Efficient Resource Placement in Hypercubes Using Multiple-Adjacency Codes

Hsing-Lung Chen, *Member, IEEE,* and Nian-Feng Tzeng, *Senior Member, IEEE*

***Abstract*—While a certain resource in the hypercube may be shared by cube nodes to lower the cost, multiple copies of a shared resource often exist in the hypercube to reduce contention, and thus the potential delay, in fetching any shared copy. It is desirable that one employs as few resource copies as possible to ensure that every node is able to reach the resource in a given number of hops, achieving efficient resource placement. This placement method also keeps system performance degradation minimal after one resource copy becomes unavailable due to a fault.**

First, we consider placing multiple copies of a certain resource in a way that every cube node without the resource is adjacent to a specified number of resource copies. The use of our developed perfect and quasiperfect multiple-adjacency codes makes it possible to arrive at efficient solutions to this placement problem in a simple and systematic manner for an arbitrary hypercube. We then deal with the generalized resource placement in the hypercube such that every node without the resource can reach no less than a specified number of resource copies in no more than a certain number of hops, using as few resource copies as possible. Our placement results yield lowest potential access contention for a given number of resource copies (i.e., cost), particularly useful for large-scale hypercubes.

***Index Terms*—Access contention, Hamming codes, hypercubes, linear block codes, resource placement, system performance.**

I. Introduction

THE hypercube has become popular recently due to its cost-effective potential in offering high computation power needed by various applications. It involves many processing nodes that work in a cooperative fashion to solve problems efficiently. A hypercube is known to possess many attractive properties, including low diameter, easy routing, efficient support of various algorithms, rich fault tolerance, and so on [1], [2], [8].

In the hypercube computer, a certain resource might have to be shared by processing nodes because of two reasons: 1) it is too expensive to equip every node with a dedicated copy of the resource, or 2) it might be unnecessary to install at every node a copy of the resource that is used infrequently. Resources range from hardware devices (such as I/O processors, disks, special-purpose units, etc.) to software modules (such as library routines, compilers, data files, etc.). The access of a shared resource tends to involve delay attributed by possible contention with other access requests and by communication latency from a node without such a resource to reach a node with the resource. The first delay component (due to contention) can be reduced by lowering the number of nodes that share a copy, whereas the second component is kept small by minimizing the mean number of hops taken to arrive at a node with the resource.

Frequently, multiple copies of each type of resource exist in a hypercube to reduce the delay of shared resource access, thereby improving system performance. The use of multiple copies also enhances reliability because the loss of one copy would not render one type of resource totally unavailable. Distributing resource copies in a hypercube with an attempt to optimize system performance measures of interest has been investigated recently [3]–[5]. Livingston and Stout [3] have studied the minimum number of resource copies needed to meet certain specified requirements when distributing the resource in a hypercube computer. A method has been proposed in [4] for mapping the I/O processors onto a hypercube such that each cube node is adjacent to at least one I/O processor. More recently, efficient algorithms for allocating a given number of resource copies to a hypercube system in an effort to optimize a defined performance measure have been developed [5].

Most of the prior articles aimed to optimize specified measures under the situation that every cube node is connected with only one copy of the resource. Under this situation, excessive sharing of one resource copy may occur in a large system because every copy has to support an increasing number of nodes as the system dimension grows, degrading overall system performance due to unnecessarily heavy contention, and thus long access delay. In addition, the loss of one node (and its associated resources) would make several nodes redirect their requests destined for the lost node to other remote nodes. Redirected requests not only take more hops to reach their destination nodes, but also increase contention at those nodes, possibly making the access delay of shared resources significantly larger. In Fig. 1, for example, every node without the resource is immediately adjacent to a node with the resource; every copy of the resource serves four nodes. After one node with a resource copy, say node (111), loses, three nodes are no longer immediately adjacent to the resource, and the resource associated with node (000) has to serve all healthy nodes. (We assume that if a node fails,

Manuscript received January 6, 1992; revised August 15, 1992. A preliminary version of the work was presented at the 20th Annual International Conference on Parallel Processing, August 1991. H.-L. Chen was supported in part by the National Science Council of the R.O.C. under Contract NSC-81-0408-E011-04. N.-F. Tzeng was supported in part by NSF under Grants MIP-8807761 and MIP-9201308 and by the State of Louisiana under Contract LEQSF(1992–94)-RD-A-32.

H.-L. Chen is with the Department of Electronic Engineering, National Taiwan Institute of Technology, Taipei, Taiwan, ROC.

N.-F. Tzeng is with The Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504.

IEEE Log Number 9208772.

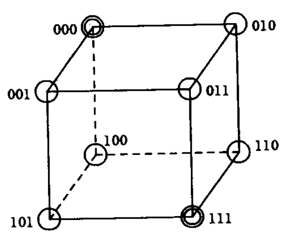

Fig. 1. Three-dimensional hypercube. (Nodes with double circles have one resource copy each.)

its associated resources, if any, would be lost as well.) As a result, after a fault arises the hypercube could suffer from considerable performance degradation simply because of the increase in access delay.

It might require that every nonresource node be in connection with multiple copies of each resource in a large-sized system to prevent any copy from becoming congested or a bottleneck, ensuring good performance. In this paper, we consider placing copies of a certain resource in a way that every node without the resource is adjacent to a specified number of resource copies, and then that every node without the resource can reach a specified number of resource copies in no more than a certain number of hops. This placement method guarantees the resource to be accessible from every node within the same number of hops even after certain resource copies are lost, reducing the potential access contention of any shared copy. The proposed resource placement strategy is based on our developed linear block codes, which can be applied easily to get a placement with any degree of adjacency in any sized hypercube using as few copies of a resource as possible. The locations to place resource copies are specified by the codewords of a linear block code, giving rise to the best result when a perfect placement exists.

It should be noted that Reddy extended his basic single-adjacency method to yield I/O embeddings such that every node is adjacent to multiple I/O processors [4]. However, both extensions of his basic method fail to produce an optimal result for any cube size that is possible to have a perfect multiple-adjacency placement. Furthermore, his extended approaches become increasingly complicated when the degree of required adjacency grows. Unlike our investigation, Reddy's study requires every node to have multiple adjacency, not just nonresource nodes.

The rest of this paper is organized as follows. In Section II, necessary notations and useful background are given. Section III deals with the resource placement in a way that every cube node without the resource is adjacent to multiple copies of the resource. The generalized resource placement is treated in Section IV. Section V concludes this paper.

II. Preliminaries

A. Notations and Background

An n-dimensional hypercube, denoted by Q_n, consists of 2^n nodes, each addressed by a unique n-bit identification number. A link exists between two nodes of Q_n if and only if their node addresses differ in exactly one bit position. A link is said to be along dimension i if it connects two nodes whose addresses differ in the ith bit (where the least significant bit is referred to as the 0th bit). Q_3 is illustrated in Fig. 1.

Two nodes in a hypercube are said to be *adjacent* if there is a link present between them. The *distance* between any two cube nodes is the number of bits differing in their addresses. For example, the distance between nodes (011) and (101) is 2. The number of *hops* (i.e., traversals) needed to reach a node from another equals the distance between the two nodes. A d-dimensional subcube in Q_n involves 2^d nodes whose addresses belong to a sequence of n symbols $\{0, 1, *\}$ in which exactly d of them are of the symbol * (i.e., the *don't care* symbol whose value can be 0 or 1). The two-dimensional subcube (1**0) in Q_4, for example, involves four nodes: (1000), (1010), (1100), and (1110).

Our resource placement strategy is based on the *linear block code,* which is briefly reviewed subsequently. For more detailed discussion, please refer to [6, ch. 3], [7, chap. 3], or any textbook on coding theory. A (binary) block code of size M over bit symbols $\{0, 1\}$ is a set of M binary sequences of length n called *codewords.* It is often that M equals 2^k for some integer k, and we are interested in this case only, referred to as an (n, k) code. An (n, k) block code, denoted by $\Psi(n, k)$, can be described concisely using a $k \times n$ matrix $\mathbf{G}$ called the *generator matrix.* For a linear block code, any codeword is a linear combination of the rows of its associated generator matrix $\mathbf{G}$. The linear combination is performed by modulo-2 additions over corresponding bits. Consider a simple example of a binary linear code $\Psi(3, 1)$ with the following generator matrix:

$$\mathbf{G} = [1 \quad 1 \quad 1]. \tag{1}$$

This linear code involves totally 2^1 codewords (as $k = 1$), which are the possible linear combinations of the row of the generator matrix $\mathbf{G}$ given above, namely, 000 and 111.

