Scalable MPI Design over InfiniBand using eXtended Reliable Connection

Matthew J. Koop, Jaidev K. Sridhar, Dhabaleswar K. Panda

Network-Based Computing Laboratory
The Ohio State University
2015 Neil Ave., Columbus, OH 43210 USA
{ koop, sridharj, panda }@cse.ohio-state.edu

***Abstract*—A significant component of a high-performance cluster is the compute node interconnect. InfiniBand, is an interconnect of such systems that is enjoying wide success due to low latency (1.0-3.0μsec) and high bandwidth and other features. The Message Passing Interface (MPI) is the dominant programming model for parallel scientific applications. As a result, the MPI library and interconnect play a significant role in the scalability. These clusters continue to scale to ever-increasing levels making the role very important. As an example, the "Ranger" system at the Texas Advanced Computing Center (TACC) includes over 60,000 cores with nearly 4000 InfiniBand ports. Previous work has shown that memory usage simply for connections when using the Reliable Connection (RC) transport of InfiniBand can reach hundreds of megabytes of memory per process at that level. To address these scalability problems a new InfiniBand transport, eXtended Reliable Connection, has been introduced.**

In this paper we describe XRC and design MPI over this new transport. We describe the variety of design choices that must be made as well as the various optimizations that XRC allows. We implement our designs and evaluate it on an InfiniBand cluster against RC-based designs. The memory scalability in terms of both connection memory and memory efficiency for communication buffers is evaluated for all of the configurations. Connection memory scalability evaluation shows a potential 100 times improvement over a similarly configured RC-based design. Evaluation using NAMD shows a 10% performance improvement for our XRC-based prototype for the `jac2000` benchmark.

***Index Terms*—ignore**

I. Introduction

Large-scale deployments of clusters designed from largely commodity-based components continue to be a major component of high-performance computing environments. A significant component of a high-performance cluster is the compute node interconnect. InfiniBand [1], is an interconnect of such systems that is enjoying wide success due to low latency (1.0-3.0μsec), high bandwidth and other features.

The Message Passing Interface (MPI) [2] is the dominant programming model for parallel scientific applications. As such, the MPI library design is crucial in supporting high-performance and scalable communication for applications on these large-scale clusters.

Clusters featuring the InfiniBand interconnect are continuing to scale. As an example, the "Ranger" system at the Texas Advanced Computing Center (TACC) includes over 60,000 cores with nearly 4000 InfiniBand ports [3]. By comparison, the first year an InfiniBand system appeared in the Top500 list of fastest supercomputers was in 2003 with a 128 node system at NCSA [4]. The latest list shows over 25% of systems are now using InfiniBand as the compute node interconnect.

Current implementations of MPI over InfiniBand, such as MVAPICH, Open MPI, HP MPI, and others, use the Reliable Connection (RC) transport of InfiniBand as the primary transport. Earlier work has shown, however, that the RC transport requires several KB of memory per connected peer, leading to significant memory usage at large-scale. MVAPICH now supports using the Unreliable Datagram (UD) transport for communication [5], however, implementing MPI over UD requires software-based segmentation, ordering and retransmission within the MPI library. Neither of these transports are ideal for MPI on large-scale InfiniBand clusters.

The latest InfiniBand cards from Mellanox include support for a new InfiniBand transport – eXtended Reliable Connection (XRC). The XRC transport attempts to give the same feature set of RC while providing additional scalability for multi-core clusters. Instead of requiring each process to have a connection to each other process in the cluster for full connectivity, XRC allows a single process to require only one connection per destination node. Given this capability, the connection memory required can potentially reduce by a factor equal to the number of cores per node, a potentially large degree as core counts continue to increase.

In this paper we design MPI over the XRC transport of InfiniBand and discuss the connection requirements and opportunities it offers. An implementation of our design is evaluated using standard MPI benchmarks and memory usage is also measured. Application benchmark evaluation shows a 10% speedup for the `jac2000` NAMD dataset over an RC-based implementation. Other benchmarks show increased memory scalability but near-equal performance using the XRC transport. For a 16 core per node cluster, XRC shows a nearly 100 times improvement in connection memory scalability over a similar RC-based implementation.

The remaining parts of the paper are organized as follows: In Section II we provide an overview of InfiniBand. In Section III we discuss previous and related work. The new XRC transport of InfiniBand is described in Section IV. We present our XRC designs in Section V. Evaluation and analysis of an implementation of our designs is covered in Section VI. Finally, conclusions and future work are presented in Section VII.

978-1-4244-2640-9/08/$25.00 © 2008 IEEE

II. INFINIBAND

InfiniBand is a processor and I/O interconnect based on open standards [1]. It was conceived as a high-speed, general-purpose I/O interconnect, and in recent years it has become a popular interconnect for high-performance computing to connect commodity machines in large clusters.

A. Communication Model

Communication in InfiniBand is accomplished using a Queue based model. Sending and receiving end-points have to establish a Queue Pair (QP) which consists of Send Queue (SQ) and Receive Queue (RQ). Send and receive work requests (WR) are then placed onto these queues for processing by InfiniBand network stack. Completion of these operations is indicated by InfiniBand lower layers by placing completed requests in the Completion Queue (CQ). To receive a message on a QP, a receive buffer must be posted to that QP. Buffers are consumed in a FIFO ordering.

