Scalable and High Performance NIC-Based Allgather over Myrinet/GM

Weikuan Yu Darius Buntinas Dhabaleswar K. Panda

OSU-CISRC-4/04-TR22
Technical Report

Scalable and High Performance NIC-Based Allgather over Myrinet/GM*

Weikuan Yu† Darius Buntinas‡ Dhabaleswar K. Panda†

Network-Based Computing Lab†
Dept. of Computer Science
The Ohio State University
{yuw,panda}@cis.ohio-state.edu

Argonne National Laboratory‡
Mathematics and Computer Science
Argonne, IL 60439
buntinas@mcs.anl.gov

Abstract

Programmable Network Interface Cards (NICs) have been leveraged to support efficient collective communication and synchronization. Previous studies have exploited NIC programmability to support efficient barrier, broadcast and reduce operations. This paper explores the design of NIC-based allgather with different algorithms. Along with these algorithms, salient strategies have been utilized to provide scalable topology management, global buffer management, efficient communication processing, as well as flow control and reliability. The resulting allgather algorithms have been incorporated into a NIC-based collective protocol over Myrinet/GM. Compared to the host-based allgather operation, over 16 nodes the NIC-based allgather operations improves allgather performance by a factor up to 3.01. Furthermore, the NIC-based allgather operations have better scalability to large systems and very little host CPU utilization. To the best of the authors' knowledge, this paper is the first in the literature to report efficient NIC-based allgather algorithms over Myrinet/GM.

1 Introduction

Modern interconnects rely heavily on communication processing at Network Interface Cards (NICs) to achieve high performance. The programmable processors in some modern interconnects, including Myrinet [2] and Quadrics [14], have been further leveraged to offload communication processing and optimize collective communication and synchronization [1, 6, 7, 21, 11, 20]. Efforts [3, 12] to offload much of MPI processing stack have been initiated and are expected to be seen in the large scale production systems in the near future. Although the memory resource and computation power of NICs are largely limited, utilizing the NIC programmability will continue to be a trend to improve the performance of communication protocols.

Besides point-to-point communication, collective communication can consist up to 70% of the execution time of a scientific application [15]. High performance and scalable collective communication is an important factor to achieve good performance and increase the effective utilization of a high-end computing platform. Previous studies [19, 1, 6, 7, 5, 21, 11, 20] have investigated the benefits of NIC-based support for

*This research is supported in part by a grant from Los Alamos National Laboratory and National Science Foundation's grants #CCR-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.

collective operations, including barrier, broadcast and reduce. It has been shown that providing NIC-based collective operations is effective in reducing the host processor involvement [7, 21], avoiding round-trip PCI bus traffic [7, 21], increasing the tolerance to process skew [5, 21] and OS effects [11]. Together, these benefits contribute to the improvement of communication performance in terms of latency and bandwidth.

However, the repertoire of collective communication that has been studied and demonstrated as beneficial with the NIC-based support is largely limited to barrier, broadcast and reduce. The benefits of NIC-based support have not been exploited for some other useful operations. For example, no algorithms have been reported to provide an efficient NIC-based allgather algorithms. Offloading allgather to the NIC will have a greater impact on the NIC's limited memory and processor resources. It is challenging to exploit the benefits under these restrictions. One must provide mechanisms to handle more design challenges in order to avoid impacts to other message passing activities and provide an efficient NIC-based allgather operation. These include at least scalable group topology management, efficient buffer management, efficient communication processing, as well as flow control and reliability.

In this paper, we take on these challenges and attempt to design a scalable and high performance NIC-based allgather operation and incorporate it into a NIC-based collective protocol. We have proposed two allgather algorithms, one with all-to-all broadcast, another with recursive doubling. Along with these algorithms, salient strategies are provided to perform scalable binomial tree-based group topology, efficient global buffer management, fast forwarding of packets for efficient communication processing, as well as scalable and efficient flow control. The resulting allgather operations have been implemented and incorporated into a NIC-based collective protocol over Myrinet/GM.

Compared to the host-based allgather operation, over 16 nodes the NIC-based allgather operations improves allgather performance by a factor up to 3.01. Furthermore, the NIC-based allgather operations have better scalability to large systems and very low host CPU utilization. To the best of the authors' knowledge, this paper is the first in the literature to report efficient NIC-based allgather algorithms over Myrinet/GM.

The rest of the paper is structured as follows. In the next section, we describe related work and the motivation for a NIC-based collective protocol. Section 3 provides an overview of Myrinet/GM. Section 4 provides the design of a NIC-based allgather operation. The implementation of the algorithms over Myrinet/GM is provided in Sections 5. Experiments and results are provided in Sections 6. Finally, the conclusion is drawn in Section 7.

2 NIC-based Allgather

In this section, we first discuss potential benefits of the NIC-based allgather operation. Then we provide a brief introduction to related work on efficient collective communications taking advantages of NIC programmability.

2.1 Potential Benefits of the NIC-based Allgather

With NIC-based allgather, the communication processing for allgather is to be done completely by the NIC. Host CPU is not involved in the intermediate steps of allgather. Work [20] demonstrates that the barrier operation can benefit from efficient communication processing with a separate NIC-based collective protocol. Fig. 1 shows that allgather is targeted to be part of such a separate collective protocol at the NIC. The allgather operation is then presented as an application interface to user applications at the host. Further discussion on why the NIC-based collective protocol needs to provide its own communication processing is provided in Section 4. Offloading the allgather communication to the NIC and processing it with a separate collective protocol could potentially provide the following benefits.

Low Latency and High Bandwidth – By offloading the allgather communication processing, received data packets can be processed and forwarded directly without making a round-trip across PCI-bus.

