

Zero-Copy Protocol for MPI using InfiniBand Unreliable Datagram

Matthew J. Koop, Sayantan Sur, Dhabaleswar K. Panda

Network-Based Computing Laboratory, The Ohio State University
2015 Neil Ave., Columbus, OH 43210 USA
{koop, surs, panda}@cse.ohio-state.edu

Abstract—Memory copies are widely regarded as detrimental to the overall performance of applications. High-performance systems make every effort to reduce the number of memory copies, especially the copies incurred during message passing. State of the art implementations of message-passing libraries, such as MPI, utilize user-level networking protocols to reduce or eliminate memory copies. InfiniBand is an emerging user-level networking technology that is gaining rapid acceptance in several domains, including HPC. In order to eliminate message copies while transferring large messages, MPI libraries over InfiniBand employ “zero-copy” protocols which use Remote Direct Memory Access (RDMA). RDMA is available only in the connection-oriented transports of InfiniBand, such as Reliable Connection (RC). However, the Unreliable Datagram (UD) transport of InfiniBand has been shown to scale much better than the RC transport in regard to memory usage. In an optimal design, it should be possible to perform zero-copy message transfers over scalable transports (such as UD).

In this paper, we present our design of a novel zero-copy protocol which is directly based over the scalable UD transport. Thus, our protocol achieves the twin objectives of scalability and good performance. Our analysis shows that uni-directional messaging bandwidth can be within 9% of what is achievable over RC for messages of 64KB and above. Application benchmark evaluation shows that our design delivers a 21% speedup for the `in.rhodo` dataset for LAMMPS over a copy-based approach, giving performance within 1% of RC.

I. INTRODUCTION

Large-scale deployments of clusters designed from commodity components continue to be a major component of high-performance computing environments. InfiniBand [1], is an interconnect of such systems that is enjoying wide success due to low latency (1.0-3.0 μ sec) and high bandwidth and other features. Recent deployments of large-scale InfiniBand clusters, such as the 9024-processor Sandia Thunderbird [2], NASA/Ames Columbia [3], 9216-core LLNL Atlas [4], and the 5200-processor TACC Lonestar [5], are a testament to the success and potential of the interconnect.

The Message Passing Interface (MPI) [6] is the dominant programming model for parallel scientific applications. The MPI library design is crucial in supporting high-performance and scalable communication for applications on these large-scale current and next-generation clusters.

As clusters continue to scale, clusters of tens-of-thousands of cores will soon be deployed with InfiniBand. Earlier studies have shown that connection memory for InfiniBand communication contexts can be significant at such scales [7], [8],

impeding the ability to increase problem resolution due to memory constraints. Using the connection-oriented Reliable Connection (RC) transport, several KB of memory must be dedicated per connected peer. Strategies such as lazy connection setup and using the Shared Receive Queue (SRQ) support of InfiniBand have been employed to reduce resource consumption [9], [10], [11]. Even combining all of these solutions, however, the MPI library can still require significant (140MB for 8K processes) of memory per process at scale.

One recently proposed solution [8], an MPI based on the connection-less Unreliable Datagram (UD) transport of InfiniBand, appears promising due to the lack of additional memory per connected peer. However, despite a near-constant memory usage per process, large message bandwidth remains a challenge due to message segmentation and memory copies. Since the UD transport can only send messages of up to the MTU size, 2 KB on current Mellanox [12] Host Channel Adapters (HCAs), message segmentation must be performed for messages above that threshold. Additionally, due to the lack of RDMA support, memory copies are still used to copy message segments and add headers in the proposed approach.

In this paper we build on our previous work [8] of a UD-based MPI and propose a novel protocol for reliably transferring large messages over UD based on a zero-copy method. Although also readily applicable to other domains, we discuss our design in relation to MPI communication. An implementation of the design is evaluated using standard MPI benchmarks to assess performance improvement. An improvement of 50% is observed for unidirectional bandwidth over a standard copy-based approach. Application benchmark evaluation shows up to a 21% speedup for LAMMPS over a copy-based approach, giving performance within 1% of RC. Performance on the NAS Parallel Benchmarks was improved by up to 16% for IS.

The remaining parts of the paper are organized as follows: In Section II we provide an overview of zero-copy as well as an overview of InfiniBand features. In Section III we address the need for a UD-based zero-copy protocol. We present a novel design to enable zero-copy transfers over UD in Section IV. Evaluation and analysis of an implementation of our design is covered in Section V. In Section VI related work in this area is discussed. Finally, conclusions are presented in Section VII.

