Subcube Fault Tolerance in Hypercube Multiprocessors

Yeimkuan Chang, *Student Member, IEEE,* and Laxmi N. Bhuyan, *Senior Member, IEEE*

***Abstract*—In this paper, we study the problem of constructing subcubes in faulty hypercubes. First a divide-and-conquer technique is used to form the set of disjoint subcubes in the faulty hypercube. The concept of irregular subcubes is then introduced to take advantage of advanced switching techniques, such as wormhole routing, to increase the sizes of the available subcubes. We present a subcube partitioning technique to form an irregular subcube of maximum size. The n-cube containing two faults is studied first because, in the worst case, two faults are sufficient to destroy all the possible regular $(n - 1)$-cubes. It is shown that the subcube partitioning technique is able to tolerate $\lceil \frac{n}{2} \rceil$ faults while maintaining a fault-free $(n - 1)$-cube in a faulty n-cube. In general, we show that a fault-free $(n - m - 1)$-cube is guaranteed when there are $(\lceil \frac{n-m}{2} \rceil + 1) \times 2^m + 2^{m-1} - 1$ or fewer faults. We also develop a two-phase subcube allocation strategy in order to show the average case performance of our subcube construction technique. Extensive simulation is conducted to show the effectiveness of the two-phase subcube allocation strategy.**

***Index Terms*—Hypercube, subcube partitioning, fault tolerance, wormhole routing.**

I. Introduction

Hypercube architecture has received much attention because of its attractive properties which includes logarithmic network diameter, regularity, fault tolerance, embeddability, and multitasking capability [1], [2], [3]. As the size of a system grows, the probability of some processors or links in the system failing increases. Hence, executing these hypercube programs in the presence of faults is a critical issue. In a normal situation, many parallel programs are executed concurrently in different subcubes allocated by the host system [4], [5], [6], [7], [8]. The intent of this paper is to develop subcube allocation strategies in the presence of faults.

This paper attempts to provide strategies to overcome the effects of faulty elements and simulate a fault-free architecture on the faulty system. Hastad, Leighton, and Newman [9] proved that, with a high probability, the faulty hypercube can simulate the fault-free hypercube with only a constant factor slowdown. Other researchers studied the fault tolerance of hypercubes in the worst case of fault distributions [10], [11], [12]. The problem of determining the number of faulty processors and faulty links in an n-cube such that no m-cube is fault-free is considered in [10], [11]. Another approach proposed by Bruck, Cypher, and Soroker [12] uses subcube partitioning. It has been proven that an n-cube can tolerate more than $O(n)$ arbitrarily placed faults with a constant slowdown. In [13], Raghavendra, Yang, and Tien also used the subcube partitioning technique to effectively achieve the embedding of rings, routing, and global operations in faulty hypercubes. However, none of the above studies considers the construction of subcubes to preserve the multitasking capability in a faulty hypercube.

The multitasking capability of the hypercube allows several incoming tasks or programs to run concurrently in the system. Concurrency is achieved by assigning different subcubes to these incoming tasks. The main idea in assigning subcubes to incoming tasks is to avoid interference among different subcubes. This allocation problem of hypercubes has been studied in the literature. It was introduced by Chen and Shin [4], using the Gray code. Other researchers have proposed different strategies to tackle the problem, including the maximal set of subcubes (MSS) [5], free list strategy [6], tree collapse (TC) strategy [7], and graph-based approaches [14], [15]. The full subcube recognition ability and the efficiency of the allocation algorithms are the two main issues of the hypercube allocations [16].

In this paper, we address this allocation problem on the n-cube system with faults. We consider wormhole routing in addition to store-and-forward routing. The communication delay between any two nodes is almost the same irrespective of the distance between the two communicating nodes in the wormhole routing employed in current commercial n-cube systems [2], [3], [17]. This is subject to the condition that there is no interference of the messages on the communicating links. In this paper, we assume that all the faults are static and are detected before the reconfiguration algorithm starts. Also, we consider only node faults, and as a result, the links incident on the faulty nodes are also discarded. The faulty nodes cannot perform either computations or communications.

We study how to allocate subcubes in the faulty hypercube and thus maintain the multitasking capability of the system. A subcube partitioning technique is presented to facilitate subcube construction and allocation in the presence of faults. No spare nodes and links are used in the process of constructing subcubes. By using the subcube partitioning technique, a subcube is first selected, and then the faults in the selected subcube are replaced by the fault-free neighbors in the other subcube. We first discuss the allocation with two faults since, in the worst case, two faults in an n-cube are sufficient to destroy every possible regular $(n - 1)$-cube. The subcube partitioning technique used for a two-fault case is also applied to overcome the situation with more than two faults.

Manuscript received Aug. 23, 1993; revised Feb. 9, 1995.

Y. Chang is with the Department of Information Management at the Chung-Hua Polytechnic Institute, Taiwan, Republic of China

L.N. Bhuyan is with the Department of Computer Science, Texas A&M University, College Station, TX 77843-3112; e-mail: bhuyan@ cs.tamu.edu.

IEEECS Log Number C95091.

Finally, we develop a two-phase fault-tolerant subcube allocation strategy in the presence of faults. The first phase is the reconfiguration process based on the subcube partitioning technique which finds the set of disjoint subcubes in a faulty hypercube. The second phase applies an existing fault-free subcube allocation strategy, such as the Buddy strategy, to each disjoint subcube for assigning the available fault-free subcubes to the incoming tasks. Simulation results using the Buddy and single Gray code strategies [4] also show that the completion time and utilization of the two-phase approaches are superior to non-reconfiguration approaches.

The rest of the paper is organized as follows. Definitions and notations are given in Section II. In Section III, we show that the divide-and-conquer technique can be used effectively to form the set of disjoint subcubes. In Section IV, a subcube partitioning technique is proposed to construct the irregular subcubes. A two-phase subcube allocation strategy is presented in Section V. The experimental results are presented in Section VI. Concluding remarks are given in the last section.

II. Preliminaries

A hypercube of dimension n, denoted by Q_n, consists of 2^n nodes. Q_n can be topologically represented as an n-dimensional cube in which nodes are located on the 2^n vertices of the cube. Each of the 2^n nodes is addressed by a distinct binary string, $l_{n-1}l_{n-2}...l_0$, with bit l_i corresponding to dimension i and $l_i \in \{0, 1\}$. Two nodes are connected by a link if and only if their addresses differ in exactly one bit. The *Hamming weight* of a node in Q_n is defined as the number of ones in its address.

Each subcube can be represented uniquely as a ternary string over the set {0, 1, *}, called its address, where * is a *Don't Care* symbol. Specifically, a d-dimensional subcube Q_d, called *d-cube*, has exactly d *s in its address, as it involves a group of 2^d nodes. For example, 000**, or equivalently 0^3*^2, represents the 2-cube formed by nodes 00000, 00010, 00001, and 00011 in a 5-cube. We define Q_d^i to be the d-cube that has the same address as Q_d except that the ith bit of Q_d^i is complement of the ith bit of Q_d. So, we call Q_d^i as the *complementary* d–cube of Q_d across dimension i. We refer to the subcube defined above as a *regular* subcube. We also introduce the following definition.

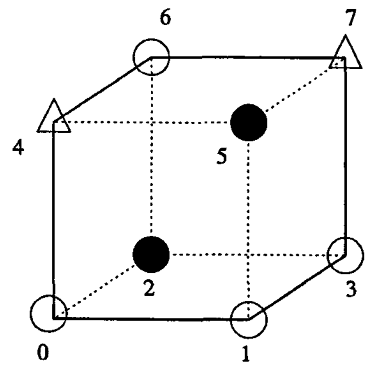


Fig. 1. An example of an irregular 2-cube.

An irregular subcube is defined as a subcube in which there exist one or more pairs of logically adjacent nodes such that the physical distance between the adjacent nodes is greater than one. Fig. 1 gives an example of an irregular 2-cube in a 3-cube with two faulty nodes. The faulty nodes, 2 and 5, are marked as shaded circles. The fault-free nodes 0, 1, 3, and 6, marked as empty circles, are used to form the 2-cube. Since nodes 0 and 3 are not physically adjacent to node 6, the communication paths between 0 and 6 and between 3 and 6 go through nodes 4 and 7, respectively. Nodes 4 and 7 are called *intermediate* nodes and are marked as triangles.

Efficient point-to-point routing is a key to the performance of hypercube multiprocessors. The current commercial hypercube multiprocessors are all based on a dimension ordered (e-cube) routing scheme which assumes a fault-free system [2], [3]. Since no hardware modification is allowed on the routers of existing machines, a software approach is needed to route the messages on faulty systems. As an example, the table-filling technique [18] can be applied based on the e-cube routing algorithm. In the table-filling method, the messages are routed from a source node to a destination node through the logical path between them. If the logical path is the same as the path routed by e-cube routing, no change is necessary. However, if they are different, an intermediate node, called a *bypassing* node, is used to receive the message and reroute it to the destination node. The table records whether or not a bypassing node is used. A centralized or distributed algorithm can be used to fill the table with necessary information about bypassing nodes. If modification on routers is allowed, slightly modifying the current router design can achieve an optimal routing performance. According to the proposed subcube partitioning technique, the messages entering an input channel of an intermediate node always go out through a specific output channel in a reconfigured subcube. Therefore, the input channel in an intermediate node is directly connected with its corresponding output channel. By direct connection, we mean an extra setting is added in the routers such that the messages from an input channel are forced to go through its corresponding output channel without executing the e-cube algorithm.

