All-to-All Broadcast on Mesh Connected Parallel Computers*

David S. Greenberg and Kevin S. McCurley

November 18, 1992

When trying to achieve the highest levels of performance on massively parallel machines, the biggest problem is often in organizing the work so that time spent on communication between processors is kept to a minimum. Very often the natural way in which to implement an algorithm requires a communication procedure known as "all-to-all broadcast", sometimes referred to as "global concatenation". In this operation, every processor sends a message to every other processor, and at the end of the operation all of the processors end up with the messages of all the other processors arranged in numerical order according to the numbering of the processors.

Such operations are used in particle dynamics simulations, in which particles are arranged in space, and every processor needs information about the position of every other particle. Work is divided among all of the processors by having a given processor compute the interactions between a subset of the particles, and computing from this information the new positions of this subset of particles. The updated positions for this subset of particles must then be communicated to all of the other processors before the process can start all over again.

In any distributed memory message-passing machine, the goal of optimizing any global communication operation is to organize the flow of data through the machine in such a way as to avoid conflicts between messages moving through the same channel, and to minimize the total number of messages sent. Which of these is more important depends on the total amount of data being sent. The design of an optimal algorithm for such an operation depends strongly on the topology of the interconnection network, the overhead associated with sending a message, and the relative speeds at which data can be transferred from a processor to the network and across a channel in the network.

Computers such as the nCUBE-2 and the Intel iPSC/860 use what is known as a hypercube topology for the network that connects processors. More recently released machines such as the Intel Delta Touchstone and the Intel Paragon use a mesh topology. In a hypercube topology, the number of processors is of the form 2^d, and each processor is connected to

*This work was supported by the Applied Mathematical Sciences Program, U.S. Department of Energy, Office of Energy Research, and was performed at Sandia National Laboratories, operated for the U.S. Department of Energy under contract number DE–AC04–76DP00789.

d other processors. Figure 1 shows a hypercube of dimension 3, which is simply a cube. In a machine using a two dimensional mesh topology, processors on the edge are connected to either 2 or 3 processors, whereas processors in the interior are connected to four neighbors.

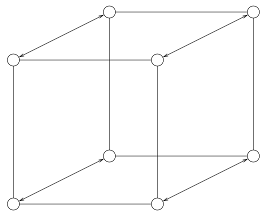

Figure 1: A three dimensional hypercube, showing one step of the binary exchange, in which data is exchanged along one dimension of the hypercube.

When designing an algorithm for all-to-all broadcast on a hypercube computer with 2^d processors, the most commonly used algorithm is commonly referred to as a binary exchange. In a binary exchange, we simply loop through the d dimensions, and in each stage of the loop all processors exchange their information with their neighbor across the dimension of the hypercube. Each processor sends d messages, and there is no contention between messages for the channels of the network, making this an extremely good algorithm. Note however that the messages roughly double in size at each step.

For mesh-connected computers, the best choice for an algorithm is anything but obvious. Experiments done by us have shown that the choice of the best algorithm depends on the size of the initial messages, the dimensions of the mesh, and the speed of the network. Intel provides a library routine for both their Delta Touchstone and iPSC/860 computers called `gcolx()`. In the process of trying to perform a very large computation on the Delta, the second author discovered that the performance of this routine was less than should be expected. In particular, the time for every one of the 512 processor to send 8192 bytes of information to every other processor was between 5.6 and 19.4 seconds, with an average across the processors of 12.5 seconds! We have now implemented several different algorithms, and in particular we have now found one routine that can perform the same operation in between 1.1 and 1.2 seconds, achieving an improvement by a factor of approximately 16 in the worst case.

When the hypercube algorithm is simply applied to a mesh consisting of 2^d processors, multiple messages will compete along the same channel. For a machine like the Delta that has larger bandwidth on the network than between a processor and the network, the impact of this fact is diminished somewhat.

So far, we have implemented multiple variations on three basic algorithms, which we refer to as tree based, ring based, and torus based. The simplest of these is the ring algorithm, in which a ring of processors is embedded in the mesh. We chose to use a snakelike mesh embedding (see figure 2), which requires us to have an even number of rows and columns in the mesh. At the beginning, each processor sends its contribution to its successor along the ring, and receives the contribution from the predecessor. From then on, each processor