TABLE I
FEATURE COMPARISON OF UD VS RC

	UD	RC
Reliable Delivery	–	✓
In-Order Delivery	–	✓
RDMA Support	–	✓
Large Messages	–	✓
Max QPs Required (per job)	N	N^2

II. BACKGROUND

In this section we give the required background information on InfiniBand [1] and Zero-Copy protocols.

A. Overview of InfiniBand Architecture

InfiniBand is a processor and I/O interconnect based on open standards [1]. 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.

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) which 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. InfiniBand also defines a Shared Receive Queue (SRQ) which allows receive requests to be shared across multiple QPs. This allows scalable receive buffer management.

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. In addition, remote atomic operations are also supported. Both communication semantics require communication memory to be registered with InfiniBand hardware and pinned in memory. The registration operation involves informing the network-interface of the virtual to physical address translation of the communication memory. The pinning operation requires the Operating System to mark the pages corresponding to the communication memory as non-swappable. Thus, communication memory stays locked in physical memory, and the network-interface can access it as desired.

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) 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. Being connection-oriented in nature, every communicating peer requires a separate QP. In regard to resources required, it is identical to RC, while not providing reliable service. Thus, it appears unattractive for implementing MPI over this transport.

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 [12] hardware). This means the communication upper layer, such as MPI, must manage fragmentation and re-assembly of messages. The UD transport does not provide any sort of reliability, as in no lost packets are reported neither is their arrival order guaranteed. The UD transport does not enable RDMA either. All communication must be performed using channel semantics, i.e. send/receive. A summary of the feature availability in UD and RC transports is given in Table I.

B. Zero-Copy Protocols and RDMA

Memory copies lead to overall degradation of application performance and ineffective use of resources of the computing platform. Memory copies become a bottleneck especially when transferring large messages. State-of-the-art MPI implementations over high-performance networks avoid memory copies for large messages by using zero-copy protocols. In zero copy protocols the sender and receiver use small control messages to match the message and then the message data is placed directly in user memory. A zero-copy protocol can significantly increase bandwidth and reduce cache pollution. Figure 1 shows the difference between a typical copy-based approach and a zero-copy approach. Instead of performing data copies in the send and receive paths, within user-space or the kernel, a zero-copy approach directly sends the data from the source application buffer to the final destination buffer.

Many zero-copy mechanisms have been implemented, including in U-Net [13], BIP [14], and PM [15]. Modern high-performance interconnects such as InfiniBand, Myrinet, and Quadrics use zero-copy communication. In order to support zero-copy messaging, the network and drivers must support

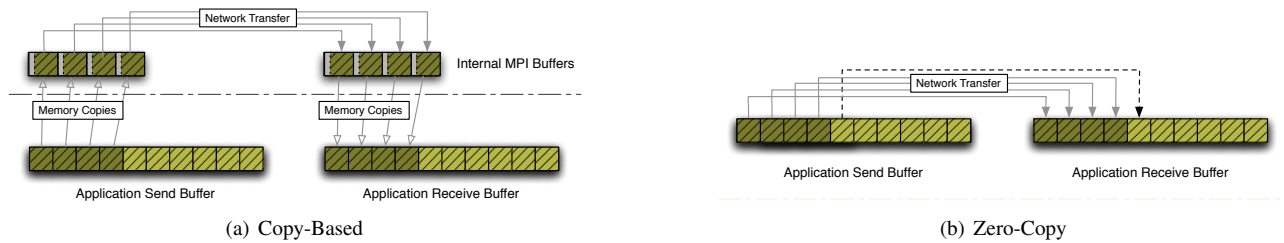


Fig. 1. Copy vs. Zero Copy

OS-bypass to avoid kernel copies. There are two major ways zero-copy protocols are implemented in MPI implementations over modern interconnects:

- *Rendezvous Protocol using RDMA*: In this method, a handshake protocol (rendezvous) is used. The sending process sends a Request to Send (RTS) message with message tag information. Upon discovery of the message by the MPI layer at the receiver end, the receiver can either use RDMA Read to get the message data directly into user application buffer or send a Clear to Send (CTS) to the sender. If the sender receives a CTS, then it can use RDMA Write to send the message data directly to user application buffer at remote side. Thus, RDMA Read/Write provide a convenient method to perform zero-copy protocols when MPI message tags are matched by the MPI library. Typically, this method is used with networking stacks which do not have the capability to match MPI tags, e.g. InfiniBand/OpenFabrics [16].
- *Matched Queues Interface*: In this method, the sending process directly sends the message to a remote process. This requires that either the network device or network software layers be able to decipher the message tags from the sent message and match it with the posted receive operations. Upon a successful match, the rest of the message may be directly received into the user application memory. This method is used in Myrinet/MX [17], InfiniPath/QLogic [18] and Quadrics [19].

