Subcube Allocation and Task Migration in Hypercube Multiprocessors

MING-SYAN CHEN, MEMBER, IEEE, AND KANG G. SHIN, SENIOR MEMBER, IEEE

Abstract—**This paper addresses two important issues in the management of processors in a hypercube: *subcube allocation* and *task migration* to eliminate the system fragmentation caused by allocation and deallocation of subcubes.**

We prove that the subcube allocation strategy using a binary reflected Gray code (BRGC), called the *GC strategy*, possesses the best subcube recognition ability among all strategies that use sequential searches. A binary code (BC) is defined as the binary representation of a nonnegative integer, and an extended binary code (EBC) is the one obtained by permuting the bits of a BC. Similarly, an extended Gray code (EGC) is obtained from a BRGC. The subcube recognition ability of an allocation strategy using multiple EBC's is analyzed and compared to that using multiple EGC's. The minimal number of EBC's required for complete subcube recognition in an n-cube or Q_n is proved to be $C^n_{\lfloor n/2 \rfloor}$, where C^q_p stands for the number of combinations of choosing p out of q possibilities.

Allocation and deallocation of subcubes usually result in a fragmented hypercube, where even if a sufficient number of hypercube nodes are available, they do not form a subcube large enough to accommodate an incoming task. As the fragmentation in conventional memory allocation can be handled by memory compaction, the fragmentation problem in a hypercube can be solved by *task migration*, i.e., relocating tasks within the hypercube to remove the fragmentation. Note that the procedure for task migration is closely related to the subcube allocation strategy used, since active tasks must be relocated in such a way that the availability of subcubes can be detected by that allocation strategy. Specifically, we develop the task migration strategy for the GC strategy. A goal configuration (of destination subcubes) without fragmentation is determined first. Then, the node-mapping between the source and destination subcubes is derived. Finally, a routing procedure to obtain shortest deadlock-free paths for relocating tasks is developed.

Index Terms—**Binary reflected Gray code (BRGC), deadlock, fragmentation, hypercube computers, stepwise adjoint, subcube recognition ability, task migration.**

I. INTRODUCTION

OWING to their structural regularity and high potential for the parallel execution of various algorithms, hypercube computers have drawn considerable attention in recent years from both academic and industrial communities [1]–[6].

Each task arriving at the hypercube multiprocessor must be allocated to an unoccupied or available subcube for execution. Upon completion of the task, the subcube used for that task is released and made available for other tasks. More specifically, subcube allocation[1] in a hypercube multiprocessor consists of two steps: 1) *determination* of the dimension of a subcube required to execute each incoming task, and 2) *location* of a subcube of the dimension determined by step 1) within the hypercube. Various approaches have been developed to deal with the first step [4], [7]. The second step was addressed in [3], where a subcube allocation strategy was proposed based on a binary reflected Gray code (BRGC), called the *GC strategy*, as opposed to the one based on a binary code (BC), called the *buddy strategy* [8]. The former is shown to outperform the latter due mainly to its superiority in recognizing the existence of available subcubes within the hypercube.

Similarly to conventional memory management, the allocation and deallocation of subcubes usually results in a fragmented hypercube, where even if a sufficient number of nodes are available or unoccupied, they do not form a subcube large enough to accommodate an incoming task. Fig. 1 shows an example of a fragmented hypercube where four available nodes cannot form a 2-cube or Q_2; thus, if a task requiring a Q_2 arrives, it has to be either queued or rejected. As shown in the simulation results in [3], such fragmentation leads to poor utilization of hypercube nodes, and the improvement achieved by the GC strategy is thus limited. As the fragmentation problem in conventional memory allocation can be handled by memory compaction, the fragmentation problem in a hypercube can be solved by *task migration*, i.e., relocating and compacting active tasks[2] within the hypercube at one end so as to make large subcubes available at the other end. Note that the procedure for task migration depends strongly on the subcube allocation strategy used, since active tasks must be relocated in such a way that the availability of subcubes can be detected by that allocation strategy. For this reason, we shall in this paper address both the problems of subcube allocation and task migration.

We shall extend the results on subcube allocation in [3] significantly. To facilitate our discussion, the buddy and GC strategies introduced in [3] are described briefly. A BC is de-

Manuscript received December 11, 1987; revised September 15, 1988. This work was supported in part by the Office of Naval Research under Contracts N00014-85-K-0122 and N00014-85-K-0531.

M. S. Chen was with the Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109. He is now with the IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598.

K. G. Shin is with Real-Time Computing Laboratory, Department of Electrical Engineering and Computer Science, The University of Michigan, Ann Arbor, MI 48109.

IEEE Log Number 9037192.

[1] In view of the fact that each task is allocated to a subcube, we use the term *subcube allocation*, instead of processor allocation which was used in [3].

[2] Those tasks which are allocated to subcubes but not completed yet.

0018-9340/90/0990-1146$01.00

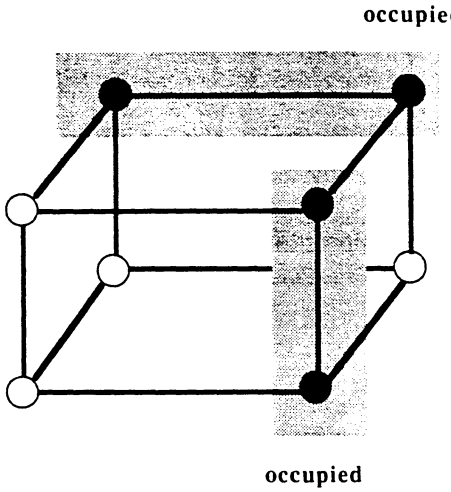

Fig. 1. An example of hypercube fragmentation.

fined as the binary representation of a nonnegative integer, and an extended binary code (EBC) is the one obtained by permuting the bits of a BC. Similarly, an extended Gray code (EGC) is defined as a code obtained by bit-permutation of a BRGC. The subcube recognition ability of an allocation strategy using multiple EBC's is analyzed and compared to that using multiple EGC's. Since, as will be proved later, no other strategy that involves a sequential search can recognize more subcubes than the GC strategy, we shall focus on the development of a task migration strategy under the GC strategy.

A collection of occupied subcubes is called a *configuration*. We first determine the goal configuration to which a given fragmented hypercube must change by relocating active tasks. Since the GC strategy is proved in [3] to be optimal for static allocation,[3] fragmentation can definitely be removed by task migration. When a task is allocated to a subcube, the portion of the task located in each node of this subcube is called a *task module*. The action for a node to move its task module to one of its neighboring nodes is called a *moving step*. The cost of each task migration is then measured in terms of the number of required moving steps while task migrations between different pairs of source and destination subcubes are allowed to be performed *in parallel*. Note that to move tasks in parallel, it is very important to avoid deadlocks during task migration. We not only formulate the node-mapping between each pair of source and destination subcubes in such a way that the number of required moving steps is minimized, but also develop a routing procedure to follow shortest deadlock-free paths for task migration.

The paper is organized as follows. Section II introduces the necessary definitions and notation. Some important results of subcube allocation strategies are presented in Section III. Our main results on task migration are given in Section IV. In three subsections are, respectively, presented the three steps for task migration under the GC strategy: 1) determination of a goal configuration, 2) determination of the node-mapping between the source and destination subcubes, and 3) determination of shortest deadlock-free routing for moving task modules. The paper concludes with Section V.

[3]By static allocation, we mean the allocation of subcubes to a sequence of incoming tasks without considering the deallocation of subcubes.

