연속적인 서브메쉬 할당기법에서 단편화를 최소화하는 기법

서 경 희[†] · 김 성 천[††]

요 약

대규모 멀티컴퓨터 시스템에서 단편화를 줄일 수 있는 적응성 있는 프로세서 할당 기법을 제안한다. 큰 크기의 서브메쉬 할당을 요구하는 작업들의 개수가 적을 경우에도 나머지 작업들의 대기 큐의 지연시간이 증가될 수 있다. 이런 상황에서 할당이 불가능한 기존의 기법들과 달리, L-모양 서브메쉬를 할당할 수 있으며, 할당 가능한 L-모양 서브메쉬를 효율적으로 탐색하는 알고리즘을 개발하였다. 그러므로 FCFS로 스케쥴링되어도, 대기 큐의 지연시간을 줄임으로써 평균응답시간을 줄일 수 있다. 시뮬레이션 결과를 통해서 제안하는 기법이 외부 단편화, 작업응답시간, 그리고 시스템의 활용도 면에서 다른 기법들보다 우수함을 보인다.

Minimizing Fragmentation in Contiguous Submesh Allocation Scheme

Kyung Hee Seo[†] · Sung Chun Kim[††]

ABSTRACT

This paper presents an adaptive processor allocation strategy to reduce fragmentation in a large multi-user multicomputer system. A small number of jobs with unexpectedly large submesh allocation requirements may significantly increase the queuing delay of the rest of jobs. Under such circumstances, our strategy further tries to allocate L-shaped submeshes instead of signaling the allocation failure unlike other strategies. We have developed the efficient algorithm to find the allocatable L-shaped submeshes. Thus, our strategy reduces the mean response time by minimizing the queuing delay, even though jobs are scheduled in an FCFS to preserve fairness. The simulations show that our strategy performs more efficiently than other strategies in terms of the job response time and the system utilization

키워드 : 단편화(fragmentation), 프로세서 할당(processor allocation), 서브메쉬(submesh)

1. 서 론

멀티컴퓨터 운영체제의 중요한 기능 중의 하나는 프로세싱 자원들을 효율적으로 관리함으로써 사용자들의 요구를 빠르고 공정하게 지원할 수 환경을 제공하는 것이며, 이를 위해서 효율적인 프로세서 할당 알고리즘이 요구된다. 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. 이는 작업 응답 시간을 줄이면서, 동시에 시스템의 활용도를 증가시킴으로써 가능해진다. 높은 시스템의 활용도를 얻기 위해서는 발생되는 단편화(fragmentation)의 양을 최소화 시켜야 한다. 또한, 할당 알고리즘의 할당 가능한 서브메쉬를 인식할 수 있는 능력이 증가되기 위해서는 높은 실행시간 오버헤드가 수반된다[11,18]. 기존의 대규모 멀티컴퓨터 시스템에서 제안되었던 할당 알고리즘들은 성능과 복잡도 사이에서 다양한 트레이드 오프를 보여주고 있다.

하나의 작업에 할당되는 프로세서들이 서로 이웃하고 있는가, 그렇지 않은가에 따라서 프로세서 할당 기법은 연속적(contiguous) 또는 비연속적(non-contiguous) 할당 기법으로 구분지울 수 있으며, 상업용 멀티컴퓨터는 연속적인 할당 기법으로 제한되어왔다[17]. 연속적인 프로세서 할당 기법의 성능에 병목현상을 초래하는 주요 요인은 단편화이며, 이 문제를 해결하기 위해 비연속 할당 기법이 제안되었다[12]. 그러나 한 작업에 할당되어있는 프로세서들이 서로 통신하기 위해 다른 작업에게 할당되어있는 프로세서들의 통신 중재가 요구되며, 아는 각 작업이 자신에게 할당된 프로세서들을 독점적으로 사용할 수 없게 하며, 메시지 전송 충돌 가능성을 상당히 증가시키게 된다. 또한, 예측할 수 없는 통신 지연으로 인해 비연속적인 할당 기법이 여러 응용분야에 적합하지 않을 수 도 있다[3].

연속적인 프로세서 할당 기법에서 발생할 수 있는 단편화의 종류는 내부, 외부, 그리고 가상 단편화이다. 초기의 할당 기법인 2차원 Buddy(2DB) 기법은 심각한 내부 단편화을 발

† 정 회 원 : 성신여자대학교 컴퓨터정보학부 초빙교수
†† 정 회 원 : 서강대학교 컴퓨터학과 교수
논문접수 : 2005년 2월 19일, 심사완료 : 2005년 3월 25일

