Electrical Engineering & Computer Science Faculty
Publications

Electrical Engineering & Computer Science
Department

5-2003

Isomorphic Strategy for Processor Allocation in k-Ary n-Cube Systems

Moonsoo Kang
Information and Communications University, kkamo@icu.ac.kr

Chansu Yu
Cleveland State University, c.yu91@csuohio.edu

Hee Yong Youn
Sung Kyun Kwan University, youn@ece.skku.ac.kr

Ben Lee
Oregon State University, benl@eecs.oregonstate.edu

Myungchul Kim
Information and Communications University, mckim@icu.ac.kr

Publisher's Statement

Original Citation

Moonsoo, K., Chansu, Y., Hee, Y. Y., Ben, L., & Myungchul, K. (May 01, 2003). Isomorphic strategy for processor allocation in k-ary n-cube systems. IEEE Transactions on Computers, 52, 5, 645-657.

Repository Citation

Kang, Moonsoo; Yu, Chansu; Youn, Hee Yong; Lee, Ben; and Kim, Myungchul, "Isomorphic Strategy for Processor Allocation in k-Ary n-Cube Systems" (2003). *Electrical Engineering & Computer Science Faculty Publications*. 98.
https://engagedscholarship.csuohio.edu/enece_facpub/98

Isomorphic Strategy for Processor Allocation in k-Ary n-Cube Systems

Moonsoo Kang, Chansu Yu, *Member*, *IEEE*, Hee Yong Youn, *Senior Member*, *IEEE*, Ben Lee, *Member*, *IEEE Computer Society*, and Myungchul Kim

Abstract—Due to its topological generality and flexibility, the k-ary n-cube architecture has been actively researched for various applications. However, the processor allocation problem has not been adequately addressed for the k-ary n-cube architecture, even though it has been studied extensively for hypercubes and meshes. The earlier k-ary n-cube allocation schemes based on conventional *slice partitioning* suffer from internal fragmentation of processors. In contrast, algorithms based on *job-based partitioning* alleviate the fragmentation problem but require higher time complexity. This paper proposes a new allocation scheme based on *isomorphic partitioning*, where the processor space is partitioned into higher dimensional isomorphic subcubes. The proposed scheme minimizes the fragmentation problem and is general in the sense that any size request can be supported and the host architecture need not be isomorphic. Extensive simulation study reveals that the proposed scheme significantly outperforms earlier schemes in terms of mean response time for practical size k-ary and n-cube architectures. The simulation results also show that reduction of external fragmentation is more substantial than internal fragmentation with the proposed scheme.

Index Terms—k-ary n-cube, processor allocation, job scheduling, partitioning, performance evaluation.

✦

1 INTRODUCTION

EXECUTING an incoming task on a parallel computer system requires decomposing the task into subtasks and allocating a set of processors with appropriate connectivity to the subtasks. The *processor allocation problem* deals with finding a particular set of processors (or subcube) with the required topology and size. The goal is to maximize the system utilization by improving the recognizability of subcubes, thereby minimizing fragmentation of processors within the system. The processor allocation problem has been extensively studied for multicomputer systems with hypercube [1], [2], [3], [4], [5] and mesh [6], [7], [8], [9] interconnection topologies. However, the problem has not been adequately addressed for the general k-ary n-cube network, even though it has been extensively studied [10], [11], [12], [13], [14], [15] for parallel computing due to its topological generality and flexibility.

The processor allocation problem is much more challenging for k-ary n-cube networks than hypercubes or meshes because reducing fragmentation within the system involves recognizing both the dimension of the network (as in hypercubes) and the number of processors in each dimension (as in meshes). However, existing processor allocation strategies for the k-ary n-cube system either recognize only the dimensionality of the subcubes or allow arbitrary partition sizes at the cost of complex search operations. For example, schemes based on *slice partitioning* [16], [17], [18], which are basically simple extensions of the processor allocation schemes for hypercubes, partition a higher dimensional cube into lower dimensional subcube "*slices*" with each lower dimension still containing k nodes, i.e., only k-ary m-cube subcubes (where $m \leq n$) are recognized. Therefore, allocation of processors to a job request is limited to one or more partitions of base-k and the remaining nodes are wasted resulting in internal fragmentation. On the other hand, processor allocation schemes based on *job-based partitioning* [16], [19], [20] alleviate the internal fragmentation problem by relaxing the base-k restriction and allowing arbitrary partition sizes. However, these schemes require time-consuming exhaustive search operations to combine external fragmentation of processors caused by dynamic allocation and deallocation of jobs.

This paper proposes *isomorphic allocation strategy* for k-ary n-cube systems that significantly improves the subcube recognition capability, fragmentation, and complexity compared to existing methods. The proposed scheme is based on *isomorphic partitioning*, which recursively partitions a k-ary n-cube into 2^n number of $\frac{k}{2^i}$-ary n-cubes, where i represents the ith *partition step*. The resulting partitioned subcubes are said to be "isomorphic" in the sense that they are also n-cubes and, for this reason, they retain many attractive properties of k-ary n-cube networks, including symmetry, low node degree ($2n$), and low diameter (kn) [13]. Moreover, the proposed strategy is extended to recognize *semi-isomorphic* subcubes by combining a number of smaller isomorphic subcubes to handle

- M. Kang and M. Kim are with the School of Engineering, Information and Communications University 58-4 Hwa-am, Yu-sung, Taejon, 305-348 Korea. E-mail: {kkamo, mckim}@icu.ac.kr.
- C. Yu is with the Department of Electrical and Computer Engineering, Cleveland State University, Stilwell Hall 340, Cleveland, OH 44115. E-mail: c.yu91@csuohio.edu.
- H.Y. Youn is with the School of Information and Communication Engineering, SungKyunKwan University, 300 Janganggu, Chunchun-dong, Suwon, 440-746 Korea. E-mail: youn@ece.skku.ac.kr.
- B. Lee is with the Department of Electrical and Computer Engineering, Oregon State University, Owen Hall 302, Corvallis, OR 97331. E-mail: benl@ece.orst.edu.
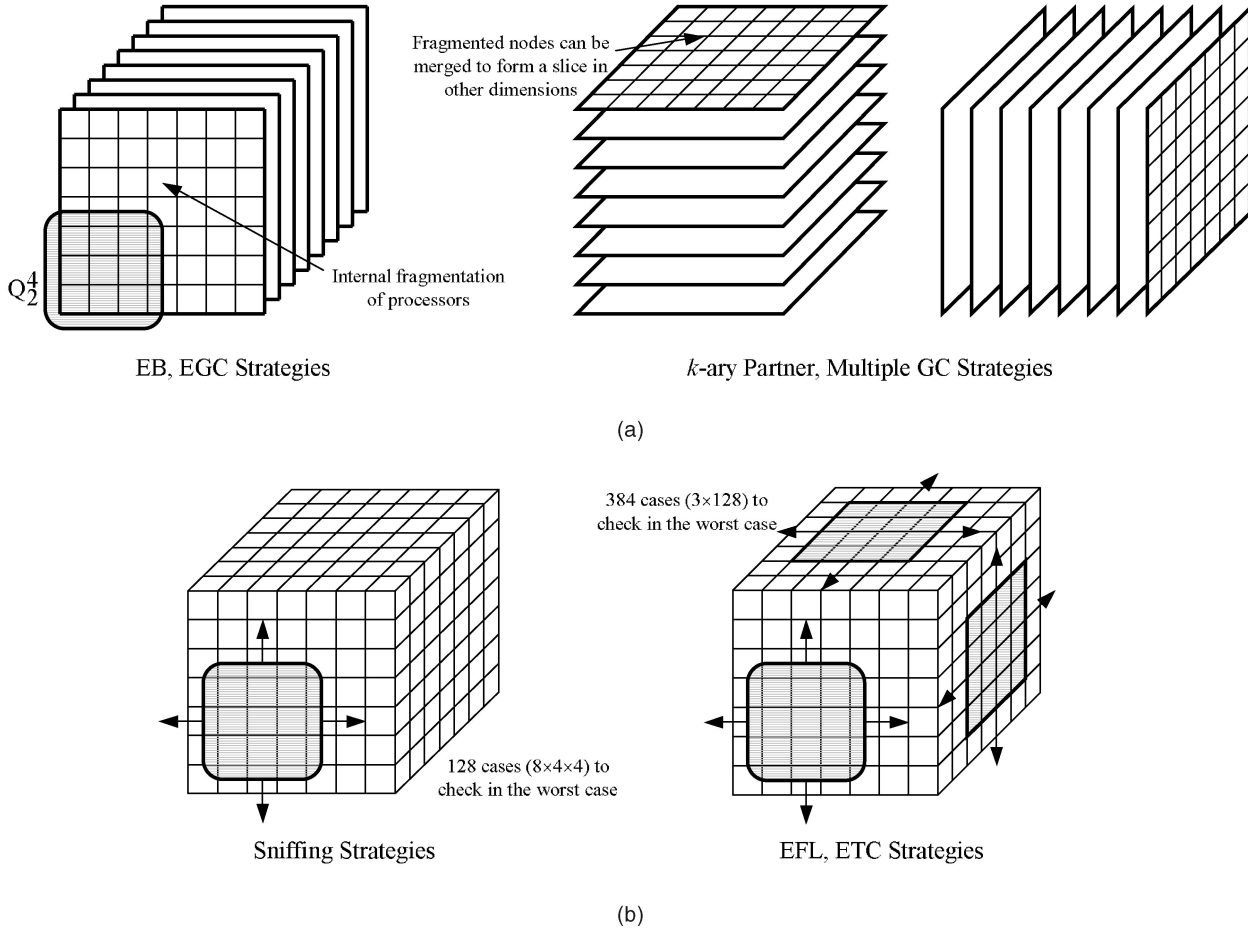
Fragmented nodes can be merged to form a slice in other dimensions

Internal fragmentation of processors

Q_2^4

EB, EGC Strategies

k-ary Partner, Multiple GC Strategies

(a)

384 cases (3×128) to check in the worst case

128 cases (8×4×4) to check in the worst case

Sniffing Strategies

EFL, ETC Strategies

(b)

Fig. 1. Slice and job-based allocation strategies on an 8-ary 3-cube system. (a) With slice partitioning. (b) With job-based partitioning.

incoming job requests of arbitrary topologies. Finally, the proposed strategy is also applied to systems that are not k-ary n-cube. This is important in practice since, for example, Cray T3D/T3E uses a three-dimensional torus network but the sizes of the three dimensions are different.

