In *Proc. 4th Conf. on Hypercube Concurrent Computers and Applications* (1989), ACM, pp. 59–66.

PARALLEL ALLOCATION ALGORITHMS FOR HYPERCUBES AND MESHES

(Preliminary Version)

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

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

Abstract

We consider the problem of subsystem allocation in the mesh, torus, and hypercube multicomputers. Although the usual practice is to use a serial algorithm on the host processor to do the allocation, we show how the free and non-faulty processors can be used to perform the allocation in parallel. The algorithms we provide are dynamic, require very little storage, and work correctly even in the presence of faults.

For the 2-dimensional mesh and torus with n processors, we give an optimal $\Theta(\sqrt{n})$ time algorithm which identifies all rectangular subsystems that are not busy and not faulty. For the d-dimensional mesh and torus of size $n = m \times m \times \cdots \times m$, we show how to find all submeshes of dimensions $k \times k \times \cdots \times k$, for all $k \leq m$, in optimal $\Theta(dn^{1/d})$ time.

Since the number of subcubes in a hypercube of dimension d is 3^d, the current practice is to allocate only a fraction of the possible subcubes, which degrades the fault tolerance and dynamic allocation ability of the system. We consider two approaches to this problem. In one approach, we limit the dimensions of the subcubes to be allocated, and show, for fixed q, how to determine all non-faulty and non-busy subcubes of dimension $d - q$ in a hypercube of dimension d in time $\Theta(d)$. The second approach involves allocating only a subset of the possible subcubes in all dimensions. We give optimal parallel algorithms for implementing several previously suggested allocation schemes of this type, including single and multiple versions of buddy, Gray-coded-buddy, and k-cube buddy systems. The parallel versions of these are significantly faster than the known serial allocation algorithms, and they provide a significant improvement in the fault tolerance of the system. We also introduce a new allocation system, the cyclical buddy system, which has a simple, efficient parallel implementation but which does not naturally arise as a serial allocation system.

1 Introduction

In MIMD parallel computers containing large numbers of processors it is desirable to be able to allocate subsystems. This capability is needed in multiuser environments such as those provided by the Intel or NCUBE series of hypercubes, and also in single user systems that allow multiple subtasking. In addition, subsystem allocation can be used to increase fault tolerance by allocating only subsystems with nonfaulty processors. Unfortunately, the existence of a large number of subsystems in a network results in an allocation problem that is computationally intensive, if one tries to allocate all possible subsystems. Thus, in practice, only a small fraction of the possible subsystems are checked for availability.

One consequence of using a scheme that allocates only a part of the possible subsystems is that a request for a particular size may be refused even when one is available. To see how this affects the fault tolerance of the system, consider the performance of the buddy system in allocating q-dimensional subcubes in a d-dimensional hypercube. This is the system currently used on the NCUBE hypercubes, and it is discussed in more detail in Section 3. While there is a total of $\binom{d}{q}2^{d-q}$ subcubes of dimension q, the standard buddy system allocates only those q-dimensional subcubes determined by fixing the high-order $d - q$ address bits. There are only 2^{d-q} of these and each processor is in exactly one such subcube. Now, with $n = 2^d$ and $m = 2^q$, let $B_w(n, m)$ denote the least number of faulty (or busy) processors which make all the buddy system subcubes consisting of m processors unavailable for allocation in a hypercube of n processors, and let $K_w(n, m)$ denote the analogous number for the case of complete allocation. That is, $K_w(n, m)$ is the least number of faulty processors which cause all subcubes of m processors to be unavailable. Taking $d = 20$ and $q = 18$ we see that the buddy system allocates only 4 of the possible 760 subcubes of dimension 18. Also, it is straightforward to check that $B_w(2^{20}, 2^{18}) = 4$ and $K_w(2^{20}, 2^{18}) = 8$. Extending this example, we find that $B_w(2^d, 2^{d-2}) = 4$ while $K_w(2^d, 2^{d-2}) = \log d + \Theta(\log\log d)$ ([BeSi, GHLS, KlSp]). Thus, the buddy system becomes progressively less fault tolerant as d increases. Further, if we consider the expected case behavior, where $B_e(n, m)$ and $K_e(n, m)$ denote the corresponding numbers for the situation in which the faulty (or busy) processors are distributed independently and uniformly throughout the hypercube, it is shown in [LiSt] through simulation that $B_e(2^{20}, 2^{18}) \approx 8.1$ and $K_e(2^{20}, 2^{18}) \approx 24.6$. Thus, in the worst case, we suffer a 50% decrease and, in the expected case, a decrease of 67% in the fault-tolerant allocation ability of the system.

*Partially supported by National Science Foundation grant CCR-8808839

†Partially supported by National Science Foundation grant DCR-8507851 and an Incentives for Excellence Award from Digital Equipment Corporation

To increase the number of subsystems which can be allocated, without increasing the time required to do the allocation, we abandon the current practice of having only the host computer decide the allocation, and instead utilize the parallel computer. In this paper, we give optimal algorithms to allocate subsystems in parallel for the hypercube, mesh, and torus. We use only free and nonfaulty processors to determine the available subsystems, thereby avoiding any interference with currently running tasks and assuring that the process works correctly even in the presence of faults. Moreover, apart from the transmission of final availability information to the host, only neighbor-to-neighbor communication among the processors is used.

Our allocation algorithms actually find every available subsystem that is allowed under the given allocation scheme. This information allows choices to be exercised to minimize fragmention of the whole system. For example, if a subsystem of size k were requested, a desirable choice among several available ones would be one that is not contained in any available subsystem of size greater than k. A further advantage of our parallel allocation algorithms is that they are dynamic and require very little storage. By determining locally which processors are nonfaulty and free at the time of the request, there is no need to maintain this information centrally. These allocation algorithms should be of considerable practical interest, particularly in large systems, for not only are they efficient, most of them are relatively simple to implement, and a significant improvement in fault tolerance is attained by their use.

