Resource Placement with Multiple Adjacency Constraints in k-ary n-Cubes

Parameswaran Ramanathan and Suresh Chalasani

***Abstract*— The problem of placing resources in a k-ary n-cube ($k > 2$) is considered in this paper. For a given $j \geq 1$, resources are placed such that each nonresource node is adjacent to j resource nodes. We first prove that perfect j-adjacency placements are impossible in k-ary n-cubes if $n < j < 2n$. Then, we show that a perfect j-adjacency placement is possible in k-ary n-cubes when one of the following two conditions is satisfied: 1) if and only if j equals $2n$ and k is even, or 2) if $1 \leq j \leq n$ and there exist integers q and r such that q divides k and $q^r - 1 = 2n/j$. In each case, we describe an algorithm to obtain perfect j-adjacency placements. We also show that these algorithms can be extended under certain conditions to place j distinct types of resources in a such way that each nonresource node is adjacent to a resource node of each type. For the cases when perfect j-adjacency placements are not possible, we consider approximate j-adjacency placements. We show that the number of copies of resources required in this case either approaches a theoretical lower bound on the number of copies required for any j-adjacency placement or is within a constant factor of the theoretical lower bound for large k.**

***Index Terms*— Resource allocation, multiprocessors, hypercubes, mesh connected computers, interconnection network, fault-tolerance.**

I. Introduction

RESOURCES in a multiprocessor system can be of several different types: hardware units like disks, printers, and I/O devices, and software units like compilers, library routines, and data files. In a large multiprocessor system, it is usually very expensive to provide a copy of a resource to each processor/node in the system. It also often leads to poor utilization of some of the copies of the resource and thus results in poor price-performance ratio. On the other hand, providing a system with very few copies of a resource leads to contention and hence loss of performance. It also makes the system susceptible to failures because a loss of a few copies can result in unavailability of that resource to some nodes. Therefore, there is a tradeoff among cost, performance, and availability that has to be taken into account in placing resources in a multiprocessor system.

Traditionally, this tradeoff is achieved by placing copies of a resource in such a way that multiple processors share a copy. Processors that do not have a copy of a resource use the copies from "nearby" processors. An *optimal* placement of a resource in a multiprocessor system should be such that there is a balance between the time spent by a processor in accessing a shared copy and the total number of copies of that resource in the system. Finding an optimal placement of resources for multiprocessor systems with arbitrary constraints on the time spent by a processor and an arbitrary number of copies is a very difficult problem. In this paper, we study and solve the problem of placing resources in a class of multiprocessor systems commonly referred to as k-ary n-cubes [4].

A k-ary n-cube (k^n) is a multiprocessor system with k^n processors/nodes arranged along n dimensions, with k nodes in each dimension. Hypercubes and two-dimensional tori are special cases of k-ary n-cubes. An n-dimensional hypercube is a 2-ary n-cube [8] while a two-dimensional torus is a k-ary 2-cube. The Intel iPSC-2 and Ametek 2010 systems [9] are examples of commercially available k-ary n-cubes.

Each processor/node in k^n can be uniquely addressed by an n-digit radix-k number. That is, each processor a in k^n can be uniquely addressed by an n-tuple $(a_n, \cdots, a_1)$, where $a_i \in \{0, 1, \cdots, (k-1)\}$, for $1 \leq i \leq n$. In k^n, two processors $a \equiv (a_n, \cdots, a_1)$ and $b \equiv (b_n, \cdots, b_1)$ are connected to each other if and only if there exists an $i, 1 \leq i \leq n$, such that $a_i \equiv b_i \pm 1 (\mathrm{mod}\, k)$ and $a_l = b_l$, for all $l \neq i$. It follows from this definition that each processor in $k^n, k > 2$, is connected to exactly $2n$ other processors. Furthermore, it can be shown that k^n is node-symmetric in the sense that for every pair of nodes a and b there exists a homomorphism that maps node a to node b.

For this class of systems, the problem addressed in this paper can be stated as follows. First, we find the minimum number of copies of a resource required to ensure that each node either has a copy of the resource, or is adjacent to j other nodes that have a copy of the resource. Then, given the minimum number of copies we determine where these resource copies should be placed in order to satisfy the above condition. The above condition is commonly referred to as the j-adjacency constraint. This constraint implies that a node that does not have a resource copy can find j copies of that resource among neighboring nodes.

In the rest of this paper, a solution that satisfies the j-adjacency constraint in k^n is referred to as a j-adjacency placement. Also, nodes that have a copy of a resource are referred to as *resource nodes* and those that don't have a copy are referred to as *nonresource nodes*. Furthermore, as in [2], a

Manuscript received August 1992; revised February 1993 and August 1993. This work was supported in part by the National Science Foundation under Grants MIP-9009154, MIP-9213716, and CCR-9308966. This paper was presented in part at the International Conference on Parallel Processing, 1992.

The authors are with the Department of Electrical and Computer Engineering, University of Wisconsin-Madison, Madison, WI 53706 USA.

IEEE Log Number 9409333.

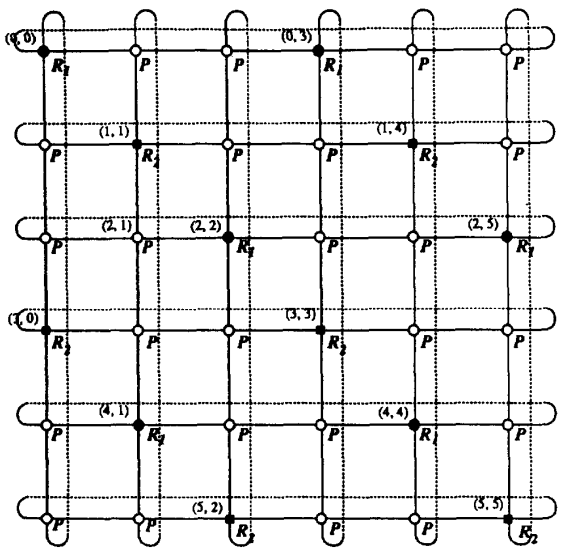


Fig. 1. Perfect placement of two distinct types of resources in 6^2.

j-adjacency placement is said to be *perfect* if and only if no two resource nodes are adjacent and each nonresource node is adjacent to exactly j resource nodes. Similarly, a j-adjacency placement is said to be *quasi-perfect* if and only if no two resource nodes are adjacent and each nonresource node is adjacent to at least j but no more than $(j+1)$ resource nodes.

