Optimal and Suboptimal Processor Allocation for Hypercycle-based Multiprocessors

Nikitas J. Dimopoulos and Vassilios V. Dimakopoulos

Abstract—Allocating nodes in a concurrent computer system depends on the topology of the system. In this work, we present a number of processor allocation strategies for Hypercycle based concurrent systems. Hypercycles is a class of multidimensional interconnection networks which includes such widely used networks as the binary n-cubes, k-ary n-cubes, generalized hypercubes etc. The allocation strategies presented include a statically optimal first-fit allocation, a suboptimal-first fit, and strategies with extended search space through the inclusion of additional search lists formed by permuting the base through which a hypercycle is defined. For all these strategies, we examine their optimality and present simulation results characterizing their performance relative to each other.

I. Introduction

Message passing concurrent computers such as the Hypercube [17], Cosmic Cube [20], MAX [18], [19], consist of several processing

Manuscript received May 1992; revised June 21, 1993 and February 8, 1994. This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant OGP001337, by the Institute for Robotics and Intelligent Systems under the National Networks of Centers of Excellence Program, and by the Canadian Microelectronics Corporation.

The authors are with the Department of Electrical and Computer Engineering, University of Victoria, Victoria, BC V8W 3P6, Canada.

IEEE Log Number 9408136.

nodes that interact via messages exchanged over communication channels linking these nodes into one functional entity.

There are many ways of interconnecting the computational nodes, the Hypercube and Cosmic Cube, have adopted a regular interconnection pattern corresponding to a binary n-dimensional cube, while MAX adopts a less structured, yet unspecified topology. Binary n-cubes have been extensively studied mainly because of the nice properties of their structure. Recently, other more efficient topologies have been proposed, an example being the k-ary n-cubes ([9]). The networks we consider in this paper are a unification of most of these topologies under a single class.

Hypercycles [11]–[14] can be considered as products of "basic" graphs. The set of component "basic" graphs are circulants [5], ranging in complexity from simple rings to fully connected graphs. Hypercycles are generalizations of many popular interconnection networks including binary n-cubes, k-ary n-cubes, generalized hypercubes [4], rings, toruses, etc.

Many properties and algorithms used for example in routing and processor allocation can be extended to the entire class of hypercycles making it possible to choose a topology that best suits the system requirements of a specific class of applications.

In this work, we are examining the problem of processor allocation for hypercycle-based multiprocessors. A number of different allocation strategies have been proposed for the hypercube multiprocessors. Chen and Shin [6] proposed a strategy that searches a list of allocation bits corresponding to the nodes of the multiprocessor. Nodes whose addresses have identical most significant bits, belong to the same subcube and can be allocated to a request. The allocation strategy is similar to the well known buddy-strategies, and has been proven to be statically optimal. The use of gray codes to form the addresses of nodes [6] doubles the number of recognized subcubes over the buddy strategy. The authors also proposed the utilization of more than one distinct gray codes to recognize more subcubes, $O(\binom{n}{\lfloor n/2 \rfloor})$ being the number of gray codes necessary to recognize all the possible subcubes in an n-cube. This strategy of performing a linear search over a list of 2^n allocation bits has a complexity of $O(k2^n)$ if k gray codes are employed. Another family of bit-mapped strategies is given in [2] allowing a more complex search over the allocation bits, resulting in improved subcube recognition. The simplest of the algorithms recognizes $(n-k+1)$ as many k-subcubes as the buddy strategy does and has an estimated complexity of $O(n2^n)$ in the worst case. The extended buddy tree strategy presented in [1] genaralizes some of the search strategies presented in [2] by controling how deep in the buddy tree the search for a free subcube is allowed to proceed. Mutliple lists, as a means for parallel allocation, are also suggested in [1] and complete subcube recognition can be had by adjusting the number of trees each processor is responsible for, and the search depth parameter. A best-fit allocation is also considered in [1].

A strategy that does not utilize allocation bits was proposed in [15]. A list of all the free subcubes is used instead, decomposing free subcubes to smaller ones if there is not a matching subcube during the allocation of a request, and coalescing freed nodes to form higher dimension subcubes during relinquishment. The objective of the actions taken is to keep a list with the largest number of high-dimensioned subcubes, called maximal set of subcubes (MSS). Finding an MSS is proved in [15] to be NP-hard so approximating algorithms were proposed that reduce the complexity to $O(n2^n)$ in the worst case. The free-list strategy proposed in [16] keeps separate $(n+1)$ lists, list k contains the free k-subcubes. Allocation is done the same way as in [15] but relinquishment is much faster due to a simpler coalescing scheme.The free-list strategy is thus faster (but does not keep an MSS as in [15]), statically optimal and has complexity of $O(n2^n)$. Finaly, the strategy presented in [8] achieves complete subcube recognition by a means of tree-collapsing transforms, is statically optimal and on the average is usually faster than the multiple gray codes and the free-list strategies.

In this work we are proposing several allocation strategies applicable to the general class of hypercycles. These strategies include a generalization of the buddy allocation scheme for the case of the hypercycles and the proof of its static optimality, the introduction of a nonstatically optimal strategy that exhibits a better performance for dynamic loads as well as the inclusion of additional search lists, based on permutations of the hypercycle base, for both the optimal and the nonstatically optimal strategies. We have also identified the permutation producing the most efficient allocation. The results are directly applicable to any of the subclasses of hypercycles, so in essence the proposed strategies are actually allocation strategies for the k-ary n-cubes, the generalized hypercubes, etc.

Thus, Section II introduces the Mixed Radix System, and presents a brief introduction to hypercycles and the routing policies used. Section III introduces the first-fit allocation strategy and proves that it is statically optimal. In addition, a nonstatically optimal first-fit allocation is also introduced in Section III. In Section IV we present a number of alternative startegies that use more than one allocation lists and we derive the more efficient one for the case of uniform hypercyles. Section V discusses the simulation experiments performed to characterize the allocation strategies proposed, while Section VI concludes the work.

II. Introduction to Hypercycles

A. Mixed Radix Number System

The mixed radix representation [3], is a positional number representation, and it is a generalization of the standard b-base representation, in that it allows each position to follow its own base independently of the others.

Given a decimal number M factored into n factors as $M = m_n \times m_{n-1} \times \cdots \times m_1$ then any number $0 \le X \le M - 1$ can be represented as

$$(X)_{m_n m_{n-1} \cdots m_1} = x_n x_{n-1} \cdots x_1 |_{m_n m_{n-1} \cdots m_1}$$

where $0 \le x_i \le (m_i - 1); i = 1, 2, \cdots, n$. The x_i's are chosen so that

$$X = \sum_{i=1}^{n} x_i w_i$$

where $w_i = m_{i-1} m_{i-2} \cdots m_1$, and $w_1 = 1$.

B. Hypercycles

An n-dimensional hypercycle is the regular undirected graph $\mathcal{G}^{\rho}_{\boldsymbol{m}} = \{\mathcal{N}^{\rho}_{\boldsymbol{m}}, \mathcal{E}^{\rho}_{\boldsymbol{m}}\}$, where

$$\boldsymbol{m} = m_n, m_{n-1}, m_{n-2}, \cdots, m_1.$$

is a mixed radix constituting the basis of the hypercycle,

$$\rho = \rho_n, \rho_{n-1}, \cdots, \rho_{n-2i}; \rho_i \le m_1/2,$$

is the connectivity vector, determining the connectivity in each dimension which ranges from a cycle ($\rho_i = 1$) to fully connected ($\rho_i = \lfloor m_i/2 \rfloor$), and

$\mathcal{N}^{\rho}_{\boldsymbol{m}} = \{0, 1, 2, \cdots, M - 1\}$ is the set of nodes.

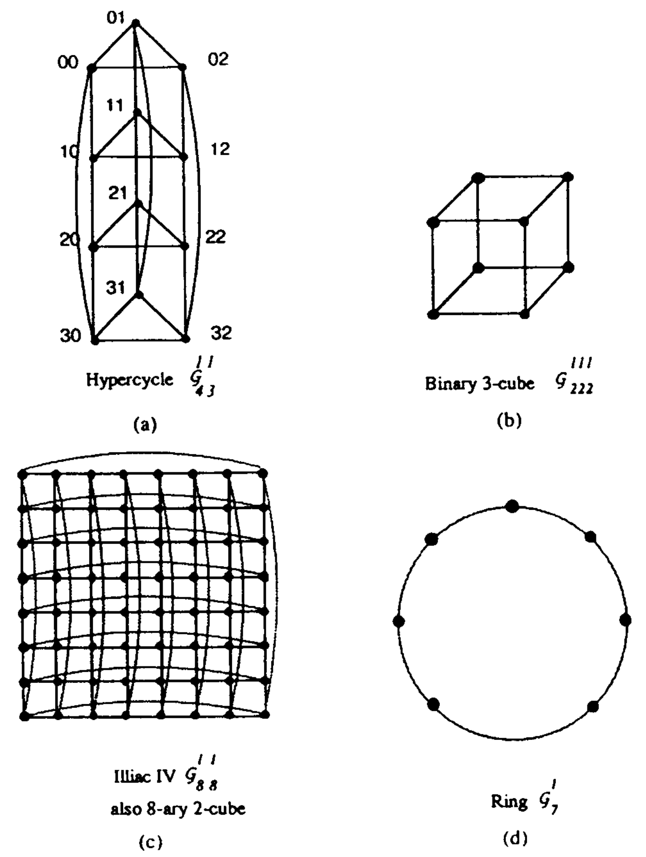
01
00
02
11
10
12
21
20
22
31
30
32
Hypercycle $\mathcal{G}^{11}_{43}$
(a)
Binary 3-cube $\mathcal{G}^{111}_{222}$
(b)
Illiac IV $\mathcal{G}^{11}_{88}$
also 8-ary 2-cube
(c)
Ring $\mathcal{G}^{1}_{7}$
(d)

Fig. 1. Examples of hypercycles.

