Broadcast/Multicast over Myrinet using NIC-Assisted Multidestination Messages*

Darius Buntinas Dhabaleswar K. Panda Jose Duato P. Sadayappan

Network-Based Computing Laboratory
Dept. of Computer and Information Science
The Ohio State University
Columbus, OH 43210
email:{buntinas, panda, duato, saday}@cis.ohio-state.edu

Abstract

Broadcasting and multicasting are common operations in parallel and distributed programs. Some modern Network Interface Cards (NICs) have programmable processors which can be used to provide support for these operations. However these processors are 5-15 times slower than the host processor. In this paper we propose a design and an implementation of a multi-send primitive to support efficient broadcast/multicast that requires minimal assistance from the NIC. Our scheme is designed with the idea that as much processing as possible should be done by the host processor. This gives us more flexibility with, for example, creating multicast trees which would be optimal for a particular message size, or choosing a multicast tree dynamically based on requirements of bandwidth versus latency for a particular message. We have designed a multi-send primitive and implemented it as an addition to Fast-Messages (FM) 2.1 running over a Myrinet network. The proposed scheme does less processing at the NIC. The impact of adding such NIC-assisted multicast operation to a run-time system is also very small, less than 500ns for non-multi-send packets. To fully utilize the benefits of this primitive, we propose a method for constructing an optimal multicast tree using the new primitive. We have evaluated this scheme and obtained a speedup factor of up to 1.85 for multicasting 16K messages with 16 nodes.

1 Introduction

Broadcasting and multicasting are common operations in parallel and distributed programs. For example, MPI [8] has a broadcast operation defined as part of the standard. It would be beneficial to be able to reduce the latency of this operation as much as possible. Some modern network interface cards (NICs) have programmable processors which can be used to provide support for collective communications such as broadcast/multicast.

Using programmable network interface cards (NICs) to support broadcasting/multicasting in a cluster is not a new idea. Bhoedjang [3] and Verstoep [15] have implemented NIC-supported multicasting, and Kesavan [6] and Sivaram [12] have evaluated different aspects of NIC-supported multicasting. In each of these schemes, the NIC is responsible for all aspects of the operation, including creating the tree and handling flow control. However, the processors found on current generation NICs are usually much slower than the host processors, for example, the processors on current generation clusters are 300-500MHz while the processors on the Myrinet NICs are 33-66MHz, this is roughly 5-15 times slower. This limits the amount of work that can be done by the NIC without compromising performance.

This raises a challenge whether new communication mechanisms can be developed for clusters to support broadcast/multicast with minimal NIC assistance, while delivering good performance. In this paper we take on such a challenge. We introduce a multidestination message passing mechanism. Such a mechanism has been developed earlier for router-based parallel systems [11, 10, 13] to support efficient collective communication.

In this paper we design and implement a *multi-send* primitive to support efficient broadcast/multicast that requires minimal assistance from the NIC. Our scheme is designed with the idea that as much processing as possible should be done by the host processor. This gives us more flexibility with, for example, creating multicast trees which would be optimal for a particular message size, or choosing a multicast tree dynamically based on requirements of bandwidth versus latency for a particular message. Also, because the proposed scheme does

*This research is supported in part by an NSF Career award MIP-9502294, NSF Grant CCR-9704512, and Ameritech Faculty Fellowship Award, and grants from the Ohio Board of Regents.

less processing at the NIC, the impact of adding such NIC-assisted multicast operation to a run-time system is very small (less than 500ns for non-multi-send packets) when compared with others (a 5μs additional per-packet overhead in the NIC alone in [15]).

We have implemented the multi-send primitive as an addition to Illinois Fast-Messages (FM) 2.1 [9, 7] running over a Myrinet [5] network. To fully utilize the benefits of this primitive, we propose a method for constructing an optimal multicast tree using the new primitive. We have evaluated this scheme and obtained a factor of improvement of up to 1.85 for multicasting 16K messages with 16 nodes.

Section 2 describes the new multi-send primitive, followed by Section 2 which describes broadcasting and multicasting using the new primitive. Construction of the optimal multicast tree is described in Section 2. The implementation details are discussed in Section 2 followed by our experimental results in Section 6. Related work is discussed in Section 7. Finally we conclude and discuss future work in Section 8.

2 NIC-Assisted Multidestination Message Passing

The basic idea is to create a *multi-send* primitive in which the host writes the packet data to the NIC only once followed by a list of destinations. The NIC would then transmit copies of the packet to each of those destinations. Figure 1 shows two diagrams where host 0 is sending packets to hosts 1 through 3. The top diagram shows host 0 making three unicast (point-to-point) sends, each of which is forwarded by the NIC to its destination. The diagram on the bottom shows the host making a single multi-send operation to the NIC which then forwards a copy of the packet to each of the destinations.

Figure 2a shows the timing diagram for a multi-send operation sending a packet to four destinations, and the receive time for the last destination. Figure 2b shows the corresponding timing diagram for host-based point-to-point operations. In the figure the interval marked *Send* corresponds to the time it takes the host to assemble a packet and write it to the send queue on the NIC, and the interval marked *Xmit* corresponds to the time it takes the NIC to transmit the packet from the send queue to the network. The interval marked *Recv* corresponds to the combined time for the NIC to receive the packet, (including the network latency), for the NIC to forward the packet to the host, and for the host to process the packet. Notice that in both diagrams the receive time for a packet at the last receiver is overlapped with the transmission time at the sender for that same packet. In Figure 2b, for the first three packets, the network transmit time of one packet is overlapped with the host send time of the next and so it is not shown. As indicated below, timing parameters for FM over Myrinet are such that this will always be the case, regardless of the packet size. Though it is not shown in these figures, packet reception can also be pipelined between the NIC and the host.

