MVAPICH-Aptus: Scalable High-Performance Multi-Transport MPI over InfiniBand *

Matthew J. Koop[†‡] Terry Jones[‡] Dhabaleswar K. Panda[†]

[†] *Network-Based Computing Laboratory*
The Ohio State University
Columbus, OH 43210
{koop, panda}@cse.ohio-state.edu

[‡] *Lawrence Livermore National Laboratory*
Livermore, CA 94550
trj@llnl.gov

Abstract

The need for computational cycles continues to exceed availability, driving commodity clusters to increasing scales. With upcoming clusters containing tens-of-thousands of cores, InfiniBand is a popular interconnect on these clusters, due to its low latency (1.5µsec) and high bandwidth (1.5 GB/sec). Since most scientific applications running on these clusters are written using the Message Passing Interface (MPI) as the parallel programming model, the MPI library plays a key role in the performance and scalability of the system. Nearly all MPIs implemented over InfiniBand currently use the Reliable Connection (RC) transport of InfiniBand to implement message passing. Using this transport exclusively, however, has been shown to potentially reach a memory footprint of over 200MB/task at 16K tasks for the MPI library. The Unreliable Datagram (UD) transport, however, offers higher scalability, but at the cost of medium and large message performance.

In this paper we present a multi-transport MPI design, MVAPICH-Aptus, that uses both the RC and UD transports of InfiniBand to deliver scalability and performance higher than that of a single-transport MPI design. Evaluation of our hybrid design on 512 cores shows a 12% improvement over an RC-based design and 4% better than a UD-based design for the SMG2000 application benchmark. In addition, for the molecular dynamics application NAMD we show a 10% improvement over an RC-only design. To the best of our knowledge, this is the first such analysis and design of optimized MPI using both UD and RC.

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-CONF-400676). This research also supported in part by U.S. Department of Energy grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; National Science Foundation grants #CNS-0403342 and #CCF-0702675; grant from Wright Center for Innovation #WCI04-010-OSU-0.

1. Introduction

Commodity cluster computing has been growing at a furious speed over the last decade as the need for additional computational cycles continues to exceed availability. One of the most popular interconnects used on these clusters is InfiniBand [8], an open-standard with one-way latencies approaching 1μsec and bandwidth of 1.5GB/sec. Introduced in 2000, InfiniBand was developed as a general system I/O fabric. Since then, however, it has found particularly wide acceptance in High Performance Computing (HPC).

As InfiniBand clusters continue to expand to ever increasing scales, the need for scalability and performance at these scales remains paramount. As an example, the upcoming "Ranger" system at the Texas Advanced Computing Center (TACC) includes over 60,000 cores with nearly 4000 InfiniBand ports [26]. By comparison, the first year an InfiniBand system appeared in the Top500 list of fastest supercomputers was in 2003 with a 128 node system at NCSA [2]. The latest list shows over 25% of systems are now using InfiniBand as the compute node interconnect.

Its popularity growing, InfiniBand-based MPI benefits from ongoing research and improved performance and stability.

The Message Passing Interface (MPI) [15] is the dominant parallel programming model on these clusters. Given the role of the MPI library as the communication substrate for application communication, the library must take care to provide scalability both in performance and in resource usage. As InfiniBand has gained in popularity, research has continued on improving the performance and scalability of MPI over it. Various optimizations and techniques have been proposed to optimize for performance and scalability, however, as InfiniBand reaches unprecedented sizes for commodity clusters it is necessary to revisit these earlier works and take the best

aspects of each.

As an example, recent studies have shown that the memory footprint for InfiniBand communication contexts can be significant at such large scales [9, 10], impeding the ability to increase problem resolution due to memory constraints. A significant reason for this is the Reliable Connection (RC) transport used in most MPIs over InfiniBand, which requires a few KB of dedicated memory for each communicating peer. Our earlier proposed solution [10] to this issue uses the Unreliable Datagram (UD) transport exclusively. Even this method has limitations, however, since the performance of UD is below that of RC in many situations, particularly for medium and large messages.

In this paper we seek to address two main questions:

- What are the current message channels developed over InfiniBand and how do they perform with scale?
- Given this knowledge, can an MPI be designed to dynamically select suitable transports and message channels for the various types of communication to improve performance and scalability for the current and next-generation InfiniBand clusters?

As part of this work we develop a multi-transport MPI for InfiniBand, MVAPICH-Aptus[1], which dynamically selects the underlying transport protocol, Unreliable Datagram (UD) or Reliable Connection (RC), as well as the message protocol over the selected transport on a per message basis to increase performance over MPIs that only use a single InfiniBand transport. We design flexible methods to enable the MPI library to adapt to different network and applications.

Our results on a variety of application benchmarks are very promising. On 512 cores, MVAPICH-Aptus shows a 12% improvement over an RC-based design and 4% better than a UD-based design for the SMG2000 [5] application benchmark. In addition, for the molecular dynamics application NAMD [18] we show a 10% improvement over an RC-only design.

The rest of the paper is organized as follows: In Section 2, we provide the requisite background information on InfiniBand. Following in Section 3, we examine existing message passing channels over InfiniBand and evaluate their characteristics in Section 4. Our multi-transport design, Aptus, is presented in Section 5. We evaluate our prototype on various application benchmarks in Section 6. Related work is noted in Section 7; we conclude and discuss future work in Section 8.

