Scalable, High-Performance NIC-Based All-to-All Broadcast over Myrinet/GM*

Weikuan Yu† Darius Buntinas‡ Dhabaleswar K. Panda†

Network-Based Computing Lab†
Department of Computer Science and Engineering
The Ohio State University
{yuw,panda}@cse.ohio-state.edu

Mathematics and Computer Science Division‡
Argonne National Laboratory
buntinas@mcs.anl.gov

Abstract

All-to-all broadcast is one of the common collective operations that involve dense communication between all processes in a parallel program. Previously, programmable Network Interface Cards (NICs) have been leveraged to efficiently support collective operations, including barrier, broadcast, and reduce. This paper explores the characteristics of all-to-all broadcast and proposes new algorithms to exploit the potential advantages of NIC programmablity. Along with these algorithms, salient strategies have been used to provide scalable topology management, global buffer management, efficient communication processing, and message reliability. The algorithms have been incorporated into a NIC-based collective protocol over Myrinet/GM. The NIC-based all-to-all broadcast operations improve all-to-all broadcast bandwidth over 16 nodes by a factor of 3, compared to host-based all-to-all broadcast operation. Furthermore, the NIC-based operations have been demonstrated to achieve better scalability to large systems and very low host CPU utilization.

1. Introduction

Collective communication can constitute up to 70% of the execution time of a scientific application [10]. High-performance, scalable collective communication is an important factor for parallel programs in achieving good performance. The programmable processors in some modern interconnects, including Myrinet [2] and Quadrics [9], have been leveraged to offload communication processing and optimize collective communication and synchronization [1, 4, 5, 15, 6, 14]. It has been shown that collective operations based on programmable Network Interface Cards (NICs) reduce the host processor involvement [5, 15], avoid round-trip PCI bus traffic [5, 15], and increase the tolerance to process skew [3, 15] and operating system effects [6]. Together, these benefits help improve communication performance in terms of latency and bandwidth.

To date, the collective communication operations that have been studied and demonstrated as beneficial with the NIC-based support have largely been limited to barrier, broadcast, and reduce. No algorithms have been reported to provide efficient NIC-based support to a common collective operation with a very dense communication pattern, such as all-to-all broadcast. In all-to-all broadcast, every process sends the same message to all the other processes, a procedure that implies that each process also receives a message from every other process. In distributed computing, all-to-all broadcast is often referred to as gossip. Because of its dense communication pattern, offloading all-to-all broadcast to the NIC imposes a greater demand on the NIC's limited memory and computation resources compared to other collective operations. To avoid impacts on other message-passing activities and enable efficient NIC-based all-to-all broadcast operation, one must provide mechanisms to handle additional design challenges, such as group topology, buffer management, and message reliability.

In this paper, we take on these challenges. We present scalable, high-performance NIC-based all-to-all broadcast operations and incorporate them into a NIC-

*This research is supported in part by a grant from Los Alamos National Laboratory and National Science Foundation's grants #CNS-0204429 and #CCR-0311542, and also in part by U.S. Department of Energy, under Contract W-31-109-ENG-38 and Grant #DE-FC02-01ER25506. Los Alamos National Laboratory is operated by the University of California for the National Nuclear Security Administration of the United States Department of Energy under contract W-7405-ENG-36.

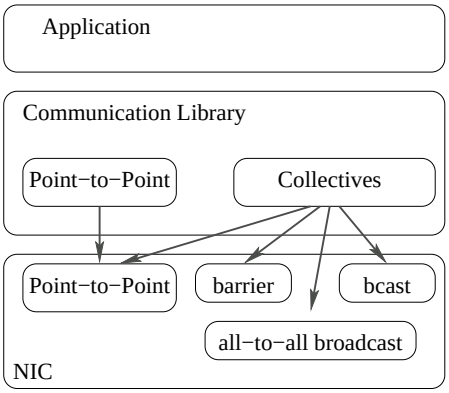


Fig. 1. NIC-Based All-to-All Broadcast in a NIC-Based Collective Processing Protocol

based collective protocol. We propose two all-to-all broadcast algorithms: concurrent broadcasting and recursive doubling. Along with these algorithms, we provide salient strategies for a scalable, binomial tree-based group topology; efficient global buffer management; efficient communication processing with fast forwarding of packets; and reliability. The resulting all-to-all broadcast operations have been implemented and incorporated into a NIC-based collective protocol over Myrinet/GM.

Compared to the host-based all-to-all broadcast operation, the NIC-based operations improve broadcast bandwidth over 16 nodes by a factor of 3. Furthermore, the NIC-based all-to-all broadcast operations have been demonstrated to achieve better scalability to large systems and very low host CPU utilization. To the best of our knowledge, this paper is the first in the literature to report efficient NIC-based all-to-all broadcast algorithms over Myrinet/GM.

The rest of the paper is structured as follows. In the next section, we describe the motivation and related work. Section 3 gives an overview of Myrinet/GM. Section 4 provides the design. The implementation of the algorithms is described in Section 5. Experiments and results are provided in Section 6. Conclusions and future work are described in Section 7.

2. NIC-Based All-to-All Broadcast

In this section, we first discuss potential benefits of NIC-based all-to-all broadcast. Then we briefly introduce related work on efficient NIC-based collective communications.

