High Performance and Reliable NIC-Based Multicast over Myrinet/GM-2

Weikuan Yu, Darius Buntinas and Dhabaleswar K. Panda

Technical Report
OSU-CISRC-5/03-TR33

High Performance and Reliable NIC-Based Multicast over Myrinet/GM-2*

Weikuan Yu Darius Buntinas Dhabaleswar K. Panda

Dept. of Computer and Information Science
The Ohio State University
2015 Neil Avenue
Columbus, OH 43210
{yuw, buntinas, panda}@cis.ohio-state.edu

Abstract

Multicast is an important collective communication primitive for parallel and distributed programming. Some Network Interface Cards (NICs), such as Myrinet, have programmable processors which can be programmed to support multicast. This paper proposes a high performance and reliable NIC-based multicast scheme and the associated mechanisms. Solutions were implemented over Myrinet by extending Myricom's GM-2. MPICH-GM was also modified to take advantage of the NIC-based multicast. Our evaluation shows that, at the GM-level, the NIC-based multicast improves the multicast latency by a factor up to 1.48 for messages $\leq$ 512 bytes, and a factor up to 1.86 for 16KB messages over 16 nodes compared to the traditional host-based multicast approach. On the same system, at the MPI-level, the NIC-based multicast scheme improves the multicast latency by a factor up to 1.78 for $\leq$ 512 byte messages, and a factor up to 2.02 for 8KB messages. Moreover, the NIC-based multicast scheme promises good scalability and it benefits larger size systems with reduced effects of process skew.

Keywords: *Myrinet, GM, MPICH, broadcast, multicast, collective communication, process skew, clusters*

1 Introduction

Multicast is an important collective operation in parallel and distributed programs. Message passing standards, such as PVM [11] and MPI [14], often have the multicast operation, also called broadcast, included as part of their specifications. Moreover, a resource management architecture such as STORM [10], needs an efficient multicast mechanism to replicate data across a set of nodes.

With the recognized importance of multicast operation, some interconnects such as QsNet [17], Infiniband [9] and BlueGene/L [12], provide hardware primitives to support multicast communication. Other interconnects, such as Myrinet, do not have hardware multicast and provide unicast communication along point-to-point links. Thus, a multicast operation is usually implemented at the user level with point-to-point unicast communications. Such an approach can lead to higher multicast latency. Thus it would be beneficial to reduce the latency of this operation as much as possible.

*This research is supported in part by a DOE grant #DE-FC02-01ER25506 and NSF Grants #EIA-9986052 and

Some modern network interface cards (NICs) have programmable processors which can be customized to provide support for collective communications. The use of programmable NICs to offload protocol processing for collective communications has been studies in [20, 2, 7, 8, 5, 6, 4]. Also through simulation based studies, Kesavan [13] and Sivaram [19] have evaluated various aspects of multicast with programmable NIC support. Among these studies, NIC-supported multicast communication has been implemented with several different schemes, namely, FM/MC by Verstoep [20], LFC by Bhoedjang [2] and a NIC-assisted scheme by Buntinas [7].

In this paper, we start with characterization of features that are important to NIC-based multicast in modern parallel systems. These features include reliability, forwarding, scalability, protection, tree construction etc. In this context, we analyzed the existing multicast scheme [20, 2, 7] and determine that these solutions lack one or more features. Accordingly we propose a new NIC-based multicast scheme that provides a complete set of features. Next, we explore different design alternatives for our scheme. We demonstrate how our proposed scheme can be implemented by taking advantage of recent alpha releases of Gm-2.0, which provides features to allow efficient implementation of NIC-based multicast.

We describe the design and implementation of our proposed NIC-based multicast scheme in detail, the modification of MPICH-GM to use the NIC-based multicast, as well as the evaluation of the NIC-based multicast at both the GM-level and the MPI-level. Compared to the traditional host-based multicast, this NIC-based multicast achieves an improvement factor up to 1.48 for multicasting messages ≤ 512 bytes, and an improvement factor up to 1.86 for multicasting 16KB messages over 16 nodes at the GM-level. At the MPI-level, the NIC-based multicast improves the broadcast latency by a factor up to 1.78 for ≤ 512 byte messages, and a factor up to 2.02 for 8KB messages over 16 nodes. Moreover, at the MPI-level, we have shown that larger size systems can benefit from the NIC-based multicast.

The rest of the paper is structured as follows. In the next section, we describe and compare general features of the multicast schemes with NIC-support, followed by Section 3 where we describe our scheme. In Section 4, we give an overview of Myrinet and GM. We describe design issues and implementation details for the NIC-based multicast scheme in Section 5, followed by Section 6 where we describe the modification of MPICH-GM to use the NIC-based multicast. We then present the performance results and evaluation of our implementation in Section 7. Finally, we discuss the results and conclude the paper in Section 8.

2 NIC-based Multicast

In this section we describe and characterize the features of NIC-supported multicast schemes. We identify the following features that are important to the multicast in modern parallel systems:

Forwarding, Tree Construction, Tree information – Since broadcast/multicast on point-to-point networks is typically done by having the message forwarded to the destinations along a spanning tree, the three major features of any multicast scheme are the spanning tree construction, how the tree information is specified and how messages are forwarded. The spanning tree can either be constructed at the host or at the NIC, however, it is more efficient to construct the tree at the host since the NIC processor is typically much slower than the host processor. Note, a node in a network consists of both the host and the NIC. The tree information can be either specified along with each multicast message [19, 7] or preposted to the NIC before the multicast [20, 2]. With preposted tree information the intermediate nodes are not required to provide the tree information before the message can be forwarded to their children, which can lead to maximally reduced host involvement at the intermediate nodes. Message forwarding in the multicast can either be done by the NIC or by the host. Using host-based forwarding, a node must pass the received message to the host first and which must then copy the same message back to the NIC in order to forward the message to its children. This leads to a large overhead to the multicast latency.

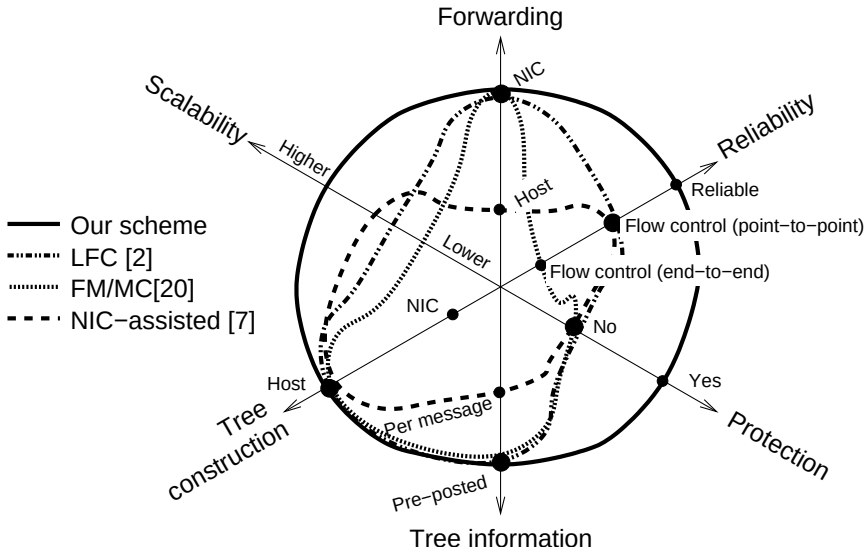


Figure 1: An axes diagram comparing the features of different multicast schemes.

