오류가 있는 메쉬 시스템에서의 프로세서 할당 기법

서 경 희†

요 약

오류가 발생할 수 있는 대규모 멀티컴퓨터 시스템의 프로세싱 자원들을 효율적으로 사용하기 위해서는 신뢰도 높은 프로세서 할당 알고리즘이 요구된다. 본 논문에서는 오류가 있는 메쉬 시스템의 성능을 높일 수 있는 동적이면서 신뢰도 높은 프로세서 할당 기법을 제안한다. 비경계 오류 노드들을 보상할 수 있는 오류프리 상한 또는 하한 경계 노드들을 사용해서 오류가 있는 메쉬 시스템을 최대 볼록 시스템으로 재구성한다. 이 재구성된 비직사각형 메쉬 시스템을 효율적으로 활용하기 위해 기존의 사각형 서브메쉬를 할당 할 수 없을 때 L-모양 서브메쉬를 할당할 수 있다. 시뮬레이션 결과를 통해서 제안하는 기법이 작업응답시간과 시스템 활용도 면에서 다른 기법들보다 우수함을 보인다.

키워드 : 재구성 메쉬, 최대 볼록 시스템, 단편화, 프로세서 할당

Processor Allocation Scheme on the Mesh-connected System with Faults

Kyung Hee Seo†

ABSTRACT

Efficient utilization of processing resources in a large multicomputer system with the possibility of fault occurrence depends on the reliable processor management scheme. This paper presents a dynamic and reliable processor allocation strategy to increase the performance of mesh-connected parallel systems with faulty processors. The basic idea is to reconfigure a faulty mesh system into a maximum convex system using the fault-free upper or lower boundary nodes to compensate for the non-boundary faulty nodes. To utilize the non-rectangular shaped system parts, our strategy tries to allocate L-shaped submeshes instead of signaling the allocation failure. Extensive simulations show that the strategy performs more efficiently than other strategies in terms of the job response time and the system utilization.

Key Words : Reconfigurable Mesh, Maximal Convex System, Fragmentation, Processor Allocation

1. 서 론

대규모 메쉬 시스템에서 오류가 발생할 확률은 높으므로, 시스템으로 계속해서 들어오는 작업들에게 프리 서브메쉬(free submesh)들을 할당해 주는 동적이면서 신뢰도 높은 프로세서 할당 기법에 대한 연구는 매우 중요하다. 오류가 있는 시스템을 효율적으로 활용하기 위해서 최대한 오류 프리(fault free)인 노들들로 재구성할 필요성이 있으며, 2, 3차원 메쉬들의 재구성에 관한 연구는 활발히 진행되어 왔다[3, 8, 14]. 재구성 메쉬(reconfigurable mesh)는 계산이나 통신의 필요에 대응해서 프로세서들 간에 형성되는 다양한 상호연결 유형들을 동적으로 제공해 줄 수 있는 재구성 가능한 버스 시스템으로 구성된 메쉬이다. 다른 병렬 컴퓨터 구조에 비해 재구성 메쉬의 장점은 문제에서 요구되는 특정 위상을 동적으로 바로 형성할 수 있다는 것이다.

프로세서 할당 기법은 각 작업들에게 독점적으로 이용할 수 있는 요구하는 크기의 서브메쉬를 할당하며, 작업들이 도착되는 순서대로 서비스를 하는 공정성을 유지하면서 시스템의 성능을 높이는 것을 목적으로 한다. 시스템의 성능을 높이기 위해서는 작업 응답 시간을 줄이면서, 동시에 시스템 활용도를 증가시킴으로써 가능해지며, 이를 위해서는 발생되는 단편화의 양을 최소화 시키는 것이 중요하다.

프로세서 할당 기법에서 발생할 수 있는 단편화의 종류는 내부, 외부, 그리고 가상 단편화이다. 초기의 할당 기법인 2차원 Buddy(2DB) 기법은 각 변이 2의 멱승인 정사각형 서브메쉬만을 할당할 수 있으므로 심각한 내부 단편화를 발생시킨다[9]. 대부분의 프로세서 할당 기법들은 직사각형 서브메쉬를 할당할 수 있으며, 이용 가능한 서브메쉬들을 인식할 수 있는 능력을 증진시킴으로써 내부 및 가상 단편화를 제거할 수 있지만[4, 5, 10, 12, 15, 16], 이런 시도로 얻을 수 있는 시스템 성능 향상의 정도는 크지 않다[13]. 시스템의

† 정 회 원 : 성신여자대학교 컴퓨터정보학부 초빙교원
논문접수 : 2005년 4월 20일, 심사완료 : 2005년 6월 20일

성능을 향상시키기 위해서는 외부 단편화 문제를 해결해야 한다. 외부 단편화는 시스템에 작업이 요구하는 프로세서들의 개수를 충분히 수용할 수 있는 프리 프로세서들이 존재하지만, 작업이 요구하는 크기를 갖는 이용 가능한 하나의 사각형 서브메쉬가 형성될 수 없을 때 발생한다. 외부 단편화를 줄이기 위한 최초의 시도는 AS(Adaptive Scan) 기법에서 찾을 수 있으며[5], 또 하나의 시도는 Flexfold 기법이다[6]. 그러나 Flexfold 기법을 포함한 기존의 사각형 서브메쉬 할당 알고리즘들은 전체 시스템 구조와 같은 위상을 가지는 사각형 서브메쉬를 발견하지 못하면, 작업을 할당할 수 없으며, 그 작업을 대기 큐에 넣는다. 시스템에서 할당해제가 발생했을 때, 대기 큐의 헤드에 있는 작업에게 할당할 수 있는 서브메쉬를 찾기 위한 시도가 이루어지므로, 그 작업이 가능한 빨리 처리되지 않으면, 병목현상이 심각해진다.

본 논문에서는 오류가 있는 단편화된 메쉬 시스템으로 들어오는 작업이 요구하는 서브메쉬의 모양을 유연성 있게 조작할 수 있는 동적이면서 신뢰도 높은 LSA (L-Shaped Submesh Allocation)기법을 제안한다. LSA 기법은 사각형 메쉬 시스템에서의 LSSA 기법[11]을 오류가 있는 메쉬 시스템에 적용하기 위해 확장시킨 것으로서, 비경계 오류 노드들을 보상할 수 있는 오류프리 상한 또는 하한 경계 노드들을 사용해서 오류가 있는 메쉬 시스템을 최대 볼록 시스템으로 재구성한다. 이 재구성된 비직사각형 메쉬 시스템을 효율적으로 활용하기 위해 할당 가능한 L-모양 서브메쉬의 탐색 프로시듀어를 이용하여 빠르고 효율적인 프로세서 할당을 제공한다. 특히, 대기 큐의 헤드에 있는 작업 때문에 다음 작업이 모양의 변형 없이 수용될 수 있는데도 불구하고 계속 기다리는 상황을 방지하고자 한다.