2.1. Potential Benefits of the NIC-Based All-to-All Broadcast

By taking advantage of NIC programmability, the communication processing for all-to-all broadcast is done completely by the NIC. Host CPU is not involved in the intermediate steps of all-to-all broadcast. Research [14] demonstrates that the barrier operation can benefit from efficient NIC-based communication processing using a separate NIC-based collective protocol. Figure 1 shows that all-to-all broadcast can also be targeted to be part of such a separate collective protocol at the NIC. The all-to-all broadcast operation is then presented as an application interface to user applications at the host. All-to-all broadcast may be viewed as the same communication pattern as barrier except for the larger amount of data communicated between processes. Presumably, all-to-all broadcast could also take advantage of the following benefits of NIC-based communication processing.

Low Latency and High Bandwidth – By offloading the all-to-all broadcast communication processing, received data packets can be processed and forwarded directly without making a round trip across the PCI bus. Offloading these can speed the communication in all-to-all broadcast and improve performance in terms of latency and bandwidth.

High Scalability – The NIC-based all-to-all broadcast has the potential of increasing the scalability of this operation. Compared to the host-based all-to-all broadcast, the NIC-based all-to-all broadcast comes closer to utilizing full network capacity because of fewer trips across the PCI bus and less involvement of host processors.

Low Host CPU Utilization – Since the communication processing for an all-to-all broadcast operation is completely undertaken by the NIC, a user application needs only to invoke the all-to-all broadcast operation through the host-side interface. It can then do other computation that does not depend on the results of the all-to-all broadcast operation, and check back later for the completion of the all-to-all broadcast operation. The effective utilization of the computation resources is very low. So the NIC-based all-to-all broadcast allows user applications to have the high overall host CPU utilization if they are able to overlap the all-to-all broadcast communication with other useful computation. However, it does not guarantee high host CPU utilization because user applications or higher-layer middleware libraries may still use continuous polling to check the completion of operations.

2.2. Related Work on NIC-Based Collectives

Previous work [13, 1, 4, 5, 15, 6, 14] has exploited the benefits of efficient collective communications by taking advantage of NIC support. Researchers [13, 1, 15] have implemented different algorithms to provide efficient NIC-based broadcast support. All these algorithms use NIC-based packet forwarding as one of the means to speed broadcast, although they differ significantly in their reliability and flow control mechanisms. Researchers [5, 3, 6, 14] have also studied the benefits of offloading barrier and reduce operations to the NIC. Results indicate that the barrier and reduce operations can benefit from reduced host involvement, efficient communication processing, and better tolerance to process skew. However, none of the work cited has studied the benefits of all-to-all broadcast operations with NIC-based support.

Earlier work [14] provided a separate NIC-based collective protocol to eliminate the communication processing redundancy and introduced additional collective application programming interface in the user-level protocol. The work did not, however, investigate how other collective operations can fit into and benefit from this NIC-based protocol. Two factors must be considered. First, an all-to-all broadcast operation essentially involves multiple parallel broadcast operations, each with every participating process being the root. This factor suggests that an efficient NIC-based all-to-all broadcast operation could also be possible by taking advantage of packet forwarding and resending similar to that used by NIC-based broadcast operations [13, 1, 5, 15]. Second, as stated earlier, all-to-all broadcast has a communication pattern similar to that of barrier, except for the amount of data communicated. This factor suggests all-to-all broadcast can also benefit from the NIC-based collective protocol [14].

In this work we investigate the feasibility and benefits of the NIC-based all-to-all broadcast. In studying this communication-intensive collective operation, we also address other design issues related to an efficient NIC-based collective protocol, including group topology management, buffer management, scalable communication processing, and reliability.

3. Overview of Myrinet and GM

Myrinet is a high-speed interconnect technology using wormhole-routed crossbar switches to connect all the NICs. GM is a user-level communication protocol that runs over the Myrinet [2] and provides a reliable, ordered delivery of packets with low latency and high bandwidth. The basic send and receive operations work as follows.

Sending a Message – To send a message, a user application generates a send descriptor, referred to as a *send event* in GM, for the NIC. The NIC translates the event to a *send token* (a form of send descriptor that the NIC uses) and appends it to the send queue for the desired destination. With outstanding send tokens to multiple destinations, the NIC processes the tokens to different destinations in a round-robin manner. To send a message for a token, the NIC also has to wait for the availability of a send buffer to accommodate the data. Then the data is DMAed from the host buffer into the buffer for the send packet and the packet is injected into the network. The NIC keeps a *send record* of the sequence number and a timestamp for each packet it has sent. If the acknowledgment is not received within the timeout period, the sender retransmits the packet. When all the send records are acknowledged, the NIC passes the send token back to the host.

Receiving a Message – To receive a message, the host provides some registered memory as the receive buffer by preposting a receive descriptor. A posted receive descriptor is translated into a *receive token* by the NIC. When the NIC receives a packet, it checks the sequence number. Unexpected packets are dropped immediately. For an expected packet, the NIC locates a receive token, DMAs the packet data into the host memory, and then acknowledges the sender. When all the packets for a message have been received, the NIC sends a *receive event* to the host process indicating that a new message has arrived.

