Embedding Cube-Connected Cycles Graphs into Faulty Hypercubes

Jehoshua Bruck, *Senior Member, IEEE*, Robert Cypher, *Member, IEEE*, and Danny Soroker

***Abstract*— We consider the problem of embedding a cube-connected cycles graph (CCC) into a hypercube with edge faults. Our main result is an algorithm that, given a list of faulty edges, computes an embedding of the CCC that spans all of the nodes and avoids all of the faulty edges. The algorithm has optimal running time and tolerates the maximum number of faults (in a worst-case setting). Because ascend–descend algorithms can be implemented efficiently on a CCC, this embedding enables the implementation of ascend-descend algorithms, such as bitonic sort, on hypercubes with edge faults. We also present a number of related results, including an algorithm for embedding a CCC into a hypercube with edge and node faults and an algorithm for embedding a spanning torus into a hypercube with edge faults.**

***Index Terms*— Cube-connected cycles, fault tolerance, graph embedding, gray code, hypercube, mesh, parallel computing, torus.**

I. Introduction

THE n-dimensional hypercube (n-cube) is a popular interconnection topology for parallel computers. It has been studied extensively in the literature and it is used in several machines that are built and sold commercially, such as the Intel iPSC-860, the nCUBE-2 and the Connection Machine CM-2. An important issue related to such parallel machines is how they can compute in the presence of faults. In this paper we study this issue for the class of ascend-descend algorithms.

Ascend-descend algorithms are an important class of algorithms that have efficient parallel implementations on the hypercube and several related topologies. They were first described by Preparata and Vuillemin [14], who showed that many widely used parallel algorithms, including bitonic sort, FFT and several matrix operations, are either ascend-descend algorithms or are composed entirely of subroutines that are ascend-descend algorithms. In an ascend-descend algorithm communication is synchronous and occurs in phases. In any one phase all of the processors communicate across a single dimension of the hypercube, while successive phases use either successively higher numbered dimensions (i.e., the dimensions are used in ascending order) or successively lower numbered dimensions (i.e., the dimensions are used in descending order). Because ascend-descend algorithms use all of the hypercube edges (communication links) and nodes (processors), even a single faulty edge or node can seriously affect their efficiency. The goal of this paper is to obtain efficient, simple implementations of ascend-descend algorithms on hypercubes with faulty components.

The issue of computing with faulty hypercubes and related computers has been addressed in a number of recent papers [1]–[5], [7]–[10], [12], [15]. Some of the previous work has considered randomly located faults [2, 10, 15], while the current paper assumes a worst-case distribution of faults. Furthermore, much of the previous work has considered the effects of node faults [1]–[3], [7], [8], [10], [12]. In particular, Aiello and Leighton have shown that any hypercube algorithm can be run on an n-cube with $n^{O(1)}$ node and/or edge faults with only a constant factor slowdown [1]. However, the slowdown factor is rather large (although constant), thus making that approach unsuitable in practice.

In contrast, the main result presented here addresses the effects of edge faults. By focusing on edge faults, we are able to obtain implementations of ascend-descend algorithms which use all of the hypercube nodes. More specifically, we show that any ascend-descend algorithm can be implemented on an n-cube which has $n-3$ or fewer faulty edges with only a small constant factor slowdown in communication and *no* slowdown in computation relative to its implementation on a fault-free n-cube. This is the first result known which obtains this performance given such a large number of edge faults. In addition, we show how a similar approach can be used to tolerate node faults.

Our approach is to find a fault-free subgraph of the n-cube on which ascend-descend algorithms can be implemented efficiently (i.e., with only a constant factor slowdown in communication relative to their implementation on a fault-free hypercube). It is well-known that the shuffle-exchange and the cube-connected cycles (CCC), both of which are constant degree graphs, can implement ascend-descend algorithms efficiently. However, for any $n \geq 3$ the shuffle-exchange with 2^n nodes contains a cycle of length 5. As a result, the shuffle-exchange is not a subgraph of the n-cube, which is bipartite. On the other hand, every n-cube does contain a CCC as a spanning subgraph. In fact, it will be shown that every n-cube with $n-3$ or fewer edge faults contains a fault-free CCC which is a spanning subgraph of the n-cube. Because each node in the CCC has degree 3, and because a worst-case distribution

Manuscript received October 5, 1992; revised August 27, 1993.

J. Bruck is with the California Institute of Technology, Mail Code 116–81, Pasadena, CA 91125 USA; e-mail: bruck@systems.caltech.edu.

R. Cypher is with the Department of Computer Science, The Johns Hopkins University, 3400 N. Charles Street, Baltimore, MD 21218 USA.

D. Soroker is with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598 USA; e-mail: soroker@watson.ibm.com.

IEEE Log Number 9402907.

of the faults is assumed, $n-3$ is also an upper bound on the number of faults that can be tolerated[1]. Furthermore, an optimal $O(n)$ time sequential algorithm will be presented which locates the fault-free CCC given the locations of the edge faults. In addition, we will give algorithms for embedding a CCC into a hypercube with node and edge faults and for embedding a spanning mesh into a hypercube with edge faults.

The remainder of the paper is organized as follows. Section II presents some definitions and notation that will be used throughout the paper. In Section III, we give an overview of our approach for embedding a CCC into a hypercube while avoiding faulty edges. Sections IV–VII contain the details of the embedding algorithm. Extensions are presented in Section VIII and conclusions are given in Section IX.

II. Definitions and Notation

Given an integer x, where $0 \leq x < 2^n$, the n-bit binary representation of x will be denoted $(x_{(n-1)} \cdots x_{(0)})$ and the ith least significant bit of x, where $0 \leq i \leq n-1$, will be denoted $x_{(i)}$. Given an integer $x_{(i)} \in \{0,1\}$, the *complement of* $x_{(i)}$, denoted $\overline{x_{(i)}}$, equals $1 - x_{(i)}$. Given n-bit binary numbers x and y, the bitwise exclusive-or of x and y will be denoted $x \oplus y$. An *n-bit Gray code* is a sequence of 2^n unique integers $x_0, x_1 \cdots x_{2^n-1}$ in the range 0 through 2^n-1 such that for any i and j in the range 0 through 2^n-1 where $j = (i+1) \bmod 2^n$, the binary representations of x_i and x_j differ in exactly one bit position.

The n-dimensional hypercube (n-cube) consists of 2^n nodes, each of which has a unique label in the range 0 through $2^n - 1$. Any pair of hypercube nodes are adjacent if and only if their binary representations differ in exactly one bit position. The edge e which connects hypercube nodes $(e_{(n-1)} \cdots e_{(i+1)} e_{(i)} e_{(i-1)} \cdots e_{(0)})$ and $(e_{(n-1)} \cdots e_{(i+1)} \overline{e_{(i)}} e_{(i-1)} \cdots e_{(0)})$ will be denoted $(e_{(n-1)} \cdots e_{(i+1)} * e_{(i-1)} \cdots e_{(0)})$ and the jth least significant bit of e, where $j \neq i$ and $0 \leq j \leq n-1$, will be denoted $e_{(j)}$. Such an edge e will be referred to as being in *dimension* i, and the terms "dimension" and "bit" will be used interchangeably throughout. The four-dimensional hypercube is shown in Fig. 1, in which edges in the same dimension appear parallel. It will be assumed that a faulty hypercube edge cannot be used for communication, while a faulty hypercube node can be used neither for computation nor for communication. It will also be assumed that the faults are static and that their locations are known.

