하이퍼큐브에서의 효과적인 프로세서할당 기법

손 유 익[†] · 남 재 열[††]

요 약

프로세서는 이용율의 최대화와 시스템 단편화의 최소화를 고려하여 들어오는 각 작업에 할당되어진다. 따라서 하이퍼큐브에서 프로세서를 효율적으로 할당하는 방법은 시스템 성능에 중요한 요인이 된다. 효율적인 프로세서 할당을 위해서는 필요한 크기의 서브큐브가 유용한지를 찾는 것과, 여러 개의 사용되지 않는 작은 서브큐브를 하나의 큰 서브큐브로 만들어 주는 것이 필요하다. 본 논문에서는, 사용 가능한 서브큐브를 표현하는 이진 트리를 얻기 위해 교환이 수행될 레벨과 파트너를 직접 결정하는 트리교환 알고리듬과 이를 이용한 할당방법에 관하여 언급한다. 제안된 알고리듬의 트리 탐색시간에 대한 복잡도는 $O\lceil n/2\rfloor \times 2^n$)으로서 기존의 다른 방법들과 비교하여 좋은 성능을 보인다.

An Efficient Processor Allocation Scheme for Hypercubes

Yoo Ek Son[†] · Jae Yeal Nam[††]

ABSTRACT

Processors must be allocated to incoming tasks in a way that will maximize the processor utilization and minimize the system fragmentation. Thus, an efficient method of allocating processors in a hypercube is a key to system performance. In order to achieve this goal, it is necessary to detect the availability of a subcube of required size and merge the released small cubes to form a larger ones. This paper presents the tree-exchange algorithm which determines the levels and partners of the binary tree representation of a hypercube, and an efficient allocation strategy using the algorithm. The complexity for search time of the algorithm is $O\lceil n/2\rfloor \times 2^n$) and it shows good performance in comparison with other strategies.

1. 서 론

최근 초고속 컴퓨터의 개발 추세에 따라 여러 종류의 병렬컴퓨터 구조에 대한 연구가 활발히 진행되어 왔으며 그중 하이퍼큐브는 좋은 구조적 특성으로 인해 많은 관심을 끌고 있다[1, 3-5, 9]. 하이퍼큐브 병렬컴퓨터에서 처리될 하나의 작업은 많은 수의 작은 작업으로 구성되며, 각 작업은 일정시간 동안 자신만의 서브큐브에서 처리되는 형태를 갖는다. 작업처리를 위해 필요한 서브큐브의 차원을 결정하는 것과 해당 차원의 서브큐브로의 할당문제가 하이퍼큐브에서의 프로세서 할당이 된다.

그러나 프로세서 할당에서 발생되는 서브큐브 탐색시간과 프래그먼트 문제는 전체 시스템 성능에 중대한 영향을 미치며 또한 적절한 크기의 사용 가능한 서브큐브를 인식하는 것이 쉬운 일이 아니다. 서브큐브 할당은 접근방법에 따라, 서브큐브의 사용가능성에 대한 정보를 나타내기 위해 리스트나 집합, 그래프 등을 이용하여 요구되는 서브큐브를 해당 리스트

† 정 회 원: 계명대학교 전자계산학과 교수
†† 정 회 원: 계명대학교 전자계산학과 전임강사
논문접수: 1995년 4월 3일, 심사완료: 1995년 7월 31일

나 집합에서 바로 찾는 Top-down 방법과, 모든 프로세서의 사용 가능성을 표현하기 위해 k 차원의 서브큐브가 요구되면 서브큐브를 이루는 2^k개의 할당비트를 조사하는 Bottom-up 방법으로 나눌 수 있다[15]. Top-down 방법은 Free list 방법[8], Heuristic processor allocation (HPA) 방법[15], Prime cube graph 방법[16], Dynamic binary tree 방법[19] 등이 있으며, 서브큐브 할당 과정은 비교적 간단하지만 할당을 해제하고 난 뒤에 사용 가능한 서브큐브를 큰 규모로 유지해야 하므로 할당해제 과정이 복잡해지는 단점이 있다. Bottom-up 방법은 Buddy 방법[2]과 Modified buddy 방법[6], Gray code (GC) 방법[2], Tree Collapsing (TC) 방법[13], Cube Coalescing (CC) 방법[20] 등이 있는데 이러한 방법들은 서브큐브의 사용 가능성에 대한 정보를 나타내기 위해 할당비트만 이용하므로 사용 가능한 서브큐브를 모두 인식하려면 일련의 할당비트를 재배열하는 과정, 즉 이진트리 생성 과정이 필수적으로 요구된다.

본 논문에서는 Bottom-up 방법을 이용하여 서브큐브를 탐색할 때 필요한 이진트리 생성 횟수를 감소시킴으로써 서브큐브 탐색시간을 줄일 수 있는 Tree Exchanging (TE) 방법을 제안하였다. 트리교환에 기초하는 이 방법에서는 사용 가능한 서브큐브를 나타내는 이진트리를 보다 빠르게 얻을 수 있는 최소한의 트리교환을 결정하여 불필요한 이진트리 생성 과정을 제거함으로써 서브큐브 탐색시간을 개선하였다.

2. 용어 및 문제 정의

n 차원 하이퍼큐브 Q_n은 2^n개의 노드를 가지며 각 노드는 n개 이웃노드로의 직접적인 통신채널이 되는 n개의 링크를 가진다. Q_n의 각 노드의 주소는 0에서 2^{n-1}까지의 정수를 이진수로 나타낸 것이다. Q_n 내의 k 차원의 서브큐브를 Q_k라 한다. 서브큐브 주소의 각 비트는 0과 1과 *로 나타내는데 이때 '*'를 *don't care symbol* 이라 한다. 예를 들어 Q5에서 노드 10001, 10011, 11001, 그리고 11011 을 갖는 2 차원의 서브큐브는 1*0*1로 표시된다. 그러므로 Q_n내에 존재하는 k 차원의 서로 다른 서브큐브의 수는 $C(n,k)\,2^{n-k}$이다.

