Assessing the Ability of Computation/Communication Overlap and Communication Progress in Modern 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

Computation/communication overlap is one of the fundamental techniques in hiding communication latency. Independent progress support in messaging layer, network interface offload capability and application usage of non-blocking communications are believed to increase overlap and yield performance benefits. In this paper, we analyze four MPI implementations on top of three high-speed interconnects (InfiniBand, Myrinet and iWARP Ethernet) in their ability to support overlap and communication progress. The results confirm that the offload ability needs to be supported with communication progress to increase the level of overlap. Our progress engine micro-benchmark results show that in all examined networks transferring small messages makes an acceptable level of progress and overlap. On the other hand, in most cases, transferring large messages does not make progress independently, decreasing the chances of overlap in applications.

1. Introduction

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, supporting independent progress for non-blocking operations at the messaging layer, and offloading communication processing to the Network Interface Cards (NIC) are the main steps in achieving efficient computation/communication 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 Message Passing Interface (MPI) [10] to hide the communication latency using non-blocking calls. To utilize the offload engines efficiently, non-blocking communications need to make progress independently. While some MPI implementations support independent progress, others 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.

The main contribution of this paper is to better understand the ability of contemporary interconnects and their MPI implementations in supporting communication progress, overlap and offload. We analyze four MPI implementations on top of three modern interconnects: Mellanox InfiniBand [9], Myricom Myri-10G [13] and the recently introduced NetEffect 10-Gigabit iWARP Ethernet [14]. We conduct our experiments using micro-benchmarks for both MPI non-blocking send and receive operations. Since the MPI libraries under study use *Eager* and *Rendezvous* protocols [11] to transfer small and large messages respectively, we analyze the results for small and large messages separately. The results suggest that offloading helps to achieve high level of overlap in all networks, if non-blocking calls make progress, at least in data transfer.

The rest of this paper is organized as follows. Section 2 provides an overview of the interconnects. Related work is reviewed in Section 3. Section 4 describes our platform. Latency and bandwidth results are presented in Section 5. Section 6 discusses the MPI overlap ability of the interconnects. In Section 7, we analyze the MPI communication progress behavior. Finally, Section 8 concludes the paper.

2. Overview of Interconnects

2.1. Mellanox InfiniBand

InfiniBand (IB) is an I/O interconnection consisting of end nodes and switches managed by a central subnet-manager [7]. End nodes use Host Channel Adapters (HCA) to connect to the network. IB verbs is the lowest level of software to access the IB protocol processing engine offloaded to the HCA. The verbs layer has queue pair (QP) based semantics, in which processes post send or receive work requests (WR) to send or receive queues, respectively. The WR is used by the user to send data from the source process address space into the send queue, or to wait for a matching data to arrive into the receive queue. A completion queue associated with the QP is used by the hardware to place completion notifications for each WR. IB verbs require registration of memory buffers prior to their usage.

Our IB network consists of 10Gb/s Mellanox MHEA28-XT (MemFree) two-port HCA cards, each with a PCI-Express (PCIe) x8 interface, connected to a Mellanox 12-port 4X MTS2400-12T4 IB switch [9]. VAPI is the software interface for Mellanox HCA.

2.2. Myricom Myri-10G

The most recent Myricom product is the 10-Gigabit Myri-10G networks, accompanied with MX-10G user-level library [13]. The Myri-10G NICs are dual-protocol cards that support 10-Gigabit Ethernet and 10-Gigabit Myrinet. Myricom has its own Myri-10G switches, but the NICs can also be used in an Ethernet environment using standard 10-Gigabit Ethernet switches. In addition to Myrinet network protocol processing, Myri-10G NICs offload some Ethernet network processing such as TCP checksum, segmentation and re-assembly, but they are not full TCP Offload Engines (TOE) [5]. The MX-10G library has semantics close to MPI. Basic communication primitives are non-blocking send and receive operations that are directly used in the implementation of MPI. The library registers user buffers internally.

We have used the single-port Myri-10G NICs (10G-PCIE-8A-C) with 10GBase-CX4 ports [13], each with a PCIe x8 interface. Myri-10G NICs are connected to a Myricom Myri-10G 16-port switch.