Reliability – Providing reliability is also important to a multicast scheme, which can be done either directly or indirectly. While a direct reliability scheme employs acknowledgments to confirm the delivery of a message, and timeout and retransmission to deal with loss of messages, an indirect reliability scheme typically assumes the network is reliable and uses a credit-based scheme to ensure that there are buffers available at the destinations before a multicast message is initiated. However, in general a network cannot be considered reliable. Though bit error-rates are low in modern networks, they are not zero. In addition, there are drawbacks with credit-based schemes. A centralized credit scheme, which uses a centralized component to manage to all the multicast credits, will have a bottleneck at the centralized component. A distributed credit scheme, in which the credits are managed from hop to hop, can lead to deadlock since a multicast message may be initiated by the root node, while an intermediate node is running out of credits to forward the packets further.

Protection and Scalability – Depending on implementation, a multicast scheme may or may not provide protection of concurrent NIC access by several processes. Without protection, a user process may modify the NIC-memory used by another process, which can lead to unpleasant scenarios. In addition, high scalability has since been a desirable feature of protocols on parallel systems, and it becomes indispensable, as the number of nodes are reaching thousands in a cluster.

Figure 1 shows a diagram, which uses six axes to represent these six features, and compares the features of available multicast schemes, as well as the scheme we are proposing in this paper. In this diagram, a line is used to connect the points on the axes to describe the features of a particular multicast scheme. All these schemes have the host construct the spanning tree to be efficient in tree construction. The NIC-assisted scheme [7] specifies the tree information along with the message, but it requires the intermediate host involvement to perform the message forwarding. FM/MC [20] provides an end-to-end flow control for multicast with host-level credits. A centralized credit manager is used to recycle multicast credits, which does not scale as the cluster size already reaches thousands. LFC [2] provides link-level point-to-point flow control with NIC-level credits. However, it is deadlock prone since a multicast packet may be injected into the network by the root NIC, while an intermediate NIC may not be able to forward the message because it has run out of credits. LFC also partitions the NIC buffer among different peer NICs, which does not scale with clusters of thousands of nodes. Notice that, protection is not provided by either FM/MC, LFC or the NIC-assisted scheme.

In this paper, we propose a high performance, reliable and scalable NIC-based multicast scheme, which has features such as NIC-based forwarding, protection of concurrent NIC-memory access between pro-

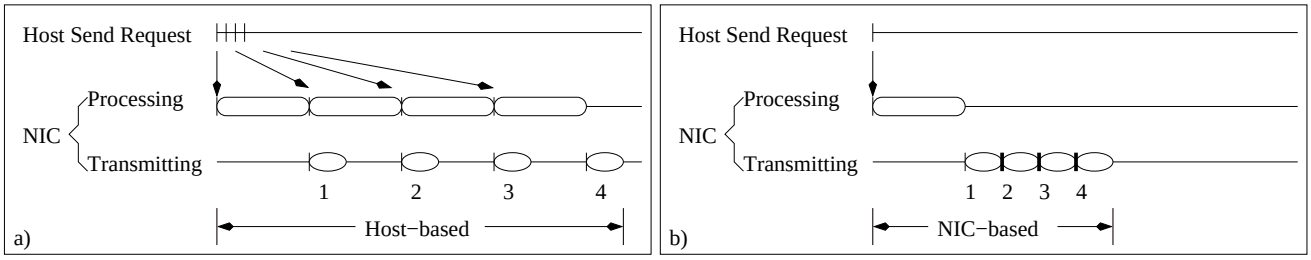


Figure 2: Abstract Timing Diagrams for sending a message to four destinations using a) multiple unicast operations and b) a NIC-based multisend operation

cesses, the tree construction at the host and preposting the tree information to the NIC before the multicast. We propose to implement this scheme over Myrinet/GM. GM [16] is a user-level communication protocol that provides a reliable ordered delivery of packets with low latency and high bandwidth. It has distinguishing features, such as, support for clusters of over 10,000 nodes and concurrent memory-protected OS-bypass access to the NIC by several user-level applications. By modifying GM to support the NIC-based multicast, while maintaining the original features of GM, it is possible to design our proposed scheme. Recent alpha releases of GM-2.0 [15] provide a myrinet packet descriptor for every network packet and also a callback handler to each this descriptor. A packet descriptor and its callback handler provide a way to get hold of a packet at any time, and take necessary action on this packet when appropriate. We have identified these features as advantages we can take, and accordingly implemented the proposed NIC-based multicast scheme using these features. This implementation provides all the features that we have proposed, i.e., a high-performance, reliable, scalable multicast scheme, supporting concurrent memory-protected access to the NIC by several applications.

3 Our Scheme

In this section, we describe our proposed scheme. Broadcast/multicast on point-to-point networks is typically done by having the message forwarded to the destinations along a spanning tree. The message originates at the root node, where multiple copies of the same message are sent to its children. Some destinations serve as secondary sources: when having received the messages, they forward the message down the tree to its children. From either kind of source nodes, multiple copies of the same data are transmitted to their children nodes. In our scheme, we use a NIC-based mechanism, by which a message is transfered only once from the host to the NIC, and from the NIC multiple replicas are transmitted to a set of destinations, referred to as the NIC-based *multisend* operation. We also extend this NIC-based multisend to a NIC-based multicast, in which NIC-based forwarding is employed at the intermediate nodes, i.e., when having received the message, the NIC at the intermediate node forwards the message without the host involvement. The NIC-based multicast has reduced multicast latency, which benefits larger size systems. We describe our scheme and its benefits over a host-based multicast scheme as we look into the process of the multicast communication.

3.1 NIC-Based Multisend and its Benefits

Figure 2 shows abstract timing diagrams for sending a message to four destinations, comparing a host-based mechanism with a NIC-based mechanism. The host-based sending can be broken into three steps. In the figure, three lines in parallel are provided to represent the timing of the three steps. At the first step, the host initiates the first stage by posting four send requests. At the second step, the NIC processes the requests in a FIFO manner, during which the messages are downloaded from the host and prepared for the next step. At the end of this step, a copy of the message is queued for

