system solvers," in *Proc. 1988 Int. Conf. Parallel Processing*. Univ. Park, PA: Penn. State Univ. Press, 1988, vol. 1, pp. 166–173.
[4] W. Abu-Sufah and A. D. Malony, "Vector processing on the Alliant FX/8 multiprocessor," in *Proc. 1986 Int. Conf. Parallel Processing*. Univ. Park, PA: Penn. State Univ. Press, 1986, pp. 559–566.
[5] Alliant Computer Systems Corp., *FX/Series Architecture Manual*, Littleton, MA, 1986.
[6] P. E. Bjorstad, "A large scale, sparse, secondary storage, direct linear equation solver for structural analysis and its implementation on vector and parallel architectures," *Parallel Comput.*, vol. 5, pp. 3–12, 1987.
[7] D. A. Calahan, "Parallel solution of sparse simultaneous linear equations," in *Proc. 11th Allerton Conf. Circuits Syst. Theory*, Univ. Illinois, Urbana, IL, 1973, pp. 729–735.
[8] G. Dahlquist and A. Bjork, *Numerical Methods*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
[9] A. K. Dave and I. S. Duff, "Sparse matrix calculations on the CRAY-2," *Parallel Comput.*, vol. 5, pp. 55–64, 1987.
[10] T. A. Davis, "PSolve: A concurrent algorithm for solving sparse systems of linear equations," CSRD Rep. 612, Center Supercomput. Res. Develop., Univ. Illinois, Urbana, IL, 1986.
[11] T. A. Davis and E. S. Davidson, "PSolve: A concurrent algorithm for solving sparse systems of linear equations," in *Proc. 1987 Int. Conf. Parallel Processing*. Univ. Park, PA: Penn. State Univ. Press, 1987, pp. 483–490.
[12] E. W. Dijkstra, "Hierarchical ordering of sequential processes," in *Operating Systems Techniques*, C. A. R. Hoare and R. H. Perrott, Eds. New York: Academic, 1972, pp. 72–93.
[13] I. S. Duff, R. Grimes, J. Lewis, and B. Poole, "Sparse matrix text problems," *SIGNUM Newsletter*, vol. 17, p. 22, 1982.
[14] I. S. Duff and J. K. Reid, "Some design features of a sparse matrix code," *ACM Trans. Math. Software*, vol. 5, pp. 18–35, 1979.
[15] ——, "The multifrontal solution of unsymmetric sets of linear equations," *SIAM J. Sci. Stat. Comput.*, vol. 5, pp. 633–641, 1984.
[16] I. S. Duff, "Parallel implementation of multifrontal schemes," *Parallel Comput.*, vol. 3, pp. 193–204, 1986.
[17] S. C. Eisenstat, M. C. Gursky, M. H. Schultz, and A. H. Sherman, "The Yale sparse matrix package, II: The non-symmetric codes," Rep. 114, Dep. Comput. Sci., Yale Univ., 1977.
[18] ——, "Yale sparse matrix package, I: The symmetric codes," *Int. J. Numer. Meth. Eng.*, vol. 18, pp. 1145–1151, 1982.
[19] A. George, M. T. Heath, J. W. H. Liu, and E. Ng, "Solution of sparse positive definite systems on a shared-memory multiprocessor," *Int. J. Parallel Programming*, vol. 15, no. 4, pp. 309–325, 1986.
[20] A. George, M. T. Heath, E. Ng, and J. W. H. Liu, "Symbolic Cholesky factorization on a local-memory multiprocessor," *Parallel Comput.*, vol. 5, pp. 85–95, 1987.
[21] R. W. Hockney and C. R. Jesshope, *Parallel Computers*. Bristol, England: Adam Hilger, 1981.
[22] J. W. Huang and O. Wing, "Optimal parallel triangulation of a sparse matrix," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 726–732, 1979.
[23] J. A. G. Jess and H. G. M. Kees, "A data structure for parallel L/U decomposition," *IEEE Trans. Comput.*, vol. C-31, pp. 231–239, 1982.
[24] H. M. Markowitz, "The elimination form on the inverse and its application to linear programming," *Management Sci.*, vol. 3, pp. 255–269, 1957.
[25] R. G. Melhem, "A modified frontal technique suitable for parallel systems," *SIAM J. Sci. Stat. Comput.*, vol. 9, pp. 289–303, 1988.
[26] J. W. H. Liu, "Computational models and task scheduling for parallel sparse Cholesky factorization," *Parallel Comput.*, vol. 3, pp. 327–342, 1986.
[27] F. J. Peters, "Parallel pivoting algorithms for sparse symmetric matrices," *Parallel Comput.*, vol. 1, pp. 99–110, 1984.
[28] A. H. Sameh, "On some parallel algorithms on a ring of processors," *Comput. Phys. Commun.*, vol. 37, pp. 159–166, 1985.
[29] D. C. Sorenson, "Analysis of pairwise pivoting in Gaussian elimination," *IEEE Trans. Comput.*, vol. C-34, pp. 274–278, 1985.
[30] R. P. Tewarson, *Sparse Matrices*. New York: Academic, 1973.
[31] J. H. Wilkinson, "Error analysis of direct methods of matrix inversion," *J. ACM*, vol. 8, pp. 281–330, 1961.
[32] O. Wing and J. W. Huang, "A computation model of parallel solution of linear equations," *IEEE Trans. Comput.*, vol. C-29, pp. 632–638, 1980.