In this paper, we consider two different designs of a node in the hypercube multiprocessor. Each node is responsible for both computation and communication in the first design. The intermediate nodes are not used for computation and are devoted entirely to communication in the reconfigured subcube. In the second node design, each node contains two units, namely, a computation unit and a communication unit that can operate independently [2], [3]. The communication unit often is called a router. The routers have the capability of forwarding the messages from other routers toward the destination nodes. Usually there is an $(n + 1) \times (n + 1)$ crossbar switch inside the router, as shown in Fig. 2 [19], [20]. The switching interface in routers includes the control logic for enabling the crossbar switch. One of the input and output channels of the $(n + 1) \times (n + 1)$ crossbar switch is used for connection between the computation unit and the router. When such a node design is used, the intermediate nodes in Fig. 1 also can be used to form other subcubes in addition to providing a path for the irregular subcubes. Since using a crossbar is very expensive, an alternative design uses a *shared-bus* as a logical crossbar, as does the nCUBE [2].

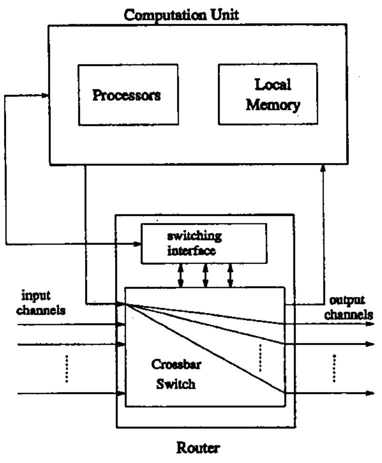


Fig. 2. Hypercube node architecture.

The hypercube multiprocessors based on wormhole routing [21] is considered in this paper. Wormhole technology has been shown to be insensitive to the distance between the sender and receiver as long as there is no interference in the links by other messages. We will assume no such interference in the system and refer to this property as *wormhole routing property*. In the shared-bus router design, data is shifted in and out of the communication ports one bit per cycle. This bit-serial data is collected into a complete flit at a receive port and is then written to memory, if this is its destination, or it is cut-through to the appropriate output port if it is to be sent on to another node. When a flit is ready to cut-through to another node, it must check the next output port to see if it is ready and then arbitrate for the shared-bus to move the flit. Since arbitration and bus allocation can occur every cycle, and since no receive port will ask for a cut-through more often than every 36 cycles (because it takes 36 cycles minimum to shift in a flit, as in nCUBE [2]), the possibility of two or more flits completely arriving at one router at the same cycle is small. Thus, multiple input ports can cut through to different output ports with little or no conflict of messages and the performance difference between a shared-bus and crossbar switch design is negligible.

As stated earlier, we will address the fault-tolerant subcube allocation problem. In [5], Dutt and Hayes define the maximum set of subcubes (MSS) to be a set of disjoint free subcubes in a hypercube that is *greater* than or equal to all other sets of disjoint subcubes in the same hypercube. A set of disjoint subcubes, A, is said to be greater than another set of disjoint subcubes, B, if there exists an i such that for all $j > i$, the number of subcubes of size j in A is equal to that in B, and the number of subcubes of size i in A is greater than that in B. Thus, if we treat busy nodes as the faulty nodes in the faulty hypercube, the problem of determining the maximum set of disjoint fault-free subcubes in a faulty hypercube is equivalent to determining the MSS. However, determining MSS is an NP-complete problem [5]. Hence, in this paper, we will develop an efficient algorithm to find the set of disjoint subcubes. We use a weight vector, which is defined below, to find the set of disjoint fault-free subcubes in a faulty hypercube. Although the set of disjoint fault-free subcubes found by means of weight vectors may not be the same as the MSS, using weight vectors is efficient.

We define the *weight vector* of an n-cube as $W = (w_1, w_2, ..., w_n)$ where w_i is the number of pairs of faulty nodes such that the Hamming distance between the two faulty nodes in each pair is i, for $1 \le i \le n$. The weight vector $W = (w_1, w_2, ..., w_n)$, of an n-cube is said to be *greater than or equal to* that of another n-cube, $Z = (z_1, z_2, ..., z_n)$, if there exists an i such that for all $j \le i$, $w_j \ge z_j$. For example, the weight vector of a 3-cube with three faulty nodes, 000, 010, and 100, is (2, 1, 0) since two pairs of faulty nodes have a Hamming distance one and one pair has a Hamming distance of two. The weight vector (2, 1, 0) is greater than that of a 3-cube containing three faulty nodes at 000, 101, and 111, which is (1, 1, 1).

III. Construction of Regular Subcubes

The aim of this section is two-fold. First, we find a subcube of the maximum size in a faulty hypercube. Secondly, we find the set of disjoint subcubes (SDS) such that independent incoming tasks can run on these disjoint subcubes simultaneously. We present an efficient algorithm for constructing SDS in a faulty hypercube by using weight vectors.

Let us consider the availability of disjoint subcubes while there are only one or two faulty nodes in an n-cube. For an n-cube containing one fault, it can be split into two $(n - 1)$-cubes, one fault-free $(n - 1)$-cube and the other faulty $(n - 1)$-cube containing one faulty node. If the above splitting process continues on the faulty subcubes, we can obtain the SDS composing of a set of subcubes of dimensions from 0 to $n - 1$. For an n-cube containing two faults, we first discuss the worst case scenario; i.e., the two faulty nodes are located at antipodal positions of the n-cube, and as a result they destroy all possible $(n - 1)$-cubes [10]. In this worst 2-fault case, we can always obtain two faulty $(n - 1)$-cubes, each containing one faulty node. Then according to the result from 1-fault case, we have the *SDS* containing two subcubes of dimension i, for all $i = 0$ to $n - 2$. In general, if the Hamming distance of the two faulty nodes is r for $1 \le r \le n$, we can obtain the SDS that contains one subcube of dimension i for all $i = r$ to $n - 1$ and two subcubes of dimension j for all $j = 0$ to $r - 2$.

For an n-cube with any number of faulty nodes, the algorithm is based on the divide-and-conquer technique and is given in the procedure Form_SDS in Fig. 3. The n and F denote the dimension and the set of faults of the hypercube, respectively. The procedure Form_Regular_n_1_cube returns a value d, which is a dimension index such that one of the two $(n - 1)$-cubes obtained from splitting the n-cube at dimension d contains the minimum number of faulty nodes compared with other possible $(n - 1)$-cubes. We split the n-cube into two $(n - 1)$-cubes with faults F_0 and F_1 at dimension d. The same algorithm is applied recursively to find all the fault-free subcubes. While partitioning the n-cube, if there exist two $(n-1)$-cubes such that both contain the same minimum number of faults, we solve it as follows. The Hamming distance between any two faulty nodes in an $(n-1)$-cube is first computed. We select an $(n-1)$-cube whose weight vector is greater than or equal to the weight vector of any other $(n-1)$-cube having the same number of faulty nodes. If there exist more than one $(n-1)$-cubes that have the same minimum number of faulty nodes and same maximum weight vector, we select the $(n-1)$-cube Q_{n-1} such that the weight vector of the complementary $(n-1)$-cube of Q_{n-1} is greater than or equal to the weight vector of the complementary $(n-1)$-cube of other $(n-1)$-cubes. However, if there is a tie again, we arbitrarily select one $(n-1)$-cube. The reason for choosing an $(n-1)$-cube with the maximum weight vector is that a large weight vector actually indicates the degree of the faulty nodes being close to each other. We shall see below that if the faulty nodes are close to each other then the probability of forming a large subcube inside the $(n-1)$-cube becomes large. Notice that the two if statements in Lines 2 and 5 of Fig. 3 are the conditions that terminate the recursion.

```
Procedure Form_SDS($n$, $F$)
/* $n$ is the dimension of the hypercube and $F$ is the set of faults */
begin
1       $d$ := Form_Regular_n_1_cube($n, F, F_0, F_1$);
2       if $|F_0| = 0$ then
3           SDS := SDS $\cup *^{n-d-1}0*^{d}$;
4       else Form_SDS($n-1, F_0$);
5       if $|F_1| = 0$ then
6           SDS := SDS $\cup *^{n-d-1}1*^{d}$;
7       else Form_SDS($n-1, F_1$);
end
```

Fig. 3. Forming the set of disjoint regular subcubes.

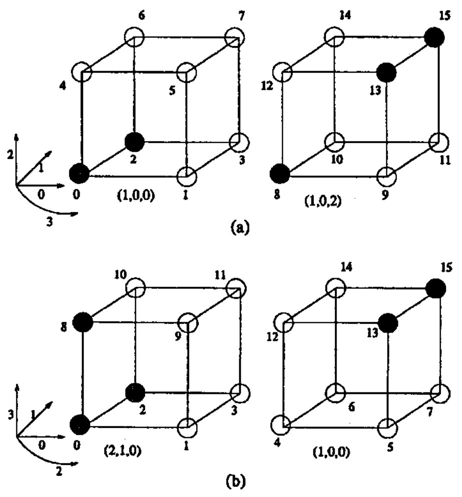


Fig. 4. Selection of a dimension for constructing regular subcubes.

EXAMPLE 1: Fig. 4 shows a 4-cube containing 5 faulty nodes, marked as shaded circles. Fig. 4a and b illustrate that the 4-cube is split into two 3-cubes along dimensions 2 and 3, respectively. It may be seen that two 3-cubes, 0*** of Fig. 4a and *1** of Fig. 4b, have the same minimum number of faulty nodes which is 2. The 3-cube, *1* * is selected since the weight vector of *0**, the complementary 3-cube of *1**, is (2, 1, 0), which is greater than (1, 0, 2), the weight vector of the complementary 3-cube of 0***. In other words, dimension 2 is selected for splitting the n-cube. After dimension d is selected, the procedure Form_SDS is continued for each $(n-1)$-cube until the subcube contains no faults or all the processors in the subcube are faulty. Thus, we obtain two 2-cubes, one 1-cube, and one 0-cube, which are better than one 2-cube, three 1-cubes, and one 0-cube obtained by splitting dimension 3 first.

