

# X-SRQ - Improving Scalability and Performance of Multi-Core InfiniBand Clusters

Galen M. Shipman<sup>1</sup>, Stephen Poole<sup>1</sup>, Pavel Shamis<sup>2</sup>, and Ishai Rabinovitz<sup>2</sup>

<sup>1</sup> Oak Ridge National Laboratory \*, Oak Ridge, TN, USA,  
{gshipman, spooler}@ornl.gov,

<sup>2</sup> Mellanox Technologies, Yokneam, Israel,  
{pasha, ishai}@mellanox.co.il

**Abstract.** To improve the scalability of InfiniBand on large scale clusters Open MPI introduced a protocol known as B-SRQ [2]. This protocol was shown to provide much better memory utilization of send and receive buffers for a wide variety of benchmarks and real-world applications. Unfortunately B-SRQ increases the number of connections between communicating peers. While addressing one scalability problem of InfiniBand the protocol introduced another. To alleviate the connection scalability problem of the B-SRQ protocol a small enhancement to the reliable connection transport was requested which would allow multiple shared receive queues to be attached to a single reliable connection. This modified reliable connection transport is now known as the extended reliable connection transport.

X-SRQ is a new transport protocol in Open MPI based on B-SRQ which takes advantage of this improvement in connection scalability. This paper introduces the X-SRQ protocol and details the significantly improved scalability of the protocol over B-SRQ and its reduction of the memory footprint of connection state by as much as 2 orders of magnitude on large scale multi-core systems. In addition to improving scalability, performance of latency-sensitive collective operations are improved by up to 38% while significantly decreasing the variability of results. A detailed analysis of the improved memory scalability as well as the improved performance are discussed.

## 1 Introduction

The widespread availability of commodity multi-core CPUs from both Intel and AMD is changing the landscape of near-commodity clusters. Compute nodes with 8 cores (2 quad core CPUs) and even 16 cores (4 quad core CPUs) are becoming more common and 8 or more cores in a single socket are expected in the next 12-18 months. A number of these multi-core clusters are connected with InfiniBand (IB), thereby increasing the need to examine the scalability of MPI in such environments.

---

\* Research sponsored by the Mathematical, Information, and Computational Sciences Division, Office of Advanced Scientific Computing Research, U.S. Department of Energy, under Contract No. DE-AC05-00OR22725 with UT-Battelle, LLC.

Open MPI [1] supports the IB interconnect using the reliable connected (RC) transport layer. RC in IB currently requires a connection to be established between each communicating pair of processes and consumes one page (commonly  $4KB$ ) of system memory for each connection. Multi-core systems increase the number of dedicated processes per node and therefore increase the number of connections per node. This additional memory consumed on the node may be substantial in a large scale multi-core system. Furthermore, maintaining a fixed amount of memory per core is becoming increasingly difficult as memory prices remain high relative to the falling price of a CPU core. Pressure on memory will increase as applications are migrated to multi-core machines.

This paper describes Open MPI's use of the extended reliable connection (XRC) which alleviates some of the memory pressure in multi-core environments. In addition to reducing overall memory consumption in Open MPI, the use of XRC in conjunction with B-SRQ improves performance. This conjunction will be referred to as X-SRQ throughout this paper.

The rest of this paper is organized as follows. Section 2 provides a brief discussion of previous work in this area as well as an overview of the XRC architecture. Section 3 describes the new protocol, including necessary modifications to our on-demand connection wire-up scheme. Section 4 describes the test platform followed by performance analysis of the results. Section 5 summarizes relevant results and concludes with a discussion of areas of possible future work.

## 2 Background

The InfiniBand specification details 5 transport layers:

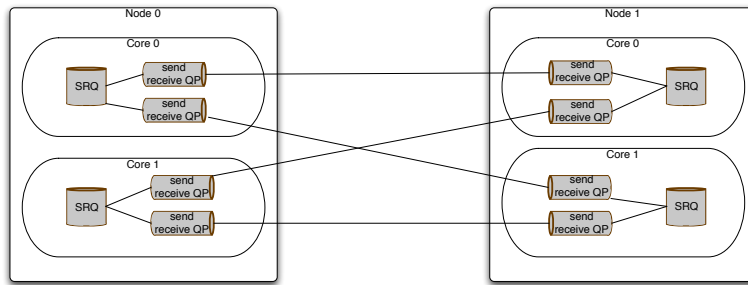
- 1) **Reliable Connection (RC):** connection-oriented and acknowledged
- 2) **Reliable Datagram (RD):** multiplexed and acknowledged
- 3) **Unreliable Connection (UC):** connection-oriented and unacknowledged
- 4) **Unreliable Datagram (UD):** connectionless and unacknowledged
- 5) **Raw Datagram:** connectionless and unacknowledged

