NIC-Based Reduction in Myrinet Clusters: Is It Beneficial?

Abstract

Reduction-to-one and reduction-to-all operations are common operations in parallel and distributed systems. These operations are collective operations *which can involve many processes. It is therefore important to make these operations fast and efficient. Some modern network interface cards (NICs) for system area networks (SANs) have programmable processors which can be used to offload protocol processing from the host processor. In this paper we investigate the use of the NIC processor to improve the performance of reduction operations. We implemented a NIC-based reduction-to-one operation which can perform integer and floating point operations. The NIC-based operation performs better than the traditional host-based approach with up to a 1.19 factor of improvement.*

1 Introduction

Reduction-to-one and reduction-to-all operations are common operations in parallel and distributed systems. These operations are *collective operations* which can involve many processes. It is therefore important to make these operations fast and efficient. Research has been done to make these operations efficient by taking advantage of the particular characteristics of the underlying architecture[6][10]. Some modern network interface cards (NICs) for system area networks (SANs) have programmable processors which can be used to offload protocol processing from the host processor. Such implementations not only allow efficient communication operations, but also significant potential for overlap of computation with communication. Programmable NICs have been used to improve the performance of certain operations, as well as reduce the host involvement in the operations allowing the host to perform other useful computation[11][1][3][4][5]. However, none of these implementations have explored the benefits of NIC-based implementation for reduction for integer and floating point data. In this paper we explore the benefits of NIC-based support for reduction operations. It is worthwhile to note that a large fraction of reduction operations are performed using small data sizes of just a few elements. This means that a specialized reduction operation which can efficiently perform the operation on a small number of elements would be useful.

We have implemented a NIC-based reduction-to-one operation to perform integer and floating-point operations on single 64 bit elements. In this paper we describe the design and implementation of this operation as well as the evaluation of the implementation. Our implementation achieves a 1.19 factor of improvment over the traditional host-based implementation when performing integer operations on a 16 node system. We show that NIC-based reduction would be also be beneficial for larger system sizes. Our initial implementation and evaluation indicates that reduction operations can benefit by NIC-based implementations, and that further work should be done to implement a more complete NIC-based reduction-to-one and reduction-to-all operations.

The rest of this paper is organized as follows. In the next section, we describe the general concept of a NIC-based reduction operation. In Section 3 we describe our design and implementation, followed by the evaluation of our implementation in Section 4. Finally we present our conclusions and future work in Section 5.

2 NIC-Based Reduction

Before we describe the general concept of the NIC-based reduction, we will briefly describe traditional host-based reduction. In traditional host-based reduction in a message-passing system, messages are passed between processes running at the host, and the arithmetic operations are performed by the host processors. When the operation is complete, the result of the operation will be located at one of the processes. Processes participating in the reduction operation are organized in a logical tree. Each process receives reduction messages from its children, which contain a partial result from the subtree of that child. Next, each process performs the arithmetic operation on its data and the partial results received from its children. The process then sends this result to its parent. Figure 1(a) shows a block diagram of a host-based reduction operation across four nodes. Node 0 sends its data to Node 1. When Node 1 receives this message it performs the arithmetic operation on the data from Node 0 and its data. Node 1 then sends this result to Node 3. Node 3 receives data from Node 2 and Node 1, and performs the arithmetic operation on its own data and the data send by Nodes 1 and 2.

In a NIC-based reduction, each process sends its data to the NIC. The NIC will then wait for the messages from its children, perform the arithmetic operation, and either send a message to its parent, or if this node is the root of the tree and has no parent, it will forward the result to the host. Figure 1(b) shows a block diagram of a NIC-based reduction operation across four nodes. Here we see each process sending its data to the NIC. The NICs at Nodes 0 and 2 immediately forward their data to their parents, since

