COLLECTIVE COMMUNICATION PATTERNS ON THE QUADRICS NETWORK[*]

Salvador Coll, José Duato, Francisco J. Mora
Technical University of Valencia, Valencia, Spain
scoll@eln.upv.es, jduato@gap.upv.es, fjmora@eln.upv.es

Fabrizio Petrini, Adolfy Hoisie
Los Alamos National Laboratory, Los Alamos, NM
fabrizio@lanl.gov, hoisie@lanl.gov

Abstract The efficent implementation of collective communication is a key factor to provide good performance and scalability of communication patterns that involve global data movement and global control. Moreover, this is essential to enhance the fault-tolerance of a parallel computer. For instance, to check the status of the nodes, perform some distributed algorithm to balance the load, synchronize the local clocks or do performance monitoring. For these reasons the support for multicast communications can improve the performance and resource utilization of a parallel computer.

The Quadrics interconnect (QsNET), which is being used in some of the largest machines in the world, provides hardware support for multicast. The basic mechanism consists of the capability for a message to be sent to any set of contiguous nodes in the same time it takes to send a unicast message. The two main collective communication primitives provided by the network system software are the barrier synchronization and the broadcast. Both of them are implemented in two different ways, either using the hardware support, when nodes are contiguous, or a balanced tree and unicast messaging, otherwise.

In this paper some performance results are given for the above collective communication services, that show, on the one hand, the outstanding performance of the hardware-based primitives even in the presence of a high network background traffic; and, on the other hand, the limited performance achieved with the software-based implementation.

Keywords: Interconnection networks, Quadrics, collective communication, multicast, performance evaluation.

[*]The work was supported by the Spanish CICYT through contract XXXXXX

1. Introduction

Current trends on high-speed interconnects include the availability of a communication processor in the network interface card [3][10], which allows the implementation of high level messaging libraries without explicit intervention of the main CPU [4]; and the support for collective communications at hardware level [11], which outperforms traditional software-based multicast implementations. Both approaches can aid in the implementation of communication patterns which involve global data movement and global control. Barrier synchronization, broadcast, gather, scatter, reduce and total exchange are typical examples of collective communication patterns.

Hardware support for multicast communication combined with the local processing power provided by network processors gives the opportunity of addressing several open problems in current and future medium- and large-scale parallel computers: scalability, responsiveness, programmability, performance, resource utilization and fault-tolerance. Many recent research results show that job scheduling techniques based on gang scheduling and coscheduling algorithms can provide solutions to these open problems [1][7][6][9].

The practical application of these scheduling algorithms relies on the efficient implementation of collective communication. In this way, not only the performance of particular applications would be improved, the overall system performance would experience significant improvements.

Hardware support for multicast communication requires many functionalities, that are dependent on the network topology, the routing algorithm and the flow control strategy. For example, in a wormhole network, switches must be capable of forwarding flits from one input channel to multiple output channels at the same time [15]. Unfortunately, these tree-based algorithms can suffer from blocking problems in the presence of congestion [16]. Also, the packets must be able to encode the set of destinations in an easy-to-decode, compact manner, in order to reduce the packet size and to guarantee fast routing times in the switches.

Software multicasts, based on unicast messages, are simpler to implement, do not require dedicated hardware and are not constrained by the network topology and routing algorithms, but they can be much slower than the hardware ones.

In previous work we analyzed in depth how hardware- and software-based multicasts are designed and implemented in the Quadrics network (QsNET) [11]. In this paper some modifications have been performed in the communication libraries to evaluate the underlying mechanisms that provide multicast support.

The initial part of the paper part introduces the mechanisms at the base of the hardware and software multicast primitives that, on their turn are at the base of more sophisticated collective communication patterns as broadcasts, barriers, scatter, gather, reduce, etc.

In the second part we provide an extensive performance evaluation of two user-level collective communication patterns, barrier and broadcast, implemented using both hardware and software multicast algorithms.

The rest of this paper is organized as follows. Section 1.2 presents the basic mechanisms that support collective communication on the QsNET, while Section 1.3 gives a detailed description of the main collective communication services. Section 1.4 presents the experimental results and performance analysis. Finally, some concluding remarks and future directions are given in Section 1.5.