The time complexity of Form_SDS is analyzed as follows. Form_Regular_n_1_cube takes $O(|F|^2 \times n^2)$ time units since $O(|F|^2 \times n)$ time units are used for computing the Hamming distance of $O(|F|^2)$ pairs of faulty nodes and $2n$ $(n-1)$-cubes are considered. The time taken from line 2 to line 7 in Fig. 3 depends on the locations of faulty nodes. Consider one extreme case where an n-cube with $|F|$ faults can only be split into two $(n-1)$-cubes with $\frac{|F|}{2}$ faults each. Recursively calling Form_SDS log $|F|$ times is sufficient to form the SDS. Thus, the time complexity of Form_SDS $T(n, |F|) = O(|F|^2 \times n^2) + T(n-1, \frac{|F|}{2})$. As a result, $T(n, |F|) = O(|F|^3 \times n^2)$. Consider another extreme case where all $|F|$ faults locate in a log $|F|$-cube. Calling Form_SDS $(n - \log |F|)$ times is sufficient to form the SDS. Thus, the time complexity of Form_SDS $T(n, |F|) = O((n - \log |F|) \times |F|^2 \times n^2)$. If log $|F|$ is much smaller than n, the complexity of Form_SDS is $O(|F|^2 \times n^3)$. Assume that $|F| = O(n)$. The time complexity of Form_SDS is $O(n^5)$ since the time complexities of all the other possible cases are between the ones of above two extreme cases. We shall see that Form_SDS does not guarantee to find the maximum set of subcubes (MSS) introduced by Dutt and Hayes [5]. However, our method, shown in Fig. 3, is efficient whereas finding the MSS is an NP-hard problem.

Let us analyze the maximum number of faults that the procedure Form_SDS can tolerate in order to maintain a fault-free subcube. After calling Form_SDS once, there exists an $(n-1)$-cube containing at most $\left\lfloor \frac{|F|}{2} \right\rfloor$ faulty nodes. Thus Form_SDS can continue m times until the finally selected $(n-m)$-cube contains only one fault, for $|F| \leq 2^m$. Obviously, a fault-free $(n-m-1)$-cube is available in an n-cube containing $|F| \leq 2^m$ faults, for $n \geq m + 1$. However, if n is much larger than m, we derive a better result as follows.

LEMMA 1. *Given 2^{n-1} faulty nodes in an n-cube, there always exists a fault-free 1-cube except when the Hamming weights of the faulty nodes are all even or all odd.*

PROOF. If the two nodes at the ends of an edge are both healthy, the lemma follows. Next, suppose that the two nodes at the ends of an edge e are both faulty. Then look at the 2^{n-1} edges in the same dimension as e. So at least one of these 2^{n-1} edges has both ends healthy. The lemma follows.

Now suppose no edge has its two ends both healthy or both faulty. Therefore, a node is healthy if and only if its neighbors are all faulty. Then, all odd weighted nodes are healthy and all even weighted nodes are faulty or vice versa. □

LEMMA 2. *Given 2^m or fewer faulty nodes in an n-cube, where $n \geq m + 2$ and $m \geq 2$, there always exists a healthy $(n - m)$-cube in the n-cube.*

PROOF. We only have to consider the case when there are exactly 2^m faulty nodes. Since $2^m \geq 4$, we claim that at least two faulty nodes have the same bit value at one or more dimension of their addresses. For example, the two faulty nodes, 0000 and 1101, have the same bit value, 0, at dimension 1. In fact, for each node, there is exactly one node, i.e., its corresponding antipodal node, that has a different bit value at all dimensions although we have $2^m \geq 4$ faulty nodes.

Let t_1 and t_2 be faulty nodes having the same bit value at dimension d. We partition the n-cube into 2^m $(n - m)$-cubes such that dimension d is internal to these $(n - m)$-cubes. So, some $(n - m)$-cube contains at least two faulty nodes t_1 and t_2. As a result, some other $(n - m)$-cube must be completely healthy. □

THEOREM 1. *There must be at least $2^m + 2^{m-2}$ faulty nodes in an n-cube in order to destroy all the possible $(n - m)$-cubes, where $n \geq m + 2$, and $m \geq 2$.*

PROOF. We prove that there exists an $(n - m)$-cube in an n-cube with $2^m + 2^{m-2} - 1$ faulty nodes. By partitioning the n-cube into 2^{m-2} $(n - m + 2)$-cubes, there must exist an $(n - m + 2)$-cube which contains at most 4 faulty nodes. Thus, according to Lemma 2, there exists a fault-free $(n - m)$-cube in the $(n - m + 2)$-cube with 4 faulty nodes where $n \geq m + 2$. □

Theorem 1 essentially states that more faults can be tolerated for maintaining a fault-free $(n - m)$-cube if n is much larger than m. The bigger the n-cube, the more the chance we can form a bigger subcube. The minimum number of faulty nodes that causes no available fault-free $(n - m)$-cube is greater than or equal to $2^m + 2^{m-2}$ according to Theorem 1. The number of faulty nodes that can be tolerated by our method is not as good as the lower bound provided in [10], which is $O(2^{m-2} \times \lceil 1.6\log(n - m + 3) - 2.1 \log\log(n - m + 3) - 4.5 \rceil)$. However, the proposed subcube partitioning technique gives us an efficient way to find the subcubes, whereas no procedure has been provided in [10] for finding the subcubes to meet the lower bound.

IV. CONSTRUCTION OF IRREGULAR SUBCUBES

In this section, we develop a subcube partitioning technique so that the maximum subcube size can be greatly increased for the same number of faulty nodes. To construct an irregular subcube of dimension d, the underlying principle is to find a d-cube, Q_d, such that the faulty nodes in Q_d can be replaced by some fault-free neighbors in a complementary d-cube of Q_d. As shown in Fig. 1, the logical topology of Q_d is maintained. We refer to the replacing nodes in the complementary d-cube as the *image nodes* of faulty nodes in Q_d. Since the image nodes in the complementary d-cube of Q_d replace the faulty nodes in Q_d, the paths between the neighboring nodes of the faulty nodes and the image nodes must be fault-free.

In the terminology of graph embedding, the subcube partitioning technique attempts to embed a fault-free $(n - 1)$-cube into a faulty n-cube. Thus, the subcube partitioning technique is a dilation-2 and congestion-1 embedding.[1] The load of this embedding depends on the number of faulty nodes in the system. The proposed subcube partitioning technique utilizes the unused links to form the disjoint subcubes of different sizes. Thus, the tasks running on the subcubes do not interfere with each other. For simplicity, let us start with the situation when there are only two faults in an n-cube. Later on, we will extend the results to an n-cube with more faults. It is known that two faulty nodes at antipodal positions are sufficient to destroy all the possible $(n - 1)$-cubes [10]. Thus, we first demonstrate how our subcube partitioning technique guarantees an $(n - 1)$-cube in an n-cube with two faults.

Without loss of generality, assume that the two faulty nodes in an n-cube are 0^n and 1^n. The regular $(n-1)$-cube, $1*^{n-1}$, is first selected. Then a fault-free irregular $(n-1)$-cube can be constructed from $1*^{n-1}$ by replacing the faulty node 1^n with its image node 01^{n-1} in $0*^{n-1}$. The links between 01^{n-1} and its neighbors and the links between the neighbors of 01^{n-1} and the corresponding neighbors of 1^n are utilized to form the fault-free $(n-1)$-cube. Let us consider the general case of an n-cube with two faults, f_0 and f_1, at any locations. Assume that the distance between f_0 and f_1 is r. Then we can easily find disjoint regular subcubes of dimension d for $r \leq d < n$ by splitting the n-cube across dimensions at which the bit values of f_0 and f_1 are the same. The remaining faulty subcube is r-dimensional with two faulty nodes at antipodal positions. It is easy to verify that the remaining faulty r-cube has no interference on these regular subcubes that were addressed. The following example shows how our technique is applied to the 2-fault case.

EXAMPLE 2. Assume that there are two faulty nodes at antipodal positions of a 4-cube, i.e., 0000 and 1111 as shown in Fig. 5. First, the 3-cube 1*** is selected. Then the faulty node 1111 in 1*** is replaced by its image node 0111 in the other 3-cube. An irregular 3-cube is formed by the links shown as bold lines and the nodes marked with Label 3 in the parentheses. The intermediate nodes are nodes 0011, 0101, and 0110. Thus, intermediate nodes may or may not be used to form smaller disjoint subcubes depending on the node design. The intermediate nodes are not used to form other subcubes in the design where nodes are responsible for both computation and communication. Thus, the three intermediate nodes are wasted. If the intermediate nodes have separate routers as shown in Fig. 2, a 2-cube consisting of nodes 0100, 0101, 0110, and 0011 can be formed. The remaining 2 nodes, 0001 and 0010, form two 0-cubes.

1. Dilation of an embedding is defined as the maximum physical distance between two logically adjacent nodes. The congestion and load of an embedding are defined as the maximum number of logical links and nodes sharing a physical link and node, respectively.

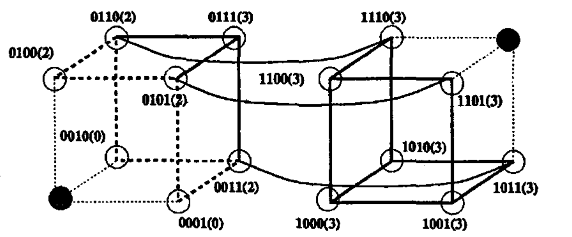


Fig. 5. Subcube allocations of 4-cube with two faults at antipodal positions.

