Improving Communication Progress and Overlap in MPI Rendezvous Protocol over RDMA-enabled Interconnects

Mohammad J. Rashti Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University, Kingston, ON, CANADA K7L 3N6
mohammad.rashti@ece.queensu.ca ahmad.afsahi@queensu.ca

Abstract

Overlapping computation with communication is a key technique to conceal the effect of communication latency on the performance of parallel applications. MPI is a widely used message passing standard for high performance computing. One of the most important factors in achieving a good level of overlap is the MPI ability to make progress on outstanding communication operations. In this paper, we address some of the communication progress shortcomings in the current polling and RDMA Read based Rendezvous protocol used for transferring large messages in MPI. We then propose a novel speculative Rendezvous protocol that uses RDMA Read and RDMA Write to effectively improve communication progress and consequently the overlap ability. Performance results based on a modified MPICH2 over 10-Gigabit iWARP Ethernet reveal a significant (80-100%) improvement in receiver side overlap and progress ability.

1. Introduction

Clusters are the predominant platforms for high-performance computing due to their cost-performance effectiveness. Interconnection networks and the communication system software play a key role on the performance of clusters. In this regard, several modern networks such as InfiniBand [7], Myrinet [15], Quadrics [2, 18], and 10-Gigabit iWARP Ethernet [16, 20] have been introduced. Such interconnects use OS bypass, and *Remote Direct Memory Access* (RDMA) to support efficient and scalable communications. RDMA is a one-sided operation, allowing direct data transfer from the source buffer to the remote destination buffer without the host CPU intervention or intermediary copies.

Most scientific applications running on clusters are written on top of Message Passing Interface (MPI) [12]. MPI implementations typically use two different protocols for transferring small and large messages: *Eager* and *Rendezvous*. In the Eager protocol, the sender sends the entire message to the receiver, where the receiver provides sufficient buffering space for the incoming messages. This protocol is mainly used for sending small messages. The Rendezvous protocol is used for large messages, where the cost of copying is prohibitive. The sender and receiver negotiate the availability of the receiver side buffer before the actual data transfer.

Overlapping computation with communication is one of the basic techniques in hiding communication latency, thereby improving application performance. Using non-blocking communication calls at the application level [6], supporting independent progress for non-blocking operations at the messaging layer [4], and offloading communication processing to the Network Interface Card (NIC) are the main steps to achieve efficient overlap.

NICs in modern interconnects are designed to offload most of the network processing tasks from the host CPU, providing excellent opportunity for communication libraries such as MPI to hide the communication latency using non-blocking calls. To efficiently utilize the offload engines, non-blocking communications need to make progress independently. Most MPI implementations require subsequent library calls in order to make progress in outstanding non-blocking calls. This may have a significant impact on performance when a computation phase follows a non-blocking call. Specifically, communication progress becomes more important for the Rendezvous protocol, where a negotiation exists prior to the actual data transfer. This is simply because a non-blocking call may return without completing the negotiation. In [19], the

authors have shown that how lack of an independent progress in the MPI Rendezvous protocol on top of modern interconnects can affect the overlap ability.

While most MPI implementations use polling-based progress engines, some use interrupts for communication progress [21]. Although interrupt-based approaches activate the progress engine any time communication progress is needed, they impose high overhead and fluctuation on the communication time, which cannot be overlooked. Our focus in this paper is therefore to address current polling-based protocols' shortcomings by proposing a novel speculative Rendezvous protocol that could increase communication progress and overlap. While current protocols rely only on the sender to initiate the Rendezvous, our proposed protocol lets either sender or receiver initiate the negotiation in order to start the data transfer before the non-blocking send or receive call returns. This will enable overlapping any computation phase following the non-blocking calls. To the best of our knowledge, this is the first study of this kind for the MPI Rendezvous protocol.

We have implemented the new protocol on MPICH2 [13] over 10-Gigabit iWARP Ethernet. Our assessment is done using overlap and progress micro-benchmarks for both sender and receiver and for two timing scenarios, where either the sender arrives first in the communication call, or the receiver arrives first. Our experimental results indicate that the new protocol is able to outperform the current Rendezvous protocol by effectively improving the receiver side progress and overlap from almost zero to nearly 100%, at the expense of only 2-14% degradation for the send side overlap and progress when the receiver arrives first.