The edge set is defined as follows: given $\alpha, \beta \in \mathcal{N}^{\rho}_{\boldsymbol{m}}$ then $(\alpha, \beta) \in \mathcal{E}^{\rho}_{\boldsymbol{m}}$ if and only if there exists $1 \le j \le n$ such that

$\beta_j = (\alpha_j \pm \xi_j) \bmod m_j$ with $1 \le \xi_j \le \rho_j$ and $\alpha_i = \beta_i; i \ne j$.

Hypercycles, have degree d and diameter k given by [11]

$$d = \sum_{i=1}^{n} f(m_i, \rho_i) \quad \text{where} \quad f(m_i, \rho_i)$$
$$= \begin{cases} 2\rho_i & \text{if } 2\rho_i < m_i \\ m_i - 1 & \text{if } 2\rho_i = m_i \end{cases}$$
$$k = \sum_{i=1}^{n} \left\lceil \frac{\lfloor m_i/2 \rfloor}{\rho_i} \right\rceil.$$

Hypercycles include the rings as one-dimensional cases. They also include the binary n-cubes ($M = 2 \times 2 \times \cdots \times 2 = 2^n$ and $\rho = 1, 1, 1, \cdots, 1$). The k-ary n-cubes are hypercylces with $M = k \times k \times \cdots \times k = k^n$ and $\rho = 1, 1, 1, \cdots, 1$. Finally, the generalized hypercubes have fully connected dimensions, i.e. $\rho_i = \lfloor m_i/2 \rfloor$ for every i. Fig. 1 shows some example hypercycles.

C. Routing in Hypercycles

Before presenting the routing equations of hypercycles, we would like to emphasize the importance of routing in the allocation procedure. The processors allocated to a job should be able to communicate efficiently which means that they should be as close as possible to each other and that other processors in the network should not interfere with the former ones. These conditions are met when the allocated portion of the network is (in the terminology of the next section) closed under the routing scheme.

Hypercycles, have routing properties that are similar to those of the n-cube. Given nodes $(\alpha)_{m_n m_{n-1} \cdots m_i \cdots m_1} = \alpha_n \alpha_{n-1} \cdots \alpha_i \cdots \alpha_1$

and $(\alpha^*)_{m_n m_{n-1} \cdots m_i \cdots m_1} = \alpha_n \alpha_{n-1} \cdots \xi \cdots \alpha_1$, a walk, from node α to node α^*, can be constructed as follows: $\alpha_n \alpha_{n-1} \cdots \alpha_i \cdots \alpha_1, \alpha_n \alpha_{n-1} \cdots \xi_1 \cdots \alpha_1, \alpha_n \alpha_{n-1} \cdots \xi_2 \cdots \alpha_1, \cdots, \alpha_n \alpha_{n-1} \cdots \xi \cdots \alpha_1$, such that[1]

$$\xi_{j_i+1} = \begin{cases} (\xi_{j_i} + \rho_i) \bmod m_i \\ \quad \text{if } [(\xi - \xi_{j_i}) \bmod m_i = |\xi_{j_i}, \xi|] > \rho_i & \text{(a)} \\ (\xi_{j_i} + |\xi_{j_i}, \xi| \bmod \rho_i) \bmod m_i \\ \quad \text{if } [(\xi - \xi_{j_i}) \bmod m_i = |\xi_{j_i}, \xi|] > \rho_i \\ \qquad \text{and } |\xi_{j_i}, \xi| \bmod \rho_i \neq 0 & \text{(b)} \\ (\xi_{j_i} - \rho_i) \bmod m_i \\ \quad \text{if } [(\xi_{j_i} - \xi) \bmod m_i = |\xi_{j_i}, \xi|] > \rho_i & \text{(c)} \\ (\xi_{j_i} - |\xi_{j_i}, \xi| \bmod \rho_i) \bmod m_i \\ \quad \text{if } [(\xi_{j_i} - \xi) \bmod m_i = |\xi_{j_i}, \xi|] > \rho_i \\ \qquad \text{and } |\xi_{j_i}, \xi| \bmod \rho_i \neq 0 & \text{(d)} \\ \xi \\ \quad \text{if } |\xi_{j_i}, \xi| \leq \rho_i & \text{(e)} \end{cases}$$

$$\xi_0 = \alpha_i \qquad \xi_{\max} = \xi. \tag{2.1}$$

Equation (2.1) defines all the minimum-length paths from a source to a destination in a single dimension. Parts (a) and (c) constitute a greedy strategy where the maximum step towards the destination is taken. Parts (b) and (d) form alternate paths by allowing the step described in part (e) to be taken earlier.

Given an origin $(\alpha)_{m_n m_{n-1} \cdots m_1} = \alpha_n \alpha_{n-1} \cdots \alpha_1$ and a destination $(\beta)_{m_n m_{n-1} \cdots m_1} = \beta_n \beta_{n1} \cdots \beta_1$ then distinct walks of minimum length that connect them are constructed by sequentially modifying the source address, each time substituting a source digit by an intermediate walk digit determined according to (2.1), until the destination is formed. The following walk connects α to β

$$\begin{aligned} \alpha = \alpha_n \cdots \alpha_3 \alpha_2 \alpha_1; \alpha_n \cdots \alpha_3 \xi_1 \alpha_1; \alpha_n \cdots \psi_1 \xi_1 \alpha_1; \\ \alpha_n \cdots \psi_1 \xi_2 \alpha_1; \alpha_n \cdots \psi_1 \xi_2 \alpha_1; \\ a_n \cdots \psi_2 \xi_2 \alpha_2; \cdots; \alpha_n \cdots \beta_3 \xi_2 \alpha_1; \\ \cdots; \beta_n \cdots \beta_3 \beta_2 \beta_1 = \beta. \end{aligned}$$

In Fig. 1(a), one can recognize the following distinct walks of equal length, that connect source 01 to destination 20: {01: 00: 10: 20}. {01; 11; 10; 20}. {01: 11: 21: 20}. {01: 31: 21: 20}. {01: 31: 30: 20}. {01; 00; 30; 20}.

In interconnection networks, deadlocks must be prevented or avoided. Routing such as the e-cube, which is applicable to hypecubes, prevents deadlocks by ordering the resources (i.e., communication links) to be allocated and preserving the order during allocation. Also, various forms of two-phase locking [22] can be employed which avoid deadlocks dynamically by forcing a blocked path to backoff [7].

We have developed a number of backing-off and backtracking strategies applicable to the entire class of hypercycles. In addition, we have generalized a form of the e-cube routing and made it applicable to a subset of the hypercycles which includes hypercubes, generalized hypercubes, some k-ary n-cubes and some dense rings and toruses. Details are reported in [12]–[14].

III. Processor Allocation

In this section, we shall investigate the problem of allocating processors in a hypercycle-based concurrent computer to incoming jobs. The hypercycle in question has M nodes and is characterized by the mixed radix base, $m_n, m_{n-1}, \cdots, m_1$. For our purposes, a job is considered as a collection of tasks which can execute concurrently. A job therefore requires a portion of the processors in order to execute.

[1] We define $|a, b| = \min\{(a - b) \bmod m_i, (b - a) \bmod m_i\}$

We consider a sequence of requests corresponding to a sequence of incoming jobs, each requesting a minimum number of nodes. We shall use a first-fit strategy, where the first unallocated fragment that can accommodate the request is assigned to the incoming job. Fragments generalize the notion of the subcube and as we shall see, fragments can accommodate all communication needs generated by member nodes internally, localizing thus jobs and preventing them from interfering with other jobs running in other parts of the system. We shall prove that this strategy is statically optimal, and it requires minimal searching. Our strategy is a generalization of a similar one reported in [6] for hypercube multiprocessors.

First, we shall define the notion of a fragment, introduce an ordering relation which arranges the nodes of the hypercycle, and establish that fragments can indeed accommodate all communication requests internally.

Definition 1: Given two nodes $b = b_n b_{n-1} \cdots b_1$ and $\check{b} = \check{b}_n \check{b}_{n-1} \cdots \check{b}_1$ we say that b *lies before* $\check{b}$ and denote $b \prec \check{b}$ iff there exists an index $1 \leq k \leq n$ such that $b_k < \check{b}_k$ and $b_i = \check{b}_i : i = n, n-1, \cdots, k+1$.

The above defined ordering relation arranges the set of nodes in a unique sequence.

Definition 2: A *region* $\mathcal{R}$ is defined as a set of consecutive nodes, i.e., $\mathcal{R}(a, b) \equiv [a, b] = \{v | a \prec v \prec b\} \cup \{a, b\}$.