There are two types of communication semantics in InfiniBand: channel and memory semantics. Channel semantics are send and receive operations that are common in traditional interfaces, such as sockets, where both sides must be aware of communication. Memory semantics are one-sided operations where one host can access memory from a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA). Remote write and read are both supported in InfiniBand. In addition, remote atomic operations are also supported. Both communication semantics require communication memory to be registered with InfiniBand hardware and pinned in memory. The registration operation involves informing the network-interface of the virtual to physical address translation of the communication memory. The pinning operation requires the operating system to mark the pages corresponding to the communication memory as non-swappable. Thus, communication memory stays locked in physical memory, and the network-interface can access it as desired.

B. Existing Transport Services

There are four transport modes defined by the InfiniBand specification: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC) and Unreliable Datagram (UD). Of these, RC, UC, and UD are required to be supported by Host Channel Adapters (HCAs) in the InfiniBand specification. RD is not required and is not available with current hardware. All transports provide a checksum verification.

Reliable Connection (RC) is the most popular transport service for implementing MPI over InfiniBand. As a connection-oriented service, a QP with RC transport must be dedicated to communicating with only one other QP. A process that communicates with N other peers must have at least N QPs created. The RC transport provides almost all the features available in InfiniBand, most notably reliable send/receive, RDMA and Atomic operations.

Unreliable Connection (UC) provides connection-oriented service with no guarantees of ordering or reliability. It does support RDMA write capabilities and sending messages larger than the MTU size. Being connection-oriented in nature, every communicating peer requires a separate QP. In regard to resources required, it is identical to RC, while not providing reliable service. Thus, it appears unattractive for implementing MPI over this transport.

C. Shared Receive Queues

Introduced in the InfiniBand 1.2 specification, Shared Receive Queues (SRQs) were added to help address scalability issues with InfiniBand memory usage. As noted earlier, in order to receive a message on a QP, a receive buffer must be posted in the Receive Queue (RQ) of that QP. To achieve high-performance, MPI implementations pre-post buffers to the RQ to accommodate unexpected messages.

When using the RC transport of InfiniBand, one QP is required per communicating peer. To prepost receives on each QP, however, can have very high memory requirements for communication buffers. To give an example, consider a fully-connected MPI job of 1K processes. Each process in the job will require 1K - 1 QPs, each with n buffers of size s posted to it. Given a conservative setting of $n = 5$ and $s = 8KB$, over 40MB of memory per process would be required simply for communication buffers that may not be used. Given that current InfiniBand clusters now reach 60K processes, maximum memory usage would potentially be over 2GB per process in that configuration.

Recognizing that such buffers could be pooled, SRQ support was added so instead of connecting a QP to a dedicated RQ, buffers could be shared across QPs. In this method, a smaller pool can be allocated and then refilled as needed instead of pre-posting on each connection.

Note that a QP can only be associated with one SRQ for RC and UD. So any channel traffic on a QP will consume a receive buffer from the attached SRQ. If another SRQ is desired instead, a second QP must be created.

III. RELATED WORK

Much recent research has focused on the scalability of MPI libraries over InfiniBand. Many of these research works focus on reducing communication buffer requirements by utilizing SRQ [6], [7]. In addition, the Reliable Connection memory utilization has also been studied previously [8]. Yu, et al. proposed a connection setup method where UD was used for the first sixteen messages before an RC connection was setup [9]; in this case RC was the primary transport and no tradeoffs were evaluated. Further, connection-less UD MPI designs have been proposed in [5]. The zero-copy protocol over UD was introduced in [10]. A hybrid approach that dynamically utilizes both the RC and the UD channels based on application communication pattern has also been designed [11].

Sur, et. al. previously evaluated ConnectX, which is the first HCA to support XRC [12], [13]. Other groups have expressed interest in providing XRC support in MPI, such as HP MPI and Open MPI [14]. However, there is no detailed study on

how this XRC implementation works, the associated design challenges and interactions with applications.

Recently Shipman, et al. in [15] presented a mechanism for efficient utilization of communication buffers using multiple SRQs for different data sizes. For example, a 500 byte message should only consume 512 bytes and a 1.5KB message should only consume 2KB. Current designs have used a single SRQ, where any message will consume a fixed size such as 8KB. This evaluation and design, however, was with the Reliable Connection (RC) transport of InfiniBand, not the XRC transport.

IV. eXTended Reliable Connection

In this section we describe the new eXtended Reliable Connection (XRC) transport for InfiniBand. We first start with the motivation for this new transport followed by the XRC connection model and addressing.

A. Motivation

The motivation for creating the XRC transport comes from the explosive growth in multi-core clusters. While node counts continue to increase, core counts are increasing at an even more rapid rate. The Sandia Thunderbird and TACC Ranger show this trend:

The Sandia Thunderbird [16] was introduced in 2006 with 4K compute nodes each with dual CPUs for a total of 8K processing cores. The TACC Ranger [3] was put into production in 2008 with nearly 4K compute nodes. Each compute node has four quad-core CPUs, for a total cluster size of nearly 64K processing cores. Each at the time of introduction were in the upper echelon of the fastest machines in the world.

Existing InfiniBand transports made no distinction between connecting a process (generally one per core for MPI) and connecting a node. Thus, the associated resource consumption increased directly in relation to the number of cores in the system. Earlier work [11] has shown that memory usage for the RC transport can reach hundreds of MB of memory/process at 16K processes.