Let $n = 2^k + k$ for some integer k. The *n-dimensional cube-connected cycles (CCC)* [14] consists of 2^n nodes, each of which has a unique label of the form (l, c) where $0 \leq l < 2^{2^k}$ and $0 \leq c < 2^k$. Each node (l, c) is connected to nodes $(l, (c+1) \bmod 2^k)$ and $(l, (c-1) \bmod 2^k)$ via forward and backward *cycle* connections and to node (m, c), where the binary representations of m and l differ in only bit position c, via a *lateral* connection. Thus a CCC consists of 2^{2^k} cycles, each of which has length 2^k. For example, the six-dimensional CCC is obtained by replacing each node in Fig. 1 by a cycle of length 4. The nodes within each cycle are numbered sequentially (modulo 2^k), and lateral edges connect nodes in different cycles. Note that each node (l, c) can be assigned a unique n-bit number where the k least significant bits give the value of c, the node's position within its cycle, and the remaining bits give the value of l, the number of the cycle to which it belongs.

[1] For some values of n, the CCC contains nodes with degree 2, in which case $n-2$ faults can be tolerated. This case is discussed in section 7.

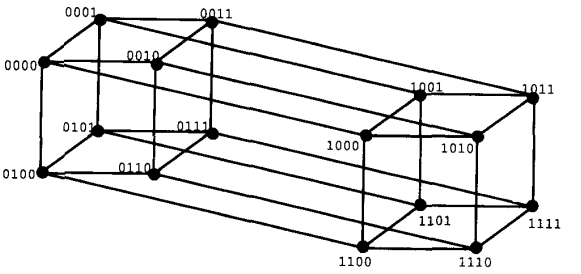


Fig. 1. The four-dimensional hypercube. Edges in the same dimension are drawn parallel.

III. The Embedding Strategy

This section gives an overview of how a fault-free n-dimensional CCC can be embedded in an n-cube with $n-3$ or fewer edge faults. We will assume throughout the discussion that $n = 2^k + k$ for some integer k. An extension of our results to arbitrary values of n will be given in Section VII.

In order to simplify the presentation, we will modify the previous definition of a CCC in three ways. First, rather than numbering the nodes within each cycle sequentially, we will number them with a Gray code. Second, the correspondence between a node's position within a cycle and the type of lateral connection which it has (that is, which bit position the lateral connection complements) will be loosened. Third, the bits which specify a node's position within a cycle can be interleaved with the bits which specify the number of the cycle to which it belongs. More formally, each node is labeled with a unique binary string of length $n = 2^k + k$. The bits in the string are of two kinds:

Lateral bits: $l_0, \cdots, l_{2^k-1}$;
Cycle bits: $c_0, \cdots, c_{k-1}$.

Each node has three neighbors, each of which differs from it in a single bit. We will call the three dimensions (bits) which connect a node, x, to its neighbors the *active dimensions (bits) of* x. A node has two *cycle neighbors* and one *lateral neighbor*. The cycle neighbors of a node each differ from it in one cycle bit, and its lateral neighbor differs from it in a single lateral bit, subject to the following requirements.

1) All of the nodes which have the same lateral bit values form a cycle (of length 2^k).
2) Along any one such cycle, each lateral bit is active exactly once.
3) Any two nodes with the same cycle bit values have the same active (cycle *and* lateral) bits.

It is straightforward to verify that the above definition is equivalent to the original definition.

Recall that in an n-cube each node is labeled with a unique binary string of length n and two nodes are adjacent if they differ in exactly one bit. It is well-known that a CCC with 2^n nodes is a subgraph of the n-cube, as we will demonstrate with the following simple embedding. First, select k dimensions in the n-cube to correspond to the cycle bits and let the other 2^k dimensions correspond to the lateral bits (recall that $n = 2^k + k$). This partitions the n-cube into disjoint k-cubes, each of which contains 2^k nodes that differ in only the cycle bits. Second, select a k-bit Gray code and use this Gray code to embed an identical cycle of length 2^k in each of the k-cubes. The edges in these cycles will be the cycle edges of the CCC. Third, select a 1–1 correspondence between the set of 2^k values of the cycle bits and the set of 2^k lateral bits. For each value of the cycle bits, connect the n-cube nodes with the given value of the cycle bits along the corresponding lateral dimension. These edges will be the lateral edges of the CCC.

We will use the same three-stage embedding in order to find a fault-free CCC in an n-cube with faulty edges. Thus our embedding will perform the following operations.

1) Partition the n-cube dimensions between cycle dimensions and lateral dimensions.
2) Construct a spanning cycle within the cycle dimensions.
3) Construct a 1–1 correspondence between the lateral bits and the set of values of the cycle bits.

In the following sections we will describe how to perform each of the above tasks. Our main result will be obtained by identifying a sufficient number of degrees of freedom in each of these stages in order to construct a fault-free embedding. We describe Step 1 last because we will need the conclusions from Steps 2 and 3 to guide us in partitioning the dimensions.

IV. Constructing a Spanning Cycle in a Faulty Cube

In this section, we describe how to construct a spanning cycle (i.e., Hamiltonian cycle) in a k-dimensional hypercube that contains some faulty edges. We assume that the k cycle bits have been selected in such a way that there are at most $k-2$ faulty edges within the cycle dimensions. This assumption will be justified in Section VI. Chan and Lee have shown that any k-cube with $k-2$ faulty edges is Hamiltonian [6], but their existence proof does not provide an efficient algorithm for finding such a cycle. Our result here is an $O(k)$ time sequential algorithm for the following problem.

Input: An integer k and a set F of faulty edges in a k-cube such that $|F| \leq k-2$.

Output: A description of a spanning cycle of the k-cube that avoids all of the edges in F.

The reason we consider all of the faults to lie within a single k-cube is that for the CCC embedding we must construct a *single* cycle within the cycle bits. In other words, given a partition of the n-cube into k-cubes, we are interested only in the location of a fault within a k-cube, not in which k-cube it is located.

g_3	g_2	g_1	g_0
0	0	0	0
0	0	0	1
0	0	1	1
0	0	1	0
0	1	1	0
0	1	1	1
0	1	0	1
0	1	0	0
1	1	0	0
1	1	0	1
1	1	1	1
1	1	1	0
1	0	1	0
1	0	1	1
1	0	0	1
1	0	0	0

Fig. 2. The 4-bit binary reflected Gray code G_4.

The main idea of the algorithm is as follows. Consider some spanning cycle, C, of the k-cube. By applying an adjacency-preserving permutation, π, to the nodes of the k-cube, C maps to another spanning cycle, C_π. Our goal will be to find a permutation for which the derived cycle avoids all of the faulty edges. The key is to start with a highly structured initial cycle and exploit its structure in order to construct π efficiently.