파라미터 $\langle t_0, t_1,...,t_{n-1}\rangle$를 갖는 (n+1)-레벨의 이진트리를 T_n으로 표기한다. 파라미터는 {0, 1,...,n−1}의 순열로서 i번째 파라미터 요소인 t_i는 이진수로 쓰여진 프로세서 주소의 $(n-1-t_i)$번째 방향을 의미한다. Q_n을 표현하는 기본 이진트리는 $T_n\langle 0, 1,...,n-1\rangle$이며 PT_n으로 표기되고 PT_n의 단말노드는 왼쪽에서 오른쪽으로 0에서 부터 2^{n-1} 까지의 주소를 갖는다. 파라미터 $\langle t_0, t_1,...,t_{n-1}\rangle$와 $\langle t_0', t_1',...,t_{n-1}'\rangle$을 각각 가지는 이진트리 T_n와 T_n'는 $0 \le i \le n-k-1$ 일 경우에, $t_i = t_i'$ 이면 구하고자 하는 서브큐브가 있는 레벨 n-k에서 서브큐브에 해당하는 노드들은 서로 중복된다[13]. 예를 들어, n=4, k=2 일 때 이진트리 $T_4\langle 0, 1, 3, 2\rangle$을 기본이진트리 $PT_4\langle 0, 1, 2, 3\rangle$와 비교하면 $0 \le i \le 1$에서 $t_i = t_i'$ 이므로 2 차원 서브큐브에 해당하는 레벨 2에 있는 노드들이 서로 중복된다고 할 수 있다.

프로세서 할당비트는 Q_n에 속하는 2^n개 프로세서의 사용 가능성을 알아보기 위해 필요한 2^n개의 비트로서 그 값이 0이면 해당하는 프로세서가 아직 할당되지 않아 사용가능하고, 1이면 이미 할당되어 사용할 수 없음을 의미한다. 이진트리 T_n에서 정수 ℓ이 $0 \le \ell \le 2^{n-k+1}-1$ 일 때, $B_{n-k+1}(\ell)$의 자식노드가 사용가능하면 $B_{n-k+1}(\ell)$도 사용가능하다. 예를 들어 n=4이고 k=2일 때 자식노드 0000과 0001이 사용가능해야 노드 000*가 사용가능하다.

$B_k(i)$는 정수 i를 k 비트로 나타낸 이진수로서 T_n에서의 한 노드를 가리키며 $B_k(i)$의 p번째 파트너를 $B_k^p(i)$라 한다. 이진트리 T_n에서 노드 $a_{k-1}a_{k-2}...a_{\ell+1}a_\ell a_{\ell-1}...a_0$의 ℓ번째 파트너는 $a_\ell = 0$이면 $a_{k-1}a_{k-2}...a_{\ell+1}1a_{\ell-1}...a_0$이고 $a_\ell = 1$이면 없다. 〈표 1〉은 n=3일 때 각 노드들의 파트너를 나타낸 것이다. 즉, 0번째 비트 값이 서로 다른 노드는 0번째 파트너이고, 1번째 비트 값이 서로 다른 노드는 1번째 파트너이고, n번째

〈표 1〉 n = 3 일때 노드들의 파트너
〈Table 1〉 Partner nodes for n = 3

노드	0번째	1번째	2번째
000	001	010	100
001	undefined	011	101
010	011	undefined	110
011	undefined	undefined	111
100	101	110	undefined
101	undefined	111	undefined
110	111	undefined	undefined
111	undefined	undefined	undefined

비트 값이 서로 다른 노드는 n번째 파트너이다. 서로 다른 값을 갖는 비트의 위치가 작을수록 가까운 파트너이다.

3. TE 알고리듬

본 장에서는 이진트리 생성 횟수를 감소시킴으로써 서브큐브 탐색시간을 개선한 새로운 방법인 Tree Exchanging (TE) 방법을 제안한다. 기존의 방법들에서는 요구되는 서브큐브가 발견될 때까지 정해진 순서대로 이진트리를 탐색[2]하거나 생성[13]하도록 하였기 때문에 불필요한 과정이 포함되어 탐색시간이 길어지는 단점이 있었다. TE 방법에서는 기본이진트리에 있는 사용가능한 단말노드의 주소를 이용하여 교환이 필요한 레벨과 파트너만을 결정하도록 함으로써 보다 개선된 탐색시간동안 원하는 이진트리를 구할 수 있도록 한다.

3.1 기본 개념

트리교환에 기초하는 TE 방법에서는 교환될 레벨과 파트너의 범위를 다음과 같이 결정한다. Q_n에서 Q_k가 요구될 때, 교환이 일어날 레벨 L의 범위는 $n-k+1 \le L \le n$ 으로 하고, 이 범위내의 각 레벨마다 교환에 사용될 파트너의 범위는 〈표 2〉와 같이 가장 먼 n−k개의 파트너로 정한다. 또한 레벨 j에서 m번째 파트너와 교환을 수행한 후에 생성된 이진트리는 $m < j-1$ 이고 $0 < r < n-j$ 이고 $1 < q < n-1$ 일 때, 레벨 j+r에서 m+q번째 파트너와 1회 더 교환될 수 있도록 한다. 그러므로, 하나의 이진트리를 생성하기 위해 연속적으로 수행되는 노드교환의 최대 횟수는 min(k, n−k)회 인데 min(k, n−k)은 $k \approx n/2$ 일 때 [정의 1]에 따라 가장 큰 값 $\lceil n/2 \rceil$를 갖는다.

[정의 1] E_N^L은 이진트리 T_n의 레벨 L에서 N 번째 파트너와의 1 회 교환을 의미한다.

〈표 2〉 교환될 레벨과 파트너의 범위
〈Table 2〉 The levels and partner nodes to be exchanged

레벨	파트너
n−k+1	$1 \le N \le n-k$. (n−k개)
n−k+2	$2 \le N \le n-k+1$.(n−k개)
...	...
n	$k \le N \le n-1$.(n−k개)

〈알고리듬 1〉에 있는 B_Exchanging은 주어진 차원의 서브큐브가 발견될 때까지 〈표 2〉에 나타난 레벨에서 정해진 파트너들을 순서대로 교환하는 알고리듬이다. 〈알고리듬 1〉로 부터 [정리 1]을 얻을 수 있으며, 이를 위해서 [보조정리 1, 2, 3]을 보인다.

