메쉬 구조 다중컴퓨터 시스템을 위한 효율적인 서브메쉬 할당방법

이원주*, 전창호
두원공과대학 인터넷프로그래밍과*
한양대학교 전자컴퓨터공학부

An Efficient Submesh Allocation Scheme for Mesh-Connected Multicomputer Systems

Wonjoo Lee*, Changho Jeon
Department of Internet Programming, Doowon Technical College *

Abstract

본 논문에서는 메쉬 구조 다중컴퓨터 시스템의 성능을 향상시킬 수 있는 새로운 서브메쉬 할당방법을 제안한다. 이 할당방법의 특징은 가용 서브메쉬의 탐색시간과 외적단편화로 인한 서브메쉬의 할당지연을 최소화함으로써 태스크의 대기시간을 줄이는 것이다. 이 할당방법은 태스크의 유형과 동일한 유형별 가용 서브메쉬 리스트에서 최적의 서브메쉬를 찾아 할당함으로써 서브메쉬 탐색시간을 줄인다. 또한 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크를 다른 가용 서브메쉬에 재배치하고, 프로세서 단편을 통합하여 할당함으로써 서브메쉬의 할당지연을 최소화한다. 시뮬레이션을 통하여 서브메쉬 탐색시간을 줄이는 방법보다 외적단편화로 인한 서브메쉬의 할당지연을 줄이는 방법이 태스크의 대기시간을 단축하는데 더 효과적임을 보인다.

I. 서론

메쉬 구조는 단순하고, 규칙적이며 확장성이 뛰어나기 때문에 상업용 또는 프로토타입의 다중컴퓨터 시스템에 널리 사용 되고 있다. 다중컴퓨터 시스템의 성능은 대기중인 태스크에 할당 가능한 가용 서브메쉬의 탐색시간과 외적단편화로 인한 서브메쉬의 할당지연에 따라 달라진다.

기존의 서브메쉬 할당방법들도 가용 서브메쉬의 탐색시간과 외적단편화를 최소화하여 시스템의 성능을 향상시키는 연구를 진행해 왔으며, 그 결과 여러 서브메쉬 할당방법들이 제안되었다[1-9]. 하지만 2차원 메쉬의 구조적인 한계로 인한 외적단편화는 해결하지 못하였다. 2차원 메쉬 구조에서는 할당 서브메쉬를 중심으로 상하, 좌우로 양분된 프로세서 단편들을 연결하여 더 큰 가용 서브메쉬를 형성할 수 없기 때문에 외적단편화로 인한 서브메쉬의 할당지연이 발생한다. 이러한 할당지연은 태스크 대기시간을 증가시키기 때문에 시스템의 성능을 저하시키는 요인이다.

본 논문에서는 가용 서브메쉬의 탐색시간과 외적 단편화로 인한 서브메쉬의 할당지연을 최소화함으로써 태스크 대기시간을 줄일 수 있는 새로운 서브메쉬 할당방법을 제안한다.

본 논문의 2장에서는 2차원 메쉬 구조에 대하여 설명한다. 3장에서는 메쉬 구조를 위한 새로운 서브메쉬 할당방법을 제안한다. 4장에서는 시뮬레이션을 통하여 본 논문에서 제안한 서브메쉬 할당방법이 기존의 할당방법에 비해 우수함을 보인다. 그리고 5장에서 결론을 맺는다.

II. 2차원 메쉬 구조

2차원 메쉬 M(W, H)는 너비와 높이가 W와 H인 사각형 격자 구조이다. M(W, H)는 W· H개의 노드로 구성되며 각 노드는 하나의 프로세서를 나타낸다. 각 노드의 주소는 <x, y> 형식으로 나타낸다. 노드 <x, y>는 좌측하단의 기본 노드 <0, 0>를 기준으로 너비는 x 만큼, 높이는 y만큼 떨어진 좌표에 위치한 노드를 가리킨다. 따라서 x, y 값은 0 x W-1 와 0 y H-1 조건을 만족한다. 태스크를 할당 받아 실행중인 노드를 할당 노드라 하며 유휴 상태의 노드를 가용 노드라 한다.

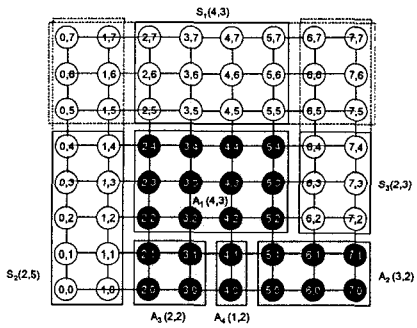

그림 1. 2차원 메쉬 구조 M(8, 8).