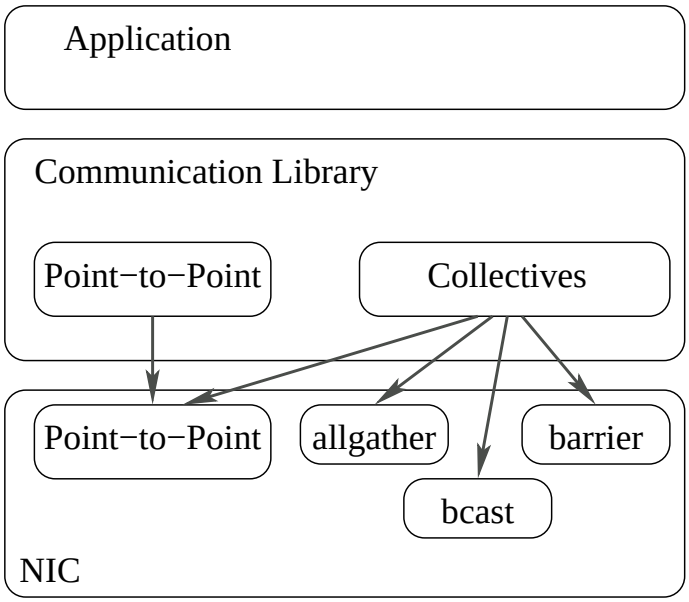


Fig. 1. NIC-Based Allgather in a NIC-Based Collective Processing Protocol

This means that the NIC can invoke next steps of an allgather operations sooner than the host-based collective operations. These advantages can speed up the allgather and contribute to its performance improvement in terms of latency and bandwidth.

High Scalability The NIC-based allgather also has the potential of increasing the scalability of this operation. Compared to the host-based allgather, the NIC-based allgather comes closer to utilizing the network capacity because of fewer trips across PCI bus and less involvement of host processors. In addition, with NIC-based allgather, flow control can be more tightly bound to the communication traffic and lead to sustained scalability.

Low Host CPU Utilization – Since the communication processing for an allgather operation is completely undertaken by the NIC, a user application only needs to invoke the allgather operation through the host-side interface. It can then proceed to do other computation that does not dependent on the results of the allgather operation, and check back later for the completion of the allgather operation. Because the host can overlap useful computation with the allgather operation, CPU utilization improves.

2.2 Related work on NIC-based collectives

As discussed in Section 1, offloading the communication processing and taking advantage of the NIC programmability will continue to be a trend to improve the performance of communication protocols. Previous work [19, 1, 6, 7, 21, 11, 20] have exploited the benefits of efficient collective communications by taking advantages of various amount of NIC support.

Work [19, 1, 21] have implemented different algorithms to provide efficient NIC-based broadcast support. All these three use NIC-based packet forwarding as one of the means to speed up broadcast, though they differ significantly in their reliability and flow control mechanisms. Work [7, 5, 11, 20] have also studied the benefits of offloading barrier and reduce operations to the NIC. In these work, they have shown 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 above has studied the benefits of allgather operations with NIC-based support.

The work [20] has 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. It remains to be investigated how other collective operations can fit into and benefit from this NIC-based collective protocol. Given the NICs' limitation and the versatility of collective operations, it would be unrealistic to provide complete support to all possible collective operations at the NIC. Thus it is necessary to identify a minimal and complete set of NIC-based collective that is also advantageous to be offloaded into the NIC. While collective operations can be classified into one-to-all, all-to-one and all-to-all categories [17, 4], allgather is a typical representative of the all-to-all category, and it can also be used to support other collective operations, e.g., all-to-all personalized broadcast and allreduce. Therefore it is natural to include allgather as one of the collective operations in the NIC-based collective protocol. In addition, an allgather operation essentially involves multiple parallel broadcast operations, each with every single participating process being the root. This suggests an efficient NIC-based allgather operation could also be possible by taking advantage of packet forwarding and resending as experienced with the NIC-based broadcast operations [19, 1, 7, 21].

In this work we attempt to investigate the feasibility and benefits of the NIC-based allgather. By studying this communication-intensive collective operation, we attempt to address more design issues related to an efficient NIC-based collective protocol, including group topology management, buffer management, scalable communication state maintenance/processing, as well as flow control and reliability. This work results in an efficient and scalable NIC-based allgather operation within the constraints of NIC resources. All the associated management mechanisms and the resulting allgather operation have been incorporated to the NIC-based collective protocol [20].

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/receive operation works as follows.

Sending a Message – To send a message, a user application generates a send descriptor, referred to as a *send event* in GM, to the NIC. The NIC translates the event to a *send token* (a form of send descriptor that 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 packet, i.e., the send buffer to accommodate the data. Then the data is DMAed from the host buffer into the *send packet* and injected into the network. The NIC keeps a *send record* of the sequence number and the time for each packet it has sent. If the acknowledgment is not received within the timeout period, the sender will retransmit the packet. When all the send records are acknowledged, the NIC will pass 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. An unexpected packet is 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 will also generate a *receive event* to the host process for it to detect the arrived message.

4 Design Challenges for the NIC-Based Allgather

In this section, we explore the design challenges for the scalable NIC-based allgather. Given the limited resource available at the NIC, minimizing the resource requirement and achieving good scalability is the foremost requirement for any collective algorithms to be a part of the NIC-based collective protocol. Within

this restriction, there are four major design issues to be considered. These include communication algorithms and group management, buffer management, communication processing, as well as flow control and reliability.

4.1 Communication algorithms and Group Topology

The choice of communication algorithms directly impacts the resource requirement of group management. A scalable NIC-based collective protocol must minimize linear scaling resource requirements. For clusters with thousands of nodes, placing the entire group membership information at the NIC imposes a large memory requirement, and the requirement would be even greater if the communication state for each peer NIC is to be maintained. To make the matters even worse, a single round of completion checking for the allgather operation is likely to take thousands of instructions. Thus it is important to investigate the allgather algorithms and group topology in order to minimize the group information and the associated resource requirements.

