The Balanced Hypercube: A Cube-Based System for Fault-Tolerant Applications

Jie Wu, *Senior Member*, *IEEE*, and Ke Huang

Abstract—In this paper, we present an interconnection structure, called the *balanced hypercube*, which is a variation of the standard hypercube with desirable properties of strong connectivity, regularity, and symmetry. The proposed structure is a special type of *load balanced graph* designed to tolerate processor failure. In balanced hypercubes, each processor has a backup (matching) processor that shares the same set of neighboring nodes. Therefore, tasks that run on a faulty processor can be reactivated in the backup processor to provide efficient system reconfiguration. Other properties of balanced hypercubes are examined. It is also shown that odd-dimensional balanced hypercubes have smaller diameters than that of standard hypercubes. As an application of balanced hypercubes, we show a fault-tolerant embedding of rings in balanced hypercubes.

Index Terms—Embedding, fault tolerance, hypercubes, interconnection networks, reconfiguration.

1 Introduction

HYPERCUBES, as one of the popular structures for multicomputers, have received much attention over the past decade. The hypercube structure offers a rich interconnection with a large bandwidth and a short (logarithmic) diameter. A number of hypercube machines have been implemented [4], since the advent of Cosmic Cubes at Caltech. To improve some desirable properties, variations of the hypercube structure have been proposed in the literature ([1], [3], [11]). Most of the hypercube variations focus on reducing diameter and/or message traffic.

As the number of processors in a system increases, the probability of system failure can be expected to be quite high unless specific measures are taken to tolerate faults within the system. Therefore, a major goal in the design of such a system is fault tolerance. These systems are made fault-tolerant by providing redundant or spare processors and/or links [8]. When an active processor (where tasks are running) fails, its tasks are dynamically transferred to spare components. The objective is to provide an efficient reconfiguration by keeping the recovery time small. In a multicomputer system, there are two types of communication in a reconfiguration process:

1) fault status exchange, which is used for each node to broadcast or to obtain some global fault information; and
2) task migration, in which tasks are actually moved around, and which involves more data movements.

It is highly desirable that the reconfiguration process is a decentralized and local one, so fault status exchange and task migration can be reduced or eliminated.

A configuration process is used to recover from faults while still maintaining the quality of the initial embedding. The communication cost of a reconfiguration process can be high due to the migration of a large number of tasks. To obtain a good initial mapping while still maintaining an efficient reconfiguration after the occurrence of faults, three integrated components need to be considered:

1) the topology of the target machine,
2) the initial embedding, and
3) the reconfiguration process.

Unfortunately, most of current existence approaches cannot handle this issue effectively. Most approaches concentrate on only one or two of the above components. One approach to achieve fault tolerance is to introduce spare nodes and/or links to hypercubes. Although this approach can reconfigure a system efficiently and avoid global task migration, the main problem is that all the spares are dedicated ones and they are normally not treated as part of the system. (If spares are treated as part of the system, then many desirable topological properties of the original system are lost.) A similar approach [2] also uses additional nodes and reconfiguration is performed through a series of *automorphic* operations that transform the task graph into a different embedding such that faulty nodes can be avoided. This approach is theoretically sound but is too expensive to be practical, because each operation corresponds to a task migration.

Another approach [13] exploits the inherent redundant processors and/or links in hypercubes to achieve fault tolerance; that is, no extra processors and/or links are added to the structure of the networks (thus all the desirable features are still maintained), but instead save some spare processors intentionally in the initial embedding such that, when faults occur, the faulty processors can be replaced by spare ones. However, hypercubes cannot effectively support an efficient reconfiguration process, especially in multiple-fault cases.

Our proposed work is in line with the later approach. Our objective is to find an embedding approach that is easy to recover from faults. The concept of *consistently recoverable embedding* is proposed. This approach provides a uniform and fast recovery independent of the number and location of faults. Moreover, all of the reconfigurations are local, i.e., tasks are migrated from one processor to one of its neighboring processors. This embedding provides a convenient vehicle for implementing a distributed reconfiguration and minimizing reconfiguration steps.

The *balanced hypercube* [6], proposed by the authors, is a variant of the hypercube structure that can support consistently recoverable embedding. Its desirable properties are similar to those in the standard hypercube. In fact, balanced hypercubes belong to a special type of *load balanced graphs* [10] that can support consistently recoverable embedding. In a load balanced graph $G = (V, E)$, with V as the node set and E as the edge set, for each node v there exists another node v', such that v and v' have the same adjacent nodes (see Fig. 1). Such a pair of nodes v and v' is called a *matching pair*. In a load balanced graph, a task can be scheduled to both v and v' in such a way that one copy is active and the other one is passive. If node v fails, we can simply shift tasks of v to v' by activating copies of these tasks in v'. All the other tasks running on other nodes do not need to be reassigned to keep the adjacency property, i.e., two tasks that are adjacent are still adjacent after a system reconfiguration. Note that the rule of v and v' as primary and backup are relative. We can have an active task running on node v with its backup on node v', while having another active task running on node v' with its backup on node v. With a sufficient number of tasks and a suitable load balancing approach, we can have a balanced use of processors in the system.

Most popular interconnection structures, including the hypercube and its extensions, are not load balanced (i.e., they cannot support consistently recoverable embedding). There is a simple (but naive) approach to extend a regular hypercube to a load balanced one: We consider two $(n-1)$-dimensional hypercubes, A and B. Within each hypercube, number the nodes from 0 to $2^n - 1$. Now add edges between the two hypercubes so that a node v in A is

• *The authors are with the Department of Computer Science and Engineering, Florida Atlantic University, Boca Raton, FL 33431.*
E-mail: jie@cse.fau.edu.