In Section 2 we consider the allocation problem for the d-dimensional mesh and torus of size $n = m^d$ and dimensions $m \times m \times \cdots \times m$. We find all submeshes of dimensions $k \times k \times \cdots \times k$, for all $k \leq m$, in optimal $\Theta(dn^{1/d})$ time. A simple modification of this algorithm determines, for a given k, the available subsystems of dimensions $k \times k \cdots \times k$ in time $\Theta(dk)$. Furthermore, when $d = 2$, our algorithm finds, for each non-faulty and non-busy processor, the size of the largest non-faulty and non-busy rectangular submesh for which the given processor appears in the upper left-hand corner.

We address the problem of allocating q-dimensional subcubes in a d-dimensional hypercube in Section 3 and show how to implement previously suggested schemes such as those based on the buddy system, the gray-coded buddy system, multiple-buddy and multiple-gray-coded buddy systems, all in time $\Theta(d)$. We also introduce a new allocation scheme, called the cyclical buddy system, which arises naturally from our parallel implementation of the buddy system, and show how to implement this system in optimal $\Theta(d)$ time, as well.

The problem of implementing the complete allocation scheme for hypercubes is considered in Section 3.4. Since there are 3^d subcubes in a hypercube of dimension d, determining all fault-free subcubes of all dimensions at the time of the request is impractical. Allocating only the subcubes of dimension $d - q$, for fixed q, is still impractical if we use a naive serial algorithm such as one which checks each of the 2^{d-q} pe's in all $\binom{d}{q}2^q$ subcubes, for this requires at least $\Theta(\binom{d}{q}2^d)$ time. Much more efficient serial algorithms are possible which require significantly less time, but they typically require extensive storage and still have poor worst-case times. At the least, any serial algorithm which must check the current fault status of processors must have a worst-case time of $\Omega(2^d)$. Here we give a parallel algorithm which finds, for fixed q, all subcubes of dimension $d - q$ in $\Theta(d)$ time.

Complete allocation in hypercubes may not be the method of choice when d, q, and $d - q$ are all large, or when allocation of all sizes is necessary. For such cases, we recommend the use of the *k-cube buddy system* first introduced in [LiSt]. For fixed k, this system allocates q-dimensional subcubes in which the last $d - k$ bits are arbitrary and the first $d - q + k$ bits are the nodes of a k-subcube of a $d - q + k$-dimensional cube. In Section 3.5 we give a $\Theta(d)$ algorithm for the allocation of all subcubes allowed by this system. With almost no increase in allocation time, our implementation of the k-cube buddy system offers a significant improvement in fault tolerance over the buddy and gray-coded buddy systems currently in use. To contrast the behavior of the k-cube buddy system with the buddy system, for example, let us take $k = 2$, $d = 20$, $q = 18$, and let $QB_w(n, m)$ and $QB_e(n, m)$ denote the quantities analogous to $B_w(n, m)$ and $B_e(n, m)$ for the 2-cube buddy system. Although $QB_w(2^{20}, 2^{18}) = 5$ which is a small improvement over $B_w(2^{20}, 2^{18})$, we have $QB_e(2^{20}, 2^{18}) \approx 12.8$ [LiSt], which represents a 50% improvement over the expected case behavior of the single buddy system.

Throughout this paper we will assume that each processing element (pe) in each of the networks under consideration has a unique identification number (id), and that each pe knows its own id. Further, we assume each pe has a fixed number of registers, each of size $\Theta(\log n)$, and can perform standard arithmetic and Boolean operations on the contents of these registers in unit time. In addition, we assume that each pe can send (receive) a word of data to (from) one of its neighbors in unit time, and that it can determine which of its neighbors, if any, are faulty. Finally, we assume that there is some host or controller which sends a message to all processors initializing the process, and that each pe can communicate back to the host.

A processing element will be termed *available* if it is neither busy nor faulty. Our algorithms are performed by all available processors, and are designed so that no messages are sent to, nor expected from, unavailable processors. While the algorithm descriptions appear synchronous, they are to be run asynchronously, with each processor waiting for the appropriate messages from its nonfaulty neighbors. Timing will be in terms of the slowest available processor. We consider only the time to locate subsystems, not the time used by the host to pick among them since that is dependent on the process used to make the selection.

2 Mesh and Torus Allocation

Before describing the allocation algorithms, we first introduce some notation for the mesh and torus.

The *one-dimensional mesh*, or linear array, of size n consists of n processing elements arranged in a line with adjacent nodes connected. We denote this array by $\mathcal{M}_1(n)$ and use P_i to denote the ith pe, $1 \leq i \leq n$. The *two-dimensional*

Algorithm 2.1 (1-Dimensional Mesh or Torus)
Each available pe P_i has integer variables s_i and a, and executes the following algorithm.

1 s_i := 1.

2 *For* a:=2 *to* n *do*

3 *If* P_{i-1} *available then send* s_i *to* P_{i-1}.

4 *If* P_{i+1} *available then*

5 *Receive* s_{i+1}.

6 $s_i := s_{i+1} + 1$.