To address this problem XRC was introduced. Instead of having a per-process cost, XRC was designed to allow a single connection from one process to an entire node. In doing so, the maximum number of connections (QPs) per process can grow with the number of nodes instead of the number of cores in the system.

B. Connection Model

XRC provides the services of the RC transport, but defines a very different connection model and method for determining data placement on the receiver in channel semantics.

When using the RC transport of InfiniBand, the connection model is purely based on processes. For one process to communicate with another over InfiniBand, each side must have a dedicated QP for the other. There is no distinction as to the node in terms of allowing communication.

Figure 1(a) shows a fully-connected job, with each node having two processes, each fully connected to the other processes on other nodes. To maintain full connectivity in a cluster with N nodes and C cores per node, each process must have $(N-1) \times C$ QPs created. In this figure and equation we do not account for intra-node IB connections since the focus of this paper is on MPI and libraries generally use a shared-memory channel for communication within a node instead of network loopback.

By contrast, XRC allows connection optimization based on the location of a process. Instead of being purely process-based, the node of the peer to connect to is now taken into account. Consider a situation where a process A on $host1$ wants to communicate with both process B and process C on $host2$. After A creates a QP with B, A is also now connected to process C. The addressing required is discussed in the next section. The additional complication here is that although A can now send to C, it is not reciprocal since C cannot send to A. To send a message, a process must have one XRC QP to the node of the destination process and in our example B has the QP to A (and can send to A). Thus, if C wants to send to A it would need to create a QP to A. Note, if C had a QP to a process D on the same node as A it would be able to communicate with A.

Figure 1(b) shows a fully-connected XRC job. Instead of requiring a new QP for each process, now each process needs to only have one QP per node to be fully connected. In the best case the number of QPs required for a fully-connected job in a cluster with N nodes and C cores per node, is only N QPs. This reduces the number of QPs required by a factor of C, which is significant as the number of cores per node continue to increase.

C. Addressing Method

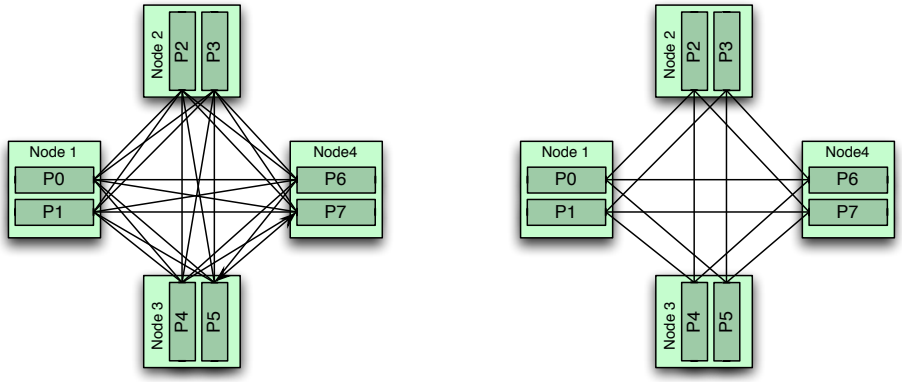
In the past, when using RC each process would communicate with a peer using a dedicated QP. Recall from Section II, there are two forms of communication semantics: channel and memory. In channel semantics the sender posts a send descriptor to the QP and the receive descriptor is consumed on the receive queue (RQ) connected to the QP. The sender does not know if the receiver is using an SRQ or a dedicated RQ. Thus, traditionally when using channel semantics the sender has no knowledge or control over the receive buffer.

XRC allows a more flexible form of placement on the receiver. When posting a send descriptor to an XRC QP, the destination *SRQ number* is specified. This allows a sender to specify a different "bucket" for different message sizes as suggested by Shipman, et. al., but without a separate QP.

This same addressing scheme is also what allows a process to communicate with other processes on the same node as one that it has a QP connection with. Only the SRQ number is needed for addressing, so as long as the SRQ number of the destination is known and at least one XRC QP is connected to a process on the node of the destination, a separate QP is not required.

V. Design

In this section we describe our MPI design using the new XRC transport. We first begin with an overview of the goals



(a) RC Connection Model

(b) XRC Connection Model

Fig. 1. InfiniBand Reliable Connection Models

in the design, and discuss issues related to the Shared Receive Queues (SRQs) possible connection setup methods.

The main goals of the design are two-fold: First, the design should reduce the memory consumption required for QPs and communication contexts. Second, the design should provide better communication buffer utilization. For example, this means that messages of 900 bytes should only consume a 1KB buffer instead of an 8KB buffer. This means we wish to reduce memory in two ways – both the connection memory as well as the communication buffer memory.

A. Shared Receive Queues

As noted earlier, previous work [15] has shown that communication buffers are not used efficiently when a single pool of receive buffers is used. Instead of using a single pool of buffers, multiple pools (SRQs) can be allocated. In this section we describe the possible configurations available by using the RC and XRC transports and multiple SRQs.

When using the RC transport this requires a QP for each SRQ available. Figure 2(a) shows the connection between two nodes, each with two processes. Despite increasing communication buffer efficiency, this requires a significant amount of connection context memory.

