Improving RDMA-based MPI Eager Protocol for Frequently-used Buffers

Mohammad J. Rashti Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University, Kingston, ON, CANADA K7L 3N6
mohammad.rashti@ece.queensu.ca ahmad.afsahi@queensu.ca

Abstract

MPI is the main standard for communication in high-performance clusters. MPI implementations use the Eager protocol to transfer small messages. To avoid the cost of memory registration and pre-negotiation, the Eager protocol involves a data copy to intermediate buffers at both sender and receiver sides. In this paper, however, we propose that when a user buffer is used frequently in an application, it is more efficient to register the sender buffer and avoid the sender-side data copy. The performance results of our proposed Eager protocol on MVAPICH2 over InfiniBand indicate that up to 14% improvement can be achieved in a single medium-size message latency, comparable to a maximum 15% theoretical improvement on our platform. We also show that collective communications such as broadcast can benefit from the new protocol by up to 19%. In addition, the communication time in MPI applications with high buffer reuse is improved using this technique.

1. Introduction

Message Passing Interface (MPI) [12] is the de-facto communication library for parallel programming in High-Performance Computing (HPC) clusters. MPI implementations over modern RDMA-enabled cluster interconnects such as InfiniBand [6] take advantage of benefits that RDMA [4] brings to inter-node communication, such as operating system bypass, lower CPU utilization, and zero-copy data transfer.

MPI implementations treat small and large messages differently. Particularly in MPICH2-based implementations [13], an Eager protocol is used to eagerly transfer small messages to the receiver to avoid extra overhead of pre-negotiation. Due to the reliability of the underlying network transport (InfiniBand reliable connection transport in this work), the Eager protocol does not require an acknowledgment to ensure completion of the data transfer, and therefore the sender can immediately finalize the communication as soon as the data is sent on the wire.

For large-size messages, a Rendezvous protocol [8] is used in which a negotiation phase makes the receiver ready to receive the message data from the sender. After the data transfer, a finalization packet is sent by the sender to inform the receiver that the data is placed in its appropriate application buffer.

RDMA-based communication requires the source and destination buffers to be registered to avoid swapping memory buffers before the DMA engine can access them [10]. Memory registration is an expensive process that involves buffer pin-down and virtual-physical address translation [10]. In addition, the registration tag needs to be advertised to the remote node. This is why in the Eager protocol the small messages are copied from application buffers into pre-registered and pre-advertised intermediate buffers. For the Rendezvous protocol though, the application buffers are registered to be directly used for zero-copy transfer.

The described implementation for the Eager protocol is optimized based on the assumption that the application buffer is not used frequently. In contrast, the reality is that most of user buffers in MPI applications are frequently used during the course of execution. This feature, the buffer reuse in MPI applications, is the main motivation behind this work. We propose to register the frequently-used application buffers so that we could initiate RDMA operations directly from the application buffers rather than the intermediate buffers (infrequently-used buffers are treated as before). This way, we can decrease the cost of communication by skipping the sender-side data copy. The registered user buffer can be kept in MPI registration cache and later on be retrieved in subsequent references to the same buffer. Therefore, the cost of one-time registration is amortized over the cost of multiple data copies. Note that the receiver-side data copy is still required, because we are

carrying out an Eager transfer, which assumes no negotiation with the receiver process.

We have implemented the proposed Eager protocol on MVAPICH2 [14] over InfiniBand and have observed up to 14% and 19% improvement for ping-pong message latency and broadcast operation, respectively. In addition, up to 11% improvement in communication time for some MPI applications is reported.

The rest of this paper is organized as follows. In Section 2, we describe the motivation behind this work. Details about our proposed Eager protocol will follow in Section 3. We also present an online adaptation mechanism to minimize the overhead on applications that do not reuse their buffers frequently. Experimental framework and performance results are covered in Section 4 and Section 5, respectively. We discuss the related work in Section 6, and conclude the paper in Section 7.

2. Motivation

The current Eager protocol copies the user buffer data into an intermediate buffer to avoid the higher cost of memory registration. However, if the user buffer is used frequently in an application, it can be registered to avoid the data copy. For this, we need to investigate whether MPI applications reuse their data buffers frequently during the execution time.

