3차원 토러스 구조를 갖는 멀티컴퓨터에서의 동적 작업 스케줄링 알고리즘

추 현 승[†] · 윤 희 용[††] · 박 경 린[†††]

요 약

멀티컴퓨터 시스템은 많은 연산 노드들을 이용함으로써 높은 성능을 얻는다. 다차원 메쉬(mesh)는 단순함과 효율성 때문에 멀티컴퓨터 구조로 널리 이용되었다. 본 논문은 3차원 토러스(torus) 시스템을 위한 최초 적합(first-fit) 방법에 기반한 효율적인 프로세서 할당 알고리즘을 제안한다. 이 알고리즘은 CST(Coverage Status Table)을 이용하여 3차원 정보를 2차원 정보로 변형하므로써 프로세서 할당 시간을 최소화 한다. 종합적인 컴퓨터 시뮬레이션 결과는 제안한 방법이 최적 적합(best-fit)에 기반한 기존 방법들과 비교해서 프로세서 이용률은 비슷하면서, 프로세서 할당 시간이 항상 짧다는 것을 보여준다. 성능 차이는 입력 부하가 증가함에 따라 더욱 두드러진다. 다른 스케줄링 환경상에서 제안된 방법의 성능을 조사하기 위해서, 전형적인 FCFS 스케줄링 기법과 함께 non-FCFS 스케줄링 기법도 연구된다.

Dynamic Task Scheduling for 3D Torus Multicomputer Systems

Hyunseung Choo[†] · Hee Yong Youn[††] · Gyung-Leen Park[†††]

ABSTRACT

Multicomputer systems achieve high performance by utilizing a number of computing nodes. Multidimensional meshes have become popular as multicomputer architectures due to their simplicity and efficiency. In this paper we propose an efficient processor allocation scheme for 3D torus based on first-fit approach. The scheme minimizes the allocation time by effectively manipulating the 3D information as 2D information using CST (Coverage Status Table). Comprehensive computer simulation reveals that the allocation time of the proposed scheme is always smaller than the earlier scheme based on best-fit approach, while allowing comparable processor utilization. The difference gets more significant as the input load increases. To investigate the performance of the proposed scheme with different scheduling environment, non-FCFS scheduling policy along with the typical FCFS policy is also studied.

키워드 : 토러스(torus), 할당기(allocator), 멀티컴퓨터(multicomputer), 작업스케줄링(task scheduling)

1. 서 론

메시지 전달 방식을 기반으로하는 멀티컴퓨터에 있어서 노드들간의 상호연결을 위한 다양한 방법들이 연구되었다 [1, 2]. 1980년대 후반까지는 n CUBE[3]와 Caltech Cosmic[4]으로 대표되는 하이퍼큐브(hypercube) 구조가 널리 이용되었으나 확장면에서의 취약한 구조가 부각되었고, 1990년대 초 이후로 Intel Paragon[5]과 Intel /DARPA Touchstone Delta[6]로 대표되는 2차원 메쉬 구조가 단순함과 효율성으로 보편화되었다. 2차원 메쉬의 노드 차수는 하이퍼큐브 구조와는 달리 고정되어 있고, Wormhole라우팅[7]으로 멀티컴퓨터 내부의 두 노드사이에 통신 지연은 노드 거리에 비례하지 않는다. 최근에는 통신 지연의 현저한 감소로 인하여 Caltesh Mosaic[2], MIT J-machine[2], Intel TeraFLOP 슈퍼 컴퓨터와 같은 3차원 메쉬에 기반한 멀티 컴퓨터가 각광받고 있다.

2차원 메쉬를 위한 많은 스케줄링 기법과 할당 알고리즘이 제안되었지만, 3차원 토러스에 대해서는 그렇지 않다. 현재 Cray T3D에서 이용되는 할당 알고리즘[13]에서 각 할당되어진 부분메쉬의 차원은 2의 거듭제곱이다. 이것은 요구된 크기가 2의 거듭제곱이 아닐 경우 내부 단편화를 일으킨다. Qiao와 Ni[14]가 제안한 자유 리스트을 이용한 3차원 토러스에서의 할당 알고리즘은 내부 단편화가 없이 완전한 부분 메쉬의 인식이 가능하다. 그 정책은 최적 적합(best-fit)에 기반을 두는데 최초 적합(first-fit)보다 훨씬 많

† 종신회원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
†† 정 회 원 : 성균관대학교 전기전자 및 컴퓨터공학부 교수
††† 정 회 원 : 제주대학교 자연과학대학 전산통계학과 교수
논문접수 : 2000년 8월 12일, 심사완료 : 2001년 6월 8일

은 검색 시간을 필요로 한다.

본 논문은 3차원 토러스를 위한 최초 적합 기반의 빠르면서 효율적인 프로세서 할당 알고리즘을 제안한다. 제안한 방법은 토러스에 있는 3차원 정보CST(coverage status table)를 이용하여 효과적으로 2차원 정보로 변형하므로써 들어오는 작업에 대한 할당 시간을 최소화한다. 유사한 접근 방법이 [9]에서 제안된 2차원 메쉬 할당 정책에서 이용되었는데, 그것은 2차원 정보를 1차원으로 매핑시켰다. 종합적인 컴퓨터 시뮬레이션은 제안된 알고리즘의 프로세서 할당 시간이 최적 적합에 기반을 둔 기존 알고리즘보다 항상 작다는 것을 보여준다[14]. 성능 차이는 입력 노드수가 증가함에 따라 더 커지고 높은 부하에서 할당 시간이 기존 방법[14]의 약 1/3 정도로 작아진다. 다른 스케줄링 환경에서 제안된 방법의 성능을 조사하기 위해 전형적인 FCFS(first-come first-served) 스케줄링 기법과 non-FCFS 스케줄링 기법을 고려한다.

