Fault-Tolerance in Augmented Hypercube Multicomputers

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

Abstract

This paper describes different schemes for tolerating faults in augmented hypercube multiprocessors. The architectures considered have a spare assigned to each subset of nodes (cluster). The approaches make use of hardware redundancy in the form of spare nodes and/or links and usually requires modifications in the communication as well as computation algorithms.

Keywords: Hypercube, fault-tolerance, spare allocation, reconfiguration

1 Introduction

The area of reconfigurability in the presence of faults is becoming increasingly important with the emergence of massively parallel architectures. Therefore, fault tolerance is an important issue that needs to be addressed in the design of node architectures, communication hardware and software design as well as parallel algorithm development. A d–dimensional hypercube multicomputer consists of $n = 2^d$ processors (nodes) interconnected as a Boolean d-cube, with each processor having only local memory. Inter–processor communication is done by explicit message passing.

Each processor in a d–cube can be represented by a ***d–tuple*** $(b_{d-1} \cdots b_i \cdots b_0)$ where $b_i \in \{0, 1\}$, and a subcube in a d-cube can be represented by a ***d–tuple*** $\{0, 1, \mathrm{X}\}^d$. Coordinate values "0" and "1" can be referred to as *bound* coordinates and "X" as *free*. A $(d - k)$–dimensional subcube ($(d - k)$–subcube) in a d–cube is represented by a ***d–tuple*** with *k bound* coordinates and $(d - k)$ *free* coordinates.

---

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

Two general approaches can be identified for fault tolerance in hypercube multiprocessors. The first approach looks into ways of utilizing the healthy processors and links of a hypercube with faulty nodes/links, to identify embedded topologies such as lower dimensional hypercubes (subcubes), meshes, etc. required by the computations. With this approach, some performance degradation is expected, however special hardware design for fault-tolerance is not necessary. A distributed fault diagnosis algorithm for identifying the set of faulty processors and links, when the number of faulty processors and links $r \leq d$ is given in [1]. Once the faulty elements are identified, the multiprocessor and the distributed algorithm running on the multiprocessor[2] must be reconfigured to run on the available processors. A faulty hypercube needs to be reconfigured to perform both local and global communication as required by the algorithm, effectively and with minimal performance degradation. A study of hypercube algorithms reveals that in many cases, the computations that require local communication are mapped onto topologies such as meshes or rings (grid problems for example [3]) and the hypercube topology is used for global data communication. Therefore, both local and global communication schemes must be considered in designing algorithms for faulty hypercubes.

The second approach to fault-tolerance makes use of hardware redundancy in the form of spare nodes and/or links and usually requires modifications in the communication hardware. Although a d-dimensional hypercube with faulty nodes still contains a number of lower dimensional subcubes and therefore can be used to run hypercube algorithms without any modification [4, 5, 6, 7], one faulty node reduces the dimension of the largest fault-free subcube to $(d-1)$ and 2 faulty nodes can reduce it to $(d-2)$. This has motivated researchers to investigate hardware redundancy and spare allocation schemes. In this paper, a survey of these schemes are presented.

2 Tolerating Node Failure

As mentioned in Section 1, faulty nodes can substantially reduce the dimension of the largest fault-free available subcube. To preserve the dimensionality of the faulty hypercube, hardware redundancy and spare allocation schemes [8, 9, 10, 7, 11, 12, 13, 14, 15] have been investigated.

A hardware scheme was first proposed by Rennels [8]. Here, $N = 2^d$ processors are grouped into $S = 2^s$ clusters of $M = 2^m$ processors each, where $d = m + s$ and one spare is assigned per cluster. Crossbar switches are employed to add spare nodes via the additional port in dimension $(d+1)$ (Fig. 1). If a processor fails in a cluster, the spare replacing it must be able to communicate in d dimensions. Therefore, it has to be linked to the m neighboring processors in that cluster as well as one processor in each of the s external subcubes to which the failed processor is connected. This is accomplished using two crossbars, namely the *CCB (Connection Crossbar)* and the *RCB (Relay Crossbar)*. The *CCB* allows a spare to be connected to each of the 2^m processors in its designated cluster via their spare port. Using the *RCB*, the