Definition 3: Given two regions $\mathcal{R}$ and $\check{\mathcal{R}}$ we say that $\mathcal{R}$ *lies before* $\check{\mathcal{R}}$ and denote $\mathcal{R} \prec \check{\mathcal{R}}$ iff for every node b in $\mathcal{R}$ and every node $\check{b}$ in $\check{\mathcal{R}}$ we have $b \prec \check{b}$.

In the subsequent, we shall use the symbol * within a sequence of factors, to indicate that the corresponding factor may attain any of the allowed m values $(0, 1, \cdots m-1)$. Similarly, the notation $*^k$ indicates a sequence of k *, i.e., a sequence of k factors each attaining any of the allowed values. This notation is very useful in defining sets of nodes. For example, with reference to hypercycle $\mathcal{G}^{111}_{532}$, $3*^2$ denotes the set of nodes $\{300, 301, 310, 311, 320, 321\}$.

Definition 4: A *fragment* is a region whose nodes have identical most significant digits. Nodes with identical most significant digits belong to the same fragment. We denote a fragment as

$$\begin{aligned} \mathcal{F}(b_n b_{n-1} \cdots b_{k+1}) \equiv b_n b_{n-1} \cdots b_{k+1} *^k = \\ \{b | b = b_n b_{n-1} \cdots b_{k+1} \beta_k \cdots \beta_1, \\ \beta_j \in \{0, 1, \cdots m_j - 1\}; \\ j = k, k-1, \cdots, 1\}. \end{aligned}$$

Definition 5: The fragments $\{b_n b_{n-1} \cdots b_{k+2} \beta_{k+1} *^k | \beta_{k+1} \in \{0, 1, \cdots m_{k+1} - 1\}\}$ that comprise the fragment $\mathcal{F}(b_n b_{n-1} \cdots b_{k+2})$ are called *buddies*.

Definition 6: A subgraph is *closed* under the hypercycle routing as defined above, iff all the intermediate nodes required to complete a path between a source and a destination are nodes of the subgraph.

Allocating closed potions of a graph ensures locality of communication, i.e., all the links needed to form a path between a source and a destination are inside the allocated subgraph. This way, tasks do not interfere with each other as long as they are allocated different subgraphs. For n-cubes it can be easily shown that subcubes are closed under the e-cube routing. For the general class of hypercycles we have the following equivallent result.

Lemma 1: Fragments are closed under the hypercycle routing.

To see why this is true, let $\alpha = b_n b_{n-1} \cdots b_{k+1} \alpha_k \cdots \alpha_1$ and $\beta = b_n b_{n-1} \cdots b_{k+1} \beta_k \cdots \beta_1$ be two nodes in the fragment $\mathcal{F}(b_n b_{n-1} \cdots b_{k+1})$. According to the routing (2.1), any intermediate node x on the path from α to β will be of the form $\xi = b_n b_{n-1} \cdots b_{k+1} \xi_k \cdots \xi_1$ and, by definition, it will belong to $\mathcal{F}(b_n b_{n-1} \cdots b_{k+1})$.

A. First-Fit Processor Allocation in Hypercycles

Because of the ordering relation $\prec$ presented in Definition 1 above, one can arrange the nodes of any hypercycle in a linear fashion and number them from 0 to $M-1$. The first-fit allocation strategy utilizes a list of allocation bits numbered from 0 to $M-1$ and corresponding to the nodes in the hypercycle. If a processor has been allocated to a job, its corresponding allocation bit is set to 1, otherwise it is cleared to 0.

The first-fit allocation strategy searches the list until it discovers a region of unallocated processors corresponding to a fragment with a size equal to that of the request. If such a region is found, the corresponding allocation bits are set to 1, and the strategy tries to accommodate a subsequent request. The allocation bits are cleared as soon as the job assigned to the corresponding nodes is terminated.

In detail, with reference to a hypercycle, the first-fit allocation strategy can be described as follows.

1) First-Fit Allocation Strategy:

1) Let $|f_j|$ be the size of the smallest fragment that will fit the jth request I_j.
2) Find the least integer m such that all the allocation bits in the region $[m|f_j|, (m+1)|f_j|-1]$ are zeroes, and set all the allocation bits in this region to 1.
3) Allocate nodes with addresses in the region to request I_j.

2) Processor Relinquishment:

1) Reset the allocation bits of the released region to zero.

The first-fit strategy has worst-case complexity of $O(M)$ where M is the number of processors in the hypercycle. This is because every allocation bit is checked only once and at most all the M allocation bits have to be checked.

Definition 7: If requests are constrained to have sizes equal to those of fragments, an allocation strategy is *statically optimal* if it can accommodate any sequence of requests $\{I_r\}_{r=1}^{j}$ such that $\Sigma_{r=1}^{j}|I_r| \leq m_n \times m_{n-1} \times \cdots \times m_1 = M$.

In what follows we adopt the terminology used in [6] and prove that the above algorithm is statically optimal.

Definition 8: A fragment $\mathcal{F}(b_n b_{n-1}\cdots b_{k+1}) \equiv b_n b_{n-1}\cdots b_{k+1}*^k$ is said to be a *hole* if and only if this region is available, but at least one of the fragments[2] $b_n b_{n-1}\cdots[b_{k+1}]_i *^k$ is not.

Let $\{h_i(j)\}_{i=1}^{u}$ denote the sequence of holes resulting from the allocation of fragments to the sequence of requests $\{I_r\}_{r=1}^{j}$. Furthermore, the sequence is arranged in lexicographical order so that the ith and kth holes, $h_i(j)$ and $h_k(j)$, satisfy $h_i(j) \prec h_k(j)$ if $i<k$.

Lemma 2: If $h_i(j) = b_n b_{n-1}\cdots b_{k+1}*^k$ for some i, then $b_{k+1} \neq 0$.

Proof: Suppose that $b_{k+1}=0$. Then the region $b_n b_{n-1}\cdots 0*^k$ is a hole, which means that at least one of the fragments $b_n b_{n-1}\cdots[0]_i *^k$ is unavailable. But this contradicts the first-fit search of the allocation strategy. ■

Lemma 3: Given a sequence of holes $\{h_i(j)\}_{i=1}^{u}$ resulting from the sequence of requests $\{I_r\}_{r=1}^{j}$, we will have:

1) $|h_i(j)| \leq |h_{i+1}(j)|; i<u$
2) If there exists i such that $|h_i(j)| < |h_{i+1}(j)|$ then $\Sigma_{\rho=1}^{i}|h_\rho(j)| < |h_{i+1}(j)|$

Proof:

1) Suppose that there exists i such that $h_1 = |h_i(j)| > |h_{i+1}(j)| = h_2$. This means that hole $h_{i+1}(j) = b_n b_{n-1}\cdots b_{k+1}*^k$ is located after hole $h_i(j) = \tilde{b}_n\tilde{b}_{n-1}\cdots\tilde{b}_{k+\sigma+1}*^{k+\sigma} (\sigma \geq 1)$. Since the fragment $b_n b_{n-1}\cdots b_{k+1}*^k$ is a hole, then the fragment $b_n b_{n-1}\cdots 0*^k$ of size h_2 is occupied. But this is contrary to the first-fit allocation strategy, since the fragment $\tilde{b}_n\tilde{b}_{n-1}\cdots\tilde{b}_{k+\sigma+1}0\cdots0*^k$ of size h_2 lying before $b_n b_{n-1}\cdots 0*^k$ is available (as part of hole $h_i(j)$).
2) Given the hole $h = b_n b_{n-1}\cdots b_{k+1}*^k$ of size $|h| = m_k \times m_{k-1} \times \cdots \times m_1$, observe that it is located in the region $\mathcal{R} = b_n b_{n-1}\cdots b_{k+2}*^{k+1}$ and by Lemma 2, the fragment $\mathcal{F} = b_n b_{n-1}\cdots b_{k+2}0*^k$ has been allocated. If there was a hole of size $|h|$ before the region $\mathcal{R}$, then the fragment $\mathcal{F}$ should not have been allocated since its allocation would have contradicted the first-fit allocation strategy. Similarly, if there is a hole of size $|h|$ in a region $\mathcal{R}' = b'_n b'_{n-1}\cdots b'_{k+2}*^{k+1}$ which lies after region $\mathcal{R}$, then, by Lemma 2, the fragment $\mathcal{F}' = b'_n b'_{n-1}\cdots b'_{k+2}0*^k$ is allocated. But this is also contrary to the first-fit strategy since the hole h is before the fragment $\mathcal{F}'$. The conclusion is that there exist at most $m_{k+1}-1$ other holes of size $|h|$. Thus if $h_{i+1}(j) = b_n b_{n-1}\cdots b_{k+1}*^k$ is a hole, there exist at most

$$(m_1-1) \text{ holes of size } 1$$
$$(m_2-1) \text{ holes of size } m_1$$
$$\vdots$$
$$(m_k-1) \text{ holes of size } m_{k-1}\times\cdots\times m_1$$

while the size of $h_{i+1}(j)$ is $|h_{i+1}(j)| = m_k \times m_{k-1}\times\cdots\times m_1$. Thus

$$\sum_{\rho=1}^{i}|h_\rho(j)| \leq \sum_{\rho=1}^{k}(m_\rho - 1)\prod_{\lambda=1}^{\rho-1} m_\lambda$$
$$= m_k \times m_{k-1}\times\cdots\times m_1 - 1$$
$$< m_k \times m_{k-1}\times\cdots\times m_1 = |h_{i+1}(j)|$$

■

Theorem 1: The first-fit allocation strategy is statically optimal.