Using the XRC transport and the SRQ addressing scheme, a different QP is no longer required to have this same functionality of selecting a receive buffer pool based on the data size. This allows two different connection models:

- *Exclusive XRC (EXRC)*: In this model each process still creates a connection (QP) to every other process in the job if needed. There is no use of the XRC ability to connect to processes on the same node with an existing connection. The destination SRQ ability is used to eliminate the additional QPs required in the RC case. This model is shown in Figure 2(b).
- *Shared XRC (SXRC)*: Using this model both the additional QPs for multiple SRQs and for processes on the same node are eliminated. This is the method that makes the most of the XRC capabilities. Figure 2(c) shows this configuration.

Table I shows the number of QPs required using these difference schemes. Additional information on best-case and worst-case connection patterns is discussed below.

B. Connection Setup

As mentioned in Section IV, XRC allows one connection per node in the optimal case.

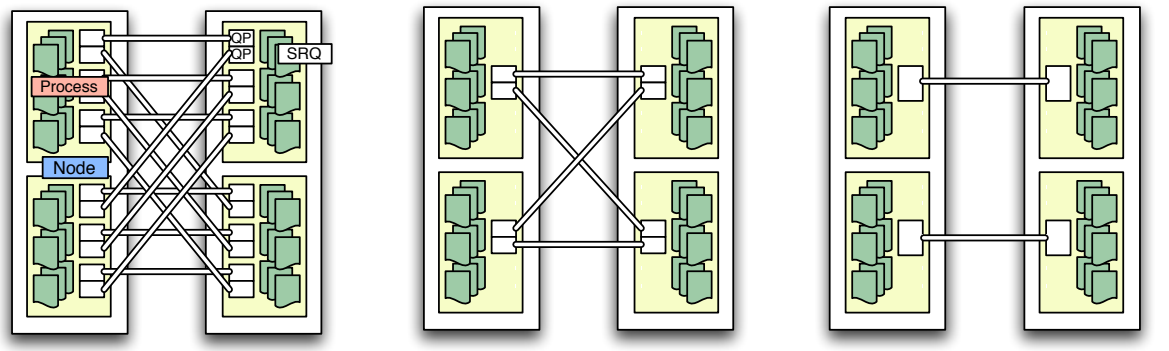
To achieve an optimal fully-connected connection pattern each process must have only a single connection to another node. In this paper, *fully-connected* means that all processes can send and receive from all other processes in the job.

Depending on the connection setup method an ideal setup or worst-case fully-connected pattern may emerge, as shown in Figure 3. We isolate two main issues that need to be addressed in a connection setup model:

- *Equal Layout*: Each node must have the same number of processes running on them. In other cases, such as 2 processes on node A and 4 on another node B, each of the two processes on node A will require 2 QPs in the best case since each process on B will require one QP to host A in order to send to that host. Clearly, there will be cases where a single connection will then not be possible.
- *Balanced Mapping*: Each process must connect with a peer that has not already created a connection with another process on its same node, otherwise the peer will create two connections to a single node.

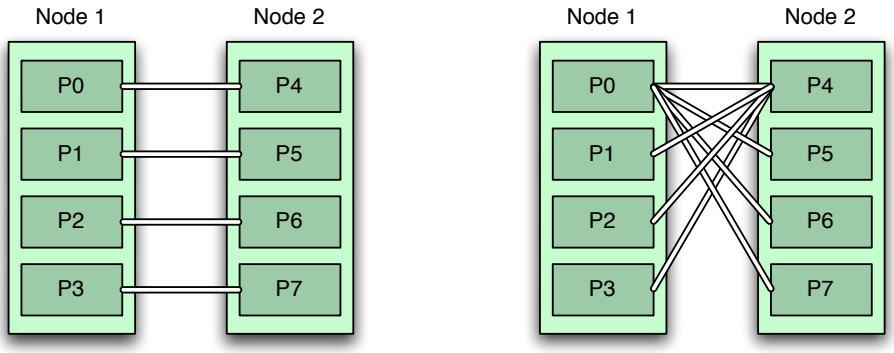
We propose three different connection models that are possible for an XRC-based MPI implementation and discuss their advantages and disadvantages:

Preconnect Method: If the job will require communication between all peers, connections can be setup at the beginning of the job. This is a static pre-connect method. In this case the optimal connection setup can be made assuming each node has the same number of processes. Even if there are non-equal numbers of nodes, the minimal number of connections can be created. This design has the problem that in many applications many processes do not directly communicate with every other



(a) RC-based Multi-SRQ (RC-MSRQ) (b) Exclusive XRC-based Multi-SRQ (EXRC-MSRQ) (c) Shared XRC-based Multi-SRQ (SXRC-MSRQ)

Fig. 2. Multi-SRQ Designs: Each figure shows two nodes, each with two processes in a fully-connected configuration. Each small white box denotes a QP.



(a) Ideal-Case Connection Setup (b) Worst-Case Connection Setup

Fig. 3. Depending on communication characteristics, connections may be created differently

process in the job. Preconnecting the connections can waste a significant amount of memory for a large job.

Predefined Method: In this alternative, the connections map is predefined (as in the preconnect method), so the minimal number of connections will be created for each process. The difference between the predefined and preconnect alternatives is that predefined is setup only as needed. The problem with such a design is that it will in many cases require a QP to be setup to a process that it doesn't need to communicate with. This process may also not be expecting any communication either and may be in a computation loop. Unless the connection setup can be done asynchronously, a deadlock could potentially occur.

