# Performance Benefits of NIC-Based Barrier on Myrinet/GM[*]

Darius Buntinas Dhabaleswar K. Panda P. Sadayappan
Network-Based Computing Laboratory
Department of Computer and Information Science
The Ohio State University
Columbus, OH 43210
{buntinas, panda, saday}@cis.ohio-state.edu

## Abstract

*Barrier synchronization is a common operation in parallel and distributed systems. A fast implementation is important because it allows fine grained parallel programs to be more efficient. It is therefore important to minimize the latency of barrier operations. Modern network interface cards (NICs) have programmable processors which can be used to support collective communications such as barrier. In [4] we have designed and implemented a* NIC-based *barrier feature over GM. This new NIC-based barrier operation raises many open questions which must be answered. Does the NIC-based barrier perform better than the host-based barrier? How does the performance of the NIC-based barrier change with better NICs? Is the NIC-based barrier scalable? How does the performance of the NIC-based barrier affect the granularity of computation? How does the NIC-based barrier affect the performance of applications? In this paper, we take on these challenges. We find that the NIC-based barrier performs better than the host-based barrier with up to a 2.22 factor of improvement on an eight node system at the MPI-level. We also find that the factor of improvement values increase with the number of nodes indicating that the NIC-based barrier is more scalable. We find that the NIC-based barrier also allows for finer grained computation without affecting the efficiency of the program. These results indicate that NIC-based barrier in current and future clusters can deliver significant performance benefits to the applications.*

## 1 Introduction

Barrier synchronization is a common operation in parallel and distributed systems. It is a common synchronization operation in both message passing as well as shared memory systems [2, 11, 14, 10, 15, 13, 6]. An efficient implementation is important because during the operation, generally no computation can be performed, unless the programming model supports split-phase, or fuzzy, barriers[9]. The commonly used MPI model does not support split-phased barriers. A fast barrier implementation also impacts the granularity of the parallel program: A fine grained parallel program will not be efficient if the barrier latency is high. It is therefore important to minimize the latency of barrier operations.

In a typical barrier operation, participating nodes exchange messages to determine whether all nodes have reached the barrier. On current clusters, the barrier protocol is typically processed at the host. For instance, a barrier message originates at the host of one node, it is transferred to the *Network Interface Card* (NIC) which transmits it to the NIC of the other node. That NIC receives the barrier message and transfers it to the host. If necessary, the host will send another barrier message to another host which must be transferred to the NIC to be transmitted as before. Thus, a typical barrier operation on current systems using host-based point-to-point operations may be very expensive. Modern NICs have programmable processors. Such programmable NICs can be used to support collective communications such as barrier in an efficient manner. In [4] we have designed and implemented a *NIC-based* barrier over GM, Myricom's message passing system[12]. In a NIC-based barrier the protocol is processed at the NIC rather than at the host. By basing the barrier protocol at the NIC, the barrier messages need not be passed up to the host to have the next message sent back down from the host to the NIC. Instead, upon the reception of a barrier message, the NIC can immediately send the next barrier message. This leads to a more efficient implementation of barrier with reduced latency.

This new NIC-based barrier operation raises many open questions which must be answered. Does the NIC-based barrier perform better than the host-based barrier? While we have shown in [4] that the NIC-based barrier gives up to a 1.83 factor of improvement over the host-based barrier at the GM level, it remains to be seen how much performance benefit can be observed at the MPI level using the new implementation. Is the NIC-based barrier scalable? How does the performance of the NIC-based barrier change with better NICs? Given that cluster sizes are growing we must see whether the operation will still be efficient over a large number of nodes. How does the performance of the NIC-based barrier affect the granularity of computation? With a more efficient barrier operation, finer grained parallel programs should be possible. How does the NIC-based barrier affect the performance of applications?

[*]This research is supported in part by an NSF Career Award MIP-9502294, NSF Grants CCR-9704512 and EIA-9986052, an Ameritech Faculty Fellow Award, and grants from the Ohio Board of Regents.

In this paper, we take on these challenges. We have modified the MPICH[8] port over GM to use the NIC-based barrier operation implemented in [4]. This was done in an efficient manner and it introduces only a 3.22$\mu$s overhead to the GM NIC-based barrier operation for 16 nodes. This allows many existing MPI applications to take advantage of this efficient operation. Furthermore, we find that the NIC-based barrier performs better than the host-based barrier at the MPI level: The latency of a 16 node MPI barrier using the NIC-based barrier operation is 105.37$\mu$s while that using the host based barrier operation is 216.70$\mu$s. We also find that the NIC-based barrier is more scalable than the host-based barrier: The factor of improvement values increase with the number of nodes participating in the barrier. We find that the NIC-based barrier also allows for finer grained computation without affecting the efficiency of the program. Finally, by using synthetic applications on an eight node system, we find up to a 1.93 factor of improvement in the applications using a NIC-based barrier versus using a host-based barrier.

The paper is organized as follows. Section 2 provides an overview of the NIC-based barrier and the basic barrier algorithm. Section 3 describes the changes we made to MPICH to use the NIC-based barrier. In Section 4 we evaluate the NIC-based barrier implementation along the dimensions discussed above. Section 5 concludes the paper.

## 2 Overview of NIC-based barrier

We presented the design and implementation of the NIC-based barrier in [4]. The implementation was done as a modification to GM, Myricom's message passing system[12]. In order to perform the evaluations described in this paper, we additionally had to modify the MPICH[8] port over GM, version 1.2..3, to use the NIC-based barrier. Here we give a short overview of the concept behind the NIC-based barrier and its implementation on GM. Readers are encouraged to refer to [4] for details.