The main contributions of this paper are two-fold: First, the Isomorphic allocation strategy is specifically targeted for the k-ary n-cube topology. That is, unlike existing strategies that are basically hypercube allocation algorithms extended to larger sizes or mesh allocation algorithms extended to higher dimensions, the proposed method exploits the unique topological characteristics of k-ary n-cube. It is also general in the sense that any size requests can be supported and the host system need not be k-ary n-cube. Second, a new way of representing the k-ary n-cube network within a graph theory framework is formalized. We believe the new formalism can be used in other research areas, such as routing, for k-ary n-cube systems.

We evaluate and compare the performance of the proposed isomorphic allocation strategy with four existing allocation schemes using CSIM simulation package [21]. In terms of *mean response time*, the proposed strategy significantly outperforms the allocation schemes based on slice partitioning. Comparison with the schemes employing the job-based partitioning approach shows that the Isomorphic allocation strategy improves the response time by $7\% \sim 49\%$ depending on system size and workload distribution.

More importantly, it is shown to be scalable, i.e., it exhibits consistent performance irrespective of the system size.

The rest of the paper is organized as follows: Section 2 presents the earlier allocation algorithms for k-ary n-cube systems. In Section 3, a formal framework for describing a k-ary n-cube and its partitioning mechanisms is introduced. Section 4 presents the isomorphic allocation strategy and its extensions. Section 5 evaluates the performance of the proposed scheme using simulation and compares it with earlier schemes. Finally, conclusions and future work are discussed in Section 6.

2 BACKGROUND AND RELATED WORK

This section overviews allocation algorithms proposed for k-ary n-cube systems. As discussed previously, there are two types of allocation algorithms. *Extended Buddy (EB)*, *Extended Gray Code (EGC)* [16], [17], k-ary *Partner* [18], and *Multiple Gray Code (Multiple GC)* [17] algorithms are based on slice partitioning. Job-based partitioning is employed in *Sniffing* [16], *Extended Free List (EFL)* [19], and *Extended Tree Collapsing (ETC)* [20].

EB and EGC algorithms are extended versions of the hypercube algorithms, *Buddy* and *Gray Code* [1], respectively. Fig. 1a illustrates an example of allocating a 4-ary 2-cube job request in an 8-ary 3-cube system. With the slice partitioning, one 8-ary 2-cube partition of 64 processors is

allocated since only 8-ary subcubes are recognized and, thus, the remaining 48 processors are wasted. This partition size limitation is an inherent problem with the underlying slice partitioning. More importantly, links as well as processors are underutilized. Unless the whole $8 \times 8 \times 8$ cube is used for one large job, communication cannot occur between the partitioned 8×8 slices. All the links along one dimension would be wasted, which accounts for one third of the total number of links. Another major drawback with the slice partitioning is it does not exploit the topological advantages of higher order architecture. For example, the nodes in an 8×1 slice (eight nodes) have longer internode distance compared to the nodes in a 2-ary 3-cube (eight nodes). The time complexity of the allocation procedure is $O(k^n)$ for a k-ary n-cube system since the availability of the processors in all possible directions is checked. Deallocation procedure takes $O(k^m)$ for releasing a k-ary m-cube job.

k-ary Partner and Multiple GC algorithms enhance the subcube recognition ability over EB and EGC. They utilize the fragmented nodes to form a slice along the other dimensions, as shown on the right side of Fig. 1a. k-ary Partner and Multiple GC algorithms require the same time complexity as EB and EGC algorithms, i.e., $O(k^n)$ for allocation and $O(k^m)$ for deallocation of a k-ary m-cube job.

In contrast, job-based partitioning addresses the internal fragmentation problem by allowing arbitrary partition sizes rather than restricting them to base-k. Fig. 1b shows an example of allocating a 4-ary 2-cube job request in an 8-ary 3-cube system using job-based partitioning. The allocation algorithms based on this approach search the processor space to find an available subcube for the job by sliding a 4×4 window frame. For example, the Sniffing algorithm checks 16 $((8-4)^2)$ window positions per 8-ary 2-cube plane, totaling 128 (16×8) positions for all eight planes, as shown on the left side of Fig. 1b. In general, for an l-ary m-cube job request in a k-ary n-cube system, the number of positions per k-ary m-cube plane is $(k-l)^m$. The total number of window positions to check amounts to $(k-l)^m k^{n-m}$ for all k^{n-m} planes and is bounded by k^n (when $k \gg l$). Thus, the allocation complexity of the Sniffing strategy is $O(k^n l^m)$ [16], where the last term l^m accounts for the availability check of the l^m processors. Deallocation of an l-ary m-cube job frees l^m processors and, thus, the deallocation complexity of the Sniffing strategy is $O(l^m)$ because the corresponding l^m bits must be reset.

EFL and ETC algorithms improve the subcube recognition ability of the Sniffing strategy by including the cases where a 4-ary 2-cube job is assigned along the other dimensions as shown on the right side of Fig. 1b. Here, the total number of window positions is n (i.e., three) times of the Sniffing strategy. In general, the allocation complexity of the ETC algorithm is

$$O \binom{n}{m} k^n l^m$$

for an l-ary m-cube job, where the combination term accounts for the selection of m dimensions out of n since the m-cube job can be allocated to any m-dimensional plane. This high time complexity is not surprising because all

possible positions are exhaustively checked. In addition to the large amount of allocation time, job response time may increase due to external fragmentation of processors. The deallocation complexity of the ETC strategy for an l-ary m-cube job is also $O(l^m)$ because the corresponding l^m bits must be reset as in the Sniffing strategy. Section 4.4 summarizes and compares the complexitites of the above-mentioned algorithms as well as the proposed Isomorphic algorithm

3 PRELIMINARIES

This section presents the formal description of the slice and isomorphic partitioning for k-ary n-cube networks.

A k-ary n-cube, which is denoted as Q_n^k, has k^n nodes, each of which can be identified by an n-tuple $(a_{n-1}, \cdots, a_1, a_0)$ of radix k, where a_i represents the node's position in the ith direction. Let Σ^k be the *alphabet* $\{0, 1, 2, \cdots, k-1\}$ and Σ_n^k be the set of all sequences of the elements in Σ^k of length n. Then, $a_i \in \Sigma^k$ and the set of the nodes of Q_n^k can be represented by Σ_n^k. Here, nodes $(a_{n-1}, \cdots, a_1, a_0)$ and $(a'_{n-1}, \cdots, a'_1, a'_0)$ are connected if and only if there exists i, $0 \leq i \leq n-1$, such that $a_i = a'_i \pm 1$ and $a_j = a'_j$ for $j \neq i$ if wrap-around links are not considered.

Given two graphs $A = (V_1, E_1)$ and $B = (V_2, E_2)$, the *cross product* $A \otimes B = (V, E)$ is defined by [13]

$$V = \{(a, b) \mid a \in V_1, b \in V_2\} \quad and$$
$$E = \{(a, b), (a', b') \mid (a = a' \quad and \quad (b, b') \in E_2)$$
$$or \quad (b = b' \quad and \quad (a, a') \in E_1)\}.$$

A cross product of an n-dimensional graph A and an m-dimensional graph B produces an $(n + m)$-dimensional graph. With $a = (a_{n-1}, \cdots, a_1, a_0) \in \Sigma_n$ and

$$b = (b_{m-1}, \cdots, b_1, b_0) \in \Sigma_m,$$

a node (a, b) in $A \otimes B$ can be represented by an $(n + m)$-tuple $(a_{n-1}, \cdots, a_1, a_0, b_{m-1}, \cdots, b_1, b_0) \in \Sigma_{(n+m)}$. In short, the cross product $A \otimes B$ stretches the graph A along the dimnensions of graph B.

