論文95-32B-7-2

회선교환방식 하이퍼큐브에서 작업이동을 위한 라우팅 알고리즘

(Routing Algorithm for Task Migration in Circuit-switched Hypercube)

金 大 榮*, 崔 相 昉**

(Dae Young Kim, Sang Bang Choi)

요 약

하이퍼큐브 다중 프로세서 시스템에서는 반복되는 프로세서 할당 및 할당해제로 인해, 이용가능한 프로세서가 충분하여도 이들 프로세서가 새로운 작업에 필요한 크기의 서브큐브를 구성할 수 없는 프레그먼테이션이 나타난다. 이러한 프레그먼테이션을 제거하기 위하여 작업이 할당된 서브큐브를 유휴 서브큐브로 작업을 이동시켜, 흩어져 있는 작은 서브큐브를 모으는 프로세서 재배치가 필요하다. 본 논문에서는 회선교환 하이퍼큐브의 프로세서 재배치에 필요한 작업이동을 위한 라우팅 알고리듬을 제안한다. 회선교환방식을 사용하는 하이퍼큐브에서의 작업이동 알고리듬은 한 서브큐브에서 다른 서브큐브로 작업을 이동시킬 수 있는 각 노드간 독립된 전용 라우팅 경로를 찾는 것이다. 본 논문에서 제안한 알고리듬은 모든 종류의 서브큐브를 검색할 수 있는 PGC(packed Gray code)를 기초로 고안했다.

Abstract

In the hypercube multiprocessor system, repeated allocations and deallocations of subcubes generate a fragmented hypercube from which, even if sufficient free processors are available, a subcube that is large enough to accomodate a new task cannot be formed. To eliminate the fragmentation, we need a processor relocation to move a task-occupied subcube to a free subcube, and gather dispersed small subcubes. In this paper, we propose a routing algorithm for task migration in circuit-switched hypercube to relocate processors. In the circuit-switched hypercube, we have to find a set of dedicated link-disjoint routing paths for each node to move a task from a busy subcube to a free subcube in fragmentation. The proposed algorithm is based on the PGC(packed Gray code) which detects all kinds of subcubes.

I. 서 론

여러개의 프로그램을 동시에 실행할 수 있는 MIMD (multiple instruction stream-multiple data stream) 병렬처리 컴퓨터상에서 어떤 작업을 처리하기 위해서는 우선 작업이 처리될 프로세서들을 결정하고, 결정된 프로세서에서 주어진 프로그램을 실행하여야 한다. 또한 프로그램의 실행이 끝나면 그 작업에 사용되었던 프로세서들을 반환하여 다른 작업의 수행에 이용될 수 있도록해야 한다. 이와 같은 과정을 프로세서 할당(allocation)과 할당해제(deallocation)라고 한다[1]. 하이퍼큐브(hypercube)[2] 다중 프로세서

* 正會員, 現代電子産業株式會社

(Hyundai Electronics Industries Co., Ltd.)

** 正會員, 仁荷大學校 電子工學科

(Dept. of Elec. Eng., Inha Univ.)

接受日字: 1995年5月28日, 수정완료일: 1995年7月12日

시스템에서는 어떤 작업이 p개의 프로세서를 요구하는 경우 k차원 ($p \le 2k$를 만족하는 가장 작은 정수) 서브큐브(subcube)를 해당 작업에 할당한다[3-6].

반복적인 프로세서 할당과 할당해제로 인해 하이퍼큐브내에는 이용 가능한 프로세서가 충분하여도, 이들 프로세서가 새로운 작업이 필요로 하는 크기의 서브큐브를 구성할 수 없는 경우가 있다. 그림 1은 이러한 프레그먼테이션(fragmentation: 작업이 할당되지 않은 유휴 서브큐브들)을 갖는 4차원 하이퍼큐브의 예를 보여준다. 이 하이퍼큐브내에는 비록 이용 가능한 프로세서가 8개(흰색 노드에 해당) 있으나, 3차원 서브큐브를 구성하지 못한다. 프레그먼테이션은 작업들이 필요로 하는 서브큐브의 크기가 다양할수록 더욱 심해져 전체 프로세서의 이용율을 저하시킨다. 이는 세그먼테이션(segmentation)을 사용하는 주기억 장치의 프레그먼테이션과 같다[7]. 이러한 프레그먼테이션을 제거하기 위하여 작업이 할당된 서브큐브를 유휴 서브큐브로 작업을 이동시켜, 흩어져 있는 작은 서브큐브를 모으는 프로세서 재배치(relocation)가 필요하다.

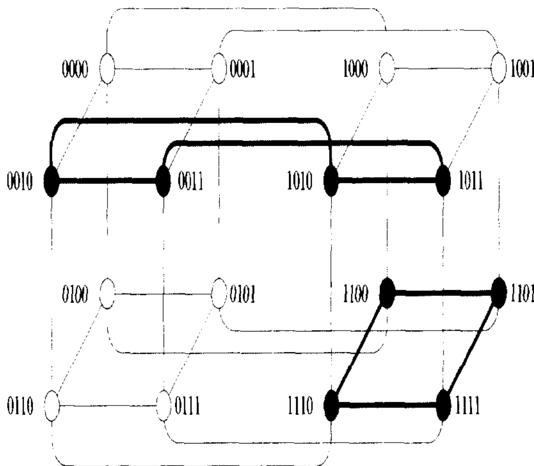


그림 1. 4차원 하이퍼큐브내의 프레그먼테이션.
Fig. 1. Fragmentation in 4 dimensional hypercube.

M. S. Chen과 K. G. Shin[8]이 제안한 그레이코드를 이용한 작업이동(task migration) 알고리듬에서는, 프로세서를 재배치하기 위하여 시작점 서브큐브내의 각 노드번지보다 낮은 번지의 도착점 서브큐브를 구해 패킷교환(packet switching)을 사용하여 작업을 이동시킴으로써 통신중에 일어날 수 있는 교착상태(deadlock)[9-10]를 제거한다. G. I. Chen과 T. H. Lai[11]가 제안한 알고리듬은 회선교환(circuit switching) 통신하에서 작업이 할당된 서브큐브번지를 사용하여 시작점 서브큐브로부터 도착점 서브큐브까지 최단거리 경로를 제공한다. 하이퍼큐브의 차원이 n이고 시스템내에 작업이 할당된 서브큐브의 갯수를 m이라 할 때 제안된 알고리듬의 시간 복잡도(time complexity)는 $O(n^2m)$이다. H. L. Chen과 N. F. Tzeng[12]은 참고문헌 [11]에서 제안한 알고리듬을 이용하여 각 노드간 두개의 서로 중첩되지 않는 경로를 제공한다. 그러나 설정된 경로는 항상 최단거리가 되지 않는다.

본 논문에서는 PGC(packed Gray code)를 이용하여 회선교환방식 하이퍼큐브내의 프레그먼테이션을 제거하기 위한 작업이동 알고리듬을 제안한다. 회선교환에서의 작업이동 알고리듬은 작업이 할당된 서브큐브(시작점)로 부터 프레그먼테이션을 이루는 유휴 서브큐브(도착점)로 작업을 이동시킬 수 있는 각 노드간의 전용 라우팅(routing) 경로를 찾는 것이다. 하이퍼큐브에서 이미 작업을 수행중인 서브큐브를 구성하고 있는 링크(link)는 사용하지 않는다. 즉, 시작점 서브큐브와 도착점 서브큐브의 각 노드사이에 유휴 링크로 이루어진 서로 중첩되지 않는(link-disjoint) 라우팅 경로를 찾아야 한다.

본 논문에서 제안한 알고리듬은 시작점 서브큐브와 도착점 서브큐브의 각 노드간 양 방향의 최단거리 통신경로를 설정한다. 통신경로는 서브큐브의 각 노드에서 병렬로 라우팅 알고리듬을 수행하여 설정할 수 있으므로 시간 복잡도는 $O(n^2)$이다. 또한 시작점 서브큐브와 도착점 서브큐브 자체를 이루는 링크는 사용하지 않는다. 이를 이용하면 프레그먼테이션을 제거하기위한 작업이동 뿐 아니라, MIMD 하이퍼큐브에서 유용하게 사용될 수 있는 두 서브큐브 노드간의 1 대 1 사상(mapping)으로 표현되는 모든 통신 패턴에 대하여 경로를 설정할 수 있다. 본 논문에서 사용된 회선교환 하이퍼큐브의 노드는 그림 2와 같이 PE(processing element)와 스위치 모듈로 구성되어 있다. 스위치 모듈은 n개의 이웃하는 노드와 자신의 PE 모듈간을 연결하는 $n \times n$ 크로스바(crossbar) 스위치이다.

본 논문은 다음과 같이 구성되어 있다. II 장에서는 두 서브큐브간 작업이동에 사용되는 그레이코드와 PGC를 설명하였다. III 장에서는 하이퍼큐브내에서,

작업중인 서브큐브들을 구성하는 링크를 통과하지 않는, 임의의 두 노드간의 라우팅 경로를 설정하는 알고리듬을 고안하였다. IV 장은 반복되는 프로세서 할당 및 할당해제에 의해 발생되는 프레그먼테이션을 제거하기 위한 PGC와 서브큐브 번지를 이용한 작업이동 알고리듬을 제안하였다. V 장에서 본 논문의 결론을 맺는다.