### 2.1 Basic idea

In a NIC-based barrier operation, the barrier protocol is performed at the NIC, rather than at the host. The host commands the NIC to start the barrier and the NIC informs the host when the barrier has completed. Figure 1 shows block diagrams for host-based and NIC-based barriers on a four node system. In both examples the same basic protocol is used, except the difference is *where* the protocol is being performed. The basic protocol for four nodes proceeds in two steps. In the first step, nodes 0 and 1 exchange messages at the same time that nodes 2 and 3 are exchanging messages. Once those messages have been exchanged, the second step proceeds where nodes 0 and 2 exchange messages while nodes 1 and 3 are exchanging messages. Note that messages which are exchanged within a step, e.g., when nodes 0 and 1 are exchanging messages, they are sent concurrently. For example, node 0 sends its message to node 1 immediately, without waiting to receive the message from 1, and similarly, node 1 sends its message to node 0 without waiting for the message from node 0. Once a node has sent the message for that step, however, it must wait to receive the corresponding message before completing that step.

In Figure 1, the diagram on the left represents the host-based barrier (as currently implemented in MPICH[8]). In

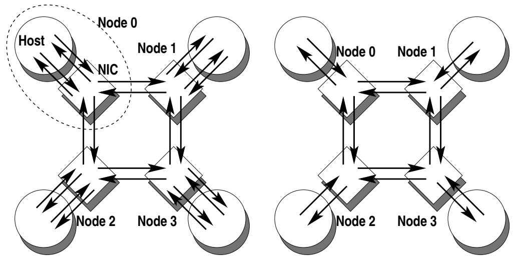


**Figure 1. Host-based barrier (left) and NIC-based barrier (right)**

this diagram, each of the messages originates at a host and is received by a host. This means that before a node completes the first step, the message sent to it in that step must be received by the NIC and passed up to the host, then the message for the second step must be passed back down to the NIC to be transmitted to the other node. The diagram on the right in Figure 1 represents the NIC-based barrier. In this diagram, the host sends a message to the NIC then waits for a notification from the NIC that the barrier has completed. The protocol messages are exchanged between the NICs. Because the messages do not have to be passed all the way to the host, the time is reduced between when the message of one step is received and the message from the next step is sent.

### 2.2 Barrier algorithm

We have implemented a NIC-based barrier on GM, Myricom's message passing system, version 1.2.3 in [4].

In [4] we implemented the NIC-based barrier using two algorithms. In this paper we only consider one algorithm, the *pairwise exchange* algorithm, since this performed better than the other. This is the same basic algorithm used in the MPICH implementation of barrier. At a conceptual level, the algorithm runs as follows. Assuming we have a power-of-two number of nodes, we start the algorithm by placing each node into its own group; i.e., if we have $N$ nodes, then there are $N$ groups each with one node. Then the algorithm proceeds recursively by merging groups, two at a time, until there is only one group. The algorithm is then finished. To merge two groups, each node in one group exchanges messages with exactly one node in the other group. The nodes in those two groups then form one new group. Because the message exchanges can happen concurrently, this can be accomplished in one step. The algorithm runs in a total of $\log_2 N$ steps if $N$ is a power of two.

If $N$ is not a power of two, then we divide the nodes into a set $S$ and a set $S'$ such that $|S|$ is the largest power of two less than $N$. We then pair each node in $S'$ with a node in $S$. Each node in $S'$ will send a message to its corresponding node in $S$, which waits for the message. Next, the nodes in $S$ perform a barrier as described above. Finally, the nodes in $S$ which had received a message from the nodes in $S'$ send a message back to their corresponding node in $S'$. For non-power-of-two number of nodes the algorithm should run in $\lfloor \log_2 N \rfloor + 2$ steps.

### 2.3 Estimated performance benefits

Figure 2 shows timing diagrams comparing the latencies of an eight node barrier using the host-based barrier and

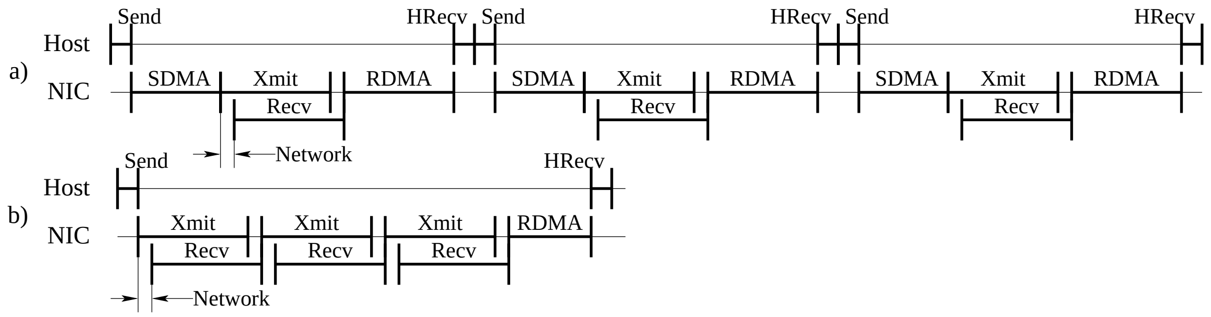


**Figure 2. Timing diagrams comparing latencies for a) host-based barrier and b) NIC-based barrier.**

a NIC-based barrier. The barrier algorithm for eight nodes proceeds similarly to the four node barrier described in Section 2.1, except that it has three steps rather than two. For simplicity, in this example we assume that each node starts the barrier at exactly the same time. We also assume that the network is wormhole routed and that the NICs have separate send and receive ports, so a message can be sent and received simultaneously.

