Fast Collective Communication Algorithms for Reflective Memory Network Clusters[1]

Vijay Moorthy Dhabaleswar K. Panda P. Sadayappan

Department of Computer and Information Science
The Ohio State University, Columbus, OH 43210
Phone: (614) 292-0053, Fax: (614) 292-2911
Email: {moorthy|panda|saday}@cis.ohio-state.edu

Contact Author: Dhabaleswar K. Panda

Abstract: *In this paper, we present algorithms for efficiently implementing collective operations on reflective memory network clusters. First, we describe attributes of reflective memory networks. We then introduce algorithms for barrier synchronization, broadcast and all-reduce. These algorithms have been implemented as an extension to the Bill-Board Protocol, a low-latency communication protocol for reflective memory networks that we previously developed [6,7]. The performance of the algorithms is evaluated in detail. We present measured performance for small network sizes as well as simulated performance for larger networks with higher bandwidths representative of the current systems. Performance comparison with the Myrinet Network indicates that reflective memory networks can deliver better communication performance than point-to-point networks.*

Keywords: Collective Communication, Reflective Memory, Cluster Computing, System Area Networks

1. Introduction

In recent years, clusters of workstations have become a popular platform for high-performance computing [1,2]. Clusters typically use commodity networks and their performance depends on the performance of the interconnecting network. Point-to-Point networks such as Myrinet [3], switched Ethernet, and ATM have become popular interconnects for clusters. Though these networks provide high bandwidth, delivering low latencies and providing good communication and computation overlap is a non-trivial problem. Consequently, a lot of research has been done on finding solutions for these problems [4,5,16,17].

Recently, there has been interest in the research community in applying other types of networks to cluster computing, such as reflective memory networks [8]. Reflective memory networks provide a limited amount of shared memory across a cluster of workstations. Each workstation is equipped with a network interface card (NIC) which possesses a memory bank. Memory on the NICs is not physically shared but updates to a location on any one of the NICs are transmitted to the other NICs on the network. This causes the memory at each NIC in each workstation to be a reflection of the memory on the other NICs.

Reflective memory networks have some unique features. The most important feature is that communication is abstracted to memory reads and writes. No operating system intervention is required for communication. Once a message is written to the NIC, the host processor can resume computation. In addition, latencies are predictable. This is especially important for some applications, such as real-time applications. Finally, the inherent broadcast nature of reflective memory networks makes them ideal for implementing collective operations, which are useful in many parallel and distributed applications.

In this paper, we develop algorithms for efficiently implementing three collective communication operations on reflective memory networks: Broadcast, Barrier Synchronization, and All-Reduce. These algorithms have been implemented as extensions to the Bill-Board protocol (BBP) [6,7], a low latency

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

communication protocol for reflective memory networks that we have previously developed. We have developed an analytical model of reflective memory networks, which is used to evaluate these algorithms. We present experimental results for the BBP implementation on a SCRAMNet [11] cluster. These results are compared with the performance of the same collective communication operations on point-to-point networks, such as Myrinet.

The paper is organized as follows: The next section gives a broad overview of reflective memory networks. Section 3 describes the algorithms. Section 4 describes the analytical modeling of the algorithms for reflective memory networks. Section 5 presents performance evaluation results. Section 6 presents our conclusions.

2. An Overview of Reflective Memory Systems

A reflective memory (RM) network provides a limited amount of shared memory across a cluster. There are many reflective memory networks available today. SCRAMNet [11] from Systran, RTNet [12] from VMIC and DEC MemoryChannel [13] are popular reflective memory products. VMMC [14] from Princeton and Merlin and Sesame [15] from Sandia Labs and SUNY, Stony Brook are research projects involving reflective memory. A detailed survey of existing RM networks is presented in [8].

The most common application of RM networks has been for Real time Systems. They have been used in aircraft simulators, telemetry, and robotics. They have also been applied to distributed applications such as on-line transaction processing (OLTP) and client-server computing, as well as parallel computing applications.