그림 1에서 할당 노드는 검정색으로, 가용 노드는 흰색으로 표현되어 있다. M(W, H)내의 서브메쉬는 크기 또는 주소를 이용하여 표현할 수 있다. 사각형 격자 구조로 w·h 개의 노드로 구성한 서브메쉬는 크기를 이용하여 S(w, h)로 표현한다. w 와 h 는 너비와 높이를 의미 하며 1 w W와 1 h H 조건을 만족한다. 또한 좌측하단과 우측상단의 노드 주소 $<x_1, y_1>$와 $<x_2, y_2>$를 갖는 서브메쉬는 S($<x_1, y_1>$, $<x_2, y_2>$)로 표현한다. 좌측하단과 우측상단 노드의 주소는 서브메쉬내 노드의 좌표값 $<x_i, y_i>$을 이용하여 $<x_1, y_1>=<\min(x_i), \min(y_i)>$와 $<x_2, y_2>=<\max(x_i), \max(y_i)>$로 구한다. 할당 서브메쉬는 할당 노드로 구성한 서브메쉬이고 가용 서브메쉬는 가용 노드로 구성한 서브메쉬이다. 가용 서브메쉬와 할당 서브 메쉬는 너비와 높이를 이용하여 각각 $S_i(w, h)$와 $A_j(w, h)$로 표현 하거나 서브메쉬의 좌측하단과 우측 상단의 노드 주소를 이용하여 각각 $S_i(<x_1, y_1>, <x_2, y_2>)$ 와 $A_j(<x_1, y_1>, <x_2, y_2>)$로 표현한다.

III. 제안한 서브메쉬 할당방법

본 논문에서 제안하는 서브메쉬 할당방법은 CFSL-TR(Classified Free submesh List-Task Relocation) 할당방법이라 하며, CFSL 할당단계와 TR 할당단계로 구성한다. CFSL 할당단계에서는 태스크와 동일한 CFSL에서 최적의 가용 서브메쉬를 찾음으로써 가용 서브메쉬의 탐색시간을 단축한다. 그리고 TR 할당단계에서는 외적단편화로 인해 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크를 재배치하고 프로세서 단편들을 통합하여 할당함으로써 태스크 대기시간을 단축한다.

3.1 용어 정의

CFSL-TR 할당방법에서 사용하는 기본적인 용어를 먼저 정의한다.

정의 **1.** 독립 가용 서브 메쉬(IFS: Independent Free Submesh)는 다른 가용 서브메쉬와 중첩되지 않게 형성 할 수 있는 최대 크기의 가용 서브메쉬이다.

그림 1 에서 실선으로 구분한 S_1(<2, 5>, <5, 7>), S_2(<0, 0>, <1, 4>), S_3(<6, 2>, <7, 4>)는 IFS 이다. 서브메쉬 S(<0, 0>, <1, 7>), S(<0, 5>, <7, 7>), S(<6, 2>, <7, 7>)은 점선으로 구분한 서브메쉬 S(<0, 5>, <1, 7>), S(<6, 5>, <7, 7>)에서 서로 중첩된다. 이때 서로 중첩되는 서브메쉬 S(<0, 5>, <1, 7>)과 S(<6, 5>, <7, 7>)을 제외하고 S_1(<2, 5>, <5, 7>), S_2(<0, 0>, <1, 4>), S_3(<6, 2>, <7, 4>)를 IFS 로 지정한다

정의 **2.** 유형별 가용 서브메쉬 리스트(CFSL: Classified Free Submesh List)는 IFS 의 너비와 높이에 따라 정방형, 가로 직사각형, 세로 직사각형으로 분류한 IFS 들의 집합이다.

본 논문에서는 정방형, 가로 직사각형, 세로 직사각형 IFS 의집합을 각각 SQ-flist, HR-flist, VR-flist 로 표현한다. 또한 할당 서브메쉬의 집합은 Alloc-list 로 표현한다. CFSL 내의 IFS 는 크기에 따라 내림차순으로 정렬되며, SQ-flist HR-flist VR-flist ≈ ∅ 조건을 만족한다. 그림 1 에서 S_1(4,3)은 가로 직사각형으로 분류되어 HR-flist 에 삽입된다. 그리고 S_2(2, 5)와 S_3(2, 3)는 세로 직사각형으로 분류되어 VR-flist 에 삽입된다. 또한 Alloc-list 는 할당 서브메쉬 A_1(<2, 2>, <5, 4>), A_2(<5, 0>, <7, 1>), A_3(<2, 0>, <3, 1>), A_4(<4, 0>, <4, 1>)를 포함한다.

