이차원 메쉬 상호 연결 망에 대한 효율적인 태스크할당 알고리즘

추 현 승[†] · 박 경 린[††] · 유 성 무[†††]

요 약

이차원적(2D) 메쉬(mesh)로 연결된 프로세서들에 있어서 새로이 시스템 내부로 진입하는 태스크에 적당한 크기를 갖는 부분메쉬(submesh) 형태로 구성된 프로세서들을 효율적으로 할당하는 일은 요구되는 높은 성능을 갖게 하기 위하여 매우 중요하다. 그러한 작업은 최소한의 오버헤드로 자유로운 부분메쉬의 인식이 보장되어야 할 필요가 있다. 본 논문에서는 2D 메쉬에 대한 효과적인 태스크할당 알고리즘을 소개한다. 간단한 1차원 배열 검색을 이용한 새로운 방식으로 할당 가능한 부분메쉬를 찾을 수 있게 함으로서 2차원 배열의 전체 검색을 이용하던 종래의 설계와는 차별화 된다. 결과적으로 새로운 알고리즘은 태스크할당 시간을 현격하게 줄일 수 있다. 종합적인 컴퓨터 시뮬레이션은 평균 할당시간 및 대기로 인한 지연시간에 있어서 전체 메쉬의 크기에 상관 없이 기존의 알고리즘들보다 효과적임을 보여준다. 하드웨어 오버헤드는 다른 알고리즘들과 비슷한 수준을 갖는다.

An Effective Task Allocation Algorithm in Two-Dimensional Mesh Interconnection Networks

Hyun-Seung Choo[†] · Gyung-Leen Park[††] · Seong-Moo Yoo[†††]

ABSTRACT

An effective allocation of requested number of processors to newly incoming tasks in two-dimensional (2D) mesh interconnection networks is very important for achieving the desired high performance and resource utilization. It also needs to guarantee the complete recognition of the free submeshes based on contiguous and available processors with minimum overhead. An efficient task allocation algorithm for 2D meshes is presented in this paper. By employing a new approach for searching the one-dimensional array, the proposed algorithm can find the available submesh without the scanning of the entire 2D array unlike earlier designs. As a result, the new algorithm can significantly reduce the task allocation time. Comprehensive computer simulation shows that the average allocation time and waiting delay are much smaller than earlier designs irrespective of the size of meshes. The hardware overhead is comparable to other algorithms.

1. 서 론

병렬과 분산계산을 위하여 개발된 다수의 중요한 상호연결 토폴로지 중에서 이차원적(2D) 메쉬는 그의 간결함과 효율성으로 인하여 넓이 사용되고 있다[1,2]. 여러 상용 또는 실험용 수퍼 컴퓨터가 2D메쉬를 기본으로 개발되었거나 개발 중에 있다. 전형적인 예로는 Intel Paragon[3], Intel/DARPA Touchstone Delta[4], 그리고 Tera Computer System[5] 등이 있다.

† 정 회 원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
†† 정 회 원 : 제주대학교 자연과학대학 전산통계학과 교수
††† 비 회 원 : 미국 콜럼버스주립대 전산학과 교수
논문접수 : 1998년 8월 12일, 심사완료 : 1999년 4월 2일

일반적인 병렬 컴퓨터 시스템[6,7]에서처럼 2D메쉬 시스템에서도 새로운 태스크들이 적당한 크기의 부분메쉬의 프로세서들을 할당 받는다. 여기에서 언급되는 부분메쉬의 크기는 단 하나의 노드에서부터 전체메쉬까지 이다. 일반적으로, 분리되어 있는 호스트 프로세서가 존재하고, 그의 운영 체제는 태스크 스케쥴러와 태스크할당기로 구성되는 태스크 분배기를 포함한다고 가정된다. 태스크 스케쥴러는 다음에 처리되어야 할 태스크를 대기 큐로부터 태스크 스케쥴링 방식에 따라서 선택한다. 태스크할당기는 선택된 태스크에 대하여 태스크 할당알고리즘을 이용하여 적당한 크기의 아직 할당되지않은 자유로운 부분메쉬를 찾는다. 본 논문은 태스크할당기에 대하여만 다루도록 한다. 메쉬의 크기가 증대함에 따라서 효율적인 부분메쉬의 할당은 더욱 시간을 요하는 일이 된다. 또한 할당알고리즘은 단편화 문제를 고려할 필요가 있다.

2D메쉬에 대한 여러 할당 알고리즘이 문헌에 발표되었다. Li와 Cheng[8]은 2D Buddy전략을, Chuang과 Tzeng[9]은 Frame Sliding (FS)알고리즘을 제안했다. Zhu의 First Fit과 Best Fit 전략[10]은 FS 알고리즘을 향상시켰다. 그러나 이러한 알고리즘들은 완전한 부분메쉬 인식 능력이 없었고, 상대적으로 간단한 경우들만이 고려되었다. 그 이후에 Ding과 Bhuyan[11]이 Adaptive Scan 알고리즘을 제안했고, Shama와 Pradhan [12]은 할당을 위하여 할당해제 리스트를 도입했다. 그들의 알고리즘은 완전한 인식을 가능하게 했지만 상대적으로 큰 할당시간을 필요로 한다.

본 논문에서 2D메쉬에 대한 새롭고 효율적인 태스크할당 알고리즘을 제안한다. 이 알고리즘은 할당 상태에 관한 각 행의 정보를 효과적으로 조작하여 어느 한 행의 일부가 이 시스템으로 들어온 태스크에게 할당되어질 수 있는지를 신속히 결정한다. 제안되는 알고리즘은 2D배열형태의 메쉬 전체 검색 없이 할당 가능한 부분메쉬를 찾을 수 있게 함으로서 종래의 설계와는 다르다. 이러한 방법은 완전한 부분메쉬 인식 능력을 갖으면서도 기존의 알고리즘들과 비교하여 할당시간을 현저하게 줄이는 결과를 낳는다. 종합적인 컴퓨터 시뮬레이션은 평균 할당시간 및 대기로 인한 지연시간에 있어서 기존의 알고리즘들보다 메쉬의 크기 및 할당되는 부분메쉬 모양의 분포와 상관 없이 시간적으로 적게 걸림을 보여준다. 하드웨어 오버헤드는 다른 알고리즘들과 비슷한 수준을 갖는다.

본 논문은 다음과 같이 구성된다. 제 2절에서는 본 논문을 통해서 사용되는 정의와 표기방법이 소개되고, 또한 기존의 태스크할당 알고리즘들이 간략하게 논의된다. 제 3절에선 제안되는 태스크할당 및 할당해제 알고리즘이 상세히 서술된다. 제안되는 알고리즘의 시간 복잡도 및 하드웨어 오버헤드가 제 4절에서 분석되고 다른 알고리즘들과 비교된다. 그 알고리즘의 성능이 분석되고 컴퓨터 시뮬레이션의 결과와 비교된다. 마지막으로 본 논문의 결론이 제 5절에 서술된다.

