Spare Allocation and Reconfiguration in a Fault Tolerant Hypercube with Direct Connect Capability

Baback A. Izadi*and Füsun Özgüner
Department of Electrical Engineering
The Ohio State University
Columbus, OH 43210

Abstract

This paper investigates hardware reconfiguration schemes to make the hypercube multicomputer fault tolerant. Two schemes are proposed; the Cluster Approach and the Enhanced Cluster Approach. The approaches are shown to be able to tolerate large number of failures without any performance degradation. It is further demonstrated that no modification to either the existing communication or computational algorithm is needed. Finally a gracefully degradable approach is presented to reconfigure when the number of faulty nodes are more than the available spares.

1 Introduction

For fault tolerance in hypercube multiprocessors, two fundamental approaches have been proposed. Some researchers have examined the software approach [1, 2, 3] with no added hardware and some performance degradation. Others have looked into hardware schemes where spare nodes and/or links are used to replace the faulty ones [4, 5, 6, 3, 7, 8, 9].

This paper investigates hardware schemes that allocate spares in a hypercube multiprocessor to make it fault tolerant. A hardware scheme was first proposed by Rennels [4]. Chau and Liestman [5] have employed a decoupling network to connect modules of active and spare processors. Several approaches have been proposed in [6, 3, 8] that perform reconfiguration using spare nodes and spare links. An augmented approach was illustrated by Witkowski and Lee [7]. Alam and Melhem [9] using hardware redundancy, developed augmented approaches to tolerate faulty nodes.

The schemes presented in this paper tolerate a large number of faults without any performance degradation and the resultant configuration does not affect either the communication or computational algorithms already developed for the hypercube multiprocessor. The allocation of spares is facilitated by using a routing element on each node similar in concept to the *Direct Connect Module* [10] of *Intel* hypercubes.

Each node of a d dimensional hypercube is represented by *d-tuple* $(b_{d-1} \cdots b_i \cdots b_0)$, where $b_i \in \{0,1\}$. A subcube is defined by a unique *d-tuple*$\{0,1,X\}^d$ where "0" and "1" are the *bound* coordinates, and "X" represents the *free* coordinates. A *(d-k)*-dimensional subcube is represented by a *d-tuple* with *k bound* coordinates and *d-k free* coordinates. Each spare is uniquely defined by *d-tuple*$\{0,1,S\}^d$ where "0" and "1" represent the bound coordinates and S corresponds to a free coordinate of its assigned cluster. A link is specified uniquely by *d-tuple*$\{0,1,-\}^d$ where "$-$" can be substituted by "0" and "1" to identify its connecting nodes. An intra-cluster spare link is defined by *d-tuple* $\{0,1,S,-\}^d$ where S signifies spare and "$-$" can be substituted by "0" and "1" to identify its connecting spare nodes. An inter-cluster spare link is defined by $(d+1)$*-tuple* $\{0,1,S\}^{(d+1)}$ where S signifies spare and ("0", "1") are the *bound* coordinates of the regular node that its respective spare node is connected to.

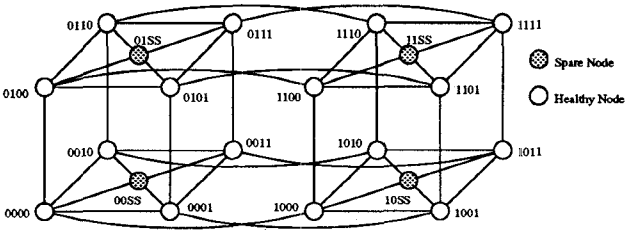

Figure 1: Hypercube of dimension 4 with clusters of dimension 2

2 Cluster Approach

An i dimensional cluster is merely a subcube of dimension i. Thus, a hypercube of dimension d can be divided into 2^j clusters where $d = i+j$. A single spare is assigned per cluster of nodes. Figure 1 depicts a hypercube of dimension 4 with clusters of dimension 2. In this scheme, the spare may logically replace the faulty node within its designated cluster. For example, in figure 1, spare $01SS$ may logically replace any one of nodes $0100, 0110, 0111, 0101$ using spare links $01S00, 01S10, 01S11, 01S01$ respectively.