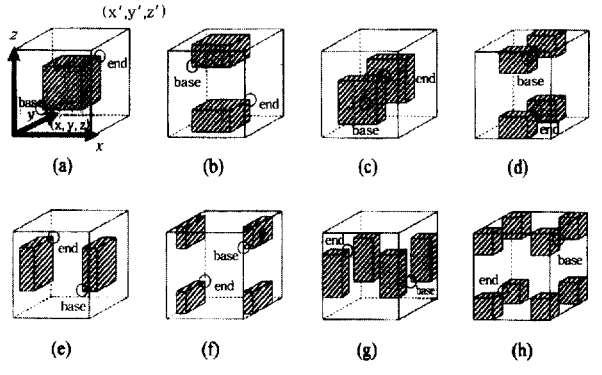
본 논문의 나머지 부분은 다음과 같이 구성된다. 2절에서는 논문에서 다뤄질 정의와 표기법이 소개된다. 기존의 프로세서 할당 알고리즘도 간략히 살펴본다. 3절에서 3차원 토러스에 대한 프로세서 할당이 제안된다. 4절에서 제안된 방법이 컴퓨터 시뮬레이션을 통해서 성능이 평가되고 기존 알고리즘들과 비교된다. 마지막으로 5절에서 본 논문을 결론 맺는다.

2. 정의와 표기법

기존 알고리즘들과의 일관성을 유지하기 위해 [14]에서 사용한 정의와 표기법을 이용한다. 3차원 토러스 $3DT(L, W, H)$인 $L\times W\times H$는 LWH개의 노드를 가진 큐빅 그리드(cubic grid) 이다. L은 토러스의 길이, W은 토러스의 너비, 그리고 H는 토러스의 높이를 말한다. 3차원 토러스는 순환 연결을 가진 3차원 메쉬의 변형이다. 토러스는 대칭형 위상이고 메쉬에 추가된 여분의 링 연결들은 임의의 두 노드사이의 가장 짧은 거리를 3차원 메쉬의 거리의 반으로 줄인다. 토러스에서 각 노드는 좌표 (x, y, z)로 표현되고, 단, $1 \le x \le L$, $1 \le y \le W$, 그리고 $1 \le z \le H$, 각 에지는 직접 통신 링크에 해당된다. 길이 인덱스는 왼쪽에서 오른쪽으로, 너비 인덱스는 앞에서 뒤로, 높이 인덱스는 아래에서 위로부터 증가한다고 가정하자.

정의 1 : $3DT(L, W, H)$에서 길이 l, 너비 w, 높이 h를 갖는 부분메쉬 $S(l, w, h)$의 식별자는 두개의 3차원 좌표 $S((x, y, z),(x', y', z'))$로 표기된다. (x, y, z)는 S의 시작점, 그리고 (x', y', z')는 S의 끝점이다.

$S((x, y, z),(x', y', z'))$가 순환 링크를 사용하지 않을 때, 즉 (그림 1) (a)처럼 $x \le x', y \le y', z \le z'$일 때 (x, y, z)가 시작점이고 (x', y', z')이 S의 끝점이다. 반면에 x-차원, y-차원, z-차원에서 순환 링크를 사용한다면, 즉, $x > x', y > y', z > z'$일 때 S의 시작점과 끝점의 상대 좌표는 (그림 1) (b)~(h)와 같이 일곱개의 가능한 경우 중의 하나이다. <표 1>에서 R_x, R_y, R_z는 각각 부분메쉬의 x 좌표, y 좌표, z 좌표 범위로 나타낸다. 예로써 $[x, x']$는 부분메쉬의 x 좌표의 (R_x) 범위를 나타낸다. <표 2>는 <표 1>의 각 경우에 해당하는 부분메쉬의 길이, 너비, 높이를 보여준다. 부분메쉬가 원래의 식별자로 표현된다 하더라도 여러 개의 작은 부분들로 나뉘어진 큐브들로 구성되는 부분메쉬로 가정하거나 조작하기는 어렵다. (그림 1) (h)의 극단적인 경우는 논리적인 큐브는 순환 링크로 연결된 3차원 토러스의 여덟 개의 코너(구석)에 자리잡은 8개의 부분 메쉬로 구성된다. 여기서 논리적으로 인접한 노드들이 연속적인 인덱스들은 갖지 않아서 프로세서 할당을 위해 배열을 조작하는 것은



(그림 1) 3차원 토러스에서 가능한 8가지 부분메쉬들

〈표 1〉 3차원 토러스에서 여덟가지 가능한 3차원 부분메쉬의 각 차원의 범위

Case	상 대 위 치	R_x	R_y	R_z
1	$x\le x', y\le y', z\le z'$	$[x. x']$	$[y, y']$	$[z, z']$
2	$x\le x', y\le y', z>z'$	$[x. x']$	$[y, y']$	$[z, H], [1, z']$
3	$x\le x', y>y', z\le z'$	$[x. x']$	$[y, W], [1, y']$	$[z, z']$
4	$x\le x', y>y', z>z'$	$[x. x']$	$[y, W], [1, y']$	$[z, H], [1, z']$
5	$x>x', y\le y', z\le z'$	$[x. L], [1, x']$	$[y, y']$	$[z, z']$
6	$x>x', y\le y', z>z'$	$[x. L], [1, x']$	$[y, y']$	$[z, H], [1, z']$
7	$x>x', y>y', z\le z'$	$[x. L], [1, x']$	$[y, W], [1, y']$	$[z, z']$
8	$x>x', y>y', z>z'$	$[x. L], [1, x']$	$[y, W], [1, y']$	$[z, H], [1, z']$

