High-Performance and Scalable MPI over InfiniBand With Reduced Memory Usage: An In-Depth Performance Analysis

SAYANTAN SUR, MATTHEW J. KOOP AND D. K. PANDA

Technical Report
OSU-CISRC-5/06-TR51

High-Performance and Scalable MPI over InfiniBand With Reduced Memory Usage: An In-Depth Performance Analysis *

Sayantan Sur Matthew J. Koop Dhabaleswar K. Panda

Network-Based Computing Laboratory
Department of Computer Science and Engineering
The Ohio State University
{surs, koop, panda}@cse.ohio-state.edu

Abstract

InfiniBand is an emerging HPC interconnect being deployed in very large scale clusters, with even larger InfiniBand-based clusters planned. The Message Passing Interface (MPI) is the programming model of choice for end applications running on these large-scale clusters. Thus, it is very critical for the MPI implementation used to be based on a scalable and high-performance design. We analyze the performance and scalability aspects of MVAPICH, a popular open-source MPI implementation on InfiniBand, from an application standpoint. We analyze the performance and memory requirements of the MPI library while executing several well-known applications and benchmarks, such as NAS, SuperLU, NAMD, and HPL on a 64-node InfiniBand cluster. Our analysis reveals that for NAS Benchmarks, NAMD and SuperLU on 64 processes, the latest design of MVAPICH requires an order of magnitude less internal MPI memory on average per process and yet delivers the best possible performance.

1. Introduction

Cluster computing has become quite popular during the past decade. The interconnect used in these clusters is very crucial for attaining the highest possible performance. InfiniBand [6] is an emerging high-performance interconnect, offering low latency (1.5-3.0 microseconds) and high bandwidth (multiple Gigabytes/second). In addition to high-performance, InfiniBand also provides many advanced features like Remote Direct Memory Access (RDMA), atomic operations, multicast, and QoS. As InfiniBand gains popularity, large scale clusters, such as the 8000-processor Sandia Thunderbird [12, 17] and NASA/Ames Columbia [12], are using it as their primary interconnect. Clusters of several tens-of-thousands of nodes have now appeared as the most powerful machines in the Top 500 list [22]. Accordingly, it is expected that the scale of InfiniBand clusters to be deployed in the near future will be even larger. MPI [11] is the de-facto standard in writing parallel scientific applications. Hence, a scalable and high performance MPI design is very critical for end HPC applications, which will run on these modern and next generation very large scale clusters.

MVAPICH [13] is an popular open-source high-performance and scalable implementation of MPI over InfiniBand. It is used by over 340 organizations spread over 33 countries. It has enabled several clusters, including the Sandia Thunderbird [17], to achieve high rankings in the Top 500 list. MVAPICH is also available from the OpenIB/Gen2 [14] stack in an integrated manner. It implements the Abstract Device Interface of MPICH [5] and was derived from MVICH [8].

MVAPICH provides various designs to perform message passing [10, 20]. Depending upon the requirement of the end MPI application and available InfiniBand hardware, different designs may be chosen by the user. In addition, all these designs are runtime tunable with various parameters. Most of these parameters are "hints" to the MPI library of the user's intentions. These parameters directly affect the memory usage and other characteristics of the MPI library. Using these parameters, the MPI library allocates internal buffers that are used for communication. In addition, depending on the requirements of the application, more memory may be allocated during its actual execution. These communication buffers represent the majority of the memory consumption of the

* This research is supported in part by Department of Energy's grant #DE-FC02-01ER25506, National Science Foundation's grants #CNS-0403342 and #CNS-0509452; grants from Intel, Mellanox, Cisco and Sun Microsystems; Equipment donations from Intel, Mellanox, AMD, IBM, Apple, Appro, Microway, PathScale, Silverstorm and Sun Microsystems.

MPI library. Allocating more buffers may allow the library to offer better communication performance. On the other hand, lack of buffers may lead to runtime allocation and management of required memory (which is costly) and hence degradation of end application performance. Thus, the following two questions are of great significance to MPI library designers, cluster system vendors, and the end users:

1. Does aggressively reducing communication buffer memory lead to degradation of end application performance?
2. How much memory can we expect the MPI library to consume during execution of a typical application, while still providing the best available performance?

To the best of our knowledge, there has been no contemporary study that comprehensively answers these questions. In this paper, we provide answers to the above two questions by analyzing the internal MPI operations during execution of well known MPI applications and benchmarks such as NAS Parallel Benchmarks [2], SuperLU [23], NAMD [16], and HPL [4]. Our analysis reveals that for the NAS Benchmarks (Class B), NAMD, and HPL on 64 processes, the latest designs of MVAPICH require less than 5MB of internal memory on average per process and yet deliver the best available performance. For SuperLU, the memory usage increases to 10MB for the evaluated data sets, but still maintains optimal performance and a 5 times reduction in memory usage over older MVAPICH designs.

The rest of the paper is organized as follows: in Section 2 we provide the required background knowledge for this paper. In Section 3 we present our end application analysis of performance as well as the memory requirements by the MPI library for NAS Parallel Benchmarks, SuperLU, NAMD, and HPL. We describe the related work in Section 4. Finally, we conclude and point out future work in Section 5.

2. Background

In this section we provide the required background for the work done in this paper. There are two major topics that are relevant: a) the InfiniBand network with its associated features, and b) design of MPI (MVAPICH in particular) protocols and buffer management.

2.1. InfiniBand Overview

The InfiniBand Architecture [6] (IBA) defines a switched network fabric for interconnecting compute and I/O nodes. In an InfiniBand network, hosts are connected to the fabric by Host Channel Adapters (HCAs). A queue based model is used in InfiniBand. A Queue Pair (QP) consists of a send queue and a receive queue. Communication operations are described in the Work Queue Requests (WQR), or descriptors, and submitted to the work queue. It is a requirement that all communication buffers be posted into receive work queues before any message can be placed into them. In addition, all communication buffers need to be registered (locked in physical memory) before InfiniBand can either send from or receive data into that memory location. This restriction is imposed to ensure that memory is present when HCA accesses the memory. Finally, the completion of WQRs is reported through Completion Queues (CQ).