The basic operation of any reflective memory network is as follows. Each node on the network has a memory bank (normally residing on the NIC). The nodes are interconnected by some network. The memory bank is mapped to the address space of the processes using the reflective memory. When a process writes to a location in the area of memory that maps to the reflective memory bank, the RM system propagates that write to every other node in the network. The propagation of the update happens asynchronously, and without intervention of the application process. When a process reads from a location in the shared area of memory, no communication is required to the other nodes, because any updates to that location by other nodes would have been automatically sent when writes occur.

Different RM networks provide similar functionality, yet their characteristics may differ. In order to evaluate the algorithms presented in this paper, it is not sufficient to test it on a single RM network alone. In this paper, we present performance over one particular network, i.e. SCRAMNet. In addition, we develop an analytical model of RM networks, which is representative of current day technology. The algorithms have been implemented and tested on SCRAMNet. Their performance has also been evaluated for the modeled RM network. Using this approach allows us to predict the performance of these algorithms on other RM networks.

3. Collective Communication Algorithms for Reflective Memory Networks

Algorithms for three collective communication operations are described here: Broadcast, Barrier Synchronization, and All-Reduce. These operations were chosen because a broadcast of information occurs in each one of these, and therefore they are good candidates for using the inherent broadcast nature of reflective memory networks.

3.1 Broadcast

In a broadcast, one process sends the same message to all other nodes. Many algorithms have been proposed for accomplishing a broadcast. However, efficient solutions making use of the broadcast nature of the underlying network do not exist.

A single-step approach is one that is used in the original BBP implementation [6]. The sender writes the message only once, to a buffer in the reflective memory. Then the sender writes a message notification containing the location of the message at a unique location for each recipient. The receivers poll for the arrival of the message notification, and on its arrival, read the message from the shared memory. The advantage of this approach is that a message can be sent to more than one recipient and the overhead for multiple receivers is independent of the message length. However, as the number of recipients increases, the overhead for writing message notifications increases linearly.

$O(log(N))$ solutions exist for broadcast on point-to-point networks. The Binomial tree algorithm is an $O(log(N))$ broadcast algorithm. This algorithm requires $log_2(N)$ steps for N participating nodes. In each step, each node that has already received the message forwards it to a node that has not received it yet. Initially, only the sender has the message. The sender sends it to one other node starting the broadcast. In the next step, two nodes have the message and they send it to two more. In this manner, at each step the number of nodes which have received the message doubles. The binomial tree algorithm has been shown to be optimal for point-to-point networks of homogenous nodes. However, the entire message is transmitted multiple times in this algorithm. Therefore, it is not the best suited for RM networks.

We propose a new algorithm that combines the advantages of both the single-step algorithm as well as the binomial tree algorithm. In this algorithm, message notifications are distributed hierarchically. The sender writes the message to a location in shared memory. Then it writes message notifications to the first r recipients. Each of these r recipients writes message notifications for r more recipients. In this way, a tree of depth $log_r(N)$ is built for distributing the message notifications, with the sender at the root of the tree. In $log_r(N)$ steps, all the recipients receive message notifications.

This algorithm allows better concurrency in the distribution of message notifications than the single-step algorithm. Since the total number of notifications remains the same, there is no increase in network usage. As in the naive scheme, for a fixed system size, the time taken to complete the broadcast increases linearly with message size. In addition, the overhead for writing message notifications increases logarithmically with system size. Therefore, for a fixed message length, the time taken to complete the broadcast increases $O(Log(N))$ with the number of nodes. It is to be noted that the time taken not only depends on N, the number of nodes, but also r, the ordinality of the tree used for distributing the notifications. For a specific system, this parameter has to be tuned, in order to determine the value of r that will give the best performance. For example, in case a binomial tree is used, $r = 2$. We study the impact of r on broadcast time in section 5.