Let us compare the latency of sending a packet to k destinations using a multi-send operation with the latency of sending a packet to k destinations using the usual host-based point-to-point operations. The time for the kth destination to receive the packet using a multi-send operation would be $(t_{Send} + (k-1) \times t_{Xmit} + t_{Recv})$, and $(k \times t_{Send} + t_{Recv})$ using host-based point-to-point operations. We have timed the host send, the NIC transmit and the receive operations in FM 2.1 on our cluster consisting of 300MHz Pentium II machines with 33MHz LANai 4.3 Myrinet cards. We estimate the send time to be (2.7863μs+0.0301μs$\times m$)[1], the NIC transmit time to be (1.3958μs+0.0075μs$\times m$) and the receive time to be (4.2820μs+0.0230μs$\times m$), where m is the packet size in bytes. Thus for sending a 1,536 byte packet to six destinations, t_{Send} would be 49.0199μs, t_{Xmit} would be 12.9158μs, and t_{Recv} would be 39.61μs. The multi-send operation would take 153.2089μs and the usual host-based point-to-point method would take 333.7294μs. This leads to a factor of improvement of

[1] See the Appendix on how we arrived at these estimates.

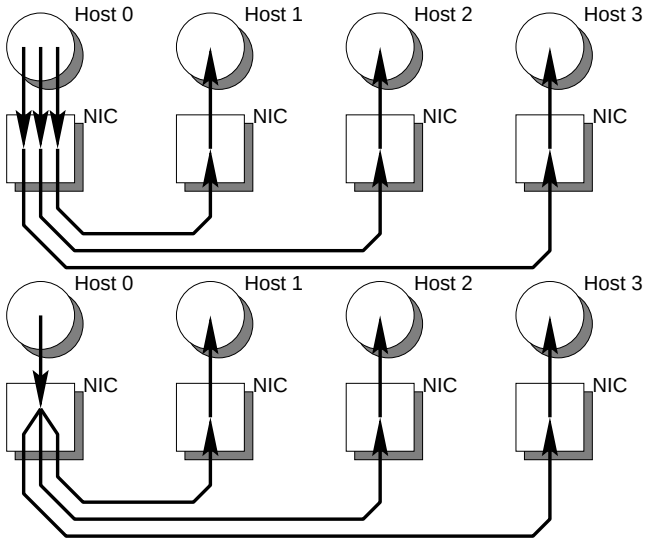

Figure 1: Multiple host-based point-to-point operations (top) and a NIC-assisted multi-send operation (bottom) to four destinations.

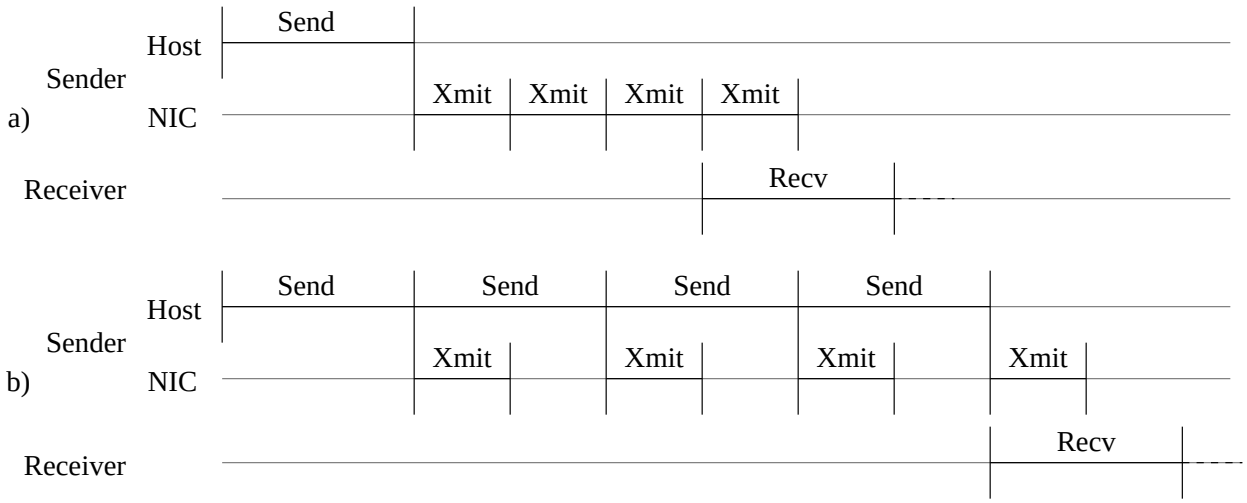

Figure 2: Timing diagram comparing latencies for sending one packet to four destination using a) a multi-send operation and b) host-based point-to-point operations.

2.18. We can see that multi-send is a powerful primitive.

The estimates above were made without write-combining support[2]. Without write-combining, we achieve about 31.7 megabytes per second throughput when the host is writing a packet to the NIC. Assuming write-combining is enabled, we could get 90 megabytes per second throughput[1]. Then the send time would be $(2.7863\mu s + 0.0106\mu s \times m)$ which would be $19.0679\mu s$ for a 1,536 byte message. The multi-send operation would then take $123.2569\mu s$ while the usual host-based point-to-point method would take $154.0174\mu s$. This would lead to a factor of improvement of 1.25. While the improvement is not as great as without write-combining, it is still significant.

3 Broadcast/Multicast with the Multi-send Primitive

While the multi-send primitive is powerful, for covering a large number of destinations we need to perform broadcasts and multicasts hierarchically in order to minimize the overall broadcast/multicast latency. This can be done by having the host at each intermediate node receive the message, then issue another multi-send operation to forward the message to its child nodes. This raises a challenge: how to create an optimal tree? It is also interesting to analyze how this scheme is different from the NIC-based multicast schemes described in [3, 15, 12, 6].