2.3. NetEffect 10-Gigabit iWARP Ethernet

The RDMA consortium [17] has developed a set of standard extensions to TCP/IP, collectively known as *iWARP*. The specification proposes a set of descriptive interfaces at the top layer, called iWARP verbs [6]. Verbs provide a user level abstract interface for direct access to the *RDMA enabled NIC* (RNIC), which gives more benefits than a simple TOE by offering direct data transfer ability and OS bypass using RDMA. Verbs have QP based semantics similar to IB network, and require user buffers to be locked down before the data transfer takes place.

Below the verbs layer, there is the *RDMA Protocol* (RDMAP) layer that is responsible for performing RDMA operations and supplying communication primitives for remote memory access calls in verbs. Below that is the *Direct Data Placement* (DDP) layer, which is used for direct transfer of data from the user buffer to the iWARP RNIC. The next layer, the *Marker PDU Aligned* (MPA) layer is used to put boundaries on DDP messages that are transferred over the stream oriented TCP protocol.

Our iWARP network consists of the second-generation NetEffect two-port NE020 10-Gigabit Ethernet RNICs [14], each with a PCIe x8 interface and CX-4 board connectivity. NetEffect verbs is the user-level interface for NE020. A Fujitsu XG700-CX4 12-port 10-Gigabit Ethernet switch is used to connect the cards together.

Figure 1 depicts the block diagram of the NetEffect NE020 RNIC [14]. It has a Virtual Pipeline Architecture, which combines state-machine based Protocol Engines, embedded cores, and a Transaction Switch integrating iWARP, IPv4 TOE and NIC acceleration logic in hardware. The NE020 has a 256MB on-board DDR2 memory bank. Utilizing its multiple parallel protocol engines, the NetEffect RNIC can support a large number of simultaneous iWARP connections [14, 16]. The RNIC can be accessed using user-level and kernel-level libraries such as NetEffect verbs, OpenFabrics verbs, standard sockets, SDP, uDAPL, and MPI.

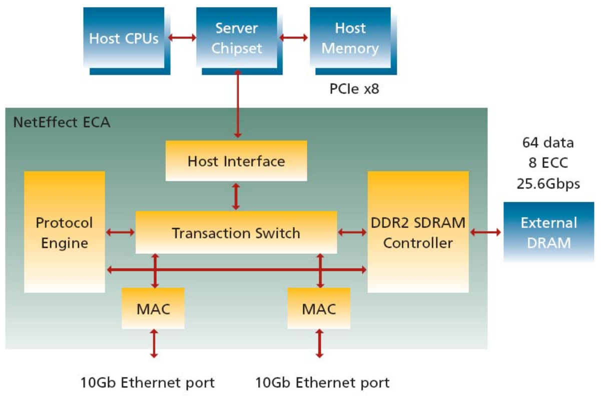


Figure 1. NetEffect NE020 iWARP Ethernet RNIC architecture (courtesy of NetEffect)

3. Related Work

A small body of work exists concerning the analysis of overlap and communication progress in contemporary networks [3, 2, 4, 19, 18]. In [3], the authors have discussed the concept of independent progress, offload and overlap, and compared the impact of six MPI implementations on application performance running on two platforms with Quadrics QsNet [15] and CNIC network interface cards (used in ASCI Red machine) [3]. Their results show that in almost all benchmarks, the combination of offload, overlap and independent progress significantly contributes to the performance. The study in [2] concerns a similar work on IB and Quadrics QsNet^{II} [1]. Our work is along the same direction as in [3, 2] with the difference that it is focused at analyzing the ability of modern interconnects and their MPI implementations in supporting communication progress and overlap rather than an application's ability in leveraging them. Moreover, we consider two state-of-the-art networks that have not been discussed yet.

Among the work on measuring the overlap ability of networks, the authors in [4] have proposed an overlap measurement method for MPI. They first compute the communication overhead, and then application availability is calculated using the overhead amount. The researchers in [18] have presented an instrumentation framework to estimate the lower and upper bounds on the achieved overlap for both two-sided and one-sided communication over IB. While [8, 20] addresses combined send and receive overlap, our proposed overlap measurement method in this paper, along with [4, 19, 18] target non-blocking send and receive overlaps separately.

