On-Line Task Migration in Hypercubes Through Double Disjoint Paths

Hsing-Lung Chen, *Member*, *IEEE Computer Society*, and Nian-Feng Tzeng, *Senior Member*, *IEEE*

Abstract—Repeated subcube allocation and deallocation in hypercubes tend to cause fragmentation, which can be taken care of by task migration. Earlier task migration dealt with the establishment of a single path from each participating node for transmitting migrated information. The time required for migration with single paths is long, if a large amount of information is moved in hypercubes. This paper considers speedy task migration in that two disjoint paths are created between every pair of corresponding nodes for delivering migrated information simultaneously, reducing the size of data transmitted over a path. All migration paths selected are pairwise disjoint and contain no link of active subcubes, so that task migration can be performed quickly and on-line without interrupting the execution of other jobs. Our approach could lead to a considerable savings in the migration time for contemporary hypercube systems, where circuit switching or wormhole routing is implemented.

Index Terms—Disjoint paths, fragmentation, hypercubes, subcubes, task migration.

1 Introduction

SEVERAL independent jobs can be executed simultaneously on a hypercube system, and each job is assigned one subcube of an appropriate size when it enters the system. Fragmentation exists in a hypercube system when no available subcube is large enough to accommodate an incoming job, even if a sufficient number of free nodes is present but those nodes are scattered around the system. It is observed that fragmentation cannot be eliminated without *task migration*, a process which relocates active tasks by migrating them to respective target subcubes so as to create a large enough free subcube for the incoming job, improving system performance. The task migration mechanism may also be needed for load balancing or for "spawning" one copy of a task itself onto an available subcube in certain "fork" situations [1]. Task migration in hypercubes has been investigated recently [2], [3]. Chen and Shin developed a task migration method under Gray code subcube allocation [2]. Their method ensures that migration paths between a subcube and its target subcube are stepwise disjoint, thus implying that the hypercube employs store-and-forward switching and migration is carried out by all participating nodes in locked steps synchronously. Chen and Lai [3] presented an algorithm for constructing parallel edge-disjoint paths between two subcubes for task migration under circuit switching. Their algorithm finds totally disjoint paths between nodes in a subcube and their corresponding nodes in the target subcube, one for each pair of nodes.

The cost of task migration is high and the job on the subcube to be migrated is suspended before migration takes place. In order to reduce the duration of job suspension, it is absolutely necessary that the time spent in migration be shortened. A hypercube system of the second generation supports either circuit switching (such as Intel's *i*PSC/2 and *i*PSC/860) or wormhole routing (such as NCUBE's *n*-Cube/2) for efficient message transmission. Task migration in a hypercube system with circuit switching or wormhole routing prefers that the migration paths are totally disjoint from a subcube to its target subcube, because under circuit switching, this allows a path to be set up successfully between every pair of corresponding nodes, whereas under wormhole routing, this eliminates message blocking. During migration, the amount of information transferred from each participating node is often quite large, possibly involving hundreds of K-bytes or more. Transmitting such a large amount of information over an established path could be long. For example, it takes roughly 36*ms* to deliver a message of 100K bytes over a path in *i*PSC/2 [4].

The communication latency consists of three components: start-up latency, network latency, and blocking time [5]. It is distance-insensitive under circuit switching and wormhole routing when no conflicts occur [4], [5]. If migration paths are totally disjoint, the communication latency is contributed only by the first two components. Start-up latency is the time required to handle the message at both the source and destination nodes. Network latency equals the elapsed time after the head of a message has entered the network at the source until the tail of the message emerges from the network at the destination [5]. Network latency is proportional to the message size, and it tends to dominate for large message transfers. To transmit a message of 100K bytes in *i*PSC/2, for example, the network latency accounts for 99% of total communication latency, since start-up latency is about 0.3*ms* only.

In this work, we investigate speedy task migration in hypercubes supporting circuit switching or wormhole routing, such that two disjoint paths exist between every pair of corresponding nodes for delivering migration messages concurrently, reducing the size of data transmitted over a path. None of the disjoint paths identified involves any link of active subcubes other than the one to be migrated. As a result, migration takes place in an on-line manner where all active jobs proceed without interruption, as links of their occupied subcubes are not utilized for any migration paths. To implement our proposed task migration, every node is assumed to have a dedicated router permitting *all-port* communication [6], so that multiple messages can be transmitted simultaneously from a node over different paths. It should be noted that job scheduling has been shown to be more effective than processor allocation in lowering the mean response time and improving processor utilization, thus reducing the possibility of fragmentation [7]. When fragmentation arises, however, task migration is the way to eliminate it and our approach could give rise to a much shorter migration interval than that of an earlier single-path migration technique introduced in [3].