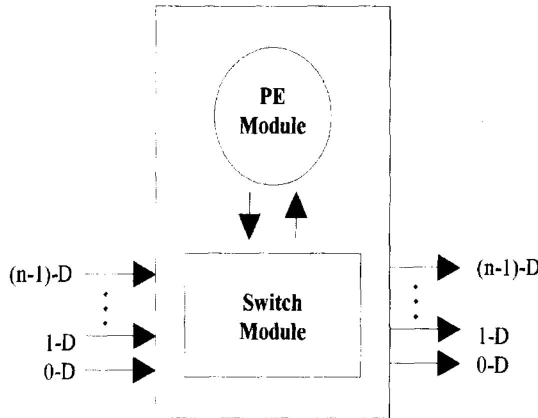


그림 2. 회선교환방식 하이퍼큐브의 노드 구조
Fig. 2. Node structure of circuit-switched hypercube.

II. 그레이 코드

1. 그레이코드

그레이코드(Gray code)는 이웃한 이진수와의 헤밍거리(Hamming distance)가 항상 1인 코드이다. n 개의 비트로 이루어진 그레이코드의 종류는 모두 n!개가 존재하는데 대표적인 것이 BRGC (binary reflected Gray code)이다. n 비트 $BRGC_n$은 다음과 같이 재귀적으로 정의된다[4].

$$BRGC_1 = \{0, 1\}$$
$$BRGC_k = \{(0)BRGC_{k-1},\ (1)BRGC^*_{k-1}\},\ 2 \le k \le n$$

(b) $BRGC_k$, b∈{0, 1} 는 $BRGC_k$를 k 비트의 이진수로 구성된 수열이라고 했을때 $BRGC_k$의 모든 원소의 최상위 비트 좌측에 비트값 b를 추가한 것이다. 예를 들어 (0) $BRGC_1$은 $BRGC_1$, {0, 1}의 각 원소의 좌측에 0을 추가한 {00, 01}이 된다. 또한 $BRGC^*_k$는 $BRGC_k$의 원소를 역순으로 나열한 것이다. $BRGC^*_1$은 $BRGC_1$, {0, 1}의 순서를 반대로한 {1, 0}이 된다.

이와 같이 $BRGC_n$을 정의하면 n 비트로 나타낼 수 있는 일반적인 그레이코드는 다음과 같이 구할 수 있다. 우선 $BRGC_n$의 각 비트에 좌측으로부터 1, 2, 3, …, n 과 같이 비트 번호를 붙인다. 그러면 n 비트 그레이코드는 $[g_1, g_2, \cdots, g_n]$, $1 \le g_k \le n$ 으로 나타낼 수 있다. 여기서 gk는 $BRGC_n$의 비트 번호 g_k를 좌측으로부터 k번째 위치로 옮기는 것을 의미한다. $BRGC_n$는 비트 위치와 비트 번호가 일치하므로 $[g_1, g_2, \cdots, g_n]$ = [1, 2, …, n]과 같이 나타낸다. $[g_1, g_2, \cdots, g_n]$으로 나타낼 수 있는 그레이코드의 종류는 1부터 n까지의 정수를 나열하는 순열의 경우의 수, n!개가 존재한다. $[g_1, g_2, g_3, g_4]$ = [3, 1, 4, 2]인 그레이코드를 그림 3에 나타내었다. 그림에서 g_1 = 3 이므로 $BRGC_4$의 비트번호 3을 첫번째 위치로 옮긴다. 마찬가지로 $BRGC_4$의 비트번호 1, 4, 2를 각각 2, 3, 4 번째 위치로 옮긴다.

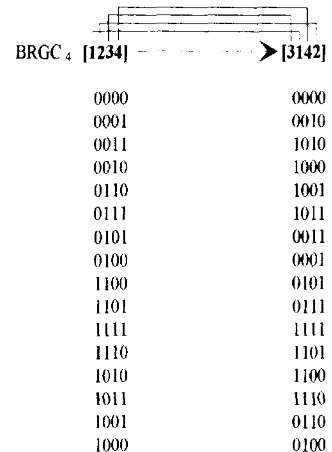


그림 3. 그레이코드의 예 [3, 1, 4, 2]
Fig. 3. Example of Gray code [3, 1, 4, 2].

2. PGC (Packed Gray Code)

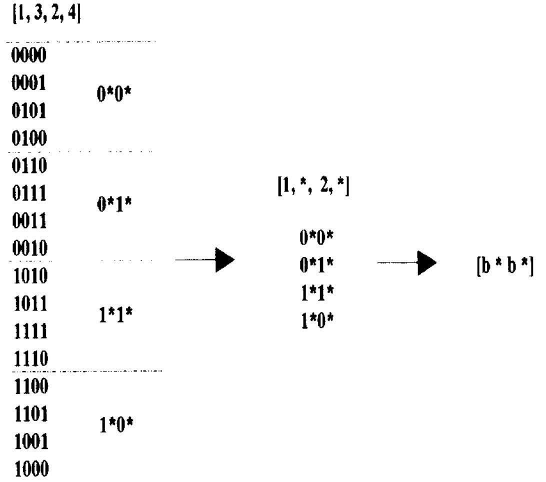
n차원 하이퍼큐브에서 k차원 서브큐브는 2^k개의 프로세서로 구성된다. 각 프로세서의 번지는 n비트 이진수로 표현되는데, 한 k차원 서브큐브에 속하는 2^k개의 프로세서의 번지는 n-k개의 동일한 비트를 갖는다. 서브큐브의 번지는 n-k개의 동일한 비트를 제외한 나머지 비트를 * (don't care)로 대치함으로써 나타낼 수 있다. 예를 들면 4개의 노드 10001, 10011, 11001,

11011로 구성된 5차원 하이퍼큐브내의 2차원 서브큐브 번지는 1*0*1과 같이 나타낸다.

[정의 1] n비트 그레이코드에서 각 원소들은 하이퍼큐브의 프로세서 번지에 해당한다. 그레이코드 $[g_1, g_2, \cdots, g_n]$의 원소들을 처음부터 순서대로 2^k개씩 나누면, 각 그룹은 하이퍼큐브상에서 k차원 서브큐브를 이루는 프로세서들의 집합이 되며, 각 프로세서들의 집합을 서브큐브 번지로 나타낼 수 있는데, 이렇게 그레이코드를 이용하여 서브큐브 번지를 나타낸 것을 PGC(packed Gray code)라 하고 $PGC^k{}_n = [g_1', g_2', \ldots, g_n']$으로 나타낸다. 여기서 아래첨자 n은 그레이코드의 비트수를 나타내고, k는 2^k개씩 묶어 k차원 서브큐브 번지로 나타냄을 의미하며, 각 비트 번호 g_i'는 다음과 같이 결정된다.

$$g_i' = \begin{cases} g_i, & 1 \le g_i \le n-k \\ *, & n-k+1 \le g_i \le n \end{cases}$$

*로 표시된 g_i'에 나열할 수 있는 비트 번호의 경우의 수는 k!이 되며, 이렇게 얻어진 k!개의 그레이코드는 동일한 $PGC^k{}_n$로 표현된다. 그리고 n비트로 표현되는 $PGC^k{}_n$의 종류는 n개의 비트위치중 (n-k)개를 선택하여 비트번호 1 부터 n-k까지 나열하는 경우의 수 C(n, n-k)•(n-k)!이 된다. *의 위치가 같은 $PGC^k{}_n$는 서로 같은 종류의 k차원 서브큐브 번지를 생성한다. 그림 4는 그레이코드 [1, 3, 2, 4]로부터 얻은 $PGC^2{}_4$ = [1,*,2,*]의 한 예이다. [1, *,2, *]와 [2, *, 1, *]는 동일한 서브큐브 번지를 생성하며 [b * b *]로 나타낸다. 여기서 'bb'에 모두 이진수 조합을 사용하면 그러한 서브큐브 번지를 얻을 수 있다. 표 1은 4 차원 하이퍼큐브에서 가능한 모든 $PGC^2{}_4$, 그리고 각 $PGC^2{}_4$로 부터 얻어지는 그레이코드와 2 차원 서브큐브 번지를 보여준다. 한 PGC로부터 얻어지는 모든 서브큐브들은 서로 다른 노드로 구성되지만 동일한 차원의 링크를 사용한다. 예를 들면 [2, 1, *, *]로부터 얻어지는 서브큐브들은 0과 1차원 링크를 이용하며, 서브큐브 00**를 구성하는 노드 {0000, 0001, 0011, 0010}와 서브큐브 11** 를 구성하는 노드 {1100, 1101, 1111, 1110}는 서로 다르다. 0과 1차원 링크를 사용하는 모든 서브큐브 {00**, 01**, 11**, 10**}는 [b b * *]로 나타낼 수 있는 [1, 2, *, *]나 [2, 1, *, *]로부터 얻을 수 있다.



(a) 그레이코드 [1, 3, 2, 4] (b) $PGC^2{}_4$ = [1, *, 2, *]

그림 4. 그레이코드 [1, 3, 2, 4]로 부터 얻은 $PGC^2{}_4$ = [1, *, 2, *]

Fig. 4. $PGC^2{}_4$ = [1, *, 2, *] obtained from Gray code [1, 3, 2, 4].

