# Efficient RDMA-based Multi-port Collectives on Multi-rail QsNet$^{II}$ Clusters

Ying Qian Ahmad Afsahi
Department of Electrical and Computer Engineering
Queen's University
Kingston, ON, Canada K7L 3N6
qiany@ee.queensu.ca ahmad.afsahi@queensu.ca

## Abstract

*Many scientific applications use MPI collective communications intensively. Therefore, efficient and scalable implementation of collective operations is critical to the performance of such applications running on clusters. Quadrics QsNet$^{II}$ is a high-performance interconnect for clusters that implements some collectives at the Elan level. These collectives are directly used by their corresponding MPI collectives.*

*Quadrics software supports point-to-point striping over multi-rail QsNet$^{II}$ networks. However, multi-rail collectives have not been supported. In this work, we propose a number of RDMA-based multi-port collectives over multi-rail QsNet$^{II}$ clusters directly at the Elan level. Our performance results indicate that the proposed multi-port gather gains an improvement of up to 6.35 for 1MB message over the native elan_gather. The proposed multi-port all-to-all performs better than the native elan_alltoall by a factor of 2.19 for 16KB message. Moreover, we have also proposed two algorithms for the scatter operation.*

## 1. Introduction

Cluster computers have become the predominant computing platform for high-performance computing and emerging commercial and networking applications due to their cost-effectiveness. In such systems, the interconnection network and the communication system software are the keys to the performance. Several contemporary interconnects such as Quadrics QsNet [15] and QsNet$^{II}$ [1], InfiniBand [9], and Myrinet [22] have been introduced to support the ever-increasing demand for efficient communication, scalability, and higher performance in clusters. Among them, QsNet$^{II}$ offers the lowest communication latency.

Most scientific applications are written on top of the Message Passing Interface (MPI) [12]. In fact, MPI has become the de-facto standard for parallel programming on clusters. MPI provides both point-to-point and collective communication operations. In QsNet$^{II}$, the MPI point-to-point implementation provided by Quadrics runs on top of a network programming interface called *Tagged Ports*, or Tports. Tports provides similar two-sided message-passing semantics as in MPI. It was initially developed by Meiko for their CS-2 interconnect [2]. QsNet$^{II}$ also supports *Remote Direct Memory Access* (RDMA) write and read operations in its lower-level Elan library via *elan_put ()* and *elan_get ()* function calls, respectively. It allows direct transfer of data from a source virtual address to a destination virtual address of two processes without the host processor intervention or any intermediate copy.

Many parallel applications need communication patterns that involve collective data movement and global control. Therefore, efficient and scalable implementation of collective communication operations is critical to the performance of such applications. Implementation of collective operations in Quadrics is very efficient. In fact, QsNet$^{II}$ provides hardware support for broadcast and barrier operations. Other collective primitives such as reduce, gather, and all-to-all are also implemented in the Elan library. These collectives are directly used by their corresponding MPI collectives.

To overcome bandwidth limitations for today's most demanding applications, and to enhance fault tolerance, using multiple independent networks known as multi-rail networks has been recently proposed to connect *symmetric multiprocessor* (SMP) nodes in high-performance clusters [6, 11]. Basically, there are two ways in distributing the traffic over multiple rails. One is to use *multiplexing*, also called message dispatching, where messages are transferred over different rails in a round robin fashion. The other method is called *message striping*, where messages are divided in multiple chunks and sent over multiple rails simultaneously.

Quadrics has a native support for message striping over multi-rail QsNet$^{II}$ networks only for large point-to-point messages through its Elan RDMA *put* and *get*, SHMEM [7] *put* and *get*, and Tports send/receive functions.

1-4244-0054-6/06/$20.00 

However, multi-port collectives with striping is not supported. Our performance results verify that only a few single-port collectives that are currently implemented on top of point to point Tports or *elan_put ()* will gain multi-rail striping from those underlying subsystems.

In this work, we take on the challenge in designing high-performance RDMA-based collectives on multi-rail QsNet$^{II}$ networks. Rather than devising single-port (MPI) collectives that utilize the underlying striping facilities, we propose and evaluate multi-port collective schemes with striping for multi-rail QsNet$^{II}$ directly at the Elan level using RDMA write. We propose algorithms for scatter, gather, and all-to-all personalized exchange operations. Our performance results indicate that the multi-port gather gains an improvement of up to 6.35 for 1MB message over the native *elan_gather*. Moreover, the proposed multi-port all-to-all performs better than the *elan_alltoall* by a factor of 2.19 for 16KB message.