The rest of this paper is organized as follows. Related work is reviewed in Section 2. Section 3 discusses the proposed Rendezvous protocol. In Section 4, we present and analyze the experimental results. Finally, Section 5 concludes the paper.

2. Related work

There has been a few works on overlap and communication progress in message passing systems. In [4], the authors compared the impact of six MPI implementations on application performance running on two platforms with Quadrics QsNet [17] and CNIC network interface cards. Their results show that in almost all benchmarks, combination of offload, overlap and independent progress significantly contributes to the performance. The study in [3] concerns a similar work on IB and QsNet^{II} [2].

The authors in [5] have proposed an overlap measurement method for MPI. In their method, the communication overhead is first computed, and then application availability is calculated using the overhead amount. While [8, 23] addresses combined send and receive overlap, our proposed overlap measurement method in this paper, along with [5, 22], target send and receive overlaps separately.

In [19], the authors analyze the overlap and progress ability in InfiniBand [11], Myrinet-10G [15] and 10-Gigabit iWARP Ethernet [16], and conclude that transferring small messages makes an acceptable level of independent progress. On the other hand, in most cases, transferring large messages does not make progress independently, decreasing the chances of overlap in applications. The results in [19] confirm that independent progress is required, at least for data transfer, to achieve high overlap ability with non-blocking communication. The paper also shows how different Rendezvous protocols affect overlap and communication progress in different networks.

On the issue of using interrupts in the Rendezvous protocol, the solution in [1] is based on RDMA Write. The authors in [9] have proposed event-based progress over TCP/IP networks. In [10], some hardware mechanisms are proposed to combine interrupts and polling. Researchers have recently proposed RDMA Read Rendezvous protocols [22] for MPI to improve communication progress and overlap. In this work, a traditional two-way handshake followed by RDMA Write is replaced by a one-way handshake followed by RDMA Read. The Authors in [22] also use an interrupt-based scheme to alleviate the bottleneck when the receiver arrives first, achieving nearly complete overlap and up to 50% progress improvement in MPI over InfiniBand relative to the RDMA Write Rendezvous protocol.

The results in [22] and also [19] show that one-way Rendezvous protocols using RDMA Read help achieve a good level of overlap and progress at the send side. However, both one-way Rendezvous (used in the MVAPICH project [14] and MPICH2 over iWARP) and two-way Rendezvous (used in the MVAPICH project [14]) protocols are not able to provide independent progress and good overlap for receiving large messages [19].

3. Speculative MPI Rendezvous Protocol

Section 3.1 explains the basics of the proposed protocol for the two timing scenarios. We will then discuss the protocol design specification in Section 3.2. Finally, Section 3.3 covers the methodologies to prevent race and deadlock conditions.

3.1. Protocol Preliminaries

In the current RDMA Read based Rendezvous protocol [22], used in the implementation of MPI non-blocking communication calls for large messages, the sender sends a *Request to Send* (RTS) message to the receiver that includes the sender's data buffer address. Upon receiving the RTS, the receiver process will transfer the data using RDMA Read.

Basically, there are two send and receive timing scenarios in the current Rendezvous protocol that could happen at runtime: 1) the sender arrives first at the send call; and 2) the receiver arrives first at the receive call. The first scenario assumes that when the receiver arrives at the receive call, the RTS negotiation message is already in the receive buffer. Thus, the data transfer can start right away. In the other scenario, the receiver posts an early non-blocking call, before receiving the RTS message. Therefore, the receiver will not be able to start the RDMA Read based data transfer. In essence, the data transfer will start afterwards by the progress engine, activated either by a costly interrupt or in the next MPI communication call [22]. Thus, any computation phase after the non-blocking receive call will delay the start of data transfer [19]. To increase communication progress and overlap, this paper proposes a novel method in which the receiver can also initiate the communication.

3.1.1 Sender Arrives First. Figure 1(a) depicts the current Rendezvous protocol when the sender arrives first. In this timing scenario, we expect that a matching RTS from the sender is present at the receiver queues, and that the non-blocking receive call will transfer the data using RDMA Read. However, due to the one-sided nature of the RDMA operation used to transfer the RTS from the sender, in addition to inefficiency in the current implementation of MPICH2 over RDMA-enabled channels, the receiver is not able to find the already arrived RTS. Therefore, a *receive request* (Rreq) will be enqueued in the *receive queue* (Recvq), leaving the communication progress to a future progress engine call.

