韓國 OA學會 論文誌
第3卷 第1號, 1998. 3.

論文 98-3-1-1-1

개선된 QA 알고리즘을 이용한 효율적인 서브메쉬 할당

강 영 욱*, 이 재 도**, 양 승 복**

Efficient Submesh Allocation Using Improved QA Algorithm

Yeong-Wook Kang*, Jae-Do Lee**, Seung-Bok Yang**

요 약

2D 메쉬(two-dimensional mesh) 시스템에서 효율적인 서브메쉬 할당이 되기 위해서는 서브메쉬의 할당 시간과 시스템 진입 태스크의 응답 시간이 작아야 한다. Quick Allocation(QA)은 서브메쉬의 할당 시간을 줄이지만 외부 단편화를 발생시킬 가능성이 높다. 이로 인해 태스크의 응답 시간이 커지게 된다. 본 논문에서는 서브메쉬 할당 시간에는 거의 영향을 미치지 않으면서 외부 단편화를 줄일 수 있도록 QA 알고리즘을 개선하였다. 수정된 알고리즘은 QA에 비해 태스크들의 평균 응답 시간을 다소 감소시켰음이 실험을 통하여 확인되었다.

Abstract

In 2D(two-dimensional) mesh system, an efficient submesh allocation scheme must have low time overhead for allocation and low response times for incoming tasks. Quick Allocation(QA) reduces submesh allocation time, but can cause high external fragmentation. Due to this, response times for incoming tasks increase. In this paper, we have improved QA to diminish external fragmentation while having not nearly effect on allocation time. Simulation revealed that modified algorithm reduces to some extent the mean response time of the tasks compared to QA.

* 대구보건전문대학 사무자동화과 전임강사
** 대구보건전문대학 사무자동화과 조교수

논문접수 : 97.11.28. 심사완료 : 98.2.28.

Ⅰ. 서 론

2D 메쉬 시스템은 그 규칙성과 구현의 용이성 때문에 병렬처리 시스템의 설계 분야에서 많은 관심을 끌고 있다. 지금까지 개발된 2D 메쉬 시스템으로는 Intel Paragon[1], Touchst-one Delta System[2], Tera Computer System[3] 등이 있다. 분할 가능한(partitionable) 2D 메쉬 시스템은 다양한 크기의 서브메쉬들로 나뉘어져서 각각 다른 태스크들을 실행시킬 수 있다. 이러한 시스템의 성능을 높이기 위해서는 프로세서들을 태스크에 효율적으로 할당하는 방법을 고려해야 한다.

태스크들은 어떤 특정 도착률로 시스템에 동적으로 제출(submitted)되어 global task queue에 저장된다. 각 태스크는 특정한 폭과 높이의 서브메쉬를 요구한다. 프로세서 할당자(proce-ssor allocator)는 이 태스크에 대해 프로세서 할당 정책에 따라 free 서브메쉬를 찾아서 그 곳에 할당한다. 태스크의 실행이 완료되면 태스크에 할당되었던 서브메쉬는 다른 태스크의 할당을 위해서 해제(released)되어야 한다. 효율적인 서브메쉬 할당 기법이 되기 위해서는 할당과 할당 해제에 있어서 모두 낮은 시간의 오버헤드를 가져야 하고 시스템 진입 태스크(incoming task)들이 낮은 대기 지연(waiting delay)을 가져야 한다.