To the best of our knowledge, this is the first work in devising multi-port collective communications on a modern multi-rail interconnect such as Quadrics. Although we have targeted Quadrics QsNet$^{II}$, we believe this study has implications beyond QsNet$^{II}$ as it can be applied to other multi-rail RDMA-capable networks such as InfiniBand.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the Quadrics QsNet$^{II}$ architecture including Tports, RDMA, and collective communications. In Section 3, we introduce our experimental platform. Section 4 explains the motivation behind this work by showing that only a couple single-port (MPI) collectives benefit from the underlying striping facility. Section 5 presents some known algorithms for scatter, gather and all-to-all personalized exchange used in this work. In Section 6, we propose and evaluate our RDMA-based multi-port collective algorithms on multi-rail QsNet$^{II}$ with its striping support on a 16-processor cluster. Related work is discussed in Section 7. Conclusions and future research are discussed in section 8.

## 2. Overview of QsNet$^{II}$

QsNet$^{II}$ [1] is the latest generation interconnect from Quadrics. It consists of two ASICs: Elan4 and Elite4. The network, arranged in a fat-tree topology, is built out of Elite4 switch components, where each can switch eight bi-directional channels. Elan4 is the communication processor on the network interface card (NIC). It provides a protected user-level access to the NIC so that multiple user processes can share it simultaneously. The command processor within Elan4 is responsible for such a protection through command queues generated by user processes.

QsNet$^{II}$ differs from other contemporary interconnects such as InfiniBand and Myrinet in the sense that it integrates a node's local memory into a globally shared, virtual-memory space. Data can be transferred from a source virtual address to a destination virtual address without any intermediate copy or host intervention. This is done with the help of Elan4's memory management unit (MMU). There is no need for memory registration as in other high-performance interconnects.

The Quadrics software includes both MPI and SHMEM. They are implemented with libelan and libelan4. Elan4 supports both *send/receive* and RDMA operations. The RDMA operations include RDMA write (*elan_put*), and RDMA read (*elan_get*). The send/receive mode, supported through connectionless Tports network programming interface, provides a similar two-sided message-passing semantics as in MPI. In fact, MPI point-to-point implementation on QsNet$^{II}$ uses Tports as its transport layer. A thread running on the Elan4's *thread processor* matches sends to their corresponding receives. If a matching is found, data is sent directly to the application's address space. If no matching receive is found, the thread processor locally buffers the data and performs a final copy when the receive call is posted. Availability of such a thread processor at the network interface enables MPI independent progress, communication/computation overlap, as well as offloading MPI tag matching and protocol processing [4].

The short message latency is minimized by pipelining messages onto the network through a dedicated *Short Transaction Engine* (STEN) processor on Elan4. The STEN processor requires the host CPU to copy the data, so latency increases for large messages. Therefore, for large messages, the RDMA engine is responsible for transferring data directly from the sender's user space to the receiver's user space. For very large messages, the sender only sends a descriptor to the sender, and then a DMA descriptor is sent back by the receiving thread processor to the sender for an efficient RDMA write.

### 2.1. Collective Communication in QsNet$^{II}$

QsNet$^{II}$ supports broadcast, barrier, reduce, gather, and all-to-all personalized exchange at the Elan level. These collectives are directly used by the MPI library, and thus it is very important to optimize them. A number of RDMA primitives such as *elan_put ()* and *elan_doput ()* are used in the implementation of these collectives. *elan_doput ()* is similar to *elan_put ()* with the exception that it sets a destination event on completion. These functions are non-blocking, and *elan_wait ()* may be used to test or wait on for the completion of the transfer.

In this work, we are interested in the design of efficient scatter, gather, and all-to-all personalized exchange operations. However, for the sake of completeness, we briefly discuss the other Elan collectives as well.

QsNet$^{II}$ supports hardware broadcast, *elan_hbcast (),* and hardware barrier, *elan_hgsync (),* operations only

across contiguous ranges of nodes [18]. Where this is not possible, software broadcast, *elan_bcast ()*, and software barrier, *elan_gsync (),* are used instead. Hardware broadcast and network conditionals are used to implement the barrier operation. The root process performs a network conditional by polling the ready flag of other nodes (a flag is set when a process other than root enters a barrier), and when polling returns successfully it broadcasts a set of done event to all other processes.

Reduction is done via *elan_reduce ()* function, where each node in the tree sends its results up using *elan_doput ()*. Parent nodes apply the reduction function, combine them, and send them up until it reaches the root [18]. An efficient NIC-based reduction scheme similar to the one proposed in [13] for QsNet has been implemented on the thread processor of QsNet$^{II}$.

A tree-based gather algorithm [18] is used in the *elan_gather ()*. Leaf nodes send data to their parents. Intermediate nodes add their own data and forward to their parents. This process continues until the root process gathers all data. To reduce host processor involvement, Elan event processor is used to chain the puts.

