Cluster Comput (2008) 11: 341–354
DOI 10.1007/s10586-008-0065-8

Efficient shared memory and RDMA based collectives on multi-rail QsNet^{II} SMP clusters

Ying Qian · Ahmad Afsahi

Received: 14 January 2008 / Accepted: 24 September 2008 / Published online: 16 October 2008

Abstract Clusters of Symmetric Multiprocessors (SMP) are more commonplace than ever in achieving high-performance. Scientific applications running on clusters employ collective communications extensively. Shared memory communication and Remote Direct Memory Access (RDMA) over multi-rail networks are promising approaches in addressing the increasing demand on intra-node and inter-node communications, and thereby in boosting the performance of collectives in emerging multi-core SMP clusters. In this regard, this paper designs and evaluates two classes of collective communication algorithms directly at the Elan user-level over multi-rail Quadrics QsNet^{II} with message striping: 1) RDMA-based traditional multi-port algorithms for gather, all-gather, and all-to-all collectives for medium to large messages, and 2) RDMA-based and SMP-aware multi-port all-gather algorithms for small to medium size messages.

The multi-port RDMA-based Direct algorithm for gather and all-to-all collectives gain an improvement of up to 2.15 for 4 KB messages over *elan_gather()*, and up to 2.26 for 2 KB messages over *elan_alltoall()*, respectively. For the all-gather, our SMP-aware Bruck algorithm outperforms all other all-gather algorithms including *elan_gather()* for 512 B to 8 KB messages, with a 1.96 improvement factor for 4 KB messages. Our multi-port Direct all-gather is the best algorithm for 16 KB to 1 MB, and outperforms *elan_gather()* by a factor of 1.49 for 32 KB messages. Experimentation with real applications has shown up to 1.47 communication speedup can be achieved using the proposed all-gather algorithms.

Keywords Collective communications · RDMA · Multi-rail networks · Clusters · Shared-memory · SMP

Y. Qian · A. Afsahi (✉)
Department of Electrical and Computer Engineering, Queen's University, Kingston, ON, Canada K7L 3N6
e-mail: ahmad.afsahi@queensu.ca

Y. Qian
e-mail: ying.qian@ece.queensu.ca

1 Introduction

SMP Clusters are the predominant platforms for high-performance computing due to their cost-performance effectiveness. Interconnection networks and the communication system software play a key role on the performance of such clusters. In this regard, several modern networks such as Quadrics [2], InfiniBand [12], Myrinet [19], and 10-Gigabit iWARP Ethernet [24] have been introduced to support scalable and efficient communications. While SMP nodes are traditionally equipped with multiple single-core processors, the emergence of multi-core SMP nodes in clusters, where each core will run at least one process with multiple intra-node and inter-node connections to several other processes, will put immense pressure on the interconnection network and its communication system software.

Scientific applications written in *Message Passing Interface* (MPI) [18] typically use communication patterns that involve collective data movement and global control. Quadrics QsNet^{II} [2] has hardware support for broadcast and barrier operations. In addition, 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.

To overcome bandwidth limitations and to enhance fault tolerance, using multiple independent networks known as *multi-rail* networks [8, 15] is very promising. Quadrics multi-rail QsNet^{II} network is constructed using multiple network interface cards (NICs) per node, in which the i-th NIC connects to the i-th rail. 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. Quadrics uses an even message striping over its multi-rail QsNet^{II} network for point-to-point messages through its *Elan Remote Direct Memory Access* (RDMA) *put* and *get*, SHMEM [9] *put* and *get*, and *Tports* send and receive functions. 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. Previous studies [22, 23] 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 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 [5, 6, 13]. Some work has also been reported on devising collective algorithms for SMP clusters [16, 17, 29, 34, 35, 37].

We are interested in the design and efficient implementation of gather, all-gather, and all-to-all operations over multi-rail QsNet^{II} SMP Clusters. All-gather and all-to-all are intensive operations, typically used in linear algebra operations, matrix multiplication, matrix transpose, as well as multi-dimensional FFT. In this work, we first argue that RDMA-based traditional multi-port algorithms for gather, all-gather, and all-to-all collectives, directly implemented at the Elan level outperform the native Quadrics implementations for medium to large messages. Our RDMA-based multi-port gather algorithms include a tree-based and a *Direct* algorithm. The all-gather and all-to-all algorithms include the *Direct, Standard Exchange* [3], and *Bruck* [4] 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 performance results indicate that the multi-port Direct algorithms for gather and all-gather gain an improvement of up to 2.15 for 4 KB messages, and up to 1.49 for 32 KB messages over the native *elan_gather()*, respectively. In addition, the multi-port Direct all-to-all performs better than the native *elan_alltoall()* by a factor of 2.26 for 2 KB messages.

To address the deficiency of the traditional algorithms for short messages, we then take on the challenge to design and implement efficient RDMA-based and SMP-aware multi-rail all-gather collective algorithms, as a case study. Our algorithms include an *SMP-aware Gather and Broadcast* algorithm, as well as *SMP-aware Direct/Bruck* algorithms for small to medium size messages. Some of the proposed algorithms overlap intra-node and inter-node communications [17, 34, 37], and use multiple outstanding RDMA operations to exploit concurrency [33]. Moreover, data buffers are shared between inter-node and intra-node communications. Our SMP-aware algorithms not only improve the performance over the native implementation, but also outperform the aforementioned traditional algorithms even for medium size messages. In essence, our performance results indicate that the proposed SMP-aware Bruck all-gather is superior among all algorithms for 512 B to 8 KB messages, and gains an improvement of up to 1.96 for 4 KB messages over the native *elan_gather()*. Not to mention, the Direct all-gather algorithm is still the best algorithm for 16 KB to 1 MB messages. Using the proposed all-gather algorithms, we have been able to reduce the communication time of real applications by a factor of up to 1.47.

The rest of this paper is organized as follows. Related work is discussed in Sect. 2. In Sect. 3, we provide an overview of the Quadrics QsNet^{II} and its gather, all-gather, and all-to-all collective algorithms. We also elaborate on the communication cost model used in the paper. Section 4 discusses the motivation behind this work. Section 5 presents the RDMA-based multi-port traditional algorithms for gather, all-gather and all-to-all collective operations, and then proposes two RDMA-based and SMP-aware multi-port all-gather algorithms. Experimental framework consisting of a 16-processor dual-rail QsNet^{II} SMP cluster is described in Sect. 6. In Sect. 7, we evaluate the performance of the proposed algorithms. Section 8 concludes the paper and plans for future work.

2 Related work

Study of collective communications has been an active area of research. Thakur and his colleagues discussed recent collective algorithms used in MPICH [32]. They have shown some algorithms perform better depending on the message size and the number of processes involved in the operation. In [36], 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. In [21], Pjesivac-Grbovic and others analyzed the performance of collective communication operations under different communication cost models.

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^{II} [26, 27]. Sur et al. proposed efficient RDMA-based all-to-all broadcast and all-to-all personalized exchange algorithms for InfiniBand-based clusters [30, 31]. In [33], Tipparaju and Nieplocha used the concurrency available in modern networks to optimize MPI_Allgather operation on InfiniBand and QsNet^{II}, but their work did not address shared memory intra-node communication.

On multi-rail systems, Coll and his colleagues [8] did a comprehensive simulation study on static and dynamic allocation schemes for multi-rail systems. Liu and others [15] designed an MPI-level multi-rail InfiniBand. However, their work was only focused on point-to-point communications. Chan et al. [7] implemented a number of multi-port MPI collectives for IBM Blue Gene/L.

