MESSAGE ROUTING SCHEMES IN A HYPERCUBE MACHINE

S. Raghupathy, M. R. Leuze, and S. R. Schach
Computer Science Department
Vanderbilt University
Box 1679, Station B
Nashville, TN 37235

ABSTRACT

The aim of this study is to determine efficient routing schemes for message passing in a hypercube machine. Two different algorithms are considered, namely static routing in which the path of a message is predetermined by the addresses of the source and destination nodes, and dynamic routing where the decision as to the next node in the path is made by the current node on the basis of local information regarding queue lengths. In addition, various different prioritization schemes are compared for both static and dynamic routing. The results show that dynamic routing can be up to twice as efficient as static routing, provided priority is given to messages which have only a few hops to traverse or were transmitted early in the computation sequence.

1. INTRODUCTION

A hypercube of order n consists of 2^n processors interconnected in the form of a binary n-cube in which the individual processors communicate with one another by means of message passing. In commercially available hypercubes *static routing* of messages is used, that is to say, given a source node N_s and a destination node N_d, the path of every message from N_s to N_d is determinate. No matter how much congestion there is along the fixed path from N_s to N_d, and irrespective of queue lengths at the intermediate nodes, the message must travel along this path, and no other.

Using a store-and-forward technique (as opposed to wormhole routing [1], or cut-through routing [2]), the static routing algorithm is implemented as follows. The addresses of the 2^n processors can be represented using n bits. Suppose that the message is currently at node N_c which is represented by bit pattern $(c_{n-1}, \ldots, c_1, c_0)$, and that the destination node N_d is represented by bit pattern $(d_{n-1}, \ldots, d_1, d_0)$. Let i denote the index of the rightmost bit in which N_c and N_d differ. Then the next node on the route from N_s to N_d is the node represented by bit pattern $(c_{n-1}, \ldots, c_1, c_0)$ with bit i flipped, that is to say, the message is routed in dimension i. The algorithm continues in this way until the message arrives at node N_d. The path from N_c to N_d can consist of at most n hops, corresponding to the case in which all n bits have to be flipped.

Superficially, static routing seems somewhat short-sighted. After all, *dynamic routing* (that is to say, allowing the message route from N_s to N_d to vary depending on circumstances) must be preferable to static routing because it allows every message to select the (locally) optimal route under the current circumstances; any sort of choice is surely preferable to no choice at all. While dynamic routing does indeed have this advantage, there is a price to pay, namely the overhead of implementing dynamic routing. At each node calculations have to be performed to determine the next node to which the message should be routed, and links have to be tested to see which ones are free. The size of the overhead will vary from hypercube to hypercube. In some machines, the additional work can be done in hardware in parallel with other operations; in other machines, it must be done in software, using machine cycles that could otherwise be used for productive computing.

The purpose of this work is threefold First, we investigate dynamic routing empirically to see just how much more efficient dynamic routing is than static routing. Since we do not know what the overhead would be on a specific existing or future hypercube machine, we ignore the overhead, and simply measure how much faster dynamic routing is than static routing. If this difference is sufficiently large, it seems reasonable to conclude that, however it is implemented in practice, dynamic routing with all its overheads will be more efficient than static routing. Second, both static and dynamic routing can be used in conjunction with a number of different prioritization schemes, such as next transmitting the message with the smallest number of remaining hops, or the message that has been the longest time in the system. In this paper we examine the effect of various prioritization schemes on both static and dynamic routing. Third, we evaluate the effect of connecting nodes by means of a bidirectional link, as opposed to two unidirectional links.

In this research we have restricted ourselves to problems which can be represented as directed acyclic graphs (DAGs) of processes. Each process receives zero or more messages, after which it computes, transmits zero or more messages, and then terminates. An example of this type of problem is parallel Gaussian elimination in which each process is responsible for maintaining and updating a single row of a large, sparse matrix. A further assumption is that there is a one-to-one correspondence between processes and processors. Finally, we ignore the problem of deadlock caused by full message buffers [3].