[1] 'Aptus' is Latin for "appropriate or fitting"

2. Background

InfiniBand is a processor and I/O interconnect based on open standards [8]. It was conceived as a high-speed, general-purpose I/O interconnect, and in recent years it has become a popular interconnect for high-performance computing to connect commodity machines in large clusters.

2.1. Communication Model

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

There are two types of communication semantics in InfiniBand: channel and memory semantics. Channel semantics are send and receive operations that are common in traditional interfaces, such as sockets, where both sides must be aware of communication. Memory semantics are one-sided operations where one host can access memory from a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA). Remote write and read are both supported in InfiniBand. Both communication semantics require communication memory to be registered with InfiniBand hardware and pinned in memory.

2.2. Transport Services

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

Reliable Connection (RC) is the most popular transport service for implementing MPI over InfiniBand. As a connection-oriented service, a QP with RC transport must be dedicated to communicating with only one other QP. A process that communicates with N other peers must have at least N QPs created. The RC transport provides almost all the features available in InfiniBand, most notably reliable send/receive, RDMA and atomic operations. *Unreliable Connection (UC)* is similar to RC but has no guarantees on ordering or reliability.

Unreliable Datagram (UD) is a connection-less and unreliable transport, the most basic transport specified for InfiniBand. As a connection-less transport, a single UD QP can communicate with any number of other UD QPs. However, the UD transport does have a number of limitations. All outgoing packets must be limited to MTU size (maximum 2KB on Mellanox [1] hardware). Given this restriction, the application or upper-level communication layer, such as MPI, must manage fragmentation and re-assembly of messages. The UD transport does not provide any reliability: lost packets are not reported and the arrival order is not guaranteed. The UD transport additionally does not enable RDMA. All communication must be performed using channel semantics, i.e. send/receive.

3. Message Channels

In this section we describe each of the communication channels that are available for message transfer in an InfiniBand-based cluster. MPI libraries in general implement all communication between tasks using methods that can be categorized under two basic protocols:

- *Eager Protocol:* In the eager protocol, the sender task sends the entire message to the receiver without any knowledge of the receiver state. In order to achieve this, the receiver must provide sufficient buffers to handle incoming unexpected messages. This protocol has minimal startup overhead and is used to implement low latency message passing for smaller messages.
- *Rendezvous Protocol:* The rendezvous protocol negotiates buffer availability at the receiver side before the message is sent. This protocol is used for transferring large messages, when the sender wishes to verify the receiver has the buffer space to hold the entire message. Using this protocol also easily facilitates zero-copy transfers when the underlying network transport allows for RDMA operations.

3.1. Eager Protocol Channels

Reliable Connection Send/Receive (RC-SR): We refer to RC-SR as the channel built directly on the channel semantics of InfiniBand. It is the primary form of communication for small messages on nearly all MPI implementations over InfiniBand. Two designs have been proposed, one with per-peer credit-based flow control and the other using the Shared Receive Queue (SRQ) support of InfiniBand. In this paper we use only the SRQ-based design since it has superior scalability (detailed and shown in earlier work [22, 19]), and since it allows receive buffers to be pooled across QPs (connections) instead of posted on a per-peer basis.

Reliable Connection Fast-Path (RC-FP): Current InfiniBand adapters only reach their lowest latency when using RDMA write operations, with channel semantics having a 2μsec additional overhead (e.g. 5μsec vs. 3μsec) on our evaluation hardware. The newest Mellanox adapter, ConnectX [14], reduces this gap to less than a microsecond, however RDMA write operations still achieve the lowest latency [24].

To leverage this capability, small message transfer has been designed over the RDMA write mechanism to facilitate the lowest latency path of communication [13]. Dedicated buffers are required for each communicating peer – the default MVAPICH configuration requires over 300KB of memory per RC-FP channel created. To limit memory usage, channels are currently setup adaptively and limited to a configurable number of channels in current MPIs over InfiniBand. In addition, each RC-FP channel requires polling an additional memory location for detection of message arrival. For example, communication with n peers using the RC-FP channel requires polling n memory locations for message arrival.

Unreliable Datagram Send/Receive (UD-SR): As designed and described in our earlier work [10], the UD-SR message passing channel is message transfer implemented over the channel semantics of the UD transport of InfiniBand. Message segmentation and reliability must be handled within the MPI library to provide the guarantees made by the MPI specification to applications. Advantages of using this channel include superior memory utilization since a single UD QP can communicate with any other UD QPs; each QP is not dedicated to a specific peer as with the RC transport.

3.2. Rendezvous Protocol Channels

Reliable Connection RDMA (RC-RDMA): The RC-RDMA channel is the mechanism for sending large messages. Using this method, the sender can use an RDMA write operation to directly write into the application buffer of the receiver without intermediate copies. Additional modes have also been suggested based on RDMA read [23] and a pipelined RDMA write [21]; however, in this paper we consider only RDMA write.

Unreliable Datagram Zero-Copy (UD-ZCopy): In [11], a zero-copy protocol for transferring large messages over the UD transport of InfiniBand was proposed. Bandwidth for large messages is significantly increased due to the lack of copies on both sides of communication. The primary motivation for this channel is to provide

high bandwidth and to avoid scalability problems with RC QPs.

Copy-Based Send: If neither of the previously noted rendezvous channels are available, large messages can be segmented within the MPI library into many small sends and sent using an eager protocol channel (after negotiating buffer availability). This method, however, introduces intermediate copies and degrades performance.

3.3. Shared Memory

Clusters with multiple tasks running per node often use shared memory communication to communicate within a single node. This reduces contention on the network device and can provide lower latency and higher performance. In this paper we will consider the design described in [6], which is included in current versions of MVAPICH [16]. This provides both an eager and rendezvous protocol design for intra-node communication.