Motivation for j-adjacency placements arises in many contexts. It can be used to assign j distinct types of resources and processors to nodes of k^n in such a way that each processor is adjacent to each type of resource. Examples of resource types include memory modules, cache units, disk controllers, and I/O processors. For instance, Fig. 1 shows a perfect 2-adjacency placement in 6^2, where nodes labeled P are nonresource nodes and nodes labeled R_1 and R_2 are resource nodes. Note that, in this placement, each processor P is adjacent to exactly one resource node of type R_1 and one resource node of type R_2. If all resources are of the same type, a j-adjacency placement may also be used to tolerate up to $(j-1)$ failures of resource nodes.

Reddy *et al.* considered the 1-adjacency problem for the placement of I/O processors in a hypercube [7]. Livingston and Stout studied the problem of placing a minimum number of resources in a hypercube subject to certain constraints [6]. Chiu and Raghavendra addressed the problem of placing a given number of resources with an objective of minimizing the resource diameter [3] where the resource diameter of a placement is defined as the maximum value, taken among all processors, of the minimum number of hops a node must traverse in order to access a resource.

More recently, Chen and Tzeng [2] proposed solutions for perfect and quasi-perfect j-adjacency placements in a hypercube subject to the general communication constraint that a nonresource node is within h hops of j resource nodes for any given $h \geq 1$. The techniques and results of Chen and Tzeng [2] cannot be extended to k-ary n-cubes due to one key difference between an n-dimensional hypercube and a k^n. The difference is that the binary addresses of all nodes in an n-dimensional hypercube form an n-dimensional vector space over the field $\{0,1\}$ whereas the radix-k addresses of k^n do not in general form a vector space.

Livingston and Stout use the theory of perfect d-dominating sets to study the problem of resource placement in several class of networks such as hypercubes, two- and three-dimensional meshes and tori, trees, cube-connected cycles, and de Bruijn graphs. In particular, they propose methods to construct resource placements in which each nonresource node can reach *exactly one* resource node within a distance of $d(d \geq 1)$ from itself [5]. In contrast, in this paper, we consider placements in which each nonresource node is adjacent to $j(j \geq 1)$ resource nodes in k^n.

The rest of this paper is organized as follows. In Section II, we prove that perfect (quasi-perfect) j-adjacency placements are not possible in k^n if $n < j < 2n (n < j < 2n-1)$. In Section III, we derive necessary and sufficient conditions for the existence of perfect 1-adjacency placements and describe an algorithm for finding them. We also derive sufficient conditions for the existence of perfect j-adjacency placements and extend the 1-adjacency algorithm to find j-adjacency solutions. We show that, under certain conditions, these algorithms can be extended to place j distinct types of resources in k^n. For values of k, n, and j for which perfect/quasi-perfect solutions do not exist, we consider approximate j-adjacency placements in Section IV. We also show that the number of resource copies used by the approximate solutions asymptotically approaches a constant multiple of a theoretical lower bound on the number of resource copies required for any j-adjacency placement. In Section V, the results in this paper are summarized and some open problems are identified.

II. Existence of j-Adjacency in Placements k-ary n-Cubes

In this section, we derive some of the necessary conditions for the existence of j-adjacency perfect and quasi-perfect placements. From these conditions it will be clear that the solutions for the hypercube do not easily generalize to higher radix k-ary n-cubes. For example, it has been shown in [2] that either a perfect or a quasi-perfect j-adjacency placement exists in a hypercube of any dimension. This is not true in a higher radix k^n as shown in the theorems below.

Theorem 1: A j-adjacency perfect placement exists in k^n, $k > 2$, only if either $j = 2n$ or $1 \leq j \leq n$.

Proof: Suppose a j-adjacency perfect placement exists in k^n, $k > 2$. Then, one of the following two complementary conditions must be true:

1) no two nonresource nodes are adjacent to each other, or
2) there exists at least one pair of adjacent nonresource nodes.

Now, suppose that the first condition is true. Since, no two nonresource nodes are adjacent to each other, all nodes adjacent to a nonresource node have a copy of the resource. Since in $k^n, k > 2$, each node is adjacent to $2n$ nodes, this implies that each nonresource node is adjacent to $2n$ resource nodes. Hence, $j = 2n$.

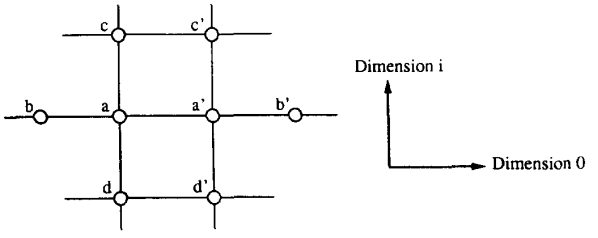


Fig. 2. Illustration of adjacencies of nodes a and a' for Theorem 1.

Next, suppose that the second condition is true. That is, there exists a pair of adjacent nonresources nodes, say a and a'. Since k^n is vertex as well as edge symmetric, we can assume without loss of generality that the addresses of a and a' differ only in dimension 0.

In dimension 0, node a is adjacent to precisely one node $b \neq a'$ (see Fig. 2). Similarly, node a' is adjacent to precisely one node $b' \neq a$ in dimension 0. Therefore, a maximum of two copies of a resource can be placed among the neighbors of a and a' in dimension 0.

Next, consider a dimension $i \neq 0$. In this dimension, node a is adjacent to two nodes, say c and d. Similarly a' is adjacent to two nodes c' and d' (see Fig. 2). Since, by definition, no two resource nodes are adjacent to each other in a perfect placement, a maximum of two resource copies can be placed among the neighbors of a and a' in this dimension.

From the above two observations, we can conclude that a maximum of $2n$ copies of a resource can be placed among the neighbors of a and a' (for a perfect placement). However, since we are considering a j-adjacency perfect placement, there must be a total of $2j$ resource copies among the neighbors of a and a'. Therefore, $2j$ must be less than or equal to $2n$, or in other words, $j \leq n$. ■

The necessary conditions proved above are usually not sufficient for the existence of perfect j-adjacency placements. In most cases, there are conditions on k depending on the values of n and j. For example, as shown in the theorem below, a perfect $2n$-adjacency placement does not exist if k is odd.

Theorem 2: A perfect $2n$-adjacency placement exists in k^n, $k > 2$, if and only if k is even.

Proof: (*Necessity*) Since each node a in $k^n, k > 2$, has $2n$ neighbors, every neighbor of a nonresource node must be a resource node. Further, by definition, no two resource nodes can be adjacent in a perfect placement. It follows from these two observations, that every alternate node along any dimension of k^n should have a copy of the resource. This implies that k must be even.