Figure 2(a) shows the timing diagram for one node performing a host-based barrier. The diagram is divided into two timelines, one for the host and one for the NIC. On the host timeline, $Send$ represents the time spent by the host to initiate a send on the NIC. Once the send has been initiated, $SDMA$ on the NIC timeline represents the time the NIC spends transferring the data for the message from the host memory to the NIC send buffer using DMA. Next, $Xmit$ represents the time the NIC spends transmitting the message onto the network. Since we assume that the network is wormhole routed, and that all other nodes started their barriers at exactly the same time, the corresponding message arrives at the NIC after a delay of $Network$ from when the message had started being transmitted. $Recv$ represents the time for the message to be received from the network into the NIC receive buffers. Once the message has been received into a receive buffer, the message is transferred to the host using DMA. This time is represented by $RDMA$. $HRecv$ represents the time for the host to process the received message. This same process is repeated two more times to exchange messages with the other two nodes.

Figure 2(b) shows the timing diagram for one node performing a NIC-based barrier. Here, $Send$ represents the time the host spends to initiate the barrier operation. Because there is no data to be transferred from the host, the NIC can immediately transmit a barrier message. $Xmit$ represents the time that the NIC spends transmitting the barrier message. After a delay of $Network$ the corresponding message arrives at the receiver. Once the message is received at the receiver NIC, it (the receiver NIC) can immediately start transmitting the next message. Once the last message has been received, the NIC transfers the completion notification to the host. This time is represented by $RDMA$. Finally, $HRecv$ represents the time for the host to process the notification. From these diagrams, we can see that the latency of an eight node host-based barrier is $3 \times (Send + SDMA + Network + Recv + RDMA + HRecv)$, while the latency for a NIC-based barrier is only $(Send + 3 \times (Network + Recv) + RDMA + HRecv)$. This suggests that the NIC-based barrier should perform better than the host-based barrier. It also suggests that performance benefits may increase with respect to system size, send/receive overheads, etc. However, the exact benefits are not known. We focus on this aspect in Section 4.

## 3 Modifications to MPICH to use the NIC-based barrier

In this section we describe the modifications we made to MPICH so that it uses the NIC-based barrier. First we give an overview of how GM works and the modifications that were made to GM to support NIC-based barrier in [4]. Then, we describe our modifications to MPICH.

### 3.1 Overview of GM

GM consists of a driver, a library and a Myrinet control program (MCP). The driver loads the MCP on to the NIC when it is loaded. During the execution of a program the driver is used mainly for opening *ports*, pinning and unpinning memory, and to put processes to sleep or wake them when blocking functions are used. A *port* is an abstraction through which a process can communicate with the NIC. Once a port is opened, the process can communicate with the NIC, bypassing the operating system and avoiding system call overhead. Each NIC can support a maximum of eight ports, some of which are reserved.

At the host level GM is connectionless, but provides reliability by maintaining reliable connections between NICs of different nodes. Flow control is used between the NIC and the host to avoid buffer overflows. To provide this reliability GM uses the concept of tokens. When a process opens a port, it has a certain number of *send tokens* and *receive tokens*. Each send token corresponds to a send event. To send a message, the process calls `gm_send_with_callback()` to fill in a send token describing the send event and queues it on the token queue. The NIC polls this queue for new send tokens. When the NIC gets a new send token it will DMA the data from the specified host buffer, and transmit the packet to the destination. Once the NIC has completed the send, and has freed the resources corresponding to that event, the send token is returned to the process. This is done implicitly when the callback function associated with the send call is executed. The host should not modify the data which is to be sent until the send token is returned from the NIC. When a deprecated version of the send call, `gm_send()`, is used, the send token is returned to the process using the receive operation instead of the callback.

In order to receive a message, the process must allocate a buffer into which the message will be received and pass a receive token describing the buffer to the NIC using `gm_provide_receive_buffer()`. Once the NIC has DMAed the data from a received message into the buffer, the receive token is returned to the process. The process detects returned receive tokens by polling `gm_receive()` or by calling `gm_blocking_receive()`. Messages may only be sent from and received into buffers which are pinned in memory. Memory is pinned using special functions supplied by GM.

### 3.2 NIC-based barrier in GM

In [4], we added two new procedures to GM called `gm_provide_barrier_buffer()` and `gm_barrier_with_callback()`. The `gm_provide_barrier_buffer()` procedure transfers a barrier receive token to the NIC. The NIC will return this token to the process when the barrier has completed. This procedure is actually a misnomer because no buffer is needed by the barrier. It was named this because it is the analog of `gm_provide_receive_buffer()`.

The `gm_barrier_with_callback()` procedure fills in a send token describing the nodes and ports with which to exchange messages and queues it on the send queue. The NIC, upon receiving the token, will perform the barrier operation, then return the receive token, previously provided by a `gm_provide_barrier_buffer()` call, to the process indicating that the barrier has completed. Note that the send token need not be returned when the receive token is returned. For instance, if there is a non-power of 2 number of nodes participating in the barrier, then some nodes in the set $S$, described in section 2.2, will be sending messages to the nodes in the set $S'$. In this case, the NIC need not wait for this last message to be sent before returning the receive token to notify the process. The send token will then be returned when this last send is complete.