4. Design Challenges for NIC-Based All-to-All Broadcast

In this section, we explore the design challenges for scalable NIC-based all-to-all broadcast. Given the high demand of memory and computation resources of its dense communication pattern and the limited resources available at the NIC, NIC-based all-to-all broadcast algorithms need to minimize resource requirements. Under this restriction, four major design issues must be considered in order to achieve good scalability and high performance. These are group topology management, buffer management, communication processing, and message reliability.

4.1. Group Topology Management

For clusters with thousands of nodes, placing the entire group membership information at the NIC incurs a large memory requirement, and this requirement is

even greater if the communication state for each peer NIC is to be maintained. Thus it is important to provide a scalable method to store and access group topology information and the associated group communication states. Typically, collective operations over point-to-point links use spanning trees to cover all the nodes. One node in a spanning tree communicates with its parent node and a limited number of child nodes. The binomial tree is one of the most commonly used topologies for collective communication. It provides two advantages over other topologies. First, it can be shared by barrier, broadcast (even with different roots), all-to-all broadcast, and other collective operations. Second, since the distance between any pair of directly communicating nodes is always some power of 2, updating the communication state of peers can be easily handled by the NIC processors using bit-shifting operations (typically, NICs are not equipped with FPU). Thus the binomial tree-based topology is a good choice for scalable group management. To achieve scalability, the entire group topology is managed distributively. Each NIC, i, maintains the information of NICs that are in the set $\{i \pm 2^j \bmod N : 0 \leq j < \log N\}$. So, each NIC only needs to maintain the topology and state information of $(2 \times \log N)$ NICs.

4.2. Buffer Management for NIC-Based All-to-All Broadcast

Processes in parallel applications can reach the same collective communication at different times. Packets can arrive at the NIC before the host even posts the receive buffer for the corresponding collective operation. In order to allow the NIC to still receive and forward the early arrived data packets, a system buffer must be present to accommodate these packets. This can avoid having to drop packets and can improve the performance of collective communication and its tolerance to process skew [15]. A common way of managing of collective buffers is to provide a global virtual memory [11, 16], which is divided into multiple channels to receive packets from multiple outstanding collective operations. The order in which these channels are used is globally synchronized across all the NICs when the collective protocol is initialized. At the end of each collective communication, the packets buffered in a global collective channel (identified with a global collective sequence number) are copied to the application buffer if needed. Collective operations with large messages can be divided into multiple collective operations.

With the inherent synchronization of all-to-all broadcast, the completion of an all-to-all broadcast operation signifies to a node that all the other nodes have at least

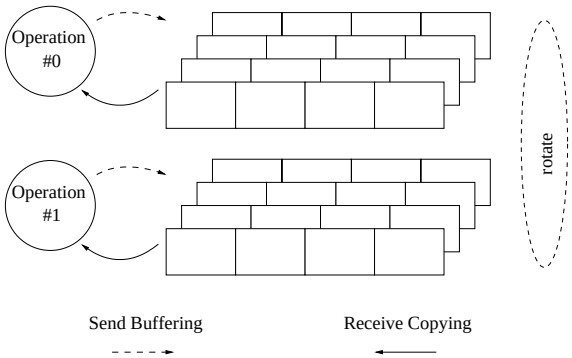


Fig. 2. Buffer Management for NIC-Based All-to-All Broadcast

reached the last all-to-all broadcast operation. When all-to-all broadcast requests are generated only after the local completion of previous broadcast requests, the following two conditions are always ensured.

1. At any moment, there is at most one outstanding all-to-all broadcast operation for which packets needs to be sent out.
2. At any moment, there are at most two outstanding all-to-all broadcast operations for which packets need to be received.

These conditions together simplify the management of all-to-all broadcast buffer channels. At any moment, only two buffer channels need to be provided for outstanding all-to-all broadcast operations. The use of these channels is inherently synchronized as operations rotate through them. Figure 2 shows the management of all-to-all broadcast buffers with two channels. Data to send can be buffered into the channel before it is sent to avoid having to register the memory. A message may be received in the buffer channel if it arrives before the corresponding request is posted, or if user applications choose to use the buffer channel and avoid the memory registration cost. At the end of the operation, the buffered data can then be copied out the channel.

4.3. All-to-All Broadcast Communication

In this subsection, we first describe the need for a different queuing and processing mechanism for all-to-all broadcast requests. Then we provide the design of two all-to-all broadcast algorithms: recursive doubling [12] and concurrent broadcasting.

4.3.1. All-to-All Broadcast Request Queuing and Processing

As described in the overview section, each request has to go through different queues, waiting for the send

buffers to become available before it can send out one of its packets and record its progress. This request is not marked as complete until all acknowledgments come back for the matching send records. This is the typical processing for point-to-point communications, and it is generally fair and efficient. However, since all-to-all broadcast needs to transmit the same data to multiple destinations, applying the same processing mechanism imposes unnecessary delays in the progress of an all-to-all broadcast operation. For example, the arrived message cannot immediately lead to the transmission of the next message until a send packet is available and the corresponding request and its queue get a turn to transmit.