In the NIC-based scheme [3, 15, 12, 6], the incoming multicast packet is transmitted to the child nodes by the NIC after it has been forwarded to the host. Figure 3 illustrates the difference in the two methods. This figure shows two diagrams where node 0 sends a message to multiple destinations, one of which is node 1. Node 1 then sends to nodes 2 and 3. The diagram on the top shows a completely NIC-based scheme where the packet coming into node 1 from node 0 is sent to nodes 2 and 3 by the NIC after being forwarded to the host. The diagram on the bottom shows node 1 forwarding the

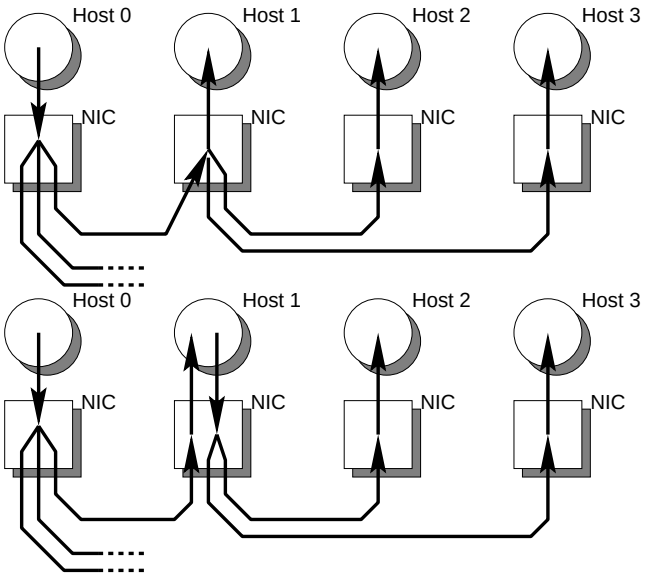

Figure 3: NIC-based multicast (top) and a NIC-assisted multicast (bottom).

[2]In our configuration, the write-combining feature does not seem to be working in the Myrinet driver supplied with the FM distribution.

incoming packet to the host. The host then issues a multi-send operation to transmit the packet to nodes 2 and 3.

While it may seem that the completely NIC-based scheme would always be better than the method we are proposing, we believe that that is not the case. The completely NIC-based approach puts more responsibility on the NIC which, as previously mentioned, has a processor 5-15 times slower than the host. We believe that the additional processing power available at the host will allow greater flexibility in, for instance, tree construction. So that depending on the message size and quality of service requirements, a tree optimal in either bandwidth or latency, for that message size can be used for sending the message. This could be done on a per-message basis. This paper will examine how to construct a multicast tree that is optimal for latency, and study the performance of such a tree.

4 Constructing an Optimal Multicast Tree

In this section, we show how to construct an optimal multicast tree using the proposed new multi-send primitive. The basic idea is to construct a tree such that the maximum number of nodes will be sending at any time. Such a tree would be optimal in terms of latency.

Bar-Noy and Kipnis [2] have shown that in the postal model, a broadcast tree optimal in terms of latency is based on the following recurrence relation. In the postal model, a broadcast to $F_\lambda(t)$ nodes can be completed in t time.

$$F_\lambda(t) = \begin{cases} 1 & \text{if } 0 \le t < \lambda, \\ F_\lambda(t-1) + F_\lambda(t-\lambda) & \text{if } t \ge \lambda. \end{cases}$$

In this recurrence relation, λ is defined as the ratio of (i) the total amount of time from when the sender of a packet starts sending it to when the receiver receives the complete packet and (ii) the amount of time that the sender spends sending the packet. One unit of time is defined as the time the sender spends sending a packet. It then takes λ time for a recipient to fully receive a packet after the sender starts sending it. It is assumed that as soon as a node receives a packet, it will start sending it to its children nodes.

Intuitively, one can think of $F_\lambda(t)$ as the number of nodes which have received the packet at time t. This is equal to the number of nodes which had already received the packet previously (i.e. $F_\lambda(t-1)$), plus the number of nodes which have just received the packet. The packets which were just fully received at time t must have been sent at λ time before then. Since, at that time there were $F_\lambda(t-\lambda)$ nodes which would have sent these packets, there would be that many new nodes receiving the packet at time t.

In our design, the NIC will be transmitting the packets to different nodes. So we need to apply the postal model from the point of view of the NIC. Specifically, when a packet is sent from a node, say, node 0, to another node, node 1, which will receive it and send it to a third node, node 2, then λ is defined as the ratio of i) the time from when the NIC at node 0 starts transmitting the packet to node 1 until the NIC at node 1 is ready to transmit the packet to node 2 and ii) the time it takes the NIC to finish transmitting the packet. Note that part i) of the ratio is simply the one way latency of a packet (i.e. the time it takes for the NIC to transmit the packet plus the time it takes for the host to receive it plus the time for the host to send the packet to the NIC).

To construct the tree, we used an algorithm similar to the "simple top-down greedy algorithm" in [4]. The tree is stored as a list of destination lists, one per host. The algorithm uses two queues called the *new* queue and the *old* queue. We start with the root node and enqueue it in the *new* queue with time 0. Then, for each node p in $(p_1, p_2, \ldots p_{n-1})$ we do the following:

- dequeue the node q that has the minimal time t of both of the queues.
- add p to the destination list of q
- enqueue p onto the *new* queue with time $t + \lambda$
- enqueue q onto the *old* queue with time $t + 1$

While there are algorithms which may construct the tree faster [4], this algorithm is simple and general. Because we were constructing the tree off-line in our test program, we were not concerned with the running time of this algorithm. This algorithm does, however, produce a tree that is optimal in the postal model for a given λ [4, 2].

