

In *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications* (1988), ACM,
pp. 222–231.

DISTRIBUTING RESOURCES IN HYPERCUBE COMPUTERS

(Preliminary Version)

Marilynn Livingston

Department of Computer Science
Southern Illinois University
Edwardsville, IL 62026-1653

Quentin F. Stout¹

Dept. of Elec. Eng. and Comp. Sci.
University of Michigan
Ann Arbor, MI 48109-2122

ABSTRACT

Given a type of resource such as disk units, extra memory modules, connections to the host processor, or software modules, we consider the problem of distributing the resource units to processors so that certain performance requirements are met at minimal cost. Typical requirements include insisting that every processor is within a given distance of a resource unit, that every processor is within a given distance of each of several resources, and that every m -dimensional subcube contains a resource unit. The latter is particularly important in a multiuser system in which different users are given their own subcubes. In this setting, we also consider the problem of meeting the performance requirements at minimal cost when the subcube allocation system cannot allocate all possible subcubes and the requirements apply only to allocable subcubes. Efficient constructive techniques for distributing a resource are given for several performance requirements, along with upper and lower bounds on the total number of resource units required.

1 Introduction

The hypercube graph provides an attractive interconnection scheme for parallel computers. It has a low diameter, high fault-tolerance, high bandwidth, and scalability over a useful range. It is also homogeneous, meaning that no node is distinguished from any other. Homogeneity is quite useful in that hypercube algorithms are not required to have different code for processors of different types, or they may choose to utilize processors differently (for example, creating a tree structure to broadcast information, with one node acting as the root) and dynamically change which processors serve which role (for example, the first processor to determine an answer may start a broadcast tree). Another useful property of hypercubes is the

¹Partially supported by NSF grant DCR-8507851 and an Incentives for Excellence Award from Digital Equipment Corp.

fact that an n -dimensional hypercube (an n -cube) has many m -dimensional subcubes (m -subcubes). There are $\binom{n}{m}2^{n-m}$ such m -subcubes, which can be grouped as $\binom{n}{m}$ partitions of the n -cube into 2^{n-m} disjoint m -subcubes. With proper hardware and software, this permits multiple users on an n -cube, each having their own subcube.

When expensive resources such as disks are attached to an n -cube, however, these properties may be lost if it is too costly to attach a resource to each processor. To regain a semblance of homogeneity, and to reduce communication, it may be desirable to distribute the disks so that no processor is very from a disk. Similarly, to retain the multiuser capability offered by the multiple subcubes, it may be desirable to insure that each m -cube has at least a minimal number of disks. This paper considers such situations, assuming that there is some performance requirement that must be met with the objective of minimizing the number of resource units needed to satisfy the requirement. A variety of performance requirements will be considered, and upper and lower bounds are derived for the number of resource units needed to satisfy them. We also give constructive techniques for distributing the resource.

In Section 2 subcube requirements are analyzed and related to previous work on fault tolerance of n -cubes and k -independent sets. One important variation considered is where each processor can contain at most one resource unit, as opposed to arbitrarily many units. It is shown that this restriction can result in more resource units being required. Section 3 considers subcube requirements where the operating system cannot allocate all possible subcubes and the requirements need to be satisfied only for allocable subcubes. Section 4 considers satisfying multiple subcube requirements simultaneously. Section 5 considers distance requirements, relating them to dominating sets and error-correcting codes. Section 6 considers partitioning the nodes into classes and insuring that a performance requirement is satisfied for each class. Section 7 concludes with comments.

Throughout, Q_n will denote the n -cube, where the nodes of Q_n are labeled with all the n -bit strings and two nodes are adjacent if and only if their labels differ in exactly one position. A string $x_1 \dots x_n$, $x_i \in \{0, 1, *\}$, denotes the subcube of Q_n containing all processors with labels of the form $a_1 \dots a_n$, $a_i \in \{0, 1\}$, such that $a_i = x_i$ whenever $x_i \neq *$. The dimension of the subcube is the number of x 's which are equal to $*$. \lg is used to denote \log_2 , and \ln denotes \log_e . Due to space limitations, proofs of results will be omitted. This paper is part of a series dealing with various aspects of resource allocation and fault tolerance, and these results are only a preliminary announcement of work in progress.