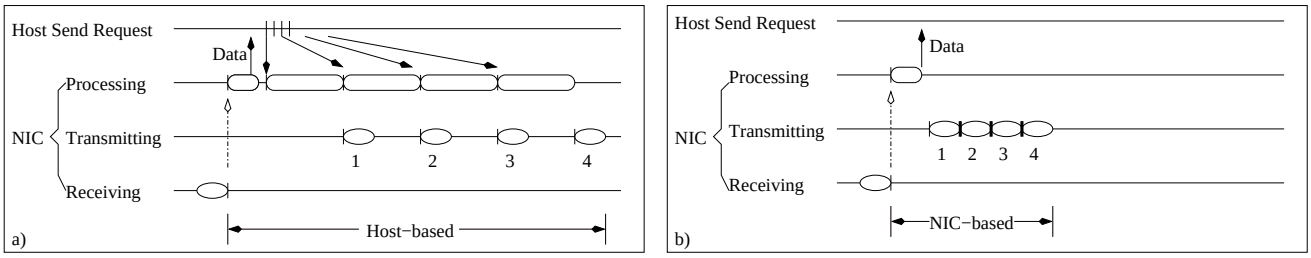


Figure 3: Abstract Timing Diagrams of an intermediate forwarding a message to four destinations using a) multiple unicast operations and b) a NIC-based multisend operation

transmitting. At the third step, the NIC transmit DMA engine completes the transmission of a queued message. As shown in Figure 2a, when the host posts 4 requests, the NIC has to repeat the second step 4 times in order to complete the sending of this message to four destinations. With a NIC-based mechanism, we can avoid repeated processing. Figure 2b shows a corresponding timing diagram for the NIC-based mechanism. The host initiates by posting only one multisend request. When the NIC notices this multisend request, it finds a corresponding list of destinations for this message, then queues the message for transmitting to the first destination. Upon the completion of the transmission to one destination, the NIC modifies only the packet header and queues the packet for transmission to another destination, and so on. The NIC can forward the same message to another destination with a small overhead to change the packet header, represented in the figure with the wider bars. However, the transmission of the same message to another destination does not have to wait for the completion of the processing of another host event. A NIC-based multisend operation can potentially transmit the messages at a faster speed, resulting in a lower latency seen from the host.

3.2 NIC-Based Forwarding and Its Benefits

In Figure 3 we show timing diagrams comparing host-based and NIC-based forwarding mechanisms. Figure 3a shows the host-based timing diagram for a single packet message. When a message arrives, the NIC copies the message into the host memory, using DMA. After receiving the message, the intermediate host initiates another set of unicast send requests to forward the message. The NIC then processes the requests and queues the messages for transmission. Notice that, at the intermediate node, for a message to be forwarded, it must be passed to the host process and must then be copied back to the NIC. This leads to a large overhead to the latency of the multicast. Figure 3b shows the timing diagram for forwarding a message with a NIC-based approach. When receiving a multicast packet, the intermediate NIC looks into the destination table to find a list of destinations for that packet. Then it DMAs the message to the host memory. The same packet will also be queued for forwarding with a changed packet header. Depending on the implementation, the forwarding can be overlapped with or after the DMA to the host. Thus in the overall multicast latency, the overhead at the intermediate host to receive the message and initiate the message forwarding is eliminated. Moreover, for multiple packet messages, an intermediate NIC can forward the packets of a message as every packet arrives using NIC-based forwarding without waiting for the arrival of the complete message. This can further reduce the multicast latency.

3.3 Tolerance to Process Skew

Besides the reduced host overhead and pipelining of packets at the intermediate nodes, the NIC-based multicast scheme has the potential of reducing the effects of process skew. When processes in a parallel program are skewed, or unsynchronized, they may not reach the same point of computation at the same

the broadcast after some delay. As the traditional host-based broadcast is implemented, an intermediate process will not forward the message until it calls the broadcast operation and receives the message. Thus, when an intermediate process is lagging behind due to the skew, the message delivery to its children will be further delayed even if they are ready to receive the message. With the NIC-based approach, the message can be forwarded by an intermediate NIC to its children even if the host process has not called the broadcast operation. So a process lagging behind at an intermediate host will not keep the processes lower in the tree from receiving the messages. Therefore the effects of process skew to the overall multicast performance can be reduced.

4 Overview of Myrinet and GM

In this section, we describe some background information on Myrinet and GM. Myrinet is a high-speed interconnect technology. It uses wormhole-routed crossbar switches to connect all the NICs to provide low latency and high performance communication. GM is a user-level communication protocol that runs over the Myrinet network [3] and provides a reliable ordered delivery of packets with low latency and high bandwidth. It has distinguishing features, such as, support for clusters of over 10,000 nodes and concurrent memory-protected OS-bypass network interface access by several user-level applications. GM provides communications to user application between end-to-end communication points, called ports.

Sending a Message – To send a message, the user application calls the appropriate function from the library. This function constructs a send descriptor, referred to as a *send event* in GM, which describes what data is to be sent and to which process to send the data to. When the NIC detects that a new send event has been posted, it transforms 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. For each send token, the NIC will DMA the data from the host buffer into a *send buffer* and transmit the message on a per packet basis. As each packet is transfered to a destination NIC, the NIC keeps a *send record* of its sequence number and the time the packet is sent. The send record will be removed when an acknowledgment from the receiver acknowledges the packet has been correctly received. If the acknowledgment is not received with the timeout time, the sender will retransmit the packet. When all the send records are acknowledged, the NIC will pass the send token back to the port, where the host can detect the completion of the send.

Receiving a Message – To receive a message, a process must provide a registered memory buffer in the host memory into which the NIC will DMA the data. The host does this by preposting a receive descriptor. A posted receive descriptor is transformed into a *receive token* by the NIC to describe the receive buffer. When the NIC receives a packet, it checks the packet sequence number with the expected sequence number for the sender NIC. When there is no discrepancy, the NIC locates a receive token, DMAs the packet data into the host memory, and also acknowledges the sender. When all the packets for a message have been received, the NIC will also generate a *receive event* to port, where the host process detects that a message has been received and the registered memory can be released.

New Features of GM-2 – In recent alpha releases of GM-2.0, a data structure, the packet descriptor, is introduced to describe every network packet that is to be sent or have been received. Inside this data structure, there is also a callback handler, which allows the possibility to take actions on the packet when appropriate. On the send side, the packets to be sent are queued for transmitting by queuing the myrinet packet descriptors, and, upon the completion of transmission, the packets are freed by freeing the myrinet packet descriptors. On the receive side, the packets that have been received are queued for copying into their final locations in the host memory, and, upon the completion of copying, the packets are also freed by freeing the myrinet packet descriptors. By using the myrinet packet descriptor and its callback handler, one can easily have a packet queued again for transmission before it is freed, by introducing an appropriate callback handler to the packet descriptor. For example, to send a message replica to another destination, a callback handler is used to change the packet header and queue the

these features as advantages we can use in our implementation of the NIC-based multicast scheme.

5 Design Issues and Our Implementation

In this section, we describe the design issues and the implementation details of the NIC-based multicast. We implemented the proposed NIC-based multicast by modifying GM version 2.0_alpha1. There are several design issues for this implementation: the sending of message replicas to multiple destinations, message forwarding at the intermediate NIC, reliability and in order delivery, deadlock, and construction of the spanning tree. For each of these issues, we describe design alternatives below and show how to select the best alternative.