Table 1 shows the Eager buffer reuse statistics for some MPI applications, described in Section 4.1. The second column shows the range (and the average) of the maximum number of times a user buffer is reused among 16 processes. The third column shows the most frequently-used buffer sizes among 16 processes for these applications. Table 1 confirms that indeed application buffers are reused frequently in most processes of the applications under study.

Table 1. Eager buffer reuse statistics for MPI applications running with 16 processes

MPI Application	Buffer reuse count	Most frequently used buffer sizes (bytes)
NPB CG Class C	7904 - 7904 (Avg: 7904)	8 – 8 (Avg: 8)
NPB LU Class C	3749 - 3750 (Avg: 3749)	1560 - 1640 (Avg: 1600)
ASC AMG2006	45 - 292 (Avg: 119)	8 - 3648 (Avg: 770)
SPEC MPI2007 104.MILC	2 - 5010 (Avg: 2049)	8 - 4800 (Avg: 2876)

In the current Eager protocol, the buffer copy at the source and destination contributes to overall message latency. Now that there is a potential to revise the Eager protocol for the frequently-used buffers, we would like to know how much improvement we can theoretically achieve if we remove the sender-side copy operation.

Figure 1 compares the ping-pong message latency with the cost of one buffer copy for Eager-size messages on our InfiniBand cluster using MVAPICH2 (refer to Section 4 for the description of our experimental platform). Performance results suggest that up to 15% improvement can be achieved by bypassing the sender-side copy for Eager-size messages. Note that this analysis does not take into account the one-time cost of registration as well as the implementation overhead. It should be mentioned that MVAPICH2 switches from Eager to the Rendezvous protocol at about 9KB on our platform. That is why the experimental results in this paper are shown for up to 8KB messages. We have not changed the default Eager/Rendezvous switching point in our study because it is the optimal value calculated for our platform. While a higher switching point is expected to provide a larger improvement for our proposed technique, it would degrade the baseline communication performance of the MVAPICH2 implementation.

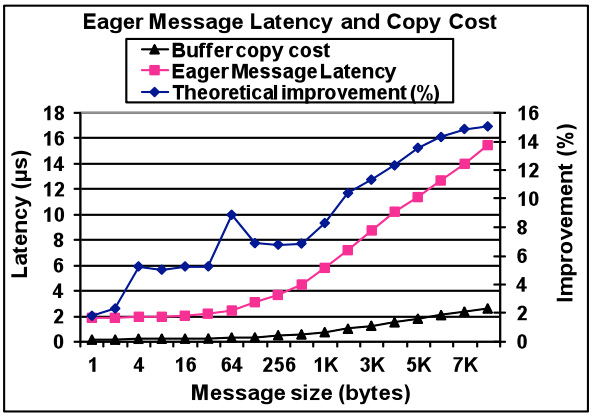

Figure 1. Eager message latency, buffer copy cost and theoretically expected improvement

3. Frequent-buffer Eager Communication

In this section, we describe the details of the proposed Eager communication improvement. Section 3.1 describes the general behavior of the proposed Eager communication. In Section 3.2, we present details about detecting frequently-used buffers at runtime. Finally, Section 3.3 proposes an adaptation mechanism to minimize the overhead on

applications that do not benefit from the new protocol due to their low buffer reuse statistics.

3.1. Eager Communication Mechanism

Figure 2 depicts the general idea behind bypassing the send-side copy through application buffer registration. The current general path for Eager messages is shown in dashed arrows, while the new path for frequently-used buffers is in solid arrows. Essentially, if an application buffer is used frequently it will be registered so that the user data can be directly transferred from the application buffer by an RDMA operation. Therefore, the communication cost is reduced by avoiding the data copy into a pre-registered buffer.

However, as noted earlier, we cannot avoid the copy into the receiver's intermediate buffer due to the Eager communication semantics. In order for the receiver side to detect reception of Eager messages, there is a need to transfer the Eager message header along with the Eager payload. The Eager header constitutes a small amount of data (especially when header caching is enabled [5]) that is still copied into the intermediate buffers. We use InfiniBand RDMA scatter/gather mechanism to gather the header information from a pre-registered buffer and the user data from the application buffer, and then transfer them together into the receiver's pre-registered buffer.

