

High Performance RDMA Protocols in HPC

Tim S. Woodall¹, Galen M. Shipman¹, George Bosilca², Arthur B. Maccabe³

Los Alamos National Laboratory, Advanced Computing Laboratory
{twoodall, gshipman}@lanl.gov

University of Tennessee, Dept. of Computer Science
bosilca@cs.utk.edu

University of New Mexico, Dept. of Computer Science
maccabe@cs.unm.edu

Abstract. Modern network interconnects that leverage Remote Directory Memory Access (RDMA) and OS bypass, such as Infiniband [2], Myrinet [9], and iWARP over TCP [3], can offer significant performance advantages over conventional send/receive network semantics. However, the high performance of RDMA often comes with hidden costs. RDMA based interconnects generally fail to provide true one-sided semantics, requiring an exchange of information prior to initiating a one-sided RDMA operation. In addition, both the initiator and target must typically preserve the physical to virtual memory mappings during the RDMA operation.

This paper describes a unique user-level ‘pipeline’ protocol that addresses these constraints while avoiding some of the pitfalls of existing techniques. By effectively overlapping the cost of memory registration with RDMA operations this protocol provides good performance even in the absence of memory buffer reuse. This protocol may also take advantage of memory buffers that have already been used in RDMA operations by avoiding the cost of memory registration. Through this approach, bandwidth may be increased up to 67% when memory buffers are not effectively reused while providing performance equal to that of existing techniques as demonstrated by both Linpack and NPB benchmark results. Several user level protocols are explored using Open MPI’s PML (Point to point messaging layer) and compared/contrasted to this ‘pipeline’ protocol.

1 Introduction

RDMA (Remote Direct Memory Access) capable interconnects are widely used in high performance computing (HPC) systems. While these interconnects provide for high bandwidth and low latency messaging, they also pose unique challenges to HPC software designers. The Message Passing Interface (MPI) standard [8], one of the most widely used HPC messaging paradigms, generally abstracts these issues from the parallel programmer. However, implementations of this standard and other messaging middleware must address these challenges to achieve balanced performance across a wide variety of application communication patterns.

One such challenge is the requirement by the majority of RDMA interconnects that memory used in RDMA operations be explicitly registered with the interconnect and pinned by the OS. Memory registration is often an expensive operation requiring a trap to the OS and an additional linear cost that is a function of the number of pages in the memory region. Various techniques exist to minimize the impact of memory registration but each make specific assumptions about application behavior or system usage. Typical approaches involve caching registrations for later reuse. However, many applications do not effectively reuse communication buffers. Additionally, the application may invalidate the cached buffer, which places constraints on the return of cached pages to the OS.

Another issue is that RDMA operations are described entirely by the initiator of the operation. This requires that both source and target buffers be made known to the initiator prior to the RDMA operation. The peer must send their local address and any memory registration information required by the interconnect to the initiator thereby incurring another roundtrip in the RDMA operation.

In this paper we describe a high performance user level RDMA protocol which minimizes the impact of memory registration while avoiding the pitfalls of other techniques. This protocol provides performance at or exceeding other methods while minimizing resource usage. In addition, this protocol makes no assumption about application behavior thereby providing improved performance for certain applications.

This protocol was implemented and evaluated in the context of Open MPI [5]. The Open MPI project draws upon prior work from LA-MPI [6], LAM/MPI [11], FT-MPI[4] and PAX-MPI [7]. Open MPI is however, a completely new MPI implementation, designed from the ground up to address the demands of current and next generation architectures and interconnects.

The remainder of this paper is organized as follows. Section 2 provides an overview of Infiniband, the RDMA interconnect used in our research. Next, section 3 discusses different approaches to minimize the impact of memory registration, while section 4 discusses our user-level pipeline protocol. Open MPI's support for multiple techniques is discussed in Section 4. Results are discussed in Section 6, followed by conclusions and future work in Section 7.

