Cluster Comput (2011) 14:345–356
DOI 10.1007/s10586-011-0165-8

Exploiting application buffer reuse to improve MPI small message transfer protocols over RDMA-enabled networks

Mohammad J. Rashti · Ahmad Afsahi

Received: 4 August 2010 / Accepted: 4 May 2011 / Published online: 3 June 2011

Abstract To avoid the memory registration cost for small messages in MPI implementations over RDMA-enabled networks, message transfer protocols involve a copy to intermediate buffers at both sender and receiver. In this paper, we propose to eliminate the send-side copy when an application buffer is reused frequently. We show that it is more efficient to register the application buffer and use it for data transfer. The idea is examined for small message transfer protocols in MVAPICH2, including RDMA Write and Send/Receive based communications, one-sided communications and collectives. The proposed protocol adaptively falls back to the current protocol when the application does not frequently use its buffers. The performance results over InfiniBand indicate up to 14% improvement for single message latency, close to 20% improvement for one-sided operations and up to 25% improvement for collectives. In addition, the communication time in MPI applications with high buffer reuse is improved using this technique.

Keywords MPI · Buffer reuse · RDMA · Eager protocol · Buffer copy · Memory registration

1 Introduction

Message Passing Interface (MPI) [1] is the de-facto communication library in high-performance computing clusters. Most commercial and open source MPI implementations utilize the latest advancements in interconnection technology to improve communication performance. *Remote Direct Memory Access* (RDMA) [2] is one of the important features of modern interconnects that substantially improves the communication performance in clusters. MPI implementations over contemporary networks such as InfiniBand (IB) [3] take advantage of benefits that RDMA brings to inter-node communication, such as operating system bypass, lower CPU utilization, and zero-copy data transfer.

RDMA-based communication requires the source and destination buffers to be registered to avoid swapping memory buffers before the DMA engine can access them [4]. Memory registration is an expensive process that involves buffer pin-down and virtual-physical address translation [4, 5]. In addition, the registration tag needs to be advertised to the remote node. These costs urge the MPI developers to treat small and large messages differently, in order to avoid extra communication overheads. For small user buffers, where the cost of buffer registration is prohibitive, the user data are copied from application buffers into pre-registered and pre-advertised intermediate buffers. These buffers are internal to MPI implementation and are used for direct memory access at both send and receive sides. On the other hand, for large buffers, where the overhead of memory copy exceeds memory registration cost, the actual user buffers are registered for RDMA transfer.

In MPICH2-based implementations [6], an *Eager* protocol is used for point-to-point communication of small messages to avoid extra overhead of pre-negotiation. The MPI Eager protocol does not use an acknowledgment at the MPI level to notify the completion of the data transfer. The MPI send operation finalizes the communication as soon as the data is reliably transferred to the other side by the IB network. The receiver polls on the MPI intermediate receive buffers for the message arrival.

M.J. Rashti · A. Afsahi (✉)
Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada K7L 3N6
e-mail: ahmad.afsahi@queensu.ca

M.J. Rashti
e-mail: mohammad.rashti@queensu.ca

For large messages, a *Rendezvous* protocol [7] is used in which a negotiation phase makes the receiver ready to receive the message data from the sender directly into the receiver application buffer. After the data transfer, a finalization message is sent by the sender to inform the receiver that the data is placed in its appropriate application buffer.

Similar mechanisms are used for MPI-2 one-sided operations (MPI_Get and MPI_Put), where a synchronization operation is required to finalize the communication. In both Put and Get operations, small messages are copied into intermediate pre-registered buffers, while the large application buffers are registered and used directly for RDMA transfer. For MPI_Put, once a message is transferred using RDMA Write, a flag is set at the remote node (using RDMA Write) to indicate that the data transfer is complete. In MPI_Get, the data is transferred using RDMA Read into a local buffer, and then a *Finalize* packet is sent to finalize the communication. In an active target communication scenario [1], an explicit synchronization is performed among processes using RDMA Write operations, after all one-sided operations are posted.

In the current MPICH/MVAPICH implementations, the aforementioned protocols for small messages are optimized based on a single usage of application buffers. In contrast, the reality is that most of the user buffers in MPI applications are frequently used during the course of execution. This feature, the buffer reuse in MPI applications, has been the main motivation behind this work.

To reduce the communication latency for small messages, 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. Obviously, infrequently-used buffers are treated as before, until they are marked as frequently-used. This way, the cost of communication is decreased by skipping a data copy. The registered user buffer can be kept in MPI registration cache and 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 copies for two-sided communications are still required, because we are carrying out an Eager transfer, which assumes no negotiation with the receiver process. For one-sided operations though, only one data copy exists which is avoided by the proposed method, a data copy at the sender-side for MPI_Put, and a data copy at the receiver-side for MPI_Get.

This paper contributes by extending the proposed protocol in [8] for Eager communication on RDMA fast-path (using InfiniBand [3] verbs layer RDMA Write operation) to two other communication paths in MPI implementation: Eager protocol on verbs layer Send/Receive operations (non-fast-path), and small message one-sided operations (both MPI Get and Put).

