제 19 회 한국정보처리학회 춘계학술발표대회 논문집 제 10 권 제 1 호 (2003. 5)

태스크 재배치를 이용한 프로세서 할당방법

이원주*, 전창호**
*두원공과대학 인터넷프로그래밍과
**한양대학교 전자컴퓨터공학부
e-mail : wonjoo@doowon.ac.kr*, chjeon@cse.hanyang.ac.kr**

A Processor Allocation Scheme Using Task Relocation

Won-Joo Lee*, Chang-Ho Jeon**
* Department of Internet Programming, Doowon Technical College
** School of Electrical and Computer Engineering, Hanyang University

요 약

본 논문에서는 메쉬 구조 다중컴퓨터 시스템을 위한 새로운 서브메쉬 할당방법을 제안한다. 이 할당방법의 특징은 외적단편화로 인한 할당지연을 최소화하여 태스크 대기시간을 단축하는 것이다. 2 차원 메쉬 구조에서는 할당 서브메쉬에 의해 상하, 좌우로 양분되는 프로세서 단편들을 연결하여 더 큰 가용 서브메쉬를 형성할 수 없는 구조적인 한계 때문에 외적단편화로 인한 서브메쉬의 할당지연이 발생한다. 이러한 할당지연은 태스크의 대기시간을 증가시키기 때문에 시스템의 성능을 저하시킨다. 따라서 본 논문에서는 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크들을 다른 가용 서브메쉬에 재배치하고 프로세서 단편들을 통합하여 할당함으로써 태스크의 대기시간을 줄인다. 시뮬레이션을 통하여 제안한 할당방법이 태스크의 대기시간을 줄이는 면에서 기존의 할당방법들 보다 우수함을 보인다.

1. 서론

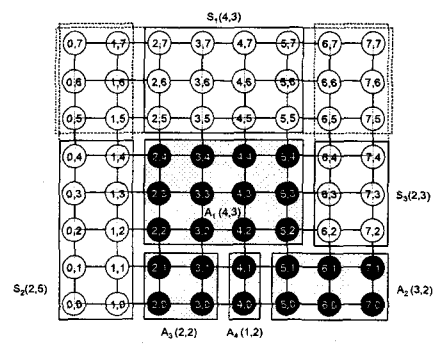
메쉬 구조는 단순하고, 규칙적이며 확장성이 뛰어나기 때문에 상업용 또는 프로토타입의 다중컴퓨터 시스템에 널리 사용되고 있다. 다중컴퓨터 시스템의 성능은 다양한 크기의 서브메쉬를 요청하는 태스크에 최적의 가용 서브메쉬를 찾아 할당하는 서브메쉬 할당방법에 따라 달라진다. 기존의 서브메쉬 할당방법은 할당 서브메쉬를 중심으로 상하, 좌우로 양분된 프로세서 단편들을 연결하여 더 큰 가용 서브메쉬를 형성할 수 없는 구조적인 한계 때문에 발생하는 외적단편화를 해결하지 못하였다. 외적단편화로 인한 서브메쉬의 할당지연은 태스크의 대기시간을 증가시키기 때문에 시스템의 성능을 저하시킨다.

본 논문에서는 외적단편화로 인한 할당지연을 최소화하여 시스템의 성능을 향상시킬 수 있는 새로운 서브메쉬 할당방법을 제안한다. 이 할당방법은 외적단편화로 인해 할당지연이 발생하면 수행중인 태스크를 재배치하고, 프로세서 단편들을 통합하여 할당함으로써 태스크 대기시간을 줄인다. 시뮬레이션을 통하여 제안한 할당방법이 태스크의 대기시간을 줄이는 면에서 기존의 할당방법들 보다 우수함을 보인다.

본 논문의 2장에서는 메쉬 구조에 대하여 설명하고, 3장에서는 기존의 서브메쉬 할당 방법에 대하여 설명한다. 4장에서는 메쉬 구조를 위한 새로운 서브메쉬 할당방법을 제안한다. 5장에서는 시뮬레이션 환경을 설명하고 성능 평가를 통하여 본 논문에서 제안한 서브메쉬 할당방법이 기존의 할당방법에 비해 우수함을 보인다. 그리고 6장에서 결론을 맺는다.