〈표 2〉 8개의 가능한 3차원 부분메쉬의 길이,너비 그리고 높이 Case

Case	l	w	h
1	$x' - x + 1$	$y' - y + 1$	$z' - z + 1$
2	$x' - x + 1$	$y' - y + 1$	$z' - z + 1 + H$
3	$x' - x + 1$	$y' - y + 1 + W$	$z' - z + 1$
4	$x' - x + 1$	$y' - y + 1 + W$	$z' - z + 1 + H$
5	$x' - x + 1 + L$	$y' - y + 1$	$z' - z + 1$
6	$x' - x + 1 + L$	$y' - y + 1$	$z' - z + 1 + H$
7	$x' - x + 1 + L$	$y' - y + 1 + W$	$z' - z + 1$
8	$x' - x + 1 + L$	$y' - y + 1 + W$	$z' - z + 1 + H$

성가신 일이다. 따라서 연속적인 인덱스로 된 확장된 논리적 인덱스들을 포함하는 가상 공간의 개념을 도입한다. 이 절의 후반부에서 가상 3차원 좌표가 매핑 함수를 이용하여 어떻게 실제 좌표로 변형되는지 소개한다. 다음으로 가상공간에 있는 가상 부분메쉬의 식별자를 정의한다.

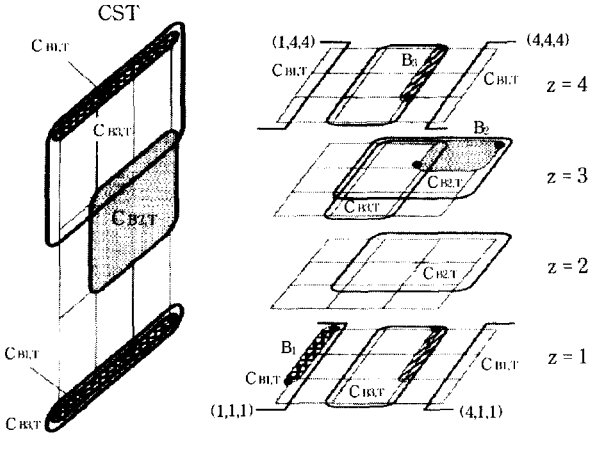
정의 2 : 부분메쉬 $S((x,y,z),(x',y',z')$의 가상 부분메쉬의 S_v식별자는 3차원 좌표로 $((x,y,z),(x_v',y_v',z_v')$는 S_v의 시작점이고 (x_v',y_v',z_v')는 S_v의 끝점이다.

단, $1 \le x \le L, 1 \le y \le W, 1 \le z \le L, 1 \le x_v' \le 2L-1, 1 \le y_v \le 2W-1, 1 \le z_v \le 2H-1$이다. $x > x'$이면 $x_v' = x' + L$이고, 그렇지 않으면 $x_v' = x'$이다. 마찬가지로, $y > y'$이면 $y_v' = y' + W$이고, 그렇지 않으면 $y_v' = y'$이다. $z > z$이면 $z_v' = z' + H$이고, 그렇지 않으면 $z_v' = z'$이다.

정의 3 : 부분메쉬 $S(l,w,h)$의 크기는 부분메쉬에 있는 노드들의 수, $l \times w \times h$로 정의된다.

정의 4 : 사용 부분메쉬 B는 부분메쉬에 있는 모든 노드들이 현재 작업에 할당되어 사용 중인 상태이다. 가상 사용 부분메쉬는 위의 정의 2에 의해서 사용 부분메쉬의 가상공간에서의 부분메쉬로써 얻어진다. 사용중인 부분메쉬들의 리스트 *B-list*는 사용 부분메쉬들의 모든 식별자를 포함한다. 본 논문의 예제에서는 사용중인 부분메쉬를 그들간의 구별을 위해서 B_i로도 표시한다. 단, i = 1, 2, 3.

예제 1 : (그림 2)에는 $3DT(4,4,4)$안에 세 개의 사용 부분메쉬들 B_1, B_2 그리고 B_3가 존재하고, 이들의 크기는 가로, 세로 및 높이에 있어서 각각 1×3×1, 2×2×1, 그리고 1×3×2 이다.

