TASK MIGRATION IN HYPERCUBE MULTIPROCESSORS

Ming-Syan Chen‡ and Kang G. Shin†

† Real-Time Computing Laboratory
Department of Electrical Engineering and Computer Science
The University of Michigan
Ann Arbor, Michigan 48109-2122

‡ IBM Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, New York 10598

ABSTRACT

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 execute 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. The procedure for task migration closely depends 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.

In this paper, we develop a task migration strategy for the subcube allocation policy based on the binary reflected Gray code. 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 achieve shortest deadlock-free paths for relocating tasks is developed.

1. 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 subcube for execution. Upon completion of a task, the subcube used for the task is released and made available to other tasks. In [3], we proposed a subcube allocation[1] strategy based on the binary reflected Gray code (BRGC), called the *GC strategy*, as opposed to the one based on the binary encoding scheme, called the *buddy strategy* [7]. The former is shown to outperform the latter due mainly to its superiority in recognizing the existence of available subcubes within a hypercube.

Similarly to conventional memory management, allocation and deallocation of subcubes usually results in a fragmented hypercube, where even if a sufficient number of nodes are available, 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 dimensional cube, or Q_2, to be used; thus, if a task requiring a Q_2 arrives, it has to be either queued or rejected.

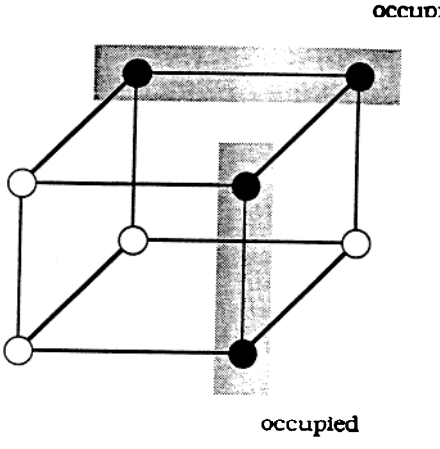

Figure 1. An example of hypercube fragmentation.

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 there is a strong dependence of task migration 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.

We shall in this paper focus on the development of a task migration strategy under the GC strategy of subcube allocation. 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

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

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

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

task is allocated to a subcube, the portion of the task located at each hypercube node of this subcube is called a *task module*. The action for a hypercube 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 moving steps required 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 moving steps required is minimized, but also develop a routing procedure to use shortest deadlock-free paths for task migration.

The paper is organized as follows. Section 2 introduces the necessary definitions and notation first, and then describes the operation of the GC strategy. Our main results on task migration are given in Section 3. In three subsections are respectively presented the three steps for task migration under the GC strategy: (i) determination of a goal configuration, (ii) determination of the node-mapping between the source and destination subcubes, and (iii) determination of shortest deadlock-free routing for moving task modules. Several illustrative examples are also given. The paper concludes with Section 4.

2. PRELIMINARIES

2.1. Notation and Definitions

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 [8]. 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 $\sum$. 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 *coding scheme* with n bits, denoted by C_n, is defined as a one-to-one mapping from an integer number between 0 and 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. For convenience, let B_n and G_n denote n-bit coding schemes associated with the binary encoding scheme and the BRGC, respectively. For example, $B_3(5)=101$, $G_3(5)=111$, while $C_4(7)=1110$ for the coding scheme in Fig. 2.

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 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.

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

Figure 2. A coding scheme with 4 bits.

2.2. Descriptions of the GC Strategy

Note that the GC strategy is based on a first-fit sequential search. Node addresses under the GC strategy are ordered in a list according to the BRGC. Such a list is called an *allocation list*. Under the GC strategy, 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 GC strategy will search for a set of 2^k consecutive unoccupied nodes in its allocation list, say $G_n(p)$, $G_n(p+1)$, $\cdots$, $G_n(p+2^k-1)$, under the constraint that p has to be a multiple of 2^{k-1}. A formal description of the GC strategy and the buddy strategy can be found in [3]. For a comparison purpose, examples for the operation under both strategies are given in Fig. 3.