4.1.1 Binomial Tree-based Topology

Typically, collective operations over point-to-point links utilize various spanning trees to cover all the nodes. One node in a spanning tree communicates with its parent node and a limited number child nodes. The binomial tree is one of the most commonly used topology for collective communication. It provides two advantages over any other topology. On the one hand, it can be shared by barrier, broadcast (even with different roots), scatter/gather, allgather and other collective operations. On the other hand, the distance between any pair of direct communicating nodes is always some power of 2, thus updating the communication state of peers can be easily handled by the NIC processors as bit shifting operations (typically, NICs are not equipped with FPU). Thus the binomial tree-based topology is a good choice for a scalable group management. The entire tree topology can be distributed to all the nodes, each maintaining only $2 * logN$ entries.

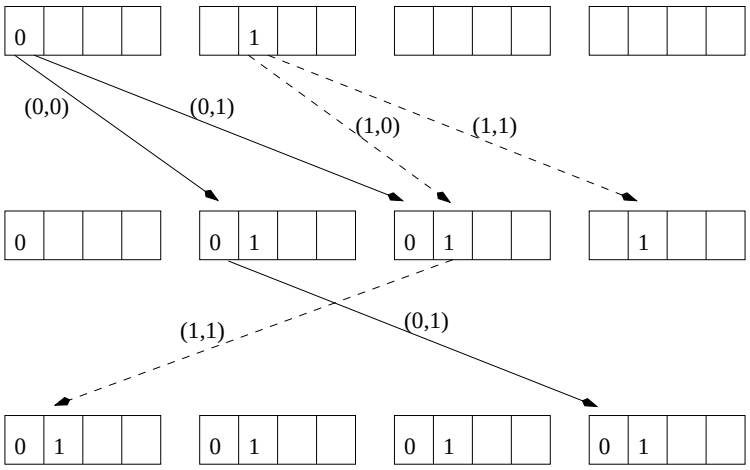


Fig. 2. All-to-all Broadcasting Algorithm for NIC-based Allgather (Showing only two nodes broadcasting)

4.1.2 Communication Algorithms: Recursive Doubling and Concurrent All-to-All Broadcasting

With the binomial tree based topology, one can implement the allgather operation with either a recursive doubling algorithm [18] or a concurrent all-to-all broadcasting algorithm. The recursive doubling algorithm

can be completed within only $logN$ steps, and it is scalable. However, if the received data can not be efficient buffered and combined at the NIC (for slow memory copy speed or insufficient NIC buffers), this algorithm has to buffer the intermediate data by copying (using DMA) the received data to the host memory and then copying it back to the NIC for the next step of communication. It then will not have the benefit of avoiding PCI-bus traffic.

The concurrent all-to-all broadcasting algorithm broadcasts the data from each node to all the other nodes. When having received or completed sending a data packet, a NIC need to reuse the data packet for fast forwarding or resending. This will reduce data copying or buffering traffic over the PCI bus (or to the NIC memory if using NIC processor cycles for buffering). However, within this algorithm, packets can come from any source and the current NIC will perform different roles for the broadcasting of the incoming packets. So the NIC has to decide whether the packets needs to be sent to a next destination and to which one. This has to be done efficiently and with only the information from the packet. To this purpose, we use a form of hopping packets, each attached with a two-tuple flag. As shown in Fig. 2, 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 group, then there is no need for the packet to make any further hops. Fig. 2 shows how the packets are broadcasted for an allgather operation with hopping packets (Only hopping packets from two NICs are given to avoid complicating the graph).

In contrast to the recursive doubling algorithm, in the all-to-all broadcasting algorithm, each NIC has to maintain the communication state about the amount of data that have arrived from each peer NIC. This algorithm has two disadvantages. For small message allgather operations, it does not combined the data into larger packets. Thus, for a system size of N nodes, each NIC has to receive (N-1) packets, plus it has to forward some of the received packets. So for small message allgather operations, the required processing time is linearly increasing with the number of system size, compared to $logN$ packets with the recursive doubling algorithm. Second, even if one provides only an integer (too small though) to record the amount of data arrived from each peer NIC, the resource requirement on the communication state is rather large, 4K bytes for a 1024-node system. Exploring both algorithms would shed more insights into their communication characteristics.

4.2 Buffer Management For NIC-based Allgather

Processes in parallel applications can reach the same collective communication at different time points. Packets can arrive the NIC before the host even posts the receive buffer for the corresponding collective operation. For barrier operations, a NIC-based collective protocol [20] can achieve efficiency without saving the packet's data. However, this is not allowed for other types of collective operations. To allow the NIC to still receive and forward the early arrived data packets, system buffer must be present for accommodating these packets. This can avoid having to drop packets and improve the the performance of collective communication and its tolerance process skew [21]. A common way of managing of collective buffers is to provide a global virtual memory [16, 22]. As shown in Fig. 3, we propose a global buffer management scheme for the NIC-based collective protocol, in which a host buffer is preallocated divided into multiple fixed-length collective channels. The order in which these channels can be used is globally synchronized across all the NICs when the collective protocol is initialized. If an unexpected packet arrives, the NIC will DMA the data to the corresponding collective channel, otherwise to the provided user buffer. Collective operations with large messages can be divided into multiple collective operations. These collective channels are used in a cyclic manner to buffer the packets for different collective operations. At the end of each collective communication, the packets buffered in a global collective channel (identified with a global collective sequence number) can be copied to the application buffer. By providing multiple collective channels, multiple

