적응적 오류 허용 라우팅 : SCP를 이용한 메쉬 구조에서의 RIFP 기법 개선

(Fault-Tolerant Adaptive Routing : Improved RIFP by using SCP in Mesh Multicomputers)

정 성 우 † 김 성 천 ††
(Sungwoo Chung) (Sungchun Kim)

요 약 다중 프로세서 환경에서의 오류 허용에 대한 적응적 라우팅 기법은 매우 중요한 요소이다. 특히 메쉬 구조를 갖는 다중 프로세서에서의 오류를 허용하는 라우팅 기법은 구현에 있어 간결함을 제공하는 환경으로, 다수의 오류를 허용하기 위해 직사각형 모양의 오류 블록으로 구성하여 라우팅을 수행한다. 이 경우, 블록 내부의 정상적인 노드 역시 오류 노드로 간주되어 전체 노드의 사용율을 저하시키는데, 오류 블록을 몇 개의 확장된 메쉬로 나누고, 구성된 확장된 메쉬들의 관계를 DAG(Directed Acyclic Graph)로 구성하고, 이 DAG에서의 확장된 메쉬간의 최단거리를 구하여 메시지를 전송하는 RIFP (Routing for Irregular Faulty Pattern)기법으로 이를 해결하였다. 그러나, 이 기법은 노드간에 주고받는 메시지가 거치는 hop의 수가 오히려 증가되는 문제가 발생하게 된다. 이러한 문제를 해결하기 위하여 본 논문에서는 증가되는 hop의 수를 억제하기 위해 목적 노드와 이웃 노드들로부터 오류 블록 경계 부분까지 직선 경로 SCP(Short-Cut Path)를 찾아 존재하는 경우, SCP를 적용하는 RIFP를 제안한다.

키워드 : 라우팅, 결함허용, DAG, 다중프로세서

Abstract *Adaptive routing methods are studied for effective routing in many topologies where occurrence of the faulty nodes are inevitable. Mesh topology provides simplicity in implementing these methods. Many routing methods for mesh are able to tolerate a large number of faults enclosed by a rectangular faulty block. But they consider even good nodes in the faulty block as faulty nodes. Hence, it results the degradation of node utilization. This problem is solved by a method which transmits messages to destinations within faulty blocks via multiple "intermediate nodes". It also divides faulty block into multiple expanded meshes. With these expanded meshes, DAG(Directed Acyclic Graph) is formed and a message is able to be routed by the shortest path according to the DAG. Therefore, the additional number of hops can be resulted. We propose a method that reduces the number of hops by searching direct paths from the destination node to the border of the faulty block. This path is called SCP(Short-Cut Path). If the path and the traversing message is on the same side of outside border of the faulty block, the message will cut into the path found by our method. It also reduces the message traverse latency between the source and the destination node.*

Key words : Fault-Tolerant, adaptive routing, mesh, faulty block

1. 서 론

현대 정보화 시대에서의 다양한 컴퓨터 응용분야는 기존의 컴퓨터 구조로는 효율적인 해결책을 제공하기가 어렵다. 이러한 이유로 대규모 병렬 처리 시스템(massively parallel processing system)의 개발이 활발히 진행되고 있다. 멀티컴퓨터(multicomputer)는 이러한 병렬 처리를 위해 설계된 분산 메모리 메시지-전송 병행 컴퓨터(distributed memory message-passing concurrent computer)이다[1]. 멀티컴퓨터의 구성 요소의 증가에 따른 오류의 증가에 대한 시스템의 신뢰성(reliability)은 대규모의 멀티컴퓨터의 설계 및 구현에 있어서 핵심적인 부분이다[2].

· 본 연구는 한국과학재단 목적기초연구 R01-2001-000-00356-0(2002) 지원으로 수행되었음
† 비 회 원 : (주) 디비엠코리아 재직
swc5@dbmkorea.co.kr
†† 종신회원 : 서강대학교 컴퓨터학과 교수
ksc@arqlab1.sogang.ac.kr
논문접수 : 2001년 4월 13일
심사완료 : 2003년 8월 6일