Manuscript received April 7, 1995; revised October 16, 1995.
For information on obtaining reprints of this article, please send e-mail to: transcom@computer.org, and reference IEEECS Log Number C96093.

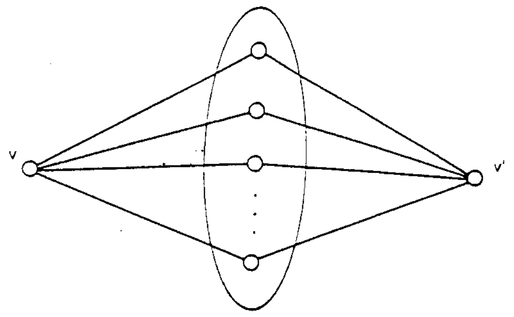


Fig. 1. Nodes *v* and *v′* that share the same set of adjacent nodes.

connected to a node *v′* in *B* if the same numbered node *v* in *B* is connected to node *v′*. Clearly, the resultant structure is load balanced and it contains 2^n nodes. However, the node degree of this *n*-dimensional hypercube is $2(n-1)$ and it is not partitionable, i.e., it cannot be partitioned into two subcubes with each being a load-balanced graph. To make the resultant structure partitionable, the above strategy can be used at each level of the recursive definition. However, the node degree doubles as we increase the dimension by one, i.e., the increasing rates of node degree and number of nodes are the same. A similar strategy is used in defining the balanced hypercube, but fewer links are used. Moreover, each balanced hypercube can be partitioned into four sub-balanced hypercubes. We also show that the balanced hypercube has desirable properties very similar to those in a standard hypercube, e.g., they have the same node degree and connectivity for the same number of nodes. In addition, we prove that odd-dimensional balanced hypercubes have smaller diameters than that of standard hypercubes.

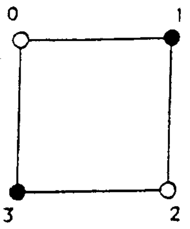
This paper is organized as follows. Section 2 defines the proposed balanced hypercube. Several properties of balanced hypercubes are listed in Section 3. The embedding of rings in balanced hypercubes is also discussed in this section. We show in Section 4 a fault-tolerant embedding of rings in balanced hypercubes that uses consistently recoverable embedding as an application of balanced hypercubes, and the paper concludes in Section 5.

2 Balanced Hypercubes (BHs)

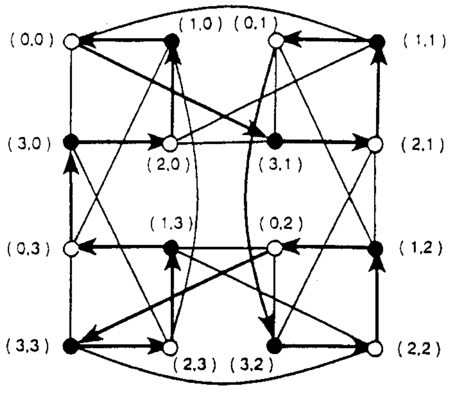
Let $G = (V, E)$ be a finite, undirected graph, with the node set *V* and the edge set *E*. Normally, a node in *V* represents a processor and an edge in *E* corresponds to a communication link connecting two processors. If an edge $e = (v, u) \in E$, then the nodes *v* and *u* are said to be *adjacent*. The *degree*, $d(v)$, of node *v* is equal to the number of edges in *G* which are incident on *v*. The *distance*, $d(v, u)$, between nodes *v* and *u* is equal to the length (in number of edges) of a shortest path connecting *v* and *u*. The *diameter* of *G* is the maximum distance between two nodes in *G* over all pairs of nodes.

A graph *G* is *load balanced* [10] if and only if for every node in *G* there exists another node matching it, i.e., these two nodes have the same adjacent nodes. Hence they are called a *matching pair.* A completely connected graph with an even number of nodes is load balanced, while several other commonly used graph structures, such as meshes, trees, and hypercubes, are not. In the following, we construct a special type of load-balanced graph called *balanced hypercube* (BH_n), where the integer *n* is called the dimension of the balanced hypercube.

Definition 1. *An n-dimensional balanced hypercube BH_n consists of 2^{2n} nodes $(a_0, a_1, \cdots, a_{i-1}, a_i, a_{i+1}, ..., a_{n-1})$, where a_o and $a_i \in \{0, 1, 2, 3\}$ $(1 \le i \le n-1)$. Every node $(a_0, a_1, ..., a_{i-1}, a_i, a_{i+1}, \cdots, a_{n-1})$ connects the following $2n$ nodes:*



(a)



(b)

Fig. 2. Structures of two balanced hypercubes: (a) BH_1, and (b) BH_2}

1) $\left((a_0+1) \, mod \, 4, a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_{n-1}\right)$,
$\left((a_0-1) \, mod \, 4, a_1, \cdots, a_{i-1}, a_i, a_{i+1}, \cdots, a_{n-1}\right)$, *and*

2) $\left((a_0+1) \; mod \; 4, a_1, \cdots, a_{i-1}, \left(a_i + (-1)^{a_0}\right) \; mod \; 4, a_{i+1}, \cdots, a_{n-1}\right)$,
$\left((a_0-1) \; mod \; 4, a_1, \cdots, a_{i-1}, \left(a_i + (-1)^{a_0}\right) \; mod \; 4, a_{i+1}, \cdots, a_{n-1}\right)$,

For convenience, we assume that all the arithmetic operations on indices of nodes in BH_n are four-modulated. Fig. 2 shows two graphs BH_1 and BH_2. Fig. 3 shows an incomplete BH_3 which consists of four BH_2s, where nodes in each of BH_2s are connected as in a complete BH_1. However, connections among nodes from different BH_2s are incomplete in Fig. 3, where only nodes (0, 0, 0), (1, 0, 0), (2, 0, 0), and (3, 0, 0) have complete connections. In BH_n, the first element a_0 of node $(a_0, a_1, \cdots, a_{i-1}, a_i, a_{i+1} \cdots, a_{n-1})$ is named *inner index,* and the other elements a_i $(1 \le i \le n-1)$ *outer indices.* Any node in BH_n has two types of adjacent nodes:

1) *inner* (based on 1 in Definition 1) and
2) *outer* (based on 2 in Definition 1).