2 Infiniband

Infiniband, similar to Myrinet GM and iWARP, provides both Remote Direct Memory Access (RDMA) and Operating System (OS) bypass facilities. RDMA enables data transfer from the address space of an application process to a peer process across the network fabric without requiring involvement of the host CPU. Infiniband RDMA operations support both two-sided send/receive and one-sided put/get semantics. Each of these operations may be queued from the user level directly to the host channel adapter (HCA) for execution, bypassing the OS to minimize latency and processing requirements on the host CPU.

Infiniband does place some constraints on these operations. As data is moved directly between the host channel adapter (HCA) and user level source/destination buffers, these buffers must be registered with the HCA in advance of their use. Registration is a relatively expensive operation which locks the memory pages associated with the request, thereby preserving the virtual to physical mappings. Additionally, when supporting send/receive semantics, pre-posted receive buffers are consumed in order as data arrives on the host channel adapter (HCA). Since no attempt is made to match available buffers to the incoming message size, the maximum size of a message is constrained to the minimum size of the posted receive buffers.

Infiniband shares many characteristics with other common RDMA interconnects including Myrinet and emerging standards such as iWARP. The common requirement for explicit memory registration, local knowledge of peer registrations prior to initiating an RDMA operation, and the associated issues with effectively managing these registrations motivated the work described in the following sections.

3 RDMA

To overcome the expense of registering memory with the interconnect, application and library developers have used several techniques. A simple solution is to restrict all RDMA operations to a static memory region. This allows the application to register the memory region once and amortize this cost over a potentially large number of RDMA operations. While this does help in hiding the costs of the memory registration, it restricts the application to a static memory region. For many applications this usage model is inappropriate and forces the user to copy in/out of the registered memory. For larger messages copy costs quickly become a bottleneck.

Another approach is to register memory on demand. The target and source buffers are registered prior to the RDMA operation and then deregistered upon completion of the operation. This approach allows the user to RDMA from any memory region providing a true zero copy transfer. Unfortunately the benefits of zero copy are mitigated by the high cost of registering the memory prior to each RDMA operation.

A third approach avoids the high cost of copying in/out of a static memory region and in some use cases allows the cost of registering the memory region to be amortized over multiple RDMA operations. Prior to the RDMA operation the memory region is registered and the registration is then cached locally. Subsequent RDMA operations first query the cache for a matching registration and if found uses this registration to immediately initiate the RDMA operation. For applications which regularly reuse source and target buffers for RDMA operations the cost of the initial registration is effectively amortized over these subsequent RDMA operations. This approach was first available in MPICH-GM [1].

A drawback to this approach is that some applications may not effectively reuse source and target buffers incurring a high cost for each RDMA operation. Additionally, cached buffers may be frequently invalidated by the application. Message buffers allocated by the application, registered and cached by the MPI layer, and later returned to the OS by the application must be removed from the cache at the MPI layer. Reuse of these cached registrations would result in corrupted data transfers due to changes in the page table mappings on subsequent allocations. Current approaches to addressing this issue involve either non-portable memory hooks and/or linker tricks to intercept `brk`, `munmap`, and/or `free` to insure that returned pages are removed from the cache, or simply disabling the return of pages to the OS by the allocator.

This paper describes an alternate approach which allows RDMA operations from arbitrary memory regions while maintaining high performance. This approach makes no assumption about the applications reuse of source and target buffers. Instead of registering the entire source and target buffer prior to an RDMA operation, memory is dynamically registered in smaller pieces and RDMA operations are overlapped with memory registration/deregistration. Additionally, send/rcv operations are employed to eagerly send data to cover the cost of initializing the pipeline. This pipeline protocol is described in further detail in the following section.

4 RDMA Pipeline Protocol

The pipeline protocol begins by eagerly sending the first part of the message data, up to a configurable eager limit, along with a MATCH header using send/receive semantics, as illustrated in Figure 1,

