손상된 하이퍼큐브상의 메세지 라우팅 알고리즘

공 헌 택[†] · 우 진 운[††]

요 약

하이퍼큐브 노드들의 커뮤니케이션은 메세지 라우팅에 의하여 이루어진다. 그러므로 효율적인 메세지 라우팅은 하이퍼큐브 병렬컴퓨터의 성능에 매우 중요한 요소가 된다. 그러나 하이퍼큐브 노드들은 하드웨어 혹은 소프트웨어상의 문제로 인하여 부분적으로 결함이 발생할 수 있는데, 이를 손상된 하이퍼큐브라 한다. 신뢰성이 높은 하이퍼큐브 시스템은 이러한 문제점을 극복해야만 한다. 손상된 하이퍼큐브상에서 신뢰성을 향상시키기 위한 방법 중의 하나는 오류회복기능을 갖는 메세지 라우팅 알고리즘을 사용하는 것이다. 본 연구에서는 독립된 경로를 이용하여 가능한 최단거리를 갖는 메세지 라우팅 알고리즘을 제안하며, 이 알고리즘의 성능을 분석하기 위하여 시뮬레이션을 통해 평가한다.

Message Routing Algorithm on an Injured Hypercube

Heon Taek Kong[†] · Jin Woon Woo[††]

ABSTRACT

Communications on hypercube nodes are done by explicit message routing. So efficient message routing is very important for the performance of hypercube multicomputers. However, hypercube nodes can be faulty due to hardware and/or software problems, which is called an injured hypercube. A reliable hypercube system should tolerate the problems. One of the methods to enhance reliability on injured hypercube is to use fault-tolerant message routing algorithms. In this paper, we propose a message routing algorithm with possible shortest distance using disjoint paths. To analyze the performance, the algorithm is simulated and evaluated.

1. 서 론

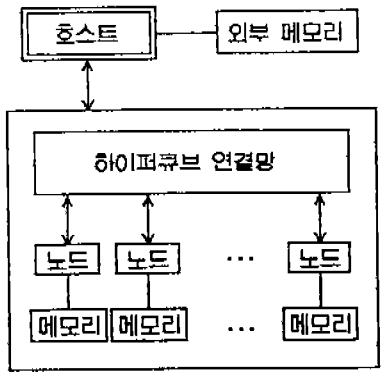
최초의 컴퓨터인 ENIAC 이래로 컴퓨터의 성능은 비약적인 발전을 거듭하여 왔다. VLSI 기술의 발전과 더불어 하드웨어 가격의 지속적인 하락은 독립된 처리 능력을 갖는 여러 개의 프로세서들을 보유한 병렬컴퓨터의 출현을 가져왔다[1, 2, 3].

하이퍼큐브 병렬컴퓨터는 다양한 알고리즘의 병렬 수행을 효과적으로 처리할 수 있는 구조적인 특성 때문에 다양한 응용분야에 도입되고 있다. 일기예보, 3차원 모델링, 원격탐사, 실시간 영상처리 등의 컴퓨터 응용분야는 재래식 순차 컴퓨터의 처리능력을 초월하는 계산빈도와 계산시간을 요구하는데 이러한 분야에 Ncube, Intel iPSC, FPS T Series 등과 같은 상업

* 이 논문은 한국과학재단 핵심전문연구(과제번호:931-0900-045-1) 연구비 지원에 의하여 연구되었음.
† 정 회 원:국립천안공업전문대학 전자계산과 조교수
†† 정 회 원:단국대학교 전산통계학과 부교수
논문접수:1995년 9월 4일, 심사완료:1996년 1월 22일

용 하이퍼큐브 병렬컴퓨터가 사용되고 있다.

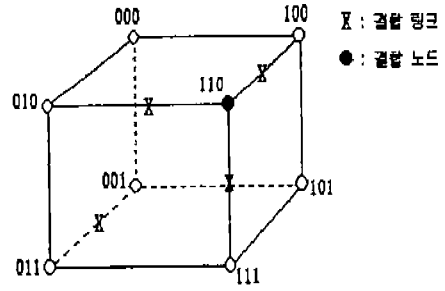
하이퍼큐브 병렬컴퓨터는 (그림 1)과 같이 로컬 및 외부메모리를 갖는 호스트(host)와 로컬 메모리만을 갖는 하이퍼큐브 프로세서(processor) 혹은 노드(node)들로 구성된다. 각 노드들은 자신의 CPU를 가지고 있어 독립적으로 처리할 수 있는 능력을 보유하며, 하이퍼큐브 구조의 연결망(interconnection network)을 통하여 다른 노드들과 커뮤니케이션을 수행한다. 그리고 $N=2^n$개의 노드들이 존재할 때 각 노드들은 0에서 2^n-1 까지의 n-비트 이진주소를 가지며, 두 개의 인접한 노드들은 이진주소에서 정확히 한 비트가 서로 다른 성질을 갖는다. 이를 n-차원 이진큐브(binary cube)라 한다[4].



