Optimal Subcube Fault Tolerance in a Circuit-Switched Hypercube

Baback A. Izadi
Dept. of Elect. Eng. Tech.
DeVry Institute of Technology
Columbus, OH 43209
bai@devrycols.edu

Füsun Özgüner
Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210-1277
ozguner@ee.eng.ohio-state.edu

Abstract

In this paper, by utilizing the circuit-switched communication modules of the hypercube nodes, we present a scheme where a $(d-1)$-dimensional subcube is allocated in a faulty d-dimensional hypercube in the presence of up to $2^{(d-1)}$ faulty nodes. The scheme is then extended to allocate a $(d-1)$-dimensional subcube in the presence of a combination of faulty nodes and faulty links. Theoretical and simulation results are presented to analyze the performance of the scheme.

1. Introduction

Multiprocessors based on the hypercube interconnection topology are being widely used for a range of scientific and real-time applications. Numerous research efforts have been undertaken to make the hypercube functional in the presence of faulty components. One of the approaches is finding the maximum dimensional fault-free subcube of a faulty hypercube [1, 8]. However, two faulty nodes in antipodal positions destroy every fault-free $(d-1)$-dimensional subcube and thus degrade the performance of the hypercube by a factor of 4. To overcome this, Chang and Bhuyan [2] have proposed a scheme that utilizes the properties of circuit-switched communication to maintain a $(d-1)$-dimensional subcube in the presence of up to $\lceil \frac{d}{2} \rceil$ faulty nodes. The procedure first constructs a $(d-1)$-dimensional subcube with the least number of faulty nodes. The faulty nodes in the selected $(d-1)$-dimensional subcube are then replaced by the healthy nodes in the other subcube. The reconfiguration can fail if any of the nodes which are two or fewer hops away from a faulty node of the selected subcube is faulty as well.

In this paper, we present a scheme for circuit-switched hypercubes where a $(d-1)$-dimensional subcube is allocated in a faulty d-dimensional hypercube in the presence of up to $2^{(d-1)}$ faulty nodes. Our scheme does not impose any restriction on the distribution of faulty nodes. We then extend the scheme so that a $(d-1)$-dimensional subcube is allocated in the presence of a combination of faulty nodes and faulty links.

The rest of the paper is organized as follows. An overview of our scheme under only faulty nodes is presented in the next section. An optimal and a near optimal reconfiguration algorithm is described in Section 3. The proof of our theoretical bound is given in Section 4. In Section 5, construction of a fault-free $(d-1)$-dimensional subcube under only faulty links and combination of faulty nodes and faulty links is discussed and simulation results are presented. Finally, concluding remarks are given in Section 6.

2. Construction of a Functional $(d-1)$-Dimensional Subcube

Figure 1 depicts the architecture of a node in a 4-dimensional hypercube. Our scheme is based on the fact that in hypercubes with circuit-switched communication modules [4], the communication time is nearly constant between any two given nodes. We assume that faulty nodes retain their ability to communicate. This is a common assumption since in hypercube multiprocessors such as the *iPSC/860* [7], the computation and the communication modules of a node are separate. Furthermore, since the complexity of the computation unit is much greater than that of the communication one, the probability of a failure in the computation unit is much higher. This assumption may be ignored by duplicating the communication module of each node. If the distribution of faulty nodes is such that every fault-free $(d-1)$-dimensional subcube is destroyed [8], we use the following procedure to construct a functional $(d-1)$-subcube.

Partition the hypercube into two faulty $(d-1)$-dimensional subcubes along a dimension j $(0 \leq j \leq d-1)$. Moreover, label the subcube with the lower number of faulty nodes *Sub1* and the other subcube *Sub2*. Although j can be chosen from any of the d coordinates, selecting dimension

1063-7133/96 $5.00 © 1996 IEEE
Proceedings of IPPS '96

Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)
1063-7133/96 $10.00 © 1996 IEEE