An important factor influencing the selection of the cluster size is the mean time to failure of a node. Re-

*Also with the Devry Institute of Technology, Columbus, Ohio 43209.

0-8186-2290-3/91/0000/0711/$01.00 © 1991 IEEE

liability studies [7] with the available components have shown that to attain 95 percent reliability in 10 years, approximately 25 percent spare overhead is needed. Therefore a cluster dimension of 2 should be acceptable.

It is assumed that a given faulty node retains its communication capability. This is a fair assumption since in hypercube multiprocessors such as the *iPSC/2*, 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.

In hypercubes with *Direct Connect Capability* [10, 11], the cost of communication is nearly constant between any two given nodes. To facilitate reconfiguration, the routing elements described below are used at each node. Figure 2 depicts the block diagram of the router for the regular and the spare nodes. The regular node router is an enhanced version of *Intel's Direct Connect Module (DCM)* with an additional channel routing element provided to connect a given node to its respective spare. The spare node router is a simple switch which is used to either link two of the regular nodes within a cluster together or to connect the spare to one of its cluster nodes. The former is used to bypass the faulty link by connecting the two appropriate channel routing elements together. For example in figure 2, connecting the routing elements of nodes 00 and 01 (linking *Node 00 In* to *Node 01 out* and *Node 01 In* to *Node 00 out*) can provide an alternative path to 010– in subcube $01XX$ of figure 1. The *Spare In* , *Spare Out* of each router of the nodes 0100 and 0101 are then connected to the *Node 00 Out, Node 00 In* and *Node 01 Out, Node 01 In* of the spare router. The latter, connecting the spare to one of its cluster nodes, is done to tolerate a node failure. This is accomplished by connecting the *Spare Routing Element* of the faulty node (figure 2(a)) to the appropriate *Node Routing Unit* of the spare node. Three cases exist which may involve routing data through the faulty node. The first one is when the message originates from the faulty node. Here the spare sends its data via the serializer (figure 2) out to the appropriate *Node XX Out* channel. The message is then routed to the *DCM* of the faulty node via the *Spare In*. Depending on the final destination node, any of *Channel Outs* may be selected. The second case is when the destination of the message is the faulty node. Once the data reaches the *DCM* of the faulty node, it is automatically routed to the channel *Spare Out* instead of channel *To Node*. The spare's *DCM* would consequently receive the message using the appropriate channel (*Node XX In*). It then deserializes the message and sends it to the spare. The third case is when the faulty node is neither the source nor the destination of the message but is used merely as an intermediate hop. In this case, the spare is not involved at all, and routing is done normally. Note that the connection of the spare router to the active node(s) is established after the reconfiguration and remains intact thereafter.