4. Channel Evaluation

Our experimental test bed is 560-core InfiniBand Linux cluster. Each of the 70 compute nodes has dual 2.33 GHz Intel Xeon "Clovertown" quad-core processors for a total of 8 cores per node. Each node has a Mellanox MT25208 dual-port Memfree HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [17], OFED 1.2 release.

The RC-based message channels we evaluate follow the design of MVAPICH [16], a popular open-source MPI implementation over InfiniBand. It is based on MPICH [7] and MVICH [12] and is used by over 610 organizations worldwide. The UD-based message channels are based on the design included in MVAPICH-UD [10, 11].

We evaluate the basic performance of latency and bandwidth of the channels, followed by an investigation into the scalability of each channel.

4.1. Basic Performance Microbenchmarks

In this section we investigate the basic characteristics of each message passing channel that is available.

Ping-Pong Latency: Figure 1 shows the latency of each of the eager message channels. RC-FP shows the lowest latency, with a minimum of slightly less than 3μsec. RC-SR and UD-SR have very similar latency results up to 2KB; at this point UD-SR is sending 2KB of data as well as the required MPI header information – resulting in a message size of over 2KB, which is the MTU on our evaluation HCA. This requires segmentation within the MPI library due to limits of the UD transport; this cost is clearly visible.

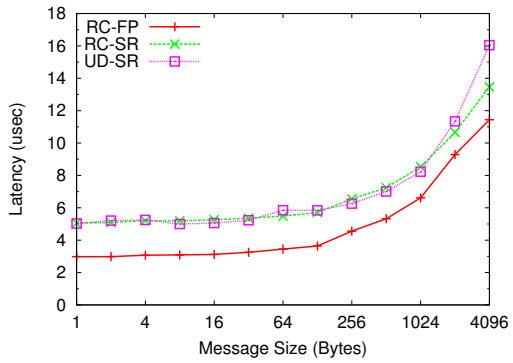


Figure 1. Channel Latency Comparison

Uni-directional Bandwidth: To measure the uni-directional bandwidth, we use the `osu_bw_mr` benchmark [16], which sends a window of sends from one task to another and measures the time until an acknowledgment from the receiver is received. In addition, the benchmark measures the aggregate bandwidth achieved by multiple pairs of communicating tasks between nodes.

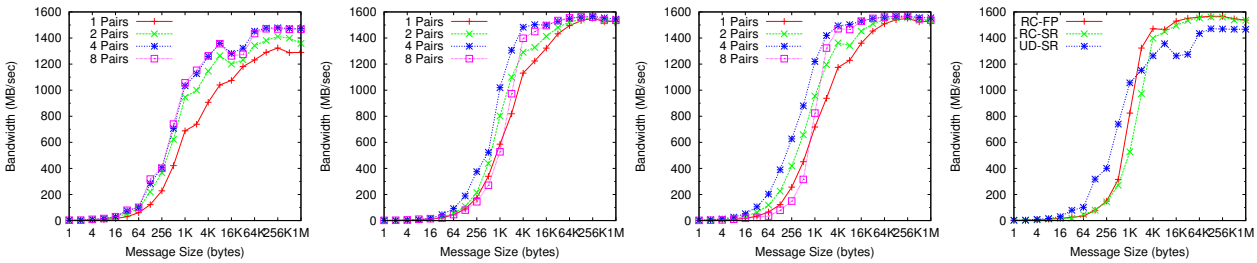
Figure 2 shows the results for each of the eager message channels, paired with the rendezvous channel of the same transport. For small message sizes we see that UD-SR demonstrates high bandwidth and there is no decrease in performance with additional pairs of tasks. By contrast, RC-FP and RC-SR show performance degradation from 4 to 8 pairs of concurrent communication. The UD transport requires less HCA overhead since hardware-level ACKs are not sent for UD messages and state information does not need to be retained on the HCA. As in the latency evaluation, we note a decrease in performance for UD-SR after 1KB due to segmentation overhead.

For large message bandwidth we note that UD-ZCopy achieves significant throughput but is slightly lower than RC-RDMA for a single pair. Additional overheads, such as posting in 2KB chunks, are required in the UD-ZCopy protocol that lower the performance below the fabric limits that RC-RDMA achieves.

4.2. Evaluation of Channel Scalability

In this section we evaluate several other characteristics of each message channel, in particular those that have scalability aspects.

Memory Footprint: While a channel may provide high-performance it may come only at the cost of host memory usage. Figure 3(a) shows our measurement of channel memory usage per task. From the graph we immediately note that RC-FP consumes significant amounts of memory, making a large number of RC-FP channels infeasible. RC-SR/RC-RDMA also have a signif-



(a) UD-SR/UD-ZCopy (b) RC-SR/RC-RDMA (c) RC-FP/RC-RDMA (d) 8 Pairs

Figure 2. Multi-Pair Uni-Directional Bandwidth Comparison

icant memory footprint as the number of connections increases since RC-SR/RC-RMDA are built on the RC transport. Recall from Section 2 that each RC QP must be dedicated to another RC QP in the peer task. Memory usage for UD-based transports is negligible since a single UD QP can communicate with any other UD QP in any task, leading to superior memory scalability.

Performance with Increasing Channel Count: Another important aspect to evaluate is the effect of multiple channels on performance. In the earlier subsection we evaluated only two tasks, which does not show scalability related effects.