5.1 Sending of Message Replicas to Multiple Destinations

To send replicas of a message to multiple destinations, one can take a simple approach to generate multiple send tokens from one multisend event and queue them to multiple destinations. Multiple replicas will be sent out as the tokens are processed. Another way to achieve this is to set up a callback handler at the time each packet of the message is queued for transmission. When the packet is to be freed at the completion of transmission, the callback handler can be invoked to change the packet header and queue the packet for transmission to another destination. This can be repeated until the packet is sent to the last destination. A third way to do this is to change the packet header right after the transmit DMA engine is done transmitting the packet header and queue the packet again for another transmission. The first approach still has to repeat the processing for each of the tokens, and it basically saves nothing but the host from posting multiple send events. The benefits from doing this is no more than 1μs, since the host overhead over GM is less than 1μs. Both the second and third approach can break the ordered delivery of packets to a destination, because packet replicas generated by these two approaches are not ordered with respect to the packets that will be created from the token that are already in the queue. One has to set up a separate ordering scheme for the the multicast packets in order to use either the second or the third approach. Moreover, the third approach takes special care and demands good timing strategy in order to avoid clobbering the packet header before the packet header is transmitted out. We implemented the second approach in our multicast scheme, since it not only has the benefits of avoiding processing multiple tokens, but also allows the relative simplicity of implementation. The benefits of the third approach could be more, but we decided to leave it for later research. The separate ordering scheme will be discussed later in this section.

5.2 Messages Forwarding at the Intermediate NIC

For a received message to be forwarded by the intermediate NIC, there are several issues to be considered. These issues are: a) how to set up timeout and retransmission mechanisms, b) which replica of the message should be made available for the retransmission, and c) whether first to forward the message or copy the message to the host memory. First, similar to the sending of multiple replicas in Subsection 5.1, the forwarding of a received packet will break the ordered delivery of packets between the intermediate NIC and its children, but beyond that, there is an issue of how to set up timeout and retransmission mechanisms. For setting up the timer, we created send records to record the time the packets are forwarded. The send records also have to use a separate ordering scheme which we will discuss in the next subsections. When the records are not acknowledged within the timeout period, retransmission of the packets should be fired up. But since the intermediate NIC does not have a send token for this multicast, one has to generate a token for the purpose of retransmission. This can be done by grabbing a send token from the free send token pool, or by transforming the receive token into a send token. Using the former approach can lead to the possibility of deadlock when the intermediate nodes are running

out of send tokens. We took the second approach since it does not require additional resources at the NIC. In this approach, the receive token is used for transferring the data to the host at the intermediate NIC, and is also used to retransmit the message when timeout.

Secondly, there is an issue about which replica of the message to be made available for the retransmission. A naive solution would be keeping the received packet available until all the children acknowledge that they have correctly received the forwarded packets. When the receive buffer is running out, more receive buffers need to be allocated in order to receive incoming packets. The problem with this approach is that the NIC memory is a limited resource, and even with a larger number of receive buffers, the free receive buffer will eventually run out for incoming packets, when the multicast traffic is high. Another approach is to release the packet as the forwarding is done, and keep the message replica in the host memory available for retransmission. Since GM can only send and receive data from registered memory, this requires the implementation to keep the host memory being registered until all the children acknowledged that the packets are correctly received. We took the second approach. The host is notified that the memory can be deregistered only when the message is acknowledged by all of its children.

Lastly, forwarding the message before DMAing to the host seems a way that could speed up the multicast. However, in GM, a simple implementation of this would violate the ordering of DMA events. We decided to copy the message to the host memory first for the robustness of following current event ordering scheme. Also we did not parallelize the copying of message to the host memory and the message forwarding in this preliminary implementation. Further investigation is planned to study this possible optimization.

5.3 Reliability and In Order Delivery

To ensure ordered sending, GM queues the *send token*s according to the destination NICs and ports, then transmits data packets for a token only when all the packets for the previous tokens to the same destination NIC and port have been transmitted. To ensure ordered receiving, each NIC keeps track of the sequence number to each peer NICs. All packets must be received in successive sequence numbers, and the receiver will always acknowledge the last received packet with the correct sequence number, ensuring ordered receiving. When a packet is not acknowledged within a timeout period, the sender NIC will retransmit the packet, as well as all the packets with greater sequence numbers.

A reliable ordered multicast requires modification to the existing ordering scheme. Since each sender is involved with multiple receivers, the sending side must keep track of the ordering of packets in a one-to-many manner to all its children. A modified ordering scheme works as described below. Multicast *send token*s are queued by group. Each multicast group has a unique group id. For each group, the NIC keeps tracks of a) a receive sequence number to record the sequence number for the packets received from its parent, b) a send sequence number to record the packets that have been sent out, and c) an array of sequence numbers to record the acknowledged sequence number from each child. A multicast packet sent from one NIC to its children has the same sequence number and send record, ensuring ordered sending for the same group's multicast packets. When an acknowledgment from one destination is received, it updates the acknowledged sequence number for that destination. If the record for a packet is timed out, the retransmission of the packet and the following ones will be performed only for the destinations that did not acknowledge. On the receiving side, only the packets with the expected sequence number for the desired group are accepted and acknowledged sequentially.

5.4 Deadlock

Deadlock is an important aspect of concern for any collective communication, which may occur if there is a cyclic dependence on using some shared resources among multiple concurrent operations. We take the following approaches to avoid the possibilities of the deadlock. First, we do not use any credit-based

flow control, avoiding one source of deadlock. In addition, we provide a unique group identifier and a separate queue for each multicast group with a sender, so that one group does not block the progress of another. The other possibility for a deadlock is when some nodes in multiple broadcast operations form a cyclic parent-child relationship, in which all of them are using its last receive token while requesting another to receive its message with a new receive token. Since the root node in a broadcast operation only uses its send token, it will not be in such a cycle. To break a possible cycle among the rest of the nodes, we sort the list of destinations linearly by their network IDs before tree construction, and a child must have a network ID greater than its parent unless its parent is the root. Thus a deadlock on the use of receive token can not form under either situation. A simple case study is provided below to show the idea. As long as receive tokens are available at the destinations, multicast packets can be received by all the destinations. The responsibility of making receive tokens available to receive multicast messages is left to client programs, the same way as is required to receive regular point-to-point messages.

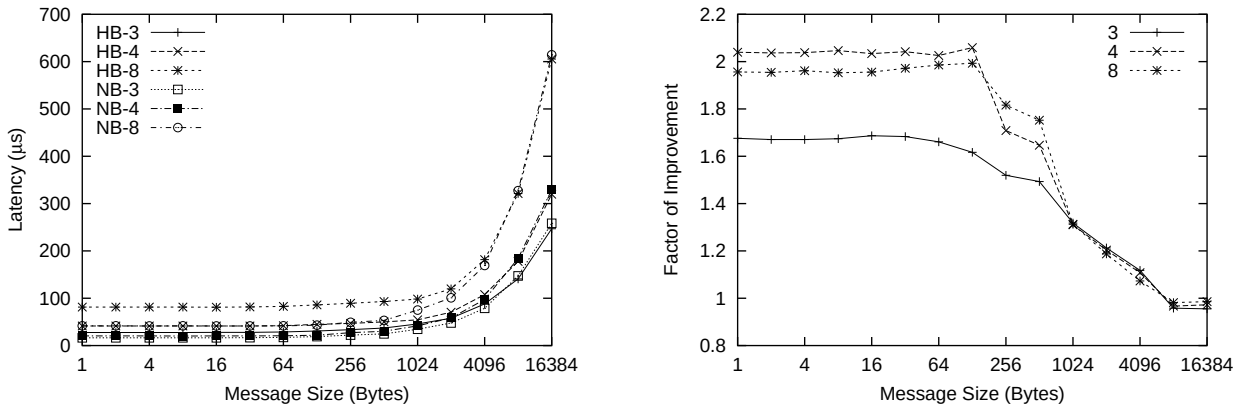
Case Study – There are two concurrent broadcast operations. In operation 1, Node with GM_ID M_0 broadcasts to receivers, M_1, M_2, ..., M_s. In operation 2, Node N_0 broadcasts to receivers, N_1, N_2, ..., N_s. M_0 and N_0 use a send token to initiate the broadcast operation, so they can not be involved such a request cycle. Assume a cycle is formed by these two trees. Any node in the cycle must be M_j or N_j, where $1 \leq j \leq s$. Take two nodes with GM_ID, P and Q, from such a cycle, the cycle must be some form of this ordering, (..., P, ..., Q, ..., P, ...). Since the nodes are sorted by the order of their IDs, a parent node must has an ID that is smaller than its children's. This implies $P < Q$ and $Q < P$, which forms a contradiction.