2 Node Model and Related Work

Let H_n denote an n-dimensional hypercube, which consists of 2^n nodes, each labeled by an n-bit string, $b_{n-1}b_{n-2} \ldots b_1b_0$, where bit b_i corresponds to dimension i. A link joins two nodes whose labels differ in exactly one bit position. All links are bidirectional and full duplex, i.e., two messages can be transmitted simultaneously in the opposite directions of any link. A link $\lambda(x, y)$ is said to be along dimension i if x and y differ in that dimension. Every path between an arbitrary pair of nodes can be specified uniquely by an ordered sequence of links, and the length of a path is the number of its constituent links. A d-dimensional subcube in H_n is represented by a string of n symbols over set {0, 1, *}, where * is a *don't care* symbol, such that there are exactly d *'s in the string.

• *H.-L. Chen is with the Department of Electronic Engineering, National Taiwan Institute of Technology, Taipei, Taiwan, Republic of China.*
• *N.-F. Tzeng is with the Center for Advanced Computer Studies, University of Southwestern Louisiana, Lafayette, LA 70504. E-mail: tzeng@cacs.usl.edu.*

Manuscript received June 15, 1995; revised Apr. 18, 1996.
For information on obtaining reprints of this article, please send e-mail to: transcom.org, and reference IEEECS Log Number C96101

2.1 Node Model

Each node in H_n has a dedicated router so that computation and communication can be overlapped at each node. The router supports either circuit switching or wormhole routing, and it has both external channels for communication among nodes and internal channels for connection with the local processor/memory [5], [6]. Every external channel is assumed to have a corresponding internal channel so that the node can send and receive on all its ports simultaneously, allowing *all-port* communication. All-port communication is realized in the *n*-Cube/2 hypercube and may become popular for future systems. With this kind of routers, each node in H_n can handle up to *n* simultaneous paths, one along a pair of external input/output channels.

The paths to be set up between two nodes in our migration are determined explicitly by the source node. This source routing requires every message to carry routing information of the entire path, because different paths from one node to another node are established using different routing data. The routing logics are not identical for source routing and for distributed routing, so if the system supports only distributed routing, additional hardware may be needed to realize our migration. When two disjoint paths exist between any given source-destination pair, each of them conveys one half of total migrated information, effectively reducing the network latency by 50%. An analysis of migration time using double disjoint paths is carried out in Section 5.

2.2 Related Work

An algorithm that constructs 2^d disjoint paths for migration between two *d*-dimensional subcubes has been introduced in [3]. It starts with finding a proper bijection function from a subcube to its target subcube, then determines a set of shortest paths between a node and its corresponding node in the target subcube according to a simple rule, and finally selects one path (from the set) that involves no *block link,* where a block link refers to a link inside a *block,* i.e., an active subcube other than the one to be migrated.

Let N denote the set $\{n-1, ..., 1, 0\}$. For any subcube S and integer $i \in N$, $S[i]$ indicates the *i*th bit of the address of S. Bit $\overline{b}$ represents the complement of b and is defined only for $b = 0$ or 1. Suppose subcubes S and T are the source and target subcubes, respectively. Let $D(S) = \{i \in N : S[i] = *\}$, $D(T) = \{i \in N : T[i] = *\}$, $I(S, T) = \{i \in N : S[i] = T[i]\}$, and $C(S,T) = \{i \in N : S\{i\} = \overline{T[i]}\}$, which mean that S and T have don't care bits at $D(S)$ and $D(T)$, respectively; S and T are bitwise identical at $I(S, T)$, and they are complemental of each other at $C(S, T)$. The four sets $I(S, T)$, $C(S, T)$, $D(S) - D(T)$, and $D(T) - D(S)$ are pairwise disjoint, and their union is N. $|D(S) - D(T)|$ is always equal to $|D(T) - D(S)|$, which denotes the number of elements in set $D(T) - D(S)$.

An *isomorphism* from S to T is a bijection from nodes of S to nodes of T such that two nodes in S are adjacent if and only if their images in T are adjacent. Chen and Lai defined an isomorphism f from S to T [3] such that for any node u in S, the address of its image $f(u)$ in T is determined as follows:

$$f(u)[i] = \begin{cases} u[i] & \text{for } \textit{all } i \in I(S,T) \\ T[i] & \text{for } \textit{all } i \in C(S,T) \cup (D(S) - D(T)) \\ u[i] & \text{for } \textit{all } i \in D(T) - D(S) \text{ and } u\left[\alpha^{-1}(i)\right] \neq T\left[\alpha^{-1}(i)\right] \\ \overline{u[i]} & \text{for } \textit{all } i \in D(T) - D(S) \text{ and } u\left[\alpha^{-1}(i)\right] = T\left[\alpha^{-1}(i)\right] \end{cases}$$

where α is any arbitrary but fixed bijective function from $D(S) - D(T)$ to $D(T)–D(S)$ and α^{-1} is its inverse. This isomorphism yields that the Hamming distance between any node u in S and its image $f(u)$ in T is exactly $|C(S, T)| + |D(S) - D(T)|$. It is clear from this isomorphism that $C(u, f(u))$ is the union of three disjoint subsets: $D_f(S \mid u) \cup D_f(T \mid u) \cup C(S, T)$, where $D_f(S \mid u) = \{i \in D(S) : u[i] \neq f(u)[i]\}$ and $D_f(T \mid u) = \{i \in D(T) : u[i] \neq f(u)[i]\}$.

