UCRL-CONF-235949

LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

Scalable High Performance Message Passing over InfiniBand for Open MPI

A. Friedley, T. Hoefler, M. L. Leininger, A. Lumsdaine

October 30, 2007

Communication in Clusters and Cluster Computer Interconnected Systems
Aachen, Germany
December 12, 2007 through December 12, 2007

Disclaimer

This document was prepared as an account of work sponsored by an agency of the United States government. Neither the United States government nor Lawrence Livermore National Security, LLC, nor any of their employees makes any warranty, expressed or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States government or Lawrence Livermore National Security, LLC. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States government or Lawrence Livermore National Security, LLC, and shall not be used for advertising or product endorsement purposes.

Scalable High Performance Message Passing over InfiniBand for Open MPI

Andrew Friedley[123] Torsten Hoefler[1] Matthew L. Leininger[23]
Andrew Lumsdaine[1]

[1]Open Systems Laboratory, Indiana University, Bloomington IN 47405, USA
{afriedle,htor,lums}@cs.indiana.edu

[2]Sandia National Laboratories, Livermore CA 94551, USA

[3]Lawrence Livermore National Laboratory, Livermore CA 94551, USA
{friedley1,leininger4}@llnl.gov

Abstract

InfiniBand (IB) is a popular network technology for modern high-performance computing systems. MPI implementations traditionally support IB using a reliable, connection-oriented (RC) transport. However, per-process resource usage that grows linearly with the number of processes, makes this approach prohibitive for large-scale systems. IB provides an alternative in the form of a connectionless unreliable datagram transport (UD), which allows for near-constant resource usage and initialization overhead as the process count increases. This paper describes a UD-based implementation for IB in Open MPI as a scalable alternative to existing RC-based schemes. We use the software reliability capabilities of Open MPI to provide the guaranteed delivery semantics required by MPI. Results show that UD not only requires fewer resources at scale, but also allows for shorter MPI startup times. A connectionless model also improves performance for applications that tend to send small messages to many different processes.

1 Introduction

The Message Passing Interface (MPI) has become the *de facto* standard for large-scale parallel computing. In large part, MPI's popularity is due to the performance-portability it offers. Applications written using MPI can be run on any underlying network for which an MPI implementation is available. Because so many HPC applications rely on MPI, all network hardware intended for high-performance computing has at least one (usually more than one) implementation of MPI available.

InfiniBand (IB) [6] is an increasingly popular high-performance network solution. Recent trends in the Top 500 list show that more and more systems are equipped with IB (from 3% in June 2005 to more than 24% in November 2007). The size of such systems also shows an increasing trend. The largest InfiniBand-based system in 2005, SystemX, connected 1,100 dual processor nodes. In 2006, the Thunderbird system appeared, consisting of 4,500 dual processor IB compute nodes.

InfiniBand provides several network transports with varying characteristics. A reliable, connection-oriented (RC) transport similar to TCP is most commonly used (especially by MPI implementations). A less-widely adopted alternative is the connectionless unreliable datagram (UD) transport, which is analagously similar to UDP. While UD does not guarantee message delivery, its connectionless model reduces the resources needed for communication and provides gains in performance when communicating with many different processes.

These characteristics of UD make it an interesting option as a transport protocol for IB support in an MPI implementation. The traditional RC (connection-based) approach with IB can lead to unacceptably high memory utilization and long startup times. Although the UD (connectionless) approach does not have the resource consumption problems of the RC approach, the burden of guaranteeing reliable message delivery is shifted from the transport protocol to the MPI implementation. This is not necessarily a disadvantage however. Providing reliability within an MPI impelmentation allows flexibility in designing a protocol that can be optimized for MPI communication.

1.1 Related Work

Several strategies have been employed to reduce the resource requirements of RC in an attempt to preserve the viability of an RC-based MPI implementation at large scale process counts. A shared receive queue (SRQ) feature allows for buffers to be posted to a single receive queue, instead of spreading resources across many different receive queues [12, 16]. MPI processes rarely receive large amounts of data over all connections simultaneously, so sharing allows the overall number of receive buffers to be reduced.