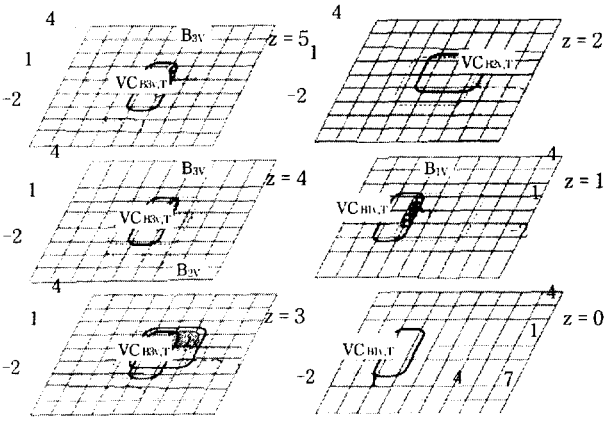


(그림 2) 사용중, 커버리지 부분메쉬와 CS에 투영된 이미지의 예제

즉, $B_1(1,3,1) = ((1,2,1),(1,4,1))$, $B_2(2,2,1) = ((3,3,3),(4,4,3))$, $B_3(1,3,2) = ((3,2,4),(3,4,1))$이다. 따라서 *B-list*는 B_1, B_2, B_3을 모두 포함한다. $B_{3v}(1,3,2) = ((3,2,4),(3,4,5)) \neq B_3$인 반면 $B_{1v} = B_1$, $B_{2v} = B_2$에 이다.

정의 5 : 가상 사용 부분메쉬 B_{iv}에 있어서 작업 $T(l,w,h)$에 대한 가상 커버 부분메쉬 $VC_{B_{iv},T} = ((x_{vc}, y_{vc}, z_{vc}),(x'_{vc}, y'_{vc}, z'_{vc}))$는 가상 사용 부분메쉬를 커버하는 가상 부분메쉬이다. 단, $x_{vc} = x - l + 1, y_{vc} = y - w + 1, z_{vc} = z - h + 1, (x'_{vc}, y'_{vc}, z'_{vc}) = (x_v', y_v', z_v')$이다. $1 \le x \le L, 1 \le l \le L$ 임을 상기하자. $x = 1, l = L$일 때 x_{vc}의 최소값을 얻을 수 있는데 그 값은 $l - L + 1 = 2 - L$이다. 반면에, $x = L, l = 1$일 때 x_{vc}의 최대값을 얻는데 그 값은 $L - 1 + 1 = L$이다. 따라서 $2 - L \le x_{vc} \le L$이다. 마찬가지로 $2 - W \le y_{vc} \le W$, $2 - H \le z_{vc} \le H$ 이다. 그러므로 가상 공간에서 가상 부분메쉬 $S((a,b,c),(d,e,f))$의 범위는 $(2-L) \le a \le L, (2-W) \le b \le W, (2-H) \le c \le H, 1 \le d \le 2L-1, 1 \le e \le 2W-1, 1 \le f \le 2H-1$이다. VC-리스트는 새 작업 T를 위한 가상 커버 부분메쉬들의 모든 식별자를 포함한다.

예제 2 : (그림 3)에서 시스템으로 들어오는 $T(2,2,2)$에 대하여, 가상 사용 부분메쉬에 대응하는 가상 커버부분메쉬들 B_{1v}, B_{2v}, B_{3v}은 각각 $VC_{B1v,T} = (0,1,0),(1,4,1))$, $VC_{B2v,T} = ((2,2,2),(4,4,3))$, $VC_{B1v,T} = ((2,1,3),(3,4,5))$이다.



(그림 3) 가상 사용중 및 커버리지 부분메쉬들

물론 다른 작업 T'에 대해서는 다른 가상 커버부분메쉬를 얻는다. (그림 3)에서 실제로 Z-차원에 10개의 평면들이 존재함을 주시하자. 4개의 평면 $z = -2, -1, 6, 7$에는 부분메쉬가 존재하지 않으므로 제외된다.

정의 6 : 좌표 매핑. 한 노드의 물리적인 3차원 좌표는 가상 3차원 좌표에서 좌표 매핑에 의해 얻어진다. 주어진 가상 3차원좌표 (x_v, y_v, z_v)에 대응되는 실제 3차원 좌표 (x, y, z)는 아래의 매핑 함수 Ψ를 통해서 얻어진다. 그러므로, $\Psi(x_v, y_v, z_v) = (x, y, z)$이다. 여기서

$$x = \begin{cases} x_v & 1 \le x_v \le L \\ x_v + L & x_v < 1 \\ x_v - L & x_v > L, \end{cases} \quad y = \begin{cases} y_v & 1 \le y_v \le W \\ y_v + W & y_v < 1 \\ y_v - W & y_v > W, \end{cases}$$

$$z = \begin{cases} z_v & 1 \le z_v \le H \\ z_v + H & z_v < 1 \\ x_v - H & z_v > H. \end{cases}$$

정의 7 : $T(l, w, h)$에 대한 사용 부분메쉬 $B = ((x, y, z), (x', y', z'))$의 커버 부분메쉬는, 가상 커버 부분메쉬 $VC_{B,T}$의 노드에서 매핑된 노드들을 구성하는 부분메쉬로써 $C_{B,T} = (((x_c, y_c, z_c), (x'_c, y'_c, z'_c)) = (\Psi(x_{vc}, y_{vc}, z_{vc}), \Psi, (x'_{vc}, y'_{vc}, z'_{vc})))$로 표기한다. 커버 부분 메쉬 안에 존재하는 어떤 노드들도 사용 부분 메쉬이기 때문에 T를 할당하기 위한 자유 부분메쉬의 끝점으로 제공될 수 없다. 여기서 자유 부분메쉬는 프로세서에 현재 어떤 작업도 동시에 할당되어지지 않은 3차원 부분메쉬이다. 최대 자유 부분메쉬는 어떤 차원으로도 확장이 불가능한 자유 부분메쉬이다. 자유 리스트는 모든 최대 자유 부분메쉬들의 리스트이다.