Reliable Broadcast in Hypercube Multicomputers

P. RAMANATHAN AND KANG G. SHIN

***Abstract*—A simple algorithm to broadcast in a hypercube multicomputer containing faulty nodes/links is proposed. The algorithm delivers multiple copies of the broadcast message through disjoint paths to all the nodes in the system. The salient feature of the proposed algorithm is that the delivery of the multiple copies is transparent to the processes receiving the message and does not require the processes to know the identity of the faulty processors. The processes on nonfaulty nodes that receive the message identify the original message from the multiple copies using some scheme appropriate for the fault model used. The algorithm completes in $n + 1$ steps if each node can simultaneously use all of its outgoing links. But if each node cannot use more than one outgoing link at a time, then the algorithm requires $2n$ steps.**

***Index Terms*—Distributed agreement, distributed fault-tolerant couting, hypercube multicomputers, quorum majority.**

I. Introduction

Several topologies have been proposed for interconnecting the processors of a distributed computing system. Among them, due to its topological richness, the binary hypercube topology has drawn considerable attention from both academic and industrial communities. Both research [7] and commercial (by Intel, NCUBE, Floating Point Systems, Ametek, Thinking Machine, to name a few) systems have been built using the hypercube interconnection topology, and significant research efforts have been made on hypercube architectures.

Most of the research effort on hypercube architecture has focused on the fault-free situation. However, the increasing use of hypercube multicomputers for critical applications has made their fault tolerance an important issue. In this paper, we present a simple algorithm for broadcasting in a hypercube in the presence of node/link faults. This work is motivated by the need for a simple broadcast algorithm for implementing fault-tolerant algorithms for problems like distributed agreement [4], [10] and clock synchronization [3], [9].

Distributed agreement and clock synchronization can be achieved only if a nonfaulty node can correctly deliver its private value to all the other nonfaulty nodes in the system [3], [4], [9], [10]. This, however, is not easy to achieve in the presence of faults because the faulty nodes can either omit, corrupt, reroute, or alter information passing through them. There are two possible approaches to overcome the problem. In the first approach, each node keeps limited information about the faulty nodes in the system. Fault-tolerant routing/broadcasting is achieved by going around the faulty nodes [1], [5]. This approach can be used only if it is possible to identify the faulty processors "on-line." Since the overhead of identifying the faulty processors and passing the fault information to the other nodes could be quite severe, this approach is not suitable for many time-critical applications. In the second approach, fault tolerance is achieved by sending multiple copies of the message through *disjoint* paths. The nodes that receive the message identify the original message from the multiple copies by using some scheme that is appropriate for the fault model, e.g., majority voting. The second approach has the advantage of not having to identify the faulty

Manuscript received February 21, 1988; revised July 12, 1988. This work was supported in part by NASA Grant NAG-1-296, and ONR Contracts N00014-85-K-0531 and N00014-85-K-0122. Any opinions, findings, and conclusions or recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding agencies.