2D 메쉬에 대한 많은 할당 정책들이 제시되었다. Two-Dimensional Buddy(2DB) [4]는 폭과 높이가 2의 누승인 정방형의 메쉬에만 적용된다. 따라서 요구되는 서브메쉬가 이러한 제약 조건을 만족시키지 않으면 내부 단편화가 발생한다. 이러한 단점을 피하기 위해 Frame Sliding(FS) [5]이 제시되었다. 이 것은 free 서브메쉬를 찾기 위해 Frame Sliding을 이용한다. 이것은 임의의 폭과 높이를 갖는 어떠한 메쉬 시스템에도 적용가능하고 내부 단편화도 없다. 하지만 가용한 free 서브메쉬를 찾지 못하는 경우가 있다. 이를 개선하기 위해First-Fit(FF)/Best-Fit(BF)[6]가 제시되었다. 즉 FF/BF는 가용한 free 서브메쉬를 항상 인식할 수 있다. Adaptive Scan(AS) 기법 역시 free 서브메쉬를 완전히 인식할 수 있지만 할당 시간이 상대적으로 크다. Quick Allocation(QA) [7]은 이상의 여러 기법들에 비해 할당 시간을 대폭 감소시키면서, 동시에 완전한 free 서브메쉬 인식 능력을 가진다. 하지만 외부 단편화에 의한 태스크의 대기 시간이 증가할 수 있다.

본 논문에서는 QA의 단점, 즉 외부 단편화를 가능한 한 최소화 하기 위해 QA 알고리즘을 개선하였다. 본 논문의 기본 사상(basic idea)은 QA에서 시스템 메쉬의 last covered를 구할 때 외부 단편화에 대한 정보를 추가적으로 저장해 둠으로써 보다 효율적인 할당이 되도록 하는 것이다. 이 것은

단지 서브메쉬 할당 시간에 있어서 약간의 증가만을 가져올 뿐이며, 실험을 통하여 개선된 알고리즘이 효과적임을 알 수 있었다.

Ⅱ. 정의 및 표기

시스템 메쉬는 M(W, H)로 표기되며, W×H 개의 노드들이 W×H 2 차원 격자(grid) 구조로 정렬되어 있다.

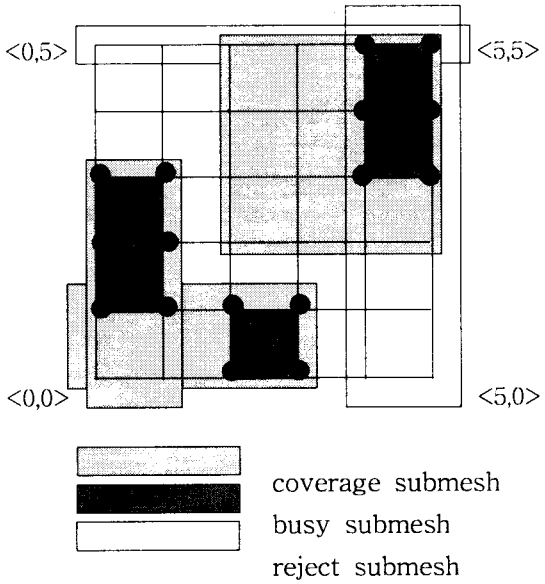


그림 1 T(3, 2)의 할당 예
Fig. 1 An example of allocation for T(3, 2)

각 노드는 하나의 프로세서를 의미한다. W는 시스템 메쉬의 폭을, H는 높이를 나타낸다. i 번째 열과 j 번째 행의 노드는 주소 <i, j>로 표시된다 ($0 \le i \le W-1$, $0 \le j \le H-1$). 행과 열의 인덱스는 0 부터 시작되며, 각각 아래에서 위로, 왼쪽에서 오른쪽으로 증가한다. 시스템 메쉬의 비 경계지역(nonboun-dary) 노드 <i, j>는 네 개의 이웃하는 노드 <i+1, j>, <i-1, j>, <i, j+1>, <i, j-1>들과 연결되어 있다. 경계지역 노드는 위치에 따라서 두 개 또는 세 개의 이웃 노드들을 가진다. Fig. 1.은 시스템 메쉬 M(6, 6)을 나타내고 있다. 또한 시스템 진입 태스크를 T(w, h)로 나타내며, w와 h는 각각 T가 요구하는 서브메쉬의 폭과 높이를 의미한다.

정의 2.1. 서브메쉬 S(w, h)의 주소는 <x, y, x′, y′>로 표기하며, <x, y>와 <x′, y′>는 각각 S의 왼쪽 하단(lower-left: ll) 노드와 오른쪽 상단(upper-right: ur) 노드를 의미한다. 또한 S의 ll 노드를 S의 베이스라 한다.