The results presented for both small and large messages in [19] (over iWARP and IB networks) highlight the existence of such inefficiency, resulting in non-independent progress for both Eager and Rendezvous protocols. Investigating the issue inside the implementation of MPI prompted us that an initial progress engine call is needed to put the RTS message from the channel-related buffers into the *unexpected queue* (Unexq). The receiver is then able to recognize the arrived RTS and take action before returning from the non-blocking call. This initial progress engine call is also beneficial for the sender, as described in Section 3.2.

3.1.2 Receiver Arrives First. Figure 1(b) depicts the current Rendezvous protocol when the receiver arrives first. In this timing scenario, the receiver will enqueue an Rreq in the Recvq, and return from the non-blocking call. The communication will progress when the progress engine becomes active by a subsequent library call. In the current protocol, the receiver does not start the Rendezvous negotiation because it is only the sender that knows the communication mode (e.g. synchronous or ready mode), and/or the exact size of the message.

However, in our proposal, the early-arrived receiver predicts the communication protocol based on its own local message size. If the predicted protocol is Rendezvous, a message similar to RTS (we call it *Request to Receive* or RTR), including the receive buffer address is prepared and sent to the sender. At the sender side, if the Rendezvous protocol is chosen, the arrived RTR message is used to transfer the data to the receiver using RDMA Write. Otherwise, if the Eager protocol is chosen, the arrived RTR will be simply discarded. Figure 2 shows a sample timing diagram for the proposed receiver-initiated Rendezvous protocol. Using this protocol, there is now potential for more progress and overlap.

3.2. Protocol Design Specification

In this section, we will discuss the design and implementation of the proposed Rendezvous protocol in MPI library. Basically, the protocol is executed by three MPI library components: the (non-blocking) send call, the (non-blocking) receive call, and the progress engine activated by subsequent library calls.

3.2.1. Non-blocking Receive. Initially, a protocol prediction is done based on the local message size in the receive call. In the case of Rendezvous protocol (large message), an initial progress engine call is made to place a possibly arrived RTS into the MPI queues. Then, for either case of the prediction outcome, the Unexq is checked for a matching message. If an Eager message is found, it will be placed into the user buffer and the communication will be finalized. Otherwise, if a matching RTS is found, the RDMA Read (data transfer) will be started.

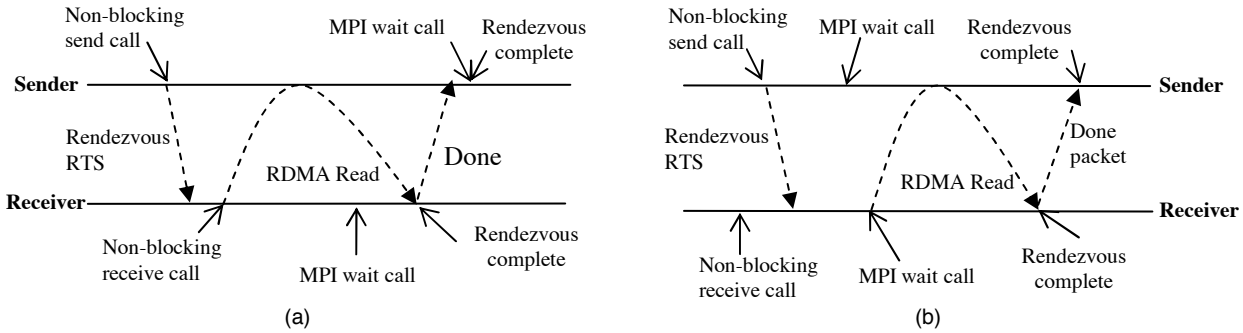

Figure 1. Timing diagram for the Rendezvous protocol in MPICH2-iWARP: (a) sender arrives first, (b) receiver arrives first.

If no matching is found in the Unexq, the receiver side will post the Rreq to the Recvq. If the Eager protocol has been predicted, the receiver returns from the non-blocking call. On the other hand, if the Rendezvous protocol has been predicted, the local memory will be registered and an RTR message will be prepared and sent to the sender to initialize the Rendezvous protocol.

