

# RDMA-based and SMP-aware Multi-port All-Gather on Multi-rail QsNet<sup>II</sup> SMP Clusters

Ying Qian

Ahmad Afsahi

*Department of Electrical and Computer Engineering  
Queen's University, Kingston, ON, Canada K7L 3N6  
ying.qian@ece.queensu.ca ahmad.afsahi@queensu.ca*

## Abstract

*Clusters of Symmetric Multiprocessors (SMP) are more commonplace than ever in achieving high-performance. Scientific applications running on clusters employ collective communications extensively. Using shared memory communication among co-located processes on SMP nodes as well as Remote Direct Memory Access (RDMA) operations for inter-node communication and trying to overlap them is a proven technique in boosting the performance of collective operations. The effect is much more pronounced when efficient multi-port collectives on multi-rail networks are devised and implemented.*

*In this work, we design and implement multi-port RDMA-based and SMP-aware all-gather algorithms with message striping over multi-rail QsNet<sup>II</sup> directly at the Elan level. We compare our algorithms against RDMA-only traditional algorithms and the native `elan_gather()`. Our performance results indicate that the proposed SMP-aware Bruck all-gather gains an improvement of up to 1.96 for 4KB messages over the native `elan_gather()`. Meanwhile, the Direct algorithm achieves up to 1.49 improvement for 32KB messages.*

## 1. Introduction

SMP Clusters are the predominant platforms for high-performance computing due to their cost-performance effectiveness. SMP nodes are traditionally equipped with multiple single-core processors. However, industry has recently adopted the development of aggressive Chip Multiprocessors for general-purpose applications. With the emergence of such multi-core SMP nodes in clusters, each core will run at least one process with multiple intra-node and inter-node connections to several other processes. This will put immense pressure on the interconnection network and its communication system software.

Scientific applications written in *Message Passing Interface* (MPI) [13] typically use communication patterns that involve collective data movement and global control. While Quadrics QsNet<sup>II</sup> [1] has hardware support for broadcast and barrier operations, a number of collectives such as reduce, gather, all-gather and all-to-all are implemented in its Elan user-level library. These collectives are directly used by their MPI counterparts. As such, efficient and scalable implementation of collectives at the Elan level is critical to the performance of MPI applications.

Quadrics has a native support for even message striping over *multi-rail* QsNet<sup>II</sup> networks for point-to-point messages through its Elan RDMA *put* and *get*, SHMEM *put* and *get*, and *Tports* send and receive functions. Previous studies [15, 16] confirm that only a few collectives that are currently implemented on top of *Tports* or `elan_put()` will gain multi-rail striping from the underlying subsystems. Recently, Quadrics has included support for multi-rail broadcast and all-to-all as well. However, to our knowledge, such collectives use single-port algorithms.

In [15, 16], the authors devised and implemented a number of multi-port collectives over multi-rail QsNet<sup>II</sup> clusters directly at Elan level. However, their work did not address optimized collectives for SMP clusters. In SMP clusters, intra-node communication is typically done via shared memory, while inter-node communication is done through the network. Recent research has focused on designing efficient intra-node communications between processes on the same SMP node [4, 5, 9]. Some work has also been reported on devising collective algorithms for SMP clusters [11, 12, 20, 25, 26, 28]. In this work, we take on the challenge to design and implement efficient all-gather collective algorithms over multi-rail QsNet<sup>II</sup> SMP clusters directly at the Elan level. All-gather is an intensive operation, used in linear algebra operations and matrix multiplication, where data provided by each process is gathered on all processes involved in the operation.

We propose and evaluate multi-port [6] RDMA-based and SMP-aware all-gather algorithms with message striping. We will compare them against the native *elan\_gather()* and our RDMA-only traditional all-gather collectives in [16]. Some of our algorithms overlap intra-node and inter-node communications [25, 28, 12], and use multiple outstanding RDMA to exploit concurrency [24]. Moreover, data buffers are shared between inter-node and intra-node communications in Quadrics software. Our traditional RDMA-only multi-port all-gather algorithms include the *Direct*, *Standard Exchange* and *Bruck* algorithms. The *Direct* algorithm is targeted for medium to large messages. The *Standard Exchange* algorithm typically performs better for short to medium size messages, while the *Bruck* algorithm is supposed to perform better for short messages. Our combined RDMA-based and SMP-aware algorithms include an *SMP-aware Gather and Broadcast* algorithm, as well as *SMP-aware Direct/Bruck all-gather* algorithms for small to medium size messages. Our performance results indicate that the proposed *SMP-aware Bruck* all-gather is superior among all algorithms for 512B to 8KB messages, and gains an improvement of up to 1.96 for 4KB messages over the native *elan\_gather()*. Meanwhile, the *Direct* algorithm achieves up to 1.49 improvements for 32KB messages. It is the best algorithm for 16KB to 1MB messages.

The rest of the paper is organized as follows. In Section 2, we provide an overview of the Quadrics QsNet<sup>II</sup> and its gather and all-gather algorithms. Related work is discussed in Section 3. The experimental platform is found in Section 4. Section 5 explains the motivation behind this work. Section 6 presents our multi-port RDMA-only, as well as RDMA-based and SMP-aware all-gather algorithms. In Section 7, we present the performance results of our collective algorithms on a 16-processor dual-rail QsNet<sup>II</sup> SMP cluster. Section 8 concludes the paper and plans for future work.