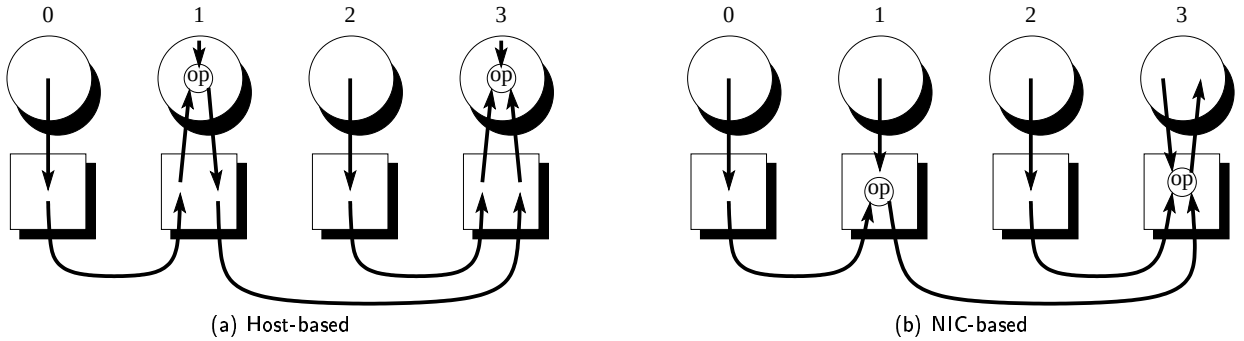

Figure 1: Block diagrams of Host-based and NIC-based reductions across four nodes. The circles represent the host processor of a node and squares represent the NIC of a node.

they are leaf nodes. The NIC at Node 1 receives this message and performs the arithmetic operation on the data in the received message and the data sent from the host. It then sends this result to the NIC at Node 3. The NIC at Node 3 receives the messages from the NICs at Nodes 1 and 2, performs the arithmetic operation on this data and on the data sent from the host. The NIC at Node 3 then forwards this result to the host.

Notice that in the host-based reduction, messages received at intermediate nodes, such as Node 2, are received by the NIC, forwarded to the host, when then performs the arithmetic operation and sends another message which is send down to the NIC to be transmitted. In the NIC-based case, because the arithmetic operation is performed at the NIC, such messages do not have to be passed between the NIC and the host. Only the initial data needs to be passed from the host to the NIC.

Another potential benefit of NIC-based reduction is reduced host involvement in the operation. For non-root nodes, once the host has sent its data to the NIC, it no longer has to be involved in the reduction operation. In the host-based case, the host process must either wait to receive the messages from its children, or it must be interrupted when the messages arrive so that it can perform the arithmetic operation on them. Since interrupts are time consuming operations, using these can lead to poor reduction latency, so this may not be a good option. Waiting for messages from children may also not be a good option. If processes are skewed, meaning that some processes are performing the reduction operation while others are lagging behind and have not yet started the operation, then intermediate processes may be waiting for other processes in their subtree to catch up. This can lead to poor overall application performance. By using NIC-based reduction operation the host need only supply the data to the NIC. It can then proceed on with other useful computation. This allows greater overlap of computation and communication operations. Furthermore, because the host is not involved in actually performing the operation, NIC-based reduction is a non-blocking operation. The root process need not wait idle for the result after it sends its data to the NIC. It can proceed with other computation and only get the result from the NIC when it needs it.

3 Design and Implementation

We implemented our NIC-based reduction operation as a modification to the GM message passing system[9] which uses the Myrinet network[2], a popular system area network for clusters. Before we describe the design and implementation of our NIC-based reduction, we will give some background on GM and Myrinet.

Myrinet is a high-performance full-duplex 2Gbps network which uses NICs with programmable processors. GM is a user-level message passing system which uses the programmable NICs for much of the protocol processing. GM consists of three components: a kernel module, a user-level library, and a control program which runs on the NIC processor. When a user application wishes to send a message it calls the appropriate function from the library. This function constructs a *send descriptor* which describes what data is to be sent and to which process to sent it to. This descriptor is then written to the NIC using PIO. The NIC detects that a new descriptor has been written and processes it, DMAing the data from the host buffers and transmitting the message. In order to receive a message, the process must provide memory buffers in host memory into which the NIC will DMA the message data. This is done by sending the NIC a *receive descriptor* which describes such a buffer. When the NIC receives a message it DMAs the data into one of the buffers, then DMAs a notification to the host process that a message has been received. The host process can either poll for these notifications, or can block while waiting. In the latter case the NIC will signal an interrupt after it DMAs the notification.

We implemented a NIC-based reduction operation by modifying GM version 1.6.3. Our implementation can perform binary *AND* and *OR* operations, integer *SUM* operations, and floating point *SUM* operations

on a single 64 bit data element. The host process passes a descriptor to the NIC describing the reduction operation. As the NIC receives reduction messages from the network, it performs the arithmetic operation using the data and stores the result. Once messages from all of the children have been received and processed, if the process initiating the reduction operation is the root, the NIC DMAs a notification to the host indicating that the reduction has completed and includes the result. Otherwise, the NIC transmits the result to the parent NIC.