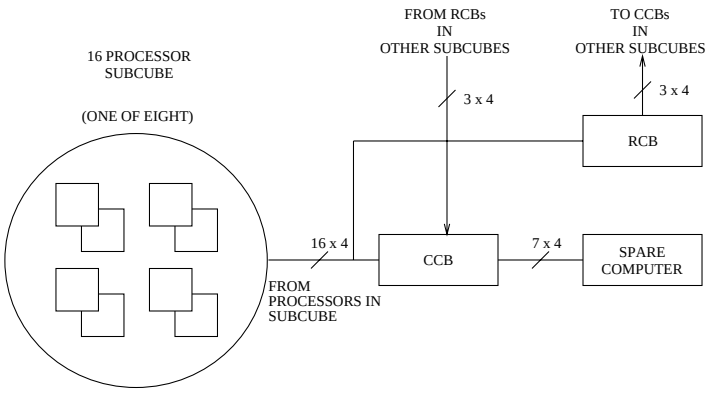


Figure 1: Rennels' scheme for assigning a spare to a cluster

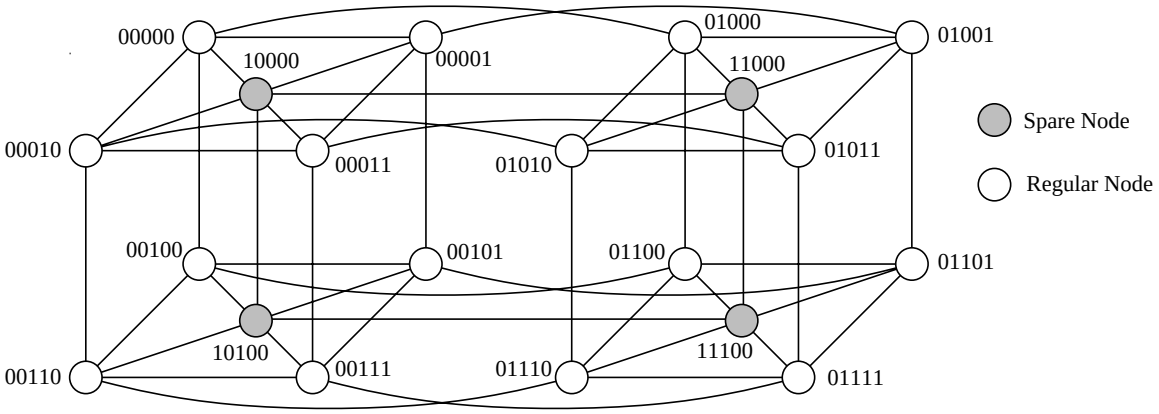


Figure 2: Augmented four-dimensional hypercube topology proposed by Banerjee

spare is linked to each of the other s clusters to which the host cluster is connected. This approach can tolerate only a single faulty node per cluster with a significant overhead. It does not tolerate any link failures. For better fault coverage, spare boards consisting of one spare processor and S crossbars are suggested. Each spare node then can replace any faulty node directly. However, the degree of the spare node would effectively be equal to the size of the hypercube. Furthermore, additional crossbars to interconnect the spares are needed.

Several approaches are proposed by Banerjee et. al. [10, 7, 12] to perform reconfiguration using spare nodes and spare links. In [12] two spares per 3-cube are assigned (Fig. 2) and the spare processors form a $(d-2)$-cube. Upon a node failure, the faulty processor is replaced with the local spare. The link connecting the spare to the faulty processor and the link connecting it to the processor diagonally opposite to the faulty processor are disabled. All other links are enabled. The spare node (S) which replaces the faulty one sends its address and the address of the faulty node to all the spare nodes connected to S. The function of each of the other spares is now that of a simple switch. Consequently each spare determines the node to which it needs to be connected by using the bitwise *XOR* of its own address, the address of the spare S, and the address of the faulty node. As an example, let us consider Fig. 2 when node 01100 is faulty. The reconfiguration steps are as follows. Replace processor 01100 with the spare

processor 11100. The links between the spare 11100 and nodes 01100 and 01111 are disabled. All other links are enabled. The spare node 11100 sends its address as well as the address of the faulty node (01100) to the its neighboring spare nodes (11000 and 10100). The spare nodes 11000 and 10100 calculate the address of the nodes they should be connected to as: $01100 \oplus 11100 \oplus 11000 = 01000$ and $01100 \oplus 11100 \oplus 10100 = 00100$. The spare nodes 11000 and 10100 then act as switches. This system can only tolerate a single node failure.