Thus it is desirable to have a separate queue for all-to-all broadcast requests and manage their communication state to different peers collectively. In doing so, not only can all-to-all broadcast requests skip the other queues and get processed faster, but also the complexity in the management and ordering of collective operations can be reduced. Because the ordering is no longer forced with respect to other requests in different queues, transmitting a packet to another destination can be accomplished by retransmitting a send packet without claiming another send buffer. Similarly, a received packet can be forwarded to other destinations immediately from the receive buffer [15]. These communication processing techniques can improve the performance of the NIC-based all-to-all broadcast.

4.3.2. Recursive Doubling Algorithm

In the recursive doubling algorithm [12], each process pairs with a peer process through bit operations and recursively doubles the exchanged message size at each step. It takes $\log N$ steps for a system size of N nodes. This algorithm is well suited for small messages because it can combine messages into a single packet, thus reducing the number of packets to be processed, and improving the communication performance. However, large packets cannot be efficiently buffered and combined at the NIC (due to slow NIC memory copy speed or insufficient NIC buffers); rather, this algorithm has to buffer the intermediate data by copying the received data, using DMA, to the host memory and then copying it back to the NIC for the next step of communication. It thus does not have the benefit of avoiding round-trip PCI bus traffic. We explore this algorithm in our design in order to minimize the latency and shed more light on the NIC communication processing.

4.3.3. Concurrent Broadcasting Algorithm

The concurrent broadcasting algorithm broadcasts the data from each node to all the other nodes. Using the distributively maintained binomial tree topology, each NIC becomes a root for a different binomial spanning tree and broadcasts its packets to the other nodes. Having received or completed sending a data packet, a NIC needs to reuse the data packet for fast forwarding or resending. However, packets can come from any source, and the current NIC has to perform different roles for the broadcasting of different incoming packets. So the NIC has to decide whether the packet needs to be sent to a next destination and to which one. This process has to be done efficiently and with information from the packet. To this purpose, we have created a form of hopping packets to facilitate the traversal of a packet in its own broadcast spanning tree. Each packet is attached with a two-tuple flag to identify the broadcast spanning tree and the position of its traversal in the tree. As shown in Figure 3, in a flag (i, j), i denotes the rank of the original NIC for this packet, and j the log of the distance it has traversed for the current hop. The distance is measured as the difference between the ranks of NICs. The current NIC finds out the next destination of a hopping packet as $(|rank - i| + 2^j)$. If it is not less than the size of the group, then there is no need for the packet to make any further hops. Figure 3 shows how the packets are broadcasted for an all-to-all broadcast operation with hopping packets (only two broadcast spanning trees for two NICs and the hopping packets are given, to avoid complicating the graph).

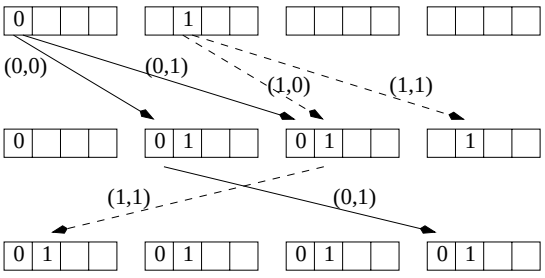


Fig. 3. Concurrent Broadcasting Algorithm for NIC-Based All-to-All Broadcast (showing only two nodes broadcasting)

The all-to-all broadcasting algorithm exploits numerous packet forwarding and resending techniques and reduces much of the traffic over the PCI bus. However, each NIC has to maintain the communication state about the amount of data that has arrived from each peer NIC. This requirement has two disadvantages. First, for small-message all-to-all broadcast operations, it does not combine the data into larger packets. Thus, for a system size of N nodes, each NIC has to receive $(N - 1)$ packets, and forward many received packets. So, for small-message all-to-all broadcast operations, the re-

quired processing time increases linearly with the system size, compared to $\log N$ packets with the recursive doubling algorithm. Second, the resource requirement for maintaining the communication state increases linearly with the number of peer NICs and can lead to a scalability constraint. To reduce this resource requirement, we use a status bit-vector to record the communication state. An additional integer can be used to count the number of packets arrived. This approach reduces both the memory requirement and the NIC processing time, because only when all expected packets have not arrived in time is this bit vector checked to find out the missing packets.

4.4. Reliability

Myrinet NIC Control Program (MCP) provides error control to ensure the reliable delivery of packets. For each packet transmitted, one send record is created. An acknowledgment must be returned by the receiver in order for the sender to release a send record created earlier. When a sender NIC fails to receive the ACK within a timeout period specified in the send record, it retransmits the packet. If the existing error control mechanism is used for all-to-all broadcast operation, each NIC must create $\mathrm{O}(N)$ number of send records and return $\mathrm{O}(N)$ number of acknowledgments.