In order to define a Q_n^k using the cross product, consider a graph L_k that has k nodes and $(k-1)$ edges, where the k nodes form a linear array and each node is connected to two nearby nodes without a wraparound edge. Thus, L_k is a one-dimensional (1D) array, $L_k \otimes L_k$ is a 2D mesh, and $L_k \otimes L_k \otimes L_k$ is a 3D mesh. Similarly, Q_n^k can be defined by a cross product of n L_ks [13].[1] That is

$$Q_n^k = \underbrace{L_k \otimes L_k \otimes \cdots \otimes L_k}_{n \ times}.$$

Instead of assuming k is a power of 2, a 2^k-ary n-cube is considered. It is represented as

$$Q_n^{2^k} = \underbrace{L_{2^k} \otimes L_{2^k} \otimes \cdots \otimes L_{2^k}}_{n \ times}.$$

---

1. Here, we do not include wraparound edges and the resulting Q_n^k is a mesh. In [13], L_k is a cycle which has wraparound edges, and the corresponding Q_n^k is a torus.
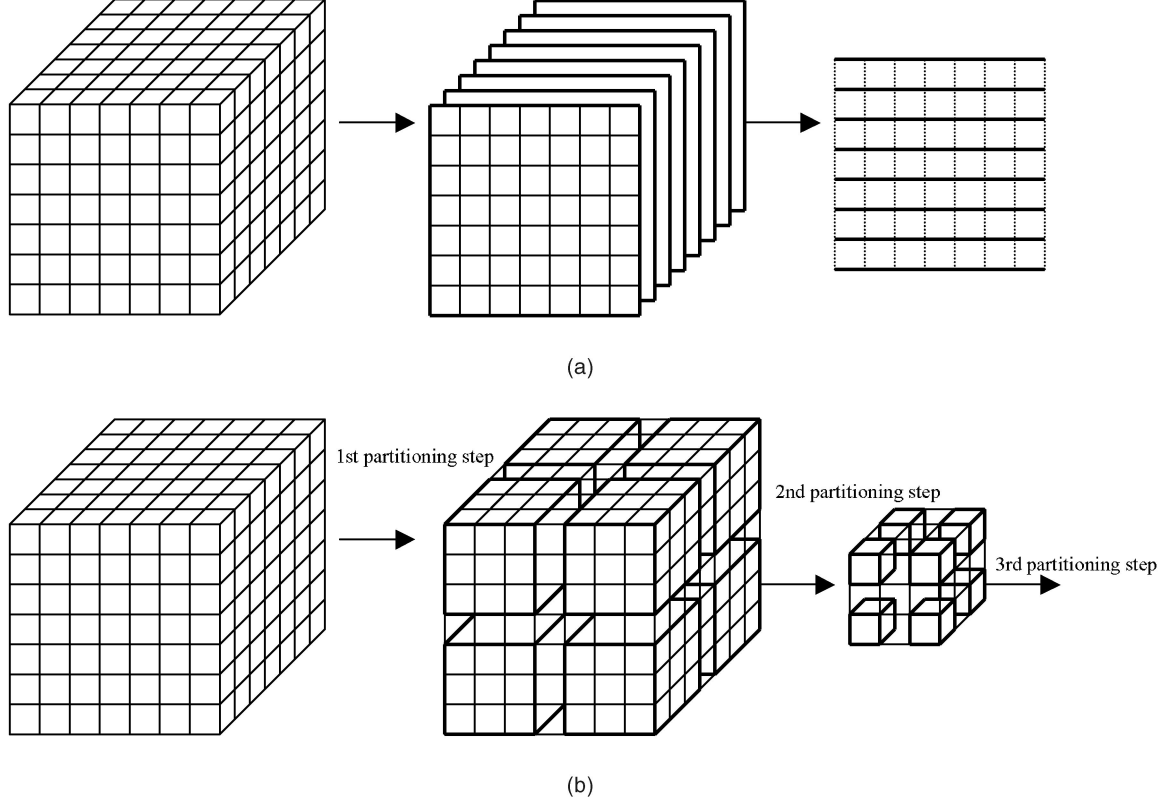
Fig. 2. Partitioning mechanisms of an 8-ary 3-cube system. (a) Slice partitioning of an 8-ary 3-cube ($8 \times 8 \times 8$). (b) Isomorphic partitioning of an 8-ary 3-cube ($8 \times 8 \times 8$).

The slice partitioning corresponds to the above-mentioned construction process of an n-dimensional $Q_n^{2^k}$ from one-dimensional L_{2^k} but in the reverse direction (this is depicted in Fig. 2a). In other words, a higher order cube is recursively partitioned into a number of lower dimensional subcubes. In the figure, $Q_3^8 = Q_2^8 \otimes L_8$, $Q_2^8 = Q_1^8 \otimes L_8$, and $Q_1^8 = Q_0^8 \otimes L_8$.

Now, an alternative way of defining a $Q_n^{2^k}$ is based on the *dot product*, where two graphs are multiplied to produce a larger graph but with the same order of dimension as that of its two subgraphs. Given two n-dimensional graphs, $A = (V_1, E_1)$ and $B = (V_2, E_2)$, the dot product $A \odot B = (V, E)$ is defined by

$$V = \{(a, b) \mid a \in V_1, b \in V_2\}$$
$$and \ \ E = \{(a, b), (a', b') \mid \exists i, 0 \ \ i \ \ n - 1,$$
$$such \ that \ a_i b_i = a_i' b_i' \pm 1 \ and \ a_j b_j = a_j' b_j' \ for \ j \neq i\}.$$

Notice that a node (a, b) in the dot product is also represented by an n-tuple $(a_{n-1}b_{n-1}, \cdots, a_1 b_1, a_0 b_0)$, where each component $a_i b_i$ is a concatenation of two subcomponents a_i and b_i. If we assume the two n-dimensional graphs A and B are 2^k-ary and 2^l-ary, then a_i and b_i are k-bit and l-bit binary numbers, respectively. Thus, $a_i b_i$ is a $(k + l)$-bit binary number and $A \odot B$ is a $(k + l)$-ary n-dimensional graph. The sets of the nodes in A and B are ${}^k_n$ and ${}^l_n$, respectively, and that of $A \odot B$ is ${}^{(k+l)}_n$. Informally, a dot product $A \odot B$ can be drawn by replacing each node of A with graph B.

In order to define a $Q_n^{2^k}$ using the dot product, consider an n-dimensional binary hypercube H_n. A node in H_n is represented by a binary n-tuple, $(a_{n-1}, \cdots, a_1, a_0)$, where

$a_i \in {}^2$. A node in $H_n \odot H_n$ can be denoted by a 4-ary n-tuple, $(a_{n-1}a_{n-1}', \cdots, a_1 a_1', a_0 a_0')$, where $a_i, a_i' \in {}^2$ and $a_i a_i' \in {}^4$. Thus, $H_n \odot H_n$ is a 4-ary hypercube and $H_n \odot H_n \odot H_n$ is an 8-ary hypercube. Similarly,

$$Q_n^{2^k} = \underbrace{H_n \odot H_n \odot \cdots \odot H_n}_{k \ times}.$$

Equivalently, $Q_n^{2^k}$ can also be represented by $Q_n^{2^{k-1}} \odot H_n$, recursively. In other words, $Q_n^{2^k}$ can be partitioned into 2^n number of $Q_n^{2^{k-1}}$'s because H_n has 2^n nodes. We call this *isomorphic partitioning* because each partition $Q_n^{2^{k-1}}$ or

$$\underbrace{2^{k-1} \times \cdots \times 2^{k-1}}_{n \ times}$$

keeps the same order of dimension (n) and, therefore, retains the topological advantages of a higher dimensional cube. Graphical representation of the Isomorphic partitioning is shown in Fig. 2b. In the figure, $Q_3^8 = Q_2^8 \odot H_3$, $Q_2^8 = Q_1^8 \odot H_3$, and $Q_1^8 = Q_0^8 \odot H_3$.

4 ISOMORPHIC ALLOCATION STRATEGY

Based on the above-mentioned discussion, we now introduce the *isomorphic allocation strategy* for $Q_n^{2^k}$. The basic allocation strategy in Section 4.1 produces *isomorphic* subcube partitions only and thus restricts the job requests to be *isomorphic* ($Q_n^{2^a}$). Section 4.2 relaxes the job size restriction to allocate *cubic job requests* in the form of $2^{a_{n-1}} \times \cdots \times 2^{a_1} \times 2^{a_0}$. (An isomorphic job is considered a
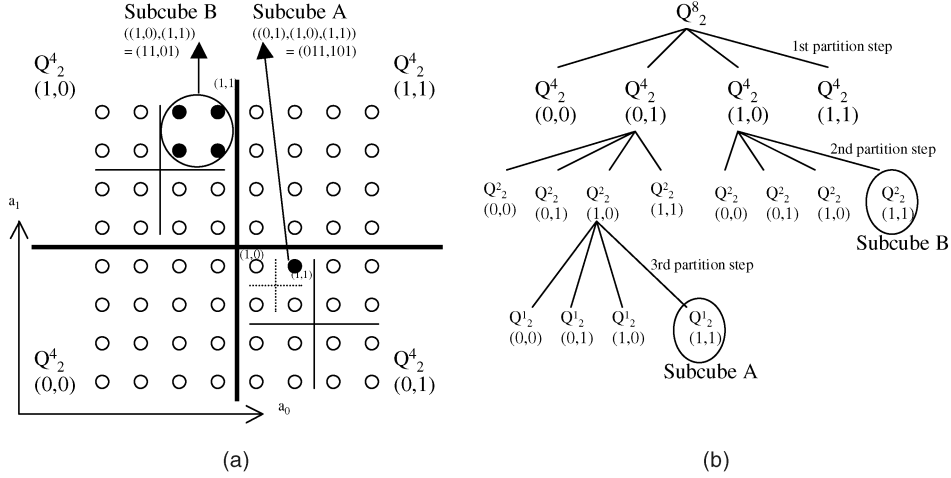
Fig. 3. Representation of the isomorphic partitioning of Q_2^8. (a) Subcubes in Q_2^8. (b) 4-ary tree representation of Q_2^8.

cubic job, where $a_i = a$ for all i.) It is further extended to allocate *noncubic jobs* in the form of $l_{n-1} \times \cdots \times l_1 \times l_0$, where l_i is not necessarily a power of two. Processor allocation in *cubic systems* in the form of $2^{k_{n-1}} \times \cdots \times 2^{k_1} \times 2^{k_0}$ is also considered in Section 4.3. This is important in practice since a system may not always be k-ary n-cube. For example, a Cray T3D/T3E uses a three-dimensional torus as the internal interconnection, but the sizes of the three dimensions are usually different mainly due to the packaging problem. Finally, Section 4.4 presents the algorithm complexity of the isomorphic allocation strategy and compares with that of previous allocation algorithms.