5.5 The Spanning Tree

The tree topology is also an important aspect that could affect the performance of multicast. One tree topology may give better performance over another depending on the communication characteristics and also the desired performance metrics, i.e., latency or throughput. In addition, *where* to generate the tree is also a design issue, since the NIC processor is typically rather slow to perform intensive computation. To better expose the potential the NIC-based multicast protocol, we use an algorithm similar to [7] for constructing an optimal tree in terms of latency. The optimality of such trees has been shown by Bar-Noy and Kipnis [1]. The basic idea of constructing an optimal tree is to have the maximal number of nodes involved in sending at any time. In other words, we construct the tree such that a node will send to as many destinations as possible while the first destination it sent to has not yet received the message and started sending itself. We computed the number of destinations a sender can send to before its first receiver can start sending as the ratio of a) the total amount of time for a node to send a message until the receiver receives it, and (b) the average time for the sender to send a message to one additional destination. The message delivery time is calculated as end-to-end latency. Different message lengths would lead to different optimal tree topologies. The group memberships can be updated when necessary. This flexibility is provided by the following division of labor between the NIC and the host: The host to generate a spanning tree and insert into a group table stored in the NIC and the NIC being only reponsible for the protocol processing related to communication. Since the LANai processor is much slower compared to the host processor, this division of labor tends to be more efficient for a large tree.

6 Incorporation of the NIC-based multicast into MPICH-GM

MPICH is a widely used MPI implementation. The layered design of MPICH lends itself the great portability to any new low level platform with just the efforts to write a new *channel interface*. For the channel interfaces that do not provide collective communications, MPICH provides its default collective communication with the point-to-point function. MPICH-GM is a port of MPICH on top of GM



(a) Latency

(b) The Performance Improvement

Figure 4: The performance of the NIC-based (NB) multisend operation, compared to Host-based (HB) multiple unicasts

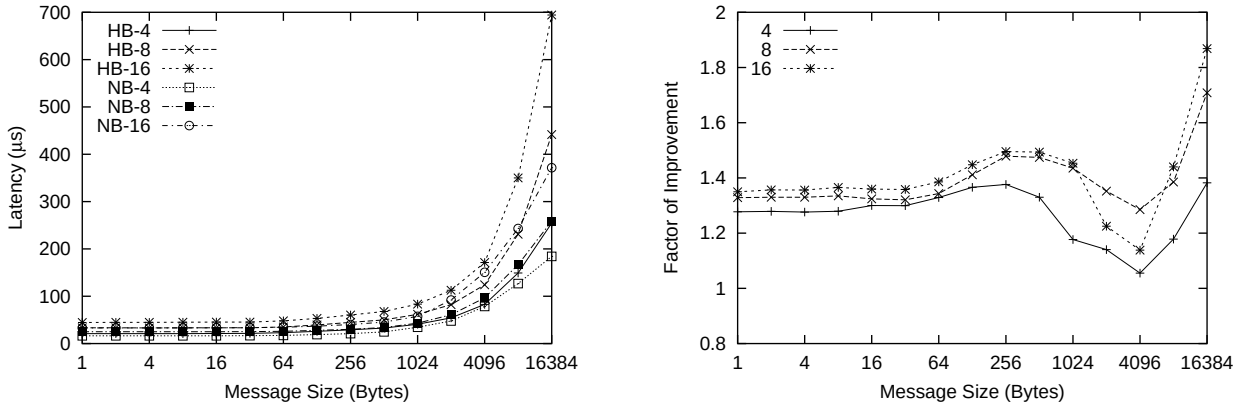
with the channel interface *ch_gm*, which does not have such collective communication support. It is of our interests to investigate how MPICH-GM performance can benefit from the NIC-based multicast support. Since MPICH-GM uses remote DMA operations in its rendezvous protocol for transferring messages larger than 16K, the modified channel interface still performs broadcast operation in the same manner as the original MPICH function does for larger than 16K messages. For messages less than 16K, all the nodes first check to see if there is an existing group context for the current broadcast root in the provided communicator. For the first broadcast operation from a particular root in a communicator, a new group context will be created and the group membership will be updated into the NIC. Although the first broadcast operation for any group will pay the cost of creating group membership, we consider this demand-driven approach to be appropriate for dealing with the unbounded number of possible combinations of communicators and broadcast source nodes. Once the unique group context is identified, the root node will initiate a NIC-based multicast operation, while the destinations will invoke blocking receiving operations as MPI_Recv does. The delivery of the messages to all the destinations are performed at the NIC. All the messages copying and filling of message envelope are performed in the same manner as the original MPICH-GM does in the eager protocol.

7 Performance Evaluation

In this section, we describe the performance evaluation of our implementation. The experiments were conducted on a 16 node cluster consisting of 16 quad-SMP 700 MHz Pentium-III nodes with 66MHz/64bit PCI bus. The nodes have Myrinet NICs with 133MHz LANai 9.1 processors and are connected to a Myrinet 2000 network. Each of these nodes run the 2.4.18 Linux kernel. We compared our NIC-based implementation, which is based on GM-2.0_alpha1, to the host-based implementation using the same version of GM. MPICH-GM version 1.2.4..8a was modified to use the NIC-based multicast. The same version of MPICH-GM was used as the comparison at the MPI-level.

7.1 GM Level

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 non-multicast communications.