Clearly, every node in BH_n has two inner adjacent nodes and $2n-2$ outer adjacent nodes. For example, in BH_3 shown in Fig. 3, node (1, 3, 2) has inner index 1 and two outer indices: 3 and 2. By adding 1 to and subtracting 1 from the inner index of node (1, 3, 2), two inner adjacent nodes, (0, 3, 2) and (2, 3, 2), are derived. Since the inner index of (1, 3, 2) is odd, the four outer adjacent nodes (0, 2, 2), (2, 2, 2), (0, 3, 1), and (2, 3, 1) can be determined by subtracting one of outer indices by 1 and by changing the inner index by 1.

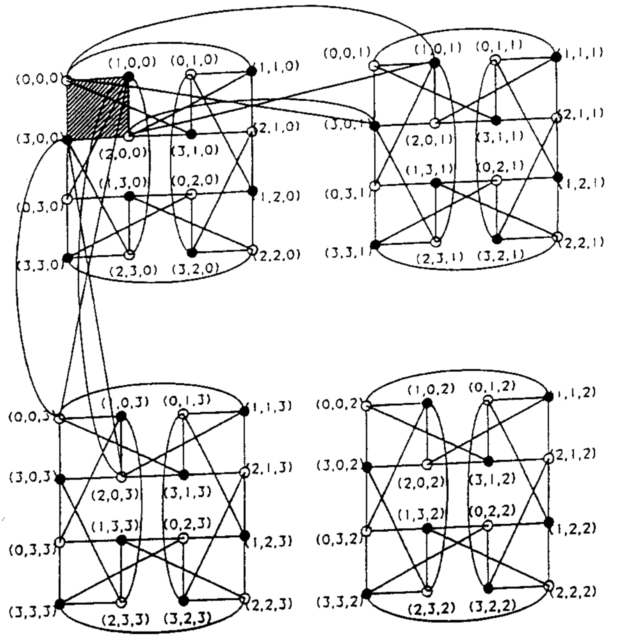


Fig. 3. The structure of BH_3.

An n-dimensional balanced hypercube is very similar to a $2n$-*dimensional hypercube* Q_{2n} in terms of their topological properties. Actually, these two structures share similar features in terms of symmetry and connectivity, as will be discussed in the following section. Based on Definition 1, it is clear that BH_n can be derived from four BH_{n-1}s by adding a new dimension as the nth outer index of every node in BH_n. If we name nodes with odd inner index *odd nodes* (represented as black nodes as in Figs. 2 and 3), similarly nodes with even inner index are named *even nodes* (represented as white nodes), we redefine BH_n as follows:

DEFINITION 2. *BH_n is constructed hierarchically as follows:*

1) *BH_1 is constructed from four nodes connected as a ring. These four nodes are labeled* 0, 1, 2, 3, *respectively.*

2) *BH_{k+1} is constructed from four BH_ks. These four BH_ks are labeled* $BH_k^{(0)}, BH_k^{(1)}, BH_k^{(2)}$, and $BH_k^{(3)}$, *where each node in* $BH_k^{(i)}(0 \le i \le 3)$ *has i attached as the new kth outer index.*

Every node $v = (a_0, a_1, \cdots, a_{k-1}, i)$ in $BH_k^{(i)}(0 \le i \le 3)$ *has two extra connections:*

a) $BH_k^{(i+1)}$: $(a_0 + 1, a_1, \cdots, a_{k-1}, i + 1)$ *and* $(a_0 - 1, a_1, \cdots, a_{k-1}, i + 1)$ *if a_0 is even.*

b) $BH_k^{(i-1)}$: $(a_0 + 1, a_1, \cdots, a_{k-1}, i - 1)$ *and* $(a_0 - 1, a_1, \cdots, a_{k-1}, i - 1)$ *if a_0 is odd.*

Clearly, Definitions 1 and 2 are equivalent. The addressing schemes used in both definitions are *quaternary*. It is easy to replace the quaternary representation by *binary* one as is used in the hypercube addressing scheme. This can be done by replacing every quaternary number by two binary numbers.

3 PROPERTIES OF BALANCED HYPERCUBES

In this section, we list some basic properties of the balanced hypercube BH_n. We compare these properties with the ones of the comparable hypercube, i.e., $2n$-dimensional hypercube Q_{2n}. Due to the space limitation, all the proofs to the following properties are omitted, see [6] for detail.

PROPERTY 1. *BH_n is a load balanced graph, and nodes in BH_n can be partitioned into a set of matching pairs $v = (a_0, a_1, \cdots, a_{n-1})$ and $v' = (a_0 + 2, a_1, \cdots, a_{n-1})$.*

For example, two nodes (0, 0, 0) and (2, 0, 0) form a matching pair in BH_3, because they have the same adjacent node set (1, 0, 0), (3, 0, 0), (1, 1, 0), (3, 1, 0), (1, 0, 1), (3, 0, 1).