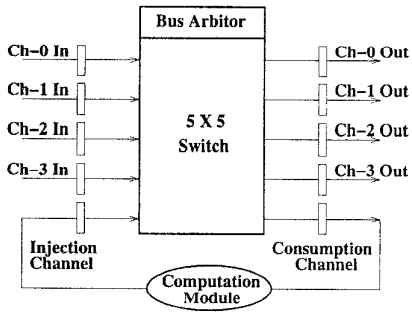

Figure 1. The architecture of a node in a 4-dimensional circuit-switched hypercube

j such that *Sub1* contains the least number of faulty nodes normally results in a faster reconfiguration. Let f_1 and h_1 respectively be the number of faulty and healthy nodes in *Sub1*. Similarly, let f_2 and h_2 be the number of faulty and healthy nodes in *Sub2*. Since $f_1 + h_1 = f_2 + h_2 = 2^{(d-1)}$ and $f_1 + f_2 \leq 2^{(d-1)}$, then $f_1 \leq 2^{(d-1)} - f_2$ or $f_1 \leq h_2$; the number of healthy nodes in *Sub2* is greater than or equal to the number of faulty nodes in *Sub1*.

Note that nodes of *Sub1* and *Sub2* form a $2^{(d-1)}$ matching along dimension j. In our scheme, *Sub1* becomes a functional $(d-1)$-dimensional subcube by replacing each of its faulty nodes with a healthy node in *Sub2* via edge-disjoint paths along dimension j and other possible edges in *Sub2*. Any healthy node that replaces a faulty node assumes its address label. Figure 2 illustrates a case in a 4-dimensional hypercube where nodes 0000, 0001, 0101, 0110, 1001, 1010, 1011, and 1110 are faulty. Setting $j = 3$, the subcube $0xxx$ is maintained by replacing the faulty nodes 0000, 0001, 0101, and 0110 with the healthy nodes 1000, 1100, 1101, and 1111 respectively. The edge-disjoint paths between the healthy nodes in *Sub2* and the faulty nodes in *Sub1* are shown in the figure with bold solid and dashed lines. These paths are established during the reconfiguration and remain intact thereafter. Each of the edge-disjoint paths then becomes an extension of the communication module of the faulty node in *Sub1*. Figure 3 illustrates the construction of edge-disjoint paths associated with the faulty nodes 0000 and 0001 of Figure 2. The bold solid and dashed lines in Figure 3 correspond to the similar lines in Figure 2.

If a message is to originate from a faulty node in *Sub1*, its healthy replacement in *Sub2* sends the data to the injection channel as normal. Using the established path, the message reaches the communication module (*CM*) of the faulty node in *Sub1*; the CM of the faulty node receives the data via input channel j (*Ch-j In*). The appropriate channel out is then used per normal operation. As an example in Figures 2 and 3, if an algorithm requires node 0000 to send a message to node 0100, node 1000 (replacement of node 0000) sends the message along dimension 3 to the CM of node 0000. For

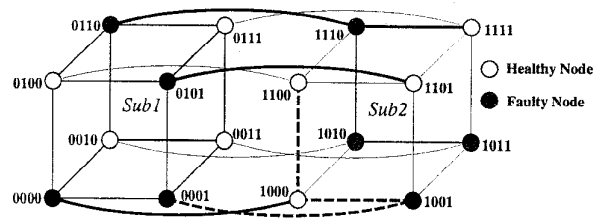

Figure 2. Construction of a 3-cube in a faulty 4-cube

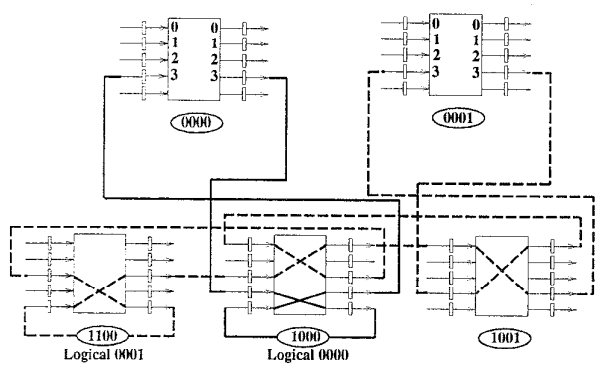

Figure 3. Replacing faulty nodes 0000 and 0001 with healthy nodes 1000 and 1100

e-cube routing, the CM of node 0000 then uses its channel 2 to forward the message to node 0100.

If the destination of a message is a faulty node in *Sub1*, the CM of that faulty node, instead of sending the message out on its consumption channel, routes it automatically to output channel j. Consequently, CM of its healthy replacement receives the message and forwards it automatically to its computation module via the consumption channel. Note that some nodes in *Sub2* are used as intermediate switch connections. For example in Figures 2 and 3, node 1000 is used to connect channel 0 to channel 2 and at the same time link channel 3 to the local computation module. Therefore, the regular nodes in *Sub2* have dual functions: one is to be logical replacements for the faulty nodes in *Sub1* and the other is to function as permanent intermediate switch connections. Both functions may be active at the same time. The faulty nodes in *Sub2* may only be used as permanent intermediate switch connections.