An approach called receiver-driven retransmission is provided to ensure reliable delivery of all-to-all broadcast messages. With the recursive doubling all-to-all broadcast algorithm, one NIC communicates with only $\mathrm{O}(\log N)$ number of peers. Detailed bookkeeping is possible for incoming packets from different peers without leading to a scalability constraint. For this algorithm, arrays of records are used to record the dynamics of packets that have been sent to, or received from each peer NIC. With the concurrent broadcasting algorithm, however, the same approach is not scalable since the number of records needed is $\mathrm{O}(N)$. For scalable bookkeeping with the concurrent broadcasting algorithm, we provide only a bit vector as a send record to keep track of the arrival of packets. This allows at most one packet from each sender to be sent at a time. Larger all-to-all broadcast messages have to be fragmented and completed with multiple all-to-all broadcast operations. Note that all this communication state information is maintained at the NIC; it is not carried along with any packet. With the receiver-driven approach, the receiver NICs of the all-to-all broadcast messages no longer need to return acknowledgments to the sender NICs. If any expected all-to-all broadcast message is timed out, a NACK is generated by the receiver NIC and sent to the corresponding sender NIC. The sender NIC then retransmits the all-to-all broadcast message. For the all-to-all broadcast algorithm, the sender NIC is the parent in the binomial tree that is supposed to send that packet. The NACK can be propagated up further, or even to the original root for the packet, if the parent NIC does not have the packet. If the sender NIC is congested by packets from other communication traffic, however, the receiver has to keep retransmitting the NACK. We choose a rather large timeout value to reduce the frequency of the NACK retransmission and allow the network to recover from a congested state. This reliability scheme allows fast normal communication processing for a low error rate interconnect, such as Myrinet, while still maintaining a sentinel mechanism for error recovery.

5. Implementation of NIC-Based All-to-All Broadcast over Myrinet/GM

In this section we describe the implementation of the NIC-based all-to-all broadcast in Myrinet/GM. Our implementation is based on gm-2.0.3_Linux. We focus on the initiation of NIC-based all-to-all broadcast support and communication processing.

5.1. Group Initiation

To initialize NIC-based all-to-all broadcast support within a group, a list of peer node IDs describing the topology of the group is first created and posted to the NIC, along with the preregistered system buffer. The preallocated buffer is divided into two all-to-all broadcast channels. A contiguous 8 KB staging buffer is also preallocated at the NIC to allow the group to temporarily buffer received packets. At the end, a group handle is returned to the user application for identifying the group.

5.2. All-to-All Broadcast Communication

An interface function, `gm_gossip()`, is added to the GM API. User applications generate all-to-all broadcast requests by specifying the group handle, the algorithm to use, the sequence number, and send and receiver buffer information. Having detected the all-to-all broadcast request, each node prepares the packet and transmits the packet to the corresponding NIC according to the specified all-to-all broadcast algorithm. At the receive side, as packets come in, the receiving NIC either forwards the packets to other NICs, in the case of the concurrent broadcasting algorithm, or updates the communication state and proceeds to the next communication steps in the case of the recursive doubling algorithm. When an all-to-all broadcast request is com-

pleted, a new receive event is generated to the host receive queue. A user application then detects the completion through such an event.

6. Performance Evaluation

To evaluate the performance of our protocol, we conducted experiments on a 16-node cluster of 512 MB DRAM dual-SMP 1 GHz Pentium-III, each equipped with 33 MHz/32 bit PCI bus. Myrinet NICs on this cluster have 133 MHz LANai 9.1 processors and 2 MB SRAM. All nodes are connected to a Myrinet 2000 network. Our NIC-based implementation over Myrinet is based on GM-2.0.3. Our modification is done by leaving the code for other types of communications mostly unchanged. Our evaluation indicates that the modification has no noticeable impact on the performance of other types of communication. To avoid any possible impact from the network topology and the allocation of nodes, we initially performed our experiments with random permutation of the nodes. We observed only negligible variations in the performance results.

The NIC-based all-to-all broadcast performance is measured as the time between when an all-to-all broadcast request is sent to the NIC and when its completion is detected through a receive event. A microbenchmark is used to measure the average time of 5,000 iterations after the first 20 warm-up iterations. The bandwidth is taken as the total number of bytes broadcast by each process divided by the time to perform an all-to-all broadcast operation. From the two typical host-based all-to-all broadcast algorithms, our experiments indicate that ring-based pipelining provides better performance than the does recursive doubling. Thus we choose the recursive doubling algorithm in our microbenchmark for measuring host-based all-to-all broadcast performance.