LSA 기법의 성능을 기존의 다른 기법들과 비교하기 위해서 이산 사건 중심의 시뮬레이터를 AweSim[1]을 사용하여 구성하였으며, LSA기법을 적용하여 작업 응답 시간을 줄이고, 시스템의 활용도가 증가함을 보인다. 본 논문의 구성은 다음과 같다. 2장에서 기존의 서브메쉬 할당 기법들을 간략하게 소개한다. 제안하는 LSA 기법을 3장에서 설명하고, 4장에서 LSA 기법의 성능을 시뮬레이션을 통해 분석한다. 5장에서 결론을 맺는다.

2. 기존의 할당 기법들

2DB 기법은 서브메쉬 $a \times b$를 요구하는 작업에 대해서 $2^k \times 2^k$, $k = \lceil \log(\max(a,b)) \rceil$, 프리 서브메쉬 만을 할당 할 수 있다. 따라서 직사각형 모양의 메쉬 시스템에는 적용할 수 없으며, 심각한 내부 단편화를 발생시킨다[9]. 2DB 기법의 단점을 보완하기 위해서, Frame Sliding(FS) 기법이 제안되었다[4]. 이 기법은 직사각형 모양의 메쉬 시스템에도 적용 가능하며, 작업이 요구하는 크기와 같은 크기의 서브메쉬를 할당함으로써 내부 단편화를 발생시키지 않는다. 그러나 고정된 stride를 사용하여 프레임을 찾기 때문에 가상 단편화를 발생시키며, 또한 외부 단편화도 많이 발생시킨다.

FF(First Fit) 와 BF(Best Fit) 기법들은 직사각형 모양의 메쉬 시스템에 적용 가능하며, 내부 단편화도 발생시키지 않는다[16]. 할당 가능한 서브메쉬를 빨리 찾기 위해서, 각 프로세서를 각 비트에 대응시키는 두 개의 2차원 배열, 비지(busy) 배열과 적용범위(coverage) 배열을 사용한다. 그러나 요청된 서브메쉬를 할당할 수 없을 때, 90° 회전시킨 서브메쉬를 고려하지 않으므로, 가상 단편화를 발생시키며, 외부 단편화도 심각하다.

FS 기법에서 발생되는 가상 단편화를 제거하기 위해서 제안된 AS 기법은[5] 고정된 stride를 사용하지 않고, 적응적으로 스캔함으로써 할당 가능한 서브메쉬를 완전하게 인식할 수 있으며, 요청된 서브메쉬를 할당할 수 없을 때, 90° 회전한 서브메쉬를 할당하려는 시도를 처음으로 하였다. 그러나 이 할당 알고리즘의 최초적합 특성 등으로 인해, 여전히 외부 단편화 문제가 심각하게 남아있다.

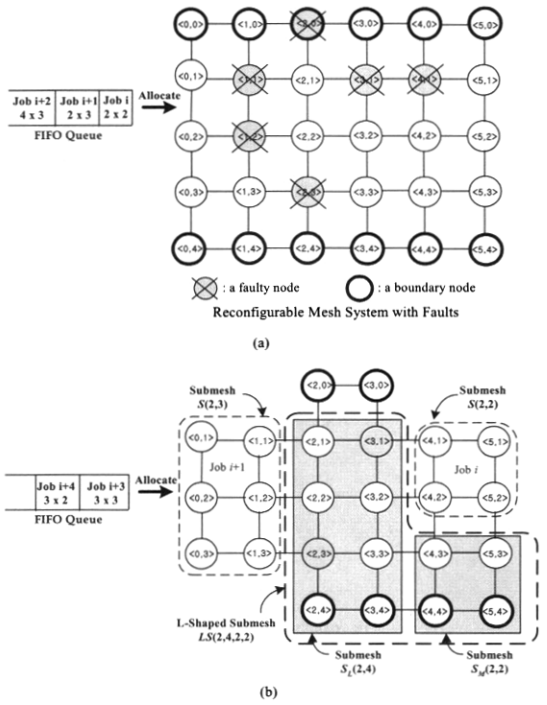
요청된 서브메쉬의 모양을 90° 회전시키는 것 외에, 폴디드(folded) 서브메쉬로 변환하여 할당을 고려하는 Flexfold 기법[6]은 다른 기법들보다 더 많은 할당 기회를 제공한다. 그러나 Flexfold 기법에서 사용되는 folding은 요청된 서브메쉬의 두 변 a 와 b가 모두 짝수일 때만 적용될 수 있다. 또한 할당할 수 있는 서브메쉬는 사각형 모양이어야 하므로, 여전히 외부 단편화의 발생량이 높게 되어, 시스템의 활용도는 높지 않다.

LSSA 기법은 요청된 사각형 서브메쉬를 기존의 기법으로 할당할 수 없을 때, L-모양 서브메쉬로 변환하여 프로세서 할당을 수행하므로, 내부, 가상 단편화를 발생시키지 않으며, 외부 단편화 발생량도 줄인다[11]. 그러나 평균 작업 응답 시간을 줄이기 위해 할당 가능한 L-모양 서브메쉬를 탐색하는 과정을 효율적으로 개선할 수 있으며, 사각형 서브메쉬에서 L-모양 서브메쉬로의 변형을 형식화하여 효율적인 탐색 알고리즘을 제공함으로써 시스템의 성능을 개선할 수 있는 가능성이 있다.

3. LSA 기법

3.1 L-모양 서브메쉬

재구성 가능한 버스를 갖는 프로세서 배열 중에서 가장 일반적인 w x h 재구성 메쉬는 격자 모양의 재구성 가능한 버스로 상호연결된 w x h 메쉬로 구성된다. w x h 메쉬 $M(w,h)$ 는 wh 개의 노드들로 구성되며, 너비 w와 높이 h를 가지는 사각형의 격자 모양을 가진다. 메쉬 시스템에서 각 프로세서는 각 노드에 대응된다. 열 i 와 행 j에 위치한 노드의 주소는 $<i, j>$로 표기하며, 행과 열들은 메쉬의 상단-좌측 모서리부터 셈하기 시작한다. 오류가 있는 메쉬 시스템에서의 L-모양 서브메쉬 할당을 보이고 있는 (그림 1)에서 2차원 메쉬 $M(6,5)$와 각 노드들의 주소를 나타내고 있다. 메쉬 $M(w,h)$에 대해서 너비 a와 높이 b를 가지는 서브메쉬 $S(a,b)$는 $a \le w$ 이고 $b \le h$을 만족하면서 $M(w,h)$에 포함되는 2차원 메쉬이다. 서브메쉬 $S(a,b)$의 주소는 $<x_1,y_1,x_2,y_2>$



(그림 1) 재구성 메쉬 시스템에서의 프로세서 할당

로 나타내며, 여기서 $<x_1,y_1>$은 그 서브메쉬의 하단-좌측 모서리를 가리키며, $<x_2,y_2>$는 상단-우측 모서리를 가리킨다. (그림 1) (b)에서 주소가 <0,3,1,1>인 서브메쉬 S(2,3)를 보이고 있다.