---

mesh of size $n = m^2$, denoted $\mathcal{M}_2(n)$, consists of n pe's arranged in an $m \times m$ two-dimensional grid. For $1 \leq i, j \leq m$, the pe in row i and column j will be denoted by P_{ij} and is connected to $P_{i\pm1,j}$ and $P_{i,j\pm1}$, when they exist. In general, a mesh of dimension d and size $n = m^d$, denoted $\mathcal{M}_d(n)$, is made up of n pe's arranged in an $m \times \cdots \times m$ grid of dimension d. Each pe has an id which is a d-tuple representing its coordinates in this grid. Two pe's are connected if and only if their coordinates differ by one in exactly one position, that is, for $1 \leq i_1, i_2, \ldots, i_d \leq m$, processing element $P_{i_1,i_2,\ldots,i_d}$ is connected to $P_{j_1,j_2,\ldots,j_d}$ provided $\sum_{t=1}^{d}(i_t - j_t)^2 = 1$.

The d-dimensional *torus* of size $n = m^d$, denoted by $\mathcal{T}_d(n)$, is the d-dimensional mesh $\mathcal{M}_d(n)$ enhanced with *wrap-around* connections. Two pe's will have a wrap-around connection if their id's are the same in all but one component and in that one component one pe's id has value 1 and the other has the value m. For example, $\mathcal{T}_1(n)$ is a ring and $\mathcal{T}_2(n)$ is a cylinder open on both ends.

In all of our algorithms, the components of a pe's id in the torus are to be interpreted modulo m, while, in the mesh, any reference to a nonexistent pe is treated as a reference to a nonavailable pe.

2.1 Dimension One

A *subsystem* of size t of $\mathcal{M}_1(n)$ or $\mathcal{T}_1(n)$ is a string of t pe's of the form $P_i, P_{i+1}, \ldots, P_{i+t-1}$. Thus, there is a total of $n-t+1$ subsystems of size t in $\mathcal{M}_1(n)$ and n subsystems of size t in $\mathcal{T}_1(n)$ for $1 \leq t \leq n-1$. We will consider processor P_i to be the *leader* of a subsystem of size t, for some $t < n$, provided that each of $P_i, P_{i+1}, \ldots, P_{i+t-1}$ is available but P_{i+t} is not. We say P_1 is the leader of the subsystem of size n in case all pe's are available.

The allocation algorithm for the one-dimensional mesh and torus, Algorithm 2.1, results in each available pe determining the size of the subsystem for which it is the leader. This information can then be used by a variety of algorithms to choose which of the available subsystems should be used in satisfying the request.

At the end of each iteration of the for-loop, if $P_i, \ldots, P_{i+a-1}$ are available then s_i equals a, otherwise it equals the size of the subsystem for which P_i is the leader. Algorithm 2.1 has worst case time complexity $\Theta(n)$, and is therefore asymptotically optimal for both the mesh and torus since the communication diameter of $\mathcal{M}_1(n)$ is $n-1$ and of $\mathcal{T}_1(n)$ is $\lfloor n/2 \rfloor$. Note that if we are interested only in determining if there are any subsystems of size k, then we need only have the for-loop run from 2 to k, reducing the time to $\Theta(k)$.

Algorithm 2.2 (2-Dimensional Mesh or Torus)
Each available pe $P_{i,j}$ has integer variables $s_{i,j}, t_{i,j}, u_{i,j}$ and a, and executes the following algorithm.

1 *Compute* $s_{i,j}$ *using the 1-dimensional algorithm, as if each column were a 1-dimensional computer.*

2 $u_{i,j} := s_{i,j}$; $t_{i,j} := 1$;

3 *For* a :=2 *to* $\sqrt{n}$ *do*

4 *If* $P_{i,j-1}$ *available then send* $u_{i,j}$ *to* $P_{i,j-1}$.

5 *If* $P_{i,j+1}$ *available then*

6 *Receive* $u_{i,j+1}$;

7 $u_{i,j} := \min(u_{i,j}, u_{i,j+1})$

8 *else* $u_{i,j}$:= 0;

9 $t_{i,j} := \max(t_{i,j}, \min(a^2, u_{i,j}^2))$.

2.2 Dimension Two

Let n and t be squares of integers. By a *subsystem of size* t of $\mathcal{M}_2(n)$ or $\mathcal{T}_2(n)$ we will mean a square subgrid isomorphic to $\mathcal{M}_2(t)$. The processor id's of a subsystem of size t form a set of the form $\{(a+i, b+j) : 0 \leq i, j < \sqrt{t}\}$ for some a, b in the range $[1..\sqrt{n}]$. There are $(\sqrt{n} - \sqrt{t} + 1)^2$, subsystems of size t in $\mathcal{M}_2(n)$, and n^2 systems of size t in $\mathcal{T}_2(n)$, for $1 \leq t \leq n$. A processor $P_{a,b}$ will be called a *leader* of a subsystem of size t, for some $t < n$, provided (1) all processors with id's in the set $\{(a+i, b+j) : 0 \leq i, j < \sqrt{t}\}$ are available but (2) not all processors in the set $\{(a+i, b+j) : 0 \leq i, j \leq \sqrt{t}\}$ are available. Processor $P_{1,1}$ is considered the leader of the system of size n if all the pe's are available.

Algorithm 2.2 proceeds by first finding, for each pe, the largest 1-dimensional subsystem along the first coordinate for which that pe is a leader. We then use the fact that a processor is a leader of a system of size at least t provided it and each of the $t-1$ processors following it along the second dimension are leaders of 1-dimensional systems of size t or greater. At the end of each iteration of the for-loop, if processor $P_{i,j}$ is the leader of a subsystem of size a^2 or greater, then $t_{i,j}$ equals a^2, otherwise it equals the size of the subsystem for which $P_{i,j}$ is the leader.

It is straightforward to show that Algorithm 2.2 has $\Theta(\sqrt{n})$ time and thus is optimal.