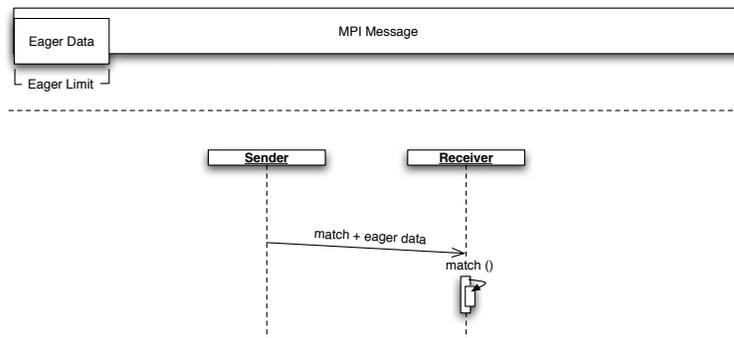


Fig. 1. Sending Eager Data with Match Header

From figure 2 upon receipt and match of the header to a posted receive, the receiver responds with an ACK to the source and begins registering blocks (RDMA

fragments) of the target buffer across the available RDMA capable HCAs. The number of blocks registered at any given time is bound by a maximum pipeline depth. The size of each block is constrained by the maximum configured RDMA size for each interconnect. As each registration in the pipeline completes, an RDMA target fragment READY control message is sent to the source to initiate a registration of the source RDMA fragment followed by an RDMA write on the block.

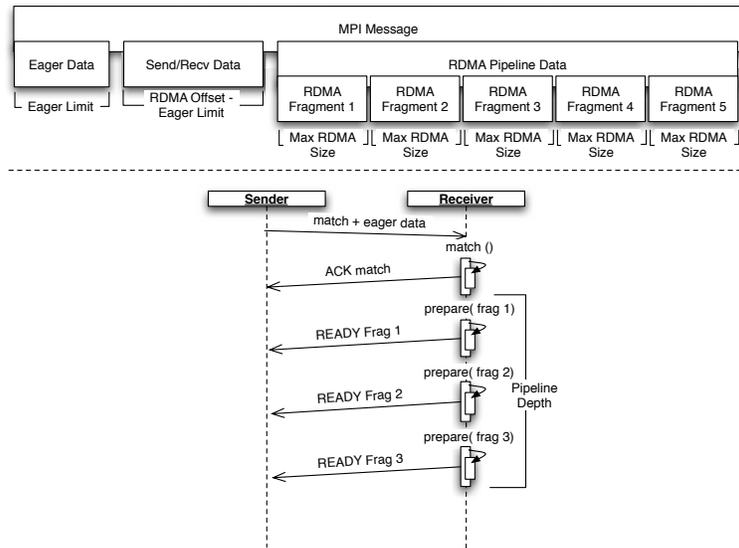


Fig. 2. Receiver Registers RDMA Target Fragments

To cover the cost of initializing the pipeline, on receipt of the initial ACK at the source, send/receive semantics are used to deliver data from the eager limit up to the initial RDMA write offset returned from the peer in the message ACK, as illustrated in Figure 3. If the rdma offset is larger than the maximum configured send size, the data is again fragmented and delivered across the available interconnects.

From Figure 4 we see that as RDMA READY control messages are received at the source, the corresponding block of the source buffer is registered and an RDMA write operation is initiated on the current block. On local completion at the source, an RDMA FIN message is sent to the peer. Registered blocks are deregistered (released) upon local completion or receipt of the RDMA FIN message. If required, the receipt of an RDMA FIN messages may also further advance the RDMA pipeline.

This protocol effectively overlaps the cost of registration/deregistration with RDMA writes. Resources are released immediately and the high overhead of a

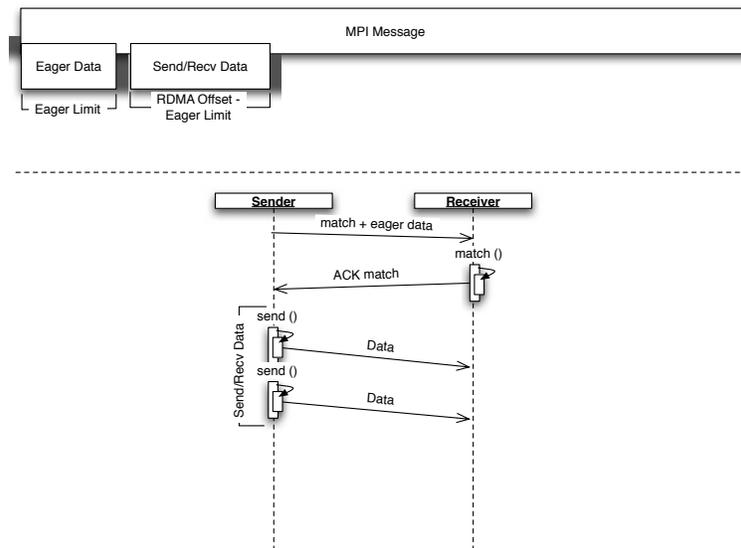


Fig. 3. Sender Sends Data up to the RDMA Offset to Cover Pipeline Initialization Costs

