RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits *

Sayantan Sur Hyun-Wook Jin Lei Chai Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
{surs, jinhy, chail, panda}@cse.ohio-state.edu

Abstract

Message Passing Interface (MPI) is a popular parallel programming model for scientific applications. Most high-performance MPI implementations use Rendezvous Protocol for efficient transfer of large messages. This protocol can be designed using either RDMA Write or RDMA Read. Usually, this protocol is implemented using RDMA Write. The RDMA Write based protocol requires a two-way handshake between the sending and receiving processes. On the other hand, to achieve low latency, MPI implementations often provide a polling based progress engine. The two-way handshake requires the polling progress engine to discover multiple control messages. This in turn places a restriction on MPI applications that they should call into the MPI library to make progress. For compute or I/O intensive applications, it is not possible to do so. Thus, most communication progress is made only after the computation or I/O is over. This hampers the computation to communication overlap severely, which can have a detrimental impact on the overall application performance. In this paper, we propose several mechanisms to exploit RDMA Read and selective interrupt based asynchronous progress to provide better computation/communication overlap on InfiniBand clusters. Our evaluations reveal that it is possible to achieve nearly complete computation/communication overlap using our RDMA Read with Interrupt based Protocol. Additionally, our schemes yield around 50% better communication progress rate when computation is overlapped with communication. Further, our application evaluation with Linpack (HPL) and NAS-SP (Class C) reveals that `MPI_Wait` time is reduced by around 30% and 28%, respectively, for a 32 node InfiniBand cluster. We observe that the gains obtained in the `MPI_Wait` time increase as the system size increases. This indicates that our designs have a strong positive impact on scalability of parallel applications.

Categories and Subject Descriptors C [2]: 2

General Terms Design, Performance

Keywords MPI, InfiniBand, Communication Overlap

1. Introduction

Cluster based computing systems are becoming popular for a wide range of scientific applications, owing to their cost-effectiveness. These systems are typically built from commodity PCs connected with high speed Local Area Networks (LANs) or System Area Networks (SANs). MPI [21] is the *de-facto* standard in writing parallel scientific applications which run on these clusters. MPI provides both *point-to-point* and *collective* communication semantics. Of these, point-to-point communications are used very widely. In fact, most of the implementations of collective communications are written on top of basic point-to-point communication functions. Further, many MPI applications [2, 17] use point-to-point communication with large messages [29, 28]. Thus, a high performance MPI point-to-point design for large messages is very critical for such applications.

For transferring large messages, typically a *Rendezvous Protocol* is used. In this protocol, the sender and the receiver negotiate the buffer availability on both sides before the message transfer actually takes place. For achieving high performance message passing for large messages, it is critical that message copies are avoided. The Rendezvous Protocol provides a way to achieve zero-copy message transfer because sender can know the location of the receiver's buffer or vice-versa.

Remote Direct Memory Access (RDMA) is a technique by which a message can be directly placed in a remote node's memory thereby avoiding intermediate copies. InfiniBand [11] is an emerging high-performance interconnect with RDMA capabilities. It can provide low latencies (around 4 μs) and high bandwidth (around 900 MB/s). Remote memory access can be of two types: RDMA Write, in which the process sending the message buffer can directly write into the memory of the receiving process; or RDMA Read, in which the process receiving the message can directly read from the sending process' memory into its own. Either of these RDMA Write or RDMA Read can be utilized to design the Rendezvous Protocol. The design choice of the RDMA semantics has impact on computation and communication overlap. Many MPI applications use non-blocking message passing calls in an attempt to overlap computation and communication [8, 15]. However, most contemporary MPI implementations are not able to provide true overlap between computation and communication even with non-blocking message passing interface. This is usually detrimental to the performance of these applications.

In this paper, we analyze in detail the design alternatives in implementing the Rendezvous Protocol using the best of these RDMA

* This research is supported in part by Department of Energy's grant DE-FC02-01ER25506, National Science Foundation's grants CNS-0403342 and CNS-0509452; grants from Intel, Mellanox, Cisco, Linux Networx and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, IBM, Appro, Microway, PathScale, Silverstorm and Sun Microsystems.

[copyright notice will appear here]

semantics. We propose a set of novel designs to use RDMA Read or RDMA Read with Interrupt for implementing the Rendezvous Protocol. Though our design and evaluation has been done on InfiniBand, we believe that it is applicable to most RDMA capable high-performance interconnects. To the best of our knowledge, no other research work proposes an event-driven RDMA Read based Rendezvous Protocol. In addition, no other research work has thoroughly analyzed these design alternatives in the context of computation/communication overlap and communication progress.

The new designs have been implemented on MVAPICH [23] [1] implementation of MPI over InfiniBand. MVAPICH is an implementation of the Abstract Device Interface (ADI) for MPICH [10]. MVAPICH is based on MVICH [16]. Compared to the current RDMA Write based Rendezvous Protocol, the new RDMA Read based designs have been able to nearly completely overlap computation and communication. Additionally, our schemes yield around 50% better communication progress rate when computation is overlapped with communication. Further, our application evaluation with Linpack (HPL) [7] and NAS-SP (Class C) reveals that `MPI_Wait` time is reduced by around 30% and 28%, respectively, on a 32 node InfiniBand cluster.

The rest of the paper is organized as follows: In section 2, we provide an overview of the InfiniBand Architecture. In section 3, we provide an overview of the Existing RDMA Write based Rendezvous Protocol and describe its limitations. In section 4, we outline the design alternatives for the Rendezvous Protocol using RDMA Read and interrupts. We evaluate the performance of our new designs in section 5. We discuss the related work in this area in section 6. Finally in section 7, we conclude this paper and present future research directions.