To generalize the above result, we first have the following theorem which assumes that the intermediate nodes cannot be used and $N(n)$ is the set of disjoint fault-free subcubes in an n-cube containing two faults located at antipodal positions.

THEOREM 2. *Given two faulty nodes at the antipodal positions of an n-cube,*

$$N(n) = Q_{n-1} + N(n-2) + \sum_{k=1}^{n-3} (kQ_{n-k-3}).$$

PROOF. Consider an n-cube with two faults 0^n and 1^n at antipodal positions. As stated above, an irregular $(n - 1)$-cube can be obtained by replacing the faulty node 1^n in $1*^{n-1}$ with its image node 01^{n-1} in $0*^{n-1}$. Since the $n - 1$ intermediate nodes cannot be used to construct the other subcubes, the remaining $(n - 1)$-cube can be treated as an $(n - 1)$-cube containing $n + 1$ faulty nodes, which are 0^n, 01^{n-1}, and all the neighbors of node 01^{n-1} in $0*^{n-1}$, i.e., 01^i01^{n-i-2} for $0 \le i \le n - 2$. The remaining faulty $(n - 1)$-cube can be further divided into an $(n - 2)$-cube containing two faults at antipodal positions, and an $(n - k)$-cube containing one fault, for all $k = 3$ to $n - 1$. Notice that an $(n - k)$-cube with one fault leads to a group of fault-free disjoint subcube of dimension $n - k - i$, where $1 \le i \le n - k$. If we sum all the subcubes of the same size up, the theorem is proved. □

For the node design with router, we consider the general case of an n-cube with two faults, f_0 and f_1, whose distance is r. Then we can easily find disjoint regular subcubes of dimension d for $r \le d < n$ by splitting the n-cube across dimensions at which the bit values of f_0 and f_1 are the same. The remaining faulty subcube is r-dimensional with two faulty nodes at antipodal positions. Without loss of generality, assume that faulty node f_0 is $1^{n-r}0^r$ and f_1 is $1^{n-r}1^r$. An irregular $(r - 1)$-cube can be constructed as described previously by selecting the $(r - 1)$-cube, $1^{n-r}0*^{r-1}$, and replacing node $1^{n-r}0^r$ with its image node, $1^{n-r}10^{r-1}$. The remaining $(r - 1)$-cube, $1^{n-r+1}*^{r-1}$, can be treated as an $(r - 1)$-cube with two faulty nodes at antipodal positions $1^{n-r+1}0^{r-1}$ and $1^{n-r+1}1^{r-1}$. This will avoid the interference between the already constructed regular and irregular subcubes, and any new subcubes to be formed. Fig. 5 shows an example for $r = 4$. By induction, the following theorem is derived.

THEOREM 3. *Given any two faulty nodes in an n-cube, two 0-cubes and one d-cube for all d = 2 to n − 1 can be constructed.*

When compared to the corresponding results in Section III, it can be seen that a 2-fault situation in an irregular subcube construction has similar characteristics of a 1-fault case in regular subcube construction. The foregoing results solve the worst case problem for an n-cube with two faulty nodes at antipodal positions, while maintaining a fault-free $(n - 1)$-cube. However, our aim is to find the maximum number of faulty nodes under our proposed subcube partitioning technique for a fault-free $(n - 1)$-cube to exist in a faulty n-cube. Remember that in the 2-fault case, a certain $(n - 1)$-cube, Q_{n-1}, is first selected. By careful observation, we have the following criteria to select an $(n - 1)$-cube such that the faulty nodes in the selected $(n - 1)$-cube, Q_{n-1}, can be replaced by the fault-free nodes in the complementary $(n - 1)$-cube, Q'_{n-1}, and also the hypercube topology is maintained.

1) If there are two faulty nodes that have a distance of 1 or 2 between them, then Q_{n-1} should contain either none or both of these two faulty nodes.
2) If Q_{n-1} contains two faulty nodes that have a distance of 2 between them, then the nodes, which are adjacent to both of these two faulty nodes in Q_{n-1} must be replaced by their fault-free image nodes in Q'_{n-1}.
3) The replacing fault-free nodes cannot have more than one faulty neighbor.

Criterion 1 rules out the following two situations. First, a faulty node in Q_{n-1} has a faulty image node in Q_{n-1}'. This situation causes the replacing node to be faulty. The second situation is that the image node is fault-free, but one of the neighbors of the image node is faulty. This situation causes one of the intermediate nodes to be faulty. We describe the process of selecting an $(n - 1)$-cube as *selecting a dimension* for *splitting the n-cube* into two $(n - 1)$-cubes such that one of these two $(n - 1)$-cubes satisfies the above three criteria. Thus, the first criterion essentially prevents one (or two) dimension(s) which is (or are) internal to these two faulty nodes from being selected. The second criterion states that if the selected $(n - 1)$-cube contains both the faulty nodes separated by a distance of 2, the nodes adjacent to both faulty nodes also must be replaced since each of the vertices of an $(n - 1)$-cube must have a degree of $n - 1$. In other words, we replace the 2-cube in Q_{n-1} spanned by these two faulty nodes by the corresponding 2-cube in the Q_{n-1}'. Note that the replacing nodes must also have a degree of at least $n - 1$. Thus, Criterion 3 is applied to avoid the replacing nodes having a degree less than $n - 1$, i.e., having more than one faulty neighbor.

EXAMPLE 3. We consider a 5-cube with six faulty nodes as shown in Fig. 6. Here, dimensions 0 and 2 are not selected due to faulty nodes 11001 and 11100, and dimensions 1 and 3 are not selected due to faulty nodes 11100 and 10110. Thus, dimension 4 is selected with the 4-cube 0**** since it contains only two faulty nodes, whose distance is 2. The only way to form a fault-free 4-cube is to replace the 2-cube 000** by 100**. However, one of the replacing nodes, 10001, has two faulty neighhbors 10101 and 11001. Since both of the nodes 00001 and 10001 have two faulty neighbors, we can't form a fault-free irregular 4-cube based on either 0**** or 1****.

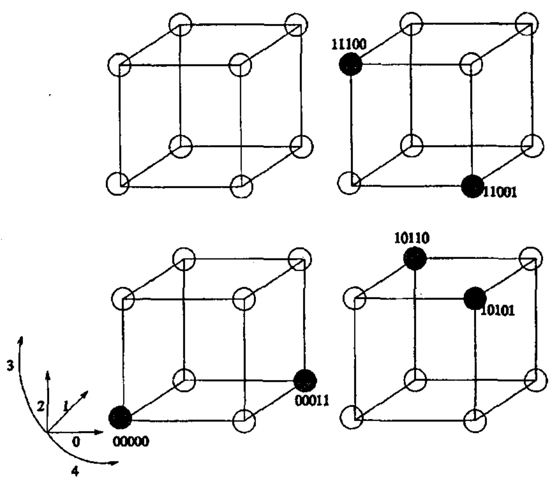


Fig. 6. Illustration of constructing irregular subcubes.

Now let $k(n, m)$ be the maximum number of faulty nodes in an n-cube such that the proposed scheme can find a fault-free irregular $(n - m)$-cube. We give our main result as follows.

THEOREM 4. $k(n, 1) = \lceil \frac{n}{2} \rceil$ *for all* $n \geq 3$.

PROOF We prove the theorem by induction on n. For $n = 3$, it is trivial with two faults. By induction, we assume that there exists a fault-free $(k - 1)$-cube in a k-cube for $k \leq n - 1$. We will show that there exists a fault-free $(n - 1)$-cube in a faulty n-cube containing $\lceil \frac{n}{2} \rceil$ faults as follows. Any dimension can be selected for splitting the n-cube and forming an $(n - 1)$-cube such that all the three criteria are satisfied if the distance between any two faulty nodes is greater than 2. Hence, examining the case where the distance between the pair of faulty nodes is 1 or 2 is important.

(1) Let the distance between the two faulty nodes be 1. Without loss of generality, we assume these two faulty nodes are 10^{n-1} and 00^{n-1}. Note that the dimension that spans 10^{n-1} and 00^{n-1} is $n - 1$. We consider the original n-cube as an $(n - 1)$-cube in which the vertices are called *supernodes* consisting of two original nodes across the dimension $n - 1$. We say that a supernode is faulty if it contains at least one faulty original node. Thus, this $(n - 1)$-cube contains at most $\lceil \frac{n-2}{2} \rceil$ faulty supernodes. By induction there exists a fault-free $(n - 2)$-cube. Since each node of the $(n - 2)$-cube is a 1-cube, an $(n - 1)$-cube is thus formed.

(2) Let the distance between the two faulty nodes be 2. Without loss of generality, we assume these two faulty nodes are 110^{n-2} and 000^{n-2}. The dimensions which span these two faulty nodes are n–1 and n–2. Similarly as in (1), we consider the original n-cube as an $(n - 2)$-cube in which the supernodes are composed of original 2-cubes across the dimensions $n - 1$ and $n - 2$. Thus, the $(n - 2)$-cube contains at most $\lceil \frac{n-2}{2} \rceil$ faulty supernodes. By induction, there exists an $(n - 1)$-cube and the theorem is proved. □

Theorem 4 also states that in the worst case $\lceil \frac{n}{2} \rceil + 1$ faulty nodes can destroy all the $(n - 1)$-cubes. However, the simulation results presented in Section VI indicate that more faulty nodes can be tolerated in the average case for larger systems.