(그림 1) 하이퍼큐브 병렬컴퓨터
(Fig. 1) Hypercube multicomputer

하이퍼큐브 노드들의 커뮤니케이션은 상호 메세지 전달에 의하여 이루어지므로 효율적인 메세지 전달은 하이퍼큐브 병렬컴퓨터의 성능에 중요한 요소가 된다. 노드들의 상대주소(relative address)를 이용하여 메세지를 최단거리로 전달하는 알고리즘이 현재 사용되고 있다[5, 6].

손상된 하이퍼큐브란 어떤 노드나 링크에 결함이 발생하여 전체적으로 정상적인 기능을 수행할 수 없는 상태의 하이퍼큐브를 말하는데, 하이퍼큐브에서 결함은 노드나 링크에서 발생 가능하며, 특히 P_{110}와 같이 노드에서 결함이 발생하면 그 노드에 인접한 모든 링크들은 결함 링크(faulty link)로 간주할 수 있다. (그림 2)는 3-차원 하이퍼큐브에서 노드 P_{110}와 0*1 링크가 손상된 예를 보여준다. 표기의 간략화를 위해 001과 011 사이의 링크는 심볼 "*"를 사용하여 0*1으로 나타낸다.



(그림 2) 손상된 3-차원 하이퍼큐브
(Fig. 2) An injured 3-dimensional hypercube

손상된 노드나 링크는 하드웨어 혹은 소프트웨어상의 문제로 인하여 정상적인 기능을 수행할 수 없으므로 정상적인 하이퍼큐브에서 사용되는 최단거리 메세지 라우팅 알고리즘은 작동되지 않을 수도 있다. 이것은 부분적인 시스템 결함이 시스템 전체의 기능을 마비시키는 결과를 초래할 수 있음을 의미한다. 그러나 손상된 하이퍼큐브에서도 정상적인 노드와 링크들을 잘 이용하면 원하는 작업을 계속 수행할 수 있다. 예를 들면, (그림 2)에서 프로세서 P_{010}가 프로세서 P_{111}에게 메세지를 전달할 때 최단경로 알고리즘은 010→110→111의 경로로 메세지를 전달하는데 P_{110}와 링크 0*1에 결함이 있으므로 이 메세지는 목적노드에 전달될 수 없게 되어 전체 하이퍼큐브가 정상적인 기능을 수행할 수 없게 되지만, 이 메세지는 정상적인 노드와 링크들을 갖는 다른 경로를 이용하여 전달될 수 있다. 가능한 경로들은 010→011→111과 010→000→100→101→111 그리고 010→000→001→101→111 등이다. 이들 중 최단거리를 갖는 경로 010→011→111를 선택하여 메세지를 전달할 수 있다.

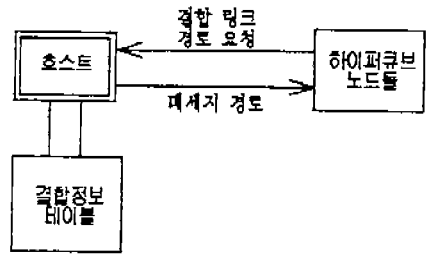
이처럼 시스템상에 결함이 있는 노드나 링크가 존재하더라도 정상적인 노드와 링크들만을 통하여 메세지를 목적지까지 전달할 수 있는 기능을 오류회복 기능(fault-tolerance)이라 한다. 일반적으로 노드나 링

드의 결함은 운영 중에 발생하므로 어느 노드나 링크에 결함이 있는가를 미리 예측하는 것이 불가능하다. 따라서 손상된 하이퍼큐브에서의 오류회복기능 알고리즘은 결함이 있는 노드나 링크의 위치에 따라 동적으로 작동하는 기능을 가져야하므로 효율적인 알고리즘 개발은 어려운 문제들 중의 하나이다.

현재까지 오류회복기능을 갖는 여러 가지 메세지 라우팅 알고리즘들이 제안되어 왔다 [7, 8, 9, 10, 11]. 이러한 알고리즘들은 결함 링크에 대한 지역 정보(local information)를 이용하거나 전역 정보(global information)를 이용한다. 지역 정보를 이용하는 알고리즘에서는 각 노드가 인접한 결함 링크에 대한 정보만을 저장하며, 메세지 전달과정에서 결함 링크를 통한 메세지 전달이 시도될 때 정상적인 링크를 이용하도록 메세지 전달 경로를 깊이우선탐색(depth-first search)의 개념과 이의 변형된 방법으로 우회(detour)시켜 메세지를 목적지까지 보낸다[7, 8, 9, 10]. 이 접근방법은 노드들이 결함 정보에 대한 저장 공간을 적게 사용한다는 장점이 있는 반면에 메세지의 교착상태(deadlock)가 발생하거나 교착상태를 방지하기 위하여 부수적인 시간을 소요하여야 하며, 경우에 따라 메세지를 최단거리로 전달하지 못하는 단점이 있다. 그러나 전역 정보를 사용하는 알고리즘에서는 각 노드가 지연 테이블(delay table)을 이용하여 목적지까지 정상적인 링크들로 구성된 메세지 전달 경로를 산출하여 메세지를 전달한다[8, 11]. 이 접근방법은 결함 정보를 저장하기 위한 공간이 각 노드마다 중복해서 사용된다는 단점이 있는 반면에 결함 링크 수에 관계없이 항상 최적의 전달 경로를 이용할 수 있는 장점이 있다.