## 2. Overview of All-gather in QsNet<sup>II</sup>

QsNet<sup>II</sup> is the latest generation interconnect from Quadrics used in high-performance clusters. It supports RDMA Write and Read operations via *elan\_put()* and *elan\_get()* functions, respectively. RDMA allows direct transfer of data from a source process virtual address to a destination process virtual address without the host processor intervention or any intermediate copy. 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.

QsNet<sup>II</sup> supports broadcast, barrier, reduce, gather, all-gather and all-to-all personalized exchange at the Elan level. In this work, we are interested in the design of efficient all-gather operation. *elan\_gather()* in the Elan library takes care of the gather and all-gather collectives. The gather algorithm uses a tree-based algorithm among the processes [18]. Leaf processes send data to their parents. Intermediate processes 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 on the NIC is used to chain the RDMA puts [19]. In SMP clusters, data (up to 2KB) are gathered in the node's shared memory buffer. Inter-node gather is then performed on a tree formed by the first process of each node. For medium size messages, a tree-based algorithm is used among all processes in the system. For messages larger than 4KB, Tports Send/Recv is used among all processes, which benefits from message striping in multi-rail QsNet<sup>II</sup>. For all-gather, *elan\_gather()* uses the gather algorithm followed by broadcast for messages up to 32KB. For larger messages, it switches to the ring algorithm. Note all the algorithms in *elan-gather()* are single-port algorithms.

## 3. Related Work

Study of collective communications has been an active area of research. Thakur and his colleagues discussed recent collective algorithms used in MPICH [23]. They have shown some algorithms perform better depending on the message size and the number of processes. In [27], Vadhiyar et al. introduced the idea of automatically tuned collectives in which collective communications are tuned for a given system by conducting a series of experiments on the system.

Some work has been reported on the use of RDMA in the design and implementation of collectives on modern networks. Roweth and his associates studied how collectives have been devised and implemented on QsNet<sup>II</sup> [18, 19]. Sur et al. proposed efficient RDMA-based all-to-all broadcast and personalized exchange algorithms for InfiniBand-based clusters [21, 22]. In [24], Tipparaju and Nieplocha used the concurrency available in modern networks to optimize MPI\_Allgather operation on InfiniBand and QsNet<sup>II</sup>.

On multi-rail systems, Coll and his colleagues [7] did a comprehensive simulation study on static and dynamic allocation schemes for multi-rail systems. Liu and others [10] designed an MPI-level multi-rail InfiniBand clusters. However, their work was only focused on point-to-point communications. On Multi-rail collectives, Qian and Afsahi [15, 16] designed and implemented RDMA-based multi-port collectives on multi-rail QsNet<sup>II</sup> networks. Their work did not address

shared memory communication in SMP clusters. Chan et al. [6] implemented a number of multi-port MPI collectives for IBM Blue Gene/L.

On the SMP clusters, some recent work has been devoted to improve the performance of intra-node communications on SMPs [4, 5, 9]. Buntinas et al. [4] have used shared buffers, message queues, Ptrace system call, kernel copy, and NIC loopback mechanisms to improve large data transfers in SMP systems. In [5], Chai and others improved the intra-node communication by using the system cache efficiently, and requiring no locking mechanisms. In [9], Jin et al. implemented a portable kernel module interface to support intra-node communications.

On collectives for SMP clusters and Large SMP nodes, Sistare and his colleagues presented new algorithms taking advantage of high backplane bandwidth of shared-memory systems [20]. In [25], Tipparaju and his colleagues overlapped shared memory intra-node and remote memory access inter-node communications in devising collectives for IBM SP. A leader-base scheme was proposed in [11] to improve the performance of broadcast over InfiniBand. In [28], Wu and others used MPI point-to-point across the network and shared memory within the SMP node to improve the performance of a number of collectives. In [26], Traff devised an optimized all-gather algorithm for SMP clusters. Ritzdorf and Traff [17] used similar techniques in enhancing NEC’s MPI collectives. Mamidala et al. [12] designed all-gather over InfiniBand using shared memory for intra-node and single-port recursive doubling algorithm for inter-node communication via RDMA.

## 4. Experimental Platform

The experiments were conducted on a 4-node dedicated SMP cluster interconnected with two QM500-B Quadrics QsNet<sup>II</sup> NICs per node, and two QS8A-AA QsNet<sup>II</sup> E-series 8-way switches. Each node is a Dell PowerEdge 6650 that has four 1.4GHz Intel Xeon MP Processors with 256KB unified L2 cache, 512KB unified L3 cache, and 2GB of DDR-SDRAM on a 400MHz Front Side Bus. Each NIC is inserted in a 64-bit, 100MHz 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 1.5.9-1, and QsNet<sup>II</sup> Library 2.2.11-2. Test codes were launched by the `pdsh` [14] task launching tool, version 2.6.1. The MPI implementation is the Quadrics MPI, version MPI.1.24-49.intel81.