The rest of this paper is organized as follows. In Sect. 2, we describe the motivation behind this work. Details about our proposed method for small message transfer will follow in Sect. 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 presented in Sect. 4 and Sect. 5, respectively. We discuss the related work in Sect. 6, and conclude the paper in Sect. 7.

2 Motivation

Despite architectural and technological advances in memory subsystems of modern computing systems, buffer copy is still a major source of overhead in inter-process communication. As explained earlier, the current MPI protocols for short message transfer use memory copies from the application buffer into intermediate buffers (or vice-versa), to avoid the higher cost of memory pinning required for direct memory access by the network interface card. If the user buffer is used frequently in an application, it may be beneficial to register the user buffer and avoid the memory copy. To find out whether this method is helpful, we first need to investigate the buffer reuse pattern of MPI applications at runtime.

Table 1 shows the Eager buffer reuse statistics for some MPI applications described in Sect. 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. The last column shows the buffer reuse percentage among all Eager-size buffer accesses for each application, averaged over 16 processes. 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	Range of buffer reuse counts	Range of most frequently used buffer sizes (bytes)	Buffer reuse percentage
NPB CG Class C	7904–7904 (Avg: 7904)	8–8 (Avg: 8)	42.94%
NPB LU Class C	3749–3750 (Avg: 3749)	1560–1640 (Avg: 1600)	98.71%
ASC AMG2006	45–292 (Avg: 119)	8–3648 (Avg: 770)	76.86%
SPEC MPI2007 104.MILC	2–5010 (Avg: 2049)	8–4800 (Avg: 2876)	44.26%

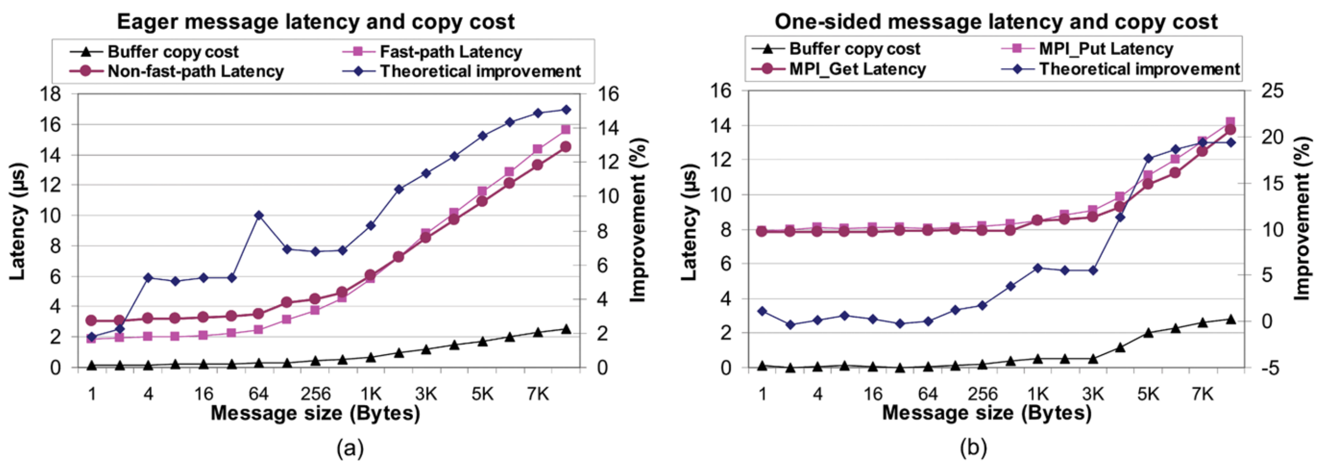

Fig. 1 Small message latency, buffer copy cost and theoretically expected improvement: (**a**) two-sided operations (Eager), (**b**) one-sided operations

Now that it appears that there is a potential to revise the small message transfer protocols for the frequently-used buffers, we would like to know how much improvement we can theoretically achieve if we remove one data copy operation. Figure 1 compares the ping-pong message latency against the cost of one buffer copy for small-size messages on our InfiniBand cluster using MVAPICH2 (refer to Sect. 4 for the description of our experimental platform). Figure 1(a) shows the results for two-sided communications while Fig. 1(b) considers the case for one-sided operations.

Performance results suggest that up to 20% improvement can be achieved by bypassing one buffer copy for small 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 9 KB on our platform. That is why the experimental results in this paper are shown for up to 8 KB 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.

For one-sided operations, this switching happens at 4 KB on our platform. However, experiments show that we can get a better performance by increasing this switching point. For consistency, we will report the performance results for one-sided operations up to the 8 KB messages.