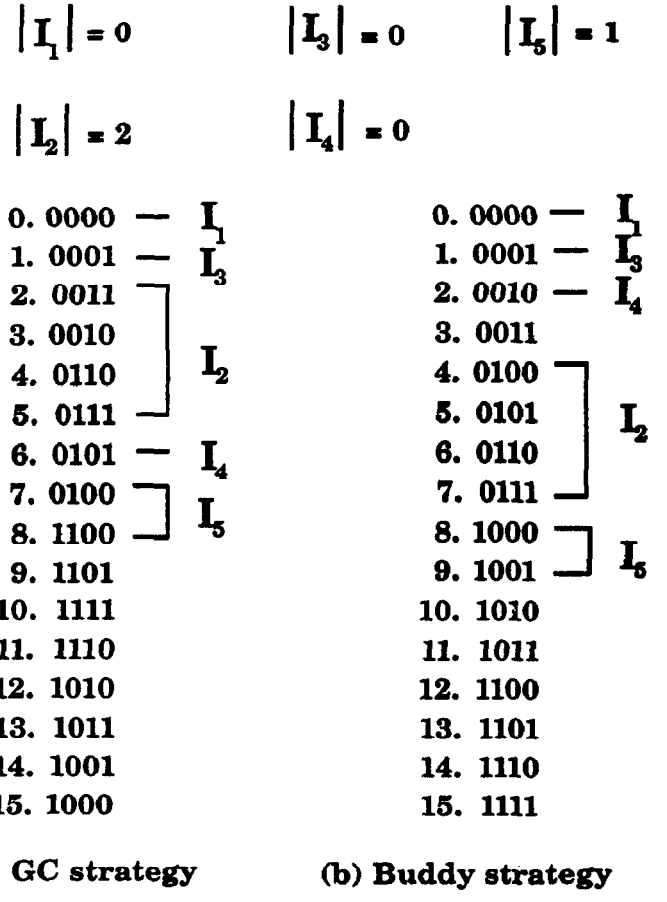

Figure 3. Example operations of both strategies.

3. TASK MIGRATION UNDER THE GC STRATEGY

Due to a similar reason for the memory fragmentation in conventional memory allocation, allocation and deallocation of subcubes may result in a fragmented hypercube. Although the GC strategy outperforms the buddy strategy, the improvement achieved by using multiple codes is rather limited [3]. 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 very important 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.

3.1. Determination of Goal Configuration

There are usually many ways to relocate active tasks and compact occupied subcubes. However, since it is desirable to perform the task migration between each pair of subcubes in parallel, it is very important to avoid any deadlock during the migration. Clearly, a deadlock might occur if there is a circular wait among nodes. 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 $G_n(q)$, if and 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 algorithm below.

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. 4b can be derived from the initial fragmented configuration in Fig. 4a.

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 $\sum_{i=1}^{k} 2^{|I_i|} \le 2^n$, where $|I_i|$ is the subcube dimension required by request I_i. As it was proved in [3], the GC allocation strategy is statically optimal. This fact implies that fragmentation will definitely be removed by A_1.

3.2. 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 1 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

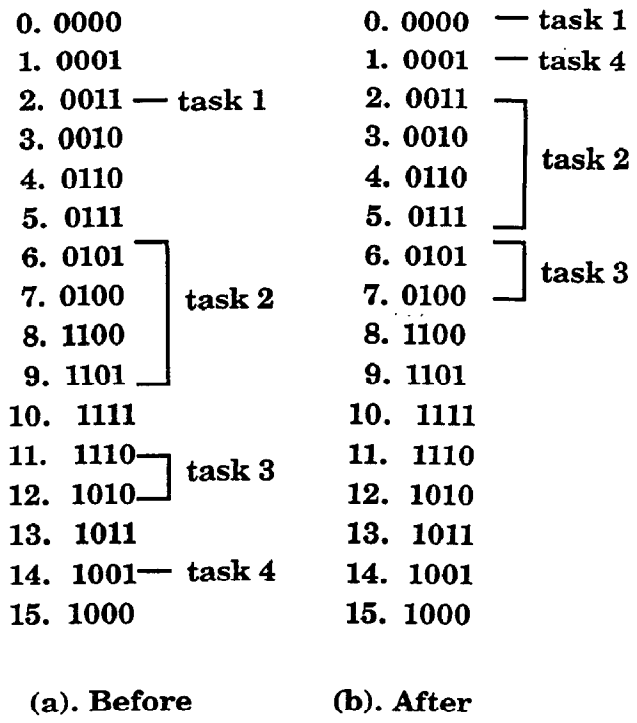

Figure 4. Task migration under the GC Strategy.