생시킨다[10]. 그러나 최근에 제안된 프로세서 할당 기법들은 내부 단편화를 발생시키지 않으며, 이용 가능한 서브메쉬들을 완전히 인식할 수 있는 능력을 갖고 있다[4,5,11,14,18,19]. 따라서 내부 및 가상 단편화는 더 이상 문제가 되지 않는다. 여전히 문제가 되는 것은 외부 단편화로서, 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 단편화를 줄이기 위한 최초의 시도는 AS(Adaptive Scan) 기법에서 찾을 수 있으며[5], 또 하나의 시도는 Flexfold 기법이다[6]. 그러나 Flexfold 기법을 포함한 모든 연속적인 프로세서 할당 알고리즘들은 전체 시스템 구조와 같은 위상을 가지는 사각형 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 사각형 서브메쉬를 형성하지 못하면, 그 작업은 대기 큐로 보내진다. 시스템에서 할당해제가 발생했을 때, 대기 큐의 헤드에 있는 작업에게 할당할 수 있는 서브메쉬를 찾기 위한 시도가 이루어지므로, 그 작업이 가능한 빨리 처리되지 않으면, 병목현상이 발생한다. 그 결과, 외부 단편화 문제는 여전히 심각한 상태이며, 이로 인해 34%에서 46%의 시스템의 활용도를 보이고 있다[12].

이러한 문제를 해결하기 위해서 FCFS 방식으로 작업을 처리하면서 효율적인 프로세서 할당 기법을 연구하는 대신에, 다른 작업 스케줄링을 채택하는 것이 더 바람직하다는 연구결과가 있다[15]. 그러나 FCFS 방식이 아닌 다른 작업 스케줄링을 채택하는 것은 사용자에게 공정하지 않다는 단점을 가지며, 또한 크기가 큰 작업을 특히 더 차별하게 되는 경향으로 인해 기근 문제를 유발시킬 수 있다.

본 논문에서는 공정한 FCFS 스케줄링을 채택하면서도 발생되는 단편화를 최소화하여 시스템의 활용도를 높이고, 동시에 작업 응답 시간을 줄일 수 있는 LSA(L-Shaped Submesh Allocation) 기법을 제안한다. LSA 기법은 LSSA 기법[13]을 확장시킨 것으로서, L-모양 서브메쉬를 형식화하여 사각형 서브메쉬가 L-모양 서브메쉬로 효율적으로 임베딩 될 수 있음을 보였으며, 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 시간과 작업 완료 시간들을 줄이고, 시스템의 활용도가 증가함을 보인다. 본 논문의 구성은 다음과 같다. 2장에서는 기존의 서브메쉬 할당 기법들을 간략하게 소개하고, 3장에서는 기본적인 용어 및 확장된 LSM을 정의한다. 제안하는 LSA 기법을 4장에서 소개하고, 5장에서는 LSA 기법의 성능을 시뮬레이션을 통해 분석한다. 6장에서 결론을 맺는다.

2. 기존의 할당 기법들

2DB 기법은 한 변의 길이가 2의 멱승인 정사각형 모양의 서브메쉬 만을 할당할 수 있다[10]. 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. 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 폭을 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[19]. 할당 가능한 서브메쉬를 더 빨리 찾기 위해서, 각 프로세서를 각 비트에 대응시키는 두 개의 비지 배열과 적용범위 배열을 사용한다. 그러나 요청된 서브메쉬를 할당할 수 없을 때, 90° 회전시킨 서브메쉬를 고려하지 않으므로, 가상 단편화를 발생시키며, 외부 단편화도 심각하다.

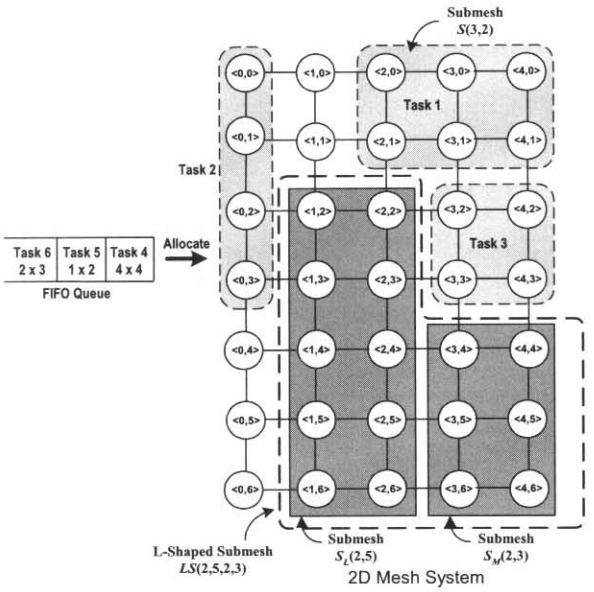
FS 기법에서 발생되는 가상 단편화를 제거하기 위해서 제안된 AS 기법은[5] 고정된 폭을 사용하지 않고, 적응적으로 스캔함으로써 할당 가능한 서브메쉬를 완전하게 인식할 수 있으며, 요청된 서브메쉬를 할당할 수 없을 때, 90° 회전한 서브메쉬를 할당하려는 시도를 처음으로 하였다. 그러나 이 할당 알고리즘의 최초적합 특성 등으로 인해, 여전히 외부 단편화 문제가 심각하게 남아있다.