주어진 태스크는 T(w, h)로 표현한다. w 와 h 는 각각 태스크의 너비와 높이를 의미한다. T(w, h)를 처리하기 위해서는 최소 w · h 개의 노드를 가진 가용 서브메쉬가 필요하다. 본 논문에서는 T(w, h)를 w 와 h 에 따라 정방형, 가로 직사각형, 세로 직사각형으로 분류하여 서브메쉬 탐색과정에서 활용한다. 태스크에 최적인 서브메쉬를 찾기 위해 태스크 유형과 동일한 CFSL 을 먼저 탐색한다. 즉, 태스크의 유형이 가로 직사각형일 경우, HR-flist 에서 먼저 최적의 서브메쉬를 찾는다. 전체 서브메쉬를 대상으로 최적의 서브메쉬를 찾는 것보다 태스크의 유형과 동일한 CFSL 에서 먼저 찾음으로써 서브메쉬의 탐색시간을 줄인다.

정의 **3.** IFS 의 확장지수(EI: Expansion Index)는 다른 가용 서브메쉬와 중첩을 허용하면서 최대 크기로 확장할 수 있는 범위값이다.

확장지수는 전체 메쉬 구조를 탐색하여 IFS 를 형성하는 과정에서 구하며, 각 IFS 의 속성으로 저장된다. IFS 인 S_i(<x_1, y_1>, <x_2, y_2>)의 확장지수는 EI(S_i)=(<α, β>, < α', β' >) 형식으로 표현한다. IFS 인 S_i(<x_1, y_1>, <x_2, y_2>)를 다른 가용 서브메쉬와 중첩하여 형성한 최대 크기의 가용 서브메쉬는 S_i'(<x'_1, y'_1>, <x'_2, y'_2>) 이다. EI(S_i)=(<α, β>, < α', β' >)는 S_i(<x_1, y_1>, <x_2, y_2>)와 S_i'(<x'_1, y'_1>, <x'_2, y'_2>)의 좌측하단 노드와 우측상단 노드의 좌표값으로 구한다. 즉, <α, β>는 각각 <$x'_1 - x_1$, $y'_1 - y_1$> 이며 <α', β'>는 각각 <$x'_2 - x_2$, $y'_2 - y_2$>로 구한다.

예를 들어 그림 1 에서 IFS 인 S_1(<2, 5>, <5, 7>)은 다른 가용 서브메쉬와 중첩하는 서브메쉬 S(<0, 5>, <1, 7>)과 S(<6, 5>, <7, 7>)를 포함하여 S_1'(<0, 5>, <7, 7>)를 형성할 수 있다. 변위값은 S_1(<2, 5>, <5, 7>)과 S_1'(<0, 5>, <7, 7>)의 좌측하단 노드와 우측상단 노드의 좌표값을 이용하여 <α, β> = <0-2, 5-5>와 <α', β' > = <7-5, 7-7>로 구한다. 따라서 EI[S_1(<2, 5>, <5, 7>)] = (<-2, 0>, <2, 0>) 이다. 그림 1 에서 태스크 T(6, 3)이 입력되면 S_1(<2, 5>, <5, 7>)과 S_2(<0, 0>, <1, 4>), S_3(<6, 2>, <7, 4>) 모두 할당 할 수 없다. 그러나 S_1(<2, 5>, <5, 7>)을 점선으로 표현된 S(<0, 5>, <1, 7>)과 S(<6, 5>, <7, 7>)로 확장하면 할당이 가능하다. 이 때 확장지수 EI(S_1)=(<-2, 0>, <2, 0>)를 사용하여 S_1(<2, 5>, <5, 7>)을 S_1'(<0, 5>, <7, 7>)로 확장하여 할당한다.

그림 1 에서는 할당 서브메쉬에 의해 좌우로 양분된 S_3(<6, 2>, <7, 4>)와 S_2(<0, 0>, <1, 4>)를 연결하여 더 큰 가용 서브메쉬를 형성할 수 없다. 이때 태스크 T(4, 6)이 주어진다면 할당 가능한 서브메쉬를 찾을 수 없기 때문에 T(4, 6)에 대한 서브메쉬의 할당은 지연된다. 이러한 서브메쉬의 할당지연을 줄이기 위해 본 논문에서는 할당 서브메쉬에서 수행중인 태스크를 재배치하고 프로세서 단편들을 통합한다. 태스크 재배치 과정에서는 먼저 그림 2 와 같이 A_i(<x_1, y_1>, <x_2, y_2>)와 기본 노드 <0, 0> 사이의 영역을 3 개의 영역으로 분할한 후, 각 영역별로 다른 할당 서브메쉬가 존재하는지 탐색한다.