The *elan_alloall* () does an all-to-all personalized exchange amongst all the processes. Quadrics uses a pairwise exchange algorithm for up to 8KB messages and then switches to a permission to send algorithm [18, 17]. In the pairwise exchange, performance deteriorates when there are hot spots due to communication with the same destinations. In the permission to send algorithm, a node starts exchanging when it has received permission from previous process. Quadrics uses a non-blocking put algorithm for short messages, where each process posts some *elan_put ()* functions. When one transfer is done, another one will be issued [18]. It has also been shown that the *Bruck's index* algorithm [5] works well for very small messages. Note in all the collectives discussed, shared memory wrapper functions are used when multiple processes exist per node.

## 3. Experimental Platform

The experiments were conducted on a 4-node dedicated SMP cluster interconnected with two QM500-B Quadrics QsNet$^{II}$ NICs per node, and two QS8A-AA QsNet$^{II}$ E-series 8-way switches. Each node is a Dell PowerEdge 6650 that has four 1.4 GHz Intel Xeon MP Processors with 256KB unified L2 cache, 512KB unified L3 cache, and 2GB of DDR-SDRAM on a 400 MHz Front Side Bus. Each NIC is inserted in a 64-bit, 100 MHz PCI-X slot. The operating system is the Vanilla kernel version 2.6.9. Our Quadrics software is the latest release "Hawk" with the kernel patch qsnetp2, kernel module 5.10.5qsnet, QsNet Library 1.5.6-1, and QsNet$^{II}$ Library 2.2.4-1. Test codes were launched by the *pdsh* [14] task launching tool, version 2.6.1. The MPI implementation is the Quadrics MPI, version MPI.1.24-47.intel81.

## 4. Motivation

To overcome bandwidth limitations for today's most demanding applications, and to enhance fault tolerance, using multiple independent networks known as *multi-rail networks* has been recently proposed to connect SMP nodes in high-performance clusters [6, 11]. Basically, there are two ways in distributing the traffic over multiple rails. One is to use *multiplexing*, where messages are transferred over different rails in a round robin fashion. The other method is called *message striping*, where messages are divided in multiple chunks and sent over multiple rails simultaneously.

A straightforward way to have a multi-rail network is to have multiple NICs installed on each cluster node and connect them to a single or a separate switch. Coll and his colleagues [6] did a simulation study of different static and dynamic allocation schemes for multiple communication rails. Recently, Liu and his associates [11] designed multi-rail InfiniBand clusters with support for point-to-point message striping in MPI.

QsNet$^{II}$ uses multiple NICs per node to construct a multi-rail cluster network, in which the *i*-th NIC connects to the *i*-th rail. Quadrics has a native support for a simple even message striping over multi-rail QsNet$^{II}$ networks only for large point-to-point messages through its Elan *put* and *get*, SHMEM *put* and *get*, and Tports send/receive functions. Our intention in this section is to show while point-to-point messages benefit from message striping, only a few (Elan and MPI) collectives that are currently implemented on top of point to point Tports or *elan_put ()* will gain multi-rail striping.

In the following, we first present the performance of Elan *put* and *get*, Tports send/receive, as well as MPI point-to-point under single-rail and dual-rail QsNet$^{II}$ on our platform (SHMEM *put* and *get* also stripe large messages but we do not show their performance here). We will then demonstrate the performance of Elan collectives, and two of MPI collectives (MPI_Scatter and MPI_Allgather that do not have Elan counterparts).

Our point-to-point experimentation is done with uni-directional, bi-directional, and both-way traffics. In the unidirectional bandwidth test, the sender transmits a message repeatedly to the receiver, and then waits for the last message to be acknowledged. The bi-directional test is the ping-pong test where the sender sends a message and the receiver upon receiving the message, immediately replies with the same message size. This is repeated sufficient number of times to eliminate the transient conditions of the network. In the both-way test, both the sender and receiver send data simultaneously. This test

puts more pressure on the communication subsystem, and the PCI-X bus.

### 4.1. Elan RDMA Performance

Figure 1 presents the bandwidth performance of the RDMA write using the *pgping* microbenchmark available in the Elan Library. It is evident that the bandwidth is doubled in the dual-rail system. The both-way single-rail and dual-rail *elan_put ()* bandwidths are 670MB/s and 1332 MB/s, respectively. The bandwidth for *elan_get ()* is almost the same as *elan_put ()* in each case (not shown).

The Elan RDMA write short message latency does not change much between single-rail and dual-rail. The latency varies between 2 µs to 2.77 µs for a 4-byte message. The *elan_get ()* short message latency is slightly larger than the RDMA write. That is why we decided to use *elan_put ()* in our collective implementations.