DEFINITION 1. *A Hamming path between $u \in S$ and $f(u) \in T$ is* regular, *if, on the path from u to $f(u)$, all links along the dimensions in $D_f(T \mid u) \cup C(S, T)$ precede all links along the dimensions in $D_f(S \mid u)$.*

It has been proved in [3] that *any two regular paths with different source nodes are totally disjoint*. An algorithm was given to choose one regular path from each node u in S to its image $f(u)$ in T as the migration path, which avoids all block links. Since all the migration paths are regular paths with different source nodes, they are pairwise totally disjoint.

3 PERTINENT ISOMORPHISMS AND PATH PROPERTIES

Consider task migration from a subcube (say S) to its target subcube (say T). Here, the target subcube is assumed to be predetermined, depending on the processor allocation strategy used, the locations of busy subcubes, the size of the requested subcube, etc., and it is known to our migration procedure. In order to identify two sets of pairwise disjoint paths from S to T such that no path involves block links, proper isomorphisms have to be devised first.

3.1 Pertinent Isomorphisms

In addition to isomorphism f described in Section 2.2, two new isomorphisms are needed for our speedy migration: One is from S to T, called isomorphism g, and the other is from T to T, called isomorphism h. Isomorphism g is defined such that for a node u in S, its image $g(u)$ in T satisfies

$$g(u)[i] = \begin{cases} u[i] & \text{for } \textit{all } i \in I(S,T) \\ T[i] & \text{for } \textit{all } i \in C(S,T) \cup (D(S) - D(T)) \\ u[i] & \text{for } \textit{all } i \in D(T) - D(S) \text{ and } u\left[\alpha^{-1}(i)\right] = T\left[\alpha^{-1}(i)\right] \\ \overline{u[i]} & \text{for } \textit{all } i \in D(T) - D(S) \text{ and } u\left[\alpha^{-1}(i)\right] \neq T\left[\alpha^{-1}(i)\right] \end{cases} \quad (1)$$

where α is any arbitrary but fixed bijective function from $D(S) - D(T)$ to $D(T) - D(S)$ and α^{-1} is its inverse, as defined earlier. It is readily observed that g is indeed an isomorphism from S to T such that for all $i \in D(T) - D(S)$, $g(u)[i] \neq u[i]$ if and only if $g(u)[\alpha^{-1}(i)] \neq u[\alpha^{-1}(i)]$. In addition, $C(u, g(u))$, which equals $\{i \in N : u[i] = \overline{g(u)[i]}\}$, is the union of three disjoint subsets, i.e.,

$$C(u, g(u)) = C(S,T) \cup D_g(S \mid u) \cup D_g(T \mid u)$$

where $D_g(S \mid u) = \{i \in D(S) : u[i] \neq g(u)[i]\}$ and $D_g(T \mid u) = \{i \in D(T) : u[i] \neq g(u)[i]\}$. From the second situation of (1), we have, for any node u in S, $k \in D_g(S \mid u)$ if and only if $\alpha(k) \in D_g(T \mid u)$, since $D_g(S \mid u) \subseteq D(S) - D(T)$, which is mapped to $D(T) - D(S)$ by bijective function α.

DEFINITION 2. *A Hamming path between $u \in S$ and $g(u) \in T$ is* aggressive, *if on the path from u to $g(u)$, all links along the dimensions in $D_g(T \mid u)$ precede all links along the dimensions in $C(S, T) \cup D_g(S \mid u)$.*

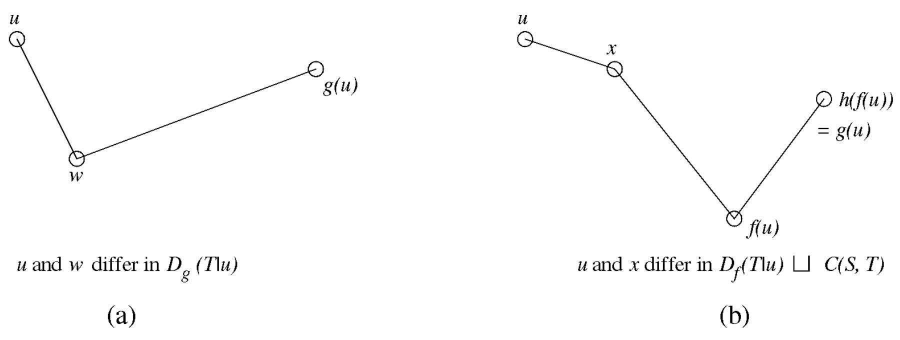
Isomorphism h is defined such that the image of a node v in T, $h(v)$, is in T and satisfies

$$h(v)[i] = \begin{cases} \overline{v[i]} & \text{for } \textit{all } i \in D(T) - D(S) \\ v[i] & \text{for } \textit{the remaining } i. \end{cases} \quad (2)$$

It is obvious that for any node u in S, $g(u) = h(f(u))$.