3 Frequent-buffer communication

In this section, we describe the details of the proposed small-message communication mechanism, both for Eager protocol and one-sided operations. Section 3.1 describes the general behavior of the proposed Eager method. We show the extension of this method for one-sided operations in Sect. 3.2. Section 3.3 presents the details about detecting frequently-used buffers at runtime. Finally, Sect. 3.4 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 Small message communication mechanism

Figure 2 depicts the general idea behind bypassing the user buffer copy through application buffer registration. Figure 2(a) illustrates the two-sided (Eager) protocol. The current general path for Eager messages is shown in dashed arrows, while the new path for frequently-used buffers is in solid arrows. RDMA represents both RDMA Write (memory semantics—one-sided) and Send (channel semantics—two-sided) operations. 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 (as shown in Fig. 2(a), for two-sided protocols) 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 [9]) that is still copied into the intermediate buffers. We use InfiniBand RDMA scatter/gather mechanism in which, the header information from a pre-registered buffer and the user data from the application buffer are gathered by the NIC in a single message using an RDMA gather list. The message is then transferred together into the receiver's pre-registered intermediate buffer. Therefore, the

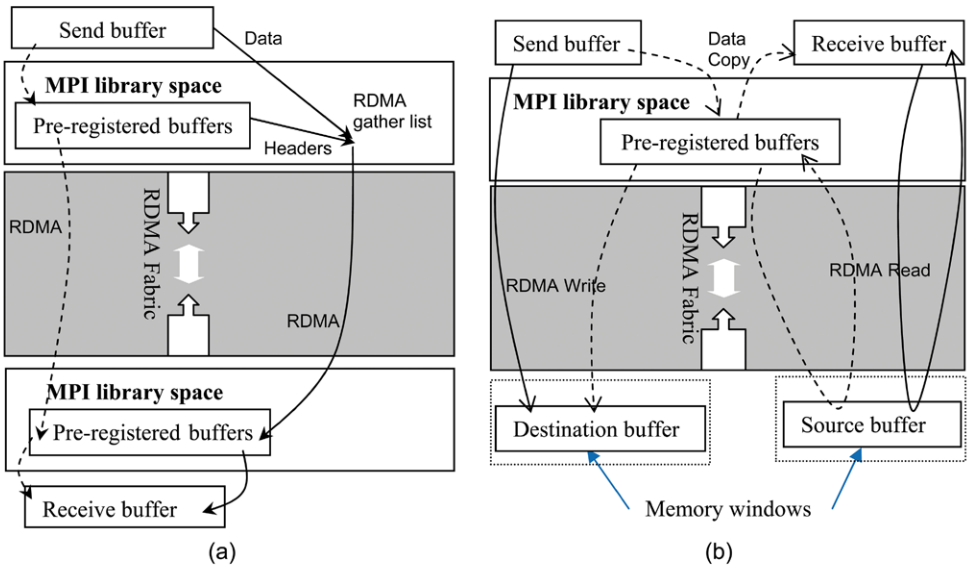

Fig. 2 Data flow diagram for the current and proposed protocols: (**a**) two-sided (Eager) protocol, (**b**) one-sided protocols

cost of copying a buffer containing both the user data and the header is decreased to the cost of only copying the header which is included in the calculation of the theoretical improvement, as shown in (6) in Sect. 5.1.1.

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 [10]. In this mechanism, a 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 [11].

3.2 Improving one-sided operations

Figure 2(b) shows the mechanism used for MPI one-sided operations. MPI_Put is shown on the left hand side and MPI_Get is on the right hand side of Fig. 2(b). Dashed lines show the current protocol, while the solid lines represent the new protocol in which a memory copy is bypassed. Similar to the Eager protocol for point-to-point communications, if an application buffer (send buffer or receive buffer) is used frequently it will be registered for the data transfer in order to reduce the communication cost by avoiding the data copy into a pre-registered buffer. In both protocols, after initiating RDMA (Write or Read) operations, a remote flag is set using RDMA Write to inform the remote party that the operation is performed.

3.3 Detection of frequently-used buffers

To be able to detect frequently-used buffers, we need to keep track of the buffer usage statistics. When a small buffer is accessed for communication, 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, then it is a candidate for buffer registration. At this step, 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 and placed in the cache for future reference.

Now the question is that 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 Fig. 2.

3.3.1 Two-sided communication

The current Eager communication path involves a copy to/from the pre-registered buffer at sender/receiver plus an RDMA Write or Send/Receive based 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 the new communication times, T_c and T_n, when we reuse the

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

Fast-path	Non-fast-path	One-sided
Message size (bytes): Minimum n	Message size (bytes): Minimum n	Message size (bytes): Minimum n
512 B: 432	512 B: 1568	512 B: 196
1 KB: 200	1 KB: 448	1 KB: 128
2 KB: 110	2 KB: 224	2 KB: 128
3 KB: 76	3 KB: 131	3 KB: 125
4 KB: 60	4 KB: 95	4 KB: 56
5 KB: 48	5 KB: 75	5 KB: 33
6 KB: 40	6 KB: 65	6 KB: 29
7 KB: 36	7 KB: 55	7 KB: 26
8 KB: 32	8 KB: 49	8 KB: 25

buffer n times, will be defined as:

$$T_c = n \times (2 \times C_m + NT_m), \tag{1}$$

$$T_n = n \times (C_m + NT_m + V) + R_m \tag{2}$$

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

$$n > \frac{R_m}{C_m - V} \tag{3}$$

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 Sect. 3.3.3), 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 4 KB buffer is registered when it is reused 15 times, hoping that it will be reused 60 times or more so that the registration cost is amortized.