Proof: Let $\{I_r\}_{r=1}^{j}$ be a sequence of requests such that $\Sigma_{r=1}^{j}|I_r| \leq m_n \times m_{n-1}\times\cdots\times m_1 = M$, where M is the number of nodes in the hypercycle. We will prove the theorem by induction on j. If we had only one request, it would trivially be allocated as long as it requires at most M nodes. Assume now that the sequence of requests $\{I_r\}_{r=1}^{j-1}$ can be accommodated by the first-fit strategy, and denote by $\{h_i(j-1)\}_{i=1}^{u}$ the sequence of holes resulted from this allocation. We shall prove that $\{I_r\}_{r=1}^{j}$ can also be accommodated, i.e. $|h_u(j-1)| \geq |I_j|$, provided that $\Sigma_{r=1}^{j}|I_r| \leq M$.

1) Assume that $|h_u(j-1)| > |h_{u-1}(j-1)|$. Then, by Lemma 3,

$$\sum_{\rho=1}^{u-1}|h_\rho(j-1)| < |h_u(j-1)| \tag{3.1}$$

Also, the number of nodes allocated plus the number of nodes still unallocated (i.e., in holes) equals the total number of nodes in the network, that is,

$$\sum_{\lambda=1}^{u}|h_\lambda(j-1)| + \sum_{r=1}^{j-1}|I_r| = M \Rightarrow$$
$$|h_u(j-1)| + \sum_{\lambda=1}^{u-1}|h_\lambda(j-1)| + \sum_{r=1}^{j-1}|I_r| = M \Rightarrow$$
$$|h_u(j-1)| + \sum_{\lambda=1}^{u-1}|h_\lambda(j-1)|$$
$$= M - \sum_{r=1}^{j-1}|I_r| \geq |I_j| \tag{3.2}$$

[2] $[b_{k+1}]_i$ denotes the ith element of the set $[b_{k+1}] = \{0, 1, \cdots, m_{k+1}-1\} - \{b_{k+1}\}$.

because of the assumption $\Sigma_{r=1}^{j}|I_r| \geq M$. Combining (3.1) and (3.2), one obtains

$$2|h_u(j-1)| > |I_j|$$

Since both $h_u(j-1)$ and I_j are fragments, and since there are no fragments available such that $2|h_u(j-1)| > |I_j| > |h_u(j-1)|$, then $|I_j| \leq |h_u(j-1)|$.

2) Assume that $|h_u(j-1| = |h_{u-1}(j-1)|$ and denote by a the number of holes in the region where holes $|h_u(j-1|$ and $|h_{u-1}(j-1)|$ are located. If $h_u(j-1) = b_n b_{n-1}\cdots b_{n_u}{*}^{n_u-1}$, then from the definition of a hole and footnote 2, $a < m_{n_u}$. Then, in the sequence of holes $\{h_i(j-1)\}_{i=1}^{u}$ produced by the first $(j-1)$ requests, there are a holes of equal size followed by zero or more smaller holes. Thus,

$$|h_u(j-1)| = |h_{u-1}(j-1) = \cdots = |h_{u-(a-1)}(j-1)| > $$
$$> |h_{u-(a-1)-1}(j-1)| \geq \cdots \geq |h_1(j-1)|$$

Lemma 3, gives

$$\sum_{\lambda=1}^{u-(a-1)-1} |h_\lambda(j-1)| < |h_{u-(a-1)}(j-1)|. \qquad (3.3)$$

Also, because the sum of all the holes and allocated fragments sums up to the number of nodes in the hypercycle, we have

$$\sum_{\lambda=1}^{u}|h_\lambda(j-1)| + \sum_{r=1}^{j-1}|I_r| = M \Rightarrow$$
$$\sum_{\lambda=u-(a-1)}^{u} |h_\lambda(j-1)|$$
$$+ \sum_{\lambda=1}^{u-(a-1)-1} |h_\lambda(j-1)| + \sum_{r=1}^{j-1}|I_r|$$
$$= M \Rightarrow a|h_u(j-1)| + \sum_{\lambda=1}^{u-(a-1)-1} |h_\lambda(j-1)|$$
$$= M - \sum_{r=1}^{j-1}|I_r| \geq |I_j|. \qquad (3.4)$$

because of the assumption $\Sigma_{r=1}^{j}|I_r| \leq M$. Combining now (3.3) and (3.4), we obtain

$$(a+1)|h_u(j-1)| > |I_j|$$

Since both $h_u(j-1)$ and I_j are fragments, and since there is no fragment $\mathcal{F}$ available such that $m_{n_u}|H_u(j-1)| > |\mathcal{F}| > |h_u(j-1)|$, then $|I_j| \leq h_u(j-1)|$.

■

B. Nonstatically Optimal First-Fit Allocation

The first-fit allocation as described in Section III-A above is statically optimal, but it allocates processors in fragments. These fragments (depending on the population of nodes in each dimension) offer a limited way to partition the hypercycle, and thus waste resources by allocating the next larger fragment to a given request. The next larger fragment is normally larger than twice the size of the immediately smaller fragment.

In order to alleviate this problem, we are proposing a *nonstatically optimal first-fit* allocation strategy which can allocate portions of a fragment (called *segments*). In this section, we shall describe the nonstatically optimal first-fit allocation strategy, and show that this strategy is not statically optimal. Yet, simulations as presented in Section V, have determined that it outperforms the optimal first-fit allocation under dynamic loading, while retaining similar complexity.

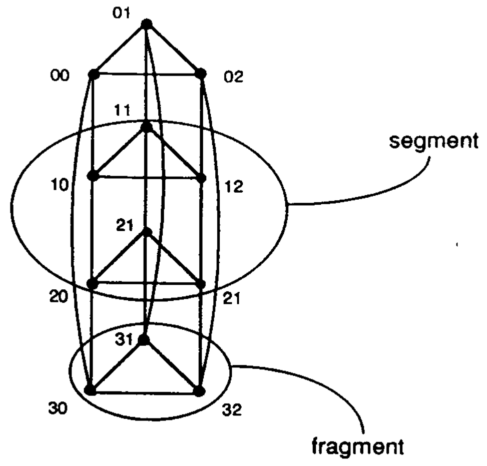

Fig. 2. Fragments and segments in a hypercycle.

Definition 9: A *segment* is a subset of a fragment and is defined as

$$\mathcal{S}(b_n b_{n-1}\cdots b_{k+1}[l_k, r_k])$$
$$= \{b | b = b_n b_{n-1}\cdots b_{k+1}\sigma_k \beta_{k-1}\cdots\beta_1;$$
$$l_k \leq \sigma_k \leq r_k, \beta_j \in \{0, 1, \cdots m_j - 1\};$$
$$j = k-1, \cdots, 1\},$$

where $l_k < r_k$ and

$$r_k - l_k < \begin{cases} \frac{m_k}{2}, & \text{if } \rho_k < \left\lfloor \frac{m_k}{2} \right\rfloor \\ m_k + 1, & \text{otherwise} \end{cases}.$$

A segment is a collection of contiguous fragments of the immediately lower dimension; it includes $(r_k - l_k)$ fragments of dimension $k-1$. Observe that if $\rho_k = \lfloor m_k/2 \rfloor$, then full connectivity exists in dimension k and a segment may include any number of the m_k $(k-1)$-dimensional fragments. In the case that dimension k is not fully connected, a segment may include up to only half this number. This is a necessary limmitation in order to maintain closure for segments under the hypercycle routing.

Fig. 2 further illustrates the notions of a segment and a fragment. There, the following sets of nodes are examples of fragments $\mathcal{F}_0 = \{00, 01, 02, 10, 11, 12, 20, 21, 22, 30, 31, 32\}, \mathcal{F}_1 = \{30, 31, 32\}$, while $\{10, 11, 12, 20, 21, 22\}$ is a segment of $\mathcal{F}_0$, and $\{30, 31\}$ is a segment of $\mathcal{F}_1$.

Lemma 4: Segments are closed under the hypercycle routing.

Proof: Let $\alpha = b_n b_{n-1}\cdots b_{k+1}\alpha_k\cdots\alpha_1$ and $\beta = b_n b_{n-1}\cdots b_{k+1}\beta_k\cdots\beta_1$ be two nodes in the segment $\mathcal{S}(b_n b_{n-1}\cdots b_{k+1}[l_k, r_k])$. According to the routing (2.1), any intermediate node ξ on the path from α to β will be of the form $\xi = b_n b_{n-1}\cdots b_{k+1}\xi_k\cdots\xi_1$. In the case where $\rho_k < \lfloor m_k/2 \rfloor$, since $l_k \leq \alpha_k, \beta_k \leq r_k$ and $r_k - l_k < m_k/2$, we must have $l_k \leq \xi_k \leq r_k$. In case of full connectivity, i.e. $\rho_k = \lfloor m_k/2 \rfloor$, then necessarily $l_k \leq \xi_k = \beta_k \leq r_k$. Thus $\xi \in \mathcal{S}(b_n b_{n-1}\cdots b_{k+1}[l_k, r_k])$ always.

