

# Reducing Connection Memory Requirements of MPI for InfiniBand Clusters: A Message Coalescing Approach \*

Matthew J. Koop<sup>†‡</sup>

Terry Jones<sup>‡</sup>

Dhableswar K. Panda<sup>†</sup>

<sup>†</sup> *Network-Based Computing Laboratory  
Department of Computer Science and Engineering  
The Ohio State University  
{koop, panda}@cse.ohio-state.edu*

<sup>‡</sup> *Lawrence Livermore National Laboratory  
Livermore, CA 94550  
trj@llnl.gov*

## Abstract

*Clusters in the area of high-performance computing have been growing in size at a considerable rate. In these clusters, the dominate programming model is the Message Passing Interface (MPI), so the MPI library has a key role in resource usage and performance. To obtain maximal performance, many clusters deploy a high-speed interconnect between compute nodes. One such interconnect, InfiniBand, has been gaining in popularity due to its various features including Remote Data Memory Access (RDMA), and high-performance. As a result, it is being deployed in a significant number of clusters and has been chosen as the standard interconnect for capacity clusters within the DOE Tri-Labs. As these clusters grow in size, care must be taken to ensure the resource usage does not increase too significantly with scale. In particular, the MPI library resource usage should not grow at a rate which will exhaust the node memory or starve user applications.*

*In this paper we present our findings of current memory usage when all connections are created and design a message coalescing method to decrease memory usage significantly. Our models show that the default configuration of MVAPICH can grow to 1GB per process for 8K processes, while our enhancements reduce usage by an order of magnitude to around 120 MB per process while maintaining near-equal performance. We have validated our design on a 575-node cluster and shown no performance degradation for a variety of applications. We also increase the message rate attainable by over 150%.*

---

\*This work was primarily performed under the auspices of the U.S. Department of Energy by the University of California, Lawrence Livermore National Laboratory. This work was also supported in part by DOE grants #DE-FC02-06ER25749 and #DE-FC02-06ER25755; NSF grants #CNS-0403342 and #CNS-0509452. LLNL Technical Report UCRL-CONF-226304.

## 1 Introduction

High-speed interconnects have become of increasing importance for optimal performance as clusters have continued to increase in size. The InfiniBand [7] interconnect is becoming increasingly popular in cluster computing due to low latency (1.0-3.0 $\mu$ sec) and high bandwidth as well as other features such as Remote Data Memory Access (RDMA), atomic operations, multicast, and quality of service. Many large clusters, such as the 9024-processor Sandia Thunderbird, NASA/Ames Columbia, and the LLNL Peloton clusters are using InfiniBand as their primary interconnect. The trend in HPC is clusters of rapidly increasing size, so issues seen on current generation clusters are likely to be of even greater importance for future machines.

The Message Passing Interface (MPI) [12] is the dominant programming model on these large clusters. Given its involvement in many applications, the MPI library has a key role in providing scalability in performance as well as resource usage. A key area of importance for applications is the amount of memory available per process. Additional memory can often allow an application to run problems not otherwise possible or to achieve increased precision. The MPI library should use as little memory as possible to allow this flexibility for the application.

In this paper, we explore the memory scalability issues for MPI libraries over InfiniBand. In particular, we evaluate the memory usage of MVAPICH [11, 14], a popular MPI implementation over InfiniBand, and quantify the effect of the number of allowed outstanding send operations on memory usage. As part of our study we propose reducing the number of allowed outstanding send operations and a coalescing method to eliminate the resulting performance degradation. While we examine MPI in-depth, this same analysis and design can apply to other applications or protocols that make use of

InfiniBand connections.

Our design reduces memory usage at 8K processes by an order of magnitude and maintains performance equivalent to the existing design. Our results show an increase in small message performance of up to 150% and near identical performance for message sizes above that level. We additionally validate our design with the NAS Parallel Benchmarks, sPPM, SMG2000, and Sweep3D and note that performance remains unchanged despite reducing memory usage significantly.