2. Collective comunication on the Quadrics network

The QsNET [10]is a butterfly bidirectional multistage interconnection network with 4 x 4 switches, which can be viewed as a quaternary fat-tree [8]. It is based on two building blocks, a programmable network interface called Elan [13]and a low-latency high-bandwidth communication switch called Elite [14]. It uses wormhole switching with two virtual channels per physical link, source-based routing and adaptive routing. Some of the most important aspects of this network are: the integration of the local memory into a distributed virtual shared memory, the support for zero-copy remote DMA transactions and the hardware support for collective communication [11].

The basic hardware mechanism that supports collective communication is provided by the Elite switches. The Elite switches can forward a packet to a set of physically contiguous output ports. Thus, a multicast packet can be sent to any group of adjacent nodes by using a single *hardware-based* multicast transaction. When the destination nodes are not contiguous a *software-based* implementation which uses a tree and point-to-point messages is used.

2.1 Hardware-based multicast

Hardware-based broadcasts are propagated into the network by sending a packet to the top of the tree and then forwarding the packet to more than one switch output as the packet is sent down the tree. Deadlocks might occur on the way down when multiple broadcasts are sent simultaneously [5]. This situation is avoided by sending broadcast packets always to a fixed top tree switch, thus serializing all broadcasts. In Figure 1 (a) it is shown that the top leftmost switch is chosen as the logical root for the collective communication, and every request, in the ascending phase, must pass through one of the dotted paths until it gets to the root switch. In Figure 1 (b) we can see how a multicast packet reaches the root node; the multiple branches are then propagated in parallel. If another collective communication is issued while the first one is still in progress, it is serialized in the root switch. The second multicast packet will be able to proceed only after an EOP token cleans the circuit of the first communication (Figure 1 (c) and (d)). All nodes connected to the network are capable of receiving the multicast packet, as long as the multicast set is physically contiguous.

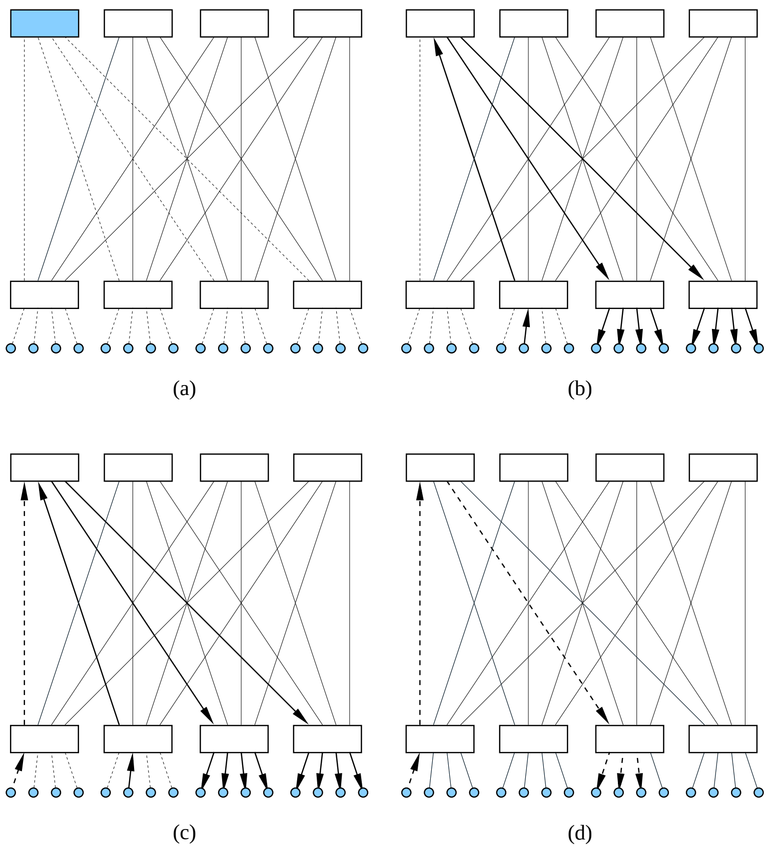


Figure 1. Hardware multicast

For a multicast packet to be successfully delivered, a positive acknowledgement must be received from all the recipients of the multicast group. The Elite switches combine the acknowledgements, as pioneered by the NYU Ultracomputer [2][12], returning a single one to the source. Acknowledgements are combined in a way that the "worst" ack wins (a network error wins over an unsuccessful transaction, which on its turn wins over a successful one), returning a positive ack only when all the partners in the collective communication complete the distributed transaction with success.