There are several design issues in this implementation, namely, dealing with unexpected messages, dealing with multiple instances of the reduction operation, generating and specifying the tree structure, and performing floating point operations at the NIC.

Unexpected messages — Because processes are not always synchronized, it is possible that some processes may execute the reduction operation before others. This means that a NIC may receive reduction messages from other NICs before the host process has initiated the reduction operation and send its data. Since the host has not informed the NIC which processes to expect data from, and what arithmetic operation to perform, the NIC cannot process the messages. Such a message can be handled in one of two ways. One option is to reject the message and request that the sender retransmit it later. Another option is for the NIC to store the data until the host has initiated that reduction operation. The first option can lead to high latency because the messages need to be retransmitted after a delay. While the second option gives better performance, it requires NIC memory to be allocated for storing this data. Since NIC memory is limited, this may limit the number of messages that can be stored. We used a hybrid approach where we provided a limited number of buffers to store unexpected data, and reject messages once these are full. When the NIC receives a descriptor from the host for a reduction operation, it checks the list of unexpected messages. If it finds any unexpected messages that match, it performs the operation on that data, and frees that unexpected message buffer.

Multiple instances of the reduction operation — When a non-root process initiates a reduction operation, after it sends the data to the NIC, it can proceed with its computation. This means that a process can initiate a second reduction operation before the NIC has completed the first. The NIC needs to be able to process multiple instances of the operations in the correct order. We did this by keeping a queue of instances of reduction operations for each host process. When a reduction message is received from the network for a particular process, the NIC searches the list of instances for that process for a matching instance. If a matching instance is found, the arithmetic operation is performed for that instance, otherwise the message is an unexpected message and is handled as described above.

Generating and specifying the tree structure — The tree structure can be generated by either the NIC or the host process. However, because NIC processors are typically much slower than host processors, it would be more efficient to have the host construct the tree and pass a list of children and the parent to the NIC. We used this option. The send descriptor was only 64 bytes so we are limited as to the number of children that can be specified. Four bytes are needed to specify each child or parent. Since we also include the eight-byte data in the descriptor, and 12 more bytes are used in the descriptor for other fields, there is only room to specify nine children, and one parent.

The shape of the reduction tree is also an important design issue. The latency of the operation increases with each level of the tree, so a very deep tree may not be desirable. On the other hand, a very shallow tree increases network contention as many child nodes transmit their data to one parent node. The exact shape of the tree depends on the performance characteristics of the reduction operation. We have not fully investigated the optimal tree shape for NIC-based reduction. For our evaluation we used a binomial tree because this is the most common tree used for reduction operations, e.g., MPICH[7] uses a binomial tree.

Performing floating point operations at the NIC — The Myrinet NIC processors do not have floating point units. So in order to be able to perform floating point operations, we had to use floating point operations implemented in software. We used the SoftFloat[8] library for these operations. SoftFloat is a free software implementation of the IEC/IEEE Standard for Binary Floating-point Arithmetic, and supports all functions dictated by the standard for 32, 64, and 128 bit floating point formats. We used only the 64 bit format in our implementation.

4 Experimental Results

In this section, we evaluate our implementation on a cluster of 16 quad-SMP 700MHz Pentium-III nodes with 66MHz/64bit PCI. The nodes are connected to a Myrinet2000 network. The NICs are PCI64B cards with 2MB or memory and 133MHz LANai 9.1 processors and are connected to a 16 ports of a 32 port switch. We compare our NIC-based reduction implementation, which is based GM version 1.6.3, to a host-based reduction implementation using the same version of GM.

To evaluate the performance of our implementation, we compare the time from when the last leaf node in the tree initiates the operation until when the root node receives the result. We performed the test in the following manner. All of the nodes perform the reduction operation. As soon as the root node completes the operation and receives the result, it sends a message to the last leaf node of the tree. Once this node receives the message it takes the time