본 연구에서 제안하는 알고리즘은 전역 정보를 사용하나 저장 공간의 중복을 피하기 위하여 결함 정보를 각 노드에 저장하지 않고 하이퍼큐브의 호스트에만 저장하여 이용한다. 그리고 호스트의 처리 능력은 매우 높고, 노드들과 빠른 시간에 통신할 수 있으며, 큰 용량의 메모리를 갖는다고 가정한다. 따라서 각 노드들이 메세지를 전달할 때 호스트에게 메세지 전달 경로를 요청하는데, 이로인한 호스트의 부하 문제와 혼잡(traffic) 문제는 고려하지 않았다. 그 이유로는 하이퍼큐브가 손상된 경우, 손상된 상태는 정상적인 상태로 고쳐질 때까지 일시적으로 존재하며, 손상된 상태동안 신뢰성을 유지하기 위하여 호스트가 이러한 문제를 감수할 수 있기 때문이다.

그리고 어떤 노드가 인접한 노드나 링크의 결함을 인지했을 때 결함에 대한 정보를 호스트에 전달하며 호스트는 이를 결함 정보 테이블에 저장한다. 손상된 하이퍼큐브에서 임의의 노드(source)가 다른 노드(destination)에게 메세지를 전달하려고 할 때 자신의 노드 번호와 목적지의 노드 번호를 호스트에 전달하면 호스트는 결함 정보 테이블을 이용하여 정상적인 메세지 경로를 결정한 후, 이를 시작 노드에게 보낸다. 이때 시작 노드는 이 경로를 따라 목적 노드에게 메세지를 전달한다. 이러한 관계를 그림으로 나타내면 (그림 3)과 같다. 호스트가 결함 정보 테이블을 이용하여 최단경로의 정상적인 메세지 전달 경로를 찾는 알고리즘은 3장에서 언급하기로 한다.



(그림 3) 호스트와 노드들간의 관계
(Fig. 3) The relation between a host and nodes

2. 독립된 경로

n-차원 하이퍼큐브(Q^n으로 표시)는 다음과 같이 정의된다.

(i) Q^0는 한 개의 노드만을 갖는다.

(ii) $Q^n = K_2 \times Q^{n-1}$, 이때 K_2는 두 개의 노드를 갖는 완전 그래프(complete graph)이고, $\times$는 두 그래프의 프로덕트(product) 연산이다[12].

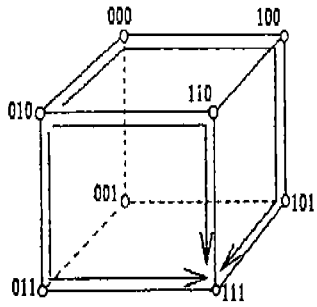
따라서 Q^n은 2^n개의 노드를 포함하며, Q^n의 각 노드의 차수가 n이므로 $n \cdot 2^{n-1}$ 개의 링크를 포함한다. 그리고 각 노드의 주소는 $a_n a_{n-1} \cdots a_1$으로 나타낸다. a_i는 0 또는 1 이다.

하이퍼큐브에서 두 노드 사이의 해밍거리(hamming

distance)는 다음과 같이 정의된다. 서로 다른 두 노드의 주소가 $u=u_nu_{n-1}\cdots u_1$와 $w=w_nw_{n-1}\cdots w_1$일때, 해밍거리는 $H(u, w)=\sum h(u_i, w_i)$이다. 만약 $u_i=w_i$이면 $h(u_i, w_i)=0$이고, 만약 $u_i\neq w_i$이면 $h(u_i, w_i)=1$이다.

독립된 경로(disjoint paths)란 시작 노드에서부터 목적 노드까지의 여러 경로들 중 경로들 간에 노드나 링크가 서로 중복되지 않는 경로들을 의미한다.

Q^n에서 u와 w가 임의의 두 노드이고 두 노드간의 해밍거리가 k일 때, n개의 독립된 경로가 존재하며 그 중 k개의 경로는 길이가 k이고, 나머지 $n-k$개의 경로는 길이가 $k+2$ 이다[13]. Q^3에서 독립된 경로가 존재하는 예를 살펴보자.



(그림 4) Q^3에서 독립된 경로들의 예
(Fig. 4) An example of disjoint paths in Q^3

(그림 4)의 Q^3에서 노드 010과 111 사이에는 해밍거리가 2이고 3개의 독립된 경로가 존재한다. 3개의 경로들 중 $k=2$개의 경로의 길이는 2이고, 나머지 $n-k=3-2=1$개의 경로의 길이는 $k+2=2+2=4$이다. 길이가 2인 경로는 010→110→111과 010→011→111 이며, 010→000→100→101→111은 길이가 4인 경로이다. 이들 3개의 경로는 중복된 링크나 노드를 갖지 않으므로 독립된 경로가 된다.

