

High Performance MPI Design using Unreliable Datagram for Ultra-Scale InfiniBand Clusters

Matthew J. Koop Sayantan Sur Qi Gao Dhableswar K. Panda

Network-Based Computing Laboratory
The Ohio State University
Columbus, OH 43210

{koop, surs, gaoq, panda}@cse.ohio-state.edu

ABSTRACT

High-performance clusters have been growing rapidly in scale. Most of these clusters deploy a high-speed interconnect, such as InfiniBand, to achieve higher performance. Most scientific applications executing on these clusters use the Message Passing Interface (MPI) as the parallel programming model. Thus, the MPI library has a key role in achieving application performance by consuming as few resources as possible and enabling scalable performance. State-of-the-art MPI implementations over InfiniBand primarily use the Reliable Connection (RC) transport due to its good performance and attractive features. However, the RC transport requires a connection between every pair of communicating processes, with each requiring several KB of memory. As clusters continue to scale, memory requirements in RC-based implementations increase. The connection-less Unreliable Datagram (UD) transport is an attractive alternative, which eliminates the need to dedicate memory for each pair of processes.

In this paper we present a high-performance UD-based MPI design. We implement our design and compare the performance and resource usage with the RC-based MVAPICH. We evaluate NPB, SMG2000, Sweep3D, and sPPM up to 4K processes on an 9216-core InfiniBand cluster. For SMG2000, our prototype shows a 60% speedup and seven-fold reduction in memory for 4K processes. Additionally, based on our model, our design has an estimated 30 times reduction in memory over MVAPICH at 16K processes when all connections are created. To the best of our knowledge, this is the first research work that presents a high-performance MPI design over InfiniBand that is completely based on UD and can achieve near identical or better application performance than RC.

Categories and Subject Descriptors

D.4.4 [Operating Systems]: Communications Management; D.4.9 [Operating Systems]: Systems Programs and Utilities; J.0 [Computer Applications]: General

Keywords

Unreliable Datagram, MPI, InfiniBand, Memory Scalability

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICS '07 June 18-20, Seattle, WA, USA.

Copyright 2007 ACM 978-1-59593-768-1/07/0006 ...\$5.00.

1. INTRODUCTION

Cluster computing has grown significantly in popularity in recent years. High-performance interconnects are often deployed to obtain increased performance. InfiniBand [13] is an increasingly popular interconnect in cluster computing due to low latency (1.0-3.0 μ sec) and high bandwidth, as well as other features such as Remote Data Memory Access (RDMA). Many large clusters, such as the 9024-processor Sandia Thunderbird, NASA/Ames Columbia, and the upcoming 9216-core LLNL Atlas cluster use InfiniBand as their primary interconnect. The trend in HPC is clusters of rapidly increasing size, so issues seen on current generation clusters are likely to be of even greater importance for future machines.

The Message Passing Interface (MPI) [22] is the dominant parallel programming model on these large clusters. Given its involvement in many applications, the MPI library has a key role in providing scalability in both resource usage and message passing performance. State-of-the-art MPI implementations over InfiniBand primarily use the Reliable Connection (RC) transport of InfiniBand since it is the most feature-rich – supporting high-throughput, reliability, atomic operations, and operations for zero-copy transfers of large buffers. The popular open-source MPI implementations over InfiniBand, MVAPICH [19] and Open MPI [8], use this transport layer for communication. As such, research studies on MPI implementations over InfiniBand have focused on managing resources for implementations using the RC transport. Strategies such as lazy connection setup and using the Shared Receive Queue (SRQ) support of InfiniBand have been employed to reduce resource consumption [33, 27, 26].

Using the RC transport, however, has the drawback that the worst-case memory usage per process of the MPI library across the cluster increases linearly with increasing processes. Since each connection requires several KB of memory and clusters continue to scale to tens of thousands of processors and above, the connection-less Unreliable Datagram (UD) transport of InfiniBand is a potentially attractive alternative. As a connection-less transport, the maximum memory used for connections across the cluster static even as the number of processes increases. Using the UD transport for an MPI library, however, brings several challenges, including providing reliability. Additionally, the UD transport limits the maximum message size to the Message Transfer Unit (MTU) size and lacks support for RDMA. The MTU on current Mellanox [1] hardware is 2KB.

In this paper we analyze the design alternatives of the MPI library over the UD transport of InfiniBand. We propose three messaging protocols that provide reliability without sacrificing performance, and while quickly reacting to the expected number of packet drops on high-performance networks. We prototype our design and

compare the performance and resource usage of our prototype versus the RC-based MVAPICH. We evaluate the NAS Parallel Benchmarks, SMG2000, Sweep3D, and sPPM up to 4K processes on an 1152-node InfiniBand cluster. For SMG2000, our prototype shows up to a 60% speedup and seven-fold reduction in memory for 4K processes. Additionally, based on an analytical model, our design has a potential 30 times reduction in memory usage over MVAPICH at 16K processes. To the best of our knowledge, this is the first research work that presents a high performance MPI design over InfiniBand that is completely based on UD and can achieve near identical or better application performance as compared to RC.

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 motivate our work by evaluating the memory usage of the different InfiniBand transports and examining the connection management methods of current MPI implementations. Our UD-based design is presented in Section 4, and the details of our prototype implementation are noted in Section 5. We evaluate our prototype in Section 6. Related work is noted in Section 7; we conclude and discuss future work in Section 8.

2. INFINIBAND

In this section we give the required background information on InfiniBand. We start with an overview, explain the communication model, and available transport services.

InfiniBand was designed 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

A Queue Pair (QP) model is used for all communication in InfiniBand. A QP consists of two queues, a Send Queue (SQ) and a Receive Queue (RQ) for initiating send and receive operations, respectively. Each QP is associated with a Completion Queue (CQ), allowing an application to poll or use an event-based interface to receive notification of operation completion.