III. WHY DO WE NEED ZERO-COPY OVER UD?

As clusters continue to scale to larger numbers of total processing cores, the need for scalability is crucial. Previous work has shown a UD-based MPI has superior memory scalability, as shown in Figure 2. Even with 8K processes, the memory usage of the entire MPI library per process is less than 40 MB. Less than 1 MB is used for communication contexts. By contrast, even with optimized settings RC communication context memory usage can cause the memory required by the MPI library to grow to nearly 140 MB/process when 8K connections are established. Clusters are expected to continue to grow in scale, so this issue of resource usage is likely to become even more significant with RC-based MPIs.

Using UD as a transport, however, loses a significant benefit of InfiniBand – the use of RDMA for high-performance zero-copy message transfers. Previous designs proposed in [8] used a pipelined-copy-send approach in which each message was

segmented, copied into internal buffers and sent. However, using this approach, uni-directional bandwidth was only around 50% of what could be achieved over RC transport for 1 MB message size. While the scalable nature of the UD transport is highly desired, good performance for large messages is also a critical component for an optimal MPI library. Thus, in this research work, we focus on improving the performance of large messages using a zero-copy protocol over UD. To the best of our knowledge, this is the first work which combines the twin goals of scalability and performance using InfiniBand UD transport.

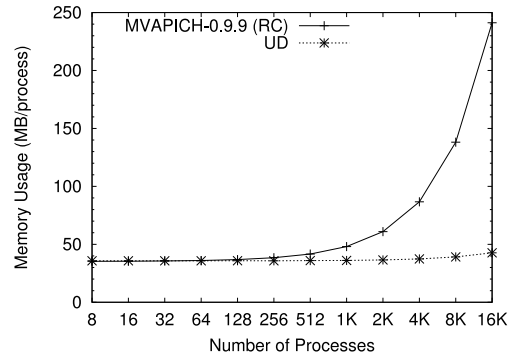


Fig. 2. Fully-Connected MPI Memory Usage, Courtesy [8]

IV. DESIGN

As noted in Section II-A.2, the UD transport does not allow RDMA semantics. Also, InfiniBand/OpenFabrics [16] does not implement any Matched Queues interface. Thus, two of the popular methods mentioned in Section II to design a zero-copy protocol are not available over UD transport. However, for scalability reasons mentioned in the previous sections, using the UD transport is highly desired on very large scale clusters. In this section we present a novel design for zero-copy protocol over UD transport. We first describe the various design challenges in detail, followed by the proposed design.

A. Design Challenges

The challenges of providing high-performance reliable transport at the software level over UD are significant. The earlier UD-based design, presented in [8] used the traditional

TCP-style reliability over UD with significantly lower performance for large messages due to extra memory copies. In this section we discuss these challenges that must be overcome for a high-performance true zero-copy design.

Limited MTU Size: A significant disadvantage of using the UD transport is that messages are limited to the maximum MTU size. The maximum MTU available on current Mellanox HCAs is 2 KB; the specification allows for an MTU of up to 4 KB. Segmentation of larger messages is required before posting the send operation since the hardware will not perform this action for UD.

Lack of Dedicated Receive Buffers: As discussed in Section II, to receive messages using channel semantics the receive buffers must be posted to a QP. While using the UD transport, the UD QP is shared for all remote peer processes. In this scenario, it is difficult to post receive buffers for a particular peer as they are all shared. Additionally, using the UD transport, if no buffer is posted to a QP, any message sent to that QP is silently dropped. A high-performance design must always strive to avoid dropping packets due to unavailability of buffers.

Lack of Reliability: There are no guarantees that a message sent will arrive at the receiver. Traditional approaches used by TCP-style reliability algorithms can assist in achieving reliability, but adding packet headers for sequence numbers, acknowledgments, and tracking send operations is non-optimal for high-performance. Adding these fields inside packets can lead to memory copies, since these fields would have to be placed in each packet individually.

Lack of Ordering: Messages may not arrive in the same order they were sent. Each buffer is consumed in a FIFO manner, so the first buffer filled may not be the first packet sent from a process. The MPI library must now take care of re-arranging the arrived packets in the correct order and form the message from the individual packets.

Lack of RDMA: The lack of RDMA is the most significant deficiency of UD in terms of implementing a high-performance large message transfer mechanism. If RDMA were provided the sender and receive buffers could be pinned and transferred directly without regard to ordering since messages are directly placed in the user buffer. Reliability could be ensured through tracking of completions on the receiver side. This mechanism is similar to the implementation of RDMA on the IBM HPS adapter where the HCA can directly place the incoming data to the user buffer since each packet describes its own placement location [20]. Unfortunately, InfiniBand does not provide such a capability for UD.

