Processor allocation strategies for modified hypercubes

S.G. Ziavras
N.G. Haravu

Indexing terms: Hypercube, Parallel processing, Processor allocation

Abstract: Modified hypercubes (MHs) have been proposed as building blocks for hypercube-based parallel systems that support the application of incremental growth techniques. In contrast, systems implementing the standard hypercube network cannot be expanded in practice. However, processor allocation for MHs is a more difficult task due to a slight deviation in their topology from that of the standard hypercube. The paper proposes two strategies to solve the processor allocation problem for MHs. The proposed strategies are characterised by perfect subcube recognition ability and superior performance. Furthermore, two existing processor allocation strategies for standard hypercube networks, namely the buddy and free-list strategies, are shown to be ineffective for MHs, in the light of their inability to recognise many available subcubes. A comparative analysis that involves the buddy strategy and the new strategies is carried out using simulation results.

1 Introduction

In the n-dimensional hypercube (or n-cube) parallel computer, n physical communication channels are attached to each of the 2^n processors. If distinct n-bit binary addresses are assigned to the processors, then two processors are neighbours if their addresses differ by a single bit. Hence, the expansion of existing hypercube systems can be accomplished only by replacing the processor chips with others containing more communication ports. This, together with long wires in large systems, are the major drawbacks of the hypercube network, in spite of its rich properties: (*a*) regular topology, (*b*) simple routing, (*c*) high degree of fault tolerance, (*d*) small diameter (for a 2^n processor system, the farthest node is only n links away), and (*e*) efficient emulation of other topologies [10, 11]. Another major drawback of the hypercube network is that its total number of processors is always a power of two.

Many hypercube variations reported in the literature fail to provide a solution to this incremental growth problem without expending extra resources for individual processors [7, 8], Ziavras [10], on the other hand, has proposed the family of modified hypercubes (MHs) which are slightly modified hypercubes that support incremental growth techniques without introducing extra resources for individual hypercubes. Results have shown that the performance of MHs is comparable to that of standard hypercubes.

The processor allocation problem for hypercube-based systems is that of determining the availability and allocating subcubes of requested size in real time. A processor allocation scheme can be either static or dynamic [9]. An allocation scheme is static if incoming requests are considered for allocation only at specific time intervals. Such schemes do not consider deallocation of resources. On the other hand, a dynamic scheme can handle processor allocation and deallocation at any time. Dynamic allocation schemes provide higher utilisation of resources. The majority of practical instances of allocation problems are NP-complete [1, 2].

The allocation of processors in standard hypercube multiprocessors may be achieved by using either bottom-up or top-down approaches. These are based on different bit-mapping techniques wherein each time the processor allocation algorithm attempts to locate a k-cube with 2^k free nodes. Two simple bottom-up subcube allocation strategies, namely the buddy and Gray code (GC) strategies, apply a first-fit sequential search [9]. The buddy and GC strategies order node addresses in an allocation list according to their binary code and binary reflected Gray code, respectively. The reflected Gray code is such that any two consecutive elements differ only by a single bit. The subcube recognition ability of the GC strategy is much better than that of the buddy strategy [4]. It is shown in Section 5 that these strategies do not perform well for MHs.

The top-down free-list strategy [9] keeps track of free subcubes available in a hypercube; a similar scheme was proposed elsewhere [5, 6]. Both schemes modify the free list dynamically, each time a subcube is allocated or deallocated. These strategies also do not perform well for MHs because they cannot recognise several available subcubes.

This paper proposes two processor allocation strategies for MHs which are based, partially or entirely, on a table look-up approach. The first strategy, which implements the table look-up approach in its entirety, maintains a list of all possible subcubes. An allocation bit associated with each subcube address determines its

This work was supported in part by the National Science Foundation under Grants CCR-9109084 and CDA-9121475, and by NJIT under Grant SBR-421240.

Paper 1036E (C2, C3), first received 13th May and in revised form 23rd December 1993

The authors are with the Department of Electrical and Computer Engineering, New Jersey Institute of Technology, Newark, New Jersey 07102, USA

availability. Although the space complexity of this strategy is high, it is presented here for the purposes of comparison and simplification of the discussion for the second strategy. The second strategy reduces the space complexity associated with the first strategy by maintaining only a small, partial look-up table. Both strategies have a perfect subcube recognition ability and time complexities comparable to that of other top-down strategies for standard hypercubes.

2 Modified hypercubes (MHs)

Modified hypercubes [10] which are amenable to incremental growth techniques are obtained by slightly modifying standard hypercubes without introducing any extra hardware resources. In addition, all resources in MHs are available for use. An MH is constructed from a standard hypercube as follows. A few gateway processors are selected for interconnections with other systems according to a unique method described in the next paragraph, one link associated with each such processor is then broken, and finally half of these links are joined together in pairs to enhance the topological properties of the resulting system while the other half can be used for incremental growth. Various interconnection schemes for clusters of MHs may be introduced. Reference 10 describes the family of hypertorus interconnection networks; a hypertorus is produced by replacing each node in a torus with an MH.

Gateway processors, used for interconnecting MHs, are referred to here as I/O processing elements (I/O PEs). Note that I/O PEs do not necessarily implement communication with the outside world. Given a standard hypercube, the methodology that produces an MH is as follows. Let $H(n, nil)$ denote the standard n-dimensional hypercube (nil stands for 0 I/O PEs). For the sake of simplicity, let the total number of I/O PEs in the corresponding MH be a power of two. Then $H(n, \lambda)$ denotes the modified form of the hypercube $H(n, nil)$ that contains 2^{λ} I/O PEs . The I/O PEs are chosen according to the following procedure. If $\alpha = 2^{\beta+1}$, where $\beta = n - \lambda$, then the decimal addresses of the I/O PEs in the transformed system $H(n, \lambda)$ are $\Gamma_{\gamma} = \gamma\alpha$ and $\Gamma_{\zeta+\gamma} = (2^n - 1) - \Gamma_{\gamma}$, where $0 \leqslant \gamma \leqslant (2^{\lambda-1} - 1)$ and $\zeta = 2^{\lambda-1}$. Any two PEs with addresses Γ_{γ} and $\Gamma_{\zeta+\gamma}$ are diametrically opposite in $H(n, nil)$ as second address is the one's complement of the first address. This technique chooses PEs to serve as gateways for inter-MH data transfers in such a way that they are uniformly distributed in the MH.