RC provides a connection-oriented transport between two queue pairs (QPs). Work requests posted to a QP are implicitly addressed to the remote peer. The scalability limitations of connection-oriented transports are well known [2], [3] requiring  $\binom{N}{2}$  connections for  $N$  peers.

Fig. 1 illustrates two nodes, each with two cores connected via RC. In this example each core is running a single process and is connected to each of the remote processes. If we assume that shared memory is used for intra-node MPI communication then the total number of QPs is 4 per node.

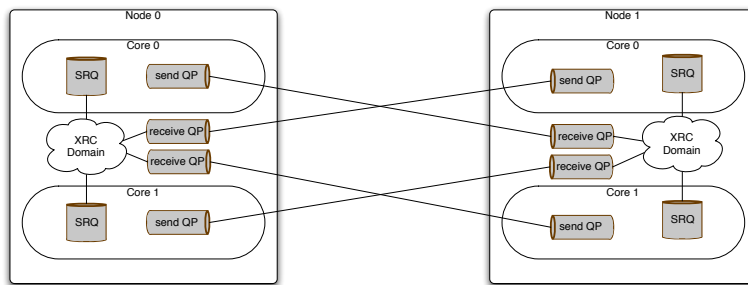
RD allows using a single QP to send and receive from any other addressable RD QP. RD was designed to provide a number of desirable scalability features but in practice RD has proven difficult to implement with no manufacturer currently supporting this transport layer.

While some are examining the use of UD to enhance scalability [4], the additional costs of user-level reliability and implementation complexity are still being examined.



**Fig. 1.** RC - 2 Nodes

To improve the scalability of InfiniBand in multi-core environments Mellanox has introduced XRC - the new transport layer. Requiring changes both in the HCA and in the software stack; XRC allows a single receive QP to be shared by multiple shared receive queues (SRQs) across one or more processes. Note that these SRQs can exist in any process which shares the same XRC domain as the receive QP. The SRQ “destination” is specified in the work queue entry (WQE) at the send QP. The receive QP consumes a buffer from the specified SRQ and enques a WQE to the completion queue (CQ) connected to this SRQ. This mechanism allows each process to maintain a single send QP to each host rather than to each remote process. A receive QP is established per remote send QP. These receive QPs can be shared among all the processes on the host.



**Fig. 2.** XRC - 2 Nodes

Fig. 2 illustrates two nodes with two processes per node (PPN) using XRC for communication. Note that each receive QP is a machine resource and exists within an XRC domain. SRQs are setup in each process and are connected to one or more receive QPs. Send QPs are a per process resource (as in RC), however each process can use a single send QP to communicate with any process on the remote machine by specifying the SRQ of the target process. If we assume that shared memory is used for intra-node MPI communication, then each node uses

4 QPs. In general, XRC can be used to reduce the number of QPs by a factor of  $PPN - 1$ . Thus for applications running  $P$  processes on  $N$  nodes, XRC decreases the number of required transport connections from  $P^2 * (N - 1)$  to  $P * 2 * (N - 1)$ . Note that the QPs in Fig. 2 are used either as send only QPs or as receive only QPs although they are currently implemented as bidirectional send/receive QPs. Work is ongoing within the OpenFabrics community to trim the QP size for the new XRC usage model.

To support the XRC hardware feature the OpenFabrics API defines two new APIs to create an XRC QP. The first API creates an XRC QP in userspace (just as in RC). This QP may be used for both send and receive operations. When using the QP for receive, a single process creates the QP and others share this by attaching SRQs to the QP. The same process which creates the QP must eventually destroy the QP, but it must defer the destruction of the QP until after all other processes have finished using the QP.