3. The Reconfiguration Algorithm

A reconfiguration algorithm establishes edge-disjoint paths between faulty nodes in *Sub1* and healthy nodes in *Sub2*. To illustrate our reconfiguration algorithm, the following sets are used. Given a faulty node $\alpha \in$ *Sub1*, let's denote the node across its dimension j as $\beta \in$ *Sub2*. If β is a healthy node, it can replace α as discussed before. We

Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)
1063-7133/96 $10.00 © 1996 IEEE

then label β as a *used* node and denote the set of used nodes as S_U. If β is a faulty node, a dedicated path from β to an *unused* healthy node in *Sub2* needs to be established. β and the healthy node are then referred to as the *source* and the *target* nodes respectively. We refer to the set of source nodes in *Sub2* as S_S. The set of unused healthy nodes in *Sub2* represents potential target nodes and are labeled as S_T. The remaining nodes in *Sub2* consist of faulty nodes. Since edge-disjoint paths can be established through them, they are also marked as used nodes and are assigned to S_U. Therefore a node in *Sub2* is a source node if both the node and its neighbor in *Sub1* are faulty. Allocated healthy nodes and remaining faulty nodes in *Sub2* are called used nodes. Finally, non-allocated healthy nodes in *Sub2* are labeled target nodes. For example, in Figure 2, S_S={1001, 1110}, S_T={1100, 1111}, and S_U={1000, 1010, 1011, 1101}. Then, a reconfiguration algorithm first has to connect the faulty nodes in *Sub1* to the respective nodes along dimension j in *Sub2*. Subsequently, it needs to establish edge-disjoint paths between the nodes in S_S and the nodes in S_T within *Sub2*.

3.1. An Optimal Algorithm

An optimal reconfiguration algorithm can be developed by utilizing the maxflow/mincut algorithm [9]. Here, optimality is measured as the ability to assign a healthy node in *Sub2* to every faulty node in *Sub1* whenever such an assignment is feasible vis-a-vis Menger's theorem [3]. To apply the maxflow/mincut algorithm, a digraph G' needs to be constructed as specified below. First a digraph representation of *Sub2* (digraph G) is constructed by replacing each link of *Sub2* with two parallel and oppositely directed links. Next, two super-nodes s and t are added and are connected to each node in S_S and S_T via a single directed link respectively. The flow capacity of each link in G' is set to 1. Figure 4 depicts the construction of G' for the example in Figure 2. By Menger's theorem, there exists sufficient number of edge-disjoint paths between s and t to make the reconfiguration feasible provided the number of nodes in S_S is smaller than or equal to the size of the mincut of G'. In Section 4, we will prove that this condition always holds. Since each of the edge-disjoint paths passes through a unique source node and a unique target node, each node in S_S can be connected to a unique healthy node in S_T. The maxflow/mincut algorithm can then construct these paths and hence assign a healthy node in S_T to each node in S_S.

3.2. A Near-Optimal Algorithm

The main drawback to reconfiguration using the above algorithm is that a new graph G' has to be constructed and the target to source assignments have to be done by the host processor. To reduce the reconfiguration time, we next present a near optimal reconfiguration algorithm which can be implemented in a distributed manner. The algorithm is near optimal since there can be cases where reconfiguration fails even though Menger's theorem holds. To find a set of candidate target nodes that can be assigned to a source node, we utilize Lee's path-finding algorithm[6]. The algorithm begins by constructing a breadth-first search of minimum depth in *Sub2* from each node in S_S. If a target node is found, a path is formed between the source and the target node. The algorithm guarantees that a path to a target node will be found, if there exists one, and the path will be the shortest possible [6]. Therefore, all target nodes that are one link away from the source nodes (at distance 1) are assigned first. Once a path is formed, the algorithm removes the links associated with that path from *Sub2*. It also marks both the source node and the target node as used nodes and assigns them to S_U. The process is repeated on the new resultant structure for a higher depth i. The reconfiguration is completed if S_S becomes an empty set. Reconfiguration fails if distance i becomes greater than $2^{(d-1)} - 1$, which is the longest acyclic path in *Sub2*.