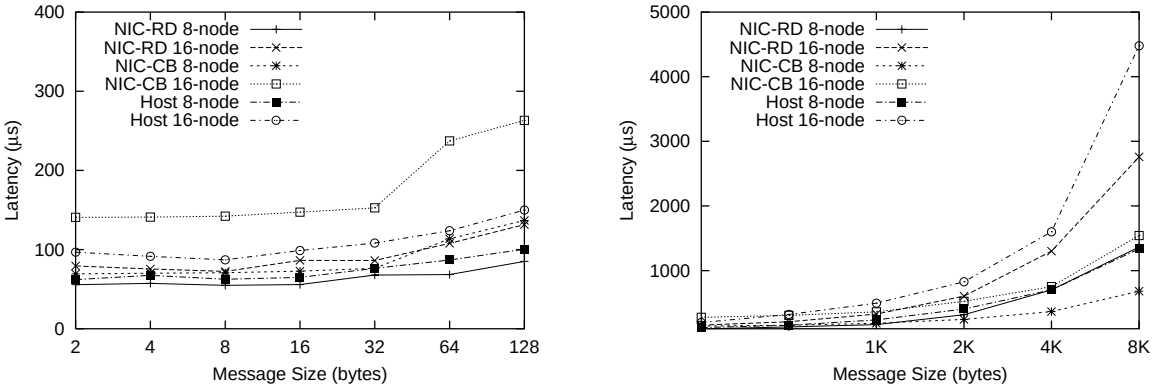
6.1. Latency and Bandwidth

Figure 4 shows the latency comparisons between the host-based all-to-all broadcast operation and the NIC-based all-to-all broadcast with the concurrent broadcasting (NIC-CB) and recursive doubling (NIC-RD) algorithms. As shown in Figure 4(a), for small messages, NIC-RD provides the best performance, compared to NIC-CB or the host-based all-to-all broadcast. The reason is that the NIC-RD algorithm can enjoy the benefits of fast communication processing at the NIC, while, at the same time, it is able to combine small messages into larger packets and does not suffer from having to process a linear scaling number of packets from all the other processes, as is the case for NIC-CB. In contrast, for large messages, the NIC-CB algorithm provides the best performance compared to NIC-RD or the host-based all-to-all broadcast. The reason is that the combined messages can no longer fit into a single MTU and they are still fragmented into packets at the NIC, so the recursive doubling algorithm no longer benefits from message combining, whereas NIC-CB still benefits from fast forwarding and retransmitting of the received (or transmission completed) packets. Both NIC-based algorithms perform better than the host-based all-to-all broadcast for large messages. Figure 4(b) shows the latency comparisons for large messages. NIC-based algorithms can improve performance by a factor of 3.

Figure 5 shows the bandwidth comparisons between the host-based and the NIC-based all-to-all broadcast algorithms. The bandwidth performance for the host-based all-to-all broadcast drops with multipacket messages (greater than 4 KB). In contrast, the NIC-based all-to-all broadcast operations are able to sustain their bandwidth performance. Compared to NIC-RD, NIC-CB can achieve better bandwidth with the benefits from fast packets forwarding and retransmitting. Figure 6 shows the performance improvement factor of the NIC-based all-to-all broadcast algorithms. Over the 16-node cluster with LANai 9.1 cards, the concurrent broadcasting algorithm provides an improvement factor of up to 3.01 for large messages, and the recursive doubling algorithm provides an improvement factor of up to 1.54 for small messages.

6.2. Scalability

Figure 7 shows the scalability comparisons between the host-based and the NIC-based all-to-all broadcast algorithms with 4 byte small messages and 4 KB large messages. For small messages, NIC-RD provides the best scalability because it benefits from offloaded communication processing and combining of small messages into larger packets. In contrast, NIC-CB performs the worst as the system size increases. The reason is that it does not combine small messages and hence the communication processing time for a large number of packets dominates over its benefits of message forwarding. These results are shown in Figure 7(a). For large messages, the NIC-CB algorithm provides the best scalability because it has the benefits the communication offloading and also the benefits of fast message forwarding. The other two algorithms do not have these features and have lower scalability, while the NIC-based all-to-all broadcast with recursive doubling still provides better scalability than the host-based all-to-all broadcast because of communication offloading. These results are shown in Figure 7(b).



(a) Small Messages (b) Large Messages

Fig. 4. All-to-All Broadcast Latency Comparisons of NIC-Based Operations with the Concurrent-Broadcasting Algorithm (CB) and the Recursive-Doubling (RD) Algorithm to Host-Based operations

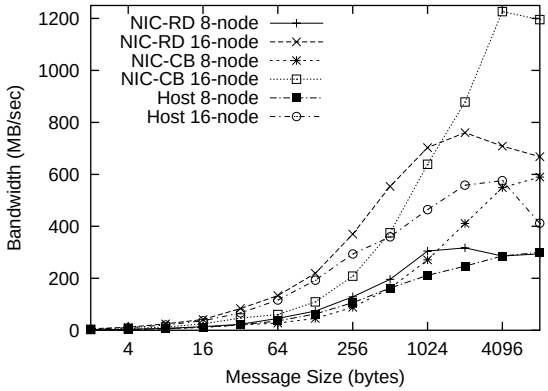


Fig. 5. Bandwidth Comparisons of NIC-Based All-to-All Broadcast with the Concurrent Broadcasting Algorithm (CB) and the Recursive Doubling Algorithm (RD) to Host-Based All-to-All Broadcast