One link of every I/O PE is broken in the original hypercube for the support of inter-MH communications through direct connections to other MHs. The chosen link is the one that implements a connection in the lowest dimension (i.e. dimension 0). However, such a technique leaves one communication link unused for any non-I/O PE which is attached to a broken link. These unused links are then interconnected in pairs, so that all resources (including communication ports) become available for use in the resulting system. The I/O PEs in $H(n, \lambda)$ form $2^{\lambda-1}$ pairs of diametrically opposite PEs in $H(n, nil)$. Since the addresses of non-I/O PEs which are attached to broken links differ from the addresses of the corresponding I/O PEs only in the least significant bit (LSB), the 2^{λ} non-I/O PEs which are attached to broken links are also diametrically opposite in pairs. To reduce the average distance and the diameter of MHs, pairs of diametrically opposite non-I/O PEs which are attached to broken links are directly connected. Fig. 1 shows the structure of $H(4, 1)$ and $H(4, 2)$. The I/O PEs are shown in circles while the repositioned links that connect together pairs of diametrically opposite nodes are represented by dotted lines.

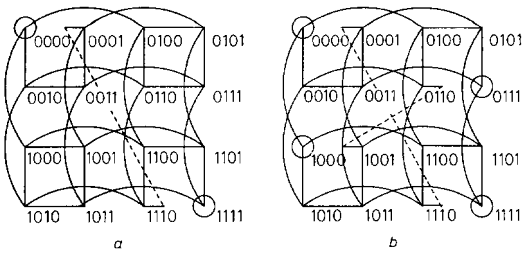
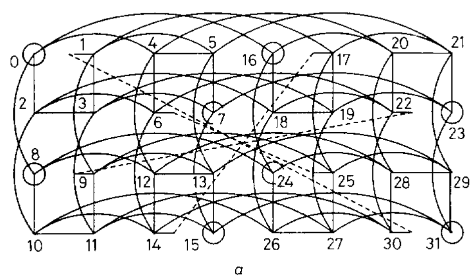


Fig. 1 *Modified hypercube structures H(4, λ)*
a $\lambda = 1$
b $\lambda = 2$

Fig. 2*a* illustrates the structure of $H(5, 3)$ with 32 processors and eight I/O PEs. The superior performance of MHs is demonstrated by the following set of their important properties. Respective proofs can be found in Reference [10].



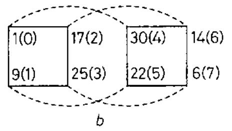


Fig. 2 *Subcubes due to repositioned links*
a The MH(5, 3)
b The λ-cube with repositioned links mapped onto a regular λ-cube

(*a*) The I/O PEs in $H(n, \lambda)$ form two disjoint $(\lambda - 1)$-dimensional hypercubes. This property is rather important because it implies ease of accessing I/O PEs. These hypercubes in Fig. 1*b* contain the pairs of nodes (0000, 1000) and (0111, 1111), respectively.

(*b*) The non-I/O PEs which are attached to repositioned links in $H(n, \lambda)$ form a λ-dimensional hypercube that was not present in the original hypercube $H(n, nil)$. The importance of this property stems from the formation of new subcubes. The new 2-cube in Fig. 1*b* contains the nodes 0001, 1110, 1000 and 0110. The new 3-cube in Fig 2*a* contains the nodes 1, 9, 17, 25, 6, 14, 22 and 30.

(*c*) In the modified hypercube $H(n, \lambda)$, the maximum $\Delta_{I/O}$ of the shortest distances from I/O PEs is

$$\Delta_{I/O} = \begin{cases} \lceil \beta/2 \rceil & \text{if } \beta > 4 \text{ or } \beta = 1 \\ 3 & \text{if } 3 \leqslant \beta \leqslant 4 \\ 2 & \text{if } \beta = 2 \end{cases}$$

A small value for this parameter is crucial because it guarantees fast access to resources external to the MH.

For practical cases, $\beta > 4$ and the value of $\Delta_{I/O}$ is small. For example, $\Delta_{I/O} = 3$ in $H(16, 10)$ that contains 65 536 processors and 1024 I/O PEs.

(*d*) The average shortest distance from I/O PEs in $H(n, \lambda)$ is equal to

$$2^{-\beta}\left(\sum_{\varepsilon=1}^{\lceil\beta/2\rceil} \varepsilon\binom{\beta}{\varepsilon} + \sum_{\varepsilon=0}^{\beta-\lceil\beta/2\rceil-1} (\varepsilon+1)\binom{\beta}{\varepsilon} + u_\beta\right)$$

where

$$u_\beta = \begin{cases} 2 & \text{if } \beta \geqslant 3 \\ 1 & \text{if } \beta = 2 \end{cases}$$

For $\beta = 1$, the value of this measure is equal to 0.5. The smaller the value of this parameter, the faster the exchange of values with resources external to the particular MH. This parameter is small for practical cases.

(*e*) The diameter Δ of $H(n, \lambda)$ is

$$\Delta = \begin{cases} \min\{\Delta_{I/O} + \lceil n/2\rceil, n\} & \text{if } \beta > 1 \\ n+1 & \text{if } \beta = 1 \end{cases}$$

For practical cases, the diameter of $H(n, \lambda)$ is smaller than that of the hypercube $H(n, nil)$ with the same number of nodes. For example, the diameter of $H(16, 10)$ is 11, while it is 16 for $H(16, nil)$; both systems contain 65 536 processors.