3. 제안된 할당 알고리즘

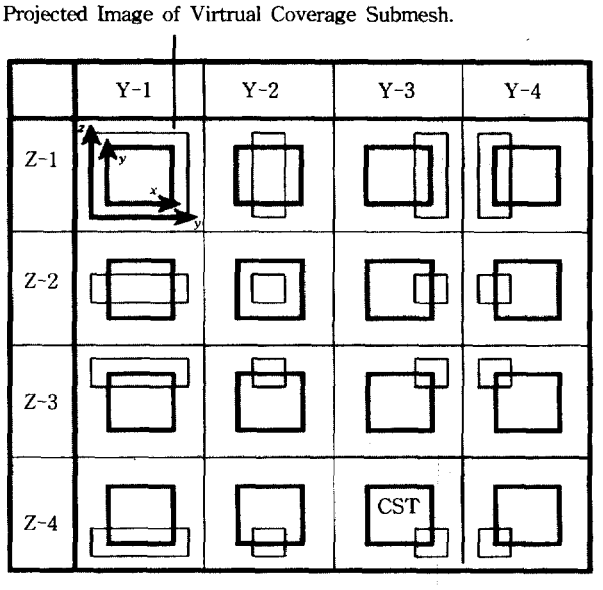
3.1 Coverage Status Table(CST)

세그먼트(segment)는 3차원 토러스 시스템의 x-차원 행에 있는 연속적인 노드들이라 하자. 세그먼트가 어떤 한 행에 있는 첫번째 노드에서 시작하고 세그먼트에 있는 모든 노드들이 어떤 커버 부분메쉬에 속한다면 '*covered segment from 1*'이라 한다. 유사하게 세그먼트가 마지막 노드에서 끝나고 그것이 어떤 커버 부분메쉬에 속하면 '*covered segment from L*'이라 한다. 세그먼트가 어느 행 전체를 덮고 세그먼트의 모든 노드들이 어떤 커버 부분메쉬에 속하면 '*complete coverage segment*'라 한다. CST는 $W \times H$ 크기의 2차원 배열로 $CST(W, H)$로서 표기한다. 단, W은 앞에서처럼 토러스의 너비, 그리고 H는 높이이다. 이 배열의 요소는 각 x-차원의 행에서 *covered segment from 1*의 가장 오른쪽 노드의 x 좌표와 *covered segment from L*의 가장 왼쪽 노드의 x 좌표이다. CST는 VC-리스트를 구성한 후에 각각의 행에 대한 정보를 저장한다.

왼편 CST에 세 개의 커버 부분메쉬가 존재한다. 그림의 오른쪽에서 왼쪽 방향으로 불을 밝힐 경우의 3차원 토러스로 가정하자. 그러면 (그림 2)에서 보는 것과 같이 각 커버 부분메쉬의 투영된 이미지는 왼편 2차원 평면에 있는 어두운 부분으로 나타나 있다. 어두운 부분에 대응되는 $CST(W, H)$의 요소들은 위에서 언급했던 것처럼 커버된 세그먼트의 x 좌표의 범위에 따라 결정된다. 각 커버 부분메쉬는 해당하는 가상 커버 부분메쉬와 좌표 매핑 함수 Ψ에 의해서 얻어짐을 볼 수 있다. 가상 커버 부분메쉬 $S((a, b, c), (d, e, f))$가 주어지면 CST안에는 16가지의 투사된 이미지와 CST의 구성이 존재한다. 여기서,

$$(2-L) \le a \le L, (2-W) \le b \le W, (2-H) \le c \le H,$$
$$1 \le d \le 2L-1, 1 \le e \le 2W-1, 1 \le f \le 2H-1$$

이다. CST의 W에 관하여 S의 y-차원의 존재 범위에 따라 4개의 경우가 있다. 각 경우는 **Y-1)** $e-b+1 \ge W$, **Y-2)** $1 \le b, e \le W$, **Y-3)** $e-b+1 < W, e > W$, **Y-4)** $e-b+1 < W, b < 1$이다. z-차원의 존재범위에 따라 4가지의 경우가 있다. 각 경우는 **Z-1)** $f-c+1 \ge H$, **Z-2)** $1 \le c, f \le H$, **Z-3)** $f-c+1 < H, f > H$, **Z-4)** $f-c+1 < H, c < 1$이다. 그러므로 (그림 4)에서처럼 전체 16가지의 결합이 가능하다.



(그림 4) CST와 가상커 버리주부분메쉬의 투영된 이미지

예제 4 : (그림 2)와 (그림 3)에서 $VC_{B1,T} = ((0,1,0), (1,4,1))$은 Y-1과 Z-4에 해당한다. 고로, $R_y = [1,4]$이고, $R_z = [1.1]$과 $[1,4]$이다. 이것은 $C_{B1,T} = ((4, \underline{1}, \underline{4}), (1, \underline{4}, \underline{1}))$로부터 얻어진다. $VC_{B2,T} = ((2,2,2), (4,4,3))$은 Y-2와 Z-2경우에 대응한다. 그러므로, R_y와 R_z는 각각 $[2,4]$와 $[2,3]$가 된다. $VC_{B3,T} = ((2,1,3), (3,4,5))$에는 Y-1과 Z-3의 경우가 적용된다. $R_y = [1,4]$와 $R_z = [1,1], [3,4]$이며 $C_{B3,T} =$

((2, 1, 3),(3, 4, 1))로부터 얻어진다.

