A New Strategy for Processors Allocation in an N-Cube Multiprocessor

Abdullah Al-Dhelaan
Bella Bose
Department of Computer Science
Oregon State University
Corvallis, Oregon 97331

88-70-2

A New Strategy for Processors Allocation in an N-Cube Multiprocessor

Abdullah Al-Dhelaan[*]
Bella Bose

Department of Computer Science
Oregon State University
Corvallis, OR 97331
(Tel. No. 503-754-3273)

[*] The first author is supported by King Saud University, Riyadh, Saudi Arabia

Abstract

In this paper we will describe two known strategies for static processors allocation in an n-cube multiprocessor, namely the buddy system strategy and the gray code strategy and then propose a new strategy that outperforms the first by (n-k+1) and the second by (n-k+1)/2 in cube recognition. Furthermore, our strategy is suitable for static as well as dynamic processors allocation and it results in a less system fragmentation, more subcubes recognition, and higher fault tolerance.

We also introduce an extension to our strategy that will enhance the performance drastically so that our algorithm together with the extension will outperform the buddy system by a factor of [k(n-k)+1] and the gray strategy by [k(n-k)+1]/2 in cube recognition. The implementation details of these algorithms are also described.

I. Introduction

During the the last five years, a movement from the SIMD (Single Instruction Multiple Data) to the general purpose MIMD (Multiple Instruction Multiple Data) machines has taken place, and the latter is drawing a lot of attention as a numerous research has been undertaken [1-12]. One common MIMD machine is the hypercube [2-12] which is becoming very popular for its attractive features to be addressed later.

The hypercube is a network of a loosely coupled processors connected in such a way that two processors are linked if and only if their binary representation differ in exactly one bit position. i.e the indices of neighboring processors differ by a power of 2.

A n-dimensional hypercube , denoted as n-cube or Q_n, is a hypercube with 2^n processors and is defined recursively as: A 0-dimensional hypercube, Q_0, is a single processor, and an n-dimensional hypercube is two (n-1)-dimensional hypercubes with links between corresponding processors in each of them. Fig. 1 and 2 show a Q_3, and a Q_4, hypercubes respectively.

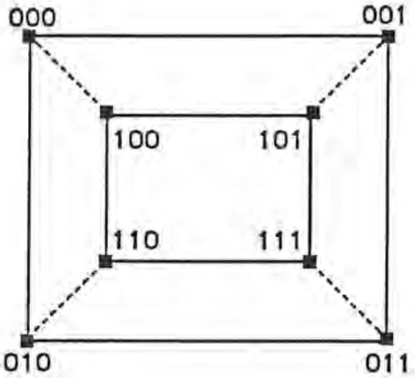


Fig.1 , 3-dimensional hypercube, Q_3

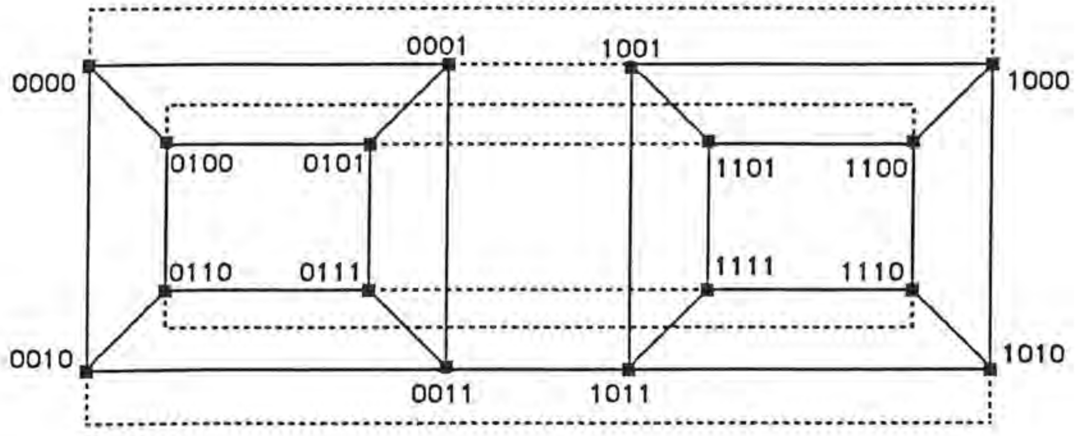


Fig. 2, A 4-dimensional hypercube, Q_4

Some of the hypercube characterizations are :

1. Each processor has a local memory and no shared memory is used.
2. Processors communicate by sending messages direct or through some intermediate processors.
3. Synchronization occurs by the availability of data and messages.

Some attractive features of the hypercube are:

1. Regularity and high potential for the parallel execution of various algorithms.
2. Its architecture allows high level of concurrency and efficiency.
3. The number of links among processors is small allowing us to have a very large hypercube with a reasonable number of links. At present machines with up to 16384 processors are available [2].

Processors allocation in a hypercube is a two steps process:

1. Determination of the size of the incoming task in terms of the number of processors needed in order to accommodate it.

2. Recognizing and locating a subcube that can accommodate the incoming task.

The first step is investigated in [3], and some algorithms for step 2 are given in [4]. In this paper more efficient algorithms for processor allocation are described.