In particular, the spanning cycle C which we use is given by the *binary reflected Gray code*. Let $G_k = g_{k,0}, g_{k,1}, \cdots, g_{k,2^k-1}$ denote the binary reflected Gray code with k bits, and given an $X \in \{0,1\}$ let $Xg_{k,j}$ denote the $(k+1)$-bit binary number obtained by concatenating X and $g_{k,j}$. The binary reflected Gray code is defined recursively as follows:

- $G_1 = 0, 1$
- If $k \geq 1$, $G_{k+1} = 0g_{k,0}, 0g_{k,1}, \cdots, 0g_{k,2^k-1}, 1g_{k,2^k-1}, 1g_{k,2^k-2}, \cdots, 1g_{k,0}$.

The binary reflected Gray code for the case $k = 4$ is shown in Fig. 2, and graphically in Fig. 3(a).

The permutations we consider are of two kinds. The first is the set of permutations of the bit positions. The second is the set of "translations" given by the bitwise exclusive-or with a constant. A translation is specified by a k-bit integer b, which we will call the *bias*. Given a bias b, the translation maps each node x to node $x \oplus b$. The cycle G_4 and its translation by bias 0001 are shown in Fig. 3.

Our analysis proceeds as follows. First, we select a permutation of the bit positions in G_k in order to produce a modified Gray code which is less likely to contain faulty edges than the original Gray code, G_k. Then, for any given edge, e, we characterize the set of biases that map some edge of the modified Gray code cycle onto e. We call a bias in this set *bad for e*, since if e is faulty, this bias cannot be used. Carrying this methodology further, we construct a *good* bias, i.e., one that is not bad for any of the faulty edges. By translating the modified Gray code cycle by this good bias, we thus obtain a fault-free cycle.

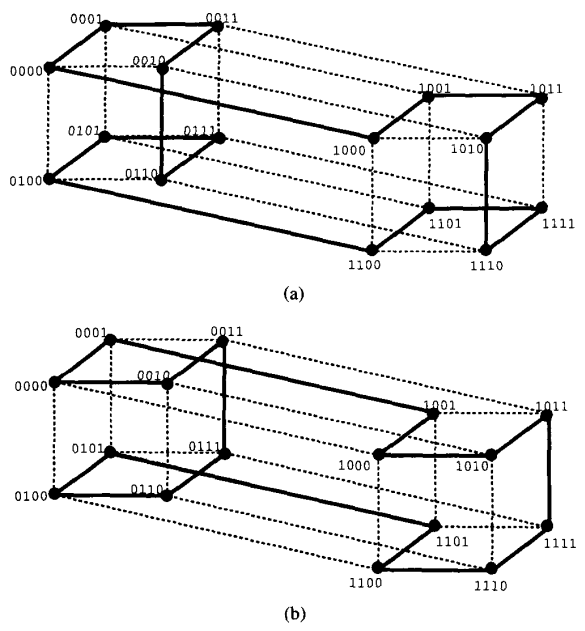


Fig. 3. (a) The four-dimensional Gray code cycle, G_4. (b) The cycle obtained by translating G_4 by bias 0001.

Our permutation of the bit positions will be based on the following properties of binary reflected Gray codes.

Lemma 1: The cycle G_k, defined by the k-bit binary reflected Gray code, contains all of the edges in dimension 0 and edges $(*0\cdots0)$ and $(*10\cdots0)$ in dimension $k-1$. Furthermore, an edge e in dimension i, $0 < i < k-1$, is in G_k, if and only if $e_{(i-1)} = 1$ and $e_{(j)} = 0$ for all $0 \le j < i-1$.

Proof: The fact that the above conditions are sufficient for an edge to be in the cycle G_k follows immediately from the construction. The fact that the above conditions are also necessary can be shown by a simple counting argument. Specifically, there are 2^{k-1} edges in dimension 0, 2 edges in dimension $k-1$ which meet the stated conditions and 2^{k-i-1} edges in dimension i, where $0 \le i \le k-2$, which meet the stated conditions. Therefore, a total of $2+\sum_{i=0}^{k-2} 2^{k-i-1} = 2+\sum_{i=1}^{k-1} 2^i = 2^k$ edges meet the stated conditions. Because G_k contains 2^k edges and all of those edges meet the conditions, all edges which meet the conditions must be in G_k. □

Corollary 2: The cycle G_k, defined by the k-bit binary reflected Gray code, contains 2^{k-1-i} edges in dimension i, for all i in the range 0 through $k-2$, and two edges in dimension $k-1$.

Another way of stating the above corollary is that G_k contains all of the dimension-0 edges, half the dimension-1 edges, a quarter of the dimension-2 edges and so on. Only dimension $k-1$ behaves differently in that it is active twice, rather than once. Therefore, our first step is to permute the dimensions so as to achieve the following property.

Property A (Cycle Dimensions): The number of faults in dimension i is no more than the number of faults in dimension j for all $i < j$.

In the remaining description we will assume that the dimensions have been permuted so that Property A holds. We will now characterize the cases in which a bias is bad for a given edge.

Lemma 3: Let e be an edge in dimension i of the k-cube. A bias, b, is bad for e according to the following necessary and sufficient conditions.

$i = 0$: unconditionally

$1 \le i \le k-2$: when $b_{(i-1)} = \overline{e_{(i-1)}}$ and $b_{(j)} = e_{(j)}$ for all $j < i-1$.

$i = k-1$: when $b_{(j)} = e_{(j)}$ for all $j < k-2$.

Proof: Recall that a bias is bad for e if it maps an edge of G_k to e. Since the mapping is through the exclusive-or function, which has the property that if $x \oplus y = z$ then $z \oplus y = x$, it follows that b is bad for e if and only if $e \oplus b$ is an edge of G_k. Thus the claim follows from Lemma 1. □

Now we are ready to present an efficient and simple algorithm for constructing a good bias. The idea is to use a diagonalization technique on the set F. For each element, $e \in F$, we choose one bit of the bias in a way that ensures that the bias is good for e. In particular, for an edge e in dimension i, if, for some $j < i-1$, we set bit j of the bias to be $\overline{e_{(j)}}$, then the bias will be good for e, as implied by Lemma 3. It turns out that, due to Property A, this can be done simultaneously for all of the elements of F. This is formalized in the following lemma.

Lemma 4: Let $F = \{f^0, \cdots, f^{k-3}\}$ be a set of faulty edges for which Property A holds, such that the dimension of f^i is no more than that of f^j if $i < j$. Let b^F be a bias defined by

$$b^F_{(k-1)} = b^F_{(k-2)} = 0 \text{ and } b^F_{(i)} = \overline{f^i_{(i)}} \text{ for all } 0 \le i \le k-3.$$

Then b^F is good for F.

Proof: By Property A, edge f^{k-3} is in dimension $k-1$. For each i where $0 \le i < k-3$, the dimension of edge f^i is either equal to the dimension of edge f^{i+1} or it is one less. By induction, for all i where $0 \le i \le k-3$, the dimension of f^i is at least $i+2$. Let f^i be an arbitrary edge in F. From the construction, $b^F_{(i)} = \overline{f^i_{(i)}}$. From Lemma 3 and the fact that the dimension of f^i is at least $i+2$, it follows that b^F is good for f^i. It follows that b^F is good for all of the edges in F. □

Thus. we obtain the following simple linear time algorithm for constructing a fault-free spanning cycle.