As described earlier, RC-FP requires dedicated receive buffers for each communicating peer. As a result, a byte for each RC-FP channel must be polled to detect message arrival. To explore the cost of this polling we modify the RC-FP channel to poll on a configurable numbers of buffers. Figure 3(b) shows the 4-byte latency with increasing poll counts. We also plot the line for RC-SR latency, since polling RC-FP buffers also delays the detection of messages arriving on any other channel. Based on this result it becomes clear that more than 100 RC-FP channels will lead to performance degradation over a RC-SR only design.

By contrast, RC-SR and UD-SR maintain the same latency as the number of allocated channels increases. All completions are placed in a single CQ, where the library can poll for message arrival.

Impact of HCA Architecture: Although RC-SR shows similar performance to UD-SR with increasing numbers of allocated channels, performance differs from UD-SR when each of the channels is *used* instead of simply allocated.

InfiniBand HCAs cache QP information using on-card memory, called the InfiniBand Context Memory (ICM) cache. The ICM cache has a limited size and cannot hold more than a limited number of QP entries at any one time; context information outside of the cache must be fetched from host memory.

We replicate the evaluation from [25] to measure the cache size for our newer-generation HCAs by evaluating the 4-byte latency at the lowest software layer, the InfiniBand "verbs." Figure 3(c) shows that the ICM cache of the HCA is still limited in size with large increases in latency when multiple QPs are accessed in a round-robin order. All protocols based on the RC transport have this issue. Furthermore, this problem is exacerbated by the increase in core counts which lead to larger number of tasks sharing the same HCA (and ICM cache). UD-SR does not have this issue since a single UD QP can communicate with any other number of UD QPs – thus remaining in the HCA cache.

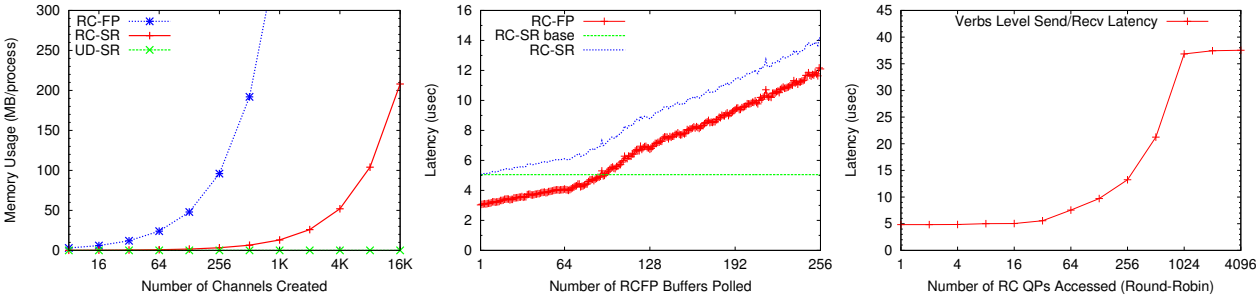
5. Proposed Design

In this section we describe our multi-transport design that incorporates all available communication channels. Since neither the RC or UD transport provides all of the desired features – scalability and best performance – a hybrid of the two transports is required. In this section we propose our design, MVAPICH-Aptus, that encompasses all of the available channels into a unified design that allows flexibility in channel selection. Based on the results from the previous section, a summary of channel characteristics is provided in Table 1. Figure 4 shows the general overview of the design.

We first describe the initial state of Aptus at startup, followed by a discussion of how Aptus multiplexes channels and provides reliability. Next we explain the channel selection algorithm of Aptus and the channel allocation strategies used.

5.1. Initial Channel Allocation

At job startup, `MPI_Init`, Aptus only creates UD QPs and exchanges their information to the other tasks in the job. At this point all tasks in the job are able to communicate with any other task using the UD-SR and UD-ZCopy channels. If tasks are sharing the same physical host, the SMP channel is also automatically allocated.



(a) Channel Memory Usage (b) RC-FP Polling (c) Effect of HCA ICM Cache

Figure 3. Channel Scalability Evaluation

Table 1. Channel Characteristics Summary

Type	Channel	Transport	Latency	Throughput	Scalability
Eager	RC Send/Receive (RC-SR)	RC	Good	Fair	Fair
	RC Fast-Path (RC-FP)	RC	Best	Good	Poor
	UD Send/Receive (UD-SR)	UD	$<$ 2KB, Good $\geq$ 2KB, Poor	$<$ 2KB, Best $\geq$ 2KB, Poor	Best
Rendezvous	RC-RDMA	RC	-	Best	Fair
	UD Zero-Copy (UD-ZCopy)	UD	-	Good	Best
	Copy-Based	UD or RC	-	Poor	-

After communication begins, Aptus tracks communication characteristics to determine the peers that each task communicates most frequently with as well as the message sizes. Using these statistics Aptus dynamically allocates more resource intensive channels, such as RC-SR or RC-FP, to reduce the communication overhead between sets of peers. The interpretation of these statistics must be done in consideration with the architecture and characteristics of each channel. Details are provided later in Section 5.4.

In addition, since performance characteristics for different message sizes vary with the channel, multiple channels between tasks may be required to obtain the best performance. For this reason Aptus allows for multiple channels to be active between tasks.