On SMP clusters, some recent work has been devoted to improve the performance of intra-node communications on SMPs [5, 6, 13]. Buntinas et al. [5] have used shared buffers, message queues, Ptrace system call, kernel copy, and NIC loopback mechanisms to improve large data transfers in SMP systems. In [6], Chai and others improved the intra-node communication by using the system cache efficiently, and requiring no locking mechanisms. In [13], Jin et al. implemented a potable 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 [29]. In [34], Tipparaju and others overlapped the shared memory intra-node and remote memory access inter-node communications in devising collectives for IBM SP. A leader-base scheme was proposed in [16] to improve the performance of broadcast over InfiniBand. In [37], 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 [35], Traff devised an optimized all-gather algorithm for SMP clusters. Ritzdorf and Traff [25] used similar techniques in enhancing NEC's MPI collectives. Mamidala et al. [17] designed all-gather over InfiniBand using shared memory for intra-node and single-port recursive doubling algorithm for inter-node communication via RDMA.

3 Background

3.1 Overview of quadrics QsNet^{II}

QsNet^{II} is the latest generation interconnect from Quadrics used in high-performance clusters. It differs from other contemporary interconnects in the sense that it integrates a node's local memory into a globally shared, virtual-memory space. QsNet^{II} supports both *Send/Recv* and RDMA operations. The RDMA operations include RDMA write (*elan_put*), and RDMA read (*elan_get*). The send/receive mode, supported through a connectionless network programming interface called *Tagged Ports* (Tports), provides a similar two-sided message-passing semantics as in MPI. In fact, Quadrics MPI point-to-point implementation uses Tports as its transport layer. As stated earlier, Quadrics multi-rail networks boost the point-to-point bandwidth using even message striping.

QsNet^{II} supports hardware broadcast and barrier, where possible. It also implements reduce, gather, all-gather and all-to-all at the Elan level. *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 [26]. 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. In SMP clusters, data (up to 2 KB) 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 4 KB, Tports Send/Recv is used among all processes, which benefits from message striping in multi-rail QsNet^{II}. For the all-gather operation, *elan_gather()* uses the gather algorithm followed by broadcast for messages up to 32 KB. For larger messages, it switches to the ring algorithm.

The *elan_alloall()* does an all-to-all personalized exchange amongst all the processes. It uses a pairwise exchange algorithm for up to 8 KB messages, and then switches to a permission to send algorithm [27]. 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 the previous process. It has also been shown that the Bruck algorithm works well for very small messages. As in the gather and all-gather collectives, *elan_alltoall()* uses shared memory for messages up to 2 KB. To our knowledge, *elan-gather()* and *elan_alloall()* use single-port algorithms.

3.2 Communication cost model

To estimate the communication cost of different algorithms, we use the *Hockney's* model [11]. This is a simple model that assumes the cost of sending a message between any two processes is equivalent to $t_s + l_m \times \tau$, where t_s is the message startup time, l_m is the message size in bytes, and τ is the time to transfer one byte. The assumption is that communication between any pair of processes has the same cost (this is almost true with wormhole routing in modern networks such as QsNet^{II}), independent of any other simultaneous network transactions. Although other elegant communication cost models such as *LogP* [10], *LogGP* [1], and *PLogP* [14] exist, the Hockney's model is sufficient for our analysis.

We assume each process has the ability to simultaneously send and receive k messages on its k links, the so-called k-port (or multi-port) model. In essence, k is the number of ports in the multi-port algorithms, equal to the number of available rails. We should also point out that although this paper does not analytically model the cost associated with shared memory transactions used in the SMP-aware algorithms in Sect. 5.2, it empirically compares its performance with intra-node network transactions.

4 Motivation

4.1 Multi-rail networks

Multi-rail networks are used to boost bandwidth and to enhance fault tolerance. In this section, we first do a feasibility study of the potential performance that could be gained using message striping by presenting the performance of Elan RDMA Write, *elan_put()*, under single-rail and dual-rail QsNet^{II} on our platform, described in Sect. 6. Figure 1 presents the both-way bandwidth of the RDMA write using the *pgping* micro-benchmark available in the Elan Library. In the both-way test, both the sender and receiver send and receive data simultaneously. This is repeated sufficient number of times to eliminate the transient conditions of the network.

As shown in Fig. 1, it is clear that the bandwidth is doubled in the dual-rail QsNet^{II} network. Evidently, striping occurs at 1 KB boundary. The bandwidth for RDMA Read (*elan_get(),* not shown) is almost the same as the bandwidth for RDMA Write. The 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 RDMA Read short message latency is slightly larger than the RDMA write. Therefore, we use RDMA Write in implementing our collective algorithms. Given the multi-rail performance offered at the Elan level, excellent opportunities exist for devising efficient multi-port collectives and utilizing message striping over multi-rail systems.

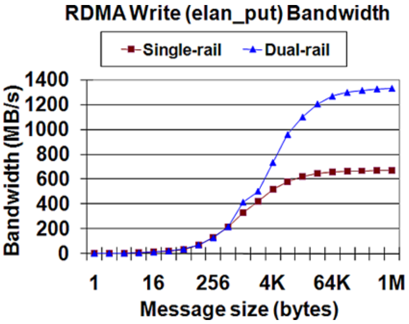


Fig. 1 Elan RDMA Write both-way bandwidth

4.2 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 [5]. 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 2 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 Fig. 2, one can conclude that shared memory implementation is the preferred method for intra-node communication, but only up to 2 KB messages; afterwards, RDMA is better. We believe, in implementing collectives, this is the main reason why Quadrics uses shared memory intra-node

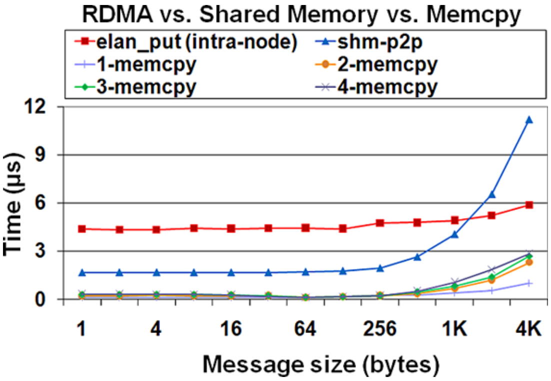


Fig. 2 Intra-node communication

communication among co-located processes only for messages smaller than 2 KB.

Prior research [5, 6] 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 Fig. 2 as all *k*-*memcpy()* operations take much less time than an intra-node RDMA operation (in fact, this is true up to 128 KB messages). Intuitively, one can argue shared memory regions can be effectively used for per-node collectives for messages larger than 2 KB as well, where they should potentially provide better performance than RDMA implementations.

Our SMP-aware all-gather algorithms in Sect. 5.2 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 Fig. 7 for node zero. 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 3 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 256 B to 32 KB 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 8 KB messages. Therefore, on our platform, we use shared memory for messages up to 8 KB. It is clear that this message size can be found empirically for other single-core/multi-core SMPs.

5 Collective algorithms

In this section, we first present the RDMA-based multi-port traditional algorithms for gather, all-gather and all-to-all collective operations, and then propose RDMA-based and SMP-aware multi-port all-gather algorithms.

5.1 RDMA-based traditional algorithms

Basically, there are two ways to improve the performance of collectives on multi-rail systems. One is to implement single-port algorithms that gain multi-rail striping from the underlying communication subsystem. This approach is currently used by Quadrics for some of the collectives. The other approach that we propose is to design and adapt multi-port algorithms that could also benefit from striping. In the following, we provide an overview of well-known traditional algorithms for gather, all-gather, and all-to-all collectives. By traditional algorithms, we mean algorithms that are designed for flat systems, and executed across all processes in the system. We assume N is the number of processes involved in a collective operation.

5.1.1 Gather algorithms

In gather, the root process gathers the data contributed by every other process. We use two multi-port gather algorithms: *Spanning Binomial Tree*, and *Direct* algorithms.

