Short Notes

An Efficient Modular Spare Allocation Scheme and Its Application to Fault Tolerant Binary Hypercubes

M. Sultan Alam and Rami G. Melhem

Abstract— **In this paper, we consider fault tolerant systems that are built from modules called fault tolerant basic blocks (FTBB's), where each module contains some primary nodes and some spare nodes. *Full spare utilization* is achieved when each spare within an FTBB can replace any other primary or spare node in that FTBB. This, however, may be prohibitively expensive for larger FTBB's. Therefore, we show that for a given hardware overhead more reliable systems can be designed using bigger FTBB's without *full spare utilization* than using smaller FTBB's with *full spare utilization*. We also present sufficient conditions to maximize the reliability of a spare allocation strategy in an FTBB for a given hardware overhead. The proposed spare allocation strategy is applied to two fault tolerant reconfiguration schemes for binary hypercubes. The first scheme uses hardware switches to replace a faulty node and the other scheme uses fault tolerant routing to bypass faulty nodes in the system and deliver messages to the destination node.**

Index Terms— **Binary hypercube, fault tolerance, modules, multiprocessors, reconfiguration, redundancy, routing, spare allocation.**

I. Introduction

We investigate modular techniques that efficiently use spare nodes to achieve fault tolerance in multiprocessor systems. Modular reconfiguration techniques have been previously proposed for binary tree architectures [7], [8], [11] and hypercubes [3], [6], [9]. The advantages of using modular techniques are 1) local and fast fault detection and reconfiguration, 2) ease of construction, scalability, and module replacement, and 3) simple fault tolerant routing algorithms. A module consists of some primary and spare nodes, and is called a *fault tolerant basic block (FTBB)*. If each spare in an FTBB can replace any primary node from that FTBB, then *full spare utilization* in the FTBB is achieved. With *full spare utilization* within each FTBB, the reliability may be enhanced by using larger FTBB's. However, *full spare utilization* in large FTBB's may require a large number of switches and therefore may not be practical. In such cases, it may be more advantageous not to use *full spare utilization* in FTBB's. In other words, for a fixed hardware overhead, systems built from large FTBB's without *full spare utilization* may be more reliable than systems built from smaller FTBB's with *full spare utilization*. In Section II, we show that the reliability of an FTBB without *full spare utilization* may be optimized for a given hardware overhead if the spares are allocated such that coverage overlap is minimized. This spare allocation strategy can be used for any FTBB irrespective of the interconnection among the primary nodes. That is, the proposed spare allocation strategy is architecture independent. In Section III, we apply this spare allocation strategy to two reconfiguration schemes for fault tolerant binary hypercubes. The first scheme uses hardware switches to preserve the hypercube topology and the second scheme uses fault tolerant routing to deliver a message correctly to its destination without considerable delay.

The use of hardware switches to tolerate faults in hypercubes has first been suggested by Rennels [9]. Recently, Chau *et al.* [3], Dutt *et al.* [6] and Banarjee *et al.* [1] have suggested fault tolerant reconfiguration schemes for hypercubes. Chau *et al.*'s scheme uses decoupling switches to achieve the same reliability as Rennels' scheme using less hardware overhead and fewer spare nodes. The drawback of Chau *et al.*'s scheme is that it takes longer to reconfigure since, on average, the states of half the nodes in a module have to be shifted to neighboring nodes. Dutt *et al.*'s scheme uses hardware switches to replace a faulty node and can be used to design any fault tolerant multiprocessor system with any degree of redundancy. This scheme is shown to be more efficient than Rennels' modular scheme in terms of the area/hardware overhead. In Section III-A, we introduce a scheme which requires the same number of spares and less switches to achieve the same reliability as Chau *et al.*'s scheme. This scheme, however, does not have the overhead of shifting processor states, and thus, reduces the reconfiguration time and overhead significantly. Our scheme also compares favorably in terms of the area/hardware overhead for the switches with that of Dutt *et al.*'s. Banarjee *et al.*'s scheme is different from the above schemes since it does not use hardware switches. Instead it uses the spare nodes to act as switching elements.

The second technique, introduced in Section III-B, uses a general two-phase fault tolerant routing technique. This technique preserves the logical topology of a binary hypercube by modifying the routing algorithm rather than preserving the physical topology. A node failure in this scheme is assumed to be the failure of the processor, the router, and the links of the node. If a node P fails, then a spare node that replaces P inherits the address of P. The routing algorithm is completely distributed and requires local fault knowledge, in the sense that only the neighbors of the failed nodes need to know about the fault in order to take corrective action. We derive an upper bound on the number of steps the algorithm requires to deliver a message to its destination. This routing algorithm is different from the routing algorithms used in injured hypercubes [2], [4], [5] since it routes messages in fault tolerant hypercubes that utilize spare nodes.

II. Modular Fault Tolerance

In order to formally describe the notion of modular fault tolerance, we introduce some terminology. The primary set PS(S) of a spare node S is the set of primary nodes that S can replace. The flexibility flex(S) of a spare node S is the cardinality of PS(S). For a primary node P, the spare set SS(P) of P is the set of spare nodes that can replace P and the degree of coverage deg (P) of P is the cardinality of SS(P). If, in a given system, flex(S) is constant for each S and deg(P) is constant for each P, then, these constants are referred to by flex and deg, respectively. In terms of system implementation, a constant degree of coverage and a constant flexibility means that the spare nodes and the primary nodes may be homogeneous. In this paper, we only consider systems with constant deg and flex.

A fault tolerant basic block F is defined as a minimal set of nodes such that, for any primary node P and spare node S in F, the nodes of SS(P) and PS(S) are in F. For example in Fig. 1(a), the primary nodes 1, 2, 3 and 4 (denoted by the large circles) and the spare nodes 1 and 2 (denoted by the small circles) belong to one FTBB. The size of an FTBB is given by a tuple (M, K) where M and K are the number of primary and spare nodes, respectively. In an FTBB of size (M, K), *full spare utilization* is achieved if flex = M and deg = K. In this case, the FTBB is K-fault tolerant in the sense that it can tolerate any combination of up to K faults, and cannot tolerate any combination of more than K faults.

Constructing fault tolerant systems using small size FTBB's with *full spare utilization* has the advantage of localized recovery mechanisms. However, using large FTBB's leads to more reliable systems. For instance, consider a system constructed from two FTBB's, f_1 and f_2 of sizes (M_1, K_1) and (M_2, K_2) respectively, with *full spare uti-*

Manuscript received September 26, 1989; revised April 12, 1990.

The authors are with the Department of Computer Science, University of Pittsburgh, Pittsburgh, PA 15260.

IEEE Log Number 9040816.

1045-9219/91/0100-0117$01.00 © 1991 IEEE

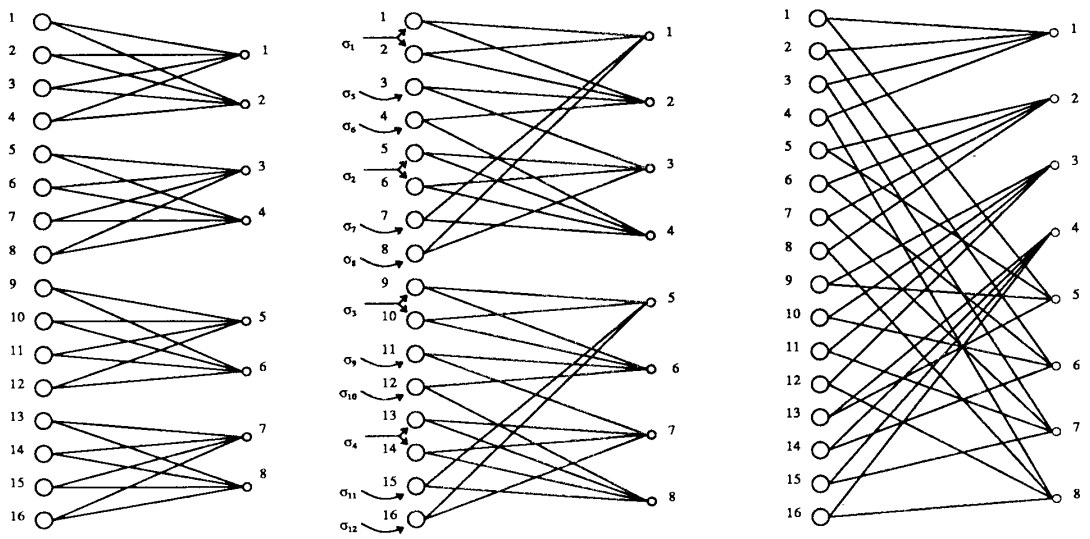

Fig. 1. Systems built from small and large FTBB's (deg = 2, flex = 4).

lization within each FTBB. Such a system is less reliable than another system formed by a single FTBB, f_b, of size $(M_1 + M_2, K_1 + K_2)$ with *full spare utilization.* Specifically, the first system can tolerate up to any combination of $\min\{K_1, K_2\}$ faults and cannot tolerate any combination of more than $K_1 + K_2$ faults. The system may or may not tolerate configurations with a number of faults between $\min\{K_1, K_2\}$ and $K_1 + K_2$, depending on the fault distribution. The second system, however, can tolerate any combination of up to $K_1 + K_2$ faults. Hence, the fault configurations that f_b can tolerate is a proper superset of the fault configurations that the system of f_1 and f_2 can tolerate.