Upon detecting a node failure, the spare within the re-

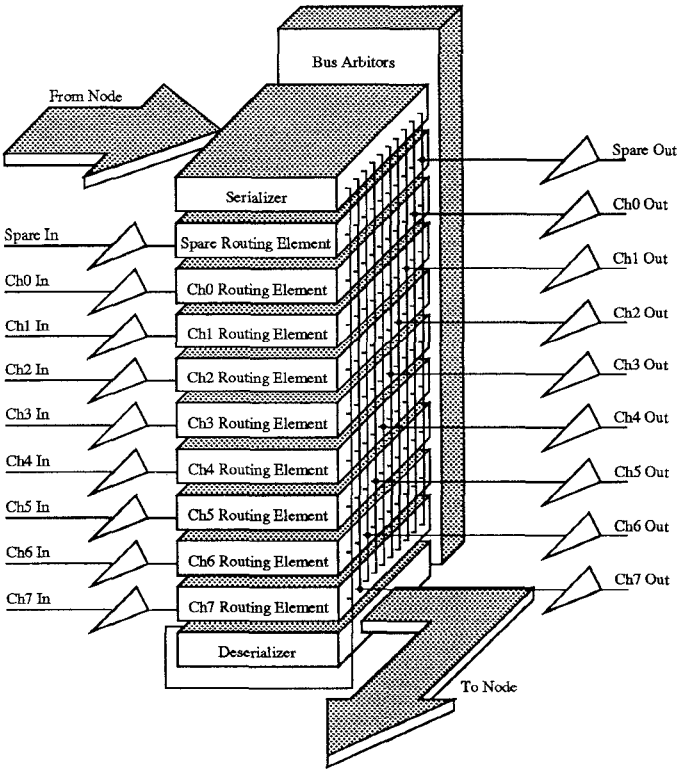

Figure 2: Communication Module Architecture for (a) Regular Node (b) Spare Node

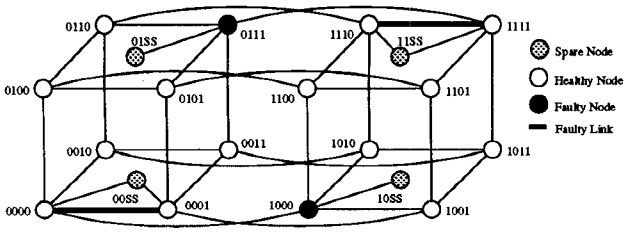

Figure 3: Reconfiguration of a faulty hypercube

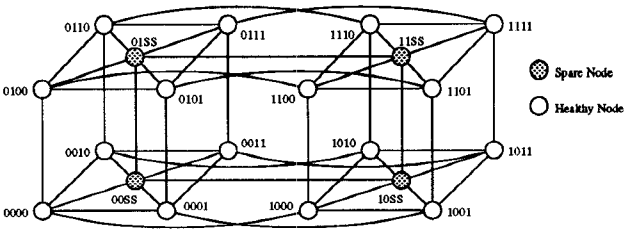

Figure 4: ECH of dimension 4

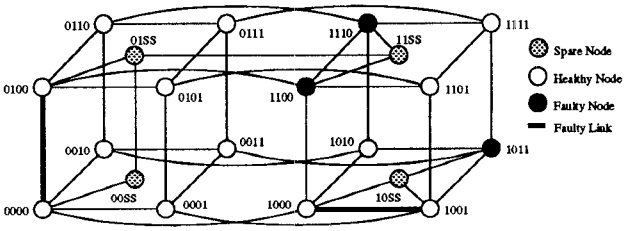

Figure 5: Reconfiguration of ECH

spective cluster logically replaces the faulty node. The spare consequently activates its link to the faulty node and disable the rest of its links. The spare's communication module then forwards/receives its data to/from other nodes via the *DCM* of the faulty node as discussed above.

One intra-cluster link failure per cluster can be tolerated. Upon detection of a link failure, the router of the spare node is used to establish a parallel path to the faulty link. The *Processing Elements* at the two ends of the faulty link would logically replace their *channel routing elements*, which connects them to the faulty link (figure 2), with the *spare channel routing element*. An Inter-cluster link failure is fatal.

Example: Figure 3 illustrates the reconfiguration of hypercube upon detection of faults in nodes 0111, 1000 and links $111-, 000-$. The *Ch0 routing element* of nodes 0000, 0001, 1110, and 1111 are replaced with the *spare channel routing element*. The faulty nodes are replaced by their designated spare as discussed before. If there exists a need to establish a link between node 1111 and 1000, the path ($1111 \rightarrow 11SS \rightarrow 1110 \rightarrow 1100 \rightarrow 1000 \rightarrow 10SS$) is set up. Note that no modification of the communication algorithm is needed.