The paper is organized as follows. Section II introduces the necessary notations. In section III, we will describe two existing allocation strategies namely the buddy system, and the gray code, [4], for the n-cube multiprocessor; then we will propose a new strategy that outperforms both of the above. Our strategy is suitable for static as well as dynamic processors allocation and it results in a less system fragmentation, higher subcubes recognition and higher fault tolerance. In section IV, we will explain an efficient way to implement our algorithm. The paper concludes with section V.

II. Notations:

Below are some operations that we will be using in our next definitions.

1) $A^{\propto\backslash\beta}$ means for each element $a_{k-1},a_{k-2},...,a_{\beta},a_{\beta-1},...,a_0$ in A, insert $\propto$ in the βth position as $a_{k-1},a_{k-2},...,a_{\beta},\propto,a_{\beta-1},...,a_0$

Example 1:

If A = { 00,01,11,10 } then we have

$A^{1\backslash 1}$ = { 010,011,111,110 }

$A^{1\backslash 2}$ = { 100,101,111,110 }

2) A^* means reverse the elements of A.

Example 2:

If A = { 00,01,11,10 } then A^*= { 10,11,01,00 }

3) $B_n(m)$ is the binary representation of an integer m with n bits.

Example 3:

$B_4(1) = 0001, B_3(1) = 001$

4) For a set A , |A| is the number of elements in A and is called the cardinality of A.

Example 4:

The cardinality of a set $A = \{a_1,a_2,....,a_n\}$ is $|A| = n$

Definition:

The product of any two graphs $G_1 = (V_1,E_1)$ and $G_2 = (V_2,E_2)$, denoted by $G_p = G_1 \text{ X } G_2$, is the graph $G_p = (V_p,E_p)$, where $V_p = V_1 \text{ X } V_2$ and any two nodes $u = (u_1,u_2)$ and $v = (v_1,v_2)$ are adjacent if and only if

$(u_1 = v_1$ & u_2 is adjacent to v_2 in G_2) or
$(u_2 = v_2$ & u_1 is adjacent to v_1 in G_1)

Definition:

An N-Cube Q_n is defined recursively as

a) Q_0 is a trivial graph with one node.
b) $Q_n = K_2 \text{ x } Q_{n-1}$ $n > 0$

The address of the subcube Q_2 which consists of the processors {0000, 0011, 0110, 0111 } in a 4-cube is written as 0X1X . i.e. X is the don't care term and it can take values 0 or 1.

III. Processor Allocation Strategies

In an n-cube multiprocessor, processors must be allocated to incoming tasks in a way that will maximize the processors utilization and minimize the system fragmentation. In order to achieve this goal, it is necessary to detect the availability of a subcube of required size and merge the released small cubes to form a larger ones.

First, we will briefly describe the known methods [4], the buddy strategy and the gray code strategy. Then we will propose a new strategy and show that it out performs both of these by a high factor. Next, an extension to our strategy that will enhance the performance drastically at a cost of a little overhead is introduced. It is shown that our strategy results in a less fragmented system and recognizes more subcubes.

A. The Buddy Strategy :

The buddy strategy can be described using a binary tree. An example is shown for the 4-cube in Fig. 3, where the external nodes are the processors. The nodes in level i are associated with subcubes of dimension n-i and a node is available if all of its offsprings are available. When an incoming task requests a cube of certain size, say Q_k, the level n-k is searched from left to right for an internal node which is free. When a free internal node is found all the descendant nodes are allocated for this task. In this method 2^n allocation bits are needed to keep track of which node is available; a processor with its bit set to 0 (1) is available (not available) .

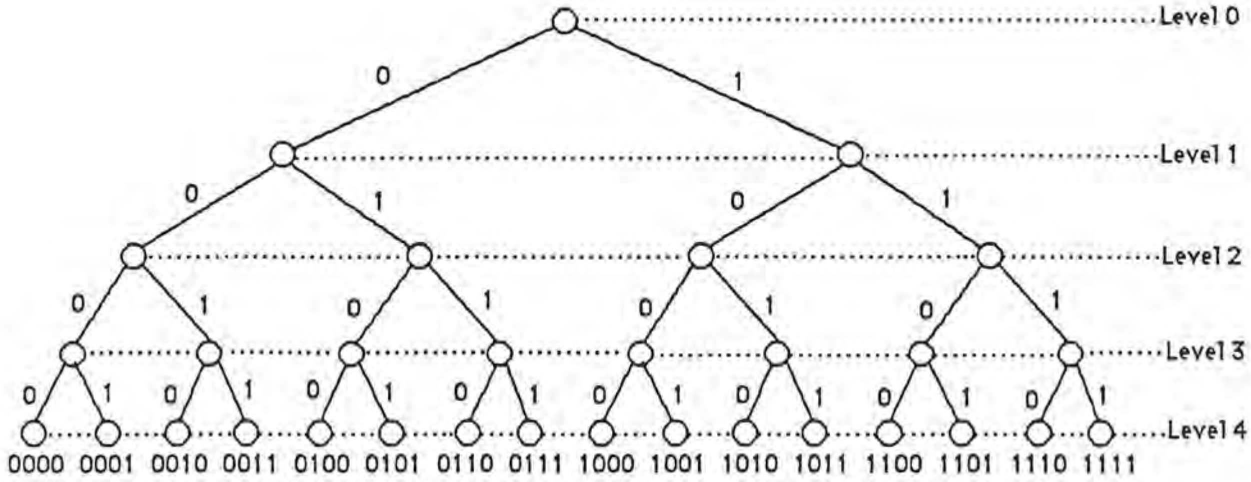


Fig. 3, Processor allocation using the buddy strategy

This method was studied in [4] and implemented in the NCUBE/six multiprocessor [5]. The algorithm is given below.

