메쉬 멀티컴퓨터에서 L-모양 서브메쉬 할당기법

(L-shaped Submesh Allocation Scheme for Mesh-Connected Multicomputers)

서 경 희 † 김 성 천 ††
(Kyung Hee Seo) (Sung Chun Kim)

요 약 프래그멘테이션은 다수의 사용자를 지원하는 대규모 멀티컴퓨터 시스템의 성능에 병목현상을 초래하는 주요 요인이다. 본 논문에서는 프래그멘테이션을 줄이기 위해서, 할당되는 프로세서들이 형성하는 모양이 직사각형이어야 한다는 제약조건을 완화시킨 LSSA (L-Shaped Submesh Allocation) 기법을 제안한다. LSSA 기법은 단편화된 메쉬 시스템에도 적용될 수 있도록 직사각형 뿐 만 아니라, 변형된 L자 모양의 서브메쉬를 할당할 수 있다. 그러므로 LSSA 기법은 시스템으로 들어오는 작업을 다른 기법들보다 빨리 수용할 수 있으며, 이로 인해 작업응답시간을 줄일 수 있다. 시뮬레이션 결과를 통해서 LSSA 기법이 외부 프래그멘테이션, 작업응답시간, 그리고 시스템의 활용도 면에서 다른 기법들보다 우수함을 보인다.

키워드 : 메쉬, 프래그멘테이션, 시스템 활용도

Abstract Fragmentation is the main performance bottleneck of large, multi-user multicomputer system. This paper presents an L-Shaped Submesh Allocation(LSSA) strategy, which lifts the restriction on the rectangular shape formed by allocated processors in order to address the problem of fragmentation. LSSA can manipulate the shape of the required submesh to fit into the fragmented mesh system. Thus, LSSA accommodates incoming jobs faster than other strategies and results in the reduction of job response time. Extensive simulations show that LSSA performs more efficiently than other strategies in terms of the external fragmentation, the job response time and the system utilization.

Keyword : mesh, fragmentation, system utilization

1. 서 론

복수개의 프로세서들을 고속의 상호연결 망으로 연결하여 구성하는 대규모 멀티컴퓨터 시스템은 여러 응용분야에서 점차 보편적인 컴퓨팅 플랫폼으로 자리잡고 있으며, 메쉬는 그 구조가 간단하고, 규칙성이 좋아서 확장이 쉽고, VLSI 구현에 적합하다는 장점 때문에, 상호연결 망의 위상으로 각광 받고 있다[1, 2, 3].

메쉬 멀티컴퓨터 OS의 중요 기능 중의 하나는 효율적인 시스템 자원관리를 통해 각 사용자들이 요구하는 병렬처리 작업을 동시에 수행시킬 수 있는 복수 사용자 환경을 지원하는 것이다. 이 기능은 OS의 기본적인 프로세서 관리 시스템인 프로세서 할당(processor allocation)과 작업 스케줄링에 의해 수행된다. 프로세서 할당 기법은 시스템으로 들어오는 각 작업들에게 독점적으로 이용할 수 있는, 요구하는 크기의 서브메쉬를 할당할 수 있도록 하며, 작업 스케줄링은 각 작업이 처리되는 순서를 결정한다. 프로세서 할당 기법의 주된 목적은, 작업들이 시스템에 도착되는 순서대로 서비스를 하는 공정성을 유지하면서 시스템의 성능을 높이는 것이다. 이는 작업 응답 시간과 시스템 프래그멘테이션(fragmentation)을 줄이면서, 동시에 시스템의 활용도를 증가시킴으로써 가능해진다. 높은 시스템의 활용도를 얻기 위해서는 발생되는 프래그멘테이션의 양을 최소화

· 본 연구는 한국과학재단 목적기초연구 R01-2001-000-00356-0 (2002)지원으로 수행되었음.
† 장 회 원 : 성신여자대학교 컴퓨터정보학부 교수
khseo@cs.sungshin.ac.kr
†† 종신회원 : 서강대학교 컴퓨터공학과 교수
ksc@arqlab1.sogang.ac.kr
논문접수 : 2002년 8월 23일
심사완료 : 2002년 11월 18일

시켜야 한다.

프로세서 할당 기법이 수행되면서 발생할 수 있는 프래그멘테이션의 종류는 내부, 외부, 그리고 가상 프래그멘테이션이다. 초기의 할당 기법인 2차원 Buddy 알고리즘은 각 변의 길이가 2의 멱승인 정사각형 서브메쉬 만을 할당할 수 있으므로, 심각한 내부 프래그멘테이션을 발생시킨다[4]. 그러나 메쉬 시스템에 적용되는 대부분의 프로세서 할당 알고리즘들은 내부 프래그멘테이션을 발생시키지 않는다[3, 5, 6, 7, 8, 9]. 최근에 제안된 프로세서 할당 알고리즘들은 이용 가능한 서브메쉬들을 완전히 인식할 수 있는 능력을 갖고 있다[3, 7, 8]. 따라서 내부 및 가상 프래그멘테이션은 더 이상 문제가 되지 않는다. 여전히 문제가 되는 것은 외부 프래그멘테이션으로서, 시스템에는 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 프래그멘테이션을 줄이기 위한 최초의 시도는 AS(Adaptive Scan) 기법에서 찾을 수 있다[6]. 시스템으로 들어오는 작업이 a개의 열과 b개의 행을 가지는 서브메쉬 $a\times b$를 요구할 때, 그 서브메쉬 $a\times b$를 할당할 수 없으면, AS 기법은 요구하는 서브메쉬를 90° 회전한, b개의 열과 a개의 행을 가지는 $b\times a$ 서브메쉬를 할당하려고 시도한다[6]. 외부 프래그멘테이션을 줄이기 위한 또 하나의 시도는 1996년에 제안된 Flexfold 할당 기법이다[10]. Flexfold 할당 기법은 요청된 크기의 서브메쉬 $a\times b$ 와 $b\times a$가 모두 할당이 불가능하고 a 와 b가 모두 짝수이면, $a/2\times 2b$, $2a\times b/2$, $2b\times a/2$, 또는 $b/2\times 2a$ 서브메쉬를 할당하기 위해서 이들을 차례로 탐색한다.

Flexfold 할당 기법을 포함해서 이전에 발표되었던 모든 프로세서 할당 알고리즘들은 전체 시스템 구조와 같은 위상을 가지는 직사각형 서브메쉬를 발견하지 못하면, 작업을 할당할 수 없었다. 즉, 요청된 크기를 충분히 수용할 수 있는 개수의 프리 프로세서들이 시스템 내에 존재하더라도, 그 프리 프로세서들이 요청된 크기를 갖는 하나의 직사각형 서브메쉬를 형성하지 못하면, 그 작업은 수행되지 못하고 대기 큐로 보내진다. 하나의 작업이 시스템을 떠났을 때, 즉, 할당해제(deallocation)이 발생하면, 대기 큐의 헤드에 있던 그 작업에게 할당할 수 있는 서브메쉬를 찾기 위한 시도가 이루어진다. 이때, 시스템으로 들어오는 작업들은 대기 큐에 도착되는 순서대로 쌓이게 된다. 즉, 대기 큐의 헤드에 있는 작업이 가능한 빨리 처리되지 않으면, 병목현상이 발생한다. 그 결과, 외부 프래그멘테이션 문제는 여전히 심각한 상태이며, 이로 인해 34% 에서 46%의 시스템의 활용도를 보이고 있다[11].