IBA provides several types of transport services: Reliable Connection (RC), Unreliable Connection (UC), Reliable Datagram (RD), and Unreliable Datagram (RD). RC and UC are connection-oriented and require one QP to be connected to exactly one other QP. On the other hand, RD and UD are connection-less and one QP can be used to communicate with many remote QPs. To the best of our knowledge, the RD transport has not yet been implemented by any InfiniBand vendor.

On top of these transport services, IBA provides services to upper level software; however, all software services are not defined for all transport types. Figure 1 depicts the particular software services defined for the various transports, as of IBA specification release 1.2. As shown in the figure, the send/receive operations are defined for all classes of transport. For connection-oriented transport, a new type of software service called Shared Receive Queue (SRQ) has been introduced. This allows the association of many QPs to one receive queue even for connection oriented transport. Thus, any remote process that is connected by a QP can send a message that is received in buffers specified in the SRQ.

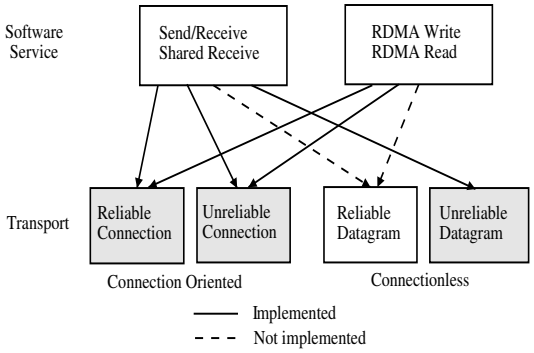


Figure 1. IBA Transport and Software Services

Apart from the basic send/receive operations, IBA

also defines Remote Direct Memory (RDMA) operations. Using this service, applications can directly access memory locations of remote processes. In order to utilize RDMA, the requesting process is required to know the virtual address and a memory access key of the remote process. RDMA operations typically have lower end-to-end latencies, since there is no receiver side software involvement in the critical data flow path. RDMA is supported on all reliable transports, the only exception being RDMA Read is not supported on UC.

In addition to these features, IBA provides a host of other exciting features like hardware multicast, QoS, Atomic operations. These features are not described here because they are not related to the research direction discussed in the paper. Additional details on these features can be obtained from IBA specification [6].

2.2. MVAPICH Design Overview

MVAPICH [13] is a popular implementation of MPI over InfiniBand. It uses several InfiniBand services like Send/Receive, RDMA-Write, RDMA-Read, and Shared Receive Queues to provide high-performance and scalability to end MPI applications. There are two major protocols used by MVAPICH. The first is the *Eager Protocol*, which is used to transfer small messages. These small messages require MPI internal communication buffers to help achieve low latency. The second protocol used is the *Rendezvous Protocol*. This protocol is used for large messages. In order to avoid buffering large messages inside the MPI library, availability of buffer at the receiver end is first negotiated with control messages. After the negotiation phase, the messages are sent directly to receiver user memory with the use of RDMA. In this paper we focus on the requirement and usage of MPI internal buffers; hence, we will describe the Eager protocol in detail. For more information on the Rendezvous protocol, please refer to [21].

MVAPICH provides several implementations for the Eager protocol based on different designs and utilizing different InfiniBand features. Users can choose a particular implementation based on their needs when compiling MVAPICH. In addition, these designs may be tuned dynamically at runtime. Among the various designs of the Eager protocol, there are three that stand out as the best. These are described in the following subsections.

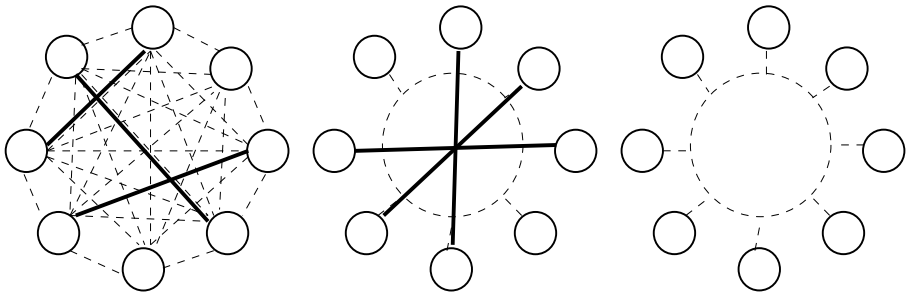
2.2.1. Adaptive RDMA with Send/Receive Channel The RDMA feature of InfiniBand offers very low latency due to the absence of receiver side software involvement, which is desirable for small messages. The RDMA channel [10] in MVAPICH provides a design by which the RDMA feature can be fully exploited to deliver low latency. The use of RDMA requires that communication buffers be made available for each remote process that may send messages. In order to avoid a memory-scalability problem when there are thousands of remote processes, this channel has an "adaptive" nature (hence the name Adaptive RDMA). At the time of communication initialization, only a limited number (typically only two or three) of buffers are allocated per remote process. These initial buffers are posted on the InfiniBand Send/Receive channel. Accordingly, all processes initially communicate using the InfiniBand Send/Receive channel semantics. MVAPICH maintains an internal counter of the number of messages exchanged by each pair of processes, and if this count increases beyond some threshold (runtime tunable), buffers are allocated and made available to the remote process over RDMA.

For the sake of brevity, this design will be referred to as ARDMA-SR for the rest of the paper and the connection between a pair of process that uses RDMA for Eager protocol will be called a RDMA Connection in this paper. Figure 2(a) illustrates this channel. The dotted lines show that all the processes have provided some limited number of buffers for the Send/Receive channel. However, this has to be done for each pair of processes. The bold lines indicate that six of the most frequently communicating processes actually communicate over RDMA.