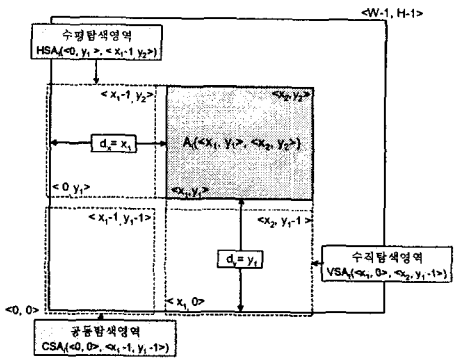

그림 2. 할당 서브메쉬의 탐색영역.

정의 **4.** 탐색영역(SA: Search Area)은 기본 노드 <0, 0>와 A_i(<x_1, y_1>, <x_2, y_2>) 사이에 위치한 서브메쉬이다.

본 논문에서는 탐색영역을 공동탐색영역(Common Search Area), 수평탐색영역(Horizontal Search Area), 수직탐색영역(Vertical Search Area)로 나누며, 각각 CSA_i(<0, 0>, <x_1-1, y_1-1>), HSA_i(<0, y_1>, <x_1-1, y_2>), VSA_i (<x_1, 0>, <x_2, y_1-1>)로 나타낸다. 각 탐색영역에 다른 할당 서브메쉬가 존재하지 않으면 A_i(<x_1, y_1>, <x_2, y_2>)에서 실행중인 태스크는 x 축의 좌측방향과 y 축의 하단 방향으로 각각 x_1과 y_1 만큼 이동하여 S<0, 0>, <x_2-x_1, y_2-y_1>)에 재배치된다. 그러나 다른 할당 서브메쉬들이 존재하면 각 탐색영역내의 할당 노드들을 탐색하여 할당범위를 생성한다. 할당범위는 다음과 같이 정의한다.

정의 **5.** 할당범위(AD: Allocation Domain)는 각 탐색영역내에 위치한 할당 노드들로 구성한 서브메쉬이다.

할당범위는 공동할당범위 (Common Allocation Domain), 수평할당범위 (Horizontal Allocation Domain), 수직할당범위(Vertical Allocation Domain)로 구분하며 각각 CAD_i (<x_{c1}, y_{c1}>, <x_{c2}, y_{c2}>), HAD_i(<x_{h1}, y_{h1}>, <x_{h2}, y_{h2}>), VAD_i(<x_{v1}, y_{v1}>, <x_{v2}, y_{v2}>)로 나타낸다. 할당범위는 이미 다른 태스크를 실행중인 할당 노드들로 구성되기 때문에 태스크를 재배치 할 수 없는 영역이다. 만약 HSA_i(<0, y_1>, <x_1-1, y_2>)에 하나 이상의 할당 서브메쉬가 존재한다면 HSA_i(<0, y_1>, <x_1-1, y_2>)내의 각 할당 노드들을 탐색하여 HAD_i(<x_{h1}, y_{h1}>, <x_{h2}, y_{h2}>)=(<min(x_i), min(y_i)>, <max(x_i), max(y_i)>)를 생성한다. 이때 x_i 와 y_i 는 각각 $0 \le x_i \le x_1-1$, $y_1 \le y_i \le y_2$ 조건을 만족한다.

정의 **6.** 할당 서브메쉬 A_i(<x_1, y_1>, <x_2, y_2>)에서 수행중인 태스크의 이동거리(D)는 수평이동거리 (horizontal shift distance) d_x와 수직이동 거리(vertical shift distance) d_y의 합이다.

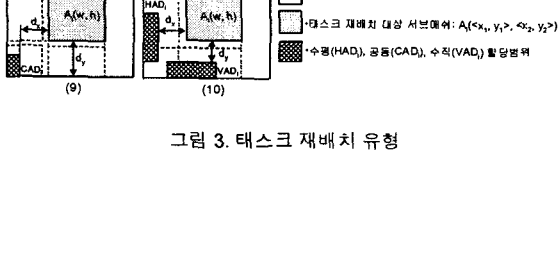

그림 3. 태스크 재배치 유형

그림 3 에서 d_x 와 d_y 는 $A_i(<x_1, y_1>, <x_2, y_2>)$와 할당범위의 위치에 따라 달라진다. d_x 와 d_y 는 $A_i(<x_1, y_1>, <x_2, y_2>)$에서 수행중인 태스크를 각각 x 축과 y 축 방향으로 이동할 수 있는 거리이며, $0 \le d_x \le x_1$, $0 \le d_y \le y_1$ 조건을 만족한다. 그림 3 할당 서브메쉬의 재배치 유형에 따른 d_x 와 d_y 는 표 1 과 같다.