The rest of the paper is organized as follows. In Section 2 we provide the requisite background information on InfiniBand and MPI. Following in Section 3 we provide an evaluation of the memory usage by the MPI library with different numbers of allowed outstanding sends. Section 4 describes our design of coalescing packets when many small messages are sent within a short time period. Our enhanced design is evaluated in Section 5. We discuss related work in Section 6 and we present our conclusions and future work in Section 7.

## 2 Background

In this section we give the necessary background information. We start with a brief overview of the InfiniBand Architecture. Next, we discuss MVAPICH and finish with an explanation of the two main connection methods used in MPI over InfiniBand.

### 2.1 InfiniBand Architecture Overview

InfiniBand was designed as a high-speed general-purpose I/O interconnect. Host Channel Adapters (HCAs) connect the InfiniBand fabric to the I/O bus. In recent years it has become a popular interconnect for high-performance computing to connect commodity machines in large clusters. It is even the preferred interconnect for DOE Tri-Lab capacity clusters.

There are two sets of communication semantics in InfiniBand, channel semantics and memory semantics. Channel semantics are send and receive operations that are common in traditional interfaces, such as sockets. In channel semantics both sides of the communication must be aware of the communication. Memory semantics are one-sided operations where one host can access memory from a remote node without a posted receive; such operations are referred to as Remote Direct Memory Access (RDMA).

In both sets of semantics all memory used for communication must be registered and pinned, not to be swapped out. The startup cost for registering memory is high. Due to this restriction, small messages are generally copied into pre-registered buffers before being sent to the receiver for optimal performance. At larger sizes, it can become advantageous to directly register the application send and receive buffers and perform a RDMA

operation to directly copy the data, termed zero-copy. This requires a “rendezvous” handshake to exchange memory keys, buffer addresses and availability.

There are four communication modes defined by the InfiniBand specification: Reliable Connection (RC), Reliable Datagram (RD), Unreliable Connection (UC) and Unreliable Datagram (UD). Of these, RC, UC, and UD are required to be supported by HCAs in the InfiniBand specification. RD is not required and is not available in current hardware. The rest of the paper will assume the use of RC since it is the most often used for MPI libraries.

When using a connection-based model, a pair of hosts that wishes to communicate must each set up a dedicated Queue Pair (QP) for communication with that peer. Each QP is linked to a Completion Queue (CQ) for notification of completion. In this connection-based model, *there is additional memory usage with each additional connection.*

To send a message a descriptor is posted to the QP. This descriptor contains information about the message to be sent, including the data address, memory keys, and message length. To receive a message using channel semantics a receive descriptor must be posted containing the address and length of the buffer. Upon posting a descriptor, a send Work Queue Entry (WQE), pronounced “wookie,” is used to track the progress of the request.

Upon completion of a WQE a Completion Queue Entry (CQE), “cookie,” is placed in the CQ. This method is used in both channel and memory semantics. CQEs can be obtained by polling the CQ or through an event-based methods.

When a QP is created, the number of send and receive WQEs must be defined. The number of WQEs allocated determines the number of outstanding send and receive operations allowed on a single QP. Using a Shared Receive Queue (SRQ) allows receive WQEs and buffers to be shared rather than per QP, which allows far better scalability. Benefits are demonstrated in [17] and we will assume SRQ is being used. Even using a SRQ, however, send WQEs must be posted per QP. Thus, the number of send WQEs allocated for a QP determines how many outstanding send operations are allowed for that connection.

### 2.2 MVAPICH

MVAPICH is a MPI library derived from MVICH [9] and MPICH that is optimized for InfiniBand [11, 14]. It is developed at the Ohio State University and is used by over 450 organizations worldwide. We will use MVAPICH to assess performance and MPI memory usage with an increasing number of processes as well as to prototype our design.

**Table 1. Average Number of Connections Per Process Used in NAS Benchmarks (64 processes)**

NAS Benchmarks						
BT	CG	FT	IS	LU	MG	SP
10	6.98	63	63	7.5	9	10

### 2.3 Connection Models in MPI over InfiniBand

There are two main connection management methods used in MPI implementations: static and on-demand [19, 20]. MVAPICH has the ability to use both options.