The rest of this paper is organized as follows. In section 2 the various routing techniques are outlined. In section 3 we describe the simulator. Results obtained using the simulator are given in section 4. Our conclusions are in section 5.

2. ROUTING TECHNIQUES

The choice of routing scheme is a critical factor in the design of an interconnection network. While a simple routing scheme may not be able to exploit all the capabilities of a network, a complex routing scheme making use of detailed global knowledge about flow patterns and message queue lengths in every node will have large computational overheads. In this work we consider two routing schemes, namely static routing and dynamic routing. Both schemes use local knowledge only. In *static* routing, the next node to which a message is sent is determined using the algorithm described in section 1. Blocking is a common consequence of static routing, because the path to be followed by a message is determined solely from the source and destinations addresses; no account is paid to the current distribution of messages in the system. In contrast, in *dynamic* routing if the next hop (as determined by the static routing algorithm) is blocked, then an attempt is made to send the message along a different route to its destination. Suppose that the message is currently at node N_c which is represented by bit pattern $(c_{n-1}, \ldots, c_1, c_0)$, and the destination node N_d is represented by bit pattern $(d_{n-1}, \ldots, d_1, d_0)$. Let the bit patterns representing N_c and N_d differ in bit positions $p_j, p_{j-1}, \ldots, p_1$ with $1 \leq j \leq n-1$. If the static algorithm is used, then the next node on the route from N_s to N_d will be the node represented by bit pattern $(c_{n-1}, \ldots, c_1, c_0)$ with the bit in position p_1 flipped. It makes no difference if the appropriate link is in use, and if so, how long the queue for that link is. In dynamic routing, however, if the indicated link is blocked then an attempt is made to route the message along one of the other dimensions in which it still has to travel. Thus if dimension p_1 link is blocked, an attempt is made to route it along dimension p_2, and so on. Only if the links in all j of the dimensions in which the message must still travel are in use can the message be considered to be blocked.

A second aspect of routing is prioritization. If a number of messages are waiting to use a link, one method of choosing which message to transmit is on the basis of first in, first out (FIFO), the method used in commercial hypercubes. In this work we consider various alternative prioritization schemes, such as LIFO, giving priority to the message with the maximum number of remaining hops, or the minimum number of remaining hops. In

addition, since the processes form a DAG, each process can be assigned a sequence number such that every message is sent to a process with a higher sequence number than the sequence number of the process that generated the message. The sequence number of the generating process can then be used to prioritize messages. The complete list of prioritization schemes appears in Figure 1.

Third, we considered two types of interconnecting link, namely a single bidirectional link between nodes (as in commercial hypercubes), and two unidirectional links, one in each direction.

3. THE SIMULATOR

The simulator was constructed to investigate routing strategies. Each message in the system essentially consists only of header fields; the data field is ignored because it is irrelevant from the viewpoint of routing. The header contains information such as source and destination node, as well as information needed when the order of transmission of messages is done on the basis of prioritization, such as sequence number, time generated, arrival time at the current node, and number of hops that still have to be traversed. Figure 2 depicts the structure of a message.

The execution cycle of the simulator consists of three phases: message generation, message ordering, and message routing. During the *message generation* phase, each active process is checked to see if it has received all the messages it requires. If so, the messages it is to transmit are generated, and placed in the message buffer. The process then terminates. After all possible messages have been generated, the simulator enters the *message ordering* phase. Here, the messages in each buffer are ordered according to the prioritization scheme currently being evaluated (in the case of equal priorities, ties are broken randomly). Finally, the *message routing* cycle commences. Here, each message is fetched from the message buffer and an attempt is made to transmit it to a neighboring node. If static routing is being used, and the predetermined link is in use, then that particular message is blocked. When dynamic routing is used, an attempt is made to transmit the message over the first unused link that will move it closer to its destination.