This channel provides reasonably good memory scalability along with the low latency offered by RDMA.

2.2.2. Adaptive RDMA with SRQ Channel The Shared Receive Queue (SRQ) is a hardware feature provided by InfiniBand that allows upper-level software to post receive buffers to only one receive queue. Incoming messages from all remote processes in the MPI application can then consume buffers from this queue in a first-come-first-serve (FCFS) basis. This feature allows very efficient sharing of receive buffers across many InfiniBand connections. Thus, reducing the memory requirement by an order of magnitude for MPI applications that execute on very large process counts (up to tens of thousands).

One drawback of the SRQ is that the processes sending messages do not have an accurate picture of the receiver buffer availability. As such, if senders keep injecting packets into the network that do not have any destination buffer available, the performance of the application is degraded. In order to alleviate this situation, we have designed a novel, receiver-driven flow-control mechanism [20]. The receiving MPI process sets a "low-watermark" for the SRQ. When the number of available buffers in that queue drops below this threshold, an interrupt is generated by the HCA, which is caught by the MPI library. If there are more receiver buffers allocated already, then they are posted to the HCA to keep the SRQ full; however, if no buffers are available, new ones are allocated and posted to the SRQ to fill it. Figure 3 de-



(a) Adaptive RDMA with Send/Receive Channel (with 6 processes using RDMA)

(b) Adaptive RDMA with SRQ Channel (with 6 processes using RDMA)

(c) SRQ Channel

Figure 2. Various Eager Protocol Designs in MVAPICH

picts the triggering of a low-watermark event by a sender and the subsequent filling of the SRQ.

In this channel, at communication initialization, all processes have full SRQs and communicate using these buffers. When a certain number of runtime tunable buffers have been consumed from the SRQ, RDMA buffers are made available for that remote process. Hence, similar to the design described in the previous section, this design also achieves scalable memory usage along with low latency of RDMA.

Again, for the sake of brevity, the design will be referred to as ARDMA-SRQ for the rest of the paper. Figure 2(b) illustrates this design. Each process provides a set of receive buffers that are shared for every remote connection (shown by the dotted line). As before, the bold lines indicate that six frequently communicating processes are using RDMA.

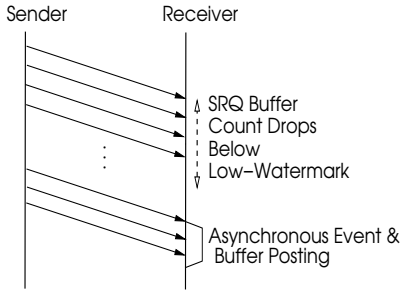


Figure 3. SRQ Low-Watermark Mechanism

2.2.3. SRQ Channel This channel exclusively utilizes the SRQ feature of InfiniBand. It employs the same receiver-driven flow-control mechanism as described in the previous section. The only difference in this channel from the previous one is that no RDMA buffers are allocated, even for frequently communicating pairs of processes. Even though RDMA channels can achieve lower-latency message passing, they consume more memory. This channel, which is exclusively based on SRQ, may have slightly increased point-to-point latency (only by around 1μs), but can provide very scalable message passing. Figure 2(c) illustrates this channel. For the rest of the paper, this design will be simply referred to as SRQ.

3. Experimental Evaluation

In this section, we present our analysis of the performance and the memory utilization of the MPI library (specifically MVAPICH [13]) while executing several well known MPI applications and benchmarks. The designs of Eager protocol evaluated are the Adaptive RDMA with Send/Receive (called ARDMA-SR), Adaptive RDMA with SRQ (called ARDMA-SRQ), and the SRQ channel (called SRQ). The descriptions of these channels are in Section 2.2.

Much of the data required for our analysis are not obtainable through any other publicly available tools. This is mainly because we aim to analyze information that is specific and *internal* to MVAPICH. In addition to this, our analysis requires the size and volume information of the messages actually sent by the MPI library. Most MPI profiling tools can provide information only about messages that were generated by the MPI application. As mentioned in Section 2.2, large message transfer may in fact involve several small message transfers as required by the Rendezvous protocol. The information about these messages is lost if we simply use MPI-level profilers.

In order to obtain an accurate picture of the various events occurring inside MVAPICH, we design an extremely low overhead profiling mechanism *internal* to MVAPICH. Our profiling implementation records information inside internal data structures of MVAPICH dur-

ing the actual application execution. All the information is then collected at the root process by `MPI_Reduce` during `MPI_Finalize`. Since the profiler need only update a few memory locations during the execution, there is almost no perceivable impact on the performance. Our profiling mechanism records important information such as:

1. Allocation of communication buffers
2. Message size and data volume profiles
3. Number of processes communicating over RDMA Eager Protocol
4. Number of "low-watermark" events experienced by the SRQ

In addition to our internal profiling of MVAPICH, we used `mpiP` [7], which is a lightweight, scalable MPI profiling tool. This tool provides us with information about which MPI calls were issued by the application. Combining this information (generated by `mpiP`) with our internal profiling of MVAPICH, provides an in-depth look into several aspects of the MVAPICH designs for the Eager protocol.

Table 1 shows the results of our profiling various applications on 64 processes. SuperLU profiling results are presented separately in Table 2. The percentage MPI time is reported by `mpiP` and the rest of the parameters are given by the MVAPICH internal profiling. This table will be referred to later as part of analysis of the results of each individual benchmarks.

3.1. Experimental Platform

Our experimental platform is a 64 node dual Opteron 2.4GHz (Processor 250) cluster. Each node is equipped with 8GB of main memory and PCI-Express interface. The nodes have MT25204 Mellanox HCAs with firmware version 1.0.1 and the OpenIB/Gen2 [14] software stack. The Linux kernel version used is 2.6.15.