주어진 *VC-list*로부터 *CST*의 값을 어떻게 갱신하는지 논의해보자. 처음에 모든 j와 k에 대하여 $CST[j,k].fromL=0$, $CST[j,k].fromL=L+1(i\le j,k\le L)$이다. 이전에 언급했던 것과 같이, CST의 값들은 커버 부분메쉬들의 x좌표의 존재범위에 의해 결정된다. 가상 커버 부분메쉬 S의 어두운 부분이 이미 결정되었다고 가정하자. **x-1)** $d-a+1<L$, **x-2)** $1\le a,d\le L$, **x-3)** $d-a+1<L$, **x-4)** $d-a+1<L$, $a>1$과 같은 x-차원에 있는 S의 존재 범위에 대해 4가지의 경우가 있다.

x-1)경우에 있어서 $CST[j,k].fromL=L$이다. 이는 이 범위에 있는 모든 노드들이 좌표 매핑함수 Ψ에 의해 부분메쉬에 있는 1과 L사이의 모든 노드와 매핑되어서 완전 커버 세그먼트가 되기 때문이다. **x-2)**경우에 만약 $a<CST[j,k].fromL+1\le d$이면 $CST[j,k].fromL=d$이다. 이것은 현재 $CST[j,k].fromL$의 값이 1에서 시작하는 가장 오른쪽의 커버된 노드가 아니고 또한 같은 커버 부분메쉬에 속하는 a와 b사이의 노드이기 때문이다. d는 1에서 시작하는 가장 오른쪽의 커버된 노드가 된다. **x-3)**경우에 만약 $(d-L)>CST[j,k].fromL$이면 $CST[j,k].fromL=d-L$이고 또한, $a<CST[j,k].fromL$이라면, $CST[j,k].fromL=a$이다. 이것은 순환 연결이 사용 되어서 가상 커버 부분메쉬가 *CST*와 교차되고 가상 커버 부분메쉬의 길이가 L보다 작기 때문이다. 좌표 매핑 함수 Ψ에 의해 얻어진 $[1,d-L],[a,L]$과 같은 커버 부분메쉬의 x좌표에 따라 두 개의 존재범위가 있다. 만약 $d-L\le a$이면, $CST[j,k].fromL=d-L$이고 $CST[j,k].fromL=a$이다. **x-4)**경우에서, 만약 $d>CST[j,k].fromL$이면 $CST[j,k].fromL=d$이고 또한, $(a+L)<CST[j,k].fromL$이면 $CST[j,k].fromL=a+L$이다. 이것 역시 순환경우이지만 가상 커버 부분메쉬의 길이가 L보다 작다. 좌표 매핑에 의해 커버 부분메쉬의 x-차원에 따른 $[1,d],[a+L,L]$의 두 가지 존재 범위가 있다. 다시 말해서 $d\ge a+L$이면 $CST[j,K]fromL=d$이고 $CST[j,k].fromL=a+L$이다.

들어오는 $T(j,w,h)$와 n개의 사용 부분메쉬 $B_i(1\le i\le n)$에 대하여 *CST*를 얻기 위한 의사코드가 아래에 설명되어 있다. 처음에 *VC-list*는 널(null)로 $CST[j,k].fromL=0$, $CST[j,k].fromL=L+1$로 초기화된다 $(1\le j\le W,2\le k\le H)$.

```
Procedure BUILD-CST
Input : Busy submeshes B_i(1 ≤ i ≤ n), and a new incoming
        task T ;
Output : Coverage Status Table
         CST[j,k] for 1 ≤ j ≤ W and 1 ≤ k ≤ H
For each B_i = ((x,y,z),(x',y',z')) do
    Compute VC_{B_i,T} = ((x_vc, y_vc, z_vc),(x'_vc, y'_vc, z_vc)).
    Insert VC_{B_i,T} into the VC-list by the increasing
          order of x_vc.
    /* If x_vc > L, x_vc is assumed to 1 when VC_{B_i,T} is inserted
       since the submesh has the wraparound connection. */
For i=1 to do
          /* beginning from the head of the VC-list */
Calculate C_{B_i,T} from VC_{B_i,T} by the coordinate
          mapping function Ψ.
          Obtain R_y and R_x of C_{B_i,T}.
/* The residence ranges in y and z-dimension will be used as
x-range and y-range of CST, respectively.*/
 Obtain R_x of C_{B_i,T} and update the CST(W,H) accordingly.
```

여기서 *CST*의 초기화는 $O(WH)$가 걸리고 $VC_{B_i,T}$를 계산하고 *VC-list*에 삽입하는 것이 $O(nlogn)$이 걸리고 *CST*를 갱신하는 것은 $O(nWH)$가 걸린다. WH가 3차원 토러스 내부의 실질적인 작업에 대해 $\log n$보다 크므로 $O(nWH)$가 **BUILD - CST** 프로시저의 시간복잡도가 된다.