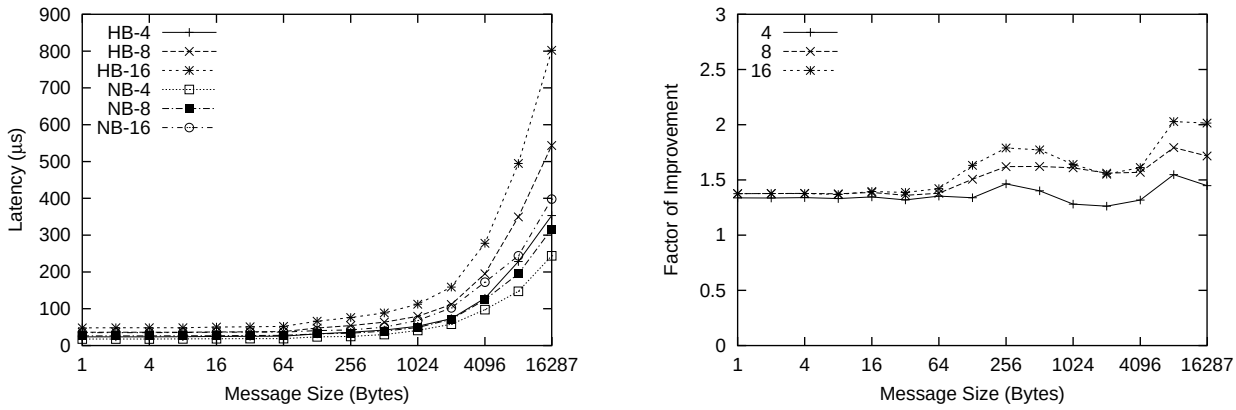
(a) Multicast Latency

(b) The Performance Improvement

Figure 5: The GM-level performance over 4, 8 and 16 node systems of the NIC-based (NB) multicast, compared to the host-based multicast (HB)

NIC-based multisend – We first evaluated the performance of the NIC-based multisend operation. Our tests were conducted by having the source node transmit a message to multiple destinations, and wait for an acknowledgment from the last destination. All destinations received the message from the source node, and none of them forwarded the message. The first 20 iterations were used to synchronize the nodes. Then 10,000 iterations were timed to take the average for the latency. Figures 4(a) and 4(b) show the performance and the improvement of using the NIC-based multisend operation to transmit messages to 3, 4 and 8 destinations, compared to the same tests conducted using host-based multiple unicasts. For sending messages $\leq$ 128 bytes to 4 destinations, an improvement factor up to 2.05 is achieved. This is due to the fact that the NIC-based multicast was able to save repeated processing for sending message replicas, so that the messages can be sent out faster. As the message size gets larger, the improvement factor decreases and eventually levels off at a little below 1. This is to be expected because large message sizes leads to longer transmission time. Using host-based multiple unicasts, better overlapping of request processing and message transmission is achieved. Eventually the overhead for changing the header and forwarding the packet to another destination becomes a constant overhead to the transmission time of a packet.

NIC-based multicast – We evaluated the performance of the multicast with NIC-based forwarding using an optimal tree. Our tests were conducted by having the root initiate the NIC-based multicast operation, and wait for an acknowledgment from one of the leaf nodes in the spanning tree. The first 20 iterations were used to synchronize the nodes. Then 10,000 iterations were timed to take the average. The same test was repeated with different leaf nodes returning the acknowledgment. The maximum from all the tests was taken as the multicast latency. The traditional host-based multicast was also evaluated in the same manner using the same version of GM as a comparison. Figures 5(a) and 5(b) shows the performance of the NIC-based multicast compared to the performance of host-based multicast. For broadcasting messages $\leq$ 512 bytes on a 16 node system, the NIC-based multicast achieves an improvement factor up to 1.48. This is to be expected because an optimal tree has a large average degree and so a shallower depth, compared the same size of binomial tree used in the traditional host-based multicast. However, multiple replicas of small messages can be sent out faster with the NIC-based multicast, therefore the larger fan-out degree does not impact the latency, and the shallower depth leads to the reduced multicast latency. When broadcasting a 16KB message on a 16 node system, the NIC-based multicast achieves an improvement factor up to 1.86. This is due to the fact that, in the NIC-based multicast, intermediate nodes do not have to wait for the arrival of the complete message to forward the message. Thus the NIC-based multicast achieves its performance benefits for the reduced intermediate host involvement and the capability of pipelining messages. As



(a) Multicast Latency

(b) The Performance Improvement

Figure 6: The MPI-level performance over 4, 8 and 16 node systems of the NIC-based (NB) multicast, compared to the host-based multicast (HB)

shown in Figure 5(b), there are dips in the improvement factor curves for multicasting 2KB and 4KB messages, that is due to the fact that the NIC-based multisend is not able to gain much benefits for single packet, larger than to equal to 1KB messages, and also it does not have benefits from message pipelining. Thus the performance improvement factors are low for these messages.