Extra headers in application UD packets: The Global Routing Header (GRH) is used by InfiniBand switches to route packets between subnets. The InfiniBand specification mandates that these headers be present in UD based multi-cast packets. However, current InfiniBand hardware and software stack place this 40-byte GRH header inside the application

packets. This is particularly frustrating since even if reliability and ordering were guaranteed, it would not be straight-forward to simply post the user application receive buffers directly to the QP.

B. Proposed Design

Large message transfers are typically done through a “rendezvous” protocol: the sender notifies the receiver of intent to send a message, a Request To Send (RTS). The receiver will reply with a Clear To Send (CTS) message as soon as the receive has been posted. Such a technique is used in traditional large message transfers in InfiniBand to perform an RDMA Write/Read operation to directly place the data into the receive buffer. In this case the CTS message will contain the address of the receive buffer to allow RDMA operations. Finally, a finish message (FIN) is sent to notify the receiver of data placement.

Our design uses a similar CTS/RTS protocol as is common in other MPI implementations, however, there are important differences due to the lack of RDMA and unreliable transfer. An overview of our proposed protocol is shown in Figure 3.

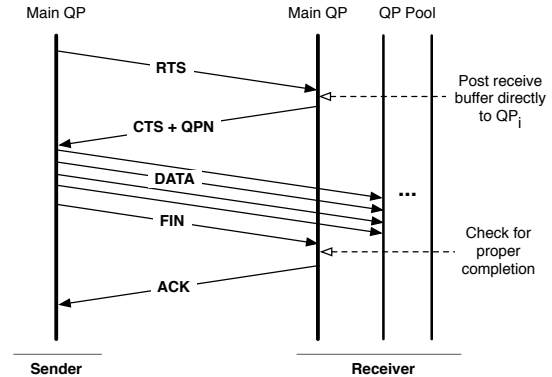


Fig. 3. UD Zero-Copy Protocol

1) *Enabling Zero-Copy:* Our proposed design to implement zero-copy transfers over UD is based on a *serialized communication* model since RDMA and tag matching are not specified for the UD transport. “Serialized” is used here to denote that communication on a given QP can be serialized – the order of transfer is agreed on beforehand and only one sender will transmit to a QP at a single time.

After receiving a RTS message from a sender, the receiver will select a UD QP for this transfer; we will denote this $QP_{transfer}$. If the receive operation has not been posted, the receiver queues the RTS message until a matching receive is posted. When the receive is posted, the buffer is directly posted in MTU size chunks to $QP_{transfer}$. In this way, the first message sent to $QP_{transfer}$ will be directly placed into the first MTU size chunk of the receive buffer. The rest of the received data will continue to fill the buffer in the order of arrival.

2) *Efficient Segmentation:* One of the challenges for using the UD transport is the lack of support for messages larger than the MTU size. Although this is not a problem that can be

solved without changing the specification, steps can be made to reduce the detrimental effects. One of the major overheads is the mechanism used to report completion of packet send operations. Since large messages can generate lots of packets, the overhead might be higher. In our design, we choose to get “completion signal” only for the last packet of the message. The underlying reliability layer would mark packets as missing at the receiver side and the sender would be notified. Hence, the intermediate completions at the sender are unnecessary, as long as we know the last packet was sent out of the node, it is enough.

3) *Zero-Copy QP Pool*: Receive requests for UD QPs are shared for all remote peers. This makes it harder to dedicate a set of resources for a particular remote process for sending large messages. In order to solve this problem, we maintain a pool of UD QPs which can be used as required. When a large message transfer is initiated, a QP is taken from the pool and the application receive buffer is posted to it (in MTU chunks). For the duration of the message transfer, this QP is bound to the specific remote process. When the transfer is finished, the QP is returned to the pool. Obtaining and returning QPs from a pool are very cheap, hence our design is expected to perform reasonably well for up to as many large incoming sends as the size of the pool. For the purpose of our evaluation, the pool size is 64, although this is tunable at runtime.

4) *Optimized Reliability and Ordering for Large Messages*: As mentioned earlier in this section, the UD transport does not guarantee any reliability or ordering. In our zero-copy protocol, we need to take care of these issues for maintaining data integrity and MPI semantics.