3.2.2. Non-blocking Send. A non-blocking send call will first decide the appropriate protocol. The protocol selection criteria vary in different implementations, but they are mainly based on the message size and the MPI communication mode. In the case of Eager, the message is transferred eagerly.

Similar to the case for non-blocking receive, if the Rendezvous protocol is chosen, an initial progress engine call is invoked, and the Unexq is then searched for any possibly arrived matching RTR from the receiver. In case a matching RTR is found, the sender will immediately initiate the data transfer using RDMA Write. The *send request* (Sreq) will be also enqueued into a *send request queue* (Sendq) until the communication is complete. On the other hand, if no RTR is found, the sender will register the local memory and start the negotiation by sending an RTS.

3.2.3. Progress Engine. The progress engine's job in the proposed protocol is in handling the incoming RTR and RTS messages as well as finalizing the communication (that is, sending done packet and deregistering memory). Handling an incoming RTS is similar to the current protocol. It is assigned to the first matching Rreq in the Recvq. Then, the corresponding data buffer is registered, and the data is transferred using RDMA Read. If no matching request is found in the Recvq, the RTS is placed in the Unexq for future use.

Dealing with RTR is a bit different, though. If no Sreq is matched against the arriving RTR, the RTR will be placed in the Unexq to be assigned to a future Sreq. Otherwise, the RTR will be assigned to the first matching Sreq in the Sendq that has no RTR assigned to it. However, there is a possibility that the RTR from the receive call has been sent simultaneously with an RTS from the peer send call. In other words, the RTR and RTS messages may have crossed each other. In such a case, we disregard the RTR and let the receiver to perform the RDMA Read. Therefore, the sender side buffer registration and RDMA Write will take place only if no RTS has been sent to the receiver before.

3.3. Race and Deadlock Hazards

Unlike the current Rendezvous negotiation model, in which only the sender is responsible for starting the Rendezvous, in the proposed protocol both the sender and receiver are able to start the negotiation. Therefore, the new protocol should be checked for race and deadlock conditions. Due to space limitations, we will briefly discuss the race and deadlock conditions, and our proposed solutions.

3.3.1. Ambiguous RTS/RTR Destination Hazard. Both peer MPI send and receive calls could be the initiator of the Rendezvous protocol, and neither of them can determine whether the other peer has already sent an RTR/RTS message. Thus, when a matching RTS/RTR arrives, the protocol cannot determine the matching receive/send call for the arrived RTS/RTR.

To prevent the above condition, an acknowledgement (ACK) message will be sent to the other side when an RTS/RTR is received. This way, the other side will no longer expect any RTS/RTR from the send/receive call that sends the ACK. Thus, the subsequent incoming RTS/RTRs will be assigned to the next matching send/receive calls.

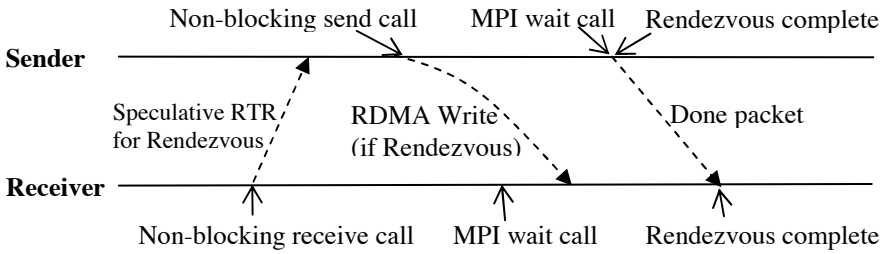

Figure 2. Timing diagram for the proposed Rendezvous protocol when the receiver arrives first.

3.3.2. Mispredicted RTR Race Hazard. Based on the sender buffer size as well as its preference in protocol selection, the receiver protocol prediction may be different than the sender's decision. If the receiver mispredicts the Rendezvous protocol, it will send an RTR, which will not be used by the sender that is using the Eager protocol. This extra RTR can be mistakenly assigned to another matching send call that selects the Rendezvous protocol.

To avoid this hazard, for each message envelope, each send call transferring an Eager message sets an *Eager-flag*. This is used as a warning for the subsequent matching Rendezvous send calls from the same process to be aware of the possibility of mispredicted RTRs. Such RTRs should be dropped. In addition, a *stop-RTR flag* is sent to the receiver to temporarily stop generating the RTR for that message envelope, until all mispredicted RTRs are dropped.