<Table 1> $A_i(<x_1, y_1>, <x_2, y_2>)$의 수평이동거리와 수직이동거리

Type	CSA	HSA	VSA	Condition	d_x	d_y
(1)	F	F	F	-	x_1	y_1
(2)	F	F	T	$(x_1<x_{v1})$ AND $(w \le x_{v1})$	x_1	y_1
(3)	F	F	T	$(x_1<x_{v1})$ AND $(w>x_{v1})$	x_1	$y_1 - y_{v2}$
(4)	F	T	F	$(y_1<y_{h1})$ AND $(h \le y_{h1})$	x_1	y_1
(5)	F	T	F	$(y_1<y_{h1})$ AND $(h>y_{h1})$	$x_1 - x_{h2}$	y_1
(6)	T	F	T	$(x_1<x_{v1})$ AND $(w>x_{v1})$	x_1	$y_1 - y_{v2}$
(7)		T	F	$(y_1<y'_{h1})$ AND $(h>y'_{h1})$	$x_1 - x_{h2}$	Y_1
(8)		F	F	$(x_1>x_{c2})$ AND $(y_1>y_{c2})$ AND $(w_c>h_c)$	x_1	$y_1 - y_{c2}$
(9)		F	F	$(x_1>x_{c2})$ AND $(y_1>y_{c2})$ AND $(w_c<h_c)$	$x_1 - x_{c2}$	y_1
(10)		T	T	$((x_1>x_{h2})$ AND $(y_1 \le y_{h2}))$ AND $((x_1 \le x_{v2})$ AND $(y_1>y_{v2}))$	$x_1 - x_{h2}$	$y_1 - y_{v2}$

표 1 에서 CSA, HSA, VSA 는 각 탐색영역에서 각각의 할당범위를 생성하면 T(true) 이고 그 반대의 경우에는 F(false)이다. 표 1 에 따라 d_x 와 d_y 가 결정되면 $A_i(<x_1, y_1>, <x_2, y_2>)$에서 실행중인 태스크는 다른 위치의 $S(<x_1-d_x, y_1-d_y>, <x_2-d_x, y_2-d_y>)$로 재배치된다. 이때 할당 서브메쉬의 모든 노드들은 교착 상태(deadlock)를 방지하기 위해 이동거리 $(D=d_x+d_y)$만큼 같은 방향 으로 함께 이동한다. 본 논문에서는 태스크를 재배치하는데 소요되는 시간을 다음과 같이 정의한다.

정의 **7.** 태스크 $T_i(w, h)$의 재배치 시간(RT_i: Relocation Time)은 할당 서브메쉬 $A_i(<x_1, y_1>, <x_2, y_2>)$에서 수행중인 태스크를 가용 서브메쉬 $S(<x_1-d_x, y_1-d_y>, <x_2-d_x, y_2-d_y>)$로 이동하는데 소요되는 시간이다.

본 논문에서는 $T_i(w, h)$의 재배치 시간을 식(1)과 같이 구한다.

$$RT_i = D \cdot \tau \quad \text{----------(1)}$$

식(1)에서 D 는 태스크의 이동거리이고 τ 는 태스크 단위이동시간으로 태스크를 이동거리 1 만큼 이동하는데 소요되는 시간이다. 태스크 재배치시간은 태스크 대기시간에 합산되어 할당방법의 성능을 평가할 때 고려한다.

3.2 서브메쉬 할당 알고리즘

CFSL-TR 할당방법의 서브메쉬 할당 알고리즘은 그림 4와 같다.

```
CFSL-TR_Allocation() //Request(Ti<w, h>)
{
  Ti = Dispatch_Qi(Ti<w, h>); //대기 큐에서 태스크 dispatch
  Make_CFSL(); //메쉬 구조를 탐색하여 CFSL 생성
  Select_Submesh() { //유형별 할당 과정 선택
    •태스크 유형에 따라 CFSL 에서 최적의 가용 서브메쉬 선택
  }
  if(최적의 가용 서브메쉬 = ∅) {
     if (External fragmentation){ // 외적단편화
        • Submesh_Compaction(); //프로세서 단편 통합
        • Goto Make_CFSL( );
     }else{
        Waiting();//할당 가능한 가용 서브메쉬가 생성될 때까지 대기
     }
  }
  Do_Allocate () {  //서브메쉬 할당 과정
    Allocate() { •최적의 가용 서브메쉬를 할당하고 Alloc-list에 삽입 }
    Delete(){•가용 서브메쉬를 CFSL 에서 삭제}
  }
}
Submesh_Compaction () {
  for(i=0; i <= N; i++){  // N: The number of Allocated submesh
    HSA_ CSA_VSA_Scan( );
    if( (dx != 0) or (dy !=0) ){ Task_ Relocation(dx, dy); }
  }
}
HSA_CSA_VSA_Scan() {
  • 태스크 재배치 유형에 따라 수평(dx)/수직(dy) 이동거리 반환
}
Task_ Relocation( dx, dy ) {
  // Ai(<x1, y1>, <x2, y2>)내의 태스크 재배치
  • <x1, y1> = <x1 - dx, y1 - dy > // 좌측하단 노드(<x1, y1>)
  • <x2, y2> = <x2 - dx, y2 - dy > // 우측상단 노드(<x2, y2>)
}
```