The first challenge is determining if message re-ordering has occurred. One method would be to perform a checksum of the final receive buffer, however, such a scheme negates much of the benefit of using a zero-copy protocol since the data buffer must now be traversed. Instead we propose leveraging the *immediate data* field of InfiniBand. In addition to the normal data payload, send operations can also specify a 32-bit immediate field that will be available to the receiver as part of the completion entry. In our design the immediate field is filled with a sequence number that identifies the order in which data was sent and should be received. Since each QP is associated with a CQ, upon receiving the FIN message the associated CQ can be polled for completions. If any completion is out of order or missing, re-ordering or drops have occurred during transfer. If a message has been dropped or re-ordered there are various techniques that can be used. The most simple, and the approach we have implemented, will send a negative acknowledgment (NACK) to the sender to request retransmission of the entire message. More optimized approaches, such as copying only re-ordered parts of the receive buffer or selective retransmission may be helpful in high-loss environments. In our previous work, however, we have observed extremely low data loss for even large-scale systems [8] with thousands of processes. Upon a successful reception of the message, the receiver will send an ACK to the sender, allowing the sender to mark the send operation

complete.

5) *Dealing with GRH Data*: As noted in Subsection IV-A, the GRH data is placed into the first 40 bytes of the receive buffer. Since the GRH data is for routing and not data that we wish to have in the receive buffer, we must avoid placing the GRH data there. As such, we use a *scatter list* of InfiniBand to specify two buffers in a receive descriptor – the first with a length of 40 bytes that maps to a temporary buffer and a second entry that maps to the receive buffer.

V. EXPERIMENTAL EVALUATION

Our experimental test bed is an InfiniBand Linux cluster. Each compute node has dual 2.8 GHz Opteron 254 single-core processors. Each node has a Mellanox MT25208 dual-port Memfree HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [16], OFED 1.1 release. The proposed design is integrated into the UD communication device of MVAPICH [21] previously designed in [8]. MVAPICH is a popular open-source MPI implementation over InfiniBand and iWARP. It is based on MPICH [22] and MVICH [23].

We evaluate the following three transfer methods at the MPI layer:

- *Reliable Connection (RC-rdma)*: For measuring RC MPI characteristics we use MVAPICH 0.9.9. Large messages are transferred using the rendezvous RDMA write operation described in Section II-B.
- *Unreliable Datagram - Copy-Based (UD-copy)*: The MPI library used for evaluating this is the same as used in [8], which was shown to scale to thousands of processes with good performance.
- *Unreliable Datagram - Zero-Copy (UD-zcopy)*: This is an implementation of the proposed design in Section IV-B. Aside from the zero-copy protocol, the MPI library is identical to that of UD-copy. The zero-copy threshold is set to be 32 KB, meaning all messages 32 KB and above will use the zero-copy protocol.

A. Basic Performance

In this subsection we evaluate the microbenchmark performance of latency, uni-directional bandwidth, and bi-directional bandwidth using the OSU Benchmarks [24].

1) *Latency*: To measure latency we use a ping-pong test and report one half of the value as the one-way latency. The latency results for large messages are shown in Figure 4. Latencies for messages smaller than 32 KB do not change with our method since we have set the threshold to 32 KB for zero-copy transfers. The gap between UD-copy and RC-rdma is very significant – UD-copy latency is nearly double that of RC-rdma for a 1 MB message. Due to the overhead incurred through segmentation and posting of receive buffers, UD-zcopy does not fully close the gap. The proposed design, however, shows only a 28% increase in latency over the RC-rdma implementation, a significant improvement over the copy-based approach.

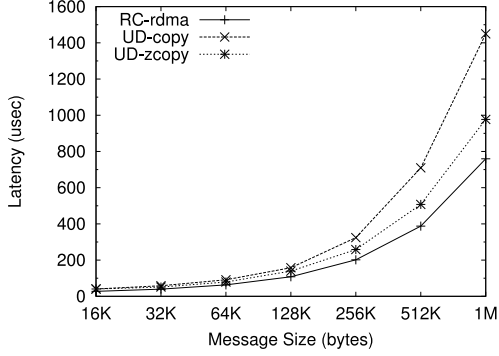


Fig. 4. One-way Latency

When compared to RC-rdma, the performance of the UD-zcopy design performs very well, although many overheads discussed in Section IV-B do reduce the performance from the RC-rdma level. In particular, posting explicit receive buffers instead of RDMA and an additional DMA operation to split the GRH field are significant costs.

2) *Uni-Directional Bandwidth*: In the uni-directional bandwidth benchmark, we evaluate the bandwidth of increasing message sizes for each of the transfer methods. This benchmark performs a throughput test between a sender and receiver where for each message size the sender sends a window of 64 messages and waits for an acknowledgment of the entire window from the receiver. This step is repeated for many iterations and the average is reported.