〈알고리듬 1〉 기본 알고리듬
〈Algorithm 1〉 Basic algorithm

```
B_Exchanging(tree, prolevel, num_neigh)

  Input  : 길이 n의 기본이진트리 tree, 노드교환 레벨 prolevel,
           각 레벨에서 교환되어야 하는 이웃노드의 갯수 num_neigh.
  Output : k차원의 서브큐브.

단계 1 : k = 0 이거나 k = n 이면 알고리듬 중지.
단계 2 : level = prolevel.
단계 3 : neighbor = level - num_neigh.
단계 4 : 노드교환으로 new_tree 생성.
단계 5 : new_tree에 원하는 서브큐브가 있으면 Return.
단계 6 : level < n 이고 neighbor < level-1 이면
         Basic(new_tree, level+1, level-neighbor-1)를 재귀 호출.
         new_tree에 원하는 서브큐브가 있으면 Return.
단계 7 : neighbor = neighbor + 1.
         neighbor ≤ level - 1 이면 단계 4로 간다.
단계 8 : level = level + 1.
         level ≤ n이면 단계 3으로 간다.
```

[보조정리 1] 이진트리 Tn이 파라미터 $\langle t_0, t_1, \ldots, t_{n-1} \rangle$을 가질 때 레벨 L에서 m번째 이웃노드와 교환 E_m^L이 일어난 후의 이진트리 T'_n의 파라미터 $\langle t_0', t_1', \ldots, t_{n-1}' \rangle$는 다음과 같이 결정된다.

1) $i = L-1$ 일 때, $t_i' = t_{i-m}$, $t_{i-m}' = t_i$,

2) $i \neq L-1$ 일 때, $t_i' = t_i$.

증명 : 노드교환후 비롯되는 $\langle t_0', tt_1', \ldots, t_{n-1}' \rangle$는 $\langle t_0, t_1, \ldots, t_{n-1} \rangle$에서 t_{L-1}과 t_{L-1-m}을 서로 교환한 것과 같다. 예를 들어, 기본이진트리 $T_4\langle 0, 1, 2, 3 \rangle$에서 E(3, 2)가 수행된다면 $T_4\langle 2, 1, 0, 3 \rangle$이 될 것이다.□

[보조정리 2] 기본 알고리듬에서, 트리 T_n에서 새로운 이진트리를 생성하기 위해 연속되는 노드교환 $E(i_1, l)$, $E(i_2, l+m), \ldots, E(i_j, l+(j-1)m), \ldots, E(i_p, l+(p-1)m)$을 하는 과정에서 생성되는 $T_n^{(1)}, T_n^{(2)}, \ldots, T_n^{(j)}, \ldots, T_n^{(p)}$는 $n-k+1 \le i_j \le n$, $1 < m < n-1$이므로 레벨 n-k에서 서로 중복되지 않는다.

증명 : $n-k+1 \le i_j \le i_{j+1} \le n$ 일 때, 이진트리 $T_n^{(j-1)}\langle t_0^{j-1}, t_1^{j-1}, \ldots, t_{n-1}^{j-1} \rangle$에서 $E(i_j, 1+(j-1)m)$의 결과로 생성되는 $T_n^{(j)}\langle t_0^j, t_1^j, \ldots, t_{n-1}^j \rangle$는 $T_n^{(j-1)}$의 i_{j-1}번째 파라미터와 $i_j+(j-1)m$번째 파라미터를 교환한 것과 같다. 또한 $T_n^{(j)}\langle t_0^j, t_1^j, \ldots, t_{n-1}^j \rangle$에서 $E(i_{j+1}, 1+jm)$의

결과로 생성되는 $T_n^{(j+1)}\langle t_0^{j+1}, t_1^{j+1},...,t_{n-1}^{j+1}\rangle$는 $T_n^{(j)}$의 i_{j+1}번째 파라미터와 $i_{j+1}+jm$번째 파라미터를 교환한 것과 같다. 이때 i_j-1를 a, $i_j+(j-1)m$를 b, $i_{j+1}-1$를 c, $i_{j+1}+jm$를 d라 하면, $1<m<n-1$이기 때문에 항상 $0\le a<c\le n-k-1$이고 $n-k\le d<b\le n$이 되므로 t_i^{j+1}이다. 그러므로 보조정리 1에 따라 서로 중복되지 않는다.□

[보조정리 3] 기본 알고리듬에서, u와 v가 서로 다른 상수인 경우 서로 다른 두번의 연속되는 노드교환 $E(i_1, u), E(i_2, u+m),..., E(i_p, u+(p-1)m)$과 $E(i_1', v), E(i_2', v+m),..., E(i_s', v+(s-1)m),..., E(i_q', v+(q-1)m)$으로 생성되는 트리의 집합 $\{T_n^{(1)}, T_n^{(2)},..., T_n^{(j)},..., T_n^{(p)}\}$와 $\{T_n'^{(1)'}, T_n'^{(2)'},..., T_n'^{(\omega)'},..., T_n'^{(q)'}\}$는 $1\le p, q\le \min(k, n-k)$이고 $n-k+1\le i_j, i_s'\le n$ 일때, 각 집합에서 비롯되는 하나의 트리 T_n과 T_n'은 레벨 $n-k$에서 서로 중복되지 않는다.