3.2 Barrier Synchronization

In Barrier Synchronization, N processes synchronize at a point called the barrier. When a process crosses that point, it is guaranteed that every other process has reached that point. It is a special case of phase synchronization as defined by Misra in [10].

The original implementation of BBP [6,7] uses a naive algorithm. In this implementation, there is one designated process, called the root. Each process sends a message to the root on arriving at the barrier. The root, on reaching the barrier, waits for messages from every other node, and then broadcasts a message to all nodes notifying them that the barrier has been completed. This algorithm does not make any use of the broadcast nature of the network. In addition, the time required to complete the barrier grows linearly with the number of participating nodes.

An algorithm, which makes use of the broadcast nature of the network, is given below. Each process sets a unique flag on the shared memory and then polls on the entire set of flags to see if all of them are set. For small system sizes, this method is extremely fast. However, as the system size increases, the number of flags to be polled increases linearly.

We can further improve on the above algorithm. Given below is a hierarchical algorithm for barrier synchronization on RM networks. The N nodes are split into groups of r each. Each group synchronizes itself by the above method. Then one designated node from each of these groups executes a barrier recursively, until there are *r* or fewer nodes left. When these processes synchronize, the barrier is complete. Instead of all nodes polling on all flags, all the nodes which are done writing flags poll on a single flag, which is set on completion of the last step with *r* or fewer nodes, by one designated node. This algorithm takes $log_r(N)$ steps to complete. At any step, the maximum number of flags to be polled is *r*. Again, as in the case of the hierarchical broadcast algorithm, the performance of the algorithm depends on the value of *r*. The impact or *r* on the barrier time is studied in section 5.

3.3 All-Reduce

In an All-Reduce operation, each process has an array of elements. The result required is an associative operation on all the arrays. Each process should have the result array at the end of the all-reduce.

Several algorithms have been suggested for the all-reduce operation on point-to-point networks. We implemented these solutions but their performance on reflective memory networks was not very good, as none of these algorithms use the broadcast capability of the network.

A commonly used algorithm is the ring algorithm. The nodes are arranged in a logical ring. In N successive steps, each node receives an array from the node logically to its left and sends an array to the node logically to its right. In the first step, each node sends its own array to the node on its right. In subsequent steps, it forwards arrays from other nodes that were received in previous steps from the node on the left. At the end of N such steps, each node has all the arrays and can independently compute the result. The ring algorithm is essentially an all-to-all broadcast, followed by the computation. However, it is not suited for RM networks because the same array is transmitted multiple times.

Presented below is a better solution for RM networks. Each node simultaneously broadcasts its array to all others; waits for the arrays from the others and then independently computes the result. The broadcast itself is done using the hierarchical algorithm described in section 3.1. In addition, to reduce contention on the network during distribution of broadcast message notifications, each node sends them in a different order. A node takes its destination set, (which is all the nodes except itself) and starts from the node logically next to it in rank order, that is, node *i* will first send to node $(i+1) \bmod N$. This also means that it will receive notifications in reverse rank order starting from $(i-1) \bmod N$.

4. Modeling Collective Communication Algorithms for Reflective Memory Networks

The algorithms presented in this paper were implemented and tested on a 4-node SCRAMNet cluster of Pentium PCs. We also developed an analytical model of a reflective memory network in which we could choose parameters such as network bandwidth, I/O bus read and write times which are representative of current RM networks and systems. The model has been evaluated for correctness, by comparing timings measured for the algorithms presented in this paper on the test bed with timing values generated using the model. Section 5 presents a comparison of measured and modeled timings. It can be observed that modeled timings track our measurements closely.

Since considerable variation exists in RM networks available today, we have attempted to pick some common features of RM networks for use in the model. We have chosen the register insertion ring topology used in SCRAMNet and RTNet, since it is common. We have modeled a non-coherent RM network. The granularity of sharing was chosen to be one word. This modeled network will be referred to as the “RM model network”.