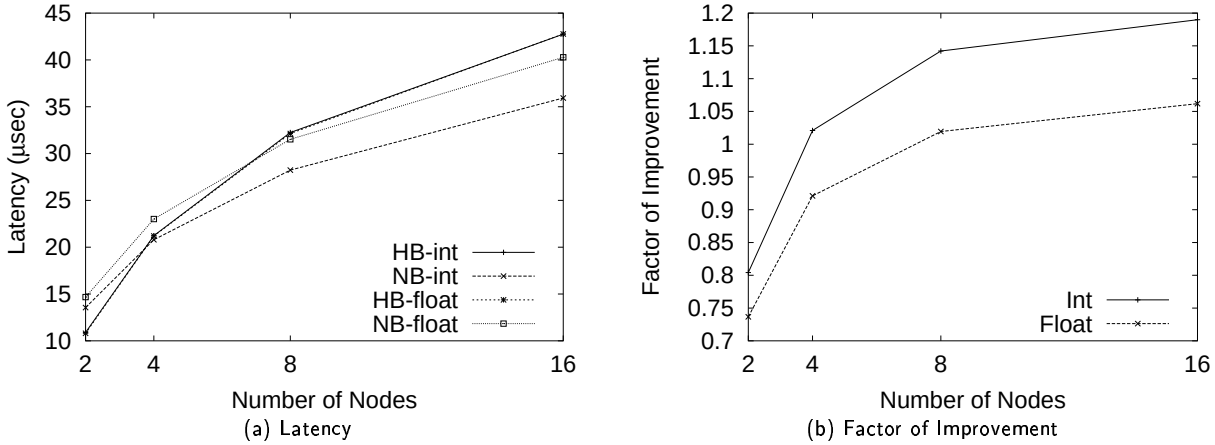

Figure 2: Comparison of NIC-based reduction (NB) and host-based reduction (HB) for integer (int) and floating-point (float) operations

between when it initiated the reduction operation and when it received the message, and subtracts off the one way latency time. We take the average time over 10,000 iterations.

We performed the evaluation for 2, 4, 8 and 16 nodes using integer operations and floating-point operations. Figure 2 shows the results of this evaluation. The figures show the results for the integer SUM operation. The results for the binary AND and OR operations were virtually identical. Notice also that the results for the host based floating point and integer operations were very similar, so the two lines on the graph are on top of one another. The graphs show that for integer operations, the NIC-based reduction performs better than the host-based when the number of nodes is four or greater. We see that the floating-point operations add some overhead, but that the NIC-based reduction is still better than the host based when the number of nodes is eight or greater. We see up to a 1.19 factor of improvement for the integer operation, and up to a 1.06 factor of improvement for floating point operations.

The factor of improvement for the NIC-based reduction increases with the number of nodes. This indicates that for larger system sizes, the NIC-based reduction operation may be even more beneficial. In order to investigate how the relative performance of the NIC-based reduction would change with an increase in system sizes we compared the performance of the operations using a 1-degree tree, in other words a chain, and varied the depth of the tree. Figure 3 show the results of this comparison. Notice again in this graph that the lines for the host-based floating-point and integer operations overlap. This graphs shows us that as the depth of the tree increases the latency of the host-based operation increases faster than the NIC-based operation. In fact the time for the host-based integer reduction increases at a rate of 3.70µs per level of depth faster than that for the NIC-based integer reduction. Similarly, the latency of the host-based floating-point reduction increases at a rate of 2.64µs per level of depth faster than that of the NIC-based floating-point reduction. We see that for a tree of depth 1 the host-based reductions perform better than the NIC-based reductions. Similarly, the host-based floating-point reduction performs better than the NIC-based floating-point reduction for the tree of depth 2. We believe that this is because of the overhead of the more complicated operation at the slower NIC processor. As the depth increases, the number of times messages have to be sent between the NIC and the host at intermediate nodes increases in the host-based reduction. Since the NIC-based reduction avoids this overhead, it performs better for deeper trees.

As system sizes increase, and trees get larger, the maximum degree of the tree also increases. To study the effect of increasing the degree of a tree, we com-

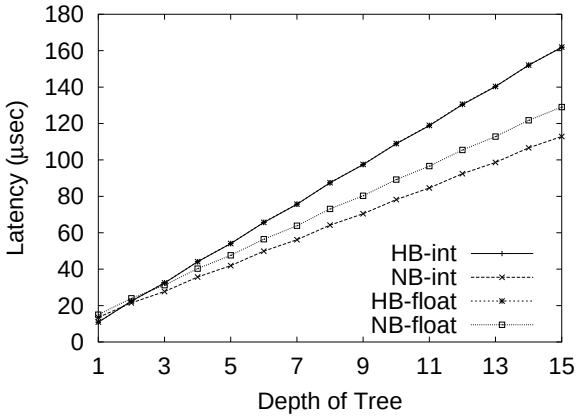

Figure 3: Latency of NIC-based reduction (NB) and host-based reduction (HB) for integer (int) and floating-point (float) operations using a 1-degree tree (a chain) of varying depth