3.2 프로세서 할당과 해제

```
Procedure PROCESSOR-ALLOCATION

Step 1. If (the number of free processors < L · W · H)
        go to Step 6.
Step 2. Determine the orientation of T. There are up
        to 6 orientation of T.
Step 3. Based on current B-list and T, run the procedure
        BUILD - CST.
Step 4. For k=1 to H do
          For j=1 to W do
            If (CST[j,k].fromL + 1 < CST[j,k].fromL)
               x = CST[j,k].fromL + 1 ;
               y = j; z = k ;
               base_found = TRUE ; go to Step 5.
Step 5. If (base_found = TRUE)
        Allocate the free submesh whose base is (x,y,z)
        to T of the current orientation.
        Add T to B-list. Stop.
        Otherwise, if all possible orientations have been checked,
        go to Step 6.
                else go to Step 2.
Step 6. Wait until a deallocation occurs.
```

작업 할당기는 단계 1에서 사용 가능한 프로세서의 수를 검사한다. 단계 2에서 원래의 T의 방향을 가지고 프로시저는 시작한다. 나중에 현재의 방향이 할당되지 못하면 작업의 방향은 변한다. 만일 l,w,h가 모두 다르면 6개의 다른 방향이 있고 두개가 같다면 3개의 다른 방향이 있다. $l=w=h$이면 오직 한 개의 방향만을 갖는다. 단계 3에서 *CST*는 BUILD-CST를 호출함으로써 생성된다. 단계 4에서 *CST*는 $CST[1,1]$에서 $CST[W,H]$까지 스캔한다. *CST*를 스캔하는 동안에 조건이 만족되면 T의 시작점은 결정되고, base_found는 TRUE로 세트된다. 또한 단계 5에서 T를 B-리스트

에 삽입한다. 만약 할당이 실패하면 모든 가능한 다른 방향을 체크한다. 만약 할당이 불가능하다면 할당해제가 발생할 때까지 기다린다. 명백하게 단계 1과 2는 $\Theta(1)$이 걸린다. 단계 4에서 두개의 for-loop때문에 $\Theta(WH)$가 걸린다. For-loop의 주요 부분은 많아야 WH번 실행된다. 단계 5에서 base_found = TURE일때는 $\Theta(n)$걸리고 그렇지 않다면 $\Theta(1)$이 걸린다. 단계 3에서 BUILD-CST프로시저 실행과 CST 구성을 위해 $O(nWH)$가 걸린다. 그리고 PROCESSOR-ALLOCATION 프로시저보다 크다. 부분 메쉬 S를 할당 해제하는 프로시저는 B-리스트에서 S를 지움으로써 매우 간단하게 할 수 있다.

4. 성능 평가

제안된 방법의 할당과 해제의 시간복잡도가 [14]와 <표 3>에서 비교된다.

〈표 3〉 시간과 공간 복잡도의 비교

	제안한 방법	[14]
할 당	$O(IW^2H^4)$	$O(L^4W^4H^4)$
해 제	$O(LWH)$	$O(L^4W^4I^4)$
메모리	$O(LWH)$	$O(L^3W^3H^3)$

시뮬레이션은 작업들의 할당과 해제되는 이벤트들을 가진 이벤트 중심 구현 방식이다. 시뮬레이션은 16×16×16에서 64×64×64까지의 3차원 토러스들을 생성했다. 다른 크기들에서 시뮬레이션 연구 결과들이 유사한 유형을 따르므로, 여기서 우리는 16×16×16 토러스에 대한 시뮬레이션 결과만를 가지고 보고한다. 모든 시뮬레이션들은 ±5%의 오차를 가진 90% 신뢰구간을 사용한다. 시뮬레이터는 Sun 4/190에서 동작하는 C언어로 개발되었다. 시뮬레이션 결과들은 5개의 독립적인 실행들로부터 얻어진다. 비교적 입력 부하가 클 때 시스템은 포화상태가 될 수 있고 큐에서 대기하고 있는 작업은 끊임없이 커진다. 이것은 무한 평균 반응 시간을 초래한다. 따라서 우리는 안정된 시스템을 보장하는 시스템 부하만을 고려한다.

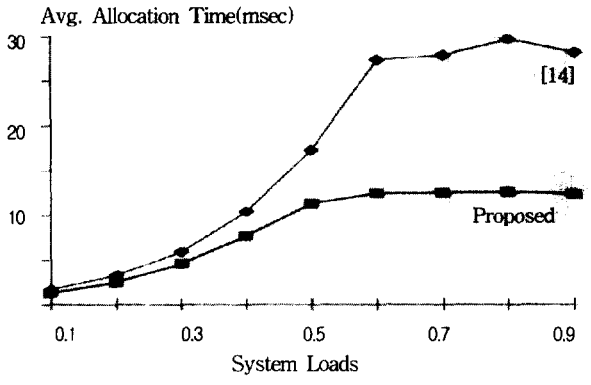
시스템 모델은 다음과 같다. 처음에 모든 메쉬들은 사용가능하고 5000개의 작업을 생성하고 처리한다. 작업 서비스 시간과 도착시간은 지수 분포를 따른다고 가정한다. 시간 단위는 [14]와 같이 성능을 연구하기 위해 부분 메쉬 할당을 위해 필요로 하는 만큼 충분하다고 가정한다. 들어오는 작업의 l, w, h은 작업의 크기의 다양한 분포에 대한 3개의 분포(균일분포, 감소분포, 증가분포)중 하나를 따른다고 가정하자.