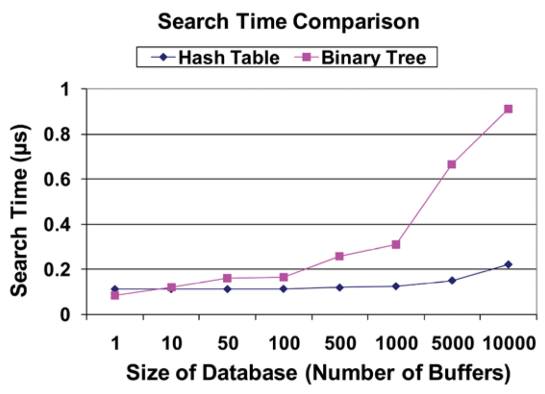

Fig. 3 Comparing search time between hash table and binary tree

3.3.2 *One-sided communication*

MPI one-sided communication follows a different path. For MPI_Put, a memory copy is followed by an RDMA Write to the remote buffer exposed by MPI window. Thus, the communication cost for the current and new methods, when a buffer is reused n times, can be formulated as in (4) and (5), respectively, in which NW_m is the latency of Network Write (RDMA Write) operation:

$$T_c = n \times (C_m + NW_m) \tag{4}$$

$$T_n = n \times (NW_m + V) + R_m \tag{5}$$

For $T_n < T_c$, n follows the same inequality (3). A similar analysis can be done for MPI_Get, in which the RDMA Read cost (NR_m) is used instead of NW_m in (4) and (5), leading to the same lower bound for n. Table 2 shows the minimum value of n for different message sizes for one-sided communication on our platform.

3.3.3 *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 [10]. Our hash table has 1 M 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 [12] that is known to be efficient for hashing buffer addresses.

As shown in Fig. 3, with the growth of the database (the number of small 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.

3.4 Adaptation

The idea of registering small 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 small messages, the overhead of manipulating buffer usage statistics can affect the overall performance of the communication protocol, especially for small messages with latencies comparable to the overhead.

One approach 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 requires extra provisioning especially at the compiler stage for application profiling, and in addition, it does not work for applications with dynamic profiles. The approach we have used in this work is based on the 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 mechanism 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 need to be added to the same hash bucket linked list, causing the linked-list to grow. Obviously, if no conflict has occurred in the hash table search, 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 on four Dell PowerEdge 2850 servers, each with two dual-core 2.8 GHz Intel Xeon EM64T processors (2 MB of L2 cache per core) and 4 GB of DDR-2 SDRAM. Each node has a Mellanox ConnectX DDR InfiniBand HCA [13] 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) 1.4 [15], 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 [16], AMG2006 from ASC Sequoia suite [17], and 104.MILC from SPEC-MPI 2007 suite [18].

CG [16] solves an unstructured sparse linear system using the conjugate gradient method. CG mostly uses MPI send/receive and barrier operations [19]. LU [16] 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 [8, 19].

AMG2006 [17] 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 [18] 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 the performance results of the proposed small message transfer protocols using point-to-point, one-sided and collective communication micro-benchmarks as well as MPI applications.

5.1 Micro-benchmark results

5.1.1 Two-sided communication

Two micro-benchmarks are used to evaluate the potential of the proposed method in improving MPI two-sided communications (Eager protocol):

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 message communication range, from 128 bytes to 8 KB. We keep send and receive buffers separated to avoid affecting each other in terms of memory access. 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. 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. The maximum improvement is around 14% for 8 KB messages, close to the 15% theoretical improvement discussed in Sect. 2.

Note that we have calculated the theoretical improvement by reducing the current communication latency by the cost

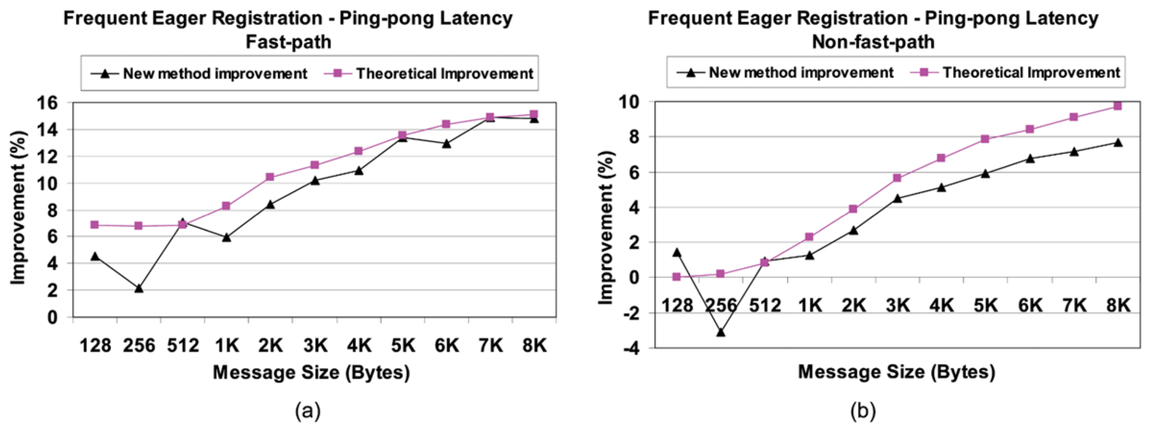

Fig. 4 Message latency improvement: (**a**) Fast-path, (**b**) Non-fast-path

of one buffer copy minus the implementation overhead for the new method, as shown in (6):

$$\begin{aligned}&\mathit{Theoretical\_improvement}\\&\quad = (\mathit{Copy\_overhead} - \mathit{New\_overhead})/\mathit{Old\_latency} \qquad (6)\end{aligned}$$