A register insertion ring [9] is a more sophisticated form of a slotted ring. In a register insertion ring, NICs have two shift registers: a ring shift register and an input shift register. Traffic from the ring enters the ring shift register. Updates from the host enter the input shift register. The NIC switches between the two shift

registers to transmit data on to the ring. If there is data in the ring shift register, it is always transmitted before any data in the input register. Data in the input register is queued in a FIFO, so that the host can make multiple writes even if there is traffic on the network. The host blocks on writes to shared memory if the FIFO is full.
It is important to model performance of the I/O Bus as well as the underlying network because access times to the shared memory depend on the performance of the I/O bus. In order to model the I/O bus, we measured the performance of the PCI bus on a 300 MHz Pentium II PC with the Intel 440 FX PCI chipset. Access times for reads and writes tend to be asymmetric on the PCI bus, so we measured read times as well as write times. We have chosen to model the PCI bus as it is widely used.

The following are the major parameters in the model that can be tuned according to the characteristics of the network being modeled: ring slot time, I/O bus read latency, and I/O bus write latency. The ring slot time is the time that elapses between two successive transmission slots on the ring. It depends on the network bandwidth. The read and write latencies are the times required for a minimum size read or a write.

The slot time for SCRAMNet was measured to be 571 ns in our cluster, corresponding to a network bandwidth of 7 MB/s. We use a slot time of 58 ns for the RM model network, which corresponds to a network bandwidth of 70 MB/s. This bandwidth was chosen because it is representative of current RM networks. VMIC RTNet, has a bandwidth of 69.5 MB/s. PCI read and write latencies measured for the Intel 440 FX chipset with the SCRAMNet NIC were used in the model. These were measured to be 950 ns and 330 ns respectively.

The time taken for any communication operation can be split into message-length independent processing overhead time and message length dependent message copying and transmission time. From the implementation of that operation, it is possible to determine how these individual components overlap. A message can be viewed as going through a number of pipelined stages such as copy from sender host memory to sender NIC, transmission from sender NIC to receiver NIC, and transmission from receiver NIC to receiver host memory. The time taken for a message to go through the entire pipeline, is the sum of the latencies of the individual pipeline stages. In case parts of the message are pipelined, for instance, as packets or words, then the time taken for the entire operation is given by the sum of the time spent in the longest pipeline stage and the latency of the other pipeline stages.

For each algorithm presented in this paper, we have determined the time spent in individual stages and derived a mathematical equation to estimate time for each operation. These equations are given below:

The time taken for a broadcast using the hierarchical broadcast algorithm is given by the following equation:

$$\begin{aligned}\text{Broadcast Time} = {} & \mathrm{FLOOR}(\log_r(N-1)) * (\textit{Message Notification Read Time} + (r-1) * \textit{Notification write time}) \\ & + \left(\mathrm{FLOOR}\left(\frac{N-1}{2^{\mathrm{FLOOR}(\log_r(N-1))}}\right) - 1\right) * (\textit{Notification write time}) \\ & + (\textit{Total number of words written} + (r-1) * \textit{slot time}) \\ & + (\textit{PCI write latency}) \\ & + (\textit{PCI read time for message}) \\ & + (\textit{Fixed message processing overhead}) \qquad \text{..........}\mathbf{(4.1)}\end{aligned}$$

The time taken for a Barrier using the hierarchical barrier algorithm is given by the following equation:

$$\begin{aligned}\text{Barrier Time} = {} & \mathrm{CEILING}(\log_r(N)) * ((r-1) * (\textit{flag read time}) \\ & + (\textit{PCI Write time for one word}) \\ & + 3 * (N-1) * \textit{slot time}) \quad . \qquad \text{.........}\mathbf{(4.2)}\end{aligned}$$

The time for an all-reduce operation using the simultaneous broadcast algorithm is given by the following equation:

All-Reduce time = 2 * (*PCI write latency* + (*total no. of words* –1) * *N* * *slot time*)
+ (*N-1*) * (*PCI read time for message*)
+ (*Fixed message processing overhead*) ....(**4.3**)

5. Performance Evaluation

The three algorithms described in the preceding section have been implemented and incorporated into the BBP. Their performance was evaluated on a 4-node SCRAMNet Cluster of 300 MHz Pentium II workstations and using the analytical model discussed above.