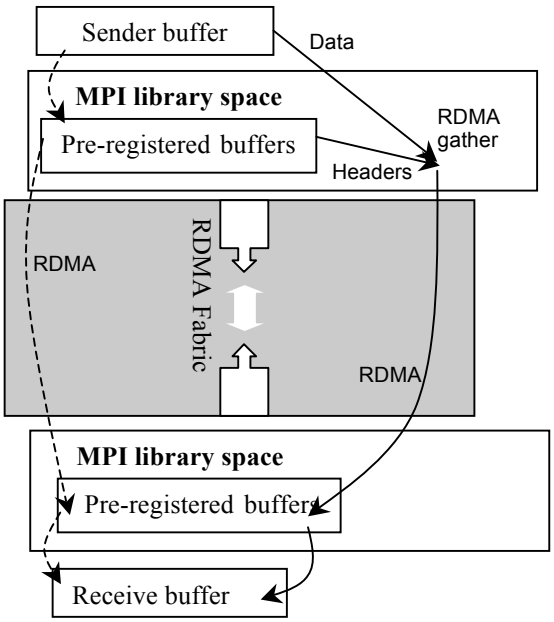

Figure 2. Data flow diagram for the current and proposed Eager protocol implementation

We keep the registered application buffers in MVAPICH2 registration cache so they could be retrieved in subsequent references. MVAPICH2 uses a lazy de-registration mechanism to avoid re-registrations in future buffer reuses [11]. In this mechanism, each previously registered buffer remains registered and its pointer is stored in a binary tree. Instead of actually registering and deregistering the buffer, its reference count is increased/decreased each time the buffer is needed to be registered/deregistered. Only when the registration cache is full, a buffer with zero reference count will be evicted from the cache and then deregistered. It is worth mentioning that to keep the information in the registration cache coherent with the operating system virtual memory changes, a synchronization with the operating system kernel is required that may negatively affect the performance of the registration cache [18].

3.2. Detection of Frequently-used Buffers

The proposed mechanism is employed only for buffers that are frequently used. To be able to detect frequently-used buffers, we need to keep track of the buffer usage statistics. When a buffer is accessed for an Eager transfer, the buffer is searched in a data structure containing buffer usage statistics. If the buffer is found in the table and its usage statistics has surpassed a pre-calculated threshold, the algorithm looks for the buffer in the registration cache (for possible previous registration). A buffer that is not found in the cache will be registered.

Now the question is: which buffer is considered frequently-used? In other words, what is the lowest value of the reuse threshold from which this mechanism can yield benefit? To realize the answer, we need to calculate the timing costs associated with both communication paths depicted in Figure 2.

The current Eager communication path involves a copy to/from the pre-registered buffer at sender/receiver plus an RDMA transfer between the two pre-registered buffers. In the proposed technique, we do not have the first copy, but we have an extra memory registration at the sender side. If we consider C_m as the copy cost, NT_m as the network transfer time, and R_m as the registration cost for a single message with size m, and V as the implementation overhead for the new method, the current and new communication times (T_c and T_n, respectively) when we reuse the buffer n times will be:

$$T_c = n \times (2 \times C_m + NT_m) \quad (1)$$

$$T_n = n \times (C_m + NT_m + V) + R_m \quad (2)$$

In order to benefit from the new method, we should find the minimum n such that $T_n < T_c$:

$$n > \frac{R_m}{C_m - V} \quad (3)$$

The value V is the overhead incurred in searching both the table containing buffer usage statistics and the buffer registration cache. The buffer usage statistics are stored in a hash table structure (Refer to Section 3.2.1), and the registration cache is in a balanced binary tree.

Inequality (3) shows the minimum number of times a buffer of size m needs to be reused after registration, in order to benefit from the new method. Table 2 shows the minimum value of n for different message sizes on our platform. The value, n, is negative for messages smaller than 64 bytes due to $V > C_m$. Even with very high number of reuses, there will be no benefit, and therefore we have disabled the proposed method for such messages.