So far we have focused on multicasting a single packet. For multi-packet messages, there are two ways for intermediate nodes to forward the message to it children. One way would be for the node to receive the whole message, then send the message to its children. The other method would be for the node pipeline the message in a packet-wise fashion. In other words, the node would receive the message one packet at a time, and forward the packets to its children as soon as they are received rather than waiting for the whole message to be received.

The decision on whether to pipeline the message or not will mostly depend on whether the run-time system

supports it. FM, for instance, does not support message pipelining in general (though we were able to do this in a special case). In Section 6 we will show performance results with and without pipelining.

5 Our Implementation of the Multi-send Primitive

We used Illinois Fast Messages (FM) version 2.1 from the HPVM 1.0 distribution for the base of our experiments. This version was used because it is the latest version of FM available for Linux.

In FM 2.1, a message is associated with a stream. In order to send a message, a stream is created between the node and the destination with the FM API call `FM_begin_message()`. Data can then be sent on the stream using the `FM_send_piece()` API call. When enough data is sent to fill a packet, FM creates a packet and writes it to the NIC which in turn transmits it to the destination node. On the receiving side, FM calls a message handler to optionally re-assemble the message, packet by packet, into the user buffer. A stream is terminated using the `FM_end_message()` API call, at which time any remaining data is assembled into a packet and written to the NIC.

To avoid loosing packets due to buffer overflow at the receiver, FM uses a flow control scheme using credit management. One credit corresponds to one packet buffer at the receiver. On starting FM, each sender starts with a certain amount of credits for each receiver. Before any packet is sent FM checks if the sender has sufficient credits for the receiving node. If there are no available credits at the receiver, then the sender blocks until it receives more credits, otherwise, the sender decrements the number of credits it has for that receiver and sends the packet. Whenever a node receives a packet it increments a counter of the number of credits it needs to return to the sender. These credits are returned to the sender either by piggybacking the credits on a data packet sent to that node, or via an explicit credit packet.

We modified FM by adding a new API call, `FM_begin_message_multi()` which creates a stream between the sender and all destinations specified in the destination list given as an argument. As with a regular FM stream, data is sent using `FM_send_piece()` and the stream is terminated with `FM_end_message()`. The flow control mechanism was also modified. Before a packet is sent credits are checked for every one of the destinations. If there are not enough credits for any destination, the operation blocks until there are. Basically, rather than just verifying that we have credits for a single destination, we check that we have enough credits for each destination. For returning credits normal FM unicast packets have a field for piggybacking them. In our scheme we use piggyback credits for each destination as described below.

When a packet is ready to be sent, the packet is assembled for the first destination and written to the send frame on the NIC. Next a list of information on each additional destination is assembled and written to a new field which is added to the send frame. Each entry in the list holds the logical node number, the physical node number and the returned credits for that destination. These are be used to update the corresponding fields in the packet header so that the packet can be sent to each destination.

Since the host assembles the packet such that it is ready to be sent to the first destination, the NIC can transmit the packet without checking whether it is a multi-destination packet. Only after the NIC has initiated the transmit DMA and is waiting for it to complete, will it check for additional destinations. This way our modifications add no overhead at the NIC for sending standard FM messages. The NIC then updates the logical node number, credit and route fields of the packet for the next destination. This is done as soon as the transmit DMA pointer has passed those fields but without waiting for the entire DMA to finish. This allows the header field updates to be overlapped with the DMA for all but the smallest messages. After updating the fields, the NIC waits for the DMA to finish then the DMA is immediately initiated to transmit the packet to the next destination. This is repeated for each additional destination.

6 Experimental Results

The performance tests were run on a cluster of 16 300MHz Pentium II machines each with 128MB of RAM running RedHat 5.2 Linux with kernel version 2.0.36. The machines are connected by a Myrinet LAN network with LANai 4.3 cards via a 16 port switch.

We tested the performance of the multi-send primitive and compared it with multiple unicast sends. In every iteration of our test routine, the root would send messages to the destinations, and then the last destination would send a zero byte message back to the root after receiving a copy of the message. This was timed for 1,000 iterations, then the average was taken for the result. This was done varying the message size and number of destinations. Figures 4 and 5 show the results of this test. Notice that for messages less than 32 bytes, our scheme performs slightly worse than the host-based

scheme. This is due to the fact that the NIC transmit time is not smaller than the host send time and due to the overhead of adding the multi-send primitive to FM. However, the NIC-assisted scheme performs clearly better than the host-based scheme for larger messages. It can be observed that the multi-send primitive achieves a factor of improvement of 3.51 for sending a 16K message to 15 destinations.

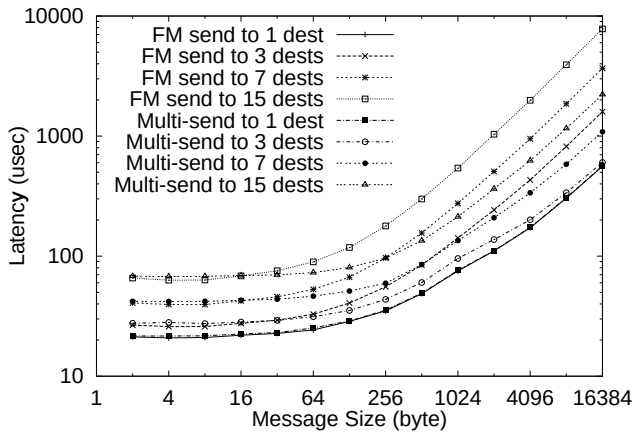

Figure 4: Latency for NIC-assisted multi-send operation and multiple FM send operations.