Spanning binomial tree algorithm The spanning binomial tree algorithm, used for a scatter operation, extended for k-port modeling can be used for the gather operation. Communications start from the leaf processes and messages are combined in the intermediate processes until they reach the root process. At each intermediate step, the length of a message sent to the parent process is multiplied by $(k+1)$. The

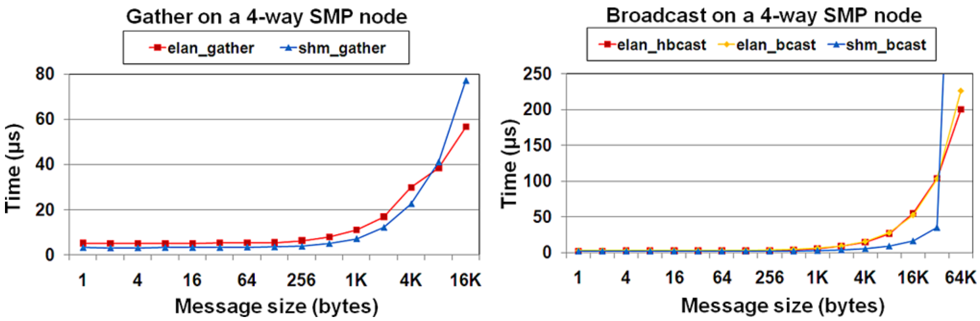


Fig. 3 Intra-node gather and broadcast

gather operation is completed after $\lceil \log_{k+1} N \rceil$ steps. Therefore, the total communication cost, T, is:

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

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

Direct algorithm The tree-based 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 medium to large messages, is for each process to send its message directly to the root process. At each step, the root process receives k different messages from k other processes. There are a total of $\lceil \frac{N-1}{k} \rceil$ communication steps. Therefore, the total communication cost, T, is:

$$T = \left\lceil \frac{N-1}{k} \right\rceil \times (t_s + l_m \times \tau). \quad (2)$$

5.1.2 All-gather algorithms

In all-gather, each process will gather the data contributed by all processes. We study three multi-port all-gather algorithms: *Direct, Standard Exchange* [3], and *Bruck* [4].

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. That is, at step i, process p sends its message to processes $(p + (i-1)k + 1) \bmod N, (p + (i-1)k + 2) \bmod N, \ldots, (p + ik) \bmod N$. There are a total of $\lceil \frac{N-1}{k} \rceil$communication steps. The total communication cost is the same as (2).

Standard exchange algorithm The Standard Exchange all-gather algorithm is the extension of *recursive doubling* algorithm for k-port modeling, and works for power of $(k+1)$ processes. It should generally perform well for short 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 $(0, 1, \ldots, k), (k+1, k+2, \ldots, 2(k+1)-1), \ldots, (N-(k+1), N-(k+1)+1, \ldots, N-1)$. 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 $(p+(k+1)) \bmod N, (p+2(k+1)) \bmod N, \ldots, (p+k(k+1)) \bmod N$. 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 the number of processes is not a power of $(k+1)$. If the number of processes is a power of $(k+1)$, the total communication time is the same as (3); otherwise (4). Figure 4 illustrates the 2-port Standard exchange algorithm for nine processes.

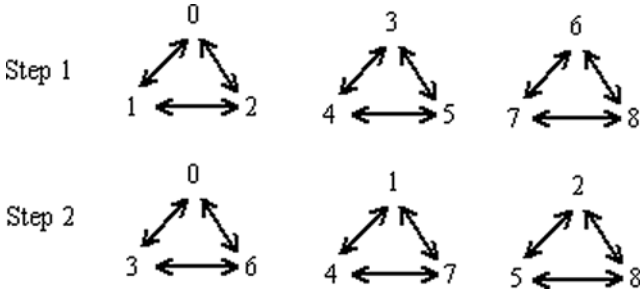


Fig. 4 Standard exchange all-gather algorithm for 9 processes under 2-port modeling

$$T = (t_s \times \log_{k+1} N) + \frac{N-1}{k}(l_m \times \tau), \quad (3)$$

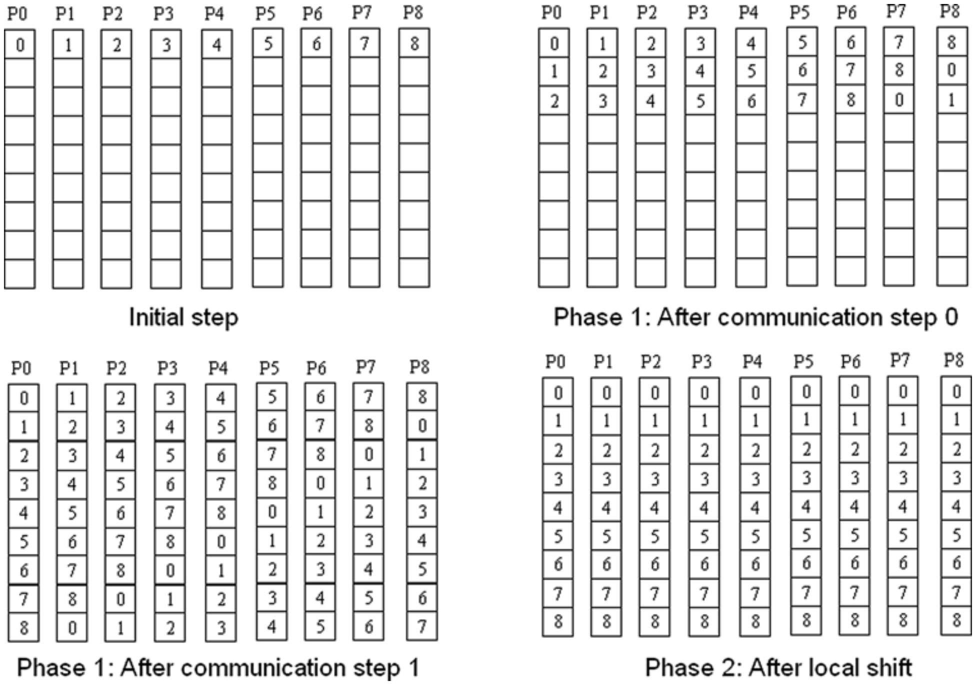
$$T = (t_s \times (k+1) \times \lfloor \log_{k+1} N \rfloor) + \frac{N-1}{k}(l_m \times \tau). \quad (4)$$

Bruck algorithm The Bruck all-gather algorithm works on any number of processes, and is proposed to improve the performance for small messages. Figure 5 illustrates the 2-port Bruck algorithm for nine processes. 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 $(p-(k+1)^i), (p-2(k+1)^i), \ldots, (p-k(k+1)^i)$, and stores the data it receives from processes $(p+(k+1)^i), (p+2(k+1)^i), \ldots, (p+k(k+1)^i)$ 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 local memory shift by p blocks downwards on a process p. The total communication cost is the same as (1), for any N.

5.1.3 All-to-all algorithms

In all-to-all, each process has a distinct data for every other process. In the following, we provide an overview of three all-to-all algorithms: *Direct, Standard Exchange* [3], and *Bruck* [4].



Initial step

P0	P1	P2	P3	P4	P5	P6	P7	P8
0	1	2	3	4	5	6	7	8

Phase 1: After communication step 0

P0	P1	P2	P3	P4	P5	P6	P7	P8
0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	0
2	3	4	5	6	7	8	0	1

Phase 1: After communication step 1

P0	P1	P2	P3	P4	P5	P6	P7	P8
0	1	2	3	4	5	6	7	8
1	2	3	4	5	6	7	8	0
2	3	4	5	6	7	8	0	1
3	4	5	6	7	8	0	1	2
4	5	6	7	8	0	1	2	3
5	6	7	8	0	1	2	3	4
6	7	8	0	1	2	3	4	5
7	8	0	1	2	3	4	5	6
8	0	1	2	3	4	5	6	7

Phase 2: After local shift

P0	P1	P2	P3	P4	P5	P6	P7	P8
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8	8

Fig. 5 Bruck all-gather algorithm for 9 processes under 2-port modeling

Direct algorithm The same grouping scheme and algorithm as the Direct all-gather is used for the Direct all-to-all. The communication cost is the same as (2).