In [10] Banerjee proposes two schemes to tolerate faulty processors. The first scheme uses two sets of nodes; *P-nodes* and *S-nodes*. *P-nodes* are the regular nodes of the hypercube. An *S-node* consists of a spare attached to a *P-node*. *S-nodes* are allocated such that every regular node is at a Hamming distance of one from a spare node. For example in a 3-cube, spares can be attached to nodes 0 and 7. When a node fails, the spare node at a distance of one from the faulty node is brought on line. The *S-node's* communication module is used to handle both the communication needs of the faulty node and that of the attached regular node. Messages that would have been routed to the faulty node, need to be routed to the spare using a routing table [7]. Therefore, some physical links may experience added traffic which is defined as *congestion*. If more than one node fails and the failed nodes are at a Hamming distance of one from an *S-node*, a weighted bipartite graph is set up to match the faulty nodes with the spares with the minimum *link dilation*. The link dilation in a faulty hypercube is defined by $d(v, \phi(v))$ where v represents the faulty node, $\phi(v)$ identifies the spare replacing v, and d denotes the distance between them. Both dilation and congestion are high with this scheme.

In the second method, spare nodes are placed between links connecting pairs of nodes. For example in a *3-cube*, one spare can be placed between nodes 0 and 1, and another spare between nodes 6 and 7. Similar to the previous scheme, a weighted bipartite graph is used to assign spares to the faulty nodes. Again the scheme results in high dilation and congestion.

Alam and Melhem [16, 13, 14] using hardware redundancy, developed augmented approaches to tolerate faulty nodes. In [16, 13], they have proposed two schemes. In the first one, a single spare is added to each cluster of 4 nodes. If one of the four nodes fails, the spare replaces it and inherits its address. The *e-cube* [17, 18] routing algorithm is no longer valid. The new routing algorithm takes up to $(2d + 2)$ steps as compared to the d steps of the *e-cube* algorithm, to send a message from a source to a destination. The system can tolerate one faulty node per cluster. To allow more faulty nodes per cluster, in the second scheme 50% redundancy is used as shown for $d = 3$ in Fig. 3. Note that each node can be replaced by one of two spares, making the spare assignment nondeterministic. Therefore, an even more complicated routing algorithm is needed. The system can handle two faulty nodes per cluster.

In [14] two spares are assigned per cluster of 4 nodes. Each spare is connected to every node in its cluster using multiple links (channels). The spares are also connected to form a cube of their own. Upon a node failure, the faulty node is assigned to one of the spares within the cluster. The links of the faulty node are then discarded and the mirror image of the spare in the other clusters are used to connect the spare to the neighboring nodes of the faulty node. Since

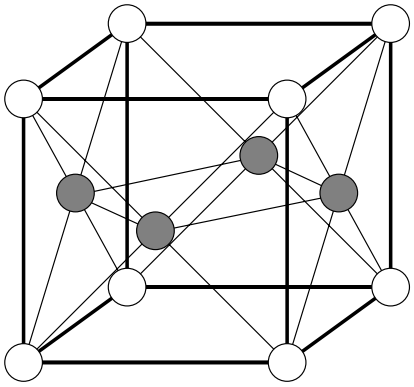


Figure 3: A 3-dimensional reconfigurable hypercube proposed by Alam and Melhem

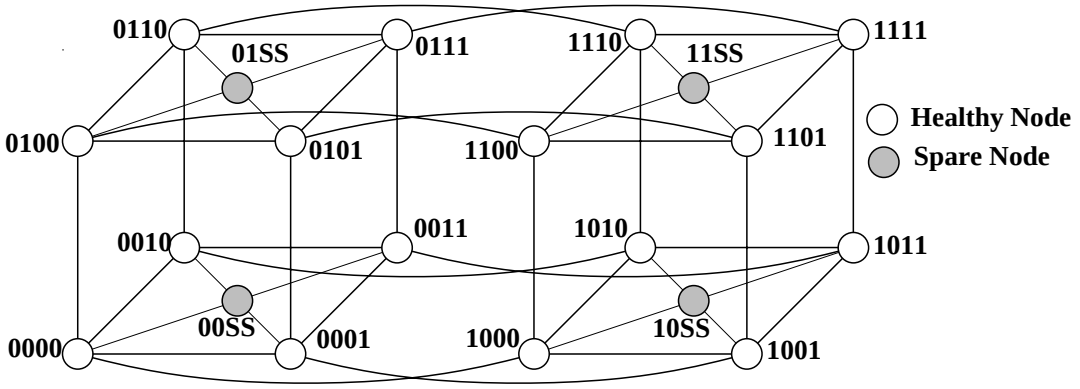


Figure 4: Hypercube of dimension 4 with clusters of dimension 2

more than one neighbor of a node could be faulty, multiple channels are necessary to connect the node to the spares of the faulty nodes. Three faults within a cluster is fatal.