Obviously, this calculation does not take into account the presumably amortized one-time overhead of buffer registration. This is a factor that can lead to the difference between the theoretical and actual improvements, which sometimes leads to low and even negative improvement (performance loss) in smaller message sizes. This gap is narrowed down as the message size or the number of buffer reuses increases. Therefore, we do not see a performance loss for larger messages.

In calculating the theoretical improvement, we use the buffer copy cost incurred in our actual micro-benchmark test, rather than using a separate micro-benchmark for measuring the copy cost. This is because the buffer copy cost varies in different situations, depending on the alignment of both source and destination addresses, with respect to CPU word size and the memory page size [20, 21]. That is why in our Send/Receive based micro-benchmark (non-fast-path), as shown in Fig. 4(b), the improvement (both theoretical and actual) is lower than the RDMA-based fast-path one. Our investigation has shown that the memory copy cost in the former case is lower.

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: $\{buf_i | 1 < i < 1000\}$, 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 (4 KB 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, even leading to performance loss for smaller messages.

The results for this micro-benchmark using both buffer allocation schemes are shown in Fig. 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 8 KB messages.

5.1.2 One-sided communications

In this section, we present the results for two micro-benchmarks similar to the ones used for two-sided communication, but using one-sided communication primitives (i.e. MPI_Get and MPI_Put operations). In both tests, the application buffers are allocated in separate memory pages to avoid overlap of registration cost.

In the first test, each one-sided communication call (MPI_Get or MPI_Put) is followed by a one-sided active target synchronization (i.e. MPI_Fence). The loop is repeated for 10000 times and the average time is measured. One-sided synchronization in the micro-benchmark is required, because the MPI implementation performs one-sided operations in the subsequent synchronization calls. Figure 6 presents the latency results of this micro-benchmark. The results are compared to the theoretical improvement, calculated using (6). As one can observe, the MPI_Put actual

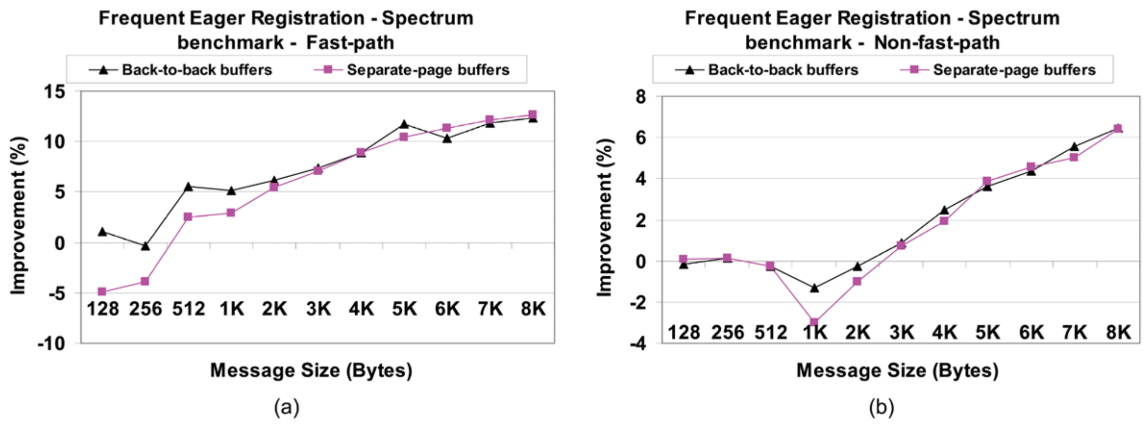

Fig. 5 Improvement as observed in spectrum buffer usage benchmark: (**a**) Fast-path, (**b**) Non-fast-path