요청된 서브메쉬의 모양을 90° 회전시키는 것 외에, 폴디드 서브메쉬로 변환하여 할당을 고려하는 Flexfold 기법[6]은 다른 기법들보다 더 많은 할당 기회를 제공한다. 그러나 Flexfold 기법에서 사용되는 폴딩(folding)은 요청된 서브메쉬의 두 변 a 와 b가 모두 짝수일 때만 적용될 수 있다[6]. 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[13]. 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. L-모양 서브메쉬, LSM

2차원 메쉬 $M(w,h)$ 는 wh 개의 노드들로 구성되며, 너비 w 와 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. (그림 1)에서 2차원 메쉬 $M(5,7)$과 각 노드들의 주소를 보이고 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 $<x_1,y_1,x_2,y_2>$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)에서 주소가 <2,1,4,0>인 서브메쉬 S(3,2)를 보이고 있다.

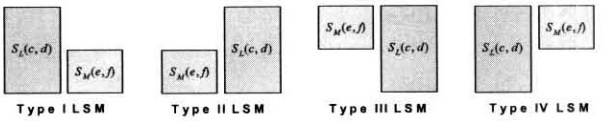


(그림 1) 2D 메쉬 M(5,7)에서의 사각형 또는 L-모양의 서브메쉬 할당

[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, $S(c,d)$는 $S(e,f)$에 L-adjacent하다고 한다:

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 정의 1의 어떤 조건을 만족하는가에 따른 4가지 L-adjacent 경우들을 보이고 있다. 두 개의 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 <1,6,2,2>인 서브메쉬 $S(2,5)$와 주소 <3,6,4,4>인 서브메쉬 $S(2,3)$은 정의 1의 첫 번째 조건을 만족하는 L-adjacent한 경우로 $S_L(2,5)$와 $S_M(2,3)$로 각각 표기할 수 있다.



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

[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 L-adjacent 서브메쉬 $S_L(c,d)$ 와 $S_M(e,f)$의 프로세서들의 집합으로 구성된다.

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 $S_M(e,f)$의 하단-좌측 모서리와 상단-우측 모서리를 각각 나타낸다. (그림 1)에서 두 L-adjacent 서브메쉬 $S_L(2,5)$와 $S_M(2,3)$는 긴 너비 4와 긴 높이 5를 가지는 LSM $LS(2,5,2,3)$을 구성하며, 좌표는 [<1,6,2,2> & <3,6,4,4>]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 만들어지는 과정은 다음과 같다.

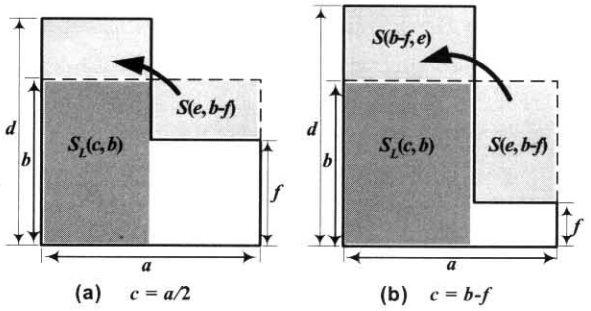
Step 1: $S(a, b)$의 커팅 사이드 결정.
if ab is even, then the larger even side is chosen as the cutting side
else, the smaller odd side is the cutting side.

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. 예를 들어, (그림 4)에서 같이, $S(4,4)$로 부터 만들어질 수 있는 LSM들은 $LS(2,5,2,3)$, $LS(2,6,2,2)$, $LS(2,7,2,1)$, 그리고, 폴디드 서브메쉬 $S(2,8)$이다. 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.



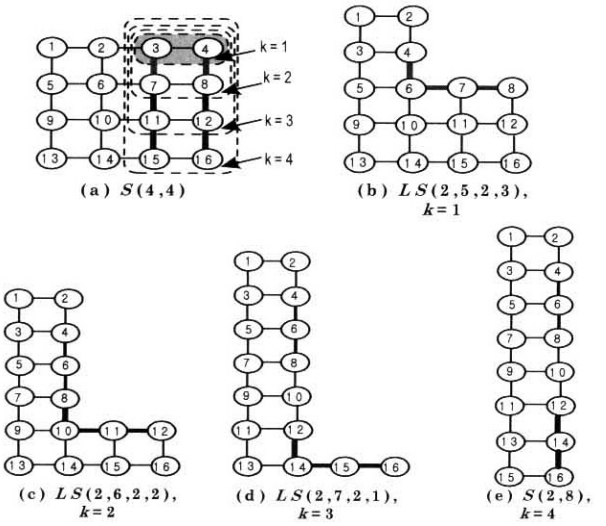
(그림 3) 서브메쉬 S(a,b)로부터 LS(c,d,e,f)의 구성 (a) a가 짝수인 경우 (b) a가 홀수인 경우

초기의 2DB 기법이 할당하는 서브메쉬의 모양은 각 변이 2^n인 정사각형이었으며[10], 그 이후에 제안된 모든 서브메쉬 할당 기법들은 사각형 모양의 서브메쉬를 고려하였다[4-6,11,14,19]. 이제 사각형의 서브메쉬에서 각 사분면이 존재하지 않는 더 유연성 있는 메쉬, 확장된 LSM을 고려한다. 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.