오류 허용 라우팅 기법은 시스템의 구조(topology)에 상관없이 메시지의 상호연결 통신망의 성능향상을 위한 매우 중요한 요소이다. 메쉬 구조는 오류를 허용하는 라우팅 기법의 구현에 있어 간결함을 제공하는 환경으로 많은 기법이 연구되고 있다. Glass와 Ni[3]는 턴 모델(turn model)을 기반으로 한 n-차원 메쉬 구조에서 n-1개의 오류를 허용하는 부분 적응적 라우팅 알고리즘을 제안하였다. Linder와 Harden[4]은 가상 상호연결 통신망의 개념을 기반으로 한 완전 적응적 오류 허용 라우팅을 제안하였다. 또한, 물리적 링크 당 특정한 수의 가상 채널을 사용함과 동시에 오류들을 직사각형의 오류 블록(faulty block)으로 구성함으로써 많은 수의 오류를 허용하는 것을 가능하게 해주는 라우팅 알고리즘이 있다. Boppana와 Chalasani[5]는 물리적 링크 당 네 개의 추가적인 가상 채널을, 또한 Su와 Shin[6]은 물리적 링크 당 단 두 개만의 가상 채널을 사용하는 적응적 라우팅 알고리즘을 제안했다.

그러나, [5, 6]에서 제안된 기법들은 오류 블록 내에 있는 모든 노드들을 오류가 발생한 노드들로 간주한다. 즉, 오류가 나지 않은 정상적인 노드들 마저 오류가 난 것으로 처리되어 메시지 교환을 불허하여 노드 사용율을 저하시킨다. 이러한 노드 사용율 저하를 극복하기 위해 Tsai와 Wang[7]은 오류 블록을 확장된 메쉬로 나누고 이를 *DAG*(Directed Acyclic Graph)에 적용하여 메시지를 전송하는 라우팅 기법 RIFP(Routing for Irregular Faulty Patterns)를 제안했다. 그러나 이들이 제안한 알고리즘은 *DAG*를 기반으로 구한 최단 거리를 고려하기에 메시지 전송이 많은 거리를 우회하여 hop의 수를 증가시키는 경우가 생기는 단점이 있다.

본 논문에서는 기존의 기법의 단점인 증가되는 hop의 수를 억제하기 위해 목적 노드와 이웃 노드들로부터 오류 블록 경계 부분까지 직선 경로인 SCP(Short-Cut Path)를 찾아 라우팅에 적용시키는 조건을 제시한다. 탐색된 SCP는 이에 인접한 오류 블록 외부의 노드에서 기억되고 라우팅 시에 오류 블록 외부에 처음으로 전송된 메시지와 같은 면에 있는 경우 SCP를 적용한 라우팅을 진행시킨다.

2. 비정규적 오류 패턴에 대한 라우팅 알고리즘 (RIFP)

2.1 기초 과정

메쉬 구조에서 오류 블록이 구성되려면 우선 오류가 발생한 노드(faulty node)가 있어야 한다. 정상적인 노드들은 안정 노드(Safe node)와 불안정 노드(Unsafe node)로 나뉘어지는데, 어떤 정상적인 노드에 대하여 이웃 노드 중 두 개 이상의 오류 노드 혹은 불안정 노드

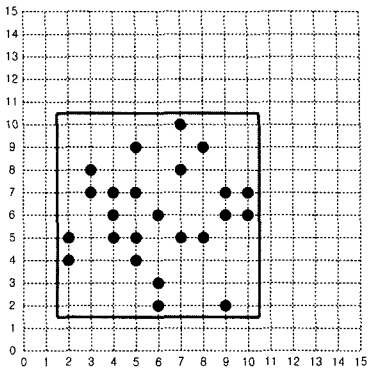

그림 1 불안정 노드의 정보를 이용하여 구성된 오류 블록의 예

가 인접해 있을 경우, 그 노드를 가리켜 불안정 노드라고 한다. 이에 해당하지 않는 노드는 안정 노드가 된다. 이러한 정보를 이용하여 불안정 노드들을 계속 탐색해 나가면 그림 1과 같은 직사각형의 오류 블록이 생성된다. 직선의 블록 내의 점선이 교차하는 부분에 검은 원이 오류가 발생한 노드를 가리키며 나머지 점선이 교차하는 부분이 불안정 노드를 나타낸다. 기존의 RIFP 기법을 구현하기 위해서는 오류 블록 내부를 확장된 메쉬로 나누어야 한다. 확장된 메쉬는 오류 블록 내부를 특정한 조건을 만족시키며 확장시켜나가는 서브 메쉬(sub mesh)를 의미한다. 확장시키는 방향은 2차원 메쉬 구조를 크게 ± 0 방향(좌우), ± 1 방향(상하)으로 나누어 진행되며, *0*과 *1*은 확장해 나가는 '차원'으로 생각할 수 있다. 상기한 조건에 위배되는 경우는 다음의 세 가지이다.

① 임의의 확장된 메쉬 M_j가 $\pm k(k = 0, 1)$ 방향으로 오류 블록 B의 상한, 혹은 하한을 넘어서는 경우.