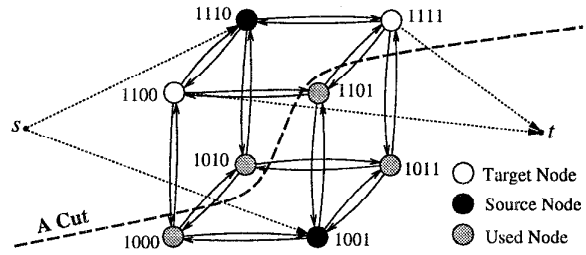

Figure 4. Construction of G' for *Sub2*

We implemented the near optimal algorithm using the C language for various dimensions of the hypercube (up to d = 10). 10000 simulation runs were performed for randomly placed $2^{(d-1)}$ faulty nodes. Our simulations resulted in 100% reconfiguration.

4. Correctness of Our Theoretical Bounds

In this section, we show that a functional $(d-1)$-dimensional subcube can be found in a hypercube with up to $2^{(d-1)}$ faulty nodes; we assume static faults and no faulty links. If every faulty node in *Sub1* is matched with a healthy node in *Sub2* along dimension j, the reconfiguration can be accomplished by simply assigning the matched healthy nodes to the faulty nodes. In the worst case, each faulty node in *Sub1* is matched with a faulty node in *Sub2* along dimension j. Furthermore, there exists maximum number of faulty nodes; *Sub1* and *Sub2* each contain $2^{(d-2)}$ faulty nodes. Therefore, $2^{(d-2)}$ edge-disjoint paths from the faulty nodes in *Sub2* to the healthy nodes in *Sub2* have to be constructed. Other cases where the number of faulty nodes

Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)
1063-7133/96 $10.00 © 1996 IEEE

in the hypercube is less than $2^{(d-1)}$ or the number of faulty nodes in *Sub1* is less than the number of faulty nodes in *Sub2* or some of the faulty nodes in *Sub1* are matched with healthy nodes in *Sub2* along dimension j are covered by the above case; they require fewer number of edge-disjoint paths from the nodes in *Sub1* to the nodes in *Sub2*. As an example, the case depicted in Figure 4 only requires two edge-disjoint paths. If 2 of the used nodes in Figure 4 are relabeled as source nodes and the remaining used nodes are relabeled as target nodes, the required number of edge-disjoint paths becomes 4. Obviously, the latter case covers the former one.

To examine whether there exists $2^{(d-2)}$ edge-disjoint paths between the faulty nodes of *Sub2* and the healthy nodes of *Sub2*, the undirected version of the digraph G' (Figure 4) has to be constructed. We will next show that each node in S_S can be connected to a unique node in S_T via a dedicated path in *Sub2*.

Lemma 1 *In a d-dimensional hypercube, let a set of nodes be assigned to P and the rest be grouped under Q. The minimum number of links that connects P to Q is* $min(|P|, |Q|)$.

Proof: The proof is omitted due to space limitation [5]. ■

Theorem 1 *In a d-dimensional hypercube, let half of the nodes be randomly labeled source nodes and the remaining nodes be called target nodes. Within such a hypercube, there exists edge-disjoint paths connecting each of the* $2^{(d-1)}$ *source nodes to a distinct target node.*

Proof: Given a hypercube graph $G(V, E)$, let's partition V into two subsets P and $Q = V - P$ such that $P = S_S$ and $Q = S_T$ ($|P| = |Q| = 2^{(d-1)}$). Moreover, let's construct a new graph G' by adding two nodes s and t to G such that they be connected to every node in the set P and Q respectively (Figure 5). The number of edge-disjoint paths between s and t in G', according to Menger's theorem [3], is equal to the mincut of G'. The theorem is proven by showing that there always exists an (s, t) mincut in G' whose cutsize is greater than or equal to $2^{(d-1)}$.