Q^n에서 임의의 두 노드 사이의 해밍거리가 k 일 때 경로의 길이가 k인 최단경로의 독립된 경로 수는 k개이다. 여기에서는 최단경로가 k개인 독립된 경로들을 구하는 방법에 대해 살펴본다.

하이퍼큐브에서 임의의 두 노드가 $u=u_nu_{n-1}\cdots u_1$와 $w_nw_{n-1}\cdots w_n$일 때, 배타적(exclusive) 연산 $u\oplus w=r_nr_{n-1}\cdots r_1$로 정의된다. 이때 만약 $u_i=w_i$이면 $r_i=0$이고, 만약 $u_i\neq w_i$이면 $Y_i=1$이다. 그리고 두 노드 사이의 상대주소는 배타적 연산의 표현으로 나타낼 수 있다. 노드 u와 w가 주어질 때, 노드 u에 대한 w의 상대주소는 $w/u=u\oplus w$로 정의한다. 예를 들면, $u=010$이고 $w=111$일 때, 상대주소는 $010\oplus 111=101$이다.

이러한 상대주소를 이용하여 라우팅 순서(routing sequence)를 정의할 수 있다[8]. 시작 노드가 u이고 목적 노드가 w일 때 라우팅 순서는 다음과 같이 정의된다. 노드 u와 w의 해밍거리가 k이면, 라우팅 순서 $R=\langle c_1, c_2,\cdots,c_k\rangle$는 노드 u에 대한 w의 상대주소 $w/u=r_nr_{n-1}\cdots r_1$에서 $r_i=1$인 비트 위치(bit position)들을 순서대로 나열한 것이다. 예를 들어, Q^5에서 $u=01000$ 이고 $w=11011$ 이면, $w/u=10011$ 이므로 라우팅 순서는 $R=\langle 1, 2, 5\rangle$로 결정된다. 라우팅 순서를 이용하여 시작 노드에서부터 목적 노드까지 메세지 경로를 결정할 수 있다. 메세지 경로를 결정하는 방법은 라우팅 순서에 포함된 c_i $(1\leq i\leq n)$의 위치에 해당하는 시작 노드의 비트를 차례로 보수(complement)시키는 것이다. 위의 예에서 라우팅 순서는 $R=\langle 1, 2, 5\rangle$이므로 시작 노드 01000에서 해당 위치의 비트를 차례로 보수시키면 01001, 01011, 11011로 되어 01000→01001→01011→11011의 경로를 얻을 수 있다.

하나의 라우팅 순서를 이용하여 시작 노드에서부터 목적 노드까지의 최단경로를 갖는 독립된 경로들을 구하는 방법을 살펴보자. 시작 노드가 u이고 목적 노드가 w 일 때 노드 u와 w의 해밍거리가 k이면 하나의 라우팅 순서를 위와 같은 방법으로 구하여 R_1이라 한다. R_1의 c_i $(1\leq i\leq n)$을 순열(permutation)시켜 다음과 같이 $R_2, R_3,\cdots,R_k$를 구할 수 있다[14].

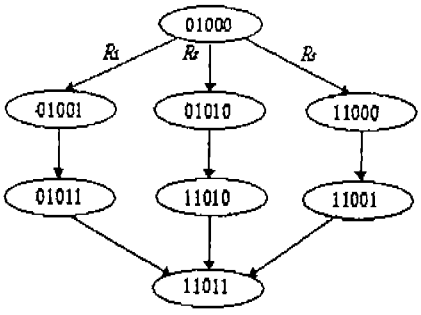
$$R_1=\langle c_1, c_2, c_3,\cdots c_{k-1}, c_k\rangle$$
$$R_2=\langle c_2, c_3, c_4,\cdots c_k, c_1\rangle$$
$$R_3=\langle c_3, c_4, c_5,\cdots c_1, c_2\rangle$$
$$\cdot \qquad \cdot$$
$$\cdot \qquad \cdot$$
$$R_{k-1}=\langle c_{k-1}, c_k, c_1,\cdots c_{k-3}, c_{k-2}\rangle$$
$$R_k=\langle c_k, c_1, c_2,\cdots c_{k-2}, c_{k-1}\rangle$$

이러한 k개의 라우팅 순서는 모두 독립된 경로들이고 행렬(matrix)로 간단히 표현될 수 있다. 예를 들어,

Q^5에서 $u=01000$ 이고 $w=11011$ 이면 $w/u=10011$ 이므로 하나의 라우팅 순서는 $R_1=\langle 1,2,5\rangle$이다. 이를 순열하여 R_2와 R_3를 얻는다.