On-Demand Method: In the on-demand method, the minimal connection map is not computed at all. Instead, whenever a process needs to send a message to a process on a node it doesn't have a connection to already, it sends a connect request to the process it needs to communicate with. In this way a non-minimal connection pattern may emerge. The pattern is dependent on the application.

VI. Experimental Evaluation

In this section we evaluate the designs we described in the previous section. We first start with a description of the experimental platform and methodology. Then we evaluate the memory usage and performance on microbenchmarks and application benchmarks.

A. Experimental Platform

Our experimental test bed is a 64-core ConnectX InfiniBand Linux cluster. Each of the 8 compute nodes has dual 2.33 GHz Intel Xeon "Clovertown" quad-core processors for a total of 8 cores per node. Each node has a Mellanox ConnectX DDR HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [17], OFED 1.3 release. The proposed designs are integrated into the MVAPICH-Aptus communication device of MVAPICH [18] previously designed in [11]. We extend the device to allow multiple RC QPs per peer and multiple SRQs. We also extend it to support the XRC transport in both the ESXRC and SXRC modes with any number of SRQs. MVAPICH is a popular open-source MPI implementation over InfiniBand and iWARP. It is based on MPICH [19] and MVICH [20].

All of the designs are implemented into the same code base and the same code flows. As a result, performance differences

can be attributed to the transport instead of software differences.

B. Methodology

We evaluate six different combinations:

- *Reliable Connection*: Using the RC transport with a single SRQ (*RC-SRQ*) as well as multiple SRQs (*RC-MSRQ*).
- *Exclusive eXtended Reliable Connection*: No sharing connections, but using XRC. Both single SRQ (*EXRC-SRQ*) and multiple SRQs (*EXRC-MSRQ*).
- *Shared eXtended Reliable Connection*: Share connections across nodes. This is using the on-demand connection setup from Section V. Both single (*SXRC-SRQ*) and multiple (*SXRC-MSRQ*) SRQ configurations

Table I shows a summary of the characteristics of each of these combinations where n is the number of nodes in the job and c is the number of cores per node. We assume for this table that processes are equally distributed. Note that RC-SRQ and EXRC-SRQ are equivalent in the amount of resources required as well as memory efficiency for communication buffers. The EXRC-SRQ case is a control to evaluate whether there are inherent performance differences between the XRC and RC hardware implementations and if the addressing method of XRC adds overhead.

In all of our evaluations we use the "on-demand" connection setup method for XRC. The other connection setup methods will have standard patterns. This method will provide the most insights.

For the multiple SRQ modes, we use 6 SRQs. We use the following sizes: 256 bytes, 512 bytes, 1KB, 2KB, 4KB, and 8KB. Messages above 8KB follow a zero-copy rendezvous protocol.

C. Memory Usage

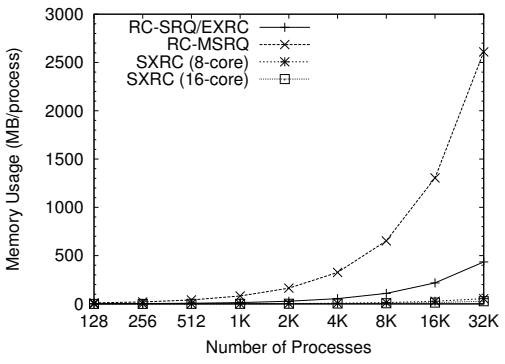


Fig. 4. Fully-Connected MPI Memory Usage

We first assess the scalability of each of the configuration. Figure 4 shows the memory usage when fully-connected. RC-SRQ is the default configuration for most MPIs, one connection per peer process. RC-MSRQ shows the memory usage when 6 SRQs are created per process and therefore the memory usage is six times higher than that of RC-SRQ. The last two lines are the memory usage for the Shared XRC implementations in the best case when in 8-core/node and 16-core/node configurations. EXRC has the same memory footprint as RC-SRQ since a single QP is required still to each process in the job.

From the figure we can observe that a fully-connected job at 32K processes will consume 2.6GB of memory with the RC-MSRQ configuration and 400 MB/process for the RC-SRQ and EXRC-{M}SRQ configurations. The SXRC designs reduce the memory usage in the best case to 54MB/process and 26MB/process for the 8-core and 16-core configurations, respectively. In the worst case the SXRC design will consume as much as the "RC-Single" model.

D. MPI Microbenchmarks

To assess if there are basic performance differences between the different combinations we ran various standard microbenchmarks. The basic latency, bandwidth, and bi-direction bandwidth results remained very similar across all combinations and are not presented here.

To further assess performance when many peers are being communicated with simultaneously we design a new microbenchmark. In this benchmark each process communicates with a variable number of random peers in the job during each iteration. In this throughput test each process sends and receives a message from 32 randomly selected peers 8 times. We ran this benchmark with 64 processes and report the results in Figure 5. From the figure we can see the SXRC mode is able to achieve higher throughput than the EXRC and RC configurations. In top-end bandwidth the XRC modes are able to outperform the RC mode.

E. Application Benchmarks

In this section we evaluate the configurations against two application-based benchmarks. These are more likely to model real-world use than microbenchmarks. We evaluate using the molecular dynamics application NAMD and the NAS Parallel Benchmarks (NPB). We evaluate all application benchmarks using 64 processes.