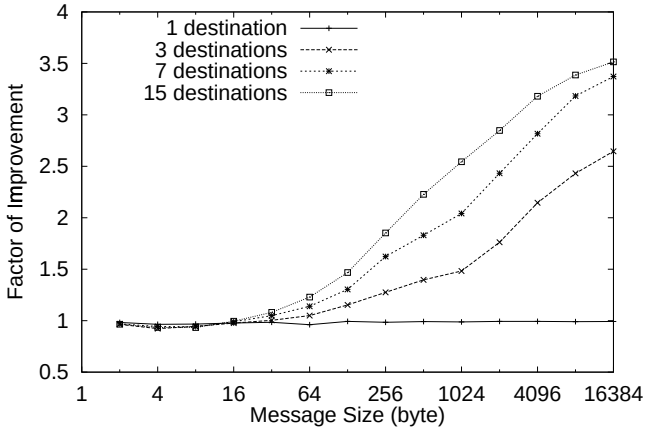

Figure 5: Speedup for NIC-assisted multi-send operation versus multiple FM send operations.

To test the performance of a multicast operation using the multi-send primitive, we ran a test similar to the one described above. One iteration of the test routine would send a message along a multicast tree, then one of the leaf nodes would send a zero byte message back to the root. This was timed for 1,000 iterations then the average was taken. Because we couldn't be sure which leaf node would receive the message last, we ran the loop several times, each time changing the leaf node which would return the message, then taking the maximum value. In order to cut down on the number of leaf nodes to test we only tested the leaf nodes which were the last children of their parents. This was then varied for message size and number of destinations. When we were using the optimal multicast tree in our tests, we needed to use a value for λ which would produce the best performance. Since the value of λ depends on the message size, in our test program, for each message size we constructed a new tree based on integer λ values from 1 to the number of nodes participating, and took the minimum. So each point in our multicast performance graphs is the minimum over each tree, of the maximum for each responding leaf node in that tree, of the average of 1,000 iterations.

Also, to study the performance impact of pipelining for multi-packet messages, we incorporated modifications to our test program as outlined in Section 2. As soon as a packet was received by the intermediate node, it would be sent out, rather than wait for the whole message to arrive. To pipeline a message, the application would have to open a stream to its destination(s), then receive the incoming message one packet at a time until it has received it completely, then close the stream.

This cannot normally be done in FM 2.1 because while a stream is open, FM does not allow the application to make a call to receive parts of a message. This is done to avoid certain deadlock conditions. When a stream is opened with `FM_begin_message()` or `FM_begin_message_multi()`, FM sets a lock so that any calls to receive a message return immediately without receiving. To get around this, we used a version of this call, `FM_unsafe_begin_message_multi()` which does not set a lock. This version is intended to be used only inside a message handler. We used it outside a message handler in our main routine. Since the lock is not set when we open the stream, we are then able to receive the packets of the incoming message and send them out. Though this approach may lead to deadlock in the general case, because our test programs ran one multicast at a time, there were no cyclical dependencies and no deadlock could occur. We used this method, not to show how to pipeline messages in FM, but rather to demonstrate the performance of our scheme when pipelining is possible.

We will next compare NIC-assisted multicasting to host-based multicasting. Because the binomial tree is most often used for host-based multicast operations, as used in MPI, we will use the binomial tree for the host-based multicast, and compare it to the NIC-assisted multicast using an optimal tree as discussed in Section 2. Then, because the binomial tree may not be the best tree for host-based multicast, we will use an optimal tree for the host-based multicast and compare that with

the NIC-assisted multicast also using an optimal tree. In each case, we will show the impact of message pipelining.

Figures 6 and 7 compare NIC-assisted multicast using an optimal tree with pipelining against the host-based multicast using a binomial tree. Notice that the NIC-assisted scheme is better for every message size and every number of destinations. The dip in the factor of improvement for 2048 byte messages with 16 nodes is due to packetization effects. The graph also shows a factor of improvement of 1.86 for multicasting a 16K message to four nodes and a factor of improvement of 1.85 for multicasting a 16K message to 16 nodes.

Figures 8 and 9 show the same comparison as above but without message pipelining. We can see that the when compared to the pipelined case, performance does not change for one packet messages (FM packets are 1,536 byte) or for four nodes of any size. This is because pipelining does not occur for single packet messages, and for a four node broadcast the optimal tree is flat ($\lambda = 2$), i.e., the root sends the message directly to all the three destinations, so again no pipelining would occur. While the performance improvement is not as great for multi-packet messages with eight and 16 nodes, when compared to the case when messages are pipelined, there is still a 1.53 and 1.30 factor of improvement for 16K messages with eight and 16 nodes, respectively.

To see what impact the shape of the tree had on the performance we are seeing, the optimal tree algorithm was used with the host-based unicast primitive and com-

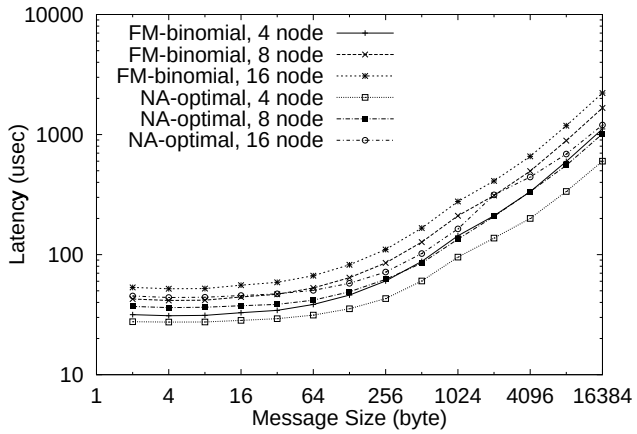

Figure 6: Multicast latency for NIC-assisted multicast using an optimal tree *with packet-wise pipelining* (NA-optimal), and multicast using binomial tree with FM unicast send (FM-binomial).