Constructing a system using FTBB's with *full spare utilization* requires more hardware than constructing it using smaller FTBB's, and therefore may not be practical. An alternative to using *full spare utilization* in small FTBB's is to use large FTBB's with less than *full spare utilization.* We use bipartite graphs to explain this. The primary and spare nodes in a system are represented by the left- and right-hand sides, respectively, of a bipartite graph (see Fig. 1), and each edge in the bipartite graph specifies, for each primary node, a spare that can replace that node. In Fig. 1(a), a system which consists of 16 primary nodes and 8 spare nodes is constructed from four FTBB's, each having four primary and two spare nodes and uses *full spare utilization.* Therefore, flex = 4 and deg = 2. Two systems with the same flex and deg are given in Fig. 1(b) and in Fig. 1(c) for FTBB's with 8 and 16 primary nodes, respectively. Each of the FTBB's in Fig. 1(b) and (c) has less than *full spare utilization.* In the remainder of this section, we study the effect of different coverage schemes on the reliability of the system, where a coverage scheme is defined by a corresponding bipartite graph. In all the figures in this paper, the larger circles denote primary nodes and the smaller ones denote spare nodes.

Assume that a system is to be built from M primary nodes and K spares, with specific values for flex and deg (note that M deg = K flex). For any coverage scheme C satisfying the above conditions, we can group the M primary nodes into a number of sets, $\sigma_1, \cdots, \sigma_e$, such that the nodes in each σ_u, where $1 \le u \le e$, are covered by the same set of deg spares. That is, $\mathrm{SS}(P_i) = \mathrm{SS}(P_j)$ for any two P_i and P_j in σ_u. Clearly, the intersection of any two sets σ_u and σ_v is empty. Thus, if λ_u is the size of σ_u, then $\sum_{u=1}^{e} \lambda_u = M$. For example in Fig. 1(a), the primary nodes can be grouped in to four sets $\sigma_1 = \{1, 2, 3, 4\}$, $\sigma_2 = \{5, 6, 7, 8\}$, $\sigma_3 = \{9, 10, 11, 12\}$, and $\sigma_4 = \{13, 14, 15, 16\}$. Also, in Fig. 1(b), the primary nodes can be grouped into 12 partitions, where $\sigma_1 = \{1, 2,\}$, $\sigma_2 = \{5, 6\}$, $\sigma_3 = \{9, 10\}$, $\sigma_4 = \{13, 14\}$ and each one of $\sigma_5, \sigma_6, \cdots, \sigma_{12}$ contains one of the primary nodes 3, 4, 7, 8, 11, 12, 15, and 16. Finally, in Fig. 1(c), we have $\sigma_u = \{u\}$ for $u = 1, 2, \cdots, 16$.

Since each primary node is covered by deg spares, the system can tolerate any combination of deg faults. However, its ability to tolerate more than deg faults highly depends on the coverage scheme C. The number of combinations of deg + 1 faults that the system cannot tolerate is given by the following lemma.

Lemma 1: If the primary nodes in a system can be partitioned into a number of sets, $\sigma_1, \sigma_2, \cdots, \sigma_e$, as defined before, then the number of deg + 1 fault configurations that a coverage scheme C cannot tolerate is given by

$$\Phi(C) = \sum_{u=1}^{e} \binom{\lambda_u + \deg}{\deg + 1}.$$

Proof: The λ_u primary nodes in σ_u are covered by the same set of spare nodes μ_u, where $|\mu_u| = \deg$. Let $\psi_u = \{\sigma_u \bigcup \mu_u\}$ and assume that the system has deg + 1 faults which are distributed over τ sets of ψ_u's, where $1 \le \tau \le e$. If $\tau > 1$, then without loss of generality we can denote the τ sets of ψ_u's by $\psi_1, \psi_2, \cdots, \psi_\tau$. For each $u, v = 1, 2, \cdots, \tau$ and $u \ne v$, we have $|\mu_u| = |\mu_v| = \deg$ and $\mu_u \ne \mu_v$, therefore the cardinality of $\bigcup_{u=1}^{\tau} \mu_u \ge (\deg + 1)$. This implies that the primary nodes in the set $\bigcup_{u=1}^{\tau} \sigma_u$ are covered by at least deg + 1 spare nodes. Therefore, it can be shown that the set $\bigcup_{u=1}^{\tau} \psi_u$ can tolerate at least deg + 1 faults. Hence, if the deg + 1 faults are distributed over more than one ψ_u then any combinations of deg + 1 faults can be tolerated. However, if $\tau = 1$, i.e., the deg + 1 faults are confined to a single ψ_u, then any combination of deg + 1 faults in ψ_u will cause the system to fail. This is because the primary nodes in each ψ_u are covered by deg spares. Therefore, the total number of combinations of deg + 1 faults that any set ψ_u cannot tolerate is $\binom{\lambda_u + \deg}{\deg + 1}$. Since there are e such ψ_u's in the system the result follows. □

Therefore, by choosing a configuration which minimizes the function Φ, we increase the probability of the system to tolerate deg + 1 faults. The following proposition specifies the minimum possible value of Φ:

Proposition 1: For any configuration C, the value of $\Phi(C)$ is always larger than or equal to M.

Proof: Let $\sigma_1, \cdots, \sigma_e$ be as defined earlier. By construction, $\lambda_1 + \cdots + \lambda_e = M$ and $e \leq M$. The result follows directly by using in the definition of Φ the relation

$$\binom{\lambda_u + \deg}{\deg + 1} \geq \lambda_u \qquad \text{if } \lambda_u \geq 1.$$

□

Clearly, the lower bound on $\Phi(C)$ is obtained if $e = M$ and $\lambda_u = 1$ for $u = 1, \cdots, M$. A coverage scheme that satisfies this condition is called a coverage with *minimal overlap.* This name is justified by the following corollary.

Corollary 1: A coverage scheme C satisfies $\Phi(C) = M$ if and only if, for any set of deg spares, $\{S_1, S_2 \cdots, S_{\deg}\}$, the cardinality of $\mathrm{PS}(S_1) \bigcap \mathrm{PS}(S_2) \bigcap \cdots \bigcap \mathrm{PS}(S_{\deg})$ is at most one.

Proof: We prove by contradiction that a coverage scheme C satisfies $\Phi(C) = M$ only if for any set of deg spares, $\{S_1, S_2 \cdots, S_{\deg}\}$, the cardinality of $\mathrm{PS}(S_1) \bigcap \mathrm{PS}(S_2) \bigcap \cdots \bigcap \mathrm{PS}(S_{\deg})$ is at most one. For any set of deg spares, $\{S_1, S_2 \cdots, S_{\deg}\}$, if the cardinality of $\mathrm{PS}(S_1) \bigcap \mathrm{PS}(S_2) \bigcap \cdots \bigcap \mathrm{PS}(S_{\deg})$ is x, where $x > 1$, then the x primary nodes in the intersection must be in the same partition. This implies that $|\sigma_u| = x > 1$, for some u, and hence $\Phi(C) > M$.