이러한 문제를 해결하기 위해서, 선입선출 (FCFS : First Come First Served) 방식으로 작업을 처리하면서 효율적인 프로세서 할당 기법을 연구하는 대신에, 다른 작업 스케줄링을 채택하는 것이 더 바람직하다는 연구결과가 있다[12]. 그러나 선입선출 방식이 아닌 다른 작업 스케줄링을 채택하는 것은 사용자에게 공정하지 않다는 단점을 가지며, 또한 크기가 큰 작업을 특히 더 차별하게 되는 경향으로 인해 starvation problem을 유발시킬 수 있다.

본 논문에서는 사용자에게 공정성을 유지하면서, 외부 프래그멘테이션의 발생 양을 줄임으로써 시스템의 활용도를 높이고, 동시에 작업 응답 시간을 줄일 수 있는 LSSA (L-Shaped Submesh Allocation) 기법을 제안한다. LSSA 기법은 CJ(Cut-and-Join) 기법[13]을 확장시킨 것으로서, 할당할 수 있는 L자 모양의 서브메쉬(LSM: L-shaped submesh)들을 확장하였으며, 할당·해제 알고리즘의 시간 복잡도를 줄임으로써, 효율적인 프로세서 할당 기법을 제공한다. 기존의 기법들을 사용해서 사용자가 요구하는 직사각형 모양의 서브메쉬를 할당할 수 없을 때, 이 작업이 현재의 단편화된 메쉬 멀티컴퓨터 시스템에서 수행될 수 있도록 하기 위해서 LSSA 기법은 신속하게 LSM을 할당할 수 있는 적응성을 가진다.

요청된 작업이 LSM 등의 변형된 메쉬 모양에서 수행될 경우 그 작업의 수행 시간이 증가될 수 도 있다. 그러나 직사각형 모양의 메쉬를 할당 받기 위해서 대기 큐에서 기다리는 대신에, 바로 LSM을 할당 받아서 작업을 수행시킴으로써, 결과적으로 작업 응답 시간을 줄일 수 있다면, 또한 이로 인해서 외부 프래그멘테이션을 줄이고 시스템의 활용도를 높일 수 있다면, LSM 등의 서브메쉬 모양의 변형은 충분한 가치가 있다. 또한, 대기 큐의 헤드에서 기다리게 하지 않고, 변형된 모양의 서브메쉬를 할당하여 그 작업을 수행시킴으로써, 그 다음에 도착된 작업들에게 가능한 빨리 할당받을 수 있는 기회를 제공할 수 있다. 즉, 다음 작업은 크기가 작아서 모양의 변형 없이도 수용할 수 있는 프리 서브메쉬가 많이 있는데도 불구하고, 대기 큐의 헤드에 있는 작업 때문에 계속 기다리는 상황을 방지하고자 한다.

LSSA 기법의 성능을 기존의 다른 기법들과 비교하기 위해서 이산 사건 중심의 시뮬레이터를 AweSim V. 1.4를 사용하여 구성하였으며, LSSA기법을 적용하여

작업 응답 시간과 작업 완료 시간들을 줄이고, 시스템의 활용도가 상당히 증가함을 시뮬레이션 결과를 통해 보인다. 본 논문의 구성은 다음과 같다. 2장에서는 기존의 서브메쉬 할당 기법들을 간략하게 소개하고, 3 장에서는 기본적인 용어 및 확장된 LSM을 정의한다. 우리가 제안하는 LSSA 기법을 4장에서 소개하고, 5장에서는 LSSA 기법의 성능을 시뮬레이션을 통해 분석한다. 6장에서 결론을 맺는다.

2. 기존의 할당 기법들

2차원 Buddy (2DB) 기법은 한 변의 길이가 2의 멱승인 정사각형 모양의 서브메쉬 만을 할당할 수 있다[4]. $a\times b$ 서브메쉬를 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 프래그멘테이션을 발생시킨다. 2DB 기법의 단점을 보완하기 위해서, Frame Sliding (FS) 기법이 제안되었다[5]. 이 기법은 직사각형 모양의 메쉬 시스템에도 적용이 가능하며, 작업이 요구하는 크기와 같은 크기의 서브메쉬를 할당함으로써 내부 프래그멘테이션을 발생시키지 않는다. 그러나 고정된 stride를 사용하여 프레임을 찾기 때문에 가상 프래그멘테이션을 발생시키며, 또한 외부 프래그멘테이션도 많이 발생시킨다.

간단한 최초적합 (FF : First Fit)과 최적적합 (BF: Best Fit) 기법들은 직사각형 모양의 메쉬 시스템에 적용이 가능하며, 내부 프래그멘테이션도 발생시키지 않는다[9]. 할당 가능한 서브메쉬를 좀더 빨리 찾기 위해서, 각 프로세서를 각 비트에 대응시키는 두 개의 2차원 배열, Busy Array와 Coverage Array를 사용하여 검색한다. 그러나 요청된 서브메쉬를 할당할 수 없을 때, 그 서브메쉬를 90° 회전시킨 모양의 서브메쉬에 대해서는 전혀 고려하지 않으므로, 가상 프래그멘테이션을 발생시키며, 외부 프래그멘테이션도 심각하다.

FS 기법에서 발생되는 가상 프래그멘테이션을 제거하기 위해서 제안된 AS 기법은[6] 고정된 stride를 사용하지 않고, 적응적으로 스캔함으로써 할당 가능한 서브메쉬를 완전하게 인식할 수 있다. 또한 요청된 서브메쉬 $a\times b$를 할당할 수 없을 때, 이 서브메쉬를 90° 회전한 서브메쉬 $b\times a$를 할당하려는 시도를 처음으로 하였다. 그러나 이 할당 알고리즘의 최초적합 특성 등으로 인해, 여전히 외부 프래그멘테이션 문제가 심각하게 남아있다.

요청된 서브메쉬의 모양을 90° 회전시키는 것 외에, 폴디드 서브메쉬(folded submesh)를 고려하려는 시도가 Flexfold 할당 기법이다[10]. Flexfold 기법은 요청된 서브메쉬 $a\times b$ 와 $b\times a$가 모두 할당이 불가능하고 a, b가 모두 짝수이면, 요청된 작업을 대기 큐에 넣는 대신에 $a/2$ $2b$ 와 $2a$ $b/2$를 탐색한다. 이 기법은 요청된 서브메쉬의 모양을 폴디드 서브메쉬로 변환하여, 다른 기법들보다 더 많은 할당 기회를 제공하기 때문에 시스템의 활용도를 증가 시킬 수 있다. 그러나 Flexfold 할당 기법에서 사용되는 폴딩(folding)은 요청된 서브메쉬의 두 변 a 와 b가 모두 짝수인 때만 적용될 수 있다[10]. 또한 할당할 수 있는 서브메쉬는 직사각형 모양이어야 하므로, 여전히 외부 프래그멘테이션의 발생량이 높게 되어, 시스템의 활용도는 높지 않다.

CJ 프로세서 할당 기법은 사용자가 요구하는 직사각형 모양의 서브메쉬를 기존의 기법으로 할당할 수 없을 때, 요청된 서브메쉬를 LSM으로 변환하여 프로세서 할당을 수행하므로, 내부, 가상 프래그멘테이션을 발생시키지 않으며, 외부 프래그멘테이션 발생량도 줄인다[13]. 그러나 90°, 180°, 270° 회전된 모양의 LSM을 할당하려는 시도를 하지 않았으며, Free list 기반의 할당 알고리즘을 사용하였기 때문에, 회전된 모양의 LSM을 할당할 경우 알고리즘의 시간복잡도가 증가할 수 있다.