Spanning Cycle Algorithm:

1) Sort the dimensions according to the number of faults (using bucket sort) so that the number of faults increases with the dimension.
2) Sort the faults by dimension (again using bucket sort) so that lower numbered faults belong to lower dimensions.
3) Construct a good bias by complementing the appropriate bits of the faulty edges according to Lemma 4.

◇

In Section VIII-B, we show how this method can be extended to handle a larger number of faults, provided that they are distributed in a certain manner.

V. Matching Lateral Bits to Cycle Positions

In the CCC, any two nodes with the same cycle bit values have the same active lateral bit (see Section 3). Thus there is a 1-1 correspondence between the set of lateral bits and the set of cycle positions. In this section we show how to construct such a correspondence that avoids all of the faulty lateral edges. We cast the problem at hand in terms of *bipartite matching* and discuss it in terms of an adjacency matrix, M, that has a row for each cycle position, a column for each lateral bit, and an "×" in each entry that contains a faulty edge. The task is to select one location in each row and column so that none of the selected locations has an ×. We will assume that the dimensions of the n-cube have been partitioned between cycle dimensions and lateral dimensions in such a way as to guarantee a certain property (Property B given below). In the next section we will describe how to efficiently construct a partition that has this property.

Property B (Lateral Dimensions): No row or column of the matrix is completely filled with ×'s.

Recall that $n = 2^k + k$ and that the total number of faulty edges in the n-cube is at most $n - 3$. Let $m = 2^k$. Thus the matrix M in the matching problem is of size $m \times m$, and the number of ×'s in M is at most $m + k - 3$, which is less than $(1.1)m$ for all values of k. It will therefore be sufficient to obtain a linear time algorithm for the following problem.

Input: An integer m and a list of at most $(1.1)m$ faulty locations in an $m \times m$ matrix, M, such that no row or column is completely faulty.

Output: A perfect matching of rows to columns that avoids all of the faulty locations in M.

It is important to note that the input is given by a sparse representation, so the running time of the algorithm should be proportional to m, and not to the size of M, which is m^2.

Intuitively, if we start by matching rows and columns that have many ×'s, then the chance of being unable to extend the matching at a later stage will be small. As a result, we will start by considering a simple greedy algorithm for solving the problem. This algorithm is given below.

Greedy Matching Algorithm:

1) Sort the rows and columns by the number of ×'s, and let $r_0, \cdots, r_{m-1}$ and $c_0, \cdots, c_{m-1}$ be the lists of rows and columns, respectively, in order of decreasing numbers of ×'s.
2) For i equals 0 through $m - 1$, find the least j such that c_j is unmatched and $M[i, j]$ is not marked, and match r_i to c_j.

◇

Lemma 5: The greedy algorithm given above constructs a fault-free matching of all but at most one row/column pair. Furthermore, if there is a row with no faults (×'s), then the algorithm constructs a fault-free matching of all of the rows and columns.

Proof: Assume that step 2 of the algorithm gets stuck when trying to match row i. This means that when reaching stage i, all of the remaining entries in row i are marked. (By *remaining entries* we mean those in columns that are still unmatched at stage i.) Thus row i of M has at least $m - i$ ×'s. Now, since the rows are sorted in decreasing order in terms of the number of ×'s, it follows that r_j has at least $m - i$ ×'s, for all $j < i$. Therefore the number of ×'s in M is at least $(i + 1) \cdot (m - i)$. But since the total number of ×'s in M is at most $(1.1)m$, i can be only 0 or $m - 1$. The case $i = 0$ is disallowed, since it implies that r_0 is totally marked. Therefore, the only place where the algorithm can get stuck is in matching the last row to the remaining column. Furthermore, if the last row has no ×'s, then the algorithm will not get stuck. □

Modified Matching Algorithm:

1) If $m = 1$, match row r_0 to column c_0 and return. Otherwise, perform the following operations.
2) Run step 1 of the greedy matching algorithm.
3) Run step 2 of the greedy matching algorithm for i equals 0 through $m - 3$.
4) Match the last two rows (r_{m-2} and r_{m-1}) to the remaining two columns.

◇

Lemma 6: The modified algorithm given above always constructs a fault-free matching of all of the rows and columns.

Proof: Assume for the sake of contradiction that the modified algorithm fails to find a fault-free matching. First, note that the modified algorithms succeeds whenever $m \leq 2$, so it follows that $m \geq 3$. The proof of Lemma 5 implies that the greedy matching algorithm will successfully match the first $m - 2$ rows. Thus the modified algorithm must fail when attempting to match the last two rows. The fact that the rows are sorted implies that each of the last two rows, r_{m-2} and r_{m-1}, contains at most one marked entry. Therefore, one of the two columns that remains when attempting to match the last two rows must be marked in both of the last two rows. Let c_i denote this column. Note that $i \neq 0$, because c_0 is matched in the first row in which it is not marked, and no column is marked in every row. The fact that c_i remains when attempting to match the last two rows implies that c_i contains at least $m - i$ marked entries. Because the columns are sorted in decreasing order in terms of the number of marked entries, column c_j has at least $m - i$ marked entries for all $j \leq i$. Thus there are at least $(i+1)(m-i)$ marked entries. Because $(i + 1)(m - i) \leq (1.1)m$ and $m \geq 3$, it follows that either $i = 0$ or $i = m - 1$. It was shown above that $i \neq 0$, so $i = m - 1$. However, column $i = m - 1$ contains at least two marked entries, so every column must contain at least two marked entries. As a result, there are at least $2m > (1.1)m$ marked entries, which is a contradiction. □

The modified algorithm can be implemented in linear time as follows. Step 1 takes constant time. Step 2 can be performed using bucket sort. For Step 3, have a linked list of "remaining columns" and for each row have a list of the columns in which it has ×'s. Have all these column lists sorted (starting with lower index). Now, at stage i, scan the list of remaining columns and the list of ×'s in row i. Find the first column that does not have an × at row i, and remove it from the list of remaining columns. The running time of this stage is proportional to m plus the total number of ×'s, which is $O(m)$. Step 4 takes constant time.

The faults:

Faults $b_5b_4b_3b_2b_1b_0$	Projection on b_1 b_0	Projection on b_2 b_1
0 * 1 0 0 0	0 0	0 0
0 1 * 0 0 0	0 0	0 0
0 0 * 0 0 1	0 1	0 0
1 1 * 0 1 1	1 1	0 1
0 1 * 1 1 0	1 0	1 1

The matching matrices:

b_1 b_0	5	4	3	2
0 0		×	×	
0 1			×	
1 0			×	
1 1			×	

b_2 b_1	5	4	3	0
0 0		×	×	
0 1			×	
1 0				
1 1			×	

Fig. 4. Two partitions which result in different matching matrices.

VI. Partitioning the Dimensions

So far, we have described how to efficiently construct a fault-free embedding of the cycles and the lateral edges of a CCC in a hypercube. The missing ingredient, which will be presented in this section, is how to partition the dimensions of the n-cube between cycle dimensions and lateral dimensions. By the discussion so far, the following two requirements are sufficient to guarantee that a partition, P, of the dimensions will yield a solution to the embedding problem.