그림 4. 서브메쉬 할당알고리즘

Make_CFSL() 모듈에서는 전체 메쉬 구조에서 탐색한 IFS를 유형에 따라 분류하여 CFSL을 생성한다. 그리고 Select_Submesh() 모듈에서는 태스크의 유형과 동일한 CFSL에서 최적의 가용 서브메쉬를 찾는다. 최적의 가용 서브메쉬가 존재하면 Do_Allocate() 모듈에서는 그 가용 서브메쉬를 태스크에 할당하고 CFSL에서 삭제한다. 그러나 최적의 가용 서브메쉬를 찾지 못하면 외적단편화 여부를 판별한다. 만약 외적단편화로 인해 할당지연이 발생한 경우에는 Submesh_Compaction() 모듈에서 수행중인 태스크를 재배치하고, 프로세서 단편들을 통합하여 할당한다. 하지만 외적단편화가 아닌 경우에는 할당 서브메쉬들이 할당 해제되어 더 큰 가용 서브메쉬를 형성할 때까지 대기한다.

T(4, 6)이 주어진 경우 그림 1 에서는 외적단편화로 인해 할당 가능한 서브메쉬를 찾을 수 없기 때문에 할당지연이 발생한다. 이때 Submesh_Compaction() 모듈에서 프로세서 단편을 통합한다.
그림 1 의 $A_3(<2, 0>, <3, 1>)$는 그림 3 의 유형 (1)에 해당하기 때문에 수평 이동거리 $d_x = 2$ 와 수직이동거리 $d_y=0$ 만큼 이동하여 $S(<0, 0>, <1, 1>)$에 재배치된다. $A_1(<2, 2>, <5, 4>)$, $A_2(<5, 0>, <7, 1>)$, $A_4(<4, 0>, <4, 1>)$도 $A_3(<2, 0>, <3, 1>)$와 같이 재배치 유형에 따라 재배치하여 프로세서 단편을 통합하면 그림 5 와 같다.

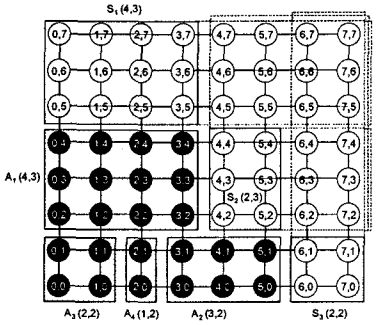

그림 5. 프로세서 단편 통합 결과.

그림 5 에서는 $S_2(<4, 2>, <5, 4>)$를 최적의 가용 서브메쉬로 선택한다. 그리고 확장지수를 사용하여 $S_2'(<4, 2>, <7, 7>)$로 확장하면 T(4, 6)에 할당하면 T(4, 6)의 대기시간을 줄일 수 있다.

3.3 서브메쉬 할당해제 알고리즘

할당 서브메쉬에서 수행중인 태스크를 완료하면 할당 서브메쉬를 가용 서브메쉬로 환원하는 할당해제 알고리즘은 그림 6 과 같다.

```
CFSL-TR_Deallocation ( )  //Deallocation(A<w, h>)
{
  if( Alloc-list = ∅ ){  // Alloc-list: 할당 서브메쉬 리스트
     SQ-flist={ M(W, H)}; HR-flist = VR-flist = ∅;
  } else {
    if (A(w, h)  == IFS){
        Insert_CFSL() {• IFS 유형에 해당하는 CFSL 삽입}
    } else{
        Deallocation() {• 서브메쉬 할당해제}
        Make_CFSL() {• 가용_서브메쉬를 탐색하여 CFSL 재생성}
    } //end if (A(w, h)  == IFS )
  } //end if( Alloc-list = ∅ )
} //end
```

그림 6. 서브메쉬 할당해제 알고리즘

그림 6 의 할당 해제 알고리즘에서는 할당 해제할 서브메쉬가 IFS 인지 아닌지를 먼저 판별한다. IFS 가 아니면 Deallocation() 모듈에서 할당 서브메쉬를 할당 해제한다. 그리고 Make_CFSL()모듈에서는 전체 메쉬 구조를 탐색하여 CFSL 을 재생성 한다. 그러나 IFS 이면 Insert_CFSL() 모듈에서 서브메쉬의 유형에 해당하는 CFSL 에 삽입함으로써 할당 서브메쉬를 해제한다.