② 임의의 확장된 메쉬 M_j의 노드가 M_1, M_2, ..., M_{j-1}의 노드 중 하나와 중복이 되는 경우.

③ 임의의 확장된 메쉬 M_j의 오류가 난 노드의 수가 확장해 나가는 차원(k)보다 같거나 더 큰 경우.

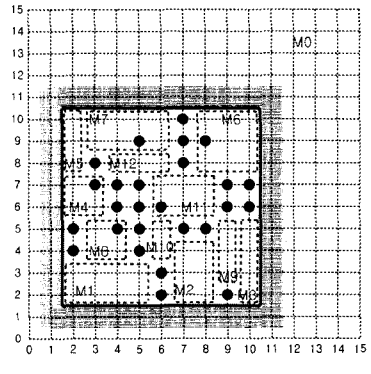

그림 2 그림 1에 대한 확장된 메쉬를 구성한 오류 블록

k는 2차원에서의 x, y 방향을 의미하며, 각각 0과 1의 값을 갖는다. 확장된 메쉬의 확장해 나가는 방향의 순서는 $-x$, $+x$, $-y$, 그리고 $+y$ 방향이다. 편의상 각각의 확장된 메쉬는 M_i로 표기되며 오류 블록 외부는 디폴트로 M_0이 된다. 오류 블록 내부에서 우선 순위가 가장 낮은 (즉, 좌표값이 가장 낮은) 정상적인 노드부터 확장된 메쉬는 오류 블록 내부에서 우선 순위가 가장 낮은 정상적인 노드부터 확장하기 시작하며, 이를 M_1로 지정한다. M_1의 확장이 끝나면 M_2로 넘어가고 더 이상 확장된 메쉬를 구성하지 못 할 때까지 계속한다. 그림 2는 그림 1에 의해 구성된 오류 블록을 확장된 메쉬로 구성한 것으로, 직선으로 처리된 오류 블록 내부에 점선으로 블록을 구성하는 것이 확장된 메쉬가 된다. 먼저 M_1은 오류 블록 내부의 좌표값이 가장 낮은 (2, 2)에서 확장이 $-x$(0차원) 방향으로 시작된다. 그러나, 오류 블록의 범위를 넘어가게 되어 상기한 조건의 첫 번째를 위배하게 되어 다시 (2, 2)로 돌아온다. 이번에는 $+x$(0차원) 방향으로 확장이 진행되며, (2, 6)에 도달하면 오류 노드의 수가 확장하는 차원의 수(0) 보다 크게 되며 이는 상기한 세 번째 조건에 위배됨으로 다시 (2, 5)로 돌아온다. 이제 $-y$(1차원) 방향으로 확장이 진행되지만 오류 블록의 범위를 벗어나게 됨으로 다시 (2, 2), (2, 5)로 돌아온다. 마지막으로 $+y$(1차원)으로 확장을 진행하게 되는데, (2, 4), (5, 4)까지 확장하게 되면 역시 상기한 세 번째 조건에 위배하게 되어 (2, 3), (5, 3)으로 돌아오게 되어 M_1의 범위를 결정하게 된다((2, 2), (5, 3)). 같은 방식으로 조건과 대조해 가며 확장을 진행하면 M_{12}까지 얻게된다.

2.2 알고리즘 RIFP와 문제점

알고리즘 RIFP를 구현하기 위해서는 먼저 확장된 메쉬를 기반으로 *DAG*를 구성해야 한다. 그림 3은 그림 2에서 구성된 확장된 메쉬를 가지고 *DAG*를 구성한 예이다. 여기서 V_i는 확장된 메쉬 M_i에 상응하는 정점(vertex)을 의미하고 시작은 M_0에 상응하는 V_0에서 시작하여 M_0과 인접하며 서로 경로가 존재하는 확장된 메쉬(즉, 이웃 메쉬)가 바로 밑의 레벨의 정점이 된다. 또 마찬가지 방법으로 M_0에 인접하며 경로가 존재하는 확장된 메쉬들은 오류 블록 내부의 나머지 확장된 메쉬들과 인접하고 서로 경로가 존재하는 경우 이들과 연결해 나가는 절차로 알고리즘이 진행된다.

실제 라우팅에서는 이렇게 구성된 *DAG*에서의 최단거리를 구하여 메시지를 전송을 시키게 되는데, *DAG*의 정점들의 관계를 이용하여, 예를 들어, M_i에 속해있는 노드 w_i가 M_k에 속해 있는 w_k로 메시지를 보내려면 어떤 정점(확장된 메쉬)들을 통과하여 가는 것이 가장 짧게 경로인가에 대한 값을 테이블에 저장한다. 메시지는 이 테이블을 참조하여 경로를 선택하게 되고, 또한, 가상 채널을 선택하게 된다. RIFP에서는, 오류 블록 외부에서는 Su와 Shin이 제안한 알고리즘을, 오류 블록 내부에서는 Glass와 Ni가 제안한 알고리즘을 이용하여 메시지를 전송한다.