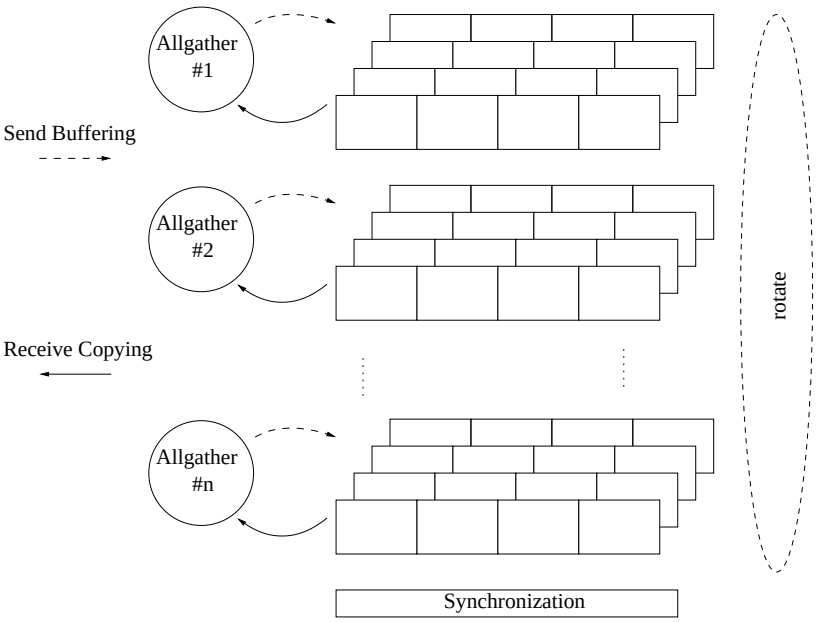


Fig. 3. Buffer Management for NIC-Based Allgather

collective operations can be performed in a pipeline. As collective operations are invoked to the NICs, one process may start a new collective operation while others are still working on the earlier operations. To avoid overwriting earlier packets that are still in a global collective channel, global synchronization among processes is enforced before a collective channel can be reused [22]. We can use an efficient barrier [20] for synchronization of the global collective channels. With multiple channels, the synchronization of previous channels and the associated cost can be overlapped with the next allgather operations.

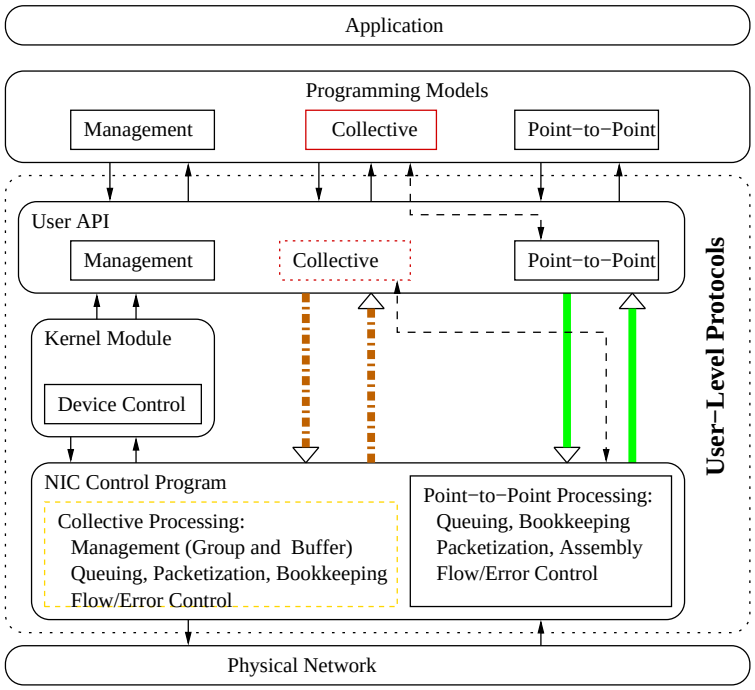


Fig. 4. A NIC-Based Collective Processing Protocol

4.3 Communication Processing for NIC-based Allgather

Within a communication protocol, a NIC typically must handle multiple communication requests to different peer NICs. Each request is placed in one of the connection queues to a particular NIC. For this request to get its turn, it must go through multiple queues and then wait for send buffers before transmiting its message. Once there are send buffers avaiable, data will be DMAed to the NIC and prepared as packets. Send records are maintained to bookkeep the progress of each request. The request is not marked as complete until all acknowledgments come back. As described in the overview section and also shown in Fig. 4, this is the typical communication processing for point-to-point communications, and it is rather fair and efficient. However, for the collective communication, much of the communication processing is rather redundant and inefficient [20], and is likely to be a hurdle to the maximum exposure of hardware capacity. For example, multiple DMAs across PCI-bus are invoked to send the same data to different destinations, and immediate resending or forwarding of packets are not available.

For the NIC-based allgather with recursive doubling algorithm, if the existing queues are not empty, the arrived message cannot immediately lead to the transmission of the next message until the corresponding request gets its turn in the relevant queues. This imposes unnecessary delay in the progress of an allgather operation. Thus it is desirable to have separate queues for allgather operations. The allgather data packets can then skip other queues and get transmitted in a much faster manner. In addition, a separated collective protocol reduces the complexity in the management and ordering of collective operations, compared to manage them inside point-to-point operations. On the other hand, to transmit a message, a NIC usually has to allocate a NIC send buffer, DMA a fragment of the message into this send buffer, packetize the buffer, then inject it into the network. For a NIC-based allgather with concurrent all-to-all broadcast, the same data is usually transmitted to multiple different receiver NICs. Thus NIC-based allgather can have a much more efficient transmission scheme if the packets with data can be repeatedly injected into the network. By doing this, transmission of the same message 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, skipping the time to wait for send buffers and the time to wait for the arrival of the whole message with the host-based allgather. These efficient processing mechanisms can improve the performance of the NIC-based allgather.

4.3.1 Flow control and 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 earlier created send record. 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 was to be used for allgather operation, each NIC must create $O(N)$ number of send records, and return $O(N)$ number of acknowledgments.