2. 2D 메쉬에서의 태스크할당

먼저 여러 변수들과 표기방법을 정의한다. 그리고 기존의 2D 메쉬에 있어서 제안된 태스크할당 방법들에 대해서 간략하게 논의한다.

2.1 정의 및 표기방법

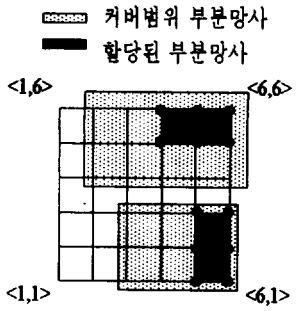
이차원 메쉬 $M(a, b)$는 ab개의 노드들로 구성된 $a \times b$ 직사각형 격자로서 여기서 a와 b는 메쉬의 폭(너비)과 높이를 각각 나타낸다. 메쉬에서의 각 노드는 좌표 $<x,y>(1 \le x \le a,\ 1 \le y \le b)$로 표현되는 하나의 프로세서를 의미한다. 행과 열의 색인은 1부터 시작하여 좌에서 우로 아래에서 위로 증가함을 가정한다. 또한 새로 진입하는 태스크의 폭과 높이는 각각 w와 h라고 가정한다.

정의 1 : 부분메쉬 $S(w,h)$의 주소는 4원소 주소 $<x,y,x',y'>$로 표현되는데, $<x,y>$와 $<x',y'>$는 S의 좌측하단(ll)과 우측상단(ur)의 노드를 각각 나타낸다. 분명히 $w=x'-x+1$이고 $h=y'-y+1$이다. 부분메쉬 S의 *기준*은 ll노드를 가리키고, 그의 *면적*은 노드들의 개수 wh로 정의된다.

정의 2 : *사용중인 부분메쉬* β는 그 내부의 모든 프로세서가 현재 어느 한 태스크에 할당되어 있음을 의미하고, *사용중인 집합* B는 모든 사용중인 부분메쉬들의 합집합이다. 예를 들어 (그림 1)에는 사용중인 두개의 부분메쉬들이 $M(6,6)$에 존재한다. $B=\{\beta_1, \beta_2\}$ 여기서 $\beta_1=<5,1,6,3>$이고 $\beta_2=<4,5,6,6>$이다.

정의 3 : 진입하는 태스크 T에 대한 β의 *커버범위* 부

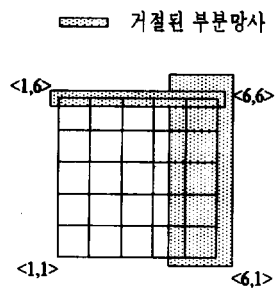
분메쉬 $\xi_{\beta,T}$는 사용중인 부분메쉬 β로 인하여 T에게 부분메쉬의 기준으로 할당되어질 수 없는 프로세서들의 모임이다. β<x,y,x'y'>과 T(w,h)에 대하여 $\xi_{\beta,T}$는 <x_c,y_c,x',y'>이고 여기서 x_c=max(1,x-w+1)과 y_c=max(1,y-h+1)이다. 또한 커버범위 부분메쉬에 있는 노드들을 *커버된* 노드들이라 정의한다. T에 관한 커버범위 집합 Ξ_T는 T에 관한 커버범위 부분메쉬들의 집합이다. 즉 Ξ_T={$\xi_{\beta,T}$ | $\beta \in$ B}. 예로써 (그림 1)에서 T(3,2)에 대하여 $\xi_{\beta1,T}$=<3,1,6,3>, $\xi_{\beta2,T}$=<2,4,6,6>, 그리고 Ξ_T={<3,1,6,3>, <2,4,6,6>}.



(그림 1) T(3,2)에 대한 할당된 부분망사 및 커버범위 부분망사들

정의 4 : T에 대한 *거절된 부분메쉬* δ_T는 T에게 부분메쉬의 할당을 고려하는데 있어서 임의의 자유로운 부분메쉬의 기준으로 선택되어질 수 없는 프로세서들로 구성된다. 각 T에 대하여 두개의 거절된 부분메쉬가 존재하며 이들은 수평(δ_{TH}) 및 수직(δ_{TV})방향에 하나씩 존재한다. <a,b>가 전체 메쉬의 ur노드라고 했을 때 이들은 δ_{TV}=<a',1,a,b>와 δ_{TH}=<1,b',a,b>이고, 여기서 a'=a-w+2이고 b'=b-h+2이다. *거절된 집합* Δ_T는 δ_{TH}와 δ_{TV}의 합집합이다. 예로써 (그림 2)에서 δ_{TV}=<5,1,6,6>이고 δ_{TH}=<1,6,6,6>이다.

정의 5 : *내부 단편화*는 실질적으로 요구되는 프로세서수에 대하여 요구된 개수 이상으로 과대 할당된 프로세서수의 비율이다. *외부 단편화*는 진입하는 태스크에 있어서 충분히 많은 수의 자유로운 프로세서가 있음에도 할당작업이 실패했을 경우에 전체 메쉬의 프로세서수에 대한 자유로운 프로세서수의 비율이다.



(그림 2) T(3,2)에 대한 거절된 부분망사

2.2 기존의 할당 알고리즘

여기에서 2D메쉬에 대한 기존의 태스크할당 방식들을 간단히 정리함으로써 태스크할당 문제의 특성을 확인한다.

① *이차원적 Buddy* (2DB) 알고리즘 : 이 방법은 Li와 Cheng[8]에 의하여 제안되었고 단지 한 변의 길이가 2의 거듭제곱인 정방형 메쉬에만 적용된다. 임의의 크기 $w \times h$를 갖고 시스템으로 진입하는 태스크에 이 알고리즘은 언제나 $2^i \times 2^i$ 여기서 $i = 2^{\lceil \log\ 2\ (\max(w,h))\rceil}$ 인 부분메쉬만을 할당한다. 이러한 방식은 태스크가 필요로 하는 만큼 이상의 과대할당으로 인한 내부 단편화문제가 발생하고, 이는 대부분의 작업들이 반드시 정방형메쉬를 필요로 하는 것은 아니기 때문이다.