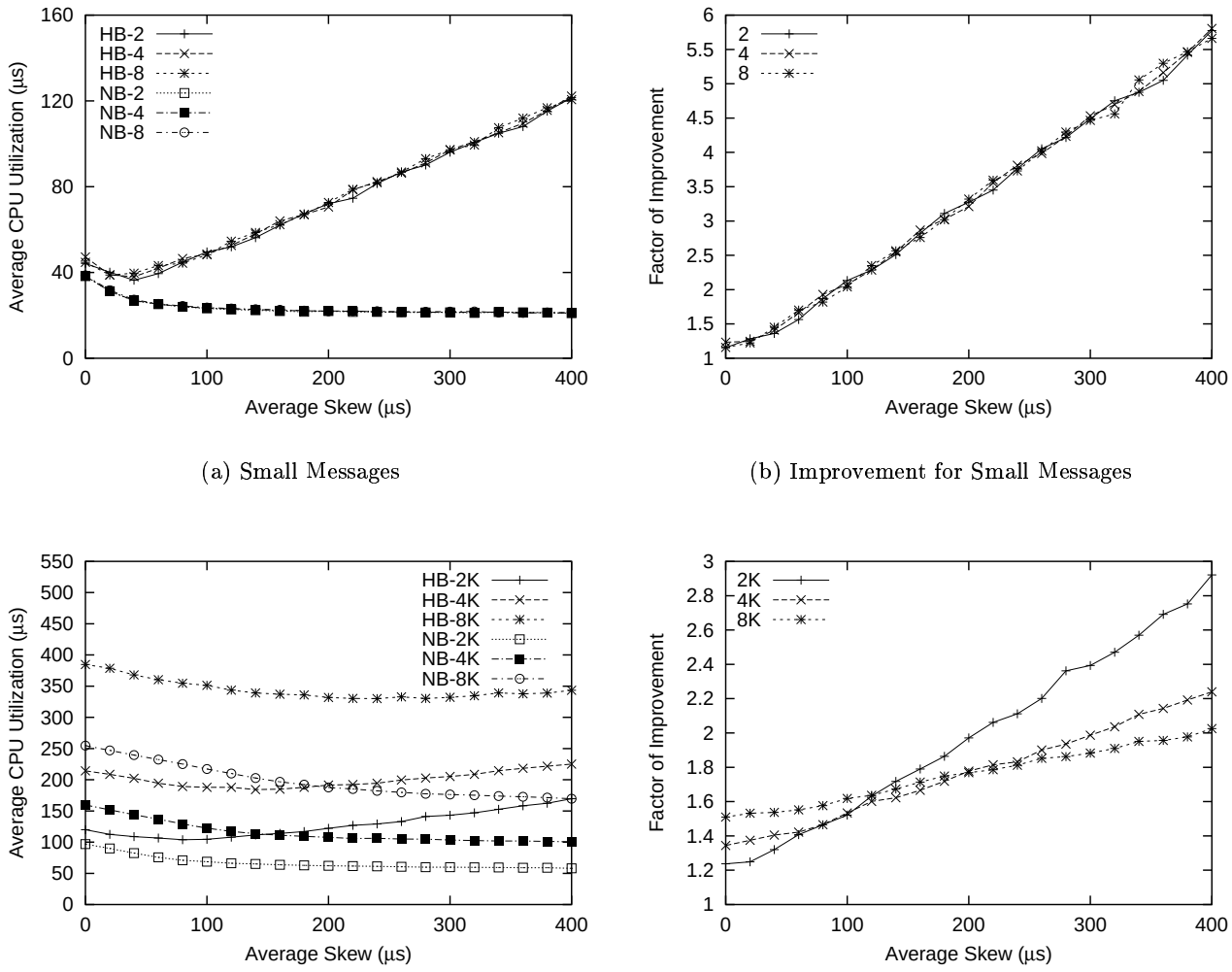
7.2 MPI Level

Since our modification to MPICH-GM only uses the NIC-based multicast support for the eager mode message passing, the largest message that uses the NIC-based multicast is the largest eager mode message, which is 16,287 bytes.

MPI broadcast – We measured the broadcast latency at the MPI level in the same manner as that at the GM level. The maximum latency obtained was taken as the broadcast latency. Figure 6(a) and Figure 6(b) show the latency performance and the improvement factor of the NIC-based multicast at the MPI level, respectively. We observed an improvement factor of up to 2.02 for broadcasting 8KB messages over 16 node system, an improvement factor of more than 1.3 for broadcasting any size messages. Also the trend of the performance improvements are similar to the trend at the GM-level (Figure 5(b)). However, when broadcasting 16,287 byte messages, there is a dip of the improvement factor curve. That is due to the larger cost of copying the data to their final locations. So the broadcast latency for a 16,287 byte message with the NIC-based multicast is comparatively high, which leads to a lower improvement factor.

7.3 Tolerance to Process Skew

Another major benefit of the NIC-based multicast is the tolerance to process skew. Each process, when performing the MPI_Bcast(), utilizes some amount of the host CPU time until the completion of the broadcast. Typically, with the blocking implementation of MPI_Bcast(), the host CPU time used to perform the MPI_Bcast() becomes larger if the multicast latency is prolonged due to a delayed process at an intermediate node. In reality, all processes skew at random. Some processes call MPI_Bcast() before the root node does, and others do after the root node. Because all the processes that have called MPI_Bcast() earlier inevitably have to wait before the multicast can be initiated by the root process, the effects of the former can not be reduced by a multicast operation, but those from the latter can be reduced when possible. We only intended to evaluate the effects of the delayed processes, relative



(a) Small Messages

(b) Improvement for Small Messages

(c) Large Messages

(d) Improvement for Large Messages

Figure 7: Average host CPU time on performing the MPI_Bcast() for small messages (2, 4 and 8 bytes) and large messages (2KB, 4KB, 8KB) over 16 nodes, under different amount of average skew with both the host-based approach (HB) and the NIC-based (NB) approach

to the root processes, to the average host CPU time. To this purpose, we measured the average host CPU time to perform the MPI_Bcast() with varying amount of process skew in the following manner. All the processes were first synchronized with a MPI_Barrier(). Then each process, except the root, chose a random number between the negative half and the positive half of a maximum value as the amount of skew they might have. The processes with a negative number perform the MPI_Bcast() with no delay; and the others with a positive skew time perform computation for this amount of skew time, then perform the MPI_Bcast() operation. After 5,000 iterations of such tests, the root process collects the total host CPU time for performing the MPI_Bcast() operation from all the processes. To show the effect of the process skew, the average host CPU time was graphed against average skew between processes. The average skew is defined as the expected skew between two arbitrary non-root processes, which can be considered as the expected distance between two random distributed points on a given interval $[\,0,\ max\,]$. The following formula gives the average skew, avg, for a maximum skew value, max:

$$avg = max \int_0^1 \int_0^1 |\, x - y \,| dx dy$$

Performing integration on the formula arrives the result, $avg = \frac{1}{3}max$. Our empirical results also confirmed that the average skew is a third of the maximum skew value. Theoretical statistical support to this conclusion can also be found in [18], page 118.

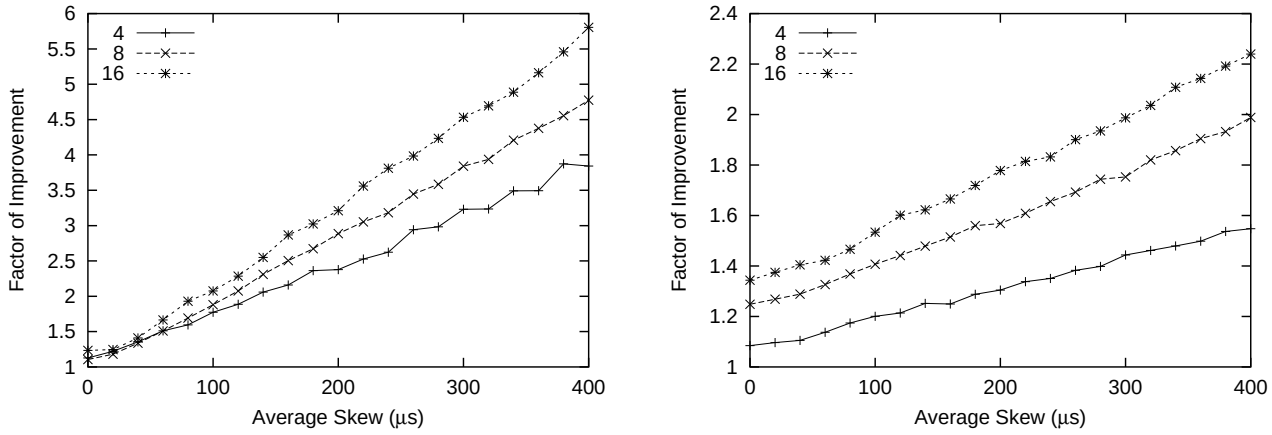
Figure 7(a) shows the average host CPU time for broadcasting small messages (2, 4 and 8 bytes) over 16 nodes with varying amount of average skew. The graph does not show much differentiation between the message sizes, however, a large difference is seen between the NIC-based broadcast and the host-based broadcast. With an average skew under 40μs, the average host CPU time decreases using either broadcast. This is to be expected because a small amount of skew time can overlap with some of the message broadcasting time. When the average skew goes beyond 40μs, the average host CPU time increases for the processes performing MPI_Bcast() using the host-based approach. This is because, as the average skew between processes increases, more intermediate processes gets delayed, therefore more processes wait longer for their ancestors to call MPI_Bcast(). In contrast, with the increasing average skew between processes, the average host CPU time decreases for the processes performing MPI_Bcast() using the NIC-based approach. This is due to the fact that the probability that the message has arrived before the host process calls MPI_Bcast() becomes higher, so that more processes spend less time in waiting for the arrival of the message, which leads to less average host CPU time to perform MPI_Bcast(). Figure 7(b) shows that the factor of improvement for performing the MPI_Bcast() using the NIC-based approach over the host-based approach for small messages. With an average skew of 400μs, the NIC-based multicast achieves an improvement factor up to 5.82. We also saw that the improvement factor becomes greater as the average skew increases.