위의 그림들을 토대로 한 라우팅의 예를 들어보면, 그림 4에서 (10, 15)에 위치한 노드를 소스 노드, (6, 7)에 위치한 노드를 목적 노드라고 가정하자. 소스 노드는 확장된 메쉬 M_0, 목적 노드는 확장된 메쉬 M_{11}에 속한다. 메시지는 오류 블록 외부에서 Su와 Shin의 알고리즘에 따라 전송이 되다가 오류 블록을 만나면 *DAG*에 의해 확장된 메쉬 M_0에서 M_{11}로 가는 최단거리를 참조한다. 이 예에서는 확장된 메쉬 M_6을 거쳐서 가는 것이 최단거리이므로 오류 블록을 만난 메시지는 확장된 메쉬 M_6으로 전송되어지고, 또 M_6에서 M_{11}로 전송되어진다.

위의 예에서, 메시지가 노드 (6, 11)에 도착한 후, *DAG*에서의 최단거리를 이용하기 때문에 바로 밑에 있는 목적 노드로의 실질적으로 더 짧은 직선적인 경로를 무시하고 우회하여 이동하는 것을 볼 수 있다. 이러한 존재 가능한 실질적으로 더 짧은 hop의 수를 갖는 경로를 무시하고 *DAG*에 의한 최단거리의 경로를 찾아 메

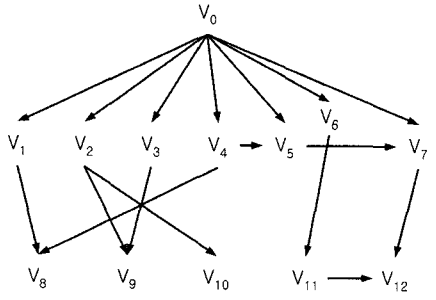

그림 3 그림 2에서 구성된 확장된 메쉬를 가지고 DAG를 구성한 결과

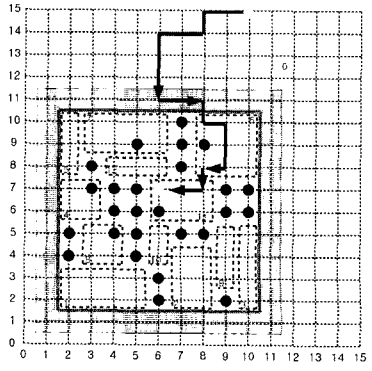
그림 4 RIFP를 적용한 그림 2의 하나의 예

시지를 전송하는 문제를, 메시지가 이를 통과해 전송되어질 수 있도록 해주는 제한된 조건을 제시하는 기법을 본 논문에서 제안하고자 한다.

3. SCP(Short-Cut Path)를 적용한 RIFP

3.1 기본 전략

오류 블록 내부에 위치하는 목적 노드에서부터 오류 블록 경계부분까지의 직선적인 경로가 존재한다고 가정하자. 본 논문에서는 이러한 경로를 가리켜 SCP (Short-Cut Path)라고 표현한다. SCP 기법의 순서를 간단히 설명하면 다음과 같다.

① 목적 노드에서 오류 블록 내부 경계까지의 직선 경로인 SCP를 탐색한다.

② 탐색에 성공하면 SCP의 오류 블록 내부 경계 노드에 인접하는 오류 블록 외부 경계 노드가 이에 대한 정보를 기억한다.

③ 메시지가 오류 블록 외부 경계에 처음으로 전송이 되어오면 SCP가 존재하는지를 확인한다.

④ SCP가 존재하면 SCP를 적용시킨다.

⑤ SCP가 존재하지 않으면 RIFP를 그대로 적용시킨다.