3. L자 모양의 서브메쉬, LSM

2차원 메쉬 $M(w,h)$ 는 wh 개의 노드들로 구성되며, 너비 w 와 높이 h를 가지는 직사각형의 격자 모양을 가진다. 메쉬 시스템에서의 각 프로세서는 각 노드에 대응된다. 열 i 와 행 j에 위치한 노드의 주소는 $<i, j>$로 표기하며, 행과 열들은 메쉬의 상단 좌측 모서리부터 세기 시작한다. 그림 1은 메쉬 $M(5,4)$ 와 각 노드들의 주소를 보이고 있다.

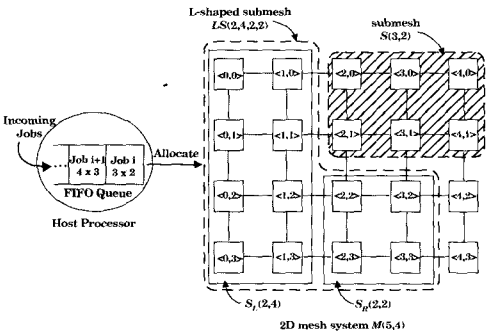

그림 1 2D 메쉬 $M(5,4)$에서의 직사각형 또는 L자 모양의 서브메쉬 할당

메쉬 $M(w,h)$에 대해서 너비 a와 높이 b를 가지는 서브메쉬 $S(a,b)$는 다음과 같은 조건, $a \leq w$ 그리고 $b \leq h$ 을 만족하면서 메쉬 $M(w,h)$에 포함되는 2차원 메쉬이다. 서브메쉬 $S(a,b)$의 주소는 $<x_1,y_1,x_2,y_2>$로 나타내며, 여기서 $<x_1,y_1>$은 그 서브메쉬의 하단-좌측 모서리를 가리키며, $<x_2,y_2>$는 상단-우측 모서리를 가리킨다. 그림 1에서 주소가 <2,1,4,0>인 서브메쉬 S(3,2)를 보이고 있다.

[정의 1] 임의의 두 서브메쉬 $S(c,d)$ 와 $S(e,f)$에 대하여, 이들의 주소가 각각$<x_{11},y_{11},x_{12},y_{12}>$ 이고, $<x_{21}, y_{21},x_{22},y_{22}>$ 일 때, 다음 조건들 중의 하나를 만족하면, 이 두 서브메쉬들은 서로 이웃한다(adjacent) 라고 한다.

1) $x_{21} = x_{12} + 1$ 이고 $y_{11} = y_{21}$,
2) $x_{21} = x_{12} + 1$ 이고 $y_{12} = y_{22}$,
3) $y_{22} = y_{11} + 1$ 이고 $x_{11} = x_{21}$,
4) $y_{22} = y_{11} + 1$ 이고 $x_{12} = x_{22}$.

[표기 1] 두 개의 서로 이웃하는 서브메쉬에 대하여, 왼쪽에 있는 서브메쉬 $S(c,d)$를 $S_L(c,d)$로 표기하며, 오른쪽에 있는 서브메쉬 $S(e,f)$를 $S_R(e,f)$로 나타낸다.

그림 1에서 주소 <0,3,1,0>인 서브메쉬 $S_L(2,4)$와 주소 <2,3,3,2>인 서브메쉬 $S_R(2,2)$은 서로 이웃하고 있다.

[정의 2] L자 모양의 서브메쉬, LSM, $LS(c,d,e,f)$은 긴 너비(longer width)가 $c+e$ 이고, 긴 높이(longer height) d를 가지는 영문자 L자 모양의 블록으로서, 두 개의 서로 이웃하는 서브메쉬 $S_L(c,d)$ 와 $S_R(e,f)$의 프로세서들의 집합으로 구성된다.

[표기 2] L자 모양의 서브메쉬 $LS(c,d,e,f)$는 [$<x_{11},y_{11},x_{12},y_{12}>$ & $<x_{21},y_{21},x_{22},y_{22}>$]로 표기하며, 여기서 $<x_{11},y_{11}>$ 와 $<x_{12},y_{12}>$는 서브메쉬 $S_L(c,d)$의 하단-좌측 모서리와 상단-우측 모서리를, 그리고 $<x_{21},y_{21}>$와 $<x_{22},y_{22}>$는 서브메쉬 $S_R(e,f)$의 하단-좌측 모서리와 상단-우측 모서리를 각각 나타낸다.

그림 1에서 서로 이웃하는 두 서브메쉬들 $S_L(2,4)$와 $S_R(2,2)$는 긴 너비 4와 긴 높이 4를 가지는 LSM $LS(2,4,2,2)$를 구성하며, 이 LSM의 좌표는 [<0,3,1,0> & <2,3,3,2>]로 표시된다.

[정의 3] L자 모양의 서브메쉬 LSM을 구성하는 모든 노드들이 이미 할당받은 어떤 작업들에게도 배정되지 않았을 때, 이 LSM을 free LSM 이라 한다.

[정의 4] L자 모양의 서브메쉬 LSM을 구성하는 모든 노드들이 하나의 작업에 할당되었을 때, 이 LSM을 busy LSM 이라 한다.

LSM $LS(c,d,e,f)$는 $cd+ef$ 개의 노드들로 구성된다. LSM $LS(c,d,e,f)$는 서브메쉬 $S(a,b)$로부터 만들어질 수 있으며, 이때 $LS(c,d,e,f)$를 구성하는 노드들의 총수와 $S(a,b)$의 노드들의 총수는 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 이분하면서 만들어지는데, 이 잘려지는 변 즉, 커팅 사이드(cutting side)의 짝홀에 따라서 만들어지는 과정은 다음과 같다: 커팅 사이드 a가 짝수일 때, 서브메쉬 $S(a,b)$의 우측-상부의 일부분인 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 2 (a)에서 보여주고 있다. 커팅 사이드 a가 홀수일 때, 서브메쉬 $S(a,b)$의 우측-상부의 일부분인 $S(e, b-f)$를 잘라서 (여기서, $b-f = c$) 그 잘려진 부분을 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 2 (b)에서 보여주고 있다. 이와 같은 L-shaping 과정은 서브메쉬 $S(a,b)$의 다른 변 b에도 똑같이 적용될 수 있다.

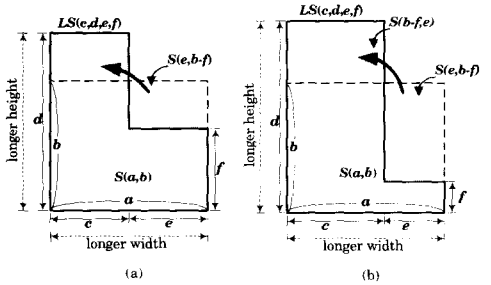

그림 2 서브메쉬 $S(a,b)$로부터 $LS(c,d,e,f)$의 구성 (a) a가 짝수인 경우 (b) a가 홀수인 경우

초기의 2DB 기법이 할당하는 서브메쉬의 모양은 각 변이 2^n인 정사각형이었으며[4], 그 이후에 제안된 모든 서브메쉬 할당 기법들은 직사각형 모양의 서브메쉬를 고려하였다[5, 6, 7, 8, 9, 10]. 이제 직사각형의 서브메쉬를, 그림 3에서와 같이, 사분면으로 나누어 보았을 때, 각 사분면이 존재하지 않는 더 유연성 있는 메쉬, 확장된 LSM을 고려하고자 한다. 직사각형의 서브메쉬에서 어느 사분면이 존재하지 않는가에 따라서 네 가지 유형의 LSM들이 존재한다. 제 1 사분면이 없는 서브메쉬는 Type 1 LSM, 제 2 사분면이 없는 서브메쉬는 Type 2 LSM, 제 3 사분면이 없는 서브메쉬는 Type 3 LSM, 그리고, 제 4 사분면이 없는 서브메쉬는 Type 4 LSM 이라 하고, 그림 3에서 보이듯이, 각각 LSM_1, LSM_2, LSM_3, 그리고 LSM_4로 표기한다. LSM_2, LSM_3, 그리고

LSM_4는 LSM_1을 시계 반대 방향으로 90°, 180°, 그리고 270° 회전시킨 형태이다. 직사각형 서브메쉬는 긴 높이 d와 짧은 높이 f가 같은 LSM의 특별한 경우로 간주될 수 있다.