2. 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. 본 논문에서는 각 노드의 주소를 <x, y> 형식으로 표현한다. 노드 <x, y>는 좌측하단의 기본 노드 <0, 0>를 기준으로 너비는 x만큼, 높이는 y만큼 떨어진 좌표에 위치한 노드를 가리킨다. 따라서 x, y값은 0 x W-1와 0 y H-1 조건을 만족한다. 태스크를 할당 받아 실행중인 노드를 할당 노드라 하며 유휴 상태의 노드를 가용 노드라 한다. <그림 1>에서 할당 노드는 검정색으로, 가용 노드는 흰색으로 표현되어 있다. M(W, H)내의 서브메쉬는 크기 또는 주소를 이용하여 표현할 수 있다. 사각형 격자 구조로 w·h개의 노드로 구성한 서브메쉬는 크기를 이용하여 S(w, h)로 표현한다. w와 h는 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 서브메쉬 이다. 가용 서브메쉬와 할당 서브메쉬는 너비와 높이를 이용하여 각각 $S_i(w, h)$와 $A_j(w, h)$로 표현하거나 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.



<그림 1> 2차원 메쉬M(8, 8)

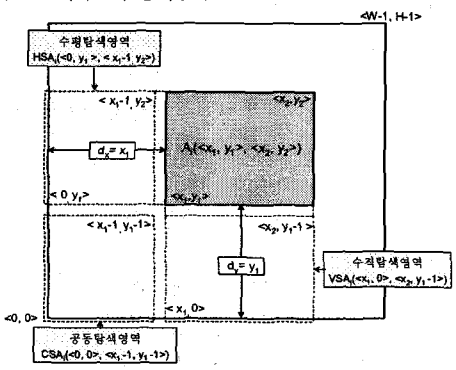
<그림 1>에서 $A_1(<2, 2>, <5, 4>)$, $A_2(<5, 0>, <7, 1>)$, $A_3(<2, 0>, <3, 1>)$, $A_4(<4, 0>, <4, 1>)$는 할당 서브메쉬이고 $S_1(<2, 5>, <5, 7>)$, $S_2(<0, 0>, <1, 4>)$, $S_3(<6, 2>, <7, 4>)$는 독립 가용 서브메쉬[1] 이다.

3. 관련 연구

메쉬 구조를 위한 기존의 서브메쉬 할당방법들은 가용 서브메쉬의 탐색시간과 프로세서 단편화를 줄임으로써 시스템의 성능을 향상 시키는 방향으로 연구를 진행해 왔으며, 그 결과 여러 서브메쉬 할당방법들이 제안되었다[2-10]. 기존의 할당방법들은 서브메쉬를 할당하는 방법에 따라 최초할당과 최적할당으로 분류된다. 최초 할당방법[3, 4]은 태스크에 할당 가능한 서브메쉬를 찾으면 바로 할당하는 방법이다. 이 할당방법은 할당 알고리즘이 단순하고 서브메쉬 탐색시간이 짧지만 최적할당에 비해 외적단편화가 심하다. 최적 할당방법[2, 5-10]은 전체 메쉬 구조에서 탐색한 가용 서브메쉬로 구성한 가용 서브메쉬 리스트에서 최적의 서브메쉬를 찾아 할당하는 방법이다. 이 할당방법은 할당 알고리즘이 복잡하고, 가용 서브메쉬 리스트내의 서브메쉬 수가 증가하면 서브메쉬 탐색시간이 길어지는 문제점이 있다. 그러나 최초할당에 비해 외적단편화가 적고 시스템의 성능 향상에 유리하다. 기존의 최적 할당방법들은 최초 할당방법에 비해 외적단편화를 많이 줄였지만 2차원 메쉬의 구조적인 한계로 인한 외적단편화는 줄이지 못하였다. 이 외적단편화는 할당 서브메쉬를 중심으로 상하, 좌우로 양분된 프로세서 단편들을 연결하여 더 큰 가용 서브메쉬를 형성할 수 없기 때문에 서브메쉬의 할당을 지연시킨다. 즉, <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 외적단편화로 인한 할당지연을 줄이기 위해 본 논문에서는 프로세서 단편들을 통합하여 할당하는 새로운 서브메쉬 할당방법을 제안한다.

4. SATR 할당방법

본 논문에서는 제안하는 서브메쉬 할당방법을 SATR(Submesh Allocation using Task Relocation)할당방법이라 한다. 이 할당방법은 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크를 다른 가용 서브메쉬로 재배치한 후 프로세서 단편들을 통합하여 할당한다. 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 3개의 영역으로 분할한 후, 각 영역별로 다른 할당 서브메쉬가 존재하는지 탐색한다.