single large memory registration is avoided. Additionally, this protocol results in improved performance for applications which may not reuse buffers for MPI operations effectively.

5 Other RDMA protocols

In addition to the RDMA pipeline protocol, which is enabled by default, Open MPI supports additional techniques for managing RDMA registrations. These approaches have been developed to contrast the relative merits of each, and allow the behaviour to be tuned at run-time to best match the application characteristics.

5.1 RDMA Cache

Open MPI provides the capability to optionally register memory on first use and cache these registrations for later re-use. When this approach is used, the entire source/destination buffer is registered and a single RDMA operation is initiated on receipt of an ack from the peer. If multiple network interfaces are available, the message is divided across the available interfaces, and a single operation is initiated on each interface.

While other MPI implementations prevent physical pages from being released to the OS, Open MPI provides the capability to use memory hooks to

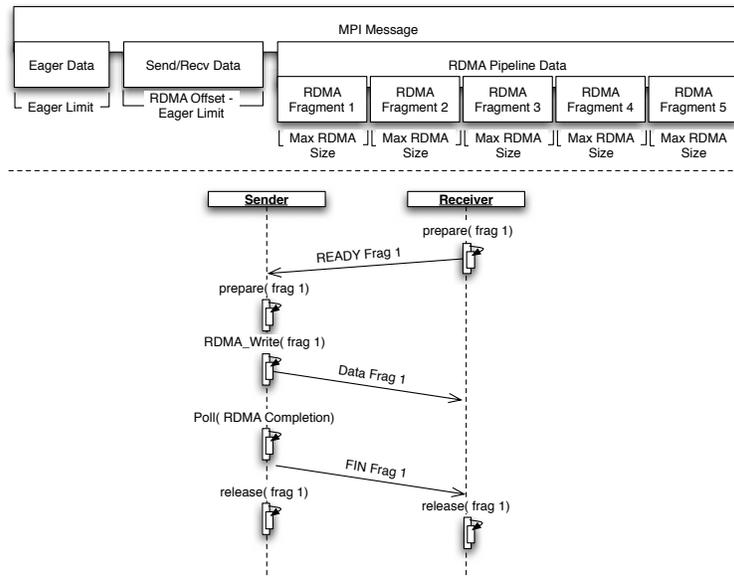


Fig. 4. Sender Prepares and RDMA Writes a Fragment

intercept the deallocation of memory and its return to the OS. When pages are returned to the OS via sbrk/munmap, the pages are checked and any matching entries de-registered. This prevents future use of an invalid memory registration while allowing memory to be returned to the host operating system. Intercepting memory deallocation introduces additional overhead and additional research into reducing this overhead is ongoing.

5.2 RDMA Caching Pipeline

A hybrid approach was developed to explore the benefit of caching individual registrations within the RDMA pipeline protocol. In this approach, the pipeline protocol described in section 4 is modified to cache each registration as it occurs in the pipeline. Subsequent pipeline RDMA operations from the same or overlapping buffer space then re-use the cached registrations. Registrations are aligned to the segment size to further promote reuse.

This hybrid approach leverages the pipeline protocol for good performance in the case of low buffer reuse, and achieves performance closer to the RDMA cache when existing registrations can be re-used. The drawback to the introduction of the cache is the added requirement for memory hooks to intercept the deallocation of memory as described above.