Standard exchange algorithm The Standard Exchange algorithm for all-to-all has the same grouping scheme as the Standard Exchange algorithm for all-gather. However, each process sends $\frac{N}{k+1}$ times larger message at each step. The total communication cost is:

$$T = (t_s \times \lceil \log_{k+1} N \rceil) + \left(\frac{N}{k+1}\right) \times \log_{k+1} N \times (l_m \times \tau). \tag{5}$$

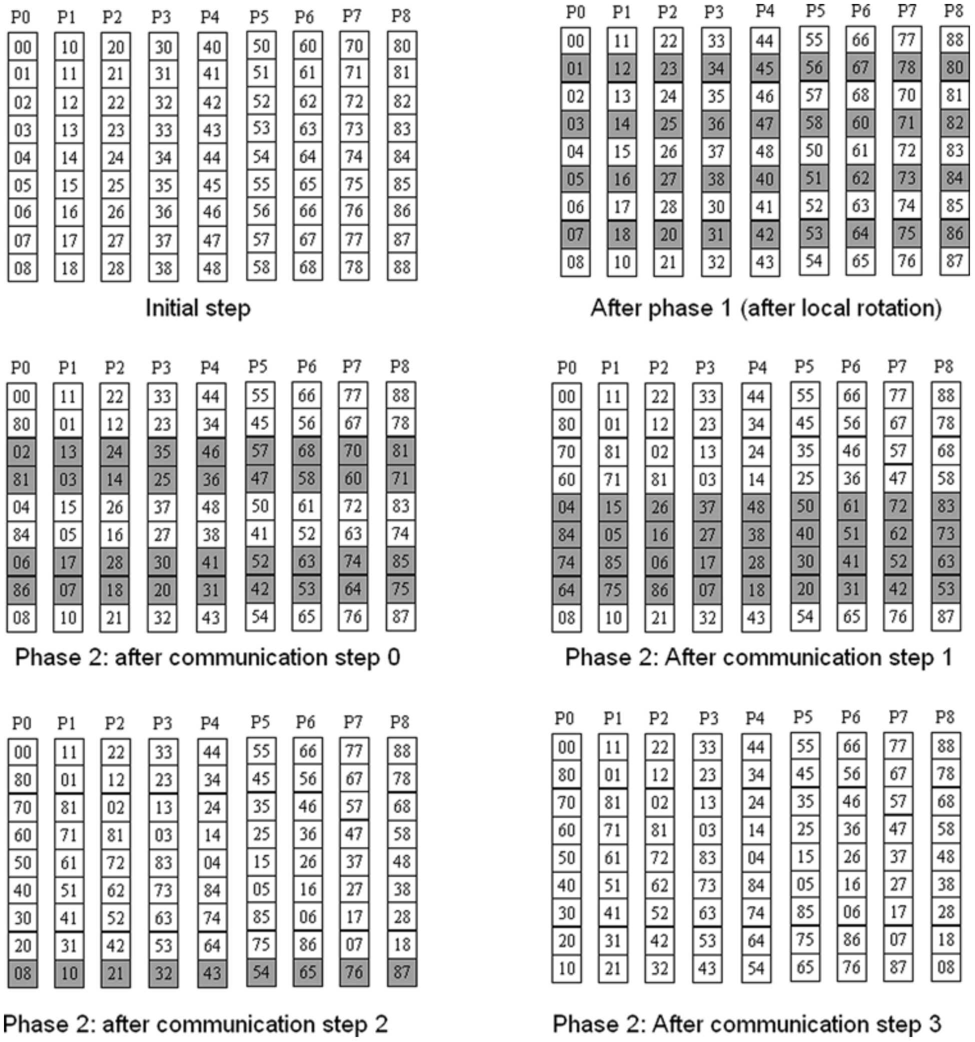
Bruck algorithm The Bruck algorithm for all-to-all among N processes can also be represented as a sequence of process-memory configurations. Each process-memory configuration has N columns of N blocks each. Columns i represents the processor P_i , and the block j represents the data j to be sent to process P_j.

Bruck algorithm for all-to-all consists of three phases. The first and the third phase only require local memory movement in each process. The first phase does a local copy and shift of the data blocks such that the data block to be sent by each processor to itself is at the top of the column. In each communication step j of the second phase, process i sends to process with rank $(i + k^j)$ (with wrap-around) all those data blocks whose j-th bit is 1. It also receives data from process with rank $(i - k^j)$, and stores them into blocks whose j-th bit is 1. The final phase does a local inverse shift of the blocks to place the data in the right order. This algorithm takes $\lceil \log_{k+1} N \rceil$ steps. The total communication cost is the same as (5), when N is a power of $(k + 1)$. Figure 6 shows an example with four communication steps for nine nodes.

5.2 RDMA-based and SMP-aware all-gather algorithms

While the performance of the traditional algorithms in Sect. 5.1 is excellent for medium to large messages (refer to Sect. 7), they lag behind the native QsNet$^{\text{II}}$ implementations for very short to medium size messages. Therefore, as a case study in this section, we propose efficient SMP-aware all-gather algorithms that not only do they address this deficiency, but also they outperform the traditional multi-port all-gather algorithms for medium size messages.

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. Instead, we adapt the traditional multi-port (Direct and tree-based) gather and (Direct and Bruck) all-gather algorithms to SMP clusters by performing them across the SMP nodes rather than processes. 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 use shared memory gather and broadcast operations within the nodes, along with (Direct and Bruck) all-gather among the nodes. We call these algorithms *SMP-aware Direct/Bruck* algorithms.



P0	P1	P2	P3	P4	P5	P6	P7	P8
00	10	20	30	40	50	60	70	80
01	11	21	31	41	51	61	71	81
02	12	22	32	42	52	62	72	82
03	13	23	33	43	53	63	73	83
04	14	24	34	44	54	64	74	84
05	15	25	35	45	55	65	75	85
06	16	26	36	46	56	66	76	86
07	17	27	37	47	57	67	77	87
08	18	28	38	48	58	68	78	88

Initial step

P0	P1	P2	P3	P4	P5	P6	P7	P8
00	11	22	33	44	55	66	77	88
01	12	23	34	45	56	67	78	80
02	13	24	35	46	57	68	70	81
03	14	25	36	47	58	60	71	82
04	15	26	37	48	50	61	72	83
05	16	27	38	40	51	62	73	84
06	17	28	30	41	52	63	74	85
07	18	20	31	42	53	64	75	86
08	10	21	32	43	54	65	76	87

After phase 1 (after local rotation)

P0	P1	P2	P3	P4	P5	P6	P7	P8
00	11	22	33	44	55	66	77	88
80	01	12	23	34	45	56	67	78
02	13	24	35	46	57	68	70	81
81	03	14	25	36	47	58	60	71
04	15	26	37	48	50	61	72	83
84	05	16	27	38	41	52	63	74
06	17	28	30	41	52	63	74	85
86	07	18	20	31	42	53	64	75
08	10	21	32	43	54	65	76	87

Phase 2: after communication step 0

P0	P1	P2	P3	P4	P5	P6	P7	P8
00	11	22	33	44	55	66	77	88
80	01	12	23	34	45	56	67	78
70	81	02	13	24	35	46	57	68
60	71	81	03	14	25	36	47	58
04	15	26	37	48	50	61	72	83
84	05	16	27	38	40	51	62	73
74	85	06	17	28	30	41	52	63
64	75	86	07	18	20	31	42	53
08	10	21	32	43	54	65	76	87