The authors are with the Real Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 8824094.

processors during the normal operation of the system. This advantage is especially important in critical real-time applications, because real-time systems normally cannot tolerate the time overhead required to identify the faulty processors.

In this paper, we present a broadcast algorithm that delivers multiple copies of the message to all nodes in the hypercube through disjoint paths. Srikanth and Toueg [10] have also proposed a broadcast algorithm that uses the multiple copy approach. However, their algorithm does not make explicit use of the properties of the hypercube topology. Their algorithm can be used in any topology with sufficient connectivity, but the burden of ensuring the delivery of multiple copies lies with the processes receiving the message. On receiving a broadcast message, the processes determine the number of copies of the same message they have already received. If the number of copies is less than a specified threshold, then the processes retransmit the message to the processes in the neighboring nodes.

The basic idea of our algorithm is as follows. The node that wants to broadcast a message sends the message to all its neighbors. The neighbors in turn broadcast the message they received using a simple coordinated recursive doubling algorithm. The recursive doubling algorithm executed by the neighbors is coordinated such that the copies of the message received by a node have traveled through disjoint paths. The salient feature of the proposed algorithm is that the delivery of the multiple copies is transparent to the processes receiving the message and does not require the processes to know the identity of the faulty processors. Depending on the fault model used, the algorithm can tolerate either $n - 1$ or $\lfloor n/2 \rfloor$ or $\lfloor n/3 \rfloor$ node/link faults. The algorithm completes in $n + 1(2n)$ steps if each node can use all (at most one of) its outgoing links at a time.

This paper is organized as follows. Section II describes the problem and the notation used in the paper. Section III describes the proposed algorithm. Section IV evaluates the performance of the algorithm in terms of steps required for completion for different communication capabilities at each node. Finally, Section V concludes the paper.

II. Notation and Problem Statement

Definition 1: An n-dimensional hypercube Q_n is defined recursively as follows.

1) Q_0 is a trivial graph with one node, and

2) $Q_n = K_2 \times Q_{n-1}$, where K_2 is a complete graph with two nodes and $\times$ is the product operation on two graphs [2].

Examples of Q_1, Q_2, and Q_3 are shown in Fig. 1. Each node in a Q_n can be uniquely represented by an n-bit address in such a way that the addresses of the adjacent nodes differ in exactly one bit. For convenience, we will number the bits in an address of a node in the Q_n from right to left as 0 to $n - 1$. If two adjacent nodes differ in their ith bit, then they will be said to be in *direction i* with respect to each other. For example, the node with address $u = 0111$ will be said to be in direction 1 of a Q_4 with respect to node $w = 0101$ and vice versa. It is clear from this definition that there are n distinct directions in a Q_n, denoted by $d_0, d_1, \cdots, d_{n-1}$. The node in direction i with respect to the node u will be denoted by $\oplus_i(u)$. Also let $[m]_n \equiv m \bmod n$, where m and n are integers.

Definition 2: The *direction set* of two nodes $u = u_{n-1}u_{n-2} \cdots u_0$ and $w = w_{n-1}w_{n-2} \cdots w_0$ in a Q_n is defined as $H(u, w) = \{d_i : u_i \neq w_i\}$. The elements of $H(u, w)$ will be referred to as the *Hamming directions* of the two nodes u and w.

The problem addressed in this paper can be easily stated as follows. Given 1) an n-dimensional hypercube subject to node/link failures, 2) there are a maximum of t node/link faults in the hypercube, and 3) the identity of the faulty node/links is not known, the problem is to develop a broadcast algorithm that satisfies the following condition.

C1: If the node initiating the broadcast is nonfaulty, then all the nonfaulty nodes in the hypercube must agree on the message broadcast by the initiating node.

As shown below in Example 1, C1 is not always easy to satisfy in the presence of faults. We will first describe a broadcast algorithm and then determine the maximum number of faults t that the algorithm can tolerate for different fault models. The proposed algorithm is suitable for applications that cannot tolerate the time overhead of identifying the faulty processors, which could in general be quite long.

