Infiniband Scalability in Open MPI

Galen M. Shipman [1,2], Tim S. Woodall [1], Rich L. Graham [1],
Arthur B. Maccabe [2], and Patrick G. Bridges [2]

[1]Los Alamos National Laboratory
Advanced Computing Laboratory
Los Alamos, NM USA
{gshipman, twoodall, rgraham}@lanl.gov

[2]University of New Mexico
Dept. of Computer Science
Albuquerque, NM USA
{maccabe, bridges}@cs.unm.edu

Abstract

Infiniband is becoming an important interconnect technology in high performance computing. Recent efforts in large scale Infiniband deployments are raising scalability questions in the HPC community. Open MPI, a new open source implementation of the MPI standard targeted for production computing, provides several mechanisms to enhance Infiniband scalability. Initial comparisons with MVAPICH, the most widely used Infiniband MPI implementation, show similar performance but with much better scalability characteristics. Specifically, small message latency is improved by up to 10% in medium/large jobs and memory usage per host is reduced by as much as 300%. In addition, Open MPI provides predictable latency that is close to optimal without sacrificing bandwidth performance.

1 Introduction

High performance computing (HPC) systems are continuing a trend toward distributed memory clusters consisting of commodity components. Many of these systems make use of commodity or 'near' commodity interconnects including Myrinet [16], Quadrics [3], Gigabit Ethernet and, recently, Infiniband [1]. Infiniband (IB) is increasingly deployed in small to medium sized commodity clusters. It is IB's low price/performance qualities that has made it attractive to the HPC market.

Of the available distributed memory programming models, the Message Passing Interface (MPI) standard [15] is currently the most widely used. Several MPI implementations support Infiniband including Open MPI [9], MVAPICH [14], LA-MPI [10] and NCSA MPI [17]. However, there are concerns about the scalability of Infiniband for MPI applications, partially arising from the fact that Infiniband was initially developed as a general I/O fabric technology and not specifically targeted to HPC [5].

In this paper, we describe Open MPI's scalable support for Infiniband. In particular, Open MPI makes use of Infiniband features not currently used by other MPI/IB implementations, allowing Open MPI to scale more effectively than current implementations. We illustrate the scalability of Open MPI's Infiniband support through comparisons with the widely-used MVAPICH implementation, and show that Open MPI uses less memory and provides better latency than MVAPICH on medium/large-scale clusters.

The remainder of this paper is organized as follows. Section 2 presents a brief overview of the Open MPI general point-to-point message design. Next, section 3 discusses the Infiniband architecture including current limitations of the architecture. MVAPICH is discussed in section 4 including potential scalability issues relating to this implementation. Section 5 provides a detailed description of Infiniband support in Open MPI. Scalability and performance results are discussed in section 6, followed by conclusions and future work in section 7.

2 Open MPI

The Open MPI Project is a collaborative effort by Los Alamos National Lab, the Open Systems Laboratory at Indiana University, the Innovative Computing Laboratory at the University of Tennessee and the High Performance Computing Center at the University of Stuttgart (HLRS). The goal of this project is to develop a new implementation of the Message Passing Interface. Open MPI draws upon prior work on LA-MPI, LAM/MPI [18], FT-MPI [8] and PAX-MPI [12].

Open MPI is based on a Modular Component Architecture [19]. This architecture supports the runtime selection of components that are optimized for a specific operating environment. Multiple network interconnects are supported through this MCA. Currently there are two Infiniband components in Open MPI. One supporting the OpenIB Verbs-API and another supporting the Mellanox Verbs-API. In addition to being highly optimized for scalability these components provide a number of performance and scalability parameters which allow for easy tuning.

The Open MPI point-to-point (p2p) design and implementation is based on multiple MCA frameworks. These frameworks provide functional isolation with clearly defined interfaces. Figure 1 illustrates the p2p framework architecture.

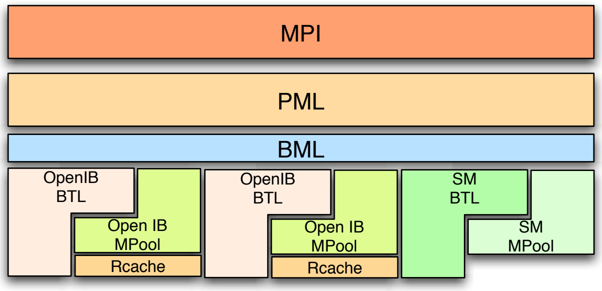

Figure 1. Open MPI p2p framework

As shown in Figure 1 the architecture consists of four layers. Working from the bottom up these layers are the Byte Transfer Layer (BTL), BTL Management Layer (BML), Point-to-Point Messaging Layer (PML) and the MPI layer. Each of these layers is implemented as an MCA framework. Other MCA frameworks shown are the Memory Pool (MPool) and the Registration Cache (Rcache). While these are illustrated and defined as layers, critical send/receive paths bypass the BML, as it is used primarily during initialization/BTL selection.