Chau and Liestman [9] have employed a decoupling network to connect modules of active and spare processors. Each module consists of 4 active processor and K spares connected in a cycle. *Soft switches* are used to bypass a total of K faulty and spare nodes. The nodes between modules are connected using k-level decoupling network. They also show that soft switches can be replaced by decoupling network as well. The system can tolerate up to K faulty nodes.

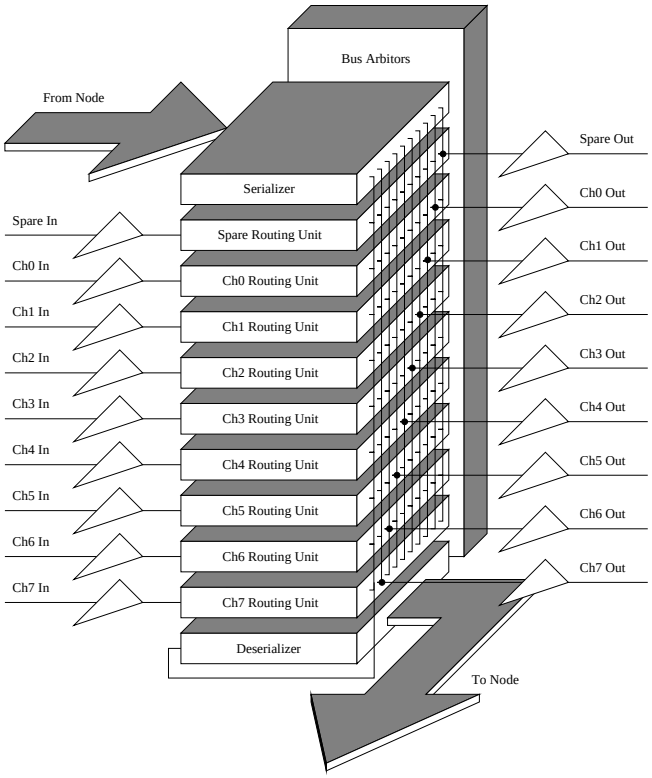
In [15] two augmented schemes are proposed to tolerate faulty nodes. The presented schemes tolerate a large number of faults without any performance degradation and the resultant configuration does not affect either the communication or computational algorithms already developed for the hypercube multiprocessor. In the first scheme (Cluster Approach), a single spare is assigned per cluster of nodes. 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 $d = i + j$. Fig. 4 depicts a hypercube of dimension 4 with clusters of dimension 2. The allocation of spares is facilitated by using a routing element on each node similar in concept to the *Direct Connect Module (DCM)* [17] of *Intel* hypercubes. It is assumed that the faulty nodes retain their communication capability. This is a fair assumption since in hypercube multiprocessors

such as the *iPSC/860*, 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.

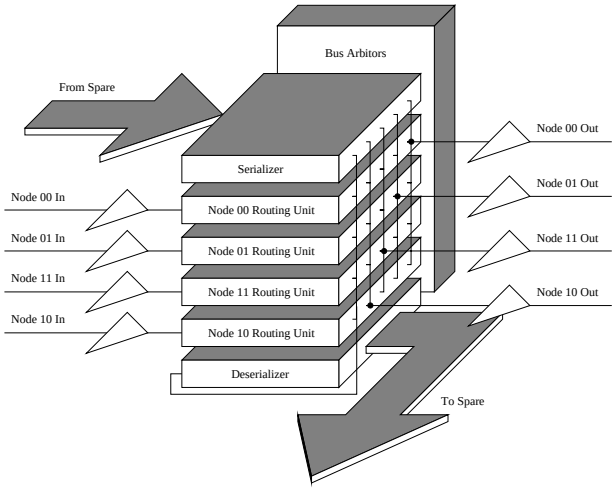
In hypercubes with *Direct Connect Capability* [17, 18], the cost of communication is nearly constant between any two given nodes. To facilitate reconfiguration, the routing elements described below are used at each node. Fig. 5 depicts the block diagram of the router for the regular and the spare nodes. 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. 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 (Fig. 5(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 (Fig. 5) 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 respective cluster logically replaces the faulty node. For example, in Fig. 4, the spare $01SS$ may logically replace any one of nodes $0100, 0110, 0111, 0101$. The spare consequently activates its link to the faulty node and disables 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.