Figure 7(c) shows the average host CPU time for broadcasting large messages (2KB, 4KB and 8KB) over 16 nodes with varying amount of average skew. Again, we see that, with the NIC-based approach, the average host CPU time spent by the processes performing MPI_Bcast() decreases as the average skew increases, while, with the host-based approach, the average host CPU time spent by the processes performing MPI_Bcast() increases once the average skew is large enough that the overlap of the delay time and the broadcast time is no longer seen. Figure 7(d) shows that the factor of improvement for performing the MPI_Bcast() using the NIC-based approach over the host-based approach for large messages. When there is no skew, the average host CPU time is actually similar to the performance of MPI_Bcast() at the MPI-level, see Figure 6(b). As the average skew increases, the benefits from the reduced skew become more pronounced. For the larger cost of coping larger messages, the improvement factors for larger message sizes are smaller. With an average skew of 400μs, the NIC-based approach achieves an improvement factor up to 2.9 for 2KB messages, up to 2.22 for 4KB messages and up to 2.01 for 8KB messages. Again the improvement factor becomes greater as the average skew between processes increases.

We also evaluated the effect of system size on the average host CPU time in the presence of process skew. Figures 8(a) and 8(b) show the factors of improvement on the average host CPU time for broadcasting 4 byte messages and 4KB messages using the NIC-based multicast compared to the host-based multicast, over 4, 8 and 16 nodes with varying amount of average skew. For both sizes of messages, the improvement factor becomes greater as the system size increases for a fixed amount of process skew. This suggests that a larger size system can benefit more from the NIC-based multicast with reduced effects of process skew.

8 Conclusions and Future Work

We have characterized features of multicast schemes that uses Myrinet programmable NICs, and proposed a NIC-based multicast scheme with a complete set of features. We have implemented this NIC-based multicast scheme, which mainly consists of a NIC-based multisend mechanism, using the NICs to send multiple replicas of a message to different destinations, and a NIC-based forwarding mecha-



(a) 4 Byte Message

(b) 4KB Message

Figure 8: The performance improvement on average host CPU time when broadcasting a) 4 bytes and b) 4KB messages over 4, 8 and 16 nodes, with the NIC-based (NB) approach and the host-based (HB) approach, under different amount of average skew between processes

involvement. This results in a high performance and reliable NIC-based multicast scheme. We also modified MPICH-GM to take advantage of this NIC-based multicast. The performance benefits of the NIC-based multicast were evaluated at both the GM-level and the MPI-level.

The NIC-based multisend operation achieves an improvement of a factor up to 2.05 over host-based multiple unicasts for $\leq$ 128 byte messages, When the NIC-based multisend is extended into multicast with NIC-based forwarding, at the GM-level, it provides an improvement factor up to 1.86 for 16KB messages and an improvement factor up to 1.48 for $\leq$ 512 byte messages over 16 nodes. At the MPI-level, the NIC-based multicast achieves an improvement factor up to 2.02 for 8KB messages, and an improvement factor up to 1.78 for small messages $\leq$ 512 bytes over 16 nodes. Moreover, at the MPI-level, the NIC-based multicast was shown to have better tolerance to process skew. In the presence of an average skew of 400μs on a 16 node system, using the NIC-based approach to perform MPI_Bcast provides an improvement factor up to 5.82 for small messages, and an improvement factor up to 2.9 for large messages in terms of the average host CPU time, compared to the host-based approach. The NIC-based multicast is also shown to be more beneficial to larger size systems.

There are several perspectives for the future work. The multicast scheme achieves its reliability and efficiency without using a centralized manager and requires minimum memory and processor resources at the NIC, which promises good scalability. So we intend to study its scalability in a large scale system. Also on larger size systems, in order to better utilize the existing groups and reduce the number of groups down the NIC, we intend to study how we can combine both the benefits of a precreated static group in the NIC and the flexibility of specifying multiple destinations along with a multisend event. Moreover, MPICH-GM employs remote DMA for transferring messages over 16K in the rendezvous protocol. We intend to study the NIC-based multicast using remote DMA operations and the performance impacts at different levels. Finally, we plan to study the benefits of such NIC-based multicasting for a large number of applications.

Software Distribution

A complete package of modified GM-2.0_alpha1 and microbenchmarks described in this paper are available. Requests can be made to Dr. D.K. Panda at panda@cis.ohio-state.edu.

Additional Information

Additional papers related to this research can be obtained from 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] A. Bar-Noy and S. Kipnis. Designing broadcasting algorithms in the postal model for message-passing systems. In *ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, 1992.

[2] R. Bhoedjang, T. Ruhl, and H. Bal. LFC: A Communication Substrate for Myrinet. In *Proceedings of the Fourth Annual Conference of the Advanced School for Computing and Imaging*, pages 31–37, June 1998.

[3] 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.

[4] D. Buntinas, D. Panda, and W. Gropp. NIC-Based Atomic Remote Memory Operations. In *Workshop on Novel uses of System Area Networks, (SAN-1)*, February 2002.

[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, and R. Brightwell. Application-Bypass Broadcast in MPICH over GM. In *Proceedings of Cluster Computing and Grid (CCGrid'03)*, May 2003.

[7] 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.

[8] 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.

[9] C. Eddington. InfiniBridge: An InfiniBand Channel Adapter With Integrated Switch. *IEEE Micro*, (2):48–56, April 2002.

[10] 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.

[11] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM Parallel Virtual Machine, A User's Guide and Tutorial for Networked Parallel Computing*. MIT Press, Cambridge, Mass., 1994.

[12] M. Gupta. Challenges in Developing Scalable Scalable Software for BlueGene/L. In *Scaling to New Heights Workshop*, Pittsburgh, PA, May 2002.

[13] R. Kesavan and D. K. Panda. Optimal Multicast with Packetization and Network Interface Support. In *Proceedings of the International Conference on Parallel Processing (ICPP'97)*, pages 370–377, 1997.

[14] Message Passing Interface Forum, MPIF. MPI-2: Extensions to the Message-Passing Interface. Technical Report, University of Tennessee, Knoxville, 1996.

[15] Myricom. Myrinet Software and Customer Support. http://www.myri.com/scs/GM/doc/, 2003.

[16] Myricom Corporations. The GM Message Passing Systems.

[17] F. Petrini, W. C. Feng, A. Hoisie, S. Coll, and E. Frachtenberg. The Quadrics Network (QsNet): High-Performance Clustering Technology. In *the Proceedings of Hot Interconnects '01*, August 2001.

[18] J. A. Rice. *Mathematical Statics and Data Analysis*. Duxbury Press, Belmont, California, 1995.

[19] R. Sivaram, R. Kesavan, D. Panda, and C. Stunkel. Where to Provide Support for Efficient Multicasting in Irregular Networks: Network Interface or Switch? In *Proceedings of the International Conference on Parallel Processing (ICPP'98)*, pages 452–459, 1998.

[20] 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.