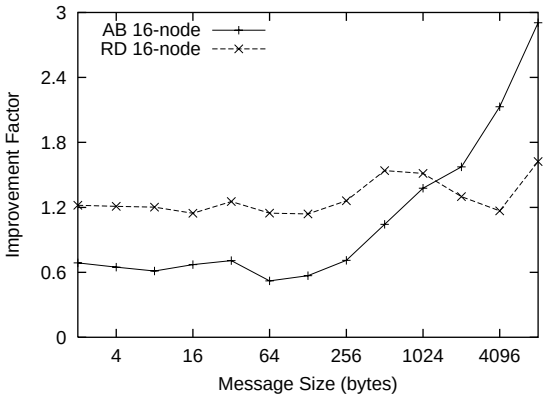
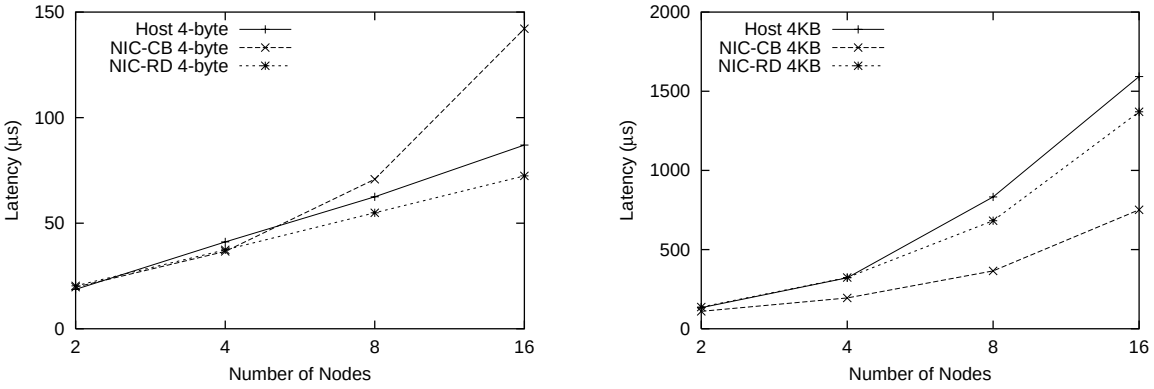


Fig. 6. Improvement Factor for NIC-Based All-to-All Broadcast with the Concurrent Broadcasting Algorithm (CB) and the Recursive Doubling Algorithm (RD)

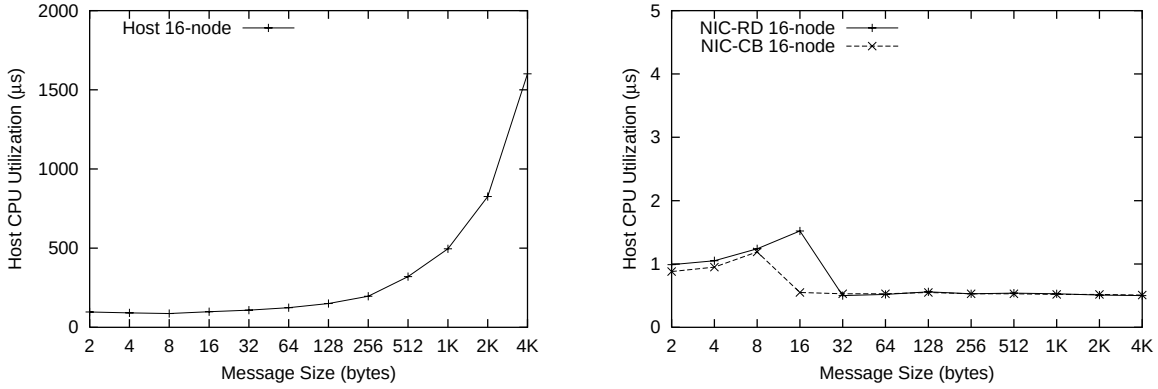
6.3. Host CPU Utilization

One of the major benefits of NIC-based all-to-all broadcast is that it has low host CPU utilization. With host-based all-to-all broadcast, the host process must constantly poll for the arrival of messages and trigger the next step for the all-to-all broadcast operation. The host CPU is largely occupied during the all-to-all broadcast operation. Figure 8(a) shows the host CPU utilization of host-based all-to-all broadcast over 16 nodes. In contrast, with the NIC-based all-to-all broadcast, once the host process sends its request to the NIC, it is free to perform other useful computation. To determine the host CPU utilization for the NIC-based all-to-all broadcast, we measured the effective time that the host CPU spends on posting the all-to-all broadcast request to the NIC and processing the arrival of a all-to-all broadcast receive event. Figure 8(b) shows the average host CPU utilization for the NIC-based all-to-all broadcast operations with both the concurrent broadcasting and the recursive doubling algorithms over various numbers of nodes. For the NIC-based all-to-all broadcast algorithms, the host CPU utilization is mostly constant over various numbers of nodes and different sizes of messages. The reason is that the time to post a send request to the NIC and the time to process a receive event does not vary with respect to the change of message size or system size. Note that for small messages the CPU utilization is higher. The reason is that for small messages the last-received data is DMAed into the receiver queue along with the event, to improve the overall performance. The host process thus has to perform an additional memory copy to obtain the message. Also note that user applications can perform nonblocking NIC-based all-to-all broadcast, where pro-



(a) Small Messages (b) Large Messages

Fig. 7. Scalability Comparisons of the NIC-Based All-to-All Broadcast with Concurrent Broadcasting Algorithm (CB) and Recursive Doubling Algorithm (RD) to the Host-Based All-to-All Broadcast



(a) Host-Based All-to-All Broadcast (b) NIC-Based All-to-All Broadcast

Fig. 8. Host CPU Utilization Comparison of the NIC-Based All-to-All Broadcast with the Concurrent Broadcasting Algorithm (CB) and the Recursive Doubling (RD) Algorithm to the Host-Based All-to-All Broadcast

cesses do not wait for the result from the NICs after they have posted the requests. Instead, these processes can perform useful computation that does not depend on the results, and they read the result from the NICs only when they need the data. Therefore, the NIC-based all-to-all broadcast allows user applications to achieve high utilization of the computation resources with its low CPU utilization.