■

The nonstatically optimal first-fit allocation, searches a list of linearly arranged nodes of the hypercycle, in a similar fashion as the first-fit discussed in Section III–A above. It searches the allocation list until it discovers a region of unallocated processors corresponding to the smallest segment capable of accommodating the request. If such a region is found, the corresponding allocation bits are set to 1, and the strategy tries to accommodate a subsequent request. The allocation bits are cleared as soon as the job assigned to the corresponding nodes is terminated.

In detail, the nonstatically optimal first-fit allocation strategy can be described as follows. Note that since all the allocation bits may

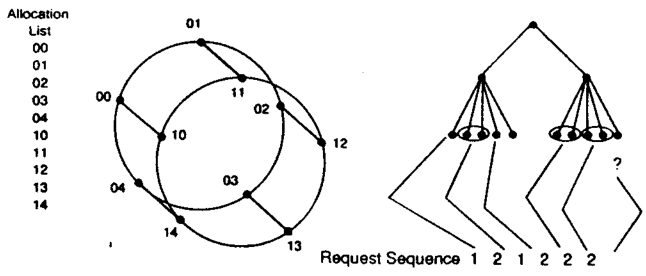

Fig. 3. Counter example for the nonstatically optimal first-fit strategy.

be checked, the worst case complexity of the algorithm is, as in the first-fit case, $O(M)$.

Nonstatically Optimal First-Fit Allocation

1) Let $|s_j|$ be the size of the smallest segment comprising r fragments of size $|f|$ that will fit the jth request I_j. Denote by $|f^*|$ the size of the immediately larger fragment that contains the segment s_j.
2) Find the least integers m and k such that $k|f| + |s_j| \leq |f^*|$ and all the allocation bits in the region $[m|f^*| + k|f|, m|f^*| + k|f| + |s_j| - 1]$ are zeroes, and set all the allocation bits in this region to 1.
3) Allocate nodes with addresses in the region to request I_j.

Processor Relinquishment:

1) Reset the allocation bits of the released region to zero.

Theorem 2: If requests are limmited to a segment, the nonstatically optimal first-fit allocation strategy is not statically optimal.

Proof: Through a counter example. As depicted in Fig. 3, given the hypercycle $\mathcal{G}_{25}^{11}$ and the sequence of segment requests $\{1, 2, 1, 2, 2, 2\}$, this sequence cannot be accommodated in the said hypercycle eventhough the total number of requested nodes exactly matches the total number of nodes in the graph. ∎

One should note that static optimality by no means guarantees a good dynamic performance. The nonstatically optimal first-fit strategy is expected to perform much better than the statically optimal first-fit stategy simply because it partitions the hypercycle in smaller portions, and is less redudant for the case where requests do not match the size of frgaments. This is indeed the case as it will be demonstrated through simulations in Section V; the nonstatically optimal strategy achieves superior results as compared to the first-fit allocation in terms of delay. Note, also, that the nonstatically optimal first-fit startegy reduces exactly to the first-fit startegy when the sizes of the requests match those of fragments.

IV. Permuting the Hypercycle Basis

Further improvement to the performance of the allocation can be had if there was a way to include in the search more fragments and segments or to choose the size of the fragments so as to best fit the size of the requests.

The use of multiple gray codes was proposed in [6] to improve the efficiency of the first-fit allocation in the case of hypercubes. The use of more gray codes permits the recognition of more subcubes and thus increases the efficiency of the allocation.

In the general case of hypercycles, the use of gray codes is not possible. Instead, we have chosen to permute the basis used to name the hypercycle in use, and perform the first-fit (or the nonstatically optimal first-fit) on the list of nodes that correspond to the chosen permutation(s). Such permutations result in isomorphic hypercycles [11], but with a different collection of fragments (segments) increasing thus the possibility that unallocated nodes would coalesce to form fragments (segments) of sufficient size to be allocated to the current request.

Of all the possible subgraphs, the first-fit strategy utilizes only a small portion. Consider for example the hypercycle with M factored as $M = m_n \times m_{n-1} \times \cdots \times m_1$. The k-dimensional fragments are limited to subgraphs described by addresses formed by placing k '*'s in the rightmost address digits. As an example, a subgraph of the form $*b_{n-1}b_{n-2}\cdots b_1$ is not recognized. The same subgraph would be recongizable (as a fragment), if the factors of M were permuted as $M = m_{n-1} \times m_{n-2} \times \cdots \times m_1 \times m_n$.

In general, k '*'s can occur in any of the n positions in a subgraph's address. Suppose that the ith placement of these k '*'s occurs in the positions described by the set p_i^k, that is, p_i^k is the set of positions where a '*' has replaced an address digit. Then the total number of possible k-dimensional fragments is given by

$$\sum_{i=1}^{\binom{n}{k}} \left(\prod_{\substack{j=1 \\ j \notin p_i^k}}^{n} m_j \right) \tag{4.1}$$

The first-fit strategy can only recognize placements of the form $p_i^k = \{k, k-1, \cdots, 1\}$ out of the $\binom{n}{k}$ possible placements of k '*'s on an n-digit address. For example, the hypercycle $\mathcal{G}_{5323}^{1111}$, has a total of 61 2-dimensional subgraphs while the first-fit allocation strategies discussed previously, can only utilize 15 2-dimensional fragments of the form b_4b_3**.

By permuting the factors of M (called the *basis* of the hypercycle) more fragments are recognizeable, and thus allocated. The search space of both the first-fit and the nonstatically optimal strategies is expanded by including lists of nodes that correspond to the chosen permutations of the basis. The search proceeds sequentially over the chosen lists of node names according to the algorithms presented in Sections III-A and -B, until a region of unallocated nodes is found that corresponds to a fragment or a segment accomodating the request. If such a region cannot be found, the search continues with the subsequent list until an unallocated region is found or the available lists are exhausted. All the lists are mapped to the same sequence of allocation bits.

In using several lists, holes which appear separate in one list, may combine to produce a larger hole in another list, and thus accommodate a larger request. One such example of the use of the permuted basis is given in Fig. 4. As it can be seen, a request for three nodes cannot be allocated since the unallocated nodes 00 10 20 do not constitute a hole under the original (unpermuted) basis (Fig. 4(b)). On the other hand, if the search is extended to include the list associated with the clock-wise rotation, the above three nodes are combined into a hole (Fig. 4(c)) which can now be allocated.

If P permutations are used then the compexity of the allocation strategy becomes $O(PM)$, that is, there exists a tradeoff between performance and running time. As (4.1) suggests, an excessive number of permutations is needed for complete fragment recognition as was also observed for the case of hypercubes in [6]. Nevertheless, our simulation studies showed that a relatively small number of permutations are enough to improve the performance considerably.

A. Multiple List First-Fit for Uniform Hypercycles

Notice that in general the factors of the hypercycle are different and, thus, permuting them results in different fragment sizes, so that the selection of the best set of permutations becomes intractable. The case is different, though, when all the factors of M are equal to each other. Such hypercycles are called *uniform*; the binary hypercubes and the k-ary n-cubes are two important topologies in this class. A

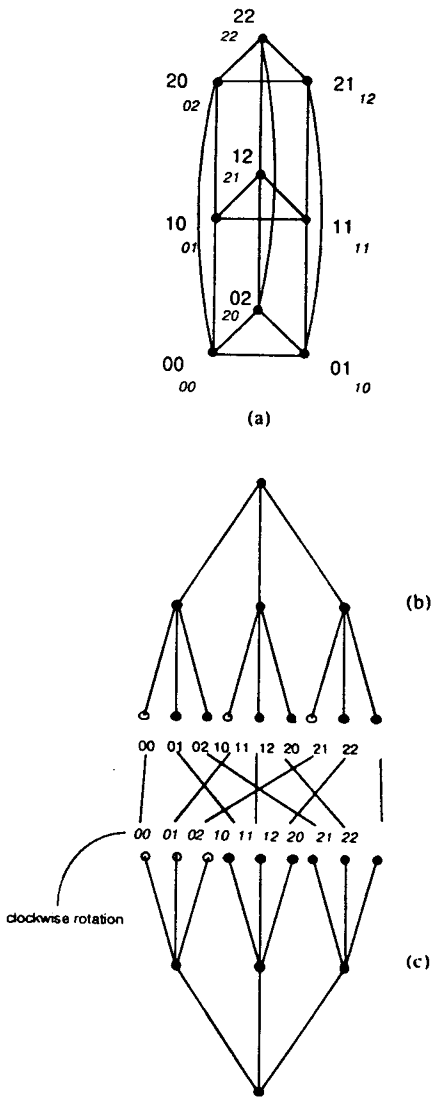

Fig. 4. Allocation utilizing two lists corresponding to naming permutations of the same hypercycle.

uniform hypercycle has $M = m^n = m \times m \times \cdots \times m$ nodes. In this section we derive the set of two permutations that result in optimal performance when used with the multiple-permutation strategies.

Moreover, the incorporation of the additional list(s), does not alter the static optimality of the strategy since any sequence of requests with a total size being less than or equal to the size of the hypercycle can be accommodated according to the first-fit strategy as discussed in Section III-A, by using the original allocation list.