1) Under P there are at most $k-2$ faulty cycle edges.
2) Under P, Property B (of Section V) holds.

Say we partition the dimensions so that the k dimensions with the least number of faults are cycle dimensions and the rest are lateral dimensions. Since the total number of faults is at most $n-3$, it follows that there will be at most $k-3$ faulty cycle edges. (The worst case is when the n-cube has 3 fault-free dimensions and $n-3$ single-fault dimensions.) However, Property B (of Section V) is not guaranteed to hold. If the problem is that a whole row is marked with ×'s, then the solution is simple: in this case, many dimensions (more than 2^{k-1}) have a single fault. Therefore we can assign 2 fault-free dimensions and $k-2$ single-fault dimensions as cycle dimensions. Under this assignment there are 2^k-1 lateral faults, so no row (or column) can be completely marked.

When a whole column is marked, the problem is more complicated. In this case, there is a single "bad" dimension with many faults (at least 2^k). Note that, consequently, the total number of faults in all of the remaining 2^k+k-1 dimensions is at most $k-3$. Thus we may choose any k of the remaining dimensions as cycle dimensions. Furthermore, there must be at least 2^k+2 fault-free dimensions, from which we will choose the cycle dimensions. We observe that the choice of which k fault-free dimensions are picked influences the structure of the matching matrix, M. In particular, it can determine whether M has a completely filled column. An example is shown in Fig. 4.

Now we can state the problem to solve when there is a highly faulty column.

Partition Problem: Select a set, G, of k fault-free dimensions as cycle dimensions so that in the resulting matching matrix, M, the bad column (corresponding to the bad dimension) will contain at least one unmarked entry.

We first prove that such a partition always exists. Our proof will provide an efficient algorithm for finding the set G.

Lemma 7: Let $k>1$, and let T_{k+2} be the table of all binary strings of length $k+2$. (T_{k+2} has $k+2$ columns and 2^{k+2} rows). Let S be a subset of the rows of T_{k+2} with the following property—for any subset, C_k, of k columns of T_{k+2}, the table whose columns are C_k and whose rows are S contains all the 2^k binary words of length k. Then the size of S is strictly greater than 2^k.

Proof: Assume that $|S|=2^k$. Then we claim that any two words in S differ in at least 3 bits (the Hamming distance between them is at least 3). If not then two words $A,B\in S$ agree on some subset, C, of k bits. But then the table whose columns are C and whose rows are S has less than 2^k different words, contradicting the assumption. Now, since the Hamming distance between any two words of S is at least 3, the distance-1 neighborhoods of any two words in S are disjoint. Each such neighborhood is of size $k+3$, since it contains a word and its $k+2$ neighbors. Consequently, in the set of words of length $k+2$ there must be 2^k disjoint sets, each of size $k+3$, so:

$$(k+3)\cdot 2^k \le 2^{k+2}.$$

But the above inequality holds only for $k\le 1$, which is a contradiction. □

Corollary 8: Let S be a set of binary words of length 4 such that for every subset of 2 bit positions, each of the 4 possible settings appears in some word of S. Then S contains at least 5 words.

Theorem 9: Let T_{k+2} and S be as in the statement of Lemma 7. Then the size of S is at least $\frac{5}{4}2^k$.

Proof: Let $c_{k+1}\cdots c_0$ denote the $k+2$ columns in T_{k+2}. Partition the 2^{k+2} strings in T_{k+2} into blocks of size 16, where in each block the values of $c_{k+1}\cdots c_4$ are fixed and $c_3\cdots c_0$ take on all the 16 possible values. Note that S must contain at least 5 words from each such block, because otherwise, from Corollary 8, there is some setting of bits in columns $c_{k+1}\cdots c_4$ and of two bits from columns $c_3\cdots c_0$ that is not covered by S. Since these blocks are disjoint, and there are 2^{k-2} of them, it follows that S contains at least $5\cdot 2^{k-2}=\frac{5}{4}2^k$ words. □

Corollary 10: There exists a set, G, of k fault-free dimensions such that when they are chosen as cycle dimensions, in the resulting matching matrix, M, the bad column will contain at least one unmarked entry.

Proof: Restricting our attention to any set D of $k+2$ fault-free dimensions, we can map each fault to a binary word of length $k+2$ by considering the position of the fault in each of the dimensions in D (of course multiple faults may map to a single binary word of length $k+2$). From Theorem 9, it follows that if there are less than $1.25\cdot 2^k$ faults, then D contains some set, G, of k dimensions such that the set of faults does not map to all of the possible strings on G. Thus if we pick G to be the cycle dimensions, the resulting matching matrix will contain an unmarked entry in the bad column. Because the total number of faults is less than $1.1\cdot 2^k$, this is always possible. □

Now we are ready to describe a linear time algorithm for the following problem.

Input: An integer n, $n = k + 2^k$, and a set, F, of $n - 3$ faulty edges in the n-cube of which at least 2^k are in one dimension.

Output: A set G of k fault-free dimensions whose choice as cycle dimensions yields a matching matrix in which no column is completely filled with $\times$'s.

Dimension Partition Algorithm:

1) Select an arbitrary set D of $k+2$ fault-free dimensions. Call the selected dimensions $c_{k+1} \cdots c_0$.
2) In an array A of length 2^{k+2}, mark the locations in which faults occur when they are projected onto the dimensions in D.
3) Partition the array A into 2^{k-2} blocks, each containing 16 consecutive entries. Find a block, B, which contains fewer than 5 faults.
4) Find a 2-dimensional subset of $c_3 \cdots c_0$ for which not all of the values are covered by the faults in B. Assign these two dimensions together with $c_4 \cdots c_{k+1}$ to be the set G of cycle dimensions.

VII. Putting it All Together

In this section we describe the complete algorithm for embedding a CCC in an n-cube with at most $n - 3$ edge faults. There is still one gap that needs to be filled—dealing with n-cubes for which n is not of the form $2^k + k$.

Let k be the smallest integer for which $k + 2^k \geq n$. The CCC graph is defined as in Section III, with k cycle bits and $n - k$ lateral bits. Thus, for general n, the cycle length is at least the number of lateral bits, but never more than twice that number. Therefore at least half of the cycle positions have lateral connections, but not necessarily all. We call the positions which have lateral connections *active*. Note that if a node is not active, its degree is 2, so if not all the nodes are active (i.e., n is not of the form $2^k + k$), $n - 2$ faults should be tolerated. Our algorithm achieves this bound.

We would like, for simplicity, to have the active cycle positions be an arbitrary fixed set (say positions $0 \cdots n-k-1$). However, if $n - 2$ faults are to be tolerated, then some extra care is needed. Specifically, even if $k - 2$ faults occur in cycle edges, the resulting matching matrix, M, could still contain a completely marked row. The solution is simple–if some cycle position in $0 \cdots n - k - 1$ is completely faulty, then replace it (in the set of active positions) by position $n - k$.

We are now ready to present the embedding algorithm in its full generality.