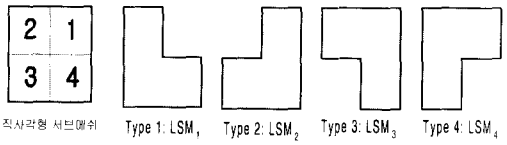

그림 3 직사각형 서브메쉬로부터 구성되는 L자 모양의 서브메쉬들

4. LSSA 기법

LSSA 기법은 다음 예에서와 같이 작업 응답 시간과 외부 프래그멘테이션을 동시에 감소시키는데 초점을 맞추고 있다. 그림 4에서 크기가 9×7 인 메쉬 시스템을 3개의 태스크들이 공유하고 있으며, 각 태스크들은 점선 안에 회색으로 칠해져 있는 영역의 프로세서들을 다른 태스크들과는 독립적으로 그 작업을 완전히 끝낼 때까지 사용할 수 있다. 각 태스크들이 요구한 서브메쉬의 크기는 2×5, 4×2, 그리고 2×2 이고, 각 태스크에게 할당된 서브메쉬의 좌표는 각각 <4,4,5,0>, <0,1,3,0>, 그리고 <2,4,3,3>이다.

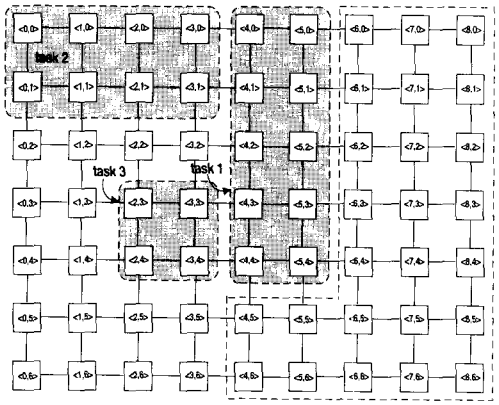

그림 4 LSSA 기법을 설명하는 예

시스템으로 들어오는 새로운 태스크 4가 서브메쉬 $S(5,5)$를 요구했을 때, 기존의 기법들은 이 태스크를 즉시 수용할 수 없다. 현재 메쉬 시스템에는 41 개의 이용 가능한 프로세서들이 존재함에도 불구하고, 기존의 기법들은 태스크 4를 대기 큐에 넣을 수밖에 없다.

그러나 LSSA 기법은 서브메쉬 5×5가 그림 5 (a)에서와 같이, LSM $LS(3,7,2,2)$로 변형될 수 있으며, 이 LSM을 90° 회전한 Type 2 LSM_2가 현재의 메쉬 시스템에서 이용 가능하며, 그 좌표가 [<4,6,5,5> & <6,6,8,0>] 임을 인식할 수 있다. 이 상황에서, LSSA 기법은 다른 기법들 보다 빨리 태스크를 수용할 수 있다.

그림 4에서 새로운 태스크 5가 $S(4,3)$을 요청하였을 때, 이를 즉시 수용하기 위해서는 $LS(2,4,2,2)$로 변형하여 할당할 수 있으며, $S(4,3)$이 $LS(2,4,2,2)$, $LS(2,5,2,1)$, 그리고 폴더드 서브메쉬 $S(2,6)$ 즉, $LS(1,6,1,6)$으로 변형될 수 있음을 그림 5 (b)와 (b')에서 보여주고 있다.

그림 5 (c)에서 볼 수 있듯이, 서브메쉬 $S(1,7)$을 L-shaping한 경우, 변환된 LSM에서의 노드들 간의 통신거리는 변환 전의 서브메쉬에서의 통신거리보다 증가되지 않는다. 실제로 노드들 간의 통신 거리는 더 가까워진다. 서브메쉬 $S(1,7)$에서 노드 1과 노드 7의 거리는 6 홉이지만, $LS(1,4,1,3)$에서는 2 홉으로 감소하고, $LS(1,5,1,2)$에서는 4 홉으로 감소한다.

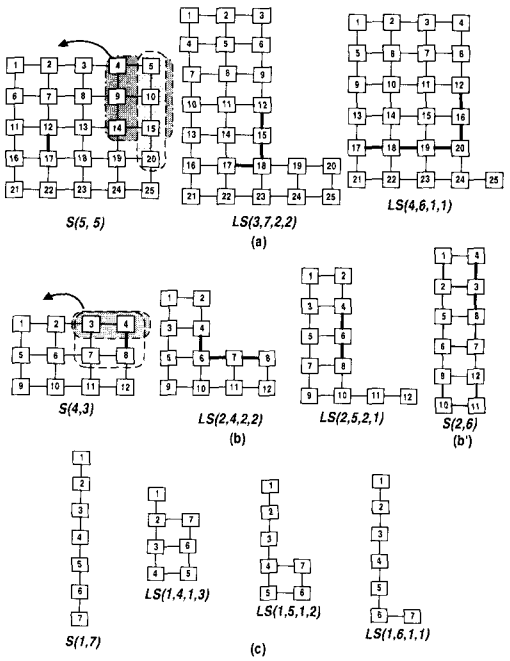

그림 5 (a) 커팅 사이드가 홀수인 $S(5,5)$로부터 만들어지는 LSM들 (b) 커팅 사이드가 짝수인 $S(4,3)$로부터 만들어지는 LSM들 (c) $S(1,7)$로부터 만들어지는 LSM들

그러나 L-sha ping이 통신거리를 항상 감소시키지는 않는다. 그림 5 (b)에서 노드 4 와 노드 8은 서브메쉬 $S(4,3)$에서 서로 이웃하고 있지만, LSM $LS(2,4,2,2)$에서는 3 홉이 떨어져있다. 또한, 노드 6과 노드 7 사이의 간선은 노드들의 두개의 쌍 {4,8} 과 {3,7} 사이에서 통신 경로로 공유되고 있다. 이러한 것이 태스크들의 실행시간을 증가 시킬 수 있다. 그러나, 각 태스크들의 할당을 기다리는 시간의 감소량이 변형된 모양의 서브메쉬를 할당 받음으로써 증가하게 되는 태스크의 실행시간보다 크다면, 전체 태스크들의 작업 응답 시간은 감소하게 될 것이다.