(*Sufficiency*) If k is even, a perfect $2n$-adjacency placement can be obtained using the following algorithm. Arbitrarily choose any node a and place a copy of the resource at that node. Then, place copies of the resource at every node that is at an even distance from node a. It can be easily shown that this simple algorithm assigns a resource to all the $2n$ neighbors of a node that is not assigned a resource and that no two resource nodes are adjacent. Hence the theorem. ■

From Theorem 1, we can conclude that a perfect $(2n-1)$-adjacency placement does not exist in k^n. However, using an algorithm similar to the one in the sufficiency proof of Theorem 2, we can show that there is a quasi-perfect $(2n-1)$-adjacency placement in k^n. Necessary conditions for the existence of quasi-perfect placements are stated in the theorem below. Note that, unlike in hypercubes [2], quasi-perfect placements do not always exist in $k^n, k > 2$.

Theorem 3: A quasi-perfect j-adjacency placement exists in $k^n, k > 2$, only if either $1 \leq j \leq n$, or $j = (2n-1)$, or $j = 2n$.

Proof: The proof is similar to that of Theorem 1, and hence is omitted. ■

Quasi-perfect placements are of interest when perfect placements do not exist for given values of k, n, and j. However, constructing quasi-perfect placements for all values of k, n, and j seems to be a difficult problem. Therefore, when perfect placements cannot be found, we present algorithms for constructing approximate placements in which each nonresource node is adjacent to at least j resource nodes.

III. Perfect Placement in k-ary n-Cubes

Algorithms for perfect placements in k^n is the subject of this section. In Section III–A, we derive necessary and sufficient conditions for the existence of perfect 1-adjacency placements in k^n. As a part of the sufficiency proof, we describe an algorithm for constructing a perfect 1-adjacency placement. Then, in Section III-B, we derive sufficient conditions for the existence of perfect j-adjacency placements. Here again, we describe an algorithm to construct the placements as a part of the proof.

Our approach for constructing perfect j-adjacency placements is similar to that of Chen and Tzeng [2]. However, there is one key difference between an n-dimensional hypercube and a k^n that distinguishes our approach from that in [2]. The difference is that the binary addresses of all nodes in an n-dimensional hypercube form an n-dimensional vector space over the field $\{0, 1\}$ whereas the radix-k addresses of k^n do not in general form a vector space. Consequently, we cannot directly use results from coding theory as in the case of hypercubes. Solutions for hypercubes may extend to a topology known as *generalized hypercubes* in which two nodes are adjacent if their radix-k addresses differ in exactly one digit [1].[1]

Before we discuss the algorithms for resource placement, we prove a lemma that characterizes the number of resource copies required in a perfect j-adjacency placement.

Lemma 1: The number of resource nodes in a perfect j-adjacency placement in $k^n, k > 2$, equals $j \cdot k^n/(2n + j)$.

Proof: Let X be the number of resource nodes in a perfect j-adjacency placement. Then, these nodes must provide j adjacencies to each of the $(k^n - X)$ nonresource nodes. That is, the total number of adjacencies required is $j \cdot (k^n - X)$. On the other hand, the total adjacencies provided by the resource nodes is $2n \cdot X$ because no two resource nodes are adjacent to each other in a perfect placement.

[1] Recall that, in k^n, two nodes are adjacent iff their radix-k addresses differ *by one* (modulo k) in exactly one digit.

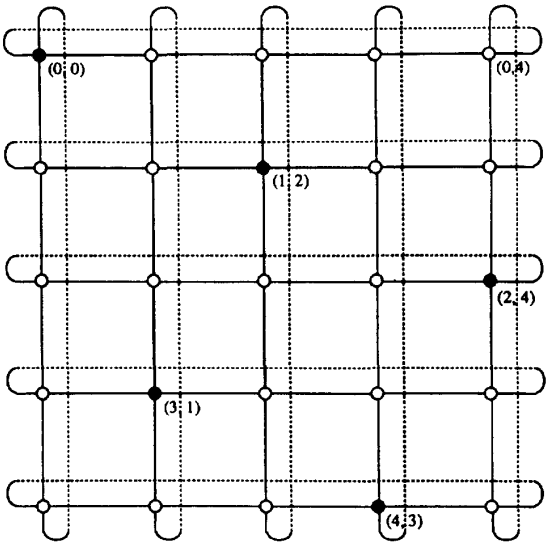


Fig. 3. Perfect 1-adjacency placement in 5-ary 2-cube.

Equating these two quantities we get

$$2n \cdot X = j \cdot (k^n - X).$$

Or, $X = j \cdot k^n/(2n + j)$. ■

Since the number of resource nodes is an integer, it follows from the above lemma that a j-adjacency perfect placement does not exist if $j \cdot k^n$ is not an integral multiple of $(2n + j)$. In Section III–B, we identify some sufficient conditions for the existence of perfect j-adjacency placements, when $j > 2$. When $X = j \cdot k^n/(2n+j)$ is not an integer, $\lceil j \cdot k^n/(2n+j) \rceil$ is a lower bound on the number of resource copies required for j-adjacency placements in k^n.

A. Perfect 1-Adjacency Placement

From Lemma 1, it follows that perfect 1-adjacency placement exists in k^n only if k^n is an integral multiple of $2n+1$. In this section, we first derive sufficiency conditions under which perfect 1-adjacency placement exists in k^n. We also show that if $n < 22$, then the necessary condition that $2n + 1$ divides k^n is also sufficient for finding perfect 1-adjacency placements.

Theorem 4: A perfect 1-adjacency placement exists in k^n if there exist integers q and r such that q divides k and $q^r = 2n + 1$.

Proof: Given q and r such that the conditions in the theorem are satisfied, we prove the theorem by constructing a perfect 1-adjacency placement in k^n. The algorithm for constructing this placement is as follows.

Consider the set, Q, of all nonzero r-tuples over the set of integers $\{0, 1, \cdots, q-1\}$. Partition Q into subsets containing only an r-tuple and its additive inverse. Note that, there are exactly $(q^r - 1)/2$ subsets in this partition because no r-tuple in Q is an additive inverse of itself. This is due to the fact that q must be an odd integer in order to satisfy the condition $q^r = 2n + 1$. Construct a $r \times n$ matrix $H = [h_1 h_2 \cdots h_n]$ using exactly one r-tuple from each subset in the partition as the column vector. Place a resource copy at a node $a = (a_n, \cdots, a_1)$ if and only if

$$a \cdot H^T \equiv 0 \pmod{q}.$$

We now prove that the resulting placement is a perfect 1-adjacency placement in k^n. To prove this result, we must prove that 1) no two resource nodes are adjacent to each other, and 2) each nonresource node is adjacent to exactly one resource node.