Let W be a subspace, then the set of vectors orthogonal to W is called the *orthogonal complement* of W. If W, a subspace of a vector space of n-tuples, has dimension k (i.e., consisting of k basis vectors), then the orthogonal complement of W has dimension $n - k$ [6, p. 41]. The subspace formed by code $\Psi(n, k)$ has dimension k, so its orthogonal complement has dimension $n - k$. Whether or not an n-tuple $\mathbf{c}$ is a codeword of code $\Psi(n, k)$ can be checked by using an $(n-k) \times n$ matrix $\mathbf{H}$, called a *parity-check matrix* of the code. $\mathbf{c}$ is a codeword if and only if it is orthogonal to every row vector of $\mathbf{H}$, namely, $\mathbf{c} \cdot \mathbf{H}^T = \mathbf{0}$, where $\mathbf{H}^T$ is the transpose of matrix $\mathbf{H}$. Matrix $\mathbf{H}$ can be constructed as follows: Every row of $\mathbf{H}$ is a basis vector of the orthogonal complement of $\Psi(n, k)$. For example, a parity-check matrix $\mathbf{H}$ of the binary code $\Psi(3, 1)$ with the generator matrix in (1) is

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix}. \tag{2}$$

Notice that the choice of $\mathbf{H}$ is not unique, and any chosen $\mathbf{H}$ satisfies $\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}$. This equality is essential and is used frequently throughout this paper.

If a parity-check matrix **H** for a binary code has r rows, then each column is an r-bit binary number and there are at most $2^r - 1$ nonzero columns. The "largest" parity-check matrix **H** that can be obtained is r by $2^r - 1$. In other words, the columns of this largest parity-check matrix **H** consist of all the possible nonzero r-tuples (totally, $2^r - 1$ of them). The matrix given in (2) is such a parity-check matrix. This largest parity-check matrix **H** is special and, in fact, is the *parity-check matrix of a Hamming code.* The generator matrix of a Hamming code is $(2^r - 1 - r) \times (2^r - 1)$. The Hamming code is a special linear code $\Psi(n, k)$ such that n equals $2^r - 1$ and k is equal to $2^r - 1 - r$. Similar to linear code $\Psi(3, 1)$, codes $\Psi(7, 4), \Psi(15, 11)$, and $\Psi(31, 26)$ are examples of the Hamming codes. One possible parity-check matrix **H** of the Hamming code $\Psi(7, 4)$ is

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{bmatrix}. \quad (3)$$

Any linear block code possesses the following basic property [7, p. 64]. Note that the minimum distance (i.e., minimum weight) of a code is the minimum distance between its two most similar codewords.

Property 1: Let **H** be a parity-check matrix of a linear code. The minimum distance of the code is equal to the smallest number of columns of **H** that sum to **0.**

It is obvious that if no $d - 1$ or fewer columns of **H** add to **0**, the code has a minimum distance of no less than d. For the parity-check matrix of Hamming code $\Psi(7, 4)$ given in (3), as an example, the minimum distance of the code is no less than three (necessary to achieve single-error correction) because no two or fewer columns could sum to **0**, as all the columns of **H** are nonzero and no two of them are alike.

B. Resource Placement Overview

The problem of placing a certain resource with multiple identical copies, say ξ copies, at nodes in Q_n is to find the set of nodes where these ξ copies are located such that a measure of interest is optimized.

1) I/O Processors Allocation: Reddy [4] considered allocating I/O processors (the resource) to cube nodes in a way that every node is adjacent to at least one I/O processor. A perfect allocation results if and only if every cube node is adjacent to exactly one I/O processor. A perfect allocation is shown to exist in only some cubes, namely, only for Q_n with $n = 2^r - 1$. Since the Hamming codes are known to be perfect codes [6], [7], a perfect allocation is obtained by placing the I/O processors at Q_n nodes with addresses being the codewords of Hamming code $\Psi(n, k)$. An example perfect allocation for Q_3 is illustrated in Fig. 1, where the two resource copies are placed at the codewords of Hamming code $\Psi(3, 1)$ defined by (1).

The allocation of I/O processors in Q_n for $n \neq 2^r - 1$ is also considered in [4]. Let n' be the largest value that is less than n and equal to $2^r - 1$. The allocation for Q_n is obtained by viewing Q_n as a union of $2^{n-n'}$ n'-dimensional subcubes and by arriving at a perfect allocation in each of the subcubes individually. For example, an allocation for Q_4 is derived by finding a perfect allocation for each of the two three-dimensional subcubes, as depicted in Fig. 2.

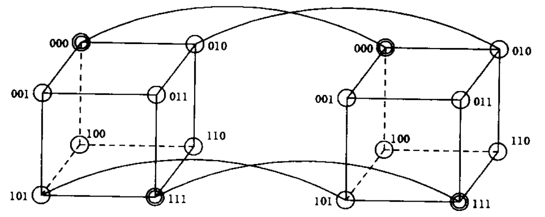

Fig. 2. Possible placement of 2^2 resource copies in Q_4. (Nodes with double circles have one resource copy each. Some links are omitted intentionally for clarity.)

2) Resource Allocation: Efficient algorithms for optimal or near-optimal resource allocation in hypercubes have been developed recently by Chiu and Raghavendra [5]. The performance measure of resource allocation to be optimized is the *resource diameter,* defined as the maximum resource distance among all the cube nodes, where the resource distance of a node is the minimum number of hops from the node to a node equipped with a copy of the resource. An allocation with the resource diameter minimized is highly desirable, as such an allocation tends to result in better mean response time because of fewer hops and thus less traffic involved.

The hierarchical allocation strategy proposed in [5] for placing 2^k copies of a resource in Q_n is equivalent to finding an optimal partition of the given system parameters n and k. Specifically, the strategy searches for a partition of n into l components, $n_1, n_2, \cdots, n_l$ (i.e., $n = n_1 + n_2 + \cdots + n_l$), and a partition of k into l components, $k_1, k_2, \cdots, k_l$ (i.e., $k = k_1 + k_2 + \cdots + k_l$), such that the summation of all the resource diameters, $d_i, 1 \leq i \leq l$, is minimized, where d_i is the resource diameter of the allocation of the ith partitioned component pair (n_i, k_i) according to a perfect code (such as the Hamming code and the Golay code [6, chap. 5.8], [7, chap. 5.3]) or a basic strategy stated subsequently.

The basic strategy of placing 2^k resource copies in Q_n is as follows [5]: Q_n is first partitioned into 2^{k-1} subcubes, each with dimension $n - k + 1$; and then two copies of the resource are placed in each $(n - k + 1)$-dimensional subcube at any two opposite corners, say at locations $(00 \cdots 0)$ and $(11 \cdots 1)$. Fig. 2 illustrates a possible placement of 2^2 resource copies in Q_4, following the basic strategy. It is shown [5] that a placement resulting from the basic strategy has the resource diameter of $\lfloor (n_i - k_i + 1)/2 \rfloor$.

If a partitioned component pair (n_i, k_i) agrees with the parameters of a perfect code, a codeword of the perfect code indicates the node to which a resource copy is attached, and the resource diameter d_i is fixed (to 1 for the Hamming code, and to 3 for the Golay code); otherwise, the basic strategy is followed, and d_i equals $\lfloor (n_i - k_i + 1)/2 \rfloor$.