Another method of improving efficiency at scale is to use a lazy connection establishment mechanism. Connections are not established until they are required for communication, resulting in lower resource utilization in applications that require fewer than $O(N^2)$ connections. However, applications exhibiting a fully-connected communication pattern do not benefit, as all connections must be established at some point.

The use of the UD transport to solve the remaining issues with “fully-wired” applications was first developed in LA-MPI [5] by Mitch Sukalski, but to our knowledge this work was never published. Another approach was recently proposed in [7, 8]. Although the implementation described by Koop et. al. has recently been released, we did not have time to perform a comparison against our work before publication.

While our implementation appears to be similar in many ways, it differs in its clear design within Open MPI’s MCA architecture as well as in its implementation of a reliability protocol. Open MPI already provides a network-agnostic reliability protocol via the Data Reliability (DR) Point-to-Point Messaging Layer (PML) component, and is used to provide reliable communication over the UD transport. Furthermore, we describe an optimization and show that it increases bandwidth significantly for UD.

2 MPI Over InfiniBand

The InfiniBand (IB) Architecture [6] is a high-performance networking standard. Several network transports are defined. The most commonly used are the reliable connection-oriented (RC) and the unreliable datagram (UD) transport. RC is analogous to TCP, as is UD to UDP. Both transports provide traditional send/receive semantics, but extend this functionality with different feature sets.

2.1 Reliable Connection Transport

The Reliable Connection (RC) transport allows for arbitrarily sized messages to be sent. IB hardware automatically fragments the data into MTU-sized packets before sending, then reassembles the data at the receiver. Additionally, Remote DMA (RDMA) is supported over the RC transport. RDMA allows one process to directly read/write memory from/to the memory of another process, without requiring any communication processing by the remote process.

A connection-oriented communication model means that a connection must be explicitly established between each pair of peers that wish to communicate. Each connection requires a send and receive queue (together referred to as a queue pair, or QP). A third type of queue, shared by many queue pairs, is used by the hardware to signal completion of send and receive requests.

Requiring a separate receive queue for each connection means that separate receive buffers must be posted to each of these queues for the host channel adapter (HCA) to copy data into. Distributing receive buffers across many connections is wasteful, as the application is not likely to be receiving data from every other connected process simultaneously. Memory utilization increases with the number of connections, reducing memory available for use by applications. To work around this issue, IB supports the concept of a shared receive queue (SRQ). Each new queue pair (QP) may be associated with an SRQ as it is created. Any data arriving on these QPs consumes receive buffers posted to the shared receive queue. Since receive buffers are shared, applications receiving from only a subset of peers at any one time require fewer total buffers to be allocated and posted.

2.2 Unreliable Datagram Transport

As its name implies, the Unreliable Datagram transport does not guarantee message delivery. Thus, a UD-based MPI implementation must implement its own reliability protocol. In principle, a hardware-based reliability mechanism should have superior performance over a software-based mechanism. In practice, the flexibility of a software-based reliability protocol has lead to performance gains [13]. Also, the reliability offered by the RC transport ends at the InfiniBand HCA. No reliability is guaranteed when the data moves from the HCA across a PCI bus and into system memory.

Hardware message fragmentation is not provided with the UD transport, so software must also deal with fragmenting large messages into MTU-sized packets before being sent. Current IB hardware allows for a maximum two kilobyte data payload. Since protocol headers must be sent with each message, a small maximum message size presents a challenge for achieving optimal bandwidth. This issue will be discussed in section 4.2.

UD features a connectionless communication model. Only address information for a single QP at each process must be exchanged; peers may communicate without any sort of connection establishment handshake. This leads to both efficient and scalable resource utilization. First, only one QP is required for communication with any number of other processes. Active connections require resources in both the application and the HCA, so a connectionless model is desireable when minimizing resource usage at scale. Second, this single QP's receive queue behaves the same as an RC shared receive queue. Our results show that a simple, fully connected MPI job with 1024 processes would require 8.8 MiB of memory per process using our UD implementation compared to 14.75 or 25.0 MiB using the existing RC implementation with or without SRQ, respectively.

3 Open MPI

Open MPI [3] is a collaborative effort to produce an open source, production quality MPI-2 [9] implementation. First introduced in LAM/MPI [15], Open MPI's Modular Component Architecture (MCA) allows for developers to implement new ideas and features within self-contained components. Components are selected and loaded at runtime.