Phase 2: After communication step 1

P0	P1	P2	P3	P4	P5	P6	P7	P8
00	11	22	33	44	55	66	77	88
80	01	12	23	34	45	56	67	78
70	81	02	13	24	35	46	57	68
60	71	81	03	14	25	36	47	58
50	61	72	83	04	15	26	37	48
40	51	62	73	84	05	16	27	38
30	41	52	63	74	85	06	17	28
20	31	42	53	64	75	86	07	18
08	10	21	32	43	54	65	76	87

Phase 2: after communication step 2

P0	P1	P2	P3	P4	P5	P6	P7	P8
00	11	22	33	44	55	66	77	88
80	01	12	23	34	45	56	67	78
70	81	02	13	24	35	46	57	68
60	71	81	03	14	25	36	47	58
50	61	72	83	04	15	26	37	48
40	51	62	73	84	05	16	27	38
30	41	52	63	74	85	06	17	28
20	31	42	53	64	75	86	07	18
10	21	32	43	54	65	76	87	08

Phase 2: After communication step 3

Fig. 6 Bruck all-to-all algorithm for 9 processes under 2-port modeling (phase 3 not shown)

5.2.1 *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 7 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 Fig. 7.

In Phase 2, the Master processes engage 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^{II} 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 proces-

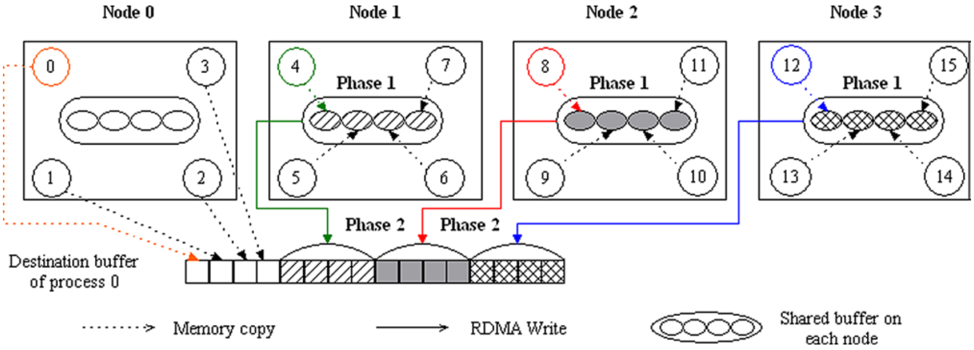


Fig. 7 Phase 1 and 2 of the SMP-aware Gather and Broadcast on a cluster of four 4-way SMP nodes

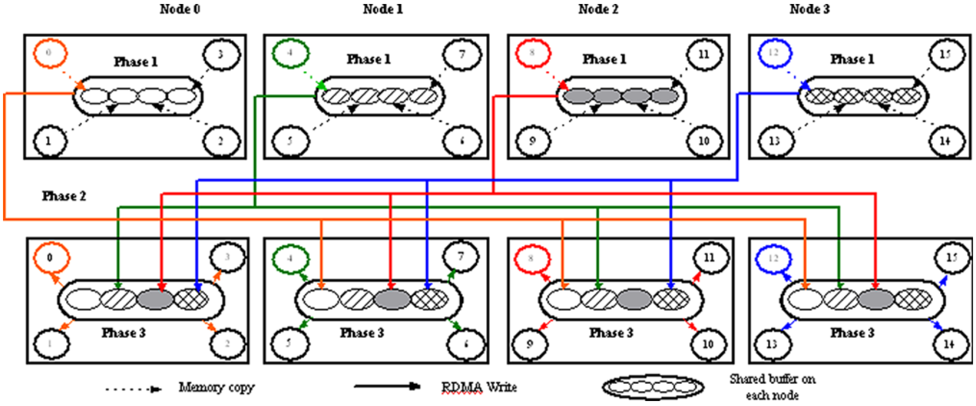


Fig. 8 SMP-aware Direct all-gather algorithm on a cluster of four 4-way SMP nodes

sor is slower than the host processor. Moreover, on-board SDRAM is a limited resource in NIC-based approaches, which limits the scalability. Our algorithms are all multi-port and use striping. For instance, a 256 B message with four processes per node will be merged into a 1 KB message in the shared buffer. This 1 KB message will then be sent in Phase 2 over the two rails using striping in our design. This is not the case for Quadrics.

5.2.2 *SMP-aware direct/Bruck algorithms*

The *SMP-aware Direct* or *Bruck* all-gather algorithm is 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 8 shows the *SMP-aware Direct* all-gather algorithm on a cluster of four 4-way SMP nodes. In Phase 1, each SMP node does a shared memory gather operation. Note that the size of the shared buffer for this algorithm is four times larger than that of the *SMP-aware Gather and Broadcast* algorithm. 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 into the respective shared memory buffers of the other nodes using the multi-port Direct or Bruck all-gather algorithm. Each Master then waits for all *devents* to make sure it has received all the data. In Phase 3, each Masters executes 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.

In Phase 2, right after we post the RDMA Write operations, we copy the messages in the shared buffer (that 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.

6 Experimental platform

The experiments were conducted on a 4-node dedicated SMP cluster, 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 256 KB unified L2 cache, 512 KB unified L3 cache, and 2 GB 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

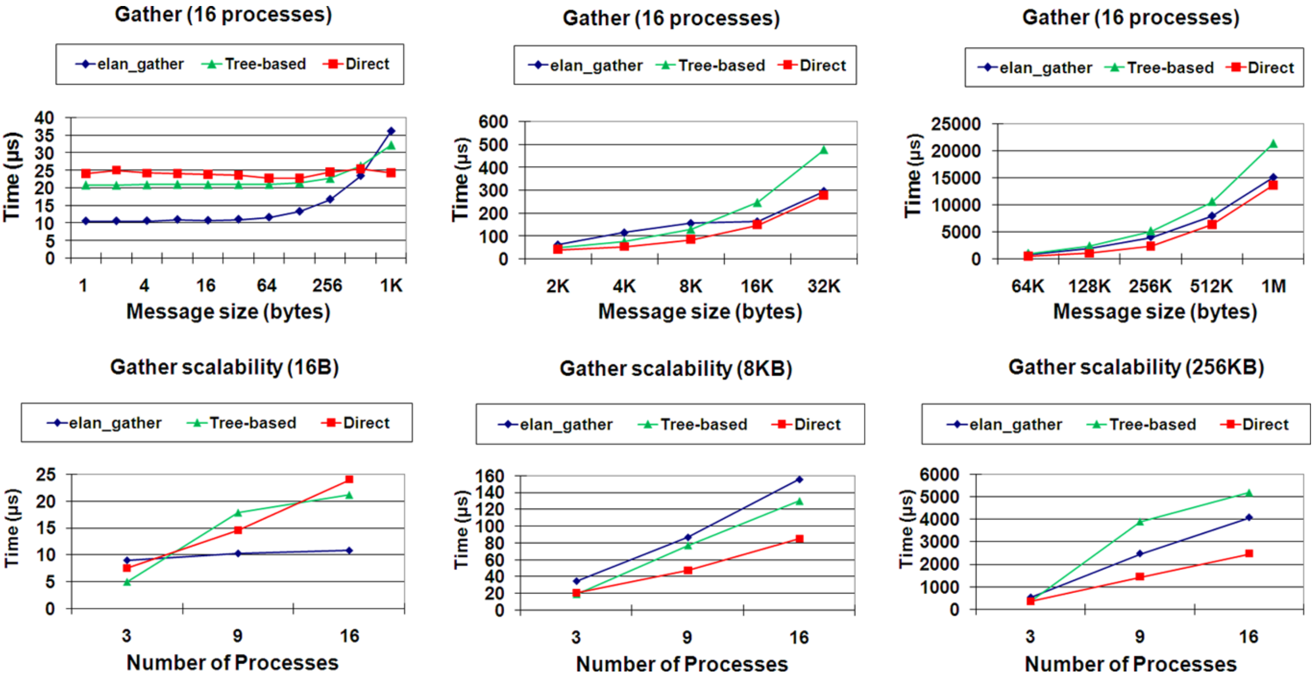


Fig. 9 Gather performance and scalability

"Hawk" release with kernel patch qsnetp2, kernel module 5.10.5qsnet, QsNet Library 1.5.9-1, and QsNet^{II} Library 2.2.11-2. Test codes were launched by the *pdsh* [20], version 2.6.1. The MPI implementation is the Quadrics MPI, version MPI.1.24-49.intel81.