Notice that at the end of each iteration of the for-loop, $u_{i,j}$ is the width of the largest available rectangle with upper left corner at $P_{i,j}$ and height a. The number of processors in this rectangle is $u_{i,j}a$, so by changing line 9 to

$$t_{i,j} := \max(t_{i,j}, u_{i,j} * a),$$

one can find the largest available rectangle with $P_{i,j}$ as its leader in time $\Theta(\sqrt{n})$ as well.

2.3 Higher Dimensions

For the case of the d-dimensional mesh and torus, subsystems of size t and leaders of subsystems of size t are defined analogously to the case $d = 2$. The method of computation used in Algorithm 2.2 can be used as a model for higher dimensions, building up information dimension by dimension. In this manner, the leaders and sizes of the subsystems of $\mathcal{M}_d(n)$ and $\mathcal{T}_d(n)$ can be found in $\Theta(dn^{1/d})$ time. If one is only interested in determining if there are any subsystems of size k, then the total time can be reduced to $\Theta(dk^{1/d})$ by changing all for-loops to run from 2 to k.

3 Hypercube Allocation

Let $\mathcal{Q}(2^d)$ denote a d-dimensional hypercube with 2^d pe's. Each pe has a unique binary d-tuple as its id number, and two pe's are connected if and only if their id numbers differ in precisely one position. Now, suppose S is a set of q distinct integers from the interval $[1..d]$ and $(b_1, b_2, \ldots, b_d)$ is a fixed binary d-tuple. The pe's with id numbers in the set $\{(c_1, c_2, \ldots, c_d) : c_i = b_i \text{ for } i \notin S, 1 \leq i \leq d\}$ form a q-dimensional subcube which will be denoted by $a_1 a_2 \ldots a_d$, where $a_i = *$ for $i \in S$ and $a_i = b_i$ otherwise. A q-dimensional subcube will be called a *q-subcube*. There are a total of $\binom{d}{q}2^{d-q}$ q-subcubes in $\mathcal{Q}(2^d)$ and each pe is in exactly $\binom{d}{q}$ q-subcubes. Moreover, since each subcube is uniquely represented as a string of length d over the alphabet $\{0, 1, *\}$, we see that $\mathcal{Q}(2^d)$ has 3^d subcubes.

Thus, for large d, allocation of all possible subcubes of $\mathcal{Q}(2^d)$ becomes computationally intensive, particularly in the presence of faults and when both dynamic allocation and deallocation is allowed. In this section we will consider two approaches to alleviate this problem. The first approach, discussed in Sections 3.1, 3.2, 3.3, and 3.5, is to allocate only a subset of the possible subcubes in all the dimensions. We provide $\Theta(d)$ time parallel algorithms to perform the allocation for each of the systems under consideration. In the second approach we consider allocation of all subcubes of dimension $d - q$, for fixed q, and give a $\Theta(d)$ algorithm here as well.

Consistent with our presentation of algorithms in Section 2, each step is to be carried out in parallel by each available pe P_α, where α denotes an arbitrary binary d-tuple. The value of the ith component of α will be denoted by $\alpha(i)$ and the neighbor of P_α along dimension k will be represented by $P_{\alpha,k}$. In addition, processor P_α will be called the *leader* of the subcube $a_1 a_2 \ldots a_d$ provided $\alpha(i) = a_i$ for each i such that $a_i \in \{0, 1\}$ and $\alpha(i) = 0$ otherwise.

3.1 Buddy Systems

For a given dimension q, the standard *single buddy system* allocates only q-subcubes in $\mathcal{Q}(2^d)$ of the form $a_1 a_2 \ldots a_{d-q} * \ldots *$, that is, the high-order $d - q$ bits of the id numbers are fixed. More generally, each permutation π of the integers $1, 2, \ldots, d$ gives rise to a single buddy system B_π which allocates only q-subcubes of the form $a_1 a_2 \ldots a_d$ where, $a_{\pi(i)} = *$ for $d - q + 1 \leq i \leq d$. The standard buddy system corresponds to the identity permutation. Notice that, for each permutation π, B_π allocates only 2^{d-q} of the available $\binom{d}{q}2^{d-q}$ q-subcubes.

Algorithm 3.1 (Buddy System)
π is a given permutation of $1, 2, \ldots, d$. Each available P_α has integer variables s_α, z_α, j,and k.

1 s_α:=0; z_α:=0;.

2 *If* $\alpha = (0, 0, \ldots, 0)$ *then* $z_\alpha := d$

3 *else while* $\alpha(\pi(z_\alpha + 1)) = 0$ *do* $z_\alpha := z_\alpha + 1$.

4 *For j := 1 to* z_α *do*

5 $k := \pi(d + 1 - j)$.

6 *If* $P_{\alpha,k}$ *available then*

7 *Receive* $s_{\alpha,k}$.

8 *If* $(s_{\alpha,k} = j - 1$ *and* $s_\alpha = j - 1)$ *then* $s_\alpha := j$.

9 *If* $(z_\alpha < d$ *and* $P_{\alpha,\pi(d-z_\alpha)}$ *available) then*

10 *Send* s_α *to* $P_{\alpha,\pi(d-z_\alpha)}$.

The buddy system is attractive since it is easy to implement. Moreover, if only static allocation is considered and the presence of faults is ignored, the buddy system is an optimal allocation strategy in the sense that it fails to grant a request only if there is an insufficient number of available nodes to satisfy the request. In dynamic allocation and deallocation, however, it is no longer optimal. Studies of the behavior of subcube allocation in this dynamic situation have been made in [ChSh, DuHa] for the purpose of evaluating various policies governing the selection of which subcube to allocate when there are several available subcubes.