On improving communication progress and overlap, researchers have proposed RDMA Read based Rendezvous protocols for MPI. They have achieved nearly complete overlap and up to 50% progress improvement relative to the RDMA Write Rendezvous protocol [19].

4. Platform

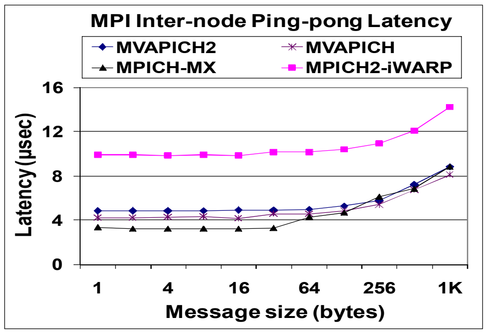
We have conducted our experiments using two Dell PowerEdge 2850 servers. Each machine is a dual-processor Intel Xeon 2.8GHz SMP with 1MB L2-cache per processor, 2GB total physical memory and an x8 PCIe slot. The machines run Linux Fedora Core 4 SMP for IA32 architecture with kernel 2.6.11.

The NetEffect iWARP MPI is based on MPICH2 version 1.0.3 [11] over NetEffect verbs 1.4.3. For Myri-10G, we use MPICH-MX based on MPICH 1.2.7..1 [11]. For IB, we use MVAPICH2 version 0.9.5 and MVAPICH 0.9.8 [12] over VAPI library. To have a fair analysis, our tests are done using single-port communication. Based on Myricom's advice, Myri-10G cards were forced to work in the PCIe x4 mode for effective performance on the nodes' Intel E7520 chipset.

5. Latency and Bandwidth

Before analyzing the ability of overlap and independent progress, we first investigate whether the networks under study have comparable MPI performance in terms of basic message latency and bandwidth. For latency, we use the standard ping-pong micro-benchmark. For bandwidth, we use a both-way communication micro-benchmark in which two processes simultaneously post a window of non-blocking send calls followed by a window of non-blocking receive calls.

Figure 2 shows the latency and bandwidth of the four MPI implementations. It is evident that the networks have comparable performance results, at least in the same order of magnitude. Although NetEffect iWARP has a considerably higher latency for small messages, it offers higher bandwidth for large messages. It is worth mentioning that the new iWARP card outperforms its previous generation, NE010, reported in [16]. Note that the low bandwidth for Myrinet is primarily due to forcing the Myri-10G NICs in PCIe x4 mode.



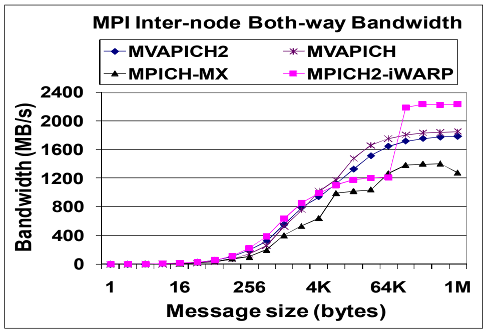


Figure 2. MPI latency and bandwidth

6. Computation/communication Overlap

In this section, we measure the overlap ability of MPI implementations for both non-blocking send and receive operations. In our tests, unlike the method used in [4] we directly measure the amount of computation that can be overlapped with communication.

In the send overlap micro-benchmark, the sender uses a loop consisting of a synchronous non-blocking send call, *MPI_Issend()*, to start the transfer, followed by a computation phase. It then waits for the message transfer to complete using *MPI_Wait()*. The receiver blocks on an *MPI_Recv()*. Note that the timing is done at the sender side.

In the receive overlap micro-benchmark, the receiver posts a non-blocking receive call, *MPI_Irecv()*, and then computes. It then waits for the communication to complete using *MPI_Wait()*. The sender blocks on an *MPI_Ssend()*. Timing is done at the receiver side. Note in both overlap tests, synchronous send operations are used to make sure that the communication completes at the same time at both ends in order to be able to calculate the overlap ability for the whole communication period.