## 5. Motivation

In [15, 16], the authors designed and implemented multi-port RDMA-only scatter, gather, all-gather and alltoall collectives directly at the Elan level over multi-rail QsNet<sup>II</sup>. While the performance of our algorithms was excellent for medium to large messages, they lagged behind the native QsNet<sup>II</sup> implementations for short to medium size messages. This paper seeks to propose efficient gather and all-gather algorithms for all message sizes. In this section, we do a feasibility study of the potential performance that could be gained in our algorithms by using multi-port message striping and shared memory communication.

To overcome bandwidth limitations and to enhance fault tolerance, using multiple independent networks known as multi-rail networks [7, 10] is very promising. Quadrics uses a simple, even message striping where messages are divided in multiple chunks and sent over multiple rails simultaneously. While point-to-point messages and some collectives may gain from message striping, it has been shown that all-gather at the Elan level does not benefit [15, 16]. It should be mentioned that `elan_gather()` uses single-port algorithms. Therefore, there is room for potential performance improvement for all-gather for medium to large messages by devising multi-port algorithms and utilizing message striping over multi-rail QsNet<sup>II</sup>.

### 5.1. Shared Memory vs. RDMA

Intra-node communication can be done using shared memory copying via shared buffers/queues, kernel-based copying, and copying through the NIC [4]. In the shared memory copying approach, a memory region is shared between the two processes. The sending process copies its message into the shared buffer and then sets a shared, synchronization flag. The receiving process polls on the flag to realize whether the sending process has finished writing. It then copies the data from the shared buffer to its own buffer. Finally, it resets the flag.

The NIC-based copying method is basically an intra-node RDMA Write operation. The kernel-based copying method eliminates one of the two copies associated with the shared memory method. However, it requires an expensive system call. Therefore, we do not consider it in our study.

We have implemented a shared memory point-to-point communication mechanism based on shared buffers. Our implementation requires no locking, and uses the `memcpy()` function. Figure 1 compares our shared memory implementation (`shm_p2p`) with intra-node RDMA write, `elan_put()`, and with concurrent `memcpy()` operations. For all the tests, results are

averaged over 1000 iterations. By *k-memcpy()*, we mean *k* processes simultaneously writing data onto *k* sections of a shared memory region. We present up to four concurrent *memcpy()* operations as our experimental cluster uses quad-way SMP nodes.

From Figure 1, one can conclude that shared memory implementation is the preferred method for intra-node communication, but only up to 2KB messages; afterwards, RDMA is better. We believe, in implementing collectives, this is the main reason why Quadrics uses shared memory intra-node communication among co-located processes only for messages smaller than 2KB.

Prior research [4, 5] has mostly focused on efficient shared-memory communication only for point-to-point transactions (such as *shm\_p2p*). However, to implement an SMP-aware per-node collective, such as gather, co-located processes just need to concurrently transfer their messages to different sections of a shared memory region using *memcpy()* operations; and then the root process copies the entire shared-memory buffer into its own destination buffer using another *memcpy()* operation (synchronization is also needed). Typically, SMP nodes support concurrent *memcpy()* operations efficiently for short to medium size messages. This is clear from the results in Figure 1 as all *k-memcpy()* operations take much less time than an intra-node RDMA operation (In fact, this is true up to 128KB messages). Intuitively, one can argue shared memory regions can be effectively used for per-node collectives for messages larger than 2KB as well, where they should potentially provide better performance than RDMA implementations.

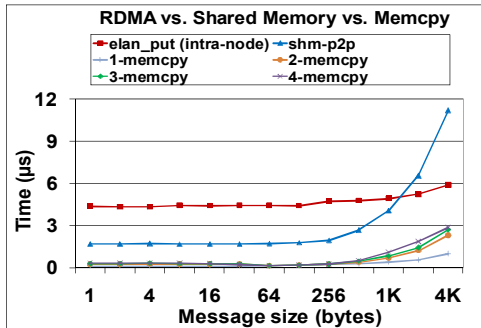


Figure 1. Intra-node communication

Our SMP-aware all-gather algorithms in Section 6 use per-node shared memory gather and broadcast. We have implemented these primitives on our 4-way SMP node in order to empirically find the maximum message size that should be transferred via shared memory for an efficient gather and broadcast operation. Our per-node shared memory gather described above includes an optimization, as shown in Figure 3. For the shared

memory broadcast, the Master (root) process copies its data to the shared buffer and then sets a synchronization flag. All other processes poll on this flag and then copy the data to their destination buffers. All processes then synchronize (using *elan\_hgsync*) to complete the operation.

Figure 2 presents the results for the shared memory gather and broadcast operations on our 4-way SMP node. While our shared-memory broadcast (*shm\_bcast*) outperforms Elan hardware broadcast (*elan\_hbcast*) and software broadcast (*elan\_bcast*) for 256B to 32KB messages (with comparable results for very short messages), our shared-memory gather (*shm\_gather*) is better than, or comparable to, the native *elan\_gather()* for up to 8KB messages. Therefore, on our platform, we use shared memory for messages up to 8KB. It is clear that this message size can be found empirically for other single-core/multi-core SMPs.