One of the problems in using a serial algorithm to implement any allocation system is in maintaining the availability information when faults occur. This is no longer a problem in our parallel implementation. Our algorithm for the single buddy system proceeds by recursive halving, determining the sizes and leaders of all of the available subcubes allowed by the buddy system. At the end of line 3 of Algorithm 3.1, z_α is the dimension of the largest subcube for which P_α would be the leader in a completely available hypercube using the buddy allocation system. At the end of the algorithm, s_α is the dimension of the largest available subcube for which P_α is the leader, among those subcubes allocable under the given buddy system.

Variations of the single buddy system have been suggested which allocate the union of several single buddy systems. For example, an orthogonal *double buddy system* allocates q-subcubes in $\mathcal{Q}(2^d)$ of the form $a_1 a_2 \ldots a_{d-q} * \ldots *$ together with the form $* \ldots * b_{q+1} b_{q+2} \ldots b_d$. Thus, for $q < n$, the orthogonal buddy system allocates twice the number of sub-

Algorithm 3.2 (Gray-Coded Buddy System)
π is a given permutation of $1, 2, \ldots, d$. Each available P_α has integer variables s_α, j, k, and an integer array $D[1..d]$.

1 $s_\alpha := 0$.

2 *For j := 1 to d do*

3 $\quad k := \pi(d + 1 - j)$.

4 $\quad$ *If $P_{\alpha,k}$ available then*

5 $\quad\quad$ *Send s_α to $P_{\alpha,k}$.*

6 $\quad\quad$ *Receive $s_{\alpha,k}$.*

7 $\quad\quad D(d + 1 - j) := s_{\alpha,k}$.

8 $\quad\quad$ *If $s_\alpha = j - 1$ and $s_{\alpha,k} = j - 1$ then $s_\alpha := j$.*

9 $\quad$ *else $D(d + 1 - j)$:= -1.*

Algorithm 3.3 (Cyclical Buddy System)
π is a given permutation of $1, 2, \ldots, d$. Each available P_α has integer variables s_α, t_α, z_α, j, and k.

1 $s_\alpha := 0$; $t_\alpha := 0$; $z_\alpha := 0$.

2 *For $j := 0$ to $(2d - 3)$ do*

3 $\quad k := \pi(d - (j \bmod d))$.

4 $\quad$ *If $P_{\alpha,k}$ available then*

5 $\quad\quad$ *Send s_α to $P_{\alpha,k}$; Receive $s_{\alpha,k}$.*

6 $\quad\quad s_\alpha := 1 + \min(s_\alpha, s_{\alpha,k})$.

7 $\quad\quad$ *If $t_\alpha < s_\alpha$ then $t_\alpha := s_\alpha$; $z_\alpha := k$.*

8 $\quad$ *else $s_\alpha := 0$.*

---

cubes as does the single buddy system. To implement this double buddy system in parallel, we run Algorithm 3.1 twice, once for the choice of permutation π_1 which corresponds to the first single buddy system, followed by a run with permutation π_2 which corresponds to the second. This gives us an algorithm with twice the overhead to implement a system which allocates twice the number of subcubes. Extending these ideas to multiple buddy systems is straightforward. Given any multiple buddy system whose allocable subcubes are the union of a fixed number, say m, of single buddy systems, m runs of Algorithm 3.1 would perform the allocation in $\Theta(md)$ time. However, while the time increases linearly with m, the number of new allocable q-subcubes does not increase as rapidly because some of the q-subcubes are allocated by more than one buddy system.

3.2 Gray-Coded Buddy Systems

Let g_d denote the binary reflected Gray code map from $\{0, \ldots, 2^{d-1}\}$ to d-bit strings. The standard *single Gray-coded buddy system* allocates q-subcubes that arise as pairs of $q-1$-subcubes of the form $\{a_1 \ldots a_{d-q+1} * \ldots *$, $b_1 \ldots b_{d-q+1} * \ldots *\}$, where $g^{-1}_{d-q+1}(a_1 \ldots a_{d-q+1})$ and $g^{-1}_{d-q+1}(b_1 \ldots b_{d-q+1})$ are consecutive mod 2^{d-q+1}. As in the case of the single buddy system, each permutation π of $1, 2, \ldots, d$ gives rise to a gray-coded buddy system G_π. Each G_π allocates roughly the same number of subcubes as does the double buddy system. With a few modifications, Algorithm 3.1 can be used to implement G_π in $\Theta(d)$ time, as given in Algorithm 3.2.

After the completion of Algorithm 3.2, the largest subcubes which are both available and allocable under the gray-coded buddy system G_π are identified as follows. For each available P_α, where $\alpha = (a_1, a_2, \ldots, a_d)$, the value of s_α shows that that a subcube of dimension s_α is allocable under the standard buddy system, which is a subset of the subcubes allocable under the Gray-coded buddy system. P_α cannot be in a gray-coded buddy allocable subcube of dimension $s_\alpha + 2$ or greater, and it is in a gray-code allocable subcube of dimension $s_\alpha + 1$ if and only if there is a j such that $D(j) \geq s_\alpha$ and $g^{-1}_{d-s_\alpha}(a_{\pi(1)} \ldots a_{\pi(d-s_\alpha)})$ and $g^{-1}_{d-s_\alpha}(a_{\pi(1)} \ldots a_{\pi(j-1)}(1 - a_{\pi(j)})a_{\pi(j+1)} \ldots a_{\pi(d-s_\alpha)})$ are consecutive mod 2^{d-s_α}.

The *double Gray-coded buddy system*, first suggested in [ChSh], allocates q-subcubes by using the Gray-coded buddy system corresponding to the identity permutation, plus the Gray-coded buddy system corresponding to the permutation which reverses the order of the bits. This allocation scheme could be implemented in $\Theta(d)$ time by running Algorithm 3.2 twice, analogous to that done for the double buddy system. This analogy extends to schemes consisting of a larger number of gray-coded buddy systems, producing a $\Theta(md)$ time algorithm for m systems.