IV. 시뮬레이션

본 논문에서는 시뮬레이션을 통하여 제안한 CFSL-TR 할당방법이 기존의 FSL 할당방법[9]에 비해 태스크의 대기시간을 줄이는 면에서 우수함을 보인다.

4.1 시뮬레이션 결과 분석

시뮬레이션에서는 지수분포(Exp), 균일분포(Uni), 정규분포(Nor)에서 각각 300,000 개의 태스크를 생성하여 할당한다. 그리고 메쉬 크기(16x16~512x512) 에서 다양한 시스템 부하($0<\rho\leq 1$)와 태스크 단위이동시간($0.01\leq\tau\leq 0.05$)에 따른 평균탐색시간과 평균대기시간을 측정한다.

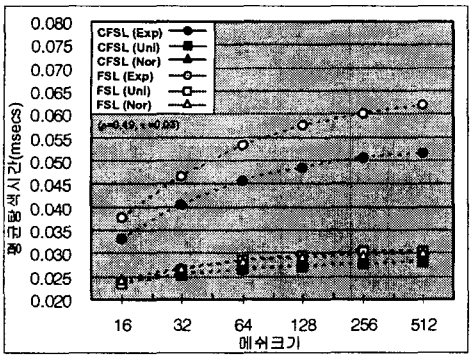

그림 7. 태스크 재배치를 적용하지 않은 경우, 메쉬크기에 따른 평균탐색시간 (ρ=0.49, τ=0.03)

그림 7을 살펴보면 CFSL 할당방법은 지수분포(Exp), 균일분포(Uni), 정규분포(Nor)에서 FSL 할당방법에 비해 각각 12.1~16.9%, 4.5~7.2%, 2.1~7.3%의 평균탐색시간을 줄였다. CFSL 할당방법은 전체 메쉬 구조에서 탐색한 가용 서브메쉬를 유형에 따라 분류하여 CFSL을 생성하고, 주어진 태스크의 유형과 동일한 CFSL에서 최적의 서브메쉬를 찾아 할당함으로써 서브메쉬의 탐색시간을 줄인다.

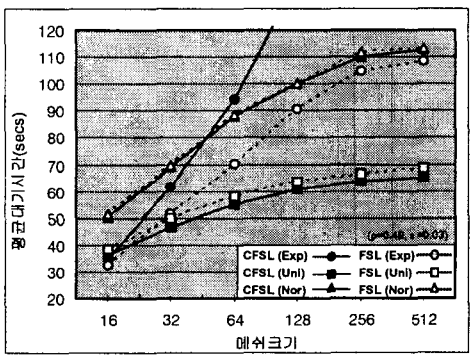

그림 8. 태스크 재배치를 적용하지 않은 경우,메쉬크기에 따른 평균대기시간 (ρ=0.49, τ=0.03)

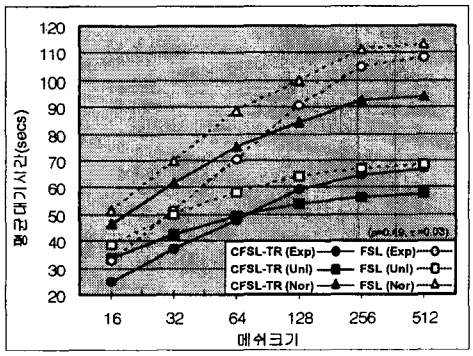

그림 9. 태스크 재배치를 적용한 경우, 메쉬크기에 따른 평균대기시간 (ρ=0.49, τ=0.03).

그림 8을 살펴보면 균일분포(Uni), 정규분포(Nor)에서 CFSL 할당방법의 평균대기시간은 FSL 할당방법에 비해 우수하지만 지수분포(Exp)에서 CFSL 할당방법의 평균대기시간은 저하되었음을 볼 수 있다. 균일분포(Uni)와 정규분포(Nor)에 비해 작은 크기의 태스크가 많은 지수분포(Exp)에서는 할당 서브메쉬에서 분할되는 가용 서브메쉬로 인해 외적단편화가 증가한다. 따라서 외적단편화로 인한 서브메쉬의 할당지연이 증가하면서 태스크 대기시간도 함께 증가한다. CFSL 할당방법은 CFSL을 사용하여 서브메쉬의 탐색시간을 단축함으로써 태스크의 대기시간을 줄일 수 있다. 하지만 외적단편화가 심해지면 서브메쉬의 할당지연이 증가하면서 태스크 대기시간도 함께 증가하는 문제점이 있다.