PROPERTY 2. *BH_n has 2^{2n} nodes, each of which has $2n$ adjacent nodes.*

This property states that BH_n and Q_{2n} have the same number of nodes and node degree. The following property shows that BH_n and Q_{2n} also have the same symmetry property.

PROPERTY 3. *BH_n is a* symmetric *graph, i.e., for any pair of nodes v and u in BH_n there is an* automorphism *T of BH_n* such that $T(v) = u$.

If $v = (a_0, a_1, \cdots, a_{n-1})$ and $u = (b_0, b_1, \cdots, b_{n-1})$, then the automorphism T can be defined as:

$$T(w) = \left(b_0 + (-1)^{a_0}(c_0 - a_0), b_1 + (-1)^{a_0}(c_1 - a_1), \cdots, b_{n-1} + (-1)^{a_0}(c_{n-1} - a_{n-1})\right)$$

for any node $w = (c_0, c_1, \cdots, c_{n-1})$ in BH_n.

PROPERTY 4. *BH_n is a bipartite graph, i.e., its node set can be divided into two disjoint subsets such that any edge must link two nodes from different subsets.*

One of the desirable properties of the $2n$-dimensional hypercube Q_{2n} is the existence of $2n$ node-disjoint paths between any pair of nodes. BH_n has the same property as shown in the following.

PROPERTY 5. *BH_n is $2n$-connected, i.e., for any pair of nodes in BH_n there exist $2n$ disjoint paths between them.*

For example, in BH_2, there are $2 * 2 = 4$ node-disjoint paths from node (1, 2) to node (3, 3): path 1: (1, 2) $\to$ (2, 2) $\to$ (3, 3); path 2: (1, 2) $\to$ (0, 2) $\to$ (3, 3); path 3: (1, 2) $\to$ (2, 1) $\to$ (1, 1) $\to$ (2, 0) $\to$ (3, 0) $\to$ (2, 3) $\to$ (3, 3); path 4: (1, 2) $\to$ (0, 1) $\to$ (3, 1) $\to$ (0, 0) $\to$ (1, 0) $\to$ (0, 3) $\to$ (3, 3).

The diameter of a graph is an important measure of communication delay. Normally, the shorter the diameter the lower the communication delay. One of the desirable properties of the hypercube structure is its short (logarithmic) diameter. Property 6 shows that BH_n has a diameter no longer than that of a $2n$-dimensional hypercube Q_{2n}.

PROPERTY 6. *The diameter of BH_n is $2n$ when n is even or $n = 1$, and is $2n - 1$ when n is odd other than 1.*

Note that a $2n$-dimensional hypercube Q_{2n} has a diameter of $2n$. Therefore, BH_n has a smaller diameter than Q_{2n} when n ($n > 1$) is odd. In addition to diameter, *average distance* and *traffic density* are important among general measures of a multicomputer structure. Since BH_n is a symmetric graph, the average distance can be defined as:

$$\bar{d}(BH_n) = \frac{1}{2^{2n}} \sum_{\text{all node } v \text{ in } BH_n} d\left(\underbrace{(0,0,\cdots,0)}_{n}, v\right)$$

Similarly, the traffic density of BH_n is

$$td(BH_n) = \frac{(\text{average distance}) * (\text{number of nodes})}{\text{number of links}} = \frac{\bar{d}(BH_n) * (2^{2n})}{n * 2^{2n}}$$

Tables 1 and 2 show the average distance and the traffic density of balanced hypercubes, respectively. Note that the average distance of a $2n$-dimensional hypercube is $\bar{d}(Q_{2n}) = n$ and the traffic density of a $2n$-dimensional hypercube is $td(Q_{2n}) = 1$. Therefore, both $\bar{d}(BH_n)$ and $td(BH_n)$ are very close to $\bar{d}(Q_{2n})$ and $td(BH_n)$, respectively.

TABLE 1
AVERAGE DISTANCES OF *BH*S

the size n	$\bar{d}(BH_n)$	the size n	$\bar{d}(BH_n)$
1	1.000000	11	11.072663
2	2.250000	12	12.066152
3	3.156250	13	13.060060
4	4.140625	14	14.054409
5	5.126953	15	15.049202
6	6.114258	16	16.044432
7	7.103638	17	17.040075
8	8.094727	18	18.036113
9	9.086861	19	19.032518
10	10.079563	20	20.029262

TABLE 2
TRAFFIC DENSITIES OF *BH*S

the size n	$td(BH_n)$	the size n	$td(BH_n)$
1	1.000000	11	1.006606
2	1.125000	12	1.005513
3	1.052083	13	1.004620
4	1.035156	14	1.003886
5	1.025391	15	1.003280
6	1.019043	16	1.002777
7	1.014805	17	1.002357
8	1.011841	18	1.002006
9	1.009651	19	1.001711
10	1.007965	20	1.001463

BH_n also supports efficient routing. An optimal routing algorithm has been presented in [6] as well as an area efficient layout of balanced hypercubes in [5].

In general, the ability to embed basic interconnection networks is one of the important measures of a newly proposed network. More formally, an embedding ϕ of an application graph $G_a = (V_a, E_a)$ into a host graph $G_h = (V_h, E_h)$ is an injective function from V_a to V_h.[1] The *expansion* of ϕ is $|V_h| / |V_a|$. The *dilation* of ϕ is $max\{d(\phi(v_i), \phi(v_j))\}$, for every $(v_i, v_j) \in E_a$. The expansion is a measure of processor utilization. The dilation represents the maximum communication delay between the communicating nodes. The load of an embedding is the maximum number of processors of G_a assigned to any processor of G_h. Throughout the paper we only consider embedding with unit dilation and load.