The second API creates a receive XRC QP at the kernel level. This allows a process to open the XRC QP and later exit without coordinating with other consumers of the QP. Each process wanting to use the XRC receive QP simply registers with the QP number which increments the internal reference count of the QP. When finished with the QP each process unregisters with the QP number which decrements the internal reference count. When the reference count drops to zero the QP is reclaimed. This method is used by Open MPI in order to support XRC with dynamic processes.

In previous work [5] Open MPI was enhanced to emulate the behavior of receive buffer pools [6] when using IB. “Buckets” of receive buffers of different sizes, with each bucket using a single SRQ, are created in each process. Each SRQ is then associated with a single QP. The sender can achieve better memory utilization on both sender and receiver sides by simply sending data on the QP with the minimum buffer of adequate size. This protocol was shown to significantly enhance memory utilization across a wide variety of applications and in some cases it enhance performance.

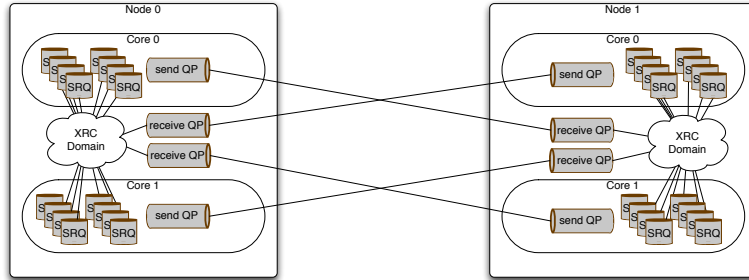
As part of this work a generic mechanism was created to allow the user or system administrator to specify any number of QPs (up to system limits) with various size buffers associated with them. Each QP can be specified to use either an SRQ or per-peer receive buffers posted to the QP directly. This mechanism allows for greater flexibility in tuning and experimentation.

Our current work involves modifying this generic mechanism to allow XRC QPs to be used in a similar fashion. Multiple SRQs can then be used to increase memory utilization without the need for creating a separate QP for each SRQ.

### 3 Protocol Description

One substantial drawback to the B-SRQ protocol is that each SRQ requires a separate QP. This limits the overall scalability of the protocol as node and core counts continue to increase. The X-SRQ protocol removes this limitation. Fig. 3 illustrates two nodes with two PPN using X-SRQ for communication. Note that

each process maintains a number of SRQs each with a different buffer size. These SRQs are then attached to a single node level receive QP per remote process.



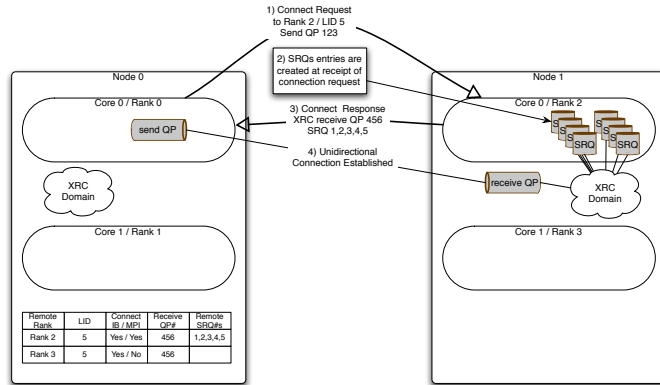
**Fig. 3.** X-SRQ - 2 Nodes

A number of changes in Open MPI were required in order to make use of XRC. Instead of addressing processes by simply using the implicit addressing of an RC QP, XRC requires specifying the remote SRQ number in the work request prior to enqueueing a send on an XRC QP. A few other minor changes were required such as extending our QP specification syntax to allow specifying XRC QPs via Open MPI's Modular Component Architecture parameter system [7].

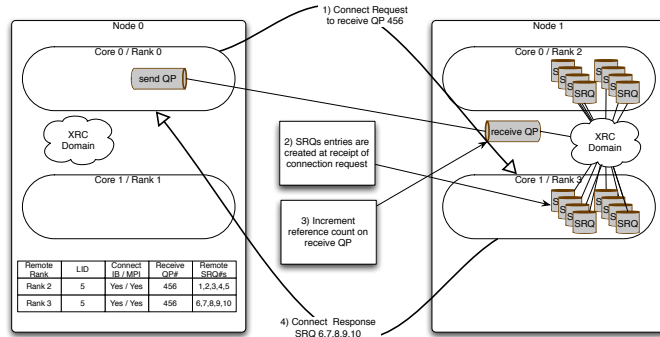
Of special interest is the connection establishment mechanism within Open MPI which supports XRC. The XRC QP establishment is considerably different from the usual RC QP wireup [3]. During process initialization, all processes perform the following:

1. exchange Local Identifiers (LIDs)
2. create an XRC domain
3. create SRQ entries

After the initialization phase each process keeps a table of LIDs for all remote processes and the connection status to the remote process. Fig. 4(a) illustrates a unidirectional connection establishment from process 0 to process 2. When process 0 initiates a send to process 2, process 0 checks the connection status to process 2. If the connection is closed then process 0 will create an XRC send QP (user level), and will send a connection request to process 0 (step 1 in Fig. 4(a)). Process 2 will create all the SRQ receive buffers as specified by the configuration (step 2 in Fig. 4(a)). Process 2 will then open an XRC receive QP (kernel level) and respond with an SRQ number for each SRQ as well as the XRC receive QP number (step 3 in Fig. 4(a)). Process 0 will receive the remote QP number and SRQ numbers and will connect the send QP to process 2 (step 4 in Fig. 4(a)). Process 0 will then update the connection status (both IB and MPI) of process 2 to "connected". Process 3's table entry is updated as well, but the MPI connection status remains "disconnected". At the end of the handshake there will be a unidirectional QP from process 0 on node 0 to process 2 on node 1.



(a) Process 0 to Process 2



(b) Process 0 to Process 3

**Fig. 4.** XSRQ Connection Establishment

Fig. 4(b) illustrates process 0 initiating a unidirectional connection to process 3 on node 1. Process 0 first checks the IB status and sees that process 0 already has an XRC send QP to node 1 (process 3). Process 0 then sends a connection request with an XRC receive QP number on node 1 to process 3 (step 1 in Fig. 4(b)). Process 3 creates all the SRQ receive buffers as specified by the configuration (step 2 in Fig. 4(b)). Process 3 will increase a reference counter on the XRC receive QP with the requested QP number (step 3 in Fig. 4(b)) and respond to process 0 with the SRQ numbers (step 4 in Fig. 4(b)). Process 0 receives the SRQ numbers and changes the connection status of process 3 to MPI “connected”.

Other aspects of the X-SRQ protocol remain similar to that of the B-SRQ protocol. As previously discussed, Open MPI supports mixing QPs with receive buffers posted directly (per-peer) on the QP with QPs using SRQs. This flexibility allows using flow-control mechanisms over the per-peer QP while using SRQs for scalability. Currently Open MPI does not support mixing XRC QPs with non-XRC QPs; this is left for future work.

## 4 Results

All experiments were conducted on a 32 node cluster located at Mellanox Technologies, USA. Each node consisted of dual quad-core Intel Xeon X5355 CPUs running at 2.66GHz with 8GB memory and a Mellanox ConnectX HCA running firmware 2.3.0. Each node ran Redhat Enterprise Linux with the 2.6.9-42 SMP kernel, OFED version 1.3 (release candidate 5) and Open MPI trunk subversion r17144. All nodes were connected via a DDR switch.

The most notable result was the significant reduction in the memory footprint of the Open MPI library when using multiple SRQs per process. X-SRQ has much better memory scaling characteristics than B-SRQ as the number of QPs is significantly smaller. The number of QPs for B-SRQ is governed by the following:  $PPN$  - number of processes per node;  $N$  - number of nodes;  $NSRQ$  - number of SRQs; and  $NQP$  - number of QPs.