In addition, flow control is important to the allgather communication since a large cluster with hundreds of nodes can potentially generate a substantially large number of packets to overflow the limited number of receive buffers at the NIC. But there are no global flow control mechanisms in MCP to prevent receivers from being congested by multiple senders. A lot of bandwidth can be wasted to retransmit the packets that are lost due to congestion. It is desirable to provide a global flow control mechanism to achieve scalable allgather operations.

With the recursive doubling allgather algorithm, one NIC is only potentially communicating with $O(logN)$ number of peers. Providing more detailed bookkeeping for the progress of an allgather operation will not lead to another scalability constraint. For this algorithm, an array of records can be used to record the dynamics of packets that have been sent or received from each peer NIC. However, with the all-to-all broadcast algorithm, it is not scalable to use the same approach since the number of records needed is $O(N)$. For scal-

able bookkeeping within the all-to-all broadcast algorithm, we provide only a bit vector as a send record to keep track of the arrival of allgather messages. This has two implications. On one hand, this allows only at most one packet from each sender to be sent at a time. To compensate for this, a larger MTU can be used for transmitting allgather packets. Even larger allgather messages can be fragmented, and completed with multiple allgather operations. On the other hand, this provides a global flow control to the allgather traffic.

For both algorithms, an approach called receiver-driven retransmission is provided to ensure reliable delivery of allgather messages. The receiver NICs of the allgather messages no longer need to return acknowledgments to the sender NICs. If any expected allgather message is timed out, a NACK will be generated from the receiver NIC to the corresponding sender NIC. The sender NIC will then retransmit the allgather 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. Taken together, these mechanisms provides global flow control to the allgather operations and still achieves the reliable delivery in a scalable manner.

5 Implementation of the Proposed NIC-Based Protocol over Myrinet/GM

In this section we describe the implementation of the NIC-based allgather in Myrinet/GM. Our implementation is based on gm-2.0.3_Linux. The allgather operation consists of three major aspects, group initiation, allgather communication and collective channel recycling. The recycling of collective channels is achieved by forcing a barrier-based [20] synchronization before the reuse of previous channels. The group initiation and communication processing are described as follows.

5.1 Group Initiation

To initialize the NIC-based allgather resources within a group, a list of peer node ID describing the topology of the group was first created and posted to the NIC, along with the preregistered system buffer. The preallocated buffer is divided into to a fixed number of collective channels, depends on the maximum number of non-blocking collective operations. A contiguous 8KB staging buffer is also preallocated at the NIC to allow the group to temporarily buffer received packets. When a receiver is expecting to receive multiple small packets, it can also save the data into this preallocated NIC buffer and later DMA the whole message to the host memory. This can save the number of DMA requests. At the end, a group handle is returned to the user application for identifying the group.

5.2 Allgather Communication Processing

As shown in Fig. 5, to start an allgather operation, each process posts an allgather request to the NIC. When having detected the allgather event, each node prepares the packet and transmits the packet to the corresponding node according to the specified allgather algorithm. Each packet contains the associated group information and the packet type. With recursive doubling algorithm, the packet contains an additional flag indicating the current step. Packets are DMAed into the user buffer directly and assembled for the next step of doubling if needed. With the all-to-all broadcast algorithm, each packet is attached with a two-tuple flag as shown in Fig. 2. When having received or completed sending a data packet with a flag (i,j), the receiver triggers a DMA request to copy the message into host buffer and it determines whether it needs to forward the packet by checking whether $|rank - i| + 2^j$ is less than the group size. To reduce the number of small DMA requests, the receiver NIC can also buffer the data temporarily into the preallocated NIC staging buffer, if a receiver is expecting to receive multiple small packets (currently less than or equal to 64 bytes). When all the expected packets are received, The buffered data are DMAed to the host buffer. As the incoming and outgoing packets are processed, the communication state is monitored and updated into a status bit vector as shown in Fig. 5. A status bit vector of only 32 integers can keep track of whether packets from 1024 NICs have come. An additional integer can be used to count the number of packets arrived. Only

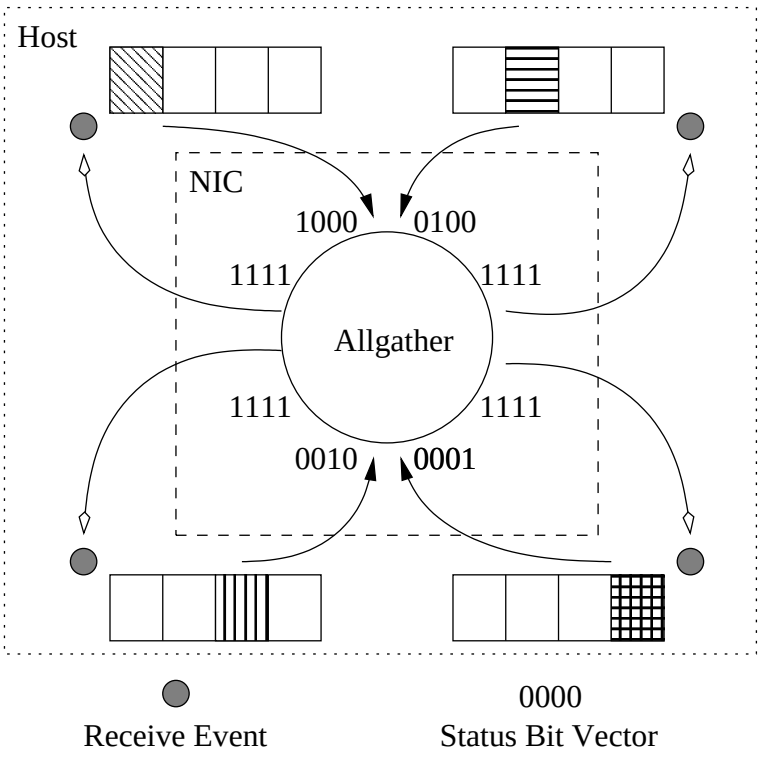


Fig. 5. NIC-Based Allgather Communication

when expected packets have not all arrived in time is this bit vector checked to find out the packets that have not come. This reduces both the memory requirement and NIC processing time. When an allgather request is completed, a new receive event is generated to the host receive queue. A user application then detects the completion of a collective communication by checking the host receive queue for the arrival of an allgather receive event shown in Fig. 5 as a filled circle.