3.3.3. Deadlock. For a message transfer, deadlock happens when neither the sender nor the receiver proceeds with the communication. In the proposed protocol, there are certain cases that the receiver can post a request into the Recvq without sending an RTR. There are also cases that the sender drops the arrived RTR. However, as described for the send side in Section 3.2, either an arrived RTR is used or an RTS is sent to the receiver. Therefore, the communication will continue either by the sender using RTR, or by the receiver using RTS. This leaves no chance for deadlock for any message size.

4. Experimental Results and Analysis

We have implemented the proposed Rendezvous protocol on MPICH2 over NetEffect 10-Gigabit iWARP Ethernet. We first describe our experimental platform, and then analyze the performance results.

4.1. Experimental Framework

The experiments were conducted on Dell PowerEdge 2850 SMP servers, each with two 2.8GHz Intel Xeon processors (with 1MB L2 cache) and 2GB SDRAM. The machines run Fedora Core 4 with kernel version 2.6.11. Our iWARP network consists of the NetEffect NE020 10-Gigabit Ethernet RNICs [16], each with PCI-Express x8 interface and CX-4 board connectivity. A Fujitsu XG700-CX4 10-Gigabit Ethernet switch connects the nodes. We use MPICH2-iWARP, based on MPICH2 1.0.3 [13] over NetEffect verbs 1.4.3, which uses the Rendezvous protocol for messages larger than 128KB.

4.2. Experimental Results

In this section, we use the micro-benchmarks proposed in [19] to evaluate how the new protocol may affect overlap and progress. We use two timing scenarios, to either force the sender to arrive earlier (to use RTS), or to force the receiver to arrive earlier (to use RTR). Assessment is done at both sender and receiver sides. Interested reader is referred to [19] for more details about the micro-benchmarks.

4.2.1. Receiver Side Overlap and Progress. Enhancing the receiver side Rendezvous overlap and progress ability has been the main goal of this work. The performance results depicted in Figure 3 and Figure 4 show that the new protocol has been highly successful in its objectives. Improving the receiver side overlap ability from less than 10% to more than 80% (more than 90% in most cases) in both timing scenarios can be clearly observed in Figure 3. In our judgment, this is a significant achievement. In the case that the sender arrives first, the initial progress engine call at the receiver side has helped to find the already arrived RTS message. Thus, we have now achieved the expected level of receiver side overlap and progress. The overlap is more than 80% and the progress benchmark latencies are not affected by the inserted delay, confirming an independent progress.

The main achievement of the proposed Rendezvous protocol is for the other scenario, in which the receiver arrives earlier. Since the early-arrival receiver speculatively sends an RTR message, the late-arrival sender will find the RTR and start the communication (unlike the current protocol where the

early-arrival receiver would start communication when it receives the RTS from the sender after its computation phase). Therefore, complete progress and almost full (more than 92%) overlap is achieved during the computation phase.

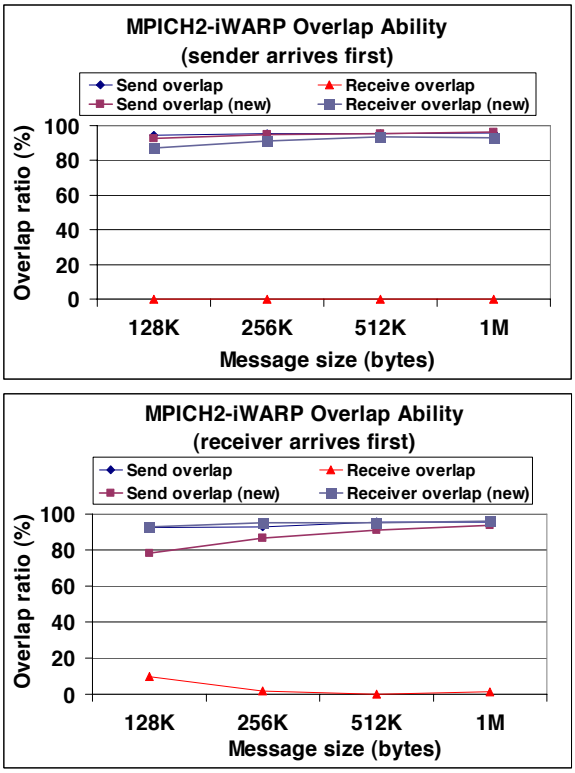

Figure 3. Current and new Rendezvous overlap ability for two timing scenarios.