$R_1=\langle 1, 2, 5\rangle : 01000 \rightarrow 01001 \rightarrow 01011 \rightarrow 11011$
$R_2=\langle 2, 5, 1\rangle : 01000 \rightarrow 01010 \rightarrow 11010 \rightarrow 11011$
$R_3=\langle 5, 1, 2\rangle : 01000 \rightarrow 11000 \rightarrow 11001 \rightarrow 11011$

이들 3개의 경로들을 (그림 5)와 같이 표현하면 R_1, R_2, R_3가 독립된 경로임을 알 수 있다.



(그림 5) 3개의 독립된 경로
(Fig. 5) Three disjoint paths

이제까지 설명한 독립된 경로와는 다른 형태의 독립된 경로들이 여러 개 존재할 수 있으나[14], 본 연구에서는 지금까지 설명한 방법에 따라 한 가지 형태인 k개의 독립된 경로를 결정하여, 이를 손상된 하이퍼큐브에서 노드들 간의 메세지 전달 경로를 얻기 위하여 사용한다.

3. 손상된 하이퍼큐브상의 라우팅 알고리즘

3.1 깊이우선 탐색

Q^n에서 시작 노드 u와 목적 노드 w 사이의 해밍거리가 k일 때 최단경로의 길이는 k이며 모든 가능한 경로의 수는 $k!$이다. 이를 위하여 2장에서 설명한 라우팅 순서를 이용한다. 시작 노드와 목적 노드 사이의 해밍거리가 k이므로 모든 가능한 라우팅 순서 $\langle c_1, c_2, \cdots, c_k\rangle$는 k개 원소들의 순열에 해당하므로 $k!$에 해당한다.

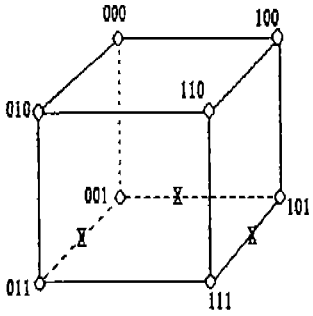
손상된 하이퍼큐브에서 최단경로의 정상적인 경로를 구하기 위한 간단한 방법으로 모든 가능한 경로들을 깊이우선탐색으로 검색하는 것이다. 깊이우선탐색에서 길이가 k인 하나의 정상적인 경로가 발견되면 탐색을 멈추고 그 경로를 반환한다. 깊이우선탐색을 알고리즘으로 표현하면 (알고리즘 1)과 같다.

```
procedure dfs(u, w)
//A solution is generated in x(1:k) and this procedure
  stops as soon as a solution is determined. //
local x(1:k)
k←hamming_distance(u, w)
S←{c1, c2,⋯,ck} such that ci is a bit position of ri
                with ri=1 in w/u=rn rn-1⋯r1
i←1
while i>0 do
    if there remains an untried x(i) such that
      x(i)∈S and x(i)≠x(j) for j=1,..., i-1
    then if <x(1),..., x(i)> is not a faulty path
        then if i=k
            then return(x(1),...,x(i)) //a path is
                                        found //
            else i←i+1 //consider the next set //
            endif
        endif
    else i←i-1 // backtrack to previous set //
    endif
endwhile
end
```

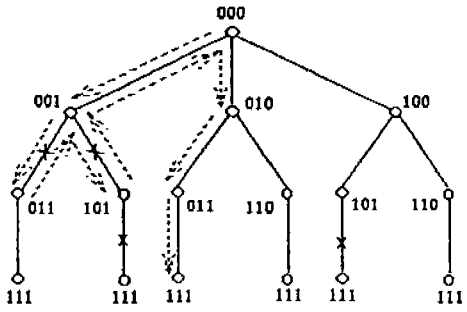
(알고리즘 1) 깊이우선탐색
(Algorithm 1) Depth-first search

(알고리즘 1)의 깊이우선탐색이 수행하는 과정을 예로 들면, (그림 6)과 같은 손상된 하이퍼큐브 Q^3에서 시작 노드가 000 이고 목적 노드가 111 일 때 $k=3$이고 S={1, 2, 3}이다. (알고리즘 1)이 진행될 때 라우팅 순서 〈1〉과 〈1, 2〉가 차례로 만들어지는데 〈1, 2〉에서 결함 링크를 포함하게 된다. 그러므로 다시 〈1,

3)을 만드나 〈1, 3〉 역시 결함 링크를 포함하므로 〈2〉로 백트랙(backtrack)하여, 다시 라우팅 순서 〈2, 1〉을 만들고 〈2, 1, 3〉을 만든다. 이때 〈2, 1, 3〉이 결함 링크를 포함하지 않고 길이가 3 이므로 하나의 정상적인 경로를 얻게되어 이를 반환하고 프로시듀어는 끝난다. (그림 7)은 시작 노드 000에서 목적 노드 111 까지의 모든 가능한 경로와 (알고리즘 1)이 탐색한 경로를 나타낸다.