Algorithm 1 (Buddy System Strategy):

Processor Allocation :

```
Step 1. Set k to the dimension of a subcube required to accommodate the
        request.
step 2. Determine the least integer $\alpha$, $0 \le \alpha \le 2^{n-k+1}-1$ such that all the $\beta$th
         allocation bits, $\alpha 2^k \le \beta \le (\alpha+1)2^k-1$, are 0's. Set all these
         bits to 1's.

step3.  Allocate processors with addresses $B_n(\beta)$ to the request, where
         $\alpha 2^k \le \beta \le (\alpha+1)2^k-1$.
```

Processor Relinquishment:

Reset every pth allocation bit to 0, where $B_n(p)$ is used in the subcube released.

Example 5:

An example of the static processor allocation using the buddy system strategy in a 4-cube multiprocessor is given in Fig . 4.

Incoming Request		Allocated processors
No.	Size	
I_1	Q_0	0000
I_2	Q_3	1000,1001,1010,1011 1100,1101,1110,1111
I_3	Q_2	0100,0101,0110,0111
I_4	Q_1	0010,0011
I_5	Q_0	0001

Fig. 4, Processor allocation using buddy system

This strategy recognizes only 2^{n-k} Q_ks within the n-cube multiprocessor. Compared to other methods, which we explain later, the buddy strategy underutilizes processors in the n-cube multiprocessor.

B. The GC Strategy :

This strategy can also be explained using a binary tree. An example is shown for 4-cube in Fig. 5, where the edges of the tree are labeled with gray code, which is explained below, and the processors are the external nodes.

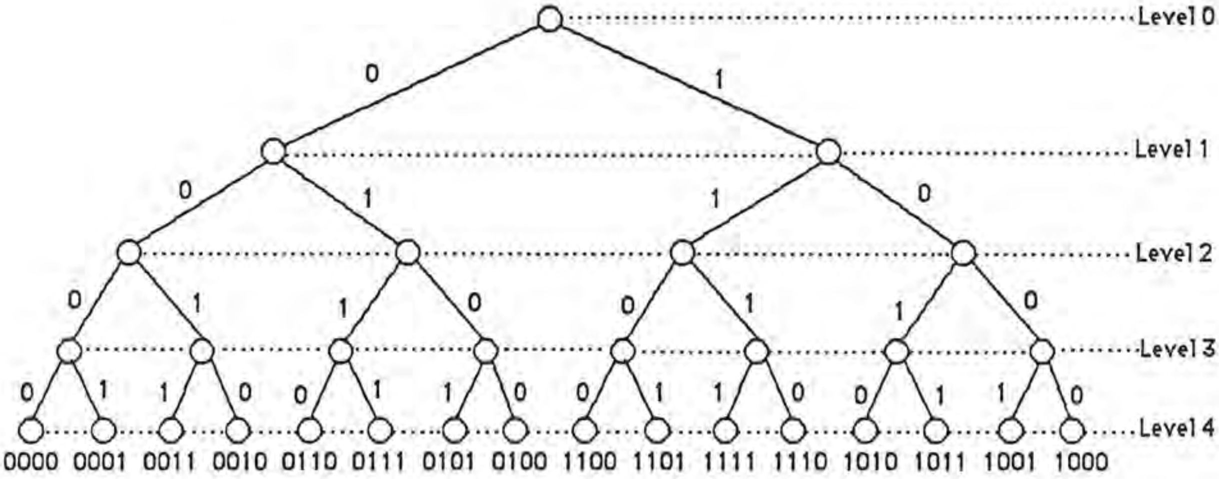


Fig. 5, Processor allocation using the gray strategy

Like the buddy strategy each node in level i is associated with a cube of dimension n-i and 2^n allocation bits are needed to keep track of the availability of the processors. A node is available if all of its offsprings are available where a processor with its bit set to 0(1) indicates the availability (unavailability) of that processor. When an incoming task requests a cube, say Q_k, level n-k+1 is searched from left to right for two adjacent $Q_{k/2}$ cubes and these two combined constitute a Q_k, instead of searching for a Q_k cube in level n-k, as in the buddy strategy. Because of this reason the number of Q_k cubes recognized by the gray code strategy is twice that of buddy system strategy. This method was studied in [4].

Before describing the algorithms some definitions are stated first.

Definition :

Let $L = \{ g_1, g_2, ..., g_n\}$ be a set. The partial rank r_i of g_i is the rank of g_i in the subset $\{g_1,g_2,...,g_i\}$ when rearranged these elements in ascending order.