We use embedding rings in balanced hypercubes as an example to show the embeddability of balanced hypercubes. Embedding of other topologies, such as trees, is studied in [12]. To embed rings in balanced hypercubes, we need to find, for every node v, the next adjacent node $f_n(v)$ on the ring. The idea is to find directly spanning rings in BH_1 and BH_2, and use the spanning ring in BH_2 as the building block to construct a spanning ring in BH_3, and so on. In general, a spanning ring in BH_n is constructed from four spanning rings in four BH_{n-1}s. This is done by breaking each spanning ring and then connecting each broken ring using four links that connect these four BH_{n-1}s. The following theorem shows one of the possible solutions based on the above idea.

THEOREM 1. *A map f_n on the node set of an n-dimersional balanced hypercube, $BH_n = (V_n, E_n)$, is constructed based on the following three rules:*

1)
$$f_n\left(2, \underbrace{2, \cdots, 2}_{l}, 0, a_{l+2}, a_{l+3}, \cdots, a_{n-1}\right) = \left(1, \underbrace{2, \cdots, 2}_{l}, 0, a_{l+2}+1, a_{l+3}, \cdots, a_{n-1}\right), \text{ where } 0 \le l < n-1,$$

2) *$f_n(0, a_1, a_2, \cdots, a_{n-1}) = (3, a_1 + 1, a_2, \cdots, a_{n-1})$, and*

3) *$f_n(i, a_1, a_2, \cdots, a_{n-1}) = (i - 1, a_1, a_2, \cdots, a_{n-1})$, if node $(i, a_1, a_2, \cdots, a_{n-1}) \in V_n$ does not match 1 or 2 as above.*

The set of directed edges $E'_n = \{(v, f_n(v)); v \in V_n\}$ forms a Hamiltonian cycle C_n in BH_n.

PROOF. In the definition of map f_n, rule 3 is used to construct a ring in each inner cube of four nodes. Therefore, when $n = 1$, the Hamiltonian cycle C_1 is generated directly by applying rule 3. Rules 2 and 3 together are used to construct a ring in BH_2. Rule 2 supersedes rule 3 in order to break and connect rings in four BH_1s. The Hamiltonian cycle C_2 generated by using rules 2 and 3 is shown in Fig. 2b. Rule 1 is used recursively to break and connect spanning rings in each subcube BH_l, $2 \le l \le n - 1$, of BH_n. By using the induction on the dimension size n of BH_n, we show the following fact: $E'_n = \{(v, f_n(v)); v \in V_n\}$ forms a Hamiltonian cycle C_n in BH_n, and edge

$$\left(\left(1, \underbrace{2, \cdots, 2}_{n-2}, 0\right), \left(2, \underbrace{2, \cdots, 2}_{n-2}, 0\right)\right) \in E'_n,$$

for $n \ge 2$. When $n = 2$, C_2 can be easily constructed in BH_2 and the edge $((1, 0), (2, 0)) \in C_2$. Assume that when $n = l$ ($l \ge 2$), the fact is true, we verify it for $n = l + 1$. Using the following equations:

$$f_{l+1}(v, i) = \begin{cases} (f_l(v), i) & \text{if } v \ne \left(2, \underbrace{2, \cdots, 2}_{l-2}, 0\right) \\ \left(1, \underbrace{2, \cdots, 2}_{l-2}, 0, i+1\right) & \text{otherwise} \end{cases}$$

we see that f_{l+1} keeps all the links in C_l (formed by f_l) except the edge

$$\left(\left(1, \underbrace{2, \cdots, 2}_{l-2}, 0, i\right), \left(2, \underbrace{2, \cdots, 2}_{l-2}, 0, i\right)\right).$$

Thus, C_{l+1} is constructed from four C_ls as shown in Fig. 4, where $C_l^{(i)}$ is the Hamiltonian cycle C_l formed by f_l in $BH_l^{(i)}$, for $0 \le i \le 3$. Therefore, when $n = l + 1$, $E'_n = \{(v, f_n(v)); v \in V_n\}$ forms a Hamiltonian cycle C_n in BH_n. Clearly the edge

$$\left(\left(1, \underbrace{2, \cdots, 2}_{n-2}, 0\right), \left(2, \underbrace{2, \cdots, 2}_{n-2}, 0\right)\right)$$

is still in the cycle C_n. □

COROLLARY. *Any rings of size of 2^{2l}, $1 \le l \le n$, can be embedded in BH_n.*

Based on the proof of Theorem 1, we can always construct a spanning ring of size 2^{2l} in each subcube BH_l of BH_n. For example, rings with 4, 16, 64, 256 can be embedded in BH_4 which consists of

1. Although we can define a more general concept of embedding by replacing the injective function by the regular function in the definition, it is beyond the scope of this paper.

$2^8 = 256$ nodes. In the next section, we show that any ring of size 2^{2l+1}, $1 \le l \le n - 1$, can also be embedded in BH_n.