표 1. $PGC^2{}_4$ 와 그에 속하는 그레이코드

Table 1. Gray codes contained in $PGC^2{}_4$.

PGC^2_4		Graycode	Subcube
[b b * *]	[1, 2, *, *]	[1, 2, 3, 4] [1, 2, 4, 3]	00** 01** 11** 10**
	[2, 1, *, *]	[2, 1, 3, 4] [2, 1, 4, 3]	
[b * b *]	[1, *, 2, *]	[1, 3, 2, 4] [1, 4, 2, 3]	0*0* 0*1* 1*1* 1*0*
	[2, *, 1, *]	[2, 3, 1, 4] [2, 4, 1, 3]	
[b * * b]	[1, *, *, 2]	[1, 3, 4, 2] [1, 4, 3, 2]	0**0 0**1 1**1 1**0
	[2, *, *, 1]	[2, 3, 4, 1] [2, 4, 3, 1]	
[* b b *]	[*, 1, 2, *]	[3, 1, 2, 4] [4, 1, 2, 3]	*00* *01* *11* *10*
	[*, 2, 1, *]	[3, 2, 1, 4] [4, 2, 1, 3]	
[* b * b]	[*, 1, *, 2]	[3, 1, 4, 2] [4, 1, 3, 2]	*0*0 *0*1 *1*1 *1*0
	[*, 2, *, 1]	[3, 2, 4, 1] [4, 2, 3, 1]	
[* * b b]	[*, *, 1, 2]	[3, 4, 1, 2] [4, 3, 1, 2]	**00 **01 **11 **10
	[*, *, 2, 1]	[3, 4, 2, 1] [4, 3, 2, 1]	

Ⅲ. 두 노드간의 라우팅 경로 설정

서브큐브 번지에서 *의 차원은 사용중인 링크의 차원를 나타내고, 이진수 (0 또는 1)로 표현된 비트의 차원은 프리(free) 링크의 차원를 나타낸다. 따라서 어느 한 노드에서 그 노드가 포함된 서브큐브 번지를 인식하면 어느 차원의 링크가 사용중이고, 어느 차원의 링크가 그렇지 않은지 알 수 있다.

이미 작업이 할당되어 수행중인 여러 서브큐브가 존재하는 하이퍼큐브내에서 임의의 한 노드에서 다른 한 노드로 라우팅 경로를 설정하는 알고리듬은 다음과 같다. 이 알고리듬에서 두 노드는 이미 결정 되어 있고, 그 노드들은 작업이 할당안된 프리 노드라 가정한다.

Procedure 1 (한 프리 노드에서 다른 프리 노드로의 라우팅 경로 설정)

Step 1: 시작점 노드번지와 도착점 노드번지를 비교하여 서로 다른 임의의 한 비트를 반전(toggle)시켜 새로운 노드번지를 얻는다.

Step 2: 새로운 노드번지가 속해있는 서브큐브의 번지를 구한다. 이 서브큐브의 번지로부터 사용중인 링크의 차원을 구한다. 새로운 노드번지와 도착점 노드번지를 비교하여 서로 다른 비트 중 이 서브큐브가 사용하지 않는 차원을 하나 선택하여 그 차원의 비트를 반전시킨다.

Step 3: 새로운 노드번지와 도착점 노드번지가 같아질 때까지 step 2를 반복한다.

위 라우팅 알고리듬에 의하여 얻어지는 노드는 시작점 노드와 도착점 노드 사이의 경로를 이루는 중간노드들이다. 이 알고리듬은 이전 단계에서 얻어진 노드로 퇴각(backtracking)하는 경우가 발생하지 않으며, 설정된 경로는 두 노드간의 최단거리이다.

［정리 1］ 시작점 노드와 도착점 노드가 모두 프리일 경우 Procedure 1을 이용하여 라우팅 경로를 설정하면 교착상태(deadlock)가 생기지 않으며, 이 라우팅 경로의 길이는 두 노드사이의 헤밍거리와 같다.

증명 : 시작점 노드와 도착점 노드를 포함하는 가장 작은 서브큐브를 만든다. 시작점 노드와 도착점 노드는 모두 프리라고 가정하였으므로 이 서브큐브 자체는 하나의 작업을 수행하는 서브큐브가 될 수 없다. 이 서브큐브내에서 Step 2에 의하여 얻어지는 새로운 중간노드와 도착점 노드를 포함하는 또다른 가장 작은 서브큐브를 형성하면, 이 서브큐브 역시 하나의 작업을 수행하는 서브큐브가 될 수 없다. 매 Step 2마다 얻어지는 서브큐브는 전체가 하나의 작업을 수행하는 서브큐브가 될 수 없으므로, 각 중간노드에는 도착점 노드와 서브큐브를 구성하는 차원중 사용하지 않는 차원이 적어도 하나 존재한다. 그러나 만약 사용하지 않는 차원이 없다면, 그것은 중간노드와 도착점 노드가 모두 작업중인 서브큐브를 구성하는 노드라는 것을 의미한다. 이 가정은 도착점 노드가 프리 노드이고 작업중인 서브큐브의 노드가 될 수 없다는 가정에 모순된다. 그러므로 라우팅 경로를 설정할 때 교착상태가 생기지 않는다. 따라서 Step 2에는 반전하기 위한 비트가 항상 존재한다. Step 2를 매번 수행할 때 마다 새로운 노드는 도착점 노드로 접근하며 헤밍거리는 1씩 줄어든다. 따라서 설정된 경로의 길이는 시작점 노드와 도착점 노드의 헤밍거리와 같다. Q.E.D.

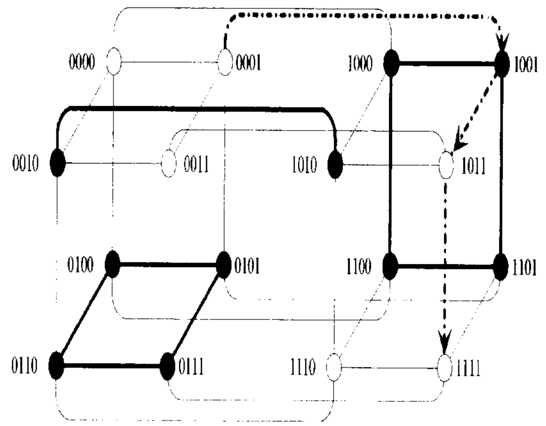


그림 5. Procedure 1에 의하여 설정된 라우팅 경로

Fig. 5. Routing path established by Procedure 1.

Step 2에 의하여 얻어지는 중간노드들은 작업을 수행중인 서브큐브에 소속된 노드일 수 있다. 그림 5는 위의 알고리듬을 적용하여 설정한 라우팅 경로를 보여준다. 이 그림에서 작업중인 서브큐브는 01**, *010, 1*0*이고, 시작점 노드와 도착점 노드는 각각 0001과 1111이다. 두 노드 0001과 1111를 포함하는 가장 작은 서브큐브는 ***1 = {0001, 0011, 0101, 0111, 1001, 1011, 1101, 1111}이다. 노드 0001은 프리 노드이므로 이 노드와 연결된 모든 링크

는 프리이다. 따라서 노드 0001은 노드 0011, 0101, 1001로 라우팅할 수 있다. Step 1에 의해 노드 0001은 그 중 한 노드인 1001로 라우팅한다. 노드 1001은 1011이나 1101로 라우팅할 수 있으나 노드 1001은 서브큐브 1*0*의 구성 노드임으로 1101로는 라우팅할 수 없으며 (1001과 1101간의 링크는 서브큐브 1*0*에 의하여 사용중임). 따라서 Step 2는 노드 1011을 새로운 중간노드로 선택한다. 노드 1011은 같은 방법으로 도착 노드 1111로 라우팅한다.

위의 알고리듬은 시작점 노드와 도착점 노드가 모두 프리인 경우 라우팅 경로를 설정하는 알고리듬이다. 시작점과 도착점 노드중 어느 한 노드가 작업중이고 다른 노드가 프리인 경우 (만약 시작점 노드가 프리이고 도착점 노드가 작업중인 경우) 라우팅 경로는 항상 가능하지 않다. 그러나 시작점 노드가 작업중이고 도착점 노드가 프리이면 라우팅 경로는 항상 설정되고 이 경우의 알고리듬은 다음과 같다.

Procedure 2 (작업중인 노드에서 다른 프리 노드로의 라우팅 경로 설정)

Step 1: 시작점 노드가 속해있는 작업중인 서브큐브의 번지를 구한다. 그 서브큐브의 번지로 부터 사용중인 링크의 차원을 구한다. 시작점 노드번지와 도착점 노드번지를 비교하여 서로 다른 비트중 작업중인 서브큐브가 사용하지 않는 차원을 하나 선택하여 그 차원의 비트를 반전시켜 새로운 중간노드번지를 구한다.

Step 2: 중간노드번지를 시작점 노드로하여 새로운 노드번지와 도착점 노드번지가 같아질 때까지 Step 1를 반복한다.

[정리 2] 시작점 노드는 작업중이고 도착점 노드는 프리인 경우, Procedure 2를 사용하여 두 노드사이의 라우팅 경로를 설정하면 교착상태가 생기지 않으며, 이 라우팅 경로의 길이는 두 노드 사이의 헤밍거리와 같다.