Example 6:

Let $\{g_1,g_2,g_3\} = \{3,1,2\}$ then we have

$\{g_1\} = \{3\} \quad ==> r_1 = 1$
$\{g_1,g_2\} = \{3,1\} \quad ==> r_2 = 1$
$\{g_1,g_2,g_3\} = \{3,1,2\} \quad ==> r_3 = 2$

Definition:

A gray code G_n with parameters $\{g_1,g_2,...,g_n\}$ where $\{g_1,g_2,...,g_n\}$ is any permutation of $\{1,2,...,n\}$ is defined recursively as

$$G_1 = \{0,1\}$$
$$G_k = \{G^{0\backslash rk}{}_{k-1}, (G^*{}_{k-1})^{1\backslash rk}\} \qquad 2 \leq k \leq n$$

where rk is the partial rank of $g_{k.}$

Example 7:

If $\{g_1,g_2,g_3\} = \{2,3,1\}$ then we have $\{r_1,r_2,r_3\} = \{1,2,1\}$. Thus
$G_1 = \{0,1\}$
$G_2 = \{00, 01, 11, 10\}$
$G_3 = \{000, 010, 110, 100, 101, 111, 011, 001\}$

If $g_i = i$ then we have Binary Reflected Gray Code (BRGC), the most frequently used case.

Example 8:

If $\{g_1,g_2,g_3\} = \{1,2,3\}$ then $\{r_1,r_2,r_3\} = \{1,2,3\}$.
$G_1 = \{0,1\}$
$G_2 = \{00, 01, 11, 10\}$
$G_3 = \{000, 001, 011, 010, 110, 111, 101, 100\}$

The Relation between GC and Binary:

To translate the gray code to its corresponding binary representation and vice versa the formula below is used [13].

$$g_i = b_i \text{ xor } b_{i+1} \quad i \neq n$$
$$g_n = b_n$$

Example 9:

The gray code that corresponds to the binary number 101 is 111.

Algorithm 2 (Gray Code Strategy):

Processor Allocation :

step 1. Set k to the dimension of a subcube required to accommodate the request.

step 2. Determine the least integer α, $0 \le \alpha \le 2^{n-k+1}-1$, such that all ($\beta \bmod 2^n$)th allocation bits are 0's, where $\alpha 2^{k-1} \le \beta \le (\alpha+2)2^{k-1}-1$. Set all these bits to 1's.

step3. Allocate nodes with addresses $G_n(\beta \bmod 2^n)$ to the request, where $\alpha 2^{k-1} \le \beta \le (\alpha+2)2^{k-1}-1$.

Processor Relinquishment:

Reset every pth allocation bit to 0, where $G_n(p)$ is used in the subcube released.

Example 10:

Using the same request sequence from example 5, the static processor allocation using the gray code strategy in a 4-cube multiprocessor is given in Fig . 6.

Incoming Request		Allocated processors
No.	Size	
I_1	Q_0	0000
I_2	Q_3	0110,0111,0101,0100 1100,1101,1111,1110
I_3	Q_2	1010,1011,1001,1000
I_4	Q_1	0011,0010
I_5	Q_0	0001

Fig. 6, Processor allocation using the gray strategy

This strategy recognizes 2^{n-k+1} Q_k within the n-cube multiprocessor and this is an improvement by a factor of two over the buddy strategy.

C. A new Strategy :

We propose a new strategy that outperforms the buddy strategy by a factor of 2(n-k+1) and thus the gray strategy by a factor of (n-k+1) in recognizing subcubes of size k. This is a significant improvement because in practical systems it is normal to have many small incoming jobs and large number of processors.

The new strategy can be described using the binary tree in Fig. 3. The external nodes in this tree correspond to the processors. The nodes in level i are associated with subcubes of dimension n-i and a node is available if all of its offsprings are available. Like the gray code strategy, when an incoming task requests a cube, say Q_k, level n-k+1 is searched from left to right for two $Q_{k/2}$ and these two combined constitute the cube Q_k but unlike gray code strategy, our strategy will recognize these two $Q_{k/2}$ cubes even if they are not adjacent. When a free internal node is found all the descendant nodes are allocated for this task. Like in the other strategies 2^n allocation bits are needed to keep track of which node is available, a processor with its bit set to 0(1) available (unavailable) . A more efficient method using 2^{n+1}-1 bits, is described in section IV.

The path from the root of the tree to any node is that node's address. This address corresponds to the subcube which consists of all the descendants processors (leaf nodes). Note that in Q_4 subcube 01, 01X or 01XX denotes the same subcube.

We will list some definitions and then describe the new algorithm.

Definition:

The $\propto$th partner of $a_{k-1},a_{k-2},..., a_{\propto+;},a_{\propto},a_{\propto-1},...,a_0$ for any $0 \le \propto \le k-1$ is defined as

$a_{k-1},a_{k-2},..., a_{\propto+1},1,a_{\propto-1},...,a_0$, if $a_{\propto} = 0$
undefined , if $a_{\propto} = 1$.

we denote the pth partner of $B_k(i)$ as $B^p{}_k(i)$.

Example 11:

All the partners for the nodes $B_3(i)$, $0 \le i \le 7$, are shown in Fig. 7.

Node	0th partner	1st partner	2nd partner
000	001	010	100
001	undefined	011	101
010	011	undefined	110
011	undefined	undefined	111
100	101	110	undefined
101	undefined	111	undefined
110	111	undefined	undefined
111	undefined	undefined	undefined

Fig. 7, The partners for B_3 (i) for all i.

Lemma 1:

The 0th partner of an even number a is a + 1 which can be represented using the same number of bits.

i.e. $B^0_\propto(2i) = B_\propto(2i+1)$, $0 \le i \le 2^{\propto-1}-1$

proof:

Since 2i is an even integer, it has 0 in the least significant bit and changing it to 1 will give 2i+1; thus 2i+1 will need no more than $\propto$ bits.