There are two sets of communication semantics in InfiniBand: channel and memory semantics. Channel semantics include send and receive operations that are similar to those found in traditional interfaces, such as sockets, where both sender and receiver must be aware of communication. Memory semantics include one-sided operations where one host can access or modify memory on a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA).

In both sets of semantics all memory used for communication must be registered and pinned, not to be swapped out. Since the startup cost for registering memory is high, small messages are generally copied into pre-registered buffers before being sent to the receiver for optimal performance. At larger sizes, it can become advantageous to perform a zero-copy transfer by directly registering the application send and receive buffers and performing an RDMA operation.

To receive a message on a QP using channel semantics, a receive buffer must be posted to that QP. Buffers are consumed in a FIFO ordering. Using a Shared Receive Queue (SRQ) allows receive buffers to be shared across QPs for scalability.

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 must be dedicated to communicating with only one other QP. An application that communicates with N other peers must have at least N QPs created. It provides RDMA capability, atomic operations, and reliable service.

Unreliable Connection (UC) provides connection-oriented service with no guarantees of ordering or reliability. It does support RDMA write capabilities and sending messages larger than the MTU size. Since the memory requirements are identical to that of RC, it appears unattractive for MPI communication.

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. UD does have a number of limitations, however. Messages larger than an MTU size, which on current Mellanox hardware is limited to 2KB, cannot be directly sent using UD. Only channel semantics are defined for UD, so RDMA is not supported. Additionally, UD does not guarantee reliability or message ordering.

3. MOTIVATION

In this section we will describe the factors motivating this work, including resource scalability as well as potential performance improvements that can be achieved with using the UD transport.

3.1 Resource Scalability

In this section we examine the memory usage of the MPI library and the resource requirements associated with a connection-oriented transport. We first classify the memory usage of the MPI library into three main categories and then examine each with regards to RC and UD.

- **Data Structure Memory:** Even when a connection is not created, memory allocations for data structures increase with job size.
- **Communication Context / QP Memory:** The memory required for QP contexts we refer to as the communication context memory. This memory requirement is the focus of this work.
- **Communication Buffer Memory:** Registered memory that is used for send or receive operations. This does not include user buffers that are registered to provide zero-copy communication, only those used for eager or packetized communication.

3.1.1 Data Structure Memory

Regardless of the transport type, the memory allocated for data structures usually grows with the number of total processes in a job. Although this memory usage per process grows linearly with increasing numbers of processes, the coefficient is quite low, generally much less than 1KB.

3.1.2 Communication Context / QP Memory

Communication context memory is the memory required for QP contexts. With the RC transport a separate dedicated QP must be created for each communicating peer. Unlike the memory required for data structures, the memory required for each additional QP is

considerably greater. The resources required for a single QP context, as created with the default settings of MVAPICH 0.9.8, consume nearly 68KB of physical memory.

In an attempt to minimize this memory, current MPI implementations over InfiniBand, including MVAPICH and Open MPI, employ an on-demand [32, 33] connection setup method to only setup connections and QPs as required. Thus, if an application communicates with only a few peers the number of connections and QPs required per process may be considerably fewer than the number of total processes in a job. This method, however, is only beneficial when the number of communicating peers is low. If an MPI application communicates with more than a small number of peers, memory usage will grow with the total number of processes in the job, potentially reaching above 1GB of memory per process for 16K processes. In [32, 30], SMG2000 [6] is found to have up to the same number of connections per process as total processes. With increasing scale the MPI library should be able to maintain a reasonable memory footprint regardless of the number of communicating peers.

Using UD as a transport, however, solves this significant problem of QP memory increasing with the number of peers. Since a single UD QP can communicate with any number of other UD QPs each MPI process need only allocate a single QP. Thus, as the number of communicating peers increases the connection memory remains constant. This reduction in connection memory can significantly increase the amount of memory available for the application.

3.1.3 Communication Buffer Memory

As mentioned in Section 2.1, to receive a message using channel semantics (send/receive), the receiver must post a receive buffer to the associated QP. To maintain high-performance, buffers are generally pre-posted to each QP; posting buffers per QP for a connection-oriented transport, like RC, however, requires significant memory usage. This prompted the InfiniBand specification to be updated to include support for Shared Receive Queues (SRQs), which allow receive buffers to be shared across QPs. Thus, a single pool of posted receives can be used for any number of QPs and MPI peers.

UD can also make use of an SRQ; however, it is not necessary since all receives can be posted directly to the single UD QP. Thus, for both RC and UD, the communication buffer memory can be allocated as a pool and does not depend directly on the number of processes in a job.

3.2 Performance Scalability

Performance can also potentially be improved using a connection-less and unreliable transport through better HCA cache and fabric utilization.

3.2.1 InfiniBand Context Memory Caching

InfiniBand HCAs cache communication context 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. Sur, et al. in [29] provide an analysis of the Mellanox MT25218 HCA show that less than 128 QPs can be stored in the cache at any one time. Furthermore, the authors show that when a QP is not in the ICM cache there is a significant latency penalty; for transfers less than 1KB the latency nearly doubles. This is a significant problem, especially considering multi-core machines where all processes on a node will be sharing the cache. Using a connection-less transport, a single QP can communicate with many peer QPs, avoiding cache thrashing when communicating with many other QPs.

3.2.2 Fabric Utilization

When using a reliable transport, the HCA must provide reliability to guarantee that packets are not lost and are delivered in order. Providing this service at the transport layer instead of the application layer does not allow the application to optimize for out-of-order messages. For example, in MPI it is not necessary for all parts of a large message transfers to arrive in order. Instead, they can be placed in the user buffer as they arrive and not dropped by the HCA. Additionally, the lack of acknowledgements (ACKs) at the transport layer means less traffic overhead on the InfiniBand fabric. The upper layer can send ACKs in a more lazy method or another optimized way depending on application needs.

4. PROPOSED DESIGN

Providing a high-performance and scalable MPI over UD requires a careful design since many features, including reliability, lack of RDMA, and a small MTU, are provided by hardware when using the RC transport are now not available or must be done in the MPI library.