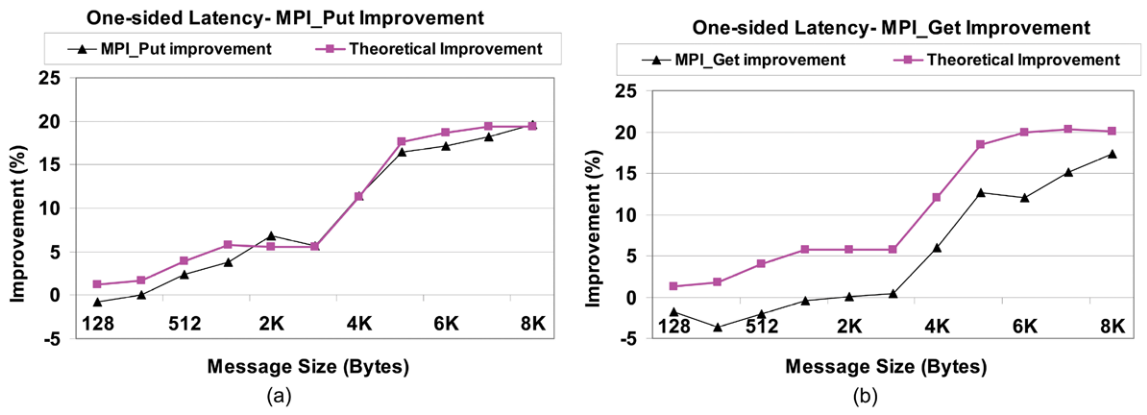

Fig. 6 MPI one-sided latency improvement: (**a**) MPI_Put, (**b**) MPI_Get

improvement results are very close to the theoretical expectation. The MPI_Put latency is improved for messages larger than 256 bytes and reaches around 20% for 8 KB messages.

On the other hand, the MPI_Get improvement is noticeably lower than the expected calculated amount and shows itself only after 3 KB buffer size. Our investigation into the MPI_Get code reveals that since the avoided buffer copy in MPI_Get is performed in the last stage (after receiving data from the source using RDMA Read) in MPI_Fence, part of it is overlapped with the previously issued RDMA Write operations used to finalize the synchronization process. Therefore, the buffer copy overhead on original MPI_Get communication time is less than the actual memory copy cost, used for the estimation of theoretical improvement. Thus, the overhead of our method is not neutralized by saving the buffer copy for smaller messages, which translates in some performance loss for 1 KB or smaller messages.

The second micro-benchmark studied is similar to the spectrum micro-benchmark used for two-sided communication. This test is used to estimate the effect of the proposed method on benchmarks with a spectrum of buffer reuse patterns. One-sided operations are called in the same way as the first micro-benchmark. The improvement results are presented in Fig. 7. Obviously, the improvement is lower than the previous micro-benchmark, because not all buffers are being reused 100% of the time.

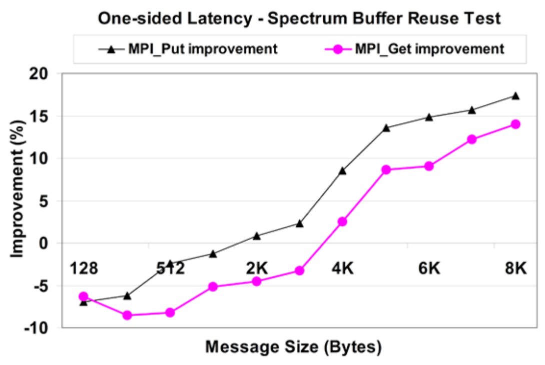

Fig. 7 MPI one-sided improvement as observed in spectrum buffer usage benchmark

5.1.3 Collective communications

Most of the MPI collective operations use MPI point-to-point primitives (i.e. MPI_Send, MPI_Recv and MPI_SendRecv) to implement the collective algorithm. Therefore, improving the performance of point-to-point communications can indirectly affect the collective performance 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 (MPI_Barrier). 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 8 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 the improved communication path, resulting in a lower improvement percentage compared to the 4-process cases. In addition, the scatter operation uses many intermediate buffers that are not re-used.

The curves for the scatter operation sharply decrease for messages larger than 4 KB. 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 (9 KB) for messages larger than 4.5 KB. The other factor is related to the use of temporary buffers in intermediate transfers. Thus, the effect of our algorithm is reduced for scattering messages larger than 4.5 KB.

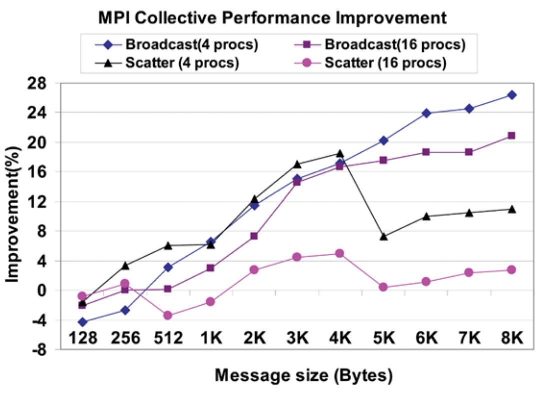

Fig. 8 Improvement of MPI collective operations

5.2 MPI Applications results

In this section, we evaluate the effect of the proposed protocols 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 9 shows the improvements achieved using the proposed method. Table 3 shows the frequently-used buffer percentage for the applications on our platform. 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 7% and 18.3% 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:

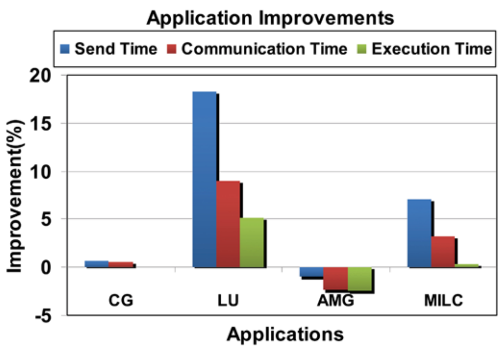

Fig. 9 Application improvement results

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

MPI application	Most frequently-reused buffer percentage
NPB CG Class C	0%
NPB LU Class C	32.9%
ASC AMG2006	8.7%
SPEC MPI2007 104.MILC	22.6%

- 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 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, we believe that process skew is the major cause for this in LU application, since its processes only synchronize once at the beginning.