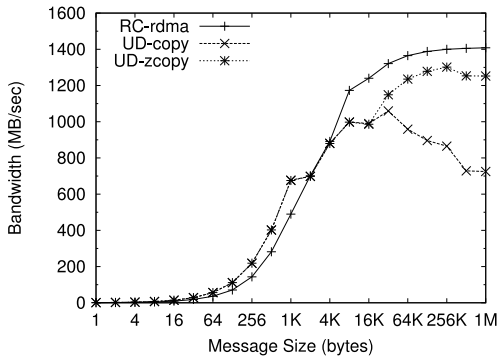


Fig. 5. Uni-Directional Bandwidth

Figure 5 shows the results of the experiment. We can clearly observe the 32 KB threshold where the zero-copy protocol begins to be used. Already at 64 KB the difference between the UD-copy and UD-zcopy methods reaches 275 MB/sec, a nearly 30% improvement. The difference is even greater for larger message sizes; 1 MB messages are transferred over 500 MB/sec faster using the proposed UD-zcopy design. Due to the various overheads incurred, the bandwidth as compared to RC-rdma is lower, however, the performance is within 9% of

the performance at 64KB.

3) *Bi-Directional Bandwidth*: The bi-directional bandwidth benchmark is similar in design to the uni-directional bandwidth benchmark presented earlier. However, in this case, both sides of the communication send as well as receive data from the remote peer.

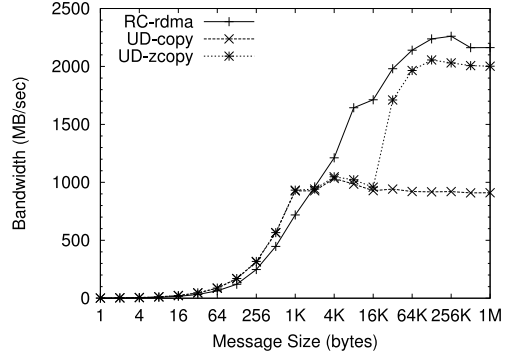


Fig. 6. Bi-Directional Bandwidth

From Figure 6 we can observe a steep increase in performance at the zero-copy threshold of 32 KB. At 64 KB the difference in performance between UD-copy and UD-zcopy is over 1 GB/sec. As compared to RC-rdma, the UD-zcopy design is slightly worse – as expected from the additional overheads, however, for all message sizes where the zero-copy protocol is being used the bandwidth difference is only 150 MB/sec, or a 7% degradation for a 1 MB message versus a nearly 60% degradation for the copy-based approach.

B. Application Benchmarks

In this section we evaluate the three messaging transfer modes against application-based benchmarks. These are more likely to model real-world use than microbenchmarks. We evaluate a molecular dynamics application and the NAS Parallel Benchmarks (NPB).

1) *LAMMPS Benchmark*: Large-scale Atomic/Molecular Massively Parallel Simulator (LAMMPS) [25] is a classical molecular dynamics simulator from Sandia National Laboratories. Several benchmark datasets are available from the LAMMPS website. They are meant to represent a range of simulation styles and computational expense for molecular-level interaction forces. In our experimental analysis, we used the Rhodospin protein (*in.rhodo*) and Polymer chain melt (*in.chain*) datasets available from the LAMMPS website. LAMMPS reports the “Loop Time” for a particular benchmark as a measure of CPU time required to simulate a set of interactions.

Figure 7(a) shows the normalized time to complete the simulation and Table II shows the messaging characteristics of the different data sets. For the *in.rhodo* dataset the performance of UD-zcopy is significantly increased over that of UD-copy and near equal to that of RC-rdma. This improvement can

TABLE II
MESSAGING CHARACTERISTICS (16 PROCESSES)

App.	Benchmark	Percentage			Total Messages		
		0-2KB	2-32KB	32KB+	0-2KB	2-32KB	32KB+
LAMMPS	in.chain	29.65	70.35	0.00	166308	394618	0
	in.rhodo	41.37	47.31	11.31	150238	171804	41088
NPB	CG.B	57.62	0.00	42.38	257902	0	189696
	EP.B	100.00	0.00	0.00	384	0	0
	FT.B	19.50	0.00	80.50	1364	0	5632
	IS.B	61.77	6.04	32.20	5402	528	2816
	LU.B	99.00	0.00	1.00	2401046	0	24192
	MG.B	57.99	24.74	17.28	56706	24192	16896
	SP.B	0.48	0.00	99.52	744	0	154080

be attributed to the 11% of messages that are greater than or equal to 32 KB and are benefiting from the zero-copy transfer mechanism. Since the `in.chain` dataset does not have messages 32 KB or over, the performance of both UD-copy and UD-copy are the same. RC-rdma performs similarly due to the lack of large messages as well.

2) *NAS Parallel Benchmarks*: The NAS Parallel Benchmarks [26] are a selection of kernels that are typical in various Computational Fluid Dynamics (CFD) applications. As such, they are a good tool to evaluate the performance of the MPI library and parallel machines.