증명 : Procedure 1은 Procedure 2를 포함하고 있다. 즉 Procedure 2는 Procedure 1에서 작업중인 새로운 중간노드에서 도착점 노드로 라우팅하는 과정과 일치한다. 그러므로 정리 2는 정리 1에 의해 증명된다. Q.E.D.

[정리 3] Procedure 1과 2를 사용하여 설정된 라우팅 경로는 시작점 노드와 도착점 노드에 의해 구성되는 가장 작은 서브큐브내의 링크만을 사용한다.

증명 : Procedure 1과 2는 시작점 노드번지와 도착점 노드번지를 비교하여 서로 다른 비트만을 반전시켜 라우팅한다. 이들 반전된 비트는 시작점 노드와 도착점 노드가 형성하는 가장 작은 서브큐브가 사용하는 링크의 차원이다. 따라서 모든 중간노드들은 이 서브큐브에 포함되며 이 서브큐브를 형성하는 링크만을 사용하게 된다. Q.E.D.

예를 들면 그림 5의 노드 0100에서 1111로의 라우팅 경로는 0100 - 1100 - 1110 - 1111이다. 이 라우팅 경로는 시작점 노드와 도착점 노드를 동시에 포함하는 가장 작은 서브큐브 *1** = {0100, 0101, 0110, 0111, 1100, 1101, 1110, 1111}내의 노드와 링크만을 사용한다. 이 라우팅 경로를 설정할 때, 0, 1, 3차원의 비트를 반전시켰으며, 이 값들은 서브큐브 *1**를 구성하는 링크의 차원들이다.

Ⅳ. 작업이동 알고리듬

1. 시작점 노드와 대응되는 중간노드번지와 도착점 노드번지의 결정

시작점 서브큐브의 한 노드로부터 도착점 서브큐브의 한 노드로 라우팅 경로를 설정하기 위해 두 서브큐브사이의 한 노드를 중간노드(intermediate node)로 이용한다. 중간노드는 시작점 서브큐브를 생성하는 PGC내의 한 서브큐브의 노드가 되도록하고, 동시에 도착점 서브큐브를 생성하는 또다른 PGC내의 한 서브큐브의 노드가 되도록 선택한다. 즉 시작점 서브큐브와 중간노드들에 의해 구성되는 서브큐브들은 한 PGC내에 속하고, 같은 중간노드들에 의해 구성되는 또다른 형태의 서브큐브들과 도착점 서브큐브는 다른 한 PGC 내에 속한다. 이러한 특성을 갖는 중간노드는 다음과 같이 선택된다.

[정의 2] 시작점 서브큐브와 도착점 서브큐브가 주어지면 시작점 서브큐브의 각 노드번지와 대응되는 중간 노드번지와 도착점 노드번지는 각각 다음과 같이 정의된다. 여기서 시작점 서브큐브와 도착점 서브큐브는 k차원으로 가정하며, 시작점 서브큐브의 번지는 우측 k개의 디지트(digit)가 모두 *이고, 도착점 서브큐브의 번지는 우측의 적당한 비트위치 i로부터 좌측으로

연속하여 k개의 디지트가 모두 *라고 가정한다. 중요한 것은 시작점 서브큐브 번지의 *와 도착점 서브큐브 번지의 *의 상대적 위치이고, 실제 *의 위치는 중간노드 선택에 영향을 미치지 않는다.

경우 1) 시작점 서브큐브 $b_{n-1}^s \cdots b_k^s *_{k-1}^s \cdots *_0^s$와 도착점 서브큐브 $b_{n-1}^d \cdots b_k^d *_{k-1}^d \cdots *_0^d$가 같은 차원의 링크를 사용하는 경우:
시작점 서브큐브의 각 노드에 대응되는 중간노드와 도착점 노드는 동일하다. 즉 $b_{n-1}^d \cdots b_k^d *_{k-1}^s \cdots *_0^s$ 이다.

경우 2) 시작점 서브큐브 $b_{n-1}^s \cdots b_k^s *_{k-1}^s \cdots *_0^s$와 도착점 서브큐브 $b_{n-1}^d \cdots b_{k+i}^d *_{k+i-1}^d \cdots *_i^d b_{i-1}^d \cdots b_0^d$가 일부 같은 차원의 링크를 사용하는 경우 (k > i):
대응되는 중간노드는 $b_{n-1}^d \cdots b_{j+1}^d b_j^s \cdots b_{k+i}^s (*_{i-1}^s)_{k+i-1} \cdots (*_0^s)_k *_{k-1}^s \cdots *_i^s *_{i-1}^s \cdots *_0^s$이다. 여기서 $(*_{i-1}^s)_{k+i-1} \cdots (*_0^s)_k$는 $*_{i-1}^s \cdots *_0^s$를 복사한 것이고, 아래첨자 j는 $\lfloor (n-1+k+i)/2 \rfloor$이다. j는 $n-2 \ge j \ge k+i$ 사이의 임의의 값을 갖을 수 있으나 편의상 그 중간위치를 사용한다. 대응되는 도착점 노드는 $b_{n-1}^d \cdots b_{k+i}^d (*_{i-1}^s)_{k+i-1} \cdots (*_0^s)_k *_{k-1}^s \cdots *_i^s b_{i-1}^d \cdots b_0^d$이다.

경우 3) 시작점 서브큐브 $b_{n-1}^s \cdots b_k^s *_{k-1}^s \cdots *_0^s$와 도착점 서브큐브 $b_{n-1}^d \cdots b_{k+i}^d *_{k+i-1}^d \cdots *_i^d b_{i-1}^d \cdots b_0^d$가 서로 다른 차원의 링크를 사용하는 경우 (k ≤ i):
중간노드는 $b_{n-1}^d \cdots b_{j+1}^d b_j^s \cdots b_{k+i}^s (*_{k-1}^s)_{k+i-1} \cdots (*_0^s)_i b_{i-1}^s \cdots b_k^s *_{k-1}^s \cdots *_0^s$이다. 여기서 $(*_{k-1}^s)_{k+i-1} \cdots (*_0^s)_i$는 $*_{k-1}^s \cdots *_0^s$를 복사한 것이고, j는 $n-2 \ge j \ge k+i$, 또는 $i-1 \ge j \ge k$의 임의의 값을 갖을 수 있으나 편의상 $\lfloor (n-1+k+i)/2 \rfloor$로 사용한다. 도착점 노드는 $b_{n-1}^d \cdots b_{k+i}^d (*_{k-1}^s)_{k+i-1} \cdots (*_0^s)_i b_{i-1}^d \cdots b_0^d$이다.

경우 4) 경우 2와 3에서 중간노드 번지의 전체 디지트 중에서 이진 디지트가 하나만 (b_{n-1}) 있는 경우:
경우 2와 3에서 j = n-1일 때이며, 중간노드와 도착점 노드의 맨 좌측 디지트는 b_{n-1}^d가 되며, 이때 일부 중간노드번지와 이에 대응되는 도착점 노드번지가 같을 수 있다. 이러한 경우 b_{n-1}에는 b_{n-1}^s를 사용한다.

위의 정의에 의해 선택된 2^k개의 중간노드는 경우 1을 제외하고는 시작점 서브큐브나 도착점 서브큐브에 소속된 노드가 될 수 없다. 그림 6의 (a), (b), (c), (d)는 각 시작점 노드번지와 대응되는 중간노드번지, 도착점 노드번지를 보여준다. 시작점 서브큐브가 0***이고 도착점 서브큐브가 ***1인 경우는 정의 2의 어느 경우에도 해당되지 않으며, 이는 시작점 서브큐브와 도착점 서브큐브의 일부 노드가 서로 중복되기 때문이다. 따라서 시작점 서브큐브나 도착점 서브큐브 번지에 * 를 포함하지 않는 차원이 하나 이상 존재해야 두 서브큐브는 서로 중첩되지 않는다.

시작점 서브큐브 = $0000*_1^s*_0^s$
도착점 서브큐브 = $1111*_1^d*_0^d$
중간 노드 = $1111*_1^s*_0^s$
도착점 노드 = $1111*_1^s*_0^s$

시작점 노드번지	중간 노드번지	도착점 노드번지
000000	111100	111100
000001	111101	111101
000011	111111	111111
000010	111110	111110

(a) 경우 1

시작점 서브큐브 = $0000*_1^s*_0^s$
도착점 서브큐브 = $111*_2^d*_1^d1$
중간 노드 = $100(*_0^s)_2*_1^s*_0^s$
도착점 노드 = $111(*_0^s)_2*_1^s1$

시작점 노드번지	중간 노드번지	도착점 노드번지
000000	100000	111001
000001	100101	111101
000011	100111	111111
000010	100010	111011

(b) 경우 2

시작점 서브큐브 = $0000*_1^s*_0^s$
도착점 서브큐브 = $11*_3^d*_2^d11$
중간 노드 = $10(*_1^s)_3(*_0^s)_2*_1^s*_0^s$
도착점 노드 = $11(*_1^s)_3(*_0^s)_211$

시작점 노드번지	중간 노드번지	도착점 노드번지
000000	100000	110011
000001	100101	110111
000011	101111	111111
000010	101010	111011

(c) 경우 3

시작점 서브큐브 = $00*_1^s*_0^s$
도착점 서브큐브 = $1*_2^d*_1^d1$
중간 노드 = $b_3(*_0^s)_2*_1^s*_0^s$
b_3 = 0: 중간 노드 = 도착점 노드
b_3 = 1: 그렇지 않은 경우
도착점 노드 = $1(*_0^s)_2*_1^s1$

시작점 노드번지	중간 노드번지	도착점 노드번지
0000	1000	1001
0001	0101	1101
0011	0111	1111
0010	1010	1011

(d) 경우 4

그림 6. 시작점 노드번지와 대응되는 중간 노드번지 그리고 도착점 노드번지
Fig. 6. Source node addresses and its corresponding intermediate and destination node addresses.