Fault-Free Embedding Algorithm:

1) *Dimension Partition:*
   - Sort the n dimensions according to the number of faults.
   - If at least $k - 2$ dimensions have a single fault, then assign 2 fault-free dimensions and $k - 2$ single-fault dimensions as cycle dimensions.
   - If some dimension has at least $n - k$ faults, then assign the cycle dimensions according to the dimension partition algorithm of Section VI.
   - If neither of the above two cases holds, assign the dimensions with the least number of faults as cycle dimensions.
2) *Cycle Construction:* Calculate the projections of the cycle faults onto the cycle dimensions. Construct a fault-free spanning cycle in the k-cube using these values as inputs to the spanning cycle algorithm of Section IV.
3) *Matching Lateral Bits to Cycle Positions:*
   - Let M be the matching matrix (described in Section V).
   - Select the set, R, of active cycle positions (active rows of M) as follows. If some row, r, in the set $\{0, \cdots, n - k - 1\}$ is completely marked, then $R = \{0, \cdots, n - k\} \backslash \{r\}$. Otherwise $R = \{0, \cdots, n - k - 1\}$.
   - Let M' be the square submatrix of M whose rows are the set R and whose columns are the columns of M. Match the lateral bits to the active cycle positions using the modified matching algorithm of Section V.

VIII. Extensions

The previous sections presented an algorithm for embedding a CCC into a hypercube with edge faults. In this section we will show how the techniques developed above can be extended to solve some related problems.

A. Node Faults

We will first consider the embedding of a CCC into a hypercube with faulty edges *and* nodes. We assume that a faulty node can neither perform computations nor route data. Since a faulty edge can be avoided by treating one of the nodes to which it is connected as being faulty, we will consider only node faults in the remaining discussion. We will concentrate on embedding a CCC using *half* of the n-cube nodes.

Our basic approach is similar to that used with edge faults. Specifically, we use a three step process.

1) Partition the hypercube dimensions between cycle and lateral dimensions.
2) Construct a cycle within the cycle dimensions.
3) Assign the lateral connections to the dimensions of the hypercube.

Since node faults do not show up in steps 1 and 3, they can both be performed arbitrarily in constant time. Thus the problem reduces to that of constructing a cycle within the cycle dimensions. In a manner similar to Section IV, the problem is characterized as follows.

Input: A set F of faulty nodes in a k-cube.

Output: A cycle of length 2^{k-1} in the k-cube that avoids all the nodes of F.

The algorithm for embedding a cycle of length 2^{k-1} in a k-cube is given next.

1) Partition the k-cube into 2-cubes (arbitrarily).
2) Embed a CCC in the resulting $(k-2)$-cube, disregarding faults.
3) Now consider a 2-cube to be faulty if any of its nodes is faulty, and find the largest connected component of the CCC embedded in Step 2.
4) (The next two steps are illustrated in Fig. 5.) Find a spanning tree of the connected component created in the previous step, and remove nodes from this spanning tree until it contains exactly 2^{k-3} nodes (it is assumed that the connected component has at least this many nodes).
5) Use the spanning tree as the basis for creating the cycle with 2^{k-1} nodes. First, replace each node of the spanning tree with the four nodes in the corresponding 2-cube, which are connected in the form of a square. Next, color the top edge of each square with the color T, the right edge of each square with R, and the bottom edge of each square with B. Then color the edges of the spanning tree with T, R and B so that no adjacent edges have the same color. Note that three colors suffice because the tree is a subgraph of a CCC and therefore has maximum degree at most 3. Now for each edge of the spanning tree, add two parallel edges between the squares that the spanning tree edge connected. These parallel edges should connect the endpoints of the square edges having the same color as the spanning tree edge to which they correspond. Finally, erase the square edges that have colors that match a spanning tree edge which is incident to the square. The resulting edges will form a cycle that corresponds to an "Euler tour" of the spanning tree (in the sense used by Tarjan and Vishkin [16]).

The success of this algorithm depends on finding a connected component in Step 3 that contains at least 2^{k-3} nodes. Because a CCC with 2^{k-2} nodes can tolerate up to $O(2^k/k)$ faults and still guarantee that there will be a connected component with 2^{k-3} nodes [14], [17], this algorithm can tolerate up to $O(2^k/k) = O(n/\log n)$ faults. This gives us the following theorem.

Theorem 11: If $n = k + 2^{k-1}$ for some integer k, then a fault-free CCC with 2^{n-1} nodes can be embedded in an n-cube with $O(n/\log n)$ faulty nodes.

B. Spanning Cycle Under Many Edge Faults

The problem of determining the existence of a spanning cycle in a hypercube with faulty edges has been shown to be NP complete [6]. Here we describe how to extend the technique of translating the Gray code cycle G_k that was described in Section IV, to handle many edge faults when they are distributed in a certain manner.

In particular, we are concerned with the distribution of faults across the dimensions. On the one hand, if there is at least one fault in every dimension then this technique must fail, since any translation of G_k contains all the edges in some dimension. On the other hand, it has the potential for tolerating exponentially many faults in a single dimension.

Let F be a set of faulty edges in a k-cube. As before, assume for the rest of this subsection that the dimensions of the k-cube have been renamed so that Property A holds, i.e., that the number of faults increases with the dimension. Let n_i denote the number of faults in dimension i. Then we have the following theorem.

Theorem 12: If the faults in the set F obey the following constraint

$$\sum_{i=0}^{k-2} n_i 2^{-i} + n_{k-1} 2^{2-k} < 1$$

then there is some bias b for which the cycle $G_k \oplus b$ is fault free.

Proof: By Lemma 3, the fraction of biases which are bad for an edge in dimension i is 2^{-i} for $0 \le i \le k-2$, and 2^{2-k} for $i = k-1$. Therefore the fraction of bad biases for the set F is given by the left hand side of the constraint above. □

Note that Theorem 12 describes a condition that is sufficient, but not necessary. If the theorem holds for a set F, then a good bias b can be constructed by the following greedy algorithm.

Greedy Spanning Cycle Algorithm:

1) Sort the dimensions according to the number of faults, so that the number of faults increases with the dimension.
2) Assign weights to the faults by giving weight 2^{-i} to each fault in dimension i, $0 \le i \le k-2$, and weight 2^{2-k} to each fault in dimension $k-1$.
3) Create the set A and initialize it to contain all of the faults in F. The set A will contain those faults which have a possibility of lying on the cycle $G_k \oplus b$.
4) For $i = 0$ to $k-2$ do the following.
   a) Create the set A_0 which consists of those faults in A that have a possibility of lying on the cycle $G_k \oplus b$ if bit i of b equals 0.
   b) Create the set A_1 which consists of those faults in A that have a possibility of lying on the cycle $G_k \oplus b$ if bit i of b equals 1.
   c) Calculate the total weight of the faults in A_0 and of the faults in A_1. If the faults in A_0 weigh less than the faults in A_1, set bit i of b to 0 and let $A = A_0$. Otherwise, set bit i of b to 1 and let $A = A_1$.
5) Set bit $k-1$ of b to 0 (actually, this assignment is arbitrary).

◇

Theorem 13: If the set F satisfies the condition of Theorem 12 then the above algorithm constructs a bias b for which the cycle $G_k \oplus b$ is fault free.