For comparison, we also measured the time taken for equivalent operations on MPIFM on Myrinet on the same cluster. Myrinet is a 1Gb/s point-to-point network which is a popular interconnect used for clustered systems. MPIFM is the most popular communications software that provides collective communication support on Myrinet. Although FM [4], the communication layer on top of which MPIFM is built is at the same layer as BBP, FM does not provide any collective communication support. In addition, MPIFM adds only a little overhead to FM. Therefore, we have chosen MPIFM for our comparisons.

For each operation, the following results are presented. First, for a 4-node system, measured results are presented for SCRAMNet and MPIFM/Myrinet. These are labeled "SCRAMNet (Measured)" and "MPIFM (Measured)" respectively in the figures. Estimated times for SCRAMNet using the model calculated using equations (4.1) – (4.3) and parameters for SCRAMNet are also presented. These are labeled "SCRAMNet (Modeled)". The purpose of presenting both numbers is to check for validity of the model. Simulated results for the RM model network are also presented. These were calculated using equations (4.1)-(4.3) and parameters for the RM model as described in section 4. These results are labeled "RM (Modeled)". Simulated results for each operation are also presented for bigger system sizes. MPIFM uses well-known algorithms for the three operations under study. These have been implemented on top of point-to-point messages. We measured times taken for point-to-point messages and used them to compute the time taken for collective operations for larger system sizes. These results are labeled "MPIFM (Modeled)" in the figures.

Figures 1,2 and 3 show the performance of broadcast. Fig. 1 compares measured values of broadcast timings for MPIFM and SCRAMNet, and the RM model for a 4-node system. Since Myrinet has the highest bandwidth, it has the lowest slope, but it has higher latency. Hence, for short messages SCRAMNet and the RM model network do better. Figs. 2 and 3 show the performance of broadcast for zero byte and 64 bytes for increasing system size. It can be observed that the performance depends on the value of r. A value of $r=4$ is the best for a 32 node system. However, this may not be the case for larger systems. Figs. 2 and 3 show that in general, as the system size increases so does the ideal value of r.

Figures 4 and 5 show the performance of the Barrier algorithm. As can be seen in Fig. 4, the modeled performance closely tracks the actual measured performance on SCRAMNet. The curve marked MPIFM shows the measured performance of the MPIFM Barrier implementation on Myrinet. A 4-node barrier takes 32 microseconds on MPIFM and only 13.9 microseconds on BBP on SCRAMNet. The hierarchical barrier algorithm on the RM model network takes only 10.7 microseconds.

Fig 5 shows the performance of the barrier algorithm for larger systems, up to 32 nodes and compares the performance of our algorithm on a reflective memory network with that of the $O(Log(N))$ scheme used in MPIFM. Note that the performance depends on the value of r used. For systems up to 32 nodes, the algorithm works best with $r=4$. A trend similar to broadcast is observed here. In general, as sytem size increases so does the ideal value of r to be used in this algorithm

Figures 6,7 and 8 show the performance of the All-Reduce algorithm. Fig. 6 shows the performance on a 4-node system. For short message sizes, SCRAMNet and the RM model network do better than Myrinet, but Myrinet does better for longer messages. Figs. 7 and 8 show the performance of All-Reduce for 4 byte and 64 byte lengths for systems larger than 4 nodes. The same pattern is observed. For the shorter message size, SCRAMNet and the RM model network do better.