[정리 4] 정의 2의 경우 1에서는 시작점 서브큐브와 도착점 서브큐브가 동일 PGC내에 포함되고, 경우 2, 3, 4에서는 중간노드들에 의해 구성되는 서브큐브들과 시작점 서브큐브는 한 PGC내에 포함되고, 같은 중간노드에 의해 구성되는 또다른 형태의 서브큐브들과 도

착점 서브큐브는 다른 PGC내에 포함된다.

증명 : 증명은 4 가지 경우로 나누어 다룬다.

경우 1) 시작점 서브큐브와 도착점 서브큐브가 같은 차원의 링크로 구성된 경우:

시작점 서브큐브 $b_{n-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$와 도착점 서브큐브 $b_{n-1}^{d} \cdots b_k^{d} *_{k-1}^{s} \cdots *_0^{s}$는 같은 차원의 링크에 의해 구성되어 있으므로 두 서브큐브는 한 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_k *_{k-1} \cdots *_0 \,]$ 내에 속한다.

경우 2) 시작점 서브큐브와 도착점 서브큐브가 일부 같은 차원의 링크로 구성된 경우:

시작점 서브큐브는 $b_{n-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$이고 도착점 서브큐브는 $b_{n-1}^{d} \cdots b_{k+i}^{d} *_{k+i-1}^{d} \cdots *_i^{d} b_{i-1}^{d} \cdots b_0^{d}$이면 (k 〉 i인 경우), 중간노드는 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s} *_{i-1}^{s} \cdots *_0^{s}$이고 도착점 노드는 $b_{n-1}^{d} \cdots b_{k+i}^{d} (*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s} b_{i-1}^{d} \cdots b_0^{d}$이다. 시작점 서브큐브 번지와 중간노드 번지의 우측 k개의 디지트 $*_{k-1}^{s} \cdots *_0^{s}$는 서로 같으므로 시작점 서브큐브 $b_{n-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$와 중간노드 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s} *_{i-1}^{s} \cdots *_0^{s}$는 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_k *_{k-1} \cdots *_0 \,]$ 내에 속하며 각 중간노드는 시작점 서브큐브내의 한 노드가 될 수 없다.

중간노드 번지와 도착점 서브큐브 번지의 $(*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s}$는 서로 같으므로, 도착점 서브큐브 $b_{n-1}^{d} \cdots b_{k+i}^{d} (*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s} b_{i-1}^{d} \cdots b_0^{d}$와 중간노드 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{i-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_k *_{k-1}^{s} \cdots *_i^{s} *_{i-1}^{s} \cdots *_0^{s}$는 2^{n-k}개의 서브큐브를 포함하는 또다른 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_{k+i} *_{k+i-1} \cdots *_i b_{i-1} \cdots b_0 \,]$ 내에 속한다. 각 중간노드는 도착점 서브큐브내의 한 노드가 될 수 없다.

경우 3) 시작점 서브큐브와 도착점 서브큐브가 서로 다른 차원의 링크로 구성된 경우:

시작점 서브큐브가 $b_{n-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$이고 도착점 서브큐브가 $b_{n-1}^{d} \cdots b_{k+i}^{d} *_{k+i-1}^{d} \cdots *_i^{d} b_{i-1}^{d} \cdots b_0^{d}$이면 (k ≤ i인 경우) 중간노드는 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_i b_{i-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$이고 도착점 노드는 $b_{n-1}^{d} \cdots b_{k+i}^{d} (*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_i b_{i-1}^{d} \cdots b_0^{d}$이다. 시작점 서브큐브와 중간노드 번지의 우측 k개의 디지트 $*_{k-1}^{s} \cdots *_0^{s}$는 서로 같으므로 시작점 서브큐브 $b_{n-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$와 중간노드 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_i b_{i-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$는 모두 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_k *_{k-1} \cdots *_0 \,]$ 내에 포함된다.

중간노드 번지와 도착점 서브큐브 번지의 $(*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})_i$는 같으므로, 도착점 서브큐브 $b_{n-1}^{d} \cdots b_{k+i}^{d} (*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_i b_{i-1}^{d} \cdots b_0^{d}$와 중간노드 $b_{n-1}^{d} \cdots b_{j+1}^{d} b_j^{s} \cdots b_{k+i}^{s} (*_{k-1}^{s})_{k+i-1} \cdots (*_0^{s})\, {}_i b_{i-1}^{s} \cdots b_k^{s} *_{k-1}^{s} \cdots *_0^{s}$는 모두 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_{k+i} *_{k+i-1} \cdots *_i b_{i-1} \cdots b_0 \,]$ 내에 포함된다.

경우 4) 경우 2와 3에서 중간노드 번지의 전체 디지트 중에서 이진 디지트가 하나인 경우:

경우 4는 경우 2와 3의 특수한 경우이므로 동일한 방법으로 증명된다. Q.E.D.

[정리 5] 두 서브큐브가 동일한 PGC (예를 들면 $PGC^{k}_{n} = [\, b_{n-1} \cdots b_k *_{k-1} \cdots *_0 \,]$) 내에 속하면, 각 서브큐브 번지에서 ∗를 이진수 패턴으로 대치하여 생성되는 두 서브큐브의 서로 대응되는 노드쌍으로 구성되는 가장 작은 서브큐브들은 모두 다른 한 PGC ($PGC^{n-k}_{n} = [\, *_{n-1} \cdots *_k b_{k-1} \cdots b_0 \,]$) 내에 속한다.

증명 : 위의 예에 대하여 증명하면, $PGC^{k}_{n} = [\, b_{n-1} \cdots b_k *_{k-1} \cdots *_0 \,]$ 내에서 두 서브큐브의 대응되는 노드쌍의 번지는 우측의 비트 0부터 k-1까지는 동일한 패턴을 갖고 좌측의 비트 k 부터 n-1까지는 서로 다르다. n차원 하이퍼큐브에서 현재 서브큐브를 구성하는 차원의 링크를 제거하면, 서브큐브에 사용되지 않는 차원의 링크로 이루어지는 새로운 (n-k) 차원 서브큐브가 형성되고 각 서브큐브에 소속된 노드들의 번지는 우측의 비트 0부터 k-1까지는 동일한 패턴을 갖고 좌측의 비트 k부터 n-1까지는 서로 다른 노드들로 구성된다. 따라서 대응되는 노드쌍으로 구성되는 가장 작은 서브큐브들은 또다른 $PGC^{n-k}_{n} = [\, *n-1 \cdots *kbk-1 \cdots b_0 \,]$ 내에 포함된다. Q.E.D.

그림 7은 한 PGC에 속하는 두 서브큐브간에 서로 대응되는 노드쌍으로 구성되는 서브큐브들은 다른 PGC내에 속하는 것을 보여준다. 두 서브큐브 00∗∗와 10∗∗는 PGC^{2}_{4} = [b b ∗ ∗] 내에 속한다. ∗∗에 11을 대치하면, 서로 대응되는 노드쌍은 0011과 1011이 되며, 이 두 노드는 4차원 하이퍼큐브에서 0과 1차원 링크를 제거하여 얻어진 서브큐브 ∗∗11에 포함된다. 이렇게 생성된 서브큐브들은 또다른 PGC^{2}_{4}

= [* * b b] 내에 속한다.

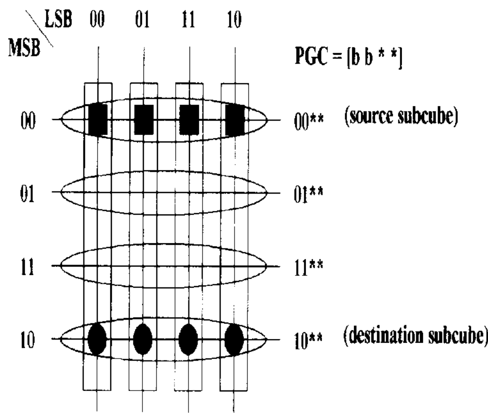


그림 7. 한 PGC에 속하는 두 서브큐브간에 서로 대응되는 노드쌍로 구성되는 서브큐브들은 다른 PGC내에 속한다

Fig. 7. The subcubes formed by each pair of nodes of two subcubes in a PGC are contained in another PGC.

[정리 6] 정의 2에서 시작점 노드와 이에 대응되는 중간노드로 구성되는 가장 작은 서브큐브가 사용하는 링크의 차원과, 중간노드와 이에 대응되는 도착점 노드로 구성되는 가장 작은 서브큐브가 사용하는 링크의 차원은 서로 다르다.