② *Frame Sliding* (FS) 알고리즘 : Chuang와 Tzeng [9]은 과대할당으로 인한 내부 단편화의 문제점을 해결하기 위하여 이 방법을 제안했다. 이 방법에서는 진입하는 태스크에게 그와 정확히 일치하는 frame이라 불리는 자유로운 부분메쉬를 할당한다. 이 방법은 임의의 태스크 T에 대하여 Ξ_T와 Δ_T 내부의 어떠한 노드도 T를 맞이할 부분메쉬의 기준으로 부적합하다는 사실에 기인한다. Frame의 sliding은 ll 노드에서부터 시작한다. 어느 한 노드 <x,y>가 B, Ξ_T, 또는 Δ_T에 포함되어 있지 않으면 자유로운 부분메쉬S <x,y,x+w-1,y+h-1>를 찾게 되고, 그렇지 않으면 sliding은 다음의 후보노드로 계속된다. Frame이 sliding되는 방향은 처음에 수평방향으로 w의 폭을 갖고 좌에서 우로 진행된다. 이러한 진행이 전체메쉬의 오른쪽 변을 넘게 되면 수직방향으로 폭h를 유지하며 sliding이 발생하고 다시 우에서 좌로의 수평방향 sliding이 진행되고, 이러한 방법을 반복하여 자유로운 부분메쉬의 기준을

찾는다. 이러한 방법은 과대할당으로 인한 문제점을 해결하기는 하나 검색작업은 할당실패를 초래할 수 도 있다. 즉, 이 알고리즘은 할당 가능한 자유로운 부분메쉬가 존재하여도 알고리즘의 특성상 인식하지 못하는 경우가 발생한다.

③ *First Fit* (FF)와 *Best Fit* (BF) 알고리즘 : 할당실패의 문제점을 해결하고자 Zhu[10]는 *busy배열*을 이용한 두 가지의 알고리즘을 제안했는데 여기서 한 원소 [i,j]는 한 노드 <i,j>의 할당상태여부 즉 (이미 다른 태스크에 할당되어) 사용 중인지 아닌지에 따라서 1 또는 0값을 갖는다. 또한 알고리즘에서 T에 관한 *coverage배열*을 사용하며 배열의 한 원소 [i,j]는 1 또는 0값을 노드 <i,j>가 Ξ_T에 속해 있는지 아닌지의 여부에 따라서 결정한다. FF알고리즘은 coverage 배열을 좌에서 우로 그리고 위에서 아래로 값이 0인 원소 [i,j] 찾을 때까지 검색한다. BF 알고리즘은 T를 배정 받기에 충분히 큰 모든 자유로운 영역들을 검색하여 사용중인 이웃하는 노드들을 수가 가장 큰 한 영역을 선택한다. 시뮬레이션 결과에 의하면 FF알고리즘이 BF알고리즘보다 성능면에서 조금 우수하다. 비록 이 두 가지의 알고리즘이 앞서 설명한 FS알고리즘의 할당실패 문제점을 해결함으로써 향상시켰지만 두 알고리즘 모두 진입하는 태스크의 오리엔테이션을 $T(w,h)$에서 $T(h,w)$로 회전시켰을 때 자유로운 부분메쉬를 인식할 수 없다. 게다가 알고리즘의 시간오버헤드는 이차원적 busy와 coverage배열의 비트맵을 조작해야 한다는 이유에서 크다.

④ *Adaptive Scan* (AS) 알고리즘 : Ding과 Bhuyan [11]은 두 가지의 주된 사상을 기본으로 하여 할당의 효율을 더욱 향상시키기 위하여 알고리즘을 제안하였다. 첫째로 이 알고리즘은 FS알고리즘의 sliding동작을 scanning동작으로 치환하였다. 현재노드가 자유로운 부분메쉬의 기준으로 선택될 수 없다면 그 노드는 Ξ_T 또는 Δ_T의 어느 부분메쉬들의 한 노드여야 한다. 만일 x_{max}가 이러한 부분메쉬들의 x좌표값들 중 최대치를 나타낸다면 이 방법에서 고려하는 다음 노드는 바로 그 행의 $x_{max}+1$에 위치한다. 현재의 행에서 기준을 찾지 못하면 scanning은 다음 행에서 진행되고 그와 같은 방식의 절차가 이루어 진다. 둘째로 진입하는 태스크의 오리엔테이션을 회전시켜서 $T(w,h)$에서 $T(h,w)$로 바꾸어 대체의 자유로운 부분메쉬를 조사한다. 그러면 원래의 호스트 메쉬에서는 할당실패 될 태스크에 대하여 오리엔테이션의 회전에 의한 수용이 가능하게 된다. 이 알고리즘은 T의 고정되지 않은 오리엔테이션과 scanning하는 동안의 폭으로 인하여 FS 알고리즘을 향상시켰으나 그래도 많은 검색시간이 걸린다.

⑤ Sharma와 Pradhan의 알고리즘[12] : 존재하는 모든 자유로운 부분메쉬들 중에서 인접하여 있으며 이미 할당된 가장 많은 노드들을 갖는 *최적 후보부분메쉬*가 T에 할당된다. 최적후보 부분메쉬를 찾기 위하여 이 알고리즘은 이미 할당된 부분메쉬들의 네 모서리들을 조사하고 또 전체메쉬의 네 모서리도 조사한다. 할당된 부분메쉬들의 한 모서리에 있는 한 후보 부분메쉬가 또 다른 할당된 부분메쉬와 겹치게 되면 후보 부분메쉬는 그 할당된 부분메쉬의 주변을 따라서 옮겨진다. 이는 할당된 부분메쉬들을 가능하면 인접되게 배치하여 외부 단편화의 가능성이 줄어 들게끔 하기 위함이다. 이 방법은 또한 완전한 부분메쉬 인식능력을 보장한다. 할당시간은 전체메쉬의 크기와는 별상관 없이 변하고 AS 알고리즘보다 적은 시간을 요한다는 것이 컴퓨터 시뮬레이션을 통하여 확인되었다.

진입하는 태스크에게 부분메쉬를 할당하기 위한 기존의 알고리즘들로부터 알 수 있듯이 본 논문에서 제안하는 알고리즘은 최소한의 시간상 오버헤드로 부분메쉬의 완전한 인식을 주된 목표로 한다. 다음 절에서 제안되는 알고리즘을 소개한다.

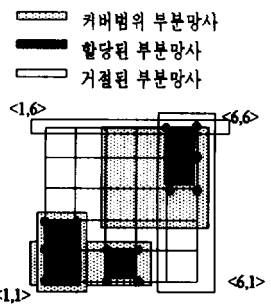
3. 태스크할당 알고리즘

제안하는 알고리즘의 주된 접근방법이 처음에 소개되고 그에 따라서 필요한 자료구조와 자세한 절차들이 연이어서 서술된다.