2. Overview of InfiniBand Architecture

The InfiniBand Architecture [11] defines a switched network fabric for interconnecting processing nodes and I/O nodes. It provides a communication and management infrastructure for inter-processor communication and I/O. In an InfiniBand network, hosts are connected to the fabric by Host Channel Adapters (HCAs). InfiniBand uses a queue-based model. A Queue Pair in InfiniBand consists of two queues: a send queue and a receive queue. The send queue holds instructions to transmit data and the receive queue holds instructions that describe where received data is to be placed. Communication operations are described in the Work Queue Requests (WQR), or descriptors, and submitted to the work queue. The completion of WQRs is reported through Completion Queues (CQs). Once a work queue element is finished, a completion entry is placed in the associated completion queue. Applications can check the completion queue to see if any work queue request has been finished. InfiniBand also supports different classes of transport services. In current products, Reliable Connection (RC) service and Unreliable Datagram (UD) services are supported.

InfiniBand Architecture supports both channel semantics and memory semantics. In channel semantics, send/receive operations are used for communication. In memory semantics, InfiniBand provides Remote Direct Memory Access (RDMA) operations, including RDMA Write and RDMA Read. RDMA operations are one-sided and do not incur software overhead at the remote side. This enables true application bypass message passing. The processor on the machine can continue its computation task without bothering about incoming messages. Thus, RDMA can positively impact the computation and communication overlap. Additionally, RDMA Write operation can gather multiple data segments together and write all data into a contiguous buffer at the receiver end. Gather/Scatter features are very useful to transfer noncontiguous data. The Gather/Scatter facility not only reduces the startup costs, but also increases network utilization. RDMA Write with Immediate data is also supported. With Immediate data, a RDMA Write operation consumes a receive descriptor and then can generate a completion entry to notify the remote node of the completion of the RDMA Write operation.

3. RDMA Write Based Rendezvous Protocol and its Limitations

MPI protocols can be broadly classified into two types:

1. **Eager Protocol:** In the Eager protocol, the sender process, *eagerly* sends the entire message to the receiver. In order to achieve this, the receiver needs to provide sufficient buffers to handle incoming messages. This protocol has minimal startup overheads and is used to implement low latency message passing for smaller messages.
2. **Rendezvous Protocol:** The Rendezvous Protocol negotiates the buffer availability at the receiver side before the message is actually transferred. This protocol is used for transferring large messages when the sender is not sure whether the receiver actually has the buffer space to hold the entire message.

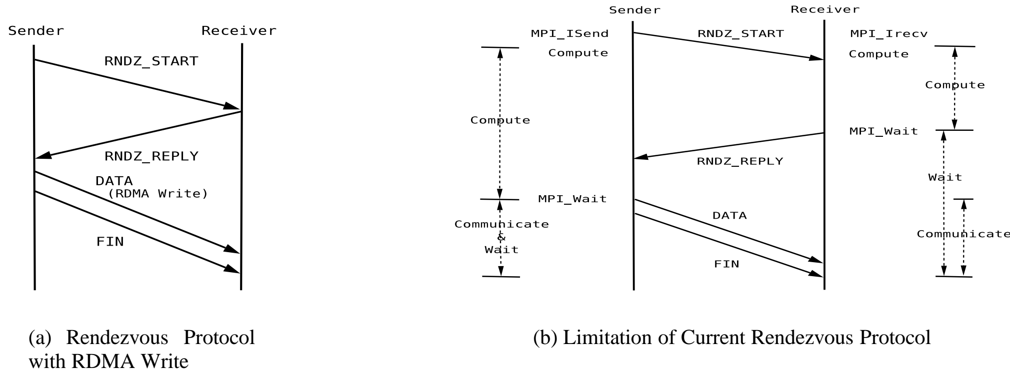
The Rendezvous protocol negotiates the buffer availability at the receiver side. However, the actual data can be transferred either by using Sockets, RDMA Write or RDMA Read. Though the socket based implementations achieve the greatest portability over various networks, it involves several levels of message copies. Thus, the Rendezvous Protocol based on sockets cannot achieve good computation and communication overlap. RDMA Write based approaches can totally eliminate intermediate copies and efficiently transfer large messages [23, 22, 24]. RDMA Read based approaches can enable both zero copy and computation and communication overlap (as will be shown in the following sections). Rendezvous protocols may also be used in other middleware such as GASNet [4].

The RDMA Write based protocol is illustrated in Figure 1(a). The sending process first sends a control message to the receiver (`RNDZ_START`). The receiver replies to the sender using another control message (`RNDZ_REPLY`). This reply message contains the receiving application's buffer information along with the remote key to access that memory region. The sending process then sends the large message directly to the receiver's application buffer by using RDMA Write (`DATA`). Finally, the sending process issues another control message (`FIN`) which indicates to the receiver that the message has been placed in the application buffer.

MPI uses a progress engine to discover incoming messages and to make progress on outstanding sends. To achieve low latency, the progress engine senses incoming messages by polling various memory locations. As can be seen in Figure 1(a), the RDMA Write based Rendezvous Protocol generates multiple control messages which have to be discovered by the progress engine. Since the progress engine is polling based, it requires the application to call into the MPI library.