2 Subcube Requirements

This section considers the problem of distributing a resource to the nodes of Q_n so that every m -subcube contains at least k units of the resource, using as little of the resource as possible. Notice that if the resource is divisible, then supplying each processor with $k/2^m$ units will satisfy the requirement, using only $k2^{n-m}$ total units. This is the minimal amount possible since each Q_n can be partitioned into 2^{n-m} disjoint m -subcubes, each of which must contain at least k units. However, it is often the case that the resource is a disk or some other indivisible object, and the problem of determining an optimal distribution satisfying the requirement is a difficult combinatorial problem. Throughout it is assumed that the resource is indivisible.

Let $C_s(n; m, k)$ denote the requirement that every m -subcube of Q_n contains at least k units of the resource, and let $C'_s(n; m, k)$ denote the same requirement with the additional restriction that there can be at most one resource unit at any processor. Denote by $R_s(n; m, k)$ and $R'_s(n; m, k)$ the minimum number of resource units needed to satisfy $C_s(n; m, k)$ and $C'_s(n; m, k)$, respectively.

If we interpret placing a resource unit at a node as making the node "faulty", then $R'_s(n; m, 1)$ is the minimum number of faulty nodes which could cause every m -subcube to be faulty. These numbers have

been extensively studied by the authors and others xx, and many of the techniques used to study them apply to general values of $R_s(n; m, k)$ and $R'_s(n; m, k)$. These techniques will be used repeatedly throughout this section, and the reader is referred to xx for an overview of them. Table 1 shows some values of $R_s(n; m, 1)$, repeating a table appearing in xx.

		n						
		0	1	2	3	4	5	6
	0	1	2	4	8	16	32	64
	1		1	2	4	8	16	32
	2			1	2	5	10	21
m	3				1	2	6	12
	4					1	2	6
	5						1	2
	6							1

Table 1: Values of $R_s(n; m, 1)$

Elementary Properties of R_s and R'_s

The following theorem lists several bounds on R_s and R'_s that follow from elementary considerations. The constraint that $m \geq \lceil \lg k \rceil$ is needed only to insure that R'_s is defined, and can be omitted for any bound involving only R_s . For (i), (ii), (iii), and (iv) the results are constructive in that any resource allocation(s) satisfying the condition(s) of the right hand side can be used in a straightforward manner to satisfy the conditions of the left hand side. For (v), an optimal distribution is obtained by having each processor receive one unit, for (vi) each processor with even parity should receive one unit, for (vii) any k processors should receive one unit, and for (viii) any k processors in $(n-1)$ -subcube can be chosen, along with their antipodal processors. Note that (iii) is a special case of (iv).

Theorem 2.1 For $n \geq m \geq \lceil \lg k \rceil$,

i) $R_s(n; m, 1) = R'_s(n; m, 1)$,

ii) $R_s(n; m, k) \leq R'_s(n; m, k)$,

iii) $R_s(n; m, k) \leq R_s(n; m, i) + R_s(n; m, m - i)$,

iv) $R_s(n; m, k) \leq R_s(n; p, i) + R_s(n; m, k - i2^{m-p})$, $p \leq m$,

v) $R_s(n; m, 2^m) = R'_s(n; m, 2^m) = 2^n$,

vi) $R_s(n; m, 2^{m-1}) = R'_s(n; m, 2^{m-1}) = 2^{n-1}$,

vii) $R_s(n; n, k) = R'_s(n; n, k) = k$,

viii) $R_s(n; n-1, k) = R'_s(n; n-1, k) = 2k$.

□

The inequality in part (iii) cannot always be replaced by equality since, for example, $R_s(4; 2, 1) = 5$ (Table 1), and $R_s(4; 2, 2) = 8$ (by part (vi)). In Theorem xx below we show that the inequality in part (ii) cannot be replaced by equality. One natural question is whether (iii) holds for R'_s , and in particular whether it is true that $R'_s(n; m, k) \leq kR'_s(n; m, 1)$? We can show that this is true whenever k is an integral power of 2, or in various other special cases, but do not know if it is always true.