Proof: We will use induction on i to show that for $0 \le i \le k-2$, immediately before setting bit i of b the total weight of the faults in A is less than 2^{-i}. The basis is $i = 0$, in which case the claim follows from the fact that the total weight of the faults in F is less than 1. The induction hypothesis is that the claim holds for $i \le j$ for some j, $0 \le j \le k-3$.

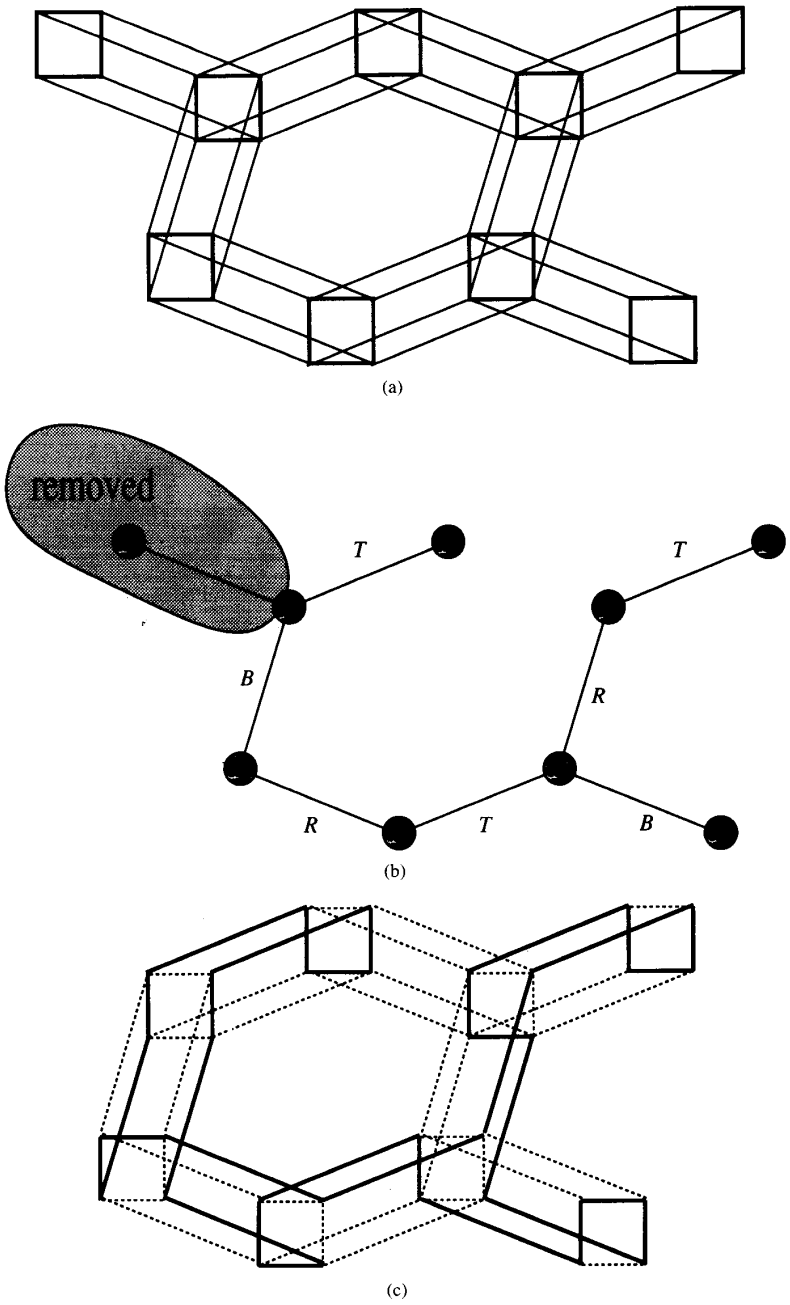


Fig. 5. An illustration of the algorithm for node faults. (a) A maximum fault free connected component of a CCC of 2-cubes. (b) A colored spanning tree of the connected component after excessive nodes have been removed. (c) The spanning cycle corresponding to the tree in (b).

From the induction hypothesis, immediately before setting bit j of b, the total weight of the faults in A is less than 2^{-j}. As a result, all of the faults in A appear in dimensions $j+1$ through $k-1$ (because even a single fault in dimension j or lower has weight at least 2^{-j}). Therefore, from Lemma 3, each fault is in either A_0 or A_1, but not both. Thus after setting bit j the total weight of the faults in A is less than 2^{-j-1}, which completes the inductive proof.

From the above inductive proof, immediately before setting bit $k-2$ of b the total weight of the faults in A is less than 2^{-k+2}, which implies that A is empty. Therefore, no fault lies on the cycle $G_k \oplus b$. □

It is easily verified that the running time of the above algorithm is $O(k|F|)$. Note that it may produce a good bias even if the total weight of F is more than 1, but this is not guaranteed.

C. Mesh and Torus Embedding

Theorem 13 can be used to obtain embeddings of meshes and tori into hypercubes with faulty edges. We will consider only the problem of embedding a $2^m \times 2^m \times \cdots \times 2^m$ d-dimensional torus into an n-cube, where $n = dm$, but it is possible to generalize the results to meshes and tori with sides of different lengths. Our approach consists of partitioning the n-cube dimensions into d sets of size m and embedding a fault-free spanning cycle within each of these d sets [11]. The number of edge faults that can be tolerated is equal to the number of dimensions in the hypercube minus the degree of the torus, and is therefore optimal.

Theorem 14: There is an $O(n)$ time sequential algorithm that embeds a $2^m \times 2^m \times \cdots \times 2^m$ d-dimensional torus into an n-cube, where $n = dm$, which has at most $n - 2d$ edge faults, provided that $n/\log(8n) \geq d$.

Proof: First sort the dimensions so that the number of faults per dimension increases with the dimension, and let a_i denote the number of faults in dimension i, $0 \leq i \leq n-1$. Next, partition the dimensions into d sets, each of which consists of every dth dimension from the sorted list (that is, partition the dimensions according to equivalence classes modulo d). Within each of the d sets, assign weights to the m dimensions in the set by giving weight 2^{-j} to each fault in the jth smallest dimension, $0 \leq j \leq m-2$, and weight 2^{2-m} to each fault in the largest dimension. Let S_0 denote the set which consists of those dimensions i such that $i \equiv 0 (\mathrm{mod}\, d)$. Note that because the dimensions were sorted according to the number of faults, S_0 must have at least as large a weight as any other set. We will show that the weight of S_0 (and thus of any set) is less than 1, so by Theorem 13 a cycle of length 2^m can be embedded into the m-cube given by the dimensions in S_0 while avoiding the edge faults.