6. Conclusions

In this paper, we have presented algorithms for Barrier Synchronization, Broadcast, and All-Reduce on reflective memory networks. These algorithms have been implemented and their performance compared with Myrinet, a high bandwidth network commonly used in cluster computing today. The results show that efficient collective communication support is indeed possible using reflective memory networks. Although, current generation reflective memory networks do not have bandwidths as high as current point-to-point networks, they have their advantages over other networks, such as bounded propagation delay and better overlap of computation and communication. It can be seen that even with a bandwidth disadvantage, they are able to provide better performance in many cases for three collective communications operations described in this paper, namely, Barrier Synchronization, Broadcast and All-Reduce. Thus, reflective memory networks can be used for implementing collective communication operations efficiently on clusters.

References

[1] T. Anderson, D. Culler, and D. Patterson. A Case for Networks of Workstations (NOW). IEEE Micro, pages 54-64, Feb 1995.
[2] D. J. Becker et al. Beowulf: A Parallel Workstation for Scientific Computation. In International Conference on Parallel Processing, 1995.
[3] N. J. Boden, D. Cohen, et al. Myrinet: A Gigabit-per-Second Local Area Network. IEEE Micro, pages 29-35,Feb 1995.
[4] S. Pakin, M. Lauria, and A. Chien. High Performance Messaging on Workstations: Illinois Fast Messages (FM). In Proceedings of Supercomputing'95.
[5] T. Von Eicken, A. Basu, V. Buch, and W. Vogels. U-Net: A User-level Network Interface for Parallel and Distributed Computing. In ACM Symposium on Operating Systems and Principles, 1995
[6] M. G. Jacunski, V. Moorthy, P. P. Ware, M. T. Pillai, D. K. Panda, and P. Sadayappan. Low Latency Message-Passing for Reflective Memory Networks. In Workshop on Communication, Architecture and Applications for Network-Based Parallel Computing (CANPC), 1999.
[7] V. Moorthy, M.G Jacunski, M. T. Pillai, P. P. Ware, D. K. Panda, T. W. Page, Jr., P. Sadayappan, V. Nagarajan, J. Daniel. Low Latency Message-Passing on Workstation Clusters using SCRAMNet. In Proceedings of International Parallel Processing Symposium (IPPS), 1999.
[8] M. Jovanovic, and Velijko Milutnovic. An Overview of Reflective Memory Systems. IEEE Concurrency, Vol 7:2,pages 56-64, 1999
[9] M. T. Liu. Distributed Loop Computer Networks. In Advances in Computers, Academic Press, 1978
[10] J. Misra. Phase Synchronization. Information Processing Letters 38, pages 101-105, 1991 .
[11] T. Bohman. Shared Memory Computing Architectures for Real-Time Simulation – Simplicity and Elegance, Technical Report, Systran Corporation, 1994.
[12] VMIC's Reflective Memory Network, VME Microsystems International Corp., 1995
[13] R. Gillet. Memory Channel Network for PCI, IEEE Micro, Vol. 16, No.1, Feb 1996, pages 12-18.
[14] M. A. Blumrich, C. Dubnicki, E. W. Felten, and K. Li. Protected User-level DMA for the SHRIMP interface. In Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-2), 1996.
[15] L. Wittie, G. Hermannsson, and A. Li. Eager Sharing for Efficient Massive Parallelism. In Proceedings of the International Conference on Parallel Processing (ICPP), 1992.
[16] J. Duato , S. Yalamanchili and L. Ni. Interconnection Networks: An Engineering Approach. The IEEE Computer Society Press, 1997
[17] 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), 1997.