### 3.3 MPICH modifications

MPICH is designed using a layered approach, such that all that needs to be done to port MPICH to use a new communications device, is to write a new *channel interface* for that device. The channel interface defines a set of low level data-transfer primitives which are used by the upper layers. The host-based `MPI_Barrier()` barrier operation is normally implemented at an upper layer using the high level `MPI_Sendrecv()` call. However, if a barrier operation is implemented in a channel interface, then by defining the `MPID_Barrier` and `MPID_FN_Barrier` macros, that operation will be used instead of the upper layer one.

We modified MPICH version 1.2..3 which has been ported to use GM[12] by Myricom. In the GM channel interface, a send is queued at the host until there is a send token available to issue a `gm_send()` call. When the token is returned, the entry for that send is marked as complete. Receive requests are similarly queued, and marked as complete when the receive token is returned. The `MPID_DeviceCheck()` procedure receives messages from the NIC and marks the completed sends as complete. It also keeps track of the token counts and sends pending messages when send tokens are available.

We implemented a low level procedure called `gmpi_barrier()` to perform the NIC-based barrier. The macros described above were defined to refer to this function. This function first determines the list of nodes with which the NIC will exchange messages. This is done using the same basic algorithm that the MPICH host-based barrier algorithm uses, which was described in Section 2.2. Next, the procedure calls `MPID_DeviceCheck()` until all pending sends and receives have completed and until there is at least one send token and at least one receive token available. Now the procedure is ready to perform the barrier. The procedure calls `gm_provide_barrier_buffer()` followed by `gm_barrier_with_callback()` and decrements the send and receive token counts. The procedure now sets a flag `barrier_done` and polls `MPID_DeviceCheck()` until the flag is set. The procedure `MPID_DeviceCheck()` was modified to set this flag when a barrier receive token is received. When the callback function associated with the barrier is called, the send token count is incremented.

## 4 Performance evaluation

In this section, we evaluate performance benefits of our implementation. The evaluation is done along multiple angles:

**MPI-level overhead:** We have evaluated the amount of overhead that the MPI layer adds to the barrier latency compared to the GM-level barrier. This will indicate whether applications at the MPI level can effectively utilize the performance of the NIC-based barrier.

**MPI-level performance and scalability:** Scalability is an important factor in collective communications. Modern clusters can have 1,000 or more nodes, so it is important that collective communication operations such as barrier perform well as the system size increases. We evaluated the performance of the NIC-based barrier and compared it to the host-based barrier at the MPI level. We also compared the scalability of the NIC-based barrier to the host-based barrier.

**Granularity of computation:** The latency of a barrier operation affects the granularity of computation. If the cost of performing a barrier is high, then the amount of computation performed between barriers will have to be large, otherwise the efficiency of the program suffers. So, by reducing the latency of the barrier, the program can be written using finer granularity without loosing efficiency. We evaluated the performance of a computational loop with barrier for varying granularity of computation.

**Varying arrival times:** In most real applications, nodes participating in a barrier may arrive at the barrier at different times. We evaluated the performance of the NIC-based and host-based barriers while varying the delay between the barriers.

**Performance evaluation with synthetic application:** Because the barrier latency in many message passing systems has been high, few benchmarks exist which use barrier heavily. For instance, the NAS[1] benchmarks use very few barriers. In order to simulate a higher granularity application, we wrote a synthetic benchmark. Each synthetic application performs several phases of computation each followed by a barrier. The length of the computation varies from one phase to the next. We compared the overall execution times of the applications using NIC-based barrier and host-based barrier.

The performance results were run on a cluster of 16 dual 300MHz Pentium II machines each with 128MB of RAM, running RedHat 6.0 with SMP kernel version 2.2.5. The machines are connected by a Myrinet[3] LAN network using NICs with 33MHz LANai 4.3 processors. These are connected to a 16 port switch. Eight of these machines are also connected by another Myrinet LAN network using NICs with 66MHz LANai 7.2 processors. These are connected to an eight port switch.

We describe the results of our evaluation in the following subsections.

## 4.1 MPI-level overhead

To evaluate the MPI overhead, we compared the latencies of the barrier operations at the GM level and at the MPI level. To determine the latency of the barriers, we performed 10,000 consecutive barriers and took the average latency of each barrier at each node. Figure 3 shows the results of these experiments using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs. When using the 33MHz NICs there was a 3.22$\mu$s overhead for the 16 node NIC-based barrier. For the 66MHz NICs, there was only a 1.16$\mu$s overhead for the eight node NIC-based barrier. Note that the overhead of MPI operation to initiate a NIC-based barrier does grow slightly with the number of nodes. For the pairwise-exchange algorithm that we are using, it grows at a rate of $\log_2 N$, where $N$ is the number of nodes participating in the barrier. By taking this into account, it can be observed that our MPI-level barrier implementation to use the GM-level NIC-based barrier is extremely efficient.