7 Performance results and analysis

In this section, we first discuss the implementation issues in our study. We then present the performance of traditional multi-port gather and all-to-all algorithms on our platform. Finally, we show the performance of the traditional and the proposed SMP-aware all-gather algorithms.

7.1 Implementation issues

For the inter-node communications, both our multi-port traditional and SMP-aware algorithms use RDMA Write on both available 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 (1 KB), 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. Note that in the implementation of our algorithms, processes do not synchronize with each other.

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 that QsNet^{II} does not need memory registration and address exchange for message transfers. This eases the implementation, and effectively reduces the communication latency.

7.2 Gather performance

The tree-based and the Direct multi-port gather algorithms have been implemented using RDMA Write, and their performance is compared with *elan_gather()* in Fig. 9. Comparing the two algorithms, as expected, the tree-based algorithm has a better performance for short messages (up to 256 bytes) than the Direct algorithm.

The overall results are very promising as the implementation of the dual-rail Direct algorithm is much better than the native implementation except for messages less than 1 KB. In fact, the proposed multi-port Direct gather gains an improvement of up to 2.15 for 4 KB message. An interesting observation is the superiority of the tree-based algorithm over *elan_gather()* for the messages in 1 KB to 8 KB range. We believe this is because *elan_gather()* uses a single-rail tree-based algorithm for medium size messages. The scalability plots in Fig. 9 confirm our findings in the 16-process case. However, it also shows that with increasing number of processes *elan_gather()* performs better for very short messages. This is because Quadrics uses shared memory for intra-node communications.

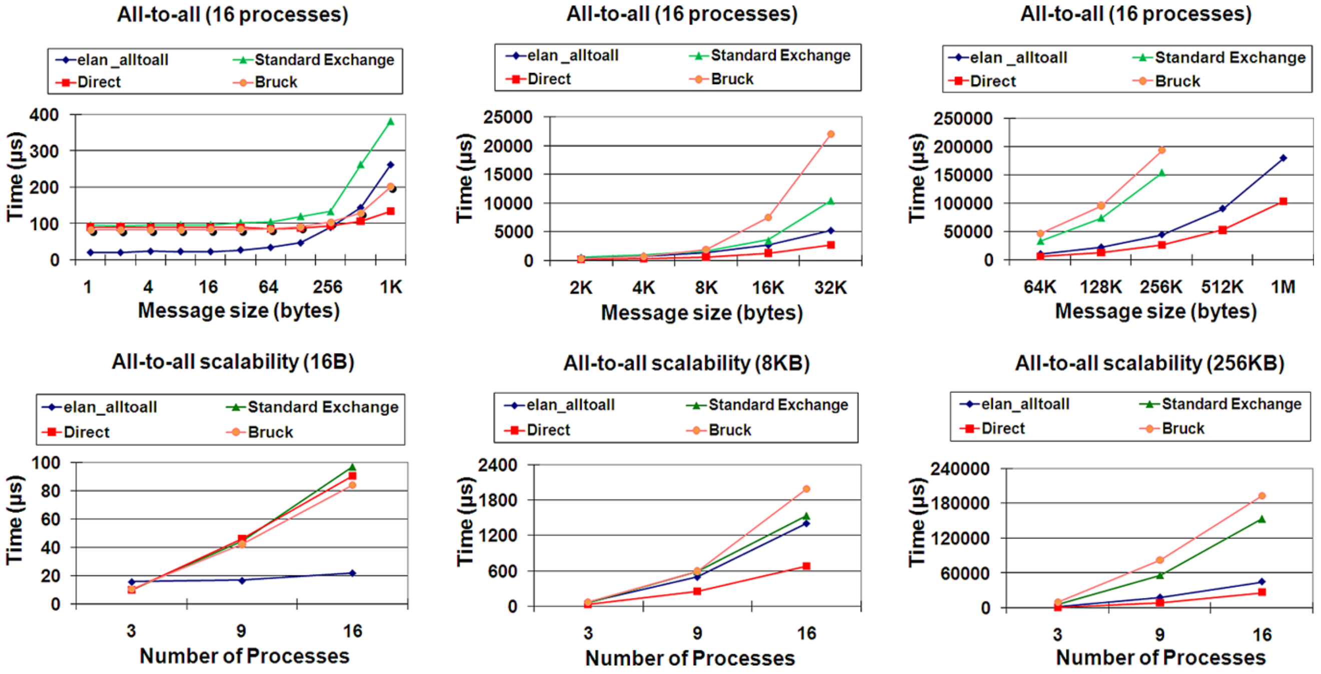


Fig. 10 All-to-all performance and scalability

7.3 All-to-all performance

Figure 10 compares the performance of the three multi-port all-to-all algorithms (Direct, Standard exchange, and Bruck) with the *elan_alltoall()* when they are implemented directly at the Elan layer using RDMA Write on our dual-rail QsNet^{II} SMP cluster. As expected, the Bruck algorithm is superior among the three traditional algorithms for very short messages (up to 64 bytes), while the Direct algorithm has a much better performance for short to large messages. The Bruck algorithm suffers because of memory copies when the message size increases. The Standard Exchange algorithm incurs a penalty for 16 processes due to correction steps. However, its performance is better than the Bruck algorithm after 4 KB.

The performance of our multi-port Direct all-to-all algorithm is much better than the native *elan_alltoall()* for messages larger than 256 bytes. In fact, an improvement of up to a factor of 2.26 for 2 KB messages has been observed. The scalability plots in Fig. 10 shows that with increasing number of processes *elan_alltoall()* outperforms our algorithms for very short messages, essentially confirming the importance of intra-node shared-memory communications for very small messages, which is lacking in our flat algorithms.

7.4 All-gather performance

In this section, we compare the performance of the proposed multi-port SMP-aware all-gather algorithms with the traditional (Direct, Standard exchange, and Bruck) all-gather algorithms and the *elan_gather()*. Figure 11 presents the performance and scalability of seven different algorithms running with 16 processes on our cluster. We should mention that we have increased the shared memory buffer size to 64 KB in order to find out the cut-off points among the different algorithms. We have considered 4, 8, and 16 processes in our scalability analysis, where processes are evenly distributed across the nodes. This resembles clusters of four uniprocessor nodes, four dual-processor nodes, and four quad-processor nodes, respectively.

Analyzing the results in Fig. 11, one can realize that the *SMP-aware Gather and Broadcast* algorithm is the best algorithm for up to 256 B messages. Its performance is in par or slightly better than the native *elan_gather()*. For short to medium size messages (512 B to 8 KB), the *SMP-aware Bruck* algorithm outperforms all other algorithms. This is in harmony with our finding in Sect. 4.1, where per-node shared-memory gather implementation outperformed *elan_gather()* for up to 8 KB messages. Note that an improvement of up to 1.96 for 4 KB messages can be observed using the *SMP-aware Bruck* algorithm.