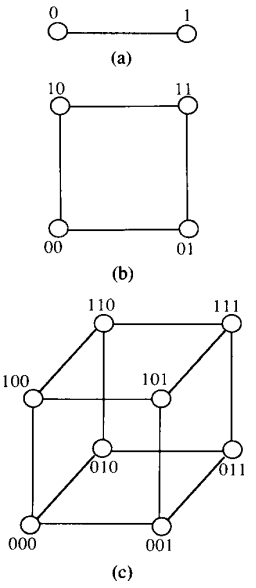

Fig. 1. Illustrative examples of hypercubes, (a) Q_1. (b) Q_2. (c) Q_3.

Example 1: Consider the hypercube Q_3 shown in Fig. 1. Suppose node 0 initiates a broadcast using the following recursive doubling algorithm.

Step 1: Node 0 sends the message to node 1.

Step 2: Nodes 0 and 1 simultaneously send the message to nodes 2 and 3, respectively.

Step 3: Nodes 0, 1, 2, and 3 simultaneously send the message to nodes 4, 5, 6, and 7, respectively.

Now suppose node 1 is faulty. Then, at the end of the algorithm, nodes 3, 5, and 7 may have received either an incorrect message or no message at all and condition C1 is violated.

III. Proposed Broadcast Algorithm

There are two aspects to any broadcast algorithm: the *delivery* and the *reception* of messages. The delivery of messages is comprised of algorithms used by the nodes to deliver multiple copies of the broadcast message to all nodes. In the proposed algorithm, n copies of the message are correctly delivered to all nodes if there are no faults in the system, where n is the dimension of the hypercube. However, in the presence of faults, some of the n copies may either get lost or corrupted.

The "messages reception" is comprised of algorithms used by the nodes to interpret and identify the correct information from the multiple copies. The identification of the correct information from the multiple copies is strongly dependent on the fault model used. Sections III-A and III-B describe the delivery and reception mechanisms of the proposed algorithm.

A. The Delivery Mechanism

The delivery mechanism proceeds in two phases. In the first phase, the node initiating the broadcast sends the message to all its neighbors. In the second phase, the neighbors use "coordinated recursive doubling" to broadcast the message to all the nodes. The sequence of directions used by these neighbors in their recursive doubling phase is coordinated so as to ensure that each node gets the broadcast message through n disjoint paths. A formal description of the algorithm is given below.

TABLE I
PATHS THROUGH WHICH THE NODES RECEIVE THE BROADCAST MESSAGE

Node	Paths via Node 1	Paths via Node 2	Paths via Node 4
1	0	0-2-3	0-4-5
2	0-1-3	0	0-4-6
3	0-1	0-2	0-4-5-7
4	0-1-5	0-2-6	0
5	0-1	0-2-6-7	0-4
6	0-1-3-7	0-2	0-4
7	0-1-3	0-2-6	0-4-5

Algorithm A:

```
procedure recursive_doubling (m, k)
  /* m: initiating node; d_k: starting direction */
begin
  for 0 ≤ l ≤ n − 1 do begin
    R := {m};
      /* R: set of nodes that have received the message*/
    for each node j ∈ R
      begin
        send message from j to ⊕_[k+l]_n(j);
        R := R ∪ {⊕_[k+l]_n(j)};
      end;
  end
end

Algorithm broadcast(s);  /* s : initiating node */
begin
  for 0 ≤ i ≤ n − 1 do begin
    send message from s to ⊕_i(s);
    recursive_doubling (⊕_i(s), i + 1);
  end;
end.
```

In the absence of faults, each node gets n identical copies of the message in the above algorithm. This is because the nodes get one message in each of the recursive doubling sequences initiated by the n neighbors of s. It is shown later in Theorem 1 that the paths through which a node receives the n copies of the message are disjoint. Given below is an example to illustrate the basic idea of the algorithm.