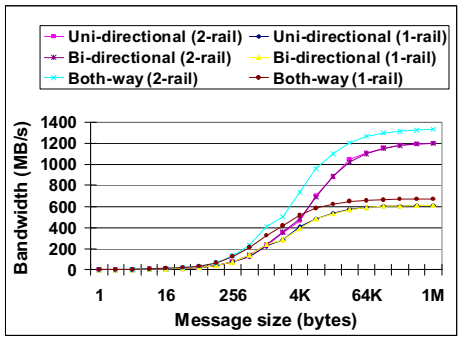


**Figure 1. Elan RDMA write performance.**

### 4.2. Tports Performance

Figure 2 shows the Tports bandwidth. Tests are done using the *tping* microbenchmark (except for the uni-directional case, where we wrote our own code). Like the Elan RDMA, the dual-rail Tports bandwidth outperforms the single-rail bandwidth in each case. The single-rail T-ports bandwidth is roughly the same as RDMA bandwidth; however, dual-rail bandwidth falls short of RDMA. The short message latency is slightly larger than the RDMA.

### 4.3. MPI Send/Receive Performance

Figure 3 compares the MPI bandwidth under different cases. Unlike the both-way, the uni-directional and bi-directional MPI bandwidths for dual-rail are almost doubled. This shows that the MPI point-to-point implementation over Tports mostly benefit from striping in the dual-rail QsNet$^{II}$. The short message MPI latency is close to that of the T-ports (not shown here).

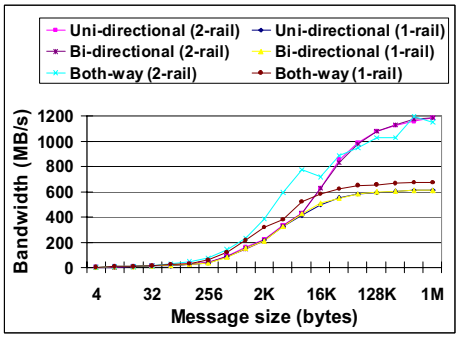


**Figure 2. T-port send/receive performance.**

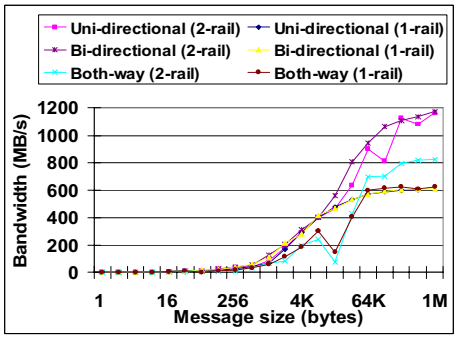


**Figure 3. MPI send/receive performance.**

### 4.4. Collective Performance

Figure 4 depicts the aggregate bandwidth for the Elan hardware and software broadcasts, gather, and all-to-all, as well as MPI_Scatter and MPI_Allgather. For the Elan collectives, we have used the *gping* microbenchmark available in the Elan Library. For the MPI collectives, we have written our own code.

From the results, except for the *elan_alltoall ()* (and *elan_reduce ()*, not shown here) other collectives at the Elan level do not benefit from the dual-rail QsNet$^{II}$. MPI_Scatter and MPI_Allgather are implemented on top of Tports, but only MPI_Scatter achieves larger bandwidth under dual-rail.

Efficient implementation of collective operations is one of the keys to the performance of parallel applications. Given the multi-rail performance offered at the Elan and Tports levels, excellent opportunities exist for devising efficient collectives for such systems.

Basically, there are two ways to improve the performance of collectives on multi-rail systems. One is to implement single-port collective communication algorithms that gain multi-rail striping from the underlying communication subsystem. This is the approach currently used for MPI_Scatter and *elan_alltoall ()*. However, this will only improve the performance for large messages. The second approach that we propose is to design and implement multi-port algorithms for multi-rail systems that also benefit from the striping feature supported by

QsNet$^{II}$. We have used some known multi-port algorithms and implemented them on our dual-rail QsNet$^{II}$ network directly at the Elan level using RDMA write.

## 5. Collective Algorithms

In this section, we provide an overview of some known algorithms for scatter, gather, and all-to-all personalized exchange. In the following discussion, $N$ is the number of processors (or processes) and $k$ is the number of ports in the multi-port algorithms (equal to the number of available rails). In the $k$-port (or multi-port) modeling, each process has the ability to simultaneously send and receive $k$ messages on its $k$ links. The assumption is that the number of processes is a power of $(k + 1)$. Otherwise, dummy processes can be assumed to exist until the next power of $(k + 1)$, and the algorithms apply with little or no performance loss.