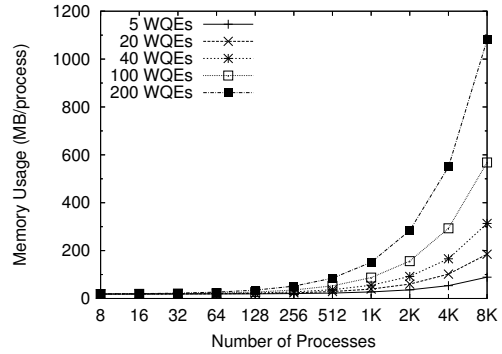
- **Static:** All connections are setup at the beginning of a MPI job.
- **On-Demand:** Setup MPI connections only when required. Each connection and InfiniBand QPs are only created when a message needs to be sent to a process for which a QP is not already created.

There are some trade-offs between these methods. A static scheme requires less logic and allows the user to know the available memory at the beginning of the job instead of memory increasing as more connections are created. A static method can also reduce startup time if all connections are used by the application. An on-demand method, however, can use a significantly lower number of connections if a process does not communicate with more than a small subset of its peers.

It is misleading, however, to show MPI memory usage information when there are no connections setup. Such graphs give the impression that memory resource issues have been solved. Applications such as NAS show only a few connections setup (Table 2.3), however, such applications are only kernels and do not necessarily reflect real application connections. Even here we observe for both FT and IS each process makes direct connections to every peer in the application. In [19, 18], SMG2000 [4] is found to have nearly the same number of connections per process as total peers.

Applications that do load balancing and dynamically redistribute work may increase the number of connections significantly as the application runs. Additionally, many MPI collective routines may setup all connections in order to obtain the best performance, such as Allgather and Alltoall.

For the purpose of this paper, we will assume a fully-connected static model to assess the worst-case memory usage, which is the case for various applications. We make this assumption since the MPI library should not



**Figure 1. Memory Usage (per process) with Varied Numbers of WQEs**

impose restrictions on application behavior to maintain a reasonable memory footprint.

## 3 Work Queue Entries and Memory Usage

In this section we discuss the purpose and memory usage of send Work Queue Entries (WQEs). In particular, we examine memory usage when all connections are made between processes in the cluster.

### 3.1 Memory Usage with Varied Numbers of WQEs

To observe the memory usage of the MPI library, we run a MPI application with the static connection settings of MVAPICH and measure the total memory usage. The average value per process is reported. To better assess only the InfiniBand connection costs, we disable shared memory support. Even when shared memory communication is used the per connection values we report are accurate and the memory usage at large numbers of processes is nearly identical to the case without shared memory support.

Figure 1 shows the memory usage per process with increasing numbers of processes and varied allocations of send WQEs. Results are experimentally obtained through 1024 processes; numbers above that level are modeled. We observe that for 8K processes, using the default allocation of 200 send WQEs requires around 1 GB of memory per process. On a machine with 8 cores per node, such as the Peloton clusters at LLNL, a total of 8 GB per node would be used only for the MPI library with this allocation. With 16GB per node, 50% of available memory is consumed only for connection memory. When the number of send WQEs per connection is decreased to 5, the memory usage drops to less than 90 MB per process at 8K processes.

Table 3.1 shows the additional memory required per connection based on the experimental evaluation up to

**Table 2. Memory usage per additional connection with increasing send WQEs**

5	10	20	40	100	200
8.82 KB	12.82 KB	20.81 KB	36.86 KB	68.73 KB	132.76 KB

1024 processes. From this information we can determine that there are approximately 4KB of connection costs other than send WQEs, but after 5 send WQEs per QP that term becomes dominant.

It should be noted that InfiniBand allows for different sizes of *inline* data. This allows the reduction of latency by  $\sim 200$  nanoseconds by storing the data, generally up to a maximum of  $\sim 200$  bytes (HCA dependent), in the request. Since inline data is stored within each WQE, it increases the size of the data structure. In the version of the InfiniBand drivers used for this paper, the memory usage remained unchanged for different values of inline data. Newer versions of these drivers have reduced memory when smaller inline limits are used in QP creation.