3.1 주요 접근방법

아래의 태스크할당 시나리오를 가정한다. (그림 3)에서 B={<1,1,2,3>, <3,1,4,2>, <5,4,6,6>} 그리고 T(3,2)라 가정한다. 그러면 Ξ_T={<1,1,2,3>, <1,1,4,2>, <3,3,6,6>}이고 Δ_T={<5,1,6,6>, <1,6,6,6>}이다. AS알고리즘과 같은 전형적인 할당 알고리즘에서처럼 노드들은 ll노드 <1,1>부터 그들이 어느 커버범위 부분메쉬에 포함되는지 조사가 시작된다. 노드 <1,1>이 두 개의 커버범위

부분메쉬 <1,1,2,3>와 <1,1,4,2>에 포함되고 하나의 할당된 부분메쉬 <1,1,2,3>에 포함될 때 4가 부분메쉬들의 가장 큰 x좌표 값이므로 다음의 조사대상 노드는 <5,1>이 된다. 이 노드가 Δ_T의 한 부분메쉬에 포함되면 그 다음 행의 가장 좌측 노드인 <1,2>가 조사된다. 둘째 행의 상황이 첫째 행과 정확히 같으므로 동일한 작업이 반복되고 다음 행의 노드인 <1,3>이 조사된다. 노드 <1,3>은 단 하나의 커버범위 부분메쉬 <1,1,2,3>에 속하므로 <3,3>이 조사되고 이는 다시 부분메쉬 <3,3,6,6>에 포함됨을 알 수 있다. 마지막으로 노드 <1,4>가 점검되고 Ξ_T와 Δ_T에 포함되지 않음이 확인된다. 따라서 부분메쉬 <1,4,3,5>가 T(3,2)에 할당된다. 이 시나리오에서 <1,1>, <5,1>, <1,2>, <5,2>, <1,3>, <3,3>, <1,4>의 일곱 개의 노드들이 Ξ_T 또는 Δ_T에 속하는 어느 부분메쉬에 포함되는지 순차적으로 조사되었다.



(그림 3) T(3,2)의 할당 예제

제안하는 알고리즘의 주된 사상은 B로부터 Ξ_T가 구성될 때 각 행의 정보를 수집함으로써 어느 한 행에 T를 수용할 수 있는 자유로운 부분메쉬의 기준이 될 노드가 있는가를 신속히 결정하는 것으로, 그 행에 있는 모든 노드들이 B또는 Ξ_T에 포함되는지를 일일이 검사하는 작업들을 배제한다. 게다가 AS알고리즘과 [12]에 소개된 열을 따라서 행하여지는 점검작업을 방지할 수 있다. 결과적으로 할당시간과 대기로 인한 지연시간이 기존의 알고리즘보다 훨씬 줄어들고, 하드웨어 오버헤드는 기존의 그것과 비슷한 수준이다. 이제부터 이 알고리즘을 Quick Allocation (QA)알고리즘이라 하겠다.

3.2 Last_Covered 알고리즘

여기에서는 전체메쉬의 각 행에서 가장 우측에 위치한 커버된 노드의 x좌표를 보관하는 일 차원 배열 last_covered를 소개한다.

정의 6 : *세그먼트*는 전체 메쉬시스템의 한 행에서 연속적인 노드들의 나열을 나타낸다. 한 세그먼트가 소속해 있는 행의 가장 좌측의 노드들로부터 시작하고 그 세그먼트의 모든 노드들이 Ξ_T의 어느 부분메쉬에 포함된다면 그 세그먼트를 *커버된 세그먼트*라 한다. 배열의 한 원소인 last_covered[j] ($1 \le j \le b'-1$)는 전체메쉬의 j번째 행에서 커버된 세그먼트의 마지막 노드의 x좌표 값을 갖는다. 여기서, 거절된 부분메쉬는 T(w,h)에 대하여 δ_{TH} =<1,b',a,b>, b'=b-h+2이다. b'부터 메쉬의 가장 위쪽인 b까지의 행들은 이미 할당된 부분메쉬들의 위치와는 상관없이 T를 수용할 수 없으므로 배열검색에서 제외된다. 원소 last_covered[j]는 j번째 행에 커버된 세그먼트가 존재하지 않을 경우 0값으로 고정된다. 예를 들어 (그림 3)에서 last_covered[j] (j=1,2,3,4,5)는 각각 4,4,6,0,0을 갖는다. 이들이 결정되는 과정은 다음과 같다.

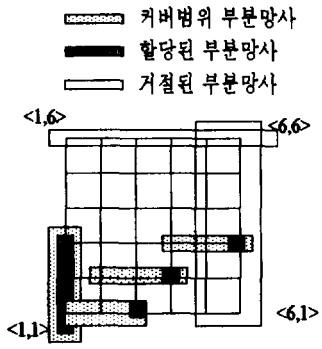
Procedure Last_covered
Step 1 last_covered[j] ($1 \le j \le b'-1$)←0;
Step 2 For each β <x,y,x',y'>
$\xi_{\beta,T}$ <x_c,y_c,x',y'>을 결정한다;
Step 3 $\xi_{\beta,T}$들을 x_c에 대한 올림차순으로 정렬한다;
Step 4 For each $\xi_{\beta,T}$(x_c의 값이 가장 작은 부분메쉬로부터 시작하여)
if (y_c < b')
For each row j'($y_c \le j \le \min(y',b'-1)$)
if ($x_c \le$ last_covered[j]+1 $\le x'$)
then last_covered[j]←x';

여기서 β_1=<5,4,6,6>, β_2=<1,1,2,3>, β_3=<3,1,4,2>, T(3,2), 그리고 b'-1=5이다. Step 1에서 last_covered[j] (j=1,2,3,4,5)=0이다. Step 2와 3에서 세 개의 할당된 부분메쉬 β_1, β_2, β_3에 의하여 세 개의 커버범위 부분메쉬 $\xi_{\beta1,T}$, $\xi_{\beta2,T}$, $\xi_{\beta3,T}$가 <3,3,6,6>, <1,1,2,3>, <1,1,4,2>로 결정되고, 그들은 $\xi_{\beta2,T}$, $\xi_{\beta3,T}$, $\xi_{\beta1,T}$의 순서로 정렬된다. Step 4에서는 $\xi_{\beta2,T}$로부터 last_covered[j] (j=1,2,3)의 값이 2로 바뀌고, $\xi_{\beta3,T}$로부터 last_covered[j] (j=1,2)의 값이 4로, $\xi_{\beta1,T}$로부터 last_

covered[3]의 값이 6으로 바뀐다. 그러나 last_covered[j] (j=4,5)는 변하지 않는다. 첫 행부터 다섯번째 행까지의 last_covered의 값은 4,4,6,0,0이다.

Lemma 3.1 : 노드 <last_covered[j]+1, j>가 거절된 부분메쉬에 포함되지 않는다면 언제나 j번째 행에서의 가장 왼쪽에 위치한 커버되지 않은 노드를 나타낸다.