3.2 Properties of Paths

Essential properties of aggressive paths and regular paths (specified in Definition 1) are characterized below. These properties serve as the basis of identifying two disjoint paths between every pair of



u
w
g(u)
u and w differ in $D_g(T|u)$
(a)
u
x
h(f(u)) = g(u)
f(u)
u and x differ in $D_f(T|u) \sqcup C(S, T)$
(b)

Fig. 1. Constructing two disjoint paths from *u* to *g*(*u*).

corresponding nodes. Their proofs are omitted here and can be found in [8].

LEMMA 1. *Let u and v be any two nodes in S. An aggressive path from u to g(u) and any aggressive path from v to g(v) are totally disjoint.*

LEMMA 2. *Let u and v be any two nodes in S and $D_f(S\,|\,u) \neq \emptyset$. A regular path from u to f(u) and an aggressive path from v to g(v) are totally disjoint.*

LEMMA 3. *Let u and v be any two nodes in T. Any pair of paths which traverse links along the dimensions in D(T) – D(S) following the same sequence, one from u to h(u) and the other from v to h(v), are totally disjoint.*

LEMMA 4. *Let u be any node in S and $D_f(S\,|\,u) \neq \emptyset$. No regular path from u to f(u) traverses any link in T.*

LEMMA 5. *Let u be any node in S. No aggressive path from u to g(u) traverses any link in T.*

LEMMA 6. *Let u and v be any two nodes in S and $D_f(S\,|\,u) \neq \emptyset$. If the addresses of nodes u and u′ differ in exactly D(T) – D(S), then a Hamming path from u to u′ and any aggressive path from v to g(v) are totally disjoint.*

4 SPEEDY TASK MIGRATION

With the properties of paths under isomorphisms *f*, *g*, and *h* characterized, we now discuss how to establish two totally disjoint paths from each node in S to its corresponding node in target subcube *T*, such that the paths involve no block links. Unlike the paths considered in [3], where all the paths created between *S* and *T* are shortest, our paths established are not necessarily shortest and are likely of unequal length. This poses no problem in the communication latency under circuit switching or wormhole routing. In fact, it has been found that the communication latency is nearly independent of the path length under circuit switching or wormhole routing, even for a message of just 1K bytes [5]. For a large message (as in the case of task migration), this would be more the case because the start-up latency then accounts for an even smaller fraction of the communication latency.

A path from node *u* to node *u*′ is represented by Λ(*u*, *u*′). An algorithm for finding a Hamming path between two nodes so as to avoid all block links has been described in [3]. This algorithm, called HAMMING-PATH, takes two nodes and a set of blocks (Ξ) in H_n as inputs, and determines a Hamming path between the two nodes, with the time complexity of $O(n^2|\Xi|)$. The algorithm is employed in our procedure of establishing two disjoint paths between every pair of corresponding nodes. In the following, our procedure is outlined for the situation of $|C(S, T)| = 0$ or $|C(S, T)| \geq 2$ first, because it is slightly different for the situation of $|C(S, T)| = 1$.

4.1 Situation $|C(S, T)| = 0$ or $|C(S, T)| \geq 2$

Under isomorphism *f*, a node *u* in *S* satisfies either $D_f(S\,|\,u) \neq \emptyset$ or $D_f(S\,|\,u) = \emptyset$. A node $u \in S$ with $D_f(S\,|\,u) \neq \emptyset$ and a node $v \in S$ with $D_f(S\,|\,v) = \emptyset$ follow different steps to construct two disjoint paths, as discussed separately below.

Case a): $D_f(S\,|\,u) \neq \emptyset$

In this case, let *w* be the node which differs from *u* in exactly $D_g(T\,|\,u)$. A Hamming path from *u* to *w* and another from *g*(*u*) to *w* are determined by HAMMING-PATH, so that they avoid all blocks. Concatenating these two paths yields an aggressive path from *u* to *g*(*u*), because on the resultant path, all links along the dimensions in $D_g(T\,|\,u)$ (which refer to those on path Λ(*u*, *w*)) precede all links along the remaining dimensions, as depicted in Fig. 1a. The resultant path apparently contains no block link. Next, a Hamming path from *u* to *x* and another from *f*(*u*) to *x* are created by HAMMING-PATH, where *x* is the node which differs from *u* in exactly $D_f(T\,|\,u) \cup C(S, T)$. Concatenating the two paths results in a regular path from *u* to *f*(*u*), based on Definition 1. According to Lemma 2, the constructed aggressive path and regular path are totally disjoint.