2.2 Software-based multicast

The Quadrics communication libraries implement a software-based multicast algorithm to be used when the hardware support is not usable. The algorithm uses a balanced tree to perform multicast transactions. Figure 2 shows the tree used for a 16-node group with the source at node 0. The source process sends a copy of the packet to its children, which, after receiving it, forward the packet to their children. Eventually all the processes will be reached. As it can be seen for the example on the figure, a broadcast would take 6 point-to-point transactions to be completed. This algorithm is performed by the thread processor included in the Elan. The thread processor can receive an incoming packet, do some basic processing and send one or more replies in few μs, without any interaction with the main processors.

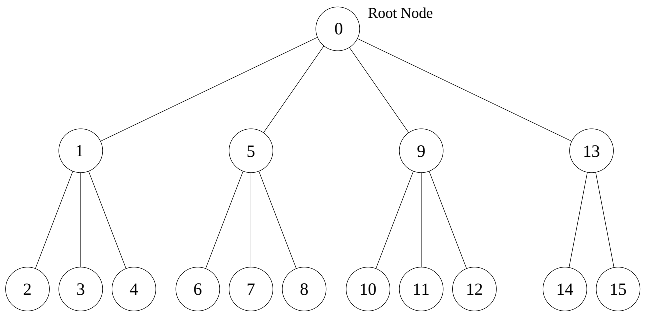


Figure 2. Balanced tree used by the software multicast algorithm

3. Barrier Synchronization and Broadcast

3.1 Barrier Synchronization

A synchronization barrier is a logical point in the control flow of a parallel program at which all processes in a group must arrive before any of the processes in the group are allowed to proceed. Typically, a barrier synchronization involves a logical reduce operation followed by a broadcast.

QsNET implements two different synchronization mechanisms in the communication libraries, a mixed software and hardware barrier called `elan_gsync()` and a purely hardware one called `elan_hgsync()`.

The algorithm implemented with `elan_gsync()` uses the balanced tree described in Section 1.2.2 to send the 'ready' signal to the process with VPID 0. Each process in the tree waits for 'ready' signals from its children, and when it receives all of them sends its own signal up to the parent process. When the root process receives all its 'ready' signals it performs a hardware broadcast which either sets an event (which all processes are waiting for) or writes a single word in a given memory location (which all processes are polling). If the destination nodes are not adjacent the same tree structure is used to distribute the data using point-to-point messages.

When the barrier is performed with `elan_hgysnc()` or `elan_hgysncEvent()`, all processes in the group set a barrier sequence number in a system memory location (Figure 3(a)). All of them but the root node (which is the process with the lowest ID in the group) wait for a 'ready' signal (busy polling on a memory location with `elan_hgysnc()` or an event mechanism with `elan_hgysncEvent()`). The root process uses an Elan thread to send a special test-and-set broadcast packet (subfigure (b)). This packet spans all the processes and checks if the barrier sequence value in each process matches with its own sequence number (it does if the corresponding process reached the barrier). All the replies are then combined by the Elites on the way back to the root node which receives a single ACK token (subfigure (c)). If all the nodes are ready an EOP token is sent to the group to set an event or write a word to wake up the processes waiting in the barrier (subfigure (d)). It has to be noted that this mechanism is completely integrated into the network flow control.

3.2 Broadcast

The main communication primitive of the QsNET is the remote DMA. A DMA operation transfers data between local and remote address spaces (including Elan memory). In addition to providing point-to-point communication, DMAs can also be used to perform group-wide operations such as broadcast and flood DMAs (a flood is similar to a broadcast but the operation completes as soon as any of the destinations accepts the DMA). A group of destination processes is defined by specifying a virtual group identifier. The effect of a write broadcast DMA is to copy the data from the source to the destination buffers of all the processes in the group. The implementation of the broadcast DMAs relies on all receiving processes having the destination buffer at the same virtual address, to obtain good performance.