주소가 각각$<x_{11},y_{11},x_{12},y_{12}>$이고, $<x_{21},y_{21},x_{22},y_{22}>$인 임의의 두 L-adjacent한 서브메쉬 $S(c,d)$ 와 $S(e,f)$는 다음 조건들 중의 하나를 만족한다. 1) $x_{21} = x_{12} + 1$ 이고 $y_{21} = y_{11}$ 2) $x_{11} = x_{22} + 1$ 이고 $y_{11} = y_{21}$, 3) $x_{11} = x_{22} + 1$ 이고 $y_{12} = y_{22}$, 4) $x_{21} = x_{12} + 1$ 이고 $y_{22} = y_{12}$. (그림 2)에서 어떤 조건을 만족하는가에 따른 4가지 L-adjacent 경우들을 보이고 있다. 두 L-adjacent 서브메쉬에 대하여, 더 큰 서브메쉬 $S(c,d)$를 $S_L(c,d)$로 표기하며, 작은 서브메쉬 $S(e,f)$를 $S_M(e,f)$로 나타낸다. (그림 1)에서 주소 <2,4,3,1>인 서브메쉬 $S(2,4)$와 주소 <4,4,5,3>인 서브메쉬 $S(2,2)$는 위의 첫 번째 조건을 만족하는 L-adjacent한 경우로 $S_L(2,4)$와 $S_M(2,2)$로 각각 표기할 수 있다.

L-모양 서브메쉬, LSM, $LS(c,d,e,f)$은 긴 너비가 $c+e$이고, 긴 높이 d를 가지는 영문자 L자 모양의 블록으로서, 두 개의 L-adjacent 서브메쉬 $S_L(c,d)$ 와 $S_M(e,f)$의 프로세서들의 집합으로 구성된다. LSM $LS(c,d,e,f)$는 [$<x_{11},y_{11},x_{12},y_{12}>$ & $<x_{21},y_{21},x_{22},y_{22}>$]로 표기하며, 여기서 $<x_{11},y_{11}>$ 과 $<x_{12},y_{12}>$는 서브메쉬 $S_L(c,d)$의 하단-좌측 모서리와 상단-우측 모서리를, 그리고 $<x_{21},y_{21}>$ 과 $<x_{22},y_{22}>$는 서브메쉬 $S_M(e,f)$의 하단-좌측 모서리와 상단-우측 모서리를 각각 나타낸다. (그림 1) (b)에서 두 L-adjacent 서브메쉬 $S_L(2,4)$와 $S_M(2,2)$는 긴 너비 4와 긴 높이 4를 가지는 LSM $LS(2,4,2,2)$을 구성하며, 좌표는 [<2,4,3,1> & <4,4,5,3>]로 표시된다.

LSM $LS(c,d,e,f)$는 서브메쉬 $S(a,b)$로부터 만들어지며, 이때 $LS(c,d,e,f)$를 구성하는 노드들의 개수와 $S(a,b)$의 노드들의 개수는 같고, 즉, $cd+ef = ab$ 이고, 또한 $c+e = a$ 가 성립한다. $LS(c,d,e,f)$는 서브메쉬 $S(a,b)$의 한 변을 이분하여 만들어지는데, 이 커팅 사이드(cutting side)는 a, b 중에서 짝수인 변으로 결정되며, 두 변 모두 짝수이면 더 긴 변으로, 모두 홀수이면 더 작은 변으로 결정되며, 짝·홀에 따라서 LSM이 만들어지는 과정은 다음과 같다:

LSM의 구성 프로시듀어

Step 1: $S(a, b)$의 커팅 사이드 결정.

if ab is even, then the larger even side is chosen as the cutting side

else, the smaller odd side is the cutting side.

Step 2: $LS(c,d,e,f)$의 구성

If the cutting side a is even,

then {

1) Determine the value of c such that $c = e = a/2$

2) An upper-right part $S(e, b - f)$of the submesh $S(a, b)$ is cut and attached to the top of left remaining submesh $S_L(c, b)$

}

else {

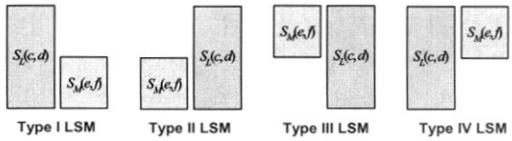
1) Determine the value of c such that $c = b - f$

2) An upper-right part $S(e, b-f)$ of the submesh $S(a, b)$ is cut and its rotated submesh $S(b-f, e)$ is attached to the top of $S_L(c, b)$.

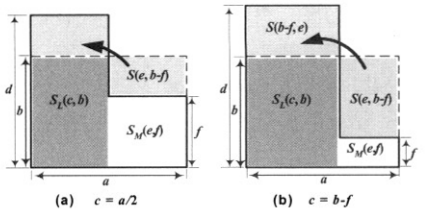
}

커팅 사이드 a가 짝수일 때, 서브메쉬 $S(a,b)$의 1사분면에 속하는 서브메쉬 $S(e, b-f)$를 잘라서 (여기서, $c = e = a/2$) $S_L(c,b)$의 상단에 다시 붙임으로써 $LS(c,d,e,f)$가 만들어진다. 이 과정을 (그림 3) (a)에서 보여주고 있으며, 만들어질 수 있는 LSM들은 $LS(a/2, b+ k, a/2, b- k)$, $1 \le k \le b-1$이고, $k=b$인 경우에는 폴디드 서브메쉬이다. 커팅 사이드 a가 홀수일 때, 서브메쉬 $S(a,b)$의 1사분면에 속하는 $S(e, b-f)$를 잘라서, $b-f = c$, 90° 회전한 서브메쉬 $S(b-f, e)$를 $S_L(c,b)$의 상단에 다시 붙임으로써 $LS(c,d,e,f)$가 만들어진다. 이 과정을 (그림 3) (b)에서 보여주고 있으며, 가능한 LSM들은 $LS(\lceil a/2 \rceil + k, b+\lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k, b-\lceil a/2 \rceil - k)$ $(0 \le k \le b-1-\lceil a/2 \rceil)$이다.

사각형 서브메쉬에서 어느 사분면이 존재하지 않는가에 따라서 네 가지 유형의 LSM들, (그림 2)에서 보이듯이, Type I LSM, Type II LSM, Type III LSM, 그리고 Type IV LSM이 존재한다. 직사각형 서브메쉬는 긴 높이 d와 짧은 높이 f가 같은 LSM의 특별한 경우로 간주 될 수 있다.