<그림 2> 할당 서브메쉬의 탐색영역

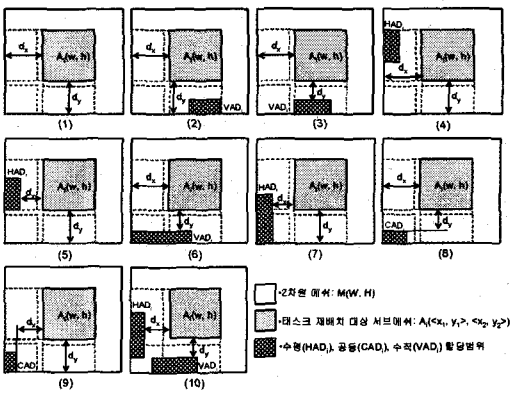
정의 **1**. 할당 서브메쉬 A_i(<x_1, y_1>, <x_2, y_2>)의 공동탐색영역(CSA: Common Scan Area) CSA_i(<0, 0>, <x_1-1, y_1-1>), 수평탐색영역(HSA: Horizontal Scan Area) HSA_i(<0, y_1>, <x_1-1, y_2>), 수직탐색영역(VSA: Vertical Scan Area) VSA_i (<x_1, 0>, <x_2, y_1-1>)는 각각 좌측하단 노드 <0, 0>, <0, y_1>, < x_1, 0>와 우측상단 노드 <x_1-1, y_1-1>, <x_1-1, y_2>, <x_2, y_1-1>로 구성된 서브메쉬이다.

<그림 2>에서 점선으로 표현된 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>)에 다른 할당 서브메쉬가 존재하지 않으면 A_i(<x_1, y_1>, <x_2, y_2>)에 실행중인 태스크는 x축의 좌측방향과 y축의 아래 방향으로 각각 x_1과 y_1만큼 이동하여 S<0, 0>, <x_2-x_1, y_2-y_1>)에 재배치된다. 그러나 다른 할당 서브메쉬들이 존재한다면 3개 영역 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>)에 위치한 할당 노드들로 각 영역의 할당범위를 생성한다. 할당범위는 다음과 같이 정의한다.

정의 **2**. 공동할당범위(CAD: Common Allocation Domain)인 CAD_i(<x_{c1}, y_{c1}>, <x_{c2}, y_{c2}>), 수평할당범위(HAD: Horizontal Allocation Domain)인 HAD_i(<x_{h1}, y_{h1}>, <x_{h2}, y_{h2}>), 수직할당범위(VAD: Vertical Allocation Domain)인 VAD_i(<x_{v1}, y_{v1}>, <x_{v2}, y_{v2}>)는 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>)에 위치한 할당 노드들로 구성한 서브메쉬이다.
할당범위는 이미 다른 할당 서브메쉬에 포함된 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_i와 y_i는 각각 $0 \le x_i \le x_1-1$, $y_1 \le y_i \le y_2$ 조건을 만족한다. 그리고 CSA_i(<0, 0>, <x_1-1, y_1-1>)와 VSA_i(<x_1, 0>, <x_2, y_1-1>)에서도 하나 이상의 할당 서브메쉬가 존재한다면 각각 CAD_i(<x_{c1}, y_{c1}>, <x_{c2}, y_{c2}>)와 VAD_i(<x_{v1}, y_{v1}>, <x_{v2}, y_{v2}>)를 생성한다. 각각의 할당범위를 생성한 후에는 태스크를 재배치하기 위한 이동거리를 구한다.

정의 **3**. 할당 서브메쉬 A_i(<x_1, y_1>, <x_2, y_2>)에서 수행중인 태스크의 이동거리(D)는 수평이동거리(horizontal shift distance) d_x와 수직이동거리(vertical shift distance) d_y의 합이다.

d_x와 d_y는 A_i(<x_1, y_1>, <x_2, y_2>)에서 수행중인 태스크를 각각 x축과 y축 방향으로 이동할 수 있는 거리이다. d_x와 d_y는 각각 $0 \le d_x \le x_1$, $0 \le d_y \le y_1$ 조건을 만족한다. d_x와 d_y는 <그림 3>과 같이 A_i(<x_1, y_1>, <x_2, y_2>)와 HAD_i(<x_{h1}, y_{h1}>, <x_{h2}, y_{h2}>), CAD_i(<x_{c1}, y_{c1}>, <x_{c2}, y_{c2}>), VAD_i(<x_{v1}, y_{v1}>, <x_{v2}, y_{v2}>)의 위치에 따라 달라진다.