(*f*) For practical cases, the average distance in $H(n, \lambda)$ is lower than the average distance in $H(n, nil)$.

Since the methodology that produces MHs modifies only some connections that correspond to the lowest dimension of the original hypercube, the hypercube topology is still the dominant topology in the resulting system.

3 Processor allocation strategies for hypercubes

Let Σ be the ternary symbol set $\{0, 1, X\}$, where X is the 'don't care' symbol. The address of a subcube in the n-cube can be uniquely represented by a string of n symbols in Σ. For example, the address of the two-dimensional subcube containing the nodes 0011, 0111, 1011 and 1111 in the four-dimensional cube in $XX11$.

3.1 The buddy strategy

Since there exist 2^n nodes in the n-cube, 2^n allocation bits are used by the buddy strategy in order to keep track of the availability of nodes. A 0/1 value for an allocation bit indicates the availability/unavailablity of the corresponding node [9]. A request for a subcube of dimension k is accommodated by determining the least integer m such that all allocation bits from $m2^k$ to $(m+1)2^k - 1$ are zeros. These allocation bits are set to 1 and the corresponding nodes are allocated. The computation time for allocation is $O(2^n)$. For the deallocation of a subcube, the allocation bits of the included nodes are reset to 0. The deallocation time is $O(2^k)$.

Static allocation strategies accommodate incoming requests without dealing with processor relinquishment (deallocation). A static allocation strategy is said to be statically optimal if it can accommodate any sequence of incoming requests as long as the sum of the required subcube dimensions is less than or equal to the total number of dimensions in the hypercube system. The buddy strategy is statically optimal. On the other hand, dynamic allocation using the buddy strategy has been observed to be far from optimal due to its poor subcube recognition ability. If a subcube request of size k is to be serviced on the n-cube, the buddy strategy looks for only those cubes with Xs in the k least significant bits of their hypercube address. That is, it can recognise only a portion of

$$1\bigg/\binom{n}{k}$$

of all possible subcubes. Nevertheless, the time complexity of the buddy strategy is $O(2^n)$.

3.2 The free list strategy

The free list strategy maintains $n+1$ independent lists, where the kth list contains the available subcubes of dimension k, for $0 < k \leqslant n$ [9]. Each element in the list is represented by a unique address, a sequence of n ternary symbols. The original n-cube is represented as a sequence of n Xs. When there is a request for a k-cube, where $k \leqslant n$, the algorithm looks for a k-cube in the free list. If there is no k-cube in the free list, one of the available nearest higher dimension subcubes is decomposed from the most significant bit side recursively until a k-cube is identified. The resulting k-cube is then assigned to the request. The computation time of the allocation algorithm is $O(n)$, because this is the number of lists that need be searched for non-emptiness.

During the deallocation cycle, a released k-cube is included in the k-cube list and its address is compared with those of other entries in the k-cube list as well as higher dimensional cube lists in order to form new cubes of more dimensions. This process is repeated until no more combinations are possible. The generated cubes in the lists are then made mutually disjoint. The deallocation time is $O(n\,2^n)$.

For the sake of comparison, the allocation times for the Gray code and multiple extended Gray code strategies referenced in the introduction are

$$O(2^n) \quad \text{and} \quad O\left(\binom{n}{\lfloor n/2\rfloor}2^n\right)$$

respectively. The deallocation times are $O(n2^n)$ and $O(2^{3^n})$, respectively.

4 Processor allocation strategies for MHs

4.1 Identification of subcubes in MHs

In order to find the possible number of k-cubes in the n-dimensional standard hypercube, one has to simply evaluate

$$\binom{n}{k}2^{n-k}$$

(see Proposition 2 below). Furthermore, the addresses of these subcubes can easily be obtained by generating all possible sequences of n-ternary symbols containing k X-bits.

The ease with which the above procedure can be accomplished is severely affected when one or more of the hypercube links are removed, and more so when some links are repositioned as in MHs. As seen earlier, the I/O and repositioned links in MHs were used to implement interconnections in the lowest dimension of the original standard hypercubes. Therefore, removing these links affects the number of 1-cubes whose addresses have an X in the least significant bit. Moreover, the number of 1-cubes is increased due to the repositioning of links. To conclude, a number of subcubes of various dimensions which were present in the original hypercube cease to exist in the MH because of the removed links, and a number of new subcubes are added to the original hypercube due to the repositioning of half of the removed links.

We may distinguish between two categories of subcubes in MHs. All the subcubes that were also present in the original hypercube fall into the first category, whereas subcubes that were not present in the original hypercube belong to the second category. Theorem 2 below gives the number of subcubes of dimension k, for $0 \leqslant k \leqslant n-1$, in the MH $H(n, \lambda)$; these numbers are required for the application of the processor allocation strategies proposed later on in his paper. Propositons 1 and 2, and Theorem 1 are pertinent.

Proposition 1: In the n-cube, with $n > 0$, there are only two nodes whose binary addresses have either all zeros or all ones.

Proposition 2: The number of subcubes of dimension k in the n-cube is

$$\binom{n}{k} 2^{n-k}$$

Proof: The number of distinct ways in which k bits can be selected among n bits is

$$\binom{n}{k}$$

For each of these combinations, the set of the remaining $n-k$ bits can take 2^{n-k} distinct values. Hence, the result.

Theorem 1: In the n-cube the number $\Phi(n, k, \lambda)$ of subcubes of dimension k that do not contain any of the $2^{\lambda+1}$ nodes whose binary addresses have in their lower $n-\lambda$ bits either all zeros or all ones is

$$\sum_{i=0}^{\min\{\lambda, n-k-2\}} \binom{\lambda}{i} 2^i (2^{n-k-i} - 2) \binom{n-\lambda}{k-(\lambda-i)} \quad \text{if } k > \lambda$$

and

$$\sum_{i=0}^{\min\{k, n-\lambda-2\}} \binom{\lambda}{k-i} 2^{\lambda-(k-i)} (2^{n-\lambda-i} - 2) \binom{n-k}{i} \quad \text{if } k \leqslant \lambda$$

iff $1 \leqslant k \leqslant (n-2)$ and $\lambda \leqslant (n-2)$. $\Phi(n, 0, \lambda) = 2^n$. $\Phi(n, k, \lambda) = 0$, for all other cases.

