Investigations on InfiniBand: Efficient Network Buffer Utilization at Scale

Galen M. Shipman[1], Ron Brightwell[2], Brian Barrett[1], Jeffrey M. Squyres[3], and Gil Bloch[4]

[1] Los Alamos National Laboratory [⋆], Los Alamos, NM USA,
{gshipman,bbarrett}@lanl.gov,
LA-UR-07-3198,
[2] Sandia National Laboratories [⋆⋆], Albuquerque, NM USA,
rbbrigh@sandia.gov
[3] Cisco, Inc., San Jose, CA USA,
jsquyres@cisco.com
[4] Mellanox Technologies, Tel Aviv, Israel,
gil@mellanox.il

Abstract. The default messaging model for the OpenFabrics "Verbs" API is to consume receive buffers in order—regardless of the actual incoming message size—leading to inefficient registered memory usage. For example, many small messages can consume large amounts of registered memory. This paper introduces a new transport protocol in Open MPI implemented using the existing OpenFabrics Verbs API that exhibits efficient registered memory utilization. Several real-world applications were run at scale with the new protocol; results show that global network resource utilization efficiency increases, allowing increased scalability—and larger problem sizes—on clusters which can increase application performance in some cases.

1 Introduction

The recent emergence of near-commodity clusters with thousands of nodes connected with InfiniBand (IB) has increased the need for examining scalability issues with MPI implementations for IB. Several of these issues were originally discussed in detail for the predecessor to IB [1], and several possible approaches to overcoming some of the more obvious scalability limitations were proposed. This study examines the scalability, performance, and complexity issues of the message buffering for implementations of MPI over IB.

The semantics of IB Verbs place a number of constraints on receive buffers. Receive buffers are consumed in FIFO order, and the buffer at the head of

[⋆] Los Alamos National Laboratory is operated by Los Alamos National Security, LLC, for the National Nuclear Security Administration of the U.S. Department of Energy under contract DE-AC52-06NA25396.

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

the queue must be large enough to hold the next incoming message. If there is no receive buffer at the head of the queue or if the receive buffer is not large enough, this will trigger a network-level protocol that can significantly degrade communication performance. Because of these constraints, MPI implementations must be careful to insure that a sufficient number of receive buffers of sufficient size are always available to match incoming messages.

Several other details can complicate the issue. Most operating systems require message buffers to be registered so that they may be mapped to physical pages. Since this operation is time consuming, it is desirable to register memory only when absolutely needed and to use registered memory as efficiently as possible. MPI implementations that are single-threaded may only be able to replenish message buffers when the application makes an MPI library call. Finally, the use of receive buffers is highly dependent on application message passing patterns.

This paper describes a new protocol that is more efficient at using receive buffers and can potentially avoid some of the flow control protocols that are needed to insure that receive buffers of the appropriate size are always available. We begin by measuring the efficiency of receive buffer utilization for the current Open MPI implementation for IB. We then propose a new strategy that can significantly increase the efficiency of receive buffer utilization, make more efficient use of registered memory, and potentially reduce the need for MPI-level flow control messages.

The rest of this paper is organized as follows. The next section provides a brief discussion of previous work in this area. Section 3 describes the new protocol, while Section 4 provides details of the test platform and analyzes results for several application benchmarks and applications. We conclude in Section 5 with a summary of relevant results and offer some possible avenues of future work in Section 5.1.

2 Background

The complexity of managing multiple sets of buffers across multiple connections was discussed in [1], and a mechanism for sharing a single pool of buffers across multiple connections was proposed. In the absence of such a shared buffer pool for IB, MPI implementations were left to develop user-level flow control strategies to insure that message buffer space was not exhausted [2].

Eventually, a shared buffer pool mechanism was implemented for IB in the form of a shared receive queue (`SRQ`), and has been shown to be effective in reducing memory usage [3, 4]. However, the IB `SRQ` still does not eliminate the need for user-level and network-level flow control that is required to insure that the shared buffer pool is not depleted [5]. The shared buffer pool approach was also implemented for MPI on the Portals data movement layer, but using fixed sized buffers was shown to have poor memory efficiency in practice, so an alternative strategy was developed [6]. In Section 3, we propose a similar strategy that can be used to improve memory efficiency for MPI over IB as well.

3 Protocol Description

IB does not natively support receive buffer pools similar to [6], but it is possible to emulate the behavior with buckets of receive buffers of different sizes, with each bucket using a single shared receive queue (SRQ). We call this emulation the "Bucket SRQ," or B-SRQ.