Example 2: Consider the hypercube Q_3 in Fig. 1. Let node 0 initiate the broadcast. In the first phase, node 0 sends its message to nodes 1, 2, and 4. In the second phase, nodes 1, 2, and 4 use coordinated recursive doubling to broadcast the message they received from node 0 to all other nodes. In its recursive doubling, node 1 first sends message to node 3 along d_1, then 1 and 3 send messages along d_2 to nodes 5 and 7, and finally nodes 1, 3, 5, and 7 send messages along d_0 to nodes 0, 2, 4, and 6. Similarly, node 2 (4) uses $d_2 \rightarrow d_0 \rightarrow d_1 (d_0 \rightarrow d_1 \rightarrow d_2)$ as the sequence of directions in its coordinated recursive doubling. Table I shows the disjoint paths through which nodes 1-7 receive their messages. In the table, columns 2, 3, and 4 indicate the path through which the node in column 1 receives its message in the recursive doubling phase initiated by nodes 1, 2, and 4, respectively.

It is important to note that if the neighbors of the initiating node do not coordinate the sequence of directions in their recursive doubling, then some nodes will not receive their copies of the message through disjoint paths. If two or more copies are received through nondisjoint paths, then a single faulty node could corrupt more than one copy of the message. As we will see later in Section III-B, this could cause severe problems in identifying the original message from the multiple copies.

The following example illustrates the effect on the paths of the multiple copies if the neighbors of the initiating node do not adhere to the sequence of directions specified in Algorithm A.

Example 3: Suppose that node 2 of Q_3 in Fig. 1 does not adhere to the sequence of directions specified in Algorithm A. Let the sequence of directions used by node 2 be d_0, d_2, and d_1. Then it is easy to verify that node 7 will receive one message from node 0 through nodes 2 and 3 and the other message from node 0 through nodes 1 and 3, i.e., node 7 receives messages from node 0 through paths that are not disjoint.

In the rest of this section, we formally prove the results indicated by Examples 2 and 3. Let R_i^s refer to the recursive doubling phase of $\oplus_i(s)$ in the above algorithm. Let $P_i^s(q)$ denote the path through which node q receives a copy of the message in R_i^s. For clarity of presentation we will drop the superscript s from the notation. Define $P_i(q) \cap P_j(q)$ to mean the set of nodes common to both $P_i(q)$ and $P_j(q)$.

Theorem 1: In Algorithm A, $P_i(q) \cap P_j(q) = \emptyset$, for all nodes q and $0 \le i, j \le n - 1$, $i \ne j$.

Proof: Suppose not. Then, there exist q, j, and k such that $P_j(q) \cap P_k(q) \ne \emptyset$. Without loss of generality, one can assume that $j = 0$. Let $p \in P_i(q) \cap P_j(q)$. Let r and t be the step number in which p receives a copy of the message in R_0 and R_k, respectively. Then, it follows from 1) all directions in R_0 are distinct, and 2) d_0 is the final direction in R_0 that $d_0 \in H(s, p)$. Using similar reasoning on R_k we get $d_k \in H(s, p)$. It also follows from the algorithm that $H(s, p) \subseteq \{d_i : 0 \le i \le r\}$ and $H(s, p) \subseteq \{d_i : k \le i \le [k + t]_n\}$. Combining these two facts with $d_0 \in H(s, p)$ and $d_k \in H(s, p)$ we get $r \ge k$ and $t \ge n - k$.

Now consider the path from p to q. Again it follows from the algorithm that

$$H(p, q) \subseteq \{d_i : r+1 \le i \le n-1\}$$
$$\subseteq \{d_i : k+1 \le i \le n-1\}$$

and

$$H(p, q) \subseteq \{d_i : [k+t+1]_n \le i \le [k+n-1]_n\}$$
$$\subseteq \{d_i : 1 \le i \le k-1\}.$$

This is impossible since $\{d_i : k + 1 \le i \le n - 1\} \cap \{d_i : 1 \le i \le k - 1\} = \emptyset$. □

B. The Reception Mechanism

As mentioned earlier, the reception mechanism is strongly dependent on the fault model used. In this section, we will consider the reception mechanisms for different fault models. Since the identity of the faulty component is not known, it is not possible to distinguish between node and link faults. So in the rest of this discussion we will treat them to be equivalent.