3.3 Cyclical Buddy Systems

While developing the allocation algorithms in this paper, we discovered a simple, efficient parallel allocation system which does not seem to arise naturally as a serial allocation system. This system, which we call a *cyclical buddy system*, allocates q-subcubes of the form $a_1 \ldots a_d$, where a_i through a_{i+q-1} are $*$ for some i, with the subscripts calculated modulo d. The cyclical buddy system allocates exactly $d2^{d-q}$ q-subcubes, giving complete allocation for $q = d - 1$. This system always allocates at least as many q-subcubes as the double buddy and Gray-coded buddy systems, and allocates strictly more than these systems when $d > 2$.

At the end of each iteration of the for-loop in Algorithm 3.3, s_α contains the largest number q such that the q-dimensional subcube $a_1 a_2 \ldots a_d$ is available, where $a_i = *$ for $i = \pi(d - j), \pi(d - j + 1), \ldots, \pi(d - j + q - 1)$, and $a_i = \alpha(i)$ otherwise. For $j \geq d$ these indices are interpreted in a modular fashion. (For $d = 1$ the upper limit of the for-loop should be increased to 0 to work properly. A natural upper limit on the for-loop is $2d - 1$, resulting in all dimensions being processed twice, but a little reflection shows that going through the last two dimensions a second time cannot produce a larger subcube than has been found earlier.) The variable t_α records the largest value of s_α, and z_α records

Algorithm 3.4 (Complete Allocation)

1 *For all subsets S of $\{1,\ldots,d\}$ of size k,*

2 *Let π be any permutation of $\{1,\ldots,d\}$ such that $\{\pi(1),\ldots,\pi(k)\} = S$.*

3 *Perform Algorithm 3.1 to find all $d-k$ cubes with S as their defining positions.*

---

the dimension along which it occurred. At the end of the algorithm each available processor stores the largest dimension of an available cyclical buddy subcube containing that processor, along with the starting dimension of the subcube.

The time of Algorithm 3.3 is clearly $\Theta(d)$. Just as for the buddy and Gray-coded buddy systems, one can extend to a multiple system by using m different cyclical buddy systems, utilizing m permutations, in $\Theta(md)$ time.

3.4 Complete Allocation

As we have seen, there are $\binom{d}{q}2^{d-q}$ subcubes of dimension q in $\mathcal{Q}(2^d)$. This number reaches its maximum value when $q = \lfloor d/3 \rfloor$, giving a value which exceeds $3^d/d$ for $d > 2$. Thus we cannot expect to find an algorithm to identify all available q-subcubes with a running time polynomial in d if we have at most 2^d processors. On the other hand, if we restrict the dimensions to be allocated, then it is possible, as we shall see, to have a polynomial time algorithm provided we use the hypercube.

We consider first a parallel algorithm based on the corresponding simple serial algorithm for allocating $d-k$-subcubes in $\mathcal{Q}(2^d)$. The serial algorithm proceeds by checking the availability of each of the 2^{d-k} pe's in each of the $\binom{d}{k}2^k$ $d-k$-subcubes, which gives us a $\Theta(\binom{d}{k}2^d)$ time algorithm. In any $d-k$-subcube, the k bit positions which are the same in all processors will be called the *defining positions*. For each choice of k defining positions there are 2^k $d-k$-subcubes. Our first algorithm sequentially goes through all possible defining positions, using the buddy system algorithm to find all the available subcubes with the selected defining positions. These steps are displayed formally as Algorithm 3.4. Since there are $\binom{d}{k}$ choices for defining positions, and Algorithm 3.1 takes $\Theta(d)$ time for each choice of defining positions, the total time is $\Theta(\binom{d}{k}d)$.

Our second algorithm, although more difficult to implement than Algorithm 3.4, finds all available $d-k$-subcubes in $\Theta(d)$ time, for fixed k. Let $C_k = \binom{2k+1}{2}$. If $d < C_k$ then use Algorithm 3.4. Otherwise, enumerate all pairs (without replacement) of the integers $1,\ldots,(2k+1)$ in lexicographic order. Let $\rho_1(i)$ denote the smallest element of the i^{th} pair, and $\rho_2(i)$ denote the largest element, for $1 \le i \le C_k$. For example, for $k = 1$ the pairs in order are (1,2), (1,3), and (2,3), and $\rho_1(2) = 1$ and $\rho_2(2) = 3$. Let D_k denote $\lfloor (d-C_k)/(2k+1) \rfloor$. We define subsets S_i of the coordinates $\{1,\ldots,d\}$, for $1 \le i \le 2k+1$, by

$$\begin{aligned} S_i &= \rho_1^{-1}(i) \cup \rho_2^{-1}(i) \cup \\ &\quad \{C_k + (i-1) * D_k + 1, \ldots, C_k + i * D_k\} \end{aligned}$$

for $i \le 2k$, and

$$\begin{aligned} S_{2k+1} &= \rho_1^{-1}(2k+1) \cup \rho_2^{-1}(2k+1) \cup \\ &\quad \{C_k + 2k * D_k + 1, \ldots, d\}. \end{aligned}$$

Notice that each set contains exactly $2k$ elements in $\{1,\ldots,C_k\}$, and each contains at least D_k elements in $\{C_k+1,\ldots,d\}$, with S_{2k+1} containing more if $2k+1$ does not divide $d-C_k$.

Each set S_i is used to find all available $d-k$-subcubes with defining positions in the complement of S_i. For, given any $d-k$-subcube, there is some i such that the k defining positions of the subcube must be in the complement of S_i. This is because there are $2k+1$ sets S_i, and any defining position j is in only 2 of the S_i, if $1 \le j \le C_k$, and in only one S_i, if $C_k < j \le d$. Therefore every $d-k$-subcube will be found.