<그림 3> 할당 서브메쉬의 재배치 유형

<그림 3> 할당 서브메쉬의 재배치 유형에 따른 d_x와 d_y는 <표 1>과 같이 구할 수 있다.

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

유형	CS	ISA	VSA	조건	d_x	d_y
(1)	F	F	F	-	x_1	y_1
(2)	F	F	T	(x_1<x_{v1}) AND (w≤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≤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 $((x_1\le x_{c2})$ AND $(y_1>y_{c2}))$	x_1-x_{h2}	y_1-y_{c2}

<표 1>의 CSA, HSA, VSA는 $CSA_i(<0, 0>, <x_1-1, y_1-1>)$, $HSA_i(<0, y_1>, <x_1-1, y_2>)$, $VSA_i(<x_1, 0>, <x_2, y_1-1>)$에서 각각 $CAD_i(<x_{c1}, y_{c1}>, <x_{c2}, y_{c2}>)$, $HAD_i(<x_{h1}, y_{h1}>, <x_{h2}, y_{h2}>)$, $VAD_i(<x_{v1}, y_{v1}>, <x_{v2}, y_{v2}>)$를 생성하면 T(true) 이고 그 반대의 경우에는 F(false)이다. <표 1>에 따라 d_x와 d_y가 결정되면 $A_i(<x_1, y_1>, <x_2, y_2>)$는 다른 위치의 $S(<x_1-d_x, y_1-d_y>, <x_2-d_x, y_2-d_y>)$에 재배치된다. 이때 할당 서브메쉬의 모든 노드들은 교착상태(deadlock)를 방지하기 위해 이동거리($D=d_x+d_y$)만큼 같은 방향으로 함께 이동한다. 본 논문에서는 태스크를 재배치하는데 소요되는 시간을 다음과 같이 정의한다.

정의 4. 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.

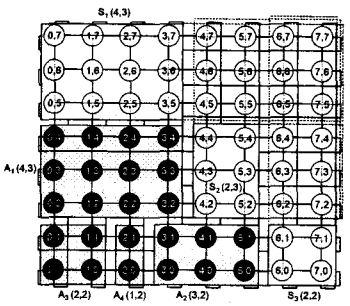
$$RT_i = D \cdot \tau \quad \text{----------(1)}$$