7. Conclusions and Future Work

In this paper, we have explored the challenges of supporting efficient all-to-all broadcast taking the advantage of NIC programmability. We have proposed strategies for scalable binomial tree-based topology management, efficient global buffer management, efficient communication processing, and message reliability. We have designed scalable, high-performance NIC-based all-to-all broadcast with concurrent broadcasting and recursive doubling algorithms. The resulting NIC-based operations have been implemented and incorporated into a NIC-based collective protocol [14] over Myrinet/GM.

Compared to the host-based all-to-all broadcast, the NIC-based all-to-all broadcast operations improves all-to-all broadcast bandwidth over 16 nodes by a factor of 3. The NIC-based all-to-all broadcast with recursive doubling algorithm is more scalable for small messages, and the concurrent broadcasting algorithm more scalable for large messages. Furthermore, the NIC-based all-to-all broadcast operations have very low host CPU utilization, which allows user applications to achieve high CPU utilization when the operation is used in a non-blocking manner.

We intend to extend the design of this NIC-based all-to-all broadcast to MPI [7] layer and study its benefits to applications. In addition, we intend to study its benefits in other programming models, such as distributed shared-memory [8], and their applications.

Acknowledgment

We thank Argonne National Laboratory for providing access to the Chiba City cluster. We also thank Myricom for providing access to GM-2 firmware source code.

Additional Information – Additional papers related to this research can be found on the following Web pages: Network-Based Computing Laboratory (http://nowlab.cis.ohio-state.edu) and Parallel Architecture and Communication Group (http://www.cis.ohio-state.edu/∼panda/pac.html).

References

[1] R. A. Bhoedjang, T. Ruhl, and H. E. Bal. Efficient Multicast on Myrinet Using Link-Level Flow Control. In *27th International Conference on Parallel Processing*, 1998.

[2] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W.-K. Su. Myrinet: A Gigabit-per-Second Local Area Network. *IEEE Micro*, 15(1):29–36, 1995.

[3] D. Buntinas and D. K. Panda. NIC-Based Reduction in Myrinet Clusters: Is It Beneficial? In *SAN-02 Workshop (in conjunction with HPCA)*, Feb. 2003.

[4] D. Buntinas, D. K. Panda, J. Duato, and P. Sadayappan. Broadcast/Multicast over Myrinet Using NIC-Assisted Multidestination Messages. In *Communication, Architecture, and Applications for Network-Based Parallel Computing*, pages 115–129, 2000.

[5] D. Buntinas, D. K. Panda, and P. Sadayappan. Fast NIC-Level Barrier over Myrinet/GM. In *Proceedings of International Parallel and Distributed Processing Symposium*, 2001.

[6] A. Moody, J. Fernandez, F. Petrini, and D. Panda. Scalable NIC-based Reduction on Large-Scale Clusters. In *SC '03*, November 2003.

[7] MPI Forum. MPI: A Message Passing Interface. In *Proceedings of Supercomputing '93*, November 1993.

[8] R. Noronha and D. K. Panda. Implementing TreadMarks over GM on Myrinet: Challenges, Design Experience, and Performance Evaluation. In *International Workshop on Communication Architecture for Clusters, in Conjunction with International Parallel and Distributed Processing Symposium*, April 2003.

[9] F. Petrini, W.-C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network: High Performance Clustering Technology. *IEEE Micro*, 22(1):46–57, January-February 2002.

[10] F. Petrini, D. Kerbyson, and S. Pakin. The Case of the Missing Supercomputer Performance: Achieving Optimal Performance on the 8,192 Processors of ASCI Q. In *SC '03*, November 2003.

[11] Quadrics Supercomputers World, Ltd. Quadrics Documentation Collection. http://www.quadrics.com/onlinedocs/Linux/html/index.html.

[12] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Proceedings of EuroPVM/MPI '03*, September 2003.

[13] K. Verstoep, K. Langendoen, and H. E. Bal. Efficient Reliable Multicast on Myrinet. In *the Proceedings of the International Conference on Parallel Processing '96*, pages 156–165, 1996.

[14] W. Yu, D. Buntinas, R. L. Graham, and D. K. Panda. Efficient and Scalable Barrier over Quadrics and Myrinet with a New NIC-Based Collective Message Passing Protocol. In *Workshop on Communication Architecture for Clusters, in Conjunction with International Parallel and Distributed Processing Symposium '04*, April 2004.

[15] W. Yu, D. Buntinas, and D. K. Panda. High Performance and Reliable NIC-Based Multicast over Myrinet/GM-2. In *Proceedings of the International Conference on Parallel Processing '03*, October 2003.

[16] W. Yu, S. Sur, D. K. Panda, R. T. Aulwes, and R. L. Graham. High Performance Broadcast Support in LA-MPI over Quadrics. In *Los Alamos Computer Science Institute Symposium*, October 2003.