MCA components are organized into a set of frameworks responsible for managing one or more components that perform the same task. Each framework defines an interface for each component to implement as well as an interface for other parts of the implementation to access the components' functionality. Although many frameworks exist within Open MPI, only those most relevant to the work presented are discussed in detail here.

3.1 Point to Point Messaging Layer

One of the most important frameworks in Open MPI's communication path is the Point-to-point Messaging Layer (PML). PML components implement MPI send and receive semantics in a network-independent fashion. Message matching, fragmentation and re-assembly are managed by the PML, as well as selection and use of different protocols depending on message size and network capabilities. Other frameworks, like the Memory Pool (MPool) and Registration Cache (RCache) are enabled for networks that require memory registration (such as InfiniBand). Figure 1 illustrates how all of these frameworks fit together to form Open MPI's point to point architecture. [11, 12] discusses the related frameworks in greater detail.

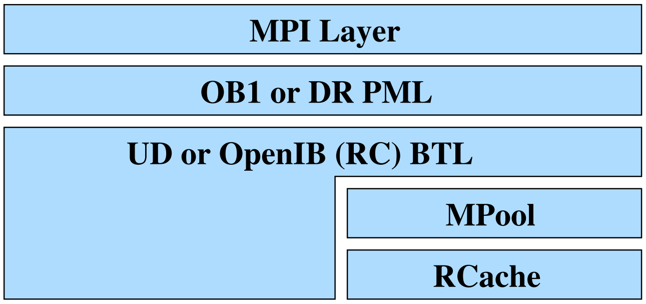

Figure 1. Point-to-point Architecture

An interesting PML component in the context of this paper is the data reliability (DR) component [10]. DR uses an explicit acknowledgement protocol along with data checksumming to provide network failover capabilities. This allows DR to detect when network links are no longer functional, and will automatically switch over to a different network when failure occurs. Furthermore, DR provides verifies checksums once data reaches main memory. This allows for detection of data corrupted by the system busses, which IB's hardware-based reliability does not provide. Such reliability comes at a price however; performance is slightly lower than that of the default PML component, referred to as OB1.

What makes the DR component interesting is its use in conjunction with InfiniBand's UD transport. As discussed in section 2.2, a software reliability protocol is needed to meet MPI's guaranteed message delivery semantics. Rather than implementing a reliability protocol specifically for UD, the DR component may be used to achieve guaranteed message delivery.

3.2 Byte Transfer Layer

Both the DR and OB1 PML components rely on another framework, the Byte Transport Layer (BTL), to implement support for communicating over specific networks. The BTL framework is designed to provide a consistent interface to different networks for simply moving data from one peer to another. This simplicity allows for fast development of new components for emerging network technologies, or to explore research ideas.

The BTL interface is designed to provide a simple abstraction for communication over a variety of network types, while still allowing for optimal performance. Traditional send semantics must be supported by each BTL, while RDMA put/get semantics may be supported by networks that provide it. Receives of any form (traditional or RDMA) are initiated only by registering a callback function and a tag value with the BTL. The BTL then calls this function whenever a message arrives with a corresponding tag value, providing the upper layers access to the received message. Some networks (including IB) require that memory be registered for direct access by the HCA, so a memory allocation interface is provided by the BTL as well.

Each BTL component implements support for a particular network and transport. InfiniBand is primarily supported via the OpenIB component, which uses the RC transport type exclusively. For clarity, this is referred to as the RC BTL. RDMA is supported for both small and large messages. While using RDMA for small messages (referred to in Open MPI as eager RDMA) may reduce latency, such an approach is not scalable due to the requirement that RDMA buffers be polled to check for completion [12].

Connection management is a major issue for large scale MPI jobs. With a connection-oriented transport like RC, a connection and associated resources must be allocated for each pair of peers that wish to communicate. In the worst case this results in $O(N^2)$ connections for an all-to-all communication pattern.