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

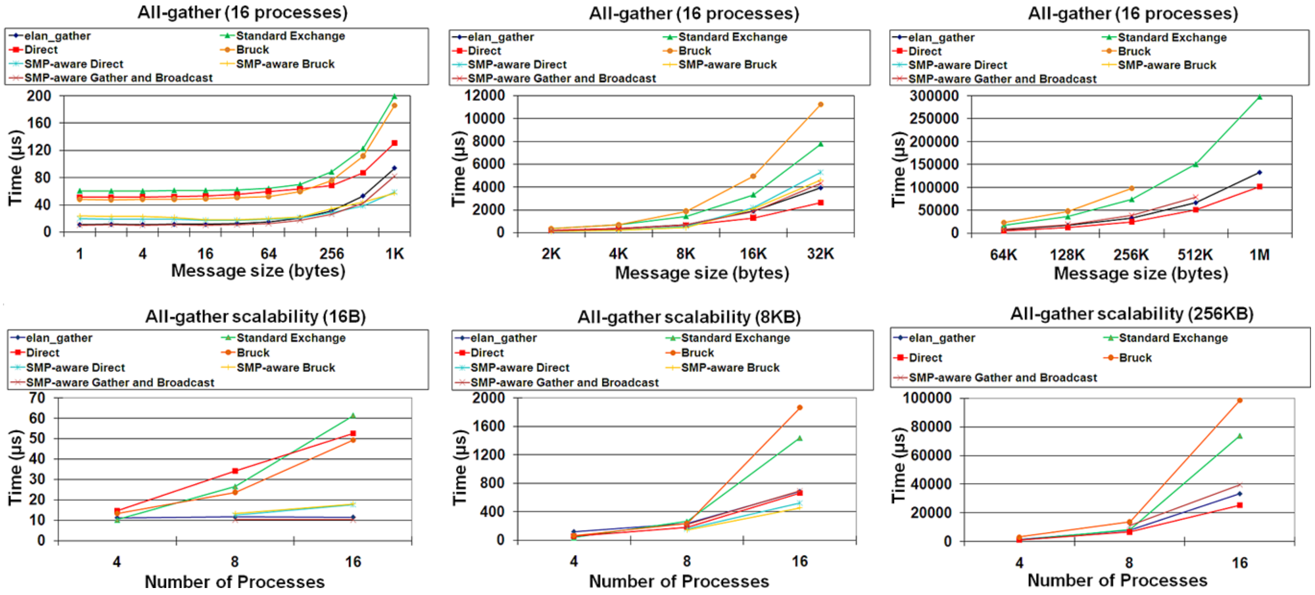


Fig. 11 All-gather performance and scalability

Table 1 Application and communication speedup (16 processes) using the proposed all-gather algorithms

	N-BODY	RADIX
Application speedup	1.01	1.13
Communication speedup	1.09	1.47

7.4.1 *Application performance*

In this section, we consider two real MPI applications, N-BODY and RADIX [28]. These applications are irregularly structured and use MPI_Allgather collective operation as well as point-to-point communications. N-BODY simulates the interaction of a system of bodies in three dimensions over a number of time steps, using the Barnes-Hut algorithm. Radix sorts a series of integer keys in ascending order using the radix algorithm.

Table 1 shows the application speedup and the communication speedup of N-BODY and RADIX running with 16 processes when using the proposed all-gather algorithms. The achieved speedups are within expectation given the size of messages that the MPI_Allgather uses in these applications. MPI_Allgather in RADIX only uses 4 KB payload, and the communication speedup of 1.47 is close to the 1.96 speedup that our *SMP-aware Bruck* algorithm achieves in our microbenchmark test. On the contrary, although N-BODY uses a larger number of MPI_Allgather collectives, 91% of the payloads are less than 64 bytes. The remaining payloads are less than 1 KB. Given our *SMP-aware Gather and Broadcast* algorithm is only slightly better than the native *elan_gather()* for messages up to 64 bytes, the 9% communication speedup for N-BODY is justified. One has to bear in mind that our microbenchmark all-gather tests at the Elan level were run under a controlled and in a synchronized fashion, while real applications typically suffer from process skew and process arrival pattern due to imbalanced computation.

8 Conclusions and future research

Scientific applications written in MPI use collective communications extensively. Quadrics QsNet^{II} is one of the high-speed interconnects used for high-performance cluster computing. Quadrics software directly implements some collectives at its Elan user-level messaging layer. Its MPI implementation uses the Elan collectives directly, and therefore optimizing collectives at the Elan level is crucial to the performance of MPI applications.

Using shared memory communication among co-located processes on SMP nodes as well as RDMA operations for inter-node communication and trying to overlap them is a promising technique in increasing the performance of collective operations. The effect is much more pronounced when efficient multi-port collectives on multi-rail networks are devised and implemented. This paper proposes and evaluates two class of collective communications algorithms directly at the Elan level over multi-rail QsNet^{II} SMP clusters with message striping: RDMA-based traditional multi-port algorithms for medium to large messages, and RDMA-based and SMP-aware multi-port algorithms for small to medium size messages.

Our RDMA-based multi-port gather algorithms include a tree-based and a Direct algorithm. The all-gather and all-to-all algorithms include the Direct, Standard Exchange, and

Bruck algorithms. Our performance results indicate that the multi-port RDMA-based Direct algorithm for gather and all-to-all collectives gain an improvement of up to 2.15 for 4 KB messages over *elan_gather()*, and up to 2.26 for 2 KB messages over *elan_alltoall()*, respectively.

For the all-gather operation, and for very short messages up to 256 B, the *SMP-aware Gather and Broadcast* algorithm performs slightly better than the native *elan_gather()*. Our *SMP-aware Bruck* algorithm outperforms all algorithms including *elan_gather()* for 512 B to 8 KB messages, with a 1.96 improvement factor for 4 KB messages. Our multi-port Direct all-gather is the best algorithm for 16 KB to 1 MB, and outperforms *elan_gather()* by a factor of 1.49 for 32 KB messages. In addition, up to 1.47 communication speedup was achieved using the proposed all-gather algorithms when running real applications.

Our platform represents a small cluster. However, the scalability results verify the superiority of the proposed algorithms for various message sizes. Although we have used Quadrics QsNet$^{\mathrm{II}}$ in this study, we believe the proposed algorithms have implications beyond this network, and can be applied directly to any other RDMA-enabled network such as InfiniBand or iWARP Ethernet. As for future work, we intend to test our algorithms on a larger cluster with large multi-core SMP nodes. We also plan to extend our study by devising other RDMA-based and multi-core/SMP-aware collectives over multi-rail systems.

Acknowledgements This work is supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) through grant RGPIN/238964-2005, Canada Foundation for Innovation (CFI) grant #7154, and Ontario Innovation Trust (OIT) grant #7154, and Ontario Graduate Scholarship (OGS).