Subsequently, we will determine $k(n, m)$ for $m > 1$. The divide-and-conquer technique is applied to get the lower bound of $k(n, m)$. For example, $k(n, 2) \geq 2(\lceil \frac{n-2}{2} \rceil) + 1$ because there exists an $(n - 1)$-cube that contains at most $\lceil \frac{n-2}{2} \rceil$ faulty nodes. However, we can prove that $k(n, 2) \geq 2(\lceil \frac{n-1}{2} \rceil + 1)$, which is very close to the value obtained by an exhaustive search of a computer program. This result can also be used to obtain $k(n, m)$ for $m > 2$. First we need the following lemma:

LEMMA 3. $k(n, 2) \geq 2(\lceil \frac{n-1}{2} \rceil + 1)$ *for* $n = 5, 6$, *or* 7.

PROOF. We show this lemma by contradiction. We first assume that there is no fault-free $(n - 2)$-cube in an n-cube with $2(\lceil \frac{n-1}{2} \rceil + 1)$ faults. For each dimension, we split the n-cube into two $(n-1)$-cubes. We will examine whether these two $(n-1)$-cubes contain the same number of faulty nodes, i.e. $\lceil \frac{n-1}{2} \rceil + 1$ faulty nodes. Then we examine whether there exists a fault-free $(n - 2)$-cube in these two $(n - 1)$-cubes.

(1) $n = 5$: We first create a set of faulty nodes such that the two 4-cubes, $0*^4$ and $1*^4$ contain three faulty nodes and no fault-free 3-cubes are available in $0*^4$ and $1*^4$. Without loss of generality, we assume that the 4-cube $0*^4$ contains three faulty nodes $f_1 = 00000, f_2 = 00011$, and $f_3 = 01100$, and $1*^4$ contains three faulty nodes $f_4 = 10000 \oplus e, f_5 = 10011 \oplus e$, and $f_6 = 11100 \oplus e$, where $e = 0l_3l_2l_1l_0$, $l_i = 0$ or 1 for $0 \leq i \leq 3$), and $\oplus$ is the bitwise *exclusive-or* operation. Since we want two 4-cubes obtained from splitting the 5-cube at any dimension to contain the same number of faulty nodes, e must be 11111. By examining dimension 0, we see that there exists a fault-free 3-cube in $*^40$ which contains faulty nodes $f_1 = 00000$, $f_3 = 00011$, and $f_5 = 11100$, when $e = 01111$. This is a contradiction. Thus the lemma follows.

(2) $n = 6$ and 7: The proofs are similar to (1) and thus omitted. □

LEMMA 4. $k(n, 2) \geq 2(\lceil \frac{n-1}{2} \rceil + 1)$ *for* $n \geq 5$.

PROOF. We prove this lemma by induction on n. The basis of the induction is proven in Lemma 3. We then assume that the lemma follows for the hypercube of size $n - 1$ or less. According to Theorem 4, $\lceil \frac{n}{2} \rceil + 1$ faulty nodes can prevent all the n dimensions from being selected for splitting the n-cube and forming an $(n - 1)$-cube. We first examine the dimension $n - 1$, i.e. the $(n - 1)$-cubes, $0*^{n-1}$ and $1*^{n-1}$. Without loss of generality, we assume that the faulty nodes in $1*^{n-1}$ are $f_1 = 0^n$ and $f_2 = 0^{n-2}11$ which prevent dimensions 0 and 1 from being selected. We also assume dimensions 0 and 1 are not blocked from being selected by other pairs of nodes if $n - 1$ is odd. Now we examine dimensions 0 and 1. Since b_1b_0 of f_1 is 00 and b_1b_0 of f_2 is 11, we must have a faulty node (say f_5) whose b_1b_0 is 10. The bit b_0 of f_5 is 0 because both $*^{n-1}0$ and $*^{n-1}1$ must contain equal number of faulty nodes which is $\lceil \frac{n-1}{2} \rceil + 1$. The bit b_1 of f_5 is 1 because Dimension 1 in $*^{n-1}0$ should be blocked from being selected. The partial address, bits $n-2$ to 2 of f_5, denoted as $y_{n-2}...y_2$, must be the same as that of a faulty node f_3 in $0*^{n-1}$ since f_5 must be less than three hops from one of the faulty nodes in $0*^{n-1}$. Notice that f_3 may or may not be the same as f_1. Similarly, b_1b_0 of a faulty node f_6 in 1^{n-1} is 01 and the partial address $z_{n-2}...z_2$ of f_6 is the same as that of f_4 in $0*^{n-1}$. The bits $b_{n-2}...b_2$ of faulty node f_1 is equal to $x_{n-2}...x_2$ which is 0^{n-3}. Thus, if we shrink the n-cube along dimensions 0, 1, and $n-1$, we will get an $(n-3)$-cube consisting of supernodes that are 3-cubes. If we treat a supernode as faulty if it contains faulty nodes, then the $(n-3)$-cube contains $2(\lceil \frac{n-4}{2} \rceil + 1)$ faulty nodes because faulty nodes f_1 and f_2, f_3, and f_5, or f_4 and f_6 are in the same supernode. By induction, there exists a fault-free $(n-5)$-cube consisting of 3-cubes, i.e., a fault-free $(n-2)$-cube. □

To generalize the above result, we deduce the following theorem.

THEOREM 5. *$k(n, m+1) \geq (\lceil \frac{n-m}{3} \rceil + 1)2^m + 2^{m-1} - 1$, for $n \geq 3 + m$ and $m \geq 1$.*

PROOF. By splitting the n-cube into 2^{m-1} $(n-m+1)$-cubes, we find that there must exist an $(n-m+1)$-cube which contains at most $2(\lceil \frac{n-m}{2} \rceil)$ faulty nodes. According to Lemma 4, there must exist a fault-free $(n-m-1)$-cube. □

The foregoing results provide a lower bound on the number of faulty nodes that can be tolerated while maintaining a fault-free $(n-m-1)$-cube. The following upper bound results are also useful.

LEMMA 5. *$k(n, 2) \leq 2n - 1$ for $n \geq 3$.*

PROOF. Let the fault distribution of $2n$ faulty nodes be such that no fault-free $(n-2)$-cube is available. Let all the neighbors of 0^n and 1^n be faulty. The fault distribution is symmetric according to the dimensions. Thus by splitting the n-cube at any dimension, we obtain two $(n-1)$-cubes, each containing n faulty nodes. Since $n-1$ faulty nodes in an $(n-1)$-cube are the neighbors of a node, all the dimensions are prevented from being selected. Thus, in order to have a fault-free $(n-1)$-cube, the maximum number of faulty nodes must be less than $2n$. □

Lemma 5 leads to a general result as follows:

THEOREM 6. *$K(n, m) \leq 2 \times \sum_{i=1}^{m-1} C_i^n - 1$ for $1 \leq m \leq \lfloor \frac{n}{2} \rfloor$.*

Proof: Let these $2 \times \sum_{i=1}^{m-1} C_i^n$ faulty nodes be 1, 2, ..., and m hops away from 0^n and 1^n. Then any $(n-m)$-cube (Q_{n-m}) contains at least $2(n-m)$ faulty nodes which are adjacent to two antipodal nodes of Q_{n-m}. According to Lemma 5, no fault-free $(n-m-1)$-cube is available. □

Finally, we summarize all the expressions of $k(n, m)$ in Table I based on the results of the proposed technique and the results from [10]. We give the results of exhaustive search algorithms (XR and XI) and the proposed algorithms (FR and FI) for regular and irregular subcubes, respectively. It is of interest to know the exact number of faults, $k(n, m)$ in an n-cube such that a fault-free m-cube is available. Table II shows $k(n, m)$ of various approaches for $1 \leq m \leq n \leq 6$. The values shown in the exhaustive search algorithm for regular subcubes are obtained from [10]. The values shown in the proposed algorithms for regular and irregular subcubes are from the divide-and-conquer approaches given in Sections III and IV, respectively. Since it is hard to get the exact value of $k(n, m)$, we use a computer program to do the exhaustive search for irregular $(n-m)$-cubes (XI).

TABLE I
EXPRESSIONS OF $k(n, m)$ BY EXHAUSTIVE SEARCH FOR REGULAR m-CUBES (XR), EFFICIENT SEARCH FOR REGULAR m-CUBES (FR), EXHAUSTIVE SEARCH FOR IRREGULAR m-CUBES (XI), EFFICIENT SEARCH FOR IRREGULAR m-CUBES (FI)

	XR	FR	XI	FI
$k(n,1)$	2	2	$= \lceil \frac{n}{2} \rceil$	$= \lceil \frac{n}{2} \rceil$
$k(n,2)$	$< \log n + \frac{1}{2} \log\log n$	4	$\geq 2\lceil \frac{n-1}{2} \rceil + 2$	$\geq 2\lceil \frac{n-1}{2} \rceil + 1$
$k(n,m)$	$< \Phi(n,m)$	2^m	$\geq 2^{m-1}(\lceil \frac{n-m+1}{2} \rceil + 1.5) - 1$	$\geq 2^{m-1}(\lceil \frac{n-m+1}{2} \rceil + 1) - 1$
$\Phi(n,m) = 2^{m-2}\lceil 1.6 \log(n-m+3) - 2.1 \log\log(n-m+3) - 4.5 \rceil$				

TABLE II
$k(n, m)$ OF AN n-CUBE FOR $1 \leq m \leq n \leq 6$

m	n															
	6				5				4				3			
	XR	FR	XI	FI	XR	FR	XI	FI	XR	FR	XI	FI	XR	FR	XI	FI
5	1	1	3	3	0	0	0	0	-	-	-	-	-	-	-	-
4	5	3	8	7	1	1	4	4	0	0	0	0	-	-	-	-
3	11	7	15	13	5	3	6	6	1	1	2	2	0	0	0	0
2	21	13	28	27	9	7	11	11	4	3	5	5	1	1	2	2
1	31	31	31	31	15	15	15	15	7	7	7	7	3	3	3	3

V. SUBCUBE ALLOCATION

In this section, we propose a two-phase fault-tolerant subcube allocation strategy that allocates the regular and irregular subcubes to the incoming tasks. Since the faults do not occur as frequently as the processors are allocated in the system, we assume that the faults are detected and located before the allocation starts. The first phase is to construct the set of disjoint subcubes by using the subcube partitioning technique. The second phase is to sort the disjoint subcubes by their sizes in an increasing order and apply the existing fault-free subcube allocation strategy such as the Buddy strategy, etc., to each disjoint subcube. When an incoming task requesting a d-cube arrives in the system, starting from d-cube of the sorted disjoint subcubes, an available d-cube is searched and assigned to the incoming task. The search starts with the list of disjoint subcubes.