Our algorithm dynamically decides when a buffer needs to be registered. The registration threshold is based on the pre-calculated values, shown in Table 2. However, we have devised the algorithm in such a way that it can speculatively decide to register a buffer when it is reused at least by a certain portion (e.g. 25%, 50% or 100%) of the minimum number, n, hoping that the buffer will be reused for more than that later. As an example for the 25% case (used in our experiments), a 4KB buffer is registered when it is reused 14 times (25% of 59), hoping that it will be reused again 59 times or more so that the registration cost is amortized.

Table 2. Minimum required buffer reuse number, n, for different message sizes on our platform

Message size (bytes): Minimum *n*	Message size (bytes): Minimum *n*
128B: 1280	2KB: 109
256B: 755	4KB: 59
512B: 432	8KB: 33
1KB: 200	

3.2.1. Searching the Buffer Usage Table. In order to minimize the overhead, V, we have experimented with two different data structures for buffer usage table: hash table and balanced binary tree [11]. Our hash table has 1M hash buckets, and each buffer address is hashed into a 20-bit index. In each hash bucket, the buffers with conflicting hash values are stored in a linked list. For the hashing function, we have chosen a multiplicative hash method proposed for Linux buffer cache [7] that is known to be efficient for hashing buffer addresses.

As shown in Figure 3, with the growth of the database (the number of Eager-size buffers in the table), the search time in the hash table grows very slowly, compared to that of the binary tree. Based on these results, we have chosen the hash table for our buffer usage search structure.

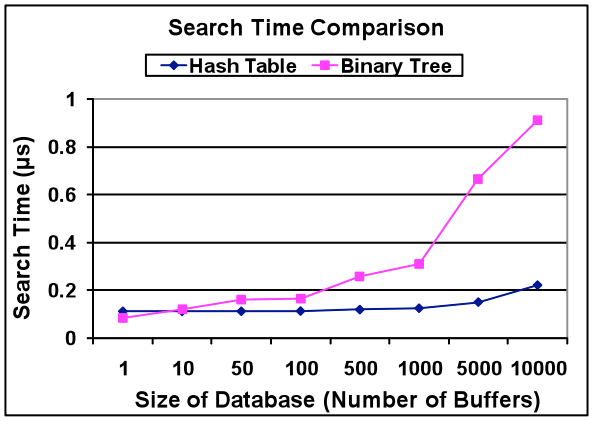

Figure 3. Comparing search time between hash table and binary tree.

3.3. Adaptation

The idea of registering the Eager message buffers is useful for applications with a high buffer reuse profile. For applications with low buffer reuse, although we normally do not incur the buffer registration cost for Eager messages, the overhead of manipulating buffer usage statistics can affect the overall performance of the Eager protocol, especially for small messages with latencies comparable to the overhead.

There are two solutions to such a problem. First, is to get the static profiles of the applications and use them to register frequently-used buffers when the program starts running. However, this approach does not work for applications with dynamic profiles. In addition, it requires extra provisioning for application profiling.

The approach we have used in this work is based on the dynamic monitoring of application buffer usage at runtime. We dynamically monitor the overall buffer usage statistics of the application, and adaptively stop the growth of the buffer usage table when the overall buffer usage statistics is low.

To avoid early stoppage, the adaptation acts only when the hash table starts growing linked lists in its hash buckets. This is because the overhead of hash table search/insertion increases dramatically when the hash values for message buffers conflict and the hash

bucket linked-lists grow. Otherwise, the search time is of order O(1) regardless of the number of buffers in the hash table. In summary, adding more buffers to the hash table is stopped when the following two conditions are satisfied:

(1) When the number of (registered) buffers in the hash table that are marked as frequently-used is less than 20% (a typical value) of all buffers in the table; and

(2) When the hash table has started to grow linked lists (due to linear search/insertion costs).

4. Experimental Framework

We have conducted our experiments using four Dell PowerEdge 2850 servers, each with two dual-core 2.8GHz Intel Xeon EM64T processors (2MB of L2 cache per core) and 4GB of DDR-2 SDRAM. Each node has a Mellanox ConnectX DDR InfiniBand HCA [9] installed on an x8 PCI-Express slot, interconnected through a Mellanox 24-port Infiniscale-III switch.