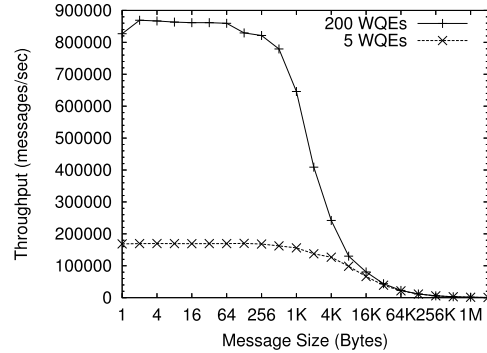
### 3.2 Effects of Reducing WQEs

As noted in the previous subsection, the MPI library memory usage is highly dependent on the memory allocation of the QP and WQE resources. Reducing the number of send WQEs leads to a large decrease in connection memory usage, however, there are performance consequences to lowering the number of available WQEs.

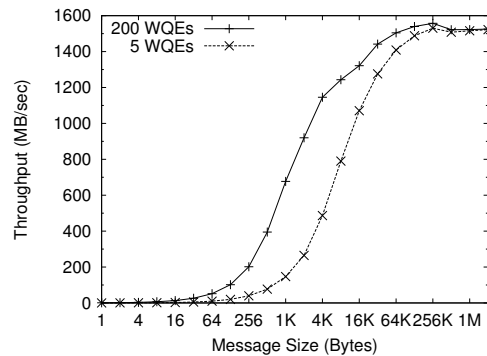
As noted in Section 2.1, the number of send WQEs restrict the number of send operations that can be outstanding on a given QP. In this section we evaluate the issues with lowering the number of send WQEs. The default value of 200 was empirically derived so buffering in the MPI library is extremely rare.

Results from throughput microbenchmarks show that small message throughput is diminished considerably when the number of send WQEs is reduced. Figure 2 shows that performance drops over 70% from nearly 900 thousand to under 200 thousand messages per second. From Figure 3 we observe the overall data throughput is also reduced significantly for small to medium-sized messages.

It is important to note that with these throughput microbenchmarks the sending process sends a window of 64 MPI messages before an acknowledgment is sent by the receiver. This process is repeated many times to get an average value. Thus, the performance degrades since the send WQEs are exhausted before the window is complete and additional sends cannot be pipelined by the HCA. This means that the bandwidth for a single message is the same in the case of both 200 and 5 WQEs, however, if messages are sent in bursts then the perfor-



**Figure 2. Uni-Directional Messaging Rate with Existing Design**



**Figure 3. Uni-Directional Bandwidth with Existing Design**

mance will be degraded when fewer WQEs have been allocated. Regardless, it is not possible to simply decrease the number of WQEs, doing so has performance consequences.

## 4 Message Coalescing Design

In this section we describe our design to alleviate any issues that may arise as a result of reducing the number of send WQEs available to the MPI library.

### 4.1 Motivation

As mentioned in Section 2.1, a send WQE is needed to issue a send request to the HCA. If one is not available, the message must be queued internally in the library until a previous send operation completes. If the number of outstanding sends is to be reduced without sacrificing performance, a method is needed to efficiently send the messages that may be in the queue as quickly as possible.

From Figures 2 and 3 we observe that when the number of WQEs is low, small message performance is degraded significantly. With larger messages, the bandwidth can be saturated with only a few simultaneous sends. Since this is the case, our attention will focus on increasing the performance for messages up to 8K, where the performance gap is the largest.

The key issue is that the network fabric is not being efficiently utilized when only a few WQEs are available. In this case the HCA is not able to pipeline the requests optimally. To make better use of the network fabric we propose a scheme to coalesce the messages that are being queued to increase the network performance since there is a startup cost associated with each send operation. By coalescing the messages we are able to amortize the startup cost of the operation across the number of messages that we are able to pack together.