3.2. NAS Benchmarks

The NAS Parallel Benchmarks [2] are a set of programs that are designed to be typical of several MPI applications, and thus, help in evaluating the performance of parallel machines. We used the MPI version of the benchmark suite. For the purposes of our evaluation, we include all the NAS Benchmarks except the Embarrassingly Parallel (EP) benchmark. We excluded this benchmark from our paper, since it has very little MPI communication and as such is of lesser significance when analyzing the operations inside the MPI library.

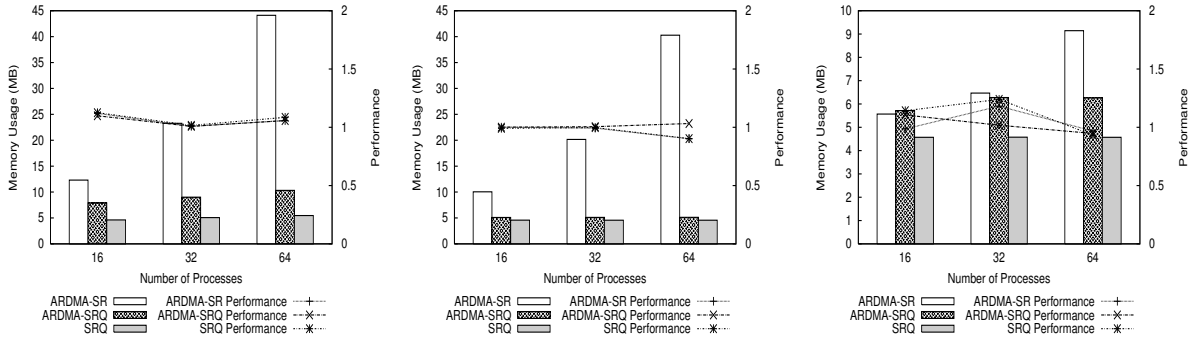
Figure 4 shows the performance of the NAS Benchmarks (Class B) using all three designs of the Eager protocol. The number of processes is varied from 16-64 for IS, FT, CG, LU, and MG. The SP and BT benchmarks are run on 49-81 processes since they require the total number of processes to be a square. Each graph has two y-axes. The left y-axis shows the communication memory used by MVAPICH while executing that particular benchmark, whereas the right y-axis shows the relative performance achieved by that benchmark execution. All the performance ratios have been normalized with respect to the best possible benchmark number obtained by the default configuration of MVAPICH version 0.9.7. A ratio > 1 indicates better performance than the default configuration of MVAPICH 0.9.7, while a ratio < 1 indicates worse performance.

The results indicate that the SRQ channel is able to provide almost the same level of performance as the other two schemes: ARDMA-SR and ARDMA-SRQ. While the SRQ channel provides almost the same performance, it does so with markedly less communication memory. In fact, in all the Figures 4(a) through 4(g), the SRQ channel consumes lesser than 5MB of communication buffers.

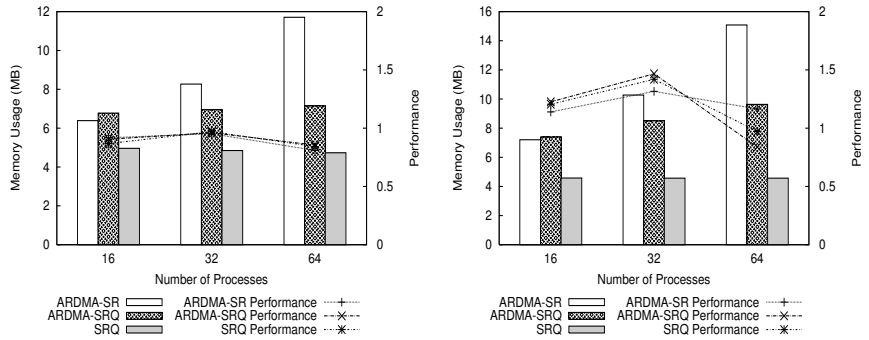
Memory utilization numbers for benchmarks IS, FT, BT, and SP are shown in Figures 4(a), 4(b), 4(f), and 4(g), respectively. These show an order of magnitude improvement (around 10 times for 64 and 81 process executions) in memory usage when ARDMA-SR is compared with ARDMA-SRQ or SRQ. However, Table 1 shows that the average number of RDMA connections (Section 2.2.1) is in fact not that high. To answer this apparent contradiction, we examine the message and volume profile graphs in Figure 5(a) and 5(c). By looking at these graphs, we can make out that these benchmarks do the major part of their communications using very large messages. As explained in Section 2.2, every large message transfer is associated with several smaller messages. These smaller messages are never sent over RDMA, rather exclusively use the Send/Receive channel. In order to transfer these small messages, an increasing number of communication buffers are allocated for the Send/Receive channel. Once the number of messages over the Send/Receive channel exceeds a certain amount, a much larger communication buffer set (64 in number) is required to be allocated *per* remote process for the Send/Receive channel. This consumes the most memory and exposes an inherent scalability issue even while using an adaptive protocol. The other NAS Benchmarks LU, MG, and CG show an improvement in memory usage as well as seen in Figures 4(d), 4(e), and 4(c). The SRQ channel consumes around half the memory required by ARDMA-SR. The difference in memory usage between ARDMA-SRQ and SRQ can be explained by the number of processes using RDMA. For example, in the LU benchmark (for 64 processes), there are on an average 3.92 RDMA connections. According to default MVAPICH 0.9.7 parameters, each RDMA connection utilizes around 500KB of memory, so analytically, the difference in memory usage between ARDMA-SRQ