Lemma 2:

For any two integers $\propto$ and β such that $0 \le \beta \le \propto-1$, out of all the nodes $B_\propto(i)$, $0 \le i \le 2^\propto-1$, there are exactly $2^{\propto-1}$ nodes that have a βth partner.

proof:

Only those that has a 0 in the βth bit will have a βth partner. If we fix the βth bit to be 0 then the other $\propto$ - 1 can take any value. This will give $2^{\propto-1}$ possible nodes.

Example 12:

Consider the nodes $B_3(i)$, $0 \leq i \leq 7$ shown in Fig. 7. We can see that exactly 4 nodes have a 0th partner, 4 nodes have a 1st partner, and 4 nodes have 2nd partner defined.

Definition:

For any integer $\propto$, $0 \leq \propto \leq 2^{n-k+1}-1$, the node $B_{n-k+1}(\propto)$ is free if and only if all of its descendants are free. For example for n = 4 and k = 2, the node 000 is free if and only if the processors 0000, 0001 are free.

Algorithm (3) (new strategy):

Processor Allocation :

Step 1. Set k to the dimension of a subcube required to accommodate the request.

Step 2. Determine the least integer $\propto$, $0 \leq \propto \leq 2^{n-k+1}-1$, such that $B_{n-k+1}(\propto)$ is free and it has a pth, $0 \leq p \leq$ n-k, partner $B^p{}_{n-k+1}(\propto)$ which is also free. Take p as small as possible.

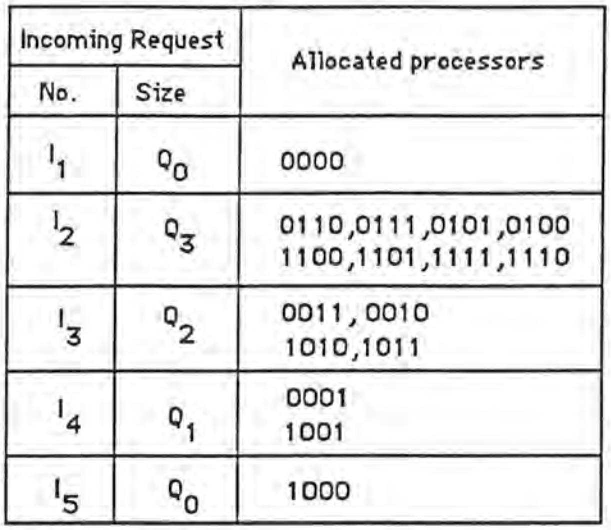
step3. Allocate these processors to the request and set their allocation bits to 1.

Processor Relinquishment:

Reset the allocation bits of all the processors that correspond to the descendants of the nodes $B_{n-k+1}(\propto)$ and $B^p{}_{n-k+1}(\propto)$ to 0. (See section IV for detail).

Example 13:

In Fig 8 , we show an example of the allocation strategy for a 4-cube multiprocessor using the same request sequence as in Examples 5 and 10.



Incoming Request		Allocated processors
No.	Size	
I_1	Q_0	0000
I_2	Q_3	0110,0111,0101,0100 1100,1101,1111,1110
I_3	Q_2	0011, 0010 1010,1011
I_4	Q_1	0001 1001
I_5	Q_0	1000

Fig. 8, Processor allocation using our strategy

From this example we can see that our strategy compacts things to the left which result in less system fragmentation; thus our strategy recognizes more subcubes.

The following lemmas give the number of subcubes recognized by the new algorithm.

Lemma 3:

Algorithm (3) generates **(n-k+1) 2^{n-k}** Q_k cubes.

proof:

In step (2), Algorithm (3) considers each of the nodes , $B_{n-k+1}(\propto)$, $0 \leq \propto \leq 2^{n-k+1}-1$ with each of its partners and then combine them to form a Q_k.

Summing over the partners, we can conclude that the number of Q_k

cubes generated by the algorithm is equal to

$$\sum_{\beta=0}^{n-k} \text{(number of nodes which have a } \beta\text{th partner)} = (n\text{-}k+1)\, 2^{n-k}$$

Example 14:

Consider requesting a Q_2, in a 4-cube multiprocessor. In step (2), Algorithm (3), will consider each of the 8 nodes $B_3(\propto)$, $0 \le \propto \le 7$ with all of its partners.

To ease the counting, we can group them by partners as follows:

The nodes which have a 0th partner are {000,010,100,110}
The nodes which have a 1th partner are {000,001,100,101}
The nodes which have a 2th partner are {000,001,010,011}

A cube will be formed by combining any node with any of its partners.
Thus, the number of cubes = $(3)(2)^2$

Lemma 4:

The Q_ks generated by Algorithm (3) are disjoint among themselves .

proof:

The cubes recognized by algorithm (3) can be listed in $\mathbf{2^{n-k}}$ X **(n-k+1)** matrix as shown in Fig. 9.

$$\left[\begin{array}{cccccc}
\overbrace{\text{00...00X}}^{n-k+1} & \text{00...0X0} & \text{00...X00} & \text{..........} & \text{0X0...00} & \text{X00...00} \\
\text{00...01X} & \text{00...0X1} & \text{00...X01} & \text{..........} & \text{0X0...01} & \text{X00...01} \\
\text{..........} & \text{..........} & \text{..........} & \text{..........} & \text{..........} & \text{..........} \\
\text{11...10X} & \text{11...1X0} & \text{11...X10} & \text{..........} & \text{1X1...10} & \text{X11...10} \\
\text{11...11X} & \text{11...1X1} & \text{11...X11} & \text{..........} & \text{1X1...11} & \text{X11...11}
\end{array}\right]_{2^{n-k} \text{ X } (n-k+1)}$$