For each i, the available $d-k$-subcubes with defining positions in the complement of S_i will be found by recursive halving to a subcube $\mathcal{T}_i$ of dimension $d-|S_i|$, and then recursively solving the problem in $\mathcal{T}_i$. The subcube $\mathcal{T}_i$ consists of all processors which have 0 as their j^{th} coordinate for $j \in S_i \cap \{C_k+1,\ldots,d\}$, and for $j \in S_i \cap \{1,\ldots,C_k\}$, have 0 if $\rho_1(j) = i$, and 1 if $\rho_2(j) = i$. Notice that there is no overlap in subcubes corresponding to two different subsets, since for any given subsets S_i and S_j, $i < j$, if m is the number of the pair (i,j), then in coordinate m all processors in the subcube $\mathcal{T}_i$ corresponding to S_i must have a 0, while in the subcube $\mathcal{T}_j$, corresponding to S_j, the processors must have a 1 in that coordinate.

The recursive halving to create the subcubes corresponding to the S_i is performed in two stages. First, the halving along dimensions in $S_i \cap \{1,\ldots,C_k\}$ is performed for the S_i one at a time. This takes exactly $2C_k$ total steps, and at its end, the information needed to create $\mathcal{T}_i$ is contained in a larger cube, which we denote by $\mathcal{T}_i^*$. Observe that these larger cubes have the property that they are pairwise disjoint. (This can be seen by the discussion in the preceeding paragraph). Therefore once the initial recursive halving in dimensions $\{1,\ldots,C_k\}$ is completed, the remaining halving along the rest of the dimensions in S_i can be done in parallel. The results of this second stage of recursive halving will be stored in the subcubes $\mathcal{T}_i$. The recursive solution can now take place, in parallel, in each of the subcubes $\mathcal{T}_i$. These steps are displayed formally in Algorithm 3.5.

The worst-case time of this algorithm will correspond to those subsets having the largest complement. The size of the largest complement is $d-(2k+D_k)$, which is $\lceil d2k/(2k+1) \rceil - k$. The worst-case number of communication steps therefore obeys a recurrence of the form

$$T_k(d) = 2C_k + \left\lfloor \frac{d-C_k}{2k+1} \right\rfloor + T_k(\left\lceil \frac{d2k}{2k+1} \right\rceil - k)$$

which is $\Theta(d)$ for any fixed k. Further, for any fixed k, $T_k(d)/d \sim 1$.

Although Algorithm 3.5 is fast, there are circumstances when allocation of all subcubes of dimension $d-k$, for fixed k, or, say bounded k, may not be the system of choice. In such cases, it may be preferable to allocate subcubes of all dimensions but restrict the type of subcube such as is done

Algorithm 3.5 (Complete $d-k$ Allocation)
This algorithm determines all $d-k$-dimensional subcubes of a d-dimensional hypercube.

1 *If $d < C_k$ then use Algorithm 3.4*

2 *else For $i := 1$ to $2k+1$ do*

3 *Form $\mathcal{T}_i^*$ by recursive halving for each coordinate $j \in S_i \cap \{1, \ldots, C_k\}$.*

4 *In parallel within each subcube $\mathcal{T}_i^*$,*

5 *Form $\mathcal{T}_i$ by recursive halving in each coordinate $j \in S_i \cap \{C_k+1, \ldots, d\}$.*

6 *Recursively call this algorithm to find all $d-k-|S_i|$-subcubes in $\mathcal{T}_i$.*

---

in the buddy or gray-coded systems. We must then contend with the trade-off between the number of allocable subcubes in a given allocation scheme and the time required to do the allocation. In the next subsection, we consider a generalization of the buddy and Gray-coded buddy systems that offers significant improvement over the schemes described in subsections 3.1 and 3.2.

3.5 k-Cube Buddy Systems

We describe here a family of allocation schemes which we first introduced in [LiSt]. Let $k \geq 1$ and consider an allocation scheme for $\mathcal{Q}(2^d)$ that will allocate q-subcubes in which the last $q-k$ bits are arbitrary and the first $d-q+k$ bits form the nodes of a subcube of dimension k in $\mathcal{Q}(2^{d-q+k})$. We call this system the standard single k-*cube buddy system*. In general, if π denotes a permutation of $1, 2, \ldots, d$, and k is a positive integer, then $QB_{k,\pi}$ denotes the single k-cube buddy system that allocates q-subcubes in which bits $\pi(d-q+k+1), \ldots, \pi(d)$ are arbitrary and bits $\pi(1), \ldots, \pi(d-q+k)$ form a k- subcube in $\mathcal{Q}(2^{d-q+k})$. We see that $QB_{k,\pi}$ allocates $\binom{d-q+k}{k} 2^{d-q}$ q-subcubes in $\mathcal{Q}(2^d)$. Note that $QB_{0,\pi}$ is the single buddy system, $QB_{1,\pi}$ extends the single Gray-coded buddy system, and $QB_{d,\pi}$ is the complete allocation system.

To implement the k-cube buddy system in parallel, suppose π is a given permutation of $1, 2, \ldots, d$. Sequentially perform recursive halving along dimensions $\pi(d)$, $\pi(d-1)$, $\ldots$, $\pi(d-q+k+1)$. For an available pe P_α with $\alpha(\pi(j)) = 0$ for $d-q+k+1 \leq j \leq d$, if the resulting s_α is $q-k$ then P_α represents a completely available $q-k$-subcube that might be combined with others to form an allocable $d-q$-subcube, and otherwise it represents an unavailable subcube.