Proof:
Case (a): $k > \lambda$ and $1 \leqslant k \leqslant (n-2)$ and $\lambda \leqslant (n-2)$
The objective here is to find all possible subcubes of dimension k so that the lower $n-\lambda$ bits of their address does not contain either all zeros or all ones. With the usual notation of subcubes, let the upper λ bits of the address contain all Xs. Then, the remaining $(k-\lambda)$ Xs can be selected among the available $n-\lambda$ lower address bits in

$$\binom{n-\lambda}{k-\lambda}$$

distinct ways. For each of these combinations, the remaining $n-k$ bits should not be identical. From Proposition 1, this equals $2^{n-k}-2$. Now, let the upper λ bits contain $(\lambda-1)$ Xs. From Proposition 2, this can be selected in

$$\binom{\lambda}{\lambda-1} 2^1$$

distinct ways. The remaining $(k-(\lambda-1))$ Xs can be accommodated in the lower $n-\lambda$ bits in

$$\binom{n-\lambda}{k-(\lambda-1)}$$

distinct ways. Again, for every combination, the remaining bits can be selected in $2^{n-(k+1)}-2$ distinct ways, from Proposition 1. The above procedure continues until there are no Xs in the upper λ bits or until there are no more than $(n-\lambda-2)$ Xs in the lower $n-\lambda$ bits. The numbers of subcubes of dimension k, $\Phi(n, k, \lambda)$, is equal to the sum of all the above combinations and is given by

$$\sum_{i=0}^{\min\{\lambda, n-k-2\}} \binom{\lambda}{i} 2^i (2^{n-k-i} - 2) \binom{n-\lambda}{k-(\lambda-i)}$$

Case (b): $k \leqslant \lambda$ and $1 \leqslant k \leqslant (n-2)$ and $\lambda \leqslant (n-2)$
Let all the Xs be in the upper λ bits of the address. From Proposition 2, this can be selected in

$$\binom{\lambda}{k} 2^{\lambda-k}$$

distinct ways and the remaining $n-\lambda$ bits can be selected in $2^{n-\lambda}-2$ distinct ways. All other valid combinations for the existence of the subcube can be obtained by transferring the Xs from the upper λ bits to the lower $n-\lambda$ bits, one-by-one, until there are no more Xs in the upper λ bits or until there are at least two bits in the lower $n-\lambda$ bit field which are not Xs. Then, $\Phi(n, k, \lambda)$ is given by

$$\sum_{i=0}^{\min\{k, n-\lambda-2\}} \binom{\lambda}{k-i} 2^{\lambda-(k-i)} (2^{n-\lambda-i} - 2) \binom{n-k}{i}$$

Case (c): $\lambda > (n-2)$
Since the subcubes under consideration should contain no nodes with either all zeros or all ones in their lower $n-\lambda$ bits, the ternary addresses of the nodes must contain at least a 0 and a 1. Hence, $\Phi(n, k, \lambda) = 0$, for $\lambda > (n-2)$.

Case (d): $k > (n-2)$
This implies $\Phi(n, k, \lambda) = 0$, for all cases where, $k = n$ and $k = n-1$. Therefore, $\Phi(n, k, \lambda) = 0$, for $k > (n-2)$.

Case (e): $k = 0$
This is a trivial case and $\Phi(n, k, \lambda)$ is not affected by the modification in the hypercube. Hence, $\Phi(n, 0, \lambda) = 2^n$.

Theorem 2: The number $N(n, k, \lambda)$ of subcubes of dimension k in the MH $H(n, \lambda)$ is equal to

$$2\binom{n-1}{k} 2^{n-1-k} + \Phi(n-1, k-1, \lambda) + 2^{\lambda-k}\left[\binom{\lambda}{k} - \binom{\lambda-1}{k}\right] \quad \text{if } k > 0$$

where $\Phi(n', k', \lambda')$ is given by Theorem 1. $N(n, 0, \lambda) = 2^n$.

Proof: $H(n, \lambda)$ has 2^λ I/O nodes, therefore 2^λ links are removed from the original hypercube $H(n, \mathit{nil})$. The removed links have addresses similar to those of the I/O PEs except that their LSB is X. The I/O PEs have addresses such that, for every possible combination in the upper $\lambda - 1$ bits there are two such nodes with one of them containing all zeros in the lower $n - \lambda + 1$ bits and the other one containing all ones in the lower $n - \lambda + 1$ bits.

The problem of determining the number of subcubes of dimension k in $H(n, \lambda)$ can be broken down into two cases: determining the number of subcubes in $H(n, nil)$ with 2^λ links removed, as mentioned above, and determining the number of subcubes in the hypercube of dimension λ formed by the non-I/O PEs which are attached to repositioned links.

Case (*a*): This can be further divided into two subcases.
Subcase (*i*): The subcube address is selected in such a way that there is no X in the least significant bit of the hypercube address. Since k Xs can be selected among $n - 1$ bits in

$$\binom{n-1}{k} 2^{n-k-1}$$

distinct ways (Proposition 2) and the least significant (nth) bit can be either 0 or 1, the number of subcubes is equal to

$$2\binom{n-1}{k} 2^{n-k-1}$$

Subcase (*ii*): The subcube address is selected in such a way that there is an X in the least significant bit (LSB) of the hypercube address. This is a special case in Theorem 1, with $n' = n - 1$, $\lambda' = \lambda$ and $k' = k - 1$. Hence, the number of subcubes is given by $\Phi(n - 1, k - 1, \lambda)$.