(그림 2) 두 L-adjacent 서브메쉬들로 구성될 수 있는 L-모양 서브메쉬들



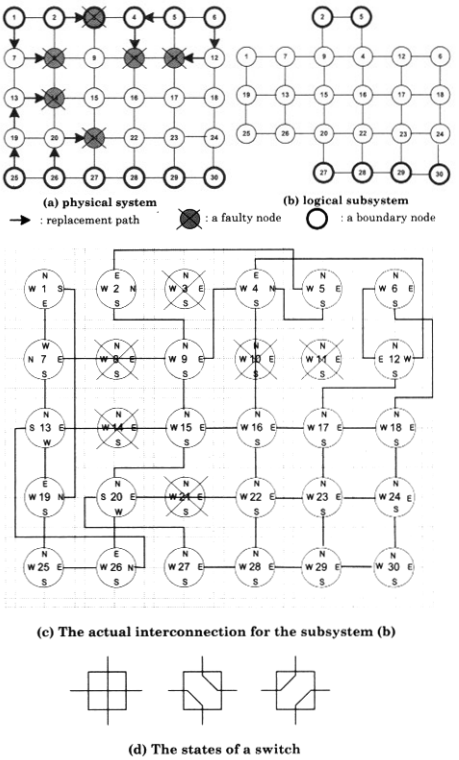
(그림 3) 서브메쉬 S(a,b)로부터 LS(c,d,e,f)의 구성 (a) a가 짝수인 경우 (b) a가 홀수인 경우

3.2 오류가 있는 메쉬 시스템의 재구성

오류 노드들이 존재하는 시스템을 재구성하지 않은 상태로 모든 오류 프리 노드들을 계속해서 사용하는 경우에는 간단한 데드락 프리인 경로배정 알고리즘을 유도할 수 없다[14]. 따라서 모든 오류 프리 노드들을 포함하도록 메쉬 시스템을 재구성하는 것이 더 효율적이다. 재구성 하드웨어 오버헤드는 낮게 유지하면서 재구성 후의 시스템 사이즈는 최대한 크도록 오류가 있는 시스템을 재구성하는 것은 매우 바람직하다. 대부분의 재구성 기법은 타깃 서브시스템이 직사각형 또는 정사각형이 되도록 요구하므로, 시스템에서의 오류 프리 노드들의 활용을 낮게 만든다. LSA 기법은 오류 프리 노드들의 활용을 거의 최적화할 수 있는 최대 재구성을 보장하면서, 재구성 후에 가능한 많은 정상 노드들을 보유할 수 있는 메쉬 재구성 기법을 도입한다. 재구성된 메쉬 시스템은 직사각형 또는 정사각형 모양으로 제한되지 않으며 볼록 모양이 보장된다. 볼록 2차원 네트워크에서는 메시지를 항상 최단경로를 사용해서 전송할 수 있는 데드락(deadlock) 프리인 간단한 경로배정 알고리즘이 존재한다[14].

오류가 있는 메쉬 시스템을 최대 볼록 시스템으로 재구성하기 위해서 내부 오류 노드들을 겹치지 않는 대체 경로를 통해 보상할 수 있는 오류 프리 상한, 하한 경계 노드들을 사용한다. 겹치지 않는 대체 경로를 찾는 문제는 효율적인 알고리즘이 존재하는 min-cost flow 문제로 변환될 수 있다[14]. 경계 노드는 내부 노드와 같은 역할을 하지만, 오류가 발생하면 사용되지 않거나, 연결 노드로 바뀌게 된다. 내부 오류 노드들만 오류 프리 경계 노드들로 대체될 필요가 있다.

(그림 4)에서 오류가 있는 메쉬 시스템 M(6,5)의 재구성을 보이고 있으며, 경계 노드들은 굵은 선으로 표시했고, 오류 노드들은 X표로 표시했다. 내부 오류 노드와 오류 프리 경계 노드를 연결하는 대체 경로는 유향 간선으로 표시했다. 대체 경로에 있는 각 노드는 대응하는 내부 오류 노드 방향으로 한자리씩 논리적으로 이동한다. 예를 들어, (그림 4) (a)에서 보이듯이, 오류 노드 8은 노드 7로 대체되며, 노드 7은 노드 1로 차례로 대체된다. 네 개의 노드 27, 28, 29, 와 30이 하한 경계 노드들로 남으며, 노드 2와 5가 상한 경계 노드들이 된다. 이들 노드들은 연속적이며, 최종 서브시스템에 포함되고, (그림 4) (b)와 같은 최대 볼록 서브시스템이 구성된다. 오류 발생 시 재구성을 수행하기 위해서 각 채널당 두 개의 수평 트랙들과 수직 트랙들이 필요하며, 각 노드는 네 개의 I/O 포트, North(N), East(E), South(S), 와



(그림 4) 오류가 있는 메쉬 시스템 M(6,5)의 재구성

West(W)를 가진다. 이 볼록 시스템의 실제 상호연결은 (그림 4) (c)에서와 같으며, 각 교차점이 스위치에 대응된다. 스위치가 가질 수 있는 모든 가능한 상태들은 (그림 4) (d)와 같다. 노드가 벤트 대체 경로(bent replacement path)에 있을 경우에는 두 이웃하는 포트 위치가 바뀔 수 있다. 예를 들어, (그림 4)에서 노드 1이 벤트 대체 경로에 있고, 논리적 서브시스템에서 노드 1과 노드 7을 연결하는 간선이 수평이므로, 노드 1에서 E와 S 포트의 위치가 바뀌었다. 또한, 대체 경로에 있는 노드는 두 개의 유향 간선, 즉, 진입 간선과 출력 간선을 가지며, 네 방향, 왼쪽, 오른쪽, 위, 아래 중의 하나를 취한다.

3.3 적응적인 프로세서 할당

오류를 가진 메쉬 시스템의 최대 재구성이 비실시간으로 이루어진 후에 이 비사각형 모양의 시스템을 활용하기 위해 L-모양 서브메쉬를 할당하려는 시도를 한다. LSA 기법은 다음 예에서와 같이 작업 응답 시간과 외부 단편화를 동시에 감소시키는데 초점을 맞추고 있다. (그림 1)에서 재구성된 메쉬 시스템을 2개의 작업들이 공유하고 있으며, 각 작업은 점선으로 표시한 영역의 프로세서들을 다른 작업과는 독립적으로 그 작업을 완전히 끝낼 때까지 사용할 수 있다. 각 작업이 요구한 서브메쉬의 크기는 2×2와 2×3 이고, 각 작업에 할당된 서브메쉬의 좌표는 각각 <4,2,5,1>과 <0,3,1,1> 이다.

시스템으로 들어오는 새로운 작업 i+2가 서브메쉬 $S(4,3)$를 요구했을 때, 기존의 기법들은 이 작업을 즉시 수용할 수 없다. 현재 메쉬 시스템에는 14개의 이용 가능한 프로세

서들이 존재함에도 불구하고, 기존의 기법들은 작업 i+2를 대기 큐에 넣을 수밖에 없다. 그러나 LSA 기법은 $S(4,3)$이 LSM $LS(2,4,2,2)$로 변형될 수 있으며, 현재의 메쉬 시스템에서 이용 가능하며, 그 좌표가 [<2,4,3,1> & <4,4,5,3>] 임을 인식할 수 있다. 이 상황에서, LSA 기법은 다른 기법들 보다 빨리 태스크를 수용할 수 있으며, 시스템의 활용도도 91%로 향상시킬 수 있다.