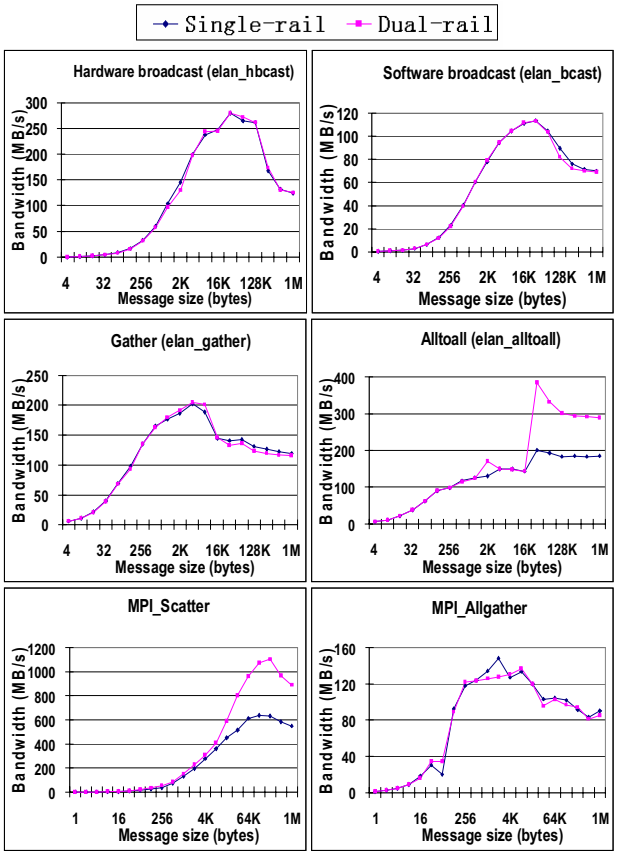


**Figure 4. Collective performance on 16 processes.**

### 5.1. Scatter

The *spanning binomial tree* algorithm [10] can be extended for $k$-port modeling. In this algorithm, the scattering process sends $k$ messages of length $N/(k + 1)$ each, to its $k$ children. Therefore, there are $(k + 1)$ processes having $N/(k + 1)$ different messages. These processes, at step 2, send one $(k + 1)$-th of their initial message to each of their immediate $k$ children. This process continues and all processes are informed after $log_{k+1} N$ communication steps. Using *Hockney's* model [8], the total communication time, $T$, is:

$$T = (t_s \times \log_{k+1} N) + (l_m \times \tau) \sum_{i=1}^{\log_{k+1} N} (k+1)^{(\log_{k+1} N)-i}$$

$$T = (t_s \times \log_{k+1} N) + \frac{N-1}{k}(l_m \times \tau) \qquad (1)$$

where, $t_s$ is the message startup time, $l_m$ is the message size in bytes, and $\tau$ is the time to transfer one byte.

The above algorithm has a logarithmic number of steps, therefore suitable for short messages and networks where the cost of message transfer is dominated by the startup latency. Another algorithm, for large messages, is the extension of *sequential tree* algorithm for $k$-port modeling. At each step, the source process sends its $k$ different messages to $k$ other processes. There are a total of $(N - 1)/k$ communication steps. Therefore, the total communication cost, $T$, is:

$$T = \frac{N - 1}{k} \times (t_s + l_m \times \tau) \qquad (2)$$

### 5.2. Gather

Gather is the exact reverse of scatter so the same spanning binomial tree algorithm extended for k-port modeling can be used. However, the communication starts from the leaf processes and messages are combined in the intermediate processes until it reaches the root. The total communication cost is the same as in Equation (1).

### 5.3. All-to-all Personalized Exchange

A lower bound for all-to-all personalized exchange time is $(N - 1)/k$ since each process must receive $N - 1$ different messages and it can only receive at most $k$ messages at a time. A simple algorithm is based on the extension of the direct algorithm for $k$-port modeling. The processes are arranged in a virtual ring. That is, at step $i$, process $p$ sends its message to processes $(p + (i - 1)k + 1)$ *mod* $N$, $(p + (i - 1)k + 2)$ *mod* $N$, …, $(p + ik)$ *mod* $N$. Modulus operation avoids sending messages to a single destination. The communication cost is the same as in Equation (2).

## 6. RDMA-based Implementation and Performance

In this section, the intention is to show the effectiveness of the multi-port algorithms introduced in Section 5 on multi-rail QsNet$^{II}$ with striping support, when they are implemented directly at the Elan layer using Elan RDMA write.

Memory registration/deregistration is a costly operation. However, contrary to InfiniBand and Myrinet, QsNet$^{II}$ does not need memory registration and address exchange for message transfers. This eases the implementation, and effectively reduces the communication latency.

Our algorithms are two-port *put*-based algorithms, where a sending process has direct control in sending messages simultaneously over the two rails using the *elan_doput ()* function. When a message is larger than a threshold (1KB) even message striping is used over the two rails. When a message is sent, the sending process uses the *elan_wait ()* to make sure the user buffer can be re-used.

In the latest Quadrics Hawk distribution, release 2.2.4-1, remote event notification is enabled in *elan_doput ()* for both single-rail and multi-rail systems. This allows multi-rail striped (ELAN_RAIL_ALL) *put* messages to have a *devent* (destination event). In this case the *devent* will be set once in each rail and in the target process one will need to call *elan_initEvent ()* once for each rail and then wait on each *ELAN_EVENT* to be returned. This guarantees a message has been delivered in its entirety.