The first condition is proved by contradiction. Let, if possible, $a \equiv (a_n, \cdots, a_1)$ and $b \equiv (b_n, \cdots, b_1)$ be two adjacent resource nodes. That is, $a \cdot H^T \equiv b \cdot H^T \equiv 0 \pmod{q}$. Without loss of generality, we can assume that $b \equiv a \oplus e_i$, where $\oplus$ denotes addition modulo k and e_i is an n-tuple with a 1 in the ith dimension and zeros in all other dimensions. Thus,

$$\begin{aligned} & b \cdot H^T \equiv 0 \pmod{q} \\ \Rightarrow\ & (a \oplus e_i) \cdot H^T \equiv 0 \pmod{q} \\ \Rightarrow\ & (a + e_i) \cdot H^T \equiv 0 \pmod{q} \text{ (since } q \text{ divides } k) \\ \Rightarrow\ & e_i \cdot H^T \equiv 0 \pmod{q} \text{ (since } a \text{ is a resource node)} \\ \Rightarrow\ & h_i \equiv 0 \pmod{q}. \end{aligned}$$

This is a contradiction because all column vectors in H are nonzero. Therefore, no two resource nodes are adjacent to each other.

To prove the second condition, consider a nonresource node a. From the definition of k^n, the addresses of all the neighbors of a are of the form $a \oplus e_i$ or $a \ominus e_i$, for some $1 \le i \le n$; here $\oplus$ and $\ominus$ are, respectively, addition and subtraction modulo k. We now show that there is an unique i for which either $(a \oplus e_i) \cdot H^T \equiv 0 \pmod{q}$ or $(a \ominus e_i) \cdot H^T \equiv 0 \pmod{q}$.

Since a is a nonresource node, $a \cdot H^T \pmod{q} \in Q$. Therefore, it follows from the construction of H that there are two mutually exclusive cases: $a \cdot H^T \pmod{q}$ is either a column vector of H or is an additive inverse of a unique column vector of H. We now show that, in either case, a is adjacent to a unique resource node.

Case 1: $a \cdot H^T \pmod{q} = h_i$ for an unique $i, 1 \le i \le n$. In this case,

$$\begin{aligned} (a \ominus e_i) \cdot H^T &\equiv (a - e_i) \cdot H^T \pmod{q} \text{ (since } q \text{ divides } k) \\ &\equiv a \cdot H^T - h_i \pmod{q} \\ &\equiv 0 \pmod{q}. \end{aligned}$$

Therefore, a is adjacent to exactly one resource node, $a \ominus e_i$.

Case 2: $a \cdot H^T \pmod{q} \neq h_i$ for all $i, 1 \le i \le n$. From the definition of H, there is an unique $i, 1 \le i \le n$, such that $a \cdot H^T + h_i \equiv 0 \pmod{q}$. In this case,

$$\begin{aligned} (a \oplus e_i) \cdot H^T &\equiv (a + e_i) \cdot H^T \pmod{q} \text{ (since } q \text{ divides } k) \\ &\equiv a \cdot H^T + h_i \pmod{q} \\ &\equiv 0 \pmod{q}. \end{aligned}$$

Therefore, a is adjacent to exactly one resource node, $a \oplus e_i$. This proves the second condition and hence the theorem. ■

To illustrate Theorem 4, consider the construction of a 1-adjacency perfect placement in 5^2. From the proof of the above theorem, the matrix H in this case is [1 2]. Node addresses that have resources in this case are (0, 0), (1, 2), (2, 4), (3, 1), and (4, 3). Fig. 3 shows this resource placement for a 5^2;

nodes with resource copies are shown with filled circles. In this figure, note that, each nonresource node is adjacent to exactly one resource node. For example, nonresource node (0, 4) is adjacent to resource node (0, 0) through a wrap-around edge.

Notice that a placement obtained by translating each resource node in a perfect placement by a constant is also perfect. As an example, let us add (1, 1) to the address of each resource node in Fig. 3. It can be easily seen that the new resource nodes (1, 1), (2, 3), (3, 5), (4, 2), and (5, 4) also constitute a perfect 1-adjacency placement in 5^2. Thus, even though the above theorem gives exactly one perfect placement, one can construct multiple disjoint perfect placements by appropriate translation.

Theorem 4 identifies a sufficient condition for the existence of perfect 1-adjacent placements. In the general case, this condition is not equivalent to the necessary condition that k^n must be a multiple of $2n + 1$. For example, if $k = 15$ and $n = 22, 2n + 1$ divides k^n. However, there are no integers q and r such that q divides k and $q^r = 2n + 1 = 45$. Therefore, although perfect 1-adjacency placement may exist in 15^{22}, Theorem 4 cannot be used to find such a placement. However, this is not a serious limitation because we next show that for all practical systems, the necessary condition implies the sufficient condition.

Theorem 5: If $n < 22$, perfect 1-adjacency placements exist in k^n if and only if k^n is an integral multiple of $2n + 1$.

Proof: (*Necessity*) Follows from Lemma 1.

(*Sufficiency*) If $n < 22$, then $2n+1$ has only one prime factor and/or all prime factors of $2n + 1$ have a multiplicity of one.

If $2n + 1$ has only one prime factor, then $2n + 1 = p^m$ for some prime p and integer m. Since $2n + 1$ divides k^n, p must also be a prime factor of k. Therefore, by setting $q = p$ and $r = m$, the existence of perfect 1-adjacency placement in k^n follows from Theorem 4.

On the other hand, if all prime factors of $2n + 1$ have a multiplicity of one, then $2n + 1 = p_1 p_2 \cdots p_l$ for some primes $p_1, p_2, \cdots, p_l$. Since $2n + 1$ divides k^n, each $p_1, p_2, \cdots, p_l$, must also be a prime factor of k. That is, $2n + 1$ divides k. Thus, by setting $q = 2n+1$ and $r = 1$, the existence of perfect 1-adjacency placement in k^n follows from Theorem 4. Hence, the theorem. ■

B. Perfect j-Adjacency Placement

From Lemma 1, we know that a perfect j-adjacency placement can exist in k^n only if $j \cdot k^n$ is a multiple of $2n + j$. In this section, we derive a sufficient condition under which such a placement can be found.

Theorem 6: A perfect j-adjacency placement exists in k^n if there exist integers q and r such that q divides k and $q^r - 1 = 2n/j$.

Proof: Given q and r such that the conditions in the theorem are satisfied, we prove the theorem by constructing a perfect j-adjacency placement in k^n. Consider the set, Q, of all nonzero r-tuples over the set of integers $\{0, 1, \cdots, q - 1\}$. Partition Q into subsets containing only an r-tuple and its additive inverse. Construct a set Q' by picking exactly one r-tuple from each subset in the partition. Then, identify a matrix H such that for each $h \in Q'$

1) if $h + h \equiv 0 \pmod q$, then $j/2$ column vectors of H are equal to h, and
2) if $h + h \not\equiv 0 \pmod q$, then j column vectors of H are equal to h.