In the next theorem inequalities (i), (ii), and (iii) are satisfied by both R_s and R'_s , but to conserve space they are stated only for R'_s . The results in (i) and (ii) are constructive in that any resource allocations which satisfy the right hand side can be directly used to generate an allocation which satisfies the left hand side. Similarly, in (iii) and (iv), any resource allocation for the left hand side satisfies the constraints for each of the components of the right hand side.

Theorem 2.2 For $n \geq m \geq \lceil \lg k \rceil$,

$$i) R'_s(n; m, k) \leq 2R'_s(n-1; m-1, \lceil k/2 \rceil),$$

$$ii) R'_s(n; m, k) \leq R'_s(n-1; m-1, k) + R'_s(n-1; m, k),$$

$$iii) R'_s(n; m, k) \geq \max\{2R'_s(n-1; m, k), R'_s(n; m+1, 2k)\},$$

$$iv) R_s(n; m, k) \geq R_s(n-1; m-1, k).$$

□

The *weight* of a processor is the total number of 1's in its label. Suppose some distance d and integer a , $0 \leq a < d$, are chosen, and one resource unit is attached to each processor with weight congruent to $a \pmod d$. The number of units used is $\sum \binom{n}{i} : i \equiv a \pmod d, 0 \leq i \leq n$, which for fixed d is minimized when $a = \lfloor (n-d)/2 \rfloor$. Any given m -subcube contains exactly $\sum \binom{m}{i} : i \equiv b \pmod d, 0 \leq i \leq m$ units for some b which depends on a and the subcube chosen, and hence any given m -subcube contains at least $K(m, d) = \sum \binom{m}{i} : i \equiv \lfloor (m-d)/2 \rfloor \pmod d, 0 \leq i \leq m$. Therefore we have:

Theorem 2.3 For $n \geq m \geq d+1$,

$$R'_s(n; m, K(m, d)) \leq \sum_{i \equiv \lfloor (n-d)/2 \rfloor \pmod d} \binom{n}{i} \leq 2^n/d. \quad \square$$

The next result shows that the R_s and R'_s functions are not the same, and hence it can be that allowing only one resource unit per processor will increase the number of units required.

Fact $R_s(5; 2, 3) < R'_s(5; 2, 3)$.

Proof: By inequality (iv) of Theorem xx, $R_s(5; 2, 3) \leq R_s(5; 1, 1) + R_s(5; 2, 1) = 16 + 10$ (see Table 1). Let S denote a set of $R'_s(5; 2, 3)$ nodes of Q_5 satisfying $C_s(5; 2, 3)$. Consider the four disjoint 3-subcubes of Q_5 given by $A = 00***$, $B = 01***$, $C = 10***$, and $D = 11***$. Each of these must have at least 6 elements in common with S to satisfy $C_s(3; 2, 3)$. On the other hand, if $A \cap S$ contains only 6 elements then the nodes of $A - S$ must be diagonally opposite. Both B and C must then be contained in S in order to satisfy $C_s(5; 2, 3)$. It follows that $R'_s(5; 2, 3) \geq 28$. □

Multiply t -Independent Sets

The function $R_s(n; m, 1)$ is intimately connected with t -independent sets [xx]. We extend these ideas to the

notion of multiply t -independent sets and find an analogous relation to $R_s(n; m, k)$. A family F of subsets of a set X is (k, t) -independent if for every pair of subsets U and V of F such that $|U| + |V| = t$ there are at least k elements of X common to all sets in U and which are in no set in V . A (k, t) -independent family F with n members can be used to construct a multiset S of n -bit strings with the property that if a resource unit is placed at each processor with its label in S (taking multiplicities into account), then the resulting distribution satisfies $C_s(n; n - t, k)$. To construct S , fix an ordering on the elements of X and let M be the $r \times n$ matrix with columns equal to the characteristic functions of the sets in F . The rows of M are the elements of S . Moreover, every multiset of processor labels listing a distribution of resources satisfying $C_s(n; n - t, k)$ must arise in this manner. This yields the following result.