증명: $u\neq v$이므로 각 집합의 처음 트리인 $T_n^{(1)}$과 $T_n'^{(1)'}$은 서로 중복되지 않고 $E(i_j, u+(j-1)m)$와 $E(i_s', v+(s-1)m)$에서 $u+(j-1)m\neq v+(s-1)m$이다. 그러므로 보조정리 1에 따라 집합 $\{T_n^{(1)}, T_n^{(2)},..., T_n^{(j)},..., T_n^{(p)}\}$와 $\{T_n'^{(1)'}, T_n'^{(2)'},..., T_n'^{(\omega)'},..., T_n'^{(q)'}\}$에서 하나의 트리 T_n과 T_n'은 레벨 $n-k$에서 서로 중복되지 않는다.□

[정리 1] 기본 알고리듬을 이용하여 서브큐브를 탐색할 경우 서로 중복되지 않는 이진트리를 $C(n, k)-1$개 생성할 수 있다.

증명: {1, 2, 3,...,m}은 일련의 m개 숫자라고 하면 다음의 두 operator가 정의될 수 있다.

$$\phi(m)=\{1, 2, 3,..., m-1\}$$
$$\phi^r(m)=\phi(m-r)$$
$$\wedge(\phi(m))=\wedge(\{1, 2, 3,..., m-1\})=\sum_{i=1}^{m-1} i$$

기본 알고리듬의 노드교환에서 i회의 연속적인 노드교환으로 생성되는 이진트리의 수를 F_k^i라고 하면 다음과 같다.

$$F_k^1=k\times(n-k)=C(k, 1)\times C(n-k, 1)$$
$$F_k^2=\wedge(\phi(k))\times\wedge(\phi(n-k))$$
$$=(1+2+...+k-1)\times(1+2+...+n-k-1)$$
$$=C(k, 2)\times C(n-k, 2)$$
$$F_k^3=\wedge(\phi^2(k))\times\wedge(\phi^2(n-k))$$
$$=(1+2+...+k-2)\times(1+2+...+n-k-2)$$
$$=C(k-1, 2)\times C(n-k, 2)$$
$$=C(k, 3)\times C(n-k, 3)$$
$$F_k^4=\wedge(\phi^3(k))\times\wedge(\phi^3(n-k))=C(k, 4)\times C(n-k, 4)$$
$$=(1+2+...+k-2)\times(1+2+...+n-k-3)$$
$$=C(k-2, 2)\times C(n-k-2, 2)$$
$$=C(k-1, 3)\times C(n-k, 3)$$
$$=C(k, 4)\times C(n-k, 4)$$

그러므로,

$$F_k^i=\wedge(\phi^{i-1}(k))\times\wedge(\phi^{i-1}(n-k))=C(k, i)\times C(n-k, i) \quad (1)$$

기본 알고리듬의 수행중 모든 노드교환에 의해 생성되는 이진트리의 수 T_k가 $C(n, k)-1$개 임을 귀납법을 이용해 증명할 수 있다.

Base: $T_1=F_1^1=n-1=C(n, 1)-1$이다.

Induction: 만약 T_{m-1}이 $C(n, m-1)-1$개 이라면,

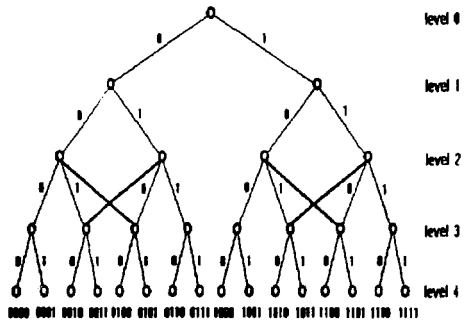
$$F_{m-1}^1+F_{m-1}^2+...+F_{m-1}^{m-1}$$
$$=C(m-1, 1)\times C(n-m+1, 1)+C(m-1, 2)$$
$$\times C(n-m+1, 2)+...$$
$$+C(m-1, m-1)\times C(n-m+1, m-1)$$
$$=C(n, m-1)-1$$

이 되고, 그러면

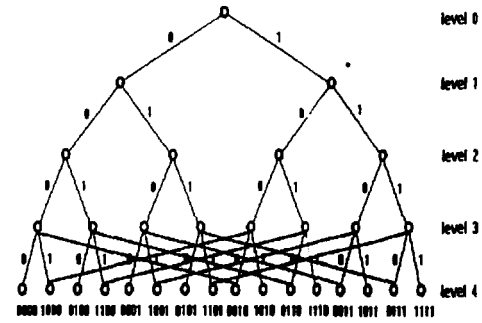
$$T_m=F_m^1+F_m^2+...+F_m^{m-1}+F_m^m$$
$$=C(m, 1)\times C(n-m, 1)+C(m, 2)\times C(n-m, 2)$$
$$+...+C(m, m-1)\times C(n-m, m-1)$$
$$+C(m, m)\times C(n-m, m)$$
$$=C(n, m)-1$$

따라서 기본 알고리듬은 노드교환을 통하여 $C(n, k)-1$개의 새로운 이진트리를 생성한다. 또한 생성된 $C(n, k)-1$개의 이진트리들이 서로 중복되지 않음을 보조정리 2, 3으로부터 알 수 있다.□

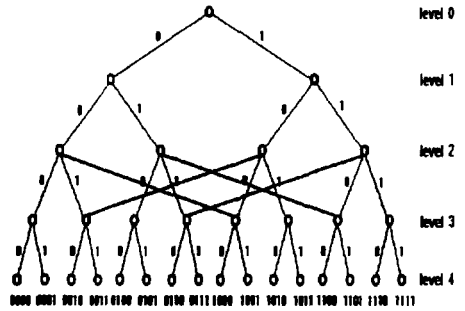
예를 들어, 하이퍼큐브의 차원 $n=4$ 이고 요구되는 서브큐브의 차원 $k=2$ 일때 교환에 이용될 레벨의 범