LSA 알고리즘은 다음과 같다. 태스크가 서브메쉬 $S(a,b)$를 요구하면 첫째, 현재 시스템에 남아있는 프리 프로세서들의 개수가 ab 보다 크거나 같은지를 검사한다. 프리 프로세서들의 개수가 ab 보다 작으면, 즉시 할당할 수 없다는 신호를 보내고, 그 태스크를 대기 큐에 넣는다. 두 번째 단계는 태스크가 요구하는 서브메쉬가 정사각형인지를 결정하여, Procedure *DetermineSearchSequence*에 의해 a, b의 값에 따라 후보 프리 서브메쉬들의 탐색순서를 결정한다. Procedure *L-shaping*은 이용 가능한 LSM을 찾는다.

Procedure *DetermineSearchSequence*(a, b, ct)
/* a, b: 서브메쉬의 두변, ct:커팅사이드 */
if ($a = b$) then
{ /* square submesh */
case (a is even) : $S(a, b)$, $S(a/2,2b)$, $S(2b,a/2)$, LSMs using *L-shaping*(a)
case (a is odd) : $S(a, b)$, LSMs using *L-shaping*(a)
}
else
{
case (both a and b are even) : $S(a, b)$, $S(b, a)$, $S(a/2, 2b)$, $S(2b, a/2)$, $S(2a, b/2)$, $S(b/2, 2a)$, LSMs using *L-shaping*(ct)
case (both a and b are odd) : $S(a, b)$, $S(b, a)$, LSMs using *L-shaping*(ct)
case (a is even and b is odd) : $S(a, b)$, $S(b, a)$, $S(a/2, 2b)$, $S(2b, a/2)$, LSMs using *L-shaping*(ct)
case (a is odd and b is even) : $S(a, b)$, $S(b, a)$, $S(2a, b/2)$, $S(b/2, 2a)$, LSMs using *L-shaping*(ct)
}

Procedure *L-shaping*(ct) /* ct is a cutting side */
case : ct is even
if ($b \geq 4$), then $LS(a/2, b+k, a/2, b-k)$, $k = \lfloor b/4 \rfloor$, $\lfloor b/2 \rfloor$, $\lfloor 3b/4 \rfloor$
else $LS(a/2, b+k, a/2, b-k)$, $1 \leq k \leq b-1$.
case : ct is odd
if ($b \geq 9$), then $LS(\lceil a/2 \rceil + k, b+\lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k, b-\lceil a/2 \rceil - k)$, $k = 0, \lfloor b/4 \rfloor$, $b-1-\lceil a/2 \rceil$,
else $LS(\lceil a/2 \rceil + k, b+\lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k, b-\lceil a/2 \rceil - k)$, $0 \leq k \leq b-2-\lceil a/2 \rceil$.

LSAFF(LSA with First Fit)와 LSABF(LSA with Best Fit) 기법은 할당 가능한 서브메쉬를 탐색하는 과정에서 탐색하게 될 LSM들의 가짓수를 제한하고, 기존의 내부 단편화를 발생시키지 않는 할당 기법들 중에서 가장 간단한 FF와 BF 기법을 응용한 효율적인 할당 알고리즘이다. 또한, FF 와 BF 기법이 가지는 단점인 가상 단편화를 LSAFF 와 LSABF 기법에서는 효과적으로 제거할 수 있으며, 이들 기법의 장점인 낮은 할당·해제 시간 복잡도를 그대로 유지한다. $LS(c,d,e,f)$를 할당하기 위해서, $S_L(c,d)$를 FF 또는 BF 기법을 적용하여 할당할 수 있는 좌표를 찾은 후에, 그 좌표를 이용하여 $S_M(e,f)$를 프레임으로 간주하여 할당할 수 있는지 탐색한다. 이 탐색 과정 중에서 할당 가능한 프리 서브메쉬를 발견해서 할당이 성공적으로 이루어질 수 있으면, 태스크 분배기는 그 프리 서브메쉬의 크기, 위치와 함께 가중치 α, $\alpha \in \{1, 4, a+2, 6+4k\}$를 결정한다. 이 가중치 α는 임베딩 비용이며, 그 할당이 네트워크 전파 시간을 얼마나 증가시키게 할지를 반영한다. 시스템 $M(w,h)$로 들어오는 작업 $S(a,b)$에 대해 LSAFF와 LSABF 기법의 할당 시간 복잡도는 $\theta(wh)$이고, 해제 시간 복잡도는 $\theta(ab)$이다.

LSA 기법은 변형된 서브메쉬를 할당하는 것이 예상되는 네트워크 전파 시간의 증가로 인해서, 대기 큐에서 기다리는 것보다도 작업의 응답시간을 증가시키게 되는 것으로 판단되면, 그 작업에게 변형된 모양의 서브메쉬를 할당하지 않는 보존 검사(conservative check)를 수행한다. 시스템으로 들어오는 각 작업 i에 대해서 t_i는 통계적으로 예측된 수행시간을 나타내며, f_i는 통신 프랙션으로서, 형태 변형으로 인해 영향을 받게 될 t_i의 프랙션이다. 시스템의 구조적인 특성, 예를 들어, 엔드-노드 프로세싱 시간에 대한 네트워크 전파 시간의 비율이 f_i에 대한 상한선을 결정한다. 변형된 모양의 서브메쉬를 할당할 경우의 예상되는 수행시간의 증가분은 $\alpha f_i t_i$로서, 이와 비교하여 할당여부를 결정한다. 예를 들어, (그림 1)에서 작업 i, 작업 i+1의 순서로 끝나도록 스케쥴되어 있다고 가정할 때, 새로운 작업 i+2가 들어와서 $S(4,3)$를 요구했을 경우, LSA 기법은 작업 i가 $6f_{i+2}t_{i+2}$ 단위 시간 내에 끝나지 않는 경우에만 $LS(2,4,2,2)$를 할당하도록 결정한다. LSA 기법은 이와 같은 보존 검사를 하므로, 변형된 모양의 서브메쉬를 할당하지 않을 수도 있다.