One way to avoid this problem is to establish connections only when communication between two peers is requested. Each time a process sends a message, it checks the state of the connection to the destination process. If no connection is established, a new connection is initiated. Rather than waiting for the connection to be established, the message to be sent is queued and the application or MPI library continue execution. If the process tries to send another message to the same destination before the connection is established, that message is queued as well. Finally, when the connection is available, any queued messages are sent. Subsequent messages are sent immediately, but a small cost is still incurred to check the state of the connection. Since applications rarely exhibit an all-to-all communication pattern [17], the reduction in resources required for established connections outweighs the small cost of managing connection state dynamically.

4 Implementation

To support communication over unreliable datagrams, a new BTL component was developed. Currently named the UD BTL, its design is far simpler than that of the existing RC-based BTL. RDMA is not available over UD, so support for RDMA protocols is not needed. More importantly, UD is a connectionless network transport; no connection management is necessary. A single queue pair is used for receiving all data from all peers. During MPI initialization, QP address information is exchanged with every other peer using an all-to-all algorithm over an existing TCP communication channel established by the runtime environment. Unlike lazy connection establishment, no logic is needed in the send path for managing connections – UD connections are always available.

Bandwidth over the UD transport may be increased by exploiting the ability of IB hardware to process multiple send requests in parallel. This is done by initializing a small constant number of QPs and posting sends among them in a round-robin fashion. Four QPs were chosen to give near-optimal bandwidth without affecting latency. Figure 2 shows peak bandwidth reported by NetPIPE [14] when striping over a varying number of queue pairs. Unlike the send side, distributing receive buffers across several QPs did not yield any performance gains. In our implementation, only one QP is used for receiving messages, while four are used for sending.

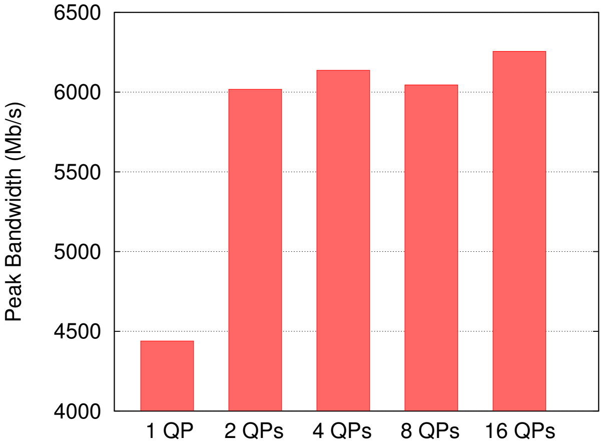

Figure 2. QP Striping Bandwidth

4.1 Buffer Management

Ensuring that enough receive buffers remain posted is a critical aspect of maximizing performance over the UD transport. When data arrives for a QP that has no receive buffers posted, the HCA silently drops the data. This problem is exacerbated by the fact that applications must call into the MPI library in order for consumed receive buffers to be process and re-posted to the QP, and may not do so for long periods of time. Therefore, the UD BTL should always have as many receive buffers posted as the number of messages that might be received between MPI calls.

Flow control is difficult for two reasons. First, receive buffers are shared, so any combination of remote sender activity may exhaust the receive queue. Many senders each sending small messages may overwhelm a receiver just as easily as one or a few senders sending large messages. Flow control is difficult, as all potential senders must be notified of congestion at the receiver in order to be effective. Doing this means sending a message to each sender, which is not feasible at scale.

Second, accurately detecting a shortage of receive buffers with enough time to react is difficult or even impossible, especially with data arriving while applications are not making frequent calls into the MPI library. Due to the simplicity of the BTL abstraction, it is not possible to take advantage of a PML-level rendezvous protocol to prepost receive buffers when large messages are expected. Nor does the IB verbs interface provide any information about how many receive buffers are currently posted; the UD BTL must keep its own counter of currently posted buffers based on completion events. Furthermore, time required to either process currently filled buffers or allocate and register new buffers is often greater than the time remaining before currently posted resources are exhausted.

Our solution attempts to strike a balance between minimizing resource usage and keeping a large number of buffers posted to the receive queue at all times. Rather than trying to dynamically adapt to the communication load, a static pool of buffers is allocated and left unchanged after initialization. For tuning to particular applications, an MCA parameter may be used to specify the number of buffers in the pool at runtime. The receive path has been optimized to process and re-post buffers as quickly as possible.