To allow multiple faults within a cluster, in the second scheme (Enhanced Cluster Approach), the spares are connected to form a $(d-2)$-cube. Fig. 6 depicts an *Enhanced Cluster Hypercube (ECH)* of dimension 4. Each regular and spare node is connected to $d+1$ and $4+(d-2) = d+2$ nodes respectively. The block diagram of the router for the regular node is the same as before (Fig. 5(a)). The router for the spare node needs $d+2$ routing elements instead of 4 elements as shown in Fig. 5(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.



(a)



(b)

Figure 5: Communication Module Architecture for (a) Regular Node (b) Spare Node

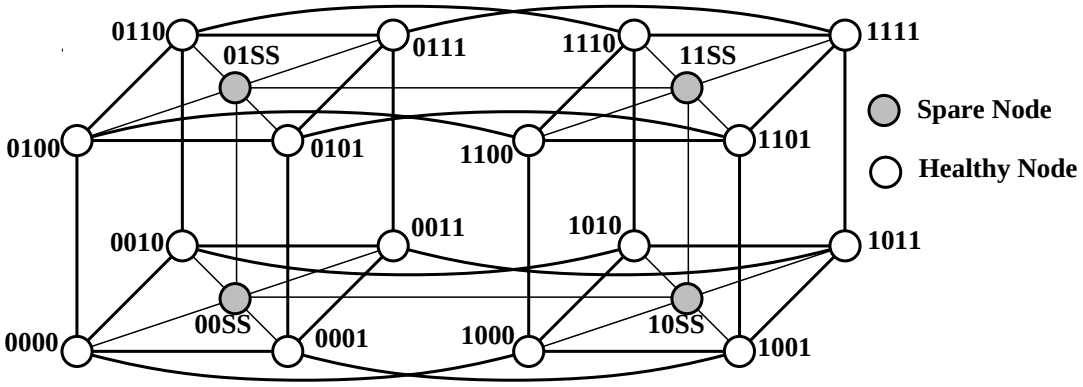


Figure 6: Enhanced cluster hypercube of dimension 4

The approach can tolerate a larger number of faulty nodes [**?**], by establishing dedicated paths, in the spare hypercube, between the spare of the faulty cluster and spares of the non-faulty clusters.

3 Tolerating Link Failure

To tolerate a single link failure Banerjee et. al. [12] have proposed a scheme where the diagonal nodes (nodes with complementing address bits i.e. 010 and 101) are connected with a spare link. Upon detecting a link failure in dimension i, all links in that dimension are discarded and every processor with the *i*–th bit equal to 1 assumes a new address by complementing its current address bits except for the *i*–th bit. As an example, if in a *3*–dimensional hypercube the link -11 (the link between the nodes 011 and 111) becomes faulty, all links in dimension 2 are discarded and are replaced by the diagonal spare links. The node address transformation is as follow; $000 \rightarrow 000, 001 \rightarrow 001, 010 \rightarrow 010, 011 \rightarrow 011, 100 \rightarrow 111, 101 \rightarrow 110, 110 \rightarrow 101, 111 \rightarrow 100$.

In [15], two approaches are proposed to tolerate link failures. In the first scheme, 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 (Fig. 5), with the *spare channel routing element*. For example in Fig. 5, 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 Fig. 4. 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. An Inter-cluster link failure is fatal. Fig. 7 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