	IS	MG	CG	FT	LU	BT	SP	NAMD	HPL
Avg. RDMA Connections	6.14	9.0	3.09	0.98	3.92	3.89	1.17	53.15	6.26
Avg. Low-Watermark events	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.03	0.0
Unexpected Messages (%)	2.7	10.2	13.5	11.9	38.1	0.3	0.7	48.2	13.6
Total Messages	$1.9e5$	$3.1e5$	$2.7e6$	$3.6e5$	$5.8e6$	$1.6e6$	$4.7e6$	$3.7e6$	$7.8e5$
MPI Time (%)	47.25	9.16	33.87	37.85	14.23	10.17	11.88	23.54	24.68

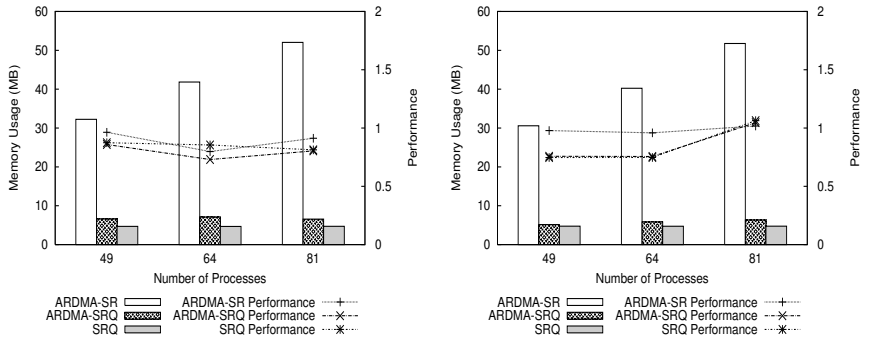
Table 1. Profiling Results on 64 processes of NAS (Class B), NAMD (apoa1), HPL



(a) IS Class B (b) FT Class B (c) CG Class B

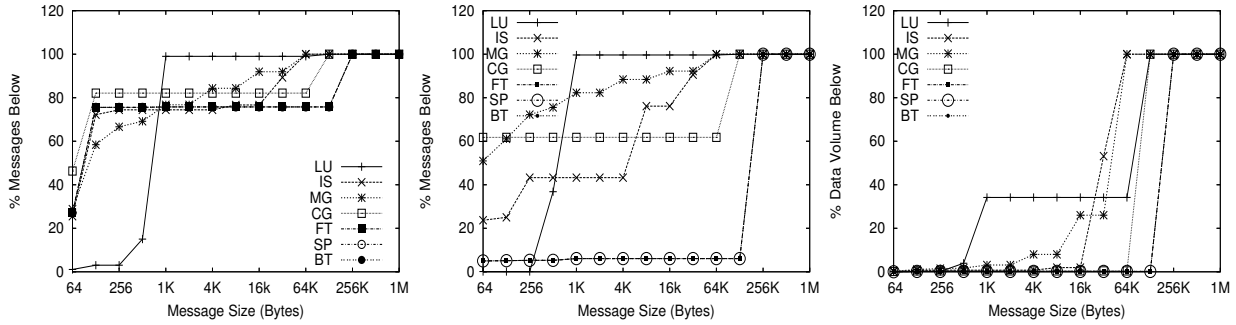


(d) LU Class B (e) MG Class B



(f) BT Class B (g) SP Class B

Figure 4. Performance of NAS Benchmarks



(a) Percentage messages below a certain message size

(b) Percentage Unexpected messages below a certain message size

(c) Percentage of Data Volume below a certain message size

Figure 5. Network-Level Message and Volume Profile of NAS Benchmarks

and SRQ should be $(500 * 3.92)/1024$ MB $= 1.9$ MB. From Figure 4(d), we can observe that the memory usage difference is indeed around 2MB for 64 processes.

We did, however, notice some variations in the performance results of the NAS benchmarks. The cause of these fluctuations may be due to NUMA issues such as process migration or due to the manner in which processes are mapped onto the physical processors. These fluctuations are observed for all the three Eager protocol designs, and as such are not an artifact of the MVAPICH designs. In all our benchmark executions, we used the same process to processor mapping for all Eager protocols. Further, no fluctuation in the memory usage was observed.

In the final version of the paper we will include NAS Benchmark results that are averaged over 25 times to rule out fluctuation effects. We could not acquire adequate time on the cluster for such large number of NAS runs during preparation of this version of the paper.

3.3. Super LU

SuperLU is a general purpose library for the direct solution of large, sparse, non-symmetric systems of linear equations on high performance machines [23]. SuperLU is offered in three different versions: sequential, multithreaded (for shared memory machines), and an MPI version to be used on distributed memory machines. We used the MPI version, called SuperLU_DIST [9] that contains a set of subroutines to solve a sparse linear system $A * X = B$. Currently, the program SuperLU_DIST parallelizes the LU factorization and triangular solution routines, which are the most time consuming.

The communication characteristics of SuperLU have been studied previously by Shalf, et al [18]. It has a variety of MPI calls which are predominantly `MPI_Isend`, `MPI_Irecv`, `MPI_Wait`, `MPI_Bcast`, and `MPI_Alltoall`. There are various data sets available for SuperLU_DIST. In our experiments, we have used `garon2.rua` and `rim.rua` from [3].

As seen in Figure 7(a), 94.99% of messages are less than 2KB for the `garon2` data set and 94.33% for the `rim` data set. While most messages are of small size, Figure 7(c) shows a few large messages that comprise most of the data volume.

Figures 6(a) and 6(b) show the performance and memory usage observed from our internal library profiling. As in the case of the NAS Benchmarks, the results indicate the ability of the SRQ channel to provide near-identical performance with significantly lower allocation of communication buffers. In the case of the `garon2` data set, usage remains roughly constant between the range of 7 to 9MB. Interestingly, with both data sets the memory usage for the SRQ design per process is higher for 16 processes than 32 or 64. From Table 2, we observe that using 16 processes, the average number of "low-watermark" events (when SRQ buffers are low) is approximately 1.5, while 32 and 64 processes have significantly less usage. This result suggests a communication pattern with significant bursts of unexpected messages and additionally that these bursts occur less frequently with a larger number of processes. These significant traffic bursts wake a thread to post additional buffers to the shared received queue, increasing the overall memory usage.