Case (*b*): One of the properties of MHs is that the non-I/O PEs which are attached to repositioned links form a hypercube of dimension λ. Now this hypercube is formed by two hypercubes of dimension $\lambda - 1$ which belong to the original hypercube. All the links that connect these two hypercubes to form the hypercube of dimension λ are due to the repositioned links. The number of subcubes of dimension k due to the original hypercubes of dimension $\lambda - 1$ is

$$2\binom{\lambda-1}{k} 2^{\lambda-1-k}$$

(Proposition 2), and the number of subcubes of dimension k due to the new hypercube of dimension λ is

$$\binom{\lambda}{k} 2^{\lambda-k}$$

Since we are interested in only those subcubes of dimension k that include the repositioned links (to ensure the subcubes selected in this fashion are mutually exclusive from the earlier selected subcubes), we subtract the former number from the latter to get

$$\binom{\lambda}{k} 2^{\lambda-k} - 2\binom{\lambda-1}{k} 2^{\lambda-k-1}$$

which is the same as

$$2^{\lambda-k}\left[\binom{\lambda}{k} - \binom{\lambda-1}{k}\right]$$

This term exists only for $\lambda > k$. It is 0 for $\lambda \leqslant k$.

The trivial case when $k = 0$ is directly given by Proposition 2.

4.2 The table look-up strategy

Since MHs are obtained by modifying standard hypercubes, direct application of the processor allocation strategies of Section 3 to MHs would lead to inefficient results. In this section, a processor allocation strategy, which is based on a table look-up approach, is proposed for MHs. Although the space complexity of this strategy is high, its introduction here will simplify the discussion for the second proposed strategy. The table look-up strategy can recognise all possible subcubes in $H(n, \lambda)$ by maintaining n independent lists, one list per possible subcube dimension k, where $0 \leqslant k \leqslant (n - 1)$. The addresses of all possible subcubes in the MH are determined and stored in the form of linked lists, with each list representing subcubes of the same dimension. In fact, the structure of each element in the linked lists consists of the following fields:

(*a*) a unique address consisting of n ternary symbols,
(*b*) a bit to indicate the availability of the subcube,
(*c*) a bit to indicate whether the subcube contains any repositioned links,
(*d*) an integer to represent the number of busy processors in the subcube.

Initially, the n independent lists are formed so that the ith list consists of $N(n, i, \lambda)$ entries. The address of the subcube in each entry is initially determined and stored in the address field of the entry, whereas the subcube allocation bit and the variable representing the number of busy processors are set to zero.

Whereas the subcubes due to the original hypercube can be represented in the usual manner with n ternary symbols, the addresses of the subcubes due to the repositioned links are not obtained in a straightforward manner. The following procedure is undertaken to represent these subcube addresses. The λ-cube formed by non-I/O PEs attached to repositioned links is mapped onto a regular hypercube j of dimension λ, with PE addresses from 0 to $2^\lambda - 1$; i.e. the lower λ bits in subcube addresses (which are n bits long) in the lists resulting from the λ-cube contain the addresses of the corresponding subcubes in the regular hypercube j and the bit which signifies that the subcube results from repositioned links is set to one. Note that the λ-cube is made of two $(\lambda - 1)$-cubes which belong to the original hypercube. Also, one of the $(\lambda - 1)$-cubes has Xs in the $\lambda - 1$ most significant bits of its address, followed by all zeros in the lower field, except the LSB which is one. Similarly, the other $(\lambda - 1)$-cube has Xs in the $\lambda - 1$ most significant bits of its address, followed by all ones in the lower field, except the LSB which is zero. The repositioned links connect pairs of nodes from the two $(\lambda - 1)$-cubes if they are diametrically opposite in the original hypercube. Fig. 2*a* shows a 5-cube with $\lambda = 3$ (i.e. the $H(5, 3)$). Fig. 2*b* shows the λ – cube with the repositioned links connecting the two $(\lambda - 1)$-cubes and the mapping of this λ-cube onto the regular λ-cube; the addresses of nodes in parentheses are the addresses in the regular λ-cube.

To obtain the n-bit actual address of any node present in the λ-cube from its λ-bit address A_x in the regular λ-cube convert the address A_x to n bits by inserting $n - \lambda$ trailing zeros. If the most significant bit (MSB) of A_x was zero, then the actual node address is the value obtained when this n-bit value is shifted once left with a 1 as the bit shifted into the LSB. If the MSB of A_x was one, then the n-bit value is complemented and then shifted once left with a 0 as the bit shifted into the LSB. For example, if $A_x = 5_{10}$ or 101_2 in Fig. 2*b*, then the first conversion to n bits gives 10100_2. Since the MSB of A_x was 1, the address in the original hypercube is 10110_2 or 22_{10}. If $A_x = 2_{10}$ or 010_2 in Fig 2*b*, then the first conversion to n bits gives 01000_2. Since the MSB of A_x was 0, the address in the original hypercube is 10001_2 or 17_{10}.

When the subcube lists are created, approximately the first 2^{n-k} addresses in the kth list are selected in such a way that the same addresses would have been selected if the buddy strategy were to be implemented. The allocation and deallocation procedures of the table look-up strategy follow.

Allocation:

Step 1: Let k be equal to the dimension of the subcube required to accommodate the current request. If there exists an available subcube in the k-cube list (i.e. there exists an available k-cube with busy-processor count equal to zero), allocate it. For each of the nodes in the allocated subcube, determine the other subcubes among all the lists that contain the node under consideration, reset their availablity bit, and increment their busy-processor count.

Step 2: If a subcube of the requested size is not found, then keep the request in the waiting queue (until a subcube of the required size becomes available).

Deallocation:

Step 1: After a task is completed, the corresponding subcube is deallocated by setting its availability bit.

Step 2: For each of the nodes in the subcube, determine other subcubes among all the lists that contain this node and decrement their busy-processor count.

Step 3: Set the deallocated subcube's busy-processor count to zero.

The allocation and deallocation procedures take time $O(N(n, k, \lambda) + 2^k \times 2^n)$ or $O(2^{n+k})$. This processor allocation strategy can also be used for standard hypercubes. Of course, in this case there will not exist any subcubes containing repositioned links. The next two theorems are pertinent.