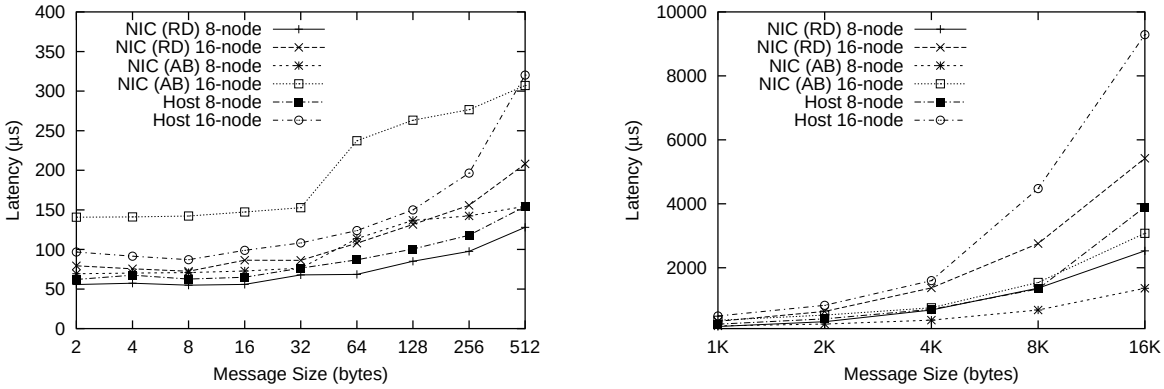
6 Performance Evaluation

In this section, we describe the performance evaluation of our protocol. Experiments were conducted on a 16-node cluster of 512MB DRAM dual-SMP 1GHz Pentium-III. Myrinet NICs on this cluster have 133MHz LANai 9.1 processors and 2MB SRAM. This cluster is connected to a Myrinet 2000 network. Our NIC-based implementation over Myrinet is based on GM-2.0.3.

Our modification to GM was done by leaving the code for other types of communications mostly unchanged. Our evaluation indicated that it has no noticeable impact on the performance of other types of communications. To avoid any possible impact from the network topology and the allocation of nodes, initial experiments were performed with random permutation of the nodes. We observed only negligible variations in the performance results. We also implemented micro-benchmarks to measure the performance of the host-based allgather operation with the recursive doubling algorithm. We chose the recursive doubling algorithm because our experiments indicated that it performs better than the host-based allgather operation with ring-based pipelining algorithm. Unless otherwise specified, the following experiments used recursive doubling for the host-based allgather. All experiments were performed by having the processes execute consecutive allgather operations with the 20 warmup iterations at the beginning.

6.1 Latency and Bandwidth

Fig. 6 shows the latency comparisons between the host-based allgather operation and the NIC-based allgather with all-to-all broadcast (AB) or recursive doubling (RD) algorithms. As shown in Fig. 6(a), for



(a) Small Messages (b) Large Messages

Fig. 6. The Latency Comparisons of the NIC-based Allgather with All-to-All Broadcasting Algorithm (AB) and Recursive Doubling (RD) Algorithm, to the Host-Based (HB) Allgather

small messages, the NIC-based allgather with recursive doubling algorithm provides the best performance compared to the all-to-all broadcasting algorithm or the host-based allgather. This is because the NIC-based allgather with recursive doubling algorithm can have benefits of fast allgather communication processing at the NIC, in the mean time it is able to combine small messages into larger packets and does not suffer from having to process $O(N)$ number of packets. In contrast, for large messages, the NIC-based allgather with all-to-all broadcasting algorithm provides the best performance compared to the recursive doubling algorithm or the host-based allgather. This is because the combined messages can no longer fit into a single MTU and it is still fragmented into packets at the NIC, so the recursive doubling algorithm no longer has the benefit from message combining. For the same messages, the NIC-based all-to-all broadcasting benefits greatly from fast forwarding and retransmitting of the received (or transmission completed) packets. For large messages, both algorithms performs better than the host-based allgather. Fig. 6(b) shows the latency comparisons for large messages. This is a 3.38 factor of improvement over host-based barrier operations.

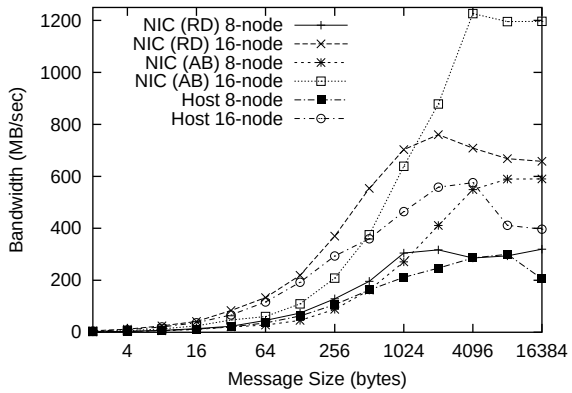


Fig. 7. The Bandwidth Comparisons of the NIC-based Allgather with All-to-All Broadcasting Algorithm (AB) and Recursive Doubling (RD) Algorithm, to the Host-Based (HB) Allgather