The benefits of the SRQ Eager protocol design are most prominent at a process group size of 64. We observe from Figures 6(a) and 6(b) that the communication buffer memory usage for `garon2` is nearly an order of magnitude less than the ARDMA design, yet maintains the same level of performance. The SRQ results for the `rim` data set yield similar results, with a 9 and 4 times improvement over the ARDMA-SR and ARDMA-SRQ designs, respectively, with near-identical performance. Most importantly, our evaluation shows a near-constant

	garon2			rim		
Processes	16	32	64	16	32	64
Avg. RDMA Connections	12.44	25.75	40.25	7.25	12.06	14.25
Avg. Low-Watermark events	1.56	0.06	0.12	1.56	0.66	0.64
Unexpected Messages (%)	33.5	22.0	31.6	29.4	24.2	30.0
Total Messages	$2.9e5$	$4.8e5$	$7.5e5$	$3.8e5$	$7.4e5$	$1.1e6$

Table 2. Profiling Results for SuperLU

memory usage per process, regardless of the process group size.

3.4. NAMD

NAMD is a fully featured, production molecular dynamics program for high performance simulation of large biomolecular systems [16]. NAMD is based on Charm++ parallel objects [15], which is a machine independent parallel programming system. Of the various data sets available with NAMD, we use the one called `apoa1`, which models a bloodstream lipoprotein particle.

The communication characteristics, as reported by `mpiP`, show the calls are primarily to `MPI_Isend`, `MPI_Send`, `MPI_Recv`, and `MPI_Barrier`. Our profile of the messages sent by the MPI library show 50% are under 128 bytes and the remaining 50% are between 128 and 32K bytes.

In Figure 9 we observe the same trends in performance and memory usage as in previous applications. For a process group size of 16 the SRQ design uses on average 6.1MB of memory and drops to 5.5MB and 5.2MB for the 32 and 64 process groups. As in SuperLU, we see the ability of the SRQ Eager protocol design to consume less memory with larger process groups due to a more balanced application communication pattern between all nodes. However, even with patterns with short bursts of unexpected traffic, such as the 16 process run, we observe a 50% improvement in memory usage over both of the ARDMA designs.

In contrast, the communication buffer usage in the ARDMA-SR design scales linearly with the number of processes. Table 1 shows one of the reasons for this scale. The number of RDMA connections also scales linearly with the number of processes due to a balanced communication pattern. This pattern triggers the creation of an RDMA channel after communicating a set number of messages, as discussed in Section 2.2.1. For 64 processes, our evaluation shows an average of 53.15 RDMA connections. The ARDMA-SRQ design also shows a significant increase over the SRQ design in memory usage due to RDMA channels. The difference in memory usage between the SRQ and ARDMA-SRQ designs is 28MB, which matches our previous model of the RDMA channel overhead: (RDMA Connections $\times$ 500KB) $=$ 53.15 Connections $\times$500KB $=$ 26.6MB.

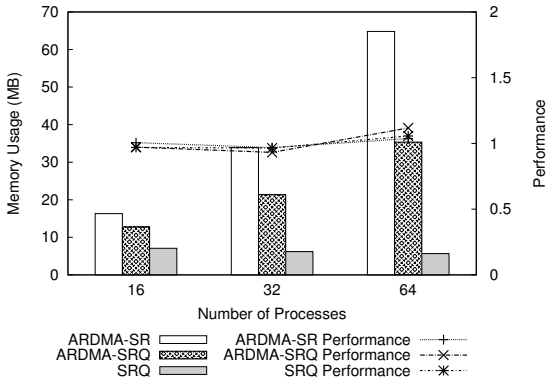


Figure 9. Performance of NAMD (apoa1)

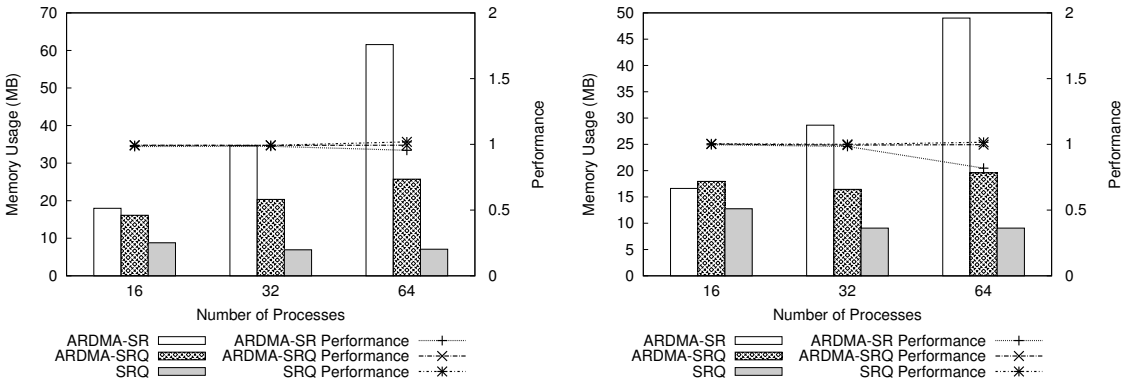
3.5. High Performance Linpack (HPL)

High Performance Linpack (HPL) is benchmark based on solving systems of linear equations [4]. It is used as the primary measure for ranking a bi-annual Top 500 list [22] of the world's fastest supercomputers.