Two different broadcast implementations are provided by the QsNET communication libraries: `elan_bcast()` and `elan_hbcast()`. Both must be called by all the processes in the group involved in the broadcast operation to guarantee that the receivers have allocated the buffers by the time the transaction is performed by the sender process. As a result, the broadcast is composed of two transactions: first, a barrier synchronization and, second, the broadcast itself. In both implementations, two types of memory resources can be used. On the one hand a global destination buffer, which has the same virtual address in all the processes (the communication library provides special memory allocation functions to do that), allows DMA transactions directly from one

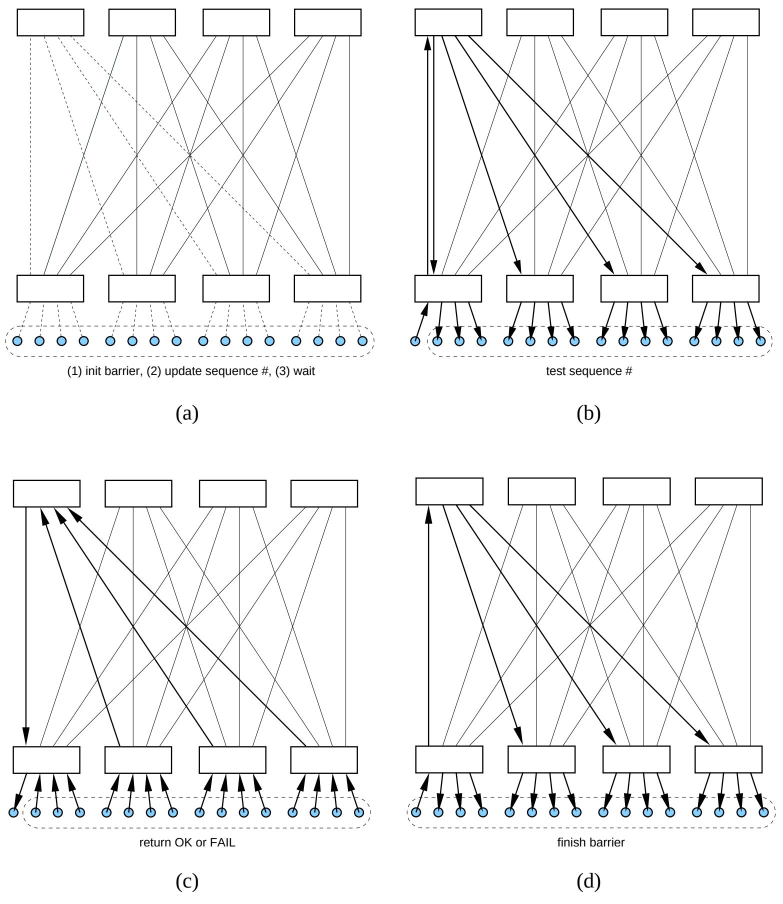


Figure 3. elan_hgsync() Barrier implementation

source to multiple destinations. On the other hand, if this memory allocation is not used, system buffers are utilized as intermediate copy space (this approach implies one copy at the source, and another copy at the destination).

The `elan_bcast()` implementation uses a software-based synchronization for the first phase similar to that utilized by the first phase of `elan_gsync()` (Section 1.3.1). The second phase is triggered by an event set in the source node and is done using the hardware broadcast mechanism (if all the destination Elans are contiguous) or by means of a software-based broadcast (if the destination Elans are not). This transaction distributes the data and wakes up the processes waiting in the barrier performed during the first phase. This implementation provides better performance than a call to `elan_gsync()` (which involves a software-based synchronization and a broadcast) and a later broadcast to send the data.

The `elan_hbcast()` primitive calls `elan_bcast()` if the hardware broadcast mechanism is not available, for example when the nodes are not contiguous. If this mechanism is available, it performs a barrier to synchronize all the nodes using `elan_hgsync()` (Section 1.3.1) and a hardware broadcast to distribute the data.

The Elan hardware broadcast can only write to the memory space of a single process per node since there is only a single context specified by the virtual process identifier. Hence, with multiple processes per node, the only way to use the hardware broadcast facility is to broadcast into an area of shared memory and then get the processes to copy from there. This has been optimized by using a FIFO like scheme that tries to overlap the broadcast with the copies.

4. Experimental Results

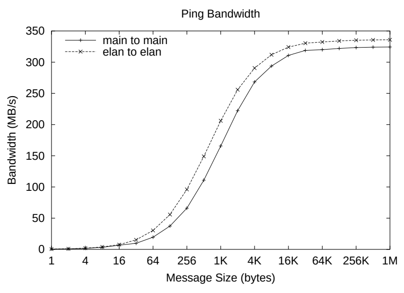
The performance of the collective communication services of the Quadrics network was evaluated on a 32-node cluster of Dell 1550, running Red Hat 7.1 Linux. Each node has two 1.13 GHz Pentium-III with 1GB of ECC RAM, and a Quadrics QM-400 Elan3 NIC attached to the network though a 66MHz/64-bit PCI bus.