III. Multiple-Adjacency Resource Placement

Placing multiple copies of a resource in a way that every cube node is guaranteed to have access to one and only one resource copy within a resource diameter tends to suffer from considerable performance degradation after a copy is lost, because the resource diameter then increases and a longer response time results. In a large system, it might be desirable to have every node adjacent to multiple resource copies (rather than one single copy) so as to ensure good performance. In a 15-dimensional hypercube, for example, if every node is adjacent to one single copy of a resource (say, the I/O processor), then every resource copy is shared by 16 nodes [following the Hamming code $\Psi(15, 11)$ perfect allocation] and totally 2^{11} copies of the resource are needed. The throughput of such a large system may be compromised by insufficient I/O bandwidth, and more I/O processors should be incorporated into the system to overcome this potential I/O limitation. From the performance standpoint, it seems interesting and important to consider the placement of a resource in a way that every node is assured to get access to multiple resource copies within a specified number of hops. Such a resource placement guarantees that the resource diameter of the placement remains unchanged even in the absence of one resource copy.

In the following, we discuss the linear block code-based resource placement strategy, which ensures every cube node without the resource is adajcent to a given number of resource copies.

A. Perfect Placement

Definition 1: A hypercube is said to have a *j-adjacency perfect placement* if it is possible to place the resource copies in the hypercube such that 1) each node without the resource is adjacent to exactly j copies of the resource, and 2) the number of resource copies involved is minimum.

The necessary condition for hypercube Q_n to have a j-adjacency perfect placement is provided in the following lemma.

Lemma 1: Q_n has a j-adjacency perfect placement only if n is equal to $j(2^r - 1)$ for an integer r.

Proof: Each node in Q_n is adjacent to n nodes, one along each dimension. If the number of resource copies involved is minimized, then no resource copies will be placed at neighboring nodes. Assuming that there are y resource copies in total, the mean adjacency of a nonresource node is then given by $y \times n/(2^n - y)$, which equals j (since every nonresource node is adjacent to exactly j copies of the resource). The preceding result leads to $y = 2^n/(n/j + 1)$. Therefore, when a j-adjacency perfect placement exists, $(n/j + 1)$ should divide 2^n (as y is an integer), implying that $(n/j + 1)$ has to be a power of 2. As a result, $(n/j + 1)$ equals 2^r for some integer r, yielding $n = j(2^r - 1)$. □

According to Lemma 1, a 2-adjacency perfect placement exists in Q_n for $n = 2, 6, 14, 30$, and so on. A 2-adjacency perfect placement for Q_2 is shown in Fig. 3. Naturally, one would wonder if a j-adjacency perfect placement always exists in Q_n, for any $n = j(2^r - 1)$. In fact, the condition of $n = j(2^r - 1)$ is also sufficient for Q_n to have a j-adjacency perfect placement, and a systematic procedure is obtained next to arrive at such a placement.

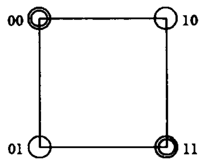

Fig. 3. A 2-adjacency perfect placement in Q_2. (Nodes with double circles have one resource copy each.)

A linear block code satisfying the property that each noncodeword is adjacent to exactly j codewords serves as the basis of our placement procedure. The codewords of this code specify the locations where the resource copies should be placed. Let this block code be referred to as a *j-adjacency perfect code,* then the parity-check matrix $\mathbf{H}$ of the j-adjacency perfect code for Q_n with $n = j(2^r - 1)$ is composed of j cascade copies of all the possible nonzero r-tuples. Since the parity-check matrix of Hamming code $\Psi(2^r - 1, 2^r - 1 - r)$ comprises (one copy of) all the possible nonzero r-tuples (totally, $2^r - 1$ columns, as described in Section II), the parity-check matrix $\mathbf{H}$ of the j-adjacency perfect code can be represented as

$$\mathbf{H} = [\mathbf{H}_a \ \mathbf{H}_a \cdots \mathbf{H}_a], \tag{4}$$

where $\mathbf{H}_a$ is the parity-check matrix of Hamming code $\Psi(2^r - 1, 2^r - 1 - r)$. With $\mathbf{H}$ obtained, we can derive the generator matrix of the desired code, $\mathbf{G}$, using the equality $\mathbf{G} \cdot \mathbf{H}^T = \mathbf{0}$. $\mathbf{H}$ is an $r \times n$ matrix, and $\mathbf{G}$ is an $(n - r) \times n$ matrix. In fact, the j-adjacency perfect code is the linear block code $\Psi(n, n - r)$.

To give an example, consider the case of a 2-adjacency perfect code for $Q_n, n = 6$. The parity-check matrix of the perfect code, according to (4), can be put in the form

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 \end{bmatrix}, \tag{5}$$

where the first three columns are all the possible nonzero 2-tuples [and constitute the parity-check matrix of Hamming code $\Psi(3, 1)$, see (2)], and the remaining three columns are another copy of all nonzero 2-tuples. The generator matrix of the code with parity-check matrix $\mathbf{H}$ is a $(6 - 2) \times 6$ matrix:

$$\mathbf{G} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Since all the codewords of this code are obtained from linear combinations of the four rows of $\mathbf{G}$, we have the locations 000000, 111000, 100100, 010010, 110001, 011100, 101010, 001001, 110110, 010101, 100011, 001110, 000111, 011011, 101101, and 111111, at which resource copies should be placed, as depicted in Fig. 4. This placement result satisfies that every node without the resource is adjacent to exactly two copies of the resource.

The linear code so constructed remains to be shown that each noncodeword is adjacent to exactly j codewords and no two codewords are adjacent. The following lemma reveals this fact, and its proof is given in the Appendix.

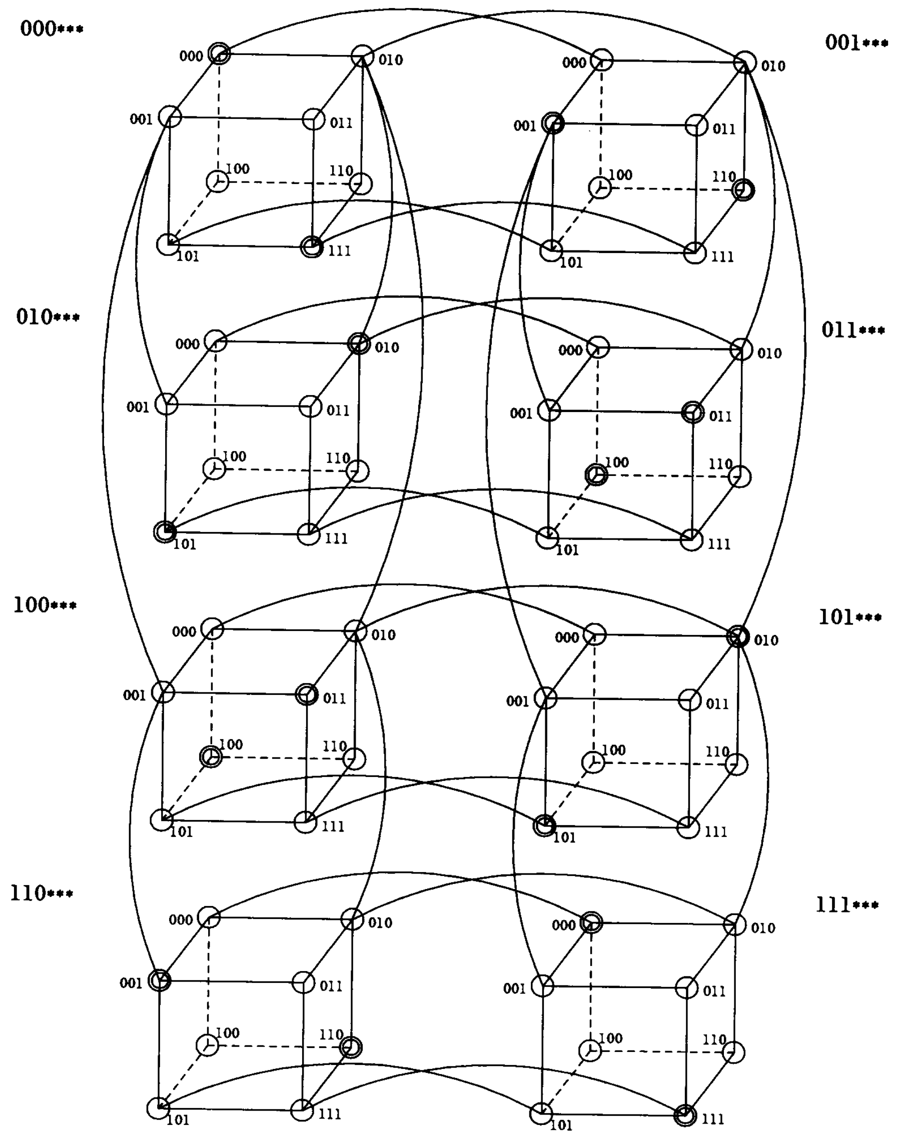

Fig. 4. A 2-adjacency perfect placement in Q_6. (Nodes with double circles have one resource copy each. Some links are omitted intentionally for clarity.)