If for any set of deg spares, $\{S_1, S_2 \cdots, S_{\deg}\}$, the cardinality of $\mathrm{PS}(S_1) \bigcap \mathrm{PS}(S_2) \bigcap \cdots \bigcap \mathrm{PS}(S_{\deg})$ is at most one, then by construction, each σ_u will contain no more than one primary node. Since there are M primary nodes in a module, there are M distinct σ_u's containing one primary node each. This implies that $\lambda_u = 1$ for $u = 1, \cdots, M$, which in turn implies that $\Phi(C) = M$. □

For example in Fig. 1(c), deg = 2, and for any deg spare nodes S_1 and S_2 the cardinality of $\mathrm{PS}(S_1) \bigcap \mathrm{PS}(S_2)$ is at most one. Therefore, the coverage scheme in Fig. 1(c) has minimum overlap. It is worth mentioning that $\Phi(C)$ for the coverage schemes in Fig. 1(a) and (b) are 80 and 24, respectively.

For a fixed hardware overhead (i.e., fixed deg and flex) it can be shown that building a system from large FTBB's with minimal coverage overlap is better than building it using small FTBB's with *full spare utilization.* Specifically, consider a system, SYS1, composed of Q FTBB's, each having M primary nodes, K spare nodes, and *full spare utilization.* The number of deg + 1 fault configurations that this system cannot tolerate is

$$\Phi(\mathrm{SYS1}) = \sum_{u=1}^{Q} \binom{M + \deg}{\deg + 1}.$$

If a system, SYS2, is built from a single FTBB of size (QM, QK) and with the same degree of coverage deg (and consequently the same flex), then it has the same hardware complexity as SYS1 but has less than full spare utilization within the FTBB. If SYS2 is organized to minimize the coverage overlap, then, the number of deg + 1 fault configurations that it cannot tolerate is $\Phi(\mathrm{SYS2}) = QM$.

Clearly, $\Phi(\mathrm{SYS1})$ is larger than $\Phi(\mathrm{SYS2})$. However, this only means that SYS2 has a higher probability of surviving deg + 1 faults. This result is expected to extrapolate to larger numbers of faults, but the general analytical expressions become more complex and less manageable. For this reason, we consider few specific cases and we present simulation results which show that SYS2 has a higher probability of survival for any number of faults. Specifically, in Fig. 2(a), we compare a single FTBB of size $(16, 8)$ and minimal coverage overlap [Fig. 1(c)] with a system composed of 4 FTBB's of size $(4, 2)$ and *full spare utilization* [Fig. 1(a)]. In Fig. 2(b), we compare a single FTBB of size $(64, 16)$ and minimal coverage overlap with a system composed of 8 FTBB's of size $(8, 2)$ and *full spare utilization.* Finally, in Fig. 2(c), we compare a single FTBB of size $(64, 48)$ and minimal coverage overlap with a system composed of 16 FTBB's of size $(4, 3)$ and full spare utilization. In all of these examples, systems composed of large FTBB's with minimal coverage overlap are more reliable than systems composed from smaller FTBB's with full spare utilization.

Given specific M, flex, and deg it is desirable to build the system from FTBB's which may be arranged to minimize Φ. However, it is not always possible to design a coverage scheme C such that $\Phi(C) = M$. It is only possible to do so if we can find M different subsets of $\{1, 2 \cdots, K\}$, each of size deg, such that each integer j, $1 \leq j \leq K$ appears in exactly flex subsets. For example, $\Phi(C) > M$ for any coverage scheme C, for a system of $M = 8$, $K = 4$, flex = 4, and deg = 2. This is because it is impossible to find eight different subsets of $\{1, 2, 3, 4\}$, each of size 2. In the following proposition, we give conditions which are sufficient, but not necessary, to find a coverage with minimal overlap:

Proposition 2: Given deg and flex, a coverage scheme C with $\Phi(C) = M$ can be designed if $M = \mathrm{flex}^{\deg}$.

Proof: Assume that the primary nodes are logically arranged as a deg-dimensional mesh with flex primary node along each dimension. Clearly, the total number of primary nodes required for such an arrangement is $\mathrm{flex}^{\deg}$. For each dimension i, and each set of flex primary nodes along that dimension, add a spare node to cover for those primary nodes. For example in Fig. 3(b), spare S_1 covers the flex primary nodes along dimension 1, enclosed in a slim rectangle pointed to by an arrow from S_1. Similarly, the primary sets of the spare nodes S_2 and S_3 are shown in Fig. 3(b). Clearly, with this arrangement each primary node is covered by deg spares and each spare covers flex nodes. Also, for any deg spares, $S_1, \cdots, S_{\deg}$, the sets $\mathrm{PS}(S_1), \cdots, \mathrm{PS}(S_{\deg})$ intersect at at most one primary node. From Corollary 1, this implies that for such coverage schemes $\Phi = M$. Examples of such spare allocation for systems logically arranged as a *4-ary two-dimensional* mesh and a *4-ary three-dimensional* mesh are shown in Fig. 3(a) and (b) respectively.□

In the next section, we describe fault tolerant hypercube architectures that are built in a modular way. Specifically, an n-dimensional hypercube system with $N = 2^n$ primary nodes is viewed as an $(n - m)$-dimensional hypercube of m-dimensional subcubes, for some $m \leq n$. This n-dimensional hypercube is called the *primary cube* and its links are called the *primary links.* To each m-dimensional subcube, K spares are added to cover for the $M = 2^m$ primary nodes in that subcube, thus forming an FTBB.

For the hypercube system of Section III-B, we have chosen to use flex = 4 and deg = 2. For these values, minimum coverage overlap can be achieved if the system is built from FTBB's with $M = \mathrm{flex}^{\deg} = 16$ primary nodes that are arranged as in Fig. 3(a). But if we want to build a fault tolerant system from FTBB's with $M = 8$ and the same flex and deg as above, then we have to build a system with $\Phi > 8$. For such an FTBB it is easy to show that the minimum value of Φ is 12 and is obtained if $\lambda_1 = \lambda_2 = 2$ and $\lambda_u = 1$, $u = 3, \cdots, 6$. An FTBB with this specification is shown in Fig. 4(a), where spare nodes adjacent to a primary node P can replace P. However, the spare distribution in the FTBB of Fig. 4(a) is rather irregular, and thus, if used as a building block of a hypercube system, results in a complex and irregular reconfiguration and routing algorithm. For this reason, the FTBB shown in Fig. 4(b) is used, in which, $\lambda_u = 2$, $u = 1, \cdots, 4$ and thus $\Phi = 16$. For the given hardware complexity of deg = 2 and flex = 4, let sys3 and sys2 be the systems containing eight primary nodes which are built from FTBB's of Fig. 4(a) and (b), respectively. Also, let sys1 be the system constructed from two FTBB's of size $(4, 2)$ with *full spare utilization.*

Let $R_{\mathrm{sys3}}, R_{\mathrm{sys2}}$, and R_{sys1} be the reliabilities of the sys3, sys2, and sys1, respectively. In order to calculate the reliability of the systems we use C_j^i to denote the number of combinations of j nodes selected from a set of i nodes, and use r to denote the reliability of a single node ($r = e^{-\omega t}$, where ω is the failure rate and t is the time). The reliability of a module of sys1 is $R_{\mathrm{sys1}}^M = r^6 + C_1^6 r^5 (1 - r) + C_2^6 r^4 (1 - r)^2$. The reliability of sys1 is therefore $\left(R_{\mathrm{sys1}}^M\right)^2$. The same method can be used to calculate the reliability of sys2 except that we need to know how many 3 and 4 faults can sys2 survive. The number of 3 faults that sys2 can survive is given by $C_3^{12} - \Phi$, this is because Φ is the number of deg + 1 faults that the system cannot survive (deg = 2). In sys2, there are C_3^4 different ways to pick a set of three spare nodes. Each such set is denoted by χ_u, where $0 \leq u \leq 3$. Let δ_u denote the set of primary nodes such that if $p \in \delta_u$, then $\mathrm{SS}(p) \subset \chi_u$. Also,