B-SRQ begins by allocating a set of per-peer receive buffers. These per-peer buffers are for "tiny" messages (128 bytes plus space for header information) and are regulated by a simple flow control protocol to ensure that tiny messages always have a receive buffer available.[5] The "tiny" size of 128 bytes was choosen as an optimization to ensure that global operations on a single `MPI_DOUBLE` element would always fall into the per-peer buffer path. The 128-byte packet size also ensures that other control messages (such as rendezvous protocol acknowledgments) use the per-peer allocated resources path as well.

B-SRQ then allocates a large "slab" of registered memory for receive buffers. The slab is divided up into N buckets; bucket B_i is S_i bytes long and contains a set of individual receive buffers, each of size R_i bytes (where $R_i \neq R_j$ for $i, j \in [0, N-1]$ and $i \neq j$). Bucket B_n contains $\frac{S_n}{R_n}$ buffers. Each bucket is associated with a unique queue pair (QP) and a unique SRQ. This design point is a result of current IB network adapter limitations; only a single receive queue can be associated with a QP. The QP effectively acts as the sender-side addressing mechanism of the corresponding SRQ bucket on the receiver.[6] Other networks such as Myrinet GM [7] allow the sender to specify a particular receive buffer on the send side through a single logical connection and as such would allow for a similar protocol as that used in B-SRQ. Quadrics Elan [8] uses multiple pools of buffers to handle unexpected messages, although the specific details of this protocol have never been published.

In our prototype implementation, $S_i = S_j$ for $i \neq j$, and $R_i = 2^{8+i}$, for $i \in [0, 7]$. That is, the slab was divided equally between eight buckets, and individual receive buffers were powers of two sizes ranging from 2^8 to 2^{15}. Fig. 1 illustrates the receive buffer allocation strategy.

Send buffers are allocated using a similar technique, except that free lists are used which grow on demand (up to a configurable limit). When a MPI message is scheduled for delivery, a send buffer is dequeued from the free list; the smallest available send buffer that is large enough to hold the message is returned.

4 Results

Two protocols are analyzed and compared: Open MPI v1.2's default protocol for short messages and Open MPI v1.2's default protocol modified to use B-SRQ for short messages (denoted "v1.2bsrq").

[5] The flow control protocol, while outside the scope of this paper, ensures that the receiver never issues a "receiver not ready" (or RNR-NAK) error, which can degrade application performance.

[6] Open MPI uses different protocols for long messages. Therefore, the maximum B-SRQ bucket size is effectively the largest "short" message that Open MPI will handle.

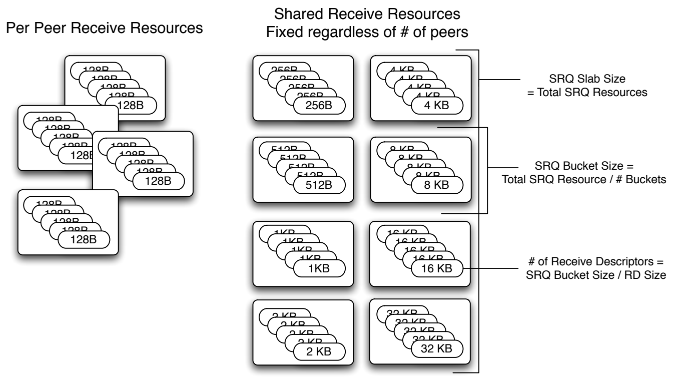
Per Peer Receive Resources
Shared Receive Resources
Fixed regardless of # of peers
128B
256B
4 KB
512B
8 KB
1KB
16 KB
2 KB
32 KB
SRQ Slab Size
= Total SRQ Resources
SRQ Bucket Size =
Total SRQ Resource / # Buckets
# of Receive Descriptors =
SRQ Bucket Size / RD Size

Fig. 1. Open MPI's B-SRQ receive resources.

The default protocol for short messages in Open MPI v1.2 uses a credit-based flow-control algorithm to send messages to fixed-sized buffers on the receiver. When the sender exhausts its credits, messages are queued until the receiver returns credits (via an ACK message) to the sender. The sender is then free to resume sending. By default, SRQ is not used in Open MPI v1.2 because of a slight latency performance penalty; SRQ may be more efficient in terms of resource utilization, but it can be slightly slower than normal receive queues in some network adapters [3, 4].