In terms of software, we are using MVAPICH2 1.0.3 [14] over OpenFabrics Enterprise Distribution (OFED) 1.3, installed on Linux Fedora Core 5, kernel 2.6.20.

4.1. MPI Applications

We have considered four applications in evaluating the proposed Eager protocol: CG and LU benchmarks from NPB 2.4 benchmarks [15], AMG2006 from ASC Sequoia suite [1], and 104.MILC from SPEC-MPI 2007 suite [16].

CG [15] solves an unstructured sparse linear system using the conjugate gradient method. CG mostly uses MPI send/receive and barrier operations [3]. LU [15] is a simplified compressible Navier-Stokes equation solver. LU mostly relies on MPI blocking (and a few non-blocking) send/receive, and some broadcast, all-reduce and barrier operations [3, 16].

AMG2006 [1] is a parallel algebraic multi-grid solver for linear systems arising from problems on unstructured grids. It uses blocking and non-blocking MPI send/receive calls and collectives such as broadcast, gather, scatter, all-reduce, all-to-all, and all-gather.

104.MILC [16] simulates four-dimensional SU(3) lattice gauge theory. The benchmark is for the conjugate gradient calculation of quark propagators in quantum dynamics. It uses non-blocking MPI send/receive calls and some broadcast and all-reduce collectives.

5. Experimental Results

In this section, we present our experimental results in evaluating the proposed Eager protocol using point-to-point and collective communication micro-benchmarks as well as MPI applications.

5.1. Micro-benchmark Results

5.1.1. Ping-pong Latency. The ping-pong operation is repeated 10000 times and the average one-way latency is measured for different messages in the Eager communication range, from 128 bytes to 8KB. Two different buffers are used in each process: one for send and the other for the receive operation. While keeping send and receive buffers separated, this method leads to minimal hash table and registration cache overhead.

Figure 4 shows the amount of improvement in the Eager message latency when the send-side copy is removed and the user buffer is registered for RDMA transfer. The maximum improvement is around 14% for 8KB messages, very close to the 15% theoretical improvement discussed in Section 2. As stated earlier, the new method is disabled for messages smaller than 128 bytes due to registration cost and implementation overhead. Starting from 128 bytes (where the data is being transferred using DMA, instead of PIO inline method), the experimental results are getting closer to the theoretical benefit.

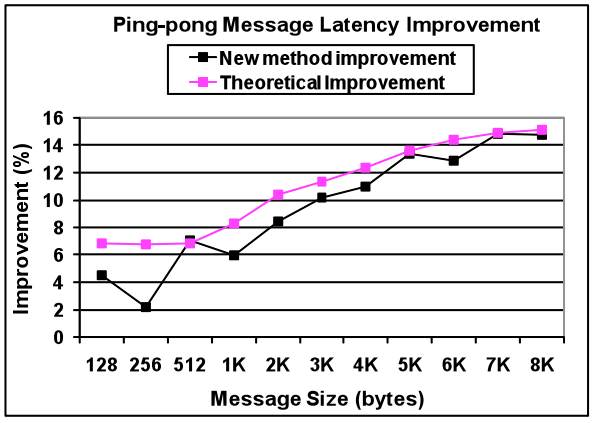

Figure 4. Message latency improvement

5.1.2. Spectrum Buffer Reuse Latency. The ping-pong test examined the case with maximum buffer reuse percentage and high reuse numbers. To simulate an application with different buffer reuse patterns for different buffers, our next micro-benchmark is using a spectrum of buffer reuse patterns by increasing the reuse count of each buffer from 1 to 1000. We have considered a 1000-unit buffer set: $\{buf_i \mid 1 < i < 1000\}$, in which buf_i is being reused i times.

For efficiency and integration purposes, we are directly using MVAPICH2 registration mechanism and registration cache. MVAPCIH2 is registering application buffers in one-page chunks (4KB on our system). Thus, we have chosen two buffer allocation schemes: in one case, the buffers are allocated back to back in memory. In the other case, the buffers are allocated in separate memory pages, without any page overlap. In the case that the buffers are allocated back to back, registration of one buffer may result in registration of a page that is overlapped with the next buffer, slightly decreasing the registration cost for the next buffer. Thus, the separate-page allocation case essentially shows the worst case scenario for buffer registration cost.