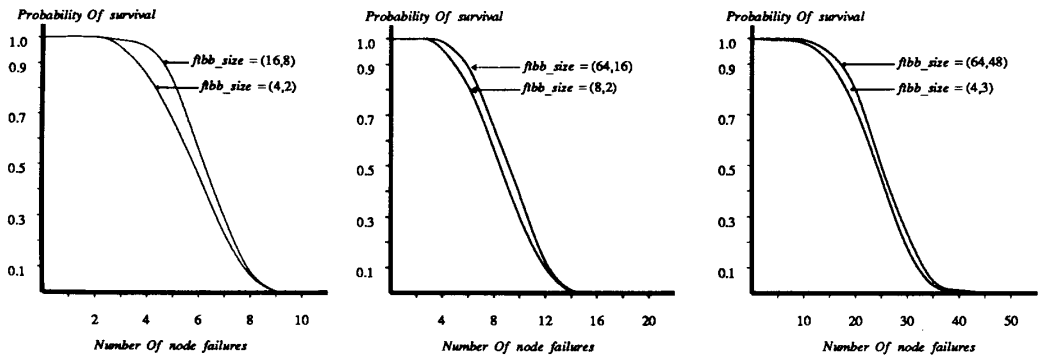

Fig. 2. Probability of survival for given number of faults. (a) N = 16, deg = 2, and flex = 4. (b) N = 64, deg = 2, and flex = 8. (c) N = 64, deg = 3, and flex = 4.

let $\eta_u = \{\delta_u \bigcup \chi_u\}$, for $u = 0, \cdots, 3$. For instance, in Fig. 6, $\chi_1 = \{s_2, s_3, s_4\}, \delta_1 = \{P_3, P_4, P_5, P_6\}$, and $\eta_1 = \{\delta_1 \bigcup \chi_1\}$. Thus, sys2 cannot tolerate any combination of four faults in an η_u since there are only three spares in it. In the four η_u's there are $4C_4^7 - 4$ combinations of four faults that the system cannot tolerate. The minus 4 is because there are four combinations out of the $4C_4^7$ that are counted twice. In Fig. 6, $\psi_{(u+1) \bmod 4} = \{\eta_{u \bmod 4} \bigcap \eta_{(u+1) \bmod 4}\}$, for $u = 0, \cdots, 3$. Therefore, ψ_1 contains the four nodes that are in the intersection of η_0 and η_1. The dotted lines in Fig. 6 denote the wrap around. Also, sys2 cannot tolerate any four faults, where three of the faults are in a ψ_u and the other fault is elsewhere. All combinations of four such faults have been counted before except for the $4C_3^4C_1^2$ combinations which were not counted since for each ψ_u, $\{\eta_{u \bmod 4} \bigcup \eta_{(u+1) \bmod 4}\}$ contains all but two primary nodes of sys2. Therefore, the total number of four faults that sys2 cannot tolerate is $4C_4^7 + 4C_3^4C_1^2 - 4$. The reliability of sys3 can be calculated in the same way. $R_{\text{sys1}}, R_{\text{sys2}}$, and R_{sys3} are given below.

$$
\begin{aligned}
R_{\text{sys1}} &= r^{12} + 12r^{11}(1-r) + 66r^{10}(1-r)^2 + 160r^9(1-r)^3 \\
&\quad + 225r^8(1-r)^4 \\
R_{\text{sys2}} &= r^{12} + C_1^{12}r^{11}(1-r) + C_2^{12}r^{10}(1-r)^2 \\
&\quad + \left(C_3^{12} - \Phi\right)r^9(1-r)^3 \\
&\quad + (C_4^{12} - 4C_4^7 + 4C_3^4C_1^2 - 4)r^8(1-r)^4 \\
&= r^{12} + 12r^{11}(1-r) + 66r^{10}(1-r)^2 + 204r^9(1-r)^3 \\
&\quad + 327r^8(1-r)^4 \\
R_{\text{sys3}} &= r^{12} + C_1^{12}r^{11}(1-r) + C_2^{12}r^{10}(1-r)^2 \\
&\quad + \left(C_3^{12} - \Phi\right)r^9(1-r)^3 \\
&\quad + (C_4^{12} - 2C_4^7 - 2C_4^6 + 2 - 2C_1^5 - 2C_1^2 - 6C_3^4) \\
&\quad \times r^8(1-r)^4 \\
&= r^{12} + 12r^{11}(1-r) + 66r^{10}(1-r)^2 + 208r^9(1-r)^3 \\
&\quad + 359r^8(1-r)^4.
\end{aligned}
$$

The values of $r^i(1-r)^{12-i}$ for $i = 8, \cdots, 12$, are positive for any value of ω and t. Let the coefficients of $r^i(1-r)^{12-i}$ for sys$N$ be denoted by Z_N^i. In the reliability equations given above, $Z_1^i \leq Z_2^i \leq Z_3^i$ for $i = 8, 9, \cdots, 12$. Therefore, $R_{\text{sys3}} > R_{\text{sys2}} > R_{\text{sys1}}$ for any value of ω and t. In Fig. 5, $R_{\text{sys1}}, R_{\text{sys2}}$, and R_{sys3} are shown for typical values of ωt. Given that for sys1, sys2, and sys3, the values of Φ are 80, 32, and 24, respectively, this result shows that smaller Φ results in a more reliable system.

III. Fault Tolerant Reconfiguration Schemes for Binary Hypercubes

A. Reconfiguration Using Hardware Switches

Given an FTBB with M primary nodes, $P_1, \cdots, P_M$, and K spare nodes, $S_1, \cdots, S_K$, *full spare utilization* within this FTBB means that any P_j may be replaced by any S_i. Such a replacement may be accomplished if hardware switches are used to redirect the n communication links connected to P_j into S_i. The switching logic may be described in terms of multiplexers and demultiplexers as shown in Fig. 7, where, for simplicity, it is assumed that multiplexers and demultiplexers can multiplex and demultiplex duplex links, respectively. Specifically, a *one_to_K* + 1 demultiplexer is used for each P_j to divert, when needed, the links of P_j to any S_i. Also, an *M_to_one* multiplexer is used for each S_i to connect it to the appropriate links. If a primary node P_j is nonfaulty then the demultiplexer associated with P_j is set to select the 0th output. In case P_j is faulty, the replacement of P_j by S_i requires that the demultiplexer associated with P_j be set to its ith output, and the multiplexer associated with S_i be set to select its jth input. This scheme can also support spare failures. For example, assume that spare node S_i replaced primary node P_j and then S_i fails. Now, spare node S_k may replace S_i, if the demultiplexer associated with P_j is set to select the kth output and the multiplexer associated with S_k is set to select the jth input.

A different fault tolerant hypercube architecture is suggested in [3], where the failure of a primary node P_j requires that, on the average, $M/2$ of the active nodes in the FTBB that contains P_j be relocated to other primary or spare nodes. This relocation requires the setting of two multilayer decoupling networks, one to restore the logical hypercube connections within the FTBB and the second to restore the logical connections with other FTBB's. Both our architecture and that of [3] support full spare utilization within each FTBB. Hence, as shown in [3], they achieve better reliability than the hierarchical and the global scheme suggested by Rennels [9].

The switch complexity of an N-node hypercube, $N = 2^n$, constructed using FTBB's of size (M,K) may be easily calculated. Specifically, each n-line, *one_to_X* multiplexer or *X_to_one* demultiplexer may be constructed from $n(X-1)$ two_to_one switches, where X is an integer greater than one. Hence, the number of switches used for the demultiplexers associated with the 2^n primary nodes is 2^nKn. Similarly, if $M = 2^m$, then the number of switches used for the multiplexers associated with the spare nodes in the 2^{n-m} FTBB's is $2^{n-m}K(2^m-1)n$. That is, the total number of switches used is $2^{n-m}nK(2^{m+1}-1)$. By comparing that number to the $2^{n-m}n(2^{m+1}+K-1)K$ two_to_one switches used in the configuration suggested in [3], it becomes clear that Chau and Liestman's scheme requires $2^{n-m}nK$ more switches than ours to

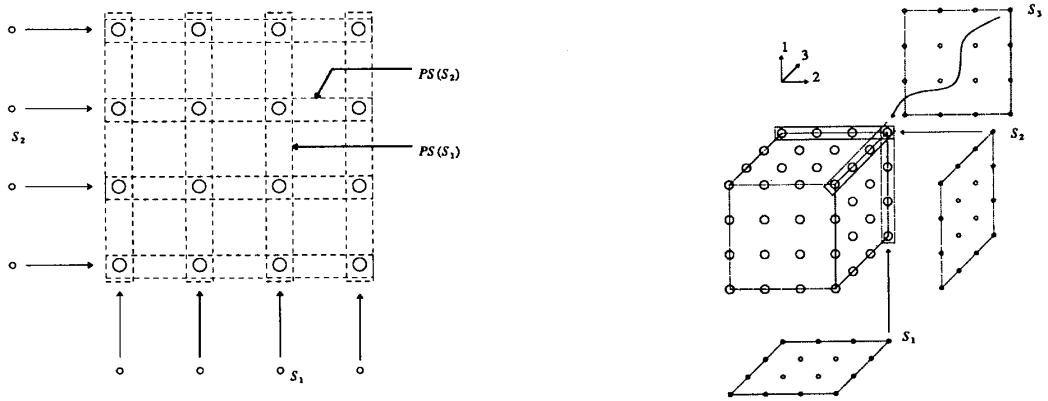

Fig. 3. d-dimensional meshes. (a) 4-ary two-dimensional mesh with deg = 2 and flex = 4. (b) 4-ary three-dimensional mesh with deg = 3, flex = 4.