이러한 문제점을 해결하기 위해 CFSL-TR 할당방법은 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크를 재배치하고, 프로세서 단편들을 통합하여 할당한다. 그림 9를 살펴보면 CFSL-TR 할당방법은 지수분포(Exp), 균일분포(Uni), 정규분포(Nor)에서 FSL 할당방법에 비해 각각 23.2~38.1%, 12.3~15.6%, 9.4~16.9%의 평균대기시간을 줄였다. 따라서 CFSL을 사용하여 가용 서브메쉬의 탐색시간을 줄이는 방법보다 외적단편화로 인한 서브메쉬의 할당지연을 줄이는 CFSL-TR 할당방법이 태스크의 대기시간을 단축하는 효과가 크다는 것을 알 수 있다.

V. 결론

본 논문에서 제안한 CFSL-TR 할당방법은 먼저 태스크와 동일한 CFSL에서 최적의 가용 서브메쉬를 찾음으로써 평균탐색시간을 단축하였다. 하지만 외적단편화로 인한 할당지연이 증가하면 평균대기시간이 오히려 길어지는 문제점이 있었다. 이러한 문제점을 해결하기 위해 CFSL-TR 할당방법은 할당 서브메쉬에서 수행중인 태스크를 재배치하고 프로세서 단편들을 통합하여 할당한다. 그 결과 지수분포(Exp), 균일분포(Uni), 정규분포(Nor)에서 FSL 할당방법에 비해 각각 23.2~38.1%, 12.3~15.6%, 9.4~16.9%의 평균대기시간을 줄였다. 따라서 CFSL-TR 할당방법이 기존의 FSL 할당방법[16]에 비해 태스크 대기시간을 단축하는 면에서 우수함을 알 수 있었다. 또한 태스크의 대기시간을 단축하는 방법으로는 서브메쉬의 탐색시간을 줄이는 방법보다 외적단편화로 인한 서브메쉬의 할당지연을 줄이는 방법이 더 효과적임을 알 수 있었다. 그리고 태스크 재배치과정에서 오버헤드로 발생하는 태스크 재배치시간은 단축한 태스크 대기시간에 비해 매우 작은 값임을 알 수 있었다. 따라서 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 태스크를 재배치하고 프로세서 단편들을 통합하여 할당하는 방법이 시스템의 성능 향상에 유리하다는 것을 알 수 있었다.

참고문헌

[1] K. Li and K. H. Cheng, "A Two-Dimensional Buddy System for Dynamic Resource Allocation in a Partitionable Mesh Connected system", IEEE Journal of Parallel and Distributed Computing, vol. 12, pp. 79-83, May 1991

[2] P.J. Chuang and N.F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems", Proc. Intl Conf. Distributed Computing Systems, pp.256-263, Aug. 1991

[3] J. Ding and L.N. Bhuyan, "An Adaptive Submesh Allocation Strategy for Two-Dimensional Mesh Connected Systems", Proc. Intl Conf. Parallel Processing, pp.II-193-200, Aug. 1993

[4] Y. Zhu, "Efficient Processor Allocation Strategies for Mesh Connected Parallel Computers", IEEE Journal of Parallel and Distributed Computing, vol. 16, pp. 328-337, Dec. 1992

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

[6] S.Bhattacharya, W.-T. Tsai, "Lookahead Processor Allocation in Mesh-Connected Massively Parallel Multocomputer", Proc. Intl Parallel Processing Symp., pp. 868-875, 1994

[7] T.Liu, W.-K. Huang, F. Lombardi, and L.N.Bhuyan, "A Submesh Allocation Scheme for Mesh-Connected Multiprocessor Systems", Proc. Intl Conf. Parallel Processing, pp. II-159-II-163, 1995

[8] S. M. Yoo, H. Y. Youn, Behrooz Shirazi, "An Efficient Task Allocation Scheme for 2D Mesh Architecture", IEEE Trans. on Parallel and Distributed Systems, vol. 8, no 9. pp. 934-942, Sep. 1997

[9]Geunmo Kim, Hyunsoo Yoon, "On Submesh Allocation for Mesh Multicomputers: A Best-Fit Allocation and a Virtual Submesh Allocation for Faulty Meshes", IEEE Trans. on Parallel and Distributed Systems, vol. 9, no 2. pp. 175-185, Feb. 1998