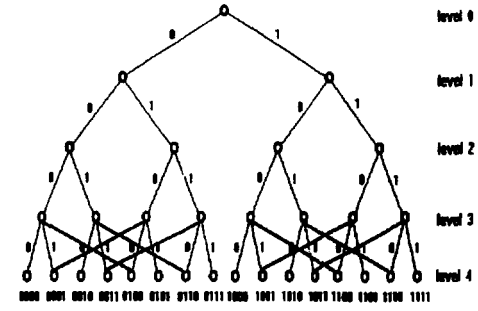
a) 기본이진트리에서 E(3, 1)의 과정



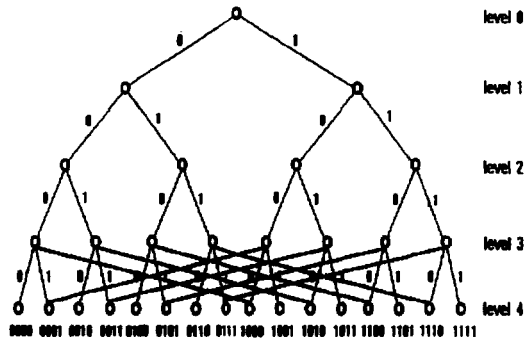
b) a의 결과트리에서 E(4, 3)의 과정



c) 기본이진트리에서 E(3, 2)의 과정



d) 기본이진트리에서 E(4, 2)의 과정



e) 기본이진트리에서 E(4, 3)의 과정

(그림 1) Exchange 알고리듬을 이용한 노드교환
(Fig. 1) The node exchange using the exchange algorithm

위는 $3 \le L \le 4$, 각 레벨당 1회 교환하는 파트너의 갯수는 $n-k=2$, 새로운 이진트리 생성을 위한 연속 노드교환의 최대 횟수는 $k=2$ 이다. 구해진 레벨과 파트너의 범위에 따라 수행되는 노드교환 과정은 〈표 3〉과 같다. ① ② ③...은 노드교환이 수행되는 순서를 의미한다.

〈표 3〉 $n=4, k=2$ 일때 기본 알고리듬을 이용한 노드교환
〈Table 3〉 Node exchanges using the Basic algorithm for $n=4, k=2$

기본이진트리	1회	2회
$T_4\langle 0, 1, 2, 3\rangle$	① $E(3, 1)\ T_4\langle 0, 2, 1, 3\rangle$	② $E(4, 3)\ T_4\langle 3, 2, 1, 0\rangle$
	③ $E(3, 2)\ T_4\langle 2, 1, 0, 3\rangle$	
	④ $E(4, 2)\ T_4\langle 0, 3, 2, 1\rangle$	
	⑤ $E(4, 3)\ T_4\langle 3, 1, 2, 0\rangle$	

(그림 1)에서는 〈표 3〉의 모든 노드교환 과정을 이진트리로 보이고 있다. a)와 c), d), e)는 기본이진트리 $T_4\langle 0, 1, 2, 3\rangle$에서 노드교환이 일어나는 과정이고 b)는 (a)의 결과 $T_4\langle 0, 2, 1, 3\rangle$에서 노드교환을 하는 과정이다.

3.2 교환 알고리듬

본 절에서는 정해진 순서에 따라 수행되는 기본 알고리듬을 수정하여 불필요한 교환 과정을 제거하고 꼭 필요한 레벨과 파트너만을 정확히 선택하여 원하는 이진트리를 바로 결정할 수 있도록 하는 교환 알고리듬을 제안한다. 교환 알고리듬은 〈알고리듬 2〉에 나타나 있다. 단계 1은 노드교환에 이용할 레벨과 파트너를 결정하는 과정으로서 기본이진트리의 단말노드를 왼쪽에서 오른쪽으로 찾아 할당비트 값이 0인 노드들의 주소를 〈표 2〉에서 구해진 범위내의 레벨에 각각 적용하여 파트너를 결정하는 과정이며, 단계 2는 비록 노드교환이 수행될 레벨과 파트너가 단계 1에서 구해졌더라도 충분한 갯수의 프로세서가 없는 경우에는 알고리듬을 종료하도록 하는 과정이며, 단계 3은 단계 1에서 구해진 레벨과 이웃노드를 이용하여 실제로 노드교환을 수행하는 과정이다. 특히, 단계 1에서는 사용가능한 단말노드들이 요구되는 서브큐브를 형성할 수 있는지 알아보기 위해, 작업에 할당되지 않은 프로세서 즉, 이진트리에서 할당비트 값이 0인 단말노드들의 주소를 이용하여 레벨과 파트너를 결정하도록 한다. 이 과정은 기본이진트리에서 단말노드를 왼쪽에서 오른쪽으로 탐색하여 가면서 할당비트의 값이 0인 노드를 찾아 처음으로 발견되는 사용가능한 단말노드의 주소가 0이 아니면 0으로 치환하고 그 이후로 발견되는 노드들의 주소도 그에 따라 상대화시킨다. 노드 A의 로그값의 정수부분을 취하는 과정을 함수 log_i(A)로 표시한다면, 상대화된 주소의 값은 log_i()이 된다. 이것은 이웃노드간의 상대적인 거리를 의미하므로 구해진 log_i() 값이 0보다 큰 경우를 이용하여 노드교환에 사용될 레벨과 이웃노드를 결정하게 된다.

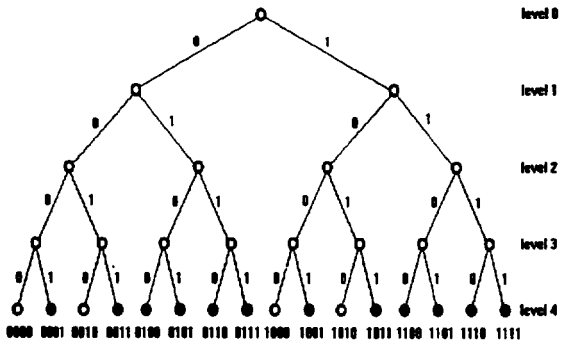
(그림 2)는 Exchanging 알고리듬이 적용되는 과정을 보여준다. (그림 2)의 a)와 같은 이진트리에서 Q_2를 찾아내기 위해 Exchanging 알고리듬이 적용되는 과정을 살펴보면 할당비트가 0인 단말노드는 4, 6, 12, 14임을 알 수 있다. 그 노드들의 주소의 log_i()값을 구하여 그 값이 0 이상 인 것만 노드교환에 이용한다. 교환될 레벨과 파트너는 〈표 3〉의 범위 내에 있어야 하므로 여기서 선택되는 레벨과 파트너를 결정한 후, 각 레벨마다 부여된 log_i() 값으로부터 레벨 n과 자신의 레벨과의 차이에 의해 결정된다. 즉,