Lemma 2: For any linear block code with parity-check matrix **H**, each noncodeword is adjacent to exactly j codewords and no two codewords are adjacent.

Since resource copies are never placed at two neighboring nodes and each node without the resource is adjacent to exactly j resource copies following a j-adjacency perfect code (from Lemma 2), the number of resource copies involved cannot be less and it is minimum. From this fact together with Lemma 1, we immediately arrive at the next theorem.

Theorem 1: A j-adjacency perfect placement exists in Q_n if and only if n is equal to $j(2^r - 1)$ for an integer r.

The number of resource copies needed to achieve a j-adjacency perfect placement in Q_n with $n = j(2^r - 1)$ is $\xi = 2^{n-r}$ [because the linear code dictating such a placement is $\Psi(n, n-r)$], and ξ cannot be reduced, leading to an optimal result. The ratio of the number of nodes with the resource to the number of nodes without the resource in Q_n equals $1\colon(2^r - 1)$.

A perfect placement, when it exists, is optimal, but it is present only for a few cube sizes. For most of the cases, perfect placements cannot be obtained. Fortunately, we may arrive systematically at a *quasiperfect* placement for any cube size.

B. Quasiperfect Placement

Definition 2: A hypercube is said to have a *j-adjacency quasiperfect placement* if it is possible to place the resource copies in the hypercube such that 1) each node without the resource is adjacent to at least j copies of the resouce, and 2) as few resource copies as possible are involved.

The number of resource copies to be placed is assumed to be a power of 2. In the subsequent discussion on the quasiperfect placement, we focus our attention on the cases where a specified adjacency is accomplished using ξ copies of the resource such that ξ is the smallest power of 2.

Again, a linear block code is to be derived whose codewords specify the locations where the resource copies are placed. The parity-check matrix $\mathbf{H}_q$ of a code for a j-adjacency quasiperfect placement can be constructed from the parity-check matrix $\mathbf{H}$ of a code for a j-adjacency perfect placement described earlier. Specifically, $\mathbf{H}_q$ of a code for a j-adjacency quasiperfect placement in $Q_n, n \neq j(2^r - 1)$, is derived from $\mathbf{H}$ of a code for a j -adjacency perfect placement in Q_{n_p} such that n_p is the largest integer less than n and satisfies $n_p = j(2^r - 1)$ for an integer r. After $\mathbf{H}$ is constructed according to (4), we get $\mathbf{H}_q$ as follows:

$$\mathbf{H}_q = [\mathbf{H} \quad \mathbf{C}_0 \quad \mathbf{C}_1 \quad \cdots \quad \mathbf{C}_{n-n_p-1}], \tag{6}$$

where $\mathbf{C}_i, 0 \leq i \leq n - n_p - 1$, is the ith$(mod\, n_p)$ column of $\mathbf{H}$, with column 0 being the leftmost column. Because $\mathbf{H}$ is an $r \times n_p$ matrix, $\mathbf{H}_q$ is an $r \times n$ matrix. The generator matrix $\mathbf{G}_q$ corresponding to $\mathbf{H}_q$ can be solved from the relationship $\mathbf{G}_q \cdot \mathbf{H}_q^T = \mathbf{0}$, giving rise to an $(n - r) \times n$ matrix. The code for a quasiperfect placement in Q_n with $n \neq j(2^r - 1)$ is again $\Psi(n, n - r)$, which contains 2^{n-r} codewords.

As an example, let us derive $\mathbf{H}_q$ for a 2-adjacency quasiperfect placement in Q_7. In this case, $n_p = 6$, and $\mathbf{H}$ for Q_6 is given in (5), leading to

$$\mathbf{H}_q = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 0 \end{bmatrix},$$

where the last column is a repetition of the first column. Its generator matrix is

$$\mathbf{G}_q = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

This code includes 32 codewords, which determine the locations where resource copies are placed in Q_7. Among those 96 cube nodes without involving the resource, 64 of them are adjacent to two resource copies each, and the rest are adjacent to three resource copies each. Interestingly, the ratio of the number of nodes with the resource to the number of nodes without the resource equals 1:3, which is the same as the ratio of the 2-adjacency perfect placement in Q_6 discussed earlier. This is because the code for Q_n is always $\Psi(n, n - r)$, no matter whether n equals $j(2^r - 1)$ or not.

It is to be noted that the placement so constructed for Q_7 requires the same number of resource copies (32 copies) as does that obtained by Reddy [4] (where the resource is the I/O processor) using a somewhat complicated enumeration method or an overlapping method. The two prior methods, however, are not generally applicable as our systematic technique in two aspects: 1) they can apply to Q_n with n limited only to $2^r - 1$ for an integer r, and 2) they become more complex as the degree of resource adjacency increases. In contrast, our technique can be applied universally to obtain a placement with any degree of adjacency in any sized hypercube, while keeping the number of resource copies involved ξ (which is a power of 2) minimum.

In the following, we show that a j-adjacency quasiperfect placement in Q_n according to the linear code $\Psi(n, n - r)$ with the parity-check matrix given by (6) requires the fewest resource copies. Such a placement needs 2^{n-r} resource copies, and r is the largest integer satisfying $j(2^r - 1) < n$. Since r is the largest integer such that $j(2^r - 1) < n$ and $j(2^r - 1)$ is not equal to n, we have $n < j(2^{r+1} - 1)$, which implies $n/(2^{r+1} - 1) < j$. If the number of resource copies is 2^{n-r-1}, the average adjacency for nodes without the resource equals

$$(2^{n-r-1} \times n)/(2^n - 2^{n-r-1}) = n/(2^{r+1} - 1) < j,$$

indicating that there is at least one node without the resource that would be adjacent to less than j resource copies. As a result, the minimum number of resource copies required to achieve j-adjacency quasiperfect placement is $\xi = 2^{n-r}$ (because ξ is restricted to a power of 2). This means that our quasiperfect placement uses the fewest possible resource copies.

Since the parity-check matrix $\mathbf{H}_q$ of the linear block code $\Psi(n, n - r)$ given in (6) contains $\mathbf{H}$, it is straightforward to prove that each noncodeword is adjacent to at least j codewords and the minimum distance of the code is 2, by following the same arguments as provided in the proof of Lemma 2. A placement in Q_n obtained according to the preceding strategy thus meets the j-adjacency quasiperfect requirement using the fewest possible resource copies, as indicated by the next theorem.

Theorem 2: For any linear block code with parity-check matrix $\mathbf{H}_q$, each noncodeword is adjacent to at least j codewords with the total number of codewords kept minimum.

The preceding treatment of quasiperfect placements assumes that the number of resource copies involved is a power of 2. If a placement is allowed to take any number of resource copies, it becomes extremely difficult, if not impossible, to derive the most efficient quasiperfect placement for any given n and j. A necessary condition on the minimum number of resource copies (not necessarily a power of 2), denoted by $c_{\min}$, can be obtained easily according to the inequality

$$(c_{\min} \times n)/(2^n - c_{\min}) \geq j,$$

yielding $c_{\min} \geq (j \times 2^n)/(j + n)$. The sufficient condition on $c_{\min}$ is unknown, and how to place a resource that takes $c_{\min}$ copies systematically is still an open problem. Our solution for the quasiperfect placement requires $\Upsilon((j \times 2^n)/(j + n))$ resource copies, where $\Upsilon(x)$ is the smallest power of 2 integer greater than or equal to x. The gap between $\Upsilon((j \times 2^n)/(j + n))$ and $(j \times 2^n)/(j + n)$ is clearly dependent on n and j.