CG and AMG2006 are applications with lower buffer reuse statistics. Our method has a slight effect on their performance (less than 1% for CG and around 2% for AMG). 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.

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) that 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 AMG.

6 Related work

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

The authors in [22] 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. This method is for earlier versions of RDMA-enabled interconnects in which the registration cost was comparable to the cost of data transfer. The contribution of this work is toward eliminating the need for a pin-cache in MPI, in order to avoid associated memory management complications.

In [9], 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. [7] 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 work has also looked at the cost of memory registration in RDMA-enabled networks, especially its high costs for small buffers [11, 23]. In a recent work presented in [24], researchers have proposed a pinning model in Open-MX based on the decoupling of memory pinning from the application, as a step toward a reliable pinning cache in the kernel. Their proposal enables full overlap of memory pinning with communication. The pinning of each page can be overlapped with the communication of the previous memory page. This method works in Open-MX, which does not bypass the kernel. Such a method cannot be used in systems that are exclusively using user-level libraries and OS bypass.

In another work, the authors in [25] present an improved memory registration cache that performs memory region registration instead of individual buffer registrations. This method also uses batch de-registrations to reduce the overhead. They also propose a method to overlap data transfer with remote node registration. The work in [25] uses simulation to evaluate the proposed mechanisms for RDMA-based web applications.

In [26] the authors propose a number of memory management techniques to improve memory registration issues in file servers. They pipeline subsequent file transfers over the network, in order to overlap data copies with data transfers. In another approach they try to make the OS kernel ready for using Linux *sendfile* mechanism for RDMA connections. This mechanism helps to avoid copying files into user space.

7 Conclusions and future work

In this paper, we have proposed a novel technique to improve the MPI small message transfer protocols over RDMA-enabled networks. In the proposed method that addresses both two-sided and one-sided communications in MPI implementation, we avoid one buffer copy and use the application buffer to transfer the data directly. Our technique is suited for applications with high buffer reuse statistics. However, the employed 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, using RDMA Write operation. This is very close to the maximum theoretical improvement of about 15% on our platform. The improvement for Eager protocol using Send/Receive operations is less, but still close to its theoretical maximum.

Micro-benchmark results for MPI one-sided operations also show significant improvement (close to 20% for MPI_Put and ∼17% for MPI_ Get operations). We have also shown that collective communications, such as broadcast, can achieve even higher, over 25%, 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, 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 would like to have the opportunity to test the proposed method on a larger test-bed with more nodes involved in the communication.

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), and Queen's University. We thank Mellanox Technologies Inc. for the resources.