Now it merely remains to find all k-subcubes among the pe's P_α with $\alpha(j) = 0$ for $d-q+k+1 \leq j \leq d$ which represent available $q-k$-subcubes. If Algorithm 3.4 is used, we obtain all available q-subcubes which are allocable by $QB_{k,\pi}$ in $\Theta(\binom{d-q+k}{k} k)$ time. By a fairly straightforward procedure, one can modify the approach of Algorithm 3.4 to move from one choice of defining bits to another in a manner than allows partial results to be reused. By incorporating this technique,

Algorithm 3.6 (k-Cube Buddy System)
Assume π is a given permutation of $1, 2, \ldots, d$, and each available P_α has an integer variable s_α.

1 $s_\alpha := 0$.

2 *For $t := d$ downto $d-q+k+1$*

3 *Perform recursive halving along dimension $\pi(t)$.*

4 *Find all k-subcubes in the remaining $d-q+k$-subcube, where if $s_\alpha < q-k$ then P_α is treated as being unavailable.*

---

one could reduce the time to $\Theta(\binom{d-q+k}{k} k/2^k)$. A more significant time reduction can be obtained by not solely reducing to those pe's P_α with $\alpha(j) = 0$ for $d-q+k+1 \leq j \leq d$. If instead all available pe's are used, in a manner similar to that used in Algorithm 3.5, one can reduce the time to $\Theta(d)$ for any fixed k and q. Due to space limitations we omit the details.

Since the k-cube buddy system can be implemented efficiently in parallel, we see that it provides an attractive alternative to the buddy systems. As we noted earlier, even for $k = 2$, we find a 50% improvement in its expected case behavior over that of the single buddy system.

Of course, multiple k-cube buddy systems can also be employed, and allocation can be performed using multiple runs of Algorithm 3.6.

4 Conclusion

We have considered the problem of allocating subsystems in MIMD parallel computers, a problem which becomes increasingly important and as the number of processors in the system grows.

Using only the non-busy and non-faulty pe's in the parallel computer to do the allocation, we have given algorithms which determine the available subsystems for the d-dimensional mesh and torus and for the d-dimensional hypercube.

We have given a simple, $\Theta(\sqrt{n})$ time algorithm to determine all rectangular subsystems in the two-dimensional mesh and torus with dimensions $\sqrt{n} \times \sqrt{n}$. In addition, we have given an algorithm which determines, for all d, all subsystems of the form $k \times k \times \cdots \times k$ in a d-dimensional mesh and torus of dimensions $m \times m \times \cdots \times m$ in optimal time $\Theta(dm)$.

To deal with subsystem allocation in hypercubes, we considered two approaches: one approach is to allocate only a subset of the possible subcubes in each dimension, the other approach is to limit the dimensions of the subcubes to be allocated. Using the first approach, we considered several allocation schemes including the buddy system, the gray-coded buddy system, the cyclical buddy system, and the k-cube buddy system, and provided optimal parallel algorithms for these. We found that with only small time and memory requirements there are several options available to increase the

number of allocable subcubes, thereby significantly improving the fault tolerance of the system. For the second approach to the problem, we gave a parallel algorithm which finds, for fixed k, all $d-k$ dimensional subcubes in time $\Theta(d)$, which is optimal. Depending on the specific requirements of the users of large systems, it may be advantageous to use some combination of complete allocation and the partial allocation schemes. Simulation studies are needed to evaluate the effectiveness of such a scheme in a given environment, however.

There is another approach to subsystem allocation which attempts to reconfigure or reroute to avoid a faulty node. For example, suppose we wish to allocate a subcube of dimension $d-1$ in a d-dimensional hypercube and suppose all nodes of $0 * \cdots *$ are available except the node $\alpha = (0,1,1,\ldots,1)$ is faulty. If a nearby node, such as $\beta = (0,0,1,\ldots,1)$, is available, we could allocate the "reconfigured" subcube of dimension $d-1$ in which α is replaced by β. Since any message sent to α from a neighbor of node α now must travel twice as far, we say this cube has *dilation* 2. Thus, our allocation problem could be extended to the allocation of subsystems with some limited dilation. This situation was investigated in [Hals]. Under the assumption that faults are distributed uniformly and randomly with probability $p < 0.5$ in a hypercube of dimension d, it is shown that, with high probability, it is possible to assign a $d-1$-dimensional subcube with dilation at most 7. Further studies need to be done and algorithms for allocation need to be developed in which dilation of some bounded size is allowed.

References

[BeSi] B. Becker and H. Simon, "How robust is the n-cube?", *Proc. 27th IEEE Symp. on Foundations of Comp. Sci.* (1986), 283-291.

[ChSh] M.-S. Chen and K. Shin, "Processor allocation in an n-cube multiprocessor using gray codes", *IEEE Trans. Computers C-36* (1987), 1396-1407.

[DuHa] S. Dutt and J. P. Hayes, "On allocating subcubes in a hypercube multiprocessor", *Proc. Third Conference on Hypercube Computers and Applications* (1988), 801-810.

[GHLS] N. Graham, F. Harary, M. Livingston, and Q.F. Stout, "Subcube fault-tolerence in hypercubes", Univ. Michigan Comp. Res. Lab. Tech. Rept. CRL-TR-12-87 (1987).

[Hals] J. Hastad, T. Leighton, and M. Newman, "Reconfiguring a hypercube in the presence of faults", *Proc. 19th ACM Symp. Theory of Comp.* (1987), 274-284.

[KlSp] D. Kleitman and J. Spencer, "Families of k-independent sets", *Discrete Math.*6 (1973), 255-262.

[LiSt] M. Livingston and Q.F. Stout, "Fault tolerance of allocation schemes in massively parallel computers", *Proc. 2nd Symp. on the Frontiers of Massively Parallel Computation* (1988), (to appear).