Results for this micro-benchmark, using both buffer allocation schemes are shown in Figure 5. The effect of buffer allocation scheme is evident for smaller messages, but almost vanishes as messages approach the page size and beyond. The highest improvement (12.7%) is again for 8KB messages.

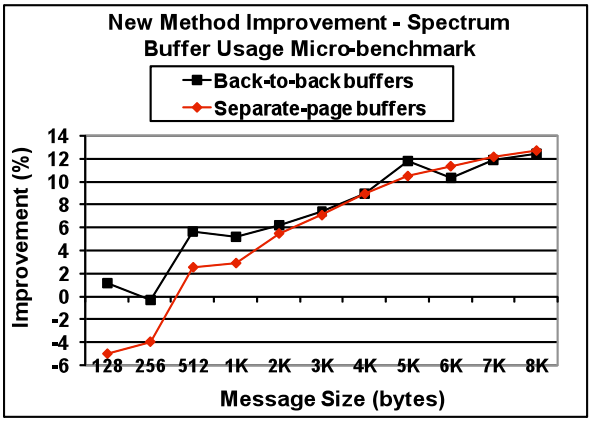

Figure 5. Improvement as observed in spectrum buffer usage benchmark

5.1.3. Collective Communications. Most of the MPI collective operations use MPI point-to-point primitives to implement the collective algorithm. Therefore, improving the performance of point-to-point Eager communication can have an indirect effect on collective operations as well. In this section, we present the effect of our Eager protocol improvement on MPI broadcast and scatter collective operations.

In each iteration of the micro-benchmark, all processes are engaged in the collective operation, followed by a synchronization operation. We repeat the test for 10000 times and calculate the average time of the collective across all processes. The synchronization is used to prevent process skew from propagating in subsequent iterations of the micro-benchmark.

Figure 6 shows the performance improvement for MPI-Broadcast and MPI-Scatter operations running with 4 processes and 16 processes on our 4-node cluster. Broadcast and Scatter use the binomial tree algorithm for data transfer in the range of messages under study. The only difference is that scatter distributes data from distinct but back-to-back buffers, while broadcast distributes data from a single source buffer among processes.

The results for the 4-process cases are relatively high, even higher than the point-to-point results (for broadcast), because more than one data copy is saved per collective. In the 16-process case, intra-node communications are done through shared memory, and thus such communications will not use our Eager technique, resulting in a lower improvement percentage compared to the 4-process cases.

The curves for the scatter plunge to around 0% for messages larger than 4KB. The reason is that in the first subdivision step of the binomial tree algorithm the root transfers half of the data (at least two buffers) to the first intermediate node. Therefore, the data size exceeds the Eager/Rendezvous switching point (9KB) for messages larger than 4.5KB. In addition, since for intermediate transfers temporary buffers are used, we cannot expect reuse of those buffers. Therefore, our algorithm is inactive for scattering messages larger than 4.5KB.

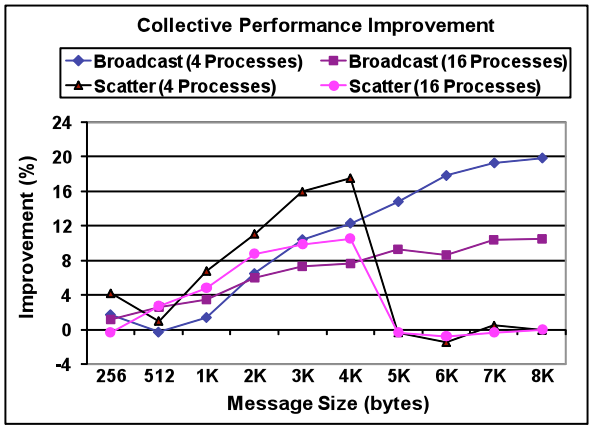

Figure 6. Collective improvement results

5.2. Results for MPI Applications

In this section, we evaluate the effect of the new Eager transfer method on some MPI applications that have been chosen for this study because of their different buffer reuse characteristics.