In the implementation of our algorithms, processes do not synchronize with each other. Note that our implementation is completely put-based, but we are in the process of optimizing them for the cases where processes are co-located on the same 4-way nodes.

### 6.1. Evaluation of Scatter

We have implemented the multi-port spanning **B**inomial tree algorithm for **S**catter operation on multi-rail QsNet$^{II}$ systems using **R**DMA **W**rite. We call this scheme as BSRW. Likewise, we call the sequential tree implementation for scatter as SSRW.

Figure 5 compares the performance of the two scatter algorithms, BSRW and SSRW, on our dual-rail QsNet$^{II}$. As expected, BSRW is superior for short messages, while SSRW has a much better performance for medium and large messages. Figure 5 also presents the scalability of our implementation. The scalability figures show that indeed the BSRW is the better algorithm for short messages with increasing system size.

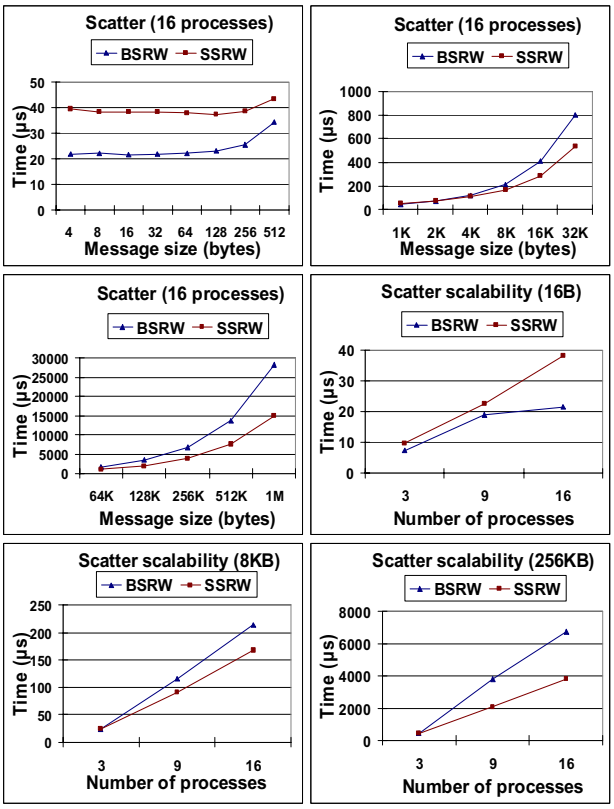


**Figure 5. Scatter performance and scalability.**

### 6.2. Evaluation of Gather

The multi-port spanning **B**inomial tree algorithm for **G**ather operation has been implemented on multi-rail QsNet$^{II}$ systems using **R**DMA **W**rite feature. We call this scheme as BGRW.

Figure 6 compares the performance of our gather algorithm, BGRW, with the *elan_gather ()*. The results are very promising as our implementation is much better than the native implementation except slightly for short messages less than 64 bytes. Interestingly, the proposed multi-port gather gains an improvement of up to 6.35 for 1MB message. The scalability plots in Figure 6 verify the superiority of our gather algorithm for short to medium, medium, and large messages. However, it does show that with increasing number of processes *elan_gather ()* is better for very short messages.

### 6.3. Evaluation of All-to-all Personalized Exchange

We have also implemented the multi-port **D**irect algorithm for **A**ll-to-all personalized exchange on multi-rail QsNet$^{II}$ systems using **R**DMA **W**rite. We call this scheme as DARW.

Figure 7 compares the performance of our all-to-all algorithm, DARW, with the *elan_alltoall ()*. The results are again encouraging. Our multi-port all-to-all algorithm and its implementation is much better than the native *elan_alltoall ()* for medium size messages. In fact, the improvement is up to a factor of 2.19 for 16KB message. However, *elan_alltoall () is* better than DARW for short messages up to 256 bytes, and for 32KB message size when it switches its algorithm. For large message sizes, our two-port algorithm is slightly better. The scalability plots confirm these findings. We are currently working on the *standard exchange* algorithm [3], and *Bruck's index* algorithm [5] for multi-rail QsNet$^{II}$. These algorithms are known to be superior for short messages.

## 7. Related Work

Study of collective communication operations has been an active area of research. Recently, the authors in [16] analyzed the performance of collective communication operations under different communication cost models. Thakur and his colleagues discussed recent collective algorithms used in MPICH [21]. They have shown some algorithms perform better than the others depending on the message size, and the number of processes.