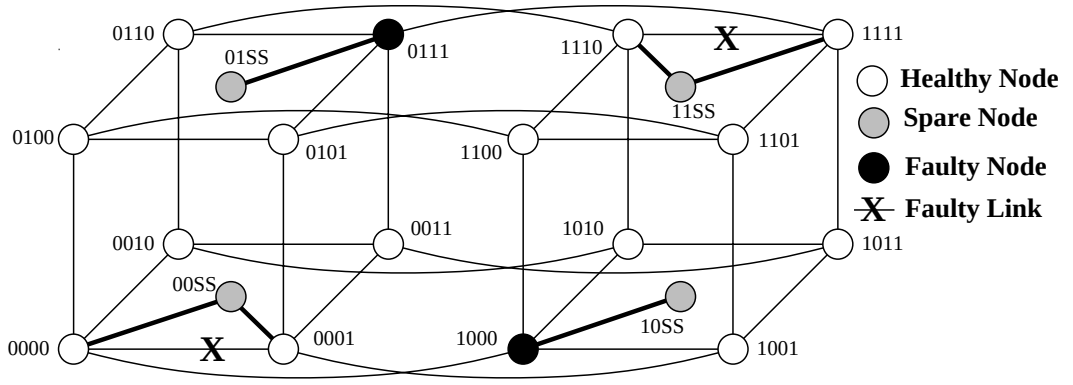


Figure 7: Reconfiguration of a faulty cluster hypercube

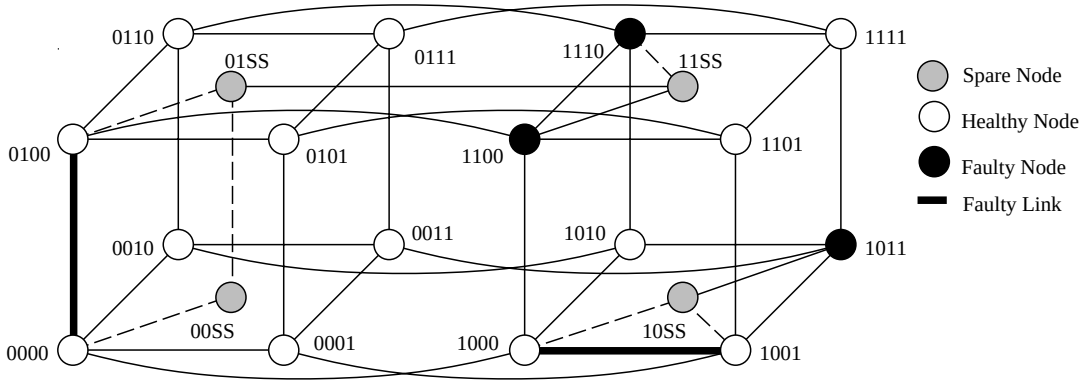


Figure 8: Reconfiguration of a faulty enhanced cluster hypercube

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.

The second approach can also tolerate both intra-cluster and inter-cluster link failures by establishing parallel path(s) to the faulty link(s). Fig. 8 demonstrates the reconfiguration of a hypercube upon detection of faults in links 0–00, 100– and nodes 1011, 1100, 1110. Multi-channel communication capability of the spare node $10SS$ is used to logically replace the node 1011 and at the same time establish a path between the nodes 1001 and 1000. As shown in Fig. 8 the faulty inter-cluster link 0–00 is replaced by the path connecting the nodes $0100,\ 01SS,\ 00SS,$ and 0000. Similarly, spare nodes $01SS$ and $11SS$ logically replace faulty nodes 1100 and 1110, shown by the dark and dashed lines respectively.

4 Conclusions

A faulty hypercube needs to be reconfigured to perform the computational task and communication as required by the algorithm with minimal or no performance degradation. The goal is to avoid the faulty nodes/links and perform useful computation with the fault-free nodes. This

paper outlines research which uses hardware redundancy in the form of spare nodes and links that logically replace the faulty node(s)/link(s). A survey of various schemes is presented.

References

[1] E. Dilger and E. Amman, "System level self diagnosis in n-cube connected multiprocessor networks," in *Proc. 14th Int. Symp. on Fault Tolerant Computing*, pp. 184–189, 1984.

[2] C. Aykanat and F. Özgüner, "A concurrent error detecting conjugate gradient algorithm on a hypercube multiprocessor," in *IEEE 17th International Symposium on Fault Tolerant Computing*, pp. 204–209, July 1987.

[3] C. Aykanat, F. Özgüner, P. Sadayappan, and F. Ercal, "Iterative algorithms for solution of large sparse systems of linear equations on hypercubes," *IEEE Transactions on Computers*, vol. c–37, pp. 1554–1568, December 1988.

[4] F. Özgüner and C. Aykanat, "A reconfiguration algorithm for fault tolerance in a hypercube multiprocessor," *Information Processing Letters*, vol. 29, pp. 247–254, November 1988.

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

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

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

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

[9] 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.

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

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

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

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

[14] M. Alam and R. Melhem, "Channel multiplexing in modular fault tolerant multiprocessors," *Proceedings of the IEEE International Conference on Parallel Processing*, pp. I492–I496, 1991.

[15] B. Izadi and F. Özgüner, "Spare allocation and reconfiguration in a fault tolerant hypercube with direct connect capability," *Proceedings of the Sixth Conference on Distributed Memory Computing Conference*, pp. 711–714, April 1991.

[16] M. Alam and R. Melhem, "Fault tolerance and reliable routing in augmented hypercube architectures," *IEEE 18th Annual Phoenix Int. Conf. on Computer Communication Proceeding*, pp. 19–23, 1989.

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

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