### 4.2 Design

As discussed in Section 2, memory used in communication must be registered. Since memory registration has a high startup cost, small messages are copied into pre-registered communication buffers before being sent in MVAPICH. The general send flow for a small message send operation therefore is to copy the contents of the user buffer into an internal registered MPI buffer, at which point the library determines if there are available send WQEs. If at least one WQE is available the message is immediately posted to the QP and sent by the HCA, otherwise it is placed on a linked-list queue. When completion of previous send operations are detected through the CQ, messages are sent in order from the queue.

To efficiently coalesce messages, we alter the send flow operation. Before copying the contents of the user

buffer into a registered buffer we check to the availability of send WQEs. If there is availability, the flow continues as in the original case. If not, we first check the queue for any other messages waiting to be sent on that QP. Assuming another message is present we verify it is of a compatible type – we do not combine sends with rendezvous data messages or other special message types. If enough space is available in the communication buffer the message is coalesced and copied into the buffer. If there are no other pending sends for that QP or there is insufficient buffer space available, a new communication buffer is used and the message is copied into it. Note that we are not using any additional memory to coalesce the messages; we are potentially using less since we can use the same buffer for more than one message.

Another design alternative is to use the InfiniBand scatter/gather capabilities instead of packing into the same buffer. This, however, introduces an unnecessary overhead. Since the user buffer is already being copied into pre-registered buffers it is more efficient to coalesce into a buffer initially. In this way we only have one copy on the sender side as well.

To further optimize the throughput of messages we cache the MPI tag matching information for each message. If the tag information of a message matches that of the previous coalesced message, a one byte `COALESCED_CACHED` flag is set and the header information is omitted. Otherwise the entire header information is included in the coalesced message. This caching increases performance by nearly 10% for messages less than 64 bytes and is negligible for messages with payloads above 512 bytes. Thus, even without caching the coalescing design has performance significantly higher than that of the original design with 200 WQEs for these message sizes. In this way we achieve a general solution with no requirement that all coalesced messages need to have the same tag matching information (size, tag, context), while obtaining optimal performance for those with identical tag information.

## 5 Evaluation

Our experimental testbed is a 575-node InfiniBand Linux cluster at Lawrence Livermore National Laboratory. Each compute node has four 2.5 GHz Opteron 8216 dual-core processors for a total of 8 cores. Total memory per node is 16GB. Each node has a Mellanox MT25208 DDR HCA. InfiniBand software support is provided through the OpenFabrics/Gen2 stack [15]. The Intel v9.0 compiler is used for compilation of the MVAPICH library and applications.

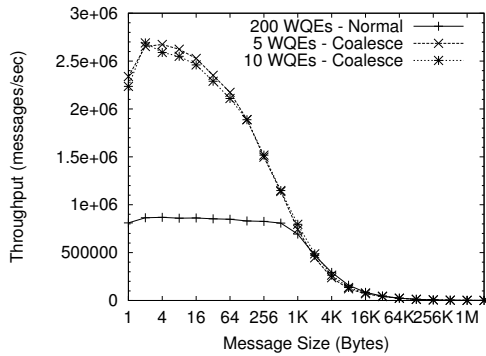


Figure 4. Coalesced Messaging Rate

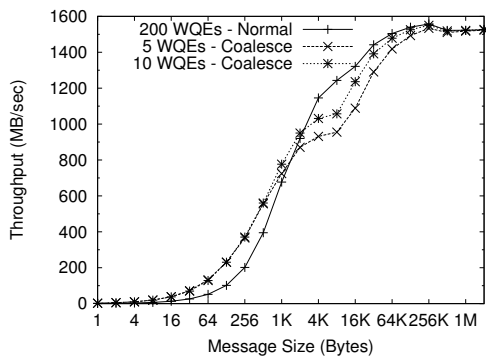


Figure 5. Coalesced Uni-Directional Bandwidth

## 5.1 Microbenchmark Performance

Figure 4 shows that small message throughput for the coalesced design with 5 and 10 send WQEs significantly exceeds that of the existing design for message sizes through 1K. This is an effect of the amortization of startup costs for posting a send in InfiniBand. By packing more messages per send buffer we are able to lower the average overhead for sending a message. In this regard we have more than succeeded in removing the performance degradation that otherwise occurs with such a small number of send WQEs.