정의 2.2. busy 서브메쉬는 β로 표기하며, 소속된 모든 프로세서(노드)들이 현재 어떤 태스크에 할당되어져 있는 서브메쉬를 의미한다. B는 모든 busy 서브메쉬들의 집합을 나타낸다. 예를 들면, Fig. 1.에서 M(6, 6)에는 세 개의 서브메쉬들

이 존재한다; B= {β1, β2, β3}, 여기서 β1 = <4, 3, 5, 5>, β2 =<0, 1, 1, 3>, β3=<2, 0, 3, 1> 이다. 반대로 free 서브메쉬는 소속된 모든 노드들이 모두 free 상태, 즉 어떠한 태스크에도 할당되지 않은 노드들로 구성된 서브메쉬를 의미한다.

정의 2.3. T에 대해 β의 coverage 서브메쉬를 $\xi\beta,T$로 표기하며, 이것은 β로 인해, 소속된 어떠한 프로세서(노드)도 T를 할당하기 위한 free 서브메쉬의 베이스가 될 수 없는 서브메쉬를 의미한다.
β<x, y, x′, y′>와 T(w, h)에 대해, $\xi\beta,T$는 <xc, yc, x′, y′> 이다. 여기서 xc = <0, x-w+1>, yc = < 0, y-h+1> 이다. 또한 T에 대한 coverage 서브메쉬들의 집합을 ΞT로 표기한다. 즉 ΞT = {$\xi\beta,T$|β∈B} 이다. 예를 들면, Fig. 1.에서 T(3, 2)에 대해, $\xi\beta1,T$=<2, 2, 5, 5>, $\xi\beta2,T$ =<0, 0, 1, 3>, $\xi\beta3,T$=<0, 0, 3, 1> 이고, ΞT = {<2, 2, 5, 5>, <0, 0, 1, 3>, <0, 0, 3, 1>} 이다.

정의 2.4. T(w, h)에 대한 reject 서브메쉬는 δ로 표기하며, T를 할당하기 위해 어떠한 free 서브메쉬의 베이스로도 될 수 없는 프로세서(노드)들로 구성된 서브메쉬이다. 각 T에 대해서 2 개의 reject 서브메쉬가 존재한다. 하나는 수평방향(δTH), 다른 하나는 수직방향(δTV)의 reject 서브메쉬이다. 이것은 간단히 계산된다. 즉 δTH = <w′, 0, W-1, H-1>, δTV = <0, h′, W-1, H-1> 이고, 여기서 w′ = W-w+1, h′ = H-h+1 이다. reject set △T는 δTH와 δTV의 합집합이다.
Fig. 1.에서 δTH = <0, 5, 5, 5>, δTV = <4, 0, 5, 5> 이다.

정의 2.5. 내부 단편화는 요구된 프로세서 수와 실제 할당된 프로세서 수의 비율을 의미한다. 또, T가 요구하는 프로세서 수(wh)에 대해 시스템 메쉬의 전체 free 프로세서 수가 큼에도 불구하고 T를 프로세서에 할당할 수 없을 경우, 외부 단편화를 시스템 메쉬의 전체 프로세서 수(WH)에 대한 free 프로세서 수의 비율로 나타낸다.

Ⅲ. Quick Allocation(QA)

QA [7]는 B로 부터 Ξ_T를 생성시킬 때

각 행에 대한 정보를 수집함으로써, 그 행의 노드들이 B나 Ξ에 속하는지를 개별적으로 체크하지 않고도, T를 수용할 free 서브메쉬의 베이스로 가능한 노드가 그 행에 존재하는지를 빨리 알 수 있다는 것이 QA의 핵심 내용이다. 더욱이 AS의 scan/sliding을 피할 수 있어 결과적으로 할당 시간과 대기 지연이 현저히 줄어들게 된다.

3.1 Last_covered

last_covered라고 하는 1 차원 배열을 하나 둔다. 이 배열의 초기값은 -1이고, 시스템 메쉬의 각 행에 대해 최 우측(right-most) covered 노드의 x 좌표를 저장한다.