A. Subcube Construction Phase

In this section, we present the algorithm Find_Disjoint_Subcubes as shown in Fig. 7 for constructing the set of disjoint regular and irregular subcubes in a faulty hypercube. The *if* statement in Line 1 of Fig. 7 is the condition that terminates the recursion. If the number of faults, $|F|$, is equal to the number of processors in the system then the process stops. Otherwise, the procedure Find_n_1_cube (Fig. 8) is invoked to find an available $(n-1)$-cube. If an $(n-1)$-cube exists, it is put in the Disjoint_Set. Finally, Find_Disjoint_Subcubes is called recursively with the dimension decreased by one. If an $(n-1)$-cube does not exist, Split_*n*_cube in Fig. 7 is called to find a dimension at which the *n*-cube is split into two $(n-1)$-cubes such that the difference between the numbers of faulty processors in the two $(n-1)$-cubes is maximum. If there is a tie, the technique that uses the weight vector to find the set of disjoint regular subcubes given in Section III can be applied to determine the dimension to split. If the sets of faults in the two $(n-1)$-cubes are in F_0 and F_1, then two Find_Disjoint_Subcubes are called recursively for F_0 and F_1, respectively.

```
Procedure Find_Disjoint_Subcubes($n$, $F$, $P$)
/* $n$ is the dimension of the hypercube and $F$ is the set of faults */
/* $P$ is initialized as null */
begin
1     if $|F| = 2^n$ then return;
2     $Q_{n-1}$ := Find_n_1_cube($n$, $F$, $P$);
3     if an $(n-1)$-cube $Q_{n-1}$ exists then
4         Disjoint_Set := Disjoint_Set $\cup$ $Q_{n-1}$;
5         Find_Disjoint_Subcubes($n-1$, $F$, $P$);
6     else
7         $d$ := Split_n_cube($n$, $F$, $F_0$, $F_1$);
8         $P := P \cup d$;
9         Find_Disjoint_Subcubes($n-1$, $F_0$, $P$);
10        Find_Disjoint_Subcubes($n-1$, $F_1$, $P$);
11    endif
end
```

Fig. 7. Finding the set of disjoint irregular subcubes.

```
Procedure Find_n_1_cube($n$, $F$)
/* $n$ is the dimension of the hypercube and $F$ is the set of faults */
begin
1     for $i = 0$ to $n-1$ do
2         if $f_i = a$ for all $f \in F$, $a = 0$ or 1 then
3             $Q_{n-1} := *^{n-i-1}\bar{a}*^i$;
4             $P := P \cup i$;
5             return ($Q_{n-1}$);
6         endif
7     endfor
8     for all $f_i, f_j \in F$ and $i \neq j$ do
9         if Hamming_distance($f_i, f_j$) = 1 or 2 then
10            $Dim\_Set$ := $Dim\_Set \cup$ dimension_spanned($f_i$, $f_j$);
11    if $i \in \{0, .., n-1\}$ and $i \notin P \cup Dim\_Set$ then
12        $Q_{n-1}$ := construct_n_1_cube($n$, $F$, $i$);
13        if $Q_{n-1} \neq null$ then;
14            $P := P \cup i$;
15            return ($Q_{n-1}$);
16        endif
17    else
18        return ($Null$);
19    endif
end
```

Fig. 8. Finding irregular $(n-1)$-cubes.

The procedure Find_*n*_1_cube, given in Fig. 8, first checks if there exists a fault-free regular $(n-1)$-cube (i.e., if there exists a dimension *i* such that the *i*th bit values of the addresses of faulty nodes are all 0s or all 1s). If there exists a fault-free regular $(n-1)$-cube, then this $(n-1)$-cube is returned and the dimension *i* is put in *P*, which is the set of dimensions that have been processed. Otherwise, the irregular $(n-1)$-cube is searched according to the proposed subcube partitioning technique. Hamming distances of addresses of every two faulty nodes are computed. For each pair of faulty nodes f_i and f_j whose Hamming distance is 1 or 2, the procedure dimension_spanned in Line 10 of Fig. 8 computes the dimensions spanning f_i and f_j and puts them into Dim_Set. If there exists a dimension *i* that does not belong to *P* and *Dim_Set*, then *i* is put in *P* and the $(n-1)$-cube is constructed by the procedure construct_*n*_1_cube(*n*, *F*, *i*) as follows. The *n*-cube is split into two $(n-1)$-cubes along dimension *i*. One $(n-1)$-cube is selected and the faults in the selected $(n-1)$-cube are replaced with the healthy processors in the other $(n-1)$-cube. The three criteria stated in Section IV must be satisfied. To update the set of faults, *F*, the *i*th bits of the addresses of faults in the selected $(n-1)$-cube are complemented. The following example illustrates the construction process of procedure Find_*n*_1_cube. The constructed 2-cube and 3-cube are indicated by solid lines and bold dotted lines, respectively, in Fig. 5.

EXAMPLE 4. Consider the 4-cube in Fig. 5 with two faults, 0 and 15. There is no regular 3-cube available because the faults are at antipodal positions. Since the Hamming distance between the two faults is 4, any dimension from 0 through 3 can be selected for constructing the irregular 3-cube by procedure construct_*n*_1_cube.

When procedure Find_Disjoint_Subcubes terminates, Disjoint_Set contains all the fault-free disjoint subcubes. The last step is to sort the Disjoint_Set by the size of the subcubes.

Let us analyze the complexity of procedure Find_*n*_1_cube. The first part that finds the regular $(n-1)$-cube takes $O(n|F|)$ time units because we need to check the *n* bit values for the address of each faulty node. The second part that finds the irregular $(n-1)$-cube takes $O(n|F|^2)$ times units. Notice that the complexity of the second part of the procedure takes into account the time spent in checking the three criteria for selecting the appropriate image nodes. Thus, the complexity of procedure Find_Disjoint_Subcubes can be computed as $O(n^2|F|^2)$.

B. Subcube Allocation Phase

The next step is to allocate fault-free subcubes to the incoming tasks. To accomplish that, we propose the procedure Fault_Tolerant_Subcube_Allocation given in Fig. 9. The procedure selects a subcube (*A*) from the set of disjoint subcubes and uses an existing fault-free subcube allocation strategy to allocate an available subcube from *A*. The deallocation procedure does not change except that the subcubes are released to the disjoint subcube from which they were allocated. The following example illustrates the procedure.

EXAMPLE 5: Based on Fig. 5, Disjoint_Set1 = $\{Q_0, Q_0, Q_2, Q_3\}$ is returned by procedure Find_Disjoint_Subcubes, or Disjoint_Set2 = $\{Q_0, Q_0, Q_0, Q_0, Q_1, Q_3\}$ is returned if the intermediate nodes, such as nodes 0011, 0101, and 0110, cannot be used to form subcubes except being used as 0-cubes. Consider the Buddy strategy as the dedicated allocation strategy, and assume that the incoming task sequence is {1, 1, 0, 3}. For Disjoint_Set1, two 1-cubes in Q_2, a 0-cube in Q_0, and a 3-cube in Q_3 will be granted. However, for Disjoint_Set2, a 1-cube in Q_2, a 1-cube in Q_3, and a 0-cube in Q_0 are granted. In the latter case, the incoming task requesting a 3-cube must wait for the completion of the other tasks.

```
Procedure Fault_Tolerant_Subcube_Allocation(n, d)
/* n and d are the dimensions of the hypercube and requested subcube */
begin
    for all $Q_i \in$ Disjoint_Set, from $i = 1$ to $|Disjoint\_Set|$ do
        if $|Q_i| \geq d$ then
            $Q_d :=$ Fault_Free_Subcube_Allocation($Q_i, d$);
            if $Q_d \neq$ Null then
                return ($Q_d$);
        endif
    endfor
    return (Null);
end
```

Fig. 9. The second phase of fault-tolerant subcube allocation.

The complexity of the two-phase fault-tolerant allocation is analyzed as follows. To simplify the analysis, we assume the number of the faulty nodes is $O(n)$ and the intermediate nodes are not used for forming different subcubes. The total number of nodes in the disjoint subcubes formed by our proposed technique is $O(2^n)$. Thus, the complexity of the allocation and deallocation processes is nearly equal to that of the applied fault-free subcube allocation strategy. For example, the Buddy strategy with time complexity $O(2^n)$ checks all the 2^n nodes in the worst case to find a fault-free subcube. Therefore, the time complexity of the two-phase fault-tolerant subcube allocation applying the Buddy strategy is also $O(2^n)$.