Theorem 2.4 Let $F(r; k, t)$ denote the maximum size of a (k, t) -independent family of subsets of a set of r elements. Then $R_s(n; m, k) = \min\{r : F(r; k, n - m) > n\}$. \square

Using this result and Theorem xx, we can show the following.

Theorem 2.5 Fix $k \geq 1$.

- i) For $n \geq \lceil \lg k \rceil + 2$, $R_s(n; n - 2, k) = \lg(n) + O(\log \log n)$,
- ii) For $n \geq m + 2 \geq \lceil \lg k \rceil + 2$, $R_s(n; m, k) \geq 2^{n-m-2}[\lg(m + 2) + O(\log \log n)]$,

where the implied constants inside the O's are dependent on k . \square

Asymptotic Bounds

For fixed $n - m$, an asymptotic upper bound for $R_s(n; m, k)$ and $R'_s(n; m, k)$ can be established which is independent of k .

Theorem 2.6 For fixed $n - m \geq 1$ and $m \geq \lceil \lg k \rceil$, $R'_s(n; m, k) < 2^{n-m}[\ln \binom{n}{n-m} + 2(n - m) \ln(2)]$. \square

3 Subcube Requirements for Allocable Subcubes

In practice, for a multi-user or multi-tasking hypercube, the routines which allocate a requested subcube examine the availability of only a subset of the subcubes of the requested dimension. Currently the *buddy system* is the predominate allocation scheme, allocating only m -subcubes of the form $a_1 \dots a_{n-m} * \dots *$. If A is a given allocation scheme, let $C_s(A; n; m, k)$ denote the requirement that every m -subcube of Q_n that is allocable by A contain at least k resource units, and let $C'_s(A; n; m, k)$, $R_s(A; n; m, k)$, and $R'_s(A; n; m, k)$ be defined accordingly.

In addition to the buddy system, other systems that have been suggested include using multiple buddy systems, Gray-coded buddy systems, and multiple Gray-coded buddy systems. In the simplest form of the multiple buddy system, the *doubly indexed buddy system*, m -subcubes of the form $* \dots * a_{m+1} \dots a_n$ are also allocated. In general, multiple buddy systems use multiple permutations of the index set $\{1 \dots n\}$, and for each permutation π allocate m -subcubes by fixing index positions $\pi(1) \dots \pi(n - m)$ and varying the remaining m indices. For *Gray-coded buddy systems*, let g_t denote the binary reflected Gray code map from $\{0 \dots 2^t - 1\}$ to t -bit strings. The m -subcubes allocated are those pairs of $(m - 1)$ -subcubes of the form $\{a_1 \dots a_{n-m+1} * \dots *, b_1 \dots b_{n-m+1} * \dots *\}$, where $g_{n-m+1}^{-1}(a_1 \dots a_{n-m+1})$ and $g_{n-m+1}^{-1}(b_1 \dots b_{n-m+1})$ are consecutive mod 2^{n-m+1} . Multiple Gray-coded buddy systems combine Gray codes with the multiple index permutations of multiple buddy systems.

In Q_n , the buddy system allocates only 2^{n-m} m -subcubes, and the doubly indexed buddy system and Gray-coded buddy system allocate 2^{n-m+1} ($n > m$). This is significantly less than the $\binom{n}{m}2^{n-m}$ m -subcubes available, and the resource allocation problems are greatly simplified.

Theorem 3.1 *For each of the buddy system, doubly indexed buddy system, and Gray-coded buddy system, $R_s(A; n; m, k) = R'_s(A; n; m, k) = k2^{n-m}$ for $n \geq m \geq \lceil \lg k \rceil$. \square*

4 Multiple Subcube Requirements

Rather than just requiring that each m -subcube have a certain minimal amount of resource, it may be that one also needs to insure that larger subcubes have even more of the resource. Tasks run on larger subcubes presumably need more resources, and while it is true that if each m -subcube has at least k resource units then each $(m+i)$ -subcube has at least $k2^i$ resource units, it may be that the minimal requirements for larger subcubes grow faster than this rate. Suppose a resource is to be distributed so that it satisfies requirements $C_s(n; m_1, k_1)$ and $C_s(n; m_2, k_2)$. Let $R_s(n; (m_1, k_1), (m_2, k_2))$ denote the least number of resource units required to satisfy both requirements, and define $R'_s(n; (m_1, k_1), (m_2, k_2))$ accordingly. If $m_1 \leq m_2$ it is easy to see that

$$R_s(n; (m_1, k_1), (m_2, k_2)) \leq R_s(n; m_1, k_1) + R_s(n; m_2, k_2 - k_1 2^{m_2 - m_1}).$$

On the other hand, the corresponding inequality for R'_s fails infinitely often.