Throughout the paper, the basic as well as the extended algorithms are collectively referred to as isomorphic allocation strategy. When the types of job requests are all isomorphic, the basic algorithm in Section 4.1 is used. But, if some jobs require nonisomorphic cubic partitions, the extended algorithm in Section 4.2 is employed.

4.1 Isomorphic Allocation Strategy for Isomorphic Jobs

This subsection presents the basic isomorphic allocation strategy for isomorphic jobs. It assigns a subcube partition to a job requesting a 2^a-ary n-cube in a 2^k-ary n-cube system, where $a \leq k$. First, we consider how addresses are assigned to a subcube partition generated by the Isomorphic allocation strategy. Fig. 3a shows the subcubes in an 8-ary 2-cube system (8×8 mesh). They can also be described by a 2^n-ary tree (4-ary tree in this example) with k partition steps (three steps in this case), as in Fig. 3b.

Consider a subcube A consisting of one node whose address is $(3, 5)$ in Fig. 3. (Note that the digits are ordered from right to left, i.e., (a_1, a_0).) A binary representation of the node is $(011, 101)$. Since $Q_2^8 = H_2 \odot H_2 \odot H_2$, the node can alternatively be represented by $((0, 1), (1, 0), (1, 1))$, where $(0, 1)$ is the address of the node in the first subgraph H_2, $(1, 0)$ is the one in the second H_2, and $(1, 1)$ is the one in the third H_2. In other words, the subcube A can be addressed by selecting $(0, 1)$ Q_2^4 subcube after the first partition step, $(1, 0)$ Q_2^2 subcube after the second partition step, and, finally, $(1, 1)$ Q_2^1 subcube after the third partition step. Similarly, subcube B in Fig. 3 can be identified by $((1, 0), (1, 1)) = (11, 01) = (11*, 01*)$.

In general, a node in a 2^k-ary n-cube, $Q_n^{2^k}$, is denoted by an n-tuple $(a_{n-1}, \cdots, a_1, a_0)$, where $a_i \in \mathbb{Z}_{2^k}$. We can also denote the node in a full binary representation as

$$\left(a_{n-1}^{(1)} a_{n-1}^{(2)} \cdots a_{n-1}^{(k)}, \cdots, a_1^{(1)} a_1^{(2)} \cdots a_1^{(k)}, a_0^{(1)} a_0^{(2)} \cdots a_0^{(k)} \right),$$

where $a_i^{(j)} \in \mathbb{Z}_2$. The superscript j in each binary number denotes the partition step in which the binary number plays its role. As discussed in Section 3, each of k dot products contributes one binary digit in all dimensions by concatenating the subgraphs. We can, therefore, alternatively represent it as

$$\left(\underbrace{\left(a_{n-1}^{(1)}, \cdots, a_1^{(1)}, a_0^{(1)} \right)}_{1st\ n-tuple}, \underbrace{\left(a_{n-1}^{(2)}, \cdots, a_1^{(2)}, a_0^{(2)} \right)}_{2nd\ n-tuple}, \right.$$
$$\left. \cdots, \underbrace{\left(a_{n-1}^{(k)}, \cdots, a_1^{(k)}, a_0^{(k)} \right)}_{k-th\ n-tuple} \right).$$

Similarly, a subcube $Q_n^{2^a}$ can be represented by an n-tuple,

$$\left(a_{n-1}^{(1)} a_{n-1}^{(2)} \cdots a_{n-1}^{(k-a)} \underbrace{* \cdots *}_{a\ times}, \cdots, a_1^{(1)} a_1^{(2)} \cdots a_1^{(k-a)} \underbrace{* \cdots *}_{a\ times}, \right.$$
$$\left. a_0^{(1)} a_0^{(2)} \cdots a_0^{(k-a)} \underbrace{* \cdots *}_{a\ times} \right),$$

or, equivalently,

$$\left(\underbrace{\left(a_{n-1}^{(1)}, \cdots, a_1^{(1)}, a_0^{(1)} \right)}_{1st\ n-tuple}, \underbrace{\left(a_{n-1}^{(2)}, \cdots, a_1^{(2)}, a_0^{(2)} \right)}_{2nd\ n-tuple}, \right.$$
$$\left. \cdots, \underbrace{\left(a_{n-1}^{(k-a)}, \cdots, a_1^{(k-a)}, a_0^{(k)} \right)}_{(k-a)th\ n-tuple} \right).$$

It is noted that the first n-tuple identifies one of 2^n $Q_n^{2^{k-1}}$s which are generated by the first step of the Isomorphic partitioning.

We now consider how to implement the isomorphic allocation strategy with the addressing scheme discussed above. The Isomorphic allocation strategy maintains a set of linked *status bitmaps* and *free lists*, as shown in Fig. 4. Each status bitmap is 2^n-bit wide (4-bit in this case) and it
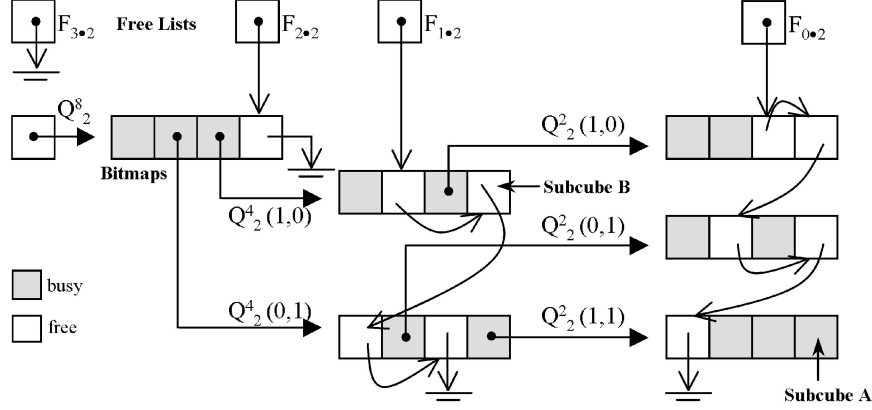
Fig. 4. Implementation of isomorphic allocation strategy with free lists and status bitmaps.

indicates the availability of its children subcubes or nodes. The tree structure in Fig. 3b is maintained by links between the bitmaps. The figure also shows another important data structure managing free and busy subcubes. A free list $F_{a \cdot n}$ is a linked list of free subcubes of $Q_n^{2^a}$

$$(\underbrace{2^a \times 2^a \times \cdots \times 2^a}_{n \; times}).$$

In this example, they are $F_{3 \cdot 2}$, $F_{2 \cdot 2}$, $F_{1 \cdot 2}$, and $F_{0 \cdot 2}$ which maintains available isomorphic subcubes of $Q_2^{2^3}$, $Q_2^{2^2}$, $Q_2^{2^1}$, and $Q_2^{2^0}$, respectively.

The isomorphic allocation strategy is summarized in `Algorithm 1` (Fig. 5), which allocates isomorphic requests ($Q_n^{2^a}$), having the same cubic lengths in all n dimensions. The `Release` algorithm is used when a job finishes its execution. Since the released subcube has ($2^n - 1$) siblings or buddies, it is necessary to check if they are all free and can be merged.

The Isomorphic allocation strategy is *statically optimal*, which means that any sequence of isomorphic requests can be accommodated if the sum of the request sizes is not larger than the system size and a static environment is assumed (i.e., the assigned nodes are not deallocated). Therefore, it is able to allocate resources as compact as possible so that a large future request can possibly be accommodated. We omit the proof here because the proof steps are almost the same as those of the free list-based allocation algorithm developed for hypercubes [2].

4.2 Isomorphic Allocation Strategy for Cubic and Noncubic Jobs

The basic Isomorphic allocation strategy presented in the previous subsection restricts the job request to be isomorphic ($Q_n^{2^a}$). This subsection extends the basic allocation strategy to handle cubic jobs by introducing *subpartitions* between any two subsequent levels of partitions. For example, if an incoming job requests $2 \times 2 \times 4$ in an 8-ary 3-cube, the basic allocation strategy tries to allocate a $4 \times 4 \times 4$ partition consisting of eight $2 \times 2 \times 2$ subpartitions and, thus, a large number of nodes are wasted. The purpose of the extended strategy is to assign two $2 \times 2 \times 2$ subpartitions instead of all eight, making it more space efficient. In

---

Algorithm 1: Isomorphic Allocation Strategy for Isomorphic Jobs

Request ($Q_n^{2^a}$):

1. If $F_{a \cdot n}$ is not empty, allocate a subcube in $F_{a \cdot n}$ to the job request.

2. Otherwise, search $F_{(a+1) \cdot n}, F_{(a+2) \cdot n}, \cdots, F_{k \cdot n}$ in order until a free subcube is found.