To calculate the overlap in each iteration *m*, we measure the portion of performed computation that can be fully overlapped with communication, without affecting the communication time. Let l_0 be the communication time with no inserted computation, c_m, the computation time inserted in the iteration *m*, and l_m, the communication time of the iteration *m*. We increase the amount of computation, c_m, by 10% in each iteration until 10% increase in the original communication time, l_0, is observed (the 10% values are approximate numbers, selected based on empirical observations). The last iteration is the largest *m*, with $l_m < (1.1)\, l_0$.

Figure 3 illustrates the timing model of our overlap micro-benchmark. In this Figure, t_1 is the time period that the non-blocking send/receive call is active. In fact, the host CPU is busy with communication processing during t_1. Obviously, the best-case scenario is when the non-blocking call is able to start the communication in the NIC offload engine, before returning from the call. The t_2 period starts when the non-blocking call returns. In this period, the NIC is performing the data transfer using its offload engine, and the CPU is available for computation. Clearly, t_2 is the upper bound for overlap. The t_3 period is the time that the progress engine, called by *MPI_Wait(),* will complete the communication. It starts right after the computation phase (c_m) or the offload phase (t_2), whichever finishes later.

Basically, the original communication latency can be calculated using Equation (1). Obviously, if the computation phase in iteration *m* (c_m) is smaller than t_2, it will be fully overlapped with the communication, and therefore l_m would be equal to l_0, as in Equation (2). It is clear that the overlapped computation amount is equal to c_m. On the other hand, if c_m is greater than t_2 (not shown in Figure 3), since no more than t_2 time is available for overlap, the start of t_3 period will be delayed, consequently increasing the l_m, as in Equation (3). Clearly, in this case, the overlap time is equal to t_2.

$$l_0 = t_1 + t_2 + t_3 \tag{1}$$

$$l_m = t_1 + t_2 + t_3 = l_0 \tag{2}$$

$$l_m = t_1 + c_m + t_3 \tag{3}$$

Using the above equations for iteration *m*, one can derive the overlap time for either of the discussed cases, as c_m - (l_m- l_0). Therefore, the *overlap ratio* is:

$$overlap\_ratio = \frac{c_m - (l_m - l_0)}{l_0} \tag{4}$$

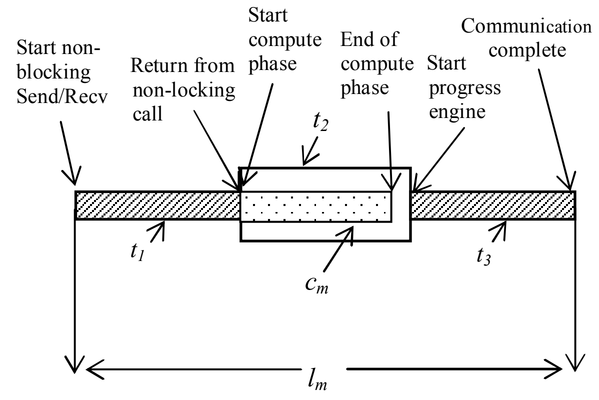
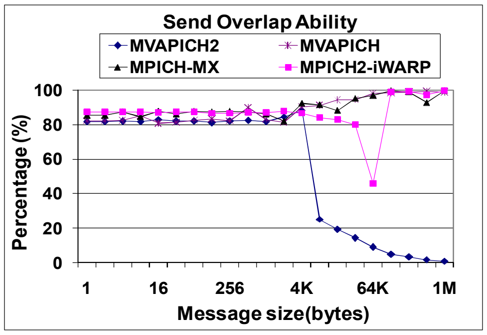


Figure 3. Timing model for overlap benchmark

6.1. Send Overlap Observations

Figure 4 illustrates the send and receive computation/communication overlap ability of the networks with their MPI implementations. One can observe a different behavior for small (Eager range) and large (Rendezvous range) messages. For sending Eager size messages, all networks show a high level of overlap ability that reflects the contribution of offloaded network processing as well as MPI progress (see Section 7.1 for more details). In our platform, the Eager protocol is used for messages up to 5980 bytes, 131052 bytes and 32KB in MVAPICH/MVAPICH2, MPICH2-iWARP and MPICH-MX, respectively.