For B-SRQ, the number of QPs created is

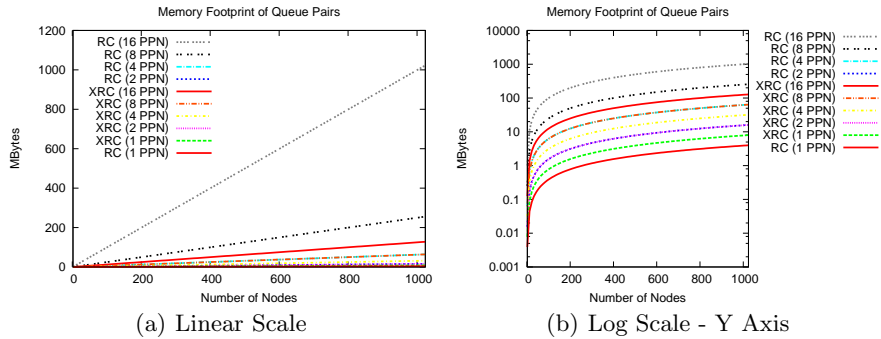
$$NQP = PPN^2 * NSRQ * (N - 1)$$

For X-SRQ, the number of QPs created is

$$NQP = PPN * 2 * (N - 1)$$

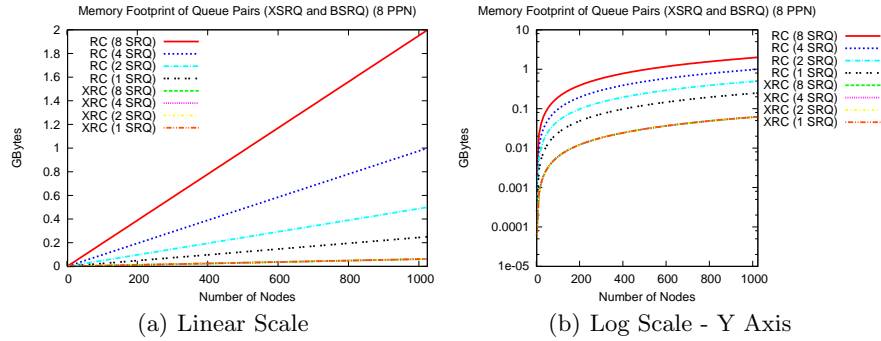
Note that for X-SRQ the  $NSRQ$  parameter is dropped. Instead of squaring  $PPN$ , we multiplied it by 2 to account for the separate send QP and receive QP. Fig. 5(a),5(b) illustrate the impact of increasing the number of processes per node - as is often the case for multi-core clusters. At 1024 nodes and 8 PPN, QP memory resources peak at 256MB when using RC as opposed to only 64MB for XRC.

Fig. 6(a),6(b) illustrate the impact of increasing the number of SRQs per process at 8 PPN. At 1024 nodes and 8 SRQs per process, QP memory resources peak at 2GB as opposed to only 64MB for XRC



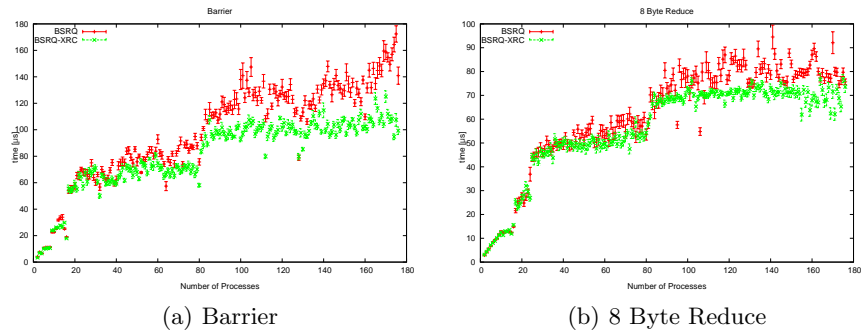
**Fig. 5.** Memory footprint of QPs (Varying PPN)]

In addition to significant memory savings, XRC improves performance. As the number of QPs is decreased, the HCA needs to manage fewer interconnect context resources. Consequently, the HCA is better able to cache context resource data and thereby to avoid a lookup on host memory. To evaluate the perfor-



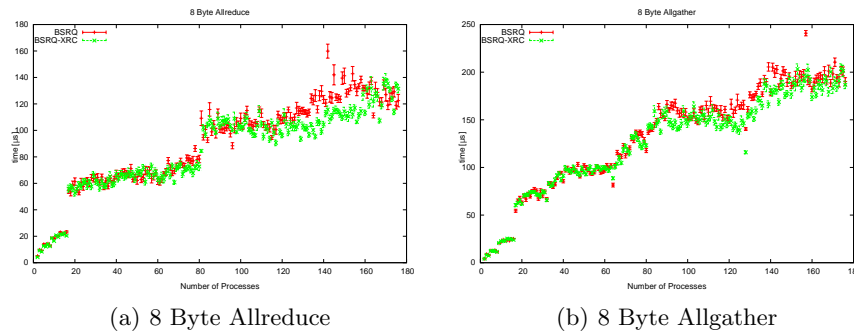
**Fig. 6.** Memory footprint of QPs (Varying SRQ Count)]