Open MPI v1.2bsrq implements the protocol described in Section 3. It has the following advantages over Open MPI v1.2's default protocol:

– Receiver buffer utilization is more efficient.
– More receive buffers can be be posted in the same amount of memory.
– No flow control protocol overhead for messages using the SRQ QPs.[7]

4.1 Experimental Setup

The Coyote cluster at Los Alamos National Laboratory was used to test the B-SRQ protocol. Coyote is a 1,290 node AMD Opteron cluster divided into 258-node sub-clusters. Each sub-cluster is an "island" of IB connectivity; nodes are fully connected via IB within the sub-cluster but are physically disjoint from IB in other sub-clusters. Each node has two single-core 2.6 GHz AMD Opteron processors, 8 GB of RAM, and a single-port Mellanox Sinai/Infinihost III SDR IB adapter (firmware revision 1.0.800). The largest MPI job that can be run utilizing the IB network is therefore 516 processors.

Both versions of Open MPI (v1.2 and v1.2bsrq) were used to run the NAS Parallel Benchmarks (NPBs) and two Los Alamos-developed applications. Wall-clock execution time was measured to compare overall application performance

[7] Without pre-negotiating a fixed number of SRQ receive buffers for each peer, there is no way for senders to know when receive buffers have been exhausted in an SRQ [3].

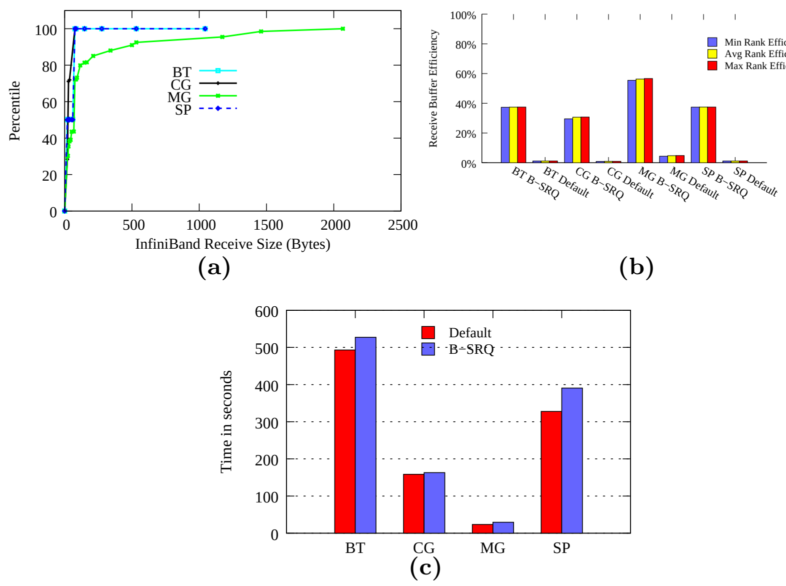

Fig. 2. NAS Parallel Benchmark Class D results on 256 nodes for (a) message size, (b) buffer efficiency, and (c) wall-clock execution time.

with and without B-SRQ. Instrumentation was added to both Open MPI versions to measure the effectiveness and efficiency of B-SRQ by logging send and receive buffer utilization, defined as $\frac{message\_size}{buffer\_size}$. Performance results used the non-instrumented versions.

Buffer utilization of 100% is ideal, meaning that the buffer is exactly the same size as the data that it contains. Since IB is a resource-constrained network, the buffer utilization of an application can have a direct impact on its overall performance. For example, if receive buffer utilization is poor and the incoming message rate is high, available receive buffers can be exhausted, resulting in an RNR-NAK (and therefore performance degradation) [3, 4].

The frequency of message sizes received via IB was also recorded. Note that IB-level message sizes may be different than MPI-level message sizes as Open MPI may segment an MPI-level message, use RDMA for messages, and send control messages over IB. This data is presented in percentile graphs, showing the percentage of receives at or below a given size (in bytes).