3. If found, perform isomorphic decomposition until a $Q_n^{2^a}$ is obtained and allocate it to the job request. Update the corresponding lists after decomposition.

4. Otherwise, enqueue the job to the system queue.

Release ($Q_n^{2^a}$):

1. Insert the released subcube into $F_{a \cdot n}$.

2. If $F_{a \cdot n}$ contains all the buddies, merge them to make $Q_n^{2^{a+1}}$ and insert it into $F_{(a+1) \cdot n}$.

3. Repeat Step 2 if all its buddies are free.

Fig. 5. Algorithm 1: Isomorphic allocation strategy for isomorphic jobs.
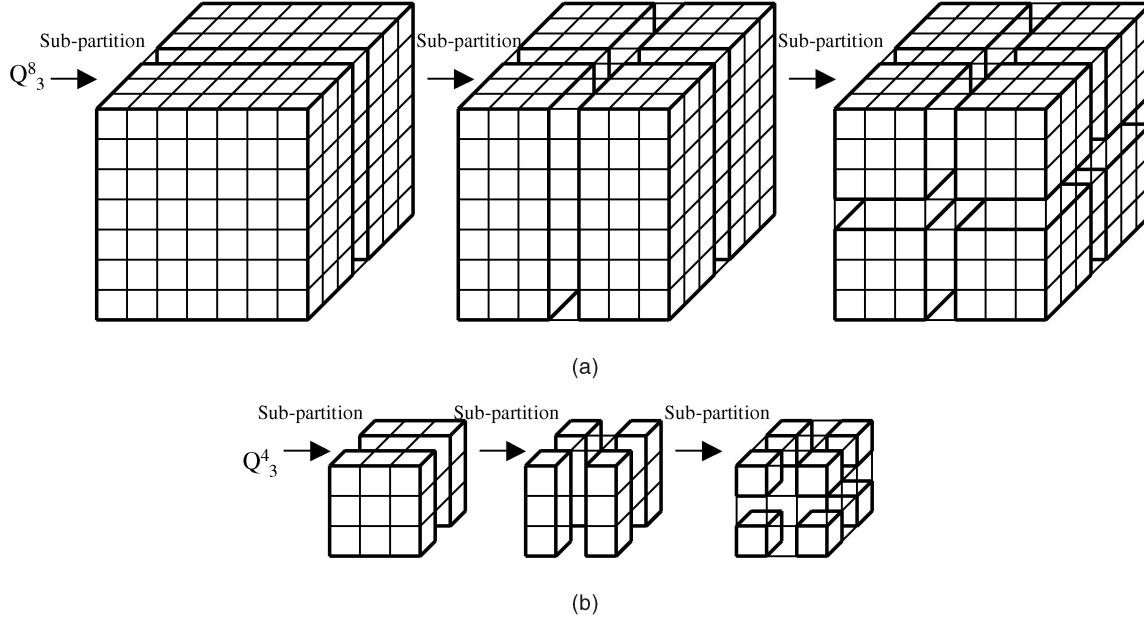
Fig. 6. Isomorphic allocation strategy for cubic jobs in an 8-ary 3-cube system. (a) First partition step. (b) Second partition step.

order to manage the subpartitions efficiently, the subpartitions are considered as nodes of a three-dimensional hypercube. A hypercube allocation algorithm (e.g., Buddy scheme [1], Gray Code [1], or Free List [2]) can be used to manage the subpartitions. Fig. 6 shows subpartitioning of an isomorphic subcube using the Buddy scheme.

In Fig. 6, notice that subpartition steps produce *semi-isomorphic subcubes* in the form of $2^{a_{n-1}} \times \cdots \times 2^{a_1} \times 2^{a_0}$, where $| a_i - a_j | \leq 1$ for all i and j, meaning the lengths of the dimensions need not be the same but different by at most one. For a cubic job requesting a $2^{a_{n-1}} \times \cdots \times 2^{a_1} \times 2^{a_0}$ subcube partition, the extended allocation strategy first adjusts the spatial pattern of the subcube to yield an equivalent semi-isomorphic subcube. For example, if an incoming job requests a $2 \times 2 \times 8$ partition, it is adjusted and allocated a $2 \times 4 \times 4$ partition.[2]

In general, the $2^{a_{n-1}} \times \cdots \times 2^{a_1} \times 2^{a_0}$ subcube partition is translated into

$$\underbrace{2^a \times \cdots \times 2^a}_{(n-l) \ times} \times \underbrace{2^{a+1} \times \cdots \times 2^{a+1}}_{l \ times},$$

where $a = (\ {}^{n-1}_{i=0} a_i) \ div \ n$ and $l = (\ {}^{n-1}_{i=0} a_i) \ mod \ n$. The job requires a partition larger than $Q_n^{2^a}$ but smaller than $Q_n^{2^{a+1}}$ and includes $2^{a \cdot n + l}$ nodes. Thus, 2^l number of $Q_n^{2^a}$ subcubes among 2^n subcubes in a $Q_n^{2^{a+1}}$ need to be merged, e.g., using the Buddy scheme, to accommodate the job. (It is equivalent to allocate an l-cube in an n-cube hypercube using the Buddy scheme.) In other words, the Isomorphic allocation strategy searches the free list $F_{a \cdot n}$ to find 2^l available subcubes according to the hypercube allocation strategy employed (Hypercube_Request and Hypercube_Release[3] procedures). Algorithm 2 (Fig. 7) shows the necessary steps for the Isomorphic allocation strategy for cubic jobs.

The isomorphic allocation strategy described above is further extended to allocate noncubic jobs, where the size of each dimension of a job request is not necessarily a power of two. Suppose that a job requests a 3×5 mesh, the isomorphic allocation strategy for noncubic jobs tries to allocate a subcube of size $2^{\lceil log_2 3 \rceil} \times 2^{\lceil log_2 5 \rceil}$ or 4×8. Fifteen processors are allocated, but the remaining 17 processors are released for future jobs. In general, for an $l_{n-1} \times \cdots \times l_1 \times l_0$ job, the allocation strategy allocates a $2^{\lceil log_2 l_{n-1} \rceil} \times \cdots \times 2^{\lceil log_2 l_1 \rceil} \times 2^{\lceil log_2 l_0 \rceil}$ subcube (in fact, an equivalent semi-isomorphic subcube) and deallocates the rest of the processors for later use.

One major drawback of this algorithm is that it may search for an overly larger subcube than requested while many subpartitions of the subcube will be deallocated immediately. For a 3×5 request, it searches for a 4×8 subcube, the size of which is more than double the requested size. The problem can be addressed by dividing the request into a set of cubic requests and trying to allocate a set of connected subcubes instead of a larger subcube. For example, a 3×5 request is considered as a combination of four cubic requests, 2×4, 2×1, 1×4, and 1×1. More enhancements are possible if the set of cubic requests is translated into a set of semi-isomorphic requests. The third subcube 1×4 becomes 2×2 and, thus, the combined request can be fit into a 4×4 subcube resulting in reduced fragmentation. We do not discuss the details of this extension but leave it as a future work.

---

2. The adjustment is, in essence, to "*fold*" a partition along the longest dimension. The folding process is simple and straightforward if we use the logical node numbers. Moreover, it improves performance by reducing the internode distance.

3. In this paper, we will refer to partition (allocation) and combine (deallocation) procedure for the hypercube algorithms as Hypercube_Request and Hypercube_Release, respectively. The Buddy scheme is the simplest but it does not always recognize a subcube even though one exists, mainly due to fragmentation. Gray code algorithm provides better *subcube recognition ability* [5].

Fig. 7. Algorithm 2: isomorphic allocation strategy for cubic jobs.

4.3 Isomorphic Allocation Strategy for Cubic Systems

The isomorphic allocation strategy for cubic jobs (`Algorithm 2`) can also be applicable to cubic systems in the form of $2^{k_{n-1}} \times \cdots \times 2^{k_1} \times 2^{k_0}$ ($k_{n-1} \quad \cdots \quad k_1 \quad k_0$). In such a system, each node is denoted by an n-tuple $(a_{n-1}, \cdots, a_1, a_0)$. Here, a_i, the node's position in the ith direction, is base 2^{k_i}, which can be represented by a k_i-digit binary number. We can also denote a node's full binary representation as

$$\underbrace{a_{n-1}^{(k_0-k_{n-1}+1)} \cdots a_{n-1}^{(k_0)}}_{k_{n-1} \ times}, \cdots, \underbrace{a_1^{(k_0-k_1+1)} \cdots a_1^{(k_0)}}_{k_1 \ times}, \underbrace{a_0^{(1)} \cdots a_0^{(k_0)}}_{k_0 \ times} ,$$

where $a_i^{(j)} \in \mathbb{Z}^2$. For example, a node in $2^2 \times 2^2 \times 2^4$ system can be represented by (a_2, a_1, a_0), where $a_2, a_1 \in \mathbb{Z}^{2^2}$ and $a_0 \in \mathbb{Z}^{2^4}$, as shown in Fig. 8. The binary representation is given by

$$a_2^{(3)} a_2^{(4)}, a_1^{(3)} a_1^{(4)}, a_0^{(1)} a_0^{(2)} a_0^{(3)} a_0^{(4)} .$$