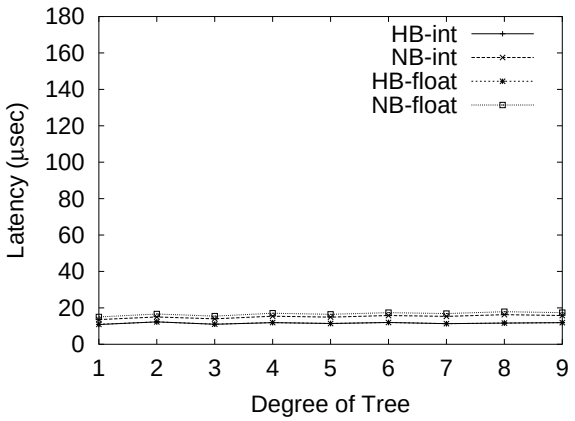

Figure 4: Latency of NIC-based reduction (NB) and host-based reduction (HB) for integer (int) and floating-point (float) operations using trees of depth 1 with varying degree

pared the latency of the NIC-based and host-based reduction operations for trees of depth 1 with varying degree. Figure 4 shows the results of this test. Again the host-based floating-point and integer lines overlap. We see here that the host-based reductions perform better than the NIC-based for any of the trees. However the host-based integer reduction performs only about 2.19µs better, and the host-based floating-point reduction performs only 3.63µs better. Furthermore, while there is a slight increase in overhead for the NIC-based reductions as the degree of the tree increases, it is quite small, 0.22µs per degree for the integer reduction, and 0.24µs per degree for the floating-point reduction. For trees such as binomial trees the depth of the tree increases at the same rate as the depth of the tree. This indicates that the NIC-based reduction will continue to perform better than the host-based reduction for large system sizes.

5 Conclusions and Future Work

We have presented an initial implementation of a NIC-based reduction operation, and evaluated it. We found up to a 1.19 factor of improvement for integer reduction and 1.06 factor of improvement for floating-point reduction. Though this improvement is not very large, the fact that the operation does not involve the host allows useful computation at the host can be overlapped with the reduction operation at the NIC. We also give evidence that the NIC-based reduction will perform better than host-based reduction in larger systems. These results indicate that NIC-based reduction is feasible and that further work should be performed to make these operations more complete.

We intend to improve the NIC-based reduction operation to allow for multiple elements, up to 64 elements of 64 bits. We also intend to increase the maximum number number of children. The current limit is nine children. Another issue that needs to be addressed is the order in which the arithmetic operations are performed. Currently the arithmetic operations are performed on the data in the order in which the messages arrive at the NIC. This may change from one run to the next. Because of the potential of rounding errors, overflow and underflow in floating-point operations, this could lead to the reduction operation giving different results for the same input. By fixing the order in which the operations are applied to the data, the result will be deterministic. We also intend to support more arithmetic operations, such as MAX and MIN. Finally, we intend to implement a NIC-based reduction-to-all operation.

References

[1] R. A. F. Bhoedjang, T. Ruhl, and H. E. Bal. Efficient Multicast on Myrinet Using Link-Level Flow Control. In *Proceedings of the 27th International Conference on Parallel Processing (ICPP '98)*, pages 381–390, August 1998.

[2] 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.

[3] D. Buntinas, D. K. Panda, J. Duato, and P. Sadayappan. Broadcast/Multicast over Myrinet using NIC-Assisted Multidestination Messages. In *Proceedings of Int'l Workshop on Communication and Architectural Support for Network-Based Parallel Computing (CANPC)*, pages 115–129, 2000.

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

[5] D. Buntinas, D.K. Panda, and W. Gropp. NIC-based atomic remote memory operations in Myrinet/GM. In *Workshop on Nove Uses of System Area Networks (SAN-1)*, February 2002.

[6] R. A. Van de Geijn. On Global Combine Operations. *Journal of Parallel and Distributed Computing*, 22:324–328, 1994.

[7] 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, September 1996.

[8] John Hauser. SoftFloat. http://www.cs.berkeley.edu/~jhauser/arithmetic/SoftFloat.html.

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

[10] D. K. Panda. Global Reduction in Wormhole k-ary n-cube Networks with Multidestination Exchange Worms. In *International Parallel Processing Symposium*, pages 652–659, Apr 1995.

[11] K. Verstoep, K. Langendoen, and H. Bal. Efficient Reliable Multicast on Myrinet. In *Proceedings of the International Conference on Parallel Processing*, pages III:156–165, Aug 1996.