Fig. 1. Broadcast Performance on a 4 node system

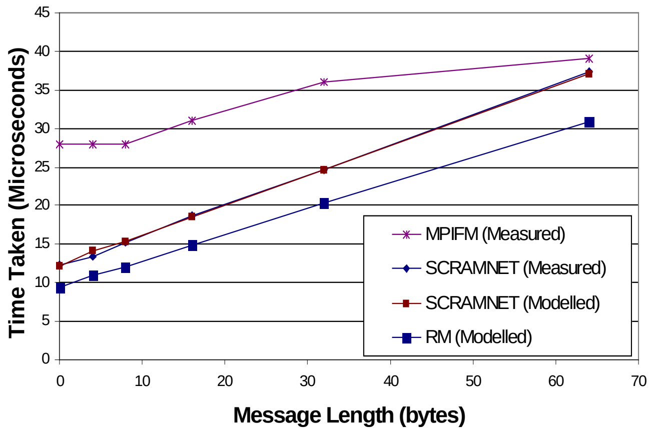

Fig. 2. Broadcast Performance (0 byte)

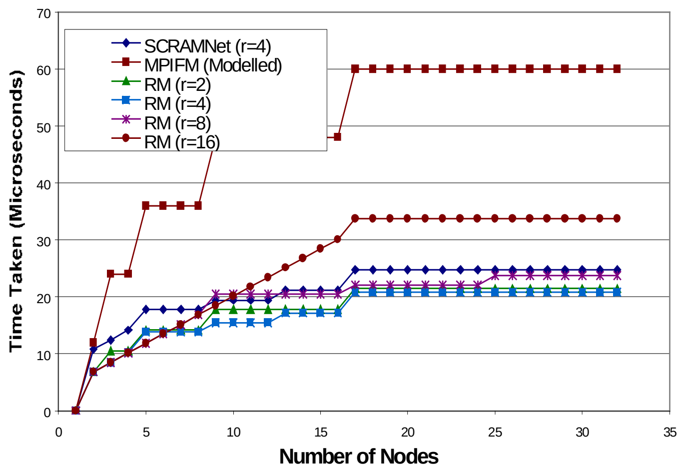

Fig. 3. Broadcast Performance (64 bytes)