6 Results

This section presents a comparison of the different protocols in Open MPI. We first demonstrate the performance of the pipeline protocol as a function of buffer reuse in terms of bandwidth. Next, we examine effective bandwidth among multiple peers using different communication patterns via the Effective Bandwidth Benchmark [10]. In general, our results provide comparisons of the pipeline protocol to:

- copy in/out - Standard send/recv semantics with copy in/out of pre-registered buffers
- leave pinned (memory hooks) - Registration cache described in section 5.1 with memory hooks to intercept memory deallocations.
- leave pinned (disable sbrk) - Registration cache described in section 5.1 with return of pages to OS disabled.
- pipeline - Pipeline protocol described in section 4
- pipeline leave pinned (memory hooks) - Caching pipeline described in section 5.2 with memory hooks to intercept memory deallocations
- pipeline leave pinned (disable sbrk) - Caching pipeline described in section 5.2 with return of pages to OS disabled.

6.1 Bandwidth

The following graphs illustrate the performance of the pipeline protocol as a function of buffer reuse. As Figure-5 illustrates, with no buffer reuse, the standard pipeline protocol achieves a speedup of up to 67 percent over the registration cache. This can be attributed to the pipeline protocol effectively overlapping the cost of registration with RDMA. In contrast, with no buffer reuse, the caching protocol is limited by the high cost of registration.

An interesting metric is the amount of reuse required for the caching protocol to achieve performance comparable to the pipeline. Figure-6 illustrates the bandwidth achieved for each protocol as a function of the number of times the buffer is reused, for an arbitrary fixed message size (8 Mbytes). As the graph illustrates, the buffer must be reused on the order of 40-50 times before the caching protocol achieves performance equivalent to the standard pipeline. Note that the hybrid caching pipeline ramps up much earlier, as the registrations are overlapped with RDMA and re-used on subsequent invocations of the pipeline. This approach improves bandwidth over the standard pipeline as we defer the cost of deregistration until memory is released by the application.

6.2 Effective Bandwidth Benchmark b_{eff}

The Effective Bandwidth Benchmark (b_{eff}) was used to examine protocol effect on bandwidth in more complex communication patterns. In this benchmark 8 nodes were used to communicate message sizes up to 1 GByte using different communication patterns. In this benchmark the benefits of the pipeline protocol

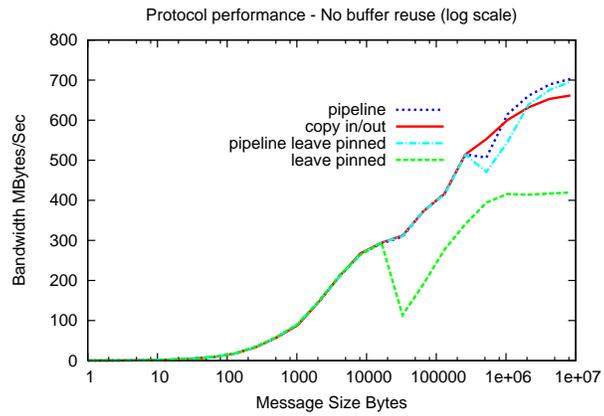


Fig. 5. Ping Pong Bandwidth - No Buffer Reuse - Log Scale

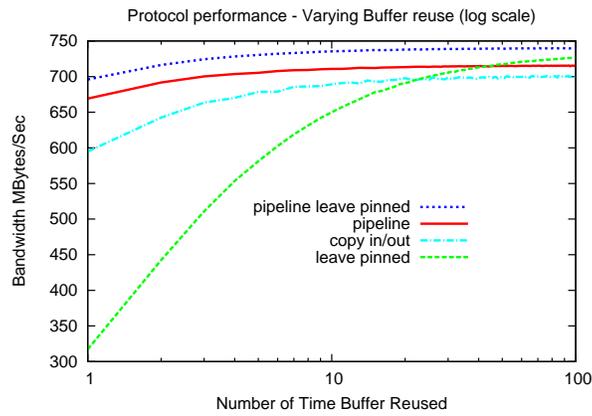


Fig. 6. Ping Pong Bandwidth - Varying Buffer Reuse - Log Scale

	b_{eff}	Lmax	b_{eff} at Lmax	b_{eff} at Lmax	ping-pong bandwidth
	MByte/s		rings& random MByte/s	rings only MByte/s	MByte/s
accumulated					
- pipeline leave pinned	1803	8 MB	5409	5426	743
- pipeline	1509	8 MB	4714	4679	710
- leave pinned	1375	8 MB	3912	3934	604
- copy in/out	1369	8 MB	3131	3129	682
per process					
- pipeline leave pinned	225	676	678		
- pipeline	189	589	585		
- leave pinned	172	489	492		
- copy in/out	171	391	391		

are apparent. When the memory cache is used (leave pinned) in conjunction with the pipeline protocol the total bandwidth achieved outperforms all the other protocols by a significant margin.