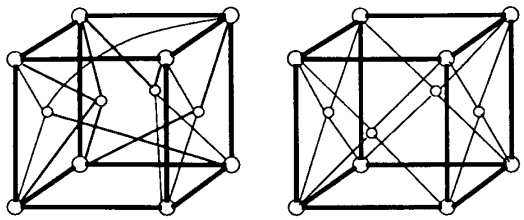

Fig. 4. FTBB's of size (8, 4). (a) Φ = 12 (sys3). (b) Φ = 16 (sys2).

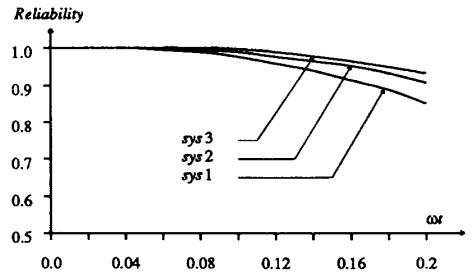

Fig. 5. Comparison of reliabilities.

achieve the same reliability using the same number of spares per module. Moreover, the reconfiguration in our scheme is faster since the state of the failed node needs to be relocated.

An automorphic design approach to design of fault tolerant multiprocessors have been proposed by Dutt *et al.* [6]. This scheme can implement any degree of redundancy in regular multiprocessor graphs that are circulant. For noncirculant multiprocessor graphs the edge-supergraph of the multiprocessor graph needs to be constructed efficiently in order to implement fault tolerance. For binary hypercubes the degree of a node of the edge-supergraph is approximately twice the degree of the node in the original multiprocessor graph. To implement a K-FT hypercube with $M = 2^m$ primary nodes the total area required for the switches is $\theta(mMK \log K)$. To construct the same K-FT system the total area required for the switches in our scheme can be shown to be $\theta(mMK)$. This is because an *X_to_one* switch can be constructed using $\theta(X)$ *two_to_one* switches. The complexity of $\log K$ in Dutt *et al.*'s scheme is due to the $\log K$ bit address registers required to store the address of a spare node in the demultiplexers.

This technique of using multiplexers and demultiplexers for reconfiguration is very flexible and may be applied to FTBB's with any coverage scheme. This implies that FTBB's without *full spare utilization* can be easily implemented using this scheme for a fixed deg and flex. An advantage of this technique is that its hardware overhead is fixed for fixed deg and flex and does not depend on the coverage scheme. Specifically, for each primary node a *(deg + 1)_to_one* demultiplexer is needed, and for each spare node, a *flex_to_one* multiplexer is needed.

B. Fault Tolerant Routing

In this section, we present an alternative to preserving the physical adjacency of the active nodes after reconfiguration. Specifically, when a node fails, it is replaced by a spare that inherits its address. Instead of changing the connections in the system to preserve physical adjacency, the e-cube routing algorithm [10] is modifed such that any message destined to P is sent to S if P has been replaced by S. An advantage of this scheme is that there are no assumptions about fault-free switches.

In order to implement fault tolerant routing, we consider n-dimensional hypercube systems built from FTBB's of the type shown in Fig. 4(b). The number of primary nodes in an FTBB is denoted by M, where $M = 2^m$ and m is an integer greater than zero. A primary node p has an n-bit address given by $p_n \cdots p_i \cdots p_1$. An FTBB containing some node p is denoted by F_p, where $F_p = p_n, \cdots, p_4$. Four spare nodes are added to an FTBB each having an address $p_n \cdots p_4 xxx$, where the x's denote generic bits. The spare nodes are interconnected as a hypercube of dimension $n - 1$. A node that is the neighbor of node p across dimension i is denoted by $C_i(p) = p_n \cdots \overline{p}_i \cdots p_1$. Similarly, an FTBB that is across dimension i from FTBB F is denoted by $C_i(F)$. When a primary node p fails, it is replaced by a spare s which inherits the active address of p. If no spare is available to replace p, then the system fails. A system that has not failed is called a *live* system, and in such a system, the *active nodes* are defined to be the set of nonfailing primary nodes and the spares that replaced the failing primary nodes. A four-dimensional hypercube system constructed from two FTBB's of the type described above is shown in Fig. 8.

We consider two-phase routing algorithms which are applicable to any architecture that uses message routing to communicate between nodes. In this kind of algorithm a message addressed to a destination node d is first routed to the FTBB that contains d, and then to d. Specifically for hypercubes, in order to deliver a message to a destination $d_n \cdots d_1$ in FTBB F_d, the following general algorithm may be applied:

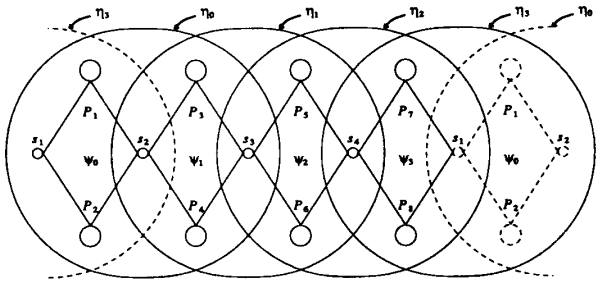

Fig. 6. FTBB of sys2 drawn in 2-D (dotted lines denote wrap around).

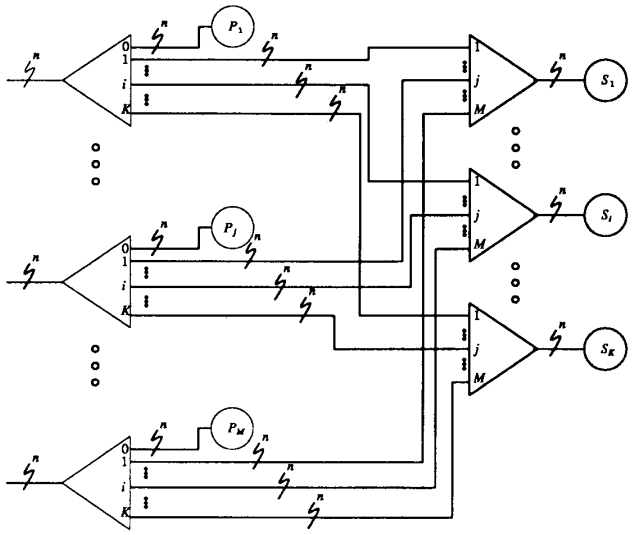

Fig. 7. Switching logic within an FTBB.

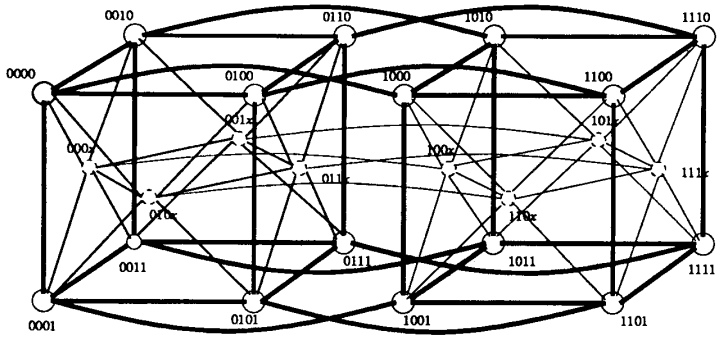

Fig. 8. A four-dimensional reconfigurable hypercube architecture (Spare links and nodes are shown using thin lines).