Since two disjoint paths between any pair of corresponding nodes, say nodes *u* and *g*(*u*), are to be determined, the path from *u* to *f*(*u*) has to be extended from *f*(*u*) to *g*(*u*), giving rise to another disjoint path from *u* to *g*(*u*). This is done by making use of isomorphism *h*. For a node *z* in *T*, let Λ(*z*, *h*(*z*)) be a Hamming path (which is surely within target subcube *T*) following an arbitrary, but fixed dimension sequence. Now, every node *z* in *T* follows the same fixed dimension sequence to create a path Λ(*z*, *h*(*z*)), so all these created paths are totally disjoint, based on Lemma 3. Concatenating regular path $\Lambda^r(u, f(u))$ and Λ(*f*(*u*), *h*(*f*(*u*))) leads to a path Λ(*u*, *h*(*f*(*u*))) from *u* and *g*(*u*), as shown in Fig. 1b. The resulting path is totally disjoint with the aggressive path $\Lambda^a(u, g(u))$, according to Lemmas 4 and 5, and it obviously contains no block link.

Case b): $D_f(S\,|\,v) = \emptyset$

From the first two situations of isomorphisms *f* and *g*, we have $D_f(S\,|\,v) = D_g(S\,|\,v)$ because under these situations, *f*(*v*)[*i*] and *g*(*v*)[*i*] are assigned identical values. Based on the last three situations of (1), $|D_g(S\,|\,v)|$ is found equal to $|D_g(T\,|\,v)|$. Since $D_f(S\,|\,v) = \emptyset$, $|D_g(S\,|\,v)| = 0 = |D_g(T\,|\,v)|$, suggesting $C(v, g(v)) = C(S, T)$ according to the expression for $C(v, g(v))$. If $|C(S, T)| = 0$, then $v = g(v)$ and there is no migration path needed for *v*. We therefore consider only $|C(S, T)| \geq 2$ subsequently.

Let $k \in C(S, T)$, and *w* be the node connected to *g*(*v*) by link *k*. A path from *v* to *w* is determined by HAMMING-PATH, and the path is concatenated with λ(*w*, *g*(*v*)) to yield a path from *v* to *g*(*v*), which is an aggressive path, according to Definition 2, because $D_g(T\,|\,v) = \emptyset$. This aggressive path evidently contains no block link. Similarly, a path from *g*(*v*) to *x* is constructed by HAMMING-PATH, where *x* is the node that is connected to *v* by link *k*. Concatenating λ(*v*, *x*) and the constructed path Λ(*g*(*v*), *x*) yields an aggressive path from *v* to *g*(*v*), and this aggressive path clearly

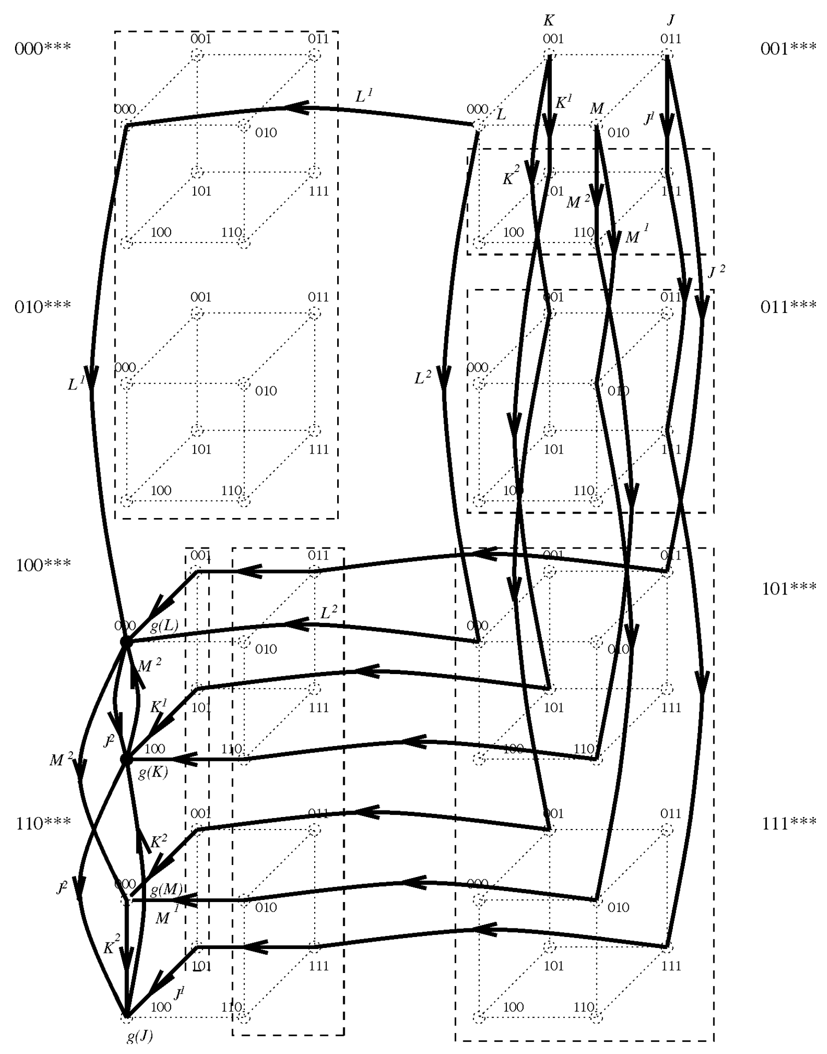


Fig. 2. Migration paths from S = 0010** to T = 1*0*00. (Some links in H_6 are omitted for clarity. The two paths from node J to its corresponding node $g(J)$ are denoted by J^1 and J^2.)