4.1 Unidirectional Ping

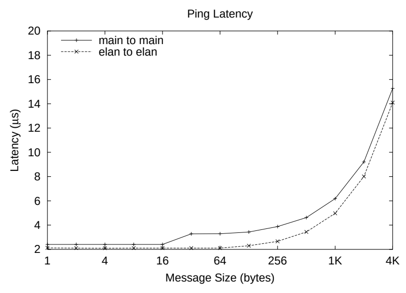
To provide some basic performance results of the QsNET on our experimental tesbed, we analyzed the latency and bandwitdh of the network for unicast messages of different sizes sent between a pair of nodes. The communication buffers were placed either in main or in Elan memory. These tests provide a performance reference to consistently analyze the results on collective communication.

The bandwidth of the unidirectional ping is shown on Figure 4 a). The asymptotic data bandwith is 336 MB/s and is obtained when the buffers are placed in the Elan memory. Taking into account the data payload and the overhead introduced by the message header (routing tags, CRC, etc.), the delivered peak bandwidth is 396 MB/s, or 99% of the nominal bandwidth (400 MB/s). With buffers in main memory the peak bandwidth is 324 MB/s. These results also show that the PCI interface running at 66 MHz provides a good performance.

Figure 4 b) shows the latency for messages shorter than 4KB. With Elan memory buffers the latency is almost constant at 2.1 μs for messages up to 64 bytes, because these messages can be sent using a single transaction. We note a slight increase in the latency with main memory buffers of 0.3μs for messages up to 16 bytes and of 1 μs for messages up to 4 KB.



(a)



(b)

Figure 4. Unidirectional Ping

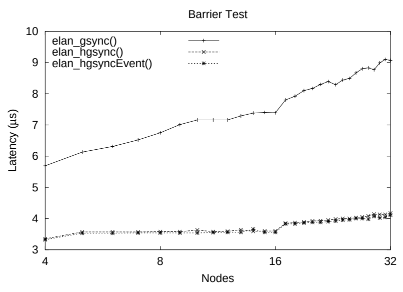


Figure 5. Barrier Synchronization

4.2 Collective Communications

The barrier synchronization and broadcast primitives provided by the QsNET have been tested using configurations ranging from 4 to 32 nodes. Results have been obtained by averaging the metrics over 10000 consecutive tests. Average latency results are reported for the barrier synchronization tests. For the broadcast tests bandwidth and latency are reported.

In addition, tests with background traffic have been performed to analyze the behavior of the collective communications under network contention. This background traffic is generated by 32 processes running in 32 nodes, with all nodes injecting messages into the network at maximum load. The goal of these tests is to identify the performance degradation experienced by the collective communication in the presence of congestion. The traffic patterns used to generate background traffic has been complement: the node with binary coordinates $a_{n-1}, a_{n-2}, \ldots, a_1, a_0$ communicates with the node $\overline{a_{n-1}, a_{n-2}, \ldots, a_1, a_0}$. This pattern was selected because it uses all the network links at the same time.

To guarantee that the performance degradation of the collective communication is only due to the network contention and not to scheduling issues, the background traffic generation and the collective communication benchmark were run in distinct processors.

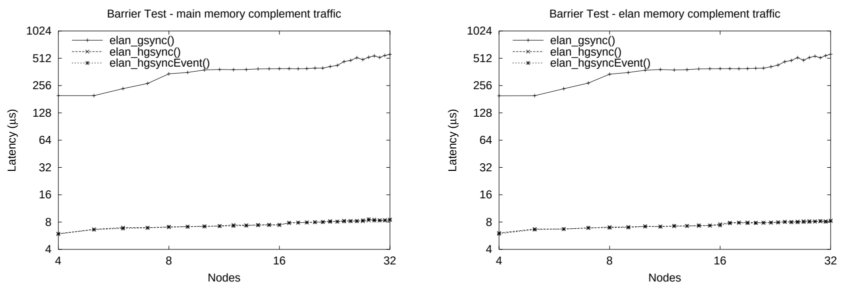
4.2.1 Barrier Synchronization. Figure 5 shows the average time required to perform a barrier synchronization in an empty network. Results for the three primitives provided by the libraries (Section 1.3.1) are shown versus the number of nodes. We can see that the hardware-based implementations of the barrier (`elan_hgsync()` and `elan_hgsyncEvent()`) provide the best results when compared to the software-based implementation (`elan_gsync()`), both in absolute performance and in scalability. The latency of the software-based implementation grows as the logarithm of the number of nodes (approximately 1μs each time the number of nodes is doubled). In this case the average latency to synchronize 32 nodes is 9.1μs. On the other hand, the hardware barriers (which show negligible differences between them) provide an average latency of 4.2μs for 32 nodes.