Phase 1: if the message is in some node p in FTBB F_p, where $F_p \neq F_d$, then send that message to some node in FTBB $C_j(F_p)$, where j is the largest integer such that $j > m$ and $p_j \neq d_j$. Repeat this step until $p_j = d_j$ for all $j > m$.

Phase 2: route the message to node d without leaving FTBB F_d.

Given a distribution of faults in the system, a link which connects two nonfaulty nodes is called a *healthy* link. A nonfaulty path between two nonfaulty nodes is either a healthy link between the two nodes or a sequence of healthy links that connect the two nodes through nonfaulty nodes. A nonfaulty node p in an FTBB F_p is said to be *not isolated* in F_p if there exists at least one nonfaulty path within F_p from p to every other nonfaulty node in F_p. In this context, a path within F_p is one that does not leave F_p.

The general two-phase routing algorithm described above may deliver messages correctly only if certain conditions are met. For instance, phase 1 may fail if there are two adjacent FTBB's that are not connected by at least one *healthy* link. Also, phase 2 may fail if there exists a node in some FTBB which is *isolated* in that FTBB.

Node isolation, however, may not take place in FTBB's with full spare utilization. Moreover, as shown by the following proposition, healthy connections between adjacent FTBB's in a hypercube may be guaranteed if less than 100% redundancy is used.

Proposition 3: If a hypercube system is constructed from FTBB's of size (M, K) such that the spare nodes are interconnected as a hypercube then the existence of a healthy link between any two adjacent FTBB's in a *live* system is guaranteed if $M > K$.

Proof: Let f_1 and f_2 be two FTBB's that are adjacent across dimension j. For each node p in f_1, there exists a link (not necessarily healthy) between p and $C_j(p)$, where $C_j(p)$ is in f_2. That is, there are $M + K$ links between f_1 and f_2. For the system to be *live,* each FTBB may not contain more than K faulty nodes. Hence, at most $2K$ of the $M + K$ links between f_1 and f_2 may be *nonhealthy,* leaving $M + K - 2K = M - K$ healthy links. The result follows directly. □

For systems with $M > K$, a message in phase 1 may be routed from a node p in a FTBB, f_1, to an adjacent FTBB, f_2, if a path π is found within f_1 which passes through at most K nonfaulty nodes. The message is then sent along π until a healthy link to f_2 is found. There are many methods to compute π for phase 1. The first is to compute π in p from global information about the closest node in f_1 which has a healthy link to f_2. The problem with this approach is that gathering global information wastes a lot of bandwidth. A second method which does not require global information is to design a backtracking algorithm similar to the one described in [2]. In such a method, a list of nodes visited so far should be kept in the message to prevent looping. This implies that the message needs to be updated at each node that it visits resulting in slower message delivery because store and forward approach with message update is used. A third method is to carefully design a routing strategy within f_1 which ensures that K distinct nodes are visited without explicitly computing or storing π. A similar argument may be given for the classification of methods that implement phase 2 of the general routing algorithm. In this section, we present a two-phase routing algorithm that uses the third method for routing messages in hypercube systems that are constructed from FTBB's of size (8, 4) (see Fig. 8). Our goal is to achieve distributed routing with only local information at each node. However, because of the flexibility of the architecture, a distributed routing algorithm that is not carefully designed may lead to a live-lock situation in which a message circulates in a loop, thus never reaches it destination. The algorithm described here is distributed, live-lock free, and requires each node to know only the status of its neighboring nodes.

Within an FTBB, six faces are identified and denoted by $\text{face}^{(b)}_{\alpha,\beta}$, $\alpha, \beta = 1, 2, 3$, $\alpha \neq \beta$, $b = 0, 1$. Specifically, if γ is the integer between 1 and 3 that is different from α and β, then $\text{face}^{(b)}_{\alpha,\beta}$ consists of the four nodes that have $p_\gamma = b$ and that are neighbors across the two dimensions α and β. For example in Fig. 8, the primary nodes 0000, 0010, 0011, and 0001 form $\text{face}^{(0)}_{1,2}$ of FTBB 0. Each FTBB, $p_n \cdots p_4$, has one spare node on each of $\text{face}^{(0)}_{1,2}$, $\text{face}^{(1)}_{1,2}$, $\text{face}^{(0)}_{1,3}$, and $\text{face}^{(1)}_{1,3}$, and is connected to the four primary nodes on that face. The spare on $\text{face}^{(b)}_{1,2}$ is identified by $p_n \cdots p_4 bbX$ and the spare on $\text{face}^{(b)}_{1,3}$ is identified by $p_n \cdots p_4 \bar{b}bX$.

In systems constructed from FTBB's of Fig. 8, any two primary nodes p and $C_1(p)$, along with their two common spares form what we call a *corner-quad* (a set ψ_u in Fig. 6), and two adjacent *corner-quads* form a *half-cube* (a set η_u in Fig. 6.) If the primary node p and the spare node s belongs to the corner-quad q, then qtwin (s, p) denotes the spare node v, where $v \in q$ and $v \neq s$. For example, in Fig. 8, nodes 0000, 0001, 010x and 000x form a *corner-quad* and 0000, 0010, 0011, 0001 and three spare nodes 010x, 000x, and 001x form a *half-cube.* Also, qtwin(0000,010x) denotes the spare node 000x. A two-phase routing algorithm of the type described above can be designed for such a system because, by Proposition 3, there exists a path from any FTBB to another FTBB and also by the following lemma no nodes can become isolated from all its neighbors within its FTBB.

Lemma 2: Node isolation within an FTBB is impossible in a *live* system which is built from the FTBB's shown in Fig. 8.

Proof: A *corner-quad* is a complete graph of four nodes. Since no more than two nodes per *corner-quad* can fail, the live nodes in a *corner_quad* are connected by a healthy link. A FTBB is made of four *corner-quads* which are labeled 0, 1, 2, and 3 such that *corner_quad* i is adjacent to *corner_quad*$((i + 1) \bmod 4)$ and *corner_quad*$((i + 3) \bmod 4)$, for $i = 0, 1, 2,$ and 3. In any two adjacent *corner-quads,* the live nodes are either connected by a healthy path or not. If the nodes in any two adjacent *corner-quads i* and $i + 1$ are connected then the result follows directly. Otherwise, three nodes in *corner-quads i* and $i + 1$ must be faulty and they must form a diagonal (for example the nodes $P_2, P_3,$ and s_2 in Fig. 6). In this case, however, only one more node in the FTBB can fail and this guarantees that there exists a path from *corner-quad i* to *corner-quad* $i + 1$ through *corner-quads* $((i + 3) \bmod 4)$ and $((i + 2) \bmod 4)$. □

Proposition 3 and the above lemma allows the design of a two-phase routing algorithm in which the path between a source and a destination may be determined distributively. That is, each node on the path may determine the next node from local information about the status of its neighbors. More specifically, if a node $p = p_n \cdots p_1$ receives a message addressed to a node $d = d_n \cdots d_1$ which is different from p, then it determines the next node on the message path by using the following information:

1) $x = x_n \cdots x_1 = p$ XOR d, the bitwise Exclusive OR of p and d.
2) For each node g adjacent to p within the current FTBB (the FTBB whose address is $p_n \cdots p_4$) an indication on whether g is faulty or is *1-accessible,* where a node is called *1-accessible* if it has only one nonfaulty neighbor in its FTBB.
3) The node $v = v_n \cdots v_1$ from which the message was received.

Given that a corner-quad contains only two spare nodes, no more than two nodes can fail in any *corner quad* in a *live* system. Since a *corner-quad* is a fully connected graph of four nodes, any two node failures within a corner quad results in a connected graph of two nodes. This implies that if we can route a message to a nonfaulty node in the *corner-quad* that contains the destination node, then it can be routed to its destination. The routing rules at p is a list of alternatives for the "next node" to which the message is to be sent. The alternatives are attempted in order, and an alternative is skipped if it specifies: 1) a faulty node, 2) the node v, 3) a node which is *1-accessible* unless this node is d or 4) an option that does not apply. The reason for not sending a message to v or to a *1-accessible* node which is not d is to prevent messages from possibly looping in a cycle.