5.2. Channel Multiplexing and Reliability

As mentioned earlier, Aptus allows for multiple communication channels to be active simultaneously. As part of the support for UD-SR, Aptus must contain a reliability and re-ordering engine since the UD transport that underlies UD-SR is unreliable and subject to message drops and reordering. To support this, we design a generalized framework to re-order messages from all message channels since depending on the channel there may be out-of-order receives. In addition, each message channel can be independently configured to use the message reliability support. For example, is is not necessary for reliability to be enabled for RC-SR or RC-FP, so we can disable reliability. This elegant integration also allows other features such as an end-to-end CRC check to be done across all channels very simply if reliability is turned on for all channels and a CRC is computed on send and receive.

5.3. Channel Selection

Given the framework described, one area left unresolved is how to determine which message channel should be used of those allocated. The factors that motivate this selection are almost entirely from the architecture characteristics. Factors such as the different overheads between RC-SR and RC-FP may change with the HCA model and should be reflected in the way messages are sent. Our main observation is that factors that drive message channel selection are fluid – they may change based on cluster architecture or the number of ranks in the MPI job.

To support a flexible model we design a *send rule chain* method of determining how messages should be sent. A single send rule is in the form of {`COND, MESG_CHANNEL`}, e.g. {`MSG_SIZE <= 1024, UD-SR` }.

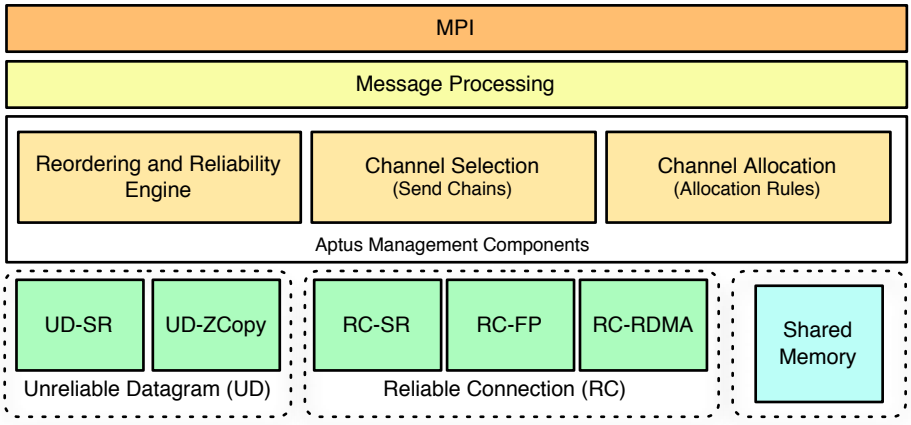


Figure 4. MVAPICH-Aptus Design Overview

If `COND` evaluates to true and `MESG_CHANNEL` is already allocated, then `MESG_CHANNEL` will be used for this message. Multiple of these send rules can be chained together, with earlier rules taking priority over later rules in the chain. The last rule in the chain must have a conditional of `TRUE` with UD-based channel to be valid.

Based on our evaluation in Section 4, we found that RC-FP has superior latency and should be used if available. In addition, RC-SR and UD-SR perform similarly in latency for small messages, however UD-SR has better throughput for messages under 2KB. Larger messages always gain performance using RC-RDMA if available. In keeping with these findings, we develop the following default send rule chain for our system:

```
{ MSG_SIZE <= 2048, RC-FP },
{ MSG_SIZE <= 2008, UD-SR },
{ MSG_SIZE <= 8192, RC-SR },
{ MSG_SIZE <= 8192, UD-SR },
{ TRUE, RC-RDMA }, { TRUE, UD-ZCOPY }
```

[*Note that these rules do not take into account whether a channel should be created, just whether to use it if it has* ***already been allocated.***]

Using this flexible framework, send rules can be changed on a per-system or job level to meet application and hardware needs without changing code within the MPI library and re-compiling.

Although our current prototype does not support any other variables besides `MSG_SIZE` and `NUM_RANKS` in the conditional, other rules could potentially be designed to allow for additional flexibility.

5.4. Channel Allocation

The previous section explored how messages can be sent over different channels using a configurable send rule chain. In this section we discuss how each of those channels is allocated initially.

As discovered earlier in our channel evaluation, it is detrimental to performance as well as the memory footprint to create certain channels after a number of them have already been created. For example RC-FP adds latency to all other channels as well as itself as more peers are communicated with over RC-FP. Similarly, too many RC connections uses a significant amount of memory and can overflow the HCA ICM cache, leading to significant latency.

It is important to use the minimal number of these channels while allocating high performance channels only to those peers that will benefit most from them. In our current prototype we use a variation of the send rules described in the previous section to increment counters to determine which peers would benefit most from a new connection. After the counter for a rule has reached its configured limit, the channel of that type is created provided per-task limits have not been passed.

Based on our evaluation, in our configuration we limit the number of RC-SR/RC-RDMA channels to 16 per task, meaning a task can only use the RC transport to a maximum of 16 of its peers. Similarly we limit RC-FP to 8 channels per task. These limits represent a tradeoff between the performance provided by each channel and the performance degradation that occurs with too many channels of these types. These limits are run-time tunable to allow flexibility based on HCA type and architecture. Limiting the RC QPs limits the potential for HCA cache thrashing.

6. Application Benchmark Evaluation

In this section we evaluate a prototype of our Aptus design on the NAS Parallel Benchmarks, NAMD, and

SMG2000. Our evaluation platform is the same as described in Section 4. To explore performance changes based on the hybrid model of Aptus, we evaluate four different configurations:

- *RC*: In this configuration we evaluate using MVAPICH 0.9.9, in the default configuration, aside from additional receive buffers posted to the SRQ. This is the baseline performance expected for an RC-based MPI. Includes RC-SR, RC-FP, and RC-RDMA.
- *UD*: Our design using only UD-SR and UD-ZCopy (MVAPICH-UD)
- *UD-copy*: Our design using only UD-SR
- *Aptus*: The prototype of our design presented in this work with the parameters mentioned in Section 5.3 using UD-SR, UD-ZCopy, RC-SR, RC-FP, and RC-RDMA.