In our evaluation, we measure three values for each application: the amount of time spent in MPI send operations; the total communication time spent in MPI send, MPI receive and MPI wait operations; and

the application execution time. Figure 7 shows the improvements achieved using the proposed method. Table 3 shows the frequently-used buffer percentage for the applications. These values show the ratio of the frequently-used registered buffers over all Eager buffers in the buffer reuse hash table.

Both MILC and LU have a high buffer reuse profile. That is why they benefit from the proposed technique. Their send times gain 8.1% and 10.6% improvement, respectively. Obviously, not all of these gains translate into total communication time improvement. In fact, there are some general factors affecting the overall gain in total communication time and application runtime. The following list summarizes them:

- Level of synchronization among processes and the time that a matching receiver is arrived: in case of skewed receivers, the send-time improvement will vanish during the skew time.
- The amount of communication/computation overlap: the more application's communication is overlapped with computation, the less we see the improvement being translated into total application runtime improvement.
- The ratio of total communication time to the application runtime: applications with lower communication/computation time ratio will see a smaller portion of the communication time gain reflected into the application runtime.

For example, MILC is frequently using MPI synchronization operations while LU is just using it at the beginning of the communication. Therefore, in MILC more of the MPI send time improvement is translated into total communication time improvement.

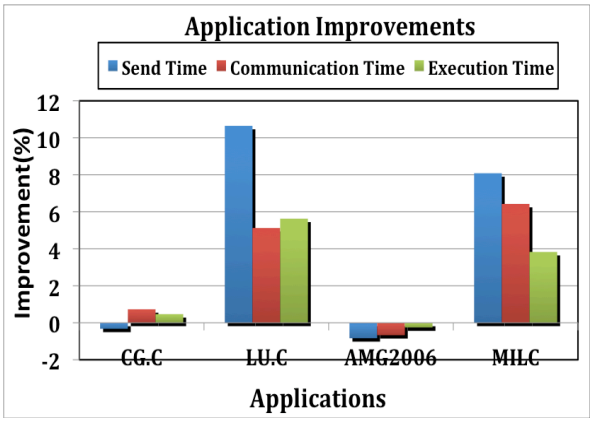

Figure 7. Application improvement results

CG and AMG2006 are applications with lower buffer reuse statistics. Our method has a slight effect on their performance (less than 1%), which is within our measurement error. Although CG buffer reuse statistics shown in Table 1 are very high, those are only for small messages (8 bytes). Since we have disabled the method for messages smaller than 128 bytes, our technique is effectively disabled for CG.

Table 3. Frequently-used buffer percentage for MPI applications (for messages > 128B)

MPI Application	Most frequently-reused buffer percentage
NPB CG Class C	0%
NPB LU Class C	32.29%
ASC AMG2006	9%
SPEC MPI2007 104.MILC	22.55%

The reason that AMG2006 does not gain is that its buffer reuse statistics are very close to the speculative threshold point from which our implementation starts to register buffers. However, those threshold values are 25% of the minimum required reuse statistics (shown in Table 2). Thus, AMG2006 suffers from the overhead of registering not sufficiently reused buffers.

AMG2006 has a low buffer reuse ratio as well (Table 3), which is lower than the minimum for our adaptation (20%). Therefore, the adaptation is activated, stopping the buffer reuse table to grow. This reduces the overhead on this application.

6. Related Work

To the best of our knowledge, no similar work has been reported on bypassing the send-side copy for Eager-size messages over RDMA-enabled interconnects. However, there exists some related work on improving the performance of Eager-size messages.

The authors in [17] describe a user-level pipeline protocol that overlaps the cost of memory registration with RDMA operations, which helps achieving good performance even in the cases of low buffer reuse.

In [5], a header cache mechanism is proposed for Eager messages in which the Eager message header is cached at the receiver side and instead of a regular header, a very small header is sent for subsequent messages with matching envelope.

Liu et al. [8] designed an RDMA-based Eager protocol with flow-control over InfiniBand. If the RDMA flow control credits of a connection are used-up without being released by the receiver, the communication falls back on the send/receive channel.

Some works have also looked at the cost of memory registration in RDMA-enabled networks, especially its high costs for small buffers [18, 2].