That is, each self-inverse in Q' appears as $j/2$ column vectors of H and each nonself-inverse in Q' appears as j column vectors of H. Note that, there are either no self-inverses or $2^r - 1$ self-inverses in Q' depending on whether q is odd or even, respectively. In either case, the total number of column vectors in H is n. Place a resource copy at a node a if and only if

$$a \cdot H^T \equiv 0 \pmod q.$$

We now prove that the resulting placement is a perfect j-adjacency placement in k^n. To prove this result, we must prove that 1) no two resource nodes are adjacent to each other, and 2) each nonresource node is adjacent to exactly j-resource nodes.

The first condition can be proved by contradiction just as in the case of perfect 1-adjacency placements (see proof of Theorem 4). To prove the second condition, consider a nonresource node a. From the placement rule, we know that $a \cdot H^T = u \in Q$. There are three mutually exclusive possibilities: 1) u is a self-inverse 2) u is not a self-inverse, but, u is a column vector of H, and (iii) u is not a self-inverse and u is not a column vector of H.

In the first case, $j/2$ columns of H are equal to u. Let i be one such column. In this case, we can verify that $(a \oplus e_i) \cdot H^T \equiv (a \ominus e_i) \cdot H^T \equiv 0 \pmod q$. That is, nodes $(a \oplus e_i)$ and $(a \ominus e_i)$ are two resource nodes adjacent to a. Since there are a total of $j/2$ such columns, a is adjacent to exactly j resource nodes.

In the second case, j columns of H are equal to u. Let i be one such column. In this case, it can be shown that $(a \oplus e_i) \cdot H^T \equiv 0 \pmod q$. In other words, node $(a \oplus e_i)$ is a resource node adjacent to a. Since there are j such columns, a is adjacent to exactly j resource nodes.

In the third case, j columns of H are equal to the additive inverse of u. Let i be one such column. In this case, it can be shown that $(a \ominus e_i) \cdot H^T \equiv 0 \pmod q$. In other words, node $(a \ominus e_i)$ is a resource node adjacent to a. Since there are j such columns, a is adjacent to j resource nodes.

Thus, in all three cases, a is adjacent to exactly j resource nodes. This proves the theorem. ■

To illustrate Theorem 6, we construct a perfect 2-adjacency placement in 6^2. The sufficient conditions mentioned in Theorem 6 are satisfied for this example with $q = 3$ and $r = 1$. Thus, the H matrix is [1 1]. The resources nodes in a perfect 2-adjacency placement in 6^2 are shown with filled circles in Fig. 4. In this figure, we note that, each nonresource node is adjacent to exactly two resource nodes. For example, nonresource node (2, 2) is adjacent to the resource nodes (2, 1) and (1, 2).

C. Perfect Placement of j Distinct Resource Types

Definition 1: A *perfect j-type j-adjacency placement* is a placement with j distinct types of resources in which 1) each nonresource node is adjacent to exactly one resource node of

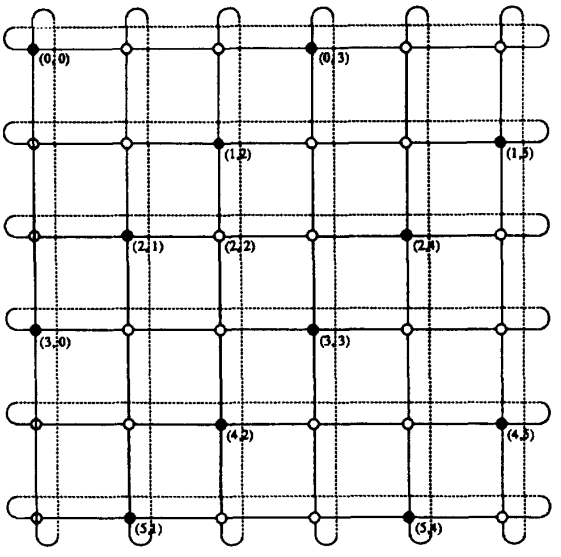


Fig. 4. Perfect 2-adjacency placement in 6-ary 2-cube.

each type, and 2) no two resource nodes are adjacent to each other.

The method described in the proof of Theorem 6 for obtaining perfect j-adjacency placements can be extended, under certain conditions, to construct a perfect j-type j-adjacency placement. The following two lemmas identify sufficient conditions under which such an extension is possible.

Theorem 7: A perfect j-type j-adjacency placement exists in k^n if

1) k, n, and j satisfy the conditions in Theorem 6 with q odd, and
2) there exist integers p and m such that p divides k and $p^m = j$.

Proof: The first condition in the theorem implies that there exists a perfect j-adjacency placement in k^n. Identify the matrix H as in the proof of Theorem 6. Let $\mathcal{P} = \{0, P_1, P_2, \cdots, P_{j-1}\}$ be the set of all m-tuples over the set of integers $\{0, 1, \cdots, p-1\}$. Construct an $m \times n$ matrix $R = (r_n, \cdots, r_1)$ such that

1) each column vector of R is a m-tuple in $\mathcal{P}$, and
2) if $h_i = h_l$, then $r_i \neq r_l$, where h_i and h_l are ith and lth column vectors of H.

Note that, the above two conditions can be satisfied because 1) a column vector can repeat at most j times in H, and 2) there are j distinct vectors in $\mathcal{P}$.

Now identify a perfect j-adjacency placement using the method described in the proof of Theorem 6. To each resource node a in this placement assign a type i if $a \cdot R^T \equiv P_i \pmod p$ for some $P_i \in \mathcal{P}$. We now show that, in this assignment, no two resource nodes adjacent to a nonresource node are of the same type.

Let, if possible, a be a nonresource node which is adjacent to two resource nodes of the same type. Also, let $a \cdot H^T \bmod q = h$. Since q is odd, $h + h \not\equiv 0 \pmod q$; thus, from the construction of matrix H, we conclude that either 1) h appears j times in H, or 2) the additive inverse of h appears j times in H.

If h appears j times in H, then any resource node adjacent to a must be of the form $(a \ominus e_i)$ such that $e_i H^T = h_i = h$. Further, two resource nodes $(a \ominus e_i)$ and $(a \ominus e_l)$ must satisfy the condition $h_i = h_l$. These two resource nodes will have the same resource type if and only if

$$\begin{aligned}
&(a \ominus e_i) \cdot R^T \equiv (a \ominus e_l) \cdot R^T \pmod p \\
\Rightarrow\ &(a - e_i) \cdot R^T \equiv (a - e_l) \cdot R^T \pmod p \\
&\qquad \text{(since } p \text{ divides } k\text{)} \\
\Rightarrow\ &e_i \cdot R^T \equiv e_l \cdot R^T \\
\Rightarrow\ &r_i \equiv r_l \\
\Rightarrow\ &h_i \not\equiv h_l \text{ (from the construction of } R\text{)}.
\end{aligned}$$

This contradicts the requirement that $h_i = h_l$.