The communication pattern, as recorded by `mpiP`, shows the calls are primarily to `MPI_Recv`, `MPI_Send`, and `MPI_Irecv`. Figure 11(a) shows the results of our profiling of the messages sent by the MPI library. We observe that while 50% of the messages are under 128 bytes, most of these are control messages for the larger application-level messages.

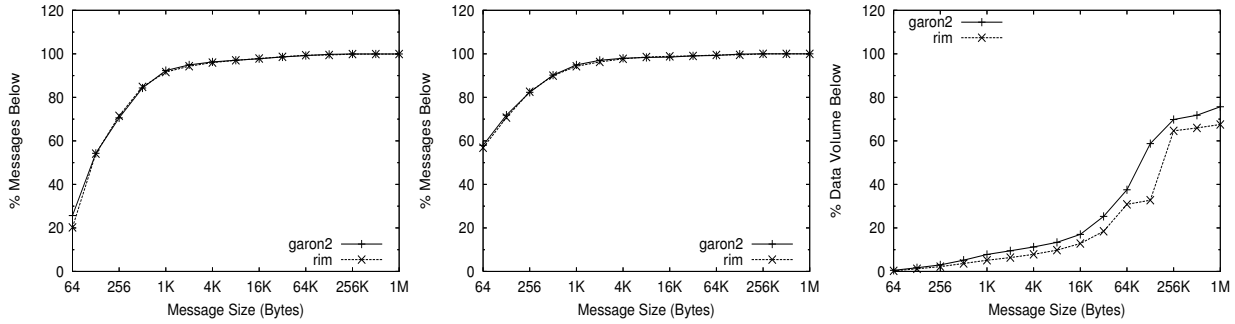
Figure 10 shows the performance and communication buffer memory usage observed for 16, 32, and 64 process runs of HPL. We once again see a relatively constant rate of performance for all of the Eager design schemes. The SRQ channel, however, is able to use a constant communication buffer size of less than 5MB for all evaluated process sizes.

Referring to Table 1 we can see that for 64 processes, on average, only 6.26 RDMA connections are established. This result explains the approximately 3.5MB difference between the ARDMA-SRQ and SRQ designs; our model relating to RDMA channel memory requirements from other sections holds here as well. There is also a marked increase in the memory usage between the ARDMA-SRQ and ARDMA-SR designs of nearly 35MB for 64 processes. Although Figure 11(a) shows that many messages sent are of medium size, there are



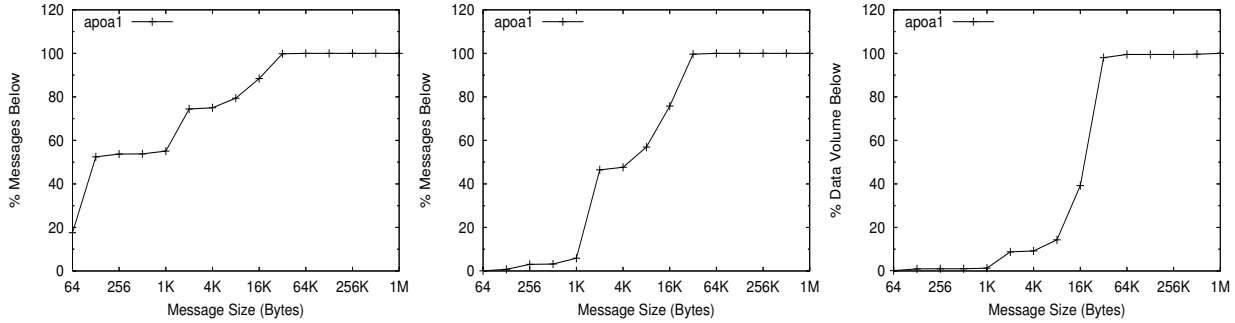
(a) garon2 (b) rim

Figure 6. Memory Usage and Performance of SuperLU



(a) Percentage messages below a certain message size (b) Percentage Unexpected messages below a certain message size (c) Percentage of Data Volume below a certain message size

Figure 7. Network-Level Message and Volume Profile of SuperLU Datasets



(a) Percentage messages below a certain message size (b) Percentage Unexpected messages below a certain message size (c) Percentage of Data Volume below a certain message size

Figure 8. Network-Level Message and Volume Profile of NAMD Datasets

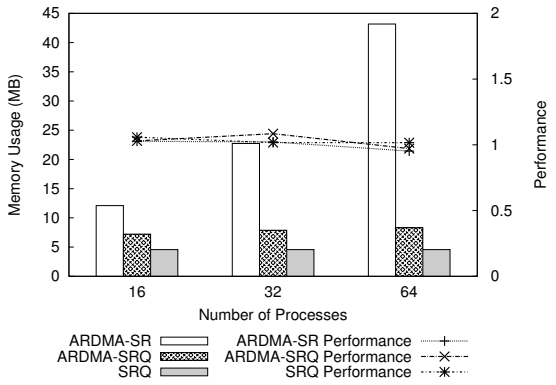


Figure 10. Performance of HPL

also a significant number of larger messages. As discussed earlier, even large messages require smaller control messages to be sent over the Send/Receive channel. When many of these smaller messages are transferred, an increasing number of communication buffers must be allocated on a per connection basis in the ARDMA-SR design, raising the memory usage of the MPI library.

4. Related Work

Memory usage with a SRQ-based design was discussed by Shipman, et al [19], however, the design proposed showed degradation of application performance and did not include MPI library-level profiling. The MVAPICH SRQ channel design was described in [20].

Communication characteristics of SuperLU have been studied previously by Shalf, et al [18]. Message profiles of NAMD and other molecular dynamics programs were studied by Alam, et al [1].

To the best of our knowledge, this is the first contemporary study that comprehensively examines the effect of MPI library memory usage on performance and the expected memory requirements of the MPI library with various adaptive schemes.