4.2 NAS Parallel Benchmarks

Results using D sized problems with the NPBs are shown in Figure 2(a). The benchmarks utilize a high percentage of small messages at the IB level, with the notable exception of MG, which uses some medium-sized messages at the IB level. Some of these benchmarks do send larger messages as well, triggering a different message passing protocol in Open MPI that utilizes both rendezvous

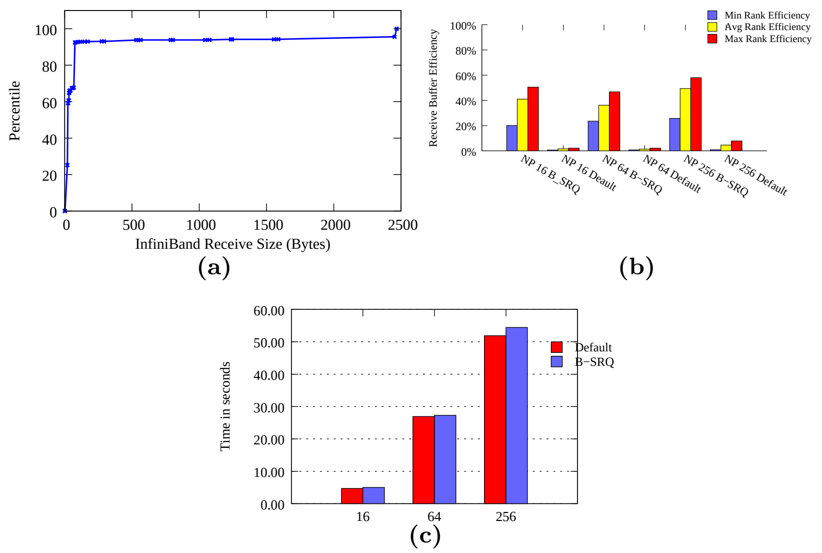

Fig. 3. SAGE results for (a) message size at 256 processes, (b) buffer efficiency at varying process count, and (c) wall-clock execution time at varying process count.

techniques and RDMA [9]. Long messages effectively avoid the use of dedicated receive buffers delivering data directly into the application's receive buffer.

Figure 2(b) shows that Open MPI's v1.2 protocol exhibits poor receive buffer utilization due to the fact that receive buffers are fixed at 4 KB and 32 KB. These buffer sizes provide good performance but poor buffer utilization. The B-SRQ protocol in v1.2bsrq provides good receive buffer utilization for the NPB benchmarks, with increases in efficiency of over 50% for the MG benchmark. Overall B-SRQ performance decreases slightly, shown in Figure 2(c); this performance penalty is currently being investigated in conjunction with several IB vendors.

4.3 Los Alamos Applications

Two Los Alamos National Laboratory applications were used to evaluate B-SRQ. The first, SAGE, is an adaptive grid Eulerian hydrocode that uses adaptive mesh refinement. SAGE is typically run on thousands of processors and has weak scaling characteristics. Message sizes vary, typically in the tens to hundreds of kilobytes. Figure 3(a) shows that most IB level messages received were less than 128 bytes with larger messages using our RDMA protocol. Figure 3(b) illustrates poor memory utilization in Open MPI v1.2 when run at 256 processors. The new protocol exhibits much better receive buffer utilization, as smaller buffers are used to handle the data received. As with the NPBs, there is a slight performance impact, but it is very minimal at 256 processors, as illustrated in Figure 3(c).

The second application evaluated was SWEEP. SWEEP uses a pipelined wavefront communication pattern. Messages are small and communication in

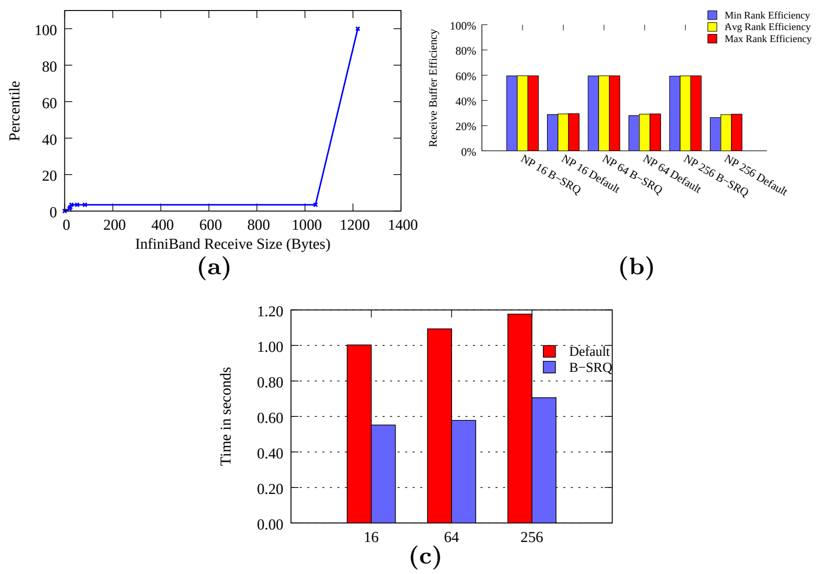

Fig. 4. SWEEP results for (a) message size at 256 processes, (b) buffer efficiency at varying process count, and (c) wall-clock execution time at varying process count.