On the other hand, if the additive inverse of h appears j times in H, then any resource node adjacent to a is of the form $(a \oplus e_i)$ such that $e_i H^T = h_i = h$. Proceeding as above, we can once again show that no two resource nodes adjacent to a are of the same type.

Hence, the theorem. ■

IV. Approximate Resource Placements in k^n

In Section II, we discussed the necessary and sufficient conditions for the existence of perfect and quasi-perfect j-adjacency placements. The advantages of perfect/quasi-perfect placements are that 1) no two resource nodes are adjacent to each other, and 2) no nonresource node is adjacent to more than $j+1$ resource nodes. These two conditions ensure j adjacencies using as few resource copies as possible. However, perfect/quasi-perfect j-adjacency placements do not exist for all values of k, n, and j.

In order to construct j-adjacency placements for all values of k, n, and j, we relax the constraints 1) and 2) stated above. This implies that in the j-adjacency placements constructed in this section two resource nodes can be adjacent to each other and a nonresource node can be adjacent to more than $(j+1)$ resource nodes. As a result, the number of copies of a resource in the constructed placements may be more than the minimum number required.

In view of the discussion above, we consider three mutually exclusive cases: 1) j divides $2n$ and $1 \leq j \leq n$, 2) j does not divide $2n$ and $1 \leq j \leq n$, and 3) $n < j < 2n$. Solutions for these cases are described in the following three subsections. In the first case, the number of resource copies used by the approximate solutions asymptotically approaches the lower bound $j \cdot k^n/(2n+j)$ as $k \to \infty$ for each n. In the other two cases, the number of resource copies used by the approximate solutions is within a constant factor of the theoretical lower bound for large k.

A. Approximate Placements when j Divides $2n, 1 \leq j \leq n$

An algorithm for constructing approximate placements in this case is given in Fig. 5. The basic idea of this algorithm is best illustrated by an example. Consider a 1-adjacency

Algorithm Resource_Place

Input Parameters: k, n, and j;

1. If $2n/j$ is not an integer
2. Print "The algorithm cannot handle this case";
3. Exit;
4. Endif
5. Find integers q and r such that q divides k and $q^r - 1 = 2n/j$.
   If such integers cannot be found, find smallest q such that $q^r - 1 = 2n/j$.
6. Let H be as in the proof of Theorem 6.
7. Place a copy of a resource at a if $a \cdot H^T \equiv 0 \pmod q$.
8. If k is an integral multiple of q
9. Return Placement; /* Perfect j-adjacency placement */
10. For $i = 1$ to n do
11. Place a copy of a resource at $a \equiv (a_n, \ldots, a_{i+1}, (k-1), a_i, \ldots, a_1)$ if
12. $\left[a_n \ \cdots \ a_{i+1} \ (q-1) \ a_{i-1} \ \cdots \ a_1 \right] \cdot H^T \equiv 0 \pmod q$;
13. Place a copy of a resource at $a \equiv (a_n, \ldots, a_{i+1}, (k-1), a_i, \ldots, a_1)$ if
14. $\left[a_n \ \cdots \ a_{i+1} \ k \ a_{i-1} \ \cdots \ a_1 \right] \cdot H^T \equiv 0 \pmod q$;
15. Endfor;
16. Return Placement.

Fig. 5. Algorithm for constructing approximate j-adjacency placement.

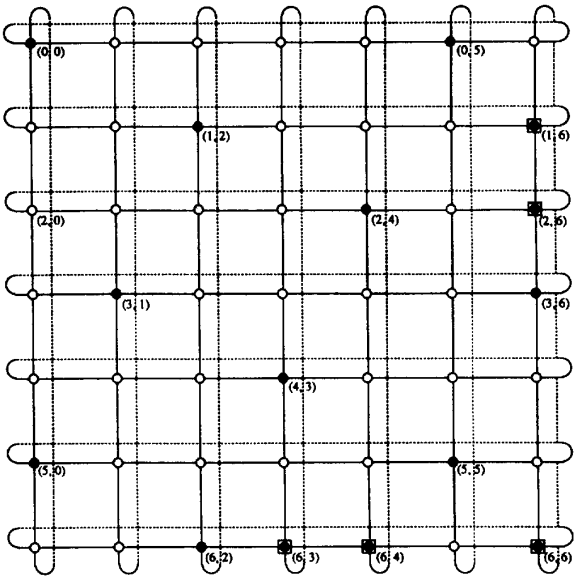


Fig. 6. 1-adjacency placement in 7-ary 2-cube.

placement in 7^2. Since there are no integers q and r such that q divides 7 and $q^r - 1 = 4$, Step 5 of the algorithm returns $q = 5$ and $r = 1$. The algorithm then places copies of a resource at all nodes orthogonal to the matrix $H = [1 \quad 2]$ (Refer to Steps 6 and 7 of algorithm Resource_Place). The resource nodes thus obtained are shown with • in Fig. 6. At the end of step 7 of algorithm Resource_Place, if k is not a multiple of q (in this case 5), then some nodes still would not have received their j adjacencies. For example, in Fig. 6, node (2, 0) is not adjacent to any resource node at the end of this step. This is because, if k were 10 (a multiple of q), this node would have received its adjacency from node (2, 9) (Node (2, 9) is orthogonal to H). Since in a $7^2, (2,0)$ is adjacent to (2, 6), we must provide a resource copy at node (2, 6). This procedure must be repeated for each node of the form $(x, 0)$ which were to receive its adjacency from $(x, 9)$ in 10^2 (Steps 11 and 12 of algorithm Resource_Place). Similarly, nodes of the form $(6, x)$ which were to receive their adjacency from nodes $(7, x)$ if k were 10, should each be provided with a copy of the resource (Steps 13 and 14 of algorithm Resource_Place). In Fig. 5, the nodes that were provided with resource copies in Steps 11–14 of the algorithm are indicated by placing a □ around the nodes.

We observe that the total number of resources required for the placement shown in Fig. 5 is 15 as compared to the lower bound 10 obtained from the formula $j \cdot k^n/(2n + j)$. However, the number of resource copies can be reduced in some instances. For example, nodes (1, 6) and (6, 4) were provided with a copy of the resource in Steps 13 and 14 because these nodes were to receive their adjacencies from nodes (1, 7) and (7, 4), respectively, in 10^2. However, since nodes (1, 6) and (6, 4) had already received their adjacencies from nodes (2, 6) and (6, 3), respectively, (which were provided with resource copies in Steps 11–12) resource copies from nodes (1, 6) and (6, 4) could have been eliminated. This optimization can be easily incorporated into algorithm Resource_Place by modifying Steps 13 and 14 as follows.