However, the MPI applications might be busy doing some computational work or I/O. In this case the applications cannot make any call into the MPI library. As a result, the message transfer has to simply wait until the control messages are discovered. This scenario is illustrated in Figure 1(b). The delayed discovery of important control messages leads to serialization of the computation and communication operations. As a result, the overlap potential of computation and communication is severely hampered as shown.

[1] MVAPICH open-source software is currently used by more than 310 organizations worldwide to extract the performance of emerging InfiniBand and other RDMA capable networks.



(a) Rendezvous Protocol with RDMA Write

(b) Limitation of Current Rendezvous Protocol

Figure 1. RDMA Write Based Rendezvous and its Limitations

4. Rendezvous Protocol: Design Alternatives

In this section we discuss in detail various design alternatives for designing a high-performance Rendezvous Protocol. The main issues for designing this high-performance protocol are: *computation/communication overlap* and *communication progress*.

4.1 RDMA Read / RDMA Write: Which is beneficial?

In this section, we compare RDMA Read and Write as design alternatives and pick the best one of them. We will compare the two based on parameters like: communication progress, computation/communication overlap, number of I/O bus transactions, etc.

Typically, small messages are sent over Eager Protocol (which is copy-based) and larger messages are set over Rendezvous Protocol. According to the MPI specification, only the sender can choose the actual protocol efficiently. Particularly, the MPI Specification [21] states that: *"The length of the received message must be less than or equal to the length of the receive buffer. An overflow error occurs if all incoming data does not fit, without truncation, into the receive buffer. If a message that is shorter than the receive buffer arrives, then only those locations corresponding to the (shorter) message are modified."* According to the requirements imposed by MPI semantics, the receiver may post a much larger buffer than what the sender chooses to send. Since, the choice of size of the message actually sent (not posted size), lies with the sender, the sender can efficiently make a choice of which protocol to use (Eager or Rendezvous).

Now, we consider the case in which the sender decides to use the Rendezvous Protocol for the message transfer. Based on program execution and timing, there can be three cases.

- **Sender arrives first:** If the sender arrives first at the send call, it can send the `RNDZ_START` message immediately. Inside the `RNDZ_START` message, it can also embed the virtual address and memory handle information about the buffer to be sent. It is to be noted that upon the receipt of this `RNDZ_START` message, all the information about the application buffer is available to the receiving process. Clearly, the receiving process does not need to send a `RNDZ_REPLY` message any more. It can simply perform a RDMA Read from the application buffer location of the sending process.
- **Receiver arrives first:** Even if the receiver arrives first at the receive call, it cannot choose which protocol the message will be actually sent over. So, it must wait for the sender's choice of protocol. The receiver waits for the `RNDZ_START` message from the sender. However, once the receiver gets the `RNDZ_START` message, it can perform the RDMA Read directly from the sender buffer, without sending any more `RNDZ_REPLY` message.
- **Sender and receiver arrive at the same time:** In this case, the sender and the receiver arrive concurrently. However, neither the sender or the receiver knows whether the other process has arrived. Hence, in this case, the receiver must wait for the protocol choice from sender (as stated before), and the sender must assume that it has arrived first. Hence, again in this case, the optimal choice would be to have the sender send a `RNDZ_START` message to the receiver. As stated above, the receiving process can simply perform a RDMA Read from the sender buffer directly.

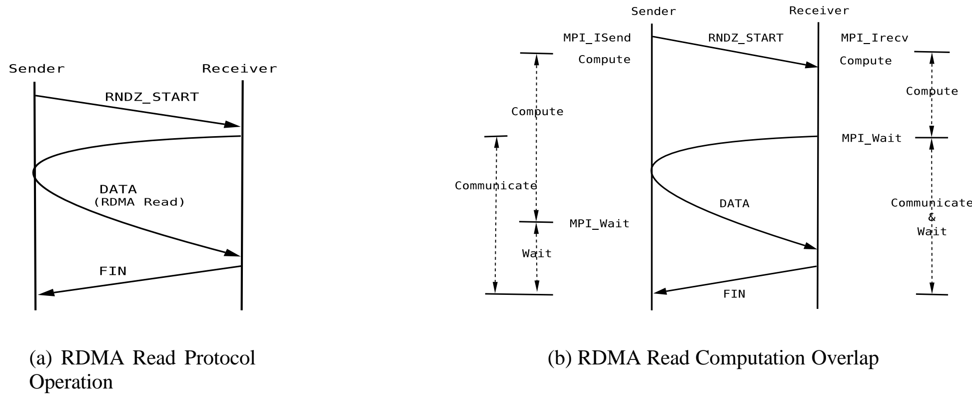
As per the above three cases, RDMA Read is chosen to reduce the number of control messages. Since the number of control messages is reduced, the total number of I/O bus transactions are reduced too. In addition, since the receiver can progress independently of the sender (once the `RNDZ_START` message is sent), we can enhance the communication progress. Further, even if the sender does not call any MPI progress, the data transfer can proceed over RDMA Read. This leads to much better overlap of computation with communication, if RDMA Read is used.

Thus, we conclude from the above that: the optimal choice of data transfer semantics is RDMA Read in all possible combinations of sender or receiver arriving at the communication point.

4.2 Design Issues for RDMA Read Based Rendezvous Protocol

In this section we describe our proposed design and implementation of the Rendezvous Protocol using RDMA Read. The basic Rendezvous Protocol over RDMA Read is illustrated in Figure 2(a). The sending process sends the `RNDZ_START` message. Upon its discovery, the receiving process issues the `DATA` message over RDMA Read. When it is done, it informs the sending process by a `FIN` message. But before we can directly utilize RDMA Read, we must address some design challenges.