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 1.1.

Definition 2 : 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, $M : \sum^n \times \sum^n \to I^+$, 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 1 : 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 1, it is necessary to introduce the following proposition whose proof can be found in [9].

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

Proof of Theorem 1: We shall prove $T(\alpha, \beta) \ge M(\alpha, \beta)2^{|\alpha|}$ 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 shor-

test 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:

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

Let r_1 be the number of dimensions in which $\{a_i, b_i\} = \{0,1\}$, r_2 the number of dimensions in which $a_i = b_i = *$, r_3 the number of dimensions in which $a_i = *$ and $b_i \in \{0,1\}$, and r_4 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,

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

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_j-th 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 is $M(\alpha, \beta) 2^{|\alpha|}$. There may be many ways to move a task from one subcube 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 1. 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 1.1. 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 3 of [3] 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 $\beta = b_n b_{n-1} \cdots b_1$ 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 1.1: 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. 4).

3.3. 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 3: 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 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 a 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 a SDF path in G_3, whereas [110, 111, 011] is not. A coding scheme, C_n, is said to be *SDF path preserving* if $\forall$ p < q there exists a 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 algorithm below.

Algorithm A_2: Determination of a SDF path

Step 1. Each node compares the destination address $d=d_nd_{n-1}\cdots d_1$ with its own address $s=s_ns_{n-1}\cdots s_1$ from left to right. Let the j-th and k-th 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 $s_j \ne d_j$, $s_k \ne d_k$.

Step 2. If $\sum_{i=k}^{j-1} s_i$ is even then send the task module to a neighboring node along the k-th dimension else send the task module to a neighboring node along the j-th 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 4-th 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 a 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 1: Let $G_n(p) = a_na_{n-1}\cdots a_1$ and $G_n(q) = b_nb_{n-1}\cdots b_1$ be two BRGC numbers. Suppose the i-th 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 $a_i \ne b_i$. 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:

$$G_{k+1} = \{d_00, d_01, d_11, d_10, d_20, d_21, \cdots, d_{2^k-1}1, d_{2^k-1}0\}.$$

It is proved in [10] that this procedure indeed generates the BRGC. This procedure can be described by the complete binary tree in Fig. 5. 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, $G_3(3) = 010$ 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 1, the following important theorem can be derived.

Theorem 2: 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 1 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 $i \ne k$, and $u_k = \bar{s}_k = d_k$. 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 $i \ne j$, and $u_j = \bar{s}_j = d_j$. Also, $f(s) > f(u)$ since $\sum_{i=j}^{n} s_i$ is odd, and $f(u) > f(d)$ since $\sum_{i=k}^{n} u_i = \sum_{i=k}^{j-1} u_i + \sum_{i=j}^{n} u_i = \sum_{i=k}^{j-1} s_i + \bar{s}_j + \sum_{i=j+1}^{n} s_i$ 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 2.

Corollary 2.1: The BRGC is SDF path preserving.

Note that the binary encoding scheme is not SDF path preserving, neither is the coding scheme given in Fig. 2. 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 $C_4(1) = 0001$. This fact demonstrate another advantage of the GC strategy.