IV. Generalized Resource Placement

This section investigates the generalized resource placement where every node in Q_n can reach at least j copies, $j \geq 2$, of a resource in no more than h hops, $h \geq 1$. A placement of this sort ensures that the resource diameter remains unchanged

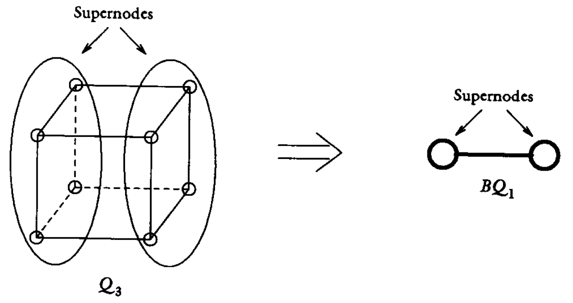

Fig. 5. One-dimensional "bulky hypercube," in which a supernode is Q_2.

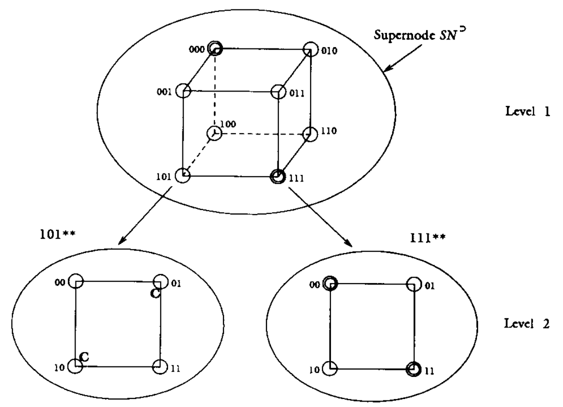

Fig. 6. Following a Hamming code strategy and a basic strategy to place resource copies respectively in levels 1 and 2 of the partitioned $SN^{\supset}$. (A node with resource is denoted by a double cycle. A critical node has the symbol "C " next to it.)

even after $(j-1)$ resource copies become unavailable, reducing potential performance degradation. Our goal is to find such a placement that uses as few resource copies as possible. This is accomplished by making use of the j-adjacency perfect code developed earlier.

A. Approach

We may envision Q_n as a "bulky hypercube" with dimension α (denoted by BQ_α) such that every node of the bulky hypercube, called a *supernode,* is a subcube with dimension $n-\alpha$ and every link between two neighboring supernodes in BQ_α represents $2^{n-\alpha}$ parallel links between the corresponding nodes in the two supernodes. For example, Q_3 shown in Fig. 1 can be envisioned as BQ_1, whose every node is a two-dimensional subcube and every link represents four parallel links, as depicted in Fig. 5. To obtain a generalized resource placement, we apply the j-adjacency perfect code to BQ_α. For the j-adjacency perfect code to exist, α should be equal to $j(2^r-1)$ for some integer r (from Theorem 1).

A certain number of resource copies are placed inside each BQ_α supernode whose address is a codeword of the chosen j-adjacency perfect code. Let the supernode involving resource copies be denoted by $SN^{\supset}$, and the supernode without resource copies be denoted by $SN^{\neg}$. Since perfect code $\Psi(\alpha, \alpha-r)$ is used for placing the resource copies in BQ_α, totally there are $2^{\alpha-r}$ $SN^{\supset}$'s. The way of placing resource copies is identical in each supernode $SN^{\supset}$, and different placement techniques require different numbers of resource copies. We choose to place resource copies in every $SN^{\supset}$ in a hierarchical fashion, as explained subsequently.

The supernode $SN^{\supset}$ (which is a cube with dimension $n-\alpha$) is viewed as composed of 2^{δ_1} subcubes with dimension $n-\alpha-\delta_1$, and each of the subcubes is a supernode with respect to $SN^{\supset}$, called SN_1. Similarly, an SN_1 in turn is considered to be composed of 2^{δ_2} subcubes with dimension $n-\alpha-\delta_1-\delta_2$, and each of the subcubes is a supernode SN_2 with respect to this SN_1, and so on. More specifically, $SN^{\supset}$ is divided into l levels, $l \geq 1$, by partitioning the number $n-\alpha$ into l positive integers, $\delta_1, \delta_2, \cdots, \delta_l$, such that $n-\alpha = \delta_1 + \delta_2 + \cdots + \delta_l$. At any level $y, 1 < y \leq l$, there are 2^{δ_y} supernodes SN_y's within a supernode SN_{y-1}. If each supernode SN_y is treated as an "entity" (i.e., a node), these 2^{δ_y} supernodes form a δ_y-dimensional cube at level y. Resource copies are placed in the δ_y-dimensional cube according to a Hamming code strategy or a basic strategy proposed by Chiu and Raghavendra [5] (explained in Section II). To allow the use of a Hamming code strategy at level y, the partition of $n-\alpha$ has to satisfy $\delta_y = 2^r - 1$, for an integer r. Fig. 6 illustrates an example where $SN^{\supset}$ is divided into two levels, with a Hamming code strategy and a basic strategy followed respectively to place resource copies in the first and the second levels.

B. Analysis

The resource diameter is 1 (meaning that each node can have access to a resource copy in no more than one hop) at the level to which a Hamming code strategy is applied. Conversely, if a basic strategy is applied to level y, the resource diameter at that level is not fixed and is dependent on the number of locations at which the resource copies are placed, namely, the resource diameter = $\lfloor(\delta_y - c_y + 1)/2\rfloor$, where the total number of locations involving resource copies is 2^{c_y}. Let d_y represent the resource diameter at level y, then a partition on $n-\alpha$ is said to be *feasible* if $d_1 + d_2 + \cdots + d_l$ is equal to $h-1$. Only the feasible partition is of our concern (because it tends to involve fewer resource copies than other partitions).

In the following, we examine the nodes in $SN^{\neg}$'s and the nodes in $SN^{\supset}$'s separately to see if they meet the requirement that each of them gains access to at least j resource copies in no more than h hops, for a feasible partition in the supernode $SN^{\supset}$. First, let us consider a node inside any supernode $SN^{\neg}$. The node can reach j $SN^{\supset}$'s in exactly one hop (as a j-adjacency perfect code is used). Since a feasible partition is followed in every supernode $SN^{\supset}$, every node inside a supernode $SN^{\supset}$ can get access to at least one resource copy in no more than $h-1$ hops (as at least one copy is placed in an $SN^{\supset}$). As a result, every node inside $SN^{\neg}$ can reach at least j resource copies in no more than h hops.

Next, a node inside any supernode $SN^{\supset}$ is considered. The resource diameter of $SN^{\supset}$ is $h-1$ (for only the feasible partition of $SN^{\supset}$ is concerned). There is at least one node $X \in SN^{\supset}$ such that X takes $h-1$ hops to reach a resource copy inside $SN^{\supset}$. A node of this type is referred to as a *critical node.* In Fig. 6, for example, critical nodes are marked and each of them takes two $(= h - 1)$ hops to reach one resource copy. A critical node X in one supernode $SN^{\supset}$ cannot reach any resource copy outside this supernode in h hops or less because 1) resource copies are placed in every supernode $SN^{\supset}$ in the same way, and 2) X takes $h-1$ hops to reach a resource copy inside this supernode, which is not adjacent to any other supernode $SN^{\supset}$ (recall that supernode $SN^{\supset}$ addresses are the codewords of a j-adjacency perfect code, whose minimum distance is exactly 2). As a result, to evaluate the number of resource copies reachable from X in no more than h hops, we need to consider the resource copies only *inside* the supernode $SN^{\supset}$ where X resides. Conversely, a noncritical node may reach a resource copy inside another supernode $SN^{\supset}$ in h hops or less. For example, the noncritical node (10100) in Fig. 6 can reach the resource copy at node (11100) inside another $SN^{\supset}$ (whose distance is 2 from the supernode $SN^{\supset}$ shown in Fig. 6; such a supernode always exists following the property of the j-adjacency perfect code) in three $(= h)$ hops. It is readily shown that a noncritical node O can get access to at least j resource copies in h hops or less, as follows. Node O may reach at least one resource copy (say, a node O') inside $SN^{\supset}$ in no more than $h - 2$ hops, and from node $O', j - 1$ other nodes with resource (i.e., the corresponding nodes in the $(j - 1)$ $SN^{\supset}$'s which are adjacent to the supernode $SN^{\supset}$ where nodes O and O' reside) can be reached in two hops (as a result of the j-adjacency perfect code). Therefore, O can have access to at least $1 + (j - 1)$ resource copies in no more than $(h - 2) + 2$ hops. Noncritical nodes thus meet our placement requirement. What remains to be considered is whether critical nodes satisfy the placement requirement. The number of resource copies reachable from a critical node in no more than h hops will be derived subsequently. To meet our placement goal, we search for the feasible partition that requires a minimum number of resource copies to satisfy that a critical node reaches no less than j resource copies in no more than h hops. Such a partition could lead to a desirable placement.