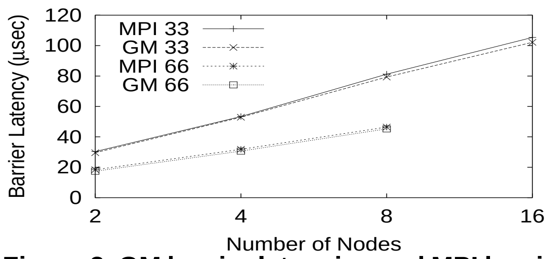


Figure 3. GM barrier latencies and MPI barrier latencies of NIC-based barriers using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs

## 4.2 MPI-level performance and scalability

To evaluate the performance of NIC-based barriers at the MPI-level, we performed 10,000 consecutive barriers using the `MPI_Barrier()` function, and took the average latency of each barrier at each node. These experiments were performed for host-based and NIC-based barriers using both LANai 4.3 NICs and LANai 7.2 NICs. Figures 4(a) and 4(b) show the results of these experiments for power-of-two numbers of nodes. Figure 4(a) shows a latency of 105.37$\mu$s for the NIC-based barrier (NB) compared to 216.70$\mu$s for the host-based (HB) barrier using the 33MHz LANai 4.3 NICs for a 16 node barrier. Similarly, a barrier latency of 46.41$\mu$s is observed for the NIC-based barrier using the 66MHz LANai 7.2 NICs for an eight node barrier, compared to 102.86$\mu$s for the host-based barrier.

Figure 4(b) shows the factors of improvement for the NIC-based barrier over the host-based barrier. Notice that the NIC-based barrier delivered a 2.09 factor of improvement for 16 nodes using the LANai 4.3 NICs and a 2.22 factor of improvement for 8 nodes using the LANai 7.2 NICs.

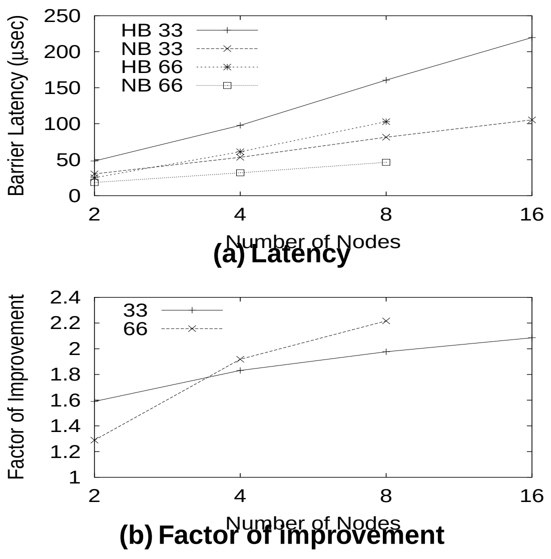
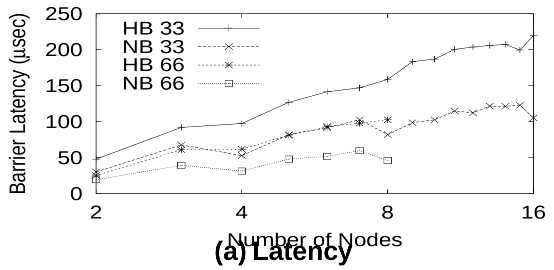


Figure 4. Performance of NIC-based barrier versus host-based barrier using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs

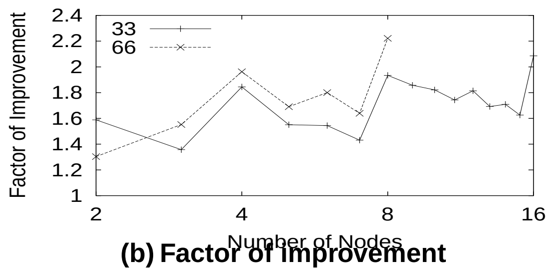
Notice also that for both NICs the factor of improvement increases with system size. This indicates that the NIC-based barrier scales better than the host-based barrier. Figure 5(a) shows the barrier latency for all (including non-power of two) nodes and Figure 5(b) the factor of improvement of the NIC-based barrier versus the host-based barrier for these nodes. From these graphs we see that even with non-power-of-two numbers of nodes, the NIC-based barrier scales better than the host-based barrier. Notice that, in some cases, the latency of performing a barrier with a non-power-of-two number of nodes is greater than the latency of performing a barrier with a greater power-of-two number of nodes (e.g., 7 nodes v.s. 8 nodes for NIC-based using the LANai 4.3). This is because for a barrier with a non-power-of-two number of nodes, two extra steps must be taken to send and receive from the nodes in set $S'$, as described in Section 2.2.

## 4.3 Granularity of computation

To examine the effects of NIC-based barrier on granularity of computation, we performed 10,000 loops of computation at the MPI-level followed by an MPI-level barrier. We varied the length of the computation to simulate different levels of granularity. In Figure 6, we varied the length of computation from 1.50$\mu$s to 129.75$\mu$s to examine the effects of NIC-based barrier for very fine levels of granularity. Figure 6 shows the average execution time (computation time and barrier time) per loop as the computation time varies. Results are presented for both NIC-based (NB) and host-based (HB) barriers on eight nodes using 33MHz LANai 4.3 (33) and 66MHz LANai 7.2 (66) NICs. Notice that for the host-based barriers, we see a flat spot where the execution time does not increase much for computation time per loop going up to around 17$\mu$s for the LANai 4.3 NICs and around 8$\mu$s for the LANai 7.2 NICs. This is due to to fact that when the host sends the last message of a barrier and completes the barrier, the NIC may still be transferring the message from the host to the transmit buffer or transmitting the message when the next barrier call is made. The



**(a) Latency**



**(b) Factor of Improvement**

**Figure 5. Performance of NIC-based barrier versus host-based barrier using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs for all number of nodes**

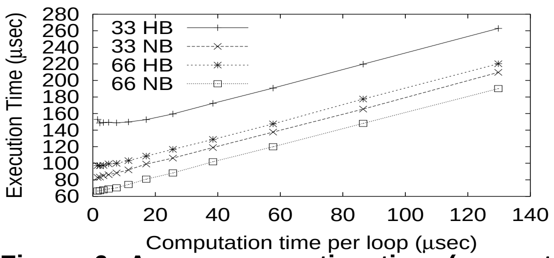


**Figure 6. Average execution time (compute time and barrier time) per loop for host- and NIC-based barrier on eight nodes using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs**

next barrier will send a message which must be delayed until the NIC has finished the previous message. We don't see the same effect with the NIC-based barrier because the NIC does not notify the host that the barrier has completed until just before it starts transmitting the last message. By the time the notification reaches the host and the host initiates the next barrier, the message will have been transmitted.