The behavior of the barrier synchronization has been analyzed by performing tests with complement background traffic. The results depicted in Figure 6 show that the buffer allocation of the background traffic has no significant effect. The network is the bottleneck in this case, not the PCI bus. The software barrier is significantly affected by the background traffic, the slowdown is 60 in the worst case of 32 nodes. On the other hand, there is little impact on the hardware barriers, whose latency is only doubled (1.8 slowdown).

The scalability is also affected by the background traffic. The latency of the hardware-based implementations with the number of nodes increases 24% (when the number of nodes varies from 4 to 32) with no backgorund traffic, and 43% with background traffic. On the other hand, the latency increase of the software-based synchronization is 60% and 180% respectively. The software-based barrier latency scalability is shown to be more sensitive to complement background traffic than the hardware-based barriers.

From a practical point of view, the hardware-based implementation of the barrier can be considered insensitive to network contention.

4.2.2 Broadcast. As mentioned in Section 1.3.2, the `elan_bcast()` primitive by default uses a hardware multicast when all the destination nodes are contiguous. In order to perform a fair comparison between the hardware- and the software-based multicast mechanisms, the original implementation of `elan_bcast()` has been modified to use the tree-based algorithm, even in the case where the destination nodes are adjacent. Thus, the results reported in this section refer to the modified version of `elan_bcast()`.



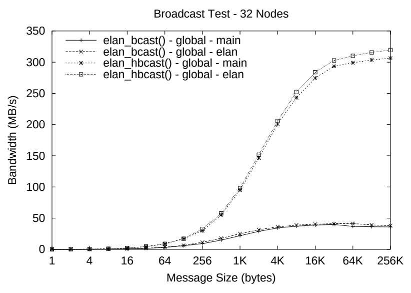
(a) (b)

Figure 6. elan_hgsync() Latency with Contention

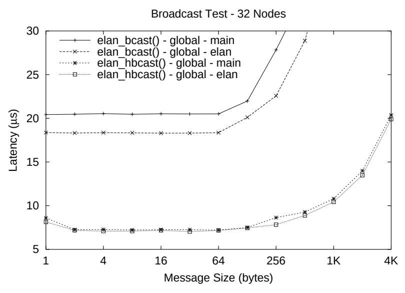
Figure 7 shows the results obtained with broadcast over 32 nodes using both algorithms supported by the system libraries (Section 1.3.2) with buffers globally allocated in main and Elan memory, that is, with the same virtual address in all processes. As expected, the best performance is obtained with the hardware-based broadcast with buffers in Elan memory. In this case the measured bandwidth for 256 KB messages is 319 MB/s, which is 95% of the unicast bandwidth (Section 1.4.1). With buffers in main memory the peak bandwidth is 306 MB/s, or 95% of the unicast bandwidth. The asymptotic bandwidth of the software-based implementation is 40 MB/s (8 sending steps to reach the last node in a 32-node network), which is worse than what it would be expected with a binary-tree implementation (5 sending steps to reach 32 nodes). For all the implementations the latency for messages shorter than 64 bytes is constant since those messages are sent using a single transaction. The hardware-based broadcast latency is lower than 8 μs for messages up to 256 bytes, with no significant effect of the memory allocation, while the software-based broadcast takes 10μs longer when using Elan memory buffers and 12μs longer with main memory buffers.

Bandwidth and latency versus the number of nodes for 256KB messages are depicted in Figure 8. Regarding the hardware-based broadcast, both performance metrics are almost insensitive to the number of nodes (for the tested configurations), slowdowns between 3 and 4% were measured. On the other hand, when the software-based broadcast is used, a significant performance degradation occurs when the number of nodes increases due to the logarithmic behavior of the tree-based implementation. In this case the slowdown is 66% when the number of nodes increases from 4 to 32.