5. Conclusions and Future Work

As InfiniBand gains popularity and is included in increasingly larger clusters, having a scalable MPI library is imperative. In this paper have have sought to address two important questions relating to this goal:

1. Does aggressively reducing communication buffer memory lead to degradation of end application performance?
2. How much memory can we expect the MPI library to consume during execution of a typical application while providing the best available performance?

Through our evaluation of the NAS Parallel Benchmarks, SuperLU, NAMD, and HPL, we have explored both of these questions. In regard to the first question, we have shown that all of the schemes in MVAPICH are able to attain near-identical performance on a variety of applications. Our evaluation showed that the latest SRQ design of MVAPICH is able to use a constant amount of internal memory per process with optimal performance, regardless of the number of processes.

To address our second question, we have evaluated the memory usage for varied numbers of processes – up to 81. Our analysis shows a near linear increase in memory usage for the ARDMA-SR design due to the requirement of allocating buffers on a per connection basis. The ARDMA-SRQ scheme improves memory usage by using a shared queue of buffers, but can suffer from a near-linear scaling of RDMA channels – as seen in our runs of the `apoa1` dataset on NAMD. The latest MVAPICH SRQ design, however, is able to maintain constant buffer usage as low as 5MB for all NAS Benchmarks (Class B), NAMD, and HPL to a maximum of just over 10MB for certain datasets of SuperLU.

In the future we plan to continue evaluating the memory usage and performance of these various designs on larger clusters. We also plan to investigate decreasing memory usage further by allocating an even lower initial number of communication buffers.

References

[1] Sadaf R. Alam, Jeffrey S. Vetter, Pratul K. Agarwal, and Al Geist. Performance characterization of molecular dynamics techniques for biomolecular simulations. In *PPoPP '06: Proceedings of the eleventh ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 59–68, New York, NY, USA, 2006. ACM Press.

[2] 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. volume 5, pages 63–73, Fall 1991.

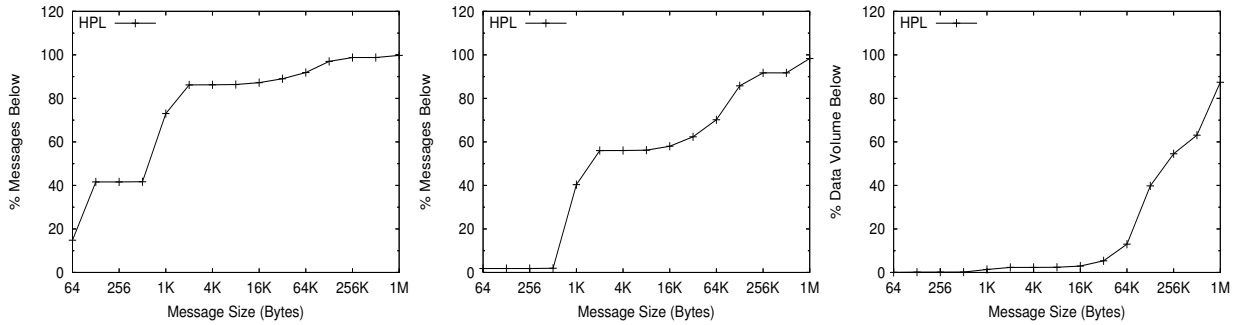
[3] T. Davis. University of Florida Sparse Matrix Collection. http://www.cise.ufl.edu/research/sparse/matrices.

[4] J. Dongarra. Performance of Various Computers Using Standard Linear Equations Software. Technical Report CS-89-85, University of Tennessee, 1989.

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

[6] InfiniBand Trade Association. InfiniBand Trade Association. http://www.infinibandta.com.

[7] J. Vetter and C. Chambreau. mpiP: Lightweight, Scalable MPI Profiling. http://www.llnl.gov/CASC/mpip/.



(a) Percentage messages below a certain message size

(b) Percentage Unexpected messages below a certain message size

(c) Percentage of Data Volume below a certain message size

Figure 11. Network-Level Message and Volume Profile of HPL

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

[9] X. Li and J. Demmel. SuperLU DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. Mathematical Software*, 29(2):110 – 140, 2003.

[10] J. Liu, J. Wu, , and D. K. Panda. High performance RDMA-based MPI implementation over InfiniBand. *Int'l Journal of Parallel Programming*, 32(3), June 2004.

[11] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, Jul 1997.

[12] NAS Project: Columbia. Columbia Supercomputer. http://www.nas.nasa.gov/About/ Projects/Columbia/columbia.html.

[13] Network-Based Computing Laboratory. MPI over InfiniBand Project. http://nowlab.cse.ohio-state.edu/projects/mpi-iba/.

[14] Open InfiniBand Alliance. Open Source InfiniBand Software Stack. http://www.openib.org/.

[15] Parallel Programming Laboratory. Charm++. http://charm.cs.uiuc.edu/research/charm/.

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

[17] Sandia National Laboratories. Thunderbird Linux Cluster. http://www.cs.sandia.gov/platforms/Thunderbird.html.

[18] J. Shalf, S. Kamil, L. Oliker, and David Skinner. Analyzing UltraScale Application Communication Requirements for a Reconfigurable Hybrid Interconnect. In *Supercomputing*, 2005.

[19] G. Shipman, T. Woodall, R. Graham, and A. Maccabe. Infiniband Scalability in Open MPI. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.

[20] 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.

[21] S. Sur, H.-W. Jin, L. Chai, and D. K. Panda. RDMA Read Based Rendezvous Protocol for MPI over InfiniBand: Design Alternatives and Benefits. In *Symposium on Principles and Practice of Parallel Programming (PPOPP)*, 2006.

[22] The Top 500 Project. The Top 500. http://www.top500.org/.

[23] Xiaoye Sherry Li, James Demmel, John R. Gilbert. SuperLU. http://crd.lbl.gov/~xiaoye/SuperLU/.