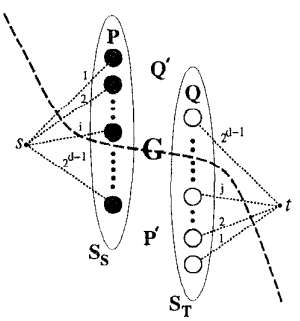

Figure 5. An (s, t) cut in G'

An (s, t) mincut in G' may exist at s, t, G, or some combination of them. By construction, the cutsize at s and t is equal to $2^{(d-1)}$. Consider a general cut in G' as depicted in Figure 5, crossing i of the edges connecting s to the nodes within S_S, k of the edges of G, and j of the edges connecting the nodes of S_T to t ($0 \leq i, j \leq 2^{(d-1)}$). Only the case where $i+j < 2^{(d-1)}$ needs to be investigated since the other meets the minimum cutsize on its own. The cut in Figure 5 splits the nodes into two sets P' and Q'. The number of source nodes in the set Q' is i. The number of target nodes in the same set is $2^{(d-1)} - j$. Hence, the total number of source and target nodes in Q' is $i+2^{(d-1)}-j$. Following the same reasoning, the total number of source and target nodes in P' is $j + 2^{(d-1)} - i$. Either $|P'|$ or $|Q'|$ is less than or equal to $2^{(d-1)}$. Without loss of generality, let it be P'. From Lemma 1, the minimum number of links connecting P' and Q' is $k = min(|P'|, |Q'|)$. Thus, the above cut must cross at least $k = j + 2^{(d-1)} - i$ links in G. The size of the (P', Q') cut is then given by $|(P', Q')| \geq i + j + j + 2^{(d-1)} - i$ or $|(P', Q')| \geq 2j + 2^{(d-1)} \geq 2^{(d-1)}$. A similar inequality results if $|Q'| < 2^{(d-1)}$, $|(P', Q')| \geq 2i + 2^{(d-1)} \geq 2^{(d-1)}$.

From the above inequalities it follows that there always exists $2^{(d-1)}$ edge-disjoint paths from s to t. Since there always exists $2^{(d-1)}$ edges from s to $2^{(d-1)}$ source nodes and $2^{(d-1)}$ edges from t to $2^{(d-1)}$ target nodes, each of the $2^{(d-1)}$ paths must connect a source node to a target node. Therefore, there exists edge-disjoint paths connecting each of the $2^{(d-1)}$ source nodes to a distinct target node. ■

5. A $(d-1)$-Subcube in Presence of Faulty Nodes and Faulty Links

A link is specified uniquely by *d-tuple*$\{0, 1, -\}^d$ where "-" can be substituted by "0" and "1" to identify its connecting nodes. Reconfiguration in the presence of faulty links is accomplished by searching for a subcube with all of its nodes and links intact. There exists $2d$ subcubes of dimension $d-1$ in a d-dimensional hypercube [8]. Each faulty link destroys $d-1$ of these subcubes. A $(d-1)$-dimensional fault-free subcube can be found in a d-dimensional hypercube by searching for a bit position that does not contain both 0 and 1 (ignoring "-") in the d-tuples representing the faulty links. Once such a bit is found, the d-tuple obtained by complementing the value of the bound bit position and assigning x's to the remaining $d-1$ coordinate positions defines a fault-free $(d-1)$-dimensional subcube. For example, consider the faulty links -010, 1-00, 10-1, and 111- in a 4-dimensional hypercube. Only the bit values of dimension 3 do not contain both 0 and 1. Therefore, the subcube $0xxx$ is the only fault-free 3-dimensional subcube. We implemented the above procedure in the C language for a hypercube of dimension 10. The simulation result for randomly placed faulty links is shown in Figure 6.

Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)
1063-7133/96 $10.00 © 1996 IEEE

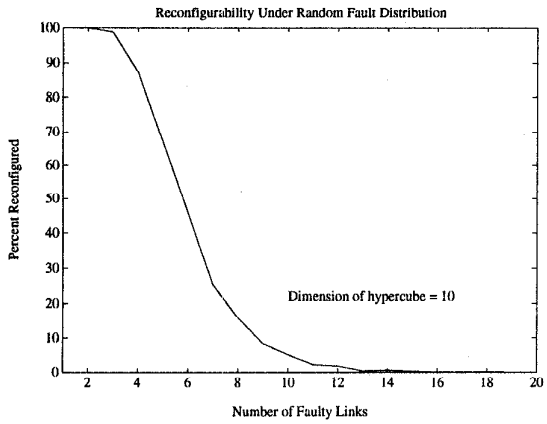

Figure 6. Percentage of sustained $(d-1)$-subcubes in the presence of faulty links

To find a fault-free $(d-1)$-dimensional subcube in the presence of a combination of faulty nodes and faulty links, our algorithm first uses the above procedure to check whether a $(d-1)$-dimensional subcube without any link failure exists. If there exists one, it is labeled *Sub1*. Next, Lee's path-finding algorithm is used to find edge-disjoint paths from the faulty nodes in *Sub1* to the healthy nodes in *Sub2* via the healthy edges from *Sub1* to *Sub2* and the healthy edges within *Sub2*. Figure 7 demonstrates a case where nodes 0000, 0001, 0101, 0110, 1001, 1010, 1011, 1110 and links -010, 1-00, 10-1, 111- are faulty. The subcube $0xxx$ remains functional in the presence of faulty elements.