Limited Outstanding RDMA Reads: The number of outstanding RDMA reads on any Queue Pair (QP) is a fixed number decided during the QP creation (typically 8 or 16). This means that we cannot directly issue a RDMA Read whenever an incoming `RNDZ_START` matches a posted receive. Instead, we use a token bucket for keeping track of the number of RDMA Reads already issued. Every time a RDMA Read is issued, we decrement the number of RDMA Read tokens available. If no more tokens are available, the RDMA Read request is placed in a FIFO queue. When the



(a) RDMA Read Protocol Operation

(b) RDMA Read Computation Overlap

Figure 2. RDMA Read Based Rendezvous Protocol

MPI progress engine is active, first requests from this FIFO queue are processed, before issuing any other RDMA Reads.

Issuing `FIN` Message: According to InfiniBand specification [11], Send or RDMA Write transactions, are not guaranteed to finish in order with outstanding RDMA Reads. In order to deal with this, we have to wait for the RDMA Read completion, before we issue the `FIN` message (over Send or RDMA Write). Alternatively, the `FIN` message can also be posted as a *fenced* operation. Completion of a fenced operation means that all outstanding reads on the particular connection are now complete. However, this fenced operation is not utilized in the current design evaluated in this paper.

The RDMA Read based Rendezvous Protocol can make progress independent of the sender (after the `RNDZ_START` message is sent). Since the sender does not need to explicitly call MPI progress function, we can achieve good computation/communication overlap on the sender side. This can be seen in Figure 2(b). However, if the receiver does not discover the `RNDZ_START` message (i.e. it is busy doing computation), then the RDMA Read will be delayed. This effect can be seen in the same Figure 2(b). Hence, the RDMA Read based Rendezvous Protocol can achieve computation/communication overlap only at the sender side, not at the receiver. The solution for this case is discussed in the next section.

4.3 Design Issues for RDMA Read with Interrupt Based Rendezvous Protocol

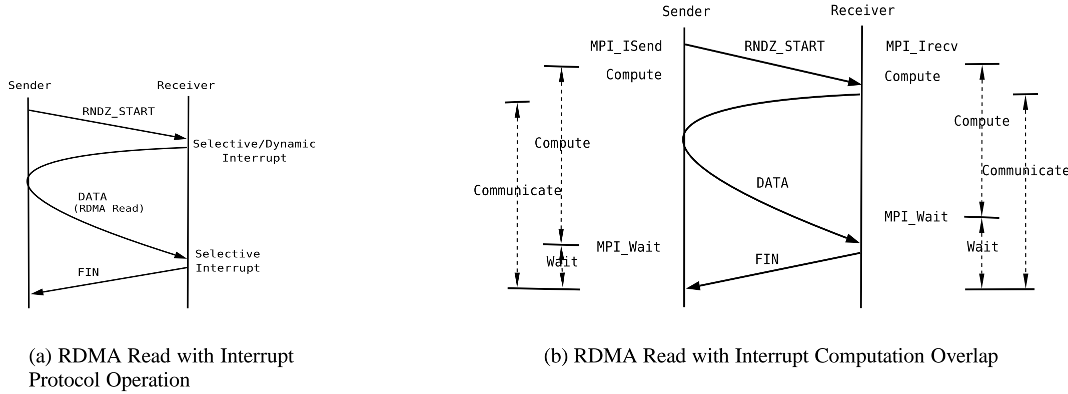
In this section we describe the design of Rendezvous Protocol using RDMA Read with interrupt. As we described earlier in this section, RDMA Read is the best data transfer mechanism when the sender arrives first. However, if the receiver arrives first, it still needs to wait for the `RNDZ_START` message from the sender. In the meantime, the receiver might be busy computing. The discovery of this `RNDZ_START` message is critical to achieving good overlap between computation and communication. Since this control message is critical, we can generate an interrupt on its arrival. This message should be handled by an *asynchronous completion handler*. The basic protocol is illustrated in Figure 3(a).

Selective Interrupt: Interrupts are usually associated with various overheads. Causing too many interrupts can harm the overall application performance. We devise a method by which we can cause a selective interrupt only on the arrival of `RNDZ_START` message and completion of RDMA Read `DATA` message. The Mellanox implementation of the Verbs Level API(VAPI) [20] provides such a feature. In order to have selective interrupts, two things must be done. First, the sender has to set a solicit bit in the descriptor (`solicit_event`) of the message which is intended to cause the interrupt. Secondly, the receiver must request for interrupts from the completion queue by setting `VAPI_SOLIC_COMP` prior to the arrival of the message.

Interrupt Suppression: Even though we have a selective interrupt scheme, back-to-back `RNDZ_START` messages should not generate multiple interrupts. This will harm the overall application performance. For designing this scheme, we disable any interrupts on the completion queue automatically after the asynchronous event handler is invoked. The event handler then keeps on polling the completion queue until there are no more completion descriptors. Thus, in this design even though back-to-back `RNDZ_START` messages might arrive, only one interrupt is generated. Finally, when there are no more completion descriptors left, the asynchronous event handler resets the request for interrupts before exiting.

Dynamic Interrupt Requests: The approximate cost of an interrupt is 18 μs (experimental platform description is given in Section 5). However, the cost of the receiver requesting an interrupt and clearing it is only 7 μs. Our design of RDMA Read with Interrupt, has such a *dynamic* scheme, in which the receiving process requests for interrupts only when pending receives are posted. If no receives are pending, then the request for interrupts is turned off, and the MPI goes into polling based progress. Whenever the interrupt is set, an internal flag indicates this status. On posting of subsequent receives, this interrupt does not need to be re-requested. Similarly, when the interrupt is cleared, an internal flag indicates that status too. This dynamic scheme can reduce the number of interrupts in the case where the sender arrives first, but the receive application hasn't posted the receive as yet.