0. 000	0. 000
1. 001	1. 100
2. 011	2. 101
3. 010	3. 001
4. 110	4. 011
5. 111	5. 111
6. 101	6. 110
7. 100	7. 010
(a)	(b)

Fig. 2. Illustration of EGC's. (a) BRGC. (b) EGC with $\{g_1, g_2, g_3\} = \{3, 1, 2\}$.

II. Preliminaries

An n-dimensional hypercube is defined as $Q_n = K_2 \times Q_{n-1}$, where K_2 is the complete graph with two nodes, Q_0 is a trivial graph with one node and $\times$ is the product operation on two graphs [9]. Let $\sum$ be the ternary symbol set $\{0, 1, *\}$, where $*$ is a DON'T CARE symbol. Every subcube in a Q_n can then be uniquely represented by a string of symbols in $\sum$. Such a string of ternary symbols is called the *address* of the corresponding subcube. For example, the address of the subcube Q_2 formed by nodes 0010, 0011, 0110, and 0111 in a Q_4 is $0*1*$. Also, the *Hamming distance* between two hypercube nodes is defined as follows.

Definition 1: The Hamming distance between two nodes with addresses $u = u_n u_{n-1} \cdots u_1$ and $w = w_n w_{n-1} \cdots w_1$ in a Q_n is defined as

$$H(u, w) = \sum_{i=1}^{n} h(u_i, w_i), \text{ where } h(u_i, w_i) = \begin{cases} 1, & \text{if } u_i \neq w_i, \\ 0, & \text{if } u_i = w_i. \end{cases}$$

For convenience, the rightmost coordinate in the address of a subcube or a node will be referred to as *dimension 1*, the second to the rightmost coordinate as *dimension 2*, and so on. A formal definition for extended Gray codes (EGC's) was introduced in [3]. An EGC with parameters g_i, $i = 1, \cdots, n$, can be obtained by permuting the bits in the BRGC in such a way that dimension i of the BRGC becomes dimension g_i of the EGC. Fig. 2(a) gives an example of a BRGC which is also an EGC with parameters $\{g_1, g_2, g_3\} = \{1, 2, 3\}$, and Fig. 2(b) shows an EGC with parameters $\{g_1, g_2, g_3\} = \{3, 1, 2\}$. Similarly, an extended binary code (EBC) is defined as follows.

Definition 2: An EBC with parameters g_i, $i = 1, \cdots, n$, is a code resulting from permuting the bits in a BC in such a way that dimension i of the BC becomes dimension g_i of this EBC. (Recall that a BC is the binary representation of a nonnegative integer.)

Obviously, the BC is also an EBC with parameters $\{g_1, g_2, g_3\} = \{1, 2, 3\}$. Examples for the BC and the EBC with parameters $\{g_1, g_2, g_3\} = \{3, 1, 2\}$ are presented in Fig. 3. Also, a set of contiguous integers is called a *region* and let $\#[a, b] = \{k | a \leq k \leq b, k \in I^+\}$. *A coding scheme*

0. 000	0. 000
1. 001	1. 100
2. 010	2. 001
3. 011	3. 101
4. 100	4. 010
5. 101	5. 110
6. 110	6. 011
7. 111	7. 111
(a)	(b)

Fig. 3. Illustration of EBC's. (a) BC. (b) EBC with $\{g_1, g_2, g_3\} = \{3, 1, 2\}$.

0. 0000
1. 0001
2. 0011
3. 0010
4. 0110
5. 0111
6. 1111
7. 1110
8. 1100
9. 1000
10. 1010
11. 1011
12. 1001
13. 1101
14. 0101
15. 0100

Fig. 4. A coding scheme with 4 bits.

with n bits, denoted by C_n, is defined as a one-to-one mapping from a number within $\#[0, 2^n - 1]$ to a binary representation with n bits, and $C_n(m)$ denotes the representation of a number m with n bits under a given coding scheme. Note that an EGC with given parameters corresponds to some coding scheme, and so does an EBC. Figs. 2 and 3 are examples of coding schemes with 3 bits. Fig. 4 gives an example coding scheme with 4 bits which is neither an EGC nor an EBC. For convenience, let B_n and G_n denote n-bit coding schemes associated with the BC and a BRGC, respectively. For example, $B_3(5) = 101$, $G_3(5) = 111$, while $C_4(7) = 1110$ for the coding scheme in Fig. 4.

In addition, a *path* is defined as an ordered sequence of hypercube nodes in which any two consecutive nodes are physically adjacent to each other in the hypercube. Also, we assume that the hardware of the hypercube computer system under consideration is so designed that each hypercube node has separate input and output ports. Thus, each node can receive a task module while sending another task module to its next hop. Each moving step is assumed to take the same amount of time, which is defined as one time unit. $|S|$ denotes the cardinality of the set S and $\bar{b}$ denotes the complement of a bit $b \in \{0, 1\}$. Unless stated otherwise, a set is referred to as an unordered set, i.e., $\{a, b\} = \{c, d\}$ implies that ($a = c$ and $b = d$) or ($a = d$ and $b = c$).

III. Subcube Allocation Strategies

The results on subcube allocation in [3] are extended here and will later be applied to task migration. The subcube recognition ability of the GC strategy was shown to be twice that of the buddy strategy in [3]. This is due to the fact that under the GC strategy the binary strings to be sequentially searched are ordered in a sequence different from those under the buddy strategy. One may naturally raise the following question: "Is there any subcube allocation strategy which can provide an even better subcube recognition ability than the GC strategy by ordering the binary strings to be searched in some other sequence?" To answer this question, we shall prove in Theorem 1 below that the GC strategy is optimal in the sense that no other strategy that involves a sequential search can recognize more subcubes than the GC strategy. By a "sequential search" we mean a search that checks the availability of each hypercube node in a given sequence until a set of consecutive unoccupied nodes forming a subcube of required size is encountered. The subcube recognition ability of an allocation strategy using multiple EBC's will also be analyzed and compared to that using multiple EGC's. The minimal number of EBC's required for complete subcube recognition in a Q_n is proved to be $C^n_{\lfloor n/2 \rfloor}$, whereas the number of EGC's required for complete subcube recognition is less than or equal to $C^n_{\lfloor n/2 \rfloor}$. Our results are shown to favor the use of the GC strategy over any other strategy based on a sequential search for subcubes.

A. Subcube Allocation Strategies Using a Single Code

Note that both the buddy and GC strategies are a first-fit sequential search. Node addresses under the buddy and GC strategies are ordered in a list according to the BC and the BRGC, respectively. Such a list is called an *allocation list*. Under each of these strategies, the availability of hypercube nodes is then kept track of by its allocation list. Suppose $k = |I_i|$ is the dimension of a subcube required to execute an incoming task I_i. The buddy strategy will search for a set of 2^k consecutive unoccupied nodes in its allocation list, say $B_n(p), B_n(p+1), \cdots, B_n(p+2^k-1)$, under the constraint that p is a multiple of 2^k. On the other hand, the GC strategy searches for a set of 2^k consecutive unoccupied nodes in its allocation list, say $G_n(p), G_n(p+1), \cdots, G_n(p+2^k-1)$, but in this case p only has to be a multiple of 2^{k-1}, instead of 2^k. A formal description of both strategies is given in Appendix A. Illustrative examples for the operations under both strategies can be found in Fig. 5.