증명 : 두 노드로 구성되는 가장 작은 서브큐브는 두 노드번지에서 서로 다른 디지트의 차원으로 이루어진다. 정의 2의 경우 1을 제외한 나머지 경우에서 시작점 노드와 중간노드 번지간에 서로 다른 디지트의 차원과, 중간노드와 도착점 노드 번지간에 서로 다른 디지트의 차원은 중복되지 않는다. 따라서 시작점 노드와 중간노드가 구성하는 서브큐브의 차원과 중간노드와 도착점 노드가 구성하는 서브큐브의 차원은 서로 다르다. 경우 2를 예로 들면 중간노드가 시작점 노드와 비교하여 서로 다른 디지트는 $b_{n-1}^{d} \cdots b_{j+1}^{d}$와 $(*_{i-1}^{s})_{k+i-1} \cdots (*_{0}^{s})k$이고, 도착점 노드와 비교하여 서로 다른 디지트는 $b_{j}^{s} \cdots b_{k+i}^{s}$와 $*_{k-1}^{s} \cdots *_{0}^{s}$이다. 시작점 노드와 중간노드에 의하여 구성되는 서브큐브의 차원은 (n-1) 부터 (j+1), ($_{k+i-1}$) 부터 (k)이며, 중간노드와 도착점 노드에 의해 구성되는 서브큐브의 차원은 (j) 부터 (k+i), (i-1) 부터 (0)이다. 따라서 두 서브큐브가 사용하는 차원은 서로 다르다. Q.E.D.

그림 6의 (b)를 보면, 시작점 서브큐브는 $0000*_{1}^{s}*_{0}^{s}$이고 도착점 서브큐브는 $111*_{2}^{d}*_{1}^{d}1$이며, 대응되는 중간노드는 $100(*_{0}^{s})2*_{1}^{s}*_{0}^{s}$이고 도착점 노드는 $111(*_{0}^{s})_{2}*_{1}^{s}1$이다. 시작점 노드와 중간노드로 구성되는 서브큐브는 5와 2차원의 링크를 사용하며 도착점 노드와 중간노드로 구성되는 서브큐브는 4, 3, 0차원의 링크를 사용한다. 정의 2의 (b)에 의해 시작점 노드 번지 {000000, 000001, 000011, 000010}에 대응하는 중간노드번지는 {100000, 100101, 100111, 100010}이고 도착점 노드번지는 {111001, 111101, 111111, 111011}이다. 정리 5에 의해 시작점 노드와 이 노드에 대응하는 중간노드로 구성되는 서브큐브들은 5와 2차원 링크를 사용하며 다음과 같다.

{000000, 100000} : *00000 = {0 32}
{000001, 100101} : *00*01 = {1 5 33 37}
{000011, 100111} : *00*11 = {3 7 35 39}
{000010, 100010} : *00010 = {2 34}

위의 예에서 *00000의 경우는 *00*00를 대신 사용할 수 있으나, *00000가 000000과 100000을 포함하는 더 작은 서브큐브이다. 정리 5에 의해 중간노드와 이 노드에 대응되는 도착점 노드로 구성되는 서브큐브들은 4, 3, 0차원의 링크를 사용하며 다음과 같다.

{100000, 111001} : 1**00*
= {32 33 40 41 48 49 56 57}
{100101, 111101} : 1**101 = {37 45 53 61}
{100111, 111111} : 1**111 = {39 47 55 63}
{100010, 111011} : 1**01*
= {34 35 42 43 50 51 58 59}

2. 서브큐브 번지를 이용한 라우팅 알고리듬

정리 5에 의해 시작점 서브큐브의 각 노드는 이에 대응되는 중간노드와 서브큐브를 구성하며 이들은 모두 한 PGC내에 속한다. 따라서 이 PGC내의 각 서브큐브는 시작점 서브큐브의 한 노드와 그에 대응되는 중간노드 하나를 포함한다. 그리고 각 중간노드와 도착점 노드가 구성하는 서브큐브들은 다른 PGC내에 속하며, 이 PGC내의 각 서브큐브도 하나의 중간노드와 도착점 노드를 포함한다.

[정리 7] 한 PGC내에 속하는 두 서브큐브사이의 서로 대응되는 노드가 정리 5와 같이 쌍을 이룰 때, 대응되는 두 노드중 어느 한 노드가 프리이면 다른 한 노드에서 프리 노드로의 라우팅 경로는 항상 존재하며, 모든 대응되는 노드간의 라우팅 경로는 서로 노드가 중첩되지 않는다.

증명 : 정리 5에 의해 대응되는 두 노드를 포함하는 가장 작은 서브큐브들은 한 PGC내에 속한다. 그리고 대응되는 두 노드간의 라우팅 경로는 어느 한 노드만 프리이면 항상 존재함을 정리 2에서 증명하였고, 그 두 노드를 포함하는 가장 작은 그 서브큐브를 구성하는 링크만을 사용함을 정리 3에서 증명하였다. 한 PGC내의 각 서브큐브는 서로 다른 노드로 구성됨으로 각각의 경로는 서로 노드가 중첩되지 않는다. Q.E.D.

시작점 서브큐브내의 모드 노드가 정의 2에 의해 중간노드와 1대1 대응하면, 중간노드들에 의해 구성되는 서브큐브들과 시작점 서브큐브는 정리 4에 의해 한 PGC내에 포함되며, 서로 대응되는 노드는 정리 5의 노드쌍과 같다. 따라서 시작점 서브큐브가 프리이면 정리 7에 의해 모든 대응되는 노드간에는 노드가 중첩되지 않는 (node-disjoint 또는 parallel) 라우팅 경로가 존재한다. 각 중간노드와 도착점 서브큐브간에는 노드가 중첩되지 않은 독립된 라우팅 경로가 존재한다.

6차원 하이퍼큐브에서 시작점 서브큐브가 0000＊＊이고, 도착점 서브큐브111＊＊1인 경우를 예로들면 다음과 같다. 작업중인 6개의 서브큐브는 0＊010＊ = {000100, 000101, 010100, 010101}, ＊100＊＊ = {010000, 010001, 010010, 010011, 1100000, 110001, 110010, 110011}, 01＊11＊ = {010110, 010111, 011110, 011111}, 100＊＊＊ = {100000, 100001, 100010, 100011, 100100, 100101, 100110, 100111}, ＊01＊＊＊ = {001000, 001001, 001010, 001011, 001100, 001101, 001110, 001111, 101000, 101001, 101010, 101011, 101100, 101101, 101110, 101111}, 11＊1＊0 = {110100, 110110, 111100, 111110} 이라고 가정한다. 시작점 서브큐브 0000＊＊의 구성 노드는 {000000, 000001, 000011, 000010}이며, 정의 2의 경우 2에 의해 대응되는 중간노드는 {100000, 100101, 100111, 100010}이다. 중간노드는 이미 작업을 수행중인 서브큐브의 한 노드일 수 있으나 시작점 서브큐브는 하던 작업을 일단 멈추었으므로 모두 프리라 할 수 있다. 두 노드간의 라우팅 경로를 설정할 때 정리 2에 의해 목적지 노드는 항상 프리이어야 함으로, 시작점 노드에서 중간노드로의 경로 설정은 실제로 중간노드에서 시작점 노드로 설정해야 한다. 시작점 노드와 대응되는 중간노드간의 라우팅 경로는 다음과 같다.

(100000 → 000000) : 100000(100＊＊＊) → 000000
(100101 → 000001) : 100101(100＊＊＊)
→ 000101(0＊010＊) → 000001
(100111 → 000011) : 100111(100＊＊＊)
→ 000111(프리노드) → 000011
(100010 → 000010) : 100010(100＊＊＊) → 000010

위에서 콜론의 좌측은 중간노드에서 시작점 노드로의 경로설정을 의미하며, 우측은 그경로를 이루는 각 노드 및 그 노드가 소속된 서브큐브 번지를 나타낸다. 정의 2의 경우 2에 의해 대응되는 도착점 서브큐브의 노드번지는 {111001, 111101, 111111, 111011}이다. 도착점 노드는 항상 프리이므로 중간노드에서 도착점 노드로 경로를 설정할 수 있다. 중간노드에서 이 노드와 대응되는 도착점 노드로의 라우팅 경로는 다음과 같다.

(100000 → 111001) : 100000(100＊＊＊)
→ 110000(＊100＊＊) → 111000(프리노드) → 111001
(100101 → 111101) : 100101(100＊＊＊)
→ 110101(프리노드) → 111101
(100111 → 111111) : 100111(100＊＊＊)
→ 110111(프리노드) → 111111
(100010 → 111011) : 100010(100＊＊＊)
→ 110010(프리노드) → 111010(프리노드) → 111011

시작점 서브큐브의 한 노드에서 도착점 서브큐브의 한 노드로의 라우팅 경로는 시작점 노드에서 중간노드로의 라우팅 경로와 중간노드에서 도착점 노드로의 경로를 서로 직렬로 연결한 것이다. 따라서 작업이 할당된 서브큐브로 부터 프레그먼테이션을 이루는 프리 서브큐브로 작업을 이동시킬 수 있는 각 노드간의 전용 라우팅 경로는 다음 알고리듬에 의하여 구할 수 있다. 그림 8은 위의 예에서 설정된 경로를 6차원 하이퍼큐브상에서 나타낸 것이다.