그림 5는 위에서 보여진 그림 4에서의 SCP를 보여주고 있다. 오류 블록 외부 경계로부터 목적 노드까지 가리키는 엷은 색 화살표가 SCP이다. 이러한 SCP의 탐색은 목적 노드와 목적 노드의 이웃 노드들로부터 오류 블록 내부 경계까지의 직선 경로를 찾는 것이다. 앞서 SCP는 제한된 조건으로 구현을 한다고 했고, 또 SCP 탐색은 목적 노드와 목적 노드의 이웃 노드에서만 실행을 한다고 했다. 바로 이 부분이 제한된 조건 중 하나인데, 그 이유는 목적 노드와 그 이웃의 제한을 벗어나면 그만큼 탐색하는데 오버헤드(overhead)가 더 커지기 때문이다. SCP는 기존의 알고리즘 RIFP에 조건을 추가하는 것이므로 오버헤드를 가능한 한 적게 하는 것이 중

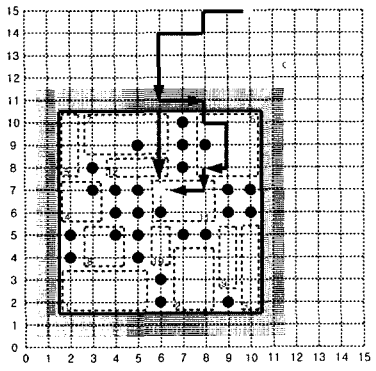

그림 5 SCP가 존재하는 그림 4의 예
(엷은 색 화살표)

[알고리즘. 1] SCP 탐색 알고리즘 : find_SCP

```
Algorithm find_SCP : // to the -0 direction
/* coordinate of ND is denoted as (d.x, d.y) */
/* the status of each node is denoted as mesh[x][y].status */
/* found is a variable which determines if SCP is found */
 1. If NS and ND are the same, then exit.
 2. Else
      for(i=d.x; i>=B.0.lb; i--) {
            2.1 if mesh[i][d.y].status is UNSAFE
                set found 1 and continue for loop
            2.2 else
                set found 0 and break out of for loop
      }
      if found is 1,
            return the address of node (B.0.lb-1, d.y)
      else
            return -1
/* do the same for d.y+1, and d.y-1 */
```

요한 사항이다. 직선 경로는 좌, 우, 상, 하 방향의 네가지 방향으로 탐색이 가능하다. 그러나, 탐색 방향은 목적 노드를 기준으로 소스 노드의 위치에 따라 결정이 되는데, 그 이유는, 언제나 네 방향으로 SCP를 탐색하는 것은 리소스의 낭비를 초래할 수 있거나 경우에 따라서는 SCP가 존재한다 하더라도 기존의 기법보다 더 우회하는 경로를 제공할 수도 있기 때문이다.

소스 노드의 위치에 따라 여덟 가지의 탐색 경우가 생기는데, 예를 들어, 소스 노드의 좌표를 $(s.x, s.y)$라고 하고 목적 노드의 좌표를 $(d.x, d.y)$라고 가정한다면, $((s.x<d.x)$ AND $(s.y<d.y))$, $((s.x<d.x)$ AND $(s.y>d.y))$, $((s.x<d.x)$ AND $(s.y=d.y))$, $((s.x>d.x)$ AND $(s.y<d.y))$, $((s.x>d.x)$ AND $(s.y>d.y))$, $((s.x>d.x)$ AND $(s.y=d.y))$, $((s.x=d.x)$ AND $(s.y<d.y))$, 그리고 $((s.x=d.x)$ AND $(s.y>d.y))$인 경우가 되겠다.

탐색한 결과 SCP가 존재할 경우 탐색이 끝난 부분(즉, 오류 블록 내부 경계)에 인접한 오류 블록 외부 경계에 위치한 노드는 자신이 SCP로 통과시켜주는 중간 노드임을 기억한다. 만약에 탐색한 결과 SCP가 존재하지 않을 경우에는 기존의 RIFP 기법을 그대로 적용하여 라우팅을 진행시킨다.

3.2 SCP 적용 여부 판단 조건

앞선 절에서 SCP가 존재한다고 가정하더라도 메시지를 SCP로 그냥 통과시키는 것이 RIFP만을 적용하여 메시지를 전송하는 것 보다 더 좋은 성능을 보장하지는 못하다. 그 이유로는 소스 노드에서 목적 노드로 메시지

를 전송 중 처음으로 오류 블록 외부 경계에 위치한 노드에 도착할 때 기존의 RIFP만을 적용해서 다음 확장된 메쉬로 가는 중간 노드가 SCP를 기억하는 오류 블록 외부 경계에 위치하는 노드보다 더 가까운 곳에 존재할 수 있는 가능성 때문이다. 예를 들면, 그림 6에서 볼 수 있듯이 목적 노드에서 오류 블록 외부 경계 사이에 SCP(위로 향한 굵은 화살표)가 존재하지만 기존의 RIFP 만을 적용해서 라우팅 하는 것이 이 경우에는 더 효율적이다. 이러한 이유로, 메시지 전송 중 처음으로 오류 블록 외부 경계에 위치한 노드에 메시지가 도착할 때 SCP가 같은 오류 블록 면에 존재할 경우에만 SCP를 적용한다. 그림 6의 경우를 예로 들면, SCP의 존재에 대한 정보를 기억하는 오류 블록 외부 경계에 위치한 노드의 좌표는 (6, 11)이다. 하지만 메시지가 처음으로 오류 블록 외부 경계에 도착한 노드의 좌표는 (11, 8)이다. 이 두 노드의 좌표값 중 같은 것이 없으므로 다른 면에 존재한다는 것을 알 수 있다. 이러한 경우에는 SCP를 적용시키지 않는다.