In the presence of network contention (Figure 9) the broadcast performance decreases significantly. Similar results are obtained either with background

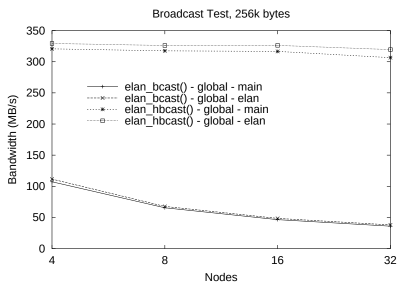


(a)

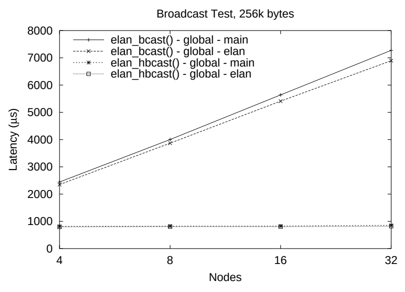


(b)

Figure 7. Broadcast



(a)



(b)

Figure 8. Broadcast Scalability

traffic using main or Elan memory (due to space limitations only results for background traffic in Elan memory are shown). The asymptotic bandwidth for the hardware-based broadcast is 37 MB/s and the software-based broadcast gets 14 MB/s. The hardware-based implementation suffers from a higher degradation in the presence of background traffic since there is higher contention with the background traffic to perform the link reservation.

In terms of scalability (Figure 10) the four alternatives suffer from the same performance degradation as the number of nodes increases. Although his effect slows down as the number of nodes is increased, we need to further investigate this behavior by analyzing larger configurations.

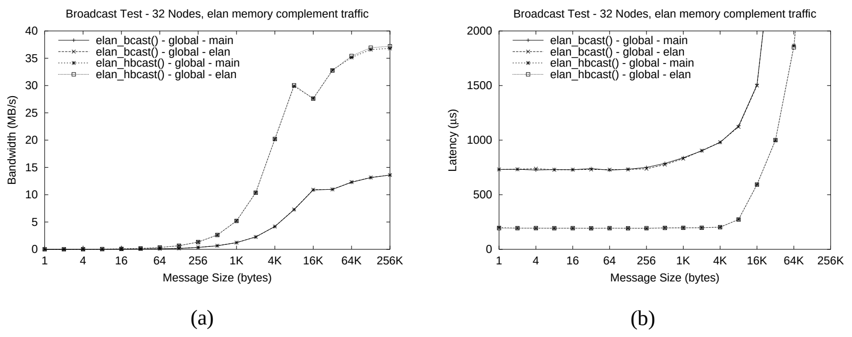


Figure 9. Broadcast with Contention

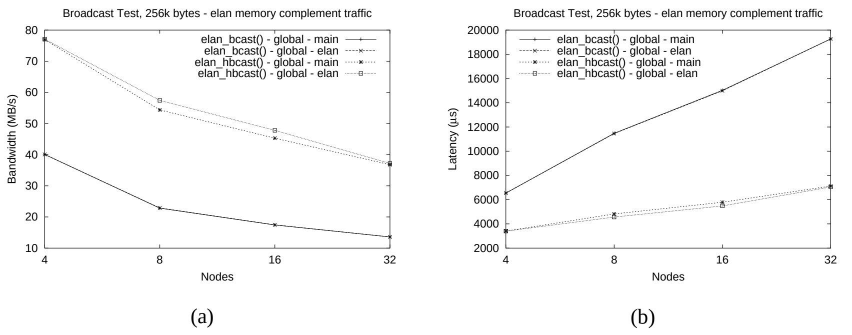


Figure 10. Broadcast Scalability with Contention

5. Conclusion

In this paper, we present a description and evaluation of the Quadrics interconnection network (QsNET) support for collective communication.

The underlying mechanism that provides the hardware support for collective communication is presented. After that, the two basic communication patterns provided by the system software, barrier synchronization and broadcast, are described.

An experimental evaluation of hardware-based and software-based implementations of these services has been performed on a 32-node cluster. Our experiments show that the time to complete a hardware-based barrier synchronization on the whole set of nodes is as low as 4.2μs, with very good scalability for the network configurations tested. In the presence of network contention, the average latency for the hardware barrier is 8.5μs, with 95% of the synchronizations taking less than 16μs. From a practical point of view the hardware-based barrier can be considered insensitive to network contention.

Good latency and scalability are also achieved with the software-based synchronization, which completes in 9.1μs on an empty network. On the other hand, it is shown to suffer a significant performance degradation with background traffic.