First, let us consider the simple omission faults, i.e., a faulty node either sends the message correctly or does not send any message at all. Note that this could have been a result of either a faulty processor or a faulty link at that node. Since copies that arrive are not corrupted, the original message can be identified from any one of the received copies. The broadcast is therefore guaranteed to satisfy condition C1, if there are fewer than n faults. Hence, in the case of simple omission fault model, the maximum number of faults that Algorithm A can tolerate is $n - 1$.

The reception mechanism is more complicated if the faulty nodes can corrupt the messages passing through them. Consider the case when the faulty nodes do not corrupt the messages maliciously, i.e., non-Byzantine faults [4]. In this case, the original message can be identified from the received copies by using simple majority voting, i.e., the information in a received message is considered correct if there are $\lceil n/2 \rceil$ copies of that information. However, since all the copies do not arrive at the same time,[1] majority voting is not as simple as in the tightly synchronous situation. The receiving processes can assume the broadcast is complete and perform the necessary voting only if they can establish a quorum. There are two alternative ways of establishing a quorum [8]. A simple way of establishing a quorum is to wait until $\lceil 2n/3 \rceil$ copies of the broadcast message arrive before voting on the information contained in them. Since copies passing through faulty nodes may not arrive at all, $\lceil 2n/$

[1] Some copies may not arrive at all.

$3\rceil$ or more copies will arrive only if there are fewer than or equal to $\lfloor n/3 \rfloor$ faults in the system. Therefore, with this approach for establishing quorum, Algorithm A can tolerate a maximum of $\lfloor n/3 \rfloor$ faults.

An alternative approach for establishing a quorum is to maintain a count of identical copies received. This approach can tolerate more faults than the first approach but at the cost of additional overhead for determining the establishment of a quorum. A quorum is established when there are at least $\lceil n/2 \rceil$ identical messages. As a result, with this approach, Algorithm A can tolerate a maximum of $\lfloor n/2 \rfloor$ faults.

Finally, the worst situation is when the faulty processors can exhibit Byzantine behavior, i.e., they can behave in any arbitrary manner including omitting, corrupting, rerouting, and even lying [4]. This case is similar to the non-Byzantine case if we are interested in satisfying only condition C1. However, if we are interested in using the proposed algorithm for broadcasting in distributed agreement or clock synchronization algorithms, then we can tolerate a maximum of $\lfloor n/3 \rfloor$ faults [3], [4], [9], [10].

It should be noted that irrespective of the behavior of the faulty processors, one can tolerate a maximum of $n - 1$ faults if each message is authenticated by an unforgeable digital signature [6]. As shown above, the number of faults that Algorithm A can tolerate and the receiving mechanism depends on the fault model used.

IV. Performance of Algorithm A

In this section, we will evaluate the performance of Algorithm A in terms of the number of steps required to complete the delivery mechanism. It is shown in Theorems 2 and 3 below that the delivery of the multiple copies can be completed in either $2n$ or $n + 1$ steps depending on the communication capability of each node.

If each node can send a message through at most one outgoing link at a time, then the algorithm requires a total of $2n$ steps. The node initiating the broadcast sends the message to all its neighbors in the first n steps. The neighbors start their coordinated recursive doubling immediately after receiving the message. The last neighbor to receive the message uses the last n steps for its recursive doubling. To prove that it is feasible to complete the algorithm in $2n$ steps, we have to prove that in every step each node has at most one message to send out. Each node will have at most one message to send only if the node initiating the broadcast (say s) sends the message to its neighbors in the following order: $\oplus_0(s)$, $\oplus_1(s)$, $\cdots$, $\oplus_{n-1}(s)$. This result is proved in the following theorem.

Theorem 2: Let s be the node initiating the broadcast. If 1) each node can use a maximum of one outgoing link at a time, 2) s sends the message to its neighbors in the following order: $\oplus_0(s)$, $\oplus_1(s)$, $\cdots$, $\oplus_{n-1}(s)$, and 3) the neighbors use coordinated recursive doubling as per Algorithm A immediately upon receiving the message from s, then all nodes will receive the n copies of the message in $2n$ steps.