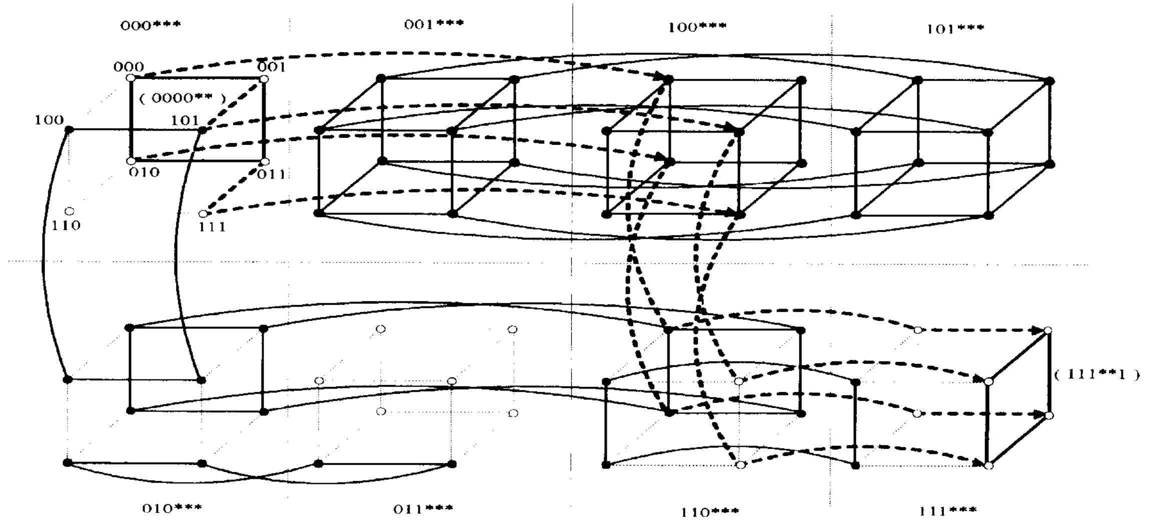


그림 8. 6차원 하이퍼큐브에서 두 서브큐브 0000**와 111**1 사이의 작업이동 경로
Fig. 8. Task migration paths between two subcubes 0000** and 111**1 in 6 dimensional hypercube.

그림에서 실선에 의하여 연결된 노드들은 작업중인 서브큐브를 나타낸다. 위의 예에 대한 설명과 그림으로부터 알고리듬의 정확함에 대한 증명은 복잡하지만 알고리듬 자체는 간단하며 매우 효율적으로 수행될 수 있음을 알 수 있다.

작업중인 서브큐브에서 프리 서브큐브로의 라우팅 알고리듬

시작점 서브큐브 번지와 도착점 서브큐브 번지에서 *의 차원을 구한다.

if (정의 2의 경우 1){

Step 1: 정의 2의 1을 사용하여 각 시작점 노드와 대응되는 도착점 노드 (중간노드와 동일) 를 구한다.

Step 2: 각 시작점 노드에서 대응되는 도착점 노드로 Procedure 1을 실행한다.

}

else (정의 2의 경우 2, 3, 또는 4){

Step 1: 정의 2의 2, 3, 또는 4를 사용하여 각 시작점 노드와 대응되는 중간노드 및 도착점 노드를 구한다.

Step 2: 각 중간노드에서 대응되는 시작점 노드로 Procedure 2를 실행한다.

Step 3: 각 중간노드에서 대응되는 도착점 노드로 Procedure 2를 실행한다.

Step 4: Step 2와 3에서 얻은 라우팅 경로를 서로 직렬로 연결한다.

}

[정리 8] 위 알고리듬을 사용하면, 시작점 서브큐브의 각 노드로 부터 도착점 서브큐브의 대응되는 노드로의 라우팅 경로는 서로 다른 링크(link-disjoint)를 사용하며, 최단거리이고, 경로 설정중에는 교착상태가 발생하지 않는다.

증명 : if와 else절의 Step 2에 의하여 얻어지는 경로는 서로 노드가 중첩되지 않음을 정리 7에서 증명하였다. else절의 Step 3에 의하여 얻어지는 경로는 서로 노드가 중첩되지 않으나, Step 4에서 두 경로를 서로 직렬로 연결할때 두 경로사이에는 노드가 서로 중첩되는 경우가 발생할 수 있다. 그러나 정리 6에서 두 경로가 사용하는 링크의 차원이 서로 다르다는 것을 증명하였으므로, Step 4에서 얻은 모든 경로는 서로 다른 링크를 사용한다.

Procedure 1과 2에 의하여 설정된 경로의 길이는 두 노드간의 헤밍거리이고, 경로 설정중에 교착상태가 발생하지 않음은 정리 1과 2에서 이미 증명하였다. 시작점 노드에서 중간 노드간의 경로길이와 중간 노드에서 도착점 노드간의 경로길이의 합은 시작점 노드번지와 도착점 노드번지간의 헤밍거리와 같으므로, 알고리

듬에 의하여 설정된 시작점 서브큐브에서 도착점 서브큐브간의 모든 경로는 최단거리이다. Q.E.D.

[정리 9] 시작점 서브큐브와 도착점 서브큐브간에는 링크가 중첩되지 않는 양 방향의 라우팅 경로가 존재한다.

증명 : 회선교환 하이퍼큐브에서 이웃하는 두 노드는 서로 다른 방향의 두 링크에 의하여 연결된다. 제안된 알고리듬에 의해 설정된 시작점 서브큐브에서 도착점 서브큐브간의 라우팅 경로상에서, 두 인접 노드는 어느 한쪽 방향의 링크만을 사용한다. 따라서 나머지 반대 방향의 링크를 사용하면 도착점 서브큐브에서 시작점 서브큐브로의 라우팅 경로가 된다. Q.E.D.

그림 8은 시작점 서브큐브와 도착점 서브큐브간의 경로는 최단거리이며 또한 양방향의 경로가 존재하는 것을 보여준다. 시작점 서브큐브내의 각 노드에 도착점 서브큐브와 현재 작업을 수행중인 서브큐브의 번지가 주어진다면, 제안된 알고리듬은 각 노드에서 병렬로 수행되어질 수 있다. 만약 서브큐브 번지가 크기 n의 어레이(array)로 표현된다면, 제안된 알고리듬의 step 1에서 시작점 노드와 대응되는 중간 노드와 도착점 노드는 O(n) 시간 안에 결정될 수 있다. Step 2와 3에서 Procedure 1이나 2의 수행시간은 O(n)이고, 설정되는 경로의 길이는 최대 n이므로 O(n2)의 시간 복잡도를 갖고, step 4에서는 단순히 두 경로를 연결함으로 일정한 시간(O(1)) 안에 이루어 질 수 있다. 따라서 제안된 알고리듬의 시간 복잡도는 O(n2)이다.

3. 시작점 서브큐브의 노드와 도착점 서브큐브의 노드간의 사상

하이퍼큐브는 다단 연결망(MIN: multistage interconnection network)으로 사상(mapping)이 가능하다[13]. 그림 9는 2차원 서브큐브를 2단 연결망으로 사상한 것을 보여준다. 만약 k차원 시작점 서브큐브를 k단 연결망으로 사상하고 도착점 서브큐브는 역상(mirror image)이 되도록 사상한후, 그림 10과 같이 서로 직렬로 연결하여 2·k단 베니스 네트워크(2·k stage Benes network) 를 형성하면, 2^k개의 시작점 노드와 2^k개의 도착점 노드사이에 순열(permutation)로 표현 가능한 모든 통신패턴을 서로 연결할 수 있다. 순열 μ : $PE_i \rightarrow PE_j$는 1대1 사상이며, 예를 들어 $\mu(PE_0) = PE_3$, $\mu(PE_1) = PE_2$, $\mu(PE_2) = PE_0$, $\mu(PE_3) = PE_1$은 ($PE_0 \rightarrow PE_3$), ($PE_1 \rightarrow PE_2$), ($PE_2 \rightarrow PE_0$), ($PE_3 \rightarrow PE_1$)의 통신 형태를 나타낸다. 그림에서 두 망의 연결이 가능한 것은 연결되는 두 노드사이에 링크가 중첩되지 않는 라우팅 경로가 존재하고, 각 라우팅 경로는 시작점 서브큐브와 도착점 서브큐브를 구성하는 링크를 사용하지 않기 때문이다.

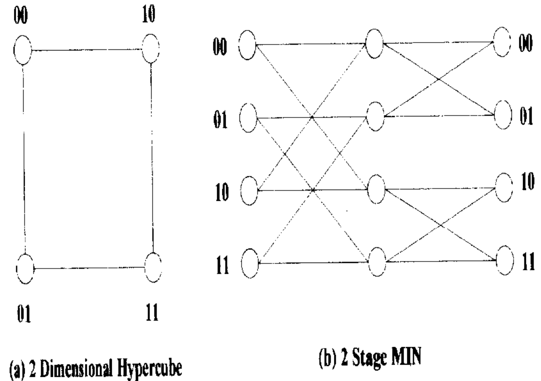


그림 9. 2단 연결망으로 사상된 2차원 하이퍼큐브
Fig. 9. 2 dimensional hypercube mapped into 2 stage MIN.

[정리 10] 두 MIN을 그림 10과 같이 서로 연결한 2·k단 베니스 네트워크(2·k stage Benes network)에서 2^k개의 시작점 노드와 2^k개의 도착점 노드사이에 순열 (μ)로 표현되는 통신패턴이 주어지면, 각 시작점 노드 i ($1 \le i \le 2^k$)로 부터 도착점 노드 μ(i)로 링크가 중첩되지 않는 라우팅 경로가 존재한다.