4.2.2. Sender Side Overlap and Progress. As we have targeted the receiver side overlap and progress, we do not expect a major change in the send side overlap or progress ability. The results in Figure 3 and Figure 4 for the send overlap and progress comply with our expectations. Figure 4 presents the results for 1MB messages. Similar results have been observed for other long messages.

Comparing the current and the new send overlap results, we observe that except for some message sizes when the receiver arrives first, the overlap is almost at the same level or even better. This is a good achievement given our protocol has added some extra overhead at the send side. For the case that the receiver arrives earlier, the sender side overlap has dropped slightly (2-14% based on the message size). This could be due to the required RTR processing at the sender, before launching the RDMA Write.

The send side progress results comply with the overlap observations. When the sender arrives first, the same negotiation scenario occurs as in the current protocol. Therefore, we see similar progress results. However, just like the overlap results for the case that the receiver arrives earlier, the new send side progress is a bit worse. Although the corresponding results presented in Figure 4 suggest that the latency has been shifted by the inserted delay (after a certain point), examining the details of latency results indicates that only 4-15% of the whole communication latency (depending on the message size) remains after the inserted delay.

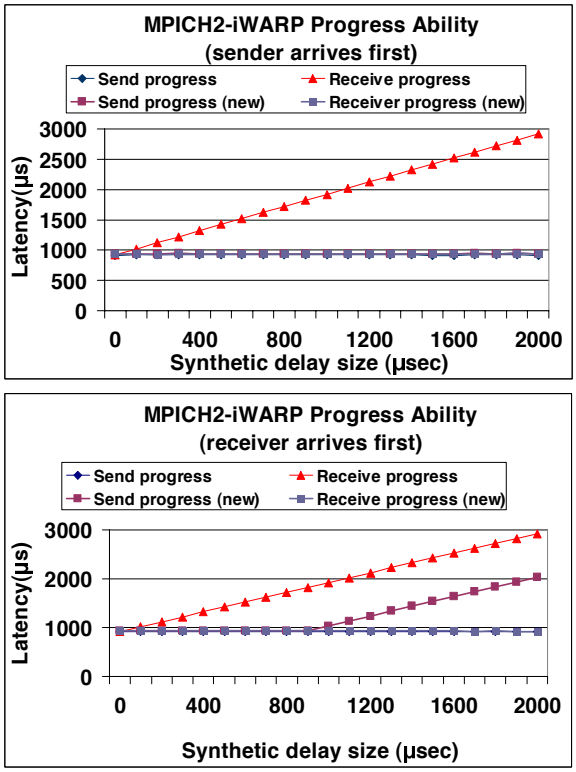

Figure 4. Current and new Rendezvous progress ability for two timing scenarios.

These results confirm that MPI has been able to make progress in more than 85% of the communication, which is in harmony with the corresponding overlap results in Figure 3. This can be attributed to the fact that we measure the latency at the receiver side to make sure the message has completely arrived. When the receiver arrives earlier and sends an RTR, the sender will start the RDMA Write after finding the RTR. Thus, the RDMA Write will be overlapped with the sender-side synthetic delay, inserted after the non-blocking call. However, the receiver does not finish the operation until receiving a *done* packet from the sender, which is sent after the delay. This affects the benchmark latency in the cases that the inserted delay is more than the message latency.

5. Conclusions and Future Work

In this work, we analyzed the overlap and communication progress ability and shortcomings of a polling and RDMA Read based MPI Rendezvous protocol on top of RDMA-enabled interconnects. To address its shortcomings, we proposed a novel speculative MPI Rendezvous protocol, and implemented it on MPICH2 over NetEffect 10-Gigabit iWARP Ethernet. Our experimental results show that the new protocol is able to improve the receiver side progress and overlap ability from almost zero to nearly 100%, at the expense of only 2-14% degradation for the send side overlap and progress. Although we targeted iWARP Ethernet, we believe this proposal can be applied directly to any other RDMA-enabled networks.