4.1 서브메쉬 형태의 변형이 작업 수행시간에 미치는 영향의 분석

폴딩 또는 L-shaping으로 인한 서브메쉬 형태의 변형들이 태스크의 실행시간에 미치는 영향을 분석하기 위해서 그러한 형태의 변형들을 하나의 서브메쉬에서 다른 서브메쉬로의 그래프 임베딩으로 간주할 수 있다 [14]. 그래프 $G(V,E)$를 다른 그래프 $G'(V',E')$로의 임베딩 ε에 대해서, V에 속하는 각 정점 v는 V'에 속하는 정점 $\varepsilon(v)$로 일대일 대응된다. E에 속하는 간선 $e = (u,v)$는 $\varepsilon(u)$ 와 $\varepsilon(v)$를 잇는 G'상의 경로로 대응된다. 그래프 G에서 이웃하는 두 정점 u 와 v에 대해서, G'로 대응되는 두 정점 $\varepsilon(u)$ 와 $\varepsilon(v)$ 사이의 거리를 그 간선 (u,v)의 연장비율(dilation)이라 한다. 임베딩 ε의 연장비율은 G의 모든 간선에 대한 연장비율 중 최대 값으로 결정된다. 혼잡비율(congestion)은 G'에서의 하나의 간선에 대해서, 이 간선을 포함하는 G' 상의 경로에 대응되는 G에서의 간선들의 최대 개수를 의미한다. 이 용어들을 사용하면, 폴디드 서브메쉬로의 임베딩은 그림 5(b)와 (b')에서 보듯이, 연장비율이 2이고, 혼잡비율이 2이다.

이제, LSSA 기법에서 수행되는 사각형 서브메쉬의 LSM으로의 형태 변형들은, 커팅 사이드 a에 대해서, 연장비율이 a를 초과하지 않으며, 혼잡비율은 2를 초과하지 않음을 보이려고 한다. 서브메쉬 $S(a,b)$에서 $S(b,a)$로의 임베딩은 연장비율과 혼잡비율이 각각 1임은 분명하다. 짝수 b 에 대해서, 서브메쉬 $S(a,b)$를 폴디드 서브메쉬 $S(2a, b/2)$로의 폴딩을 고려하자. 서브메쉬 $S(a,b)$의 상단-좌측의 모서리의 좌표를 (0,0)로 하고 각 노드에 주소를 부여하면, 하단-우측 모서리의 주소는 $(a-1, b-1)$이 된다. 이때, $S(a,b)$에 있는 노드 (i, j)를 $S(2a, b/2)$에서의 노드 (i', j')로 다음과 같이 일대일 대응시키면:

$$i' = \begin{cases} 2i+1 & \text{if } \lfloor j/2 \rfloor \text{ is odd} \\ 2i & \text{otherwise} \end{cases}$$

$$j' = \lfloor j/2 \rfloor,$$

연장비율과 혼잡비율이 각각 2임을 쉽게 알 수 있다. 이 임베딩은 그림 5(b)와 (b')에서 $a=4$, $b=3$인 경우에 대해서 보이고 있다. $S(4,3)$에서 수평으로 이웃하고 있던 노드들은 $S(2,6)$에서도, 굵은 선으로 나타냈듯이, 여전히 이웃하고 있다. 오직 수직으로 이웃하고 있던 노드들 만 연장비율이 2로 증가하였다. 그러므로, LSSA 기법에서 폴디드 서브메쉬로의 변형은 연장비율과 혼잡비율이 2를 넘지 않는다.

LSM으로의 임베딩은 커팅 사이드 a가 짝수일 경우에는 연장비율이 $a/2+1$ 이고, 혼잡비율이 2이며, 커팅 사이드 a가 홀수일 경우에는 연장비율이 a이고, 혼잡비율이 2임을 다음 정리 1에서 보인다.

[정리 1] 서브메쉬 $S(a,b)$는 홀수인 커팅 사이드 a에 대해서 연장비율과 혼잡비율이 각각 a 와 2가 되도록 LSM $LS(c,d,e,f)$로 임베딩할 수 있으며, 짝수인 커팅 사이드 a에 대해서 연장비율과 혼잡비율은 각각 $a/2+1$과 2가 되도록 LSM $LS(c,d,e,f)$로 임베딩할 수 있다. 여기서, $cd+ef=ab$, $d \geq f$ 이고 $c+e=a$ 이다.

[증명] 서브메쉬 $S(a,b)$를 LSM $LS(c,d,e,f)$로의 변형을 고려하자. 여기서, $cd+ef=ab$, $d \geq f$ 이고 $c+e=a$ 이다. 서브메쉬 $S(a,b)$의 상단-좌측 모서리의 좌표를 (0,0)로 하고 각 노드에 주소를 부여하면, 하단-우측 모서리의 주소는 $(a-1, b-1)$이 된다. LSM $LS(c,d,e,f)$에서 $S_L(c,d)$의 상단-좌측의 모서리의 좌표를 (0,0) 으로 하고 각 노드에 주소를 부여하면, $S_L(c, d)$의 하단-우측 모서리의 주소는 $(c-1, d-1)$이 되며, $S_R(e, f)$의 상단-좌측의 모서리의 좌표는 $(c, d-f)$가 되고, $S_R(e, f)$의 하단-우측 모서리의 주소는 $(c+e-1, d-1)$이 된다. $S(a,b)$에 있는 노드(i, j)를 $LS(c,d,e,f)$에서의 노드 (i', j')로 다음과 같이 일대일 대응시키고, 여기서 l은 노드의 번호로서, 그 노드의 좌표 (i, j)로부터 구해진다:

$l < c(d-f)$ 이면 $\begin{cases} i' = l \bmod c \\ j' = \lfloor l/c \rfloor, \end{cases}$

$l \geq c(d-f)$ 이면 $\begin{cases} i' = l \bmod a, \ l = i + ja. \\ j' = j + d - b \end{cases}$

위의 LSM으로의 주소변환 함수를 사용하면, 좌표 $<0, b-1, a-1, b-f+1>$로 표현되는 $S(a, f-1)$에 포함되는 노드들 간의 상호연결 관계는 $LS(c,d,e,f)$에서도 전혀 변동 없이 그대로 유지된다. 나머지 부분에서 이웃하는 노드들의 연장비율은 커팅 사이드 a에 의해 결정된다. a가 짝수인 경우에는 커팅 사이드 a를 반으로 잘

라서 $S_1(c,b)$에의 상단에 붙임으로, 잘려지는 서브메쉬의 우측-하단 모서리의 노드 번호를 가지는 노드와 수직 아래방향으로 이웃하던 노드와의 연장비율이 가상크며, $a/2+1$로 증가한다. a가 홀수인 경우에는 잘려지는 서브메쉬의 좌변 또는 남겨지는 서브메쉬의 우변에서 수직으로 이웃하던 노드들의 연장비율이 최대 a 까지 증가하는 경우가 있으므로, 연장비율은 a가 된다. yx-라우팅을 사용하여 노드간의 통신을 수행할 경우에, 커팅 사이드의 짝·홀에 관계없이 혼잡비율은 2가 된다. 그러므로 LSM으로의 변형에서 커팅 사이드 a가 홀수일 때 연장비율과 혼잡비율은 각각 a와 2보다 작거나 같으며, 커팅 사이드 a가 짝수일 때 연장비율과 혼잡비율은 각각 $a/2+1$ 과 2보다 작거나 같다.■

그림 5 (a)에서 커팅 사이드 a가 5인 경우에 연장비율이 3 또는 5로 증가하며, 혼잡비율이 2가 됨을 보여주고 있다. 그림 5(b)에서 커팅 사이드 a가 4인 경우는 연장비율이 3으로, 혼잡비율이 2로 증가하는 것을 보이고 있다.

LSSA 기법에서 사용되는 모든 서브메쉬 형태의 변형들은 각 태스크의 계산 시간에는 영향을 주지 않으며, 오직, 각 노드들 간의 통신 시간에만 영향을 줄 수 있다. 서브메쉬 $S(a,b)$로부터 LSM이 만들어졌을 때, 다음과 같은 최악의 상황 즉, $a \geq b$ 이고 a는 홀수일 때, 그 a를 커팅 사이드로 하여 구성된 LSM이 할당된 경우를 고려하자. 서브메쉬 $S(a,b)$에서 임의의 이웃하는 두 노드들의 거리는 LSM에서 많아야 a 만큼 증가할 수도 있다. 그리고 혼잡 비율이 2이므로, 메시지들이 각 통신 링크를 횡단하는 시간은 많아야 두 배만큼 소요될 것이다. 그러므로 전체적인 통신 시간은 최악의 경우에 $2a$ 만큼 증가할 수 도 있다.