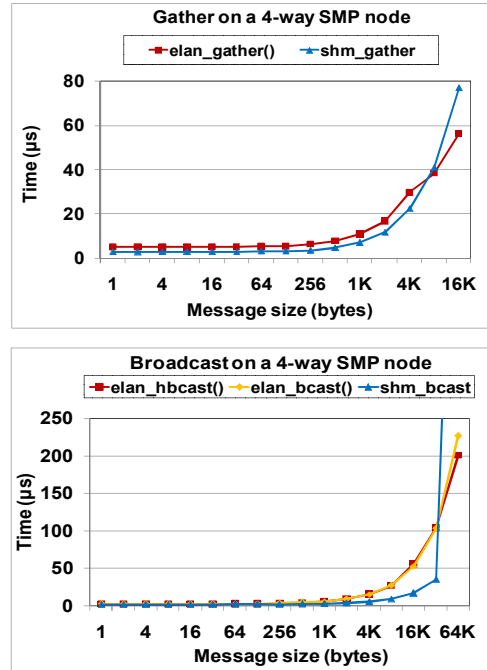


Figure 2. Intra-node gather and broadcast

## 6. Algorithms and Implementation Issues

In this section, we introduce our SMP-aware all-gather algorithms. For the sake of completeness, we briefly discuss our traditional algorithms [16].

### 6.1. Traditional All-gather Algorithms

By traditional algorithms, we mean algorithms that are designed for flat systems, and executed across all processes in the system. We study three well-known multi-port algorithms: *Direct*, *Standard Exchange* [2], and *Bruck* [3] algorithms. In the following discussion,

$N$  is the number of 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 communication between any pair of processes has the same cost.

**6.1.1. Direct Algorithm.** The *Direct* all-gather algorithm is the extension of *sequential tree* algorithm for  $k$ -port modeling, and suitable for large messages. In each step, each process sends its own message to  $k$  other processes in a wrap-around fashion. There are a total of  $\lceil N-1/k \rceil$  communication steps. Using *Hockney's* model [8], 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 total communication cost,  $T$ , is:

$$T = \lceil N-1/k \rceil \times (t_s + l_m \times \tau) \quad (1)$$

**6.1.2. Standard Exchange Algorithm.** The *Standard Exchange* all-gather algorithm [2] is the extension of *Recursive Doubling* algorithm for  $k$ -port modeling, and works for power of  $(k+1)$  processes. It performs better for up to medium size messages. In the  $k$ -port *Standard Exchange* algorithm, processes are divided into  $N/(k+1)$  groups of  $(k+1)$  processes each. Processes are grouped as  $(0, 1, \dots, k)$ ,  $(k+1, k+2, \dots, 2(k+1)-1)$ ,  $\dots$ ,  $(N - (k+1), N - (k+1)+1, \dots, N - 1)$ . In step 1, all processes within a group exchange their messages using  $k$ -port. At the end of this step, each process has  $(k+1)$  messages. In step 2, process  $p$  exchanges all its messages with processes  $(p + (k+1)) \bmod N$ ,  $(p + 2(k+1)) \bmod N$ ,  $\dots$ ,  $(p + k(k+1)) \bmod N$ . At the end of this step, each process has  $(k+1)^2$  messages. This continues to the step  $\log_{k+1} N$ . At each step  $i$  of this algorithm, each process sends messages of size  $(k+1)^{i-1}$  to  $k$  other processes. Note this algorithm needs correction steps when number of processes is not a power of  $(k+1)$ . The total communication cost,  $T$ , is:

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

**6.1.3. Bruck Algorithm.** The *Bruck* all-gather algorithm [3] works on any number of processes, and is proposed to improve the performance for small messages. The all-gather operation among  $N$  processes can be represented as a sequence of process-memory configurations. Each process has an  $N$ -block output buffer. Initially, local data is placed at the top of the output buffer.

The algorithm consists of two phases. Phase 1 has  $\lfloor \log_{k+1} N \rfloor$  steps. In each step  $i$  of phase 1, process  $p$

sends all its data to processes  $(p - (k+1)^i)$ ,  $(p - 2(k+1)^i)$ ,  $\dots$ ,  $(p - k(k+1)^i)$ , and stores the data it receives from processes  $(p + (k+1)^i)$ ,  $(p + 2(k+1)^i)$ ,  $\dots$ ,  $(p + k(k+1)^i)$  at the end of the data it already has. An additional step is required if  $N$  is not a power of  $(k+1)$ , where each process sends the first  $(N - (k+1)^{\lfloor \log_{k+1} N \rfloor})$  blocks from the top of its output buffer to the destination processes and appends the received data to the end of its current data. The second phase consists of a single round local memory shift. The total communication cost is the same as in Equation (2).