6.3 Experimental Setup

Our experiments were performed on a 256 node cluster consisting of dual Intel Xeon X86-64 3.4 GHz processors with a minimum 6GB of RAM, Mellanox PCI-Express Lion Cub adapters connected via a Voltair switch. The Operating System is Linux 2.6.9-11 with Open MPI 1.1 pre-release.

7 Conclusions - Future Work

In this section we summarize the results of this work and provide directions for future work.

7.1 Conclusions

RDMA capable interconnects pose unique challenges that require careful consideration to achieve balanced performance and scalability across a wide range of application communication patterns. The results of this work indicate the RDMA pipeline protocol effectively addresses these concerns, by overlapping the dynamic registration and deregistration of memory buffers with data transfer. This approach avoids the issues associated with maintaining a registration cache, which requires non-portable memory hooks to either intercept deallocations, or disable the return of pages to the OS. Additionally, the pipeline protocol reduces the memory footprint and resource requirements of the application over the caching approach.

7.2 Future work

The hybrid pipeline protocol which cached registrations as they occurred in the pipeline provided promising results over the dynamic pipeline. However, the caching approach is still constrained by the above issues. Additional work to address these issues would involve effectively managing cache size, investigating the potential for efficient notification from the OS on changes to registered pages, and improving the performance of cache cleanup/deregistration.

Acknowledgments

This material is based upon work supported by Subcontract No. 12783-001-05 49 issued to Rice University from the Regents of the University of California (Los Alamos National Laboratory). Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36.

Project support was provided through ASC/PSE and ASC/S&CS programs. LA-UR-06-1268.

References

1. Performance of mpich-gm.
2. I. T. Association. Infiniband architecture specification vol 1, release 1.2, 2004.
3. M. Chadalapaka, H. Shah, U. Elzur, P. Thaler, and M. Ko. A study of iscsi extensions for rdma (iser). In *NICELI '03: Proceedings of the ACM SIGCOMM workshop on Network-I/O convergence*, pages 209–219, New York, NY, USA, 2003. ACM Press.
4. G. E. Fagg, A. Bukovsky, and J. J. Dongarra. HARNES and fault tolerant MPI. *Parallel Computing*, 27:1479–1496, 2001.
5. E. Garbriel, G. Fagg, G. Bosilica, T. Angskun, J. J. D. J. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. Castain, D. Daniel, R. Graham, and T. Woodall. Open MPI: goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, 2004.
6. R. L. Graham, S.-E. Choi, D. J. Daniel, N. N. Desai, R. G. Minnich, C. E. Rasmussen, L. D. Risinger, and M. W. Sukalksi. A network-failure-tolerant message-passing system for terascale clusters. *International Journal of Parallel Programming*, 31(4), August 2003.
7. R. Keller, E. Gabriel, B. Krammer, M. S. Mueller, and M. M. Resch. Towards efficient execution of MPI applications on the grid: porting and optimization issues. *Journal of Grid Computing*, 1:133–149, 2003.
8. Message Passing Interface Forum. MPI: A Message Passing Interface. In *Proc. of Supercomputing '93*, pages 878–883. IEEE Computer Society Press, November 1993.
9. Myricom. Myrinet-on-VME protocol specification.
10. R. Rabenseifner and A. Koniges. The parallel communication and i/o bandwidth benchmarks: b_eff and b_eff_io. 2001.
11. J. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In *Proceedings, 10th European PVM/MPI Users' Group Meeting*, number 2840 in Lecture Notes in Computer Science, Venice, Italy, September / October 2003. Springer-Verlag.