Let w denote the weight of the faults in S_0 and let $b_j = a_{jd}$ for $0 \leq j \leq m-1$. Then

$$w = b_{m-1}/2^{m-2} + \sum_{j=0}^{m-2} b_j/2^j.$$

Note that for each fault which contributes to one of the b_j's, where $0 \leq j \leq m-2$, there are at least $d-1$ other faults which do not contribute to any of the b_j's. Therefore,

$$\sum_{j=0}^{m-2} b_j \leq \lfloor ((n-2d) - b_{m-1})/d \rfloor \leq m-3$$

which implies that

$$\sum_{j=0}^{m-2} b_j/2^j < 1/2.$$

Because $n/\log(8n) \geq d$, it follows that $n \leq 2^{(n/d-3)}$, so $1/2 \geq (n-2d)/2^{(n/d-2)} \geq b_{m-1}/2^{m-2}$ and $w < 1/2+1/2 = 1$. □

IX. Conclusion

This paper has examined the problem of embedding a cube-connected cycles graph into a hypercube with faulty edges. We have shown an algorithm that, given a list of faulty edges, computes an embedding of the CCC that spans all of the nodes and avoids all of the faulty edges. The algorithm is optimal both in terms of its running time and in terms of the number of faults that it tolerates (assuming a worst-case fault distribution). Because ascend-descend algorithms can be implemented efficiently on a CCC, this embedding enables the implementation of ascend-descend algorithms, such as bitonic sort, on hypercubes with edge faults.

We also presented a number of related results, including an algorithm for embedding a CCC into a hypercube with edge and node faults and an algorithm for embedding a spanning torus into a hypercube with edge faults.

Acknowledgment

We would like to thank the anonymous referees for their helpful comments and suggestions.

References

[1] B. Aiello and T. Leighton, "Coding theory, hypercube embeddings, and fault tolerance," in *Proc. 3rd Annu. Symp. on Parallel Algorithms and Architectures*, 1991, pp. 125–136.

[2] F. Annexstein, "Fault tolerance of hypercube-derivative networks," in *Proc. 1st Annu. ACM Symp. on Parallel Algorithms and Architectures*, 1989, pp. 179–188.

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

[4] J. Bruck, "On optimal broadcasting in faulty hypercubes," IBM Res. Rep., RJ7147, 1989. To appear in *Disc. Appl. Math.*, vol. 53, Sept. 1994.

[5] J. Bruck, R. Cypher, and D. Soroker, "Running algorithms efficiently on faulty hypercubes," in *Proc. 2nd Annu. ACM Symp. on Parallel Algorithms and Architectures*, 1990, pp. 37–44.

[6] M. Y. Chan and S. J. Lee, "On the existence of hamiltonian circuits in faulty hypercubes," *SIAM J. Discrete Math.*, vol. 4, no. 4, pp. 511–527, 1991.

[7] ——, "Fault-tolerant embedding of complete binary trees in hypercubes," *IEEE Trans. Parallel Distrib. Syst.*, vol. 4, no. 3, pp. 277–288, 1993.

[8] ——, "Fault-tolerant permutation routing in hypercubes," Tech. Rep., UTDCS-5-90, Univ. of Texas at Dallas, 1990.

[9] D. Dolev, J. Y. Halpern, B. Simons, and R. Strong, "A new look at fault-tolerant network routing," *Inform. Computat.*, vol. 72, no. 3, pp. 180–196, Mar. 1987.

[10] J. Hastad, T. Leighton, and M. Newman, "Fast computation using faulty hypercubes," in *Proc. 21st Annu. ACM Symp. on Theory of Computing*, 1989, pp. 251–284.

[11] C. T. Ho and S. L. Johnsson, "Embedding meshes in boolean cubes by graph decomposition," *J. Parallel and Distrib. Computing*, Apr. 1990, pp. 325–339.

[12] M. Livingston, Q. Stout, N. Graham, and F. Harary, "Subcube fault-tolerance in hypercubes," Tech. Rep. CRL-TR-12–87, Univ. of Michigan Computing Res. Lab., Sept. 1987.

[13] M. Livingston and Q. Stout, "Embeddings in hypercubes," In *Math.l Comput. Modeling*, vol. 11, pp. 222–227, 1988.

[14] F. P. Preparata and J. Vuillemin, "The cube-connected cycles: a versatile network for parallel computation," *Commun. ACM*, vol. 24, no. 5, pp. 300–309, May 1981.

[15] M. O. Rabin, "Efficient dispersal of information for security, load balancing, and fault tolerance," *J. ACM*, vol. 36, no. 2, pp. 335–348, Apr. 1989.

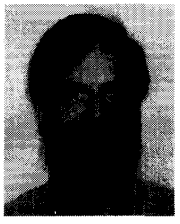
[16] R. E. Tarjan and U. Vishkin, "An efficient parallel biconnectivity algorithm," *SIAM J. Comput.*, vol. 14, 4, pp. 862–874, Nov. 1985.

[17] C. D. Thompson, "Area-time complexity for VLSI," *Proc. 11th Annu. ACM Symp. on Theory of Computing*, 1979, pp. 81–88.

Jehoshua Bruck (S'86–M'89–SM'93) received the B.Sc. and M.Sc. degrees in electrical engineering from the Technion, Israel Institute of Technology, in 1982 and 1985, respectively, and the Ph.D. degree in electrical engineering from Stanford University in 1989.

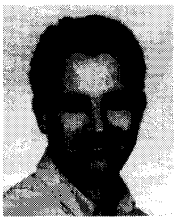
He is an Associate Professor of Computation and Neural Systems and Electrical Engineering at the California Institute of Technology. His research interests include parallel and distributed computing, fault-tolerant computing, error-correcting codes, computation theory, and neural systems. He has an extensive industrial experience, including, serving as manager of the Foundations of Massively Parallel Computing Group at the IBM Almaden Research Center from 1990 to 1994, a Research Staff Member at the IBM Almaden Research Center from 1989 to 1990 and a Researcher at the IBM Haifa Science Center from 1982 to 1985.

Dr. Bruck is the recipient of a 1994 National Science Foundation Young Investigator Award, a 1992 IBM Outstanding Innovation Award for his work on "Harmonic Analysis of Neural Networks" and a 1994 IBM Outstanding Technical Achievement Award for his contributions to the design and implementation of the SP-1, the first IBM scalable parallel computer. He also received five IBM Plateau Invention Achievement Awards and he holds 15 patents.



Robert Cypher (M'93) received the B.S. degree in mathematical sciences from Stanford University in 1982 and the M.S. and Ph.D. degrees in computer science from the University of Washington in 1987 and 1989, respectively.

He was a Research Staff Member in the IBM Almaden Research Center and a Consulting Assistant Professor in the Stanford University Computer Science Department from 1989 through 1993. He was a Research Staff Member in the IBM T. J. Watson Research Center from 1993 through 1994. Currently, he is an Assistant Professor in the Johns Hopkins University Computer Science Department. His research interests include parallel algorithms and architectures, communication protocols, the semantics of parallel programs, VLSI design, fault-tolerance, and the design of interconnection networks.



Danny Soroker was born in Jerusalem, Israel, in 1959. He received the B.Sc. degree in computer engineering and the M.Sc. degree in electrical engineering from the Technion— Israel Institute of Technology, Haifa, in 1981 and 1983, respectively, and the Ph.D. degree in computer science from the University of California, Berkeley, in 1987.

From 1988 to 1990, he was a Visiting Scientist at IBM Almaden Research Center. From 1990 to 1993, he was a Research Computer Scientist at Shell Bellaire Research Center. In 1993, he joined the Entertainment Systems Department at IBM T. J. Watson Research Center as a Research Staff Member, where he has been involved in creating a novel multiprocessor-based system for video post production. His research interests include parallel computing, algorithms, and design and implementation of software systems.

Dr. Soroker is a member of the ACM and the IEEE Computer Society.