그러나 통신 시간의 실제적인 증가분은 훨씬 더 작을 것으로 예상된다. 통신 시간은 네트워크 전파 시간과 엔드 노드 프로세싱 시간으로 구성된다. 네트워크 전파 시간은 라우터에서 발생되는 지연시간을 포함하여 메시지가 전달되기 위해 네트워크 안에서 소요되는 시간이다. 엔드-노드 프로세싱 시간은 메시지가 근원지 노드와 목적지 노드에서 소비하는 시간이다. 이 시간은 사용자와 시스템 버퍼 간에 복사하는 소프트웨어 부하, 버퍼 관리, 근원지 노드에서 라우팅 헤드를 붙이고, 목적지 노드에서 제거하는 작업 등에 소요되는 시간을 포함한다. 일반적으로, 엔드-노드 프로세싱 시간은 네트워크 전파 시간 보다 훨씬 더 크다. 예를 들어, 인텔의 델타 멀티컴퓨터의 메시지 전송시간 67μs 중에서 네트워크에서 소비되는 시간은 1μs 보다도 작으며[10], 또한 contention-free message-passing 네트워크 시스템에서 전체 통신 시간 중에 네트워크 전파 시간이 차지하는 비중이 1.5% 이하라는 보고도 있다[15]. 아주 다행스럽게도, 폴딩이나 LSM으로의 변형은 엔드 노드 프로세싱 시간에는 전혀 영향을 주지 않는다. 폴딩과 L shaping은 전체 통신 시간 중에 매우 적은 비중을 차지하는 네트워크 전파 시간만을 증가시킬 수 있으므로, 태스크 실행 시간의 실제적인 증가분은 매우 작다.

4.2 LSSAFF 와 LSSABF 기법

LSSAFF(LSSA with FF)와 LSSABF(LSSA with BF) 기법은 LSSA 기법에서 할당 가능한 서브메쉬를 탐색하는 과정에서 탐색하게 될 LSM들의 가짓수를 제한하고, 기존의 내부 프래그멘테이션을 발생시키지 않는 할당 기법들 중에서 가장 간단한 FF와 BF 기법[9]을 응용한 효율적인 할당 알고리즘이다. 또한, FF 와 BF 기법이 가지는 단점인 가상 프래그멘테이션을 LSSAFF와 LSSABF 기법에서는 효과적으로 제거할 수 있으며, 이들 기법의 장점인 낮은 할당·해제 시간 복잡도를 그대로 유지한다.

시스템으로 들어오는 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 결정한다. a 와 b가 같은 정사각형 서브메쉬이면, 폴딩과 L-shaping 모두 한 변 a에 대해서만 적용된다. 그 다음에, 다음과 같은 순서대로 프리 서브메쉬의 탐색이 시작된다. 즉, 원래의 서브메쉬 $S(a,b)$, 회전된 서브메쉬 $S(b,a)$, 짝수 a에 대한 폴디드 서브메쉬 $S(a/2, 2b)$, 회전된 서브메쉬 $S(2b, a/2)$, 짝수 b에 대한 폴디드 서브메쉬 $S(2a, b/2)$, 회전된 서브메쉬 $S(b/2, 2a)$, $S(a,b)$로부터 만들어지는 LSM, 그리고 회전된 LSM이다.

짝수 a에 대해 $S(a,b)$, $a \geq b \geq 2$로부터 만들어지는 LSM은 $LS(a/2, b+k, a/2, b-k)$, $1 \leq k \leq b$ 이며, 홀수 a에 대해 $S(a,b)$, $a \geq b \geq 2$로부터 만들어지는 LSM은 $LS(\ |a/2|+k,\ b+|a/2|-k,\ |a/2|-k,\ b-|a/2|-k)$ $(0 \leq k \leq b-1-|a/2|)$ 이다. 이용 가능한 LSM을 만날 때까지 k값을 바꾸며, $b-1-|a/2|=-1$ 인 경우에는 k값을 1로 하여 LSM이 구성된다. 효율적인 탐색을 위해, $k \geq 4$인 경우에는 k는 $|b/4|$씩 증가하며, 그렇지 않은 경우에는 1씩 증가한다. 똑같은 절차가 b에도 적용될 수 있다.

LSM, $LS(c,d,e,f)$를 할당하기 위해서, $S_L(c,d)$를 FF 또는 BF 기법을 적용하여 할당할 수 있는 좌표를 찾은 후에, 그 좌표를 이용하여 $S_R(e,f)$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, $\alpha \in \{1, 4, a+2, 2a\}$를 결정한다. 이 가중치 α는 임베딩 코스트이며, 그 할당이 네트워크 전파 시간을 얼마나 증가시키게 할지를 반영한다. 시스템 $M(w,h)$로 들어오는 작업 $S(a,b)$에 대해 LSSAFF와 LSSABF 기법의 할당 시간 복잡도는 $\theta(wh)$이고, 해제 시간 복잡도는 $\theta(ab)$이다.

LSSA 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 않는다. 즉, 가장 먼저 끝나게 되는 작업이 완전히 끝날 때까지 남아있는 시간과 예상되는 수행시간의 증가분을 비교하여, 남아있는 시간이 증가분보다도 크다면 변형된 모양의 서브메쉬를 할당한다. 시스템으로 들어오는 각 작업 i에 대해서 t_i는 통계적으로 예측된 수행시간을 나타내며, f_i는 통신 프랙션으로서, 형태 변형으로 인해 영향을 받게될 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 증가 분은 $\alpha f_i t_i$로서, 이와 비교하여 할당여부를 결정한다. 예를 들어, 그림 4에서 태스크 1부터 태스크 3까지의 순서로 끝나도록 스케쥴되어 있다고 가정할 때, 새로운 태스크 4가 들어와서 $S(5,5)$를 요구했을 경우, LSSA 기법은 태스크 1이 $6f_4t_4$ 단위 시간 내에 끝나지 않는 경우에만 $LS(3,7,2,2)$를 할당하도록 결정한다. LSSA 기법은 이와 같은 보존 검사를 하므로, 변형된 모양의 서브메쉬를 할당하지 않을 수도 있다.

5. 성능 분석

LSSA 기법, 특히 LSSAFF, LSSABF 기법을 기존의 사각형 서브메쉬 할당 기법들의 성능과 비교하기 위해서 이산 사건 중심의 시뮬레이션을 수행하였다. 이 이산 사건 시뮬레이터는 메쉬 시스템에서 일련의 작업들이 도착되고, 서비스 받고, 떠나는 과정을 공정성을 유지할 수 있는 FCFS 스케쥴링을 채택하여 모델링 하였다. 시스템 부하 (system load)는 작업들이 도착되는 간격과 그 작업들의 체재시간의 비율에 의해 결정되므로, 다음과 같이 정의하였다 [10]:

$$\text{시스템 부하} = \frac{\text{평균 체재시간} \times \text{평균 요청된 서브메쉬 크기}}{\text{시스템의 크기} \times \text{평균 도착되는 시간 간격}} \quad (1)$$

여기서, 체재 시간은 요청된 크기의 서브메쉬를 그대로 할당 받았을 때 시스템에서 그 응용 프로그램이 수행되는 시간이다. 시스템의 활용도는 메쉬 시스템에 있는 프로세서들이 평균적으로 얼마나 사용되고 있는지를 나타낸다. 시스템의 활용도를 측정하기 위해서 매 단위 시간 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[9]:

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

여기서, wh는 전체 메쉬 시스템의 크기를 의미한다. 성능분석을 하기 위해 평균 작업 응답 시간 (MRT : Mean Response Time), 1000 개의 태스크들을 완료하는 시간, 그리고 시스템 활용도를 측정하였다.