When allocating buffers for an SRQ, the RC BTL uses a similar strategy, except that it adjusts the number of buffers based on the number of processes in the MPI job. However, large process counts do not necessarily correlate to increased communication load per receiver, possibly leading to an unnecessarily large buffer pool at scale. Accurate sizing of the buffer pool depends heavily on an application's behavior; particularly its communication patterns and frequency of calls into the MPI library.

4.2 Message Format and Protocol

Both the PML and BTL are free to define their own protocol and message headers as they require. The PML builds messages of a suitable size (bounded by the maximum message size the BTL can send) and prefixes its own header data. A BTL component treats this as an opaque data payload, and prefixes its own header data as needed. When messages arrive at the receiver, the BTL uses its header data however it chooses, and passes the opaque data payload (including the PML headers) to the PML for processing.

Lack of flow control has the advantage of allowing for a simpler communication protocol. The UD BTL adds only a single byte message tag value (rounded to 4 bytes for alignment purposes), used for determining which registered callback function should be used to pass the data to the upper layer. However this low per-message overhead is offset by the UD transport's 2 Kib MTU. Packet headers for both the PML and BTL must be included in each 2 Kib message. In contrast, the RC BTL includes additional flow control information in its packet headers, but is able to send much more data with each message.

To ensure reliability, the DR PML defines the concept of a virtual fragment, or VFRAG [10], which acts as a unit of acknowledgement. Two timeout mechanisms are associated with each VFRAG. The first is used to detect when local send completion fails to occur. The second is the familiar ACK timeout, which allows the sender to detect loss by lack of positive acknowledgement from the receiver within some time frame. Retransmission occurs either when a timeout expires, or the receiver responds with an acknowledgement indicating some portion of the VFRAG was not received. After several retransmission attempts over one BTL, the PML marks that BTL as failed, and begins using a different BTL. While such an approach is intended to transparently respond to network failure, it also guarantees delivery over an unreliable network. When used in conjunction with the DR PML, our UD implementation provides MPI's guaranteed delivery semantics.

5 Results

5.1 Experimental Environment

Atlas, a production system housed at Lawrence Livermore National Laboratory (LLNL), was used for all experimental results. Atlas is a 1,152 node cluster connected via an InfiniBand network. Each node consists of four dual core 2.4 GHz Opteron processors (eight cores per node) with 16 Gib RAM and Mellanox PCI-Express DDR InfiniBand HCAs, running Linux.

A development version of Open MPI was used, based on subversion revision 16080 of the main development trunk. Source code is available to the public on the Open MPI website [http://www.open-mpi.org]. Stable UD support is planned for Open MPI version 1.3.

Results for UD are provided using both the OB1 and DR PMLs. The rationale behind including results for UD with OB1 is that the reliability overhead introduced by DR may be directly observed. Although use of OB1 with UD does not guarantee message delivery, no messages were actually dropped while running our tests.

5.2 Microbenchmarks

Microbenchmarks are used to determine the behavior of a system under certain, usually isolated conditions. We use those benchmarks to determine basic system parameters such as latency, bandwith, and memory utilization.

5.2.1 Point-to-point Performance

NetPIPE [14], a common ping-pong benchmark, was used to establish a baseline comparison of the latency and bandwidth capabilities of the RC and UD BTLs. Latency as a function of message size is presented in Figure 3, while bandwidth is presented in Figure 4. For the RC BTL, eager RDMA was disabled to approximate latencies at scale.

For messages smaller than 2 Kib, UD with OB1 is nearly equivalent to RC, with or without SRQ. UD performance

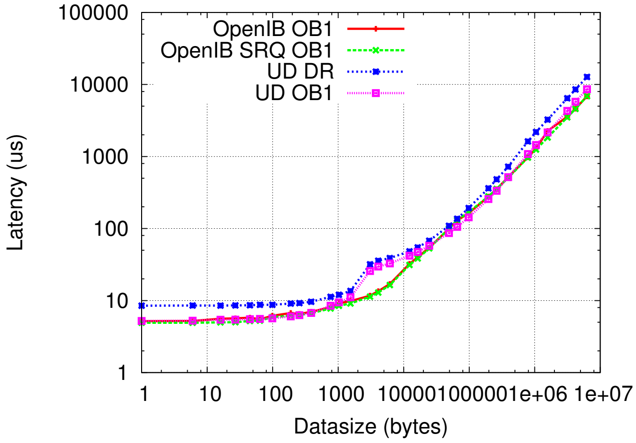

Figure 3. NetPIPE Ping Pong Latency

with DR is worse due to the DR PML reliability protocol. A study of the performance overhead inherent in the DR PML may be found in [10]. At 2Kib, UD's MTU forces the PML to switch to the slower rendezvous protocol, causing a drop in performance that is overcome as message size increases.