Hybrid Communication Progress: In this new design, our asynchronous event handler is invoked by an interrupt. It executes as a separate thread to the MPI program. As we mentioned in Section 3, many MPI implementations are based on a polling progress engine. This means that whenever a MPI call is issued by the application, the MPI implementation checks all communication channels for incoming messages and makes progress on pending sends. Hence, we can potentially have two threads of the progress engine (one polling and the other handling the event) active at the same time. Thus, we need to provide a thread safe mechanism to implement this hybrid progress engine. At the same time as providing thread safety, it should also provide high performance. If there are no interrupts caused, the overhead imposed by this thread safety mechanism should be minimal. Figure 3(b) shows the computation/communication at both the sender and receiver side. In this figure, the `RNDZ_START` message causes an interrupt at the receiver. The RDMA Read `DATA` message is issued immediately. Hence, the



(a) RDMA Read with Interrupt Protocol Operation

(b) RDMA Read with Interrupt Computation Overlap

Figure 3. RDMA Read with Interrupt based Rendezvous Protocol

computation and communication can be overlapped at both sender and receiver.

5. Experimental Evaluation

In this section we evaluate our proposed designs for the optimized Rendezvous Protocol. We compare three schemes, the first one being the RDMA Write (MVAPICH version 0.9.5) [23], the second one being the RDMA Read and the third one being RDMA Read with Interrupt based Rendezvous Protocol. Our evaluation platforms used were of two types:

- **Cluster A**: 8 SuperMicro SUPER X5DL8-GG nodes with dual Intel Xeon 3.0 GHz processors. Each node has 512KB L2 cache and 2GB of main memory. The nodes are connected to the InfiniBand fabric with 64-bit, 133 MHz PCI-X interface.
- **Cluster B**: 32 nodes, dual Intel Xeon 2.66 GHz processors. Each node has 512KB L2 cache and 2GB of main memory. The nodes are connected to InfiniBand fabric with 64-bit, 133 MHz PCI-X interface.

All the machines have Mellanox InfiniHost MT23108 Host Channel Adapters (HCAs). The clusters are connected using a Mellanox MTS 14400 144 port switch. The Linux kernel version used on Cluster A and Cluster B were 2.4.22smp and 2.4.20-8smp, respectively. The InfiniHost SDK used was 3.2 and the HCA firmware version was 3.3.

5.1 Computation and Communication Overlap Performance

In this section we evaluate the ability of our designed schemes to effectively overlap computation and communication. We designed two micro-benchmarks and carried out the evaluation on Cluster A.

Sender Overlap: In this experiment, we evaluate how well the sending process is able to overlap computation with communication. The sender initiates communication using `MPI_Isend`, then computes for W μs. At the same time, the receiver is just blocking on a `MPI_Recv`. After the sender has finished computing, it checks for completion of the pending sends. The entire operation is timed at the sender. If the entire operation lasted for T μs, then the computation to communication overlap ratio is W/T.

Figure 4 shows this ratio versus the computation time. We can see that for the RDMA Write scheme, the overlap ratio is quite low. This is because the sender process is unable to receive the `RNDZ_REPLY` message due to the computation. On the other hand, the RDMA Read and RDMA Read with Interrupt schemes show nearly complete overlap. It is to be noted that for low values of computation time (W), the value of the ratio is low, since in this case, the time for communication is dominant.

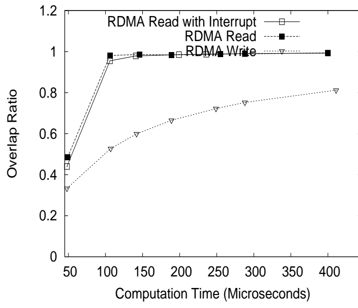
Receiver Overlap: In this experiment, we evaluate how well the receiving process is able to overlap computation with communication. This experiment is similar in nature with the sender overlap experiment. In this experiment, the receiver posts a receive using `MPI_Irecv` and computes for W μs, while the sender blocks on a `MPI_Send`. After the computation, the receiver waits for the communication to complete. The entire time is marked as T. The computation to communication ratio is W/T.

Figure 5 shows this ratio versus the computation time. We can see that for the RDMA Write and the RDMA Read schemes, the overlap ratio is quite poor. This is because the receiving process is unable to issue the `RNDZ_REPLY` or `DATA` message due to the computation. On the other hand, the RDMA Read with Interrupt scheme show nearly complete overlap, since the arrival of the `RNDZ_START` message generates an interrupt and the receiving process immediately issues the `DATA` message. As noted before, for low values of computation time (W), the communication time is dominant, resulting in a low overlap ratio.

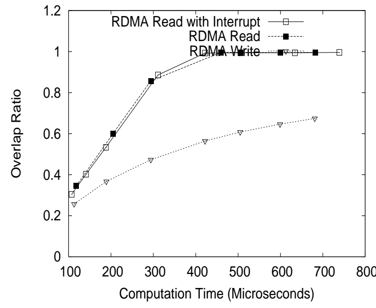
The experimental platform is dual SMPs. In the case of RDMA Read with Interrupt scheme, it may happen that the interrupt handler thread is scheduled on the "idle" processor, thus inflating the benefits of RDMA Read with Interrupt. In order to eliminate such an effect, we perform this experiment on a uni-processor kernel on the same machines. Our experiments reveal that with RDMA Read with Interrupt, we get 99.5% overlap, whereas with RDMA Read and RDMA Write we observe only 62.2% and 59% overlap, respectively, for a 1MB message size with 1800 μs computation time. These results are almost identical with the dual SMP results. This is because the interrupt handler thread consumes very little CPU time and is very short lived. It needs to be "awake" only for a few micro seconds to perform tag matching and post necessary network transactions only if it is required.