involves no block link. The two aggressive paths are not the same (for $|C(S, T)| \geq 2$), and they are totally disjoint because they both are Hamming paths with one traversing link k last whereas the other traversing link k first. Further, all aggressive paths so constructed are pairwise totally disjoint, based on Lemmas 1, 2, 3, 4, and 5. We thus arrive at the next theorem.

THEOREM 1. *Under $|C(S, T)| = 0$ or $|C(S, T)| \geq 2$, two totally disjoint paths exist between every node in S and its corresponding node in T, and they contain no block link. In addition, all the paths between S and T are pairwise disjoint.*

Under this situation, every node in S determines its two disjoint migration paths independently and concurrently. A node in Case a) calls HAMMING-PATH four times and makes use of isomorphism h once to decide its two paths, giving rise to the same time complexity as that of HAMMING-PATH, i.e., $O(n^2|\Xi|)$. A node in Case b) requires lower time complexity, as it calls HAMMING-PATH only twice to determine its two paths.

An example of this situation is demonstrated in Fig. 2, where S = 0010**, T = 1*0*00, and all the remaining nodes in H_6 are block nodes as enclosed by dashed lines. In this example, $C(S, T) = \{5, 3\}$, $D(S) - D(T) = \{1, 0\}$, and $D(T) - D(S) = \{4, 2\}$. Let mapping α be defined as: $\alpha(0) = 2$ and $\alpha(1) = 4$, and the dimension sequence for specifying $\Lambda(z, h(z))$ from every node z in T be 2, 4 (i.e., it takes dimension 2 first, followed by dimension 4). Node u = 001011 belongs to Case a), as $f(u)$ = 100000 and $D_f(S\,|\,u) = \{1, 0\} \neq \varnothing$. For this node, $g(u)$ = 110100, $D_f(T\,|\,u) = \varnothing$, $D_g(S\,|\,u) = \{1, 0\}$, and $D_g(T\,|\,u) = \{4, 2\}$. Node w = 011111 differs from u in exactly $D_g(T\,|\,u)$, and the two paths derived by HAMMING-PATH are 001011 $\rightarrow$ 001111 $\rightarrow$ 011111 and 110100 $\rightarrow$ 110101 $\rightarrow$ 110111 $\rightarrow$ 111111 $\rightarrow$ 011111. Concatenating the two paths yields an aggressive path between u and $g(u)$. Next, node x = 100011 differs from u in exactly $D_f(T\,|\,u) \cup C(S, T)$. The two paths $\Lambda(u, x)$ and $\Lambda(f(u), x)$ obtained by HAMMING-PATH are concatenated, together with path $\Lambda(f(u), h(f(u)))$ to get another disjoint path 001011 $\rightarrow$ 101011 $\rightarrow$ 100011 $\rightarrow$ 100001 $\rightarrow$ 100000 $\rightarrow$ 100100 $\rightarrow$ 110100. The two migration paths between u and $g(u)$ are indicated by J^1 and J^2 in Fig. 2.

On the other hand, node v = 001000 belongs to Case b), as $f(v)$ = 110100 and $D_f(S\,|\,v) = \varnothing$. This node exhibits $g(v)$ = 100000, $D_f(T\,|\,v) = \{4, 2\}$, $D_g(S\,|\,v) = \varnothing$, and $D_g(T\,|\,v) = \varnothing$. The two aggressive paths between v and $g(v)$ are 001000 $\rightarrow$ 000000 $\rightarrow$ 100000 and 001000 $\rightarrow$ 101000 $\rightarrow$ 100000, as marked by L^1 and L^2 in Fig. 2. All the migration paths between S and T are given in the figure, and they indeed are totally disjoint and contain no block link.