For the Rendezvous protocol, MPICH-MX, MVAPICH and MPICH2-iWARP maintain their high overlap ability. In the MPICH2-iWARP case, after a sharp decrease for 64KB messages (mostly due to buffer copy cost) the overlap ability increases from 128KB and approaches 100% for 1MB messages. This is because in the Rendezvous protocol, after a one-way handshake from the sender, called *Request To Send* (RTS), the receiver uses RDMA Read to retrieve the message from the sender's memory, and therefore the sender is almost free during this operation. The same story applies to MVAPICH over IB [19]. In MPICH-MX, data is transferred by a direct *Get* initiated by the receiver using a progression thread, which makes both the sender and receiver processors available [13].



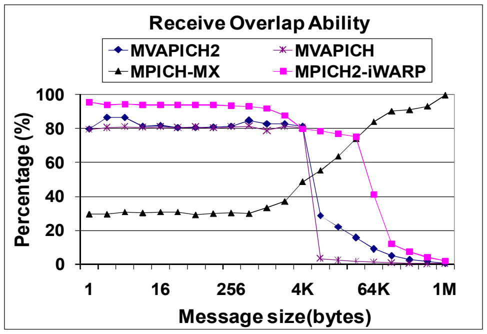


Figure 4. Computation/Communication overlap

On the other hand, the MVAPICH2 Rendezvous protocol results show significant degradation where the overlap becomes close to zero for very large messages. This is primarily because MVAPICH2 uses a two-way handshake protocol followed by RDMA Write for data transfer [19]. In this protocol, the sender sends an RTS to the receiver. The receiver will reply with a *Clear To Send (CTS)* message that enables the sender to start the data transfer. Therefore, when the sender enters a computation phase after its non-blocking call, the negotiation cycle (reception of CTS) remains incomplete, delaying the start of data transfer until the next MPI communication call at the sender side.

6.2. Receive Overlap Observations

In both iWARP and IB networks, shown in Figure 4, a high level of overlap ability exists for the MPI receive operation up to Eager/Rendezvous switching point. After the switching point, this overlap ability drops and becomes close to zero for 1MB messages. Obviously, for large messages, the non-blocking calls do not start the data transfer, serializing computation and communication phases. This could be due to inefficiencies in MPI progress engine as well as the Rendezvous protocol.

A different behavior is observed though for Myrinet network. For the Eager case, the overlap ability, in the range of 30%-75%, increases after 1KB messages (we are investigating the reason behind this). For the Rendezvous case, since most of data transfer is started with a progression thread [13], the receiver CPU is free during the data movement.

7. Communication Progress

Communication progress is called independent if a pending non-blocking communication can make progress without subsequent library calls. This ability helps to overlap the on-going communication with computation. The level of independent progress depends on the communication protocols, the progress engine and the underlying hardware offload support.

Generally, the progress can be made in protocol negotiations and/or data transfer. Inevitably, with the existence of offload, data movement can proceed independently from the host CPU. However, the start of data transfer and the completion of communication protocol may still need further library calls. In such a case, even in the existence of offload, any computation phase that follows a non-blocking call may delay the progression of the communication protocol that has already been started by the non-blocking call.

7.1. Sender Side Progress

In our send progress micro-benchmark, the sender invokes a non-blocking send call after synchronization with the receiver. It then spends a certain amount of time in a synthetic delay routine. Unlike the tests for measuring overlap, the inserted computational delay is chosen to be longer than the basic message latency to clearly highlight the progress ability. After the delay, *MPI_Wait()* is called for the completion of the send request. At the other end and after synchronization, the receiver calls a blocking receive operation. The overall receive latency is recorded for several inserted delay values for each message size. If the latency is increased by the inserted delay, then the communication has not made progress independently.