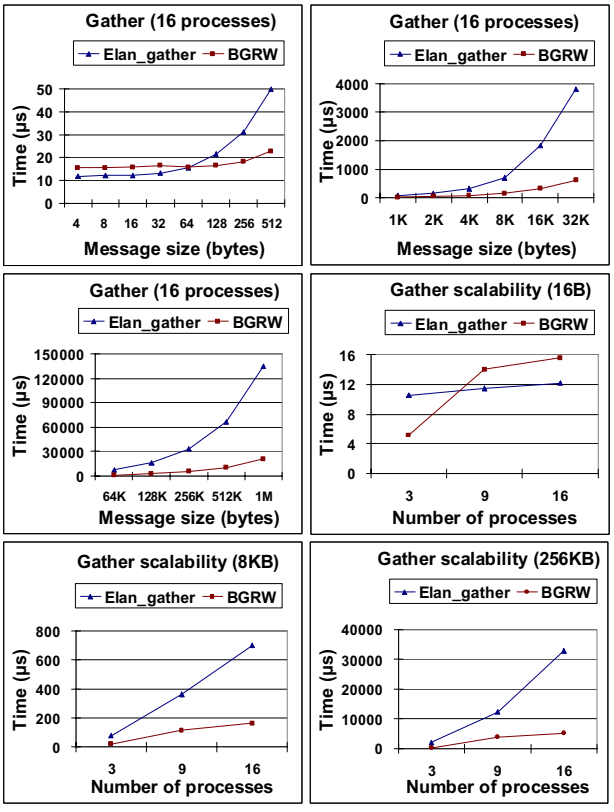


**Figure 6. Gather performance and scalability.**

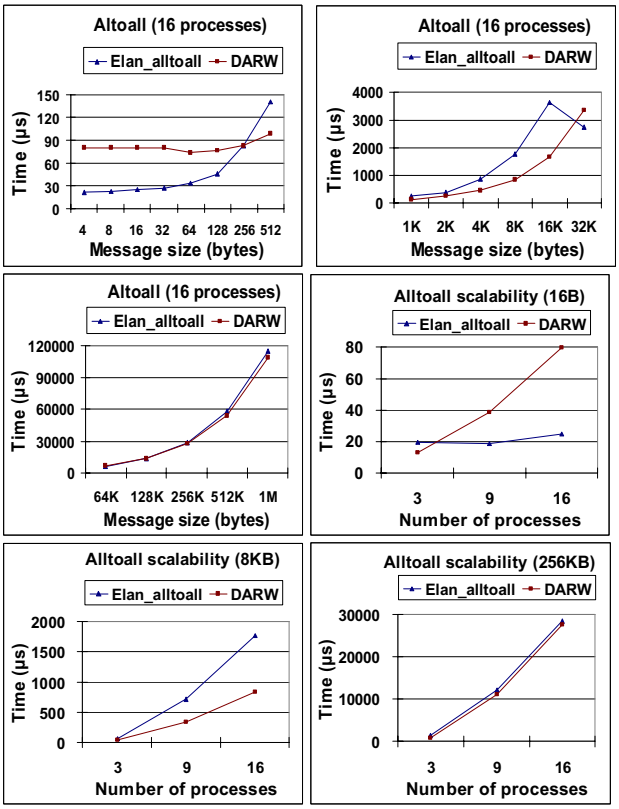


**Figure 7. All-to-all performance and scalability.**

Different collective algorithms on QsNet$^{II}$ have been studied in [18, 17]. Sur, et al. proposed efficient RDMA-based all-to-all broadcast [20] and all-to-all personalized exchange [19] algorithms for InfiniBand–based clusters.

Coll and his associates did a comprehensive simulation study on static and dynamic allocation schemes for multi-rail systems [6]. Recently, the authors in [11] designed an MPI-level multi-rail InfiniBand clusters. However, their work was only focused on point-to-point communications.

## 8. Conclusions

Interconnection networks and the supporting communication system software are the deciding factors in the performance of clusters. Specifically, efficient implementation of collective operations is critical to the performance of MPI applications.

QsNet$^{II}$ is a high-performance network for clusters that implements some collectives at the Elan level. Their MPI counterparts directly use them. Therefore, optimizations and inclusion of new collectives at the Elan level are extremely desirable.

Quadrics supports point-to-point message striping over multi-rail QsNet$^{II}$. In this work, we have proposed and implemented a number of multi-port collectives at the Elan level over multi-rail QsNet$^{II}$ systems. Our performance results indicate that our multi-port gather

gains an improvement of up to 6.35 for 1MB message over the native *elan_gather ()*. The proposed multi-port all-to-all performs better than the *elan_alltoall ()* by a factor of 2.19 for 16KB message. Moreover, we have proposed two algorithms for short and long messages for the scatter operation.