4. LSA 기법

LSA 기법은 다음 예에서와 같이 작업 응답 시간과 외부 단편화를 동시에 감소시키는데 초점을 맞추고 있다. (그림 1)에서 5×7 메쉬 시스템을 3 개의 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 3×2, 1×4, 그리고 2×2 이고, 각 태스크에게 할당된 서브메쉬의 좌표는 각각 <2,1,4,0>, <0,3,0,0>, 그리고 <3,3,4,2>이다.

시스템으로 들어오는 새로운 태스크 4가 서브메쉬 $S(4,4)$를 요구했을 때, 기존의 기법들은 이 태스크를 즉시 수용할 수 없다. 현재 메쉬 시스템에는 21 개의 이용 가능한 프로세서들이 존재함에도 불구하고, 기존의 기법들은 태스크 4를 대기 큐에 넣을 수밖에 없다. 그러나 LSA 기법은 $S(4,4)$가 (그림 4) (b)에서와 같이, LSM $LS(2,5,2,3)$으로 변형될 수 있으며, 현재의 메쉬 시스템에서 이용 가능하며, 그 좌표가 [<1,6,2,2> & <3,6,4,4>] 임을 인식할 수 있다. 이 상황에서, LSA 기법은 다른 기법들 보다 빨리 태스크를 수용할 수 있으며, 시스템의 활용도도 Flexfold 기법을 적용했을 경우의 40%에서 85.7%로 향상시킬 수 있다.



(그림 4) S(4,4)로부터 구성되는 L-모양 서브메쉬들과 폴디드 서브메쉬

서브메쉬 $S(1,b)$을 L-shaping한 경우, LSM에서의 노드들 간의 통신거리는 변환 전 서브메쉬에서의 통신거리보다 증가되지 않는다. 실제로 노드들 간의 통신 거리는 더 가까워진다. 서브메쉬 $S(1,7)$에서 노드 1과 노드 7의 거리는 6 홉이지만, $LS(1,5,1,2)$에서는 4 홉으로, $LS(1,4,1,3)$에서는 2 홉으로 감소한다. 그러나 L-shaping이 통신거리를 항상 감소시키지는 않는다. (그림 4)에서 노드 4 와 노드 8은 서브메쉬 $S(4,4)$에서 서로 이웃하고 있지만, $LS(2,5,2,3)$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 태스크의 실행시간을 증가 시킬 수 있지만, LSM을 할당하는 것은 대기 큐의 지연시간을 확실히 줄일 수 있고, 기다리고 있는 다음 태스크들에게도 빨리 서비스할 수 있는 기회를 제공한다. 따라서 각 태스크들의 할당을 기다리는 시간의 감소량이 LSM을 할당 받음으로써 증가하게 되는 태스크의 실행시간 보다 크다면, 전체 태스크들의 작업 응답 시간은 감소하게 될 것이다.

4.1 사각형 서브메쉬의 LSM으로의 임베딩

폴딩 또는 L-shaping으로 인한 서브메쉬 형태의 변형이

태스크의 실행시간에 미치는 영향을 분석하기 위해서 그러한 형태의 변형을 하나의 서브메쉬에서 다른 서브메쉬로의 그래프 임베딩으로 간주할 수 있다[8]. 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. 임베딩은 한 그래프에서 설계된 알고리즘을 다른 그래프에서 수정 없이 사용가능하도록 한다. 임베딩 비용을 측정하는 척도로 확장비율(expansion), 로드(load), 연장비율(dilation), 그리고 혼잡비율(congestion)이 있다[16]. 확장비율은 $|V|/|V'|$이며, V'에 속하는 정점 v의 로드는 $|\{u \mid u\in V, \varepsilon(u) = v\}|$이다. 그래프 G에서 이웃하는 두 정점 u 와 v에 대해서, G'으로 대응되는 두 정점 $\varepsilon(u)$ 와 $\varepsilon(v)$ 사이의 거리를 그 간선 (u,v)의 연장비율이라 한다. 임베딩 ε의 연장비율은 G의 모든 간선에 대한 연장비율 중 최대 값으로 결정된다. 혼잡비율은 G'에서 하나의 간선에 대해서, 이 간선을 포함하는 G' 상의 경로에 대응되는 G에서의 간선들의 최대 개수를 의미한다.

이 용어들을 사용하면, 사각형 서브메쉬의 LSM으로의 임베딩은 확장비율 1, 로드 1, 그리고 혼잡비율 2이다. 폴디드 서브메쉬로의 임베딩은 (그림 4) (e)에서 보듯이, 연장비율이 2이다. 이제, 사각형 서브메쉬를 커팅 사이드 a에 대해서 연장비율이 $a/2+1$(짝수 a) 또는 a(홀수 a)를 초과하지 않도록 LSM으로 임베딩할 수 있음을 보이고자 한다.

[정리 1] 짝수 변을 가지는 사각형 서브메쉬는 폴디드 서브메쉬로 연장비율 2, 혼잡비율 2로 임베딩 될 수 있다.