In a real hypercube, each node generates messages as soon as it has acquired all its input messages. The exact order and the precise time at which messages are generated will depend on delays within the system, and (as with most distributed systems) will vary from run to run. In our simulator, messages are generated "in step," that is to say, at the message generation phase of the cycle all possible messages are generated. Further, they are generated in a fixed order which does not vary from run to run. To add an element of randomness to the simulation, the contents of a message buffer are therefore randomly shuffled at the end of each generation phase. In addition, when priorities are used, after shuffling the contents of a buffer the first message of highest priority is transmitted next. In this way, if two or more messages have equal priority then shuffling has the effect of breaking ties randomly.

4. RESULTS

All possible combinations of routing scheme (static or dynamic), link (one bidirectional link or two unidirectional links), and prioritization scheme (see Figure 1) were simulated, a total of 44 different cases. Each case was run on a (simulated) hypercube of dimension $N = 8$ (256 nodes). Each case was repeated 10 times, and the mean computed. Each case was repeated many times to smooth out the effects of the random shuffling described at the end of the previous section.

A number of different input graphs were considered. Since the results were very similar in all instances, for brevity only one input case is presented here. Figure 3 depicts the 22 cases with one bidirectional link between nodes, and Figure 4 shows the 22 cases with two unidirectional links.

Not surprisingly, dynamic routing performs better than static routing, but the improvement factor varies depending on the prioritization scheme. At best, the improvement is by a factor of two. Overall, the best results occur

when priority is given to messages with the lowest sequence number (Scheme E). Results almost as good are obtained when priority is given to messages with the fewer number of hops, either in the original message (Scheme A) or remaining to be traversed (Scheme C). Messages of lowest sequence number are essentially those transmitted earliest in the computation sequence. Giving priority to such messages essentially speeds up the rate at which processes can begin transmitting, and hence speeds up the computation as a whole. With regard to messages with the fewest numbers of hops, by giving them priority, the traffic congestion in the hypercube is decreased, thus allowing the longer messages to proceed with less blocking than would otherwise be the case. In addition, messages with many hops still remaining have more choices of routes along which to travel to their final destination than messages with only one or two hops still to complete. By giving priority to messages with fewer choices, the overall amount of blocking is decreased.

From Figure 5 we see that two unidirectional links (again not surprisingly) improve throughput over one bidirectional link, and again the improvement depends on the prioritization scheme. What is perhaps unexpected is that the improvement is rarely of the order of more than fifteen per cent, and is usually much smaller. This effect may be caused by the fact that the problem graph is a DAG, thereby imposing a directionality on the flow of messages.

5. CONCLUSIONS

We have seen that the throughput of a certain class of problems on a hypercube can be increased by up an order of two through use of dynamic rather than static routing algorithms, and also by prioritizing the messages. However, the increased cost of having two unidirectional links between nodes (rather than a single bidirectional link) does not seem warranted for the class of problems considered here.

A number of different prioritization schemes have been considered. Nevertheless, it is possible that other prioritization schemes might result in a further increase in throughput for this class of problems. It is likely that different prioritization schemes would yield improved throughput for other classes of problems. In short, although the results are encouraging, further research is needed to determine whether dynamic routing algorithms should be used in all hypercubes.

REFERENCES

[1] William J. Dally, "A VLSI Architecture for Concurrent Data Structures," Ph.D. Thesis, Computer Science Department, California Institute of Technology, Technical Report 5209:TR:86, 1986.

[2] Parviz Kermani and Leonard Kleinrock, "Virtual cut-through: A new computer communications switching technique," *Computer Networks* vol. 3, pp. 267–286, 1979.

[3] William J. Dally and Charles L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," Computer Science Department, California Institute of Technology, Technical Report 5231:TR:86, 1986.