It is proved in [3] that the GC strategy can recognize $2^{n-k+1}Q_k$'s for $1 \le k \le n-1$, which is twice the number of subcubes recognizable by the buddy strategy. Moreover, we shall prove that the GC strategy is optimal insofar as the subcube recognition ability of a sequential search is concerned. To facilitate this proof, it is necessary to introduce the following lemma first.

Lemma 1: The intersection of two overlapping subcubes $\alpha = a_n a_{n-1} \cdots a_1$ and $\beta = b_n b_{n-1} \cdots b_1$ is a subcube with address $\gamma = c_n c_{n-1} \cdots c_1$, where

$$c_i = \begin{cases} *, & \text{if } a_i = b_i = *, \\ 0, & \text{if } a_i = b_i = 0 \text{ or } \{a_i, b_i\} = \{0, *\}, \\ 1, & \text{if } a_i = b_i = 1 \text{ or } \{a_i, b_i\} = \{1, *\}. \end{cases}$$

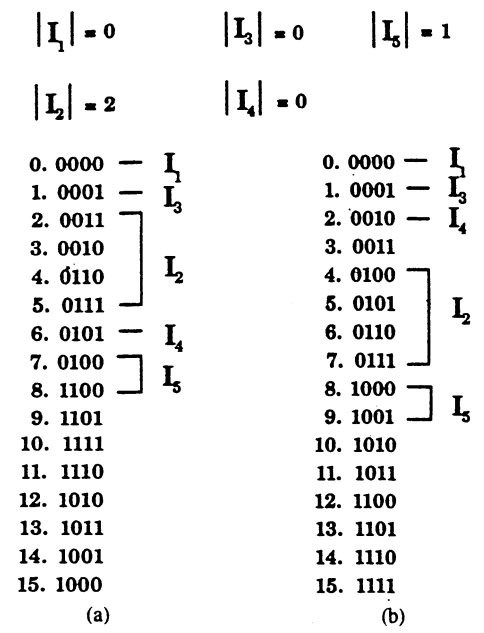

Fig. 5. Example operations of both strategies. (a) GC strategy. (b) Buddy strategy.

Proof: Note that subcubes α and β are disjoint iff there exists a k such that $\{a_k, b_k\} = \{0, 1\}$. Therefore, the above equation includes all possible combinations of a_i and b_i, $\forall i$, since α and β are overlapping. It can be seen that nodes in γ are contained in both α and β.

On the other hand, suppose there is a node m with address $m_n m_{n-1} \cdots m_1$ that does not belong to γ. Then, there must exist a j such that $\{m_j, c_j\} = \{0, 1\}$. Without loss of generality, we can assume $m_j = 0$ and $c_j = 1$. Then, we have either $a_j = 1$ or $b_j = 1$, meaning that m is not contained in the intersection of α and β. Therefore, nodes in γ are the only nodes that are contained in both α and β, and thus this lemma follows. Q.E.D.

By Lemma 1, the intersection of two overlapping subcubes must be a subcube. This leads to the following important property of the BRGC.

Theorem 1: The BRGC is an optimal coding scheme as far as the subcube recognition ability of a sequential search is concerned.

Proof: Note that there are $2^{n-k+1}Q_k$'s recognizable by the BRGC. Suppose there exists a coding scheme that can recognize more than $2^{n-k+1}Q_k$'s. Then, there must exist two distinct integers i and j such that a) $|i - j| < 2^{k-1}$, b) nodes with addresses $C_n(p)$, $p \in \#[i, i+2^k-1]$, form a Q_k, and c) nodes with addresses $C_n(q)$, $q \in \#[j, j+2^k-1]$, form another Q_k. Thus, the cardinality of $\#[i, i+2^k-1] \cap \#[j, j+2^k-1]$ must be greater than 2^{k-1} since $|i-j| < 2^{k-1}$, and less than 2^k since $i \neq j$. However, this means that the intersection of these two regions does not form a subcube, a contradiction to Lemma 1. Q.E.D.

Notice that not only the BRGC but also each EGC has an optimal subcube recognition ability since each EGC can be obtained by permuting the bits of the BRGC.

B. Subcube Allocation Strategies Using Multiple Codes

In [3], we developed the following theorem to determine the subcube recognition ability of an EGC with given parameters.

Theorem 2 [3]: A subcube Q_k with the address $b_n b_{n-1} \cdots b_1$ can be recognized by an EGC with parameters g_i, $1 \le i \le n$, iff any of the following three conditions is satisfied:

a) $b_{g_i} = *$, $1 \le i \le k$.

b) $b_{g_i} = *$, $1 \le i \le k-1$, and there exists an r such that $b_{g_{r-1}} = 1$, $b_{g_r} = *$, and $b_{g_s} = 0$, $k \le s < r-1$.

c) $b_{g_i} = *$, $1 \le i \le k-1$, $b_{g_s} = 0$, $k \le s \le n-1$, and $b_{g_n} = *$.

Similarly, the subcube recognition ability of an EBC with given parameters can be determined by the lemma below whose proof is trivial, and thus, omitted.

Lemma 2: A subcube Q_k with the address $b_n b_{n-1} \cdots b_1$ can be recognized by an EBC with parameters g_i, $1 \le i \le n$, iff $b_{g_i} = *$, for $1 \le i \le k$.

Let $R(n)$ denote the number of EBC's required for complete subcube recognition in a Q_n. We then obtain the following important result.

Theorem 3: $R(n) = C^n_{\lfloor n/2 \rfloor}$.

Proof: Note that there are $C^n_k 2^{n-k} Q_k$'s in a Q_n, and from Lemma 2, each EBC can only recognize $2^{n-k}Q_k$'s. Therefore, it requires at least C^n_k EBC's to recognize all Q_k's in a Q_n. However, $C^n_{\lfloor n/2 \rfloor} = \max_{0 \le k \le n} \{C^n_k\}$, leading to $R(n) \ge C^n_{\lfloor n/2 \rfloor}$.

The inequality $R(n) \le C^n_{\lfloor n/2 \rfloor}$ can be proved in light of the Theorem of Matching [10] by using the same procedure as in the proof of Theorem 4 in [3]. Q.E.D.

Notice that the number of EGC's required for complete subcube recognition, as proved in [3], is less than or equal to $C^n_{\lfloor n/2 \rfloor}$. Naturally, this is due to the superiority of the subcube recognition ability of EGC's to that of EBC's. Following the same procedure as in the proof of Theorem 4 in [3], we obtain the parameters of those EBC's required for complete subcube recognition in a Q_5: $\{1, 2, 3, 4, 5\}$, $\{4, 5, 2, 1, 3\}$, $\{3, 5, 4, 1, 2\}$, $\{2, 5, 1, 3, 4\}$, $\{3, 4, 2, 5, 1\}$, $\{5, 1, 3, 2, 4\}$, $\{4, 1, 5, 2, 3\}$, $\{2, 4, 1, 5, 3\}$, $\{3, 1, 4, 5, 2\}$, $\{2, 3, 5, 1, 4\}$. Let B^i denote the ith EBC in the ten EBC's and $SB_k = \cup^k_{i=1} B^i$. The subcube recognition abilities of SB_h, $1 \le h \le 10$, are shown in Table I. For the comparison purpose, the subcube recognition ability of those EGC's with the same parameters is shown in Table II, where G^i is the ith EGC and $SG_k = \cup^k_{i=1} G^i$. The number of recognizable subcubes of each dimension with multiple EBC's and that with multiple EGC's are plotted in Fig. 6, where trivial cases for Q_0 and Q_5 are omitted. From Fig. 6, it is easy to see that the subcube recognition ability of EGC's is superior to that of EBC's.