Theorem 3: The table look-up strategy is statically optimal for allocation on standard hypercubes.

Proof: For standard hypercubes, the look-up table is made up of $n + 1$ lists, one for every possible subcube dimension k, where $0 \leqslant k \leqslant n$. Let these lists be arranged in such a way that the first 2^{n-k} elements have the same subcube addresses as the ones the buddy strategy would look for, if the buddy strategy were to be implemented. Only if these subcubes are unavailable, will the table look-up strategy look for other possible combinations by searching the rest of the list. It has been proved [3] that the buddy strategy is statically optimal. Hence, it can be concluded that the table look-up strategy is also statically optimal for allocation on standard hypercubes.

Proposition 3: For standard hypercubes and the table look-up strategy, if the lists contain only the addresses of adjacent subcubes that the free list strategy can recognise, then the table look-up strategy has performance similar to that of the free list strategy.

Proof: The selection of a subcube in the free list strategy follows the K-map adjacency rule which is also used by the Gray code strategy. Thus, for a k-cube request, the free list strategy initially looks for a free subcube in the k-cube list; if no free k-cube exists then the strategy looks for a free subcube in the higher dimension list which is adjacent to the previous list, and so on. The same situation is encountered in the table look-up strategy if the addresses in the look-up table are arranged so that neighbouring addresses within lists are adjacent to each other. Thus, the k-cube list contains 2^{n-k} entries such that the most significant $n - k$ bits of the addresses follow a Gray code.

The following example illustrates the equivalence of the two strategies for standard hypercubes under the condition of Proposition 3.

Example: In a 5-cube system, let the requests for the following sequence of subcube dimensions occur: 2, 1, 2, 3, 2, 2, 2 and 1. Let the free list and table look-up strategies employ the reflected Gray code for generating adjacent subcube addresses. Also, for the sake of simplicity, let the allocation policy be static. Table 1 shows the subcube

Table 1: Free list strategy allocation

Requested dimension	Free list	Allocated cube address
—	xxxxx	—
2-cube	1xxxx	000xx
	01xxx	
	001xx	
1-cube	1xxxx	0011x
	01xxx	
	0010x	
2-cube	1xxxx	011xx
	010xx	
	0010x	
3-cube	10xxx	11xxx
	010xx	
	0010x	
2-cube	10xxx	010xx
	0010x	
2-cube	100xx	101xx
	0010x	
2-cube	0010x	100xx
1-cube	—	0010x

addresses selected by the free list strategy. Table 2 shows the addresses of the subcubes allocated by the table look-up strategy under the condition of Proposition 3. To simplify the discussion, the table shows only the lists that deal with subcubes of dimension up to three, because the maximum dimension requested in this example is three. Assuming that requests arrive in consecutive cycles, the numbers in parentheses indicate when the corresponding subcubes were allocated. In contrast, the other numbers indicate when the corresponding subcubes became unavailable because either they are a part of larger subcubes allocated at that time or they contain smaller subcubes which are allocated at that time. It can easily be seen that the same subcubes are selected each time by both strategies.

The amount of memory space used by the table look-up strategy can be reduced considerably by deleting the busy processor count associated with every subcube in the lists and instead maintaining a set of global bits,

Table 2: Table look-up strategy allocation

1-cube list		2-cube list		3-cube list	
Address	Available	Address	Available	Address	Available
0000x	1	000xx	(1)	00xxx	1
0001x	1	001xx	2	01xxx	3
0011x	(2)	011xx	(3)	11xxx	(4)
0010x	(8)	010xx	(5)	10xxx	6
0110x	3	110xx	4		
0111x	3	111xx	4		
0101x	5	101xx	(6)		
0100x	5	100xx	(7)		
1100x	4				
1101x	4				
1111x	4				
1110x	4				
1010x	6				
1011x	6				
1001x	7				
1000x	7				

Table 3: Biased normal distribution, smallest dimension first

Response time	T	Average delay		Job completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	5.16	94.90	84.83	24.65	18.21
	200	0.00	10.59	195.00	174.04	25.11	18.55
	300	0.00	16.03	294.98	263.10	25.33	18.67
10	100	0.00	5.17	90.02	80.42	48.05	35.51
	200	0.00	10.60	190.04	169.56	49.63	36.62
	300	0.00	16.03	290.00	258.65	50.12	36.98
15	100	0.25	5.50	84.73	75.69	69.53	51.43
	200	0.29	11.06	184.27	164.12	73.20	53.86
	300	0.30	16.58	284.71	253.39	74.45	54.74
20	100	2.25	7.57	76.32	67.53	80.48	61.58
	200	4.86	16.19	171.12	149.99	84.99	64.01
	300	7.47	25.06	265.57	231.66	86.48	64.79
25	100	4.89	11.08	67.05	57.24	85.35	64.64
	200	10.34	23.90	156.70	131.68	90.00	67.27
	300	15.80	37.06	245.52	204.83	91.61	67.87

one available bit per PE in the system. The allocation procedure is then similar to the earlier strategy except that the new strategy checks for the availability of every PE in the available subcube address by checking the global bits of the included PEs. However, this modification increases the time complexity of the strategy.

4.3 An improved processor allocation strategy for MHs

The strategy of Subsection 4.2 is characterised by a perfect subcube recognition ability because it maintains lists for all possible subcube addresses. However, its space complexity is relatively high due to the look-up table created at static time. This section presents a new strategy which is based on the free-list strategy but incorporates parts of the buddy and table look-up strategies also in order to reduce the space complexity, while at the same time retaining the perfect subcube recognition ability.

As for the buddy strategy, 2^n bits are maintained to keep track of the availability of the 2^n nodes in the modified n-cube. Also, $n - 1$ free lists are maintained as in the free list strategy. Finally, lists similar to the ones used in the table look-up strategy are created, except that these lists contain only the addresses of subcubes containing repositioned links. The allocation and deallocation procedures follow.