3 Enhanced Cluster Approach

The *Cluster Approach* has two shortcomings. It does not allow multiple faults within a cluster and it fails to reconfigure upon an inter-cluster link failure. To overcome these deficiencies, in this scheme, the spares are connected in such a way to form a $d-2$ cube. Figure 4 depicts an *Enhanced Cluster Hypercube (ECH)* of dimension 4. The same figure also demonstrates the scalability of this topology. Each regular and spare node is connected to $d+1$ and $4+(d-2) = d+2$ nodes respectively. Intra-cluster and inter-cluster spare links are uniquely defined by $(d+1)$-*tuple* and *d-tuple* respectively. For example, the spare link connecting the spare node $00SS$ and the node 0001 is identified as $00S01$. Also the inter-cluster spare link connecting the spare $01SS$ and $00SS$ is labeled $0-SS$. The block diagram of the router for the regular node is the same as before (figure 2(a)). The router for the spare node needs $d+2$ routing elements instead of 4 elements as shown in figure 2(b). The spares are provided with the *simultaneous multi-channel communication capability*. More specifically, each spare at the same time can connect $\lfloor \frac{d}{2} \rfloor$ pairs of its links together.

The *ECH* can tolerate up to 3 faulty nodes per cluster. Reconfiguration is accomplished using the following algorithm. The algorithm first checks whether the number of spares is sufficient for reconfiguration and if there exists a cluster with all four nodes being faulty. Next, a single faulty cell per faulty cluster of dimension 2 is selected at random and its task is assigned to the respective spare. If all faults are covered, then reconfiguration is accomplished. If there are still faulty nodes which have no spare assignment, the free spares in the next higher dimension are considered. The process continues until all faulty nodes are covered by the spares.

The *Enhanced Cluster Hypercube* can tolerate both intra-cluster and inter-cluster link failures. Parallel path(s) to the faulty link(s) are established. All other unnecessary spare links are disabled. Reconfiguration fails if any of the two nodes at the end of a faulty link is faulty too. It also fails if two faulty links have a common node.

Example: Figure 5 demonstrates the reconfiguration of the *Enhanced Cluster Hypercube* upon detection of faults in links 0-00, 100- and nodes 1011, 1100, 1110. *Simultaneous multi-channel communication capability* of the spare node $10SS$ is used to logically replace node 1011 and at the same time establish a path between nodes 1001 and 1000. Inter-cluster link failure, 0-00, requires the use of two spare nodes $01SS$ and $00SS$ to be replaced with $01S00 \rightarrow 0-SS \rightarrow 00S00$. The spare node $01SS$ logically replaces the faulty node 1100 using the path $-1SS \rightarrow 11S00$. Similarly 11SS replaces 1110 using the link 11S10.

4 Gracefully Degradable Approach

As part of a comprehensive solution to a fault-tolerant hypercube, the possibility of more faulty nodes than available spares needs to be considered. In such a case,

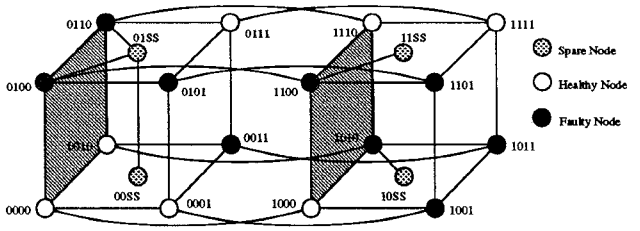

Figure 6: One step degraded ECH

the system should still be functional. This can be accomplished by gracefully degrading the hypercube to a lower dimension. A *k-step Degradable Hypercube* [12] is defined as $Q^{EC}(d, f, k)$ where d is the physical dimension and f signifies the number of faulty nodes present. The functional cube has a dimension of $d-k$. *EC* denotes *Enhanced Cluster*. It can be shown that the Enhanced Cluster Hypercube is one step degradable for up to $2^{d-1}+1$ faulty nodes ($Q^{EC}(d, 2^{d-1}+1, 1)$).