증명 : 커버되지 않은 가장 왼쪽에 위치한 노드의 상태에 관해서 세 가지 가능한 경우들이 있다. 첫째경우 : 어떠한 거절범위 부분메쉬도 j번째 행의 가장 왼쪽의 노드를 포함하지 않는다. 두번째경우 : 단 하나의 커버범위 부분메쉬가 j번째 행을 커버하고 가장 왼쪽의 노드를 포함한다. 세번째경우 : 적어도 두개의 커버범위 부분메쉬 ξ_1과 ξ_2가 j번째 행을 커버하고 적어도 한 커버범위 부분메쉬, 예를 들어 ξ_1가 그 행의 가장 왼쪽에 위치한 노드를 포함한다. 첫째경우에서 처음부터 last_covered[j]+1은 1값을 갖는다. 만일 j번째 행을 커버하는 어떤 커버범위 부분메쉬가 존재한다면 그의 x_c 값은 1보다는 크다. 그리하여 노드 <last_covered[j]+1, j> 즉 <1,j>는 분명히 j행에 있는 커버되지 않은 노드를 나타낸다. 두번째경우의 증명은 분명하다. 세번째 경우를 위해서 일반성을 상실하지 않으며 $x_c(\xi_1) \le x_c(\xi_2)$과 $x'(\xi_1) < x'(\xi_2)$을 가정한다. 이는 두개의 할당된 부분메쉬 ξ_1와 ξ_2가 같은 행에서 중복되어질 수 없기 때문에 $x'(\xi_1)$ $x'(\xi_2)$이다. 또한 last_covered[j]는 ξ_1를 처리하여 값이 일단 결정되었음을 가정한다. (그림 4)의 ξ_1 <1,1,1,3>과 다른 세 개의 커버범위 부분메쉬가 존재함을 참조하여, ξ_1과 ξ_2의 세 가지 가능한 상태를 고려한다. 처음 행에서 볼 수 있는 중첩된 상태, 둘째 행에서 볼 수 있는 인접한 상태, 그리고 셋째 행에서 볼 수 있는 분리된 상태가 그 세 가지임을 알 수 있다. ξ_2가 ξ_1과 중첩된 경우엔 $x_c(\xi_2) <$ last_covered[j]+1$\le x'(\xi_2)$이므로 last_covered[j]는 $x'(\xi_2)$로 바뀐다. 이제 last_coverd[j]+1은 확실히 j행에서 가장 왼쪽에 있는 커버되지 않은 새로운 x좌표이다. ξ_2가 ξ_1과 인접하고 $x_c(\xi_2)$=last_covered[j]+1이며 또한 $x_c(\xi_2) \le x'(\xi_2)$이므로 다시 last_covered[j]는 $x'(\xi_2)$로 바뀐다. 바뀐 last_covered[j]+1은 j행에서 새로이 가장 왼쪽에 위치한 커버되지 않은 노드의 x좌표이다. ξ_2와 ξ_1가 인접하지않고 떨어져 있으면 $xc(\xi_2) >$ last_covered[j]+1이므로 last_covered[j]의 값은 바뀌지 않으므로 last_covered[j]+1은 계속해서 j행에서 가장 왼쪽에 위치한 커버되지 않은 노드의 x좌표를 갖는다.



(그림 4) T(3,1)에 대한 거절범위 부분망사

3.3 태스크할당 및 할당해제

이 절에서는 last_covered를 이용한 할당절차에 대하여 서술한다.

Lemma 3.2 : B에 있는 한 부분메쉬에 포함된 노드는 Ξ에 있는 한 부분메쉬에도 포함된다.

증명 : 이것은 정의3으로부터 명백하다.

위에서 언급된 Lemma 3.2에 의하여 어느 한 노드가 이미 할당된 부분메쉬에 포함이 되는가의 여부를 확인할 필요가 없다. 단지, 그 노드가 커버범위 부분메쉬에 포함이 되는가 만을 확인하며 이는 last_covered의 정보로부터 제공된다. 각 T(w,h)에 대하여 M(a,b)의 할당절차는 아래와 같이 주어진다.

Procedure Allocation

Step 1 flag false;
/*이 flag는 orientation의 변화를 나타낸다.*/
Step 2 If (자유로운 프로세서의 개수 <w×h)
then go to Step 7;
Step 3 T의 orientation을 아래와 같이 정하고 거절된 집합을 결정한다;
If (flag=flase)
then T←T(w,h), a'←a-w+2, b'←b-h+2;
else T←T(h,w), a'←a-h+2, b'←b-w+2;
Step 4 현재의 B와 T에 대하여 $\Xi_{B,T}$ 및 last_covered를 3.2에서 서술한 방법으로 결정한다;
Step 5 j←1;
While (j < b' AND last_covered[j]+1 a')
/* j행에서 자유로운 부분메쉬를 찾지 못하는 동안*/

```
        j j+1;
      if (j=b')
      /* 현재의 orientation으로 자유로운 부분메쉬
         를 찾지 못한다면 */
        then
         if (flag=flase)
           then flag true and go back to Step 3;
           else go to Step 7;
      else /* 부분메쉬를 찾은 경우 */
         i←(last_covered[j]+1) and go to Step 6;
Step 6 If (flag=flase)
         then
           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;
```

Orientation의 변화에 따라서 부분메쉬의 할당을 고려하는 단계, 자유로운 프로세서의 개수와 태스크에서 요청한 개수를 비교하는 단계가 각각 step 1과 step 2이다. T의 orientation을 정하고 거절된 집합을 결정하는 step 3과 현재의 B와 T에 대하여 Ξ_{BT} 및 last_covered를 3.2에서 서술한 방법으로 결정하는 step 4가 종료되면 last_covered를 이용한 할당절차가 step 5에 의하여 수행된다. 배열 last_covered를 이용 while문을 수행하여 어느 행에 자유로운 부분메쉬가 있는지 결정되고, 따라서 현재의 orientation으로 자유로운 부분메쉬를 찾지 못한 경우와 찾은 경우로 나뉜다. 자유로운 부분메쉬를 찾지 못한 경우에는 다른 orientation을 고려하던가 step 7에 의하여 대기상태로 들어가고, 부분메쉬를 찾은 경우에는 기준을 결정하고 다음 단계인 step 6로 간다. Step 6에서는 orientation에 따라서 부분메쉬 S를 결정하고, 이를 T에 할당하고, B에 S를 첨가한 후 종료한다.

예를 들어서, (그림 3)에서 a'=5이고 b'=6라 하자. 여기서 last_covered[j] $(1\le j\le 3)+1$ a'이므로, j는 4가 된다. 그러면 last_covered[4]+1<a'이다. 이제 노드 <1,4>는 부분메쉬 <1,4,3,5>의 기준이 될 수 있으므로 T에 할당된다. 이 알고리즘은 3.1절의 예제와 비교할 때 많은 검색시간을 절약할 수 있다.

정리 3.1 : 제안하는 QA알고리즘은 완전한 부분메쉬의 인식을 보장한다.