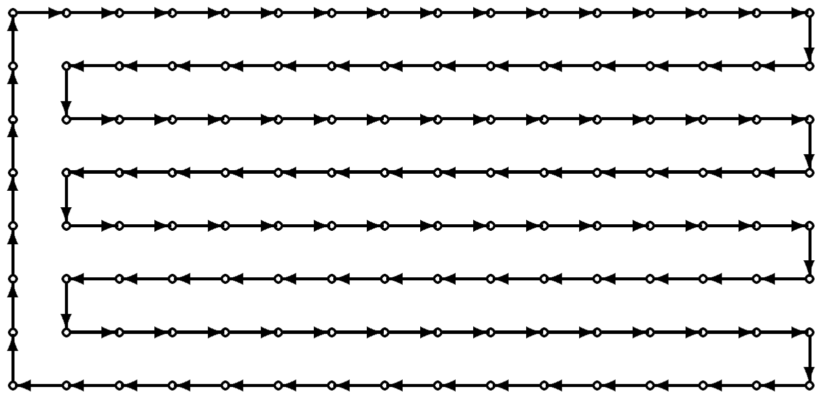

Figure 2: Ring based routing scheme for all-to-all broadcast

sends on the data that they just received. If there are n processors in the mesh, then at the end of $n-1$ steps, every processor will have the entire contribution. This algorithm has the advantage that the total message size never exceeds the maximum size of a processor's original contribution. It has the disadvantage that the number of messages sent is rather large: $n-1$ messages on each processor.

The second algorithm that we implemented is based on the use of what is known as a spanning tree for the graph whose vertices are the processors and whose edges are the network channels. We chose to collapse along a tree to get the entire contribution to processor 0, and then fan the information back out along the tree in reverse. This algorithm has the advantage that it uses relatively few messages, but it also makes rather inefficient use of the network.

The third algorithm that we implemented was a variation of the ring based method which we call the torus algorithm. In this algorithm, we treat each row of processors as a ring, with the last processor in the row connected to the first processor in the row. The messages are passed along this ring until every processor in the row knows the contribution of every other processor in the row. The process is then repeated for the columns, using as the input messages the contribution from the processor's entire row.

As we completed this work we learned of some similar work done by M. Barnett, R. Littlefield, D.G. Payne, and R. van de Geijn on the problem of what is known as a global combine operation. In this operation, each processor starts with a vector of numbers, and in the end every processor ends up with the sum of all of the vectors (the sum may be replaced by any operation that is commutative and associate). The major difference between this problem and the all-to-all broadcast problem is that in a global combine operation, processors finish with exactly the same amount of data that they started with. For an all-to-all broadcast, the processors finish with the sum of the amounts contributed at the beginning. This influences the choice of algorithm somewhat, particularly as the number of processors grows large. The techniques used for the two problems are, however, quite similar.

We chose to implement each of these algorithms to test their performance in practice, since theoretical estimates are often based on too simple a model. We used a variety of tricks to improve the performance on the Delta, primary among them the use of forced message types to avoid buffering of large messages. In addition, Robert van de Geijn of the University of Texas graciously provided us with a piece of code that uses the binary exchange algorithm that is specifically designed for a hypercube, but also performs quite well on the mesh topology of the Delta. In figure 3, we show the performance of these algorithms in comparison to the Intel supplied code for `gcolx()`. From figure 3 it can be seen that van de Geijn's code performed best for small message sizes, the torus algorithm performed best for intermediate to large sized messages. We expect that the absolute best strategy is one that would adapt to the size of messages that are being sent, and perhaps combine different stages of the binary exchange with the torus algorithm.