또 다른 예로써 그림 7의 경우를 고려해보면, 위의 예

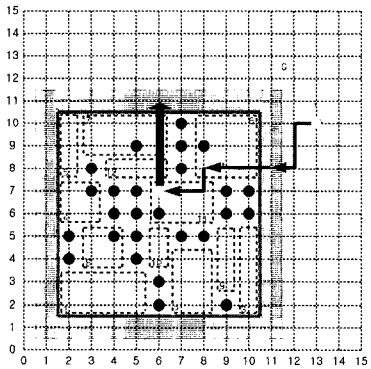

그림 6 SCP 적용 불가능 한 경우의 예
(굵은 화살표가 SCP)

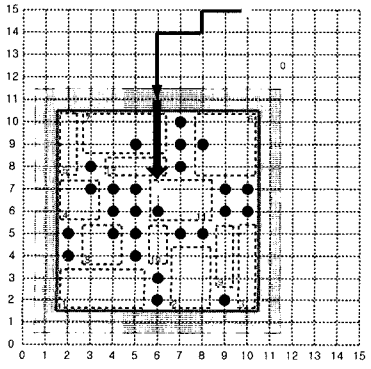

그림 7 SCP 적용 가능한 경우의 예

에서와 마찬가지로 SCP의 존재에 대한 정보를 기억하는 오류 블록 외부 경계에 위치한 노드의 좌표는 (6, 11)이고, 메시지가 처음으로 오류 블록 외부 경계에 도착한 노드의 좌표 역시 (6, 11)이다. 즉, 이 두 노드는 같은 면에 있고, 또 같은 노드임을 알 수 있다. 같은 노드일 필요는 없지만 좌표값 중 하나라도 같은 것이 있다면, 같은 면에 있다는 사실이므로, 이러한 경우에는 SCP를 적용시켜 메시지를 계속 전송시키게 된다. 위의 두 예에서와 같이, 본 논문에서는 오류 블록 외부에서 전송되어 온 메시지가 처음으로 오류 블록 외부 경계에 도착한 노드와 SCP에 대한 정보를 갖고 있는 노드가 같은 면에 있는 경우에 대해서만 SCP를 적용시키는 RIFP를 구현한다.

4. 성능 평가

기존의 RIFP 기법과 제안하는 SCP를 적용하는 RIFP 기법을 시뮬레이션을 통하여 성능을 비교 분석하였다. 우선, 시뮬레이션은 8×8, 16×16, 32×32 2차원 메쉬 구조에서 수행하였으며, 각각 메쉬 구조에서 4%, 6%, 8%, 그리고 10%의 오류율을 가지고 수행하였다. 이러한 환경은 메쉬 구조에서의 시뮬레이션에 사용되는 가장 일반적인 환경임을 반영한 것이다. 또한, 10개의 서로 다른 seed 값으로 random하게, 각 seed 값 당 20번씩 총 200번 반복 수행을 통한 결과를 수집하여 각 경우의 평균을 계산하였다. 시뮬레이션의 기본적인 환경으로 Intel Pentium III-450MHz CPU 기반으로 한 Windows NT Workstation에서 ANSI C Compiler를 통해 구현하였다.

본 논문에서의 성능 평가 비교 요소는 소스 노드에서 출발한 메시지가 목적 노드까지 도달하는데 걸리는 평균 지연 시간(Average Latency, 식 (1) 참조)과 평균 메시지 전송 hop의 수(Average number of hops)이다.