To compare the granularity of computation possible using NIC-based barriers versus using host-based barriers, we plotted graphs which show the minimum computation time required between barriers for a program to have a certain efficiency factor. We assume that the program performs computation followed by a barrier, and performs no other communication. We define our efficiency factor as the ratio of computation time to the total execution time (i.e., computation time and barrier time). Figures 7(a) through 7(d) show the computation time required to achieve efficiency factors of 0.25, 0.50, 0.75 and 0.90 for both the LANai 4.3 and the LANai 7.2 NICs.

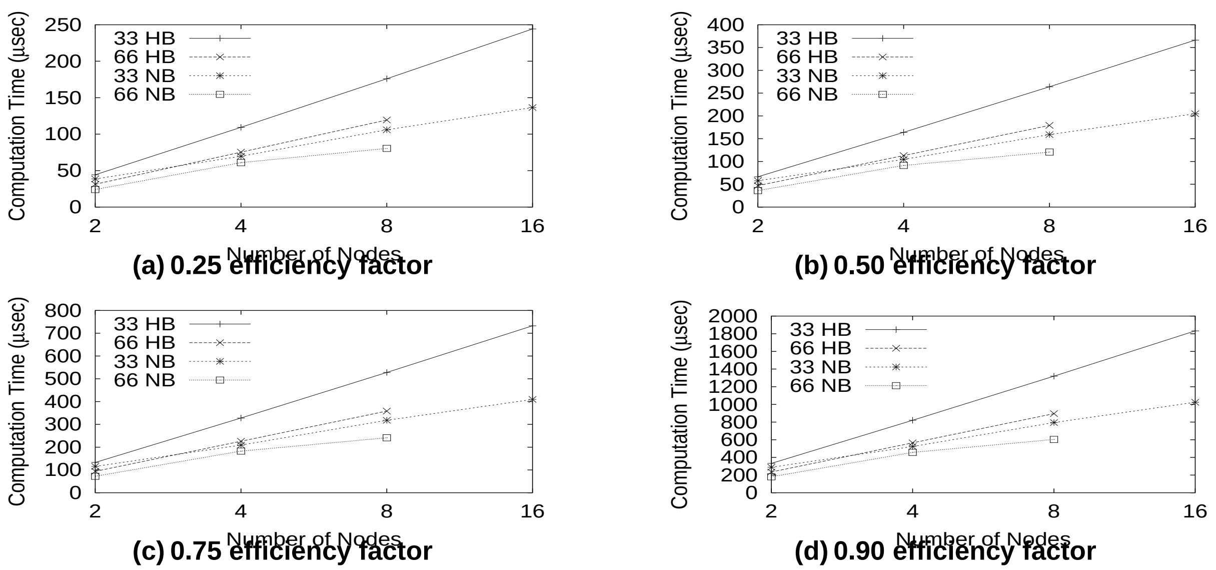
We see from these figures that the minimum computation time for a particular efficiency factor when using NIC-based barriers is less than that when using host-based barriers. For instance, Figure 7(b) shows the graph for a 0.50 efficiency factor. The minimum computation time for 16 nodes is 366.40$\mu$s for the host-based barrier and 204.76$\mu$s for the NIC-based barrier using the LANai 4.3 NICs. For the LANai 7.2 NICs over eight nodes, the host-based barrier requires 179.18$\mu$s of computation between barriers to maintain the efficiency factor, while the NIC-based barrier needs only 120.62$\mu$s of computation. For a 0.90 factor of efficiency, Figure 7(d) shows that using the LANai 4.3, the host-based barrier requires 1,831.98$\mu$s, as compared to 1,023.82$\mu$s for the NIC-based barrier to maintain the efficiency factor. Using the LANai 7.2 NIC, the host-based barrier needs 895.91$\mu$s of computation time while the NIC-based barrier needs only 603.11$\mu$s for 0.90 factor of efficiency. From these results we can see that finer granularity programs can be written using NIC-based barrier without losing efficiency.

## 4.4 Varying arrival times

In real applications, the nodes participating in a barrier do not always reach the barrier at the same time. Often some nodes reach the barrier before others. To examine the effects of varying arrival times on barrier performance, we performed 10,000 loops of computation followed by a barrier. The length of computation at each node was varied by a percentage of the mean in both directions from the mean (e.g., 4096$\mu$s $\pm$20%). Figure 8 shows the execution time of this benchmark for 16 nodes with the computation time varying from 64$\mu$s to 4096 $\mu$s with a 20% variation in the computation time using LANai 4.3 NICs. Notice that the difference in the execution time of the benchmarks using NIC-based barriers and host-based barriers gets smaller as the computation time gets larger. This is because the total variation in the arrival times gets larger. Figure 9 shows the difference between the benchmarks using host-based and NIC-based barriers over 16 nodes using LANai 4.3 NICs. Notice that the difference gets smaller as the computation time and percent variation increases, i.e., as the total variation in arrival time increases. Notice also, that for 0% variation the difference does not decrease. This indicates that the amount of computational delay itself does not affect the difference in execution time, but rather it is the total amount of variation that affects the difference. So, as the variation in arrival times increases, the execution time of the benchmark using NIC-based barriers increases slightly faster than that for the benchmark using host-based barriers. This indicates that the host-based barrier is not as sensitive to a variation in arrival time as the NIC-based barrier. Even though the NIC-based barrier is more sensitive to the variation in arrival time than the host-based barrier, it always performs better than the host-based barrier.