As mentioned earlier, a partition produces δ_y-dimensional cubes at level y. Suppose that each Q_{δ_y} has 2^{c_y} locations placed with the resource following either a Hamming code or a basic strategy, giving rise to the resource diameter of d_y. A critical node $X \in$ supernode $SN^{\supset}$ takes $d_1, d_2, \cdots,$ and d_l hops (which amount to $h - 1$ hops), respectively, in level 1, level 2, $\cdots$, and level l to reach a resource copy. Let $SN_y(X)$ denote the node SN_y, which contains critical node X in Q_{δ_y}. In Fig. 6, for example, $SN_1(X = 10101)$ represents node (101) in level 1. Let $f_y(X)$ and $g_y(X)$ be the numbers of such nodes SN_y's $\in Q_{\delta_y}$ that involve resouce copies and are reachable from $SN_y(X)$, for a critical node X, in exactly d_y hops and $d_y + 1$ hops, respectively. It is observed that $f_y(X)$ and $g_y(X)$ are identical for any critical node $X \in Q_{\delta_y}$ (because either a Hamming code or a basic strategy is followed in level y). For simplicity, $f_y(X)$ and $g_y(X)$ are denoted by f_y and g_y, respectively. The expressions for f_y and g_y are provided in the following lemmas, whose proofs can be found in the Appendix. These results are essential for us to determine if a feasible partition satisfies our placement goal.

Lemma 3: If a placement in Q_{δ_y} is obtained following a Hamming code strategy, $f_y = 1$ and $g_y = (\delta_y - 1)/2$.

Lemma 4: When a basic strategy is followed in Q_{δ_y} to have 2^{c_y} locations placed with the resource, then $f_y = 1$ and $g_y = c_y$, if $\delta_y - c_y + 1$ is odd; otherwise, $f_y = 2$ and $g_y = 2(c_y - 1)$.

With f_y and g_y given in Lemma 3 or 4 for each level $y, 1 \leq y \leq l$, we immediately get the next theorem, which reveals the number of resource copies reachable from a critical node in $SN^{\supset}$, for a given placement in the supernode $SN^{\supset}$.

Theorem 3: Let f and g be the numbers of such nodes $(\in SN^{\supset})$ that involve resource copies and are reachable from any critical node X in exactly $h - 1\, (= \Sigma_{y=1}^{l} d_y)$ hops and in exactly h hops, respectively, then

$$f = \prod_{y=1}^{l} f_y \quad \text{and} \quad g = \sum_{y=1}^{l} g_y \left(\prod_{i=1; i \neq y}^{l} f_i \right).$$

The expression for f is clear because 1) X has to take d_y hops in level y, for all $1 \leq y \leq l$, to reach a resource copy in exactly $h - 1$ hops, and 2) in level y, the number of nodes that involve resource copies and are reachable from X in exactly d_y hops is f_y. The expression for g results directly from the fact that X must take $d_y + 1$ hops in one and only one level y, and takes d_i hops in level i for the remaining levels, to reach a resource copy in exactly h hops. In Fig. 6, for instance, $f_1 = 1, g_1 = 1, f_2 = 2,$ and $g_2 = 0$, yielding $f = 2$ and $g = 2$. The two $(= f)$ resource copies reachable from a critical node, say (10101), in two hops are those at nodes (11100) and (11111). Likewise, the two $(= g)$ resource copies reachable from the same critical node in three hops are those at nodes (00000) and (00011).

C. Algorithm

To meet our placement requirement, any feasible partition of $n-\alpha$ that results in $f+g$ less than j is discarded. The desirable placement is obtained by searching possible α's and partitions of $n-\alpha$ for the result that requires the least amount of resource copies. Formally, our generalized placement strategy is given as follows.

1) For a given set of $n, j,$ and h, select a j-adjacency perfect code to determine the locations of supernodes $SN^{\supset}$'s.
2) A supernode $SN^{\supset}$ is partitioned into several levels, with resource copies placed in each subcube formed at a level according to either a Hamming code or a basic strategy. Only feasible partitions are considered.
3) Find the partition using the fewest resource copies.
4) Repeat 1)–3) above for every possible j-adjacency perfect code to search for the desirable placement.

Since the perfect codes (i.e., the j-adjacency perfect codes and the Hamming codes) exist only in a few instances for practical values of n and j. the number of possible placements is quite limited and the search is fairly efficient. The desirable placement is not necessarily unique. Consider the case of $n = 16, j = 6$, and $h = 3$ as an example. The only available 6-adjacency perfect code is $\Psi(6,5)$ in this case (as the possible $\alpha = 6(2^r - 1), \alpha \leq n$, is solely 6), and each supernode $SN^{\supset}$ is a ten-dimensional cube. We have the following possible placements in every $SN^{\supset}$.

1) Hamming code $\Psi(3,1)$ for each subcube at the first level, and Hamming code $\Psi(7,4)$ for each subcube at the second level.
2) Hamming code $\Psi(3,1)$ for each subcube at the first level, and a basic strategy using 2^{γ} resource copies for each subcube Q_7 at the second level, where γ satisfies $\lfloor(7-\gamma+1)/2\rfloor = 2-1$ (since only feasible partitions are of our concern and d_1 is 1).
3) Hamming code $\Psi(7,4)$ for each subcube at the first level, and a basic strategy using 2^{γ} resource copies for each subcube Q_3 at the second level, where γ satisfies $\lfloor(3-\gamma+1)/2\rfloor = 2-1$.
4) A basic strategy using 2^{γ} resource copies for each Q_{10}, where γ satisfies $\lfloor(10-\gamma+1)/2\rfloor = 2$.

It is clear that placement 1) needs 2^{1+4} resource copies per $SN^{\supset}$ and yields $f+g = 1+(1+3)$; placement 2) needs $2^{1+\gamma}$ resource copies per $SN^{\supset}$ with the smallest $\gamma = 5$; placement 3) needs $2^{4+\gamma}$ resource copies per $SN^{\supset}$ with the smallest $\gamma = 1$; and placement 4) needs 2^{γ} resource copies per $SN^{\supset}$ with the smallest $\gamma = 6$. Placement 1) should be discarded, as $f+g = 5 < j$. Placement 2) for $\gamma = 5$ is a desirable placement, as it takes 2^6 resource copies per $SN^{\supset}$ and leads to $f+g = 1+(1+5) > j$. In this case, placement 3) for $\gamma = 2$ and placement 4) for $\gamma = 6$ are also desirable placements.

In general, a Hamming code strategy utilizes resource copies more efficiently than a basic strategy, so a placement with as many levels following the Hamming code strategy as possible tends to require fewer resource copies; this type of placement is desirable, provided that it yields $f+g$ no less than j. It should be noted that any placement obtained from interchanging two levels of a desirable placement is also a desirable one. As an instance, placing resource copies in every $SN^{\supset}$ of the preceding example as follows is a desirable placement: A basic strategy using 2^5 resource copies for each subcube Q_7 at the first level and Hamming code $\Psi(3,1)$ for each subcube at the second level, as it is attained by interchanging the two levels of placement 2.