Figure 5 depicts the measured results for the send communication progress in the four MPI implementations. For MVAPICH2 and MPICH-MX, the latency for Eager size messages is not affected by the inserted delay. This shows that the communication completes during the delay time without any subsequent MPI calls at the sender side. This applies to MPICH2-iWARP and MVAPICH, however for all message sizes including those in the Rendezvous protocol range. The results for MPICH2-iWARP and MVAPICH are in concert with their high send overlap ability shown in Figure 4. In fact, using a one-way handshake Rendezvous makes the sender-side CPU free, after sending the RTS to the receiver.

On the other hand, for Rendezvous range of messages in MVAPICH2 and MPICH-MX, the latency is affected by the inserted delay, making the completion of communication happen after the delay. Examining the details of latency values for different inserted delays shows that for all affected cases only a portion of the original message latency remains after the inserted delay. For example in the MPICH-MX case, 15 to 35 percent of the original communication time for 64KB messages remains after the inserted delay. This post-delay work value for MVAPICH2 is as high as 84% of the original latency for 32KB messages. This shows that both MPICH-MX and MVAPICH2 do not have perfect independent progress for sending large messages.

High post-delay work values for MVAPICH2 imply that even the data movement is affected. This is mostly due to the fact that, even in the best-case scenario, both send and receive non-blocking calls are consumed for Rendezvous negotiation (RTS and CTS messages), leaving the data transfer for future progress engine calls. This is in harmony with the justification for MVAPICH2 send overlap behavior in the Rendezvous range (Section 6.1).

On the other hand, the relatively low values of post-delay work and high send overlap ability for Myrinet suggest that the large message data transfer makes progress during the computation phase, despite some post processing work that remains after the inserted delay. The post processing includes a final data copy and protocol finalization. The result for 64KB messages when inserting 100μs delay confirms our argument. In essence, the entire 100μs delay is overlapped with the 162μs message transfer latency and the delay-affected latency becomes equal to the non-delayed latency.

The best Rendezvous progress results are for MPICH2-iWARP and MVAPICH. Here, in a shortened Rendezvous negotiation, the receiver retrieves the data from the sender using RDMA Read upon reception of RTS from the sender [19]. Therefore, the receiver can start the data transfer, independently making progress by utilizing offloaded protocol engine. That is why we see a flat delay curve for large messages. The decrease in latency with 100μs inserted delay relative to no delay is speculated to be due to *MPI_Wait()* polling overhead on the NIC. When there is no inserted delay, the wait call polls the NIC continuously, affecting the performance of RDMA Read request from the peer process.

7.2. Receiver Side Progress

We run a similar test for the receive operation. The timing measurement is again performed at the receiver side. After a barrier synchronization, the receiver posts a non-blocking receive call and enters a synthetic delay. During the delay, it also checks for probable completion of the message reception by examining the receive buffer for a certain data value. This helps to detect whether the message has been completely received while the receiver is in the delay period. After the delay, *MPI_Wait()* is called to complete the request. At the send side, a blocking send is called after the synchronization.

Results of the receive progress micro-benchmark for MPI libraries under study are shown in Figure 6. Clearly, none of the networks shows independent progress for receiving Eager size messages, and their completion is delayed by the inserted delay time. In general, Eager size messages in MPICH are transferred upon posting the MPI send call. The observed delay effect is because the reception of the message is not checked in the non-blocking receive call and is left to the progress engine in future MPI calls. Thus, the completion of the receive operation remains for the time that we call *MPI_Wait()*, which is in fact after the inserted delay.

Examining the details of receive latencies for Eager-size messages in all networks confirms that only a small portion of the original (non-delayed) communication latency remains after the inserted delay. For instance, the amount of post-delay work for iWARP is up to 30% of the original message latency. This implies that the actual data transfer has been performed during the delay (using direct memory transfer), but the post processing has remained for the MPI progress engine, which is invoked after the delay. This post processing includes finding the arrived message in the unexpected queue, copying it into the user buffer and finalizing the receive data structure.