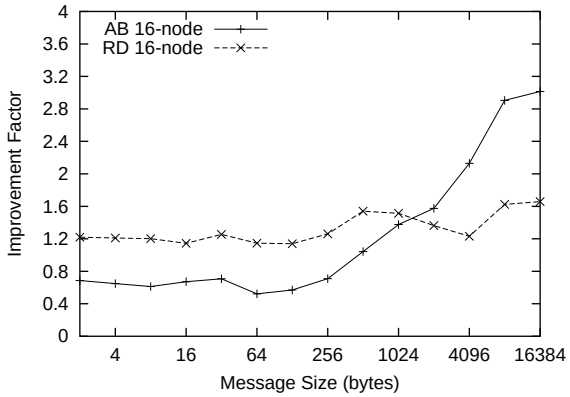
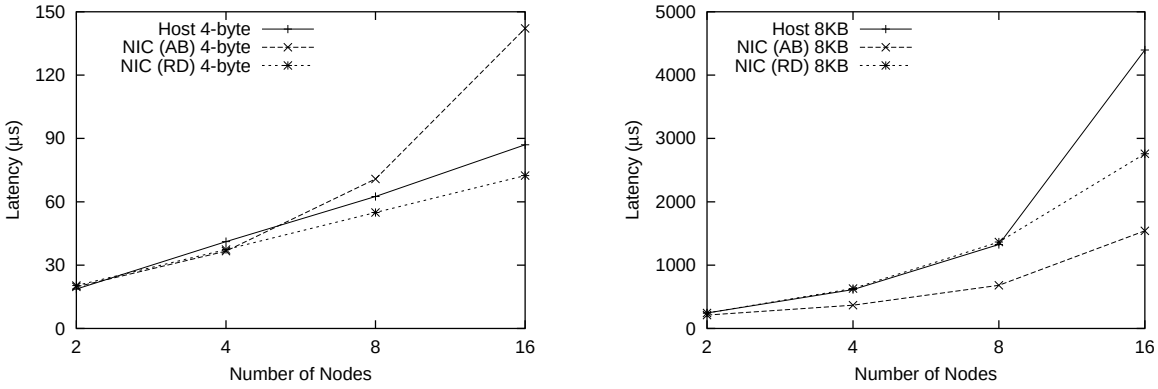


Fig. 8. The Factor of Improvement for NIC-Based Allgather with Different Algorithms. AB: all-to-all broadcasting; RD: recursive doubling

Fig. 7 shows the bandwidth comparisons between the host-based allgather operation and the NIC-based allgather with all-to-all broadcast (AB) or recursive doubling (RD) algorithms. Interestingly, the bandwidth

performance for the host-based allgather drops with multi-packet messages (greater than 4KB). This suggests that the Myrinet is not able to sustain its maximum bandwidth with its flow control-absent communication protocol. In contrast, the NIC-based allgather is able to maintain its maximum bandwidth performance which suggests that the global flow control forced in the NIC-based allgather algorithms improves the performance. The all-to-all broadcasting algorithm can achieve better bandwidth with the benefits from fast packets forwarding and retransmitting. Fig. 8 shows the performance improvement factor of the NIC-based allgather algorithms. Over the 16-node cluster with LANai 9.1 cards, the all-to-all 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



(a) Small Messages (b) Large Messages

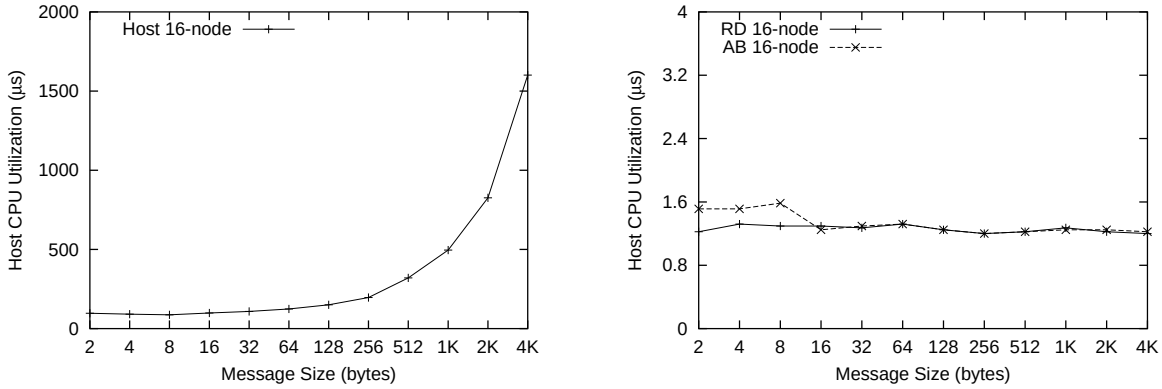
Fig. 9. Scalability Comparisons of the NIC-based Allgather with All-to-All Broadcasting Algorithm (AB) and Recursive Doubling Algorithm, to the Host-Based (HB) Allgather

Fig. 9 shows the scalability comparisons between the host-based allgather operation and the NIC-based allgather with all-to-all broadcast (AB) or recursive doubling (RD) algorithms, with 4-byte small messages and 8KB large messages. For small messages, NIC-based allgather with recursive doubling algorithm provides the best scalability because it has the benefits the communication offloading and is also able to combine small messages into larger packets. In contrast, NIC-based allgather with all-to-all broadcasting algorithm performs the worst as the system size increases. This is because it does not have to combine small messages and the communication processing time for a large number of packet dominates over its benefits of message forwarding. These results is shown in Fig. 9(a).

For large messages, NIC-based allgather with all-to-all broadcasting 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 this features and have lower scalability, while the NIC-based allgather with recursive still provides better scalability than the host-based allgather due to communication offloading. These results is shown in Fig. 9(b).