Theorem 4.1 *For $n \geq 5$, $R'_s(n; (1, 1), (2, 3)) > R'_s(n; 1, 1) + R'_s(n; 2, 1)$. \square*

The problem of satisfying multiple requirements can be extended to more than 2, and can also be combined with the notion of considering only allocable subcubes. For the some simple allocation schemes, the number of resource units needed are straightforward to calculate.

Theorem 4.2 *Let $n, (m_1, k_1), (m_2, k_2), \dots, (m_t, k_t)$ be such that $m_1 \leq \dots \leq m_t \leq n$ and $m_i \geq \lceil \lg k_i \rceil$ for $1 \leq i \leq t$. For the buddy system, doubly indexed buddy system, and Gray-coded buddy systems,*

$$R_s(A; n; (m_1, k_1), \dots, (m_t, k_t)) = R'_s(A; n; (m_1, k_1), \dots, (m_t, k_t)) = c_t 2^{n-m_t},$$

where c_i is defined recursively by $c_1 = k_1$ and $c_{i+1} = \max\{c_i 2^{m_{i+1} - m_i}, k_{i+1}\}$. \square

5 Distance Requirements

Instead of requiring that each m -subcube have a suitable number of resource units, one may instead need to insure that each node is within some distance δ of a suitable number of resource units. Let $C_d(n; \delta, k)$ denote the requirement that every node of Q_n be within δ of k or more resource units, and define $C'_d(n; \delta, k)$, $R_d(n; \delta, k)$, and $R'_d(n; \delta, k)$ correspondingly. When $k = 1$, a distribution of the units satisfying $C_d(n; \delta, 1)$ is a δ -dominating set of Q_n . Constructions of optimal distributions of some δ -dominating sets are given with the aid of error-correcting codes. For example, perfect single error-correcting codes show that $R_d(2^t - 1; 1, 1) = 2^{2^t - 1 - t}$.

	n						
	0	1	2	3	4	5	6
0	1	2	4	8	16	32	64
1	1	1	2	2	4	7	12
2	1	1	1	2	2	2	4
δ 3	1	1	1	1	2	2	2
4	1	1	1	1	1	2	2
5	1	1	1	1	1	1	2
6	1	1	1	1	1	1	1

Table 2: Values of $R_d(n; \delta, 1)$

6 Partitioning a Resource

Suppose each processor contains a unit of resource, and these are to be partitioned into different classes. For example, the resource may be memory, and it may be that multiple copies of different software modules are to be stored throughout the hypercube, with each processor containing one copy of one module. All the processors storing the same module form a single class. As another example, the resource may be a connection to another device, with all processors connected to the same type of device forming a class. Or it may be that the resource is a connection to a processor in another n -cube, where the collection of n -cubes form the nodes of a hypercube graph and the classes are determined by the dimension that the link corresponds to in this large hypercube. This situation could be considered as “cube-connected cubes”, similar to the “cube-connected cycles”. It can arise when the degree of the node is fixed, perhaps by pin limitations, and a good approximation of a hypercube of larger degree is desired.

In resource partitioning situations one may need to insure either that every processor is within a certain distance of a processor in each class, or that each m -subcube contains at least one processor in each class. In this latter case one may also invoke the notion of allocable subcubes. Let $P_s(n; m)$ denote the largest number of classes that Q_n can be partitioned into such that every m -subcube contains at least one processor of each class, and let $P_d(n; \delta)$ and $P_s(A; n; m)$ be defined accordingly.

7 Comments

We have considered some of the problems that arise when one is trying to satisfy a resource distribution requirement using a minimal number of resource units.