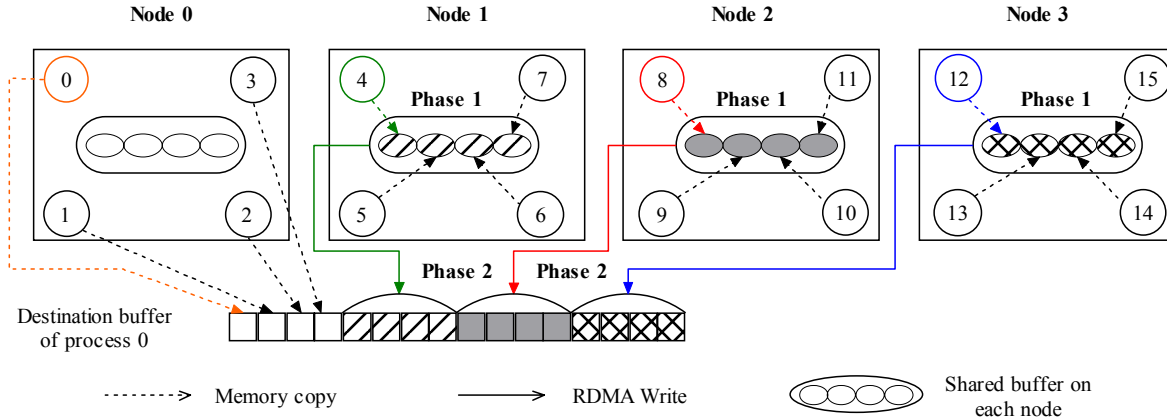
**6.1.4. Implementation Issues.** For the inter-node communications, our traditional and SMP-aware algorithms use RDMA Write on both rails. A sending process has direct control in sending messages simultaneously over the two rails, using *elan\_doput()*. When a message is larger than a threshold (1KB), even message striping is used. When a message is sent, the sending process uses the *elan\_wait()* to make sure the user buffer can be re-used safely.

Quadrics support event notification for both single-rail and multi-rail systems. The destination event (*devent*) is set once in each rail. A target process may 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 order in its entirety. It is worth mentioning QsNet<sup>II</sup> does not need memory registration and address exchange for message transfers. This eases the implementation, and effectively reduces the communication latency.

## 6.2. SMP-aware All-gather Algorithms

In the SMP-aware algorithms, we distinguish between the intra-node and inter-node communications. However, we do not just simply replace the intra-node communications in the traditional algorithms with shared memory communications. We propose two classes of SMP-aware all-gather algorithms. In the first class, we essentially do an SMP-aware gather algorithm across all processes in the system and then broadcast the gathered data to all processes, hence the name *SMP-aware Gather and Broadcast* algorithm.

In the second class, we adapt the traditional multi-port *Direct* and *Bruck* all-gather algorithms to SMP clusters by performing them across the SMP nodes rather than processes. We also do shared memory gather and broadcast operations within the nodes. We call these algorithms *SMP-aware Direct/Bruck* algorithms.



**Figure 3. Phase 1 and 2 of the SMP-aware Gather and Broadcast on a four 4-way SMP cluster**

**6.2.1. SMP-aware Gather and Broadcast Algorithm.** This algorithm is essentially done in three phases as follows:

- Phase 1:** Per-node shared memory gather
- Phase 2:** Inter-node gather among the Master processes (Tree-based or Direct)
- Phase 3:** Broadcasting gathered data to all processes

Figure 3 shows Phase 1 and Phase 2 of this algorithm for a cluster of four 4-way SMP nodes. Without loss of generality, we assume process 0 is the root process. We choose the first process of each node as the local Master process, in this case processes 0, 4, 8, and 12. In Phase 1, a local shared memory gather is done among the processes of each node. The size of the shared memory buffer is equal to the number of local processes times the message size. Each process has a shared memory flag. Local processes concurrently copy their data, using *memcpy()*, to the corresponding locations in the shared buffer, and then set their own shared memory flag. The Master process polls on all the local flags and will move on to Phase 2 once all flags are set. Note the optimization for node 0 in Figure 3.

In Phase 2, the Master processes involve in a Direct or tree-based inter-node gather operation. For instance, in a Direct inter-node gather algorithm, each Master writes the contents of its local shared memory to the corresponding position in the final destination buffer of the root process. Messages from different Masters are sent on different rails with message striping using RDMA Write. At the end, all processes synchronize using *elan\_hgsync()*, and move on to Phase 3 where the root process broadcasts the gathered data to all processes using QsNet<sup>II</sup> hardware broadcast primitive.

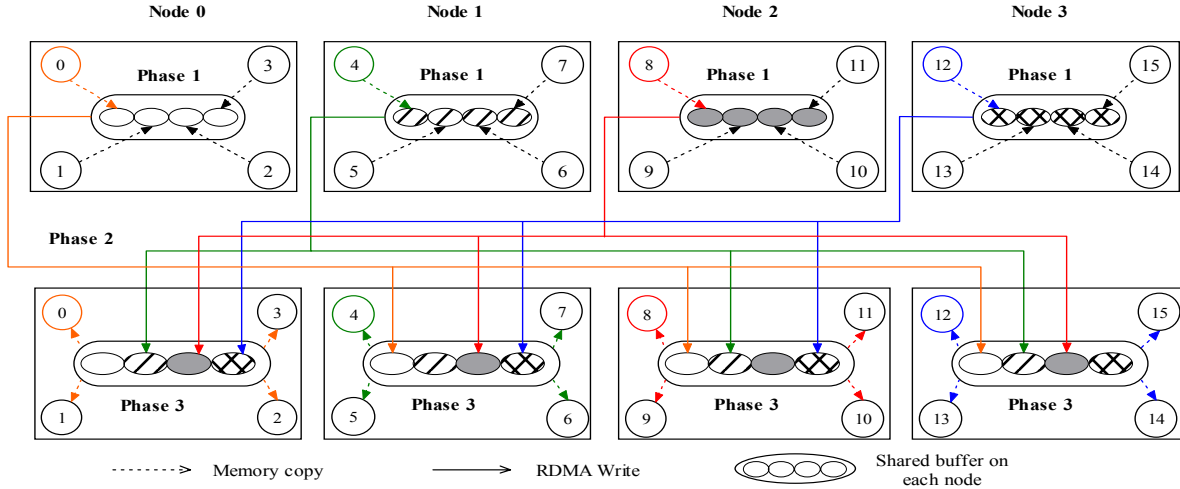
Our *SMP-aware Gather (Direct) and Broadcast* algorithm in principle is similar to the all-gather algorithm in *elan\_gather()* for short messages. Our