4. 성능 분석

4.1 시스템 모델

LSA 기법을 기존의 서브메쉬 할당 기법들의 성능과 비교하기 위해서 AweSim을 사용해서 이산 사건 중심의 시뮬레이션을 수행하였다. 이 이산 사건 시뮬레이터는 2차원 메쉬 시스템에서 일련의 작업들이 도착되고, 서비스 받고, 떠나는 과정을 공정성을 유지할 수 있는 FCFS 스케쥴링을 채택하여 모델링 하였다. 시스템의 크기는 32 x 32이고, 작업부하는 태스크들의 도착되는 시간간격의 분포, 태스크 서비스 시간의 분포, 태스크 크기의 분포로 구성된다. 태스크들의 도착되는 시간간격과 태스크 서비스 시간의 분포는 대부분 지수 분포로 가정되지만, bimodal hyper-exponential, three-stage hyper-exponential 분포도 제안되어 있다[7]. 본 시뮬레이터에서는 시스템의 일반적인 동작 환경을 수용할 수 있는 전통적인 작업부하 모델을 채택하였다. 도착되는 시간간격에 의해 결정되는 새로운 태스크는 요청하는 서브메쉬의 크기, 요청하는 체재시간, 그리고 통신 프랙션에 의해 특징지어진다[6]. 요청하는 서브메쉬의 각 변의 길이는 수행될 작업의 성격을 모르는 상황을 반영할 수 있는 균일 분포를 사용해서 발생시킨다. 체재 시간은 요청된 크기의 서브메쉬

를 그대로 할당 받았을 때 시스템에서 그 응용 프로그램이 수행되는 시간이다. 태스크의 크기와 그 태스크의 요청된 체재시간은 서로 독립적이라고 가정한다. 통신 프랙션은 요청된 서브메쉬의 형태변형에 의해 초래되는 수행시간의 증가분을 나타낸다. 공정성을 유지하기 위해 태스크들은 FCFS 스케쥴링으로 처리되며, 하나의 태스크의 할당해제가 발생하면, 대기 큐의 헤드에 있던 태스크의 할당이 시도된다.

시스템이 불안정해지는 상황을 피하기 위해, 시스템 부하는 1을 초과하지 않도록 패러미터를 선택하여 다음과 같이 정의하였다 [6]:

$$\text{시스템부하} = \frac{\text{평균체재시간} \times \text{평균요청된서브메쉬크기}}{\text{시스템크기} \times \text{평균도착되는시간간격}} \quad (1)$$

시스템의 활용도는 메쉬 시스템에 있는 프로세서들이 평균적으로 얼마나 사용되고 있는지를 나타낸다. 시스템의 활용도를 측정하기 위해서 매 단위시간 마다 프리 프로세서들의 개수를 측정한다. 각 단위 시간 i에 대해서 n_i 개의 프로세서들이 프리 상태에 있고, 시뮬레이션을 t 단위 시간 동안 수행하였을 때, 시스템의 활용도는 다음과 같이 정의 된다[15]:

$$\text{시스템 활용도} = \frac{\sum_{i=1}^{t}(wh - n_i)}{wht} \quad (2)$$

여기서, wh는 전체 메쉬 시스템의 크기를 의미한다. 성능 분석을 하기 위해 평균 작업 응답 시간 (MRT : Mean Response Time)과 시스템 활용도를 측정하였다.

시뮬레이터는 할당기법에 따라 다른 모양의 프리 서브메쉬들을 탐색하며, 다음과 같은 할당 기법들을 모델링하여 시뮬레이터에 포함하였다.

• FF

오직 요청된 크기와 모양을 가지는 서브메쉬만을 할당하려고 시도한다. 이 태스크를 수용할 수 있는 프리 서브메쉬들 중에서 알고리즘이 가장 처음에 찾게 되는 프리 서브메쉬로 결정된다.

• BF

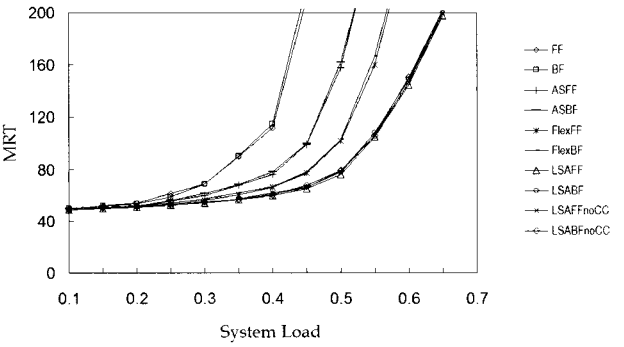
오직 요청된 크기와 모양을 가지는 서브메쉬만을 할당하려고 시도한다. 이 태스크를 수용할 수 있는 프리 서브메쉬들 중에서 가장 단편화를 적게 생성하게 되는 프리 서브메쉬로 결정된다.

• ASFF (Adaptive Scan with First Fit)

요청된 모양의 서브메쉬와 이의 90° 회전된 모양의 직사각형 서브메쉬를 할당하려고 시도한다. 이 태스크를 수용할 수 있는 프리 서브메쉬들 중에서 알고리즘이 가장 처음에 찾게 되는 프리 서브메쉬로 결정된다.

• ASBF (Adaptive Scan with Best Fit)

요청된 모양의 서브메쉬와 이의 90° 회전된 모양의 직사각형 서브메쉬를 할당하려고 시도한다. 이 태스크를 수용할 수 있는 프리 서브메쉬들 중에서 가장 단편화를 적게 생성하게 되는 프리 서브메쉬로 결정된다.



(그림 5) 시스템 부하의 변화에 따른 MRT의 개선도(평균 체재시간=50, 작업이 도착되는 간격을 변화시킴)

• FlexFF(Flexfold with First Fit)

요청된 모양의 서브메쉬와 이의 90° 회전된 모양의 직사각형 서브메쉬, 그리고 요청된 서브메쉬의 두 변이 모두 짝수인 경우에, 짝수인 변에 대한 폴디드 서브메쉬와 이를 90° 회전시킨 서브메쉬를 할당하려고 시도한다. 위 순서대로 탐색해 가는 과정에서 가장 처음에 찾게 되는 이 태스크를 수용할 수 있는 프리 서브메쉬로 결정된다.

• FlexBF(Flexfold with Best Fit)

FlexFF에서와 같이 탐색을 하면서 만나게 되는, 태스크를 수용할 수 있는 프리 서브메쉬들 중에서 가장 단편화를 적게 생성하게 되는 프리 서브메쉬로 결정된다.

• LSAFF(LSA with First Fit)

요청된 모양의 서브메쉬와 이의 90° 회전된 모양의 직사각형 서브메쉬, 짝수인 각 변에 대한 폴디드 서브메쉬와 이를 90° 회전시킨 서브메쉬, 그리고 LSM을 할당하려고 시도한다. 위 순서대로 탐색해 가는 과정에서 가장 처음에 찾게 되는 이 태스크를 수용할 수 있는 프리 서브메쉬로 결정된다. 결정된 서브메쉬가 폴디드 서브메쉬 또는 LSM이면 보존 검사를 하여 할당 여부를 결정한다.

• LSABF(LSA with Best-fit)

태스크를 수용할 수 있는 여러 개의 프리 서브메쉬들 중에서 하나를 선택하는 과정에서 가장 단편화를 적게 생성하게 되는 프리 서브메쉬로 결정하는 것을 제외하고는 LSAFF 기법과 같다.