정의 3.1. 메쉬 시스템에서 어떤 행의 연속 노드 열(a sequence of consecutive nodes)을 세그먼트(segment)라 한다. 만약 세그먼트가 어떤 행의 최 좌측(left-most) 노드에서 시작하고, 그 세그먼트의 모든 노드들이 ΞT의 하나 이상의 서브메쉬에 속한다면, 그 세그먼트를 covered 세그먼트라 부른다.

last_covered[j] ($0 \le j \le h'-1$)는 j 번째 행의 covered 세그먼트에 속한 마지막 노드의 x 좌표이다. 만약 j 번째 행에 covered 세그먼트가 존재하지 않는 다면, last_covered[j]는 -1 이 된다. 예를 들어, Fig. 1.에서 last_covered[j] (j = 0, 1, 2, 3, 4)는 각각 3, 3, 5, 5, -1 이다.

3.2 서브메쉬 할당

어떤 노드가 B의 서브 메쉬에 속한다면 Ξ에도 속한다. 따라서 노드가 coverage 서브 메쉬에 속하는지만 체크하면 되고, 그러한 정보는 배열 last_covered에서 알 수 있다. 예를 들면, Fig. 1.에서 $w' = 4$, $h' = 5$ 이다.

Fig. 1.에서 last_covered[j]($0 \le j \le 3$) +1 $\ge w'$ 이고 last_covered[j] (j = 4) + 1 < w' 이므로 노드 <last_covered[j]+1, j>, 즉 노드 <0, 4>가 free 서브메쉬 <0, 4, 2, 5>의 베이스로서 가능하다.

Ⅳ. QA 알고리즘의 개선

4.1 free 서브메쉬의 경계값 정의

QA는 할당 시간의 감소에 집중한 결과, 높은 외부 단편화 때문에 태스크의 응답 시간이 늦어질 수 있다. Fig. 2.에서 태스크 T(3, 3)에 대하여 QA 알고리즘에 의하면,

서브메쉬 <5, 0, 7, 2>가 할당된다. 이 것은 서브메쉬 <0, 2, 2, 4>가 할당되었을 때 보다 향후 외부 단편화의 가능성이 높다. 따라서 할당 가능한 서브메쉬들이 하나 이상 존재할 경우, 이 들 중 가장 적합한 서브메쉬를 선택하기 위한 기준이 필요하다. 이를 위해 다음을 정의한다.

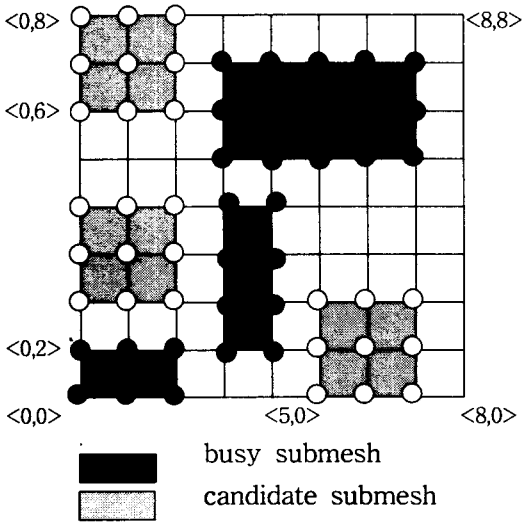


그림 2 T(3, 3)의 할당 예
Fig. 2 An eample of allocation for T(3, 3)

정의 4.1. 시스템 메쉬 M(W, H)에서 free 노드 <i, j>의 경계값(boundary value)은 <i, j>의 할당된 이웃 노드들의 수와, <i, j>가 M의 경계에 위치했을 경우 M의 경계점(boundary point)들의 수를 합한 값이다. 또, free 서브 메쉬의 경계값은 그 서브 메쉬의 주변(periphery)에 위치한 모든 노드들의 경계값을 합한 값이다. 예를 들면, Fig. 2.에서 free 서브 메쉬 <5, 0, 7, 2>, <0, 2, 2, 4>, <0, 6, 2, 8>의 경계값은 각각 5, 9, 8 이다.