IV. Task Migration under the GC Strategy

Due to similar reasons for the memory fragmentation in conventional memory allocation, allocation and deallocation of subcubes may result in a fragmented hypercube. As shown in the simulation results in [3], although the GC strategy outperforms the buddy strategy, the improvement achieved by using multiple codes is rather limited. This fact in turn implies that poor utilization of hypercube nodes is due mainly to system fragmentation, rather than the subcube recognition ability of an allocation strategy. Consequently, it is essential

TABLE I
THE NUMBER OF SUBCUBES RECOGNIZABLE BY SB_h, $1 \leq h \leq 10$

Q_k \ SB_h	h=1	h=2	h=3	h=4	h=5	h=6	h=7	h=8	h=9	h=10
k=0	32	32	32	32	32	32	32	32	32	32
k=1	16	32	48	64	64	80	80	80	80	80
k=2	8	16	24	32	40	48	56	64	72	80
k=3	4	8	12	16	20	24	28	32	36	40
k=4	2	4	6	8	10	10	10	10	10	10
k=5	1	1	1	1	1	1	1	1	1	1

TABLE II
THE NUMBER OF SUBCUBES RECOGNIZABLE BY SG_h, $1 \leq h \leq 10$

Q_k \ SG_h	h=1	h=2	h=3	h=4	h=5	h=6	h=7	h=8	h=9	h=10
k=0	32	32	32	32	32	32	32	32	32	32
k=1	32	57	72	79	79	80	80	80	80	80
k=2	16	32	47	58	65	71	75	78	79	80
k=3	8	16	22	25	31	34	37	38	39	40
k=4	4	8	10	10	10	10	10	10	10	10
k=5	1	1	1	1	1	1	1	1	1	1

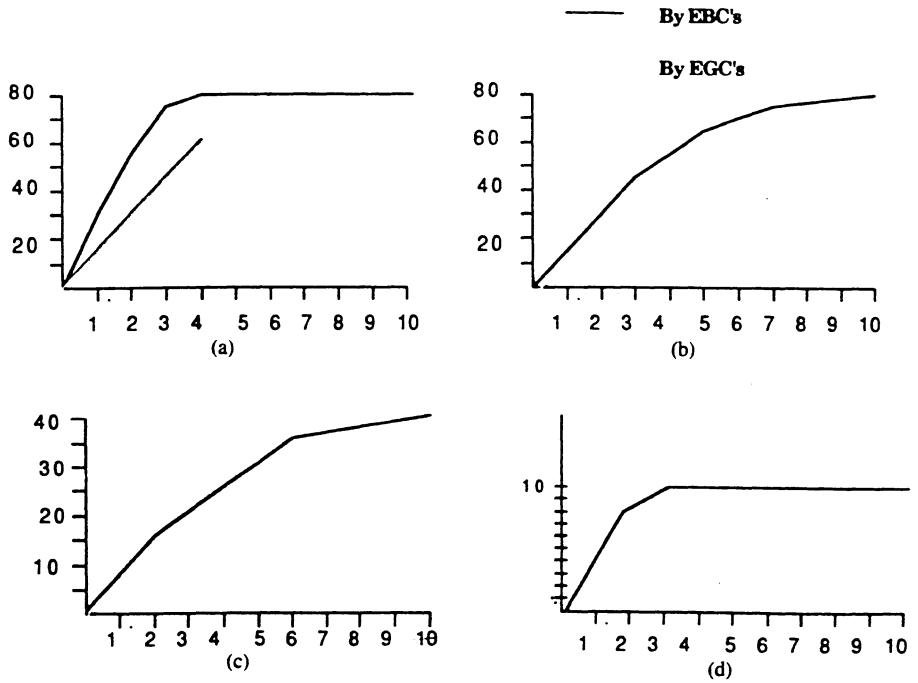

Fig. 6. The number of recognizable subcubes of each dimension. (a) One-dimensional cubes. (b) Two-dimensional cubes. (c) Three-dimensional cubes. (d) Four-dimensional cubes.

to develop an efficient procedure for task migration under the GC strategy, which relocates active tasks to eliminate the fragmentation. In the following three subsections, we shall determine, respectively, the goal configuration, the node-mapping between the source and destination subcubes, and shortest deadlock-free paths for task migration.

A. Determination of Goal Configuration

There are usually many ways to relocate active tasks and compact occupied subcubes. However, since it is desirable to perform task migration between each pair of subcubes in parallel, it is important to avoid any deadlock during the migration. Clearly, a deadlock might occur if there is a circular wait among the nodes involved. To prevent this, a linear ordering of hypercube nodes is established in such a way that each node can only move its task module to a node with a lower address, i.e., a node with address $G_n(p)$ sends its task module to another node with address $G_n(q)$, only if $p > q$. Since a node with a lower address never sends its task module to a node with a higher address, it is easy to see that the condition for any circular wait can be avoided by the above method. Thus, given a configuration of occupied subcubes, the goal configuration without fragmentation can be determined by the following algorithm.

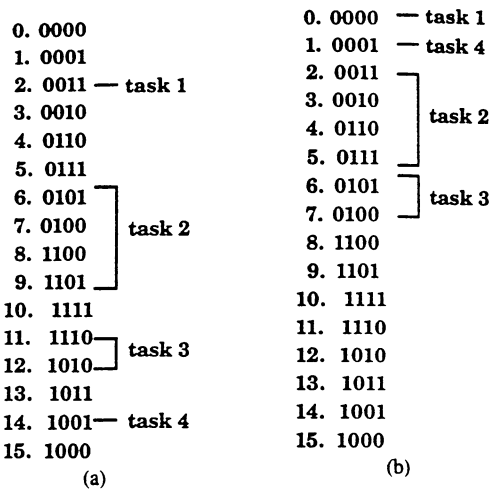

Fig. 7. Task migration under the GC strategy. (a) Before. (b) After.

Algorithm A_1: Determination of the goal configuration:

Step 1. Label each task in the availability list with a distinct number in such a way that each task allocated to a subcube with a lower address is labeled with a smaller number.

Step 2. Relocate all tasks according to an increasing order of their labels.

For example, the goal configuration in Fig. 7(b) can be derived from the initial fragmented configuration in Fig. 7(a). It can be easily verified that by using algorithm A_1, each task will always be moved from a subcube with a higher address to a subcube with a lower address, while a task with a smaller label is not necessarily ahead of a task with a larger label in the goal configuration. An allocation strategy is said to be *statically optimal*, if a Q_n using the strategy can accommodate any input request sequence $\{I_i\}_{i=1}^{k}$ iff $\sum_{i=1}^{k} 2^{|I_i|} \leq 2^n$, where $|I_i|$ is the subcube dimension required by request I_i. As was proved in [3], the GC allocation strategy is statically optimal. This fact implies that fragmentation will definitely be removed by A_1.

B. Node-Mapping Between Source and Destination Subcubes

Once the goal configuration is determined, each active task will be moved from its current or source subcube to the destination subcube. The action for a node to move its task module to one of its neighboring nodes is called a moving step. To determine the number of moving steps for a task migration, we first define the *moving distance* of a task between two subcube locations as follows. As it will be shown in Theorem 4 below, the minimal number of moving steps required to move a task from one subcube location to another can be determined by the moving distance between the two subcube locations. Furthermore, as it will become clear later, when modules of a task are migrated in parallel, the moving distance between two subcube locations is equal to the number of moving steps required to move a task module from the source node to its destination node under the node mapping scheme described in Corollary 4.1.