The results are encouraging and future work in this area is justified. Our all-to-all algorithm did not perform well for short messages. For this, we are currently experimenting with the *standard exchange* [3], and *Bruck's index* [5] algorithms. We are also trying to utilize the shared memory wrapper facility of Quadrics software to speedup the collectives for co-located processes on SMP nodes. Optimal static and dynamic striping mechanisms may also help boost the performance. We intend to extend our study by devising other collective communications of interests and testing them on larger multi-rail clusters. NIC-based or NIC-assisted collectives for multi-rail systems, and taking advantage of basic hardware collectives in their design are other areas of interests.

## Acknowledgments

The authors would like to thank the anonymous referees for their insightful comments. Many thanks goes to David Addison of the Quadrics for his help on our overall understanding of the notification events in multi-rail QsNet$^{II}$ systems. This work was supported by grants from the Natural Sciences and Engineering Research Council of Canada (NSERC), Queen's University, Canada Foundation for Innovation (CFI), and Ontario Innovation Trust (OIT).

## References

[1] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini, and J. Nieplocha. QsNetII: Defining high-performance network design. *IEEE Micro*, 25(4):34-47, July-Aug. 2005.

[2] J. Beecroft, M. Homewood, and M. McLaren. Meiko CS-2 interconnect Elan-Elite design. *Parallel Computing*, 20(10-11):1627-1638, Nov. 1994.

[3] S. H. Bokhari, Multiphase complete exchange on Paragon, SP2, and CS-2. *IEEE Parallel and Distributed Technology*, 4(3):45-59, Sept. 1996.

[4] R. Brightwell, D. Doerfler, and K. D. Underwood. A comparison of 4X InfiniBand and Quadrics Elan-4 technologies. In *2004 IEEE International Conference on Cluster Computing (Cluster 2004)*, pages 193-204, Sept. 2004.

[5] J. Bruck, C.-T. Ho, S. Kipnis, E. Upfal, and D. Weathersby. Efficient algorithms for all-to-all communications in multiport message-passing systems. *IEEE Transactions on Parallel and Distributed Systems*, 8(11):1143-1156, Nov. 1997.

[6] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie, and L. Gurvits. Using multirail networks in high performance clusters. In *3rd IEEE International Conference on Cluster Computing (Cluster'01)*, pages 15-24, Oct. 2001.

[7] Cray Man Page Collection: Shared Memory Access (SHMEM) S-2383-23, Available: http://docs.cray.com/.

[8] R. Hockney. The communication challenge for MPP, Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3):389–398, Mar. 1994.

[9] InfiniBand Architecture. Available: http://www.infinibandta.org/.

[10] S. L. Johnson and C.-T. Ho. Optimum broadcasting and personalized communication in hypercubes. *IEEE Transactions on Computer*, 38(9): 1249-1268, Sept. 1989.

[11] J. Liu, A. Vishnu and D. K. Panda. Building multirail InfiniBand clusters: MPI-level design and performance evaluation. In *2004 ACM/IEEE Conference on Supercomputing (SC'04)*, Nov. 2004.

[12] Message Passing Interface Forum: MPI, A Message Passing Interface standard, Version 1.2, 1997.

[13] A. Moody, J. Fernandez, F. Petrini, and D. K. Panda. Scalable NIC-based reduction on large-scale clusters. In *2003 ACM/IEEE Conference on Supercomputing (SC'03)*, Nov. 2003.

[14] PDSH: available: http://www.llnl.gov/linux/pdsh/.

[15] F. Petrini, S. Coll, E. Frachtenberg, and A. Hoisie, Performance evaluation of the Quadrics interconnection network. *Journal of Cluster Computing*, 6(2):125-142, Apr. 2003.

[16] J. Pjesivac-Grbovic, T. Angskun, G. Bosilca, G. E. Fagg, E. Gabriel, and J. J. Dongarra. Performance analysis of MPI collective operations. In *19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05)*, pages 272a-272a, Apr. 2005.

[17] D. Roweth and A. Moody. Performance of all-to-all on QsNet$^{II}$". Quadrics White Paper, Available at http://www.quadrics.com/.

[18] D. Roweth, A. Pittman, and J. Beecroft. Optimized collectives on QsNet$^{II}$. Quadrics White Paper, Available at http://www.quadrics.com/.

[19] S. Sur, H.-W. Jin, and D. K. Panda. Efficient and scalable all-to-all personalized exchange for InfiniBand clusters. In *2004 International Conference on Parallel Processing (ICCP'04),* pages 275-282, 2004.

[20] S. Sur, U. K. R. Bondhugula, A. Mamidala, H.-W. Jin, and D. K. Panda. High performance RDMA based all-to-all broadcast for InfiniBand clusters. *In International Conference on High Performance Computing (HiPC 2005)*, Dec. 2005.

[21] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications,* 19(1):49-66, 2005.

[22] R. Zamani, Y. Qian, and A. Afsahi. An evaluation of the Myrinet/GM2 two-port networks. In *3rd IEEE Workshop on High-Speed Local Networks, HSLN 2004*, pages 734-742, Nov. 2004.