## 4.5 Synthetic application performance

In order to evaluate how the barrier performance would affect an application, we ran three synthetic MPI-level applications. The synthetic applications consist of several steps each of which consists of computation followed by a barrier. The mean computation time varies from one step to the next. Within each step, the computation time varies randomly from one node to the next by $\pm$10% from the mean. The execution time of each synthetic application



(a) 0.25 efficiency factor

(b) 0.50 efficiency factor

(c) 0.75 efficiency factor

(d) 0.90 efficiency factor

**Figure 7. Computation time required to achieve a particular efficiency factor using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs**

was taken over 10,000 runs. The first application was designed such that it performed eight steps and had computation times of 10, 20, 30, . . ., 80$\mu$s, for the respective steps, for a total of 360$\mu$s of computation. This application can be seen as communication intensive. The second application had 20 steps and had computation times of 10, 20, 30, . . ., 200$\mu$s, for a total of 2,100$\mu$s of computation. The third application had 10 steps and had computation times of 100, 500, 1,000, 2,000, 3,000, 500, 500, 250, 600, 1,000$\mu$s for a total of 9450$\mu$s of computation. This application can be seen as computation intensive. Figure 10(a) shows these results for host-based (HB) and NIC-based (NB) barriers using 33MHz LANai 4.3 (33) and 66MHz LANai 7.2 (66) NICs. Notice that in all cases the NIC-based barrier performs better than the host-based barrier. Figure 10(b) shows the factor of improvement for the applications using NIC-based barriers versus the applications using host-based barriers. For all applications, we see that the factor of improvement is increasing. Figure 10(c) shows the efficiency factor for the applications. Notice that for each application the NIC-based barrier has a higher efficiency factor than the host-based barrier. These results indicate that a NIC-level barrier implementation on large clusters using NICs with faster processors can deliver very good performance benefits at the application level.

## 5 Conclusions

This paper evaluated the performance of the NIC-based barrier by comparing it to the traditional host-based barrier. In [4] we implemented the NIC-based barrier in GM and showed that it performed better than the host-based barrier. In this paper we modified MPICH version 1.2..3 which runs over GM, to use the NIC-based barrier. This was done in an efficient manner which added only 3.22$\mu$s overhead to the GM implementation of the NIC-based barrier over 16 nodes using the 33MHz LANai 4.3 NICs. When comparing the performance of the barriers at the MPI level, we found a 2.09 factor of improvement for 16 nodes using the LANai 4.3 NICs and a 2.22 factor of improvement for 8 nodes using the 66 MHz LANai 7.2 NICs. Furthermore, the factor of improvement increased with the number of participating nodes. This indicates that the NIC-based barrier scales better than the host-based barrier.

We also evaluated the impact of the NIC-based barrier on the granularity of computation. We found that for a program to have a 0.90 factor of efficiency using the LANai 4.3 NICs, at least 1831.98$\mu$s of computation must be performed per barrier if the host-based barriers are used, but only 1023.82$\mu$s if a NIC-based barrier is used. This value is 44% lower than for the host-based barrier. So, using the NIC-based barrier allows for finer grained programs without lowering the efficiency. We noticed that the NIC-based barrier is more sensitive to variation in arrival times than the host-based barrier. However, the NIC-based barrier always performed better than the host-based barrier. To evaluate the impact of using the NIC-based barrier on applications, we used synthetic applications. This indicates that using the NIC-based barrier in applications which perform many barrier calls will deliver significant performance benefits.

We are currently evaluating the impact of using NIC-based barriers in different programming models and applications, such as, Global Arrays[13], Bulk Synchronous Programming[7], and Buffered Coscheduling[15]. We also intend to evaluate the benefits of NIC-based barriers for larger system sizes using modeling and experimental evaluation. More generally, we intend to study whether other collective communication operations (such as reduction and all-to-all) could benefit from a NIC-based implementation.

**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). If you are interested in us-

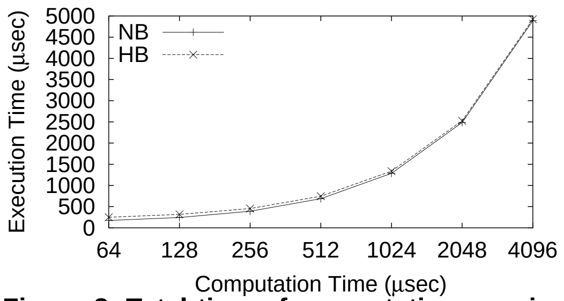


**Figure 8. Total time of computation, varying at each node by 20%, followed by a barrier for NIC-based and host-based barriers over 16 nodes using 33MHz LANai 4.3 NICs.**