Definition 3: The moving distance of a task between two subcube locations with addresses $\alpha = a_n a_{n-1} \cdots a_1$ and $\beta = b_n b_{n-1} \cdots b_1$ in a Q_n is defined as

$$M(\alpha, \beta) = \sum_{i=1}^{n} m(a_i, b_i), \text{ where } m(a_i, b_i) = \begin{cases} 1, & \text{if } \{a_i, b_i\} = \{0, 1\}, \\ 0, & \text{if } a_i = b_i, \\ \frac{1}{2}, & \text{otherwise.} \end{cases}$$

Then, we have the following theorem for the minimal number of moving steps required to move a task from one subcube location to another.

Theorem 4: Let $T(\alpha, \beta)$ be the minimal number of moving steps from a subcube location α to another location β. Then, $T(\alpha, \beta) = M(\alpha, \beta)2^{|\alpha|}$, where $|\alpha| = |\beta|$ is the dimension of the subcube.

To facilitate the proof of Theorem 4, it is necessary to introduce the following proposition whose proof can be found in [11].

Proposition 1: Given a node $u \in Q_n$, $\sum_{w \in Q_n} H(u, w) = n2^{n-1}$.

Proof of Theorem 4: We shall prove $T(\alpha, \beta) \geq M(\alpha, \beta)2^{|\alpha|}$ first. Suppose $\alpha = a_n a_{n-1} \cdots a_1$ and $\beta = b_n b_{n-1} \cdots b_1$. We define the *frontier subcube* of α towards β, denoted by $\sigma_{\alpha \to \beta}(\alpha) = f_n f_{n-1} \cdots f_1$, in such a way that, $\forall i$, $f_i = b_i$ if $a_i =^*$ and $b_i \in \{0, 1\}$, and $f_i = a_i$ otherwise. For example, if $\alpha = 00^{**}$ and $\beta = 1^*1^*$, then $\sigma_{\alpha \to \beta}(\alpha) = 001^*$ and $\sigma_{\beta \to \alpha}(\beta) = 101^*$. Clearly, $\sigma_{\alpha \to \beta}(\alpha)$ contains all the nodes in α which are closest to β. Besides, we define the Hamming distance between two subcubes as the shortest distance between any two nodes which, respectively, belong to the two subcubes, i.e., $H^*(\alpha, \beta) = \min_{u \in \alpha, w \in \beta} H(u, w)$. Since we align some bits of α with their corresponding bits of β to obtain $\sigma_{\alpha \to \beta}(\alpha)$, it is easy to see that $\forall u \in \alpha$ and $w \in \beta$, $H(u, w) \geq H^*(u, \sigma_{\alpha \to \beta}(\alpha)) + H^*(\sigma_{\alpha \to \beta}(\alpha), \sigma_{\beta \to \alpha}(\beta)) + H^*(\sigma_{\beta \to \alpha}(\beta), w)$.

Let $u' \in \beta$ denote the node to which the task module originally located at $u \in \alpha$ is to be moved. Notice that the number of moving steps required to move a task module from u to u' is greater than or equal to the Hamming distance between them, $H(u, u')$. Then, $T(\alpha, \beta) \geq \sum_{u \in \alpha} H(u, u')$. Moreover, from the above reasoning, we obtain

$$\begin{aligned} T(\alpha, \beta) &\geq \sum_{u \in \alpha} H(u, u') \\ &\geq \sum_{u \in \alpha} \{H^*(u, \sigma_{\alpha \to \beta}(\alpha)) \\ &\quad + H^*(\sigma_{\alpha \to \beta}(\alpha), \sigma_{\beta \to \alpha}(\beta)) \\ &\quad + H^*(\sigma_{\beta \to \alpha}(\beta), u')\} \\ &= \sum_{u \in \alpha} H^*(u, \sigma_{\alpha \to \beta}(\alpha)) \\ &\quad + 2^{|\alpha|} H^*(\sigma_{\alpha \to \beta}(\alpha), \sigma_{\beta \to \alpha}(\beta)) \\ &\quad + \sum_{u \in \alpha} H^*(\sigma_{\alpha \to \beta}(\alpha), u'). \end{aligned}$$

Let r_1 be the number of dimensions in which $\{a_i, b_i\} = \{0, 1\}$, r_2 be the number of dimensions in which $a_i = b_i = *$, r_3 be the number of dimensions in which $a_i = *$ and $b_i \in \{0, 1\}$, and r_4 be the number of dimensions in which $b_i = *$ and $a_i \in \{0, 1\}$. Clearly, $r_1 = H^*(\sigma_{\alpha\to\beta}(\alpha), \sigma_{\beta\to\alpha}(\beta))$, $r_2 + r_3 = |\alpha|$, $r_1 + r_3 = M(\alpha, \beta)$, and $r_3 = r_4$, because the addresses of α and β have the same number of *'s. Therefore,

$$\begin{aligned} T(\alpha, \beta) &\geq \sum_{u\in\alpha} H^*(u, \sigma_{\alpha\to\beta}(\alpha)) \\ &\quad + 2^{|\alpha|} H^*(\sigma_{\alpha\to\beta}(\alpha), \sigma_{\beta\to\alpha}(\beta)) \\ &\quad + \sum_{u\in\alpha} H^*(\sigma_{\alpha\to\beta}(\alpha), u') \\ &= 2^{|\alpha|} H^*(\sigma_{\alpha\to\beta}(\alpha), \sigma_{\beta\to\alpha}(\beta)) \\ &\quad + 2\sum_{u\in\alpha} H^*(u, \sigma_{\alpha\to\beta}(\alpha)) \\ &= 2^{|\alpha|} r_1 + 2(2^{r_2} 2^{r_3-1} r_3) \\ &\qquad \text{(From Proposition 1 and } r_2 = |\sigma_{\alpha\to\beta}(\alpha)|\text{.)} \\ &= 2^{|\alpha|}(r_1 + r_3) = M(\alpha, \beta) 2^{|\alpha|}. \end{aligned}$$

Next, we prove the inequality $T(\alpha, \beta) \leq M(\alpha, \beta)2^{|\alpha|}$ by showing the existence of a one-to-one mapping between nodes in α and β, and that the Hamming distance between each pair of mapping and mapped nodes is $M(\alpha, \beta)$. Suppose $p_1, p_2, \cdots, p_{r_3}$ are those dimensions in which $a_{p_i} \in \{0, 1\}$ and $b_{p_i} = *$, and $q_1, q_2, \cdots, q_{r_4}$ are those dimensions in which $a_{q_i} = *$ and $b_{q_i} \in \{0, 1\}$. Note that $r_3 = r_4$. Each node $u = u_n u_{n-1} \cdots u_1 \in \alpha$ can then be mapped to a node $w = w_n w_{n-1} \cdots w_1 \in \beta$ in such a way that when $i \neq p_j$ for any $1 \leq j \leq r_3$,

$$w_i = \begin{cases} b_i, & \text{if } b_i \in \{0, 1\}, \\ u_i, & \text{if } a_i = b_i = *, \end{cases}$$

and when $i = p_j$ for some j, $1 \leq j \leq r_3$,

$$w_{p_j} = \begin{cases} \overline{u_{p_j}}, & \text{if } w_{q_j} = u_{q_j}, \\ u_{p_j}, & \text{if } w_{q_j} \neq u_{q_j}. \end{cases}$$

This is a one-to-one mapping, since the possibility of a many-to-one mapping is eliminated by different assignments of bits in the p_jth dimension, $1 \leq j \leq r_3$. Moreover, we have $H(u, w) = r_1 + r_3 = M(\alpha, \beta)$. By the above node-mapping, we can determine, for each source node in α, the corresponding mapped node in β, and the total number of moving steps is $M(\alpha, \beta)2^{|\alpha|}$, thus satisfying $T(\alpha, \beta) \leq M(\alpha, \beta)2^{|\alpha|}$. Q.E.D.