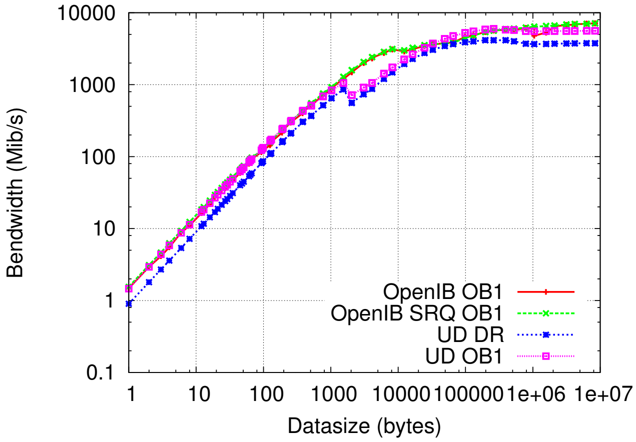

Figure 4. NetPIPE Ping Pong Bandwidth

5.2.2 Startup Performance and Memory Overhead

We developed a benchmark to measure the overhead incurred by lazy connection establishment. Each process in the MPI job iteratively sends a 0-byte message to every other process, while receiving a 0-byte message from another process. This is done in a ring-like fashion to prevent flooding any one process with messages. The total time taken by each process is averaged to produce a single measurement.

Time taken to execute the benchmark for varying process counts is shown in Figure 5. A total of 512 nodes were used; results past 512 nodes were obtained by running more than one process per node. This does not significantly affect the results, as the benchmark is not bandwidth bound. The UD BTL yields excellent performance, as no connections need to established. Since there is no connection overhead, UD's time is only that taken for every process to send a 0-byte message to every other process. RC, shown both with and without SRQ enabled, incurs significant overhead due to a connection establishment handshake protocol.

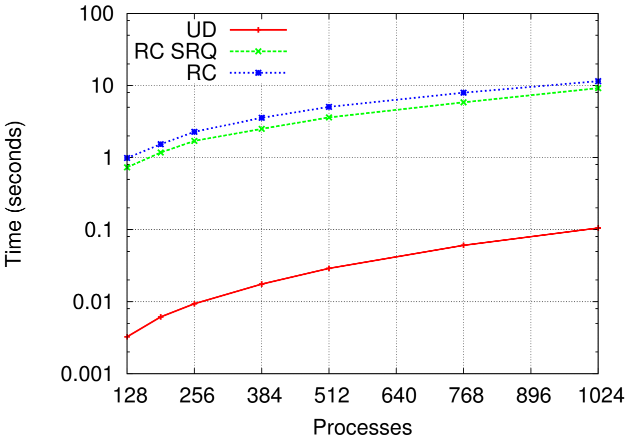

Figure 5. Startup Overhead

Figure 6 shows average memory utilization per process of MPI processes measured at the end of the allconn benchmark. Again, results past 512 nodes were obtained by running multiple MPI processes per node. UD's memory utilization grows very slowly, at a near-constant rate. Each MPI process stores a small amount of address information for every other MPI process, leading to slowly increasing utilization as scale increases. RC with SRQ does significantly better than without, resulting in lower utilization at low process counts but increasing at a faster rate than UD. This is due to the RC BTL adjusting the size of the receive buffer pool based on the number of processes, while the UD BTL receiver buffer count remains constant (cf. Section 4). Memory utilization for RC without SRQ is significantly higher since receive buffers are allocated for each connection.

5.3 Application Benchmarks

Application benchmarks help to analyze the behavior of our implementation for real-world applications. We included positive and negative results to illustrate the trading of memory and wireup costs for communication costs.