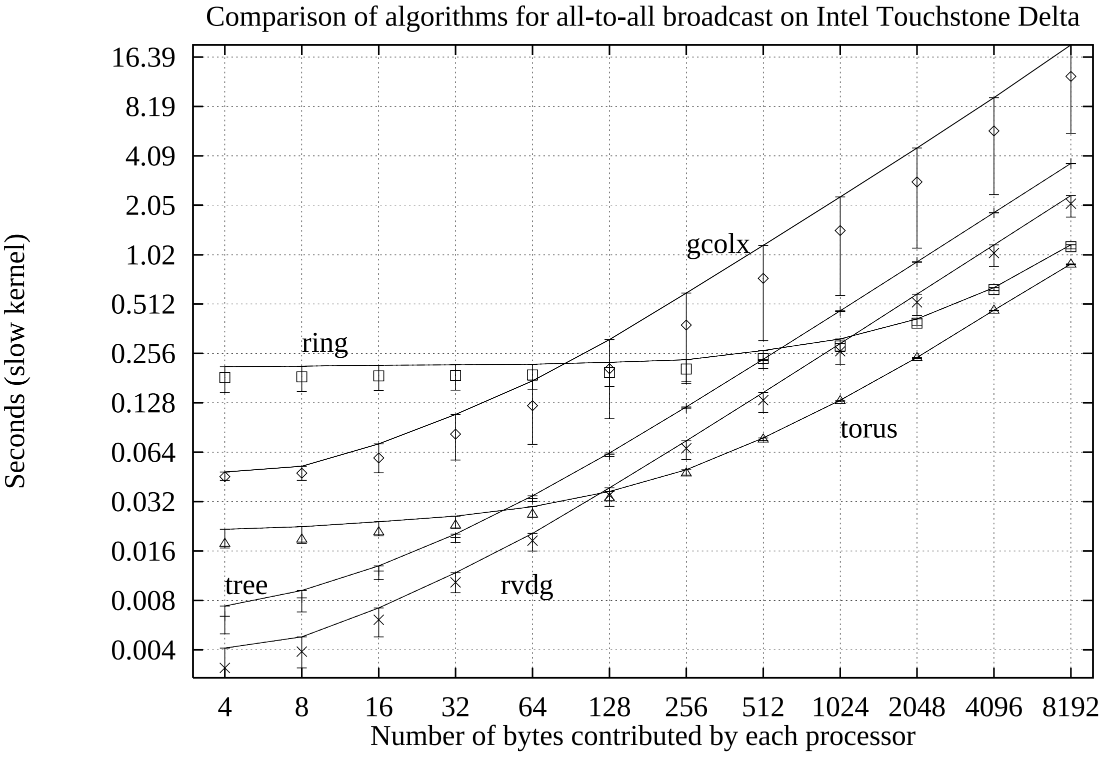

Figure 3: Comparison of running times for five different algorithms used for all-to-all broadcast. The curve labeled "rvdg" refers to the code supplied to us by Robert van de Geijn, which uses a binary exchange algorithm. The curves plot the maximum completion time taken by a node, and the error bars plot the average and minimum times. These timings were done on 512 nodes of the Touchstone Delta using the slow NX kernel.

Our tests assumed that the messages contributed by each processor are the same size, but there is no need for this restriction. One as of yet unexplored area of research is to inves-

bytes	gcolx	torus	ring	rvdg	tree
4	0.0419	0.0091	0.1285	0.0030	0.0057
8	0.0430	0.0095	0.1303	0.0037	0.0064
16	0.0525	0.0109	0.1309	0.0047	0.0084
32	0.0716	0.0130	0.1322	0.0068	0.0123
64	0.1050	0.0162	0.1353	0.0115	0.0193
128	0.1755	0.0233	0.1381	0.0207	0.0336
256	0.3194	0.0365	0.1438	0.0391	0.0623
512	0.6168	0.0644	0.1692	0.0760	0.1198
1024	1.2021	0.1204	0.1968	0.1505	0.2354
2048	2.3704	0.2243	0.2521	0.2992	0.4713
4096	4.8899	0.4426	0.3701	0.5975	0.9310
8192	10.6725	0.8679	0.6345	1.1937	1.8539

Table 1: Maximum execution times for all-to-all broadcast on the Touchstone Delta using the fast NX kernel on 512 processors.

bytes	gcolx	torus	ring	rvdg	tree
4	0.0474	0.0138	0.2110	0.0044	0.0073
8	0.0505	0.0145	0.2129	0.0050	0.0092
16	0.0700	0.0165	0.2150	0.0071	0.0129
32	0.1064	0.0189	0.2165	0.0116	0.0203
64	0.1718	0.0241	0.2187	0.0206	0.0348
128	0.3091	0.0349	0.2240	0.0387	0.0634
256	0.5949	0.0570	0.2429	0.0750	0.1219
512	1.1706	0.1054	0.2657	0.1480	0.2356
1024	2.2956	0.2051	0.3128	0.2931	0.4650
2048	4.5628	0.4007	0.4219	0.5870	0.9307
4096	9.2939	0.7835	0.6577	1.1728	1.8437
8192	19.4309	1.5679	1.1620	2.3477	3.6818

Table 2: Maximum execution times for all-to-all broadcast on the Touchstone Delta using the slow NX kernel on 512 processors.

tigate which of these algorithms perform best when the message sizes vary widely across the machine. We also hope to explore the best choice of algorithm for the forthcoming Paragon series of mesh connected computers, on which there are several different characteristics that may influence the ultimate choice for the best performing code.

For further information on this subject, contact

Kevin McCurley
Organization 1423
Sandia National Laboratories
Albuquerque, NM 87185
email: mccurley@cs.sandia.gov