In addition, each of the configurations uses the same shared memory channel for all communication to peers on the same node.

In terms of channel designs, our Aptus prototype is based on MVAPICH-UD and MVAPICH, however, the codebase itself is almost entirely new. It is implemented as a device layer for MPICH.

For our evaluation we collect message statistics for each message channel. We track both the number of messages sent and the data volume sent over each channel. In addition, we track this on a per-peer basis to determine how many message channels are allocated and to what peers. We also determine the message distribution of the application from within the MPI library, including control messages. Figure 7 shows these distributions with darker blocks denoting larger messages. For example, 95% of messages for LU are greater than 512 bytes, however, very few are greater than 4KB.

6.1. NAS Parallel Benchmarks

The NAS Parallel Benchmarks [4] (NPB) 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 evaluate using the largest of the problem datasets, Class 'D'. We run CG, EP, FT, LU, and MG with 512 tasks and both BT and SP with 529 tasks.

Figure 6 shows execution time normalized to RC of the NAS Benchmarks. In each case the Aptus prototype maintains equal or better performance than the other configurations. We note that in general UD-Copy performs the worst since large messages incur intermediate copy overheads.

For both CG and LU the RC configuration outperforms that of UD. Figure 7(b) shows the percentage of messages sent over each message channel. We observe that over 40% of messages are able to be transferred over the low-latency RC-FP channel, which is not available in a UD-only implementation. For CG, we note from Figure 7 that over 30% of messages are over 256KB, where RC-RDMA can provide a significant benefit. Aptus takes the benefits of both RC-FP for small messages and RC-RDMA for large messages, while using UD-SR for less frequently communicating peers. As a result, for both LU and CG, Aptus performs 2% better than RC and 4% better than UD.

In other benchmarks, FT in particular, we note that UD outperforms the RC configuration. From Table 2 we note that FT performs communication with all peers and in the RC case will require 504 RC QPs, leading to QP thrashing as the HCA ICM cache is exceeded. By comparison, the UD QPs in the UD configuration will remain in the ICM cache and lead to increased performance, 9% in this case. Aptus shows a small improvement over UD since a few large messages can be sent using RC, reducing the load on the host.

6.2. NAMD

NAMD is a fully-featured, production molecular dynamics program for high performance simulation of large biomolecular systems [18]. NAMD is based on Charm++ parallel objects, which is a machine independent parallel programming system. Of the various data sets available with NAMD, we use the one called `f1atpase`. We evaluate with 512 tasks.

From Figure 6 we observe that NAMD performs much better (10%) using the UD transport than the RC transport. From Table 2 we observe that each NAMD task communicates with a large number of peers (120.8) on average. With so many communicating peers RC will overflow the ICM cache. In addition, many of the messages are below the 2KB segmentation threshold for UD-SR where performance exceeds that of RC-SR. Aptus performs similarly to UD with a 10% improvement over RC, which can be explained by Figure 7(b), which shows nearly 80% of messages are sent over UD-SR, meaning Aptus receives little benefit from RC-RDMA or RC-FP.

Note that although we have set the threshold for the number of RC-SR/RC-RDMA channels to 16, both NAMD and SMG2000 have an average above 16. This is due to the handshake involved in connection setup and the fact that both sides must create the connection. This is a benign race condition that can lead to at maximum two additional connections.

6.3. SMG2000

SMG2000 [5] is a parallel semi-coarsening multigrid solver, which is part of the ASC Purple Benchmarks. We run SMG2000 for 512 tasks and report the solve time produced by the benchmark.

Figure 6 shows the execution time of SMG2000 nor-

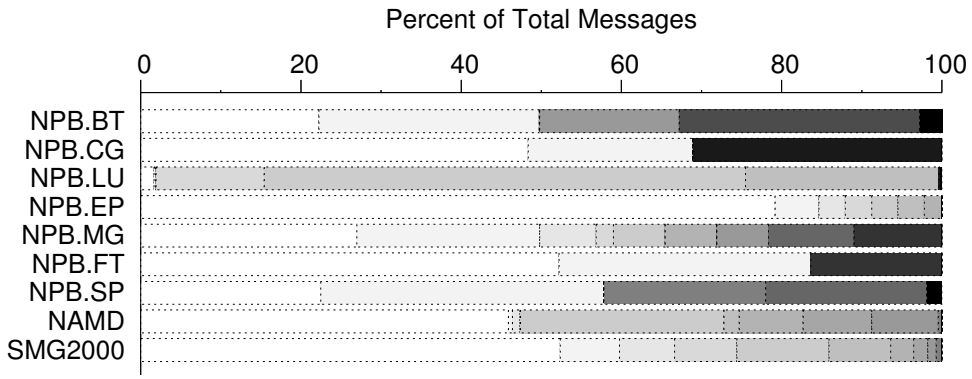


Figure 5. Message Size Distribution: Darker blocks denote larger message sizes (Brightness Index: 100%: $\leq$ 64 bytes, 50% $\leq$ 4KB, 25% $\leq$ 64KB , 0% $\geq$ 512KB)

malized to RC, where we observe clear differences between each of our configurations. Aptus delivers the highest performance, a full 12% over RC and 4% over UD. As with NAMD, from Table 2, SMG2000 communicates on average with over 120 peers of the 512 in the job. Such a large number of peers favors the UD transport, which we observe in the increased performance of UD over RC. Aptus, however, further improves on the performance of UD by using RC-SR and RC-RDMA for larger messages. From Figure 7(a) we observe Aptus is able to send 50% of data volume over one of the RC-based message channels, which have better bandwidth than UD-SR for messages over 2KB.