References

1. Message Passing Interface Forum: MPI, A Message Passing Interface Standard V2.2 (2011)
2. RDMA Consortium: Remote direct memory access protocol. http://www.rdmaconsortium.org (2009). Accessed 1 August 2010
3. InfiniBand Trade Association: InfiniBand architecture specification. http://www.infinibandta.org/ (2010). Accessed 19 July 2010
4. Mietke, F., Rex, R., Baumgartl, R., Mehlan, T., Hoefler, T., Rehm, W.: Analysis of the memory registration process in the mellanox InfiniBand software stack. In: Proceedings of the 12th International Euro-Par Conference, Dresden, Germany, pp. 124–133 (2006). doi:10.1007/11823285_13
5. Magoutis, K.: Memory management support for multi-programmed remote direct memory access (RDMA) systems. In: Proceedings of the 2nd Workshop for RDMA Applications, Implementations and Technologies (RAIT-2005); held in conjunction with IEEE Cluster 2005, Burlington, MA, October (2005). doi:10.1109/CLUSTR.2005.347031
6. Argonne National Laboratory: MPICH2 MPI Implementation. http://www-unix.mcs.anl.gov/mpi/mpich2/ (2010). Accessed 26 July 2010
7. Liu, J., Wu, J., Panda, D.K.: High performance RDMA-based MPI implementation over InfiniBand. In: Proceedings of the 17th Annual Conference on Supercomputing, pp. 295–304 (2003). doi:10.1145/782814.782855
8. Rashti, M.J., Afsahi, A.: Improving RDMA-based MPI Eager protocol for frequently-used buffers. In: 9th Workshop on Communication Architecture for Clusters (CAC 2009). Proceedings of the 23rd International Parallel and Distributed Processing Symposium (IPDPS 2009), Rome, Italy, May 25–29 (2009). doi:10.1109/IPDPS.2009.5160895
9. Huang, W., Santhanaraman, G., Jin, H., Panda, D.K.: Design alternatives and performance trade-offs for implementing MPI-2 over InfiniBand. In: Proceedings of the Euro PVM/MPI Conference, pp. 191–199 (2005). doi:10.1007/11557265_27
10. Mietke, F., Rex, R., Mehlan, T., Hoefler, T., Rehm, W.: Reducing the impact of memory registration in InfiniBand. In: 1st Kommunikation in Clusterrechnern und Clusterverbundsystemen (KiCC) (2005)
11. Wyckoff, P., Wu, J.: Memory registration caching correctness. In: Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05), pp. 1008–1015 (2005). doi:10.1109/CCGRID.2005.1558671
12. Lever, C.: Linux Kernel Hash Table Behavior: Analysis and Improvements. Technical Report 00-1, Center for Information Technology Integration, University of Michigan (2000)
13. Mellanox Technologies Inc.: http://www.mellanox.com/ (2010). Accessed 27 July 2010
14. The Ohio State University, Network-based computing laboratory: MVAPICH2, MPI-2 over InfiniBand, iWARP and RoCE Project. http://mvapich.cse.ohio-state.edu/ (2010). Accessed 1 July 2010
15. OpenFabric Alliance: OpenFabrics Enterprise Distribution (OFED). http://www.openfabrics.org (2010). Accessed 1 July 2010
16. National Aeronautics and Space Administration: NAS Parallel Benchmarks, version 2.4. http://www.nas.nasa.gov/Resources/Software/npb.html (2010). Accessed 1 August 2010
17. Lawrence Livermore National Laboratory: AMG 2006, ASC Sequoia Benchmarks. http://asc.llnl.gov/sequoia/benchmarks/ (2009). Accessed 1 August 2010
18. Standard Performance Evaluation Corporation: SPEC MPI 2007 Benchmark Suite. http://www.spec.org/mpi/ (2010). Accessed 1 August 2010
19. Faraj, A., Yuan, X.: Communication characteristics in the NAS parallel benchmarks. In: Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS), pp. 724–729 (2002)
20. Arber, L., Pakin, S.: The impact of message-buffer alignment on communication performance. Parallel Process. Lett. **15**, 49–65 (2005). doi:10.1142/S0129626405002052
21. Morrow, M.: Optimizing Memcpy improves speed. Embedded system design, April 2004. http://www.eetimes.com/design/other/4024961/Optimizing-Memcpy-improves-speed (2004). Accessed 24 July 2010
22. Woodall, T.S., Shipman, G.M., Bosilca, G., Graham, R.L., Maccabe, A.B.: High performance RDMA protocols in HPC. In: Proceedings of the Euro PVM/MPI Conference, pp. 76–85 (2006). doi:10.1007/11846802_18
23. Dalessandro, D., Wyckoff, P., Montry, G.: Initial performance evaluation of the NetEffect 10 gigabit iWARP adapter. In: Proceedings of the 3rd IEEE Workshop on Remote Direct Memory Access (RDMA): Applications, Implementations, and Technologies (RAIT 2006), held in conjunction with IEEE Cluster, pp. 1–7 (2006). doi:10.1109/CLUSTR.2006.311915
24. Goglin, B.: Decoupling Memory Pinning from the Application with Overlapped On-demand Pinning and MMU Notifiers. In: Proceedings of the IEEE International Symposium on Parallel & Distributed Processing (IPDPS2009), pp. 1–8 (2009). doi:10.1109/IPDPS.2009.5160888
25. Ou, L., He, X., Han, J.: An efficient design for fast memory registration in RDMA. J. Netw. Comput. Appl. **32**, 642–651 (2009). doi:10.1145/363095.363141
26. Dalessandro, D., Wyckoff, P.: Memory management strategies for data serving with RDMA. In: Proceedings of the 15th Annual IEEE Symposium on High-Performance Interconnects (HotI), August 22–24 (2007). doi:10.1109/HOTI.2007.21

Mohammad J. Rashti is a post-doctoral fellow in the Department of Electrical and Computer Engineering at Queen's University in Kingston, ON, Canada, under the supervision of Dr. Ahmad Afsahi. He received his Ph.D. degree in 2010 from Queen's University, his M.Sc. in 2003 from Sharif University of Technology and his B.Sc. in 2000 from University of Tehran, all in Computer Engineering. His research interests include high speed networking, high performance parallel and distributed computing, with special focus on inter-process communication.

Ahmad Afsahi is currently an Associate Professor in the Department of Electrical and Computer Engineering at Queen's University in Kingston, ON, Canada. He received his Ph.D. degree in Electrical Engineering from the University of Victoria in British Columbia, in 2000, and his M.Sc. and B.Sc. degrees in Computer Engineering from Sharif University of Technology and Shiraz University, in Iran, respectively. His research activities are in the areas of parallel and distributed processing, network-based high-performance computing, high-speed networks, and power-aware high-performance computing. His research has been mainly focused at improving the performance and scalability of communication subsystems and messaging layers for high-performance computing and data center clusters, as well as their power awareness. His research has earned him a Canada Foundation for Innovation Award as well as an Ontario Innovation Trust Award. Dr. Afsahi is a Senior Member of IEEE, and a licensed Professional Engineer in the province of Ontario.