Rearranging the representation in the order of partition step, it is equivalent to

$$a_0^{(1)} \quad , \quad a_0^{(2)} \quad , \quad a_2^{(3)}, a_1^{(3)}, a_0^{(3)} \quad , \quad a_2^{(4)}, a_1^{(4)}, a_0^{(4)} .$$

The system is partitioned along the longest dimension first so that each subcube becomes more isomorphic. As shown in Fig. 8, the first partition step is based on the binary number $a_0^{(1)}$ and, thus, the system is simply partitioned into two subsystems. This is the same for the second partition step. For the third partition step, three binary numbers $a_2^{(3)}$,
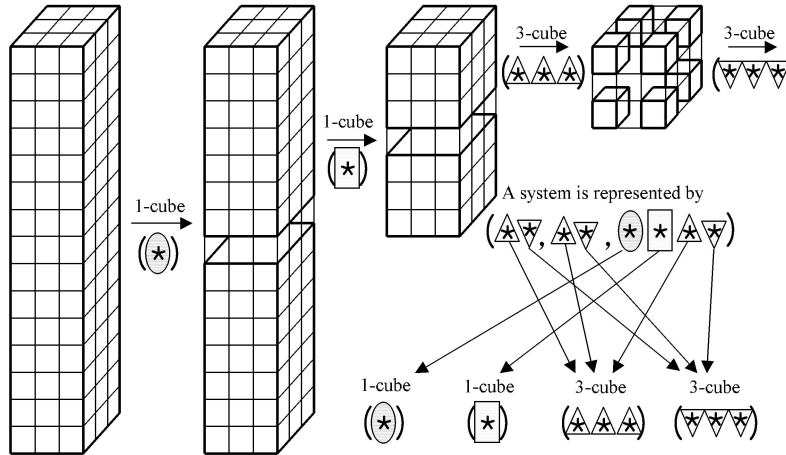

Fig. 8. Isomorphic allocation strategy on a cubic system.

TABLE 1
Algorithm Complexities for Allocating a $Q_m^{2^l}$ Job on a $Q_n^{2^k}$ System ($2^{a \cdot n + l}$-Ary m-Cube Job in Case of `Algorithm 2`)

k-ary n-cube allocation strategy		Time complexity		Space complexity
		Allocation	Deallocation	
Slice allocation	EB, EGC [16][17]	$O(2^{kn})$	$O(2^{km})$	$O(2^{kn})$
	k-ary Partner [18]	$O(2^{kn})$	$O(2^{km})$	$O(2^{kn})$
Job-based allocation	Sniffing [16]	$O(2^{kn} \cdot 2^{lm})$	$O(2^{lm})$	$O(2^{kn})$
	ETC [20]	$O\left(\binom{n}{m} 2^{kn} 2^{lm}\right)$	$O(2^{lm})$	$O(2^{kn})$
Isomorphic allocation	`Algorithm 1`	$O(k)$	$O(k)$	$O(2^{kn})$
	`Algorithm 2`	$O(2^n)$	$O(k + 2^l)$	$O(2^{kn})$

$a_1^{(3)}$, and $a_0^{(3)}$ along the three different dimensions play their roles and, thus, the system is partitioned into eight subsystems. This is also the same for the fourth partition step. In the first and second partition steps, one-bit bitmaps for managing an 1-cube hypercube are used. For the third and fourth step, 8-bit bitmaps are used for managing 3-cube hypercubes.

4.4 Complexity Analysis of the Isomorphic Allocation Algorithms

This subsection analyzes the time and space complexity of the isomorphic allocation algorithms when allocating and deallocating an $Q_m^{2^l}$ job on a $Q_n^{2^k}$ system.

The `Request` Step 1 in `Algorithm 1` (isomorphic allocation algorithm for isomorphic jobs) takes $O(1)$ and Step 2 takes $O(k)$ because the allocator searches at most k free lists. Subcube decomposition process in Step 3 takes at most $O(k)$ since, in the worst case, the largest possible subcube ($Q_n^{2^k}$) is recursively decomposed until a $Q_n^{2^a}$ subcube is obtained. Therefore, the time complexity of allocation is $O(k)$. Deallocation also requires $O(k)$ time assuming that the number of free subcubes is maintained for each free list $F_{a \cdot n}$. The `Release` Steps 1 and 2 take $O(1)$, but Step 3 is repeated at most k times.

The time complexity of `Algorithm 2` (isomorphic allocation algorithm for cubic jobs) depends on the hypercube allocation algorithm employed (`Hypercube_Request` and `Hypercube_Release`). Here, the simple Buddy scheme is assumed as it is used in the performance evaluation study in Section 5. In the Buddy scheme, 2^n allocation bits are used to keep track of the availability of nodes for an n-cube hypercube. The time complexities of the allocation (`Hypercube_Request`) and deallocation (`Hypercube_Release`) of an l-cube job are $O(2^n)$ and $O(2^l)$, respectively. In `Algorithm 2`, for each free list $F_{a \cdot n}$, 2^n allocation bits are used to keep track of the free subcubes ($Q_n^{2^a}$).

The `Request` Step 1 in `Algorithm 2` takes $O(1)$ and Step 2 takes $O(2^n)$ because it calls `Hypercube_Request`. When $F_{a \cdot n}$ is empty or Step 2 fails, the allocator needs to find a higher dimensional subcube, as in Step 3 ($O(k)$), decompose it until an $((a+1) \cdot n)$-cube is obtained, as in Step 4 ($O(k)$), and, finally, calls `Hypercube_Request` in Step 4 ($O(2^n)$). Therefore, the time complexity of allocation is $O(2^n)$. Deallocation takes $O(k + 2^l)$. The `Release` Step 1 takes $O(2^l)$ due to `Hypercube_Release`. Step 2 takes $O(1)$

because only one subcube ($Q_n^{2^{a+1}}$) is released, but it can be repeated k times, as in Step 3.

The space complexity of the Isomorphic allocation algorithm is mainly contributed by the status bitmaps, as shown in Fig. 4 in Section 4.1. Status bitmaps of size 2^n each are used in each partition step to facilitate the merging process. In the first partition pass, a single status bitmap is needed for managing 2^n $Q_n^{2^{k-1}}$s. In the second pass, there can be 2^n bitmaps in the worst case when all of the 2^n $Q_n^{2^{k-1}}$'s are subdivided. Since there are a total of k passes, the number of status bitmaps amounts to $\frac{2^{kn}-1}{2^n-1}$ ($= 1 + 2^n + 2^{2n} + \cdots + 2^{(k-1)n}$) in the worst case. Since each status bitmap is 2^n wide, the space complexity of the Isomorphic algorithm is $O(2^{kn})$. The worst case can happen when all the jobs require only one processor and the system is almost completely utilized. In the normal situation, however, the number will be much smaller and it depends on the load intensity as well as the job size distribution.

Table 1 summarizes complexity analysis of the isomorphic allocation strategy as well as the slice and job-based allocation algorithms. Note that the expressions in Table 1 appear different from those introduced in Section 2 because a $Q_m^{2^l}$ job on a $Q_n^{2^k}$ system is considered instead of a Q_m^l job on a Q_n^k system. For example, $O(k^n)$ in Section 2 is equivalent to $O(2^{kn})$ in Table 1. The space complexity of the slice and job-based allocation algorithms is $O(2^{kn})$ because the processor bitmap of 2^{kn} wide is used in order to keep track of the availability of processors.

5 PERFORMANCE EVALUATION AND COMPARISON

We evaluated and compared the performance of the Isomorphic allocation strategy with other allocation policies in the literature using the CSIM simulation package [21]. Three simulated schemes based on the slice partitioning are EB, EGC [16], [17] and k-ary Partner [18] algorithms, and one based on job-based partitioning is ETC [20]. The Buddy scheme is employed in the `Hypercube_Request` and `Hypercube_Release` procedures for the isomorphic allocation strategy. In Section 5.1, workload model and performance measures are discussed. Simulation results with cubic requests are presented in Section 5.2 and those with noncubic requests are discussed in Section 5.3.

5.1 Workload Model and Performance Parameters

The workload model consists of distribution of job inter-arrival time, job size (subcube size), and job service demand. Job interarrival time and job service demand are usually assumed to follow the exponential distribution, but different distributions such as bimodal hyper-exponential [22], [23], three-stage hyper-exponential [24], and uniform-log [25] distributions have also been suggested. In this paper, we employ a simple but traditional workload model since it covers the general operational conditions of parallel computer systems, thereby fairly assesses and compares the performances of the schemes. In fact, we expect more favorable results for the isomorphic allocation strategy when the parameters of the workload model vary widely. It is because, as will be shown later in Section 5.3 (Figs. 12 and 13), the main strength of the proposed scheme stems from its superior capability in packing subcubes with less external fragmentation, which will be more significant in the operation environment with a widely varying workload. Moreover, the time complexity of the proposed scheme is much smaller than others, as discussed in Section 4.4, which will allow consistently better performance regardless of workload conditions.