The two-phase subcube allocation strategy can also be applied to fault-free hypercubes as follows. It may be observed in [22] that the left half of the system is more fragmented than the right half in the first-fit allocation approaches. The first phase is to split the n-cube into a set of subcubes, one k-cube for $1 \leq k \leq n - 1$ and two 0-cubes. The second phase is the same as the one in the faulty system described above. Although this approach (unlike first-fit approaches) sacrifices the whole system to be assigned as a subcube, the overall system performance can be improved.

VI. Simulation Results

In this section, we first present the simulation results on finding a fault-free $(n - 1)$-cube in an n-cube containing a certain number of faulty nodes. We carry out the experiments by randomly generating faults in an n-cube. The probabilities of recognizing regular $(n - 1)$-cubes (Section III) are compared with the probabilities of recognizing irregular $(n - 1)$-cubes based on subcube partitioning techniques.

In the simulation, each process is repeated 10000 times with randomly located faults and then the average is computed. Fig. 10 and Fig. 11 show the results for 8-cube and 12-cube, respectively. Fig. 10 shows the comparison of probabilities of recognizing fault-free 7-cubes in an 8-cube with 2 to 22 faults. We can see that when the number of faulty nodes exceeds 5, the probability of successful recognition of regular 7-cubes drops under 50% and to 0% when the number of faulty nodes reaches 10. However, the probability of successful recognition of fault-free irregular 7-cubes drops to 0% when the number of faulty nodes reaches 20. Fig. 11 shows similar results on a 12-cube. Comparing the two figures, it can be seen that the probability of successful recognition of regular $(n - 1)$-cubes does not increase with the system size. But the ability of recognizing irregular $(n - 1)$-cubes increases, because the probability that the distance between any two faulty nodes is greater than two also increases with the system size.

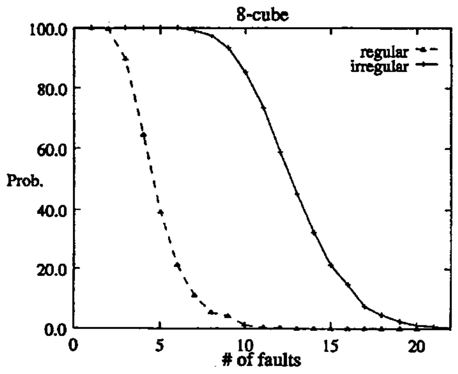


Fig. 10. Probabilities of fault-free 7-cubes in an 8-cube with faults.

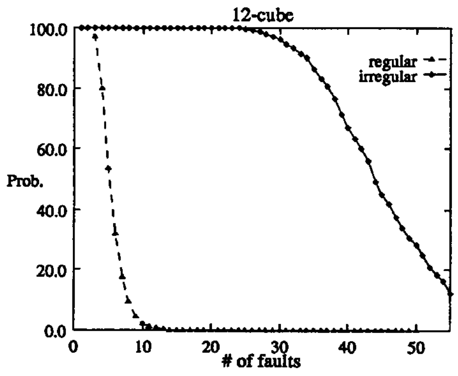


Fig. 11. Probabilities of fault-free 11-cubes in a 12-cube with faults.

Now, we present the simulation results and compare the performance of the subcube allocation strategies based on the Buddy and single Gray code strategies [4]. However, the experiment can be easily extended to other strategies as well [4], [5], [6], [7], [8]. We select the Buddy and single GC strategies because the Buddy strategy is currently employed in the commercial hypercube multiprocessors and the single GC strategy is simple to implement. The following allocation experiments are conducted in the simulation:

1) The straightforward *regular* allocation strategy, which treats the faulty processors as being allocated permanently.
2) The *modified regular* subcube allocation strategy, which is based on the procedure Form_SDS proposed in Section III.

3) The *irregular* subcube allocation strategy, which is based on our two-phase fault-tolerant subcube allocation strategy given in Section V.
4) The *pessimistic* irregular subcube allocation strategy, which is the same as the *irregular* one except that the intermediate nodes cannot be used to form other disjoint subcubes.

In the experiments, K faults are generated 20 times with $K \geq 2$. For each generation of K faults, 100 incoming tasks are generated and queued. The dimensions of the requested size by the incoming tasks are assumed to follow a given distribution such as uniform, normal, and decreasing distributions. In our experiments, the residence time of the allocated subcube is assumed to be uniformly distributed between 5 and 11. Let p_i be the probability that an incoming task requests a subcube of size i, for $0 \leq i \leq D$, where D is the size of the system. Thus we have $\sum p_i = 1$. The p_is used in our experiments are shown in Table III.

TABLE III
p_i OF NORMAL AND DECREASING DISTRIBUTIONS

8-cube normal	$p_0 \longrightarrow p_8 =$	0.0228, 0.044, 0.919, 0.1498, 0.383, 0.1498, 0.919, 0.044, 0.0228
8-cube decreasing	$p_0 \longrightarrow p_8 =$	0.393, 0.239, 0.145, 0.088, 0.053, 0.032, 0.020, 0.018, 0.012
12-cube normal	$p_0 \longrightarrow p_{12} =$	0.0057, 0.0135, 0.0245, 0.0714, 0.12, 0.1639, 0.182, 0.1639, 0.12, 0.0714, 0.0245, 0.0135, 0.0057

The service discipline of the system is assumed to be based on first come, first served (FCFS). At each time unit, the system attempts to find a fault-free subcube of the requested size to the first task in the queue and the assigned task is removed from the queue. After an incoming task in the system finishes, the subcube assigned to it is released. The process continues until all the 100 tasks are finished. Under the above simulation model, the performance is measured in terms of completion time (total time to complete all the 100 tasks), waiting time (average waiting time before a task is assigned to a subcube), and processor utilization (the percentage of a processor being utilized per unit time). For each K faults, 20 independent runs are performed. The average of these parameters for 20 runs is computed and used in the plots.

The simulations without any fault are also conducted for comparison using the Buddy and GC allocation strategies (called *no fault*). It is possible that a task requests for a system size, n-cube or $(n-1)$-cube, etc., that cannot be available due to faults in the system. Since we do not reject those tasks, we will double (or quadruple) the residence time of the incoming task and reduce the subcube size by one dimension (or by two) accordingly. This gives a fair comparison and indicates the impact of faults on the performance of the subcube allocation strategy.

We first show the simulation results based on the Buddy strategy. Figs. 12 and 13 show the completion time and utilization of allocating 100 incoming tasks with an uniform distribution of requested sizes in an 8-cube containing 2 to 12 faults. Notice that the performance of the modified regular scheme is always better than the regular scheme. We can see that the performance of allocating irregular subcubes is always better than allocating regular subcubes. The *pessimistic* irregular approach is slightly worse than the *irregular* approach. This indicates that bigger subcubes play an important role in the allocation process. The *pessimistic* approach does not lose much bigger subcubes in which the biggest possible subcube is an $(n-1)$-cube. The results for no-fault case are the best and are not affected by the number of faults in the system.

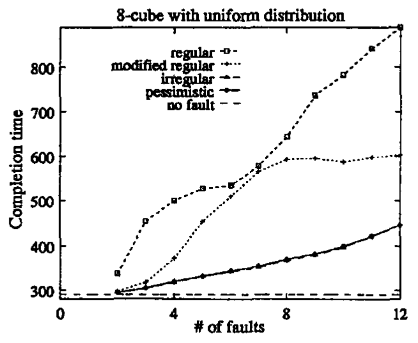


Fig. 12. Completion time of 100 incoming tasks by the Buddy strategy.

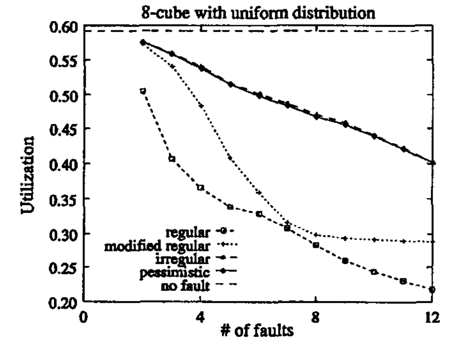


Fig. 13. Utilization of 100 incoming tasks by the Buddy strategy.

Figs. 14 and 15 show the results for the 100 incoming tasks with a normal distribution of requested sizes. The performance of irregular subcubes again is found to be better than others in the faulty systems. Figs. 16 and 17 show the results for the 100 incoming tasks with a decreasing distribution of requested sizes. The performance of irregular subcubes also continues to be better than others in the faulty systems.

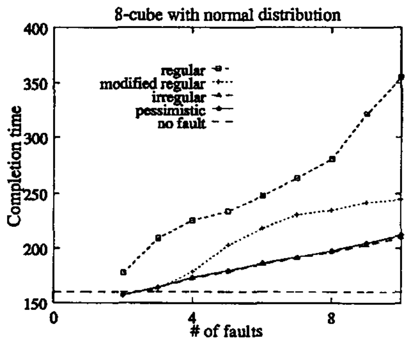


Fig. 14. Completion time of 100 incoming tasks by the Buddy strategy.

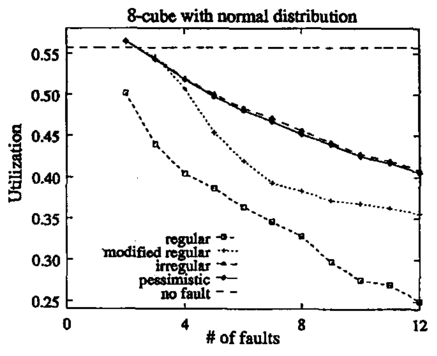


Fig. 15. Utilization of 100 incoming tasks by the Buddy strategy.