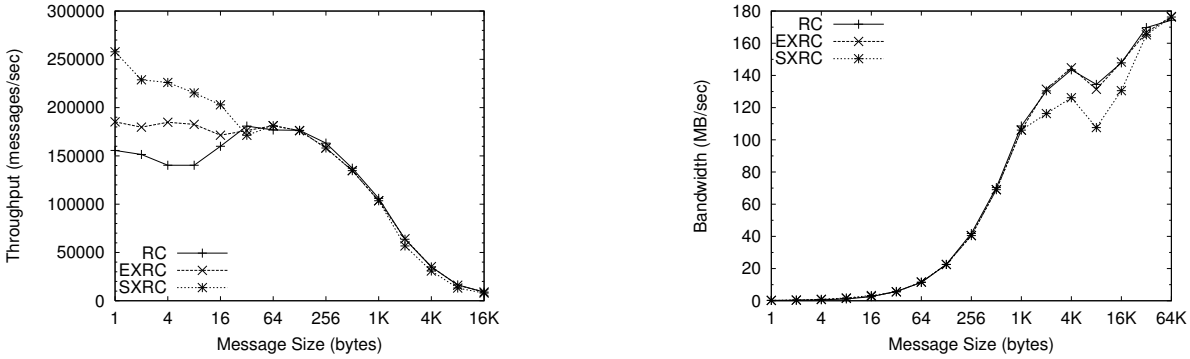
NAMD:

NAMD is a fully-featured, production molecular dynamics program for high performance simulation of large bimolecular systems [21]. NAMD is based on Charm++ parallel objects, which is a machine independent parallel programming system. Of the standard data sets available for use with NAMD, we use the `apoa1`, `f1atpase`, `er-gre`, and `jac2000` datasets.

Figure 6(a) shows the overall performance results of the different combinations on the various datasets. From the figure we observe that for both `apoa1` and `f1atpase` performance is very similar across all modes. For `jac` we see that the RC-MSRQ performs 11% worse than the standard RC-SRQ implementation. We believe this is due to HCA cache effects when large numbers of QPs are being used at the same time [11]. By contrast, we see that the SXRC modes provide

TABLE I
COMPARISON OF NUMBER OF QUEUE PAIRS FOR VARIOUS CHANNELS

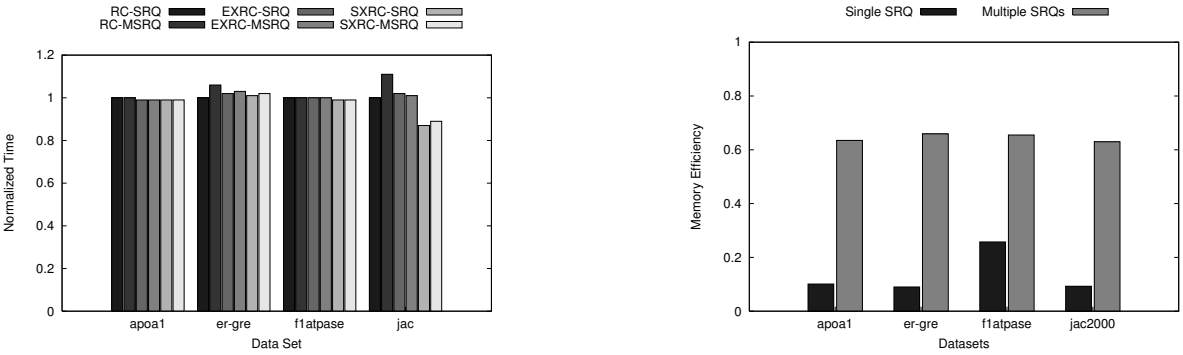
	Attributes			QPs per Process		QPs per Node	
	SRQs	Transport	Shared	Best Case	Worst Case	Best Case	Worst Case
RC-SRQ	1	RC	N	$n \times c$	$n \times c$	$n \times c^2$	$n \times c^2$
RC-MSRQ	6			$6 \times n \times c$	$6 \times n \times c$	$6 \times n \times c^2$	$6 \times n \times c^2$
EXRC-SRQ	1	XRC	N	$n \times c$	$n \times c$	$n \times c^2$	$n \times c^2$
EXRC-MSRQ	6						
SXRC-SRQ	1		Y	n	$n \times c$	$n \times c$	$2 \times n \times c$
SXRC-MSRQ	6						



(a) Message Rate

(b) Bandwidth

Fig. 5. Many-to-Many Benchmark Evaluation



(a) Normalized Time

(b) Memory Efficiency

Fig. 6. NAMD Evaluation

over 10% improvement. For the same reason as RC-MSRQ performs poorly, the SXRC modes perform well. Since a fewer number of QPs are used they are more likely to stay in the HCA cache. This mirrors what we observed in the many-to-many benchmark in Figure 5. We can see in Figure 6(b) that communication buffer usage is much improved when using multiple SRQs. This figure shows the ratio of the total amount of received messages to the total amount of memory in the communication buffers used to service those messages.

Table II shows the connection characteristics of each of the datasets. We can see that each of the datasets requires significant communication, especially `jac` where every process communicates with every other process. We observe for that dataset on the sum of the connections for the processes on a single node are only 82 as compared to 448 for RC-SRQ and EXRC modes. The RC-MSRQ configurations requires even more QPs, a total of 3136 QPs per node.