Regarding the broadcast, the hardware-based implementation can deliver a sustained bandwidth of 319 MB/s (95% of the point-to-point bandwidth) with less than 8μs latency for messages up to 256 bytes when using Elan memory buffers (306 MB/s, 8μs with main memoy buffers). The software-based broadcast delivers an asymptotic bandwidth of 40 MB/s for any memory allocation (the bottleneck in this case is the algorithm itself, not the PCI bus).

Contention tests, done in the presence of high network load, show that the broadcast maintains reasonably good performance (i.e. less than 200μs to deliver messages up to 4KB). In this case the hardware-based broadcast outperforms the software-based broadcast thanks to its hardware-based synchronization and packet transmission mechanism.

Overall, our analysis shows the potential of the interconnect to efficiently support large-scale collective communication, even in the presence of high network contention. On the other hand, while the hardware support is shown to make possible extremely good performance, its usage is limited to those cases where all the destination nodes are physically contiguous. Otherwise, for example with a single faulty node, a tree-based software implementation is used. This situation is likely to happen in current an future high-performance paralle machines, soon to reach tens of thousand of processors. This fact makes the hardware support unsable in practice. As future work, we plan to address this problem to overcome the impact of a few faulty nodes (or components) on the performance of collective communications. This will make a great impact not only on applications performance but on the behavior of the whole system. In particular if modern resource managers with job launching and process scheduling functions, which rely on efficient collective communication, are used.

Acknowledgments

The authors would like to thank the Quadrics team, David Addison, Jon Beecroft, Robin Crook, Moray McLaren, David Hewson, Duncan Roweth and John Taylor, for their invaluable support.

References

[1] Andrea Carol Arpaci-Dusseau. Implicit coscheduling: coordinated scheduling with implicit information in distributed systems. *ACM Transactions on Computer Systems*, 19(3):283–331, 2001.

[2] G. Bell. Ultracomputer: a Teraflop before its time. *Communications of the ACM*, 35(8):27–47, 1992.

[3] Nanette J. Boden, Danny Cohen, Robert E. Felderman, Alan E. Kulawick, Charles L. Seitz, Jakov N. Seizovic, and Wen-King Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, January 1995.

[4] Darius Buntinas, Dhabaleswar Panda, and P. Sadayappan. Performance Benefits of NIC-Based Barrier on Myrinet/GM. In *Workshop on Communication Architecture for Clusters (CAC '01)*, San Francisco, CA, April 2001.

[5] José Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection Networks: an Engineering Approach*. IEEE Computer Society Press, 1997.

[6] Fabrizio Petrini and Wu-chun Feng. Buffered Coscheduling: A New Methodology for Multitasking Parallel Jobs on Distributed Systems. In *Proceedings of the International Parallel and Distributed Processing Symposium 2000, IPDPS2000*, Cancun, MX, May 2000.

[7] Dror G. Feitelson and Morris A. Jette. Improved Utilization and Responsiveness with Gang Scheduling. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[8] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, October 1985.

[9] Fabrizio Petrini. Scaling to Thousands of Processors with Buffered Coscheduling. In *Scaling to New Heights Workshop*, Pittsburgh, PA, May 2002.

[10] Fabrizio Petrini, Wu chun Feng, Adolfy Hoisie, Salvador Coll, and Eitan Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.

[11] Fabrizio Petrini, Salvador Coll, Eitan Frachtenberg, and Adolfy Hoisie. Hardware- and Software-Based Collective Communication on the Quadrics Network. In *IEEE International Symposium on Network Computing and Applications 2001 (NCA 2001)*, Boston, MA, October 2001.

[12] G. F. Pfister and V. A. Norton. Hot-spot Contention and Combining in Multistage Interconnection Networks. *IEEE Transactions on Computers*, C-34(10):943–948, October 1985.

[13] Quadrics Supercomputers World Ltd. *Elan Reference Manual*, January 1999.

[14] Quadrics Supercomputers World Ltd. *Elite Reference Manual*, November 1999.

[15] Rajeev Sivaram, Dhabaleswar Panda, and Craig Stunkel. Efficient Broadcast and Multicast on Multistage Interconnection Networks using Multiport Encoding. In *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing*, New Orleans, LA, October 1996.

[16] Rajeev Sivaram, Dhabaleswar Panda, and Craig Stunkel. Multicasting in Irregular Networks with Cut-Through Switches using Tree-Based Multidestination Worms. In *Parallel Computing, Routing, and Communication Workshop, PCRCW'97*, Atlanta, GA, June 1997.