(그림 6) 손상된 3-차원 하이퍼큐브
(Fig. 6) An injured Q^3



(그림 7) 깊이우선탐색 과정
(Fig. 7) The process of depth-first search

3.2 독립된 경로의 이용

(알고리즘 1)의 깊이우선탐색은 S의 값에 따라 항상 일정한 방향으로 탐색을 수행하며 결함 링크들의 분포와 관계없이 탐색 방향이 결정된다. 이러한 점을 개선하기 위하여 2장에서 언급한 독립된 경로를 사용한다. Q^n에서 시작 노드 u와 목적 노드 w 사이의 해밍거리가 k 일 때 최단경로의 길이가 k인 k개의 독립된 경로를 구하고 결함 링크 정보를 가진 테이블을 이용하여 독립된 경로들을 확인한다. 확인하는 과정에서 정상적인 경로가 하나라도 존재하면 그 경로가 선택된다. 그러나 k개의 독립된 경로가 모두 결함 링크를 포함한다면 각 경로에 결함 링크의 위치를 표시한 후, 결함 링크 위치가 시작 노드에서 가장 먼 경로를 하나 선택한다. 이때 결함 링크의 위치가 동일한 것이 여러 개 존재한다면 임의로 하나를 선택한다. 선택된 경로의 결함 링크가 j 위치에서 발생했다면 라우팅 순서 〈x(1),...,x(j-1)〉은 정상적인 링크만을 포함하는 것을 의미하므로 x(j)에서부터 x(k) 까지 만을 대상으로 깊이우선탐색을 적용한다. 이 과정에서 하나의 정상적인 라우팅 순서 〈x(1),...,x(k)〉를 얻으면 이를 반환하고, 정상적인 경로가 존재하지 않으면 j-1 위치로 백트랙된다. 백트랙이 되는 경우, j-1 위치의 링크가 결함 링크가 아니지만 결함 링크와 같은 의미를 가지므로 j-1 위치를 결함으로 표시하고, 나머지 독립된 경로들과 결함 링크의 위치를 다시 비교하여 시작 노드에서 가장 먼 결함 링크를 갖는 경로를 얻은 후 이 과정을 반복한다. 이러한 과정을 알고리즘으로 나타내면 (알고리즘 2)와 같다.

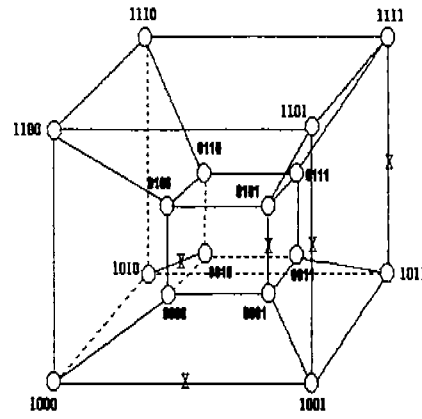
```
procedure heuristic(u, w)
//A solution is generated in x(1:k) and this pro-
  cedure stops as soon as a solution is determined. //
local x(1:k)
k←hamming_distance(u, w)
S←{c1, c2,···,ck} such that ci is a bit position of ri
                  with ri=1 in w/u=rnrn-1···r1
generate k disjoint paths using S and k, and let them
C1,...,Ck
for i←1 to k do
      if Ci is a normal path
      then return Ci
      else f(i)←position of faulty link in Ci
      endif
endfor
loop
      let f(s) be the maximum of {f(1),...,f(k)}
```

```
    i←s // s-th path //
    j←f(s) // the position of faulty link in s-th
              path //
    if j=0 then exit endif // there is no shortest
                              path //
    search C_i using depth-first search from the
    positon j
    if a normal path to the destination node is
    found
    then return it
    else f(i)←j-1 // mark the fault position to j-1 //
    endif
endloop
end
```

(알고리즘 2) 독립된 경로를 이용한 깊이우선탐색
(Algorithm 2) Depth-first search using disjoint paths

(알고리즘 2)가 수행되는 과정을 예를 들어 살펴보면, (그림 6)과 같은 손상된 하이퍼큐브에서 (알고리즘 2)는 먼저 〈1, 2, 3〉, 〈2, 3, 1〉, 〈3, 1, 2〉의 3 개의 독립된 경로를 만든다. 3 개의 독립된 경로 각각에 대하여 차례로 정상적인 경로인가를 확인한다. 이 과정에서 〈1, 2, 3〉은 결함 링크를 포함하며, 〈2, 3, 1〉은 정상적인 경로가 되므로 이를 반환하고 멈춘다.