증명 : 본 알고리즘에서 한 노드가 자유로운 부분메쉬의 기준이 될 수 없다면, 그 노드는 Ξ_T 또는 Δ_T에 포함되어야 한다. 할당절차의 Step 3에서 Δ_T에 포함된 행과 열들은 배제된다. 그러므로 임의의 행에 존재하는 어느 한 노드가 Ξ_T에 포함되어 있지 않다는 것 만 보일 수 있으면 우리가 원하는 것을 찾게 된다. Step 5에서는 나머지 행들에 대한 전수 검색이 이루어 지는데 이미 Lemma 3.1에서 증명된 것처럼 <last_covered[j]+1,j>는 j행의 가장 좌측에 있는 노드를 나타내므로, 그 노드 및 그에 대응하는 자유로운 부분메쉬를 찾게 되는 것은 보장된다.

부분메쉬의 할당해제에 관한 절차는 B로부터 β를 삭제하는 것으로써 매우 간단하다. AS알고리즘과 [12]의 알고리즘의 할당해제 방법은 동일하고 간단하다. 다음 절에서 제안된 알고리즘의 성능을 평가한다.

4. 성능평가

이 절에서는 제안된 알고리즘이 평균할당시간, 대기지연, 하드웨어 오버헤드에 관하여 평가된다. 이 작업은 먼저 알고리즘의 시간복잡도와 필요로 하는 하드웨어의 평가를 수행한다. 그리고 컴퓨터 시뮬레이션에 의하여 성능을 평가한다. 또한 제안된 알고리즘과 기존의 완전한 부분메쉬 인식능력을 갖춘 AS알고리즘 및 [12] 알고리즘을 비교한다.

4.1 할당 및 할당해제의 시간복잡도

이미 언급된 바와 같이 AS알고리즘, [12]알고리즘, 및 본 논문에서 제안되는 QA알고리즘의 할당해제는 단지 B에서 β를 삭제하는 작업을 포함하므로 시간복잡도는 (1)이다. 그러나 일반적으로 완전한 부분메쉬의 인식은 프로세서 할당작업을 매우 복잡하게 만든다. 전체가 a×b인 메쉬시스템에서의 할당된 부분메쉬의 수를 N_B라 하자. AS알고리즘의 태스크할당에 관한 시간복잡도는 최악의 경우 ab개의 노드를 모두 검색할 필요가 있으므로 $O(abN_B)$이다. 또한 각 노드는 이미 할당된 부분메쉬나 커버범위 부분메쉬에 포함이 되는지 점검을 위하여 $O(N_B)$의 시간이 걸린다. 유의사항으로써 최악의 경우 한 행에서의 검색시간은 비록 aN_B이나 실제로 평균의 경우에는 AS접근방법에 의하여 훨씬 작다. [12]에서 소개된 알고리즘은 그들의 분석에 의하면 $O(N_B^3)$의 할당시간을 갖는다.

이제 제안된 알고리즘의 시간복잡도를 분석한다. 분명하게 Step 1,2,3,6,7은 각각 단지 (1) 걸린다. Step 5는 내부의 while loop이 b'-1번 수행되고 최악의 경우엔 b'=b이므로O(b) 걸린다. Step 4의 last_covered를 구성하는데 가장 많은 시간을 필요로 하고 결국은 본 알고리즘의 시간복잡도를 결정한다. 초기화는 O(b) 걸리고, 모든 $\xi_{\beta,T}$를 결정하는데 (N_B), $\xi_{\beta,T}$ 모두를 정렬하는데 $O(N_B\log_2 N_B)$가, last_covered[j]가 필요한 값을 갖는데 $O(N_Bb)$의 시간이 걸린다. 측정 가능한 크기의 태스크들과 메쉬구조에서 b는 $\log_2 N_B$를 능가하므로 $O(bN_B)$가 Step 4의 시간 복잡도이자 본 알고리즘의 복잡도이다.

본 알고리즘의 시간복잡도는 AS알고리즘보다 훨씬 낮으며 이는 각 행이 자유로운 부분메쉬의 기준이 될 노드가 존재하는가에 대한 정보를 미리 갖게끔 하는 반면 AS알고리즘에선 각 행의 여러 노드들이 이미 할당된 부분메쉬나 거절범위 부분메쉬에 포함되는지를 점검해야 하기 때문이다. [12]에서 제안된 알고리즘의 할당시간은 메쉬의 크기에 따라서는 그리 변하지 않으나, 할당된 부분메쉬의 개수에 비례한다. [12]의 할당시간은 본 논문에서 제안된 알고리즘보다 최악 및 평균의 두 경우에서 크게 나타난다. 최악의 경우에 N_B는 ab와 같으므로 [12]의 할당시간은 $O(a^3b^3)$인 반면 새로운 알고리즘은 $O(ab^2)$이다. 평균의 경우 [12]의 할당시간은 새로 제안된 알고리즘보다 크다는 것이 다음 절에서 설명하는 컴퓨터 시뮬레이션의 결과에 의하여 확인된다. <표 1>은 새로 제안된 알고리즘과 기존의 알고리즘들을 시간복잡도, 하드웨어 오버헤드, 공간복잡도에 대하여 비교한다.

〈표 1〉 제안된 알고리즘을 포함한 다양한 태스크할당 알고리즘들의 비교

알고리즘	할당시간	할당해제	공간복잡도	완전인식	내부단편화
2DB	$\theta(\log_2 a)$	$O(ab)$	$O(ab)$	불가능	가능
FS	$O(abN_B)$	$\theta(1)$	$\theta(N_B)$	불가능	불가능
FF	$O(ab)$	$O(ab)$	$\theta(ab)$	불가능	불가능
BF	$O(ab)$	$O(ab)$	$\theta(ab)$	불가능	불가능
AS	$O(abN_B)$	$\theta(1)$	$\theta(N_B)$	가능	불가능
[12]	$O(NB^3)$	$\theta(1)$	$\theta(N_B)$	가능	불가능
QA	$O(bN_B)$	$\theta(1)$	$\theta(b+N_B)$	가능	불가능

주의사항 2DB : a=b=2^i의 정방형 메쉬에 적용

4.2 시뮬레이션 결과

이 절에서는 새로이 제안된 알고리즘과 AS 및 [12]의 알고리즘들이 컴퓨터 시뮬레이션을 통해서 얻어진 결과자료에 의하여 비교된다. 시뮬레이션은 사건 구동 방식에 따르고 사건은 태스크들의 할당 및 할당해제가 되며 할당과 할당해제의 수행에 필요한 실제의 CPU시간을 고려한다. 시뮬레이션은 크기가 10×10인 메쉬에서부터 80×80까지의 메쉬까지 그래프를 그리기에 간단한 정방메쉬를 가정하고 수행되었다. 모든 시뮬레이션은 95%의 신뢰구간에서 3%의 오차범위를 허용하였다. 시뮬레이터는 SUN 4/490워크스테이션에서 C언어로 개발되었다. 이 논문에서 보고되는 모든 결과는 다섯번의 독립된 수행으로부터 모아진 것이다.