Communication Progress: In this execution, we take consecutive time stamps from the micro-benchmark execution. These time stamps are recorded just before the application enters the computation phase, in the `MPI_Wait` and from inside the MPI library when the actual communication takes place.

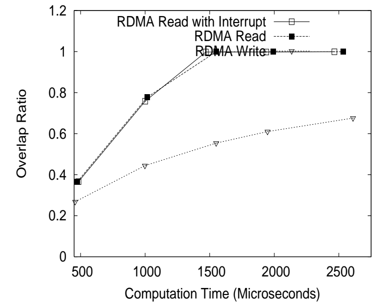
Figure 6(a) shows the progress snapshot during the sender overlap test. We observe from this figure, that in the RDMA Write based Rendezvous Protocol, the computation and communication are completely serialized. It offers no overlap at all. Whereas, in the RDMA Read based schemes, the communication happens during the application is computing. The RDMA Read based schemes



(a) Sender Overlap Performance (64KB)

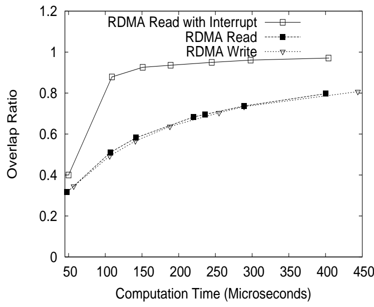


(b) Sender Overlap Performance (256KB)

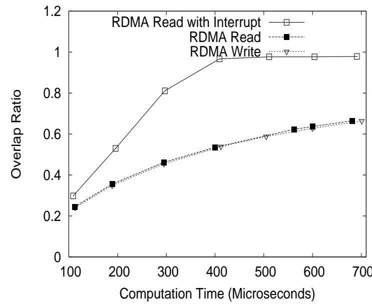


(c) Sender Overlap Performance (1MB)

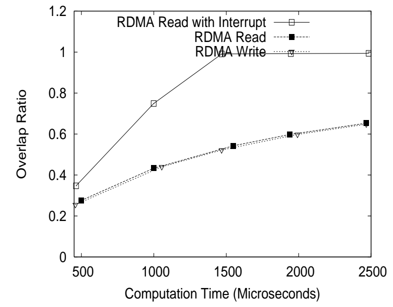
Figure 4. Sender Communication and Computation Overlap Performance



(a) Receiver Overlap Performance (64KB)

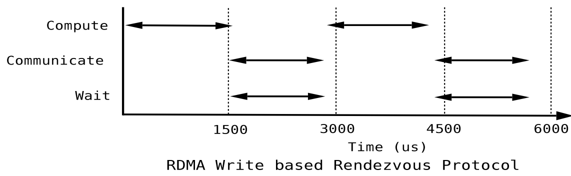


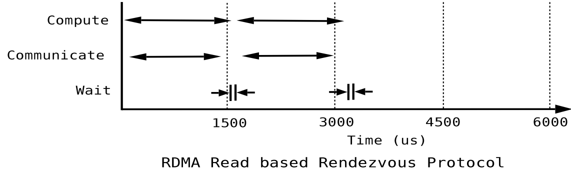
(b) Receiver Overlap Performance (256KB)

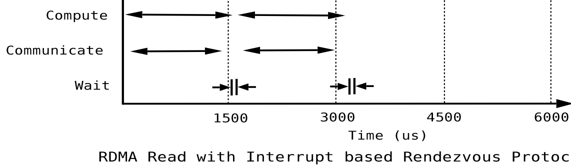


(c) Receiver Overlap Performance (1MB)

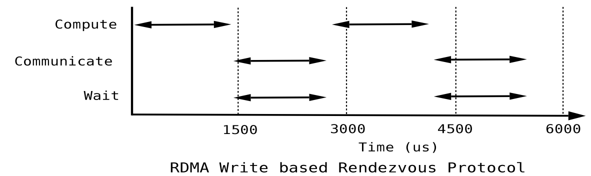
Figure 5. Receiver Communication and Computation Overlap Performance

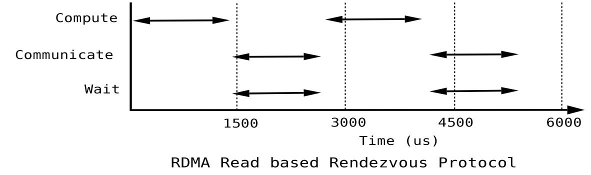


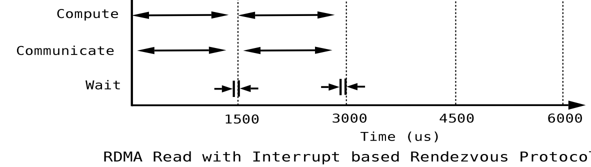




(a) Sender Overlap







(b) Receiver Overlap

Figure 6. Computation and Communication Overlap with Time Stamps

can progress 50% faster when transferring messages of 1MB and computing for 1500 μs.