식(1)에서 D는 태스크의 이동거리이고 τ는 태스크 단위이동시간으로 태스크를 이동거리 1만큼 이동하는데 소요되는 시간이다. 태스크 재배치 시간은 태스크의 수행시간에 합산되어 할당방법의 성능을 평가할 때 고려한다.

```
Submesh_Compaction ( )
{
  for(i=0; i <= N; i++)   // N: 할당 서브메쉬의 수
  {
    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> 프로세서 단편 통합 알고리즘

프로세서 단편 통합 알고리즘은 <그림 4>와 같다. <그림 1>을 예로 들어 프로세서 단편 통합 알고리즘을 설명한다. $A_3(<2, 0>, <3, 1>)$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. $A_1(<2, 2>, <5, 4>)$은 <그림 3>의 유형 (3)에 해당한다. $A_1(<2, 2>, <5, 4>)$의 $HSA_1(<0, 0>, <1, 1>)$에는 할당 서브메쉬가 존재하지 않기 때문에 수평할당범위를 생성할 수 없다. 따라서 수평이동거리는 $d_x=2$ 이다. 하지만 $CSA_1(<0, 0>, <1, 1>)$에는 $A_3(<2, 0>, <3, 1>)$가 $S(<0, 0>, <1, 1>)$에 이미 재배치되었기 때문에 $CAD_1(<0, 0>, <1, 1>)$을 생성한다. 그리고 $VSA_1(<2, 0>, <5, 1>)$에는 $A_2(<5, 0>, <7, 1>)$, $A_4(<4, 0>, <4, 1>)$의 할당 서브메쉬들이 존재하기 때문에 각 할당 서브메쉬의 할당 노드를 탐색하여 $VAD_1(<4, 0>, <5, 1>)$를 생성한다. $A_1(<2, 2>, <5, 4>)$과 $CAD_1(<0, 0>, <1, 1>)$, $VAD_1(<2, 0>, <5, 1>)$의 위치에 따라 $d_y=2-2=0$ 이다. 따라서 $A_1(<2, 2>, <5, 4>)$은 $d_x=2$와 $d_y=0$ 만큼 이동하여 $S(<0, 2>, 3, 4>)$에 재배치 된다. $A_2(<5, 0>, <7, 1>)$, $A_4(<4, 0>, <4, 1>)$도 $A_1(<2, 2>, <5, 4>)$과 동일한 과정으로 재배치된다. 프로세서 단편 통합 결과는 <그림 5>와 같다.



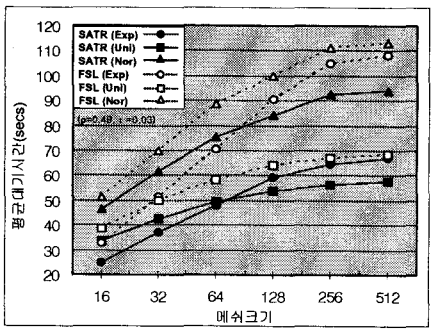
<그림 5> 프로세서 단편 통합 결과

T(4, 6)의 경우 <그림 1>에서는 외적단편화로 인해 할당 가능한 서브메쉬를 찾을 수 없었다. 하지만 <그림 5>에서는 확장지수[1]를 사용하여 $S_2(<4, 2>, <5, 4>)$를 $S_2'(<4, 2>, <7, 7>)$로 확장하면 T(4, 6)에 할당함으로써 T(4, 6)의 대기시간을 줄일 수 있다.

5. 시뮬레이션 및 결과 분석

본 논문에서는 FSL 할당방법[10]과 SATR 할당방법의 성능을 비교한다. 두 할당방법의 성능을 평가하는 척도로 평균대기시간(mean waiting time)을 사용한다. 평균대기시간은 태스크가 큐에 처음 도착한 시점에서 서브메쉬를 할당 받는 시점까지 소요되는 평균 시간을 의미한다.

시뮬레이션에서는 지수분포(Exp), 균일분포(Uni), 정규분포(Nor)에서 각각 300,000개의 태스크를 생성하여 할당한다. 메쉬 크기(16x16~512x512)에서 다양한 시스템 부하($0<\rho\le 1$)와 태스크 단위이동시간($0.01 \le \tau \le 0.05$)에 따른 평균대기시간을 측정한다. <그림 6>은 $\rho=0.49$, $\tau=0.03$에서 다양한 메쉬 크기에 따른 평균대기시간을 측정한 결과이다. <그림 6>을 살펴보면 SATR 할당방법은 FSL 할당방법에 비해 모든 확률분포에서 성능이 우수하며 Exp, Uni, Nor에서 각각 23.2~38.1%, 12.3~ 15.6%, 9.4 ~16.9%의 평균대기시간을 줄였다.



<그림 6> 메쉬 크기에 따른 평균대기시간 ($\rho=0.49$, $\tau=0.03$)

6. 결론

본 논문에서는 메쉬 구조 다중컴퓨터 시스템을 위한 새로운 서브메쉬 할당방법으로 SATR 할당방법을 제안하였다. 이 할당방법은 외적단편화로 인해 서브메쉬의 할당지연이 발생하면 할당 서브메쉬에서 수행중인 태스크를 재배치하고 프로세서 단편들을 통합하여 할당함으로써 태스크 대기시간을 줄였다. 태스크 재배치 과정에서 오버헤드로 발생하는 태스크 재배치시간은 감소하는 태스크 대기시간에 비해 매우 작은 값이다. 따라서 서브메쉬의 할당지연이 발생하면 할당 가능한 가용 서브메쉬가 형성될 때까지 대기하는 방법보다 프로세서 단편들을 통합하여 할당하는 방법이 시스템의 성능 향상에 유리하다는 것을 알 수 있었다.

7. 참고문헌

1) 이원주, 전창호, "태스크와 서브메쉬의 유형별 분류에 기반한 프로세서 할당방법", 정보과학회 2002 봄 학술발표논문집(A), 제 29 권, 제 1 호, pp 589-591, Apr. 2002

2) 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

3) 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

4) 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

5) 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

6) 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

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

8) 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

9) 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

10)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