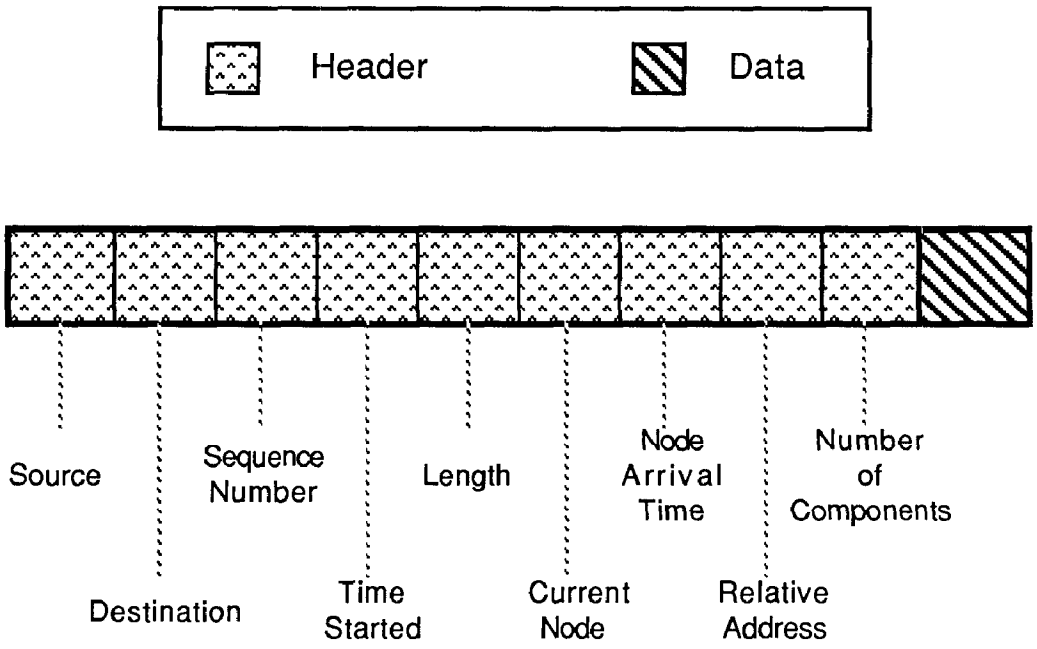

Figure 1: Message format

Priority is given to a message with :

A Minimum remaining number of hops
B Maximum remaining number of hops
C Minimum original number of hops
D Maximum original number of hops
E Minimum sequence number of originating process
F Maximum sequence number of originating process
G Earliest generation time
H Latest generation time
I Earliest arrival time at a node (FIFO)
J Latest arrival time at a node (LIFO)
K (No prioritization scheme)

Figure 2: The prioritization schemes

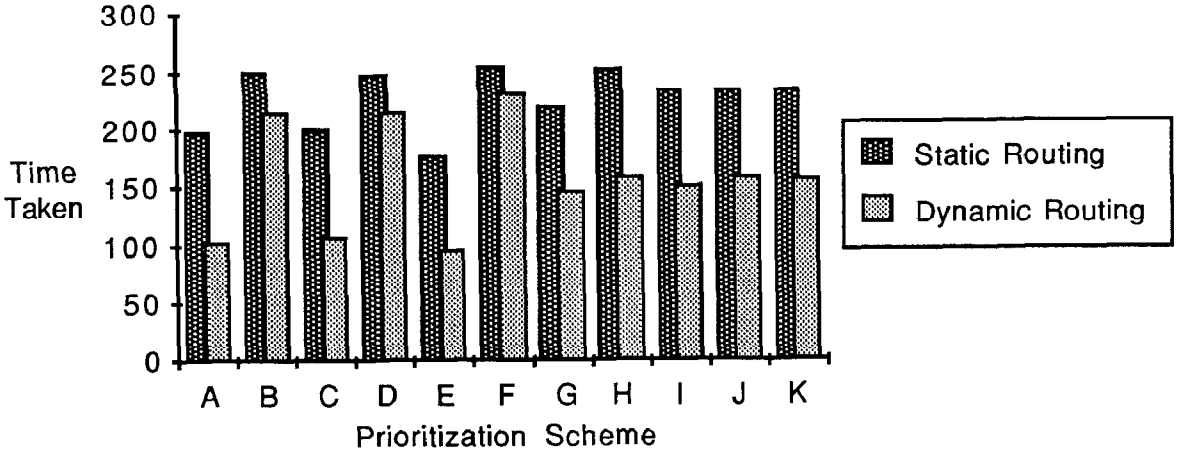

Figure 3: Running time with single bidirectional link between nodes. (The letters refer to the prioritization schemes listed in Figure 2).