This finding is also seen in Figure 5, where we observe improved performance for messages up to 1K and slightly lower performance for messages from 4K to 128K. Performance has been improved significantly for small to medium messages, but the coalescing scheme cannot pack messages that exceed the half of the buffer space. The copy-based send method is used through 9K on this platform. The number of WQEs is important; using only 5 WQEs for the coalescing design cannot match the microbenchmark performance of 200 WQEs at all

message sizes, however, 10 WQEs nearly matches the performance. Adding additional WQEs can close the performance gap.

Even higher messaging rates could likely be achieved with an even smaller number of send WQEs, or artificially restricting the number of outstanding send operations for small messages and then allowing messages greater than 2KB to use additional WQEs. This is outside the scope of this work; we instead wish to find the lowest memory usage possible without harming performance rather than just increasing the message rate.

It is important to note that microbenchmark performance represents the extreme situation for a low number of send WQEs. This performance will only be seen when there are more outstanding sends being used by the application than available send WQEs.

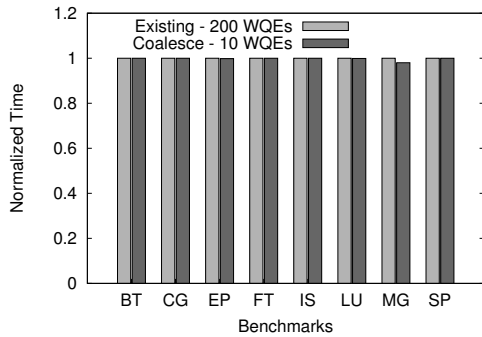
Applications that do not use more than the available send WQEs or do so infrequently will perform equally with the coalescing design or the existing design. Differences in the microbenchmark graphs are not applicable unless the send WQEs are exhausted, as they are here. In this work we present these numbers since they represent the worst-case and even in this case we are able to show near equal performance using only 15% of the memory usage.

## 5.2 Application Performance

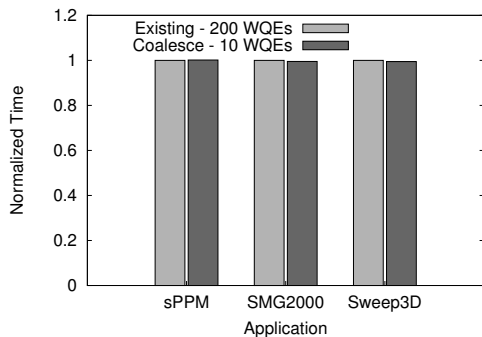
We compare the performance of the existing design with the default number of WQEs with our proposed coalescing design using a low number of existing WQEs and significantly less memory. We evaluate with the NAS Parallel Benchmarks, and three applications from the ASC Benchmark Suite [2]: sPPM, SMG2000, and Sweep3D. Communication pattern analysis of sPPM, SMG200, and Sweep3D is available in [18] by Vetter, et al.

### 5.2.1 Applications

- **NAS Parallel Benchmarks** [3] are kernels designed to be typical of various MPI applications. As such, they are a good tool to evaluate the performance of the MPI library and parallel machines.
- **sPPM** [13] is an application distributed as part of the ASC Purple Benchmarks. It solves a 3D gas dynamics problem on a uniform Cartesian mesh using the Piecewise Parabolic Method (PPM).
- **SMG2000** [4] is a parallel semicoarsening multi-grid solver, which is also a part of the ASC Purple Benchmarks.
- **Sweep3D** [6, 8] uses a multidimensional wavefront algorithm to solve the 3D, time-independent, particle transport equation on an orthogonal mesh.