Allocation:

Step 1: Determine the first available subcube from the free lists in such a way that none of the subcube's links is a link removed from the original hypercube (this can easily be determined by looking at the address of individual nodes, as described in Section 2). Allocate this subcube according to the free-list strategy and reset the availablity bits of all the PEs that form the subcube.

Step 2: If a subcube was not found in the free lists, then find the first subcube from the list of subcubes that contain repositioned links so that all the PEs in the subcube are available. Allocate this subcube in accordance to the table look-up strategy and reset the availablity bits of all the PEs forming the subcube.

Step 3: If no subcube of the requested dimension is available, then keep the request in the waiting queue.

Deallocation:

Step 1: If the subcube to be relinquished contains repositioned links, then apply the deallocation procedure of the table look-up strategy, else follow the deallocation steps of the free-list strategy.

Step 2: Set the individual availability bits of all PEs contained in the deallocated subcube.

The computation times of the allocation and deallocation procedures are $O(n + 2^k)$ and $O(n\ 2^n)$, respectively.

5 Simulation results

Our simulation procedure for MHs is similar to that used in References 3 and 9 for standard hypercubes. The results obtained for our improved processor allocation strategy are compared with those obtained for the buddy strategy. The following assumptions were made for the simulation. Incoming requests arrive in every time unit, while this process continues for a total of T time units. The residence times of subcube requests are assumed to be uniformly distributed. Since the direct application of the buddy strategy to MHs would result in subcubes of higher dimensions not being recognised, depending on the values of λ and n, the requested subcubes are restricted to relatively low dimensions. The dimensions of the subcubes required by incoming requests are assumed to have either uniform or biased normal distribution. A queue is maintained for every subcube dimension. Every time a new job arrives, it is pushed to the bottom of the respective queue. At every instance, if jobs are completed, the corresponding subcubes are released. Furthermore, all of the queues are checked for any waiting requests and if the corresponding subcubes are available, they are allocated. Servicing subcube requests can be done either from the highest dimension queue down to the smallest dimension queue, in which case the priority is highest dimension first, or in the reverse order according to the smallest dimension first priority. In either case, the priority within a given queue is first-come, first-served (FCFS).

The following data were collected by averaging the results over 100 independent runs of the simulation program: C, the number of requests that were assigned in time T; D, the total delay involving all requests until T; U, the total utilisation of the MH in time T. $U = \sum_{i=1}^{C} 2^{|I_i|} t_i$, where $|I_i|$ is the ith request size and t_i is the residence time until T of the request I_i.

The three performance measures that were obtained from the above data are the average waiting delay per request, the number of jobs completed in time T, and the efficiency of the strategy. The average waiting delay is given by D/T, the efficiency is given by $U/(T \times 2^n)$, and the job completion is directly given by C.

Table 4: Biased normal distribution, largest dimension first

Response time	T	Average delay		Job completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	5.16	94.90	84.83	24.65	18.21
	200	0.00	10.59	195.00	174.04	25.11	18.55
	300	0.00	16.03	294.98	263.10	25.33	18.67
10	100	0.00	5.17	90.02	80.42	48.05	35.31
	200	0.00	10.60	190.04	169.57	49.63	36.62
	300	0.00	16.03	290.00	258.65	50.21	36.98
15	100	0.18	5.38	84.80	75.85	69.79	51.58
	200	0.20	10.89	184.40	164.45	73.29	54.00
	300	0.20	16.36	284.78	253.78	74.54	54.85
20	100	2.16	7.16	76.25	68.09	84.37	63.69
	200	4.65	14.24	171.89	154.59	90.79	68.07
	300	6.75	21.08	267.89	241.10	93.27	69.73
25	100	6.72	12.18	63.36	54.91	88.40	66.64
	200	15.88	26.53	144.48	126.20	94.02	70.74
	300	25.97	41.14	222.72	196.87	95.98	72.15

Table 5: Uniform distribution, smallest dimension first

Response time	T	Average delay		Job completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	16.67	94.90	63.01	35.95	15.03
	200	0.00	33.56	195.00	129.08	36.58	15.27
	300	0.00	50.55	294.98	194.78	36.90	15.28
10	100	0.30	16.68	89.58	59.77	69.10	29.33
	200	0.45	33.56	189.36	125.78	71.55	30.13
	300	0.53	50.55	289.28	191.52	72.65	30.32
15	100	5.39	16.84	75.37	56.39	81.27	42.72
	200	11.51	33.78	162.83	122.00	84.17	44.51
	300	17.63	50.76	250.77	188.00	85.22	45.04
20	100	9.95	17.97	63.31	51.07	84.03	54.61
	200	21.43	36.01	140.01	115.02	86.04	56.06
	300	33.09	53.85	216.45	179.76	86.89	57.43
25	100	13.05	19.79	53.78	44.45	86.60	58.09
	200	27.55	40.79	125.48	102.46	88.76	61.16
	300	42.30	61.93	195.61	160.01	89.26	62.09

Table 6: Uniform distribution, largest dimension first

Response time	T	Average delay		Job completion		Efficiency	
		Table	Buddy	Table	Buddy	Table	Buddy
5	100	0.00	16.67	94.90	63.01	35.95	15.03
	200	0.00	33.56	195.00	129.08	36.58	15.27
	300	0.00	50.55	294.98	194.78	36.90	15.28
10	100	0.21	16.68	89.69	59.77	69.57	29.33
	200	0.23	33.56	189.76	125.78	72.05	30.13
	300	0.25	50.55	289.70	191.52	73.01	30.32
15	100	5.06	16.82	76.15	56.43	88.45	42.77
	200	11.03	33.73	163.31	122.05	93.63	44.52
	300	17.61	50.72	249.09	188.03	95.55	45.05
20	100	12.06	17.71	58.19	51.47	91.06	53.41
	200	27.23	35.08	127.88	117.18	95.38	57.45
	300	43.36	52.27	193.46	183.01	96.89	58.81
25	100	17.67	19.32	45.32	45.26	91.72	60.64
	200	39.06	38.75	102.23	106.92	95.77	66.01
	300	61.37	57.80	156.44	169.38	97.18	68.21