7. Related Work

Scalability of MPI libraries over InfiniBand has been a topic of much recent research. Many of these research works focus on reducing communication buffer requirements by utilizing SRQ [22, 19, 20]. In addition, the Reliable Connection memory utilization has been tackled in [9]. Yu, et al. proposed a connection setup method where UD was used for the first sixteen messages before an RC connection was setup [27]; in this case RC was the primary transport and no tradeoffs were evaluated. Further, connection-less UD MPI designs have been proposed in [10]. The zero-copy protocol over UD was introduced in [11]. Each of these works has been either on RC or UD; to the best of our knowledge, our work is the first to show the combination of both RC and UD.

MPIs that dynamically use all available interconnects have been designed in the past, including Intel MPI, Open MPI, Scali MPI, and others. Research work has been done with Mad-MPI [3] from the Madeleine project that seeks to incorporate multiple interconnects and provides scheduling across them. Our work is different in that we are targeting different transports on the same network device, and optimizing for the memory footprint and performance.

Current MPI designs over InfiniBand such as MVAPICH, Open MPI, and Intel MPI offer dynamic creation of RC QPs as needed; however, none of them include support for both the UD and RC transports simultaneously and cannot limit the number of RC QPs that are created. If a peer communicates with all others in the job a QP will be created to each one. Our design by contrast allows the amount of allocated resources to be limited.

The next-generation Mellanox HCA, ConnectX, provides a new transport, eXtended Reliable Connection (XRC) [24, 14]. XRC allows a single QP to be shared among tasks on the same node to potentially reduce memory usage. Driver support for XRC is still being tested and we hope to evaluate it in the future.

8. Conclusion and Future Work

As high-performance systems continue to scale, the need for the MPI library to maintain scalable in resource usage as well as performance is essential. For large-scale clusters an MPI that uses the RC transport exclusively can use up to 200 MB/task at 16K tasks, limiting the scalability and available memory. By contrast an MPI design has been proposed that uses the UD transport exclusively and has near-constant memory usage; however, performance is reduced compared to that of RC-based MPIs for some applications.

In this work we bridge this gap between RC and UD designs of MPI over InfiniBand. We first examine all available message passing channels available over InfiniBand and evaluate the performance as well as various scalability limitations for each. Based on the results from this evaluation we form a hybrid design, MVAPICH-Aptus,

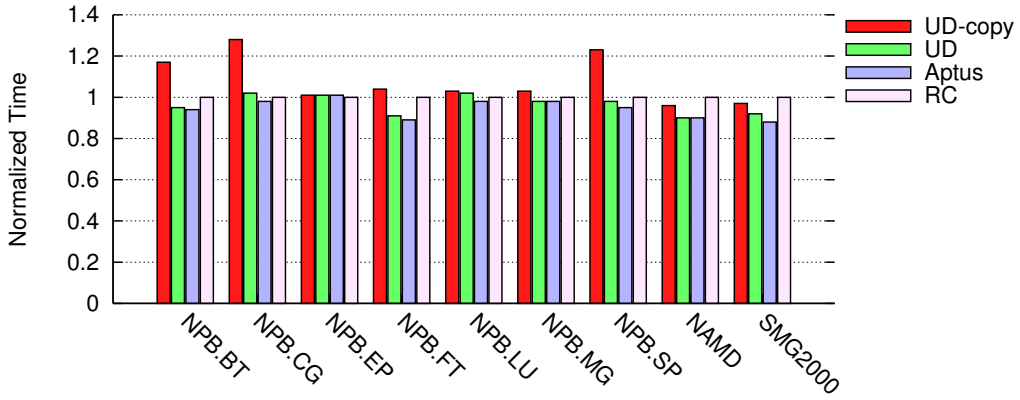
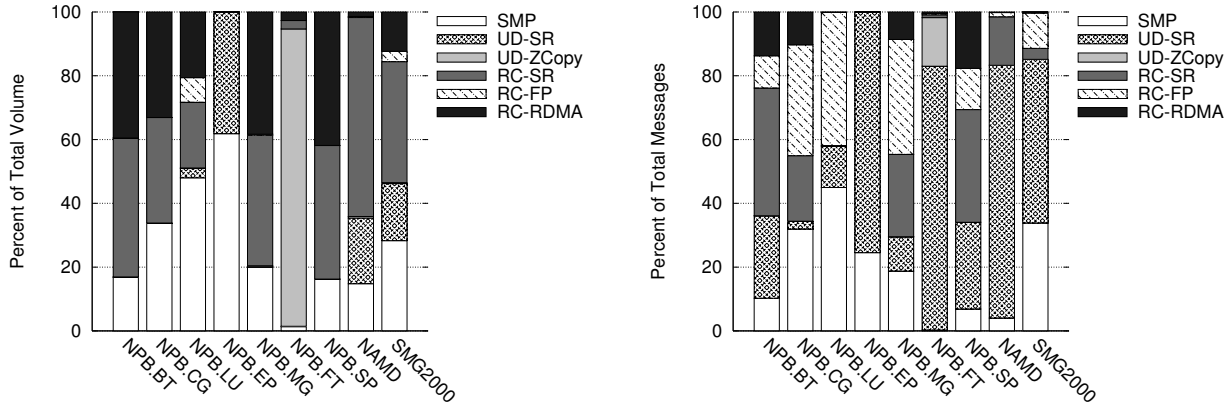