As for the future work, we would like to evaluate the proposed protocol on a larger testbed with some real applications. We also intend to investigate the protocol overhead, and study various overhead avoidance techniques. Specifically, we would like to improve the adaptability of our design to minimize the protocol overhead on applications that may not benefit from the proposed receiver-initiated Rendezvous protocol.

6. Acknowledgement

This research is supported by the Natural Sciences and Engineering Research Council of Canada through grant RGPIN/238964-2005, Canada Foundation for Innovation's grant #7154, and Ontario Innovation Trust's grant #7154. The Authors would like to thank NetEffect for the resources and technical support.

7. References

[1] G. Amerson and A. Apon. Implementation and design analysis of a network messaging module using Virtual Interface Architecture. In *2004 IEEE International Conference on Cluster Computing (Cluster'04),* pages 255-265, 2004.

[2] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, and J. Nieplocha. QsNetII: Defining high-performance network design. *IEEE Micro*, *25(4):34-47, 2005*.

[3] R. Brightwell, D. Doerfler and K.D. Underwood. A comparison of 4X InfiniBand and Quadrics elan-4 technologies. In *2004 IEEE International Conference on Cluster Computing (Cluster'04),* pages 193-204, 2004.

[4] R. Brightwell, R. Riesen and K.D. Underwood. Analyzing the impact of overlap, offload, and independent progress for Message Passing Interface applications. *International Journal of High Performance Computing Applications,* 19(2):103-117, 2005.

[5] D. Doerfler and R. Brightwell. Measuring MPI send and receive overhead and application availability in high performance network interfaces. In *EuroPVM/MPI 2006*, pages 331-338, 2006.

[6] G. Goumas, A. Sotiropoulos and N. Koziris. Minimizing completion time for loop tiling with computation and communication overlapping. In *15th IEEE/ACM International Parallel and Distributed Processing Symposium (IPDPS'01),* 2001.

[7] InfiniBand Architecture, http://www.infinibandta.org/.

[8] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff and D.K. Panda. Performance comparison of MPI implementations over InfiniBand, Myrinet and Quadrics. In *2003 ACM/IEEE Conference on Supercomputing (SC'03), 2003*.

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

[10] A. Maquelin, G.R. Gao, H.H.J. Hum, K.B. Theobald and X.-M. Tian. Polling watchdog: combining polling and interrupts for efficient message handling. In *23rd Annual International Symposium on Computer Architecture (ISCA'96),* Pages 17–188, 1996.

[11] Mellanox Technologies, http://www.mellanox.com/.

[12] MPI: A Message-Passing Interface standard, 1997.

[13] MPICH2, http://www-unix.mcs.anl.gov/mpi/mpich2/.

[14] MVAPICH, http://mvapich.cse.ohio-state.edu/.

[15] Myricom, http://www.myricom.con/.

[16] NetEffect, Inc., NetEffect NE020 10Gb iWARP Ethernet channel adapter. http://www.neteffect.com/.

[17] F. Petrini, S. Coll, E. Frachtenberg and A. Hoisie. Performance evaluation of the Quadrics interconnection network. *Journal of Cluster Computing,* 6(2):125-142, 2003.

[18] Y. Qian and A. Afsahi. RDMA-based and SMP-aware multi-port all-gather on multi-rail QsNet^{II} SMP clusters. In *36th International Conference on Parallel Processing (ICPP 2007),* 2007.

[19] M.J. Rashti and A. Afsahi, Assessing the ability of computation/communication overlap and communication progress in modern interconnects, In *15th Annual IEEE Hot Interconnects Symposium* (HotI 2007), 2007.

[20] M.J. Rashti and A. Afsahi. 10-Gigabit iWARP Ethernet: comparative performance analysis with InfiniBand and Myrinet-10G. In *7th IEEE Workshop on Communication Architecture for Clusters (CAC'07)*, 2007.

[21] D. Sitsky and K. 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.

[22] S. Sur, H. Jin, L. Chai and D.K. Panda. RDMA Read based rendezvous protocol for MPI over InfiniBand: design alternatives and benefits. In *11th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP 2006),* pages 32-39, 2006.

[23] R. Zamani, Y. Qian and A. Afsahi. An evaluation of the Myrinet/GM2 two-port networks. In *3rd IEEE Workshop on High-Speed Local Networks (HSLN 2004),* pages 734-742, 2004.