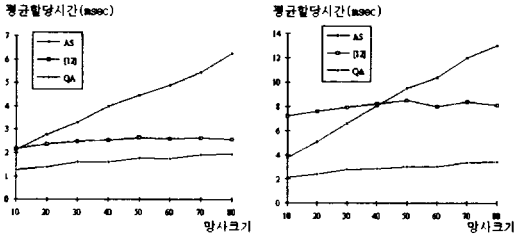
비교의 공정성을 위해서 이전에 [9,11,12]서 사용된 동일한 시뮬레이션 모델을 이용한다. 태스크할당은 전체메쉬에서 분리되어 있는 태스크 분배기의 역할을 하는 프로세서에 의하여 수행된다. 시스템의 초기에 전체메쉬는 자유롭고, 3000개의 태스크가 생성되어 태스크 분배기의 큐에 위치한다. 각각의 태스크는 상주시간에 관한 요구가 있고 이는 5에서 10단위시간 사이에서 균등하게 분포됨을 가정한다. 여기서 단위시간은 이전 알고리즘들에서 사용한 것처럼 사건 구동방식(event-driven)시뮬레이션을 수행하는 과정에서의 시간단위로 시뮬레이션의 결과로써 얻는 성능 측정치들의 실제 시간인 second (sec) 또는 millisecond (msec)과는 다른 시뮬레이션상의 시간이다. 이와 같이 태스크의 상주시간 결정방법을 비교하고자 하는 다른 알고리즘들과 같게 함으로써 태스크 할당 알고리즘들간의 성능을 측정하고 비교하는데 영향을 주지 않게 된다. 태스크들은 매 단위시간에 도착됨을 가정한다. 여기서 언급되는 단위시간은 충분히 큰 시간으로써 태스크 분배기가 전체메쉬를 판독 및 점검하는데 필요한 시간은 무시할 수 있다. 진입하는 태스크의 너비(폭)와 높이를 나타내는 한 변의 길이는 균등분포 또는 지수분포를 가정한다. 균등분포에 있어서 진입하는 태스크의 한 변의 길이는 균등하게 1과 메쉬 변의 길이(L) 사이에서 균등하게 분포된다. 지수분포에 대하여는 그 평균이 L+1값의 반으로 선택된다. [1, L+1)의 범위를 벗어나는 값들은 버려진다. 태스크들의 너비와 높이는 분리되어 위에서 언급된 분포에 근거하여 생성된다.

태스크 분배기는 First-Come-First-Serve (FCFS)의 원칙에 따라 언제나 대기 큐의 첫번째 태스크를 위한

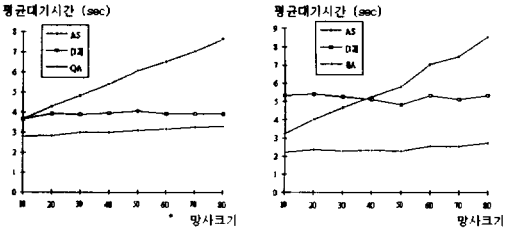
자유로운 부분메쉬를 찾는다. 자유로운 부분메쉬를 찾는데 실패한다면 분배기는 할당을 위하여 단순히 이미 할당된 부분메쉬의 해제를 기다린다. 태스크에 부분메쉬가 할당되면 그 태스크는 큐로부터 삭제되고 다음에 위치한 태스크가 다음의 단위시간에서 처리된다.

시뮬레이션을 통하여 다음의 성능 측정치들을 얻는다; 1) 평균할당시간 : 큐의 헤드에 있는 태스크에게 부분메쉬를 할당하는 데 걸리는 시간, 2) 평균대기시간 : 시스템으로 진입한 태스크가 큐의 헤드에 위치하게 되어 부분메쉬가 할당될 때까지의 시간, 3) 이미 할당된 부분메쉬의 평균개수. 다양한 크기의 전체메쉬에 대한 평균할당시간이 (그림 5)에서 균등분포 및 지수분포를 갖는 부분메쉬 변의 길이에 따라서 각각 그려진다. (그림 6)에서는 평균 대기시간이 그려진다. <표 2>는 할당된 부분메쉬의 평균개수를 나타낸다. 다음의 열거한 내용들은 그림과 표로부터 관찰된 것들이다.

- 제안된 알고리즘의 평균할당시간은 전체 호스트메쉬의 크기가 80×80까지 증가될 때 AS알고리즘 보다 25%정도 낮게 측정된다. 기본적으로 같은 결과가 대기시간에 있어서 구해진다.
- 제안된 알고리즘은 [12]의 알고리즘보다 일관성 있게 나은 결과를 보이고, AS알고리즘과의 차이와는 달리 호스트메쉬의 크기에 상관없이 거의 변하지 않는다. 다시 말해서 제안된 알고리즘과 [12]의 알고리즘 둘 다 전체메쉬의 크기에 관계없이 일정하게 수행된다.
- 할당시간은 진입하는 태스크들의 모양에 따라서 민감한 결과를 갖는다. 지수적 분포된 변의 길이를 갖을 때의 할당시간은 균등분포인 경우보다 약 두 배가량 더 걸린다. 이는 부분메쉬의 모양이 더 임의적이면 외부단편화가 될 확률이 더 높아지므로 직관적으로 예측된다. 비교된 알고리즘들 중에서 [12]의 방법은 그의 큰 중가로 보았을 때 가장 민감한 알고리즘으로 판단되고 본 논문에서 제안된 QA 알고리즘은 분명히 가장 덜 민감한 알고리즘이다.
- 대기시간은 할당시간보다 분포에 있어서 덜 민감하다.
- 할당된 부분메쉬의 평균 개수는 전체메쉬의 크기가 증가해도 별로 증가하지 않는다. 예상했던 것처럼 지수적으로 분포된 변의 길이를 갖을 때의 개수는 균등분포인 경우보다 약 두 배가량 더 많다. 본 알고리즘과 AS알고리즘은 같은 개수의 할당된 부분메쉬를 갖으며 이는 비록 두 알고리즘의 실제 할당 시간은 서로 달라도 논리적인 작동방법이 정확히 같음을 알 수 있다. [12]의 알고리즘은 무시할 수 있을 정도로 더 많은 개수의 부분메쉬들을 할당한다



(가) 균등분포된 변의 길이 (나) 지수분포된 변의 길이



(가) 균등분포된 변의 길이 (나) 지수분포된 변의 길이

(그림 5) 망사 크기에 관한 평균할당시간

〈표 2〉 할당된 부분메쉬들의 평균개수

전체메쉬 크기	균등분포		지수분포	
	AS&QA	[12]	AS&QA	[12]
10×10	1.79	1.84	3.97	4.09
20×20	1.89	1.95	4.16	4.30
30×30	1.90	1.98	4.28	4.44
40×40	1.98	2.05	4.40	4.52
50×50	1.95	2.03	4.43	4.58
60×60	1.94	2.03	4.29	4.41
70×70	1.95	2.02	4.39	4.55
80×80	1.97	2.05	4.34	4.49

5. 결 론

본 논문에서 이차원적 메쉬구조에 대하여 완전한 부분메쉬 인식능력을 갖는 효율적인 할당알고리즘이 제안되었다. 종합적인 컴퓨터 시뮬레이션 결과에 의하면 제안된 알고리즘의 할당시간과 대기지연시간은 이전의 가장 효과적인 알고리즘보다 할당시간에 있어서는 1.4배에서 3.5배 정도까지가, 그리고 대기지연시간은 1.2

배에서 2.5배정도 빠르면서 정확히 같은위치의 부분메쉬를 할당하고 있음(전체메쉬의 이용율이 같음)을 알 수 있다. 이는 자유로운 부분메쉬의 검색과정에서 last_covered배열을 사용하는 접근방법의 효율성으로 특징지울 수 있다.