• LSAFFnoCC(LSA with First Fit and without the Conservative Check)

보존 검사 단계만을 수행하지 않는 것을 제외하고는 LSAFF 기법과 같다.

• LSABFnoCC(LSA with Best Fit and without the Conservative Check)

보존 검사 단계만을 수행하지 않는 것을 제외하고는 LSABF 기법과 같다.

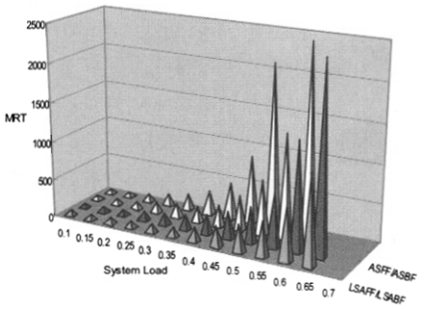
4.2 시뮬레이션 결과

(그림 5)는 32x32 메쉬 시스템에서 시스템 부하에 따른

MRT의 변화를 보이고 있다. 시스템 부하는 식 (1)에서 작업이 도착되는 간격을 변화시킴으로써 조절하였다. 각 작업의 요청된 체재시간은 50 단위시간을 평균으로 하는 지수 분포를 갖도록 하였으며, 요구되는 작업들의 크기는 균일 분포로 발생시켰는데, 각 변의 길이는 2에서 20으로 제한하였다. 네트워크 전파 시간이 그 작업의 수행시간의 10%를 차지하는 것으로 가정하고, 통신 프랙션을 0.1로 고정시켰다. 다른 기존의 연구 결과 [2, 6, 16] 에서와 같이, FF 기법과 BF 기법을 적용하였을 때의 MRT의 차이는 매우 작음을 (그림 5)에서도 알 수 있다. LSA 기법을 적용하였을 경우의 MRT가 FF, BF, ASFF, ASBF, FlexFF, 그리고 FlexBF의 경우보다 작음을 (그림 5)에서 볼 수 있다. Flexfold 기법과 비교하면, 10% 정도 높은 시스템 부하에서도 시스템이 안정되어 있음을 볼 수 있다. 시스템 부하가 증가할수록, LSA 기법의 장점은 두드러진다. 시스템 부하가 0.55일 때, Flexfold 기법과 비교하면 MRT가 35% 정도 감소하였다.

(그림 6)은 작업이 도착되는 간격을 평균 60을 갖는 지수분포로 발생시키고, 시스템 부하를 각 작업의 요청된 체재시간을 변화시킴으로써 조절하였을 때, MRT의 변화를 측정한 것이다. 나머지 조건들은 (그림 5)에서와 같이 유지하였다. 시스템 부하가 0.6일 때, FlexFF 기법과 비교하면 MRT가 37% 정도 감소하였으며, LSAFF 기법을 적용하였을 경우의 MRT가 FF, ASFF, 그리고 FlexFF 기법을 적용하였을 경우보다 작음을 볼 수 있다.

Flexfold와 LSA 기법에서 통신 프랙션이 미치는 영향을 (그림 7)에서 보였다. FF/BF와 AS 기법은 이 패러미터에 전혀 영향이 없지만, 비교하기 위해 그래프에 포함시켰다. (그림 6)에서와 같은 시뮬레이션 환경에서, 통신 프랙션이 취하는 값은 집합 {0.0, 0.2, 0.4, 0.8}에서 선택된다. LSA 기법이 FF/BF, AS, 그리고 Flexfold 기법들보다 여전히 좋은 결과를 보이고 있지만, 통신 프랙션의 값이 증가할 수 록 AS기법에 근접한 성능을 보인다. 통신 프랙션의 값이 큰 경우에는 보존 검사에서 수행시간을 증가하게 되는 형태 변형을 대부분 거부하게 되므로, 이와 같은 결과를 보인다고 판단된다. 멀티 프로세서 시스템에서 메시지 패싱의 소프트웨어 오버헤드는 이 값을 1-2%로 간주한다[6]. 본 시뮬레이션에서는 이 값을 10%로 놓고 수행하였으므로, 앞에서 보인 결과들을 볼



(그림 6) 시스템 부하의 변화에 따른 MRT의 개선도(평균 도착되는 시간 간격 =60, 체재시간을 변화시킴)

때, LSA 기법은 비교적 효율적이라고 볼 수 있다.

LSA 기법을 적용하였을 때의 시스템의 활용도는 각 작업의 요청된 체재시간은 60 단위시간을 평균으로 하는 지수분포를 갖도록 하고, 발생되는 각 작업의 크기에서 각 변의 길이를 2에서 27로 제한하여 균일분포로 독립적으로 발생시켰으며, 시스템 부하를 작업이 도착되는 간격을 변화시킴으로써 조절하였을 때, 1000 개의 개체를 발생시켜 측정하였다. 시스템 부하가 0.8 일 때, 62%까지 증가한다. 이것은 FF를 적용하였을 때의 시스템의 활용도가 20%, Flexfold 기법을 적용하였을 경우의 39% 인 것에 비하면 매우 좋은 결과이다. 이 높은 시스템의 활용도는 외부 단편화를 발생시키면서 대기 큐에서 기다리는 대신에 LSM을 할당하기 때문에, 그리고 크기가 큰 작업을 즉시 수용할 능력이 기존의 다른 기법들보다도 많기 때문에 가능한 것으로 판단된다.