**Figure 6. NAS Benchmarks Comparison, Class C, 256 Processes**



**Figure 7. sPPM, SMG2000, and Sweep3D Results, 256 Processes**

### 5.2.2 Results

All results are taken with 256 processes, 4 processes per node with shared memory support disabled to stress network performance. All application results are the average of 5 runs, although deviation in performance was low between runs.

Figure 6 shows the performance of the existing and proposed designs with performance normalized to the existing design for all of the NAS Benchmarks, Class C. As expected from the microbenchmark results, there is no observed performance degradation. MG shows some benefits using our design over the original, although additional study is required to determine the cause of this improvement.

The performance comparison for sPPM, SMG2000, and Sweep3D is shown in Figure 7. Again we observe no performance degradation for our enhanced design. All application results are within one percent of each other.

As expected from the microbenchmark results, the performance of the proposed design is no worse in any application or benchmark than the existing design, despite using nearly an order of magnitude less memory.

## 6 Related Work

Memory usage of the MPI library has also been studied by many other researchers. Early versions of MVAPICH exhibited significant memory usage as the number of connections increased as studied by Liu, et al in [10]. Followup work by Sur, et al in [17] significantly reduced the per connection memory usage in MVAPICH using InfiniBand Shared Receive Queue (SRQ) support and a unique flow control method. Similar techniques have been used in Open MPI [5] by Shipman, et al in [16]. In both of these studies, the memory usage was reduced mostly through a reduction in communication memory buffer usage. In this paper we have instead targeted the connection memory usage which remains a significant issue at scale. Adaptive connection management in MVAPICH to setup only those connections that are used was discussed by Yu, et al in [20]. This paper described a method to setup connections dynamically as needed. The methods we describe in this work can be used in conjunction with an on-demand strategy for optimal memory usage.

During the preparation for this paper, Mellanox Technologies [1] implemented and released a similar coalescing scheme to enhance small message throughput for MVAPICH. Their work is focused only on improving small message throughput of messages with identical MPI message tags rather than decreasing the memory footprint. Their design artificially limits the number of send WQEs available for small messages and still allocates 200 WQEs, leaving memory usage unchanged from the default MVAPICH case.

Our contribution in this paper is evaluating the memory connection usage and the impact of send WQEs and proposing a coalescing method to maintain optimal performance using significantly less WQEs. Our design seeks to be comprehensive in providing near-identical performance using significantly less memory resources without imposing any restrictions on application behavior, such as the number of outstanding sends an application may have, the number of connections created, or their message tags.

## 7 Conclusions and Future Work

As clusters continue to increase in size, the MPI library must be scalable in resource usage. It is important to design the MPI library to support applications that require a large number of connections. Existing allocations of send WQEs can consume a large amount

of memory when deployed at scale, starving the applications of the memory needed to run simulations with higher precision or even the ability to run at all.

In this paper we have proposed lowering the number of allocated send WQEs as a method to reduce the memory usage of the MPI library and demonstrated a message coalescing design to maximize performance. Our evaluation has shown our proposed design reduces memory usage by an order of magnitude, from 1GB to 120MB per process at 8K processes, without sacrificing performance, even in the worst-case when many messages are outstanding per connection. For small message sizes, throughput with our enhancements is increased by over 150%. We have also evaluated our design against the NAS application kernels, sPPM, SMG2000, and Sweep3D and demonstrated identical performance with significantly reduced memory usage.

In the future we hope to further quantify the number of connections needed for various real-world applications to better determine expected memory requirements of the MPI library. We also plan to continue to evaluate the effect of coalescing on application performance. Additionally, we plan to extend our design to use the send queue re-size operation as specified by the InfiniBand specification after it is included in the OpenFabrics stack. We would also like to explore a shared send queue, to further reduce the memory used for WQEs.