4.1 Overview

Our UD-based design is different from MVAPICH and Open MPI in that it uses UD QPs for data transfer rather than RC. Using the connection-less UD transport allows much better memory scalability for large-scale clusters.

Figure 1 shows the resource allocations of our proposed design versus the design used in MVAPICH. In MVAPICH each peer requires a data structure and optionally a QP, created when communication is initiated during application execution. In the worst-case a QP will be allocated for each peer in this situation. Our UD-based design by contrast uses a single UD QP for all peers since it is a connection-less transport. In both designs the CQ and communication buffers are shared across all connections.

4.2 Reliability

A key issue that must be addressed when using an unreliable transport layer is how to provide reliability in a lightweight and scalable form. Significant prior work has been done on methods to provide reliable service at the transport layer; however, in-depth study has not been done at the MPI library layer. LA-MPI [10, 3] has explored reliability against loss and corruption in the MPI library, but was more focused on I/O bus errors. We discuss this related work in Section 7.

Our design uses the traditional sliding window protocol. The sender issues packets in order as there is available space in the window. In this manner the window represents the maximum number of unacknowledged message segments outstanding. Additional send operations occurring when the send window is already full are queued until outstanding operations have been acknowledged.

At the time of each send operation, a timestamp is associated with each message segment. To maintain high-performance, we use the Read Time Stamp Counter (RDTSC) assembly instruction to read the hardware counters for these values. If an ACK has not been received within a given timeout the segment is retransmitted.

4.2.1 Optimizations

We additionally use a negative acknowledgment (NACK) based protocol to request selective retransmissions. Upon receiving a message out of order we assume the skipped messages have been lost and request re-transmission from the sender. Using this mechanism we can handle potential message drops without having to wait for timer expiration. We also borrow the traditional ACK coalescing method from TCP to reduce the number of acknowledgments.

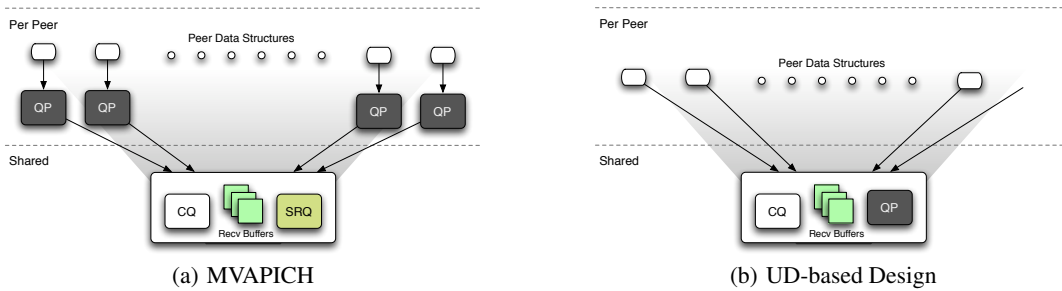


Figure 1: Resource Allocation

4.2.2 Reliability Progress Methods

The main challenge in providing reliability at the user-level within the MPI library is the potential lack of progress. In a traditional sliding window sender retransmission protocol, message acknowledgments must be sent within a timeout period otherwise the message is retransmitted from the sender. There are two issues to consider:

- Sender Retransmission: How should the sender know when to retransmit a message? Should there be a timer-based interrupt or a thread?
- Receiver Acknowledgment: How can the receiver acknowledge the message within a bounded amount of time?

While providing these guarantees may seem simple, providing them while maintaining performance is non-trivial. We present three different methods of providing reliable service within the MPI library.

Thread: This is the traditional approach in which a watchdog thread sleeps for a set amount of time before waking to process incoming messages, send ACKs, and resend timed-out messages. This method provides a solution to both of the issues of sender and receiver progress, but it does so at significant cost. To maintain a quick response to a segment drop the thread wakeup time, t_{wake} , must be set to a short interval. A thread that wakes up frequently may harm performance. Figure 2(a) demonstrates this protocol.

Progress-Engine: The progress engine of MPI refers to the core of the library that is activated to send and receive messages – to make progress on communication. In this variant, reliability is only handled in the progress engine. This has a number of advantages including the lack of the overhead of locks for various data structures that otherwise need protection. Additionally, for infrequent packet drops, there is no need to have a thread wake up frequently, which causes potential performance degradation. Since we are relaxing the timeframes for retransmission and acknowledgment, the disadvantage of such a design is that an ACK may not be returned within the timeout period since the receiver may be in a computation loop (not discovering the message), triggering an unnecessary retransmission from the sender. Additionally, sender retransmission may also be delayed if the sender is in a long compute loop. Figure 2(b) shows this protocol when the receiver is in a long compute loop and an unnecessary retransmission is generated.

Hybrid: In this method the reliability is primarily through the progress engine; however, if the timer runs out the message is retransmitted to the receiver side to a secondary UD QP that is interrupt-driven. When a message is received on the interrupt-driven QP a thread is awoken to process the message and acknowledge all other waiting messages. This guarantees an ACK will be

sent back within a Round Trip Time (RTT) if received, reducing unnecessary multiple retries while avoiding the overhead of a thread waking at a constant rate. Figure 2(b) shows the benefit of this method when the receiver is in a compute loop, avoiding multiple retransmissions. Using an interrupt-driven QP for all communication is not feasible since interrupts are costly operations that harm performance of both communication and computation.

Note that implementations of both the *thread* and *hybrid* designs require locking of internal data structures to ensure thread safety. Using the *progress engine* method avoids this as only one thread is only ever executing within the progress engine.

5. IMPLEMENTATION

We have implemented our proposed design over the verbs interface of the OpenFabrics/Gen2 stack [24]. Our prototype design is based on MPICH [11] from Argonne National Laboratory. MPICH defines an Abstract Device Interface (ADI), that allows the device and transport specific information to be encapsulated. We develop an ADI component based on the MVICH [16] and MVAPICH ADI components. MVICH was developed at Lawrence Berkeley National Laboratory for the VIA [7] interconnect. MVAPICH is a derivative of MVICH from the Ohio State University that has been significantly redesigned and optimized for InfiniBand. Both MVICH and MVAPICH were created for reliable connection-oriented transports, requiring a significant portion of the code path to be re-written to support UD.