Obviously, one needs to choose permutations so that the total numer of recognized fragments is maximized. This is done when permutations that have no fragment in common are used. Let the unpermuted basis be called *original permutation*. We have the following lemma.

Lemma 5: A permutation $\pi = \binom{n \; n-1 \; \cdots \; 1}{x_n \; x_{n-1} \; \cdots \; x_1}$ recognizes completely different fragments from the original permutation $\binom{n \; n-1 \; \cdots \; 1}{n \; n-1 \; \cdots \; 1}$ if and only if $\{x_1, x_2, \cdots, x_k\} \neq \{1, 2, \cdots k\}$ for all $k = 1, 2, \cdots, n-1$.

Proof: In the case that $\{x_1, x_2, \cdots, x_k\} = \{1, 2, \cdots k\}$ for some k, then the fragment $00..0*^k$ belongs trivially to both permutations. Suppose now that for all k, $\{x_1, x_2, \cdots, x_k\} \neq \{1, 2, \cdots k\}$. If there existed a fragment common to both permutations, then under the original permutation it would have an address $A = b_n b_{n-1} \cdots b_{k+1} *^k$. Under the permutation π, the k '*'s would also be in the rightmost positions, i.e. the corresponding address would be $B = b_{x_n} b_{x_{n-1}} \cdots b_{x_{k+1}} *^k$, and no '*' appears in positions $k+1, k+2, \cdots, n$ of B. Thus the k least signifficant digits of A are mapped to the k least signifficant digits of B, contradicting the assumption $\{x_1, x_2, \cdots, x_k\} \neq \{1, 2, \cdots k\}$.

Selecting permutations with no common fragments is not enough, though. Consider the case where a request cannot be allocated by the original permutation, although there exists space in the allocation list. This can only be the case when the existing holes are not buddies (see Definition 5) and consequently cannot be combined to form the required fragment. One needs to choose the permutation(s) so that it maximizes the number of holes which cannot be combined to form a larger fragment during the first-fit search, but can be combined to a larger fragment as they are mapped through the subsequent permutation(s). In what follows, we shall prove that in uniform Hypercycles, clock-wise rotations can map m fragments of equal size which are not buddies to a single fragment of size equal to m times the size of the individual fragments; this is true for fragments having any size between m^0 and m^{n-1}.

Definition 10: Given a set of n integers $1, 2, \cdots n$, we call a permutation π a *generalized shift* if it maps the last λ numbers onto the last $\lambda + 1$ positions for every $\lambda = 1, 2, \cdots n-1$.

The following are examples of generalized shifts

$$\pi_1 = \begin{pmatrix} n & n-1 & n-2 & \cdots & 1 \\ n-1 & n-2 & \cdots & 1 & n \end{pmatrix}$$

$$\pi_2 = \begin{pmatrix} n & n-1 & n-2 & n-3 & \cdots & 2 & 1 \\ n-1 & n & n-3 & n-2 & \cdots & 1 & 2. \end{pmatrix}$$

Lemma 6: In uniform n-dimensional hypercycles, only generalized shifts map fragments of size $m^{\lambda-1}$ into a fragment of size $m^{\lambda}, \lambda = 1, 2, \cdots n$.

Proof: Assume a generalized shift π and a fragment $\mathcal{F}_n = f_n f_{n-1} \cdots f_{\lambda} *^{\lambda-1}$.

Because π is a generalized shift, then there will be a position $\sigma, n \geq \sigma \geq \lambda$ such that $\lambda \geq \pi(\sigma) \geq 1$ and

$$\lambda \geq \pi(j) \geq 1, \forall j = \lambda - 1, \lambda - 2, \cdots, 1.$$

Then the fragments

$$\mathcal{F}_i = f_n f_{n-1} \cdots f_{\sigma+1} [f_\sigma]_i f_{\sigma-1} \cdots f_{\lambda} *^{\lambda-1}$$

and $\mathcal{F}_0$ map into fragment $\mathcal{X} \doteq x_n x_{n-1} \cdots x_{\lambda} *^{\lambda}$ where $x_j = f_{\pi^{-1}(j)}$.

Conversely, assume that any a fragments of the form $\mathcal{F}_i = f_n^i f_{n-1}^i \cdots f_{\lambda}^i *^{\lambda-1}, i = 0, 1, 2, \cdots, a-1$, map into the fragment $\mathcal{X} = x_n x_{n-1} \cdots x_{\lambda+1} *^{\lambda}$. Then obviously the $\lambda - 1$ last digits map into the λ last positions. Thus π is a generalized shift.

■

Theorem 5: In uniform n-dimensional hypercycles, the only generalized shift that maps a collection of m fragments of size $m^{\lambda-1}$, which are not buddies in the original permutation, into a fragment of size $m^{\lambda}, \lambda = 1, 2, \cdots n$ is the clock-wise rotation $\pi = \binom{n \;\; n-1 \;\; n-2 \; \cdots \; 1}{n-1 \;\; n-2 \; \cdots \; 1 \;\; n}$.

Proof: By construction.

Because of the previous lemma, only generalized shifts map m fragments of size $m^{\lambda-1}$ into a fragment of size $m^{\lambda}, \lambda = 1, 2, \cdots n$.

For $\lambda = 1$, it means that the permutation must map 1 either to itself or to 2. (i.e. $\pi(1) = 1$ or $\pi(1) = 2$). If π (1) were to be 1, then given the fragment $\mathcal{F} = f_n f_{n-1} \cdots f_1$ and according to the construction in the previous lemma, fragments $\mathcal{F}_i = f_n f_{n-1} \cdots [f_1]_i$ would map to fragment $\mathcal{X} = f_n f_{n-1} \cdots f_2 *$. But fragments $\mathcal{F}$ and $\mathcal{F}_i$ are buddies. Thus π must be such that $\pi(1) = 2$.

Suppose that

$$\pi(1) = 2, \pi(2) = 3, \cdots, \pi(\lambda) = \lambda + 1. \qquad (4.2)$$

We will show that $\pi(\lambda + 1) = \lambda + 2; \lambda \leq n - 2$. Because of Lemma 6, and assumption (4.2) above, $\pi(\lambda + 1) = \lambda + 2$ or $\pi(\lambda + 1) = 1$. If $\pi(\lambda + 1) = 1$, then fragments $\mathcal{F} = f_n f_{n-1} \cdots f_{\lambda+1} *^{\lambda}$ and $\mathcal{F}_i = f_n f_{n-1} \cdots f_{\lambda+1}[f_{\lambda+2}]_i *^{\lambda}$ would map into $\mathcal{X} = f_n f_{n-1} \cdots f_{\lambda+2} *^{\lambda+1}$. But $\mathcal{F}$ and $\mathcal{F}_i$ are buddies. Thus, $\pi(\lambda+1) = \lambda+2; \lambda \leq n-2$, and since π is a permutation, $\pi(n) = 1$. ∎

The clock-wise rotation was proven to be the only permutation that is able to combine holes that are not buddies so as to form fragments of a higher dimension. For any other permutation, the holes themselves are broken, and their constituents are mapped to different subgraphs, making thus their combining into a larger fragment impossible. The optimality of the clock-wise rotation is finally ensured by the fact that it also belongs in the class of permutations that recognize different fragments than the ones the unpermuted basis does, as per Lemma 5. The enhanced performance of the clockwise rotation was confirmed through our simulation results, to be presented in Section V.

B. Node Allocation Through an Optimal Permutation for Nonuniform Hypercycles

In the case that the hypercycle in question is not uniform, then a permutation of the factors of its basis, produces a topologically identical network, but its nodes combine to produce a different collection of fragment sizes as we noted earlier. Selecting the best permutations becomes a difficult task, heavily dependent on the load characteristics. It is advantageous, then, to choose the permutation which produces fragments whose sizes match the closest to the sizes requested in the request sequence. The following heuristic procedure can be used in order to obtain the "best" permutation.

Let π_j denote the jth permutation, out of the $n!$ possible permutations of the basis for the hypercycle $\mathcal{G}^{\rho_n \rho_{n-1} \cdots \rho_1}_{m_n m_{n-1} \cdots m_1}$. Under this permutation, a k-dimensional fragment, has size $F^j_k = m_{\pi_j^{-1}(k)} \times m_{\pi_j^{-1}(k-1)} \times \cdots \times m_{\pi_j^{-1}(1)}; k = 1, 2, \cdots n - 1$. Using the first-fit strategy, a request asking for i nodes is then allocated a fragment of size $F^j[i]$, where $F^j[i]$ will be equal to F^j_k, for k such that $F^j_{k-1} < i \leq F^j_k \equiv F^j[i]$.

If we are given the load characteristics in the form p_i = probability of a request asking for i nodes, then the quantity $A^j = \Sigma_{i=1}^{N} p_i \cdot F^j[i]$ gives the average number of allocated nodes per request for permutation π_j. The permutation that *minimizes* the average number A^j of allocated nodes per request is chosen. A similar expression can be derived for the nonstatically optimal allocation strategy.

The minimization, of course, carries a considerable overhead as n increases. Assuming, though, that the load characteristics change slowly, the proposed scheme becomes viable in that the minimization is carried out infrequently. The load characteristics may be known or established dynamically as requests are delivered.