The following algorithm is used to find the 1-step degraded cube. The algorithm first checks whether there exist enough healthy nodes. It then finds a $(d-1)$−subcube with at least 2^{d-2} fault free nodes. Finally, the faulty nodes within this subcube are assigned to the spare nodes via the spare links and the *multi channel communication module* of the spare nodes.

Example: *Enhanced Cluster Hypercube* of dimension 4 with 9 faulty nodes is one step degraded to a hypercube of dimension 3. This is illustrated in figure 6. The shaded subcube is the functional one.

If regular nodes are also provided with the *simultaneous multi-channel capability*, it is possible to use the non-faulty and unused nodes as spares. The result is even greater fault tolerance, $Q^{EC}(d, 3*2^{d-2}, 1)$.

5 Conclusion

In this paper a comprehensive solution to the problem of tolerating faulty nodes and/or links in a hypercube multiprocessors is presented. Using the *Direct Connect Capability* of the hypercube, two schemes are proposed to achieve fault tolerance while maintaining its performance and its algorithmic configuration. The *Cluster Approach* uses a simple spare communication module to enable the hypercube to tolerate a single faulty node per cluster and a single intra-cluster link failure. Using a more complex communication module for the spare, in the *Enhanced Cluster Approach*, multiple faults per cluster are allowed. Compared with the previously proposed approaches, these schemes can tolerate more failures at a lower overhead without affecting the existing communication or computational algorithms.

In case of a massive failure, rather than total system failure, a *Gracefully Degradable Approach* is suggested which finds a $d-1$ functional cube in the presence of up to $2^{d-1}+1$ faulty nodes.

References

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

[2] C. C. Li and W. K. Fuchs, "Graceful degradation scheme for hypercube multiprocessors using data redistribution," in *Proceedings of the Fifth Conference on Hypercube Concurrent Computers and Applications*, pp. 1446–1454, April 1990.

[3] P. Banerjee, "Reconfiguring a hypercube multiprocessor in the presence of faults," in *Proceedings of the Fifth Conference on Hypercube Concurrent Computers and Applications*, pp. 95–102, 1990.

[4] D. Rennels, "On implementing fault-tolerance binary hypercubes," *Proceedings of the IEEE International Symposium on Fault Tolerant Computing*, pp. 344–349, 1986.

[5] S.-C. Chau and A. L. Liestman, "A proposal for a fault-tolerant binary hypercubes architecture," *Proceedings of the IEEE International Symposium on Fault Tolerant Computing*, pp. 323–330, 1989.

[6] P. Banerjee, "Strategies for reconfiguring hypercubes under faults," *Proceedings of the IEEE International Symposium on Fault Tolerant Computing*, pp. 210–217, 1990.

[7] A. Witkowski and R. Lee, "Fault tolerance for the hypercube multiprocessor," in *Proceedings of the Fifth Conference on Hypercube Concurrent Computers and Applications*, pp. 117–122, 1990.

[8] P. Banerjee, J. Rahmeh, C. Stunkel, V. Nair, K. Roy, V. Blasubramanian, and J. Abraham, "Algorithm-based fault tolerance on a hypercube multiprocessor," *IEEE Transactions on Computers*, vol. 39, pp. 1132–1145, September 1990.

[9] M. Alam and R. Melhem, "An efficient modular allocation scheme and its application to fault tolerant binary hypercubes," in *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, pp. 117–126, January 1991.

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

[11] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, "Hyperswitch network for the hypercube computer," in *Proceedings of the 15th Annual International Symposium on Computer Architecture*, pp. 90–99, May 1988.

[12] T. C. Lee and J. P. Hayes, "One-step-degradable fault-tolerant hypercubes," in *Proceedings of the Fourth Conference on Hypercube Concurrent Computers and Applications*, pp. 87–93, March 1989.