We implement all three of the reliability methods described in the design section as well as a profiling engine to detect message retransmissions, duplicate messages, and message sizes. We compute message drop rates by comparing the number of re-transmissions from a given process to another and the number of duplicate messages the process detected from the other. This data is collected during the finalize stage.

6. EVALUATION

Our experimental testbed is an 1152-node, 9216-core, InfiniBand Linux cluster at Lawrence Livermore National Laboratory (LLNL). Each compute node has four 2.5 GHz Opteron 8216 dual-core processors for a total of 8 cores. Total memory per node is 16GB. Each node has a Mellanox MT25208 dual-port Memfree HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [24]. The Intel v9.1 compiler is used for all compilations, including MPI libraries and applications.

We compare the performance of our UD-based prototype versus the popular MVAPICH MPI library, version 0.9.8. Comparisons with other MPIs were not possible due to limited time on the cluster. For MVAPICH, we disable small message RDMA, since per process it requires roughly 512KB of memory per additional con-

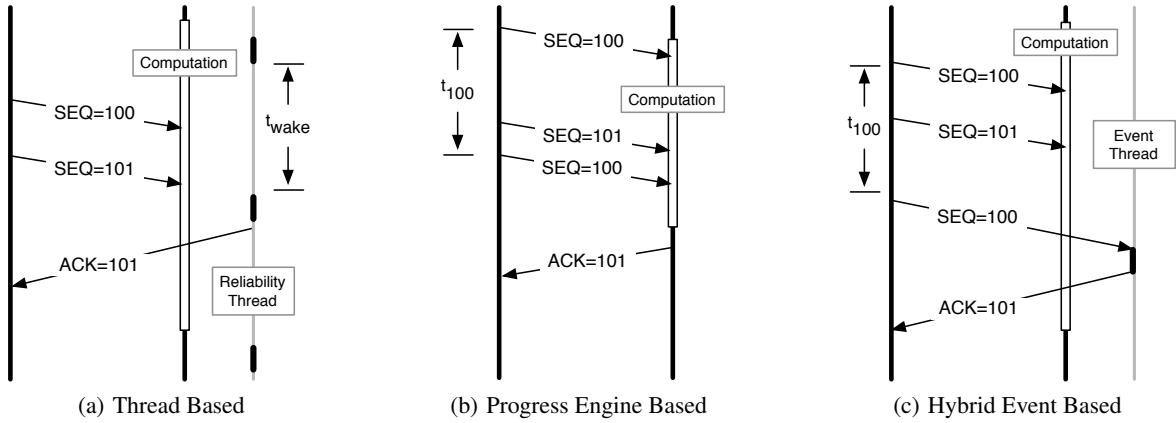


Figure 2: Reliability Protocols

nected process; large message RDMA for zero-copy transfers is still enabled. We disable this feature since we are interested in ultra-scale clusters. Earlier studies [28] have shown the SRQ path to have similar or better performance for applications than the small message RDMA path while using significantly less memory. Small message RDMA, with lower latency ($3.0\mu\text{sec}$ for RDMA, $5.0\mu\text{sec}$ for SRQ), may be beneficial for smaller clusters, but is not scalable for large-scale systems due to the large per process memory usage.

Additionally, although by default MVAPICH posts 512 buffers to the shared SRQ for receiving messages, we had to increase that value to 8K to ensure optimal performance for some applications on this larger cluster.

6.1 Memory Usage

One of the primary benefits of using UD as the transport protocol is the reduced memory usage, as noted in Section 3.1. In this section we describe our methodology and compare the memory usage of the entire MPI library at increasing scale.

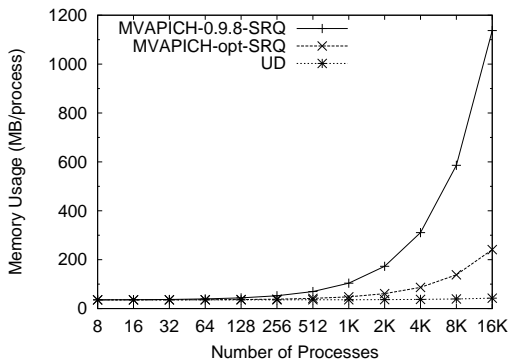


Figure 3: Fully-Connected Memory Usage

Figure 3 compares the memory usage of our proposed design as compared to MVAPICH with increasing numbers of connected peers. Values through 4K processes are measured, 8K and 16K values are modeled. From the figure we observe that MVAPICH has the potential to use 1.1GB of memory per process at 16K processes per job, and our prototype uses only 40MB per process. Expanding to total connection memory usage across the cluster, which is what scientists use to determine the problem size able to be run, the

Table 1: Additional Memory Required with additional processes and connections

MPI	Memory Required Per Process	
	Addl. Peer	Addl. Connection
MVAPICH-0.9.8	0.61KB	68KB
MVAPICH-opt	0.61KB	13KB
UD	0.44KB	0KB

default settings will consume 18TB of memory, and the UD-based design will use a much smaller 655GB of memory. This memory usage is the worst-case situation, where all peers are communicated with at least once. As mentioned in Section 3.1.2, many applications communicate with the majority of their peer processes, so the resource usage is important to consider and cannot be ignored.

Settings currently being used on our evaluation cluster for reduced memory usage are shown as “MVAPICH-opt”, these decrease the number of outstanding send operations allowed per QP as well as the amount of data that can be inlined with a request to decrease overall memory usage significantly from the 0.9.8 default. This decreases the total connection memory, however, we have not evaluated all possible effects. Our evaluation has been done with the default settings; we mention possible optimization here for completeness. Even using these optimized settings our UD-based design uses 80% less memory at 16K processes. A subsequent release of MVAPICH (0.9.9) has reduced memory usage below even the level of the optimized settings, however, UD remains the most scalable option.