NAS Parallel Benchmarks:

The NAS Parallel Benchmarks [22] are a selection of ker-

nels that are typical in various Computational Fluid Dynamics (CFD) applications. As such, they are a good tool to evaluate the performance of the MPI library and parallel machines. In this evaluation the Class "C" benchmark size was used.

The performance results for each of the configurations are shown in Figure 7(a). Very little performance variation was observed in nearly all of the benchmarks. Only one benchmark, IS, showed a consistent improvement with the SXRC transport. The dominating factor in the IS benchmark performance is the `MPI_Alltoall` collective for large message sizes. For large message sizes the `MPI_Alltoall` collective sends directly to each process in the job. In the SXRC configurations, connections can be shared and can reduce connection thrashing in the HCA cache. There seems to be little difference between the SXRC-MSRQ and SXRC-SRQ modes in terms of performance.

Figure 7(b) shows the memory efficiency obtained by using multiple SRQs compared to a single SRQ. In all benchmarks efficiency was greatly increased. In the case of SP, efficiency rose from less than 6% to 75%. Using XRC we are able to acheive this buffer efficeincy as well as a reduction in connection memory.

The connection characteristics for the NAS benchmarks are shown in Table III. The benchmarks have a variety of patterns. IS, the benchmark where the SXRC modes outperformed, we notice that all connections are required. Using SXRC each process on average only needs to create 9.25 connections as apposed to 56 connections for the ESRQ and RC-SRQ modes (the shared memory channel is used for 7 others). The on-demand connection setup method seems to work well, although not always setting up the minimal number of connections.

VII. Conclusion

Clusters are continuing to scale to ever-increasing core counts. Node counts are increasing significantly, but much of the growth in core counts is coming from multi-core clusters. On large-scale clusters MPI is the primary programming model.

As such a significant part of the cluster environment and the high impact on application performance and scalability, the MPI library has an important role. It also must maintain a low resource footprint to allow applications to use as much node memory as possible. The Reliable Connection (RC) transport of InfiniBand has previously been show to scale poorly to very large-scale process counts, restricting the ability of the MPI library to remain scalable.

To address this scalability problem a new InfiniBand transport, eXtended Reliable Connection (XRC) has been introduced. Instead of requiring separate connections for each process, it allows connections to be a perl-node basis. This allows a reduction in memory usage by a factor of the number of cores per node.

In this paper we have designed MPI for this new transport and described the various opportunities it offers. We have implemented our design and evaluated them against RC-based implementations. Our study shows large gains in connection memory scalability. Also, using the SRQ addressing we are able to show increased communication buffer efficiency through multiple SRQs without sacrificing performance or memory. We additionally show a 10% improvement in the `jac2000` dataset for NAMD using the XRC based design.

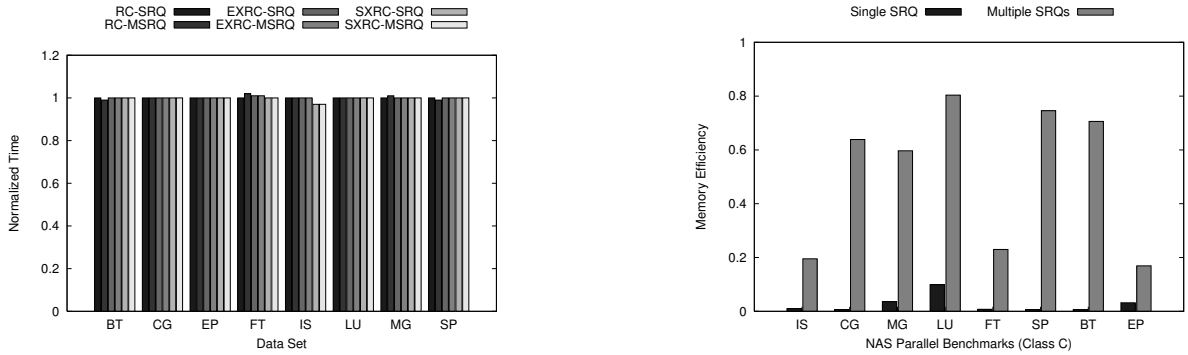
In the future we hope to evaluate our designs at a larger scale. At a larger scale we would like to compare the XRC based design with the hybrid RC-UD design proposed earlier. We would also like to explore dynamic creation of SRQs to fit the message sizes commonly used by the applications instead of the static SRQ buffer pools used currently.

Acknowledgment

This research is supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342 and #CCF-0702675; grant from Wright Center for Innovation #WCI04-010-OSU-0; grants from Intel, Mellanox, Cisco, and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, IBM, Appro, QLogic, and Sun Microsystems.