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 done in parallel. It was assumed in Section 2 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 1.1. 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

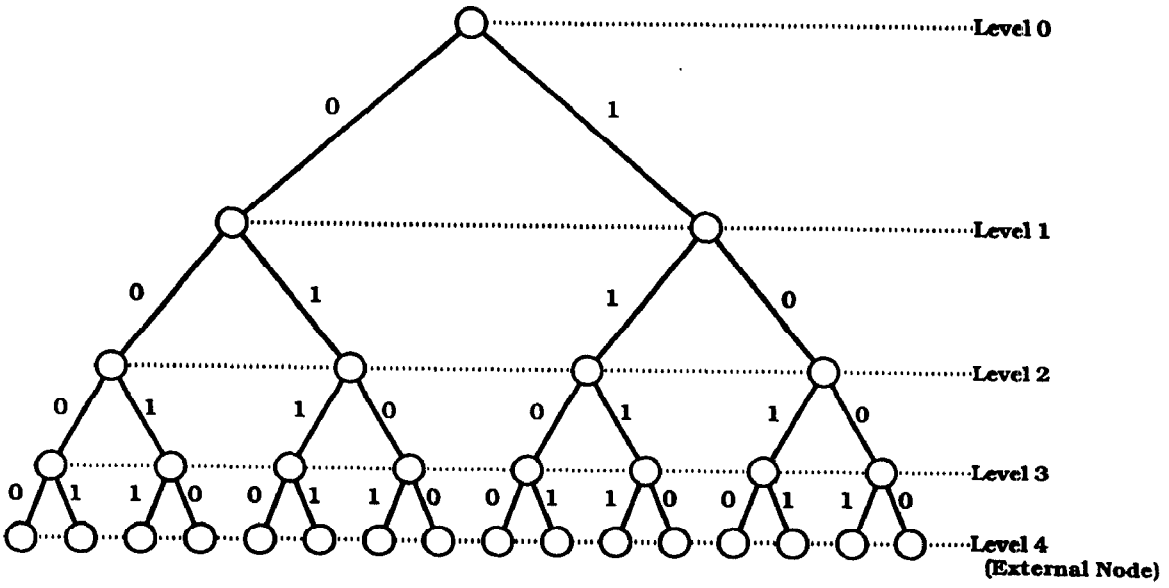

Figure 5. The labeled complete binary tree for a BRGC.

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, another advantage of using the GC strategy.

Corollary 2.2: Under the node-mapping scheme in Corollary 1.1, 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 $a_k=a_{k-1}=\cdots=a_1 = b_k=b_{k-1}=\cdots=b_1 = *$. From the node-mapping scheme in Corollary 1.1, 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, 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. 4a. From A_1, we obtain the goal configuration in Fig 4b. By the node-mapping scheme developed in Section 3.2, 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.

4. CONCLUSION

In this paper, 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 formulated. Finally, a routing procedure for obtaining shortest deadlock-free paths for task migration is derived. Moreover, the migration 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.

ACKNOWLEDGEMENT

The authors would like to thank Andre van Tilborg for his support and encouragement of this work.

This work was supported in part by the Office of Naval Research under contracts N00014-85-K-0122 and N00014-85-K-0531. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the view of the ONR.

REFERENCES

[1] T. F. Chan and Y. Saad, "Multigrid Algorithms on the Hypercube Multiprocessor", *IEEE Trans. on Comput. C-35*, 11 (Nov. 1986), 969-977.

[2] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes", *IEEE Trans. on Comput. C-37*, 7 (July, 1988), 867-872.

[3] M. S. Chen and K. G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes", *IEEE Trans. on Comput. C-36*, 12 (Dec. 1987), 1396-1407.

[4] M. S. Chen and K. G. Shin, "On the Relaxed Squashed Embedding of Graphs a into Hypercube", *SIAM J. Computing*, 1989 (in press).

[5] C. L. Seitz, "The Cosmic Cube", *Commun. of the Assoc. Comp. Mach. 28*, 1 (Jan. 1985), 22-33.

[6] N. Corp., NCUBE/ten: an overview, Nov. 1985.

[7] K. C. Knowlton, "A Fast Storage Allocator", *Commun. of the Assoc. Comp. Mach. 8*, 10 (Oct. 1965), 623-625.

[8] F. Harary, *Graph Theory*, Addison-Wesley, Mass., 1969.

[9] D. E. Knuth, *The Art of Computer Programming, vol. 1*, Addison-Wesley, Mass., 1968.

[10] E. M. Reingold, J. Nievergelt and N. Deo, *Combinatorial Algorithm*, Prentice Hall, Englewood Cliffs, Mass., 1977.