We also calculate the memory usage of MVAPICH when the number of processes in a job increases, but no direct connections are made to any peers. This reflects only the *data structure* memory. Table 6.1 shows the results of this measurement. From this table we observe that our UD-based prototype uses even less memory in data structures than MVAPICH, meaning even with no connections setup the memory of our UD prototype uses less memory than MVAPICH as the number of processes in a job increases.

6.2 Basic Performance

Figure 4 shows the latency, bandwidth, and messaging rate of both MVAPICH (with SRQ) and our UD-based prototype using the *progress engine* based reliability method. From Figure 4(a) we observe that for small messages under the MTU size message la-

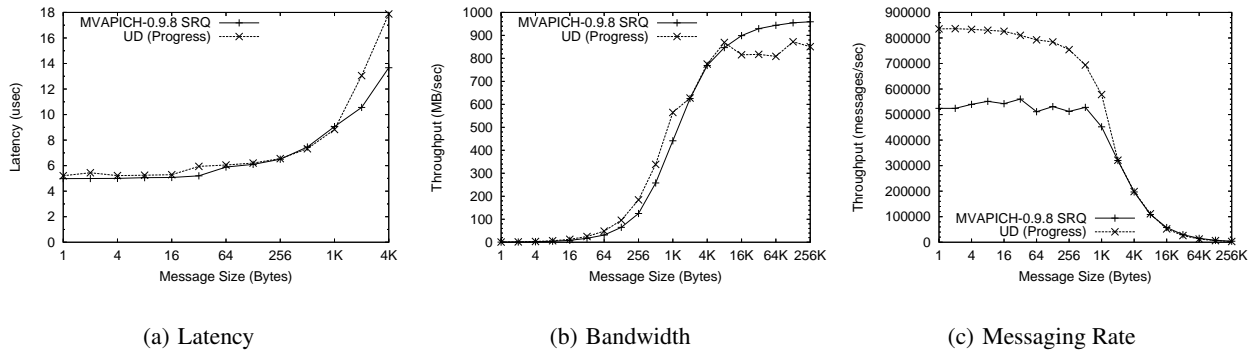


Figure 4: Basic Micro-Benchmark Performance Comparison

tency is nearly the same as MVAPICH. Messages above that size have a higher latency as the library must segment the message instead of segmenting in hardware. We also evaluate the latency penalty of using locks around all critical data structures for the *thread* and *hybrid* reliability methods and find it to have a $0.2 \mu\text{sec}$ additional overhead. Figure 4(b) shows the bandwidth comparison. For messages less than the MTU size our prototype shows increased throughput over MVAPICH. Throughput is roughly comparable until 8KB where MVAPICH maintains a bandwidth of 100-150 MB/sec higher. From Figure 4(c) we observe the messaging rate of our prototype is up to 40% higher for small messages. After studying the base InfiniBand performance with and without a SRQ, the lesser small message bandwidth and message rate for MVAPICH is due to the use of a SRQ. Recall from Section 3.1.3 that for scalability a SRQ is required when using RC.

6.3 Application Results

We evaluate with the NAS Parallel Benchmarks, and three applications from the Advanced Simulation and Computing (ASC) Benchmark Suite [2]: sPPM, Sweep3D, and SMG2000. We evaluate the NAS Parallel Benchmarks with each of our design alternatives as well as MVAPICH. Given limited large-scale cluster time, sPPM, Sweep3D, and SMG2000 are evaluated with only the *progress-engine* approach for reliability and MVAPICH. For the thread-based reliability method we set the thread-wake time to one second and a message timeout of two seconds. The progress engine and hybrid message timeout is set to 0.4 sec. All results are the average of multiple runs.

For each of the applications we evaluate the memory usage in all three categories: data structure, connection/context memory and communication buffer memory used by the MPI library. Additionally, we profile the MPI message sizes used by the application and directly communicating peers to better understand the results. Communication pattern analysis of sPPM, SMG200, and Sweep3D is available in [30] by Vetter, et al.

6.3.1 NAS Parallel Benchmarks

The NAS Parallel Benchmarks [4] (NPB) are kernels designed to be typical of various Computational Fluid Dynamics (CFD) MPI applications. As such, they are a good tool to evaluate the performance of the MPI library and parallel machines. We use the largest class (C) with datasets for all benchmarks. Table 2 shows the number of messages, RC connections, and UD packet drops observed for 256 processes. Additional study on communication patterns of NAS Parallel Benchmarks is available in [31].

We evaluate the performance of each of the benchmarks with both our UD prototype and MVAPICH for 256 processes on 256 nodes. Each of the design alternatives for reliability are evaluated for the UD prototype. Figure 5(a) shows the results of our evaluation. Comparing only the reliability methods for the UD prototype, the *progress-engine* approach gives the best observed performance for all benchmarks. The other techniques require additional locking overhead for thread safety. The thread-based technique has the highest overhead, sometimes reaching nearly 10%. We attribute this to additional system noise [25] incurred by the threads waking up and competing for computational cycles and locks. The hybrid approach incurs very little overhead as no retransmissions are made.

Comparing the UD prototype performance with MVAPICH we observe that our UD prototype performs quite well, performing with better or equal performance in all benchmarks besides CG and MG where performance is within 3%. The UD prototype performs particularly well for FT and IS, both of which make use of `MPI_Alltoall` operations, where an RC-based MPI will have to go outside the ICM cache to communicate with all other nodes.