References

[1] InfiniBand Trade Association, "InfiniBand Architecture Specification," http://www.infinibandta.com.
[2] *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum, Mar 1994.
[3] Texas Advanced Computing Center, "HPC Systems," http://www.tacc.utexas.edu/resources/hpcsystems/.
[4] "TOP 500 Supercomputer Sites," http://www.top500.org.
[5] M. Koop, S. Sur, Q. Gao, and D. K. Panda, "High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters," in *21st ACM International Conference on Supercomputing (ICS07)*, Seattle, WA, June 2007.
[6] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda, "Shared Receive Queue Based Scalable MPI Design for InfiniBand Clusters," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
[7] G. Shipman, T. Woodall, R. Graham, and A. Maccabe, "Infiniband Scalability in Open MPI," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
[8] M. Koop, T. Jones, and D. K. Panda, "Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach," in *7th IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid07)*, Rio de Janeiro, Brazil, May 2007.
[9] W. Yu, Q. Gao, and D. K. Panda, "Adaptive Connection Management for Scalable MPI over InfiniBand," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
[10] M. Koop, S. Sur, and D. K. Panda, "Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram," in *IEEE Int'l Conference on Cluster Computing (Cluster 2007)*, September 2007.
[11] M. Koop, T. Jones, and D. K. Panda, "MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand," in *IEEE Int'l Parallel and Distributed Processing Symposium (IPDPS 2008)*, April 2008.
[12] S. Sur, M. Koop, L. Chai, and D. K. Panda, "Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms," in *15th IEEE Int'l Symposium on Hot Interconnects (HotI15)*, August 2007.
[13] Mellanox Technologies, "ConnectX Architecture," http://www.mellanox.com/products/connectx_architecture.php.
[14] The Open MPI Team, "Open MPI," http://www.open-mpi.org/.
[15] G. M. Shipman, R. Brightwell, B. Barrett, J. M. Squyres, and G. Bloch, "Investigations on infiniband: Efficient network buffer utilization at scale," in *Proceedings, Euro PVM/MPI*, Paris, France, October 2007.
[16] Sandia National Laboratories, "Thunderbird Linux Cluster," http://www.cs.sandia.gov/platforms/ Thunderbird.html.
[17] OpenFabrics Alliance, "OpenFabrics," http://www.openfabrics.org/.
[18] Network-Based Computing Laboratory, "MVAPICH: MPI over InfiniBand and iWARP," http://mvapich.cse.ohio-state.edu.



(a) Normalized Time

(b) Memory Efficiency

Fig. 7. NAS Parallel Benchmarks (Class C) Evaluation

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

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

[21] J. C. Phillips, G. Zheng, S. Kumar, and L. V. Kale, "NAMD: Biomolecular Simulation on Thousands of Processors," in *Supercomputing*, 2002.

[22] 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," vol. 5, no. 3, Fall 1991, pp. 63–73.

TABLE II
NAMD CHARACTERISTICS SUMMARY (PER PROCESS)

Benchmark	Configuration	Avg. Comm Peers	Max Comm Peers	Avg. QPs/process	Max QPs/processc	Avg QPs/node	Max QPs/nodee
apoa1	RC-MSRQ	54.41	63	285.54	336	2284.50	2340
	RC-SRQ			47.59	56	380.75	390
	EXRC-{M}SRQ			47.59	51	380.75	390
	XRC-{M}SRQ			9.09	33	72.75	80
f1atpase	RC-MSRQ	62.19	63	331.50	336	2652	2688
	RC-SRQ			55.25	56	442	448
	EXRC-{M}SRQ			55.25	56	442	448
	XRC-{M}SRQ			9.22	33	15	82
jac2000	RC-MSRQ	63	63	336	336	2688	2688
	RC-SRQ			56	56	448	448
	EXRC-{M}SRQ			56	56	448	448
	XRC-{M}SRQ			9.22	33	73.75	82
er-gre	RC-MSRQ	25.38	63	122.28	336	978	1188
	RC-SRQ			19.55	56	160.25	193
	EXRC-{M}SRQ			20.03	56	160.25	194
	XRC-{M}SRQ			8.28	33	66.25	79

TABLE III
NAS CHARACTERISTICS SUMMARY (PER PROCESS)

Benchmark	Configuration	Avg. Comm Peers	Max Comm Peers	Avg. QPs/process	Max QPs/processc	Avg QPs/node	Max QPs/nodee
IS	RC-MSRQ	63	63	336	336	2688	2688
	RC-SRQ			56	56	448	448
	EXRC-{M}SRQ			56	56	448	448
	XRC-{M}SRQ			9.25	25	74	74
CG	RC-MSRQ	6.88	7	23.28	24	186	186
	RC-SRQ			3.88	4	31	31
	EXRC-{M}SRQ			3.88	4	31	31
	XRC-{M}SRQ			3.50	4	28	28
MG	RC-MSRQ	9	9	30	30	240	240
	RC-SRQ			5	5	40	40
	EXRC-{M}SRQ			5	5	40	40
	XRC-{M}SRQ			4	4	32	32
LU	RC-MSRQ	7.50	8	22.50	24	180	192
	RC-SRQ			3.75	4	30	32
	EXRC-{M}SRQ			3.75	4	30	32
	XRC-{M}SRQ			3.75	4	30	32
FT	RC-MSRQ	63	63	336	336	2688	2688
	RC-SRQ			56	56	448	448
	EXRC-{M}SRQ			56	56	448	448
	XRC-{M}SRQ			7	7	56	56
SP	RC-MSRQ	10	10	36	36	288	288
	RC-SRQ			6	6	48	48
	EXRC-{M}SRQ			6	6	48	48
	XRC-{M}SRQ			4	4	32	32
BT	RC-MSRQ	10	10	36	36	288	288
	RC-SRQ			6	6	48	48
	EXRC-{M}SRQ			6	6	48	48
	XRC-{M}SRQ			4	4	32	32
EP	RC-MSRQ	6	6	18	18	144	144
	RC-SRQ			3	3	24	24
	EXRC-{M}SRQ			3	3	24	24
	XRC-{M}SRQ			3	3	24	24