V. Simulation Studies

We have constructed a simulator capable of simulating the behavior of the first-fit, the nonstatically optimal first-fit allocation strategies and also able to search on sequences of nodes derived by permuting the basis used to describe the hypercycle under simulation.

We have conducted three classes of experiments.

The first class of experiments assumes that the requests have arbitrary sizes (up to and including the maximum number of nodes in the hypercycle), and then the request is rounded up to the closest fragment or segment that contains it. The size of the requests is uniformly distributed.

Job run times are uniformly distributed between 3 and 7 clock ticks. Job arrival is Poison distributed. Our simulator uses a FIFO queuing discipline in that jobs are queued as they arrive, while jobs from the head of the queue are dispatched as soon as nodes became free in the graph. The Average Delay, being the time spent by the tasks waiting in the queue before they were allocated, averaged over all the tasks allocated during the simulation was measured.

The purpose of these experiments was to establish the relative performance of the first-fit and the nonstatically optimal first-fit, as well as the effect of using permutations to increase the number of fragments/segments which are recognizable by the allocation strategy.

The second class of experiments is governed by the same assumptions but the load is no longer uniformly distributed. The simulations were run with an arbitrary ordering of the factors defining the hypercycle, and then with the permutation which optimally matched the traffic characteristics according to the discussion in Section IV-B.

The third class of experiments assumes that the requests arrive one every clock tick. The experiments were run for uniform hypercycles which included the 3-, 4-, and 5-cube as well as $\mathcal{G}^{1111}_{3333}$. The requests were assumed to be of sizes which matched the fragment sizes of

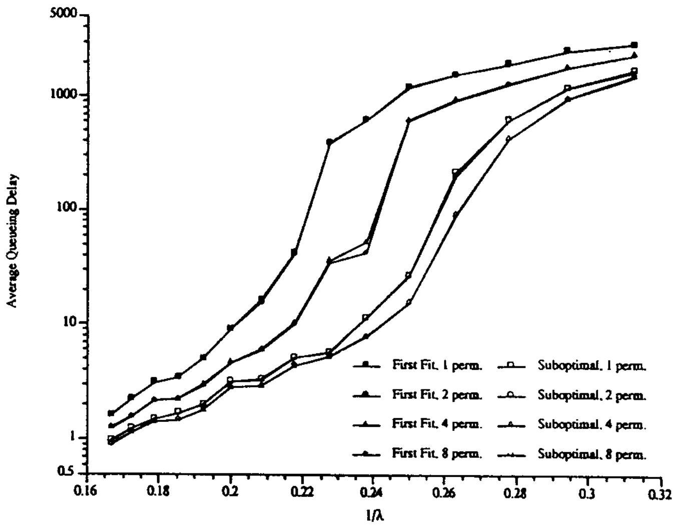

Fig. 5. Average queueing delay versus the inverse of the average interarrival time (λ) for the hypercycle $\mathcal{G}^{1111}_{3322}$.

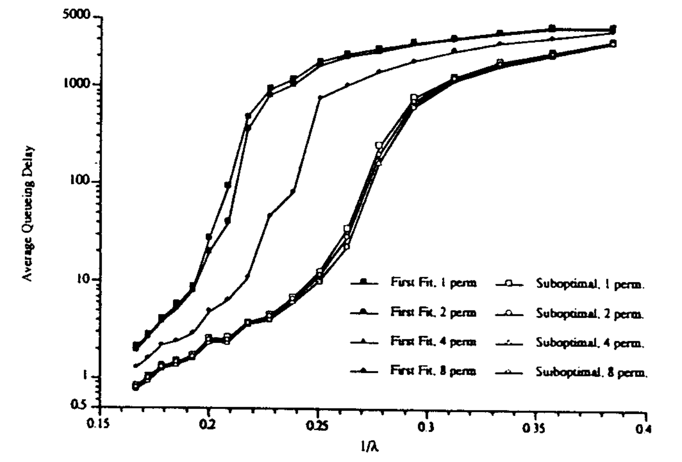

Fig. 6. Average queueing delay versus the inverse of the average interarrival time (λ) for the hypercycle $\mathcal{G}^{2211}_{5432}$.

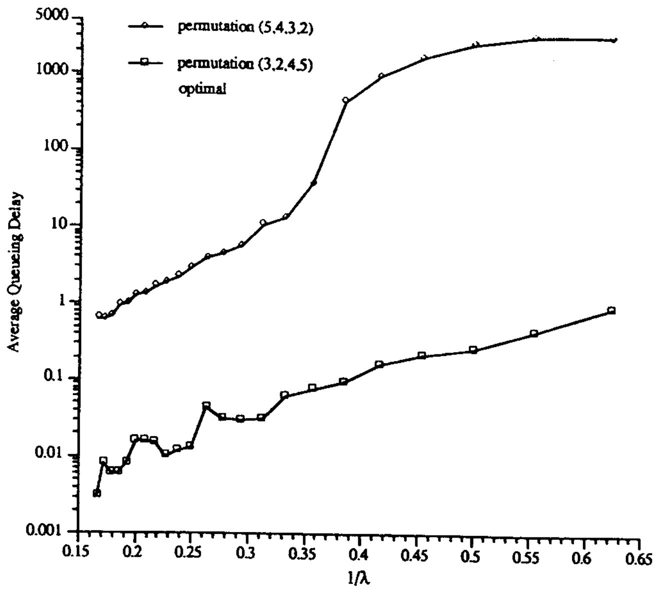

Fig. 7. Average queueing delay versus the inverse of the average interarrival time (λ) for the hypercycle $\mathcal{G}^{2211}_{5432}$. The requests are poison distributed while their sizes are following a gaussian distribution with mean of 20 and variance of 7.5. Permutation (5, 4, 3, 2) is an arbitrary permutation while permutation (3, 2, 4, 5) was chosen to best fit the characteristics of the traffic.

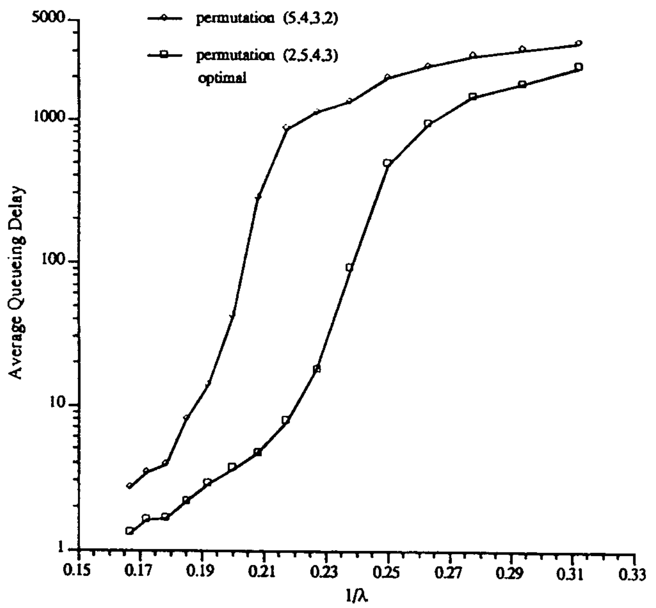

Fig. 8. Average queueing delay versus the inverse of the average interarrival time (λ) for the hypercycle $\mathcal{G}^{2211}_{5432}$. The requests are poison distributed while their sizes arefollowing a gaussian distribution with mean of 60.5. and variance of 7.5. Permutation (5,4,3,2) is an arbitrary permutation while permutation (2,5,4,3) was chosen to best fit the characteristics of the traffic.

the simulated hypercycle and were uniformly distributed between 1 and the maximum fragment for the hypercycle in question. The job duration was also uniformly distributed. The run was limited in duration to 100 clock ticks and repeated 10 000 times with the measured delay averaged over the number of times the experiment was run. The purpose of this class of experiments was to establish the relative performance of the use of gray codes as compared to the use of permuted bases in naming the nodes of a uniform hypercycle.

The results of these experiments are presented in Figs. 5–9 and Tables I–IV.

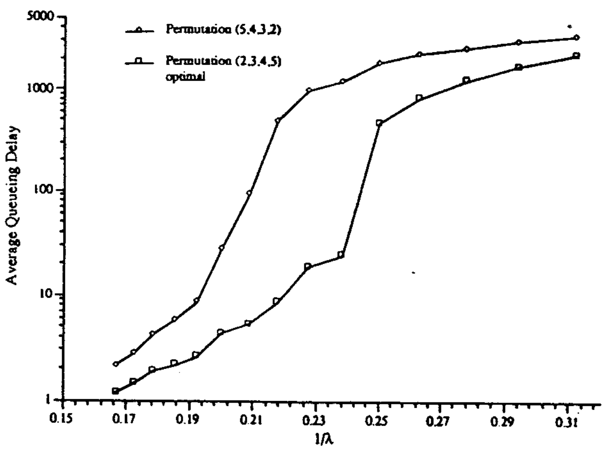

Fig. 9. Average queueing delay versus the inverse of the average interarrival time (λ) for the hypercycle $\mathcal{G}^{2211}_{5432}$. The requests are poison distributed while their sizes are uniformly distributed in the interval [1, 120]. Permutation (5, 4, 3, 2) is an arbitrary permutation while permutation (2, 3, 4, 5) was chosen to best fit the characteristics of the traffic.