4.2 free 서브메쉬의 경계값 계산

QA 알고리즘에서 시스템 진입 태스크 T(w, h)에 대해 할당 가능한 후보 free 서브 메쉬(이하 '후보 서브 메쉬'라 한다)들의 베이스는 배열 last_covered로 부터 알 수 있으며, 그 베이스들은 $0 \le$ last_covered[j] $(0 \le j < h') + 1 < w'$ 인 모든 j에 대한 노드 <last_covered[j]+1, j> 들이다. 이 후보 서브 메쉬 들의 경계값을 각각 계산하여 가장 높은 경계값을 갖는 후보 서브 메쉬를 T에 할당한다.

본 논문에서는 이 들 각 후보 서브 메쉬의 경계값을 저장하기 위해 1 차원 배열 boundary_value를 둔다. 따라서 T에 대해 노드 <last_covered[j]+1, j> 를 베이스로 하는 free 서브 메쉬의 경계값은 boundary_value[j] 이다. 또한 나중에 후보 서브 메쉬의 경계값을 계산하기 위해 Last_covered 프로시져에서 각 β<x, y, x′, y′>에 대해 $\xi_{\beta,T}$<x_c, y_c, x′,

y'>를 결정할 때, β의 베이스의 y 좌표를 추가로 저장한다. 즉 coverage 서브메쉬의 자료구조를 바꿔 $\xi_{\beta,T}$<x_c, y_c, x', y', ybase>로 저장한다. 여기서 ybase = y 이다.

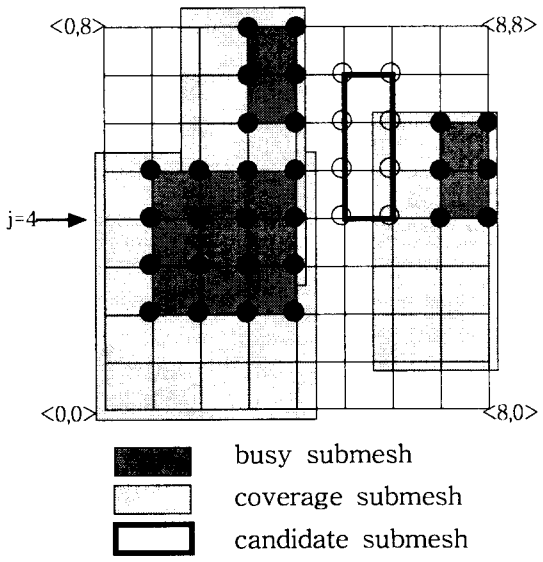


그림 3 T(2, 4) 할당 예
Fig. 3. An example of allocation for T(2, 4)

4.3 boundary_value[j]의 초기화

태스크 T(w, h)에 대해, 시스템 메쉬 M의 각 행 j (0≤j≤h'-1)에 대한 후보 서브메쉬는 <last_covered[j]+1, j, last_covered[j]+w, j+h-1>이다. 배열 last_covered의 초기값이 -1 이므로 후보 서브메쉬는 <0, j, w-1, j+h-1> 이 된다. 이 경우 베이스의 x 좌표가 0 이므로 후보 서브메쉬의 경계값은 최소한 h가 된다. 또, j = 0 이거나 j = h'-1 일 경우는 경계값이 w 만큼 증가된다. 따라서 배열 boundary_value를 다음과 같이 초기화한다.

```
For each j (0≤j≤h'-1),
  if (j = 0 OR j = h'-1)
     boundary_value[j] ← w + h
  else
     boundary_value[j] ← h
```

4.4 boundary_value[j]의 재 설정

Fig. 3.과 같이 T(2, 4)의 경우를 가정해 보자. busy 서브메쉬 <1, 2, 4, 5>, <3, 6, 4, 8>, <7, 4, 8, 6>을 각각 β_1, β_2, β_3 라 두고, 각 β_1, β_2, β_3에 대해 결정되는 coverage 서브메쉬 <0, 0, 4, 5, 2>, <2, 3, 4, 8, 6>, <6, 1, 8, 6, 4>를 각각 ξ_1, ξ_2, ξ_3 라 두자. 먼저 ξ_1 에 대해 j = 4 일 때 last_covered[4]의 초기값은 -1 이므로 x_c=0 ≤last_covered[4]+1≤x'=4 를 만족한다. 따라서 last_covered[4]의 값은 4로 변경된다. 이때 새로운 후보 서브메쉬는 초기의 <0, 4, 1, 7>에서 <5, 4, 6, 7>로 변경된다.