비록 제안된 방법이 이차원적 호스트메쉬에서 태스크할당 성능을 상당히 향상시켰으나 아직 외부단편화는 해결되지 않았다. 효과적인 외부단편화의 방지 또는 완화가 지속적으로 연구되어질 필요가 있다. 태스크의 재배치를 통한 접근방법[13] 또는 작업 스케쥴링을 이용하는 방법[14-17]이 해결책중의 하나가 될 것이다.

이차원적 메쉬 시스템은 특히 확장성 및 라우팅 효율면[18]에 있어서 다른 토폴로지들과 차별화할 수 있는 많은 바람직한 특징들을 갖고 있으므로 병렬구조로써 가장 장래성 있는 구조라고 할 수 있다. 이미 2D메쉬를 완전히 이용하는 여러 응용 분야들이 존재한다. 여러 운영조건과 함께 2D메쉬에서의 보다 효과적인 스케쥴링 방식 및 할당알고리즘이 보다 깊이 연구되어질 것이다.

참 고 문 헌

[1] P. Muzumdar, "Evaluation of on-chip static interconnection networks," IEEE Trans. Computers, Vol.C-36, pp.365-369, March 1987.

[2] G. Randade 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.477-482, Aug. 1987.

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

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

[5] Alverson et al., "The Tera computer system," Proc. 1990 Int'l Conf. on Supercomputing, pp.1-6, 1990.

[6] J. Kim, C.R. Das, and W. Lin, "A Top-down Processor Allocation Scheme for Hypercube Computers," IEEE Trans. on Parallel and Distr. Sys., Vol.2, pp.21-30, Jan. 1991.

[7] P.J. Chuang and N.F. Tzeng, "Dynamic Processor Allocation in Hypercube Computers," Proc. 17th Annual Int'l Symp. on Comp. Arch., May 1990.

[8] K. Li and K.H. Cheng, "A two-dimensional buddy system for dynamic resource allocation in a partitionable mesh connected system," Journal of Parallel and Distributed Computing, Vol.12, pp.79-83, May 1991.

[9] P.J. Chuang and N.F. Tzeng, "An efficient submesh allocation strategy for mesh computer systems," Int'l Conf. on Distributed Computing Systems, pp.256-263, Aug. 1991.

[10] Y. Zhu, "Efficient processor allocation strategies for mesh-connected parallel computers," Journal of Parallel and Distributed Computing, Vol.16, pp.328-337, Dec. 1992.

[11] J. Ding and L.N. Bhuyan, "An adaptive submesh allocation strategy for two-dimensional mesh connected systems," Int'l Conf. on Parallel Processing, pp.II-193-200, Aug. 1993.

[12] D.D. Sharma and D.K. Pradhan, "A fast and efficient strategy for submesh allocation in mesh-connected parallel computers," Symp. on Parallel and Distributed Processing, pp.682-689, Dec. 1993.

[13] H.Y. Youn, S.M. Yoo, and B. Shirazi, "Task relocation for two-dimensional meshes," Symp. on Parallel and Dist. Comp. Systems, pp.230-235, Oct. 1994.

[14] Y.K. Chu, I.L. Yen, and D.T. Rover, "Incorporating job scheduling for processor allocation on two-dimensional mesh-connected systems," Symp. on Parallel and Dist. Comp. Systems, pp.124-129. Oct. 1994.

[15] D.D. Sharma and D.K. Pradhan, "Job scheduling in mesh multicomputers," Int'l Conf. on Parallel Processing, pp.II-251-258, Aug. 1994.

[16] H.Y. Youn, H. Choo, S.M. Yoo, B. Shirazi, "Dynamic task scheduling and allocation for 3D torus multicomputer systems," Int'l Conf. on Parallel Processing, pp.III-199-206, Aug. 1996.

[17] H. Choo, H.Y. Youn, G.-L. Park, B. Shirazi, "Efficient processor allocation scheme for multi dimensional interconnection networks," Int'l Conf. on Parallel Processing, pp.114-117, Aug. 1997.

[18] K. Hwang, Advanced Computer Architecture : Parallelism, Scalability, Programmability, McGraw-Hill, 1993.

추 현 승

e-mail : choo@yurim.skku.ac.kr

1988년 성균관대학교 이과대학 수학과 졸업(학사)

1990년 텍사스 주립대(달라스) 전자계산학과(공학석사)

1996년 텍사스 주립대(알링턴) 전산공학과(공학박사)

1997년~1998년 특허청 심사4국 컴퓨터심사담당관실 심사관

1998년~현재 성균관대학교 전기전자 및 컴퓨터공학부 전임강사

관심분야 : ATM, 병렬 및 분산 처리, 알고리즘 해석, 고속통신망 등

박 경 린

e-mail : glpark@cheju.cheju.ac.kr

1986년 중앙 대학교 전자계산학과 졸업(학사)

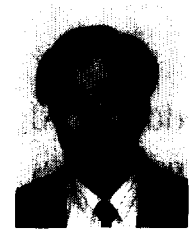
1988년 중앙 대학교 전자계산학과 대학원(공학석사)

1992년 텍사스 주립대(알링턴) 전산공학과 대학원(공학석사)

1997년 텍사스 주립대(알링턴) 전산공학과 대학원(공학박사)

1998년~현재 제주대학교 자연과학대학 전산통계학과 전임강사

관심분야 : 분산/병렬 처리 시스템, 오류 허용 시스템, 성능 평가 등



유 성 무

e-mail : yoo_seong-moo@colstate.edu

1973년 서울대학교 경제학과 졸업(학사)

1989년 텍사스 주립대(알링턴) 전산공학과 대학원(공학석사)

1995년 텍사스 주립대(알링턴) 전산공학과 대학원(공학박사)

1995년~현재 미국 콜럼버스주립대 전산학과 조교수

관심분야 : 분산 및 병렬 처리, 컴퓨터구조, 운영체제, 인터넷 상거래 및 보안등