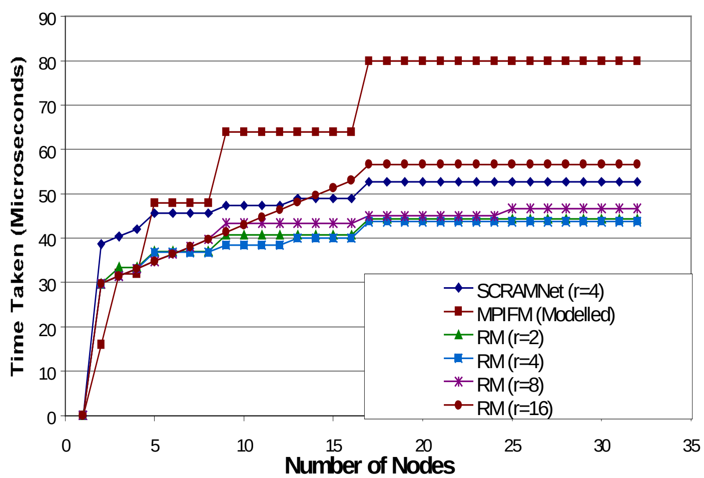

Fig. 4. Barrier Performance on a 4-node System

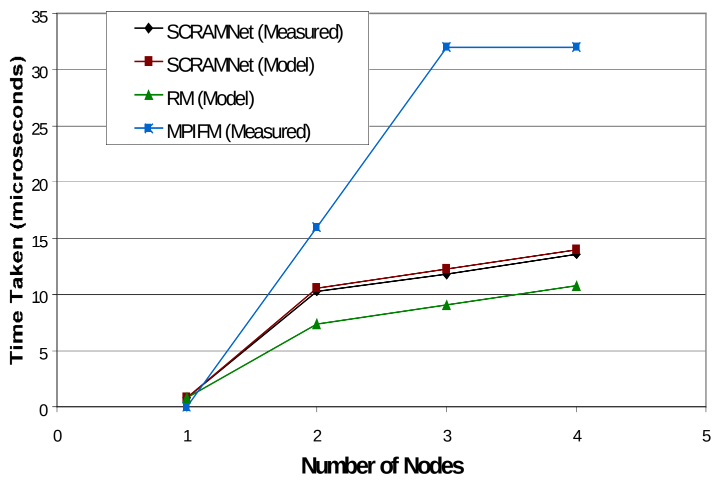

Fig. 5. Barrier Performance on up to 32 nodes

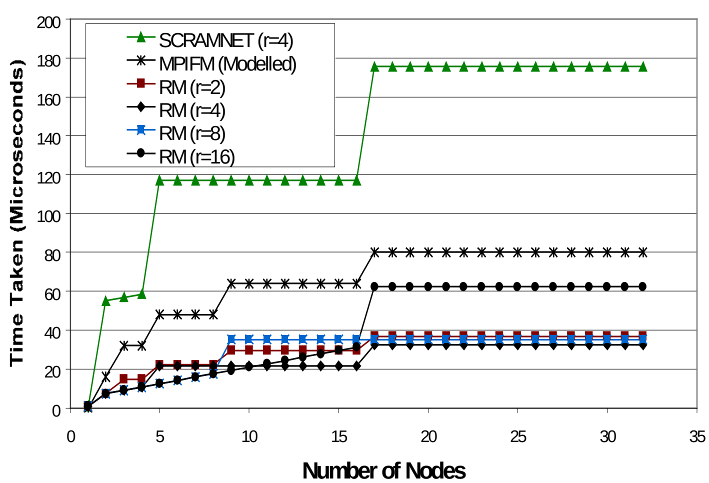

Fig. 6. All-Reduce Performance on a 4-node system

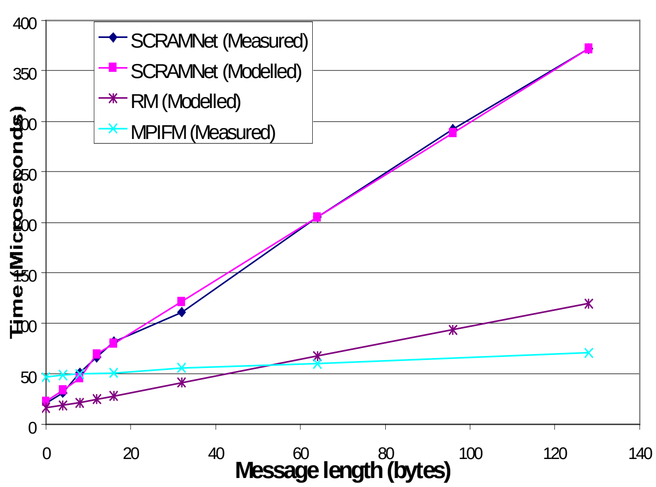

Fig. 7. All-Reduce Performance (4 bytes)

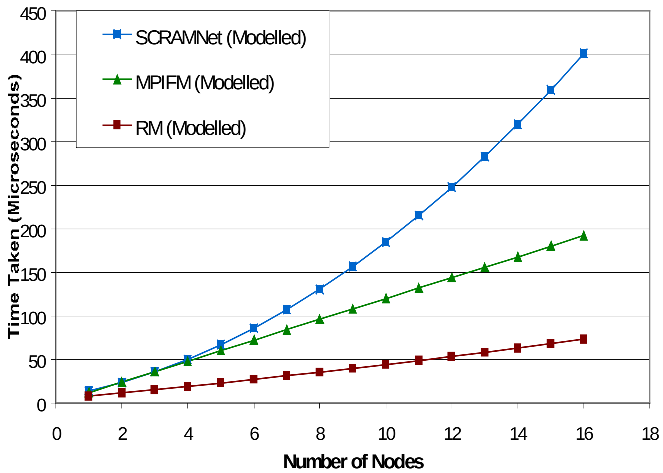

Fig. 8. All-Reduce Performance (64 bytes)