7. Conclusions and Future Work

In this paper, we have proposed a novel technique to improve the MPI Eager protocol over RDMA-enabled networks. In the proposed method, we avoid the send-side buffer copy, and instead transfer data directly from the application buffer. Our technique is suited for applications with high buffer reuse statistics. However, our adaptation mechanism minimizes the overhead on applications with low buffer reuse profiles.

Micro-benchmark results show that our implementation achieves up to 14% improvement in the point-to-point Eager communication time. This is very close to the maximum theoretical improvement of about 15% on our platform. We have also shown that collective operations, such as broadcast, can achieve even higher improvements because more than one data copy is saved in each operation.

MPI application results confirm that applications with high buffer reuse benefit from this technique. Such applications show much higher improvement for the send-time than the total communication time and application execution time. Process synchronization pattern, communication/computation overlap and the communication/computation time ratio are the deciding factors that may affect the total communication time and application runtime improvement.

For the future work, we plan to run our applications on a larger testbed. We expect our proposed Eager protocol to provide much better application results on a system with a larger node count, as more processes communicate with each other across the network. We would also like to extend our work to the user-level send/receive-based communication model as well as to some RDMA-based collective operations.

8. Acknowledgements

This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC), Canada Foundation for Innovation (CFI), Ontario Innovation Trust (OIT), Queen's University, and Mellanox Technologies. The authors would also like to thank the anonymous reviewers for their insightful comments.

9. References

[1] AMG 2006, "ASC Sequoia Benchmarks": https://asc.llnl.gov/sequoia/benchmarks/.

[2] D. Dalessandro, P. Wyckoff and G. Montry, "Initial performance evaluation of the NetEffect 10 Gigabit iWARP adapter", In *Proceedings of 3rd IEEE Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2006)*, 2006, pp. 1-7.

[3] A. Faraj and X. Yuan, "Communication characteristics in the NAS parallel benchmarks", In *Proceedings of IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2002),* pp. 724–729.

[4] J. Hilland, P. Culley, J. Pinkerton and R. Recio, "RDMA protocol verbs specification (version 1.0)", *RDMA Consortium, 2003.*

[5] W. Huang, G. Santhanaraman, H. Jin and D.K. Panda, "Design alternatives and performance trade-offs for implementing MPI-2 over InfiniBand", In *Proceedings of Euro PVM/MPI, 2005,* pp. 191-199.

[6] InfiniBand Architecture: http://www.infinibandta.org/.

[7] C. Lever, "Linux kernel hash table behavior: analysis and improvements", Technical Report 00-1, Center for Information Technology Integration, University of Michigan, 2000.

[8] J. Liu, J. Wu D.K. Panda, "High performance RDMA-based MPI implementation over InfiniBand", In *Proceedings of 17th annual conference on Supercomputing, 2003,* pp. 295 - 304.

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

[10] F. Mietke, R. Rex, R. Baumgartl, T. Mehlan, T. Hoefler and W. Rehm, "Analysis of the memory registration process in the Mellanox InfiniBand software stack", In *Proceedings of 12th International Euro-Par Conference, Dresden, Germany, 2006*, pp. 124-133.

[11] F. Mietke, R. Rex, T. Mehlan, T. Hoefler, and W. Rehm, "Reducing the impact of memory registration in InfiniBand", In *1st Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC), 2005.*

[12] Message Passing Interface Forum, "MPI, A Message Passing Interface Standard" V2.0.

[13] MPICH2 MPI Implementation: http://www-unix.mcs.anl.gov/mpi/mpich2/.

[14] MVAPICH. http://mvapich.cse.ohio-state.edu/.

[15] NAS Parallel Benchmarks, version 2.4: http://www.nas.nasa.gov/Resources/Software/npb.html

[16] SPEC MPI 2007 Benchmark Suite: http://www.spec.org/mpi/.

[17] T.S. Woodall, G.M. Shipman, G. Bosilca, R.L. Graham, and A.B. Maccabe, "High performance RDMA protocols in HPC", In *Proceedings of Euro PVM/MPI, 2006*, pp. 76-85.

[18] P. Wyckoff and J. Wu, "Memory registration caching correctness", In *Proceedings of CCGrid '05, 2005,* pp. 1008 - 1015.