13.	Place a copy of a resource at $a \equiv (a_n, \cdots, a_{i+1}, (k-1), a_i, \cdots, a_1)$
14a.	if fewer than j neighbors of node a have a resource copy and
14b.	$[a_n \cdots a_{i+1} k a_{i-1} \cdots a_1] \cdot H^T \equiv 0 \bmod q$;

Algorithm Resource_Place can be extended under the conditions identified in Theorems 7 to place j distinct types of resources to ensure that each nonresource node is adjacent to a resource node of each type. However, in this case, one cannot perform the optimizations in Steps 13, 14a, and 14b. Through examples, it can also be shown that Algorithm Resource_Place is not always optimal. For instance, one can place resource copies at nodes

$$\{(0,0),(0,4),(1,2),(2,0),(2,5),(2,6),(3,3),(4,1),$$
$$(5,4),(5,5),(5,6),(6,2)\}$$

to obtain 1-adjacency placement in 7^2. This placement uses only twelve resource copies as opposed to thirteen required by the optimized Algorithm Resource_Place. Since, at present, we are unable to determine the number of resource copies in an optimal placement, we compare the number of resource copies placed by algorithm Resource_Place with the theoretical lower bound $j \cdot k^n/(2n + j)$. We show that the difference between the number of resource copies required by the algorithm and the theoretical lower bound approaches zero as k tends to ∞ for each n.

Theorem 8: The number of resource copies required by algorithm Resource_Place for a j-adjacency placement approaches the theoretical lower bound $j \cdot k^n/(2n + j)$ as k tends to ∞ for each n.

Proof: Let $k = s \cdot q + t$ for integers s and t such that $s \geq 0$, and $0 < t < q$. At the end of Step 7, the number of resource copies placed by the algorithm is at most equal to $j \cdot k^n/(2n + j)$. Thus, the only additional copies placed by the algorithm are in Steps 10–15. An upper bound on the number of copies placed in these steps can be obtained by counting the number of resource copies placed in an $(n-1)$-dimensional hyperplane of $((s+1)q)^n$. That is, in each iteration of the for-loop, the number of copies placed is at most $j \cdot ((s+1)^{n-1} \cdot q^{n-1})/(2n + j)$.

Therefore, the ratio of the additional copies to the theoretical lower bound is

$$\frac{(s+1)^{n-1}q^{n-1}}{(s \cdot q + t)^n}.$$

Since $q^r - 1$ must equal $2n/j$, q cannot exceed $(2n + j)/j$. Hence, $k \to \infty$ is equivalent to $s \to \infty$. Taking limits as $s \to \infty$, the above ratio tends to zero. ■

B. Approximate Placements when j Does Not Divide $2n$. $1 \leq j \leq n$

Recall that, Algorithm Resource_Place is based on the assumption that j divides $2n$. In this section, we consider the case when j does not divide $2n$. We propose two different techniques to handle this case. Depending on the values of the k, n, and j, either of the two techniques could be better in terms of the number of resource copies required. The first technique finds an n' such that j divides $2n'$ and then uses schemes from the previous sections to determine the required resource placement. In contrast, the second technique finds a j' such that j' divides $2n$ and then applies our earlier schemes. These two techniques are described below.

1) Modification of n to n': Let n' be the largest integer less than n such that $2n'$ is divisible by j. Partition k^n into $k^{n-n'}$ k-ary n'-cubes and use Algorithm Resource_Place to find approximate j-adjacency placements in each one of the k-ary n'-cubes. This is possible because j divides $2n'$. Since each node in k^n belongs to one of these k-ary n'-cubes, this technique ensures that the j-adjacency constraint is satisfied, albeit using more resource copies. The theorem below compares the number of resource copies required by this technique to the theoretical lower bound on the number of resource copies required.

Theorem 9: The ratio of the number of resource copies required by the above technique to the theoretical lower bound on the number of resource copies required for a j-adjacency placement in k^n is bounded by 5/3 as k tends to ∞ for each n.

Proof: From Theorem 8, the number of resources placed by Algorithm Resource_Place in each $k^{n'}$ approaches $(jk^{n'})/(2n' + j)$ as $k \to \infty$. Hence, the total number of resources placed by the above technique approaches $(jk^{n'}/2n' + j) \cdot k^{n-n'}$ as $k \to \infty$. Since the theoretical lower bound on the number of resource copies required is $(jk^n)/(2n + j)$, the ratio of the resource copies placed by the above technique to this theoretical lower bound approaches $(2n + j)/(2n' + j)$ for large k.

Since j does not divide $2n$, let $2n = sj + t$ for some integers s and t, $s \geq 0$ and $0 < t < j$. If s and j are odd, then $n' = (s-1)j/2$ is the largest integer less than n such that $2n'$ is divisible by j. Otherwise, the largest such integer is $sj/2$. Therefore, $n - n' < j$. Hence,

$$\begin{aligned}\frac{2n + j}{2n' + j} &= 1 + \frac{2(n - n')}{2n' + j} \\ &< 1 + \frac{2j}{2n' + j} \\ &\leq 1 + \frac{2j}{3j} \text{ (since } n' \geq j) \\ &= 5/3.\end{aligned}$$

Hence the theorem. ■

2) Modification of j to j': Another technique to handle the case in which $2n$ is not divisible by j is to find the smallest $j' > j$ such that j' divides $2n$ and use Algorithm Resource_Place to find a j'-adjacency placement in k^n.

Theorem 10: The ratio of the number of resource copies required by the above technique to the theoretical lower bound on the number of resource copies required for a j-adjacency placement in k^n is bounded by $(2n + j)/(3j)$ as k tends to ∞ for each n.

Proof: From Theorem 8, the number of resources placed by Algorithm Resource_Place in k^n for a j'-adjacency placement approaches $(j'k^n)/(2n + j')$ as $k \to \infty$. Hence, the ratio of the resource copies placed by the above technique to the theoretical lower bound approaches $(j'(2n + j))/(j(2n + j'))$ for large k. Since this is an increasing function of j' and since $j' \leq n$, the above ratio is bounded by $(2n + j)/(3j)$. ■

Since $5/3 < (2n + j)/3j$ when $2j < n$, it follows from Theorems 9 and 10 that the technique of modifying n to n' is better than the technique of modifying j to j' if $2j < n$; otherwise, the latter technique is better.

C. Approximate Placements when j Does Not Divide $2n, n < j < 2n$

The placement algorithms discussed so far are based on the assumption that either $1 \leq j \leq n$ or $j = 2n$. In this section, we consider the complementary case when $n < j < 2n$. Since from Theorem 1 there exist no perfect j-adjacency placements for this case, we need to consider approximate placement schemes.