이와 같이 각 행 j에 대해 후보 서브메쉬가 변경되면 그 서브메쉬의 경계값도 새로 설정되어야 한다. Fig. 3.에서 ξ_1의 ybase

값(= 2)과 y′ 값(= 5), 그리고 후보 서브 메쉬의 ll 노드와 ur 노드의 y 좌표 값(각각 4, 7)으로부터, 후보 서브 메쉬와 busy 서브 메쉬 β_1이 서로 이웃해 있는 노드 수(= 2)를 계산할 수 있으며, 그 수를 neighbours라 둔다. 또한 후보 서브 메쉬의 right-most 노드들이 시스템 메쉬의 우측 경계에 위치했다면 boundary_value는 h만큼 증가하게 된다. 다음은 last_covered[j]가 변경 될 경우 boundary_value[j]을 재 설정하는 루틴이다.

```
if (xc ≤ last_covered[j] + 1 ≤x′)
    boundary_value[j] ← neighbours
    if (last_covered[j] + w = W-1)
        boundary_value[j]←
boundary_value[j]+h
```

4.5 boundary_value[j] 갱신

앞의 예에서, 다음으로 ξ_2에 대해 j = 4 일 때, x_c=2 ≤ last_covered[4]+1 = 5 ≤ x′=4 가 만족되지 않기 때문에 last_covered [4]의 값은 여전히 4가 된다. 역시 ξ_3에 대해서도 j = 4 일 때, x_c=6 ≤ last_covered [4]+1 =5 ≤ x′=8 가 만족되지 않으므로 last_covered[4]는 불변이다. 이와 같이 last_covered[j] 값이 변동되지 않는 경우, coverage 서브 메쉬 ξ와 후보 서브 메쉬가 서로 이웃하면, 그 이웃하는 노드 수(neighbours)만을 boundary_value[j]에 더해 주면 된다. 예에서 ξ_2, ξ_3 은 모두 후보 서브 메쉬에 이웃하고, neighbours 값은 각각 2, 3 이 된다. 다음은 boundary_value[j]을 갱신하는 루틴이다.

```
if NOT (xc ≤ last_covered[j] + 1 ≤ x′)
 if (last_covered[j] = x′ OR
    last_covered[j] + 1 = xc - 1)
    boundary_value[j] ← boundary_value[j]
                        + neighbours
```

4.6 Modified QA (MQA)

이상의 내용에 의하여 QA를 개선한 수정된 알고리즘(MQA)을 다음에 나타내었다. 굵게 나타낸 부분이 원래의 QA 알고리즘에 추가되었거나 변경된 부분이다.

Procedure Last_covered

```
last_covered[j] (0≤j≤h′-1) ← -1.
For each j (0≤j≤h′-1),
  if (j = 0 OR j = h′-1)
    boundary_value[j] ← w+h
  else
    boundary_value[j] ← h
For each β<x, y, x′, y′>,
   determine ξ β,T<xc, yc, x′, y′, ybase>
Arrange ξ β,Ts in the increasing order of xc

For each ξ β,T (starting from one whose xc
                                is smallest)
```

```
if (yc < h')
   for each row j (yc≤j≤min(y', h'-1))
     if (xc ≤ last_covered[j]+1 ≤ x')
       last_covered[j] ← x'
       boundary_value[j]←neighbours
       if (last_covered[j] + w = W-1)
          boundary_value[j] ←
                boundary_value[j] + h
     else
        if (last_covered[j] = x' OR
           last_covered[j]+1 = xc-1)
           boundary_value[j] ←
                  boundary_value[j]
                    + neighbours
```

Procedure Allocation

Step 1.

flag ← false. /* the flag representing the orientation */

Step 2.

if (number of free processors < w×h)
go to Step 7.