4 : log_i(0) = 0,
6 : log_i(2) = 1 ==〉 level 3 ==〉 neighbor node 0
12 : log_i(8) = 3 ==〉 level 4 ==〉 neighbor node 3
14 : log_i(10) = 3 ==〉 level 4 ==〉 neighbor node 3

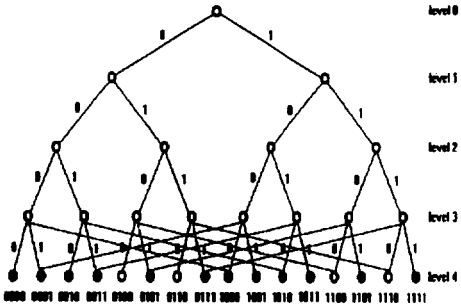
이다. 그러므로 수행되어야 할 노드교환 식은 E_3^4과 E_3^4인데 E_0^3은 이웃노드가 0이므로 수행할 필요가 없고 E_3^4만 수행하면 된다.

〈알고리듬 2〉 교환 알고리듬
〈Algorithm 2〉 Exchange algorithm

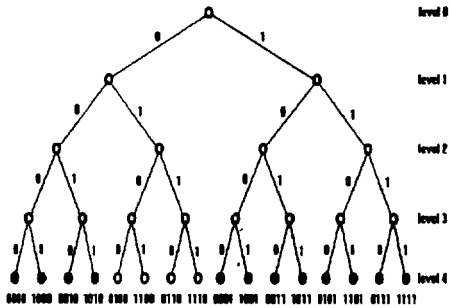
```
T_Exchange(tree) :
    input   : 기본이진트리 tree.
    output  : k 차원의 서브큐브.
    comment : log_i(a)는 단말노드 a의 로그 값의 정수부분을 취하는 함수.

단계 1 : 이진트리의 단말노드를 왼쪽에서 오른쪽으로 탐색하면서 할당비트 값이
         0인 노드의 log_i() 값을 구해 교환에 사용될 레벨과 이웃노드를 결정한다.
단계 2 : 할당비트가 0인 노드의 갯수가 2k보다 작으면 알고리듬 종지.
단계 3 : 서브큐브가 발견될 때까지 구해진 모든 레벨과 이웃노드에서 노드교환을
         수행한다.
단계 4 : 서브큐브가 발견되면 Return하고, 아니면 알고리듬 종지.
```



a) 일부 단말노드가 사용가능한 기본이진트리 PT_4



b) a의 이진트리에 E(4, 3)을 적용



c) b의 결과 이진트리

(그림 2) Exchanging 알고리듬을 이용한 노드교환
(Fig. 2) Node exchange using the exchanging algorithm

3.3 할당 알고리듬

〈알고리듬 3〉는 Exchanging 알고리듬을 이용한 TE 방법의 서브큐브 할당과 제거 과정이다. TE방법은 크게 기본이진트리에서 buddy 방법을 이용하여 서브큐브를 탐색하는 과정과 Exchanging 알고리듬의 수행 후 새로운 이진트리에서 buddy 방법을 이용하는 과정으로 구성된다.

〈알고리듬 3〉 TE 할당 알고리듬
〈Algorithm 3〉 TE allocation algorithm

```
할당 알고리듬
  단계 1 : 태스크를 수용할 수 있는 서브큐브의 크기 k를 정한다.
           k = 0 이거나 n 이면 할당 알고리듬을 종단한다.
  단계 2 : Buddy 방법과 같은 방법으로 트리의 레벨 n-k에서 서브큐브를 찾는다.
           사용가능한 서브큐브가 발견되면 단계 4로 간다.
  단계 3 : T_Exchanging 알고리듬을 수행한다. 사용가능한 서브큐브가 발견되면
           다음 단계로 가고 그렇지 않으면 단계 5로 간다.
  단계 4 : 발견된 서브큐브에 해당하는 할당비트를 1로 바꾼 다음, 그 서브큐브에
           태스크를 할당하고 할당 알고리듬을 끝낸다.
  단계 5 : 태스크를 큐에 넣고 사용가능한 서브큐브가 생길 때까지 기다린다.

제거 알고리듬
    할당된 태스크의 수행이 끝난 서브큐브의 할당비트를 0으로 바꾼다.□
```

[정리 2] TE 알고리듬을 통해 요구되는 차원의 서브큐브를 완전 인식하는데 걸리는 탐색시간은 $O(\lceil n/2 \rceil \times 2^n)$ 이다.

증명 : TE 할당방법의 단계 2에서 탐색시간 $O(2^n)$ 동안 k 차원의 서브큐브를 2^{n-k}개를 인식할 수 있다. 단계 3에서는 TE 알고리듬을 이용한다. TE 알고리듬의 단계 1은 노드교환이 수행될 레벨과 이웃노드를 결정하는 과정으로서 2^n개의 단말노드의 주소를 이용하

여 동작하므로 탐색시간이 $O(2^n)$이다. 단계 3은 단계 1에서 구해진 레벨과 이웃노드를 이용하여 실제로 노드교환을 수행하는 과정으로서 최악의 경우 $\lceil n/2 \rfloor$ 회의 연속 노드교환을 하게 되므로 각 노드교환마다 2^n개의 할당비트를 교환하게 되어 전체 탐색시간은 $O(\lceil n/2 \rfloor \times 2^n)$이 된다. 단계 2에서는 탐색시간 $O(2^n)$ 동안 k 차원의 서브큐브를 2^{n-k}개를 인식할 수 있다. TE 알고리듬을 이용하여 이진트리를 생성하는 단계 3은 TE 알고리듬이 탐색하는 이진트리들 중에서 가장 적당한 이진트리를 선택하는 과정이므로 $(C(n, k)-11)$ 2^n개의 서브큐브를 인식할 수 있다. 즉, $(C(n, k)-1)$ 2^{n-k}개의 서로 다른 서브큐브를 인식한다. 그러므로 TE 방법에서는 n 차원의 하이퍼큐브에 존재하는 k 차원의 서브큐브 $O(C(n, k) \times 2^n)$개를 탐색시간 $O(\lceil n/2 \rfloor \times 2^n)$동안 완전 인식한다.□

4. 성능분석 및 비교

요구되는 차원의 서브큐브를 완전히 인식하는 방법으로서 기존의 MGC, TC, CC 방법들과 본 논문에서 제안한 TE 방법을 탐색시간 복잡도 면에서 성능을 비교 분석한다.