algorithm is host-based, while Quadrics uses a single-port tree-based, NIC-based approach that does not use striping. While NIC-based techniques alleviate cache flushing problems in host-based methods, they will incur higher latencies as the NIC processor is slower than the host processors. Moreover, on-board SDRAM is a limited source in NIC-based approaches, which limits the scalability. Our algorithms are all multi-port and use striping. For instance, a 256B message with four processes per node will be merged into a 1KB message in the shared buffer. This 1KB message will then be sent in Phase 2 over the two rails using striping in our design. This is not the case for Quadrics.

**6.2.2. SMP-aware Direct/Bruck Algorithms.** The *SMP-aware Direct* or *Bruck* all-gather algorithms can be done in three steps as follows:

- Phase 1:** Per-node shared memory gather
- Phase 2:** Inter-node all-gather among the Master processes (Direct or Bruck)
- Phase 3:** Per-node shared memory broadcast

Figure 4 shows the *SMP-aware Direct* all-gather algorithm on a quad 4-way SMP cluster. In Phase 1, each SMP node does a shared memory gather operation. However, the size of the shared buffer for this algorithm is four times larger than its counterpart. In Phase 2, Master processes involve in a *Direct* or *Bruck* inter-node all-gather operation. Each Master writes the gathered data in Phase 1 to the respective shared memory buffers of the other nodes using the corresponding multi-port all-gather algorithm. Each Master then waits for all *devents* to make sure it has received all the data. In Phase 3, Masters use a local shared memory broadcast to copy out the overall contents of the shared buffer to the destination buffers of each process. A final synchronization among all processes completes the collective operation.



**Figure 4. SMP-Aware Direct all-gather algorithm on a cluster of four 4-way SMP nodes**

In Phase 2, right after we post the RDMA Write operations, we copy the messages in the shared buffer, which have been deposited by local processes, to the destination buffers. This way, we overlap some memory copy operations in Phase 3 with the inter-node communication in Phase 2. Meanwhile, at the end of Phase 2 of the *SMP-aware Bruck* algorithm, all data is available in the shared buffer, however data is not in the right order. Instead of doing a local memory shift, we copy each message from the shared buffer to the right position of the destination buffer for every process.

## 7. Performance Evaluation

In this section, we show the performance of the multi-port algorithms in Section 6 when they are implemented directly at the Elan layer using RDMA write and shared memory communication over dual-rail QsNet<sup>II</sup> clusters with message striping support. Figure 5 compares the performance and scalability of seven different algorithms running with 16 processes on our four 4-way SMP cluster. In our tests, we have increased the shared memory buffer size to 64KB in order to find out the cut-off points among different algorithms.

By closely analyzing the results in Figure 5, one can realize that the *SMP-aware Gather and Broadcast* algorithm is the best algorithm for up to 256B messages. Its performance is in par or slightly better than the native *elan\_gather()*. For short to medium size messages (512B to 8KB), the *SMP-aware Bruck* algorithm outperforms all other algorithms. This confirms our finding in Section 5.1, where per-node shared-memory gather implementation outperformed *elan\_gather()* for up to 8KB messages. An

improvement of up to 1.96 for 4KB message can be observed using the *SMP-aware Bruck* algorithm.

As expected, traditional algorithms for flat systems (*Direct*, *Standard Exchange* and *Bruck*) cannot compete with SMP-aware algorithms for short to medium size messages. However, for medium to large messages (16KB to 1MB), the *Direct* algorithm is superior among all algorithms, gaining an improvement of up to 1.49 for 32KB messages, benefiting from large messages and multi-rail QsNet<sup>II</sup>.

Our platform represents a small cluster. However, the scalability plots in Figure 5 verify the superiority of our algorithms for various message sizes. We have considered 4, 8, and 16 processes in our scalability analysis, where processes are evenly distributed over the nodes. This nicely resembles clusters of four uni-processor nodes, dual-processor nodes, and quad-processor nodes, respectively.

## 8. Conclusions and Future Research

Scientific applications written in MPI typically use collective communications among the parallel processes. Quadrics MPI directly calls Elan collectives, therefore optimizing Elan collectives is crucial to the performance of MPI applications. In this work, we have proposed and evaluated a number of multi-port all-gather algorithms using RDMA and shared-memory communication over multi-rail QsNet<sup>II</sup> SMP clusters directly at the Elan level. These algorithms include the traditional *Direct*, *Standard Exchange* and *Bruck* algorithms for medium to large messages, and SMP-aware algorithms for short to medium size messages.