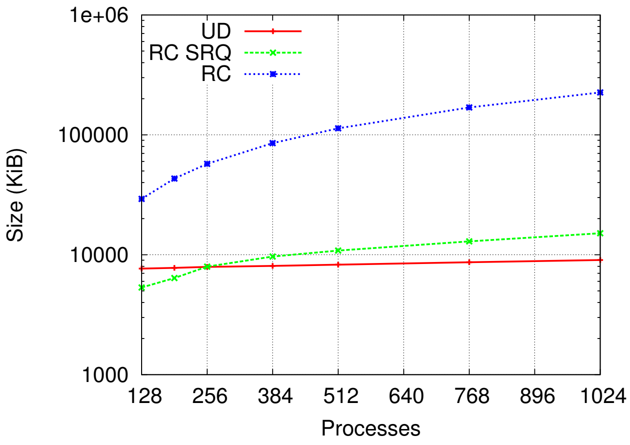

Figure 6. Memory Overhead

5.3.1 ABINIT

ABINIT is an open source code for ab initio electronic structure calculations based on the density functional theory. The code is the object of an ongoing open software project of the Université Catholique de Louvain, Corning Incorporated, and other contributors [4]. We use a sample calculation representing a real-word problem described in [1].

Overall execution times are presented in Figure 7. Only one run in each configuration was possible due to limited system availability. Even so, it is fairly clear that for a real application such as ABINIT, performance is similar for both UD and RC-based implementations. We were unable to measure memory usage, but believe the results would be positive but less pronounced than those of the microbenchmarks.

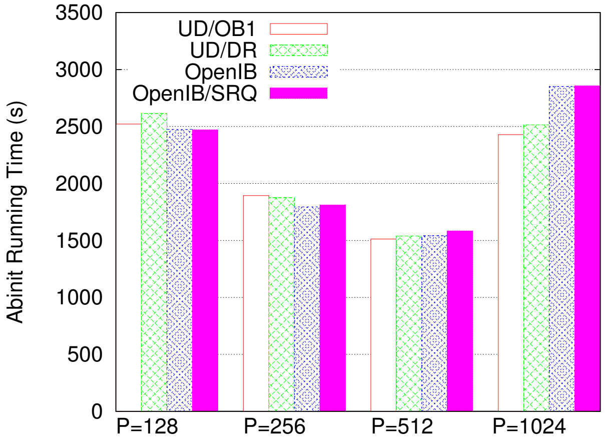

Figure 7. Abinit results

5.3.2 SMG2000

SMG2000 [2] is a benchmark written around a parallel semicoarsening multigrid solver designed for modeling radiation diffusion. An analysis presented in [17] indicates that SMG2000 tends to send many small messages to many distinct peers. Based on the microbenchmark results, our expectation is that a UD-based MPI implementation will perform well in this sort of scenario due to superior small-message latency and a connectionless communication model.

Figure 8 shows the execution time of the SMG2000 solver phase for varying number of processes. For this application, UD has a clear advantage, but trades in overhead for the data reliability protocol. The short execution time of our tests emphasize UD's better connection setup performance, especially as scale increases.

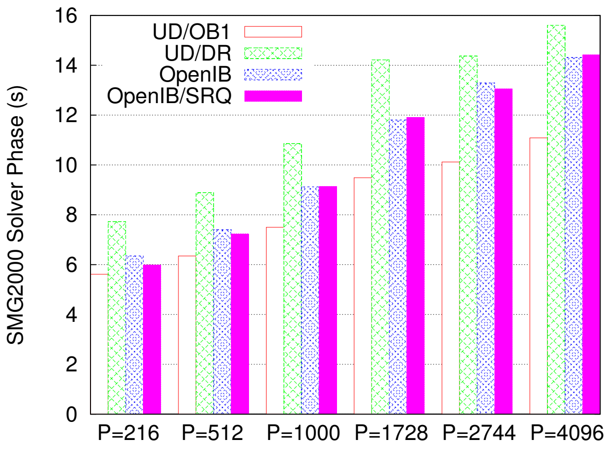

Figure 8. SMG2000 Solver Time

6 Conclusions and Future Work

There are several conclusions that can be drawn from the results in this paper. First, an unreliable-datagram based MPI implementation can be a viable alternative to reliable connection-based approaches, especially at large scale. The UD approach provides comparable message passing performance to RC approaches and provides distinct advantages in startup time and memory overhead. Second, our use of multiple queue pairs for sending messages optimizes UD communication bandwidth. Finally, our implementation demonstrates how components in a component-based software architecture can be reused to solve new problems. In particular, we were able to provide data reliability in the UD implemenation by reusing the data reliability component (Data Reliability PML), thereby reducing code complexity and maintenance costs.