Fig. 9 , Recognized subcubes

We can see from the matrix that each element a_{ij} , $0 \leq i \leq 2^{n-k}-1$, and $0 \leq j \leq n-k$, is of the form:

$$a_{ij} = (B_{n-k}(i))^{x\backslash j} = a_{n-k-1}, a_{n-k-2}, ..., a_j, X, a_{j-1}, ..., a_1, a_0$$

Suppose two cubes a_{ij} and a_{st} are equal (i.e. $a_{ij} = a_{st}$)
where

$$a_{ij} = (B_{n-k}(i))^{x\backslash t} = a_{n-k-1}, a_{n-k-2}, ..., a_t, X, a_{t-1}, ..., a_1, a_0$$

$$a_{st} = (B_{n-k}(s))^{x\backslash t} = a_{n-k-1}, a_{n-k-2}, ..., a_t, X, a_{t-1}, ..., a_1, a_0$$

This implies i = s and j = t. Thus all cubes are distinct.

The subcube recognition problem becomes more important when considering some faulty processors or when allocating processors dynamically. In these situations also out strategy does better than the buddy strategy or gray code strategy as illustrated in the following examples.

Example 15: (fault tolerance)

In a 4-cube multiprocessor if two nodes, one from (0000,0001) and the other from (1000,1001) are faulty. Then neither the buddy system allocation strategy nor the gray code strategy will be able to satisfy the requests $\{I_1 = Q_3, I_2 = Q_2\}$ but our strategy will satisfy this.

Example 16: (Dynamic allocation)

Consider the request $\{ I_1 = Q_1, I_2 = Q_2, I_3 = Q_1, I_4 = Q_3 \}$. If I_1 and I_3 released their processors and others do not then using the buddy system strategy or the gray code strategy a request like $\{ I_5 = Q_2 \}$ will not be satisfied. But our strategy will combine the two released Q_1s into a Q_2 and allocate it to I_5 .

D. Algorithm (4): (An extension to Algorithm (3))

In Algorithm (3), when a Q_k cube is requested, the nodes at level n-k+1 of the tree are searched. Suppose none of the subcubes corresponding to this level is available. In this case, the algorithm terminates without allocating a Q_k cube. However, it might be possible to get to the higher levels of the tree and get more Q_k cubes. We first explain the method by taking an example and then describe the algorithm.

Example 17:

In the case of n = 5 and k = 3, Algorithm (3) will recognize the following 12 subcubes in Fig. 10. These subcubes are searched at level 3 of the tree.

$$\begin{bmatrix} \mathrm{00XXX} & \mathrm{0X0}\overbrace{\mathrm{XX}}^{k-1} & \mathrm{X00}\overbrace{\mathrm{XX}}^{k-1} \\ \mathrm{01XXX} & \mathrm{0X1XX} & \mathrm{X01XX} \\ \mathrm{10XXX} & \mathrm{1X0XX} & \mathrm{X10XX} \\ \mathrm{11XXX} & \mathrm{1X1XX} & \mathrm{X11XX} \end{bmatrix}$$

Fig. 10, Q_3 cubes recognized in Q_5 using Algorithm 3.

Suppose none of the above subcubes are available. After cyclic right shifting the columns 2 and 3 of the subcubes we get the following 16 new subcubes as shown in Fig. 11.

$$\underset{\text{After one cyclic right shift}}{\begin{bmatrix} \mathrm{X0X0X} & \mathrm{XX00X} \\ \mathrm{X0X1X} & \mathrm{XX01X} \\ \mathrm{X1X0X} & \mathrm{XX10X} \\ \mathrm{X1X1X} & \mathrm{XX11X} \end{bmatrix}} \quad \underset{\text{After two cyclic right shift}}{\begin{bmatrix} \mathrm{XX0X0} & \mathrm{XXX00} \\ \mathrm{XX0X1} & \mathrm{XXX01} \\ \mathrm{XX1X0} & \mathrm{XXX10} \\ \mathrm{XX1X1} & \mathrm{XXX11} \end{bmatrix}}$$

Fig. 11, Q_3 cubes in Q_5 generated by Algorithm (4).

The subcubes obtained after shifting once and twice correspond to searching the tree at levels 4 and 5 respectively. Thus these new subcubes can be generated systematically.

The formal algorithm is given below. To balance the search overhead and the recognition ability, we use a parameter δ which will limit the depth of the search up to, and including, level n-k+1+δ. If $\delta \geq$ k-1, then the search is done at all levels from n-k+2 to n. A good heuristic approach is to make δ large if the cubes are not released very frequently. Note that, when δ = 0, it will not search any cubes in step 2 and when δ = infinite, it will search all the levels.

Algorithm (4):

Processor Allocation :

Step 1: Use algorithm (3) to find Q_k. If succeeded then done else goto step 2.