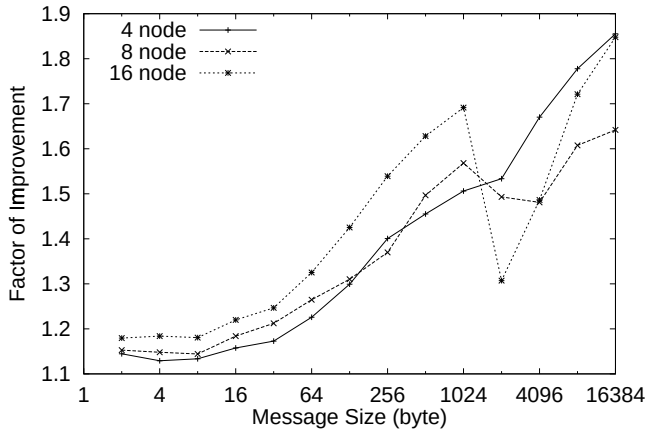

Figure 7: Multicast factor of improvement for NIC-assisted multicast using an optimal tree *with packet-wise pipelining*, compared to multicast using binomial tree with FM unicast send.

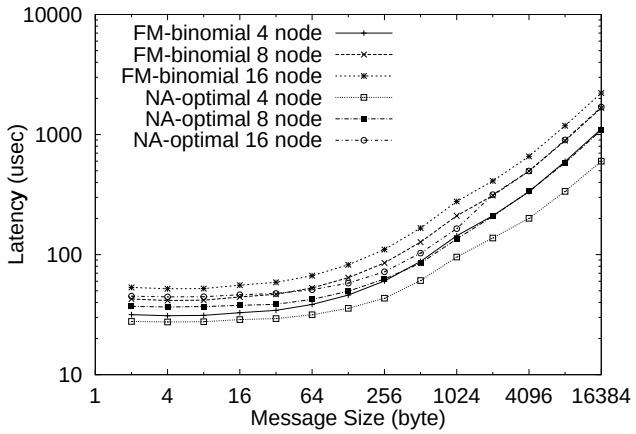

Figure 8: Multicast latency for NIC-assisted multicast using an optimal tree *without packet-wise pipelining* (NA-optimal), and multicast using binomial tree with FM unicast send (FM-binomial).

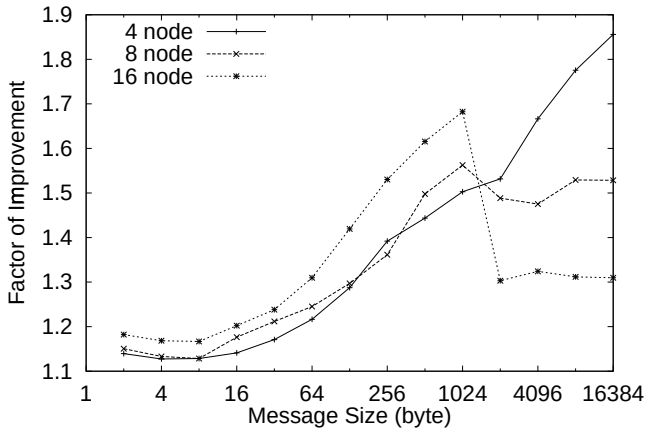

Figure 9: Multicast factor of improvement for NIC-assisted multicast using an optimal tree *without packet-wise pipelining*, compared to multicast using binomial tree with FM unicast send.

pared with the same data for the multi-send primitive above. Figures 10, 11, 12 and 13 show the results of this comparison. In this test, our scheme performs a little worse than the host-based method for messages less than 32 bytes. We believe that this is due to the overhead of the additions to FM similar to that observed for Figure 4. For multi-packet messages, the optimal tree for the host-based method turned out to be a binomial tree (i.e. $\lambda = 1$), so the performance improvement for those messages is the same as in the previous graphs.

7 Related Work

NIC-based multicasting has been previously proposed by Verstoep et.al. [15] and Bhoedjang et.al. [3]. Their schemes perform the entire multicast operation at the NIC, as opposed to our scheme which uses a multi-send operation as a primitive for multicast.

Verstoep, et.al. extend FM 1.1 to include NIC-based multicasting. In their scheme (called FM/MC)[15], the multicast is performed completely at the NIC-level. At intermediate nodes, the packet is forwarded to the host, but then immediately transmitted to the child nodes without involving the host. The host receive queues are divided into multicast queues and unicast queues. In order to prevent buffer overflow,

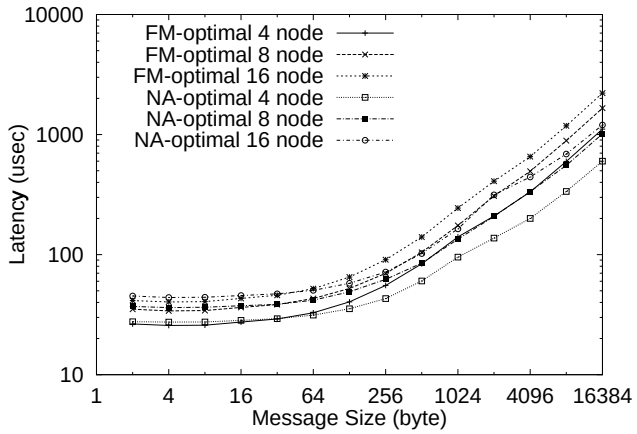

Figure 10: Multicast latency for NIC-assisted multicast using an optimal tree *with packet-wise pipelining* (NA-optimal), and multicast using an optimal tree with FM unicast send (FM-optimal).