V. Conclusion

We have investigated the efficient placement of resource copies in a hypercube such that the resource diameter remains unchanged even after resource copies become unavilable, lowering access contention to ensure good performance. The use of j-adjacency perfect codes enables the placement of resource copies in a simple and uniform way that guarantees every cube node without the resource to be adjacent to exactly j resource copies. Multiple-adjacency perfect placements exist in hypercubes with a few selected sizes. For other cube sizes where perfect placements are unattainable, j-adjacency quasiperfect codes are developed to help arrive at the resource placement satisfying that every node without the resource is adjacent to at least j resource copies, using as few resource copies as possible.

The j-adjacency perfect code is applied to a generalized resource placement problem where every cube node gets access to at least j resource copies in no more than a specified number of hops. This generalized placement problem is solved by searching for the most efficient placement that meets our placement requirement. The resource placements we obtained appear interesting and are particularly useful for a large-scale hypercube computer to improve its performance and reliability.

Appendix

Proof of Lemma 2: Let $\mathbf{e}_x$ denote an n-tuple with all positions being 0 except the xth position (note that the first position is leftmost). For example, $\mathbf{e}_2 = [0100\cdots00]$. If codeword $\mathbf{u}_1$ is adjacent to a noncodeword $\mathbf{v}$, then, there exists $\mathbf{e}_{i_1}$ (where i_1 is the position of the bit differing between $\mathbf{u}_1$ and $\mathbf{v}$) such that

$$\mathbf{u}_1 + \mathbf{e}_{i_1} = \mathbf{v}$$

(recall that the addition is in mod 2). Since $\mathbf{u}_1$ is a codeword, satisfying

$$\mathbf{u}_1 \cdot \mathbf{H}^T = \mathbf{0},$$

we have

$$\begin{aligned}\mathbf{v} \cdot \mathbf{H}^T &= (\mathbf{u}_1 + \mathbf{e}_{i_1}) \cdot \mathbf{H}^T = \mathbf{u}_1 \cdot \mathbf{H}^T + \mathbf{e}_{i_1} \cdot \mathbf{H}^T \\ &= \mathbf{e}_{i_1} \cdot \mathbf{H}^T.\end{aligned}$$

It is easy to show that $\mathbf{e}_{i_1} \cdot \mathbf{H}^T$ is nothing but the i_1th column of $\mathbf{H}$. According to the method for constructing $\mathbf{H}$, exactly j columns in $\mathbf{H}$ are identical. Suppose that the i_1th, i_2th, $\cdots$, and i_jth columns of $\mathbf{H}$ are identical columns, then $\mathbf{e}_{i_1} \cdot \mathbf{H}^T, \mathbf{e}_{i_2} \cdot \mathbf{H}^T, \cdots$, and $\mathbf{e}_{i_j} \cdot \mathbf{H}^T$ are all equal, since they are nothing but the i_1th column, the i_2th column, $\cdots$, and the i_jth column of $\mathbf{H}$, respectively. Thus, we have

$$\mathbf{e}_{i_1} \cdot \mathbf{H}^T = \mathbf{e}_{i_2} \cdot \mathbf{H}^T = \cdots = \mathbf{e}_{i_j} \cdot \mathbf{H}^T = \mathbf{v} \cdot \mathbf{H}^T.$$

Adding $\mathbf{v} \cdot \mathbf{H}^T$ to each preceding expression, we get

$$\begin{aligned}(\mathbf{v} + \mathbf{e}_{i_1}) \cdot \mathbf{H}^T &= (\mathbf{v} + \mathbf{e}_{i_2}) \cdot \mathbf{H}^T = \cdots = (\mathbf{v} + \mathbf{e}_{i_j}) \cdot \mathbf{H}^T \\ &= (\mathbf{v} + \mathbf{v}) \cdot \mathbf{H}^T,\end{aligned}$$

which is equal to $\mathbf{0}$, as $(\mathbf{v} + \mathbf{v})$ yields an all-zero n-tuple. From the preceding equation, it is clear that there are at least j codewords, namely, $\mathbf{v}+\mathbf{e}_{i_1}, \mathbf{v}+\mathbf{e}_{i_2}, \cdots$, and $\mathbf{v}+\mathbf{e}_{i_j}$, which are adjacent to any noncodeword $\mathbf{v}$.

Let us assume that a noncodeword $\mathbf{v}$ is adjacent to $j+1$ or more codewords. That is, there exist $j+1$ or more codewords $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_w, w > j$, such that

$$\begin{aligned}\mathbf{v} &= \mathbf{u}_1 + \mathbf{e}_{i_1} = \mathbf{u}_2 + \mathbf{e}_{i_2} = \cdots \\ &= \mathbf{u}_w + \mathbf{e}_{i_w}, \qquad i_1 \neq i_2 \neq \cdots \neq i_w.\end{aligned} \tag{A1}$$

Since $\mathbf{u}_1, \mathbf{u}_2, \cdots$, and $\mathbf{u}_w$ are all codewords, they satisfy

$$\mathbf{u}_1 \cdot \mathbf{H}^T = \mathbf{u}_2 \cdot \mathbf{H}^T = \cdots = \mathbf{u}_w \cdot \mathbf{H}^T = \mathbf{0}.$$

From the following result

$$(\mathbf{u}_1 + \mathbf{e}_{i_1}) \cdot \mathbf{H}^T = \mathbf{u}_1 \cdot \mathbf{H}^T + \mathbf{e}_{i_1} \cdot \mathbf{H}^T = \mathbf{e}_{i_1} \cdot \mathbf{H}^T$$

and (A1), we have

$$\mathbf{e}_{i_1} \cdot \mathbf{H}^T = \mathbf{e}_{i_2} \cdot \mathbf{H}^T = \cdots = \mathbf{e}_{i_w} \cdot \mathbf{H}^T.$$

This equation implies that there are $w > j$ equivalent columns in $\mathbf{H}$ (as $\mathbf{e}_{i_m} \cdot \mathbf{H}^T$ is the i_mth column of $\mathbf{H}$, for all $1 \leq m \leq w$). According to the construction of $\mathbf{H}$, however, there are only j equivalent columns in $\mathbf{H}$. This contradiction results from our assumption that a noncodeword is adjacent to $j+1$ or more codewords.

The j-adjacency perfect code is a linear block code, and it can be easily shown according to Property 1 that the minimum distance of a j-adjacency perfect code, $j \geq 2$, constructed from (4) is exactly 2, because 1) there are two columns alike in $\mathbf{H}$ and they sum to $\mathbf{0}$, and 2) the smallest number of columns that sum to $\mathbf{0}$ is two, as the columns of $\mathbf{H}$ are all nonzero. The lemma thus follows. □

Proof of Lemma 3: The resource diameter d_y is 1, and every noncodeword is adjacent to exactly one codeword. Thus, every noncodeword can reach exactly one codeword in $d_y = 1$ hop. Q_{δ_y} involves 2^{δ_y} SN_y's, whose locations equal to codewords have resource copies placed. Since a critical node can never be in an SN_y involving resource copies and must belong to an SN_y situated at a noncodeword location, we have $f_y = 1$.

Let the Hamming code used be $\Psi(\delta_y, k)$. Recall that the parity-check matrix $\mathbf{H}$ of $\Psi(\delta_y, k)$ consists of all nonzero $(\delta_y - k)$-tuples as its columns. If $\mathbf{e}_x$ is an $(\delta_y - k)$-tuple with all positions 0 except the xth position, then $\mathbf{e}_x \cdot \mathbf{H}^T$ is equal to the xth column of $\mathbf{H}$. Consider a noncodeword $\mathbf{v}$. Suppose that $\mathbf{v}$ can reach a codeword $\mathbf{u}$ in one hop along dimension i, that is, $\mathbf{v} + \mathbf{e}_i = \mathbf{u}$ and $\mathbf{u} \cdot \mathbf{H}^T = \mathbf{0}$.