Step 2:

```
 { This is the extension to Algorithm 3, needed if Step 1, fail }


$\delta$ = How far we want to go down, infinite to maximize.

For i = 0 to ($2^{n-k+1}$-2) do
begin
  For d = 1 to min (k-1, $\delta$ ) do   { Search level n-k+1+d }
  begin
    For j = 0 to (n-k) do
    begin
      If ($B_{n-k+1}$ (i) has a jth partner) then
      begin
        current_cube = (The address of the cube formed from
                          $B_{n-k+1}$ (i) and $B^j_{n-k+1}$ (i) )
        current_cube = concat ('X'$^d$, current_cube)
        if (current_cube is free) then
        begin
            allocate (current_cube);
            exit;
         end
      end;
   end
end
```

Lemma 5:

Step2 of Algorithm (4) , with δ = infinite, generates $\mathbf{(k\text{-}1)(n\text{-}k)2^{n\text{-}k}}$ new $Q_k$cubes.

proof:

Ignoring the case p=0, Algorithm (3) generates (n-k) 2^{n-k} subcubes. These correspond to the last n-k columns of the matrix in Fig. 9. Each of these can be expanded k-1 times. So step 2 of Algorithm (4) will generate (k-1)(n-k) 2^{n-k} new Q_k cubes.

Lemma 6:

The Q_k cubes generated step 2 of Algorithm (4) are disjoint among themselves.

proof:

Step 2 of Algorithm (4) generates cubes by right cyclic shifting addresses that are already proved to be disjoint, by lemma 4, thus have distinct (n-k+1) most significant digits. When shifting i times, $1 \leq i \leq$ k-1, the (k-1-i) least significant digits will be X's. So each shift will produce a new set of cubes that are different and disjoint from all others.

Lemma 7:

The Q_k cubes generated by step1 of Algorithm(4), i.e. Algorithm (3), are disjoint from those generated by Step 2 of Algorithm (4).

proof:

This is clear since the addresses of those recognized by Algorithm (3) when expressed with n digit have at least k-1 X's. However, those generated by step 2 of Algorithm (4) have less than k-1 X's.

From the above lemmas the following theorem holds.

Theorem 8 :

Algorithm (4) (both steps) will generate $[\mathbf{k(n\text{-}k)+1}]\ \mathbf{2^{n-k}}$ distinct Q_ks

Example 18:

As a quick comparison we consider a request for a Q_2 subcube in a Q_4 multiprocessor. Form the previous discussion one can easily see the following:

a) The buddy system strategy will recognize

00XX 01XX 10XX 11XX

b) Gray code strategy will recognize

00XX 0X1X 01XX X10X 11XX 1X1X 10XX X00X

c) Algorithm (3), i.e. step 1 of Algorithm (4), will recognize

00XX 0X0X X00X 0X1X
X01X 01XX X10X X11X
10XX 1X0X 1X1X 11XX

d) Step 2 of Algorithm (4) will recognize

X0X0 XX00
X0X1 XX01
X1X0 XX10
X1X1 XX11

e) Algorithm (4) will recognize both (c) and (d).

A summary is given in Fig. 12.

The Method		Number of Q_2s recognized	
Buddy System (Algorithm 1)		2^{n-k}	= 4
Gray Strategy (Algorithm 2)		2^{n-k+1}	= 8
Our Strategy	(Algorithm 3)	$(n-k+1)\,2^{n-k}$	= 12
	(Algorithm 4) (Step. 2)	$(k-1)(n-k)\,2^{n-k}$	= 8
	(Algorithm 4)	$[k(n-k)+1]\,2^{n-k}$	= 20

Fig. 12 , The number of Q_2 s recognized in a Q_4

IV. Implementation Issues:

When implemented using an array of 2^n bits, we will have to check sequentially 2^n allocation bits each time a cube is requested which could be extremely large. This will result in an inefficient implementation especially if dynamic allocation is used and/or the rate of releasing the cubes can't be predicted.

An efficient way to keep track of the availability of the processors and the subcubes is to organize their allocation bits as a tree in which the external nodes are the allocation bits for the processors and the internal nodes are the allocation bits for the subcubes. The external node is 0 if the corresponding processor is available and 1 if it is not. Similarly, the internal node is 0 if the subcube associated with it is available as a whole and 1 if it is not. We can represent a tree of height n using an array $A[1: 2^{n+1}-1]$ of $2^{n+1}-1$ bits. The root node corresponds to location 1. For an internal node i, the left son is stored at 2i and the right son is at 2i+1. Furthermore, the internal nodes at level i are at location 2^i to 2^{i+1} -1. So when a Q_k is requested, the nodes at level (n-k+1) of the tree needs to be searched. These nodes correspond to array positions 2^{n-k+1} to 2^{n-k+2}-1. For these nodes the pth partner, if exists, can be obtained by changing the pth bit from 0 to 1. An example is shown for Q_3 in Fig. 13 below.