## References

- [1] Mellanox Technologies. <http://www.mellanox.com>.
- [2] ASC. ASC Purple Benchmarks. <http://www.llnl.gov/ascii/purple/benchmarks/>.
- [3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, D. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, V. Venkatakrishnan, and S. K. Weeratunga. The NAS parallel benchmarks. volume 5, pages 63–73, Fall 1991.
- [4] P. N. Brown, R. D. Falgout, and J. E. Jones. Semi-coarsening multigrid on distributed memory machines. *SIAM Journal on Scientific Computing*, 21(5):1823–1834, 2000.
- [5] E. Gabriel, G. E. Fagg, G. Bosilca, T. Angskun, J. J. Dongarra, J. M. Squyres, V. Sahay, P. Kambadur, B. Barrett, A. Lumsdaine, R. H. Castain, D. J. Daniel, R. L. Graham, and T. S. Woodall. Open MPI: Goals, concept, and design of a next generation MPI implementation. In *Proceedings, 11th European PVM/MPI Users' Group Meeting*, pages 97–104, Budapest, Hungary, September 2004.
- [6] A. Hoisie, O. M. Lubeck, H. J. Wasserman, F. Petrini, and H. Alme. A general predictive performance model for wavefront algorithms on clusters of SMPs. In *International Conference on Parallel Processing*, pages 219–, 2000.
- [7] InfiniBand Trade Association. InfiniBand Architecture Specification. <http://www.infinibandta.com>.
- [8] K. Koch, R. Baker, and R. Alcouffe. Solution of the first-order form of the 3-d discrete ordinates equation on a massively parallel processor. *Trans. AMer. Nuc. Soc.*, pages 65–, 1992.
- [9] Lawrence Berkeley National Laboratory. MVICH: MPI for Virtual Interface Architecture. <http://www.nersc.gov/research/FTG/mvich/index.html>, August 2001.
- [10] J. Liu, B. Chandrasekaran, J. Wu, W. Jiang, S. Kini, W. Yu, D. Buntinas, P. Wyckoff, and D. K. Panda. Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics. In *Supercomputing(SC)*, 2003.
- [11] J. Liu, J. Wu, S. P. Kini, P. Wyckoff, and D. K. Panda. High Performance RDMA-Based MPI Implementation over InfiniBand. In *17th Annual ACM International Conference on Supercomputing (ICS '03)*, June 2003.
- [12] Message Passing Interface Forum. *MPI: A Message-Passing Interface Standard*, Mar 1994.
- [13] A. A. Mirin, R. H. Cohen, B. C. Curtis, W. P. Dannevik, A. M. Dimitis, M. A. Duchaineau, D. E. Eliason, D. R. Schikore, S. E. Anderson, D. H. Porter, P. R. Woodward, L. J. Shieh, and S. W. White. Very high resolution simulation of compressible turbulence on the IBM-SP system. In *Supercomputing '99: Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, New York, NY, USA, 1999. ACM Press.
- [14] Network-Based Computing Laboratory. MVA-PICH: MPI for InfiniBand. <http://nowlab.cse.ohio-state.edu/projects/mipi-iba>.
- [15] OpenFabrics Alliance. OpenFabrics. <http://www.openfabrics.org/>, April 2006.
- [16] G. Shipman, T. Woodall, R. Graham, and A. McCabe. Infiniband Scalability in Open MPI. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [17] S. Sur, L. Chai, H.-W. Jin, and D. K. Panda. Shared Receive Queue Based Scalable MPI Design for InfiniBand Clusters. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.
- [18] J. Vetter and F. Mueller. Communication Characteristics of Large-Scale Scientific Applications for Contemporary Cluster Architectures. In *IPDPS '02: Proceedings of the 16th International Symposium on Parallel and Distributed Processing*, page 27.2, Washington, DC, USA, 2002. IEEE Computer Society.
- [19] J. Wu, J. Liu, P. Wyckoff, and D. Panda. Impact of On-Demand Connection Management in MPI over VIA. In *CLUSTER '02: Proceedings of the IEEE International Conference on Cluster Computing*, Washington, DC, USA, 2002. IEEE Computer Society.
- [20] W. Yu, Q. Gao, and D. K. Panda. Adaptive Connection Management for Scalable MPI over InfiniBand. In *International Parallel and Distributed Processing Symposium (IPDPS)*, 2006.