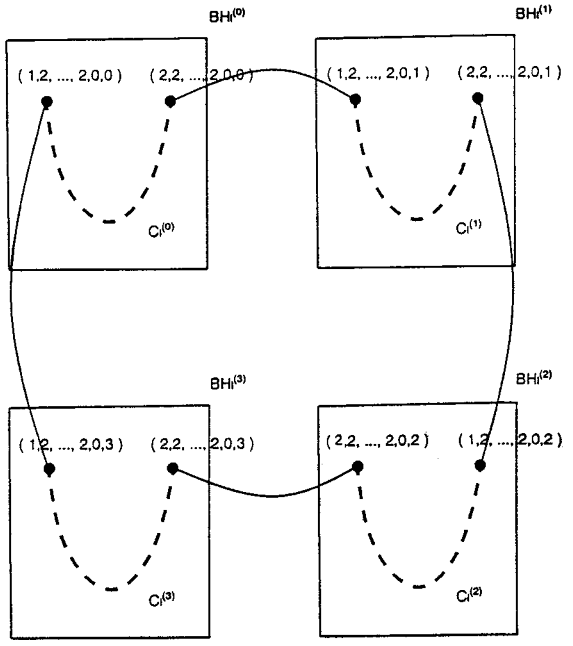


Fig. 4. Four C_ls form a C_{l+1}.

4 An Application of BHs: Fault-Tolerant Ring Embeddings

We study fault-tolerant embeddings of rings as a possible application of balanced hypercubes. In a regular embedding, once we have mapped an application graph into a host graph, it is desirable that if the host has a fault, it can reconfigure itself so that the application graph can continue operation with minimum interruption while still maintaining the same or near the same embedding quality. Therefore, a good embedding should not only try to optimize the initial embedding, by minimizing dilation, congestion, and load, but also should be easy to recover from faults and still maintain certain level of quality of the embedding. Such type of fault-tolerant embeddings are called *recoverable embeddings*.

We consider embeddings with unit dilation and load in both the initial embedding ϕ and the recovered embedding ϕ'. More specifically, we study *k-fault-tolerant t-step-recoverable embeddings*, or (k, t) embeddings [7], which means that a maximum of t steps are required to recover a system with k faulty processors and the recovered embedding is isomorphic to the initial embedding.

Efficient system reconfiguration is the key in achieving an efficient recoverable embedding. Basically, in a system reconfiguration faulty components are disconnected from the system and spares are included. We use the scheme proposed in [14], where control is distributed throughout the system. In some other reconfigurations, it is usually assumed that reconfiguration and recovery are directed by a central or global supervisor, which is often the most vulnerable part of the system. In addition, centralized control is not practical in a large multicomputer system. In distributed reconfiguration, depending on how tasks are actually moved, the reconfiguration strategies can be divided into:

1) local reconfiguration, which involves only local task movements, and
2) global reconfiguration, which involves global task movements.

Ideally, we want a distributed reconfiguration in which reconfiguration process is distributed, there is no scattered information about fault distribution, and all the reconfigurations are local ones without global data movements. Moreover, we require that such a process can handle multiple faults and ideally be independent of the structure of the application graph.

Unfortunately, most of the existing topologies cannot effectively support an efficient reconfiguration process, especially in multiple-fault cases. For example, consider embedding a ring into a hypercube; results [9] show that there exists an initial embedding such that any single fault can be recovered in one step. However, we will show (in Theorem 2) that there does not exist an embedding that can recover any simple and double faults in one and two steps, respectively. Recent results [13] show that there exists distributed reconfiguration (with degradation of the quality of the initial embedding) of multiple faults in hypercubes. However, more than k steps are used to recover from k faults and different reconfiguration processes have to be employed for different guest graphs.

We propose here the concept of *consistently recoverable embedding*, or *consistent-(k, k) embedding*, which provides a uniform (i, i) embedding with $1 \le i \le k$ for a given positive integer k; that is, it provides a uniform recoverable embedding which is independent of the number and location of up to k faults. We first formally define the concept of the *consistent-(k, k) embedding*, then we propose a consistent-(k, k) embedding of a ring in a balanced hypercube. The detailed discussion of the usefulness of consistent-(k, k) embedding and consistent-(k, k) embedding of other topologies, such as trees, are studied in [12].

Definition 3. *A consistent-(k, k) embedding is an (i, i) embedding for all i such that $1 \le i \le k$, where k is a given positive integer, and each node has a distinct spare. If each recovery step is defined as replacing each faulty node by its spare, then the resultant recovered embedding is isomorphic to the initial embedding.*

In the above definition, since each faulty node is replaced by its spare in one step, a total of i steps are required in a system with i faults, where $1 \le i \le k$. Hence it is an (i, i) embedding. Also, with each node having a distinct spare node, the configuration is independent of the number and location of up to k faults in the system. Therefore, it is consistent. We use consistent-($*$, $*$) embedding to represent consistent-(k, k) embedding without setting limits on value k, i.e., any (large) k faults can be recovered in k steps. Clearly, conditions associated with the consistent-(k, k) embedding are strong. The following result shows that there does not exist a consistent-(k, k) ($k \ge 2$) embedding of rings in standard hypercubes.

Theorem 2. *There does not exist a consistent-(k, k) ($k \ge 2$) embedding of rings in standard hypercubes.*

Proof. It is sufficient to show that there does not exist a consistent-(2, 2) embedding, because based on Definition 3 any consistent-(k, k) embedding ($k > 2$) implies a consistent-(2, 2) embedding. Suppose nodes u, v, w, x (see Fig. 5) in sequence are used in the initial embedding of a consistent-(k, k) embedding of a ring in a hypercube where v' and w' are spares for v and w, respectively. (The smallest ring in a hypercube has four nodes.) If the system has one faulty node, say v, then the recovered embedding is obtained by replacing v using v'. Since this recovered embedding is isomorphic to the initial embedding, v' must be adjacent to both neighbors, u and w, of v. Similarly, w' must be adjacent to both v and x. Now, if the system has two faulty nodes, v and w, based on the definition of consistent-(k, k) embedding, v' and w' are used to replace faulty nodes v and w, respectively. Since the recovered embedding is isomorphic to the initial embedding, nodes v' and w' are connected (see Fig. 5). This implies that there are three node-disjoint paths of length two (via nodes u, w, and w', respectively) between nodes v and v'. This contradicts the hypercube property that there are only l

node-disjoint paths of length l between two nodes separated by Hamming distance l in a hypercube. □

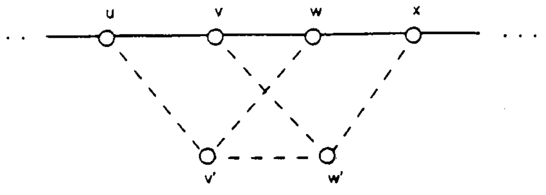


Fig. 5. The connection requirement of a consistent-(k, k) embedding of a ring.