MPool The memory pool provides memory allocation/deallocation and registration/deregistration services. Infiniband requires memory to be registered (physical pages present and pinned) before send/receive or RDMA operations can use the memory as a source or target. Separating this functionality from other components allows the MPool to be shared among various layers. For example, MPI_ALLOC_MEM uses these MPools to register memory with available interconnects.

Rcache The registration cache allows memory pools to cache registered memory for later operations. When initialized, MPI message buffers are registered with the Mpool and cached via the Rcache. For example, during an MPI_SEND the source buffer is registered with the memory pool and this registration may be then be cached, depending on the protocol in use. During subsequent MPI_SEND operations the source buffer is checked against the Rcache, and if the registration exists the PML may RDMA the entire buffer in a single operation without incurring the high cost of registration.

BTL The BTL modules expose the underlying semantics of the network interconnect in a consistent form. BTLs expose a set of communication primitives appropriate for both send/receive and RDMA interfaces. The BTL is not aware of any MPI semantics; it simply moves a sequence of bytes (potentially non-contiguous) across the underlying transport. This simplicity will enable early adoption of novel network devices and encourages vendor support. There are several BTL modules currently available; including TCP, GM, Portals, Shared Memory (SM), Mellanox VAPI and OpenIB VAPI. In the later section we discusses the Mellanox VAPI and OpenIB VAPI BTLs.

BML The BML acts as a thin multi-plexing layer, allowing the BTLs to be shared among multiple upper layers. Discovery of peer resources is coordinated by the BML and cached for multiple consumers of the BTLs. After resource discovery, the BML layer may be safely bypassed by upper layers for performance. The current BML component is named R2.

PML The PML implements all logic for p2p MPI semantics including standard, buffered, ready, and synchronous communication modes. MPI message transfers are scheduled by the PML based on a specific policy. This policy incorporates BTL specific attributes to schedule MPI messages. Short and long message protocols are implemented within the PML. All control messages (ACK/NACK/MATCH) are also managed at the PML. The benefit of this structure is a separation of transport protocol from the underlying interconnects. This significantly reduces both code complexity and code redundancy enhancing maintainability. There are currently three PMLs available in the Open MPI code base. This paper discusses `OB1` the latest generation `PML` in the later section.

3 Infiniband

The Infiniband specification is published by the Infiniband Trade Association (ITA) and was originally proposed as a general I/O technology, allowing for a single I/O fabric to replace multiple existing fabrics. This goal has faded and currently Infiniband is targeted as an Inter Process Communication (IPC) and Storage Area Network (SAN) interconnect technology.

Infiniband, similar to Myrinet and Quadrics, provides both Remote Direct Memory Access (RDMA) and Operating System (OS) bypass facilities. RDMA enables data transfer from the address space of an application process to a peer process across the network fabric without requiring involvement of the host CPU. Infiniband RDMA operations support both two-sided send/receive and one-sided put/get semantics. Each of these operations may be queued from the user level directly to the host channel adapter (HCA) for execution, bypassing the OS to minimize latency and processing requirements on the host CPU.

3.1 Infiniband OS Bypass

To enable OS bypass, Infiniband defines the concept of a Queue Pair (QP). The Queue Pair mechanism provides user level processes direct access to the IB HCA. Unlike traditional stack based protocols, there is no need to packetize the source buffer or process other protocol specific messages in the OS or at user level. Packetization and transport logic is located almost entirely in the HCA.

Each queue pair consists of both a send and receive work queue, and is additionally associated with a Completion Queue (CQ). Work Queue Entries (WQEs) are posted from the user level for processing by the HCA. Upon completion of a WQE, the HCA posts an entry to the completion queue, allowing the user level process to poll and/or wait on the completion queue for events related to the queue pair.

Two-sided send/receive operations are initiated by enqueueing a send WQE on a QP's send queue. The WQE specifies only the senders local buffer. The remote process must pre-post a receive WQE on the corresponding receive queue which specifies a local buffer address to be used as the destination of the receive. Send completion indicates the send WQE is completed locally and results in a sender side CQ entry. When the transfer actually completes a CQ entry will be posted to the receivers CQ.

One-sided RDMA operations are likewise initiated by enqueueing a RDMA WQE on the Send Queue. However, this WQE specifies both the source and target virtual addresses along with a protection key for the remote buffer. Both the protection key and remote buffer address must be obtained by the initiator of the RDMA read/write prior to submitting the WQE and as such Infiniband does not provide true one-sided communication. Completion of the RDMA operation is local and results in a CQ entry at the initiator. The operation is one sided in the sense that the remote application is not involved in the request and does not receive notification of its completion.