Figure 6. Application Benchmark Performance

Table 2. Average Number of Channels Used/Allocated Per Task (Aptus)

App.	Message Channels			
	SMP	UD-{SR,ZCopy}	RC-{SR,RDMA}	RC-FP
NPB.BT	4.11	20.17	10.60	7.88
NPB.CG	3.00	6.94	2.94	2.94
NPB.EP	3.00	6.00	0.00	0.00
NPB.FT	7.00	504.00	16.00	8.00
NPB.MG	4.31	9.00	5.63	5.63
NPB.LU	3.75	7.06	2.23	2.23
NPB.SP	4.11	20.17	10.62	7.88
NAMD	6.30	120.80	16.47	8.00
SMG2000	4.25	120.19	16.34	8.00



(a) Data Transfer Volume

(b) Number of Messages

Figure 7. Aptus Channel Usage Distributions

to utilize each of these message channels. We introduce send rule chains to efficiently use each channel.

As part of our work we develop and evaluate a prototype of our design. Our evaluation on 512 tasks reveals significant performance gains. By using each message channel within its optimal range we outperform both MPIs that are limited to only one transport of InfiniBand. We observe a 12% improvement over an RC-based design and 4% better than a UD-based design for the SMG2000 application benchmark. In addition, for NAMD we show a 10% improvement over an RC-only design. Since only a handful of RC-based resources are allocated, memory usage remains within a few MB of that of an exclusively UD-based MPI.

In the future we seek to explore more dynamic management of message channels. Some application codes use adaptive methods and change communicating peers, and we plan to explore methods to tear down and reallocate channels for these types of applications. We also plan to evaluate our design at larger scales. In addition, we plan to compare performance and scalability with the proposed XRC transport of InfiniBand.

9 Acknowledgments

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

References

[1] Mellanox Technologies. http://www.mellanox.com.

[2] TOP 500 Supercomputer Sites. http://www.top500.org.

[3] O. Aumage, L. Bougé, L. Eyraud, G. Mercier, R. Namyst, L. Prylli, A. Denis, and J.-F. Méhaut. High performance computing on heterogeneous clusters with the madeleine ii communication library. *Cluster Computing*, 5(1):43–54, 2002.

[4] 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.

[5] P. N. Brown, R. D. Falgout, and J. E. Jones. Semicoarsening multigrid on distributed memory machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000.

[6] L. Chai, A. Hartono, and D. K. Panda. Designing Efficient MPI Intra-node Communication Support for Modern Computer Architectures. In *Proceedings of Int'l IEEE Conference on Cluster Computing*, September 2006.

[7] 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.

[8] InfiniBand Trade Association. InfiniBand Architecture Specification. http://www.infinibandta.com.

[9] M. Koop, T. Jones, and D. K. Panda. Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach. In *7th IEEE Int'l Symposium on Cluster Computing and the Grid (CCGrid07)*, Rio de Janeiro, Brazil, May 2007.

[10] M. Koop, S. Sur, Q. Gao, and D. K. Panda. High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters. In *21st ACM International Conference on Supercomputing (ICS07)*, Seattle, WA, June 2007.

[11] M. Koop, S. Sur, and D. K. Panda. Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram. In *IEEE Int'l Conference on Cluster Computing (Cluster 2007)*, September 2007.

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

[13] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing (ICS '03)*, June 2003.

[14] Mellanox Technologies. ConnectX Architecture. http://www.mellanox.com/products/.

[15] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.

[16] Network-Based Computing Laboratory. MVAPICH: MPI over InfiniBand and iWARP. http://mvapich.cse.ohio-state.edu.

[17] OpenFabrics Alliance. OpenFabrics. http://www.openfabrics.org/.

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

[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] G. M. Shipman, R. Brightwell, B. Barrett, J. M. Squyres, and G. Bloch. Investigations on infiniband: Efficient network buffer utilization at scale. In *Proceedings, Euro PVM/MPI*, Paris, France, October 2007.

[21] G. M. Shipman, T. S. Woodall, G. Bosilca, R. L. Graham, and A. B. Maccabe. High performance RDMA protocols in HPC. In *Proceedings, 13th European PVM/MPI Users' Group Meeting*, Lecture Notes in Computer Science, Bonn, Germany, September 2006. Springer-Verlag.

[22] 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.

[23] 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.

[24] S. Sur, M. Koop, L. Chai, and D. K. Panda. Performance Analysis and Evaluation of Mellanox ConnectX InfiniBand Architecture with Multi-Core Platforms. In *15th IEEE Int'l Symposium on Hot Interconnects (HotI15)*, August 2007.

[25] S. Sur, A. Vishnu, H. W. Jin, W. Huang, and D. K. Panda. Can Memory-Less Network Adapters Benefit Next-Generation InfiniBand Systems? In *Hot Interconnect (HOTI 05)*, 2005.

[26] Texas Advanced Computing Center. HPC Systems. http://www.tacc.utexas.edu/resources/hpcsystems/.

[27] W. Yu, Q. Gao, and D. K. Panda. Adaptive Connection Management for Scalable MPI over InfiniBand. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.