Microbenchmarks and certain applications (e.g., NetPIPE and SMG2000) show that the UD BTL incurs a slight performance penalty in some cases when compared to Open MPI's existing RC implementation. On the other hand, our UD-based implementation offers reduced memory overhead and fast startup times, especially for applications communicating with many peers. SMG2000 and ABINIT show that applications can benefit, especially at scale. As larger IB systems are deployed, the highly scalable, low-overhead characteristics of a UD-based implementation will be preferred over RC-based implementations.

Future work will investigate the use of a UD-specific reliability schemes to minimize performance penalties associated with the current DR-based implementation. Alternative solutions to the software flow control problem will also be explored.

Acknowledgements

This work was supported by a grant from the Lilly Endowment and National Science Foundation grant EIA-0202048. This research was funded in part by a gift from the Silicon Valley Community Foundation, on behalf of the Cisco Collaborative Research Initiative of Cisco Systems.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. UCRL-CONF-235949.

References

[1] F. Bottin and G. Zerah. Formation enthalpies of monovacancies in Aluminium and Gold a large-scale supercell ab initio calculation. *submitted to Physical Review B*, 2006.

[2] 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.

[3] 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.

[4] X. Gonze, J.-M. Beuken, R. Caracas, F. Detraux, M. Fuchs, G.-M. Rignanese, L. Sindic, M. Verstraete, G. Zerah, F. Jollet, M. Torrent, A. Roy, M. Mikami, P. Ghosez, J.-Y. Raty, and D. Allan. First-principles computation of material properties : the ABINIT software project. *Computational Materials Science 25, 478-492*, 2002.

[5] R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4):285–303, August 2003.

[6] InfiniBand Trade Association. *Infiniband Architecture Specification Volume 1, Release 1.2*. InfiniBand Trade Association, 2004.

[7] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda. High performance MPI design using unreliable datagram for ultra-scale InfiniBand clusters. In *ICS '07: Proceedings of the 21st annual international conference on Supercomputing*, pages 180–189, New York, NY, USA, 2007. ACM Press.

[8] M. J. Koop, S. Sur, Q. Gao, and D. K. Panda. Zero-copy protocol for mpi using infiniband unreliable datagram. *IEEE Cluster 2007: International Conference on Cluster Computing*, Austin, TX, USA, September 17-20, 2007.

[9] Message Passing Interface Forum. *MPI-2: Extensions to the Message Passing Interface*, July 1997. http://www.mpi-forum.org.

[10] G. M. Shipman, R. L. Graham, and G. Bosilca. Network fault tolerance in open MPI. In *Proceedings, Sixth International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks*, Rennes, France, September 2007.

[11] 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.

[12] G. M. Shipman, T. S. Woodall, R. L. Graham, A. B. Maccabe, and P. G. Bridges. Infiniband scalability in open mpi. In *Proceedings of IEEE Parallel and Distributed Processing Symposium*, April 2006.

[13] R. Sivaram, R. K. Govindaraju, P. H. Hochschild, R. Blackmore, and P. Chaudhary. Breaking the connection: Rdma deconstructed. In *Hot Interconnects*, pages 36–42. IEEE Computer Society, 2005.

[14] Q. Snell, A. Mikler, and J. Gustafson. NetPIPE: A Network Protocol Independent Performace Evaluator. In *IASTED International Conference on Intelligent Information Management and Systems*, June 1996.

[15] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, pages 379–387, Venice, Italy, September / October 2003. Springer-Verlag.

[16] 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 *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 105, New York, NY, USA, 2006. ACM Press.

[17] J. Vetter and F. Mueller. Communication characteristics of large-scale scientific applications for contemporary cluster architectures. In 16th Intl. Parallel & Distributed Processing Symp., May 2002.