The job arrival pattern in our workload model is assumed to follow the Poisson distribution with a rate λ. The arrival rate (λ) is based on the system capacity. This is done to avoid saturation by ensuring that the arrival rate to the system does not exceed the service rate. Total service demand follows an exponential distribution and the mean service time is assumed to be one time unit. Job size is cubic and its distribution is assumed to be uniform in each dimension. In a $2^4 \times 2^4 \times 2^4$ (2^{12} nodes) system, for example, the requested partitions take the form of $2^a \times 2^b \times 2^c$. The probability that a (b or c) is equal to 0, 1, 2, 3, or 4 is $\frac{1}{5}$ each. Since there is a total of 125 cases, the probability of each case is set to $\frac{1}{125}$. One thousand jobs per each random number seed were generated. With 20 seeds, we observed 20,000 jobs, which is sufficient to obtain steady state results. The job size and service demand are assumed to be independent, where a large job (large subcube) has the same distribution of the service demand as that of a small job.

We measured the mean response time, which is a good metric for determining how fast a processor allocation strategy responds to incoming job requests. In order to understand the performance in more detail, the three components of response time were analyzed: service time, *numerical delay*, and *topological delay* [26]. When a job at the head of the job queue fails to be allocated, it is due to one of the following three reasons: The number of processors needed by the job is not sufficient, there is no empty subcube of the requested size in spite of having a sufficient number of processors, or the algorithm has no ability to recognize the candidate even though one exists. The job will be allocated later when all of the busy processors in one of the appropriate subcubes are freed. Based on this observation, we define the numerical delay as the queuing delay incurred when the system does not have sufficient number of available processors needed by a job, while the topological delay is the additional delay experienced when
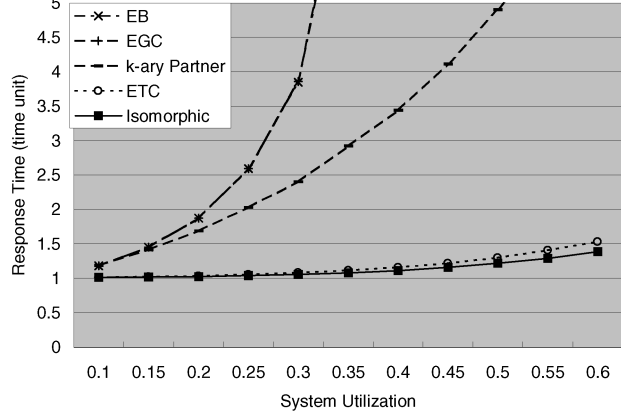

Fig. 9. Comparison of mean response time in a Q_3^4 system.

there is no available subcube in spite of having a sufficient number of processors. After numerical delay, there will be enough free processors. However, scattered placement of the free processors across several subcubes causes topological delay. While the numerical delay depends on the job arrival rate and job service time, the topological delay depends on how efficiently the allocation algorithm manages the processor space. Thus, the topological delay can be considered as a measure of the efficiency of an allocation algorithm.

5.2 Simulation Results with Cubic Requests

Fig. 9 shows the variations in mean response time with respect to system utilization for a 4-ary 3-cube (Q_3^4 or $2^2 \times 2^2 \times 2^2$). The proposed isomorphic allocation strategy outperforms other policies by a considerable margin except ETC. EB and EGC show similar performance and their performance degrades significantly when the system utilization reaches beyond 0.3. The k-ary Partner scheme shows better performance than EB and EGC due to its superior subcube recognition ability. However, the k-ary Partner also exhibits limited performance benefit. This is because internal fragmentation is unavoidable with the conventional slice partitioning. ETC based on the job-based partitioning performs comparably with the proposed scheme, as shown in Fig. 9. However, there is an extremely high cost to perform an exhaustive search during allocation. In addition, the performance improvement for the ETC algorithm is observed only for small systems, as we will see shortly.

Next, we analyze in detail the two best performing strategies, ETC and the proposed isomorphic allocation strategy, for various system sizes. System sizes simulated were 4-ary 3-cube (Q_3^4), 8-ary 3-cube (Q_3^8), and 16-ary 3-cube (Q_3^{16}). The response time of ETC allocation algorithm quickly saturates for larger systems, as shown in Fig. 10b and 10c. On a 16-ary 3-cube system, the saturation point with ETC is about 50 percent utilization of the maximum system capacity. The main cause of the saturation is external fragmentation and it becomes more critical as the system size grows. In contrast, the isomorphic allocation strategy is shown to be scalable, i.e., it exhibits consistent performance irrespective of the system size. The use of
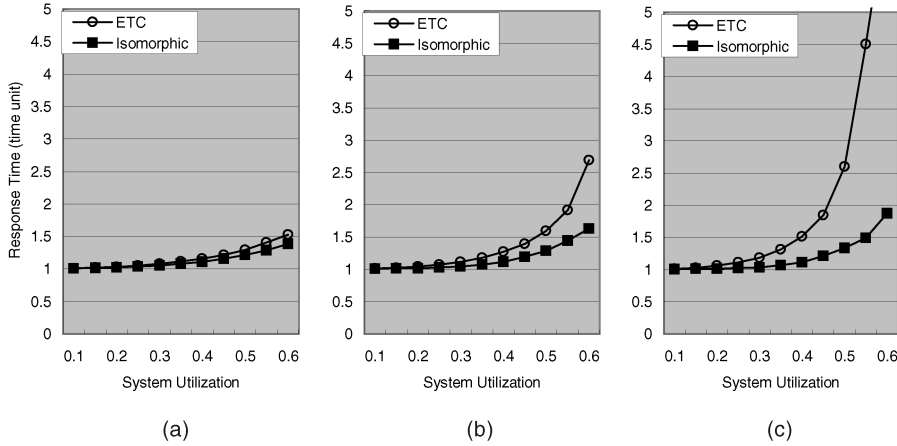
Fig. 10. Mean response time on various system sizes. (a) 4-ary 3-cube. (b) 8-ary 3-cube. (c) 16-ary 3-cube.

systematic partitioning in the isomorphic allocation strategy results in reduced external fragmentation and thus improves the response time. At 50 percent utilization of the maximum system capacity, the isomorphic allocation strategy improves the response time as much as 49 percent compared to the ETC algorithm.

5.3 Simulation Results with Noncubic Requests

This subsection compares the response times of ETC and the isomorphic allocation strategy for noncubic job requests. We assume that the job size follows the uniform distribution across all dimensions. For example, in an $8 \times 8 \times 8$ (2^9 nodes) system, a job requests a partition of the form of $l_2 \times l_1 \times l_0$. Since $0 < l_i \quad 8$, the probability that l_i is equal to $1, 2, \cdots, 8$ is $\frac{1}{8}$ each. For a noncubic job request, the isomorphic allocation strategy allocates a semi-isomorphic partition and the rest of the processors are deallocated immediately. Fig. 11 shows that, for both ETC and the isomorphic allocation strategy, the results are far worse than those with cubic requests. This is mainly due to internal fragmentation. However, the proposed scheme shows better performance than ETC. At 40 percent utilization of the maximum system capacity, the isomorphic allocation strategy improves the response time as much as 45 percent compared to the ETC algorithm.

Figs. 12 and 13 show the numerical and topological delay, respectively, for ETC and the isomorphic allocation strategy. As shown in Fig. 12, the isomorphic allocation strategy incurs less numerical delay than ETC. However, the difference in the topological delay is much more pronounced, as depicted in Fig. 13. This observation leads to the conclusion that the isomorphic allocation strategy partitions the system architecture more efficiently with less external fragmentation and, thus, reduces the topological delay. With ETC, the scattered placement of free processors increases the topological delay even when a sufficient number of processors are available.

6 CONCLUSION AND FUTURE WORK

This paper addresses the processor allocation problem for k-ary n-cube systems. Most of the prior research has been based on slice partitioning, which divides a system topology into a number of lower dimensional slices. However, they suffer from internal fragmentation due to the partitioned cube size limitation. More recently, several allocation algorithms have been proposed based on job-based partitioning. These algorithms greatly improve the system performance compared to those based on slice partitioning, but resort to time-consuming exhaustive
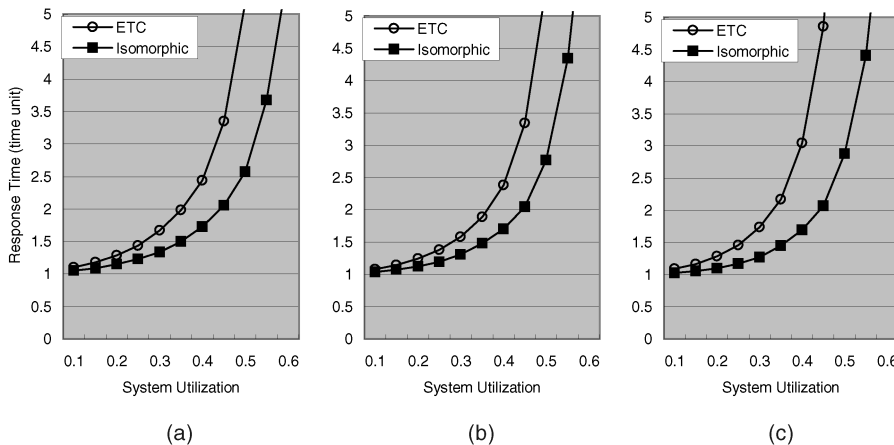