The above theorem proves that the minimal number of moving steps required to move a task from a subcube location α to another subcube location β as $M(\alpha, \beta)2^{|\alpha|}$. There may be many ways to move a task from one subcube location to another, each having the same total number of moving steps. For example, we can move an active task from 10*1 to 000* by either a) 1011 → 0000 (3 hops) and 1001 → 0001 (1 hop), or b) 1011 → 0001 (2 hops) and 1001 → 0000 (2 hops). The total number of moving steps in either case is 4. However, in order to exploit the inherent parallelism, we naturally want the total $M(\alpha, \beta)2^{|\alpha|}$ moving steps to be equally distributed among all pairs of source and destination nodes such that every node u in α requires exactly $M(\alpha, \beta)$ moving steps to transfer its task module to the corresponding node in β. Clearly, this can be accomplished by the node-mapping scheme introduced in the proof of Theorem 4. Notice that the source and destination subcubes for tasks being migrated under a strategy must be recognizable by that strategy. In light of this fact, a simplified node-mapping scheme will be introduced in Corollary 4.1. However, it is necessary to introduce the following proposition first.

Proposition 2: Suppose $\alpha = a_n a_{n-1} \cdots a_1$ and $\beta = b_n b_{n-1} \cdots b_1$ are two k-dimensional subcubes recognizable by the GC strategy. Then, there is at most one dimension, say p, in which $a_p \in \{0, 1\}$ and $b_p = *$.

Proof: From Theorem 2 and the fact that $g_i = i$, $1 \leq i \leq n$, for the BRGC, we know $a_i = *$ and $b_i = *$ for $1 \leq i \leq k - 1$. Since there are exactly k *'s in the address of a Q_k, this proposition follows. Q.E.D.

The node-mapping between two subcubes recognizable by the GC strategy can be determined as follows. Suppose $\alpha = a_n a_{n-1} \cdots a_1$ is the source subcube and $\beta = b_n b_{n-1} \cdots b_1$ is the destination subcube. Let p and q be the dimensions in which $a_p \in \{0, 1\}$ and $b_p = *$, and $a_q = *$ and $b_q \in \{0, 1\}$.

Corollary 4.1: Each source node $u = u_n u_{n-1} \cdots u_1 \in \alpha$ can be one-to-one mapped to a destination node $w = w_n w_{n-1} \cdots w_1 \in \beta$ in such a way that when $i \neq p$,

$$w_i = \begin{cases} b_i, & \text{if } b_i \in \{0, 1\}, \\ u_i, & \text{if } a_i = b_i = *, \end{cases}$$

when $i = p$,

$$w_p = \begin{cases} \overline{u_p}, & \text{if } w_q = u_q, \\ u_p, & \text{if } w_q \neq u_q, \end{cases}$$

and $H(u, w) = M(\alpha, \beta)$.

For example, when $\alpha = 1*1*$, $\beta = 00**$, and $u = 1110$, we have $p = 3$ and then $w = 0010$. It can be verified that every node in 1*1* will need exactly 2 moving steps to relocate its task module to the corresponding node in 00**, i.e., 1010 (12) → 0000 (0), 1011 (13) → 0001 (1), 1110 (11) → 0010 (3), and 1111 (10) → 0011 (2). It is worth mentioning that the order of source nodes in the BRGC is not necessarily the same as that of their corresponding destination nodes after the node-mapping (see Fig. 7).

C. Determination of Shortest Deadlock-Free Routing

After the determination of the node-mapping, we now want to develop a routing method to move each task module from its source node to its destination node. As mentioned earlier, in order to avoid deadlocks, a linear ordering among hypercube nodes is enforced such that each node can only move its task module to a node with a lower address. More formally, we need the following definition.

Definition 4: A path is said to be *shortest deadlock-free* (SDF) with respect to a coding scheme if it is a shortest path from the source node to the destination node and the reverse

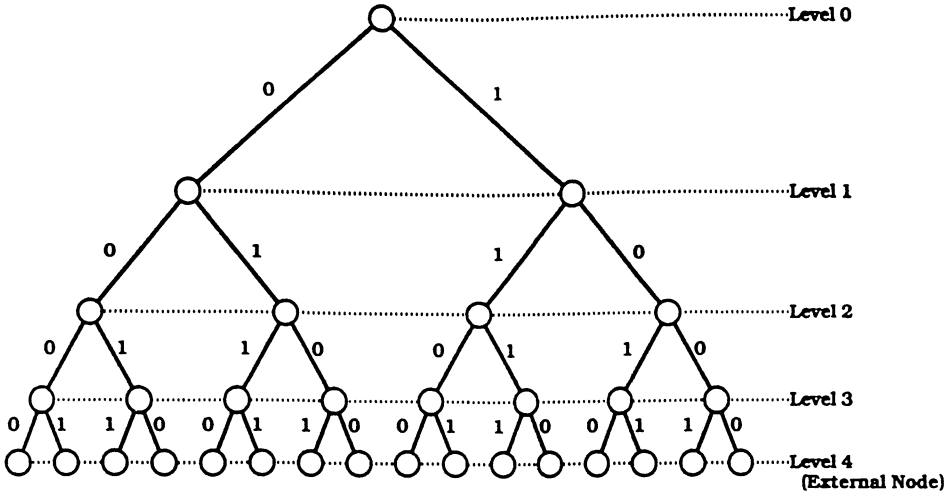

Fig. 8. The labeled complete binary tree for a BRGC.

order of nodes in that coding scheme is preserved in the node sequence of that path.

In other words, if $C_n(i)$ and $C_n(j)$ are two nodes in an SDF path, then $C_n(i)$ is ahead of $C_n(j)$ in the path iff $i > j$. For example, the path [110, 010, 011] is an SDF path in G_3 of Fig. 2(a), whereas [110, 111, 011] is not. A coding scheme C_n is said to be *SDF path preserving* if $\forall p < q$ there exists an SDF path from $C_n(q)$ to $C_n(p)$, i.e., there exists $[C_n(q), C_n(r_1), C_n(r_2), \cdots, C_n(r_{d-1}), C_n(p)]$, such that $q > r_1 > \cdots > r_{d-1} > p$.

Once the node-mapping between each pair of source and destination subcubes is determined, each source node appends to its task module the address of its destination node. Each node can then determine the next hop on which to route a task module by the following algorithm.

Algorithm A_2: Determination of an SDF path:

Step 1. Each node compares the destination address $d = d_n d_{n-1} \cdots d_1$ with its own address $s = s_n s_{n-1} \cdots s_1$ from left to right. Let the jth and kth dimensions be, respectively, the first and second dimensions in which they differ, i.e., $s_i = d_i$ for $j+1 \le i \le n$ and $k+1 \le i \le j-1$, and $s_j \neq d_j, s_k \neq d_k$.

Step 2. If $\sum_{i=k}^{j-1} s_i$ is even **then** send the task module to a neighboring node along the kth dimension **else** send the task module to a neighboring node along the jth dimension.