시뮬레이터는 할당기법에 따라 다른 모양의 프리 서브메쉬들을 탐색하며, 다음과 같은 할당 기법들을 모델링하여 시뮬레이터에 포함하였다. First-Fit (FF), Best-Fit (BF), ASFF (Adaptive Scan with First Fit), ASBF (Adaptive Scan with Best Fit), FlexFF(Flexfold with First Fit), FlexBF(Flexfold with Best Fit), LSSAFF, 그리고 LSSABF.

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 [9,10,16] 에서와 같이, FF 기법과 BF 기법을 적용하였을 때의 MRT의 차이는 매우 작음을 그림 6에서도 알 수 있다. 따라서 각 기법을 명확하게 보이기 위해, 다른 그래프에서는 FF기법을 적용한 결과만을 나타낸다.

LSSA 기법을 적용하였을 경우의 MRT가 FF, BF, ASFF, ASBF, FlexFF, 그리고 FlexBF의 경우보다 작음을 그림 6에서 볼 수 있다. Flexfold 기법과 비교하면,

10% 정도 높은 시스템 부하에서도 시스템이 안정되어 있음을 볼 수 있다. 시스템 부하가 증가할수록, LSSA 기법의 장점은 두드러진다. 시스템 부하가 0.55일 때, Flexfold 기법과 비교하면 MRT가 34% 정도 감소하였다. 그림 7은 그림 6과 같은 환경에서 각 할당 기법들이 1000 개의 작업을 완성시키는 시간을 측정한 것이다. 시스템 부하가 0.4 이하에서는 AS, Flexfold 기법들과 거의 비슷한 완성 시간을 보이지만, 시스템 부하가 0.5 이상으로 증가할 경우에는 LSSA 기법의 완성시간이 다른 기법들에 비해서 상대적으로 작다. 시스템 부하 0.6에서는 LSSA 기법의 완성시간은 Flexfold 기법에 비해서 28%, AS 기법에 비해서는 32% 정도 감소하였다.

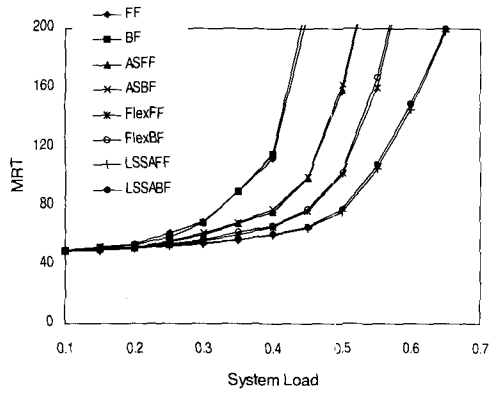

그림 6 시스템 부하의 변화에 따른 MRT의 개선도(평균 체재 시간=50, 작업이 도착되는 간격을 변화시킴)

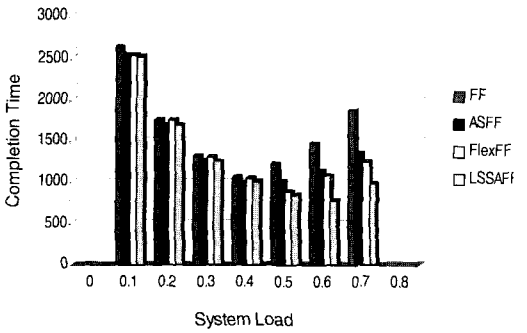

그림 7 시스템 부하의 변화에 따른 Completion time의 비교 (평균 체재 시간=50, 작업이 도착되는 간격을 변화시킴)

그림 8은 각 작업의 요청된 체재시간은 60 단위시간을 평균으로 하는 지수 분포를 갖도록 하고, 시스템 부하를 작업이 도착되는 간격을 변화시킴으로써 조절하였을 때, 1000 개의 작업을 완성시키는 시간을 측정한 것이다. 시스템 부하가 0.6 이상일 경우에 LSSA 기법의 완성시간이 다른 기법들에 비해서 상대적으로 작다. 시스템 부하 0.7에서는 LSSA 기법의 완성시간은 Flex fold 기법에 비해서 12%, AS기법에 비해서는 39%정도 감소하였다.

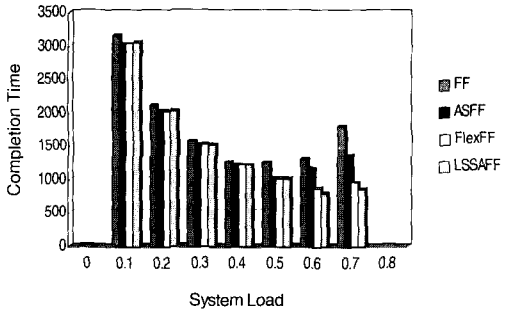

그림 8 시스템 부하의 변화에 따른 Completion time의 비교 (평균 체재 시간=60, 작업이 도착되는 간격을 변화시킴)

그림 9는 작업이 도착되는 간격을 평균 60을 갖는 지수분포로 발생시키고, 시스템 부하를 각 작업의 요청된 체재시간을 변화시킴으로써 조절하였을 때, MRT의 변화를 측정한 것이다. 나머지 조건들은 그림 6에서와 같이 유지하였다. 시스템 부하가 0.6일 때, Flexfold 기법과 비교하면 MRT가 37% 정도 감소하였으며, LSSA 기법을 적용하였을 경우의 MRT가 FF, ASFF, 그리고 Flexfold 기법을 적용하였을 경우보다 작음을 볼 수 있다.

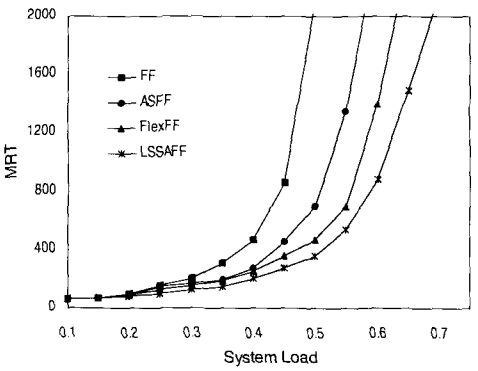

그림 9 시스템 부하의 변화에 따른 MRT의 개선도(평균 체재 시간=60, 체재시간을 변화시킴)

그림 10은 요구되는 작업들의 크기를 지수 분포로 발생시켰는데, 각 변의 길이는 2에서 27로 제한하여 앞의 실험에서보다 더 큰 크기의 작업을 발생시켰으며, 나머지 조건들은 그림 6에서와 같이 유지하였다. 각 작업이 더 큰 크기의 서브메쉬를 요구할수록 LSSA 기법은 더 효과적으로 작동함을 알 수 있다. 즉, 시스템 부하가 0.55일 때, Flexfold 기법에 비해서 MRT가 47% 정도 감소하였다.