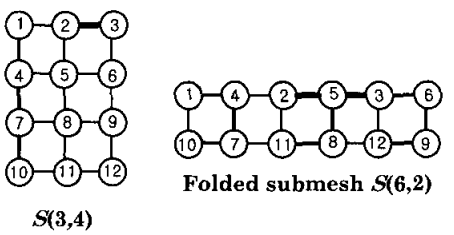
[증명]

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,

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

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

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



(그림 5) 폴디드 서브메쉬로의 임베딩의 예

[정리 2] 서브메쉬 $S(a,b)$는 짝수 커팅 사이드 a에 대해서 연장비율과 혼잡비율이 각각 $a/2+1$과 2가 되도록 LSM 으로 임베딩할 수 있다.

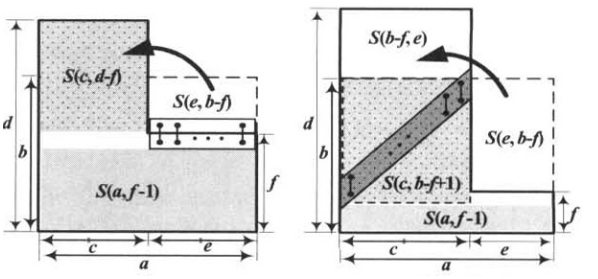
[증명]

G를 서브메쉬 $S(a,b)$라 하고 H를 LSM $LS(c,d,e,f)$, $cd+ef=ab$, $d\geq f$, $a\geq b$, 그리고 $c+e=a$라 하자. $S(a,b)$의 상단-좌측 모서리의 좌표를 (0,0)로 하고 각 노드에 주소를 부여하면, 하단-우측 모서리의 주소는 $(a-1, b-1)$이 된다. G에서 H로의 함수 ε는 $\varepsilon(i, j) = (h, k)$, 즉 G에서의 노드 (i, j)를 H에서의 노드 (h, k)로 다음과 같이 정의한다. 여기서 $l = i + ja$ 이다.

$$h = \begin{cases} l \bmod c, & l < c(d-f) \\ l \bmod a, & otherwise, \end{cases}$$

$$k = \begin{cases} \lfloor l/c \rfloor, & l < c(d-f) \\ j+d-b, & otherwise. \end{cases}$$

G에서의 간선 $e=\{(i, j), (i', j')\}$에 대해서 $\varepsilon(e)$는 $\varepsilon(i, j)$에서 $\varepsilon(i', j')$사이의 최단거리로 정의한다. 이 함수 ε를 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. 서브메쉬 $S(c, d-f)$는 $S(a, b-f)$의 폴디드 서브메쉬로 간주 될 수 있다. 이 폴딩 임베딩의 연장비율은 $a/2$이다. (그림 6) (a)에서 보듯이, $S(e, b-f)$의 하단에 위치한 노드들만이 수직 아래방향으로 이웃하던 노드들과의 연장비율이 가장 크며, $a/2+1$이다. yx-라우팅을 사용하여 노드간의 통신을 수행할 경우에, 커팅 사이드의 짝·홀에 관계없이 혼잡비율은 2가 된다. 그러므로 LSM으로의 변형에서 커팅 사이드 a가 짝수일 때 연장비율과 혼잡비율은 각각 $a/2+1$ 과 2보다 작거나 같다. ■



(그림 6) L-모양 임베딩

[따름정리 1] 서브메쉬 $S(a,b)$는 홀수 커팅 사이드 a, $a\geq b \geq 3$, 에 대해서 연장비율 $3+2k$가 되도록 LSM $LS(\lceil a/2 \rceil + k, b+\lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k,$

$b-\lceil a/2 \rceil - k$)로 임베딩 할 수 있다.

[증명]

커팅 사이드 a가 홀수인 경우 서브메쉬 S(a,b)에서 만들어질 수 있는 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. 정리 2에서 정의된 함수 ε를 사용하면, $S(a, f-1)$에 포함되는 노드들 간의 상호연결 관계는 $LS(\lceil a/2 \rceil + k,\ b+\lfloor a/2 \rfloor - k, \lfloor a/2 \rfloor - k,\ b-\lceil a/2 \rceil - k)$에서도 그대로 유지된다. (그림 6) (b)에서 보듯이, 서브메쉬 $S(c,\ b-f+1)$의 대각선에서 수직으로 이웃하던 노드들의 연장비율이 가장 크며, $3+2k$가 된다. 그러므로 이 L-모양 임베딩의 연장비율은 $3+2k$이다. ■

$a = b$ 이고, a는 홀수, k가 최대값, $b-1-\lceil a/2 \rceil$ 인 경우 S(a,a)로부터 만들어지는 LSM은 $LS(a-1,\ a+1,\ 1,1)$이며, 이 임베딩의 연장비율은 a이고, 혼잡비율은 2이다. 따라서 LSA에서의 형태변형은 연장비율 a, 혼잡비율 2를 초과하지 않는다.

LSA 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, $a \ge b$ 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 멀티 컴퓨터의 메시지 전송시간 67μs 중에서 네트워크에서 소비되는 시간은 1μs 보다도 작으며[6], 또한 충돌 프리 메시지 패싱 네트워크 시스템에서 전체 통신 시간 중에 네트워크 전파 시간이 차지하는 비중이 1.5% 이하라는 보고도 있다[7]. 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 LSA 알고리즘

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*(a,b)에 의해 a, b의 값에 따라 후보 프리 서브메쉬들의 탐색순서를 결정한다. Procedure *L-shaping*(c)는 이용 가능한 LSM을 찾는다.