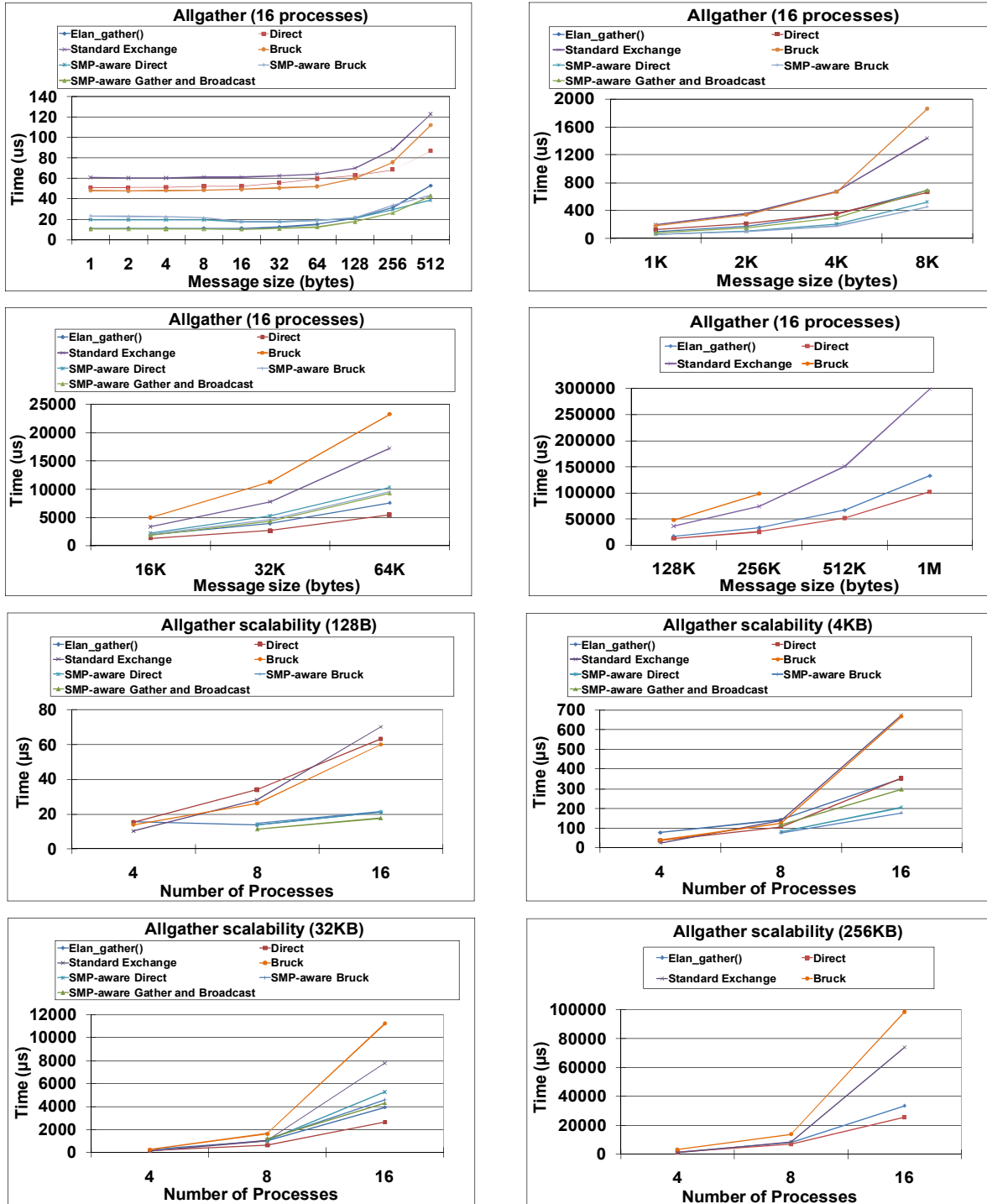


Figure 5. Performance and scalability of the proposed all-gather algorithms on a cluster of four 4-way SMP nodes with dual-rail QsNet<sup>II</sup>

For very short messages up to 256B, the *SMP-aware Gather and Broadcast* algorithm performs slightly better than the native *elan\_gather()* algorithm. Our *SMP-aware Bruck* algorithm outperforms all algorithms for short to medium size messages, from

512B to 8KB. Our multi-port *Direct* implementation outperforms *elan\_gather()* and all other algorithms for messages larger than 8KB, gaining an improvement of up to 1.49 for 32KB messages over the native *elan\_gather()*. As for future, we intend to test our

algorithms on larger systems. We plan to extend our study by devising other SMP-aware collectives over multi-rail systems.

## 9. Acknowledgments

This work is supported by the Natural Sciences and Engineering Research Council of Canada through grant RGPIN/238964-2005, Canada Foundation for Innovation's grant #7154, and Ontario Innovation Trust's grant #7154.