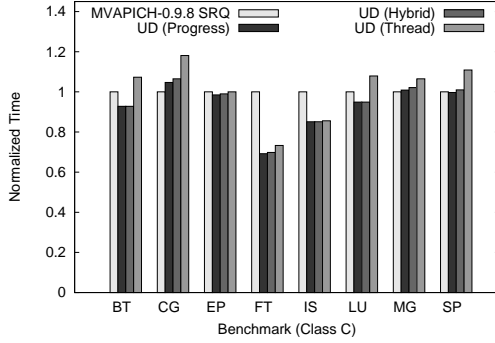
Figure 5(b) shows the memory usage of each of the benchmarks. As noted earlier, we measure the connection memory (for QPs), the communication buffers, and the memory required for data structures. We observe that even for MVAPICH, the connection memory is not significant for many of the benchmarks since each process has only a few communicating peers. IS and FT, however, connect to all 255 other peers, making their connection memory more significant when the RC transport is used. The UD-based prototype shows negligible connection memory in all cases since it is based on a connection-less transport. The communication buffer memory usage is higher for MVAPICH as each receive buffer posted is 8KB and those for our UD prototype are 2KB (the MTU size). Some applications that use a significant number of large messages, such as FT and IS, required additional memory above the base value for the UD prototype. Overall, including all forms of communication memory the UD prototype used lower memory for all benchmarks.

6.3.2 sPPM

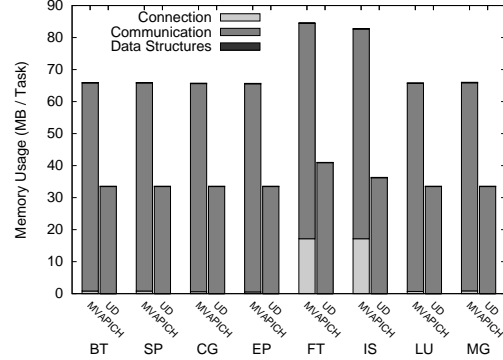
sPPM [23] is an application distributed as part of the ASC Purple Benchmarks. It solves a 3D gas dynamics problem on a uniform Cartesian mesh using the Piecewise Parabolic Method (PPM). Figure 6 shows a breakdown of MPI message volume according to size for this dataset at 4K processes. We observe from the figure that 40% of messages are 64 bytes or less and another 40% are larger than 256 KB.

Table 2: NAS Characteristics (256 processes)

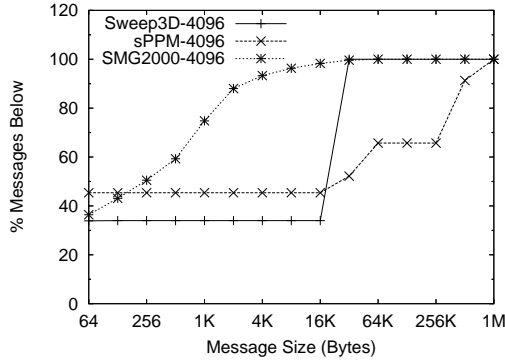
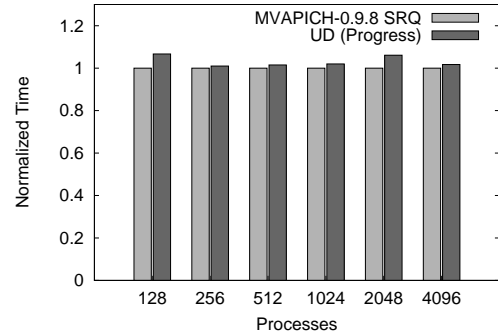
Characteristic	Benchmark							
	BT	CG	EP	FT	IS	LU	MG	SP
Total MPI Messages (millions)	4.96	13.31	0.01	1.47	1.50	77.31	1.68	9.88
RC: Connections (max per process)	12	9	8	255	255	10	13	12
UD: Total Packet Drops	0	0	0	0	0	0	0	0



(a) Normalized Time



(b) Memory Usage Per Process

Figure 5: Characteristics of NAS Benchmarks (256 processes, Class C)**Figure 6: Total MPI Message Size Distribution****Figure 7: sPPM Performance Comparison**

The performance of sPPM with increasing numbers of processes is shown in Figure 7. From the figure we observe that the performance between the default configuration of MVAPICH and our prototype is similar. For 4K processes, the UD prototype is within 1% of the performance of MVAPICH. The lower performance and lack of zero-copy for large message transfers with UD and the high percentage of messages over 256KB likely combine for the observed lower performance. Table 3 shows statistics related to the MPI messaging characteristics of sPPM. Memory usage per process is nearly constant, regardless of the total number of processes. Given the low number of communicating peers for each process, maximum 15, the MVAPICH RC connection memory is kept less than 1MB. Communication buffer memory per process is 68MB for MVAPICH and 36MB for our UD prototype. No packet drops were observed for any number of processes.

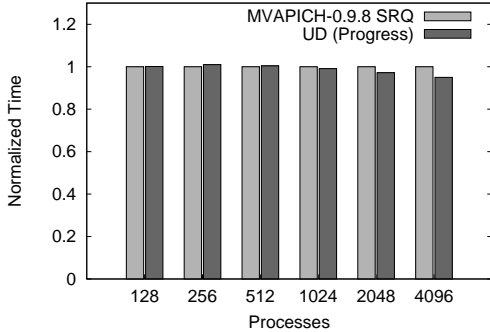
6.3.3 Sweep3D

Sweep3D [12, 14] uses a multi-dimensional wavefront algorithm to solve the 3D, time-independent, particle transport equation on an orthogonal mesh. Profiling of the MPI message sizes, shown in Figure 6 reveals that 38% of messages are 64 bytes or less and the remaining volume is between 32 and 64 KB in size.

Figure 8 shows the normalized time of running Sweep3D at increasing scale. Performance is roughly comparable, with the UD-based design getting slightly faster with increasing numbers of processes: 5% faster at 4K processes. From Table 3 we observe that as the number of processes in a job increases, the number of packet drops increases. The percentage is still low, at 4K processes only 8 packets were dropped. As with sPPM, the memory required for RC connections for MVAPICH is quite low with a maximum of 11. As such, the difference in memory required for the UD prototype and MVAPICH is minimal.