4.2 Situation $|C(S, T)| = 1$

For any node u in S, either $D_f(S\,|\,u) \neq \varnothing$ or $D_f(S\,|\,u) = \varnothing$ holds. A node u satisfying $D_f(S\,|\,u) \neq \varnothing$ follows the same way as above for Case a) to establish an aggressive path between u and $g(u)$. For a node v with $D_f(S\,|\,v) = \varnothing$, according to the argument given in Case b), we have $C(v, g(v)) = C(S, T)$, which means that v is adjacent to its corresponding node $g(v)$ in T, as $|C(S, T)| = 1$. A single-link path thus exists between v and $g(v)$. Next, a second migration path from every node in S has to be determined. A node u with $D_f(S|u) \neq \emptyset$ can decide a regular path from u to $f(u)$ based on the steps described in prior Case a). Unlike Case a) where an arbitrary dimension sequence is proper for creating path $\Lambda(f(u), h(f(u)))$, however, this situation requires that the dimension sequence be decided by a specific node before the regular path can be extended from $f(u)$ to $g(u)$ as stated earlier to generate the second disjoint path. Specific nodes are those nodes q satisfying $D_f(S|q) = \emptyset$. The following details how such a node q decides the dimension sequence. Let $\gamma = |D(S) \cap D(T)|$ and $\beta = |D(S) - D(T)| = |D(T) - D(S)|$. S can be partitioned into 2^γ β-dimensional subcubes along those dimensions in $D(S) \cap D(T)$. Consider any β-dimensional subcube so partitioned, say B. Obviously, there is exactly one node, say q, in B such that $D_f(S|q) = \emptyset$, i.e., $D_f(T|q) = D(T) - D(S)$. Node q is a specific node, which decides the dimension sequence for all the nodes inside B. There are 2^γ specific nodes totally in S. Let $C(S, T) = \{k\}$, then node q and $g(q)$ is connected by link k. If a second path exists from q to $g(q)$, it must start with a link other than k.

Consider a node in B, say node $(p \neq q)$, and the aggressive path from p (recall that each node in S determined an aggressive path individually). If node w is the last intermediate node on the aggressive path between p and $g(p)$ and if the link between w and $g(p)$ is *not* k, node q can have a second migration path formed by concatenating the subpath (determined by HAMMING-PATH) which traverses all dimensions in $D_g(T|p)$, link k, and the subpath which traverses (in any sequence) all dimensions in $D_g(T|p)$. The dimension sequence used by the last subpath has to be broadcast (by q) to all the other $2^\beta - 1$ nodes in B, so that all second migration paths from B would follow the identical (sub)sequence, ensuring that they are pairwise disjoint. The reason for the second migration path $\Lambda(q, g(q))$ so constructed to be disjoint with others lies in that its first constituent subpath is a subpath of a regular path, which is disjoint with other regular paths, and which is disjoint with any aggressive path according to Lemma 6.

On receiving this broadcast dimension sequence, every node u in B can decide its path $\Lambda(f(u), h(f(u)))$, thereby the second migration path. In order to let node q know whether or not its second migration path exists, every node (other than q) in B sends a message to q indicating if its aggressive path constructed traverses link k last. This is done in a distributed manner, and node q in the worst scenario checks all the $(2^\beta - 1)$ messages received, taking time $O(2^\beta)$. Compared with the previous situation, the situation of $|C(S, T)| = 1$ requires additional time complexity of receiving $(2^\beta - 1)$ messages and examining them at each specific node. If $\beta = 0$ or every aggressive path from B traverses link k last, the second migration path between q and $(g(q))$ is absent. The next theorem follows immediately.

THEOREM 2. *Under $|C(S, T)| = 1$, two totally disjoint paths exist between every node in S and its corresponding node in T, and they contain no block link, provided that not every aggressive path from B traverses link k last, for all B's. In addition, all the paths between S and T are pairwise disjoint.*

5 Migration Time Analysis

The speedup resulting from the use of double disjoint paths for task migration in hypercubes with circuit switching or wormhole routing is analyzed. Our time analysis is based on the following assumptions about the communication model:

1) every cube link is full duplex, i.e., two messages can simultaneously travel along the link in opposite directions;
2) a router can send and receive messages on all its ports concurrently, known as the all-port communication mode [6]; and
3) there is no blocking time or queuing delay for each path, since all the migration paths are pairwise disjoint.

Let M, B, and P be the message length, the channel bandwidth, and the path length between the source (S) and the target (T), respectively. Assume that $|C(S, T)| = \zeta$ and $|D(S)–D(T)| = \beta$.

5.1 Under Circuit Switching

When each source node sends a message through only one path (i.e., a regular path) to its corresponding destination node, the network latency is expressed by $(M_c/B)P + M/B$, where M_c is the length of the control message transmitted to establish the circuit and $P = \zeta + \beta$. During task migration, $M_c << M$ and hence P has a negligible effect on the network latency. As a result, the network latency can be approximated by M/B using only one path. On the other hand, when two disjoint paths are used between every corresponding node pair for migration and each path is assumed to transmit one half of the message, the network latency becomes $(M_c/B)P + M/(2B)$, where P equals $\zeta + 2\beta$, which is the length of the longest paths between S and T. If $(M_c/B)P$ is ignored, the network latency is approximately $M/(2B)$. Since the communication latency is equal to the start-up latency plus the network latency and the start-up latency is in the order of hundreds of μsec, much smaller than the network latency for a long message (for example, it takes about 36 *ms* to transmit a message of 100K bytes over a path in *i*PSC/2 [4], while its start-up latency is about 300 μsec), our speedy task migration yields a reduction in the communication latency by almost 50%. Under this switching technique, time overhead is involved to split a large message into two smaller ones (and possibly assemble them at the receiving node), but it is not expected to be high. The time of migration through double disjoint paths will thus be considerably shorter.