There are δ_y total columns in $\mathbf{H}$, and any given column, say column i, is the sum (in mod 2) of two other columns. In $\mathbf{H}$ given by (3), for example, the first column is the sum of the second and third columns. In addition, the first column is the sum of the fourth and seventh columns, and also the sum of the fifth and sixth columns. In general, with respect to any given column, say column i, in $\mathbf{H}$, all the remaining $\delta_y - 1$ columns can be grouped into $(\delta_y - 1)/2$ pairs, each of whose sum is equal to column i, namely,

$$\begin{aligned}\mathbf{e}_i \cdot \mathbf{H}^T &= \mathbf{e}_{z_1^1} \cdot \mathbf{H}^T + \mathbf{e}_{z_1^2} \cdot \mathbf{H}^T = \mathbf{e}_{z_2^1} \cdot \mathbf{H}^T \\ &\quad + \mathbf{e}_{z_2^2} \cdot \mathbf{H}^T = \cdots = \mathbf{e}_{z^1_{(\delta_y-1)/2}} \cdot \mathbf{H}^T \\ &\quad + \mathbf{e}_{z^2_{(\delta_y-1)/2}} \cdot \mathbf{H}^T,\end{aligned}$$

where $\mathbf{e}_{z_\beta^1} \cdot \mathbf{H}^T$ and $\mathbf{e}_{z_\beta^2} \cdot \mathbf{H}^T, 1 \leq \beta \leq (\delta_y - 1)/2$, are pair columns. Adding $\mathbf{v} \cdot \mathbf{H}^T$ to each of the preceding expressions, we get

$$\begin{aligned}(\mathbf{v} + \mathbf{e}_i) \cdot \mathbf{H}^T &= (\mathbf{v} + \mathbf{e}_{z_1^1} + \mathbf{e}_{z_1^2}) \cdot \mathbf{H}^T \\ &= (\mathbf{v} + \mathbf{e}_{z_2^1} + \mathbf{e}_{z_2^2}) \cdot \mathbf{H}^T = \cdots \\ &= (\mathbf{v} + \mathbf{e}_{z^1_{(\delta_y-1)/2}} + \mathbf{e}_{z^2_{(\delta_y-1)/2}}) \cdot \mathbf{H}^T = \mathbf{0},\end{aligned}$$

because of $(\mathbf{v} + \mathbf{e}_i) = \mathbf{u}$, a codeword. This indicates that any noncodeword $\mathbf{v}$ can reach at least $(\delta_y - 1)/2$ codewords in two hops.

Suppose that there exists another codeword, say $(\mathbf{v} + \mathbf{e}_{w^1} + \mathbf{e}_{w^2})$, that can be reached from $\mathbf{v}$ in two hops. Since the set of $\{\mathbf{e}_i, \mathbf{e}_{z_1^1}, \mathbf{e}_{z_1^2}, \mathbf{e}_{z_2^1}, \mathbf{e}_{z_2^2}, \cdots, \mathbf{e}_{z^1_{(\delta_y-1)/2}}, \mathbf{e}_{z^2_{(\delta_y-1)/2}}\}$ contains all possible $\mathbf{e}_x$'s, $\mathbf{e}_{w^1}$ should be one of them. Without loss of generality, we assume that $\mathbf{e}_{w^1} = \mathbf{e}_{z_1^1}$. The codeword $(\mathbf{v} + \mathbf{e}_{w^1} + \mathbf{e}_{w^2})$ is thus equal to $(\mathbf{v} + \mathbf{e}_{z_1^1} + \mathbf{e}_{w^2})$, which is two hops away from the codeword $(\mathbf{v} + \mathbf{e}_{z_1^1} + \mathbf{e}_{z_1^2})$, a contradiction (since the distance between two codewords of a Hamming code is no less than three). A noncodeword thereby can reach exactly $(\delta_y - 1)/2$ codewords in $d_y + 1 = 2$ hops. This completes the proof of the lemma. □

Proof of Lemma 4: The basic strategy treats Q_{δ_y} as a collection of $2^{c_y - 1}$ $(\delta_y - c_y + 1)$-dimensional subcubes, and places resource copies at two opposite corners of every subcube, giving rise to the resource diameter of $d_y = \lfloor(\delta_y - c_y + 1)/2\rfloor$. A critical node, say X, must belong to an SN_y that is d_y hops away from an SN_y with resource in Q_{δ_y}. Consider any $(\delta_y - c_y + 1)$-dimensional subcube (where each node is an SN_y). If $\delta_y - c_y + 1$ is odd (see Fig. 2 for an example of $\delta_y - c_y + 1 = 3$), an SN_y involving X, denoted by $SN_y(X)$, in the subcube can reach one of the two SN_y's involving the resource in d_y hops, and reach the other SN_y (of the two) involving the resource in $d_y + 1$ hops (since each subcube contains two SN_y's with resource). Furthermore, any SN_y with resource is adjacent to exactly $c_y - 1$ SN_y's with resource at the corresponding locations in other subcubes, which can be reached in d_y hops plus on hop. $SN_y(X)$ thus can reach one SN_y with resource in d_y hops and $1 + (c_y - 1)$ SN_y's with resource in $d_y + 1$ hops. As a result, $f_y = 1$ and $g_y = c_y$ when $\delta_y - c_y + 1$ is odd.

Next, consider a $(\delta_y - c_y + 1)$-dimensional subcube, with $(\delta_y - c_y + 1)$ being even. An $SN_y(X)$ can reach both SN_y's with resource inside the subcube in exactly d_y hops. Since every SN_y with resource is adjacent to exactly $(c_y - 1)$ SN_y's with resource in other subcubes, $SN_y(X)$ can have access to $2(c_y - 1)$ SN_y's with resource in $d_y + 1$ hops. The lemma is thus proved. □

References

[1] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. 37, pp. 867–872, July 1988.

[2] D. A. Reed and R. M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing.* Cambridge, MA: MIT Press, 1987.

[3] M. Livingston and Q. F. Stout, "Distributing resources in hypercube computers," in *Proc. 3rd Conf. Hypercube Concurrent Computers and Applications*, Jan. 1988, pp. 222–231.

[4] A. L. N. Reddy, "Parallel input/output architectures for multiprocessors," Ph.D. dissertation, Dept. of Electrical and Computer Engineering, Univ. of Illinois, Urbana, 1990.

[5] G.-M. Chiu and C. S. Raghavendra, "Resource allocation in hypercube systems," in *Proc. 5th Distributed Memory Computing Conf.*, Apr. 1990, pp. 894–902.

[6] R. E. Blahut, *Theory and Practice of Error Control Codes.* Reading, MA: Addison-Wesley, 1983.

[7] S. Lin and D. J. Costello, *Error Control Coding Fundamentals and Applications.* Englewood Cliffs, NJ: Prentice-Hall, 1983.

[8] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a hypercube in the presence of faults," in *Proc. 19th Annu. ACM Symp. Theory of Computing*, May 1987, pp. 274–284.

Hsing-Lung Chen (S'79–M'88) received the B.S. and M.S. degrees in computer science from National Chiao Tung University, Taiwan, in 1978 and 1980, respectively, and the Ph.D. degree in electrical and computer engineering from the Illinois Institute of Technology, Chicago, in 1987.

From 1987 to 1989 he was an Assistant Professor in the Department of Mathematics and Computer Science at Clarkson University, Potsdam, NY. In 1989, he joined the Department of Electronic Engineering at National Taiwan Institute of Technology, Taipei, Taiwan, where he is currently an Associate Professor. His research interests include parallel processing, distributed computing, and database systems.

Dr. Chen is a member of the Association for Computing Machinery.

Nian-Feng Tzeng (S'85–M'86–SM'92) received the B.S. degree in computer science from National Chiao Tung University, Taiwan, the M.S. degree in electrical engineering from National Taiwan University, Taiwan, and the Ph.D. degree in computer science from the University of Illinois at Urbana–Champaign in 1978, 1980, and 1986, respectively.

He is currently an Associate Professor in the Center for Advanced Computer Studies at the University of Southwestern Louisiana, Lafayette, where he has been on the faculty since June 1987. From 1986 to 1987, he was a Member of the Technical Staff, AT&T Bell Laboratories, Columbus, OH. His current research interest is in the areas of parallel and distributed processing, fault-tolerant computing, and high-performance computer systems. He is an Associate Editor of the IEEE TRANSACTIONS ON COMPUTERS and will be Co-Guest Editor of a special issue of the *Journal of Parallel and Distributed Computing*, 1995.

Dr. Tzeng is a member of Tau Beta Pi and the Association for Computing Machinery, and the recipient of the outstanding paper award of the 10th International Conference on Distributed Computing Systems, May 1990.