Table 3: Application Characteristics

Application	Characteristic	Processes					
		128	256	512	1024	2048	4096
sPPM	Total MPI Messages (millions)	0.11	0.24	0.52	1.08	2.25	4.74
	RC: Connections (max per process)	10	10	12	13	14	15
	UD: Total Packet Drops	0	0	0	0	0	0
Sweep3D	Total MPI Messages (millions)	0.17	0.32	0.63	1.28	2.65	5.48
	RC: Connections (max per process)	9	9	9	9	10	11
	UD: Total Packet Drops	0	0	0	0	1	8
SMG2000	Total MPI Messages (millions)	29.9	64.4	146.1	312.7	652.1	1376.6
	RC: Connections (max per process)	111	195	340	438	631	992
	UD: Total Packet Drops	0	0	0	0	10	27

**Figure 8: Sweep3D Performance Comparison**

6.3.4 SMG2000

SMG2000 [6] is a parallel semi-coarsening multigrid solver, which is part of the ASC Purple Benchmarks. Analysis of the MPI message distribution, as shown in Figure 6 reveals the majority of messages are quite small, 90% of MPI messages are less than 4KB.

From Figure 9(a) we observe superior performance for our UD-based design as the number of processes increases. At 4K processes our prototype is 60% faster than MVAPICH. As noted in Table 3, the number of connected peers is significant, nearly 1000 connected peers per process for 4K processes. Further inspection of the communication pattern shows a regular frequency of communication with many peers. As such, the QP context caching and management of ACKs at the application layer is the likely reason for the benefit, particularly since the size of each message is very small. Unfortunately, there is no direct mechanism to measure the cache misses on the HCA ICM cache. When sending to such a large number of peers the ICM cache is likely being thrashed for RC, while the UD QPs will remain in cache. We also observe increasing numbers of packet drops with scale; at 2K processes 10 packets are dropped and 27 packets are dropped for 4K processes.

Figure 9(b) shows the memory usage of SMG2000 with increasing numbers of processes. Since the number of connected peers increases with the overall number of processes, the connection memory similarly increases for MVAPICH. The UD prototype maintains constant connection memory. At 4K processes, the maximum number of RC connections for MVAPICH made by a single process is nearly 1000, requiring over 60MB of memory per process just for connections, totaling 240 GB of memory across the cluster.

7. RELATED WORK

The issue of memory consumption for connection-oriented transports has also been studied by other researchers. Gilfeather and Maccabe proposed a connection-less TCP method that activates and deactivates connections as needed to save memory [9]. Scalability limitations of the connection-oriented VIA interconnect were explored in [5] and an on-demand connection management method for VIA was proposed in [32].

InfiniBand, the successor of VIA, has also been studied in regards to resource usage. Early versions of MVAPICH exhibited significant memory usage as the number of connections increased as studied by Liu, et al in [17]. Yu, et al. proposed an adaptive connection setup method where UD was used for the first sixteen messages before an RC connection was setup [33]; in this case RC was the primary transport. Work by Sur, et al. in [27] significantly reduced the per connection memory usage in MVAPICH using InfiniBand Shared Receive Queue (SRQ) support and a unique flow control method. Similar techniques have been used in Open MPI by Shipman, et al in [26]. In both of these studies, the memory usage was reduced mostly through a reduction in communication memory buffer usage while still using the RC transport layer. Koop, et al., proposed a coalescing method that reduced the memory required for communication contexts when using the RC transport [15]. We instead focus on reducing the connection memory usage by leveraging the connection-less UD transport.

Other researchers have explored providing reliability at the MPI layer as part of the LA-MPI project [10, 3], however, their work had a different goal. LA-MPI is focused on providing network fault-tolerance at the host to catch potential communication errors, including network and I/O bus errors. Their reliability method works on a watchdog timer with a several second timeout, focusing on providing checksum acknowledgment support in the MPI library. Conversely, in this work we explore using the connection-less UD transport of InfiniBand to provide superior scalability and near-equal to better performance compared to RC for ultra-scale clusters.

UD has also been used in other work to support enhanced collective operations for MPI. Researchers have previously shown the benefit of using hardware-based multicast over UD to increase collective performance in MVAPICH [18, 20]. Mamidala, et al., has also shown the benefit of using UD for MPI_{Alltoall} operations [21]. Our work instead focuses on a complete MPI stack over UD for high-performance and scalable point-to-point performance.

8. CONCLUSIONS AND FUTURE WORK

As clusters continue their path toward larger numbers of processors the need for the MPI library to maintain scalability in resource usage as well as performance is of paramount importance. Infini-

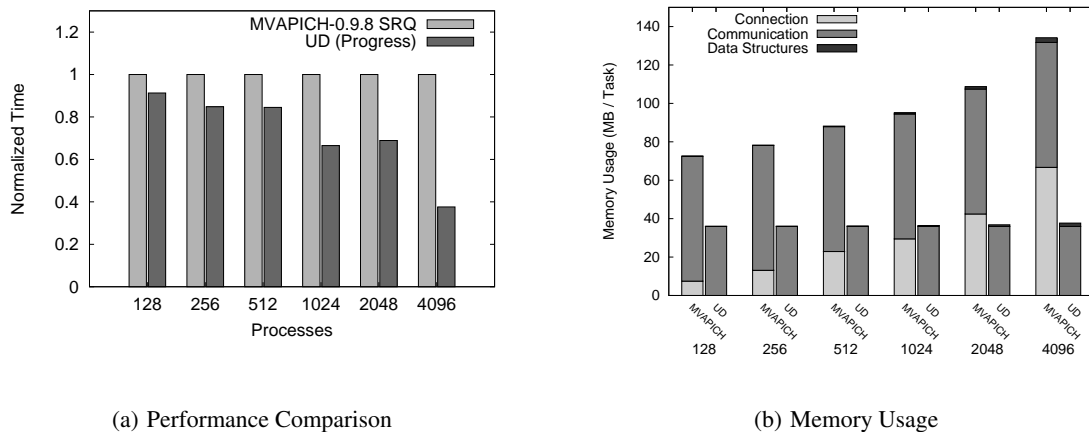


Figure 9: SMG2000 Characteristics with increasing processes

Band is an interconnect that has been growing in popularity and is being used in large-scale systems, including the Sandia Thunderbird, NASA/Ames Columbia, and LLNL Atlas clusters. Current state-of-the-art MPI implementations over InfiniBand generally use the Reliable Connection (RC) transport, which is inherently unscalable since significant additional resources are required for each additional communicating peer.