Step 3.

Decide the orientation of T as follows, and determine the reject set

```
if (flag = false)
  T←T(w, h), w'←W-w+1, h'←H-h+1
else
  T←T(h, w), w'←W-h+1, h'←H-w+1
```

Step 4.

Based on current B and T, determine $\Xi_{\beta,T}$ and last_covered as explained above

Step 5.

```
max_value ← -∞,  max_j ← -1
j ← 0
while (0 ≤ j < h')
 if (last[_covered[j]+1 < w' AND
    boundary_value[j]>max_value)
    max_value← boundary_value[j]
    max_j ← j
 j ← j + 1

if (max_j = -1) /* no free submesh is
                                found */
     if (flag = false)
        flag ← true
        go back to Step 3
     else
        go to Step 7.
  else /* a free submesh is found */
     j ← max_j
     i ← last_covered[j] + 1
     go to Step 6.
```

Step 6.

```
if (flag = false)
   S← <i, j, i+w-1, j+h-1>
else
   S← <i, j, i+h-1, j+w-1>
```

Allocate S to T and add S to B. Stop.

Step 7.

flag← false.

Wait until a submesh is released.

위의 프로시져에서 알 수 있듯이 추가 또는 변경된 코드의 시간 복잡도는 상수로서, 할당 시간에 미치는 영향은 아주 작다.

V. 실험 및 평가

5.1 실험 모델

C/CSIM [8]을 이용하여 QA와 MQA의 성능을 비교하였다. 시스템 메쉬의 크기는 64×64를 대상으로 하였으며, 태스크의 실행 시간과 태스크간 도착 시간(inter-arrival time)은 각각 EXTM, IATM의 평균을 갖는 지수 분포를 취하도록 하였다. 태스크의 다양한 특성을 반영하기 위해 태스크의 크기(w와 h)는 일양 분포, 감소 분포, 증가 분포 등 3 종류의 분포에 대해 각각 실험하였다. 일양 분포에서 w, h는 1~N (N = 64) 사이의 어느 한 값을 동일한 확률로 갖는다. 감소 분포에서는 범위 1~N 을 네 구간, 즉 [1-8], [9-16], [17-32], [33-64] 로 나누고, w, h가 특정 구간에 속할 확률을 차례대로 0.4, 0.2, 0.2, 0.2 로 정하였다. 하지만 구간 내에서 w, h는 여전히 일양분포를 취한다. 증가 분포에서는 네 구간 [1-32], [33-48], [49- 56], [57-64] 로 나누고 w, h가 특정 구간에 속할 확률을 차례대로 0.2, 0.2, 0.2, 0.4 로 정하였다. 마찬가지로 구간내에서 w, h는 일양 분포이다.

본 실험에서 매 실험 당 50,000 개의 태스크를 발생시켜 각 태스크의 응답 시간(시스템에 제출되어 실행이 완료되기 까지의 시간)을 기록하였으며 할당 시간 오버헤드는 무시하였다. 다양한 부하 하에서의 시스템의 성능을 보기 위해 다음과 같이 load를 정의한다.

$$load = (n \times EXTM) / (N \times IATM)$$

여기서, n은 태스크가 요구하는 서브메쉬의 평균 크기이고, N은 전체 프로세서 수이다. 실험에서 EXTM은 10.0 으로 정하였으며, IATM은 요구되는 load에 따라 조정하였다.

5.2 평 가

서로 다른 load 하에서 QA와 MQA의 평균 응답 시간을 비교한 실험결과를 다음의 Table에 나타내었다. load가 0.5~0.7 사이에 이를 때 시스템은 불안정 상태에 빠졌다. Table에서 보듯이 MQA가 QA에 비해 평균 응답 시간을 전체적으로 10% 감소시킨 것으로 나타났다. 또한 load가 작을 경우 응답 시간의 차이는 거의 없으나, load가 증가할수록 QA와 MQA의 평균 응답 시간 차이가

더 커진다. 결국 MQA가 QA에 비해 보다 효율적임을 알 수 있다.