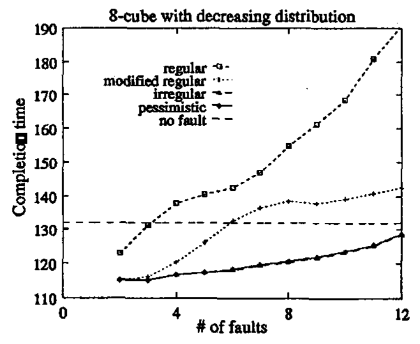


Fig. 16. Completion time of 100 incoming tasks by the Buddy strategy.

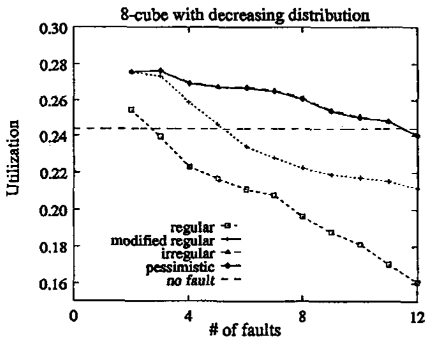


Fig. 17. Utilization time of 100 incoming tasks by the Buddy strategy.

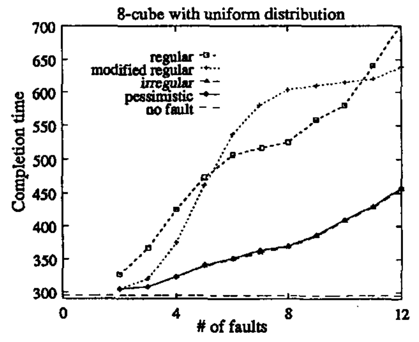


Fig. 18. Completion time of 100 incoming tasks by the single GC strategy.

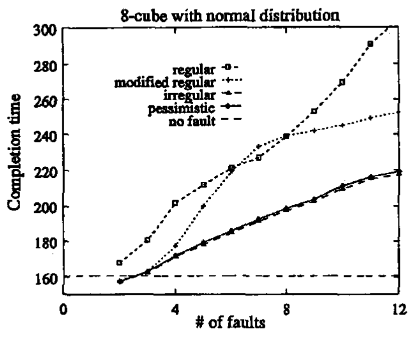


Fig. 19. Completion time of 100 incoming tasks by the single GC strategy.

The simulation results of completion time based on single GC strategy have been shown in Figs. 18 and 19 with uniform and normal distributions for an 8-cube system. Basically, the curves resemble those of the Buddy strategy. The behavior is similar for processor utilization. Notice that in some cases, the performance of the *modified regular* approach is worse than the *regular* approach. We also get similar characteristics in the 12-cube case.

VII. Conclusion

The fault-tolerant capability of a multiprocessor becomes a critical issue when the size of the system grows. In this paper, we have proposed some novel subcube partitioning techniques to allocate subcubes in a faulty hypercube. The concept of irregular subcubes is introduced to enhance subcube availability in the presence of faults. We have shown that $\lceil \frac{n}{2} \rceil$ faults in an n-cube can be tolerated while maintaining a fault-free $(n-1)$-cube. We have also derived the bounds on the number of faults that can be tolerated in order to find fault-free subcubes of smaller sizes. These results are superior to those existing schemes in the literature.

We have considered wormhole routing, currently employed in commercial hypercube multiprocessors, in constructing subcubes in the presence of faults. As long as the links of an irregular subcube that is assigned to one task are not shared by other subcubes, the task execution time will have an insignificant increase when compared to the time spent on running the same task in a regular subcube. Two-phase fault-tolerant subcube allocation strategy is then developed to show the effectiveness of average case on our irregular subcube construction technique. Simulation results show that two-phase subcube allocation is superior to fault-free regular subcube allocation.

Acknowledgments

This research was supported in part by the National Science Foundation under grant MIP-9002353 and by the Texas Advanced Technology Program under grant No. 999903-025.

REFERENCES

[1] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Computers*, pp. 323-333, Apr. 1984.

[2] nCUBE Coporation, *nCUBE 2 Processor Manual*, Dec. 1990.

[3] Intel, *Intel iPSC/2*, Intel Scientific Computers, 1988.

[4] M.S. Chen and K.G. Shin, "Processor allocation in an N-cube multiprocessor using gray codes," *IEEE Trans. Computers*, pp. 1,396-1,407, Dec. 1987.

[5] S. Dutt and J.P. Hayes, "Subcube allocation in hypercube computers," *IEEE Trans. Computers*, pp. 341-351, Mar. 1991.

[6] J. Kim, C.R. Das, and W. Lin, "A top-down processor allocation scheme for hypercube computers," *IEEE Trans. Parallel and Distributed Systems*, pp. 20-30, Jan. 1991.

[7] P.J. Chuang and N.F. Tzeng, "A fast recognition-complete processor allocation strategy for hypercube computers," *IEEE Trans. Computers*, pp. 467-479, Apr. 1992.

[8] Y. Chang and L.N. Bhuyan, "Fault tolerant subcube allocation in hypercubes," *Proc. Int'l Conf. Parallel Processing*, pp. I-132-136, Aug. 1993.

[9] J. Hastad, T. Leighton, and M. Newman, "Fast computation using faulty hypercubes," *Proc. 21st Ann. ACM Symp. Theory and Computing*, pp. 251-263, 1989.

[10] M. Livingston, Q. Stout, N. Graham, and F. Harary, "Subcube fault tolerance in hypercube," Technical report CRL-TR-12-87, Univ. of Michigan, Computing Research Lab., Sept. 1987.

[11] B. Becker and H. Simon, "How robust is the n-cube?" *Information and Computation*, pp. 162-178, 1988.

[12] J. Bruck, R. Cypher, and D. Soroker, "Tolerating faults in hypercubes using subcube partitioning," *IEEE Trans. Computers*, pp. 599-605, May 1992.

[13] C. Raghavendra, P. Yang, and S. Tien, "Free dimensions—An effective approach to achieve fault tolerance in hypercubes," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 170-177, 1992.

[14] O.H. Kang, S.Y. Yoon, H.S. Yoon, and J.W. Cho, "Heuristic subcube allocation in hypercube systems," *IEICE Trans. Information and Systems*, pp. 517-526, July 1992.

[15] Q. Yang and H. Wang, "A new graph approach to minimize processor fragmentation in hypercube multiprocessors," *IEEE Trans. Parallel and Distributed Systems*, pp. 1,165-1,171, Oct. 1993.

[16] Y. Chang and L.N. Bhuyan, "Parallel algorithms for hypercube allocation," *Proc. Int'l Parallel Processing Symp. (IPPS)*, pp. 105-112, Apr. 1993.

[17] S.H. Bokhari, "Communication overheads on the Intel iPSC-2 hypercube," Intel ICASE Interim Report 10, May 1990.

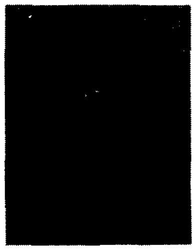
[18] M. Peercy and P. Banerjee, "Distributed algorithms for shortest-path, deadlock-free routing and broadcasting in arbitrarily faulty hypercubes," *Proc. Int'l Symp. Fault-Tolerant Computing*, pp. 218-225, 1990.

[19] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, "Hyperswitch network for the hypercube computer," *Proc. Int'l Symp. Computer Architecture*, pp. 90-99, 1988.

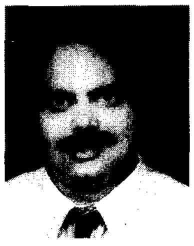
[20] S. Abraham and K. Padmanabhan, "Performance of the direct binary *n*-cube network for multiprocessors," *IEEE Trans. Computers*, pp. 1,000-1,011, July 1989.

[21] L.M. Ni and P.K. McKinley, "A survey of wormhole routing techniques in direct networks," *Computer*, vol. 26, no. 2, pp. 62-76, Feb. 1993.

[22] D. Jokanovic, N. Shiratori, and S. Noguchi, "Fault tolerant processor allocation in hypercube multiprocessors," *IEICE Trans. Information and Systems*, vol. E 74, no. 10, pp. 3,492-3,505, Oct. 1991.



Yeimkuan Chang received the BS degree in physics from the National Central University, Taiwan, Republic of China, and the MS degree in computer science from the University of Houston at Clear Lake in 1984 and 1990, respectively. He received the PhD degree in computer science from the Texas A&M University in 1995. From 1986 to 1987, he was with Lighton Inc., Taiwan, as a process engineer. Currently, he is an associate professor in the Department of Information Management at the Chung-Hua Polytechnic Institute, Taiwan, Republic of China. His research interests include cache coherence design for multiprocessors, fault tolerance, and interconnection networks.



Laxmi N. Bhuyan (S'81-M'82-SM'87) received the MSc degree in electrical engineering from Sambalpur University, India, in 1979, and the PhD degree in computer engineering from Wayne State University, Detroit, Michigan, in 1982.

Dr. Bhuyan is currently a professor in computer science at Texas A&M University, College Station, Texas. He was previously with the Center for Advanced Computer Studies at University of Southwestern Louisiana. His research interests are in the areas of computer architecture, parallel processing, fault-tolerant computing, and performance evaluation.

Dr. Bhuyan currently serves as the area editor of *Computer* magazine for systems architecture and as an editor of *Parallel Computing Journal*. He was an ACM Lecturer and a Distinguished Visitor of the IEEE Computer Society. He was the program committee chairman of the First International Symposium on High-Performance Computer Architecture (HPCA), January 1995. He is the chair of the IEEE Computer Society Technical Committee of Computer Architecture (TCCA) for 1995-96.