6.3 Host CPU Utilization

One of the major benefits of NIC-based allgather is that it has low host CPU utilization. With the host-based allgather operation, the host process has to constantly poll for the arrival of messages and trigger the next step for the allgather operation. The host CPU is largely occupied in the whole course of the allgather operation. Fig. 10(a) shows the host CPU utilzation of host-based allgather over 16 nodes. In contrast, with the NIC-based allgather, once the host process sends its request to the NIC, it is free to perform other



(a) Host-Based Allgather (b) NIC-Based Allgather

Fig. 10. The Host CPU Utilization Comparison of the NIC-Based Allgather with All-to-All Broadcasting Algorithm (AB) and Recursive Doubling (RD) Algorithm to the Host-Based Allgather

useful computation. To find out the host CPU utilization for the NIC-based allgather, we time the average latency of an allgather operation and then subtract from it the maximal time that the host processor can spend performing other computation without increasing this latency. Fig. 10(b) shows the average host CPU utilization for the NIC-based allgather operations with both all-to-all broadcasting and recursive doubling algorithms over various numbers of nodes. The host CPU utilization is mostly the time to post a send request to the NIC and detect the completion from the receive event. Therefore, it is constant over various numbers of nodes, or different sizes of messages. Note, for small messages with all-to-all broadcasting algorithm, the overhead is higher. This is because the received data is first buffered at the NIC staging buffer to reduce the number of DMA operations, and at the end of the allgather operation, it is DMAed into the receiver queue along with the event to improve the overall performance. The host process has to perform an additional memory copy to obtain the message. The same technique is utilized for point-to-point in the original GM protocol and the similar observation is noted in the work [10]. Note that NIC-based allgather allows for a non-blocking implementation, where the processes do not wait for the result from the NICs after they have posted the requests. Instead, these processes can go on to perform useful computation that does not depends on the results, and they would only read the result from the NICs when they need the data. This would allow further reduction in CPU utilization.

7 Conclusions and Future Work

In this paper, we have explored the challenges of supporting efficient NIC-based allgather at the Network Interface Cards (NICs). Then we have proposed salient strategies are provided to perform scalable binomial tree-based group topology, efficient global buffer management, fast forwarding of data packets, efficient communication processing, as well as scalable and efficient flow control. Accordingly, scalable and high performance NIC-based allgather algorithms, all-to-all broadcasting and recursive doubling, have been designed with these strategies. The resulting NIC-based allgather operations have been implemented and incorporated into a NIC-based collective protocol [20] over Myrinet/GM.

Compared to the host-based allgather, over 16 nodes the NIC-based allgather operations improves allgather performance by a factor up to 3.01. In addition, the NIC-based allgather with recursive algorithm is more scalable for small messages, and NIC-based allgather with all-to-all broadcasting algorithm for large messages, compared to host-based allgather. Furthermore, the NIC-based allgather operations has very low host CPU utilization. To the best of the authors' knowledge, this paper is the first in the literature to report efficient NIC-based allgather algorithms over Myrinet/GM.

In the future, we intend to study the applicability of this NIC-based allgather and the collective protocol to different programming models, e.g., MPI [9] and portals [3]. In addition, we intend to study its benefits to other programming models, such as distributed shared-memory [13], and their applications. Furthermore, we intend to incorporate the NIC-based collective protocol into a resource management framework, e.g., STORM [8] to investigate their benefits in increasing the resource utilization and the efficiency of resource management.

8 Acknowledgment

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

Software Download – The NIC-based collective protocol is provided as a patch to gm-2.0.3_Linux library and firmware. Requests can be made by contacting Dr. Panda at panda@cis.ohio-state.edu.

Additional Information – Additional papers related to this research can be found on 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 *ICPP: 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] R. Brightwell, W. Lawry, A. B. Maccabe, and R. Riesen. Portals 3.0: Protocol building blocks for low overhead communication. In *Proceedings of the 2002 Workshop on Communication Architecture for Clusters*, April 2002.

[4] G. Bronevetsky, D. Marques, K. Pingali, and P. Stodghill. Collective Operations in an Application-level Fault Tolerant MPI System. In *Proceedings of Intl' Conference on Supercomputing, '03*, San Franscisco, CA, June 2003.

[5] 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.

[6] 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.

[7] D. Buntinas, D. K. Panda, and P. Sadayappan. Fast NIC-Level Barrier over Myrinet/GM. In *Proceedings of Int'l Parallel and Distributed Processing Symposium (IPDPS)*, 2001.

[8] E. Frachtenberg, F. Petrini, J. Fernandez, S. Pakin, and S. Coll. STORM: Lightning-Fast Resource Management. In *Proceedings of the Supercomputing '02*, Baltimore, MD, November 2002.

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[10] J. Liu, B. Chandrasekaran, W. Yu, J. Wu, D. Buntinas, S. P. Kini, P. Wyckoff, and D. K. Panda. Micro-Benchmark Performance Comparison of High-Speed Cluster Interconnects. *IEEE Micro*, 24(1):42–51, January-February 2004.

[11] A. Moody, J. Fernandez, F. Petrini, and D. Panda. Scalable NIC-based reduction on Large-scale Clusters. In *SC '03*, Phoenix, Arizona, November 2003.

[12] Myricom, Inc. Myrinet Express (MX): A high-performance, low-level, message passing interface for Myrinet. http://www.myri.com/scs/MX/doc/MX.pdf.

[13] R. Noronha and D. K. Panda. Implementing TreadMarks over GM on Myrinet: Challenges, Design Experience, and Performance Evaluation. In *Int'l Workshop on Communication Architecture for Clusters (CAC '03), Held in Conjunction with (IPDPS '03)*, Nice, France, April 2003.

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

[15] 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*, Phoenix, Arizona, November 2003.

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

[17] A. Skjellum, S. G. Smith, N. E. Doss, A. P. Leung, and M. Morari. The design and Evolution of zipcode. *Parallel Computing*, 20(4):565–596, 1994.

[18] R. Thakur and W. Gropp. Improving the Performance of Collective Operations in MPICH. In *Proceedings of EuroPVM/MPI03*, Venice, Italy, September 2003.

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

[20] 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 (CAC '04), Held in Conjunction with (IPDPS '04)*, Santa Fe, New Mexico, April 2004.

[21] 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 (ICPP'03)*, October 2003.

[22] 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, (LACSI '03)*, Santa Fe, New Mexico, October 2003.