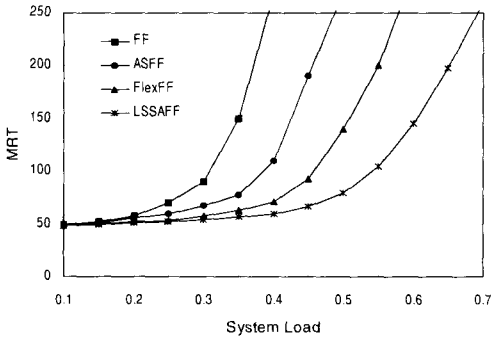

그림 10 시스템 부하의 변화에 따른 MRT의 개선도(큰 크기의 작업을 발생시킴)

LSSA 기법을 적용하였을 때의 시스템의 활용도는 발생되는 각 작업의 크기에서 각 변의 길이를 2에서 27로 제한하여, 균일분포로 독립적으로 발생시켰으며, 1000개의 entity를 발생시켜 그림 9에서와 같은 환경에서 측정하였다. 그림 11에서 보듯이, 시스템 부하가 0.8일 때, 62%까지 증가한다. 이것은 최초 적합을 적용하였을 때의 시스템의 활용도가 20%, Flexfold 기법을 적용하였을 경우의 39% 인 것에 비하면 매우 좋은 결과이다. 시스템 부하가 1 이상 일 때 시스템의 활용도는 64%에 이른다.

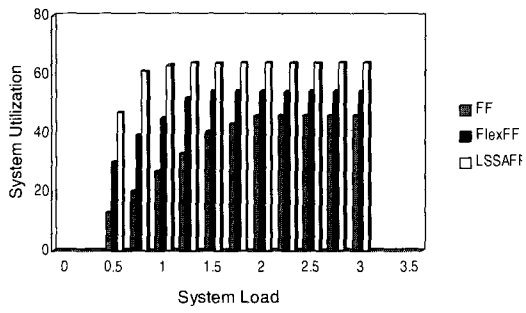

그림 11 시스템 부하의 변화에 따른 시스템의 활용도

이 높은 시스템의 활용도는 외부 프래그멘테이션을 발생시키면서 대기 큐에서 기다리는 대신에 LSM을 할당하기 때문에, 그리고 크기가 큰 작업을 즉시 수용할 능력이 기존의 다른 기법들보다도 많기 때문에 가능하였다.

시스템 부하가 작을 때는 LSSA 기법도 기존의 다른 기법들처럼 직사각형 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 때는 대기 큐에 쌓이는 작업들이 많아지므로, 요청된 서브메쉬의 모양을 L-shaping하여 LSM을 할당할 수 있는 LSSA 기법이 대기 큐 헤드의 병목 현상을 풀어 줄 수 있기 때문이다. 또한 시스템에서의 체재시간이 긴 작업들이 많을수록, 요청된 서브메쉬의 크기가 큰 경우에도 MRT, 완성시간, 그리고 시스템의 활용도에서 LSSA 기법의 성능이 기존의 다른 기법들보다 향상된다.

6. 결 론

2차원 메쉬 멀티 컴퓨터에서 서브메쉬의 모양을 L자 모양으로 변형하여 할당할 수 있는 적응적인 프로세서 할당기법을 제안하였다. 메쉬 시스템에 대한 기존의 프로세서 할당 기법들은 직사각형 서브메쉬 할당으로 제한함으로써, 심각한 외부 프래그멘테이션 문제를 갖고 있었으며, 이로 인해 시스템의 활용도가 매우 낮았다. 그러나 제안된 LSSA 기법은 직사각형 또는 L자 모양의 서브메쉬를 할당할 수 있는 적응성을 제공하므로, 기존의 다른 기법들이 즉시 할당할 수 없는 작업, 특히 큰 크기의 작업의 요청을 빨리 수용할 수 있다. 기존의 기법을 적용하였을 때의 시스템 활용도 34 ~ 46% [11] 보다 높은 62%까지 LSSA 기법은 제공할 수 있다. LSSA 기법의 평균 작업 응답 시간이 다른 기법들에 비해서 감소되는 것을 시뮬레이션 결과를 통해서 보였다. 그러므로 LSSA 기법은 평균 작업 응답 시간을 줄이고, 동시에 시스템의 활용도를 높일 수 있다. 또한 요청된 작업이 L자 모양의 서브메쉬에서 수행되는 경우에도 응용 프로그래머에게 투명성을 보장하므로, 그 프로그램을 다시 작성할 필요가 없다.

LSSA 프로세서 할당 기법을 적용하면, 메쉬 멀티 컴퓨터를 사용하는 응용 프로그래머들은 자신의 프로그램을 직사각형 서브메쉬에서 수행되도록 하기 위해 dummy 프로세서들을 포함시킬 필요가 없다. 즉, 작업을 수행하는데 필요한 가장 적당한 개수의 프로세서들

을 정확하게 요구할 수 있다. 또한, LSSA 기법은 시스템에 오류가 있는 프로세서들이 존재할 경우에도, 유연하게 대처할 수 있을 것으로 기대된다.

참 고 문 헌

[1] P. Mazumder, "Evaluation of On-Chip Static Interconnection Networks," *IEEE Trans. on Computers*, Vol. C-36, No. 3, pp. 365-369, Mar. 1987.

[2] A. G. Ranade and S. L. Johnsson, "The Communication Efficiency of Meshes, Boolean Cubes and Cube Connected Cycles for Wafer Scale Integration," *Proc. Int'l Conf. on Parallel Processing*, pp. 479-482, 1987.

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

[4] 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.

[5] 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.

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

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

[8] 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.

[9] 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.

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

[11] W. Liu, V. Lo, K. Windisch and B. Nitzberg, "Non-contiguous Processor Allocation Algorithms for Distributed Memory Multicomputers," *Proc. 1994 Int'l Conf. on Supercomputing*, pp. 227-236, June 1994.

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

[13] 서경희, 김성천, "메쉬 시스템에서 형태 변형에 의한 유연성있는 프로세서 할당 기법", 정보과학회논문지, 제24권, 제11호, pp. 1113-1123, 1997.

[14] J. W. Hong, K. Mehlhorn, and A. L. Rosenberg, "Cost Trade-offs in Graph Embeddings, with Applications," *Journal of the ACM*, Vol. 30, No. 4, pp. 709-728, October 1983.

[15] Hwang, K. and Xu, Z., Scalable Parallel Computing: Technology, Architecture, Programming, p.280, WCB/McGraw-Hill, 1998.

[16] 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.

서 경 희

1986년 서강대학교 수학과(이학사). 1989년 서강대학교 전자계산학과(공학사). 1992년 서강대학교 컴퓨터학과(공학석사). 1998년 서강대학교 컴퓨터학과(공학박사). 1999. 3 ~ 현재 성신여자대학교 컴퓨터정보학부 계약교원. 관심분야는 병렬처리, 광통신

김 성 천

1975년 서울대학교 공과대학 공업교육학(전기전공)학사. 1979년 Wayne State Univ. 컴퓨터공학 공학석사. 1982년 Wayne State Univ. 컴퓨터공학 공학박사. 1982년 ~ 1984년 캘리포니아주립대 조교수. 1984년 ~ 1985년 금성반도체(주) 책임연구원. 1986년 ~ 1989년 서강대학교 공과대학 전자계산소 부소장. 1989년 ~ 1991년 서강대학교 공과대학 전자계산학과 학과장. 1985년 ~ 현재 서강대학교 공과대학 전자계산학과 교수(1992.9~현재). 1989년 ~ 현재 한국정보과학회 병렬처리시스템 연구회 부위원장(1989년 ~ 1993년), 위원장(1994년 ~ 1997년), 대한전자공학회 및 한국통신학회 논문지 편집위원(1991년 ~ 현재, 1993년 ~ 현재). 관심 분야는 병렬처리시스템(Parallel Computer Architecture, Interconnection Network)