MGC 방법은 논문 [13]에서 언급된 바와같이 $C(n, \lceil n/2 \rfloor)$개의 그레이코드를 미리 구하여 저장해 두어야 하므로 $(C(n, \lceil n/2 \rfloor) \times 2 \times 2^n)$ 만큼의 메모리가 항상 필요하다. 2^n개 프로세서를위한 하나의 GC를 만드는 시간을 $\theta(2^n)$이라 하면 $C(n, \lceil n/2 \rfloor)$개의 다중 GC를 만들기 위해서 $\theta(C(n, \lceil n/2 \rfloor \times 2^n)$의 시간이 따로 필요하며 요구되는 서브큐브의 차원에 관계없이 구해진 모든 그레이코드를 탐색해야 하므로 탐색시간이 $O(C(n, \lceil n/2 \rfloor) \times 2^n)$이다. 또한 하이퍼큐브의 차원이 변경되었을 경우에는 $C(n, \lceil n/2 \rfloor)$개의 그레이코드를 새로 결정해야하며 하이퍼큐브의 차원이 커질수록 적절한 그레이코드를 모두 구하는 것이 쉽지 않은 단점들이 있다.

TC 방법은 기본이진트리의 단말노드의 주소와 할당여부를 위해 $O(2 \times 2^n)$의 메모리가 필요하다. 한 레벨에서 노드를 교환하는데 걸리는 시간을 $O(2^n)$이라고 하면, TC 방법에서는 새로운 이진트리 생성을 위해 레벨 i에서의 Tree Collapsing을 하는데, 1회의 collapsing을 위해서는 레벨 $i+2$부터 레벨 n까지 모든 레벨마다 노드교환을 해야 하므로 노드교환이 $n-i-2$회 수행되어 이진트리 생성 시간이 길어지는 단점이 있다. TC 방법의 전체적인 복잡도는 $O(C(n, k) \times 2^n)$이다.

CC 방법은 coalescing 될 기본이진트리를 위해 $O(2 \times 2^n)$의 메모리가 필요하다. 이 방법에서는 서브큐브를 형성하는데 필요 없는 프로세서들을 coalescing을 통해 여과시킴으로써 불필요한 이진트리 생성 횟수를 상당히 감소시켰다. 그러나 서브큐브가 발견될 때까지 기본이진트리의 각 레벨에서 coalescing이 수행되므로 전체 탐색시간이 $O(n \times 2^n)$이 된다.

제안된 TE 방법에서는 하이퍼큐브를 표현하는 기본이진트리의 2^n개 프로세서의 주소와 할당여부를 나타내기 위해 2×2^n 크기의 배열이 필요하다. 정리 2에 따르면 사용 가능한 서브큐브를 나타내는 이진트리들을 탐색하기 위해 $O(\lceil n/2 \rfloor \times 2^n)$의 시간이 걸림을 알 수 있다.

따라서 본 논문에서 제안한 TE 방법을 기존의 다른 방법들과 비교해 볼 때, 사용되는 메모리의 용량이 크고 할당과정에서의 정적인 특성으로 인한 단점이 나타나는 MGC 방법이나, 이진트리 생성 횟수가 많은 TC 방법이나 CC 방법에 비해 성능이 좋음을 알 수 있다. 〈표 4〉에서 각 방법의 복잡도를 보여준다.

〈표 4〉 복잡도 비교

〈Table 4〉 Comparison of complexity

방법	서브큐브 수	메모리	탐색 시간
Buddy	2^{n-k}	$\theta(2^{n+1})$	$O(2^n)$
MGC	$C(n, k) \times 2^{n-k}$	$\theta(C(n, \lceil n/2 \rfloor) \times 2^{n+1}$	$O(C(n, \lceil n/2 \rfloor) \times 2^n$
TC	$C(n, k) \times 2^{n-k}$	$\theta(2^{n+1})$	$O(C(n, k) \times 2^n$
CC	$C(n, k) \times 2^{n-k}$	$\theta(2^{n+1})$	$O(n \times 2^n)$
TE	$C(n, k) \times 2^{n-k}$	$\theta(2^{n+1})$	$O(\lceil n/2 \rfloor \times 2^n)$

5. 결 론

병렬처리를 위해 하이퍼큐브에 도착하는 여러 작업들은 적절한 서브큐브에 할당되어야 한다. 그러나 적절한 크기의 사용 가능한 서브큐브를 인식하는 일이 쉽지 않으며, 더우기 서브큐브 탐색시간 개선과 프래그먼트 문제해결을 위한 효율적인 프로세서 할

당방법은 전체 시스템 성능에 중대한 영향을 갖는다고 할 수 있다.

본 논문에서는 이를 위해 Tree Exchanging 기법을 이용하여 주어진 차원의 서브큐브를 표현하는 이진트리를 동적으로 생성하는 새로운 할당알고리듬을 제시하였다. 이 알고리듬은 요구되는 서브큐브를 표현하는 이진트리를 생성하기 위해 사용 가능한 프로세서의 주소를 이용하여 교환될 레벨과 파트너를 정확하게 결정하도록 하였으며 수행된 교환은 서로 중복되지 않으면서 주어진 차원의 서브큐브를 완전히 인식할 수 있다. 제시된 방법은 서브큐브 탐색시간이 $O(\lceil n/2 \rfloor \times 2^n)$이 되어 GC로 이루어진 이진트리를 미리 만들어 저장해 두는 MGC 방법보다 메모리를 적게 차지하고, 서브큐브 탐색시간이 $O(C(n, k) \times 2^n)$인 TC 방법이나 $O(n \times 2^n)$인 CC 방법에 비해 서브큐브 탐색시간이 작은 장점을 갖는다.