Before describing the routing algorithm we define some functions that are necessary to explain the routing steps. $S_\alpha(x)$ is the spare node added to the $\text{face}^{(b)}_{1,\alpha}$, which contains primary node x. Note that the value of α can either be 2 or 3. The function $C_{1,2,\cdots,n}(x) = C_1(C_2(\cdots(C_n(x))\cdots))$. Function $T_i(d, p) = p_n \cdots d_i \cdots p_2 p_1$, replaces the ith bit of p by d_i. $T_{1,2,\cdots,n}(d, p) = T_1(T_2(\cdots(T_n(d, p))\cdots))$. HD$(x, y)$ is the Hamming distance between the nodes x and y, and HF(F_x, F_y) is the Hamming distance between two FTBB's F_x and F_y.

Routing Rules for primary nodes:

If there exists at least one j such that $x_j = 1$, then try the following alternatives in order: (message is not at the destination FTBB)

1) For $j = n, n - 1, \cdots 4$ try $C_j(p)$ if $x_j = 1$
2) $C_1(p)$
3) $S_2(p)$ and $S_3(p)$: if $C_2(p) = v$ then try $S_3(p)$ first, while if $C_3(p) = v$ then try $S_2(p)$ first
4) $C_2(p)$ and $C_3(p)$.

Else, try the following alternatives in order: (message is at the destination FTBB)

1) For $j = 3, 2, 1$ try $C_j(p)$ if $x_j = 1$
2) $S_\alpha(p)$, if HD$(p, d) \leq 2$ and $p, d \in \text{face}^{(b)}_{1,\alpha}$
3) any of the two adjacent spare nodes of p
4) For $j = 1, 2, 3$ try $C_j(p)$.

Routing Rules for spare nodes:

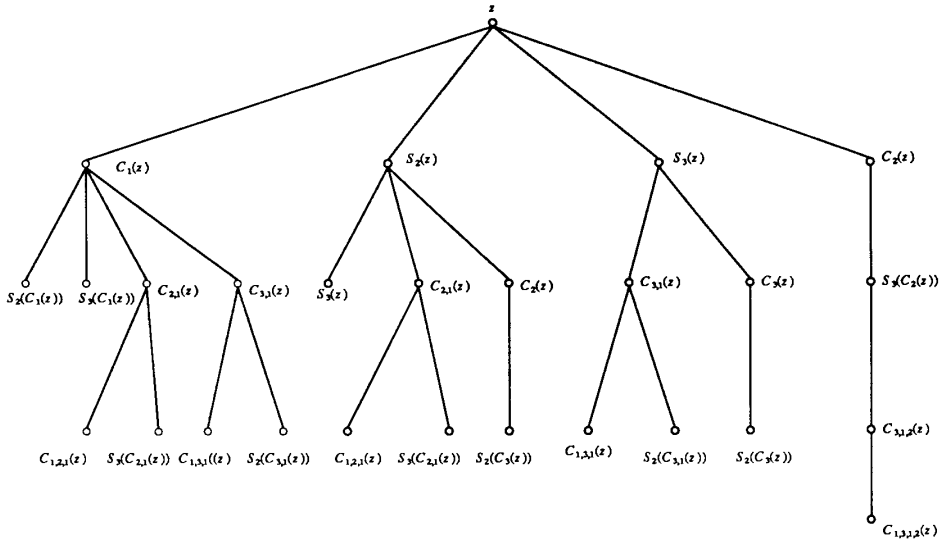

Fig. 9.

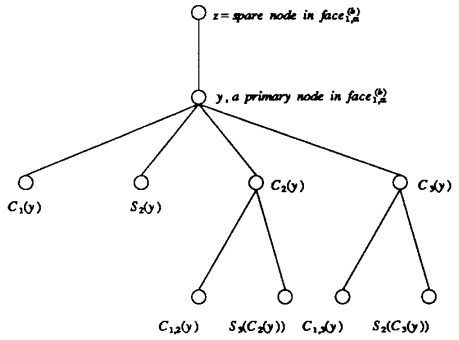

Fig. 10.

If there exists at least one j such that $x_j = 1$, then try the following alternatives in order: (message is not at the destination FTBB)

1) For $j = n, n-1, \cdots 4$ try $C_j(p)$ if $x_j = 1$
2) any adjacent primary node, if the message originates at p or came from a different FTBB
3) qtwin(p, v)
4) $C_1(C_\alpha(v))$ and then $C_\alpha(v)$, if $p \in \text{face}^{(b)}_{1,\alpha}$.

Else, try the following alternatives in order: (message is at the destination FTBB)

1) node d, if d is adjacent to p
2) adjacent spare s, such that $s_\alpha = d_\alpha$, if $p \in \text{face}^{(b)}_{1,\alpha}$
3) the other adjacent spare node in the current FTBB
4) $T_{1,\alpha}(d,p)$ and then $C_1(T_{1,\alpha}(d,p))$, if $p \in \text{face}^{(b)}_{1,\alpha}$
5) $C_1(T_\alpha(d,v))$ and then $C_{\alpha,1}(T_\alpha(d,v))$, if $p \in \text{face}^{(b)}_{1,\alpha}$.

In the above algorithm, two lists of alternative next nodes are used to route messages at each node. The first list is used for routing messages among FTBB's, and is applied whenever p is not in FTBB F_d, where $d \in f_d$. That is, whenever $x_j \neq 0$ for at least one j. $4 \leq j \leq n$. In this case, p is in the first phase of the message route. If, during this phase, routing from one FTBB to another fails then the algorithm tries to limit the routing trials, first to a *corner quad* and then to a *half-cube*. The reason is that limiting trials to a *corner quad* (or a *half-cube*) allows the message to reach the next FTBB in at most 3 steps (or 4 steps). If the fault distribution makes it impossible to stay within the same *half-cube* then at most 5 steps may be required to reach the next FTBB. The second list of alternatives in the routing algorithm is basically used to route a message to the destination node after the message is in the destination FTBB. Once the message reaches the destination FTBB, it takes at most 6 steps within the FTBB to reach d as shown in Lemma 4.

Lemma 3: In phase 1 of the routing algorithm, a message never loops in a cycle. Moreover, if a message starts at a node p in FTBB F_p, then it is routed to FTBB $C_j(F_p)$ in at most 5 steps, where $\text{HF}(C_j(F_p), F_d) = \text{HF}(F_p, F_d) - 1$.

Proof: We construct a tree starting from a given node p with all possible paths specified by phase 1 of the routing algorithm before a message is routed to a node in $C_j(F_p)$. We define 3 types of paths denoted by π_i, where $2 \leq i \leq 4$. The length of a π_i path is i and contains $i+1$ nodes. A π_2, π_3, and π_4 path contains distinct nodes from a *corner-quad*, a *half-cube*, and an FTBB, respectively. Therefore, a message taking a π_i path takes at most $i+1$ steps to reach a node in $C_j(F_p)$. This is because no more than 2, 3, and 4 nodes can fail in any *corner-quad* of $C_j(F_p)$, in any *half-cube* of $C_j(F_p)$, and in FTBB $C_j(F_p)$, respectively. From the construction of the routing trees if we can show that all the paths in the trees are of type π_i, where $2 \leq i \leq 4$, then the result follows directly.

Phase 1 of this routing algorithm may be initiated in F_p by a spare or a primary node. Therefore, we construct two trees given in Figs. 9 and 10. In Fig. 9, phase 1 is initiated at a primary node and in Fig. 10

it is initiated at a spare node. These two trees are constructed from all the possible paths that a message can take while still in F_z.

In Figs. 9 and 10 the ith branch from any given node is taken if branches $1, 2, \cdots, i-1$ specify a node that is faulty or *1-accessible.* For example in Fig. 10 the third branch from y towards $C_2(y)$ is chosen only when nodes $C_1(y)$ and $S_2(y)$ are faulty (the first and second branches, respectively, from y). We explain the construction of the tree in Fig. 10 in detail. The message originates from a spare node z, therefore, is routed to $C_j(z)$ if $C_j(z)$ is not faulty. Otherwise, the message goes to node y. Node y tries to route the message to $C_j(y)$ and may fail if $C_j(y)$ is faulty. If $C_j(y)$ is faulty then y tries to send the message to the nodes $C_1(y)$, $S_2(y)$, $C_2(y)$, and $C_3(y)$ in order. That is, if node $C_1(y)$ is faulty then y tries to send the message to $S_2(y)$ and then if $S_2(y)$ is faulty to $C_2(y)$ and so on. The paths $\{z, y, C_1(y)\}$ and $\{z, y, S_2(y)\}$ are of length 2 and each contains three distinct nodes from a *corner-quad.* Therefore, these paths are of type π_2. Similarly the paths $\{z, y, C_2(y), C_{1,2}(y)\}$, $\{z, y, C_2(y), S_3(C_2(y))\}$, $\{z, y, C_3(y), C_{1,3}(y)\}$ and $\{z, y, C_3(y), S_2(C_3(y))\}$ are of type π_3. In Fig. 9, the path from z to $C_2(z)$ is selected only if $C_1(z)$ is *1-accessible.* It can also be shown that all the paths in Fig. 9 are of type π_i, $2 \leq i \leq 4$. Therefore, the result follows. □

Lemma 4: In phase 2 of the routing algorithm, a message never loops in a cycle and is sent to the destination node in at most 6 steps.