The case for the Rendezvous protocol is quite different. For MVAPICH, MVAPICH2 and MPICH2-iWARP, the results look similar to that of Eager case, but the latency details imply a different story. In fact, all of the message latency remains after the inserted delay. This means that, unlike the discussion in Section 7.1 for send progress, even the data transfer does not start until the inserted delay ends. This is primarily because the non-blocking (send or receive) call does not find the Rendezvous control message (CTS or RTS) to start the data transfer. These messages will be recognized only at the next progress engine execution.

An interesting observation in Figure 6 is the flat curve for receiving Rendezvous range messages in Myrinet. This reflects significant offload ability of the NIC and verifies that transferring large messages with MX imposes a very small processor overhead on both sides, primarily because of using a progression thread which takes care of data transfer and protocol negotiations [13].

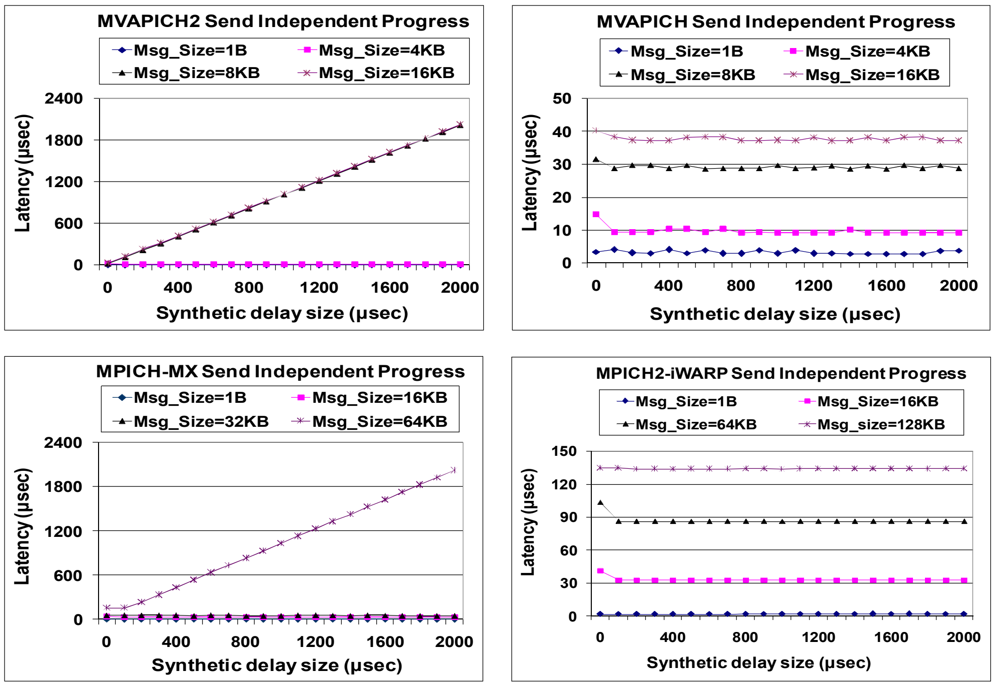


Figure 5. MPI send progress results

8. Conclusions and Future Work

In this paper, we assessed the ability of modern cluster interconnects with offload capability and their MPI libraries for computation/communication overlap and communication progress. In terms of overlap, all of the MPI libraries are able to overlap computation with sending small messages that are transferred eagerly. All libraries, except MVAPICH2, also show a high level of overlap ability for sending large messages. However, not a significant portion of the time for receiving large messages using MPICH2-iWARP and MVAPICH, receiving small messages using MPICH-MX and sending/receiving large messages using MVAPICH2 can be overlapped with computation.

The MPI communication progress results confirm that without independent progress (at least in data transfer) we cannot achieve high level of overlap. MPICH2-iWARP and MVAPICH that have shown a good send overlap ability show independent progress for send operation as well. Myrinet also shows a good progress for send especially for small messages, although not fully independent. MVAPICH2 shows the worst progress results for sending large messages just like its send overlap ability.

All networks show some level of receive progress for small messages, although it is just in data transfer. MPI over iWARP and IB show poor receive progress for large messages because the Rendezvous negotiation (reception and recognition of RTS at the receiver side) does not complete in non-blocking calls, preventing the data transfer to start. Myrinet has the best receive progress in this range due to the existence of an independent progression thread.