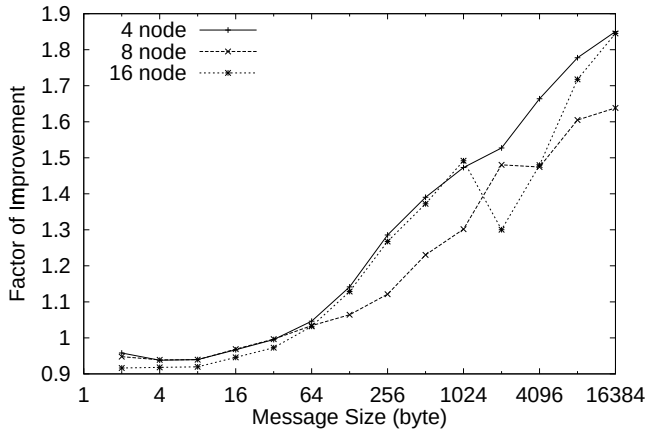

Figure 11: Multicast factor of improvement for NIC-assisted multicast using an optimal tree *with packet-wise pipelining*, compared to multicast using an optimal tree with FM unicast send.

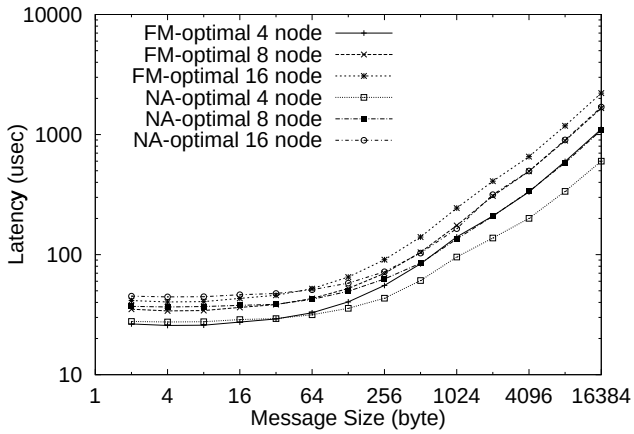

Figure 12: Multicast latency for NIC-assisted multicast using an optimal tree *without packet-wise pipelining* (NA-optimal), and multicast using an optimal tree with FM unicast send (FM-optimal).

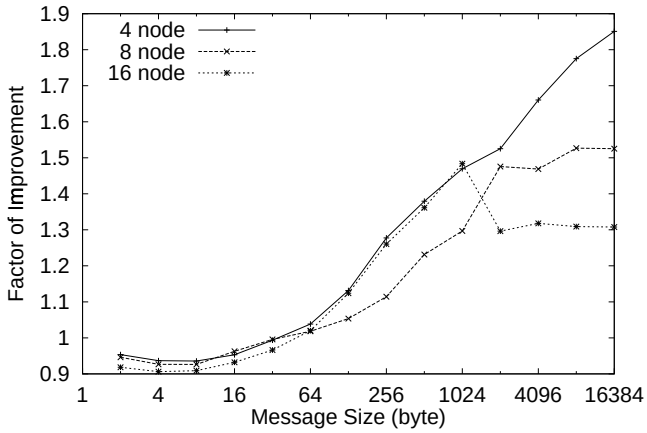

Figure 13: Multicast factor of improvement for NIC-assisted multicast using an optimal tree *without packet-wise pipelining*, compared to multicast using an optimal tree with FM unicast send.

credits are managed separately for each. One multicast credit corresponds to one packet buffer in *each* host in the network, rather than just for one host, as with unicast credits. One NIC on the network is designated as a credit manager which distributes and collects the multicast credits. A NIC must request credits from the manager before multicasting a packet. However, once a multicast has been initiated, the intermediate nodes do not need to check for credits and can immediately forward the packet to their children when they receive it.

Bhoedjang et.al. propose a new message passing protocol called Link-level Flow Control (LFC)[3]. This system does all flow control at the NIC and also performs the multicast completely at the NIC. In this system data for a packet is copied to a buffer pool on the NIC by the host, then the host writes a descriptor to the send queue. The NIC polls the send queue and transmits the packets when there are sufficient credits. In order to send a multicast packet, the host simply has to add one descriptor for each destination but refer them to the same data. At an intermediate node, the NIC receives the packet then adds descriptors for its children to the send queue.

The main difference between our scheme and the schemes described above, is that a multicast operation in the schemes described above is done completely at the NIC. Also, our scheme keeps the receive queues and credit management unified for both unicast and multicast messages.

8 Conclusions and Future Work

We have introduced a new multi-send primitive, which uses the NIC to transmit multiple copies of a packet to different destinations. We then showed how this primitive can be used in a multicast tree to further improve performance for large numbers of destinations.

The multi-send primitive gave us a 3.51 factor of improvement over conventional host-level iterative sends for 16K messages. We also observed a 1.85 factor of improvement for 8K messages and 16 nodes when using the primitive in a multicast operation using an optimal tree versus using a binomial tree with the traditional host-level unicast sends.

The current evaluations are based on a 16-node cluster with a 16 port myrinet switch. Thus, there is no network contention during the multicast operation. For larger systems, network contention will play an important role. We plan to develop a method for constructing trees for multicast using multi-send primitives which would give minimal contention for arbitrary irregular networks. Also it would be interesting to study the effects of a workload on such an algorithm and the resulting performance on the multicast.