Similarly, Figure 6(b) shows the progress during the receiver overlap test. We observe from this figure, that in the RDMA Write and the RDMA Read based protocol, the computation and communication are completely serialized. They hardly offer any overlap. Whereas, in the RDMA Read with Interrupt scheme, the communication happens during the application is computing. The RDMA Read with Interrupt schemes can progress around 50% faster when transferring messages of 1MB and computing for 1500 μs.

5.2 Application level Evaluation

In this section, we evaluate the impact of our RDMA Read and RDMA Read with Interrupt schemes on application wait times. For our evaluation, we choose two well known applications - HPL and NAS-SP (Scalar Pentadiagonal Benchmark). High Performance Linpack (HPL) is a well known benchmark for distributed memory computers [7]. It is used to rank the top 500 computers [26] twice every year. NAS-SP [2] is a CFD simulation which solves linear equations for the Navier-Stokes equation. We used the Class C benchmark for our evaluation.

To find out the communication time for these applications, we use a light-weight MPI profiling library [12], `mpiP`. This profiling tool reports the top aggregate MPI calls and the time spent in each one of them. We collect the aggregate time spent in the `MPI_Wait()` function call. This time is spent by the application just busy waiting for the pending sends and receives to be completed. Since this time is just wasted by the application waiting for the network to complete the operations, this represents time which can possibly be overlapped with computation. Figure 7(a) and 7(b) show the `MPI_Wait` times for HPL and NAS-SP (Class C) with increasing number of processes, respectively.

We observe that the wait time of HPL is reduced by around 30% for 32 processes by the RDMA Read and RDMA Read with Interrupt designs. Similarly, for the NAS-SP, we can see around 28% improvement for 36 processes. This is mainly because the RDMA Write based Rendezvous implementation waits till the `MPI_Wait()` to issue the `DATA` message, and hence cannot achieve good overlap. In addition, we observe from the figure that the benefits provided by the new design are scaling with the number of processes. Hence, our new design is capable of taking better advantage of network when there is possibility of overlap. In these results we see that the RDMA Read and RDMA Read with Interrupt perform equally well. This might be due to the fact that these applications do not require computation/communication overlap on the receiver side.

6. Related Work

Several researchers have proposed various schemes to achieve better MPI communication progress. The aspect of communication and computation overlap has also received due attention from researchers. In this section we present related work in this area. Sitsky and Hayashi [25] propose several methods of communication progress for the Fujitsu AP1000+. They propose an interrupt driven message detection approach for better communication progress. However, they do not consider specific interrupts for efficiently implementing the Rendezvous Protocol and in general their design considers every incoming message generating an interrupt. Keppitiyagama et al [13, 14] describe asynchronous message progress mechanism for MPI-NPII which is a network-processor based message manager for MPI. Their work highlights the benefits of computation and communication overlap, however it does not deal directly with optimizing host based Rendezvous Protocols. Tipparaju et al [27] suggest the use of a helper thread to achieve efficient one-sided communication supplementing hardware capabilities (especially in the context of non-contiguous data transfer). However, our work deals with designing a rendezvous protocol for two-sided communication. Bell and Bonachea [3] develop designs for speeding up one-sided communication for Global Address Space languages. Their design radically reduces protocol overhead and achieves zero-copy transfers. However, our work is related to protocol design for message-passing parallel programs. Amerson et al [1] describe the communication progress problem with large message transfer using the Rendezvous Protocol. Their solution also relies on an interrupt handler based approach. However, they only consider the RDMA write based semantics and not RDMA read. Brightwell et al [6, 5] have analyzed the impact of overlap on large scientific applications. They indicate the potential benefits RDMA read can provide to overlap. However, their study is mainly an analysis of applications, not an optimization of the Rendezvous Protocol itself. Macquelin et al [19] proposed a *Polling Watchdog* for efficient message handling. They introduced a simple hardware extension for combining polling and interrupts. Our work is different from theirs in the respect that we are analyzing the benefits of different RDMA semantics to reduce the number of control messages. Majumder et al [18] have proposed an event based progress mechanism for LA-MPI [9]. They indicate the benefits of such an approach to overlap in applications. However, their work is mainly over TCP/IP and does not consider RDMA read as a part of their design.

7. Conclusions and Future Work

In this paper, we have presented new designs which exploit the RDMA Read and the capability of generating selective interrupts to implement a high-performance Rendezvous Protocol. We have evaluated in detail the performance improvement offered by the new design in several different areas of high performance computing. We have observed that the new designs can achieve nearly complete computation and communication overlap. Additionally, our schemes yield a 50% better communication progress rate when computation is overlapped with communication. Further, our application evaluation with Linpack (HPL) and NAS-SP (Class C) reveals that `MPI_Wait` time is reduced by around 30% and 28% respectively for a 36 node InfiniBand cluster. We observe that the gains obtained in the `MPI_Wait` time increase as the system size increases. This indicates that our designs have a strong positive impact on scalability of parallel applications.

We plan on continuing work in this direction. We will evaluate the benefit offered by the *fenced* method of sending the FIN message, as described in Section 4.2. We want to evaluate the impact of our proposed schemes on larger scale clusters. We want to study a broad variety of applications and evaluate the benefits to them due to the new scheme. Finally, we want to improve the progress engine to support blocking mode support and see the impact of running several processes per node on end application performance.