다른 경우의 예를 살펴보기 위하여 (그림 8)과 같은 손상된 하이퍼큐브를 고려하여 보기로 한다. 시작 노드가 0110 이고 목적 노드가 1001 일 때 두 노드간의 해밍거리는 4가 되어 4 개의 독립된 경로인 라우팅 순서 〈1, 2, 3, 4〉, 〈2, 3, 4, 1〉, 〈3, 4, 1, 2〉, 〈4, 1, 2, 3〉를 구한다. 결함 링크 정보를 이용하여 각 경로들이 3, 4, 2, 3 위치에 결함 링크를 포함하고 있음을 알 수 있다. 이 경로들 중 시작 노드에서 가장 멀리 있는 결함 링크를 갖는 경로 〈2, 3, 4, 1〉이 선택되어 깊이우선탐색을 수행한다. 그러나 〈2, 3, 4, 1〉의 경로는 〈2, 3, 4〉 다음에 다른 링크를 선택할 수 없으므로 결함 링크의 위치를 3으로 표시하고 백트랙된다. 이때 〈1, 2, 3, 4〉, 〈2, 3, 4, 1〉, 〈4, 1, 2, 3〉은 모두 같은 위치에서 결함 링크를 가지므로 임의로 하나의 경로를 선택한다. 만약 〈1, 2, 3, 4〉가 선택되면 3의 위치에 결함 링크를 가지므로 〈1, 2, 4, 3〉 경로를 확인하게 되나 이 경로 역시 결함 링크를 포함한다. 그러므로 〈1, 2, 3, 4〉는 결함 링크의 위치를 2로 표시하고 백트랙된다. 다음으로 〈2, 3, 4, 1〉과 〈4, 1, 2, 3〉의 경로가 모두 3의 위치에 결함 링크를 가지므로 임의로 〈2, 3, 4, 1〉을 선택한다. 〈2, 3, 4, 1〉의 결함 링크의 위치가 3이므로 〈2, 3, 1, 4〉의 경로를 확인하게 된다. 이때 〈2, 3, 1, 4〉가 정상적인 경로이므로 이를 반환하고 알고리즘은 끝난다. 따라서 결정된 경로는 0110→0100→0000→0001→1001.



(그림 8) 손상된 4-차원 하이퍼큐브
(Fig. 8) An injured Q^4

(알고리즘 1)과 (알고리즘 2)를 비교해 볼 때 독립된 경로를 이용한 (알고리즘 2)가 결함 링크의 분포에 따라 정상적인 경로를 빨리 찾을 수 있다. 일반적으로 하이퍼큐브에서 노드에 결함이 발생하면 그 노드에 연결된 모든 링크들이 결함 링크로 간주되며, 그 주위의 노드나 링크들에 결함이 발생할 가능성이 높으므로 (알고리즘 2)는 이러한 결함의 분포에 따라 결함이 밀집한 부분에서 멀리 떨어진 링크들을 이용하는 경로를 먼저 탐색할 수 있는 장점이 있다. 그러나 최악의 경우, (알고리즘 2)는 (알고리즘 1)과 마찬가지로 모든 경로에 대하여 탐색을 수행하게 되므로 (알고리즘 1)과 같은 시간 복잡도(time complexity)를 가질 뿐만 아니라 부수적인 시간을 필요로 한다.

3.3 시뮬레이션 결과

앞에서 설명한 깊이우선탐색과 독립된 경로를 이

용하는 알고리즘은 IBM PC-486에서 Turbo C로 프로그래밍하여 시뮬레이션을 수행하였다. 하이퍼큐브의 크기는 4, 5, 6, 7, 8, 9, 10 차원까지 고려하였다. 그리고 손상된 하이퍼큐브를 얻기 위하여 노드에 결함이 있는 것으로 가정하고, 난수(random number)를 사용하여 결함 노드들을 결정하였다. 노드에 결함이 있는 경우 그 노드에 인접한 모든 링크들은 결함 링크로 처리하였다. 결함 노드의 비율이 10%, 20%, 30%, 40% 일 때 독립적으로 프로그램을 실행시켜 두 알고리즘이 필요로 하는 링크의 비교 회수를 측정하였으며 그 결과는 〈표 1〉과 같다.

〈표 1〉 시뮬레이션 결과
〈Table 1〉 The result of simulation

차 원	알고리즘	비 교 회 수			
		10%	20%	30%	40%
4	알고리즘 1	4	7	8	11
	알고리즘 2	4	8	7	8
5	알고리즘 1	5	7	8	10
	알고리즘 2	5	8	7	8
6	알고리즘 1	7	9	9	19
	알고리즘 2	8	9	11	17
7	알고리즘 1	9	10	10	25
	알고리즘 2	10	10	11	20
8	알고리즘 1	10	11	11	29
	알고리즘 2	10	12	15	25
9	알고리즘 1	12	13	25	30
	알고리즘 2	13	15	30	27
10	알고리즘 1	12	15	30	34
	알고리즘 2	13	16	36	30

〈표 1〉의 결과를 살펴보면, 결함의 비율이 낮을 때 두 알고리즘에서 필요로 하는 비교 회수는 별 차이가 없으나 비율이 높아 질수록 (알고리즘 2)의 비교 회수가 (알고리즘 1)보다 적어진다는 것을 알 수 있다. 이것은 결함의 분포에 따라 결함이 밀집한 부분과 멀리 떨어진 링크들을 이용하는 경로를 먼저 탐색하는 (알고리즘 2)의 특징이 결함의 비율이 높을수록 잘 나타나기 때문이다. 그러나 결함의 비율이 낮을 때 (알고리즘 2)가 (알고리즘 1)보다 더 많은 비교 회수를 수행한다. 그 이유는 (알고리즘 2)가 하나의 경로를 산출하는데 (알고리즘 1)보다 부수적으로 더 많은 비교를 필요로 하기 때문이다.