5.2 Under Wormhole Routing

If a message is transmitted over one path to its destination, the network latency is given by $(L_f/B)P + M/B$, where L_f is the length of each flit and P equals $\zeta + \beta$. For $L_f << M$, a typical situation for task migration, the path length P will not affect the network latency noticeably, provided that there is no link contention. When two disjoint paths are used for migration between each node pair, the network latency is given by $(L_f/B)P + M/(2B)$, where P equals $\zeta + 2\beta$. Again, the term of $(L_f/B)P$ and the start-up latency can be ignored for large message transmission (for example, the transmission time of 100K bytes over a path in *n*-Cube/2 is about 50 *ms* and its start-up latency is about 150 μsec). Without involving extra time overhead for splitting/assembling messages under this routing, the proposed speedy task migration hence leads to a speedup factor of roughly 2.

6 Concluding Remarks

Task migration by employing two disjoint paths between every pair of corresponding nodes has been introduced. This approach aims at reducing the migration time in hypercubes supporting circuit switching or wormhole routing. As the amount of information to be transferred is often huge and the migration duration for such a hypercube is dictated chiefly by network latency, which is proportional to the message size, utilizing double migration paths can reduce communication latency by almost 50% if the router permits all-port communication because each path then delivers only one half of total migrated information. This is made possible by taking advantage of the fact that under circuit switching and wormhole routing, communication latency is insensitive to path length, so migration paths identified can be of unequal length while maintaining same efficiency.

Our procedure for determining migration paths can be carried out at each participating node in a distributed manner, taking an identical order of time complexity as that required by the earlier single-path migration technique [3], except for one situation where message exchanges are necessary. All migration paths selected are pairwise disjoint and contain no block link. This procedure is applicable to any fragmented hypercube system, and it identifies two migration paths from each participating node, if they exist, or otherwise, tells the absence of the second migration path.

We have implemented this proposed on-line migration approach on the Intel *iPSC*/860 machine with the feature of source routing equipped to allow for specifying the routing path of a message at the originating node. Our implementation results confirm the advantage of proposed speedy task migration, even with the overhead (of reassembling the two halves of the message at the destination) taken into consideration. If the migration file is of 4M bytes, for example, our speedy migration gives rise to an improvement of roughly 35% in the migration time when compared with migration using one single path.

It is worth mentioning that speedy task migration could become more attractive for the future system where the start-up latency is lowered drastically; for instance, the recently announced *n*-Cube/3 is claimed to have a start-up latency of only 5 μsec, in sharp contrast to 150 μsec for the *n*-Cube/2. As a result, the use of double disjoint paths for transmitting even a moderate amount of information in such hypercubes would be advantageous.

ACKNOWLEDGMENTS

Hsing-Lung Chen's research was supported by the National Science Council of the Republic of China under Contract NSC83-0408-E011-001. Nian-Feng Tzeng's research was supported in part by the U.S. National Science Foundation under Grants MIP-9201308 and CCR-9300075 and by the State of Louisiana under Contract LEQSF(1994-96)-RD-A-39. A preliminary version of this paper was presented at the 23rd International Conference on Parallel Processing, August 1994.

REFERENCES

[1] T. Schwederski, H.J. Siegel, and T.L. Casavant, "Task Migration Transfers in Multistage Cube Based Parallel Systems," *Proc. 1989 Int'l Conf. Parallel Processing*, vol. I, pp. 296-305, Aug. 1989.

[2] M.S. Chen and K.G. Shin, "Task Migration in Hypercube Multiprocessors," *Proc. 16th Int'l Symp. Computer Architecture*, pp. 105-111, May 1989.

[3] G.-I. Chen and T.-H. Lai, "Constructing Parallel Paths Between Two Subcubes," *IEEE Trans. Computers*, vol. 41, no. 1, pp. 118-123, Jan. 1992.

[4] O. Frieder et al., "Experimentation with Hypercube Database Engines," *IEEE Micro*, pp. 42-56, Feb. 1992.

[5] L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, vol. 26, no. 2, pp. 62-76, Feb. 1993.

[6] P.K. McKinley and C. Trefftz, "Efficient Broadcast in All-Port Wormhole-Routing Hypercubes," *Proc. 1993 Int'l Conf. Parallel Processing*, vol. II, pp. 288-291, Aug. 1993.

[7] P. Krueger, T.-H. Lai, and V.A. Dixit-Radiya, "Job Scheduling Is More Important that Processor Allocation for Hypercube Computers," *IEEE Trans. Parallel and Distributed Systems*, vol. 5, no. 5, pp. 488-497, May 1994.

[8] N.-F. Tzeng and H.-L. Chen, "On-Line Task Migration in Hypercubes Through Double Disjoint Paths," Technical Report TR-95-8-3, Center for Advanced Computer Studies, Univ. of Southwestern Louisiana, 1995.