증명 : 시작점 서브큐브와 도착점 서브큐브는 다단 연결망으로 사상하여 그림 10과 같이 베니스 네트워크(Benes network)로 연결할 수 있다. 서로 역상인 두 다단 연결망을 서로 직렬로 결합하여 형성한 베니스 네트워크(Benes network)는 시작점 노드와 도착점 노드사이에 순열로 표현 가능한 모든 통신 패턴을 연결할 수 있다. 시작점 서브큐브와 도착점 서브큐브내의 모든 링크들은 앞 절의 재할당 알고리듬에서 경로를 설정할 때 사용되지 않는다. 그리고 두 서브큐브간에는 링크가 중첩되지 않는 서로 독립된 라우팅 경로가 항상 존재한다. 따라서 시작점 서브큐브의 각 노드에서 도착점 서브큐브의 임의의 노드로 서로 링크가 중첩되지 않는 독립된 라우팅 경로가 존재한다. Q.E.D.

예를 들어 시작점 서브큐브를 0000**라 하고 도착점 서브큐브를 111**11라 하면, 시작점 서브큐브

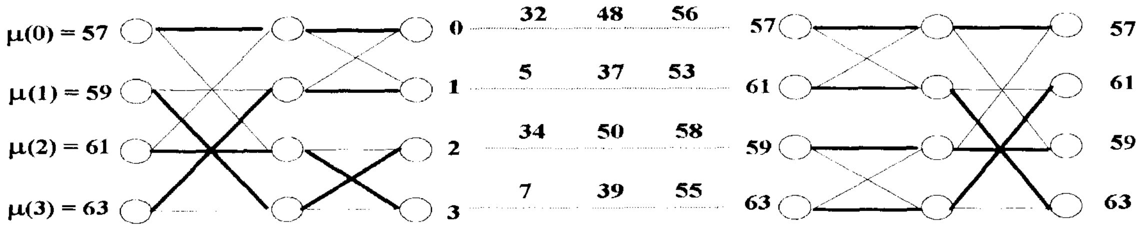


그림 10. 2단 연결망과 그것의 역상인 연결망을 서로 결합시킨 베니스 네트워크
Fig. 10. 4-stage Benes network constructed with a 2-stage MIN and its mirror image.

의 각 노드 번지와 도착점 서브큐브의 노드 번지사이에는 통신패턴 (0 → 57), (1 → 59), (2 → 61), (3 → 63)이 가능하다. 이와 같은 통신패턴은 순열 μ(0) = 57, μ(1) = 59, μ(2) = 61, μ(3) = 63이 되고, 베니스 네트워크 자체내에서의 경로설정은 다음과 같다 (그림 10 참조).

μ(0) = 57 : 0 …… 57
μ(1) = 59 : 1 → 3 → 2 …… 59
μ(2) = 61 : 2 → 3 …… 63 → 61
μ(3) = 63 : 3 → 1 …… 61 → 63

위의 예에서 [0 …… 57], [2 …… 59], [3 …… 63], [1 …… 61] 간의 경로는 앞 절의 라우팅 알고리듬을 사용하면 구할 수 있다. 따라서 작업중인 서브큐브에서 프리 서브큐브로의 라우팅 알고듬과 베니스 네트워크을 동시에 사용하면, 시작점 서브큐브의 각 노드를 도착점 서브큐브의 임의의 한 노드로 라우팅이 가능하다.

μ(0) = 57 : [0 → 32 → 48 → 56 → 57]
μ(1) = 59 : 1 → 3 → [2 → 34 → 50 → 58 → 59]
μ(2) = 61 : 2 → [3 → 7 → 39 → 55 → 63] → 61
μ(3) = 63 : 3 → [1 → 5 → 37 → 53 → 61] → 63

위의 예에서 [ ] 내의 경로는 재할당 알고리듬을 이용하여 얻은 라우팅 경로이다. 이 경로는 시작점 서브큐브와 도착점 서브큐브를 각각 사상하여 만든 두 다단 네트워크을 연결하는 역할을 한다. 베니스 네트워크를 구성하기 위하여 사용된 링크와 재할당 알고리듬에 의하여 사용된 링크는 서로 다르므로 전체의 라우팅 경로는 링크가 중첩되지 않는다.

V. 결 론

하이퍼큐브 구조를 갖는 MIMD 다중 프로세서 시스템에서는 반복적인 프로세서 할당과 할당해제로 인해, 이용 가능한 프로세서가 충분하여도 이들 프로세서가 새로운 작업에 필요한 크기의 서브큐브를 구성할 수 없는 프레그먼테이션이 발생한다. 본 논문에서는 PGC와 서브큐브 번지를 이용하여 회선교환방식 하이퍼큐브내의 프레그먼테이션을 제거하기 위한 작업이동 알고리듬을 제안하였다. 회선교환방식 하이퍼큐브에서의 작업이동 알고리듬은 작업이 할당된 서브큐브를 프레그먼테이션내에 있는 프리 서브큐브로 작업을 이동시킬 수 있는 각 노드간의 전용 라우팅 경로를 찾는 것이다.

본 논문에서 제안한 방법에 의해 설정된 두 서브큐브사이의 라우팅 경로는 유휴 링크로 이루어지며, 링크가 서로 중첩되지 않는 양 방향의 최단거리 경로이다. 따라서 하이퍼큐브내에서 이미 작업을 실행중인 다른 서브큐브는 방해하지 않고 프로세서를 재배치 할 수 있다. 또한, 경로 설정에서 시작점 서브큐브와 도착점 서브큐브 자체를 구성하는 링크는 사용하지 않는다. 이 링크를 사용하면 두 서브큐브를 베니스 네트워크 형태로 변환할 수도 있다. 변환된 네트워크를 이용하면 MIMD 하이퍼큐브에서 유용하게 사용될 수 있는, 두 서브큐브간의 1대 1 사상으로 표현되는 모든 통신 패턴에 대하여 경로를 설정할 수 있다.

참 고 문 헌

[1] M. Jeng and H. J. Siegel, "A

distributed management scheme for partitionable parallel computers," *Proc. 1989 Int'l Conf. Parallel Processing*, vol. II, pp. 57-64, Aug. 1989.

[2] Y. Saad and M. M Schutz., "Topologies properties of hypercubes," *IEEE Trans. Comput*, vol. C-37, pp. 867-872, July 1988.

[3] P. W. Purdom, Jr. and S. M. Stigler, "Statistical properties of the buddy system," *J. Ass. Comput. Mach*, vol. 17, pp. 683-697, Oct. 1970.

[4] M. S. Chen and K. G. Shin, "Processor allocation in an N-cube multiprocessor using Gray codes," *IEEE Trans. Comput*, vol. C-36, pp. 1396-1407, Dec. 1987.

[5] P. J. Chuang and N. F. Tzeng, "A fast recognition complete processor allocation strategy for hypercube computers," *IEEE Trans. Comput*, vol. 41, pp. 467-479, Apr. 1992.

[6] J. Kim, C. R. Das, and W. Lin, "A top-down processor allocation scheme for hypercube computers," *IEEE Trans. Comput*, vol. 2, pp. 20-40, Jan. 1991.

[7] A. Silberschatz, J. Peterson, and P. Galvin, Operating system concepts, Addison Wesley, pp. 229-274, 1991.

[8] M. S. Chen and K. G. Shin, "Task migration in hypercube multiprocessors," in *Proc. 16th Annu. Int. Symp. Comput. Architecture*, pp. 105-111, May 1989.

[9] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Trans. Comput*, vol. C-36, no. 5, pp. 547-553, May 1987.

[10] D. H. Linder and J. C. Harden, "An adaptive and fault tolerant wormhole routing strategy for k-ary n-cube," *IEEE Trans. Comput.*, vol 40, no. 1, pp. 2-12, Jan. 1991.

[11] G. I. Chen and T. H. Lai, "Constructing parallel paths between two subcubes," *IEEE Trans. Comput.*, vol. 41, No. 1, pp. 118-123, Jan. 1992.

[12] H. L. Chen and N. F. Tzeng, "Speedy task migration in hypercube systems," *Proc. 1994 Int'l Conf. Parallel Processing*, vol. III, pp. 74-78, Aug. 1994.

[13] F. T. Leighton, Introduction to parallel algorithms and architecture: Array, trees, hypercube, Morgan Kaufmann Publishers, pp. 392-472, 1992.

저 자 소 개

金 大 榮(正會員)
1992년 인하대학교 전자공학과 학사. 1994년 인하대학교 전자공학과 석사. 1994년 ~ 현재 현대전자(주) 컴퓨터기술전략부 근무. 관심분야는 컴퓨터 구조, Routing algorithm, Task-migration, 병렬 및 분산처리 등임.

崔 相 昉(正會員)
1981년 한양대학교 전자공학과 졸업. 1988년 University of Washington 석사. 1990년 University of Washington 박사. 1981년 ~ 1986년 금성정보통신(주) 근무. 1991년 ~ 현재 인하대학교 전자공학과 조교수. 관심분야는 컴퓨터 구조, 병렬 및 분산처리 시스템, Fault-tolerant computing 등임.