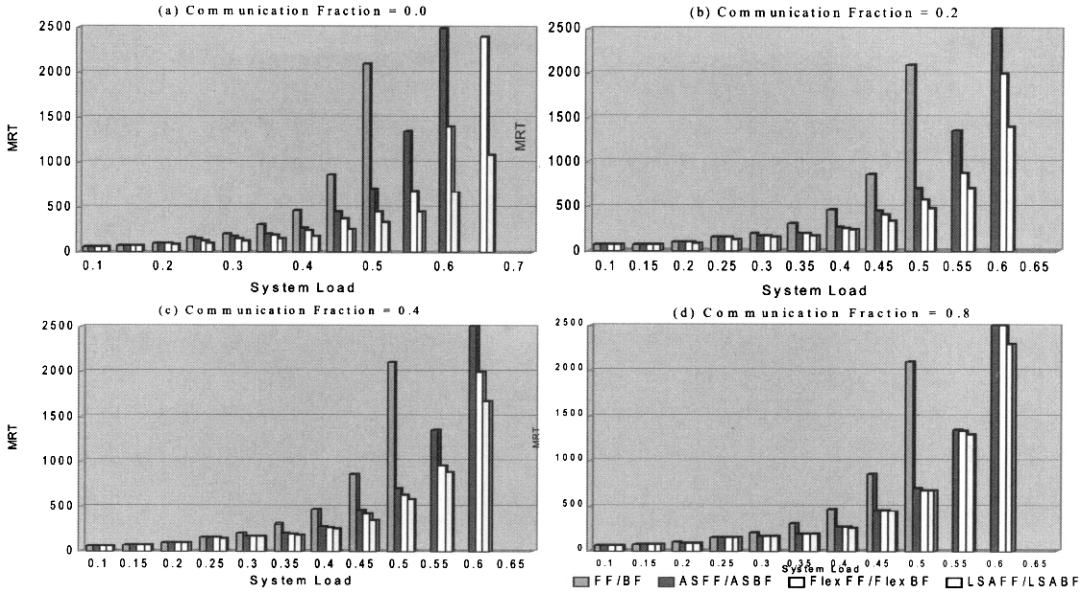
시스템 부하가 작을 때는 LSA 기법도 기존의 다른 기법들처럼 사각형 서브메쉬 모양으로 할당하는 경우가 많으므로 거의 비슷한 성능을 갖지만, 시스템 부하가 0.55 이상으로 높아지는 경우 기존의 기법에 비해 MRT와 완성시간에서 30% 이상의 성능 개선을 보인다. 시스템의 부하가 높을 때 대기 큐에 쌓이는 작업들이 많아지므로, 요청된 서브메쉬의 모양을 L-shaping하여 할당할 수 있는 LSA 기법이 대기 큐 헤드의 병목 현상을 풀어 줄 수 있기 때문이다. 또한 시스템에서의 체재시간이 긴 작업들이 많을수록, 요청된 서브메쉬의 크기가 큰 경우에도 MRT와 시스템의 활용도에서 LSA 기법의 성능이 기존의 다른 기법들보다 향상된다.

5. 결 론

오류 노드들을 가지는 2차원 메쉬 시스템을 효율적으로 이용하기 위해 동적이면서 신뢰도 높은 프로세서 할당기법을 제안하였다. LSA 기법은 오류가 있는 메쉬 시스템을 오류 프리 노드들을 최대한 포함하는 볼록 시스템으로 재구성한 후에, 기존의 사각형 서브메쉬뿐만 아니라 L-모양 서브메쉬를 할당하므로, 큰 크기의 작업을 포함해서 시스템으로 들어오는 작업들을 기존의 다른 기법들보다 빠르고 공정하게 수용할 수 있다. 재구성되는 최종 서브시스템이 사각형으로 제한되지 않으면서 볼록 모양이 보장되므로, 오류 프리인 정상 노드들을 최대한 포함할 수 있는 재구성 기법이며, 이 비사각형 메쉬 시스템에서 L-모양 서브메쉬의 할당은 시스템의 활용도를 증가시킨다. LSA 기법의 평균 작업 응답시간이 다른 기법들에 비해서 감소되는 것을 시뮬레이션 결과를 통해서 보였다. 그러므로 LSA 기법은 평균 작업 응답시간을 줄이고, 동시에 시스템의 활용도를 높일 수 있다.

참 고 문 헌

[1] A. Alan, B. Pritsker, J.J. O'reilly, and D.K. LaVal, Simulation with Visual SLAM and AweSim, John Wiley & Sons, Inc., 1997.



(그림 7) 통신 프랙션 값의 변동에 따른 영향(평균 도착되는 시간 간격 = 60, 체재시간을 변화시킴)

[2] D. Babbar and P. Krueger, "A performance Comparison of Processor Allocation and Job Scheduling Algorithms for Mesh-Connected Multiprocessors," *Proc. IEEE Symposium on Parallel and Distributed Processing*, pp.46-53, 1994.

[3] H. Chen and S. Hu, "Submesh Determination in Faulty Tori and Meshes," IEEE Trans. on Parallel and Distributed Systems, Vol.12, No.3, Mar., 2001.

[4] P. J. Chuang and N. F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer systems," *Proc. Int'l Conf. on Distributed Computing Systems*, pp.256-263, 1991.

[5] J. Ding and L. N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected systems," *Proc. Int'l Conf. on Parallel Processing*, Vol.II, pp.193-200, 1993.

[6] V. Gupta and A. Jayendran, "A Flexible Processor Allocation Strategy For Mesh Connected Parallel Systems," *Proc. Int'l Conf. on Parallel Processing*, Vol.III, pp.166-173, 1996.

[7] M. Kang, C. Yu, H. Y. Youn, B. Lee, and M. Kim, Isomorphic Strategy for Processor Allocation in k-Ary n-Cube Systems, IEEE trans. on Computers, Vol.52, No.5, pp.645-657, May, 2003.

[8] J. H. Kim and P. K. Rhee, "The Rule-Based Approach to Reconfiguration of 2-D Processor arrays," IEEE Trans. Computers, Vol.42, No.11, pp.1403-1408, Nov., 1993.

[9] K. Li and K. H. Cheng, "A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected System," *Proc. ACM Computer Science Conf.*, pp.22-28, 1990.

[10] T. Liu et al., "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems," *Proc. Int'l Conf. on Parallel Processing*, Vol.II, pp.159-163, 1995.

[11] K.H. Seo and S.C. Kim, "Improving system performance in contiguous processor allocation for mesh-connected parallel systems," ELSEVIER Journal of Systems and Software, Vol. 67, Issue 1, pp.45-54, July, 2003.

[12] D. D. Sharma and D. K. Pradhan, "A Fast and Efficient Strategy for Submesh Allocation in Mesh-Connected Parallel Computers," *Proc. IEEE Symposium on Parallel and Distributed Processing*, pp.682-689, Dec., 1993.

[13] D. D. Sharma and D. K. Pradhan, "Job Scheduling in Mesh Multicomputers," *Proc. Int'l Conf. on Parallel Processing*, Vol.II, pp.251-258, 1994.

[14] N. F. Tzeng and G. Lin, Maximum Reconfiguration of 2-D Mesh Systems with Faults, In *Proc. Int'l Conf. on Parallel Processing*, pp.I-77-I-84, 1996.

[15] B. S. Yoo and C. R. Das, "A Fast and Efficient Processor Allocation Scheme for Mesh-Connencted Multicomputers," *IEEE Trans. on Computers*, Vol.51, No.1, pp.46-60, Jan., 2002.

[16] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh-Connected Parallel Computers," *Journal of Parallel and Distributed Computing*, Vol.16, No.12, pp.328-337, Dec., 1992.

서 경 희

e-mail : khseo@cs.sungshin.ac.kr

1986년 서강대학교 수학과(이학사)

1989년 서강대학교 전자계산학과(공학사)

1992년 서강대학교 대학원 컴퓨터학과 (공학석사)

1998년 서강대학교 대학원 컴퓨터학과 (공학박사)

1999년~2003년 성신여자대학교 컴퓨터정보학부 계약교원

2003년~현재 성신여자대학교 컴퓨터정보학부 초빙교원

관심분야 : High Performance Computing and Software, Nume- rical Analysis, Optical Communication, Embedded System