3.2 Infiniband Resource Allocation

Infiniband does place some additional constraints on these operations. As data is moved directly between the host channel adapter (HCA) and user level source/destination buffers, these buffers must be registered with the HCA in advance of their use. Registration is a relatively expensive operation which locks the memory pages associated with the request, thereby preserving the virtual to physical mappings. Additionally, when supporting send/receive semantics, preposted receive buffers are consumed in order as data arrives on the host channel adapter (HCA). Since no attempt is made to match available buffers to the incoming message size, the maximum size of a message is constrained to the minimum size of the posted receive buffers.

Infiniband additionally defines the concept of a Shared Receive Queue (SRQ). A single SRQ may be associated with multiple QPs during their creation. Receive WQEs that are posted to the SRQ are then shared resources to all associated QPs. This capability plays a significant role in improving the scalability of the connection-oriented transport protocols described below.

3.3 Infiniband Transport Modes

The Infiniband specification details five modes of transport, Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC), Unreliable Datagram (UD) and Raw Datagram.

Reliable Connection provides a connection oriented transport between two queue pairs. During initialization of each QP, peers exchange addressing information used to bind the QP's and bring them to a connected state. Work requests posted on each QP's Send

Queue are implicitly addressed to the remote peer. As with any connection oriented protocol, scalability may be a concern as the number of connected peers grows large, and resources are allocated to each QP. Both Open MPI and MVAPICH currently use RC transport modes.

Reliable Datagram allows a single QP to be used to send and receive messages to/from other RD QPs. Whereas in RC reliability state is associated with the QP, RD associates this state with an end-to-end (EE) context. The intent of the Infiniband specification is that the EE's will scale much more effectively with the number of active peers. Both Reliable Connection and Reliable Datagram provide acknowledgment and retransmission. In practice, this portion of the specification has yet to be implemented.

Unreliable Connection and Unreliable Datagram are similar to their reliable counterparts in terms of QP resources. These transports differ in that they are unacknowledged services and do not provide for retransmission of dropped packets. The high cost of user reliability relative to the hardware reliability of RC and RD make these modes of transport inefficient for MPI.

3.4 Infiniband Summary

Infiniband shares many of the architectural features of VIA (Virtual Interface Architecture). Scalability limitations of VIA are well known [6] to the HPC community. These limitations arise from a connection oriented protocol, RDMA semantics and the lack of direct support for asynchronous progress. While the Infiniband specification does address scalability of connection oriented protocols through the RD transport mode this part of the specification has not been implemented and while the SRQ mechanism addresses scalability issues associated with the reliable connection oriented transport, issues related to flow control and resource management must be considered. MPI implementations must therefore compensate for these limitations in order to effectively scale to large clusters.

4 MVAPICH

MVAPICH is currently the most widely used MPI implementation on Infiniband platforms. A descendent of MPICH [11], one of the earliest MPI implementation, as well as MVICH [13], MVAPICH provides several novel features for Infiniband support. These features include small message RDMA, caching of registered memory regions and multi-rail IB support.

4.1 Small Message Transfer

The MVAPICH design incorporates a novel approach to small message transfer. Each peer is preallocated and registered a separate memory region for small message RDMA operations called a persistent buffer association. Each of these memory regions is structured as a circular buffer allowing the remote peer to RDMA directly into the currently available descriptor. Remote completion is detected by the peer polling the current descriptor in the persistent buffer association. A single bit can indicate completion of the RDMA as current Mellanox hardware guarantees the last byte of an RDMA operation will be the last byte delivered to the application. This design takes advantage of the extremely low latencies of Infiniband RDMA operations.

Unfortunately, this is not a scalable solution for small message transfer. As each peer requires a separate persistent buffer, memory usage grows linearly with the number of peers. Polling each persistent buffer for completion also presents scalability problems. As the number of peers increases the additional overhead required to poll these buffers quickly erodes the benefits of small message RDMA.

A similar design was attempted earlier on ASCI Blue Mountain with what later evolved into LA-MPI to support HIPPI-800. The approach was later abandoned due to poor scalability and a hybrid approach evolved, taking advantage of HIPPI-800 firmware for multiplexing. Other alternative approaches to polling persistent buffers for completion have also been discussed and may prove to be more scalable [4].

To address the issues of small message RDMA, MVAPICH provides a medium and large configuration option. These options limit the resources used for small message RDMA. As demonstrated in our results section this configuration option improves the scalability of small message latencies but still results in suboptimal performance as the number of peers increases.

4.2 Connection Management

MVAPICH uses static connection management, establishing a fully connected job at startup. In addition to eagerly establishing QP connections, MVAPICH also allocates a persistent buffer association for each peer. If send/receive is used instead of small message RDMA, MVAPICH allocates receive descriptors on a per QP basis instead of using the shared receive queue across QP's. This further increases resource allocation per peer.