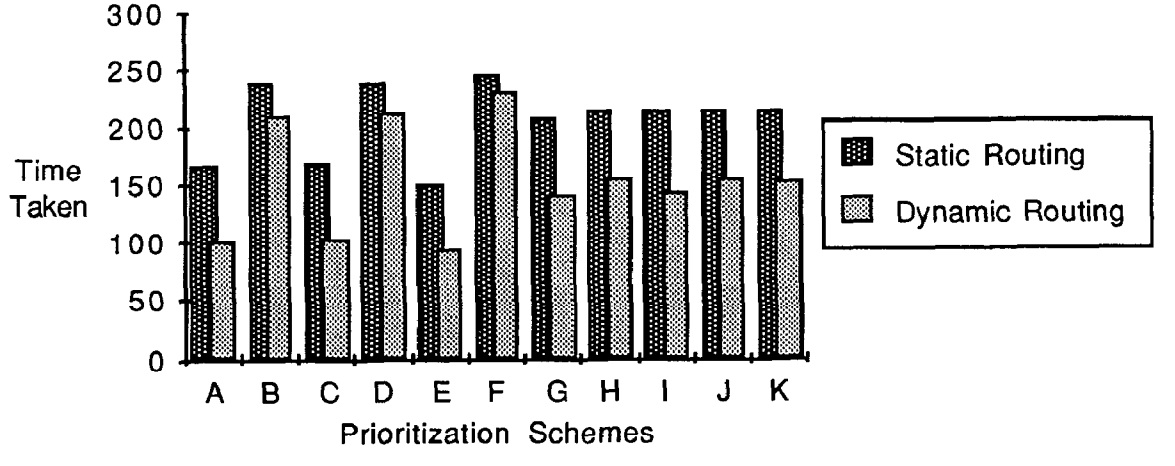

Figure 4: Running time with two unidirectional links between nodes. (The letters refer to the prioritization schemes listed in Figure 2).

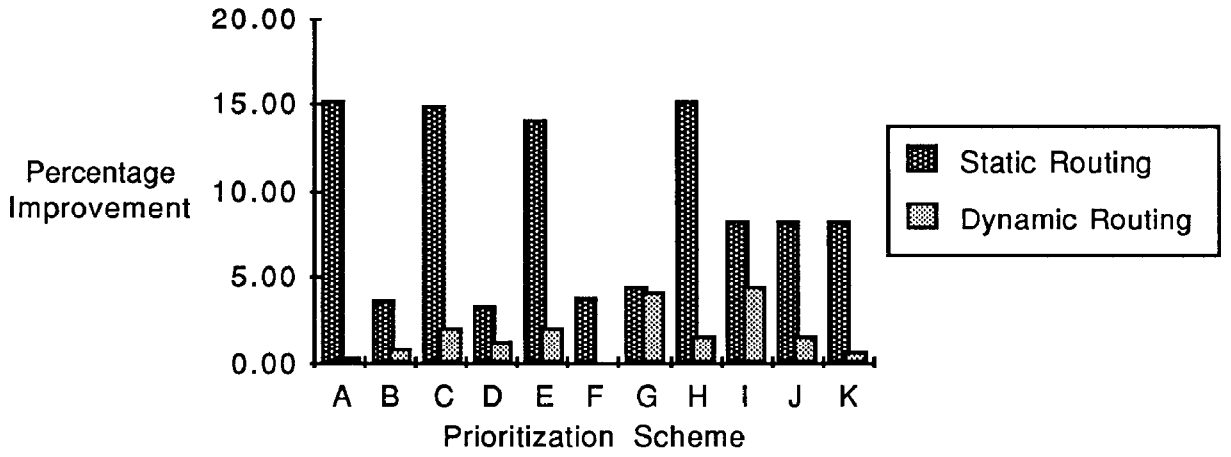

Figure 5: Percentage improvement in running time when two unidirectional links are used.