TABLE I
DELAY FOR SEQUENCES OF FRAGMENTS IN THE 3-CUBE
(TASK DURATION UNIFORMLY DISTRIBUTED IN [2, 6])

POLICY		DELAY
Gray Code		12.572
First-Fit		12.791
First-Fit with one additional permutation	321,123	12.750
	321,312	12.744
	321,132	12.710
	321,231	12.375
	321,213*	12.362

* (clock-wise rotation)

TABLE II
DELAY FOR SEQUENCES OF FRAGMENTS IN THE 4-CUBE
(TASK DURATION UNIFORMLY DISTRIBUTED IN [2, 6])

POLICY		Delay
Gray Code		6.798
First-Fit		7.048
First-Fit with one additional permutation	4321,1234	7.038
	4321,1243	7.036
	4321,1324	7.035
	4321,1423	7.029
	4321,1342	7.028
	4321,1432	7.022
	4321,2143	6.995
	4321,2134	6.995
	4321,2314	6.952
	4321,2413	6.883
	4321,3124	6.543
	4321,3142	6.534
	4321,3214*	6.520

* (clock-wise rotation)

From the figures corresponding to the first class of experiments (Figs. 5 and 6), one can surmise that the nonstatically optimal first-fit allocation has a better performance as compared to the first-fit. This is

TABLE III
DELAY FOR SEQUENCES OF FRAGMENTS IN THE 5-CUBE (TASK DURATION UNIFORMLY DISTRIBUTED IN [3, 7])

POLICY		DELAY
Gray Code		8.814
First-Fit		9.092
First-Fit with one additional permutation	54321,12345	9.091
	54321,13245	9.091
	54321,15432	9.088
	54321,21543	9.080
	54321,43215*	8.497

* (clock-wise rotation)

TABLE IV
DELAY FOR SEQUENCES OF FRAGMENTS IN THE HYPERCYCLE $\mathcal{G}_{3333}^{1111}$ (TASK DURATION UNIFORMLY DISTRIBUTED IN [5, 8])

POLICY		DELAY
First-Fit		6.229
First-Fit with one additional permutation	4321,2134	6.247
	4321,2143	6.246
	4321,2314	6.231
	4321,1342	6.229
	4321,1423	6.229
	4321,1243	6.229
	4321,1324	6.229
	4321,1234	6.229
	4321,1432	6.228
	4321,2413	6.223
	4321,3142	6.187
	4321,3124	6.187
	4321,3214*	6.186

* (clock-wise rotation)

due to the nonstatically optimal's ability to better utilize the available nodes by allocating segments. Also both the first-fit and nonstatically optimal first-fit benefit from the use of multiple permutations in that the search is extended to more fragments and/or segments in the graph.

A similar picture emerges from the second class of experiments. As it can be seen in Figs. 7–9, the permutation chosen to optimally match the request characteristics has a better performance than any randomly chosen ordering of the factors of the basis of the hypercycle; this is true for all the considered loads. For clarity we have only included the graphs corresponding to the original permutation and the "best" permutation as derived in Section IV-B.

Finally, as it can be seen in Tables I–IV the first-fit allocation benefits from the use of clock-wise rotations (denoted by an asterisk). As a matter of fact, the first-fit allocation with a clock-wise rotation outperformed the use of gray codes [6] in all the three hypercubes considered. The complexity of either strategy is similar and this was verified experimentally[3] by our simulations. In addition, the optimality of clock-wise rotations was maintained for uniform hypercycles (as it can be seen in Table IV).

[3]For example, for the case of the 3-cube and for simulations lasting for 100 ticks and repeated 100 times, the innermost loop of our simulator (the loop that checks whether a region is free) was executed 76026 times for the first-fit with an additional clock-wise rotation and 85881 times for the first-fit utilizing a Gray Code.

VI. CONCLUSIONS

In this work, we presented a number of allocation strategies for hypercycle-based multiprocessor systems. These strategies are of the first-fit type, and include a statically optimal first-fit and a syuboptimal first-fit wchich is more efficient for dynamic loads.

Additionally, we investigated the effects of searching through lists of nodes which have been permuted. We proved the existence of permutations which coalesce existing holes into larger regions so that larger requests can be accomodated. We have simulated the proposed strategies and confirmed our theoretical predictions. A more detailed treatment of the results presented in this paper can be found in [10].

The work reported here is part of a larger project to study and develope hypercycle-based multiprocessors. Some preliminary results on routing, performance and implementation are reported in [11]–[14], [21].

REFERENCES

[1] A. Al-Bassam, H. El-Rewini, B. Bose, and T. Lewis, "Efficient serial and parallel subcube recognition in hypercubes," in *Proc. Fifth Distrib. Memory Comp. Conf.*, Apr. 1990, pp. 64–71.

[2] A. Al-Dhelaan and B. Bose, "A new strategy for processors allocation in an N-cube multiprocessor," in *Proc. Int. Phoenix Conf. Comput. Commun.*, Mar. 1989, pp. 114–118.

[3] L. N. Bhuyan and D. P. Agrawal, "Design and performance of generalized interconnection networks," *IEEE Trans. Comput.*, vol. C-32, pp. 1081–1090, Dec. 1983.

[4] ——, "Generalized hypercube and hyperbus structures for a computer network," *IEEE Trans. Comput.*, vol. C-33, pp. 323–333, Apr. 1984.

[5] F. Boesch and R. Tindell, "Circulants and their connectivities," *J. Graph Theory*, vol. 8, pp. 487–499, 1984.

[6] M. S. Chen and K. G. Shin, "Processor allocation in an N-cube multiprocessor using gray codes," *IEEE Trans. Comput.*, vol. C-37, pp. 1396–1407, Dec. 1987.

[7] E. Chow, H. Madan, J. Peterson, D. Grunwald, and D. Reed, "Hyperswitch network for the hypercube computer," in *Proc. 15th Annu. Int. Symp. Comput. Architect.*, May 1988, pp. 90–99.

[8] P.-J. Chuang and N.-J. Tzeng, "Dynamic processor allocation in hypercube computers," in *Proc. 17th Annu. Int. Symp. Comput. Architect.*, May 1990, pp. 40–49.

[9] W. Dally, "Performance analysis of k-ary n-cube interconnection networks," *IEEE Trans. Comput.*, vol. 39, pp. 775–784, June 1990.

[10] V. V. Dimakopoulos, "Processor allocation and message broadcasting in hypercycle interconnection networks," M.A.Sc. thesis, Dep. Elec. Comput. Eng., Univ. of Victoria, 1992.

[11] N. J. Dimopoulos, R. D. Rasmussen, G. S. Bolotin, B. F. Lewis, and R. M. Manning, "Hypercycles, interconnection networks with simple routing strategies," in *Proc. Canadian Conf. Elec. Comput. Eng.*, Nov. 1988, pp. 577–80.

[12] N. Dimopoulos, D. Radvan, and K. F. Li, "Performance evaluation of the backtrack to the origin and retry routing for hypercycle-based interconnection networks," in *Proc. Tenth Int. Conf. Distrib. Comput. Syst.*, ACM, June 1990, pp. 278–284.

[13] N. J. Dimopoulos, R. Shivakumar, and D. Radvan, "Routing and processor allocation on a hypercycle-based multiprocessor," in *Proc. 1991 Int. Conf. Supercomput.*, ACM, June 1991, pp. 106-114.

[14] N. J. Dimopoulos, M. Chowdhury, R. Sivakumar, and V. Dimakopoulos, "Routing in hypercycles—Deadlock free and backtracking strategies," in *Proc., PARLE '92 Parallel Architect. and Languages Europe '92*, June 1992, pp. 973–997.

[15] S. Dutt and J. P. Hayes, "Subcube allocation in hypercube computers," *IEEE Trans. Comput.*, vol. 40, pp. 341–352, Mar. 1991.

[16] J. Kim, C. R. Das, and W. Lin, "A top-down processor allocation scheme for hypercube computers," *IEEE Trans. Parallel and Distrib. Syst.*, vol. 2, pp. 20–30, Jan 1991.

[17] J. C. Peterson, J. O. Tuazon, D. Lieberman, and M. Pniel, "The MARK III hypercube-ensemble concurrent computer," in *Proc. 1985 Int. Conf. Parallel Processing*, Aug. 1985, pp. 71–73.

[18] R. D. Rasmussen, G. S. Bolotin, N. J. Dimopoulos, B. F. Lewis, and R. M. Manning, "Advanced general purpose multicomputer for space applications," in *Proc. 1987 Int. Conf. Parallel Processing*, Aug. 1987, pp. 54–57.

[19] ——, "MAX: Advanced general purpose real-time multicomputer for space applications," in *Proc. IEEE Real Time Syst. Symp.*, Dec. 1987, pp. 70–78.

[20] C. L. Seitz, "The cosmic cube," *CACM*, vol. 28, pp. 22–33, Jan. 1985.

[21] R. Sivakumar, N. J. Dimopoulos, V. Dimakopoulos, M. Chowdhury, "Implementation of the routing engine for hypercycle based interconnection networks," in *Proc. Canadian Conf. VLSI*, Aug. 1991, pp. 6.4.1–6.4.7.

[22] A. S. Tanenbaum, *Operating Systems: Design and Implementation.* Englewood Cliffs, NJ: Prentice-Hall, 1987.