4.3 Caching Registered Buffers

As discussed earlier Infiniband requires all memory to be registered (pinned) with the HCA. Memory registration is an expensive operation so MVAPICH caches memory registrations for later use. This allows subsequent message transfers to queue a single RDMA operation without paying any registration costs. This approach to registration assumes that the application will reuse buffers often in order to amortize the high cost of a single up front memory registration. For some applications this is a reasonable assumption.

A potential issue when caching memory registrations is that the application may free a cached memory region and then return the associated pages to the OS [1]. The application could later allocate another memory region and obtain the same virtual address as the previously freed buffer. Subsequent RDMA operations may use the cached registration but this registration may now contain incorrect virtual to physical mappings. RDMA operation may therefore use an unintentional memory region. In order to avoid this scenario MVAPICH forces the application to never release pages to the OS [2] and thereby preserving virtual to physical mappings. This approach may cause resource exhaustion as the OS can never reclaim physical pages.

5 Design of Open MPI

In this section we discuss Open MPI's support for Infiniband, including techniques to enhance scalability.

5.1 The OB1 PML Component

OB1 is the latest point-to-point management layer for Open MPI. OB1 replaces the previous generation PML - TEG [20]. The motivation for a new PML was driven by code complexity at the lower layers. Previously much of the MPI p2p semantics such as the short and long protocols were duplicated for each interconnect. This logic as well as RDMA specific protocol logic was moved up to the PML layer. Initially there was concern that moving this functionality into an upper layer would cause performance degradation. Preliminary performance benchmarks have shown this not to be the case. This restructuring has substantially decreased code complexity while maintaining performance on par with both previous Open MPI architectures as well as other MPI implementations. Through the use of device appropriate abstractions we have exposed the underlying architecture to the PML level. As such the overhead of the p2p architecture in Open MPI is lower than that of other MPI implementations.

OB1 provides numerous features to support both send/receive and RDMA read/write operations. The send/receive protocol uses pre-allocated/registered buffers to copy in for send and copy out for receive. This protocol provides good performance for small messages transfer and is used both for the eager protocol as well as control messages.

To support RDMA operations, OB1 makes uses of the Mpool and Rcache components in order to cache memory regions for later RDMA operations. Both source and target buffers must be registered prior to an RDMA read or write of the buffer. Subsequent RDMA operation can make use of pre-registered memory in the Mpool/Rcache. While MVAPICH prevents physical pages from being released to the OS, Open MPI instead uses memory hooks to intercept deallocation of memory. When memory is deallocated it is checked against the Rcache and all matching registrations are de-registered. This prevents future use of an invalid memory registration while allowing memory to be returned to the host operating system. Intercepting memory allocation operations introduces additional overhead but is minimized through the use the rcache's logarithmic query performance. Additional research into minimizing this overhead is ongoing.

5.2 The OpenIB and Mvapi BTLs

This section focuses on two BTL components, both of which support the Infiniband interconnect. These two components are called Mvapi, based on the Mellanox verbs API, and OpenIB, based on the OpenIB verbs API. Other than this difference the Mvapi and OpenIB BTL components are nearly identical. Two major goals drove the design and implementation of these BTL components, performance and scalability. The following details the scalability issues addressed in these components.

5.2.1 Connection Management

As detailed earlier, connection oriented protocols pose scaling challenges for larger clusters. In contrast to the static connection management strategy adopted by MVAPICH, Open MPI uses dynamic connection management. When one peer first initiates communication with another peer, the request is queued at the BTL layer. The BTL then establishes the connection through an out of band (OOB) channel. After connection establishment, queued sends are progressed to the

[1] Memory is returned via the `sbrk` function in UNIX and Linux.

[2] The `mallopt` function in UNIX and Linux prevents pages from being given back the OS.

peer. This results in a shorter startup time and a longer first message latency time for Infiniband communication. Resource usage reflects the actual communication patterns of the application and not the number of peers in the MPI job. As such, MPI codes with scalable communication patterns will require fewer resources. Currently connections are opened dynamically but not closed dynamically future work will address deallocating connections at runtime.

5.2.2 Small Message Transfer

Open MPI currently avoids scalability problems of small message RDMA transfer by using Infiniband's send/receive interface for small messages. In an MPI job with 64 nodes, instead of polling 64 preallocated memory regions for remote RDMA completion, Open MPI polls a single completion queue. Instead of preallocating 64 separate memory regions for RDMA operations, Open MPI will optionally post receive descriptors to the SRQ. Unfortunately, Infiniband does not support flow control when the SRQ is used. As such Open MPI provides a simple user level flow control mechanism. As demonstrated in our results, this mechanism is probabilistic and may result in retransmission under certain communication patterns and may require further analysis.