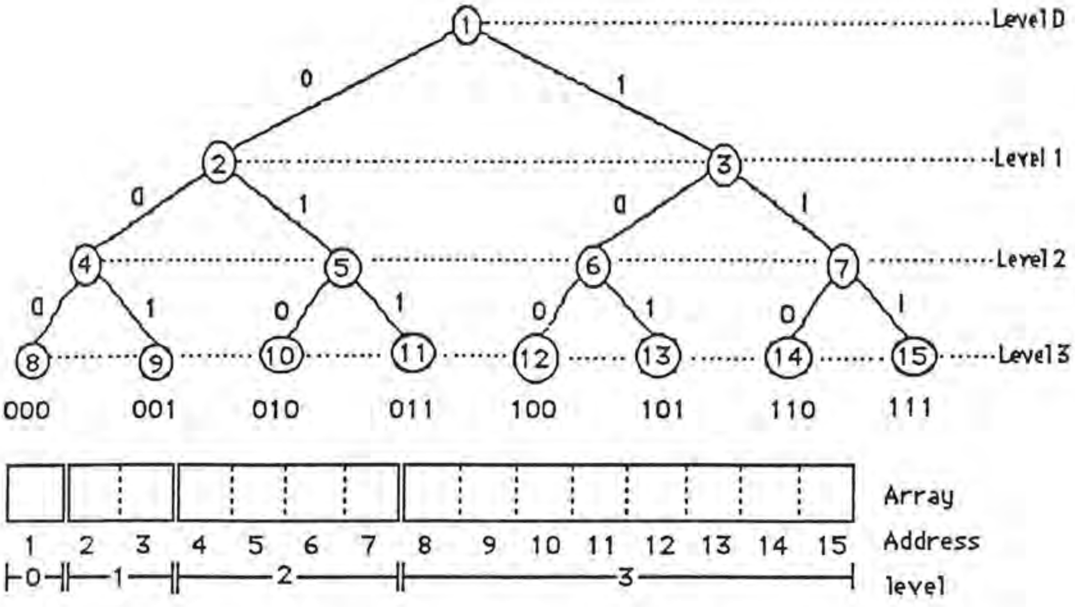


Fig. 13 , A tree and its array representation for Q_3

Allocation:

To allocate a k-dimensional subcube, Q_k :

1. Search the tree at the level n-k+1 for two free $Q_{k/2}$s that form a Q_k , when combined.We use algorithm (3) given in section III to do the search systematically.
2. Allocate these two $Q_{k/2}$s to the incoming task.
3. For each external node,v, covered by (i.e. descendents of) the subcubes obtained in the previous step do:
   a. Set v to 1.
   b. Traverse the fathers link setting the allocation bits to 1, until either the root or an unavailable cube is reached.

Deallocation:

To deallocate a k-dimensional subcube, Q_k :

1. Set the allocation bits associated with the two $Q_{k/2}$ to 0.
2. For each external node,v, covered by (i.e. descendents of) the subcubes obtained in the previous step do:

   a. Set v to 0.
   b. If its sibling is available set the fathers allocation bit to 0.
   c. Propagate upwards until nodes can't be merged anymore or the root is reached.

V. Conclusion:

We briefly reviewed the two known strategies for static processors allocation in an n-cube multiprocessor, namely the buddy system strategy and the gray code strategy and then proposed a new strategy that outperforms the first by (n-k+1) and the second by (n-k+1)/2 in cube recognition.

Furthermore, our strategy is suitable for static as well as dynamic processors allocation and it results in a less system fragmentation, more subcubes recognition, and higher fault tolerance.

An extension to our strategy that searchs the whole tree and thus improves the performance is described. When our algorithm and its extension are combined, Algorithm (4), they outperform the buddy system by [k(n-k)+1] and the gray strategy by [k(n-k)+1]/2 in cube recognition. Implementation details of these algorithms are also discussed.

References:

[1] K. Hwang and F.A. Briggs, Computer Architecture and Parallel Processing, New York: McGraw-Hill, 1984.

[2] R. M. Chamberlain, "Gray codes, Fast Fourier Transformations and Hypercubes," Parallel Computing, 6, 1988, pp. 225-233.

[3] M. Chen and K. G. Shen, "Embedment of interesting task modules into a hypercube multiprocessor," in Proc. Second Hypercube Conf., Oct. 1986, pp. 121-129, .

[4] M. Chen and K. G. Shen, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," IEEE Trans. Comput., vol. C-36, Dec. 1987, pp. 1396-1407.

[5] B. Becker and H. U. Simon, "How robust is the n-cube?," in Proc. 27th Ann. Symp. Foundations Comput. Sci., Oct. 1986, pp. 283-291.

[6] C. L. Seitz, "The cosmic cube," Commun. Ass. Comput. Mach., vol. 28, Jan. 1985, pp. 22-33.

[7] NCUBE Corp., NCUBE/ten: AN Overveiw, Beverton, OR, Nov. 1985.

[8] L.N. Bhuyan and D.P. Agrawal, "Generalized hypercube and hyperbus structures for a computer network," IEEE Transactions on Computers, C-33, pp. 323-333, April 1984.

[9] J. R. Armstrong and F. G. Gray, "Fault Diagnosis in a Boolean n Cube Array of Multiprocessors," IEEE Trans. Comput., vol. C-30, Aug. 1981, pp. 390-393.

[10] M. Pease, "The Indirect Binary n-Cube Microprocessor Array," IEEE Trans. Comput., C-26, May. 1977, pp. 458-473.

[11] H. P. Katseff, "Incomplete hypercubes," IEEE Trans. Comput., vol. 37, 5, pp. 604-608, May1988.

[12] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," IEEE Transactions on Computers, C-37, July 1988, pp 867-872.

[13] Z. Kohavi, Switching and Finite Automata Theory, New York: McGraw-Hill, 1978.

[14] F. Harary, Graph Theory. Reading, MA:Addison-Wesely, 1969.