the wavefront is limited to spatial neighbors on a cube. The wavefront direction and starting point is not fixed; it can change during the run to start from any corner on the cube. Receive buffer size is highly regular as shown in Figure 4(a). Figure 4(b) shows that receive buffer efficiency is improved by the B-SRQ algorithm, although to a lesser extent than in other applications. Figure 4(c) shows that performance was dramatically increased by the use of the B-SRQ protocol. This performance increase is because the v1.2 protocol incorporates a flow control mechanism which limits the number of sends that can be outstanding (controlled by the number of buffers available on the receiver). Receive resources are allocated per peer, so a relatively small number of receive buffers are available (defaulting to 8). SWEEP is extremely latency sensitive; this flow control not only adds overhead, it prevents SWEEP's high message rate from keeping the IB network full. Over 90% of SWEEP's messages are in the message sizes covered by the B-SRQ protocol; these messages were therefore not rate limited by a flow control mechanism, resulting in a significant performance increase.

5 Conclusions

Evaluation of application communication over Open MPI revealed that the size of data received at the IB level, while often quite small, can vary substantially from application to application. Using a single pool of fixed-sized receive buffers may therefore result in inefficient receive buffer utilization. Receive buffer depletion negatively impacts performance, particularly at large scale. Earlier work focused on complex receiver-side depletion detection and recovery, or avoiding depletion altogether by tuning receive buffer sizes and counts on a per-application (and sometimes per-run) basis.

The B-SRQ protocol focuses on avoiding resource depletion by more efficient receive buffer utilization. Experimental results are promising; by better matching receive buffer sizes with the size of received data, application runs were constrained to an overall receive buffer memory footprint of less than 25 MB. Additionally, receive queues were never depleted and no application-specific tuning of receive buffer resources was required.

5.1 Future Work

While the B-SRQ protocol results in better receive buffer utilization, other receive buffer allocation methods are possible. For example, buffer sizes in increasing powers of two may not be optimal. Through measurements collected via Open MPI v1.2bsrq, we will explore receive buffer allocation policies and their potential impact on overall receive buffer utilization.

Recent announcements by IB vendors indicate that network adapters will soon support the ability to associate a single QP with multiple SRQs by specifying the SRQ for delivery on the send side. This may have implications on both performance and QP resource utilization that will need to be studied further.

References

[1] Brightwell, R., Maccabe, A.B.: Scalability limitations of VIA-based technologies in supporting MPI. In: Proceedings of the Fourth MPI Devlopers' and Users' Conference. (2000)

[2] Liu, J., Panda, D.K.: Implementing efficient and scalable flow control schemes in mpi over infiniband. In: Workshop on Communication Architecture for Clusters (CAC 04). (2004)

[3] Shipman, G.M., Woodall, T.S., Graham, R.L., Maccabe, A.B., Bridges, P.G.: Infiniband scalability in Open MPI. In: International Parallel and Distributed Processing Symposium (IPDPS'06). (2006)

[4] Sur, S., Chai, L., Jin, H.W., Panda, D.K.: Shared receive queue based scalable MPI design for InfiniBand clusters. In: International Parallel and Distributed Processing Symposium (IPDPS'06). (2006)

[5] Sur, S., Koop, M.J., Panda, D.K.: High-performance and scalable MPI over InfiniBand with reduced memory usage: An in-depth performance analysis. In: ACM/IEEE International Conference on High-Performance Computing, Networking, Storage, and Analysis (SC'06). (2006)

[6] Brightwell, R., Maccabe, A.B., Riesen, R.: Design, implementation, and performance of MPI on Portals 3.0. International Journal of High Performance Computing Applications **17**(1) (2003)

[7] Myrinet: Myrinet GM. (http://www.myri.com/scs/documentation.html)

[8] Quadrics: Quadrics elan programming manual v5.2. http://www.quadrics.com/ (2005)

[9] Shipman, G.M., Woodall, T.S., Bosilca, G., Graham, R.L., Maccabe, A.B.: High performance RDMA protocols in HPC. In: Proceedings, 13th European PVM/MPI Users' Group Meeting. Lecture Notes in Computer Science, Bonn, Germany, Springer-Verlag (2006)