참 고 문 헌

[1] [1] J. P. Hayes, R. N. Mudge, Q. F. Stout, S. Colley, and J. Palmer, "Architecture of a Hypercube Supercomputer," *Proc. 1986 Int'l Conf. Parallel Processing*, pp. 653-660, Aug. 1986.

[2] M. Chen and K. G. Shin, "Processor Allocation in an N-Cube Multiprocessor Using Gray Codes," *IEEE Trans. on Comp.*, vol. C-36, Dec. 1987.

[3] Y. Saad and M. H. Schultz, "Topological Properties of Hypercubes," *IEEE Trans. on Comp.*, vol. 37, no. 7, July 1988.

[4] C. T. Ho and S. L. Johnson, "On the Embedding of Arbitrary meshes in Boolean Cubes," *Proc. 1987 Int'l Conf. Parallel Processing*, pp. 188-191, Aug. 1987.

[5] S. K. Chen, C. T. Liang, and W. T. Tsai, "Loops and Multi-dimensional Grids on Hypercubes: Mapping and Reconfiguration Algorithms," *Proc. 1988 Int'l Conf. Parallel Processing*, pp. I315-322, Aug. 1988.

[6] A. Al-Dhelaan and B. Bose, "A New Strategy for Processors Allocation in an N-Cube Multiprocessors," *Proc. Phoenix Conf. Comp. and Comm.*, pp. 114-118, Mar. 1989.

[7] J. Jang, S. Choi and W. Cho, "A New Approach to processor Allocation and Task Migration in an N-cube Multiprocessor," *Proc. Supercomputing'89*, pp. 314-325, Nov. 1989.

[8] J. Kim, C. R. Das, and W. Lin, "A Processor Allocation Scheme for Hypercube Computers," *Proc. 1989 Int'l Conf. Parallel Processing*, pp. II231-238, Aug. 1989.

[9] C. Y. Chen and Y. C. Chung, "Embedding Networks with Ring Connection in Hypercube Machines," *Proc. 1990 Int'l Conf. Parallel Processing*, pp. III327-334, Jun. 1990.

[10] M. Chen and K. G. Shin, "Subcube Allocation and Task Migration in Hypercube Multiprocessors," *IEEE Trans. on Comp.*, vol. 39, no. 9, Sep. 1990.

[11] M. Y. Chan and S. J. Lee, "Subcube recognition, Allocation/Deallocation and Relocation in Hypercubes," *Proc. of Second Symposium on Parallel and Distributed Processing*, pp. 87-93, Dec. 1990.

[12] S. T. Tan and D. Du, "On subcube allocation and relinquishment schemes for hypercube connected multiprocessor," *1990 Int'l Conf. on Parallel Processing*, pp. I571-I572, 1990.

[13] P. J. Chuang and N. F. Tzeng, "Dynamic Processor Allocation in Hypercube Computers," *Proc. of 17th Annual International Symposium on Computer Architecture*, pp. 40-49, 1990.

[14] C. H. Huang and J. Y. Juang, "A Partial Compaction Scheme for Processor Allocation in Hypercube Multiprocessors," *Proc. 1990 Int'l Conf. Parallel processing*, pp. I-211, May. 1990.

[15] S. Y. Yoon, O. Kang, H. Yoon, S. R. Maeng, and J. W. Cho, "A Heuristic processor Allocation Strategy in Hypercube Systems," *Proc. of Third Symposium on Parallel and Distributed Processing*, pp. 574-581, Dec. 1991.

[16] H. Wang and Q. Yang, "Prime Cube Graph Approach for Processor Allocation in Hypercube Multiprocessors," *Proc. 1991 Int'l Conf. Parallel Processing*, pp. I25-32, Aug. 1991.

[17] G. I. Chen and T. H. Lai, "Constructing Parallel Paths Between Two Subcubes," *IEEE Trans. on Comp.*, vol. 41, no. 1, Jan. 1992.

[18] Y. Chang and L. N. Bhuyan, "Parallel Algorithms for Hypercube Allocation," *7th International Parallel Processing*, pp. 105-112, Aug. 1993.

[19] D. D. Sharma and D. K. Pradhan, "Fast and Efficient Strategies for Cubic and Non-cubic Allocation in Hypercube Multiprocessors," *Proc. 1993 Int'l Conf. Parallel Processing*, pp. I118-127, Aug. 1993.

[20] G. Kim and H. Yoon, "A Processor Allocation Strategy using Cube Coalescing in Hypercube multicompyters," *Proc. of Fifth Symposium on Parallel and Distributed Processing*, pp. 596-605, Dec. 1993.

손 유 익

1952년 6월 12일생
1976년 경북대학교 전자공학과 졸업 (학사)
1979년 경북대학교 대학원 전자공학과(공학석사)
1990년 경북대학교 대학원 전자공학과 (공학박사)
1990년~1991년 미국 오레곤주립대 연구교수
1979년~1984년 한국전자기술연구소 정보시스템연구실 선임연구원
1984년~현재 계명대학교 전자계산학과 근무, 현재 교수로 재직
관심분야:병렬컴퓨터구조 및 알고리듬, Interconnection Networks

남 재 열

1960년 8월 12일생
1983년 2월 경북대학교 전자공학과(공학사)
1985년 2월 경북대학교 대학원 전자공학과(공학석사)
1991년 8월 University of Texas at Arlington(UTA) 전기공학과(공학박사)
1985년 5월~1987년 7월 한국전자통신연구소 연구원
1988년 1월~1991년 8월 UTA 연구조교
1991년 9월~1995년 2월:한국전자통신연구소 선임연구원
1993년 7월~1995년 2월:MPEG-Korea 의장
1995년 3월~현재 계명대학교 공과대학 전자계산학과 전임강사
주관심분야:영상신호처리, 영상통신, 멀티미디어 통신