The results of evaluating each of the methods on the NAS benchmarks can be found in Figure 7(b). For many of the benchmarks where there is a significant number of large send operations (FT, IS, and CG), we see a large difference in performance between UD-copy and RC-rdma – for IS the performance of UD-copy is nearly 40% worse. Our proposed zero-copy design, however, reduces this overhead considerably. Performance is increased by 4%, 3%, 16%, 2%, and 3% for CG, FT, IS, MG, and SP, respectively. As we can observe from comparing Table II and our percentage improvement, although some benchmarks use many large messages, such as SP, it may not be bandwidth limited and communication can be overlapped with communication.

Aside from IS, where the application is extremely short running and the communication time can become dominated by setup overheads, the absolute application time is very similar between RC-rdma and UD with the addition of the zero-copy protocol.

VI. RELATED WORK

Zero-copy protocols have been widely used in MPI implementations. MPICH-PM [15] used a zero-copy protocol over the PM communications library. The design of the protocol uses a Request to Send (RTS), Clear to Send (CTS) and Send Fin messages. MPI-BIP [14] also used zero-copy message transfers to reduce copy overhead and boost application performance. This work noted that the communication bandwidth was comparable to memory bandwidth and thus, zero-copy would be an attractive protocol. This observation holds good for modern systems. RDMA had been emulated over InfiniBand UD in [27] using a copy-based method, however no zero-copy operations were designed or evaluated.

RDMA is provided using a UD-based approach with reliability provided by MPI in the IBM HPS adapter [20], however, each UD packet is self-identifying as to its address. This is unlike InfiniBand where RDMA is not enabled for UD.

Although these research works focused on utilizing zero-copy protocols for performance reasons, there was no special treatment of these protocols with regard to scalability. Scalability of MPI libraries over InfiniBand has been a topic of much recent research. Efforts in these research works focuses on reducing communication buffer requirements by utilizing SRQ [10], [11]. In addition, the Reliable Connection memory utilization has been tackled in [7]. Further, connection-less MPI designs have been proposed in [8]. While there are many research works which focus individually on zero-copy protocols and scalability of MPI libraries, to the best of our knowledge, this is the first work on combining both these objectives in a scalable, high-performance MPI design and implementation for InfiniBand.

VII. CONCLUSION

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 a UD-based MPI design appears to be a promising alternative to those using RC connections, which require several KB per additional communicating peer. An issue with the traditional approach for message transfer over a non-RDMA transport, one in which copies are made, is that high-performance is not maintained.

In this work we have proposed a zero-copy protocol for large message transfers over the UD transport. Having a high-performance UD-based message transfer protocol allows the MPI to avoid using the RC transport, thus saving a significant amount of memory per process. Evaluation with microbenchmarks show an increase in bi-directional throughput of nearly 100% over a copy-based approach and only slightly degraded performance compared to RC RDMA-based approaches that use additional memory. Application benchmark evaluation shows the zero-copy design gives a 21% speedup for the `in.rhodo` dataset for LAMMPS over a copy-based approach, giving performance within 1% of RC. Performance on the NAS Parallel Benchmarks was improved by up to 16%

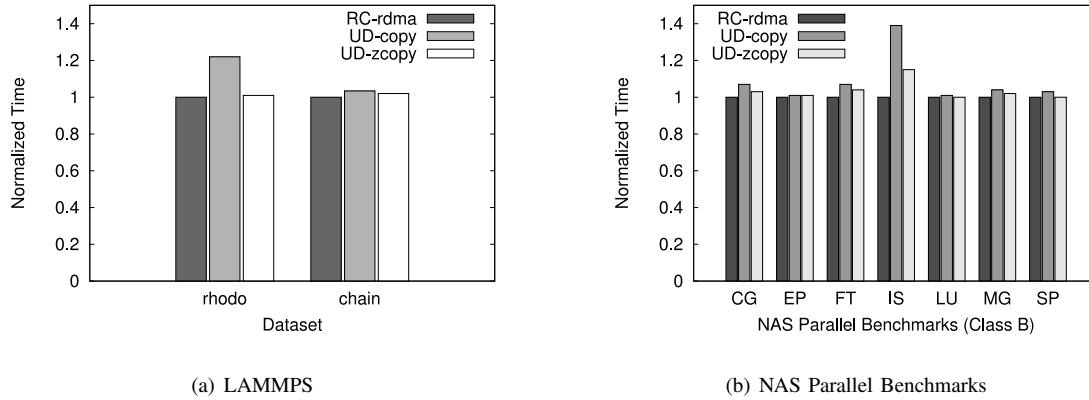


Fig. 7. Evaluation Normalized Time (16 processes)

for IS and performance on other benchmarks was improved as well.