Open MPI's resource allocation scheme is detailed in the Figure 2. Per peer resources include 2 Reliable Connection QP's, one for High Priority transfers and one for Low Priority transfers. High priority QP's share a single Shared Receive Queue and Completion Queue as do low priority QP's. Receive descriptors are posted to the SRQ on demand. The number of receive descriptors posted to the SRQ is calculated using the following method:

$x = log_2(n) * k + b$

x	Number of Receive Descriptors to post
n	Number of peers in cluster
k	per peer scaling factor for number of Receive Descriptors to post
b	base number of Receive Descriptors to post

The high priority QP is for small control messages and any data sent eagerly to the peer. The low priority QP is for larger MPI level messages as well as all RDMA operations. Using two QP's allows Open MPI to maintain two sizes of receive descriptors, an eager size for the high priority QP and a maximum send size for the low priority QP. Ordering of messages is guaranteed by Open MPI's matching logic which supports out-of-order fragments. While this framework requires an additional QP per peer, we gain a finer grained control over receive descriptor memory usage. In addition, using two QPs allows us to exploit parallelism available in the HCA hardware [7].

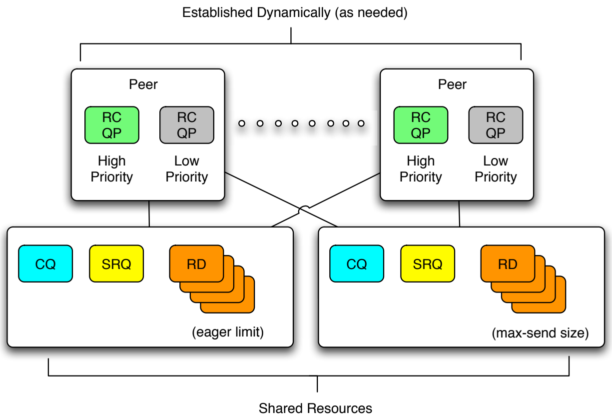

Figure 2. Open MPI Resource Allocation

6 Results

This section presents a comparison of our work. First we present scalability results in terms of per node resource allocation. Next we examine performance results, showing that while Open MPI is highly scalable it also provides excellent performance in the NAS Parallel benchmark (NPB) [2].

6.1 Scalability

Open MPI establishes connections dynamically on the first send to a peer. This allows resource allocation to reflect the communication pattern of the MPI application. Second, Open MPI optionally makes use of the Infiniband SRQ so that receive resources (preregistered memory) can be shared among multiple endpoints. As demonstrated earlier, the memory footprint of a pure RDMA protocol as used in MVAPICH increases linearly with the number of peers. This is partially due to lack of dynamic connection management as well as per peer resource allocation for the small RDMA protocol. As the number of peers increases this memory allocation scheme becomes intractable.

To examine memory usage of the MPI library we have used three different benchmarks. The first is a simple "hello world" application that does not communicate with any of its peers. This benchmark establishes a baseline of memory usage for an application. Figure 3 demonstrates that Open MPI's memory usage is constant as no connections are established and therefore no resources are allocated for other peers. MVAPICH on the other hand preallocates resources for each peer at startup so memory resources increase as the number of peers increases. Both MVAPICH small and medium configurations consume more resources.

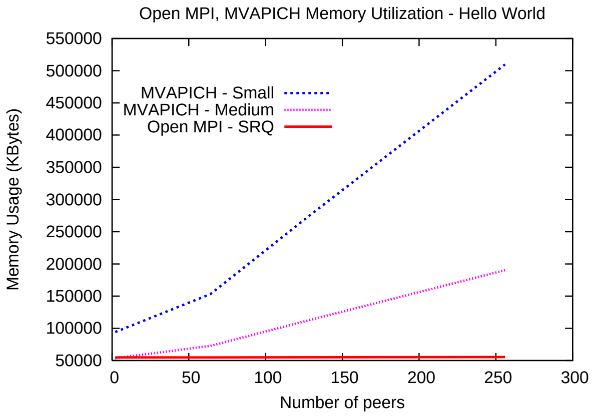

Figure 3. Hello World Memory Usage

Our next benchmark is a pairwise ping-pong, where peer's of neighbor rank ping each other, that is rank 0 pings rank 1 and rank 2 pings rank 3 and so on. As Figure 4 demonstrates, Open MPI memory usage is constant. This is due to dynamic connection management, only peers participating in communication are allocated resources. Again we see that MVAPICH memory usage ramps up with the number of peers.