TABLE 3
THE SPARE NODE SET OF A RING OF EIGHT PROCESSORS IN BH_2

node v	1	2	3	4	5	6	7	8
imaging node $\phi(v)$	(0,0)	(1,0)	(0,3)	(1,3)	(0,2)	(1,2)	(0,1)	(1,1)
spare node $\phi'(v)$	(2,0)	(3,0)	(2,3)	(3,3)	(2,2)	(3,2)	(2,1)	(3,1)

Using a similar approach, we can prove that there does not exist a consistent-(k, k) ($k \geq 1$) embedding of any nontrivial binary trees in standard hypercubes.

In balanced hypercubes, the reconfiguration method associated with a consistent-(k, k) embedding can be simply expressed as follows: Whenever a processor (a_0, a_1, ..., a_{n-1}) fails, this processor should be replaced by its matching node, i.e., processor (a_0 + 2, a_1, ..., a_{n-1}).

The spare (matching) node set of a ring of eight processors in BH_2 of Fig. 6a is shown in Table 3. To find a consistent-(k, k) embedding of an application graph in a balanced graph, as long as we use at most one node in each matching pair in the initial embedding, the result is guaranteed consistently recoverable and there is no limit on the number of faults (k) to be tolerated; that is, the embedding is consistent-($*$, $*$). A consistent-($*$, $*$) embedding of rings in *BHs* with an expansion two (which corresponds to an embedding with the best possible system utilization) is shown as follows:

THEOREM 3. *In an n-dimensional balanced hypercube, $BH_n = (V_n, E_n)$, we define a map g_n on the node subset*

$$V_n' = \{(a_0, a_1, \cdots, a_{n-1}) \in V_n;\ a_0 \in \{0, 1\}\}$$

as follows:

1) $g_n\left(0, \underbrace{2, \cdots, 2}_{l}, 0, a_{l+2}, a_{l+3}, \cdots, a_{n-1}\right) =$

$\left(1, \underbrace{2, \cdots, 2}_{l}, 0, a_{l+2}+1, a_{l+3}, \cdots, a_{n-1}\right)$, *where* $0 \leq l < n-1$,

2) $g_n(0, a_1, a_2, \cdots, a_{n-1}) = (1, a_1, a_2, \cdots, a_{n-1})$, *if node* $(0, a_1, a_2, \cdots, a_{n-1})$ *does not match* 1) *above, and*

3) $g_n(1, a_1, a_2, \cdots, a_{n-1}) = (0, a_1 - 1, a_2, \cdots, a_{n-1})$.

The set of edges $E_n'' = \{(v, g_n(v));\ v \in V_n\}$ *forms a cycle with* $2 * 4^{n-1}$ *nodes in* BH_n.

This theorem can be proved in a similar way used in proving Theorem 1. The following equations are useful to construct such a proof:

$$g_{k+1}(v,i) = \begin{cases} (g_k(v), i) & \text{if } v \neq \left(0, \underbrace{2, \cdots, 2}_{l-2}, 0\right) \\ \left(1, \underbrace{2, \cdots, 2}_{l-2}, 0, i+1\right) & \text{otherwise,} \end{cases}$$

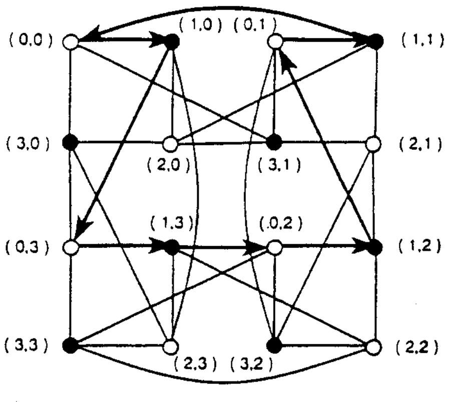
and

$$\begin{cases} g_2(0,i) = (1,i) \\ g_2(1,i) = (0,i-1) \end{cases}$$

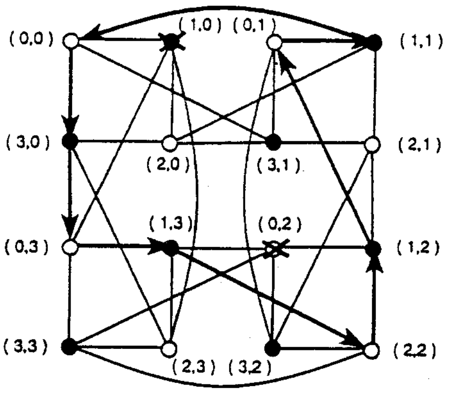
It is clear that map g_m provides a consistent-($*$, $*$) embedding of rings in balanced hypercubes. Fig. 6a shows a consistent-($*$, $*$) embedding of a ring of eight processors in BH_2.

COROLLARY. *For any rings of size* 2^{2l+1}, $1 \leq l < n$, *there exists a consistent-($*$, $*$) embedding in* BH_n.