For example, suppose the source node is $G_4(12) = 1010$ and the destination node d is $G_4(1) = 0001$, then $j = 4$ and $k = 2$. The next node determined by A_2 is $G_4(3) = 0010$ since $\sum_{i=2}^{3} s_i$ is odd, and thus, the fourth dimension of 1010 is changed. Then, the next hop determined by the intermediate node $G_4(3) = 0010$ is $G_4(2) = 0011$, since we get $j = 2$ and $k = 1$ for $s = 0010$ and $d = 0001$. It can be verified that [1010 (12), 0010 (3), 0011 (2), 0001 (1)] is an SDF path. Actually, this is not a coincidence. As it will be proved later, the paths determined by A_2 must be SDF. To facilitate the proof, it is necessary to introduce the following lemma which compares the order of two BRGC numbers.

Lemma 3: Let $G_n(p) = a_n a_{n-1} \cdots a_1$ and $G_n(q) = b_n b_{n-1} \cdots b_1$ be two BRGC numbers. Suppose the ith dimension is the first dimension in which $G_n(p)$ and $G_n(q)$ differ, when they are compared from left to right, i.e., $a_j = b_j$ for $n \ge j > i$ and $a_i \neq b_i$. Without loss of generality, we can assume $a_i = 1$ and $b_i = 0$. Then, $p > q$ iff $\sum_{j=i+1}^{n} a_j$ is even.

Proof: Consider the following procedure to generate the BRGC. Let $G_1 = \{0, 1\}$. Given a k-bit BRGC $G_k = \{d_0, d_1, \cdots, d_{2^k-1}\}$, a $(k+1)$-bit BRGC can be generated by

$$G_{k+1} = \{d_0 0, d_0 1, d_1 1, d_1 0, d_2 0, d_2 1, \cdots, d_{2^k-1} 1, d_{2^k-1} 0\}.$$

It is proved in [12] that this procedure indeed generates the BRGC. This procedure can be described by the complete binary tree in Fig. 8. As the number of bits in the BRGC increases, the corresponding tree grows. The address of every external node (leaf) is determined by the coded bits in the path from the root to the external node, and the BRGC is then obtained by the addresses of external nodes from left to right. It can be verified that every node which is reached from the root via an even number of links labeled with 1 has a 0 left-child and a 1 right-child. Note that an external node further to the right is associated with a larger number in the BRGC. Thus, it is proved that $p > q$ if $\sum_{j=i+1}^{n} a_j$ is even, and the fact that $p < q$ if $\sum_{j=i+1}^{n} a_j$ is odd follows similarly. Q.E.D.

For example, in the 3-bit BRGC of Fig. 2(a), $G_3(3) = 010$ appears after $G_3(1) = 001$, since the number of 1's in the left of their first different bit position in $G_3(3)$ is zero which is even, whereas $G_3(4) = 110$ appears before $G_3(6) = 101$ since the number of 1's in the left of their first different bit position in $G_3(4)$ is one. With the aid of Lemma 3, the following important theorem can be derived.

Theorem 5: The path determined by A_2 is SDF.

Proof: Let $f(m)$ denote the number mapped into the binary string m in the BRGC, i.e., $p = f(m)$ iff $m = G_n(p)$. Since $f(s) > f(d)$, from Lemma 3 we know that $\sum_{i=j}^{n} s_i$ is odd. Let $u = u_n \cdots u_1$ denote the address of the next hop determined by A_2. Consider the case when $\sum_{i=k}^{j-1} s_i$ is even. Clearly, from Step 2 of A_2, $H(u, s) = 1$ and $H(u, d) = H(s, d) - 1$, since $u_i = s_i$ if $i \neq k$, and $u_k = \overline{s_k} = d_k$. Besides, we have $f(u) > f(d)$ since $\sum_{i=j}^{n} u_i = \sum_{i=j}^{n} s_i$ is odd, and $f(s) > f(u)$ since $\sum_{i=j}^{n} s_i + \sum_{i=k}^{j-1} s_i$ is odd.

On the other hand, in the case when $\sum_{i=k}^{j-1} s_i$ is odd, $H(u, s) = 1$ and $H(u, d) = H(s, d) - 1$, since $u_i = s_i$ if $i \neq j$, and $u_j = \overline{s_j} = d_j$. Also, $f(s) > f(u)$ since $\sum_{i=j}^{n} s_i$ is odd, and $f(u) > f(d)$ since $\sum_{i=k}^{n} u_i = \sum_{i=k}^{j-1} u_i + \sum_{i=j}^{n} u_i =$

$\sum_{i=k}^{j-1} s_i + \overline{s_j} + \sum_{i=j+1}^{n} s_i$ is odd. Therefore, in both cases, $H(u, d) = H(s, d) - 1$, $f(s) > f(u)$ and $f(u) > f(d)$. Since the above results hold for every intermediate node, this theorem follows. Q.E.D.

The above theorem shows that task migration under the GC strategy can be accomplished via SDF paths. Also, the following corollary results from Theorem 5.

Corollary 5.1: The BRGC is SDF path preserving.

Note that the BC is not SDF path preserving, neither is the coding scheme given in Fig. 4. For example, no SDF path exists from $B_3(2) = 010$ to $B_3(1) = 001$, and nor does from $C_4(9) = 1000$ to $C_4(1) = 0001$.

Furthermore, as it will be proved below, A_2 will not send any two modules of a task to the same next hop, implying that task migration can be performed in parallel. It was assumed in Section II that a hypercube node can send and receive task modules at the same time and each moving step takes one time unit. Let α be the source subcube and $S_1, S_2, \cdots, S_{2^{|\alpha|}}$ be the nodes of α. Suppose β is the destination subcube and $S_i(t)$ is the hypercube node which receives, via the path determined by A_2, the task module originally residing at S_i after t time units under the node-mapping scheme in Corollary 4.1. Thus, $S_i = S_i(0)$, $1 \le i \le 2^{|\alpha|}$, are the nodes of α, $S_i(M(\alpha, \beta))$, $1 \le i \le 2^{|\alpha|}$, are the nodes of β, and $[S_i(0), S_i(1), \cdots, S_i(M(\alpha, \beta)]$ is the path for moving a task module from S_i to its destination. Two paths $[S_i(0), S_i(1), \cdots, S_i(M(\alpha, \beta)]$ and $[S_j(0), S_j(1), \cdots, S_j(M(\alpha, \beta)]$ are said to be *stepwise disjoint*, if at any time t, the two corresponding task modules will not be sent to the same next hop, i.e., $S_i(t) \neq S_j(t)$, $\forall t \in [0, M(\alpha, \beta)]$. Then, we have the following corollary which states the impossibility for two task modules to compete for the same next hop during task migration, showing another advantage of using the GC strategy.

Corollary 5.2: Under the node-mapping scheme in Corollary 4.1, the paths determined by A_2 are stepwise disjoint.

Proof: Suppose $\alpha = a_n a_{n-1} \cdots a_1$ and $\beta = b_n b_{n-1} \cdots b_1$ are, respectively, the source and destination subcubes, and $|\alpha| = |\beta| = k$. From Proposition 2, we get $a_{k-1} = \cdots = a_1 = b_{k-1} = \cdots = b_1 = *$. From the node-mapping scheme in Corollary 4.1, it follows that for any pair of source node $u = u_n u_{n-1} \cdots u_1$ and destination node $w = w_n w_{n-1} \cdots w_1$, we have $u_{k-1} \cdots u_1 = w_{k-1} \cdots w_1$. This in turn implies that the path from u to w must be within subcube $*^{n-k+1} u_{k-1} \cdots u_1$.