Proof: Suppose not. Then, there exists a step k in which a node p has to send out more than one message. Without loss of generality we can assume that the messages are from the recursive doubling initiated by nodes $\oplus_0(s)$ and $\oplus_i(s)$. Clearly, $k > i$ because otherwise node $\oplus_i(s)$ would not have initiated its recursive doubling. By arguments similar to that in Theorem 1, it follows that $d_0 \in H(s, p)$. Since in R_i a message is sent in d_0 only in its $(n - i)$th step, $k \geq (n - i) + i = n$. But at step $k \geq n$, R_0 is already complete. Hence, p will not have a message from R_0 to send out. This contradicts our initial assumption. □

In contrast, if a node can send messages through all its outgoing links and also receive from all its incoming links simultaneously, then Algorithm A requires only $n + 1$ steps. In the first step, the source node s sends the message to all its neighbors. In the next n steps, the neighbors use the recursive doubling in Algorithm A to deliver the message to all nodes. The following theorem proves that there is no contention for the same link at any node during the entire course of the algorithm.

Theorem 3: Let s be the node initiating the broadcast. If a node can receive and send messages simultaneously in all its incoming and outgoing links, respectively, then Algorithm A requires $n + 1$ steps.

Proof: Suppose not. Then, there exists a step k in which a node p that has to send more than one message in the same direction, say j. Without loss of generality, we can assume that the messages are from the recursive doubling initiated by nodes $\oplus_0(s)$ and $\oplus_i(s)$. This implies in step $k - 1$ of recursive doubling both nodes $\oplus_0(s)$ and $\oplus_i(s)$ send message in the same direction. Contradiction. □

V. Conclusion

A simple algorithm was proposed for reliable broadcast in a hypercube. The algorithm is particularly useful in critical real-time systems that cannot tolerate the time overhead of identifying the faulty processors on-line. The maximum number of faults the proposed algorithm can tolerate was determined for different fault models. Finally, for different communication capabilities at each node the performance of the algorithm was determined in terms of number of steps required for completion of the broadcast. The effect of multiple copies on the delivery times is not accounted for in this analysis.

References

[1] M.-S. Chen and K. G. Shin, "Message routing in an injured hypercube," in *Proc. Third Conf. Hypercube Concurrent Comput. Appl.*, to be published.

[2] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1969.

[3] L. Lamport and P. M. Melliar-Smith, "Synchronizing clocks in the presence of faults," *J. ACM*, vol. 32, Jan. 1985.

[4] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Programming Languages Syst.*, vol. 4, pp. 382–401, July 1982.

[5] T. C. Lee and J. P. Hayes, "Routing and broadcasting in faulty hypercube computers," in *Proc. Third Conf. Hypercube Concurrent Comput. Appl.*, to be published.

[6] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM*, vol. 21, pp. 120–126, Feb. 1978.

[7] C. L. Seitz, "The Cosmic cube," *Commun. ACM*, vol. 28, pp. 22–33, Jan. 1985.

[8] K. G. Shin and J. W. Dolter, "Alternative majority voting methods for real-time computing systems," *IEEE Trans. Reliability*, to be published.

[9] T. K. Srikanth and S. Toueg, "Optimal clock synchronization," *J. ACM*, vol. 34, pp. 626–645, July 1987.

[10] ——, "Simulating authenticated broadcasts to derive simple fault-tolerant algorithms," Tech. Rep. 84-623, Dep. Comput. Sci., Cornell Univ., July 1984.

Concurrent Access of Priority Queues

V. NAGESHWARA RAO AND VIPIN KUMAR

Abstract—**The heap is an important data structure used as a priority queue in a wide variety of parallel algorithms (e.g., multiprocessor scheduling, branch-and-bound). In these algorithms, contention for the shared heap limits the obtainable speedup. This paper presents an**

Manuscript received February 25, 1988; revised July 12, 1988. This work was supported by Army Research Office Grant DAA 29-84-K-0060 to the Artificial Intelligence Laboratory, and Office of Naval Research Grant N00014-86-K-0763 to the Department of Computer Science, University of Texas, Austin.

The authors are with the Department of Computer Science, University of Texas, Austin, TX 78712.

IEEE Log Number 8824093.