Procedure *DetermineSearchSequence*(a, b)

```
{
 case : a = b  /* 두 변 a, b 가 같은 정사각형 서브메쉬 */
   if (a is even) then S(a, b), S(a/2,2b), S(2b,a/2), LSMs
   using L-shaping(a)
     else S(a, b), LSMs using L-shaping(a)
 case : a ≠ b
   determine the cutting side, c, of the requested sub-
   mesh
   if (both a and b are even) then 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(c )
   if (both a and b are odd) then S(a, b), S(b, a), LSMs
   using L-shaping(c )
   if (a is even and b is odd) then S(a, b), S(b, a),
   S(a/2, 2b) , S(2b, a/2), LSMs using L-shaping(c )
   if (a is odd and b is even) then S(a, b), S(b, a), S(2a,
   b/2), S(b/2, 2a), LSMs using L-shaping(c )
 }
```

Procedure *L-shaping*(c)/* c : 커팅 사이드 */

```
{
 case : c is even
   if (b≥4 ), then LS(a/2, b+ k, a/2, b- k), k =⌊ b/4⌋ ,
   ⌊ b/2⌋ , ⌊ 3b/4⌋
     else LS(a/2, b+ k, a/2, b- k), 1≤k≤b-1.
 case : c is odd
   if (b≥9 ), then LS(⌈ a/2⌉ + k, b+⌊ a/2⌋ - k,⌊ a/2⌋ -
   k, b-⌈ a/2⌉ - k ), k = 0, ⌊ b/4⌋ , b-1-⌈ a/2⌉ ,
     else LS( ⌈ a/2 ⌉ + k, b+⌊ a/2⌋ - k,⌊ a/2⌋ - k, b-
     ⌈ a/2⌉ - k ), 0 ≤ k ≤b-2-⌈ a/2⌉ .
}
```

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, α∈{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 태스크 1부터 태스크 3까지의 순서로 끝나도록 스케쥴되어 있다고 가정할 때, 새로운 태스크 4가 들어와서 $S(4,4)$를 요구했을 경우, LSA 기법은 태스크 1이 $6f_4t_4$ 단위 시간 내에 끝나지 않는 경우에만 $LS(2,5,2,3)$을 할당하도록 결정한다. LSA 기법은 이와 같은 보존 검사를 하므로, 변형된 모양의 서브메쉬를 할당하지 않을 수도 있다.

5. 성능 분석

5.1 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, 작업 부하는 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[9]. 본 시뮬레이터에서는 시스템의 일반적인 동작 환경을 수용할 수 있는 전통적인 작업부하 모델을 채택하였다. 도착되는 시간 간격에 의해 결정되는 새로운 태스크는 요청하는 서브메쉬의 크기, 요청하는 체재시간, 그리고 통신 프랙션에 의해 특징지어진다. 요청하는 서브메쉬의 각 변의 길이는 수행될 작업의 성격을 모르는 상황을 반영할 수 있는 균일 분포를 사용해서 발생시킨다. 체재 시간은 요청된 크기의 서브메쉬를 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.

시스템 부하는 1을 초과하지 않도록 패러미터를 선택하여 다음과 같이 정의하였다 [6]:

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

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 [19]:

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

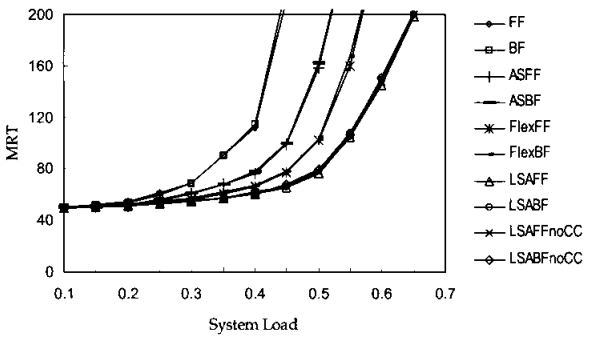
여기서, wh는 전체 메쉬 시스템의 크기를 의미한다. 성능 분석을 하기 위해 평균 작업 응답 시간 (MRT : Mean Response Time), 1000 개의 태스크들을 완료하는 시간, 그리고 시스템 활용도를 측정하였다.

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), LSAFF(LSA with First Fit), LSABF(LSA with Best-fit), LSAFFnoCC(LSA with First Fit and without the Conservative Check), 그리고 LSABFnoCC (LSA with Best Fit and without the Conservative Check).