Efficient system reconfiguration is the major advantage of the proposed consistent-(k, k) embedding over the existing ones. Because the reconfiguration is consistent, there is no need for each local supervisor to know the global information of the number and location of faulty processors. Therefore, the reconfiguration is much simpler than the one used in a regular recoverable embedding where a certain format of global information collection process is required. More specifically, in the decentralized reconfiguration method, each processor is assumed to be capable of acting as a local supervisor of reconfiguration and recovery. It is nor-



(a)



(b)

Fig. 6. The consistent-(k, k) embedding of a ring in BH_2: (a) the initial embedding before the failure of nodes (1,0) and (0,2), and (b) the recovered embedding.

mally assumed that the processor that detects a faulty node acts as a local supervisor. With the consistent-(k, k) embedding, one reconfiguration step is required for each faulty processor. In addition, based on the topological property of the balanced hypercube, each local supervisor is adjacent to both the node to be replaced (the faulty processor) and the replacing node (the matching node of the faulty processor). This condition holds no matter which neighbor of the faulty node detects the faulty and acts as a local supervisor. For the example of Fig. 6a, suppose nodes (1, 0) and (0, 2) are faulty and they are detected by nodes (0, 0) and (1, 3), respectively. Then, node (0, 0) acts as a local supervisor to replace (0, 0) by its matching node (3, 0). Note that (3, 0) is adjacent to (0, 0). Similarly, node (1, 3) acts as another local supervisor to replace (0, 2) by (2, 2). The result is shown in Fig. 6b. Suppose these faulty nodes are detected by the other two neighbors, e.g., (0, 3) and (1, 2). The same replacing nodes will be used and both local supervisors are still adjacent to their respective replacing nodes.

5 Conclusion

We have proposed the balanced hypercube structure, which is a variation of the standard hypercube. The basic properties of balanced hypercubes have been compared to those of hypercubes. Results show that balanced hypercubes still maintain many desirable properties of hypercubes. In addition, balanced hypercubes have been shown to be superior over the hypercubes in the following two aspects:

1) Balanced hypercubes support an efficient reconfiguration without changing the adjacency relationship among tasks, and
2) An odd-dimensional balanced hypercube has a smaller diameter than the comparable hypercube.

Ring embedding, including a fault-tolerant ring embedding using consistently recoverable embedding, has also been discussed. Results confirm that the balanced hypercube has better fault-tolerant embedding capability than the standard hypercube. We believe that the proposed consistent recoverable embedding is a practical and useful solution to minimize reconfiguration time in a decentralized multicomputer system. The balanced hypercube has been proved to be a suitable structure for supporting this method.

References

[1] L.N. Bhuyan and D.P. Agrawal, “Generalized Hypercube and Hyperbus Structures for a Computer Network,” *IEEE Trans. Computers*, vol. 32, no. 4, pp. 323–333, Apr. 1984.

[2] S. Dutt and J.P. Hayes, “Design of Fault-Tolerant Systems Using Automorphisms,” *J. Parallel and Distributed Computing*, vol. 12, pp. 249–268, 1991.

[3] A. El-Amawy and S. Latifi, “Properties and Performance of folded hypercubes,” *IEEE Trans. Parallel and Distributed Systems*, vol. 2, no. 1, pp. 31–42, Jan. 1991.

[4] J.P. Hayes and T.N. Mudge, “Hypercube Supercomputers,” *Proc. IEEE*, vol. 77, no. 12, pp. 1,829–1,841, Dec. 1989.

[5] K. Huang and J. Wu, “Balanced Hypercubes,” Technical Report CSE-92-6, Dept. of Computer Science and Eng., Florida Atlantic Univ., Feb. 1992.

[6] K. Huang and J. Wu, “Area Efficient Layout of Balanced Hypercubes,” *Int’l J. High Speed Electronics and Systems*, vol. 6, no. 4, pp. 631–646, 1995.

[7] T.C. Lee, “Quick Recovery of Embedding Structures in Hypercube Computers,” *Proc. Fifth Distributed Memory Computing Conf.*, pp. 1,426–1,435, 1990.

[8] T.C. Lee and J.P. Hayes, “One-Step-Degradable Fault-Tolerant Hypercubes,” *Proc. Fourth Conf. Hypercubes*, pp. 94–97, Mar. 1989.

[9] J. Liu, T.J. Sager, and B.M. McMillin, “An Improved Characterization of 1-step Recoverable Embeddings: Rings in Hypercubes,” Technical Report CS-92-07, Univ. of Missouri at Rolla, Nov. 1992.

[10] N. Sherwani, A. Boals, and H. Ali, “Load Balancing Graphs,” *Congressus Numerantium*, vol. 73, pp. 205–214, 1990.

[11] N.F. Tzeng and S. Wei, “Enhanced Hypercubes,” *IEEE Trans. Computers*, vol. 40, no. 3, pp. 284–294, Mar. 1991.

[12] J. Wu and K. Huang, “Fault-Tolerant Ring Embedding in Balanced Hypercubes,” Technical Report CSE-93-5, Dept. of Computer Science and Eng., Florida Atlantic Univ., Jan.1993.

[13] P-.J. Yang, S-.B. Tien, and C.S. Raghavendra, “Reconfiguration of Rings and Meshes in Faulty Hypercubes,” *J. Parallel and Distributed Computing*, vol. 22, pp. 96–106, 1994.

[14] R.M. Yanney and J.P. Hayes, “Distributed Recovery in Fault-Tolerant Multiprocessor Networks,” *IEEE Trans. Computers*, vol. 35, pp. 871–879, 1986.