Fig. 11. Mean response time with noncubic job requests on various system sizes. (a) 4-ary 3-cube. (b) 8-ary 3-cube. (c) 16-ary 3-cube.
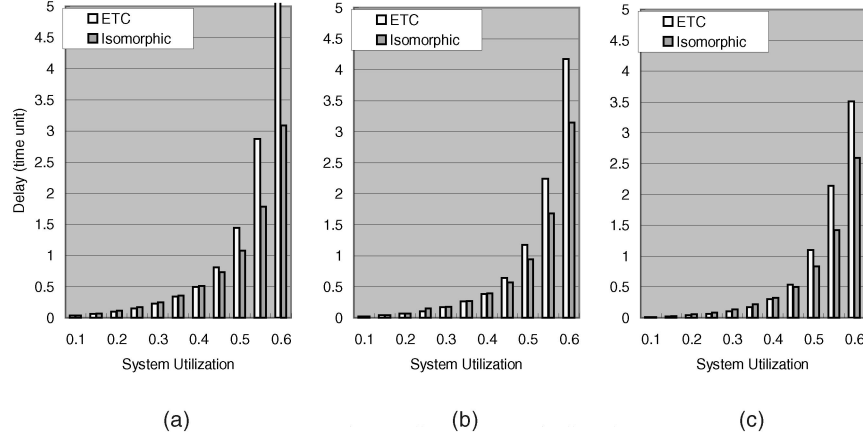
Fig. 12. Numerical delay with noncubic job requests on various system sizes. (a) 4-ary 3-cube. (b) 8-ary 3-cube. (c) 16-ary 3-cube.
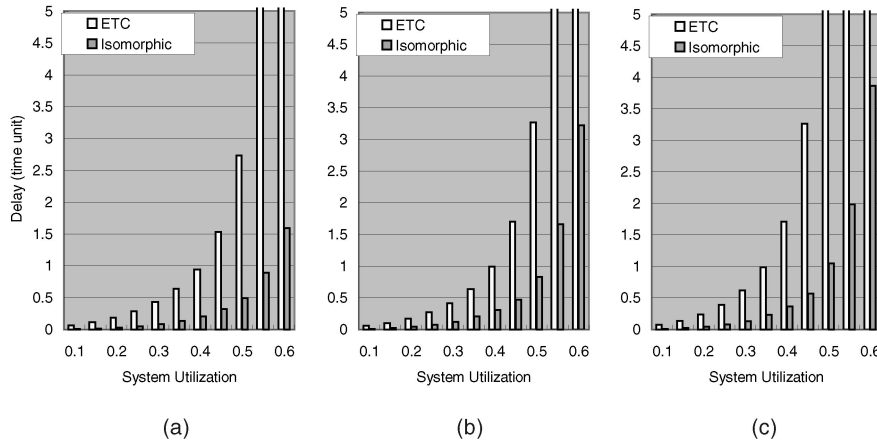

Fig. 13. Topological delay with noncubic job requests on various system sizes. (a) 4-ary 3-cube. (b) 8-ary 3-cube. (c) 16-ary 3-cube.

search. In contrast, our proposed isomorphic partitioning efficiently divides a system into the same dimensional subcubes so that the external as well as internal fragmentation are minimized. Simulation study shows the isomorphic allocation strategy outperforms all existing methods. Moreover, the resulting partitions are characterized by the same order of dimension as the whole system and, thus, retain the advantages of a high order architecture. We also extended the isomorphic allocation strategy to handle cubic and noncubic jobs and showed that it can also be applied to cubic architectures.

The isomorphic partitioning mechanism is a novel method for partitioning a k-ary n-cube topology. Allocation on other interconnection networks, such as meshes, can also be improved with the proposed isomorphic partitioning. In-depth study of numerical and topological delay will be interesting and may allow for a better understanding of the behavior of the processor allocation algorithms. As a future work, we plan to study an adaptive solution which uses a time efficient and space efficient adaptive algorithm depending on load intensity and workload distribution. Since the status bitmaps are created and relinquished dynamically as needed, they can be managed independently via separate `Hypercube_Request` and `Hypercube_Release` procedure. For example, one bitmap can be managed by a Buddy scheme while the other bitmap by a

space efficient scheme. When there are many large jobs and few small jobs, the higher dimensional subcubes need to be managed by a space efficient scheme while the lower dimensional subcubes are better managed by a time efficient Buddy scheme.

ACKNOWLEDGMENTS

REFERENCES

[1] M.S. Chen and K.G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Trans. Computers,* vol. 36, no. 12, pp. 1396-1407, Dec. 1987.

[2] J. Kim, C.R. Das, and W. Lin, "A Top-Down Processor Allocation Scheme for Hypercube Computers," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, no. 1, pp. 20-30, Jan. 1991.

[3] S. Dutt and J.P. Hayes, "Subcube Allocation in Hypercube Computers," *IEEE Trans. Computers,* vol. 40, no. 3, pp. 341-352, Mar. 1991.

[4] P.J. Chuang and N.F. Tzeng, "A Fast Recognition-Complete Processor Allocation Strategy for Hypercube Computers," *IEEE Trans. Computers,* vol. 41, no. 4, pp. 467-479, Apr. 1992.

[5] C. Yu and C.R. Das, "Limit Allocation: An Efficient Processor Management Scheme for Hypercubes," *Proc. Int'l Conf. Parallel Processing,* pp. 143-150, 1994.

[6] C. Chang and P. Mohapatra, "Improving Performance of Mesh Connected Multicomputers by Reducing Fragmentation," *J. Parallel and Distributed Computing,* July 1998.

[7] S-M. Yoo, H.Y. Yoon, and B. Shiraz, "An Efficient Task Allocation Strategies for 2D Mesh Architectures," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 9, pp. 934-942, Sept. 1997.

[8] D.D. Sharma and D.K. Pradhan, "Job Scheduling in Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 1, pp. 57-70, Jan. 1998.

[9] G. Kim and H.Y. Yoon, "On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes," *IEEE Trans. Parallel and Distributed Systems,* vol. 9, no. 2, pp. 175-185, Feb. 1998.

[10] W.J. Dally, "Performance Analysis of k-Ary n-Cube Interconnection Networks," *IEEE Trans. Computers,* vol. 39, no. 6, pp. 775-785, June 1990.

[11] A. Agarwal, "Limits on Interconnection Network Performance," *IEEE Trans. Parallel and Distributed Systems,* vol. 2, pp. 398-412, Oct. 1991.

[12] P. Ramanathan and S. Chalasani, "Resource Placement with Multiple Adjacency Constrains in k-Ary n-Cubes," *IEEE Trans. Parallel and Distributed Systems,* vol. 6, no. 5, pp. 511-519, May 1995.

[13] B. Bose, B. Broeg, Y. Kwon, and Y. Ashir, "Lee Distance and Topological Properties of k-Ary n-Cubes," *IEEE Trans. Computers,* vol. 44, no. 8, pp. 1021-1030, Aug. 1995.

[14] K. Day and A.E. Al-Ayyoub, "Fault Diameter of k-Ary n-Cube Networks," *IEEE Trans. Parallel and Distributed Systems,* vol. 8, no. 9, pp. 903-907, Sept. 1997.

[15] D.K. Panda, S. Singal, and R. Kesavan, "Multidestination Message Passing in Wormhole k-Ary n-Cube Networks with Base Routing Conformed Paths," *IEEE Trans. Parallel and Distributed Systems,* vol. 10, no. 1, pp. 76-96, Jan. 1999.

[16] V. Gautam and V. Chaudhary, "Subcube Allocation Strategies in a K-Ary N-Cube," *Proc. Int'l Conf. Parallel and Distributed Computing and Systems,* pp. 141-146, 1993.

[17] G. Dommety, V. Chaudhary, and B. Sabata, "Strategies for Processor Allocation in k-Ary n-Cubes," *Proc. Int'l Conf. Parallel and Distributed Computing and Systems,* pp. 216-221, 1995.

[18] K. Windisch, V. Lo, and B. Bose, "Contiguous and Non-Contiguous Processor Allocation Algorithms for k-Ary n-Cubes," *Proc. Int'l Conf. Parallel Processing,* 1995.

[19] H.L. Chen and C.T. King, "Efficient Dynamic Processor Allocation for k-Ary n-Cube Massive Parallel Processors," *Computers Math. Applications,* pp. 59-73, 1997.

[20] P.J. Chuang and C.M. Wu, "An Efficient Recognition-Complete Processor Allocation Strategy for k-Ary n-Cube Multiprocessors," *IEEE Trans. Parallel and Distributed Systems,* vol. 11, no. 5, pp. 485-490, May 2000.

[21] Mesquite Software, Inc., *User's Guide: CSIM18 Simulation Engine (C++ Version),* 1998.

[22] S. Majumdar, D.L. Eager, and R.B. Bunt, "Scheduling in Multi-programmed Parallel Systems," *Proc. ACM SIGMETRICS Conf. Measuring and Modeling of Computer Systems,* pp. 104-113, 1988.

[23] J. Jann, P. Pattnaik, H. Franke, F. Wang, J. Skovira, and J. Riordan, "Modeling of Workload in MPPs," *Job Scheduling Strategies for Parallel Processing,* pp. 95-116, 1997.

[24] D. Feitelson, L. Rudolph, U. Schwiegelshohn, K. Sevcik, and P. Wong, "Theory and Practice in Parallel Job Scheduling," *Job Scheduling Strategies for Parallel Processing,* pp. 1-34, 1997.

[25] M. H-Balter and A. Downey, "Exploiting Process Lifetime Distributions for Dynamic Load Balancing," *Proc. 15th ACM Symp. Operating Systems Principles,* 1995.

[26] M. Kang and C. Yu, "Job-Based Queue Delay Modeling in a Space-Shared Hypercube," *Proc. ICPP Workshop Parallel Computing,* pp. 313-318, 1999.