Although several simulation results were obtained, without loss of generality, simulation results are presented here for $n = 6$ and $\lambda = 3$. The requested subcubes are assumed to be of size ranging from 1 to 3. Table 3 shows the simulation results when the arriving requests are assumed to have a biased normal distribution; the column labelled 'table' represents the performance of our processor allocation algorithm. More specifically, the arriving requests have the following probabilities for subcube dimension: $p_1 = 0.5762$, $p_2 = 0.3142$ and $p_3 = 0.1096$, where p_k is the probability that the request is for a subcube of dimension k. The priority for subcube allocation is smallest dimension first. In the next set of runs, this priority is changed to largest dimension first and the results are shown in Table 4. Tables 5 and 6 show similar results when the arriving requests follow a uniform distribution (i.e. $p_1 = p_2 = p_3 = \frac{1}{3}$).

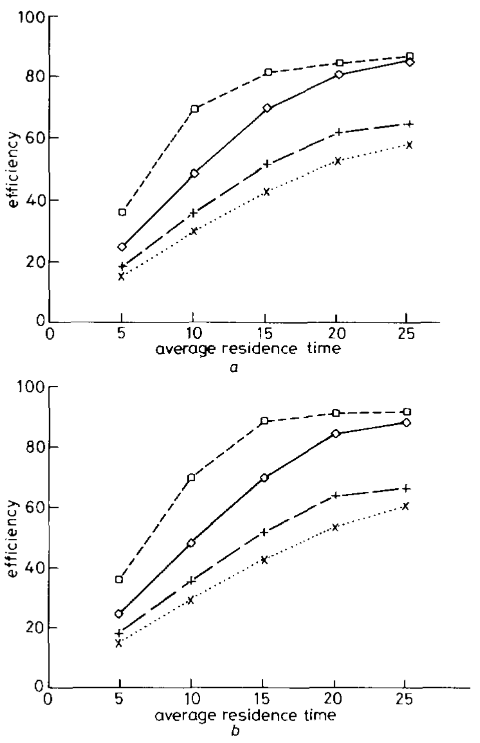


Fig. 3 *Comparison of our strategy with the buddy strategy, efficiency curves*

a smallest dimension first priority
b largest dimension first priority
—◇— Table, biased normal
-- + -- Buddy, biased normal
-- □ -- Table, uniform
··· × ··· Buddy, uniform

It can easily be seen that our strategy outperforms the buddy strategy. Plots of efficiency versus the average residence time for T equal to 100 are shown in Fig. 3. From the simulation results in Reference 9, it can be observed that for standard hypercubes the performance of other processor allocation strategies, like the Gray code, the modified buddy, and the free-list strategies varies 5–10% from that of the buddy strategy. Therefore, based on our simulation results, we may conclude that none of the other existing processor allocation strategies for standard hypercubes could outperform our proposed strategy for MHs.

6 Conclusions

Although the hypercube network is characterised by a set of very powerful properties, its major drawback is that it cannot be expanded in practice. In contrast, modified hypercubes have proved to be promising alternative hypercube-like networks prone to incremental growth techniques. The processor allocation strategies reported in the literature for standard hypercube networks work poorly on MHs. This was shown in this paper by simulating the buddy strategy on MHs. Two processor allocation strategies for MHs were introduced in this paper. The proposed strategies have a perfect subcube recognition ability and simulation results have shown that they are very efficient. It was shown that our strategies perform very well for standard hypercubes also.

7 References

1 AHMAD, I., and GHAFOOR, A.: 'Semi distributed task allocation strategy for large hypercube supercomputers'. Proceedings of *Supercomputing '90*, IEEE Computer Soc. Press, Los Almitos, California, Nov. 1990, pp. 898–907

2 CASAVANT, T.L., and KUHL, J.G.: 'Analysis of three dynamic load-balancing strategies with varying global information requirements'. Proceedings of *7th Intern. Conf. Distrib. Computing*. West Germany, April 1987, pp. 185–192

3 CHEN, M.S., and SHIN, K.G.: 'Processor allocation in an N-cube multiprocessor using Gray codes', *IEEE Trans.*, 1987, **C-36**, pp. 1396–1407

4 CHEN, M.S., and SHIN, K.G.: 'Subcube allocation of task migration in hypercube multiprocessors', *IEEE Trans.*, 1990, **C-39**, pp. 1146–1155

5 DUTT, S., and HAYES, J.P.: 'On allocating subcubes in a hypercube multiprocessor'. Proceedings of *3rd Conf. Hypercube*, 1988, pp. 801–810

6 DUTT, S., and HAYES, J.P.: 'Subcube allocation in hypercube computers', *IEEE Trans.*, 1991, **C-40**, pp. 341–352

7 GHOSE, K., and DESAI, K.R.: 'The HCN: a versatile interconnection network based on cubes'. Proceedings of *Supercomputing '89*, IEEE Computer Society and ACM SIGARCH, Reno, Nevada, 1989, pp. 426–435

8 KATSEFF, H.P.: 'Incomplete hypercubes', *IEEE Trans.*, 1988, **C-37**, pp. 604–608

9 KIM, J., DAS, C.R., and LIN, W.: 'A top-down processor allocation scheme for hypercube computers', *IEEE Trans., Parallel Distrib. Syst.*, 1991, **2**, pp. 20–30

10 ZIAVRAS, S.G.: 'On the problem of expanding hypercube-based systems', *J. Parallel and Distrib. Comput.*, 1992, **16**, pp. 41–53

11 ZIAVRAS, S.G.: 'RH: a versatile family of reduced hypercube interconnection networks', *IEEE Trans. Parallel Distrib. Syst.* (to be published)