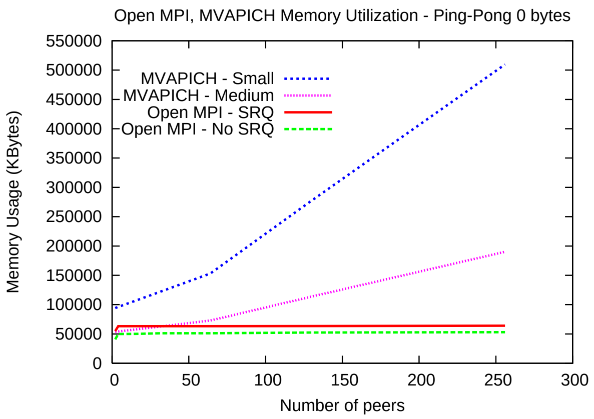

Figure 4. Pairwise Ping-Pong Memory Usage

Our final memory usage benchmark is a worst case for Open MPI, each peer communicates with every other peer. As can be seen in Figure 5 Open MPI SRQ memory usage does increase as the number of peers increases, but at a much smaller rate than that of MVAPICH. This is due to the use of the SRQ for resource allocation. Open MPI without SRQ scales slightly worse than the MVAPICH medium configuration, this is due to Open MPI's use of two QPs per peer.

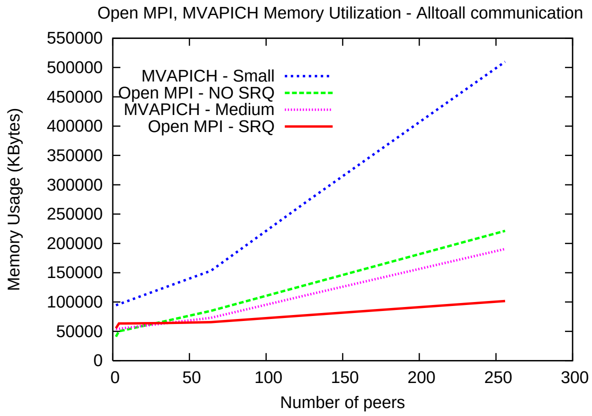

Figure 5. All-to-all Memory Usage

6.2 Performance

To verify the performance of our MPI implementation we present both micro benchmarks as well as the NAS Parallel Benchmarks

6.2.1 Latency

Ping-pong latency is a standard benchmark of MPI libraries. As with any micro-benchmark, ping-pong provides only part of the true representation of performance. Most ping-pong results are presented using two nodes involved in communication. While this number provides a lower bound on communication latency, multi-node ping-pong is more representative of communication patterns in anything but trivial applications. As such, we present ping-pong latencies for a varying number of nodes in which N nodes perform the previously discussed pair-wise ping-pong. This enhancement to the ping-pong benchmark helps to demonstrate scalability of small message transfers because in larger MPI jobs the number of peers communicating at the same time often increases.

In this test, the latency of a zero byte message is measured for each pair of peers. We have then plotted the average with error bars for each of these runs. As can be seen in Figure 6, the small message RDMA mechanisms provided in MVAPICH provides a benefit with a small number of peers. Unfortunately, the polling of memory regions is not a scalable architecture as can be seen when the number of peers participating in the latency benchmark increases. For each additional peer involved in the benchmark, every other peer must allocate and poll an additional memory region. Costs of polling quickly erode any improvements in latency. Memory usage is also higher on a per peer and aggregate basis. This trend occurs in both small and medium MVAPICH configurations. Open MPI provides much more predictable latencies and outperforms MVAPICH latencies as the number of peers increases. Open MPI - SRQ latencies are a bit higher than Open MPI - No SRQ latencies as the SRQ path under Mellanox HCAs is more costly.

The following Table 1 shows the Open MPI send/receive latencies trail MVAPICH small message RDMA latencies but are better than MVAPICH send/receive latencies. This is an important result as larger MVAPICH clusters will make more use of send/receive and not small message RDMA.

	Average Latency
Open MPI - Optimized	5.64
Open MPI - Default	5.94
MVAPICH - RDMA	4.19
MVAPICH - Send/Receive	6.51

Table 1. Two node Ping-Pong latency in μ-sec. Optimized - Limits the number of WQE on the RQ Defaults - Default number of WQE on the RQ