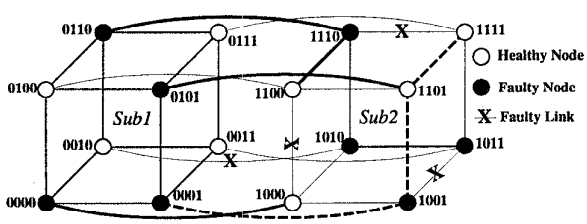

Figure 7. A functional 3-cube in a faulty 4-cube

The simulation result for randomly placed faulty nodes and faulty links is given in Figure 8. In our simulation we have assumed that the probability of having a node failure is the same as having a link failure. From Figures 6 and 8, it follows that a functional $(d-1)$-dimensional subcube can exist provided the number of faulty links are relatively small.

6. Concluding Remarks

In this paper we presented a scheme where a functional $(d-1)$-dimensional subcube can be allocated in the presence

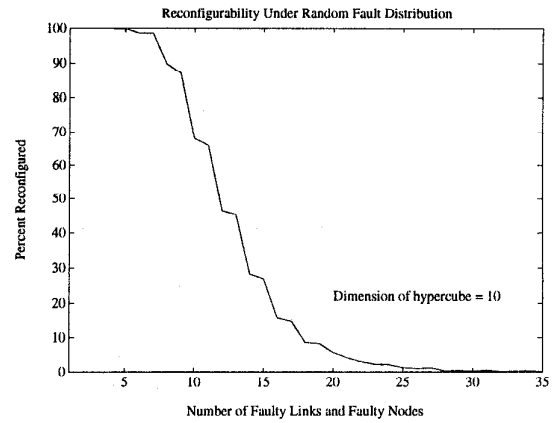

Figure 8. Sustained $(d-1)$-subcubes in the presence of faulty nodes and links

of up to $2^{(d-1)}$ faulty nodes. Moreover, we showed that a functional $(d-1)$-subcube may be allocated in the presence of a combination of faulty nodes and faulty links. To increase the number of tolerated faulty links, the requirement that *Sub1* be free of faulty links needs to be relaxed. This may be done by having each faulty link in *Sub1* bypassed by a parallel path consisting of healthy links between *Sub1* and *Sub2*, and healthy links within *Sub2*. For example, in Figure 7, if link 0-11 is faulty as well, it can be replaced by the parallel path made of links -011, 1-11, and -111.

References

[1] B. Becker and H. U. Simon. How robust is the n-cube? In *Proc. 27th Annu. Symp. Foundations Comput. Sci.*, pages 283–291, October 1986.

[2] Y. Chang and L. N. Bhuyan. Subcube fault tolerance in hypercube multiprocessors. *IEEE Transactions on Computers*, 44:1108–1120, September 1995.

[3] C. J. Colbourn. *The Combinatorics of Network Reliability*. Oxford University Press, 1987.

[4] P. T. Gaughan and S. Yalamanchili. Adaptive routing protocols for hypercube interconnection networks. *IEEE Computer*, pages 12–23, May 1993.

[5] B. Izadi. Design of fault-tolerant distributed memory multiprocessors. *Ph.D. thesis, the Ohio State University*, 1995.

[6] C. Y. Lee. An algorithm for path connection and its applications. *IRE Transaction on Electronic Computers*, ec-10:346–365, 1961.

[7] S. Nugent. The *iPSC/2* direct-connect communication technology. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 51–60, January 1988.

[8] F. Özgüner and C. Aykanat. A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor. *Information Processing Letters*, 29(5):247–254, November 1988.

[9] A. Tucker. *Applied Combinatorics 2nd ed.* Wiley, 1984.

Proceedings of the 10th International Parallel Processing Symposium (IPPS '96)
1063-7133/96 $10.00 © 1996 IEEE