## 10. References

- [1] J. Beecroft, D. Addison, D. Hewson, M. McLaren, D. Roweth, F. Petrini and J. Nieplocha. QsNet<sup>II</sup>: defining high-performance network design. *IEEE Micro*, 25(4):34-47, July-Aug. 2005.
- [2] S.H. Bokhari. Multiphase complete exchange on Paragon, SP2, and CS-2. *IEEE Parallel and Distributed Technology*, 4(3):45-59, Sept. 1996.
- [3] 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.
- [4] D. Buntinas, G. Mercier and W. Gropp. Data transfers between processes in an SMP system: performance study and application to MPI", In 35<sup>th</sup> *International Conference on Parallel Processing (ICPP 2006)*, 2006.
- [5] L. Chai, A. Hartono and D.K. Panda. Designing high performance and scalable MPI intra-node communication support for clusters. In 8<sup>th</sup> *IEEE International Conference on Cluster Computing (Cluster 2006)*, 2006.
- [6] E. Chan, R. Van de Geijn, W. Gropp and R. Thakur. Collective communication on architectures that support simultaneous communication over multiple links. In 11<sup>th</sup> *ACM SIGPLAN symposium on Principles and practice of parallel programming (PPoPP'06)*, pages 2-11, 2006.
- [7] S. Coll, E. Frachtenberg, F. Petrini, A. Hoisie and L. Gurvits. Using multirail networks in high performance clusters. In 3<sup>rd</sup> *IEEE International Conference on Cluster Computing (Cluster'01)*, pages 15-24, 2001.
- [8] R. Hockney. The communication challenge for MPP, Intel Paragon and Meiko CS-2. *Parallel Computing*, 20(3):389-398, Mar. 1994.
- [9] H.-W. Jin, S. Sur, L. Chai and D.K. Panda. LiMIC: support for high-performance MPI intra-node communication on Linux clusters", In 34<sup>th</sup> *International Conference on Parallel Processing (ICPP 2005)*, 2005.
- [10] 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)*, 2004.
- [11] A.R. Mamidala, L. Chai, H.-W. Jin and D.K. Panda. Efficient SMP-aware MPI-level broadcast over InfiniBand's hardware multicast", In 6<sup>th</sup> *Workshop on Communication Architecture for Clusters (CAC 2006)*, 2006.
- [12] A.R. Mamidala, A. Vishnu and D.K. Panda. Efficient shared memory and RDMA based design for MPI-allgather over InfiniBand. In *EuroPVM/MPI 2006*, pages 66-75, 2006.
- [13] MPI: A Message-Passing Interface standard, 1997.
- [14] PDSH: available: <http://www.llnl.gov/linux/pdsh/>
- [15] Y. Qian and A. Afsahi. Efficient RDMA-based multi-port collectives on multi-rail QsNet<sup>II</sup> clusters. In 6<sup>th</sup> *Workshop on Communication Architecture for Clusters (CAC 2006)*, 2006.
- [16] Y. Qian and A. Afsahi. High performance RDMA-based multi-port all-gather on multi-rail QsNet<sup>II</sup>. In 21<sup>st</sup> *International Symposium on High Performance Computing Systems and Applications (HPCS 2007)*, 2007.
- [17] H. Ritzdorf and J.L. Traff. Collective operations in NEC's high-performance MPI libraries. In 20<sup>th</sup> *International Parallel and Distributed Processing Symposium (IPDPS'06)*, 2006.
- [18] D. Roweth and D. Addison. Optimized gather collectives on QsNet<sup>II</sup>. In *EuroPVM/MPI 2005*, pp. 407-414, 2005.
- [19] D. Roweth, A. Pittman and J. Beecroft. Optimized collectives on QsNet<sup>II</sup>. *Quadrics White Paper*, Available: <http://www.quadrics.com/>.
- [20] S. Sistare, R. vandeVaart and E. Loh. Optimization of MPI collectives on clusters of large-scale SMPs. In 1999 *ACM/IEEE Conference on Supercomputing (SC'99)*, 1999.
- [21] 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.
- [22] S. Sur, H.-W. Jin, and D.K. Panda. Efficient and scalable all-to-all personalized exchange for InfiniBand clusters. In 33<sup>rd</sup> *International Conference on Parallel Processing (ICPP'04)*, pages 275-282, 2004.
- [23] 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.
- [24] V. Tipparaju and J. Nieplocha. Optimizing all-to-all collective communication by exploiting concurrency in modern networks. In 2005 *ACM/IEEE Conference on Supercomputing (SC'05)*, 2005.
- [25] V. Tipparaju, J. Nieplocha and D.K. Panda. Fast collective operations using shared and remote memory access protocols on clusters. In 17<sup>th</sup> *International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.
- [26] J.L. Traff. Efficient allgather for regular SMP-clusters. In *EuroPVM/MPI 2006*, page 58-65, 2006.
- [27] S.S. Vadhiyar, G.E.Fagg and J. Dongarra. Automatically tuned collective communications. In 2000 *ACM/IEEE Conference on Supercomputing (SC2000)*, 2000.
- [28] M. Wu, R.A. Kendall and K. Wright. Optimizing collective communications on SMP clusters. In 34<sup>th</sup> *International Conference on Parallel Processing (ICPP 2005)*, pages 399- 407, 2005.