식 (1)에서 mnum은 mesh에서의 hop의 수이고, 각 hop들의 delay는 일정하다고 가정하였다. 평균 지연 시간은 오류 블록을 작은 mesh들로 나눈 후 SCP를 탐색하기 위해 이 sub mesh들을 거쳐가는 횟수로 평균 지연 시간을 계산하였다.

$$T_{average} = \frac{\sum_{m_{num}=1}^{tot-num} T_{m_{num}}}{tot-num} \qquad \text{(1) 평균지연시간}$$

먼저, 메시지 전송의 평균 지연 시간을 비교한다. 그림 8, 그림 9, 그리고 그림 10은 각각 8×8, 16×16, 32×32 2차원 메쉬 구조에서 오류율 4%, 6%, 8%, 10%를 적용하여 시뮬레이션을 수행하였을 때의 기존 기법과 제안한 기법의 평균 지연 시간의 비교를 보여준다. 우선 8×8 2차원 메쉬 구조에서는 기존의 RIFP 기법이 제안

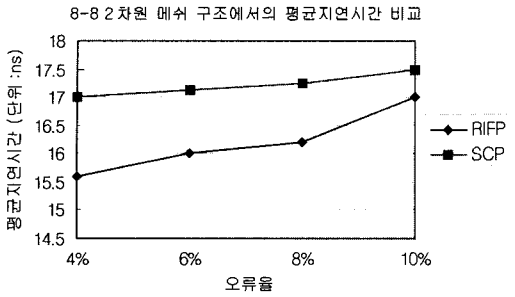

그림 8 8×8 2차원 메쉬 구조에서의 평균 지연 시간 비교

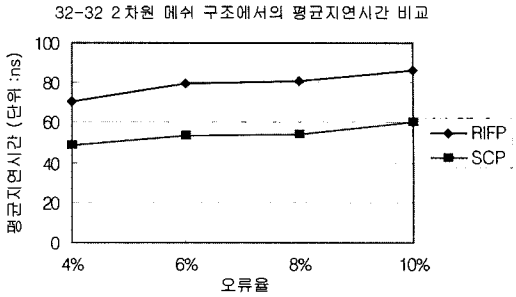

그림 10 32×32 2차원 메쉬 구조에서의 평균 지연 시간 비교

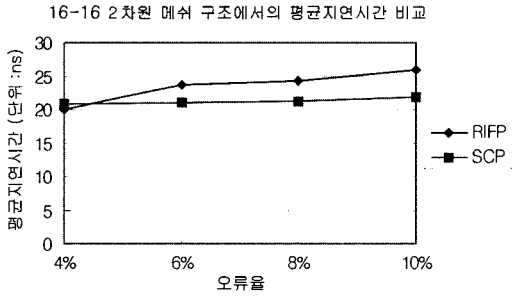

그림 9 16×16 2차원 메쉬 구조에서의 평균 지연 시간 비교

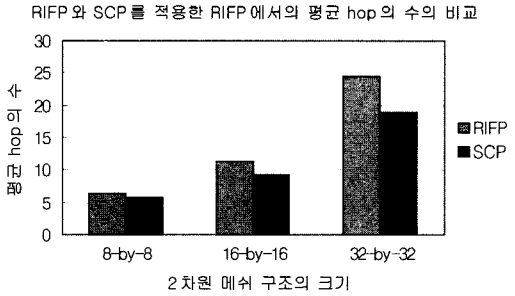

그림 11 RIFP와 SCP를 적용한 RIFP의 평균 hop의 수 비교

한 기법보다 더 짧은 평균 지연 시간을 보여주지만 16×16과 32×32 2차원 메쉬 구조에서는 제안한 기법이 더 짧은 평균 지연 시간을 보여준다. 그 이유는 8×8 2차원 메쉬 구조에서의 오류 블록의 크기는 비교적 작은 크기이므로 SCP 탐색을 하는 데 걸리는 시간이 기존의 RIFP만을 적용하여 수행하는 것 보다 조금 더 오래 걸리기 때문이다. 하지만 16×16 2차원 메쉬 구조에서 오류율 6%부터는 SCP를 적용한 라우팅을 수행하는 것이 기존 기법에서의 오류 블록을 우회하며 라우팅하는 것보다 더 짧은 평균 지연 시간을 갖게 된다. 기존의 RIFP는 오류 블록에 진입해서도 다소 우회하여 메시지를 전송하는데 반해, SCP를 적용한 RIFP에서는 메시지 전송이 직선적인 경로를 통하여 이루어지므로 그만큼 지연 시간이 짧게 걸린다. 전체적으로는 22% 정도의 평균 지연 시간 감소가 있었다.