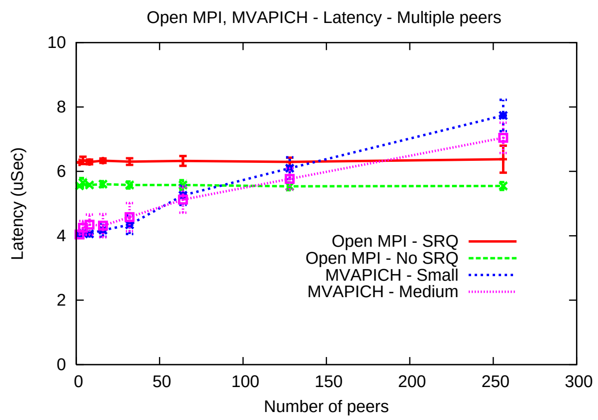

Figure 6. Multi-Node Zero Byte Latency

6.2.2 NPB

To demonstrate the performance of our implementation outside of micro benchmarks we used the NAS Parallel Benchmarks [2]. NPB is a set of benchmarks derived from computational fluid dynamics applications. All NPB benchmarks were run using the class C size of problem and all results are given in run-time (seconds). The results of these benchmarks are summarized in Table 2. Open MPI was run using 3 configurations, with SRQ, SRQ with simple flow control and without SRQ. MVAPICH was run in both small and medium cluster configurations. Open MPI without SRQ and MVAPICH performance is similar. With SRQ, Open MPI performance is similar for the BT, CG, and EP benchmarks. BT, FT and IS performance is lower with SRQ as receive resources are quickly consumed in collective operations. Our current flow control mechanism addresses this issue for the BT benchmark but both the FT and IS benchmarks are still effected. Table 3 demonstrates that by using Open MPI's tuned collective framework overall performance improves and SRQ performance improves dramatically for the FT benchmark. It should be noted that only 128 nodes were available at the time of this benchmark. Further research into SRQ flow control techniques are ongoing.

6.3 Experimental Setup

Our experiments were performed on two different machine configurations. Two node ping-pong benchmarks were performed on dual Intel Xeon X86-64 3.2 GHz processors with 2GB of RAM, and Mellanox PCI-Express Lion-Cub adapters connected via a Voltair 9288 switch. The Operating System is Linux 2.6.13.2 with Open MPI pre-release 1.0 and MVAPICH 0.9.5-118. All other benchmarks were performed on a 256 node cluster consisting of dual Intel Xeon X86-64 3.4 GHz processors with a minimum 6GB of RAM, Mellanox PCI-Express Lion Cub adapters also connected via a Voltair switch. The Operating System is Linux 2.6.9-11 with Open MPI pre-release 1.0 and MVAPICH 0.9.5-118.

7 Future Work - Conclusions

Open MPI addresses many of the concerns regarding the scalability and use of Infiniband in HPC. In this section we summarize the results of this paper and provide directions for future work.

7.1 Conclusions

Open MPI's Infiniband support provides several techniques to improve scalability. Dynamic connection management allows per peer resource usage to reflect the applications chosen communication pattern,

	BT		CG				EP			
Nodes	64	256	32	64	128	256	32	64	128	256
Open MPI - No SRQ	100.03	25.03	20.17	12.74	7.39	5.56	38.89	19.84	9.95	5.11
Open MPI - SRQ	114.92	26.92	20.45	12.86	7.49	5.61	38.85	19.72	10.04	5.26
Open MPI - SRQ FC	100.13	25.33	21.13	12.83	7.38	5.63	39.10	19.76	12.88	5.12
MVAPICH - Small	98.78	27.40	20.33	12.96	7.84	6.11	39.15	19.65	10.02	5.32
MVAPICH - Large	99.22	27.58	20.24	13.15	7.83	6.09	39.10	19.59	9.89	5.31

	SP		FT				IS			
Nodes	64	256	32	64	128	256	32	64	128	256
Open MPI - No SRQ	54.39	16.08	36.64	18.28	9.39	4.81	2.23	1.62	0.97	0.52
Open MPI - SRQ	140.81	22.53	75.48	68.36	56.92	26.96	32.21	33.29	25.06	21.97
Open MPI - SRQ FC	54.90	14.61	54.81	35.87	19.39	24.54	5.32	4.38	12.35	11.12
MVAPICH - Small	53.66	15.16	37.59	19.42	10.17	4.84	2.19	1.55	0.87	0.42
MVAPICH - Large	53.87	15.84	37.91	19.51	9.85	4.88	2.20	1.56	0.87	0.50

Table 2. NPB Results - Each benchmark uses the class C option with a varying number of nodes, 1 process per node. Results are given in seconds.

	BT	CG			EP		
Nodes	64	32	64	128	32	64	128
Open MPI - No SRQ	101.03	19.83	15.50	9.37	41.99	19.89	10.16
Open MPI - SRQ	100.92	19.81	15.54	9.37	39.00	22.57	10.49
Open MPI - SRQ FC	101.11	19.84	15.64	9.50	38.99	20.13	13.43
MVAPICH - Small	99.76	20.11	17.76	10.73	38.75	19.37	10.06
MVAPICH - Large	99.96	20.14	17.76	10.73	38.79	19.39	10.07

	SP	FT			IS		
Nodes	64	32	64	128	32	64	128
Open MPI - No SRQ	56.76	36.63	20.14	10.25	2.25	1.75	1.06
Open MPI - SRQ	56.51	36.57	19.80	10.42	13.00	1.76	11.68
Open MPI - SRQ FC	56.39	36.55	19.78	11.21	3.26	1.83	11.10
MVAPICH - Small	56.26	37.66	22.31	11.35	2.23	1.71	0.93
MVAPICH - Large	56.28	37.64	22.39	11.33	2.22	1.72	0.93

Table 3. NPB Results (With tuned collectives) - 256 Nodes not available at time of benchmark.

thereby allowing scalable MPI codes to preserve resources. Per peer memory usage in these types of applications will be significantly less in Open MPI when compared to other MPI implementations which lack this feature. Shared resource allocation scales much more effectively than per peer resource allocation through the use of the Infiniband Shared Receive Queue (SRQ). This should allow even fully connected applications to scale to a much higher level.