On a more general level we intend to investigate whether other collective communication operations, such as barriers, reductions, or personalized multicast could benefit from similar NIC-assisted primitives.

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).

Appendix

This appendix explains how we arrived at the estimates shown in Section 2. We used the RDTSC Pentium assembly instruction[14] which reads the Pentium's internal 64 bit cycle counter. If one knows the speed of the processor, one can use this counter for accurate timings.

To estimate t_{Send} we timed the send portion of a simple ping-pong program and averaged it over 10,000 iterations. We did this for various message sizes up to one packet size (1,536 bytes).

To estimate t_{Xmit} we used the RTC register on the NIC's LANai chip. This register is incremented at a regular interval. To time this interval we did the following. First, we read the NIC's RTC register from the host then immediately read the Pentium cycle counter. We then repeatedly slept for one second, read the RTC and Pentium cycle counter, and compared the values to the initial reading, then calculated the elapsed time. This was done until the resulting value converged. By looping until the value converged, we were able to eliminate, to an arbitrary degree of accuracy, the delay of reading the RTC register over the PCI bus.

Once we calibrated the RTC register period, we timed the NIC transmit time by subtracting the RTC time from the variable storing the total transmit time as soon as the NIC noticed that a packet was added to the send queue. Then as soon as the transmit DMA was completed, we added the value of the RTC register to this variable. This was done for sending 10,000 packets, then this value was read by the host and divided by 10,000. To make sure that no other packets were sent during this time, such as credit packets, the FM library code was modified to prevent such packets from being sent.

Note that $t_{Send} + t_{Xmit}$ does not include the time between when the host finishes copying the packet to the NIC and when the NIC notices that the packet is there. We expect this time to be small.

We timed the interval for the NIC to receive a message and DMA it to the host similarly to how we timed the t_{Xmit} value. We took the time from when the NIC started the DMA from the network to a receive buffer, until the packet had finished being DMAed to the host. We then timed the interval for the host to perform the copy of the packet from the DMA region where the NIC had placed it, to the application buffer. We took the sum of these two measurements for the value of t_{Recv}.

Note that this time does not include the network latency, which should be very small because we have only one switch, and the time from when the NIC finishes DMAing the packet until when the host notices that the packet is there. Again, we feel that this should be a small value.

References

[1] S. Araki, A. Bilas, C. Dubnicki, J. Elder, K. Konishi and J. Philbin, "User-Space Communication: A Quantitative Study". *Proceedings of the 1998 SC'98 Conference*. November, 1998.

[2] A. Bar-Noy and S. Kipnis, "Designing Broadcast Algorithms in the Postal Model for Message-Passing Systems". *Mathematical Systems Theory*, 27(5), pp. 431-452, September 1994.

[3] R.A.F. Bhoedjang, T. Ruhl and H.E. Bal, "Efficient Multicast On Myrinet Using Link-Level Flow Control". *International Conference on Parallel Processing*, pp. 381-390, August 1998.

[4] J. Bruck, L. De Coster, N. Dewulf, C. Ho and R. Lauwereins, "On the Design and Implementation of Broadcast and Global Combine Operations using the Postal Model". *IEEE Transactions on Parallel and Distributed Systems*, 7(3), pp. 256-265, March 1996.

[5] N. Boden, D. Cohen, R. Felderman, A. Kulawik, C. Seitz, J. Seizovic, and W. Su, "Myrinet: A Gigabit-Per-Second Local Area Network". *IEEE Micro* 15(1), pp. 29-36, February 1995.

[6] R. Kesavan and D.K. Panda, "Optimal Multicast with Packetization and Network Interface Support". *International Conference on Parallel Processing (ICPP 1997)*, pp. 370-377.

[7] M. Lauria, S. Pakin and A. Chien, "Efficient Layering for High Speed Communication: Fast Messages 2.x". *Proceedings of the 7th High Performance Distributed Computing (HPDC7) Conference*, July 28-31, 1998.

[8] Message Passing Interface Forum, *MPI-2: Extensions to the Message-Passing Interface*. July 1997.

[9] S. Pakin, M. Lauria and A. Chien, "High Performance Messaging on Workstations: Illinois Fast Messages (FM) for Myrinet". *In Supercomputing '95*.

[10] D.K. Panda, "Fast Barrier Synchronization in Wormhole k-ary n-cube Networks with Multidestination Worms". *International Symposium on High Performance Computer Architecture (HPCA '95)*, pp. 200-209, 1995.

[11] D.K. Panda, S. Singal, and R. Kesavan, "Multidestination Message Passing in Wormhole k-ary n-cube Networks with Base Routing Conformed Paths". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 10, No. 1, pp. 76-96, January 1999.

[12] R. Sivaram, R. Kesavan, D.K. Panda, and C.B. Stunkel, "Where to Provide Support for Efficient Multicasting in Irregular Networks: Network Interface or Switch?" *International Conference on Parallel Processing (ICPP 1998)*, pp. 452-459.

[13] C.B. Stunkel, R. Sivaram, and D.K. Panda "Implementing Multidestination Worms in Switch-Based Parallel Systems: Architectural Alternatives and their Impact". *International Symposium on Computer Architecture (ISCA'97)*, Vol. 25, No. 2, pp. 50-61, June 2-4 1997.

[14] *Using the RDTSC Instruction for Performance Monitoring*. available from http://developer.intel.com/drg/pentiumII/appnotes/RDTSCPM1.HTM

[15] K. Verstoep, K. Langendoen, and H.E. Bal, "Efficient Reliable Multicast on Myrinet". *1996 International Conference on Parallel Processing*, Vol. 3, pp. 156-165, August 1996.