mance improvements of XRC, the SkaMPI collectives benchmarks [8] were used. MPI\_BARRIER, MPI\_REDUCE (8 bytes), MPI\_ALLREDUCE (8 bytes) and MPI\_ALLGATHER (8 bytes) collectives were chosen in order to evaluate the performance of X-SRQ. Overall performance of X-SRQ was better than that of B-SRQ. Fig. 7(a),7(b) illustrate that X-SRQ performance improvements reach up to 42% on the MPI\_BARRIER benchmark and up to 38% on the MPI\_REDUCE benchmark. The standard deviation of the results was also much lower for X-SRQ when compared to B-SRQ.



**Fig. 7.** Collective performance

Fig. 8(a), 8(b) also show some improvement of MPI\_ALLREDUCE and MPI\_ALLGATHER benchmarks at larger process counts, although not as significant as that of MPI\_BARRIER and MPI\_REDUCE benchmarks. Larger overheads for these collectives may minimize the overall performance improvement. The standard deviation of the results was again lower for X-SRQ compared to B-SRQ.



**Fig. 8.** Collective performance (continued)

## 5 Conclusions

Through a novel use of the XRC transport layer, both the scalability and performance of Open MPI have improved. The X-SRQ protocol improves both send and receive buffer utilization while significantly improving the scalability of QP connections. While not limited to multi-core systems, these scalability improvements are significant in larger multi-core environments. Current trends point towards increased core counts for the foreseeable future thereby making these scalability enhancements a necessity for clusters using IB.

In addition to improved scalability, X-SRQ improves performance on latency-sensitive operations due to more efficient use of network resources. These performance improvements are consistent with the HCA architecture and are relevant not only for larger clusters, but for any multi-core cluster (as small as 32 nodes) using InfiniBand.

Open MPI does not currently support the use of XRC QPs and RC QPs concurrently. Future work will involve allowing these different QP “types” to be used concurrently within a single Open MPI job.

## References

- [1] Gabriel, E., Fagg, G., Bosilca, G., Angskun, T., Squyres, J.J.D.J., Sahay, V., Kambadur, P., Barrett, B., Lumsdaine, A., Castain, R., Daniel, D., Graham, R., Woodall, T.: Open MPI: goals, concept, and design of a next generation MPI implementation. In: Proceedings, 11th European PVM/MPI Users’ Group Meeting. (2004)
- [2] Brightwell, R., Maccabe, A.B.: Scalability limitations of VIA-based technologies in supporting MPI. In: Proceedings of the Fourth MPI Developers’ and Users’ Conference. (2000)
- [3] Shipman, G.M., Woodall, T.S., Graham, R.L., Maccabe, A.B., Bridges, P.G.: Infini-band scalability in Open MPI. In: International Parallel and Distributed Processing Symposium (IPDPS’06). (2006)
- [4] Koop, M.J., Sur, S., Gao, Q., Panda, D.K.: High performance mpi design using unreliable datagram for ultra-scale infiniband clusters. In: ICS ’07: Proceedings of the 21st annual international conference on Supercomputing, New York, NY, USA, ACM (2007) 180–189

- [5] Shipman, G.M., Brightwell, R., Barrett, B., Squyres, J.M., Bloch, G.: Investigations on infiniband: Efficient network buffer utilization at scale. In: Proceedings, Euro PVM/MPI, Paris, France (2007)
- [6] Brightwell, R., Maccabe, A.B., Riesen, R.: Design, implementation, and performance of MPI on Portals 3.0. *International Journal of High Performance Computing Applications* **17**(1) (2003)
- [7] Squyres, J.M., Lumsdaine, A.: The component architecture of open MPI: Enabling third-party collective algorithms. In Getov, V., Kielmann, T., eds.: Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications, St. Malo, France, Springer (2004) 167–185
- [8] Reussner, R., Sanders, P., Prechelt, L., Müller, M.: Skampi: A detailed, accurate mpi benchmark. In: PVM/MPI. (1998) 52–59