Future work along this direction is planned to evaluate the zero-copy design on larger-scale clusters. We would also like to evaluate using multiple paths through the network to avoid potential hotspots when large message transfers are occurring, a feature that can be used in UD without requiring an additional QP as with the RC transport. Additionally, the next Mellanox HCA, ConnectX, will provide a new Scalable Reliable Connection transport that we hope to evaluate [28], [29]. We would also like to continue to look at other applications and specific patterns that can be optimized for UD communication.

ACKNOWLEDGMENT

This research is supported in part by Department of Energy's grant #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CCF-0702675; grants from Intel, Mellanox, Cisco, Sun Microsystems, and Linux Networx; Equipment donations from Intel, Mellanox, AMD, Advanced Clustering, IBM, Apple, Appro, Microway, PathScale, Silverstorm and Sun Microsystems.

REFERENCES

- [1] InfiniBand Trade Association, "InfiniBand Architecture Specification," <http://www.infinibandta.com>.
- [2] Sandia National Laboratories, "Thunderbird Linux Cluster," <http://www.cs.sandia.gov/platforms/Thunderbird.html>.
- [3] NAS Project: Columbia, "Columbia Supercomputer," <http://www.nas.nasa.gov/About/Projects/Columbia/columbia.html>.
- [4] Lawrence Livermore National Laboratory, "MI&E Atlas (Peloton)," <http://www.llnl.gov/computing/hpc/resources/>.
- [5] Texas Advanced Computing Center (TACC), "Lonestar," <http://www.tacc.utexas.edu/resources/hpcsystems/>.
- [6] *MPI: A Message-Passing Interface Standard*, Message Passing Interface Forum, Mar 1994.
- [7] 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.
- [8] 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.
- [9] 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.
- [10] 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.
- [11] G. Shipman, T. Woodall, R. Graham, and A. Maccabe, "Infiniband Scalability in Open MPI," in *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [12] "Mellanox Technologies," <http://www.mellanox.com>.
- [13] T. von Eicken, A. Basu, V. Buch, and W. Vogels, "U-net: a user-level network interface for parallel and distributed computing," in *SOSP '95: Proceedings of the fifteenth ACM symposium on Operating systems principles*. ACM Press, 1995, pp. 40–53.
- [14] L. Prylli and B. Tourancheau, "BIP: a new protocol designed for high performance networking on Myrinet," in *IEEE Parallel and Distributed Processing Symposium (IPDPS)*, 1998.
- [15] F. Carroll, H. Tezuka, A. Hori, and Y. Ishikawa, "MPICH-PM: Design and implementation of zero copy MPI for PM," 1998.
- [16] OpenFabrics Alliance, "OpenFabrics," <http://www.openfabrics.org/>.
- [17] Myricom Inc., "MPICH-MX," <http://www.myri.com/scs/download-mpichmx.html>.
- [18] QLogic, "InfiniPath," <http://www.pathscale.com/infinipath.php>.
- [19] Quadrics, "QsNET II," <http://www.quadrics.com/quadrics>.
- [20] R. Sivaram, R. K. Govindaraju, P. Hochschild, R. Blackmore, and P. Chaudhary, "Breaking the connection: Rdma deconstructed," in *HOTI '05: Proceedings of the 13th Symposium on High Performance Interconnects*, 2005, pp. 36–42.
- [21] Network-Based Computing Laboratory, "MVAPICH: MPI over InfiniBand and iWARP," <http://mvapich.cse.ohio-state.edu>.
- [22] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI, Message Passing Interface Standard," Argonne National Laboratory and Mississippi State University, Tech. Rep.
- [23] Lawrence Berkeley National Laboratory, "MVICH: MPI for Virtual Interface Architecture," <http://www.nersc.gov/research/FTG/mvich/index.html>, August 2001.
- [24] Network-based Computing Laboratory, "OSU Benchmarks," <http://mvapich.cse.ohio-state.edu/benchmarks>.
- [25] S. J. Plimpton, "Fast Parallel Algorithms for Short-Range Molecular Dynamics," *Journal of Computational Physics*, vol. 117, pp. 1–19, 1995. [Online]. Available: <http://lammps.sandia.gov/index.html>
- [26] 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," vol. 5, no. 3, Fall 1991, pp. 63–73.
- [27] 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*. ACM Press, 2007, pp. 46–54.
- [28] Mellanox Technologies, "ConnectX Architecture," http://www.mellanox.com/products/connectx_architecture.php.
- [29] 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 (HotI5)*, August 2007.