이제, 기존의 기법과 제안한 기법의 hop의 수의 차이에 대해 비교한다. 그림 11은 오류율 6%에서 기존 기법과 제안한 기법의 평균 hop의 수의 차이를 비교하여 보여준다. 8×8 2차원 메쉬 구조의 경우에는 오류 블록의 크기가 작으므로 확장된 메쉬의 개수도 적고 또한 오류 블록 외부와 대부분의 확장된 메쉬가 인접해 있는 경우가 대부분이므로 hop의 수는 비교적 크지 않은 성능 향상을 보인다. 그러나, 16×16이나 32×32 2차원 메쉬 구조의 경우에는 8×8 2차원 메쉬 구조의 경우보다 오류 블록의 크기가 크므로 더 향상된 성능을 보인다. 그만큼 기존의 기법이 오류 블록의 크기가 커질수록 우회하여 가는 경로의 길이가 길어진다는 의미이다. 16×16과 32×32 2차원 메쉬 구조에서는 더 큰 성능 향상을 보여, 전체적으로 20% 정도의 평균 hop의 수의 감소가 있었다.

5. 결 론

대규모 응용 프로그램을 신속하게 수행하기 위해 멀티컴퓨터 시스템은 매우 중요한 역할을 하고 있다. 특히, 이러한 응용 프로그램들을 안정된 환경에서 수행되도록 보장하기 위해서는 수행 중에 시스템에서 발생하는 오류를 융통성 있게 처리할 수 있는 기술이 반드시 요구된다. 이러한 오류를 허용하는 기존의 기법들은 공통적으로 오류 블록을 구성하여 많은 수의 오류를 허용하는 알고리즘을 제안했다. 그러나 오류 블록 내부의 정상적인 노드 또한 오류 노드로 간주되어 노드 사용율의 저하를 초래한다. 이러한 문제는 오류 블록 내부를 확장된 메쉬로 나누고, 각 확장된 메쉬로 통하는 중간 노드를 구하여 해결하였다. 메시지는 확장된 메쉬를 *DAG*에 적용하여 구한 최단 거리를 이용하여 전송이 된다. 그러

나, 이 방법은 *DAG*에서 구한 최단 거리의 hop의 수가 오히려 증가하는 문제가 발생한다.

따라서, 본 논문에서는 증가하는 hop의 수를 억제하기 위하여 목적 노드와 이웃 노드들로부터 오류 블록의 경계 부분까지의 직선 경로인 SCP를 찾아 기존 기법의 단점을 해결하였다. 실험결과 8×8 2차원 메쉬 구조에서는 평균 hop의 수에서는 약 10% 정도의 감소를 보였으며, 16×16 2차원 메쉬 구조에서는 18% 감소하였다. 그리고, 32×32 2차원 메쉬 구조에서는 23%의 평균 hop의 수의 감소로 기존의 기법보다 전체적으로 20% 정도의 hop의 수의 감소를 가져왔다. 평균 지연 시간은 8×8 2차원 메쉬 구조에서는 오히려 6% 증가를 가져왔지만, 메쉬 구조의 크기가 커질수록 약 27% 정도의 감소로 제안한 기법은 메쉬 구조의 크기가 클수록 효과적임을 알 수 있었다.

참 고 문 헌

[1] K.M. Al-Tawil, M. Abd-Abd-Barr, and F. Ashraf, "A Survey and Comparison of Wormhole Routing Techniques in Mesh Networks," *IEEE Network*, pp.38-45, Mar. 1997.

[2] J. Duato, "A Theory of Fault-Tolerant Routing in Wormhole Networks," *IEEE Trans. Parallel and Distributed Systems*, vol.8, no.8, pp.790-802, Aug. 1997.

[3] C. Glass and L. Ni, "Fault-Tolerant Wormhole Routing in Meshes," *23rd Int'l Symp. Fault-Tolerant Computing*, pp.240-249, 1993.

[4] D. Linder and J. Harden, "An Adaptive and Fault-Tolerant Wormhole Routing Strategy for k-ary n-cubes," *IEEE Trans. Computers*, vol.40, pp.2-12, 1991.

[5] R.V. Boppana and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Trans. Computers*, vol.44, pp.848-864, 1995.

[6] C.C. Su and K.G. Shin, "Adaptive Fault-Tolerant Deadlock-Free Routing in Meshes and Hypercubes," *IEEE Trans. Computers*, vol.45, pp.666-683, June 1996.

[7] M.J. Tsai and S.D. Wang, "Adaptive and Deadlock-Free Routing for Irregular Faulty Patterns in Mesh Multicomputers," *IEEE Trans. Parallel and Distributed Systems*, vol.11, no.1, pp50-62, Jan. 2000.

정 성 우

1998년 서강대학교 컴퓨터학과 졸업
2001년 서강대학교 컴퓨터학과 석사
2001년~현재 (주) 디비엠코리아 재직
관심분야는 기획업무, 병렬처리시스템

김 성 천

정보과학회논문지 : 시스템 및 이론
제 30 권 제 1 호 참조