References

1. Alexandrov, A., Ionescu, M., Schauser, K.E., Scheiman, C.: Incorporating long messages into the logp model—one step closer towards a realistic model for parallel computation. In: Proc. 7th ACM Symposium on Parallel Algorithms and Architecture (SPAA'95), 1995
2. Beecroft, J., Addison, D., Hewson, D., McLaren, M., Roweth, D., Petrini, F., Nieplocha, J.: QsNet$^{\mathrm{II}}$: defining high-performance network design. IEEE Micro **25**(4), 34–47 (2005)
3. Bokhari, S.H.: Multiphase complete exchange on Paragon, SP2, and CS-2. IEEE Parallel Distrib. Technol. **4**(3), 45–59 (1996)
4. Bruck, J., Ho, C.-T., Kipnis, S., Upfal, E., Weathersby, D.: Efficient algorithms for all-to-all communications in multiport message-passing systems. IEEE Trans. Parallel Distrib. Syst. **8**(11), 1143–1156 (1997)
5. Buntinas, D., Mercier, G., Gropp, W.: Data transfers between processes in an SMP system: performance study and application to MPI. In: Proc. 35th Int. Conf. on Parallel Processing (ICPP 2006), 2006
6. Chai, L., Hartono, A., Panda, D.K.: Designing high performance and scalable MPI intra-node communication support for clusters. In: Proc. 8th IEEE Int. Conf. on Cluster Computing (Cluster 2006), 2006
7. Chan, E., Van de Geijn, R., Gropp, W., Thakur, R.: Collective communication on architectures that support simultaneous communication over multiple links. In: Proc. 11th ACM SIGPLAN Symposium on Principles and practice of parallel programming (PPoPP'06), pp. 2–11 (2006)
8. Coll, S., Frachtenberg, E., Petrini, F., Hoisie, A., Gurvits, L.: Using multirail networks in high performance clusters. Concurr. Comput. Pract. Exp. **15**(7-8), 625–651 (2003)
9. Cray Man Page Collection: Shared Memory Access (SHMEM) S-2383-2, http://docs.cray.com/
10. Culler, D.E., Karp, R.M., Patterson, D.A., Sahay, A., Schauser, K.E., Santos, E., Subramonian, R., von Eiken, T.: LogP: towards a realistic model of parallel computation. In: Proc. 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, 1993
11. Hockney, R.: The communication challenge for MPP, Intel Paragon and Meiko CS-2. Parallel Comput. **20**(3), 389–398 (1994)
12. InfiniBand Architecture, http://www.infinibandta.org/
13. Jin, H.-W., Sur, S., Chai, L., Panda, D.K.: LiMIC: support for high-performance mpi intra-node communication on Linux clusters. In: Proc. 34th Int. Conf. on Parallel Processing (ICPP 2005), 2005
14. Kielmann, T., Bal, H.E., Verstoep, K.: Fast measurement of LogP parameters for message passing platforms. In: Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP), 2000
15. Liu, J., Vishnu, A., Panda, D.K.: Building multirail infiniband clusters: MPI-level design and performance evaluation, In: Proc. 2004 ACM/IEEE Conf. on Supercomputing (SC'04), 2004
16. Mamidala, A.R., Chai, L., Jin, H.-W., Panda, D.K.: Efficient SMP-aware MPI-level Broadcast over InfiniBand's Hardware Multicast. In: Proc. 6th Workshop on Communication Architecture for Clusters (CAC 2006), 2006
17. Mamidala, A.R., Vishnu, A., Panda, D.K.: Efficient shared memory and RDMA based design for MPI-allgather over InfiniBand. In: Proc. EuroPVM/MPI, pp. 66–75 (2006)
18. MPI: A Message-Passing Interface standard (1997)
19. Myricom, http://www.myricom.com/
20. PDSH, http://www.llnl.gov/linux/pdsh/
21. Pjesivac-Grbovic, J., Angskun, T., Bosilca, G., Fagg, G.E., Gabriel, E., Dongarra, J.J.: Performance analysis of MPI collective operations. In: Proc. 19th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS'05), 2005
22. Qian, Y., Afsahi, A.: Efficient RDMA-based multi-port collectives on multi-rail QsNet$^{\mathrm{II}}$ clusters. In: Proc. 6th Workshop on Communication Architecture for Clusters (CAC 2006), 2006
23. Qian, Y., Afsahi, A.: RDMA-based and SMP-aware multi-port all-gather on multi-rail QsNet$^{\mathrm{II}}$ SMP clusters. In: Proc. 36th Int. Conf. on Parallel Processing (ICPP 2007), 2007
24. Rashti, M.J., Afsahi, A.: Assessing the ability of computation/communication overlap and communication progress in modern interconnects. In: Proc. 15th IEEE Symposium on High-Performance Interconnects (Hot Interconnects 2007), pp. 117–124 (2007)
25. Ritzdorf, H., Traff, J.L.: Collective operations in NEC's high-performance MPI libraries. In: Proc. 20th Int. Parallel and Distributed Processing Symposium (IPDPS'06), 2006
26. Roweth, D., Addison, D.: Optimized gather collectives on QsNet$^{\mathrm{II}}$. In: Proc. EuroPVM/MPI, pp. 407–414 (2005)
27. Roweth, D., Pittman, A., Beecroft, J.: Performance of all-to-all on QsNet$^{\mathrm{II}}$. Quadrics White Paper (2005). http://www.quadrics.com/
28. Shan, H., Singh, J.P., Oliker, L., Biswas, R.: Message passing and shared address space parallelism on an SMP cluster. Parallel Comput. **29**, 167–186 (2003)
29. Sistare, S., vandeVaart, R., Loh, E.: Optimization of MPI collectives on clusters of large-scale SMPs. In: Proc. 1999 ACM/IEEE Conf. on Supercomputing (SC'99), 1999

30. Sur, S., Jin, H.-W., Panda, D.K.: Efficient and scalable all-to-all personalized exchange for InfiniBand clusters. In: Proc. 33rd Int. Conf. on Parallel Processing (ICCP'04), pp. 275–282 (2004)
31. Sur, S., Bondhugula, U.K.R., Mamidala, A., Jin, H.-W., Panda, D.K.: High performance RDMA based all-to-all broadcast for Infiniband clusters. In: Proc. Int. Conf. on High Performance Computing (HiPC 2005), 2005
32. Thakur, R., Rabenseifner, R., Gropp, W.: Optimization of collective communication operations in MPICH. Int. J. High Perform. Comput. Appl. **19**(1), 49–66 (2005)
33. Tipparaju, V., Nieplocha, J.: Optimizing all-to-all collective communication by exploiting concurrency in modern networks. In: Proc. 2005 ACM/IEEE Conf. on Supercomputing (SC'05), 2005
34. Tipparaju, V., Nieplocha, J., Panda, D.K.: Fast collective operations using shared and remote memory access protocols on clusters. In: Proc. 17th IEEE Int. Parallel and Distributed Processing Symposium (IPDPS'03), 2003
35. Traff, J.L.: Efficient allgather for regular SMP-clusters. In: Proc. EuroPVM/MPI, pp. 58–65 (2006)
36. Vadhiyar, S.S., Fagg, G.E., Dongarra, J.: Automatically tuned collective communications. In: Proc. 2000 ACM/IEEE Conf. on Supercomputing (SC2000) (2000)
37. Wu, M., Kendall, R.A., Wright, K.: Optimizing collective communications on SMP clusters. In: Proc. 34th Int. Conf. on Parallel Processing (ICPP 2005), pp. 399–407 (2005)

Ying Qian is currently pursuing her PhD in the Department of Electrical and Computer Engineering at Queen's. She received her BSc in Electronics Engineering from Shanghai Jiao-Tong University, China, in 1998, and her MSc from Queen's University, Canada, in 2004. Her research interests include parallel processing, high-performance communications, user-level messaging, and network performance evaluations.

Ahmad Afsahi is currently an Associate Professor in the Department of Electrical and Computer Engineering at Queen's University, in Kingston, Ontario. He received his PhD in Electrical Engineering from the University of Victoria, British Columbia, in 2000, and his MSc and BSc in Computer Engineering from the Sharif University of Technology and Shiraz University, Iran, respectively. His research interests include parallel and distributed processing, network-based high-performance computing, power-aware computing, and advanced computer architecture. His research has earned him a Canada Foundation for Innovation Award and an Ontario Innovation Trust Award. Dr. Afsahi is a Senior Member of IEEE, and a licensed Professional Engineer in the province of Ontario.