As clusters reach towards tens of thousands of processors and above, a connection-oriented design fails to remain scalable in resource usage. In particular, we find that connection memory usage for 16K processes will be 1.1GB per process, for a total of 19.2TB of memory used for connections across an entire job. This severely limits the problem size that can be run on a cluster since the available memory often determines the available problem size for scientific applications.

Our proposed design eliminates this resource issue by using the connection-less Unreliable Datagram (UD) transport of InfiniBand. Our design uses a near constant memory usage, using only 40MB of memory per process at 16K processes, a reduction of over 30 times – potentially freeing over 18.5TB of memory across the cluster for use by the application. Evaluation of our prototype shows similar performance as MVAPlCH for most applications and improved performance for those with many communicating peers, particularly as the number of processes increases.

In the future we hope to further assess the specific conditions where a UD-based MPI outperforms an RC-based MPI and vice-versa. Additionally, we plan to explore potential methods of increasing the bandwidth for large messages over UD, which remains unoptimized.

9. ACKNOWLEDGMENTS

This research is supported in part by Department of Energy's grant #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grant #CNS-0509452; grants from Intel, Mellanox, Cisco, Sun Microsystems, and Linux Network; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, IBM, Apple, Appro, Microway, PathScale, Silverstorm and Sun Microsystems.

10. REFERENCES

- [1] Mellanox Technologies. <http://www.mellanox.com>.
- [2] ASC. ASC Purple Benchmarks. <http://www.llnl.gov/asci/purple/benchmarks/>.

- [3] R. T. Aulwes, D. J. Daniel, N. N. Desai, R. L. Graham, L. Risinger, M. W. Sukalski, and M. A. Taylor. Network Fault Tolerance in LA-MPI. In *Proceedings of EuroPVM/MPI '03*, September 2003.
- [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. Venkatakrisnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.
- [5] R. Brightwell and A. B. Maccabe. Scalability Limitations of VIA-Based Technologies in Supporting MPI. In *Fourth MPI Developer's and User's Conference*, 2000.
- [6] 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.
- [7] Compaq, Intel, and Microsoft. VI Architecture Specification V1.0, December 1997.
- [8] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.
- [9] P. Gilfeather and A. B. Maccabe. Connection-less TCP. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 9*, 2005.
- [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. Sukalski. 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. Technical report, Argonne National Laboratory and Mississippi State University.
- [12] A. Hoisie, O. M. Lubeck, H. J. Wasserman, F. Petrini, and H. Alme. A General Predictive Performance Model for Wavefront Algorithms on Clusters of SMPs. In *International Conference on Parallel Processing*, pages 219–, 2000.
- [13] InfiniBand Trade Association. InfiniBand Architecture

- Specification. <http://www.infinibandta.com>.
- [14] K. Koch, R. Baker, and R. Alcouffe. Solution of the First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor. *Trans. of American Nuclear Society*, pages 65–, 1992.
- [15] 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)*, May 2007.
- [16] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. <http://www.nersc.gov/research/FTG/mvich/index.html>, August 2001.
- [17] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In *Supercomputing(SC)*, 2003.
- [18] J. Liu, A. Mamidala, and D. K. Panda. Fast and Scalable MPI-Level Broadcast using InfiniBand's Hardware Multicast Support. In *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS 04)*, April 2004.
- [19] 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.
- [20] A. R. Mamidala, J. Liu, and D. K. Panda. Efficient Barrier and Allreduce on InfiniBand Clusters using Hardware Multicast and Adaptive Algorithms. In *Proceedings of IEEE Cluster Computing*, 2004.
- [21] A. R. Mamidala, S. Narravula, A. Vishnu, G. Santhanaraman, and D. K. Panda. On using connection-oriented vs. connection-less transport for performance and scalability of collective and one-sided operations: trade-offs and impact. In *PPoPP '07: Proceedings of the 12th ACM SIGPLAN symposium on Principles and practice of parallel programming*, pages 46–54. ACM Press, 2007.
- [22] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.
- [23] A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimits, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White. Very high resolution simulation of compressible turbulence on the IBM-SP system. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, page 70, New York, NY, USA, 1999. ACM Press.
- [24] OpenFabrics Alliance. OpenFabrics. <http://www.openfabrics.org/>.
- [25] F. Petrini, D. J. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *SC '03: Proceedings of the 2003 ACM/IEEE conference on Supercomputing*, page 55. IEEE Computer Society, 2003.
- [26] G. Shipman, T. Woodall, R. Graham, and A. Maccabe. Infiniband Scalability in Open MPI. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [27] 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.
- [28] S. Sur, M. J. Koop, and D. K. Panda. High-Performance and Scalable MPI over InfiniBand with Reduced Memory Usage: An In-Depth Performance Analysis. In *Super Computing*, 2006.
- [29] 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.
- [30] J. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In *IPDPS '02: Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, page 27.2, Washington, DC, USA, 2002. IEEE Computer Society.
- [31] J. S. Vetter and A. Yoo. An empirical performance evaluation of scalable scientific applications. In *Supercomputing '02: Proceedings of the 2002 ACM/IEEE conference on Supercomputing*, pages 1–18, Los Alamitos, CA, USA, 2002. IEEE Computer Society Press.
- [32] J. Wu, J. Liu, P. Wyckoff, and D. Panda. Impact of on-demand connection management in mpi over via. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, page 152, Washington, DC, USA, 2002. IEEE Computer Society.
- [33] 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.