References

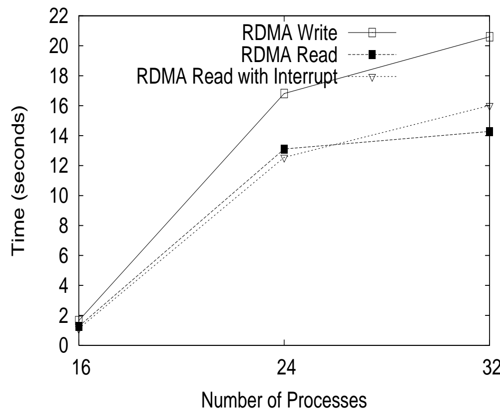
[1] G. Amerson and A. Apon. Implementation and Design Analysis of a Network Messaging Module using Virtual Interface Architecture. In *International Conference on Cluster Computing*, 2004.

[2] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.

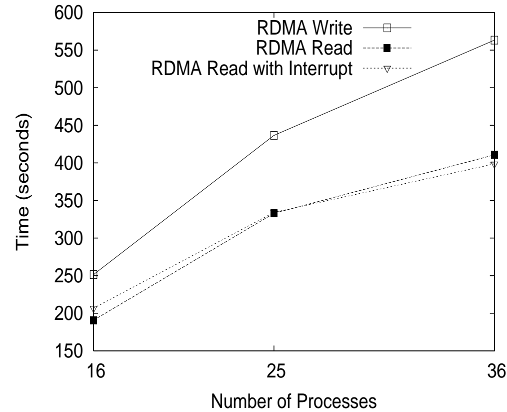
[3] C. Bell and D. Bonachea. A New DMA Registration Strategy for Pinning-Based High Performance Networks. In *International Parallel and Distributed Processing Symposium*, 2003.

[4] Berkeley Lab. GASNet. http://gasnet.cs.berkeley.edu/.

[5] R. Brightwell and K. Underwood. An Analysis of the Impact of MPI Overlap and Independent Progress. In *International Conference on*



(a) MPI Wait Time for HPL



(b) MPI Wait Time for NAS SP

Figure 7. Application Level Evaluation for Proposed Designs

Supercomputing (ICS), 2004.

[6] R. Brightwell, K. Underwood, and R. Riesen. An Initial Analysis of the Impact of Overlap and Independent Progress for MPI. In *Euro PVM/MPI*, 2004.

[7] J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85, University of Tennessee, 1989.

[8] G. Goumas, A. Sotiropoulos, and N. Koziris. Minimizing Completion Time for Loop Tiling with Computation and Communication Overlapping. In *International Parallel and Distributed Processing Symposium*, 2001.

[9] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen L. D. Risinger, and M. W. Sukalski. A Network-Failure-Tolerant Message-Passing System for Terascacle Clusters. In *International Conference on Supercomputing (ICS)*, 2002.

[10] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard. Technical report, Argonne National Laboratory and Mississippi State University.

[11] Infi niBand Trade Association. Infi niBand Trade Association. http://www.infi nibandta.com.

[12] J. Vetter and C. Chambreau. mpiP: Lightweight, Scalable MPI Profi ling. http://www.llnl.gov/CASC/mpip/.

[13] C. Keppitiyagama and A. Wagner. MPI-NP II: A Network Processor Based Message Manger for MPI. In *International Conference on Communications in Computing (CIC)*, 2000.

[14] C. Keppitiyagama and A. Wagner. Asynchronous MPI messaging on Myrinet. In *International Parallel and Distributed Processing Symposium*, 2001.

[15] V. Kumar, A. Grama, A. Gupta, and G. Karypis. *Introduction to Parallel Computing : Design and Analysis of Algorithms*. Addison Wesley / Benjamin Cummings, 1993.

[16] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. http://www.nersc.gov/research/FTG/mvich/index.html, August 2001.

[17] Lawrence Livermore National Laboratory. The ASCI Purple Benchmarks. http://www.llnl.gov/asci/platforms/purple/rfp/benchmarks/.

[18] S. Majumder, S. Rixner, and V. S. Pai. An Event-driven Architecture for MPI Libraries. In *The Los Alamos Computer Science Institute Symposium*, 2004.

[19] O. Maquelin, G. R. Gao, and H. H. J Hum. Polling Watchdog: Combining Polling and Interrupts for Effi cient Message Handling. In *International Symposium on Computer Architecture*, 1996.

[20] Mellanox Technologies. Mellanox VAPI Interface, July 2002.

[21] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[22] Myricom Inc. Portable MPI Model Implementation over GM, March 2004.

[23] Network-Based Computing Laboratory. MPI over Infi niBand Project. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[24] Quadrics. MPICH-QsNet. http://www.quadrics.com.

[25] D. Sitsky and Kenichi Hayashi. An MPI library which uses polling, interrupts and remote copying for the Fujitsu AP1000+. In *International Symposium on Parallel Architectures, Algorithms, and Networks*, 1996.

[26] The Top 500 Project. The Top 500. http://www.top500.org/.

[27] V. Tipparaju, G. Santhanaraman, J. Nieplocha, and D. K. Panda. Host-Assisted Zero-Copy Remote Memory Access Communication on Infi niBand. In *International Parallel and Distributed Processing Symposium*, 2004.

[28] J. S. Vetter and F. Mueller. Communication characteristics of large-scale scientifi c applications for contemporary cluster architectures. *Journal of Parallel and Distributed Computing*, 63(9):853 – 865, September 2003.

[29] F. C. Wong, R. P. Martin, R. H. Arpaci-Dusseau, and D. E. Culler. Architectural requirements and scalability of the NAS parallel benchmarks. In *Conference on High Performance Networking and Computing*, 1999.