7.2 Future work

This work has identified additional areas for improvement. As the NAS parallel benchmarks illustrated, there are concerns regarding the SRQ case that require further consideration. Preliminary results indicate that an effective flow control and/or resource replacement policy must be implemented, as resource exhaustion results in significant performance degradation.

Additionally, Open MPI currently utilizes an OOB communication channel for connection establishment, which is based on TCP/IP. Using an OOB channel based on the unreliable datagram protocol will decrease first message latency and potentially improve the performance of the Open MPI run-time environment.

While connections are established dynamically, once opened, all connections are persistent. Some MPI codes which randomly communicate with peers may experience high resource usage even if communication with the peer is infrequent. For these types of applications, dynamic connection tear down may be beneficial.

Acknowledgments

The authors would like too thank Kurt Ferreira of UNM and Jeff Squyres and Brian Barrett of IU for comments and feedback on early versions of this paper. This material is based upon work supported by Subcontract No. 12783-001-05 49 issued to Rice University from the Regents of the University of California (Los Alamos National Laboratory). Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36. Project support was provided through ASC/PSE and ASC/S&CS programs. LA-UR-06-0465.

References

[1] I. T. Association. Infiniband architecture specification vol 1. release 1.2, 2004.

[2] Bailey, Barszcz, Barton, Browning, Carter, Dagum, Fatoohi, Fineberg, Frederickson, Lasinski, Schreiber, Simon, Venkatakrishnan, and Weeratunga. NAS parallel benchmarks, 1994.

[3] J. Beecroft, D. Addison, F. Petrini, and M. McLaren. QsNetII: An interconnect for supercomputing applications, 2003.

[4] R. Brightwell. A new MPI implementation for cray SHMEM. In *PVM/MPI*, pages 122–130, 2004.

[5] R. Brightwell, D. Doerfler, and K. Underwood. A comparison of 4x infiniband and quadrics elan-4 technologies. In *Proceedings of 2004 IEEE International Conference on Cluster Computing*, pages 193–204, September 2004.

[6] R. Brightwell and A. Maccabe. Scalability limitations of VIA-based technologies in supporting MPI. In *Proceedings of the Fourth MPI Developer's and User's Conference*, March 2000.

[7] V. V. et al. Programming the infiniband network architecture for high performance message passing systems. In *Proceedings of The 16th IASTED International Conference on Parallel and Distributed Computing and Systems*, 2004.

[8] G. E. Fagg, A. Bukovsky, and J. J. Dongarra. HARNESS and fault tolerant MPI. *Parallel Computing*, 27:1479–1496, 2001.

[9] E. Garbriel, G. Fagg, G. Bosilica, T. Angskun, J. J. D. J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.

[10] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4), August 2003.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[12] R. Keller, E. Gabriel, B. Krammer, M. S. Mueller, and M. M. Resch. Towards efficient execution of MPI applications on the grid: porting and optimization issues. *Journal of Grid Computing*, 1:133–149, 2003.

[13] L. B. N. Laboratory. Mvich: Mpi for virtual interface architecture, August 2001.

[14] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High performance RDMA-based MPI implementation over infiniband. In *ICS '03: Proceedings of the 17th annual international conference on Supercomputing*, pages 295–304, New York, NY, USA, 2003. ACM Press.

[15] Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.

[16] Myricom. Myrinet-on-VME protocol specification.

[17] S. Pakin and A. Pant. VMI 2.0: A dynamically reconfigurable messaging layer for availability, usability, and management. In *Proceedings of The 8th International Symposium on High Performance Computer Architecture (HPCA-8)*, Cambridge, MA, February 2002.

[18] J. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, Venice, Italy, September / October 2003. Springer-Verlag.

[19] J. M. Squyres and A. Lumsdaine. The component architecture of open MPI: Enabling third-party collective algorithms. In V. Getov and T. Kielmann, editors, *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*, pages 167–185, St. Malo, France, July 2004. Springer.

[20] T. Woodall, R. Graham, R. Castain, D. Daniel, M. Sukalsi, G. Fagg, E. Garbriel, G. Bosilica, T. Angskun, J. J. Dongarra, J. Squyres, V. Sahay, P. Kambadur, B. Barrett, and A. Lumsdaine. Open MPI's TEG point-to-point communications methodology : Comparison to existing implementations. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.