As for the future work, we intend to devise novel MPI communication protocols that could address the deficiencies in the current MPI implementations, highlighted in this paper.

9. Acknowledgments

The Authors would like to thank NetEffect for the resources to conduct iWARP tests and their technical support. We acknowledge Myricom for providing the Myrinet switch. We thank the anonymous referees for their insightful comments. 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 first author is also supported in part by Sun Microsystems of Canada Scholarship in Computational Science and Engineering.

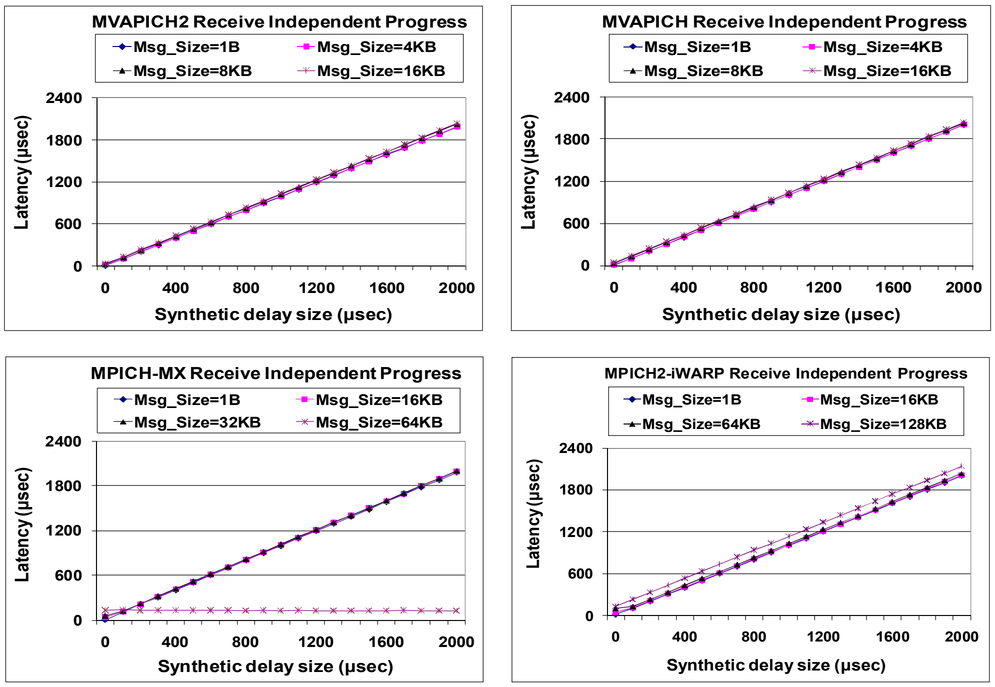


Figure 6. MPI receive progress results

10. References

[1] 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, July-Aug. 2005.

[2] 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 2004),* pages 193-204, 2004.

[3] 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.

[4] 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.

[5] W. Feng, P. Balaji, C. Baron, L.N. Bhuyan and D.K. Panda. Performance characterization of a 10-Gigabit Ethernet TOE. In *13th IEEE Symposium on High-Performance Interconnects (Hot Interconnects 2005),* 2005.

[6] J. Hilland, P. Culley, J. Pinkerton and R. Recio. RDMA protocol verbs specification (Ver1.0), 2003. http://www.rdmaconsortium.org/

[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 2003),* 2003.

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

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

[11] MPICH and MPICH2. http://www-unix.mcs.anl.gov/mpi/

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

[13] Myricom, Inc., http://www.myricom.com/

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

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

[16] 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 2007)*, 2007.

[17] RDMA Consortium. iWARP protocol specification. http://www.rdmaconsortium.org/

[18] A.G. Shet, P. Sadayappan, D.E. Bernholdt, J. Nieplocha and V. Tipparaju. A performance instrumentation framework to characterize computation-communication overlap in message-passing systems. In *2006 IEEE International Conference on Cluster Computing (Cluster 2006),* pages 1-12, 2006.

[19] 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.

[20] 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'04)*, pages 734-742, 2004.