As in Section IV–B, approximate placements can be obtained by either modifying n or j. In the first technique, find the smallest integer $n' > n$ such that $2n'$ is divisible by j. Since j exceeds n, this condition is satisfied if $n' = j$. Then, find a j-adjacency placement in k^j using Algorithm Resource_Place and modify it using techniques similar to those in Steps 10–15 of Algorithm Resource_Place. Using arguments similar to those given in the proof of Theorem 8, the number of resource copies in the resulting placement can be shown to be bounded by $(k^n/3) + k^n - (k-2)^n$.

The second technique for obtaining an approximate j-adjacency placement when $n < j < 2n$ is to construct a quasi-perfect $(2n-1)$-adjacency placement in k^n (see Theorem 3). The number of resource copies required for this method is bounded by $k^n/2$. Since this bound is greater than the bound of $(k^n/3) + k^n - (k-2)^n$ for the first technique, the first technique is superior for large k.

TABLE I
SUMMARY OF RESULTS DERIVED IN THIS PAPER

j	k ($k > 2^{\dagger}$)	# of Resource Copies	Comments
divides $2n$	$\exists q, r$, s.t. q divides k and $q^r - 1 = 2n/j$	$j \cdot k^n/(2n+j)$	perfect, optimal
$= (2n-1)$	for all k	$k^n/2$	quasi-perfect
divides $2n$, $1 \le j \le n$	$\forall q, r$, $q^r - 1 = 2n/j \Rightarrow$ q does not divide k	approaches $j \cdot k^n/(2n+j)$ as $k \to \infty$	approximate, asymptotically optimal
does not divide $2n$, $1 \le j \le n$	for all k	bounded by $\min\left\{\frac{(2n+j)}{3j}, \frac{5}{3}\right\} \cdot \frac{j \cdot k^n}{(2n+j)}$ as $k \to \infty$	approximate
$n < j < 2n$	for all k	bounded by $\min\{\frac{k^n}{2}, \frac{4k^n}{3} - (k-2)^n\}$ as $k \to \infty$	approximate

† Results for $k = 2$ are presented in [2].

V. CONCLUSIONS

In this paper, we studied the resource placement problem in k-ary n-cubes. We described algorithms for constructing j-adjacency placements in k^n. We showed that these algorithms can be extended to place j distinct types of resources in k^n. The key results are summarized in Table I. In addition, we showed the following:

- perfect (quasi-perfect) j-adjacency placements are impossible in k-ary n-cubes if $n < j < 2n (n < j < 2n-1)$;
- for $n < 22$. the necessary and sufficient condition for a perfect 1-adjacency placement is that k^n should be an integral multiple of $2n+1$;
- a sufficient condition for a perfect j-adjacency placement is that there exist integers q and r such that q divides k and $q^r - 1 = 2n/j$.

Finding the number of resource copies required for an optimal j-adjacency placement for a general k^n seems to be hard and at present it is an open problem.

The results derived in this paper for k-ary n-cubes can be extended to n-dimensional meshes in a straightforward fashion. Notice that an n-dimensional mesh does not have wraparound links that are present in the k-ary n-cube. Hence, a j-adjacency placement in k^n provides j adjacencies to all nodes in the *interior* of the mesh; however, some nonresource nodes on the boundary of the mesh might not be adjacent to j resource nodes because of the absence of wraparound links. Hence, some boundary nodes that do not have resources in k^n may have to receive resource copies in the corresponding mesh. In order to determine the exact location of resource nodes on the boundary, techniques similar to those presented in this paper can be used.

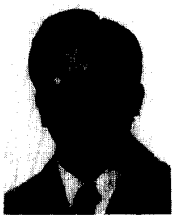
The results derived in this paper may also be generalized to the case in which each nonresource node is required to reach j resource nodes within h hops, $h > 1$, using partitioning techniques similar to those in [2]. However, these partitioning techniques do not always result in optimal placements. Finding the optimal number of resource copies required for an h-hop j-adjacency placement is an open problem.

ACKNOWLEDGMENT

The authors wish to thank F. Wang and Profs. K. Saluja and B. Bose for their constructive comments during this work.

REFERENCES

[1] L. N. Bhuyan and D. P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-33, pp. 323–333, Apr. 1984.
[2] H.-L. Chen and N.-F. Tzeng, "Fault-tolerant resource placement in hypercube computers," in *Proc. Int. Conf. Parallel Processing*, Aug. 1991, pp. 517–524.
[3] G.-M. Chiu and C. S. Raghavendra, "Resource allocation in hypercube systems," in *Proc. Distrib. Memory Comput. Conf.*, Apr. 1990, pp. 894–902.
[4] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. New York: Morgan-Kaufmann, 1992.
[5] M. Livingston and Q. Stout, "Perfect dominating sets," *Congressus Numerantium*, vol. 79, pp. 187–203, 1990.
[6] ——, "Distributing resources in hypercube computers," in *Proc. Third Conf. Hypercube Concurrent Comput. and Appl.*, Jan. 1988, pp. 222–231.
[7] A. L. N. Reddy, P. Banerjee, and S. G. Abraham, "I/O embedding in hypercubes," in *Proc. Int. Conf. Parallel Processing*, Aug. 1988, pp. 331–338.
[8] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. 37, pp. 867–872, July 1988.
[9] C. L. Seitz *et al.*, "The architecture and programming of the Ametek Series 2010 multicomputer," in *Proc. Third Conf. Hypercube Concurrent Comput. Appl.*, Jan. 1988, pp. 33–37.



Parameswaran Ramanathan received the B.Tech. degree from the Indian Institute of Technology, Bombay, India, in 1984, and the M.S.E. and Ph.D. degrees from the University of Michigan, Ann Arbor, in 1986 and 1989, respectively.

From 1984 to 1989 he was a Research Assistant in the Department of Electrical Engineering and Computer Science at the University of Michigan. At present, he is an Assistant Professor in the Department of Electrical and Computer Engineering, and in the Department of Computer Sciences at the University of Wisconsin, Madison. His research interests include the areas of real-time systems, fault-tolerant computing, distributed systems, and parallel algorithms.

Suresh Chalasani received the B.Tech. degree in electronics and communications engineering from J.N.T. University, Hyderabad, India, in 1984, the M.E. degree in automation from the Indian Institute of Science, Bangalore, in 1986, and Ph.D. degree in computer engineering from the University of Southern California in 1991.

He is currently an Assistant Professor of Electrical and Computer Engineering at the University of Wisconsin, Madison. His research interests include parallel architectures, parallel algorithms, and fault-tolerant systems.