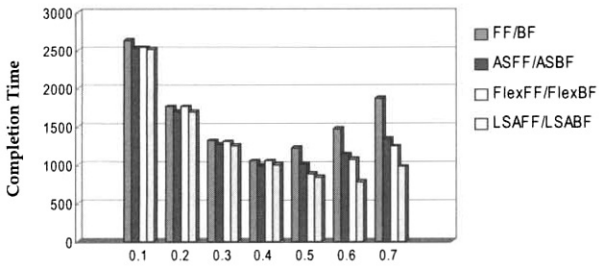
5.2 시뮬레이션 결과

(그림 7)은 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 [2,6,19] 에서와 같이, FF 기법과 BF 기법을 적용하였을 때의 MRT의 차이는 매우 작음을 (그림 7)에서도 알 수 있다. LSA 기법을 적용하였을 경우의 MRT가 FF, BF, ASFF, ASBF, FlexFF, 그리고 FlexBF의 경우보다 작음을 (그림 7)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 8)은 (그림 7)과 같은 환경에서 각 할당 기법들이 1000 개의 작업을 완성시키는 시간을 측정한 것이다. 시스템 부하가 0.4 이하에서는 FF, ASFF, FlexFF 기법들과 거의 비슷한 완성 시간을 보이지만, 시스템 부하가 0.5 이상으로 증가할 경우에는 LSAFF 기법의 완성시간이 다른 기법들에 비해서 상대적으로 작다. 시스템 부하 0.6에서는 LSAFF 기법의 완성시간은 FlexFF 기법에 비해서 28%, ASFF 기법에 비해서 32%, FF 기법에 비해서 47% 정도 각각 감소하였다.

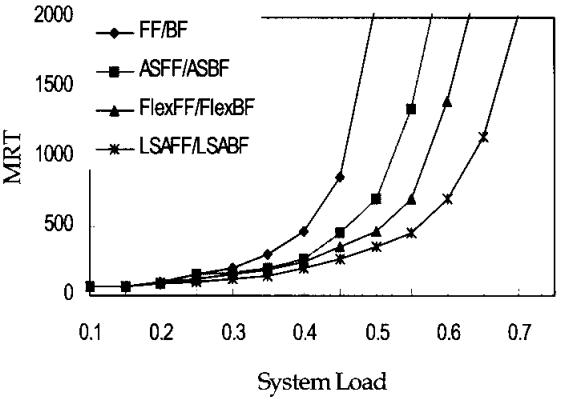


(그림 7) 시스템 부하의 변화에 따른 MRT의 개선도(평균 체재 시간=50, 작업이 도착되는 간격을 변화시킴)



(그림 8) 시스템 부하의 변화에 따른 Completion time의 비교 (평균 체재 시간=50, 작업이 도착되는 간격을 변화시킴)

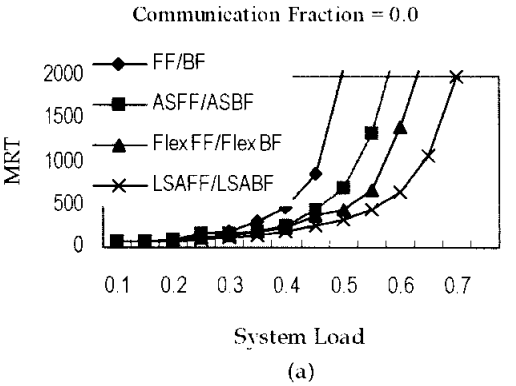
(그림 9)는 작업이 도착되는 간격을 평균 60을 갖는 지수 분포로 발생시키고, 시스템 부하를 각 작업의 요청된 체재 시간을 변화시킴으로써 조절하였을 때, MRT의 변화를 측정한 것이다. 나머지 조건들은 (그림 7)에서와 같이 유지하였다. 시스템 부하가 0.6일 때, FlexFF 기법과 비교하면 MRT가 37% 정도 감소하였으며, LSAFF 기법을 적용하였을 경우의 MRT가 FF, ASFF, 그리고 FlexFF 기법을 적용하였을 경우보다 작음을 볼 수 있다



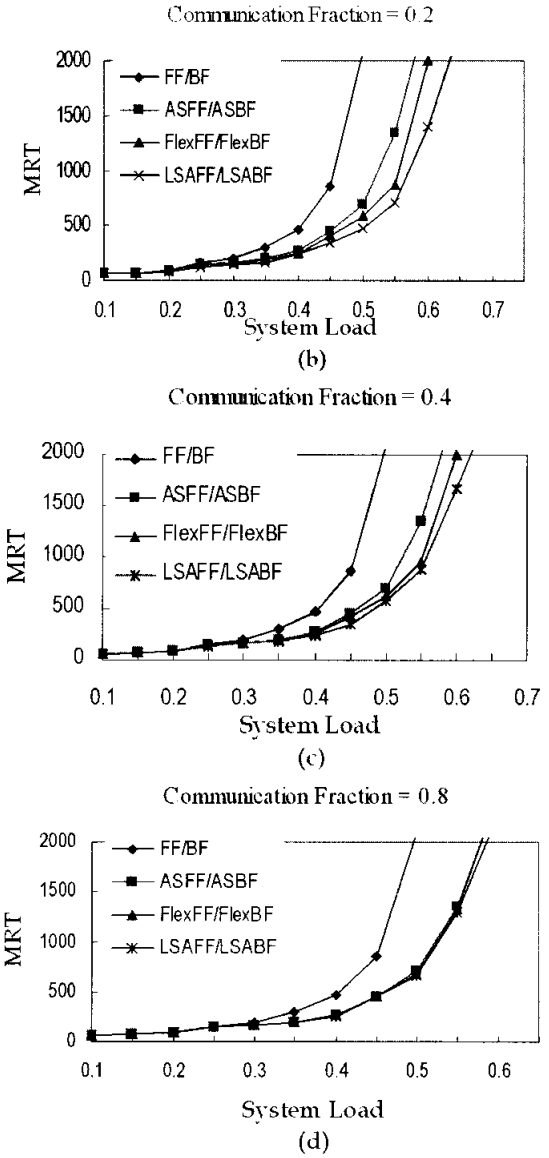
(그림 9) 시스템 부하의 변화에 따른 MRT의 개선도(평균 도착되는 시간 간격 =60, 체재시간을 변화시킴)