표 1 QA와 MQA의 평균 응답 시간
Table 1 Mean Response Time vs. load
(w, h: uniform distribution)

Load	QA	MQA	감소율
0.1	11.153	11.066	0.8%
0.2	13.297	13.048	1.9%
0.3	19.396	18.220	6.1%
0.4	35.144	31.370	10.7%
0.5	318.507	195.082	38.8%

표 2 QA와 MQA의 평균 응답 시간
Table 2 Mean Response Time vs. load
(w, h: decreasing distribution)

Load	QA	MQA	감소율
0.1	10.395	10.331	0.6%
0.2	12.137	11.683	3.7%
0.3	17.217	15.925	7.5%
0.4	42.624	31.053	27.1%

VI. 결 론

본 논문에서는 외부 단편화 문제를 개선하기 위해 QA 알고리즘을 수정하였다. 수정된 알고리즘(MQA)은 외부 단편화를 줄이기 위해 먼저 시스템 메쉬의 각 행에 대한 free 서브메쉬의 경계값을 계산하고, 가장 큰 경계값을 갖는 행의 free 서브메쉬를 할당한다.

표 3 QA와 MQA의 평균 응답 시간
Table 3 Mean Response Time vs. load
(w, h: increasing distribution)

Load	QA	MQA	감소율
0.1	11.549	11.531	0.2%
0.2	13.863	13.799	0.5%
0.3	18.454	18.149	1.7%
0.4	25.974	25.337	2.5%
0.5	45.748	42.744	6.6%
0.6	217.079	122.570	43.5%

MQA는 할당 시간에 거의 영향을 미치지 않으며, QA에 비해 태스크의 평균 응답 시간을 선체적으로 10% 감소시킴으로써 QA의 문제점으로 지적되어 온 외부 단편화가 다소 개선되었음이 실험을 통하여 입증되었다.

참 고 문 헌

[1] "Paragon XP/S Product Overview," Intel Corporation, 1991

[2] "A Touchstone DELTA System Description," Intel Corporation, 1991.

[3] R. Alverson et al., "The Tera Computer System." Proc. 1990 Int'l Conf. on Supercomputing, pp. 1-6, 1990.

[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 Conference, pp. 22-28, Feb. 1990.

[5] P.-J. Chuang and N.-F. Tzeng, "An Efficient Submesh Allocation Strategy for Mesh Computer Systems," Proc. International Conference on Distributed Computing Systems, pp. 256-263, May. 1991.

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

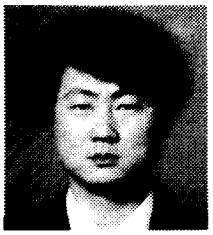
[7] S.M. Yoo, H.Y. Youn and B. Shirazi, "An Efficient Task Allocation Scheme for 2D Mesh Architectures," IEEE Trans. Parallel and Distributed Systems, vol. 8, no. 9, Sep. 1997

[8] H. Schwetman, CSIM User's Guide, Microelectronics and Computer Technology Corporation, 1991.

□ 筆者紹介

강영욱
1989년 동아대학교 컴퓨터공학과(공학사)
1990년 경남대학교 전자계산학과(공학석사)
1996년 ~ 현재 대구보건전문대학 사무자동화과 전임강사

이재도
1985년 영남대학교 전자공학과(공학사)
1989년 영남대학교 전자공학과(공학석사)
1996년 ~ 현재 영남대학교 컴퓨터공학과 박사과정
1991년 ~ 현재 대구보건전문대학 사무자동화과 조교수



양승복
1985년 영남대학교 전자공학과(공학사)
1988년 영남대학교 전자공학과(공학석사)
1992년 ~ 현재 경남대학교 컴퓨터 공학과 박사과정
1990년 ~ 현재 대구보건전문대학 사무자동화과 조교수