Divide a Q_n into $2^{k-1} Q_{n-k+1}$'s, whose addresses are $*^{n-k+1} d_{k-1} \cdots d_1$, $d_i \in \{0, 1\}$, $1 \le i \le k-1$. Call each of these Q_{n-k+1}'s a *partition*. From Proposition 2, it can be seen that each partition contains exactly two adjacent nodes in α. The entire paths for moving two task modules in a partition to their destination nodes will remain within the same partition. This means that task modules from different partitions will not collide with one another at any time. Moreover, since there is no cycle of an odd length in a Q_n, the two task modules originally residing at two adjacent nodes in the source subcube will not collide with each other at any time. This corollary thus follows. Q.E.D.

To illustrate the entire process of task migration, consider the fragmented configuration in Fig. 7(a). Using Algorithm A_1, we obtain the goal configuration in Fig. 7(b). By the node-mapping scheme developed in Section IV-B, we have 0011 → 0000 for task 1, 0101 → 0011, 0100 → 0010, 1100 → 0110, and 1101 → 0111 for task 2 (*10* → 0*1*), 1010 → 0100 and 1011 → 0101 for task 3 (101* → 010*), and 1001 → 0001 for task 4. (Note that the relative order of source nodes is changed during the node-mapping for task 3.) The SDF routing can then be determined by A_2 as follows:

Task 1: 0011 → 0001 → 0000;

Task 2: 0101 → 0111 → 0011, 0100 → 0110 → 0010, 1100 → 0100 → 0110 and 1101 → 0101 → 0111;

Task 3: 1010 → 1110 → 1100 → 0100 and 1011 → 1111 → 1101 → 0101, and

Task 4: 1001 → 0001.

V. Conclusion

In this paper, we have derived several important results for the subcube allocation and task migration in a hypercube computer system. We first proved that the GC strategy is optimal in the sense of maximizing the subcube recognition ability of a sequential search. Also, the subcube recognition ability of an allocation strategy using multiple EBC's was analyzed and compared to that using multiple EGC's. The minimal number of EBC's required for complete subcube recognition in a Q_n was proved to be $C^n_{\lfloor n/2 \rfloor}$.

Furthermore, we have developed a procedure for task migration under the GC strategy to eliminate the system fragmentation, which consists of three steps. First, a goal configuration without fragmentation is determined in light of the static optimality of the GC strategy. Second, the node-mapping between the source and destination subcubes is determined. Finally, a routing procedure for obtaining shortest deadlock-free paths during task migration is derived. Moreover, the migrating paths determined by A_2 are proved to be stepwise disjoint, thus allowing for the full parallelism in task migration. Our results confirm the inherent superiority of the GC strategy over others.

Note that an active task has to be temporarily stopped when it is to be moved to another subcube location. The task will resume its execution once it reaches its destination subcube. Hence, task migration induces some operational overhead, degrading system performance. To optimize system performance, one has to determine an optimal threshold by striking a compromise between the system's task admissibility and the operational overhead caused by task migration. Derivation of such a threshold and a decision on when to perform task migration can be made by the host computer which keeps track of the status of every hypercube node. However, design of an optimal threshold depends strongly on the computing environment and the system objective under consideration, and may well vary from one hypercube to another.

Appendix A

Subcube Allocation Strategies

The Buddy Strategy

Subcube Allocation Using a BC:

Step 1. Set $k := |I_j|$, where $|I_j|$ is the dimension of a subcube required to accommodate the request I_j.

Step 2. Determine the least integer m such that all the allocation bits in the region $\#[m2^k, (m+1)2^k - 1]$ are 0's, and set all the allocation bits in the region $\#[m2^k, (m+1)2^k - 1]$ to 1's.

Step 3. Allocate nodes with addresses $B_n(i)$ to the request I_j, $\forall i \in \#[m2^k, (m+1)2^k - 1]$.

Deallocation:

Reset every pth allocation bit to 0, where $B_n(p) \in q$ and q is the address of a released subcube.

The GC Strategy

Subcube Allocation Using a BRGC:

Step 1. Set $k := |I_j|$, where $|I_j|$ is the dimension of a subcube required to accommodate the request I_j.

Step 2. Determine the least integer m such that all $(i \bmod 2^n)$th allocation bits are 0's, where $i \in \#[m2^{k-1}, (m+2)2^{k-1} - 1]$. Set all these 2^k allocation bits to 1's.

Step 3. Allocate nodes with addresses $G_n(i \bmod 2^n)$ to I_j, where $i \in \#[m2^{k-1}, (m+2)2^{k-1} - 1]$.

Deallocation:

Reset every pth allocation bit to 0, where $G_n(p) \in q$, and q is the address of a subcube released.

Appendix B

List of Symbols

Q_n	A hypercube of dimension n.
C_n	An n-bit coding scheme.
G_n	An n-bit binary reflected Gray code (BRGC).
B_n	An n-bit binary code which converts a nonnegative integer to its binary representation.
$G_n(m)$	The n-bit BRGC representation of an integer m.
$B_n(m)$	The n-bit binary representation of an integer m.
$\|\alpha\|$	The dimension of a subcube α.
$H(u, w)$	The Hamming distance between two hypercube nodes.
$H^*(\alpha, \beta)$	The Hamming distance between two subcubes.
$M(\alpha, \beta)$	The moving distance between subcubes α and β.

References

[1] T. F. Chan and Y. Saad, "Multigrid algorithms on the hypercube multiprocessor," *IEEE Trans. Comput.*, vol. C-35, no. 11, pp. 969–977, Nov. 1986.

[2] Y. Saad and M. H. Schultz, "Topological properties of hypercubes," *IEEE Trans. Comput.*, vol. C-37, no. 7, pp. 867–872, July 1988.

[3] M.-S. Chen and K. G. Shin, "Processor allocation in an N-cube multiprocessor using Gray codes," *IEEE Trans. Comput.*, vol. C-36, no. 12, pp. 1396–1407, Dec. 1987.

[4] ——, "On relaxed squashed embedding of graphs a into hypercube," *SIAM J. Comput.*, vol. 18, no. 6, pp. 1226–1244, Dec. 1989.

[5] C. L. Seitz, "The Cosmic Cube," *Commun. ACM*, vol. 28, no. 1, pp. 22–23, Jan. 1985.

[6] N. Corp., *NCUBE/ten: An Overview*, Nov. 1985.

[7] S. L. Johnsson, "Communication efficient basic linear algebra computations on hypercube architectures," *J. Parallel Distributed Comput.*, pp. 133–172, 1987.

[8] K. C. Knowlton, "A fast storage allocator," *Commun. ACM*, vol. 8, no. 10, pp. 623–625, Oct. 1965.

[9] F. Harary, *Graph Theory*. Reading, MA: Addison-Wesley, 1969.

[10] C. L. Liu, *Introduction to Combinatorial Mathematics*. New York: McGraw-Hill, 1968.

[11] D. E. Knuth, *The Art of Computer Programming, Vol. 1*. Reading, MA: Addison-Wesley, 1968.

[12] E. M. Reingold, J. Nievergelt, and N. Deo, *Combinatorial Algorithm*. Englewood Cliffs, NJ: Prentice-Hall, 1977.

Ming-Syan Chen (S'87–M'88), for a photograph and biography, see the January 1990 issue of this Transactions, p. 18.

Kang G. Shin (S'75–M'78–SM'83), for a photograph and biography, see the January 1990 issue of this Transactions, p. 18.