한편, 〈표 1〉의 결과에 사용된 결함 정보는 모두 임의로 만들어졌으므로 결함이 군집화되는 실제 상황에서는 두 알고리즘의 성능이 크게 다를 수 있을 것으로 예측된다.

4. 결 론

본 연구에서는 손상된 하이퍼큐브상의 메세지 라우팅 알고리즘을 제안하였다. 깊이우선탐색과 독립된 경로를 이용하는 알고리즘은 모두 결함 정보를 전역 정보로 저장하며 하이퍼큐브의 호스트에서 작동된다. 하이퍼큐브상의 임의의 노드가 메세지 전달을 원할 때, 그 노드가 목적 노드를 호스트에 전달하면 호스트는 결함 링크가 포함되지 않는 최단경로를 반환하게 된다. 이와같은 상황에서는 호스트가 하이퍼큐브 시스템의 병목현상(bottleneck)이 될 수 있으나, 하이퍼큐브의 손상된 상태는 정상적인 상태로 고쳐질 때까지 일시적으로 존재하므로 손상된 상태동안 신뢰성을 유지하기 위하여 호스트의 과부하를 감수할 수 있는 것으로 가정하였다.

알고리즘의 성능을 평가하기 위하여 시뮬레이션을 수행하였으며, 그 결과 결함의 비율이 높아질수록 독립된 경로를 이용하는 알고리즘이 더 적은 수의 비교를 수행한다. 그러나 시뮬레이션에서 사용된 결함 정보는 모두 임의로 만들어졌기 때문에 결함이 군집화되는 실제 상황에서는 두 알고리즘의 성능이 다를 수 있다고 본다.

참 고 문 헌

[1] M. Quinn, 'Designing Efficient Algorithms for Parallel Computers', McGraw-Hill, 1987.

[2] E. Krishnamurthy, 'Parallel Processing: Principles and Practice', Addison Wesley, 1989.

[3] S. Ranks, Y. Won, and S. Sahni, "Programming a Hypercube Multicomputer," IEEE Software, Vol. 5, No. 5, pp. 60-77, 1988.

[4] J. Woo, "Efficient Hypercube Algorithms," Ph.

D. Dissertation, University of Minnesota, 1989.

[5] H. Sullivan and R. Bashkow, "A large scale homogeneous, fully distributed parallel machine," Proc. Fourth Symp. Comput. Architecture, pp. 105-117, Mar. 1977.

[6] H. Katseff, "Incomplete Hypercubes," IEEE Trans. on Computers, Vol. 37, No. 5, pp. 604-608, May 1988.

[7] M. Chen and K. Shin, "Depth-First Search Approach for Fault-Tolerant Routing in Hypercube Multicomputers," IEEE Trans. on Parallel Distributed Systems, Vol. 1, No 2, pp. 152-159, April 1990.

[8] M. Chen and K. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers," IEEE Trans. on Computer, Vol. 39, No. 12, pp. 1406-1416, Dec. 1990.

[9] J. Gordon and Q. Stout, "Hypercube Message Routing in the Presence of Faults." 3rd Hypercube Conference, pp. 318-327, 1988.

[10] T. Lee and J. Hayes, "Routing and Broadcasting in Faulty Hypercube Computers," 3rd Hypercube Conference, pp. 346-354, 1988.

[11] J. Kim and K. Shin, "Deadlock-Free Fault-Tolerant Routing in Injured Hypercube," IEEE Trans. on Computer, Vol. 42, No. 9, pp. 1078-1088, Sep. 1993.

[12] F. Harary, 'Graph Theory, Reading, MA:', Addision-Wesley, 1969.

[13] Y. Saad, Y. and M. Schultz, "Topological Properties of Hypercubes," IEEE Trans. on Computer, Vol. 37, No. 7, pp. 867-872, July 1988.

[14] 윤기송, "하이퍼큐브 내에서의 가능한 노드 독립 경로의 분석," 정보과학회논문지, 제21권, 제4호, pp. 724-736, 1994.

공 헌 택

1984년 미국 NMSU대학교 전자계산학과(학사)

1987년 미국 Utah State University 전자계산학과(석사)

1992년~현재 단국대학교 대학원 전산통계학과 박사과정

1988년~1990년 한국국방연구원 전산체계연구부 근무

1990년~현재 국립천안공업전문대학 전자계산과 조교수

관심분야: 병렬알고리즘, 병렬처리

우 진 운

1980년 서울대학교 수학교육과(학사)

1989년 미국 University of Minesota 전산학과(박사)

1980년~1983년 대한항공 및 국토개발연구원 전산실 근무

1989년~현재 단국대학교 전산통계학과 부교수

관심분야: 알고리즘, 병렬처리 및 분산처리