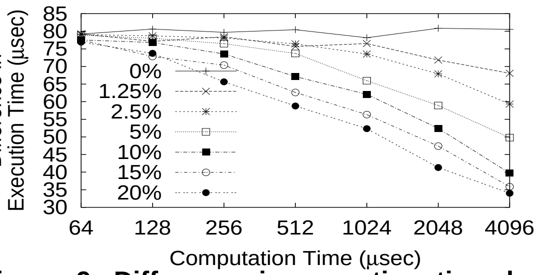
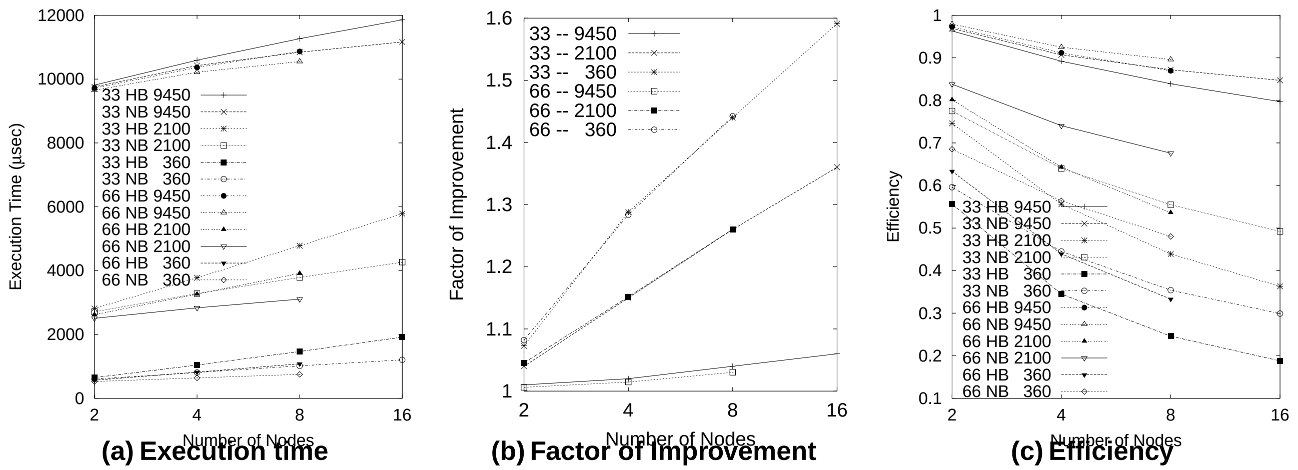


**Figure 9. Difference in execution time between using host- and NIC-based barriers performing computation (± percentage) followed by a barrier (16 nodes; 33MHz LANai 4.3 NICs).**



**(a) Execution time** **(b) Factor of Improvement** **(c) Efficiency**

**Figure 10. Performance of synthetic benchmarks (total computation time of 360$\mu$s, 2,100$\mu$s and 9450$\mu$s) for host-based and NIC-based barriers using 33MHz LANai 4.3 and 66MHz LANai 7.2 NICs**

ing this software, please contact Dr. D. K. Panda at panda@cis.ohio-state.edu.

## References

[1] D. H. Bailey, E. Barszcz, L. Dagum, and H. Simon. NAS Parallel Benchmark Results. Technical Report 94-006, RNR, 1994.

[2] M. Barnett, S. Gupta, D. G. Payne, L. Shuler, R. van de Geijn, and J. Watts. Interprocessor Collective Communication Library (Intercom). In *Scalable High Performance Computing Conf.*, pages 357–364, 1994.

[3] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. Seizovic, and W. Su. Myrinet - a gigabit per second local area network. In *IEEE Micro*, February 1995.

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

[5] D. Buntinas, D. K. Panda, and P. Sadayappan. Fast NIC-based barrier over Myrinet/GM. Technical Report OSU-CISRC-10/00-TR22, The Ohio State University, 2000.

[6] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. The IEEE Computer Society Press, 1997.

[7] M. Goudreau, K. Lang, S. Rao, T. Suel, and T. Tsantilas. Towards efficiency and portability: programming with the bsp model. In *Proceedings of the 8th annual ACM symposium on Parallel algorithms and architectures*, June 1996.

[8] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, Sept. 1996.

[9] R. Gupta. The Fuzzy Barrier: A Mechanism for the High Speed Synchronization of Processors. In *Proceedings of the Int'l Conf. on Architectural Support for Programming Languages and Operating Systems*, pages 54–63, 1989.

[10] P. Keleher, A. L. Cox, S. Dwarkadas, and W. Zwaenepoel. TreadMarks: Distributed Shared Memory on Standard Workstations and Operating Systems. In *Proceedings of the 1994 Winter Usenix Conf.*, Jan. 1994.

[11] P. K. McKinley and D. F. Robinson. Collective Communication in Wormhole-Routed Massively Parallel Computers. *IEEE Computer*, pages 39–50, Dec 1995.

[12] Myricom. Myricom GM myrinet software and documentation. http://www.myri.com/scs/GM/doc/gm_toc.html, 2000.

[13] J. Nieplocha, R. J. Harrison, and R. L. Littlefield. Global arrays: A portable "shared memory" programming model for distributed memory computers. In *Supercomputing 94*, 1994.

[14] D. K. Panda. Issues in Designing Efficient and Practical Algorithms for Collective Communication in Wormhole-Routed Systems. In *ICPP Workshop on Challenges for Parallel Processing*, pages 8–15, 1995.

[15] F. Petrini and W. Feng. Buffered coscheduling: A new methodology for multitasking parallel jobs on distributed systems. In *Proceedings of the Int'l Parallel and Distributed Processing Symposium 2000, IPDPS2000*, May 2000.