Flexfold와 LSA 기법에서 통신 프랙션이 미치는 영향을 (그림 10)에서 보였다. FF/BF와 AS 기법은 이 패러미터에 전혀 영향이 없지만, 비교하기 위해 그래프에 포함시켰다. (그림 9)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 컨서버티브 체크에서 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 때, LSA 기법은 비교적 효율적이라고 볼 수 있다.

LSA 기법을 적용하였을 때의 시스템의 활용도는 각 작업의 요청된 체재시간은 60 단위시간을 평균으로 하는 지수 분포를 갖도록 하고, 발생되는 각 작업의 크기에서 각 변의



(a)



(그림 10) 통신 프랙션 값의 변동에 따른 영향을 보여주는 시스템 부하의 변화에 따른 MRT의 개선도(평균 도착되는 시간 간격 = 60, 체재시간을 변화시킴)

길이를 2에서 27로 제한하여 균일분포로 독립적으로 발생시켰으며, 시스템 부하를 작업이 도착되는 간격을 변화시킴으로써 조절하였을 때, 1000개의 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 때는 대기 큐에 쌓이는 작업들이 많아지므로, 요청된 서브메쉬의 모양을 L-shaping하여 LSM을 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 MRT, 완성시간, 그리고 시스템의 활용도에서 LSA 기법의 성능이 기존의 다른 기법들보다 향상된다.

6. 결 론

메쉬 시스템에서 연속적인 프로세서 할당기법의 단편화 문제를 해결하기 위해, 하나의 작업에 할당되는 프로세서들이 형성하는 모양이 사각형이어야 하는 제약조건을 완화시킴으로써, 서브메쉬의 모양을 L자 모양으로 변형하여 효율적으로 할당할 수 있는 프로세서 할당기법을 제안하였다. LSSA 기법에서 할당 가능한 L-모양 서브메쉬를 탐색하는 과정을 효율적으로 개선하는 알고리즘을 제공하였으며, 사각형 서브메쉬에서 L-모양 서브메쉬로의 변형을 형식화하여 임베딩 비용을 그래프 임베딩 이론에 적용하여 산출하였다. 제안하는 기법은 기존의 사각형 서브메쉬를 할당할 수 없을 때, L-shaping의 임베딩 비용을 계산하여 보존 검사를 한 후에 L-모양 서브메쉬를 효율적으로 탐색하므로, 큰 크기의 작업을 포함해서 시스템으로 들어오는 작업들을 기존의 다른 기법들보다 빠르고 공정하게 수용할 수 있다. 그러므로 LSA 기법은 평균 작업 응답 시간을 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.

[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] C. Chang and Prasant Mohapatra, An Integrated Processor Management Scheme for the Mesh-Connected Multicomputer Systems, In *Proc. Int'l Conf. on Parallel Processing*, pp.118-121, 1997.

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

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] X. Shen and Q. Hu, On Embedding Between 2D Meshes of the Same Size, IEEE Trans. on Computers, Vol. 46, No.8, pp.880-889, Aug. 1997.

[17] K. Windisch, V. Lo, and B. Bose, Contiguous and Non-Contiguous Processor Allocation Algorithms For K-ary and N-cubes, In *Proc. Int'l Conf. on Parallel Processing*, vol. II, pp.164-168, 1995.

[18] 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.

[19] 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 졸업(이학사)
1989년 서강대학교 전자계산학과 졸업(공학사)
1992년 서강대학교 대학원 컴퓨터학과(공학석사)
1998년 서강대학교 대학원 컴퓨터학과(공학박사)
1999년~2003년 성신여자대학교 컴퓨터정보학부 계약교원
2003년~현재 성신여자대학교 컴퓨터정보학부 초빙교원
관심분야 : 병렬처리시스템, 광통신, 임베디드 시스템

김 성 천

e-mail : ksc@arqlab1.sogang.ac.kr
1975년 서울대학교 공과대학 공업교육학 졸업(학사)
1979년 Wayne State Univ. 컴퓨터공학(공학석사)
1982년 Wayne State Univ. 컴퓨터공학(공학박사)
1982년~1984년 캘리포니아 주립대 조교수
1984년~1985년 금성반도체 책임 연구원
1985년~현재 서강대학교 컴퓨터학과 교수
관심분야 : 병렬처리 시스템, 인터커넥션 네트워크