Proof: This proof is similar to the one in Lemma 3. □

Theorem 1: The routing algorithm correctly routes the message to the destination in at most $5\gamma + 6$ steps if the Hamming distance between the source and the destination is γ.

Proof: By Lemma 3, $\mathrm{HF}(F_p, F_d)$ decreases by one after at most 5 steps. Therefore, the message reaches F_d in at most 5γ steps. Once the message is in F_d, then by Lemma 4, the message reaches the destination node in at most 6 steps. Therefore, the result follows directly. □

The upper bound on routing given in the above theorem is achievable only if the source and the destination nodes are in two FTBB's that are neighbors across some dimension j, and the eight faults in the two FTBB's are such that no two nodes across dimension j are faulty. The probability that the system reaches the above configuration is very small and thus the average number of routing steps is expected to be much less than this worst case bound. Also relatively complex routing rules are needed to primarily guarantee live-lock free routing. In typical fault configurations, only the first few alternatives in the list of "next nodes" needs to be examined. Finally, we note that the routing rules may be easily translated into a binary function which determines, at each node, the output link to which a message should be relayed in terms of x, v, and a vector which specify the status of the neighboring nodes. This function is relatively simple and may be implemented in hardware if fast message relaying is desired.

IV. Conclusion

Full spare utilization in FTBB's is desirable to achieve better reliability, but it may be very expensive to implement in large FTBB's. Therefore, given a specific hardware overhead, i.e., given a set of K, deg, and flex, the goal is to design a coverage scheme in FTBB's without *full spare utilization* such that the reliability is optimal. In Section II, we show that maximum module reliability is achieved by minimizing the coverage overlap, i.e., by maximizing sharing of the spare nodes. We have given a sufficient condition to minimize Φ. Necessary conditions may be formulated in terms of choosing M different subsets of $\{1, 2, \cdots, K\}$, each of size deg such that each integer j, $1 \leq j \leq K$, appears exactly in flex subsets.

We have also presented two modular approaches to achieve fault tolerance in binary hypercubes. The first approach is very flexible and uses switches to invoke a spare node that replaces a faulty primary node. In the second approach, no attempt is made to preserve the physical adjacency of the nodes in the hypercube. Instead, fault tolerant routing algorithms are used to route messages to their destinations, thus bypassing faulty nodes. We derive an upper bound on the number of routing steps it takes for a message to be delivered to its destination. This upper bound is derived for all possible fault configurations including some configurations that occur with very low probability. If message loss is allowed when one of these less probable configurations occur then much simpler algorithms may be designed and better bounds may be derived. The fault tolerant routing approach is very robust because it does not require switches and may tolerate the failure of routers and links. This robustness, however, is attained at the expense of routing overhead.

Finally, we note that both of these schemes can tolerate up to K faults per module since each module contains only K spare nodes. However, single point failures (e.g., power line failure) that affect all the nodes in a module cannot be tolerated unless the architecture incorporates spare modules. If a system is designed such that there are y spare modules for every x module, then a faulty module may be replaced by a spare module by applying the hardware switching mechanism that is described in Section III-A.

Appendix
Definitions

FTBB	Fault tolerant basic block
PS(S)	The set of primary nodes that spare node S can replace
SS(P)	The set of spare nodes that can replace primary node P
flex(S)	The cardinality of PS(S)
deg(P)	The cardinality of SS(P)
$\Phi(C)$	The number of deg + 1 fault combinations that a coverage scheme C cannot tolerate
$C_i(P)$	If $P = p_n, \cdots, p_i, \cdots, p_1$, then $C_i(P) = p_n, \cdots, \overline{p}_i, \cdots, p_1$
$T_i(d, p)$	If $d = d_n, \cdots, d_i, \cdots, d_1$ and $p = p_n, \cdots, p_i, \cdots, ps_1$, then $T_i(d, p) = p_n, \cdots, d_i, \cdots, p_1$
$C_{1,2,\cdots,n}(P)$	$C_{1,2,\cdots,n}(P) = C_1(C_2(\cdots(C_n(P))\cdots))$
F_p	Is the FTBB that contains the node p. If the address of p is $p_n, \cdots, p_1$, then the address of F_p is $p_n, \cdots, p_4$.
$T_{1,2,\cdots,n}(d, p)$	$T_{1,2,\cdots,n}(d, p) = T_1(T_2(\cdots(T_n(d, p))\cdots))$
HD(x, y)	Hamming distance between two nodes x and y
HD(F_1, F_2)	Hamming distance between FTBB F_1 and FTBB F_2
corner-quad	Fully connected graph of two spare and two primary nodes. For example in Fig. 7, the following four nodes form a corner-quad 1100, 1101, 110x, and 111x.
qtwin(p, s)	Denotes a spare node v, such that, if the primary node p and spare node s belongs to corner-quad q, then $v \in q$ and $v \neq s$. For example in Fig. 7, qtwin(1100, 110x) = 111x.
half-cube	Formed by two adjacent corner-quads. For example in Fig. 7, the following seven nodes along with their links form a half-cube 1100, 1110, 1111, 1101, 110x, 111x, and 101x.

References

[1] P. Banarjee, J.T. Rahmeh, C.B. Stunkel, V.S. Nair, K. Roy, and J.A. Abraham, "An evaluation of system level fault tolerance on the Intel hypercube multiprocessors," in *Proc. 18th. Int. Symp. Fault-Tolerant Comput.,* June 1988, pp. 362–367.

[2] D. Blough and N. Bagherzadeh, " A new fault tolerant routing algorithm for hypercube systems," in *Proc. Int. Workshop Hardware Fault Tolerance,* Urbana, IL, June 1989, pp. 52–54.

[3] S.-C. Chau and A.L. Liestman, "A proposal for a fault-tolerant binary hypercube architecture," in *Proc Int. Symp. Fault-Tolerant Comput.,* June 21–23, 1989, pp. 323–330.

[4] M. Chen and K. Shin, "Message routing in an injured hypercube," in *Proc. Fourth Hypercube Conf.,* pp. 312–317, 1988.

[5] E. Chow, H. Madan, and J. Peterson, "An adaptive message-routing network for the hypercube computer," in *Proc. 15th Symp. Comput. Architecture,* 1988, pp. 90–99.

[6] S. Dutt and J. P. Hayes, "An automorphic approach to the design of fault tolerant multiprocessors," in *Proc. 19th Int. Symp. Fault Tolerant Comput.,* June 1989, pp. 496–503.

[7] A. S. M. Hassan and V. K. Agrawal, "A fault-tolerant modular architecture for binary trees," *IEEE Trans. Comput.,* vol. C-35, no. 4, pp. 356–361, Apr. 1986.

[8] M. Howells and V. K. Agrawal, "A reconfigurating scheme for yield enhancement of large area binary tree architectures," *IEEE Trans. Comput.,* vol. C-37, no. 4, pp. 463–468, Apr. 1988.

[9] D. A. Rennels, "On implementing fault-tolerance in binary hypercubes," in *Proc. IEEE Fault Tolerant Comput.,* 1985, pp. 344–349.

[10] H. Sullivan and T. R. Bashkow, "A large scale homogeneous, fully distributed parallel machine, I," in *Proc. 4th Symp. Comput. Architecture,* Mar. 1977, pp. 105–117.

[11] R. S. Swarz and D. P. Siewiorek, *The Theory and Practice of Reliable System Design.* Bedford, MA: Digital, 1982.