균일분포에서 들어오는 태스트의 길이는 균일하게 1과 토러스 L의 변 길이 사이에 분포한다. 감소 분포에서 $[1, L]$의 범위는 4개의 간격 - $[1, L/8]$, $[L/8+1, L/4]$, $[L/4+1, L/2]$, $[L/2+1, L]$으로 나뉜다. $[1, L/8]$범위에 있을 확률은 0.4, $[L/8+1, L/4]$은 0.2, $[L/4+1, L/2]$은 0.2, $[L/2+1, L]$은 0.2이다. 각 간격 안에 l, w, h분포는 균일하다. 예를 들어 $3DT(16, 16, 16)$의 $P[s1, s2]$로 표기되는 $[s1, s2]$변에 떨어질 확률은 다음과 같다. $P[1, 2] = 0.4, P[3, 4] = 0.2, P[5, 8] = 0.2$, 그리고 $P[9, 16] = 0.2$이다. 특히, l, w, h에 대한 감소 분포는 작업이 작은 크기 부분메쉬를 요구하는 작업들이 우세한 시스템을 표현한다. 증가분포는 감소분포와 반대다. 3DT(16, 16, 16)에서 $P[1, 8] = 0.2$, $P[9, 12] = 0.2$, $P[13, 14] = 0.2$, $P[15, 16] = 0.4$이다. 이 분포에서, 시스템은 큰 부분메쉬를 요구하는 작업을 받을 기회가 커질 것이다. 작업의 길이, 너비, 높이는 3개의 분포 중 하나에 기반하여 독립적으로 생성된다. 3개의 분포에 대한 시뮬레이션 결과들은 아주 유사하다. 따라서 우리는 균일분포의 결과만을 보고한다.

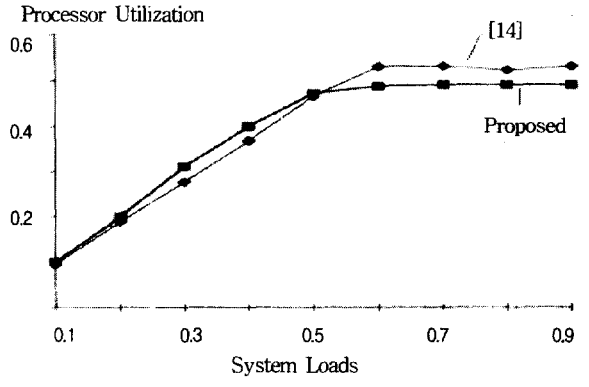
다른 부하에서 제안된 정책의 성능을 연구하기 위해 부하를 다음과 같이 정의한다. 부하는 $\frac{n \times MSVT}{N \times MIAT}$으로 정의한다. 여기서 n은 프로세서의 수에 대해서 요청한 부분메쉬의 평균 크기이고, N은 3차원 토러스 시스템에 있는 총 프로세서의 수이다. $MIAT$는 평균 작업 서비스 시간이고 는 평균 작업 도착 시간이다. 시뮬레이션에서 $MSVT$를 10단위로 고정하고 요구 부하에 따라 $MIAT$값을 조정한다. 명백히 말하면, 시스템은 너무 작은 $MIAT$값에서 불안정해 지는 것이다. 수집한 성능 측정값들은 완료시간에 대한 이용된 프로세서 수의 평균값인 프로세서 이용률(총 5000개의 작업을 완료하는데 드는 시간)이며 평균 작업 할당 시간은 대기 큐의 앞(head)에 있는 작업이 할당되는 시간이다.

작업 할당기는 FCFS를 따른다고 가정하자. 즉, 할당기는 항상 큐에 있는 첫 작업에게 자유 부분메쉬를 찾는 노력을 할 것이다. 만약 자유 부분메쉬를 찾지 못하면 할당기는 단순히 해제될 부분메쉬를 기다린다.

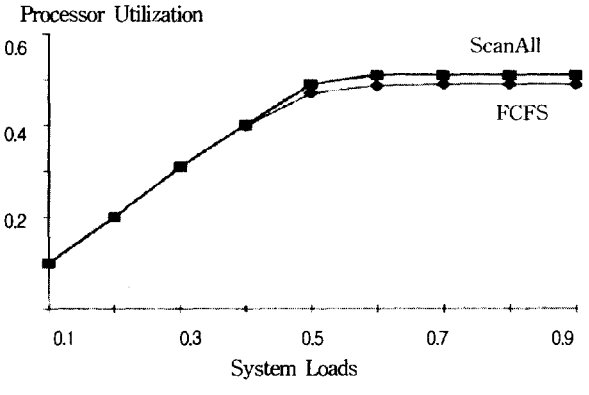
(그림 5)은 균일하게 분포된 변 길이에 대해 제안된 방법과 [14]의 평균 할당 시간을 보여준다. 제안한 방법은 입력 부하에 관계없이 다른 방법보다 꾸준히 더 작은 할당시간이 걸린다. 차이는 입력부하가 증가할수록 더 현저하다. 따라서 제안한 방법이 높은 부하의 조건에서 더 효과적임을 말할 수 있다. (그림 6)은 0.1에서 0.9까지의 다른 작업부하에서 2개의 방법의 프로세서 이용률을 비교한다. 또한 Scan All이라는 또 따른 스케줄링 기법에서 제안된 방법의 성능을 조사했다. Scan All은 FCFS와 달리, 큐에 있는 첫 번째 작업이 할당되지 않으며 첫 번째 작업 뒤에 있는 어떤 할당될 수 있는 작업을 스캔한다. 그런 다음 첫 번째 작업 전에 그 작업이 할당된다. 그러나 기아를 피하기 위해 첫 번째 작업이 이미 정해진 시간을 다 소비하면 그 작업이 할당 될수 있을 때까지 스캔을 중지한다. (그림 7)과 (그림 8)은 제



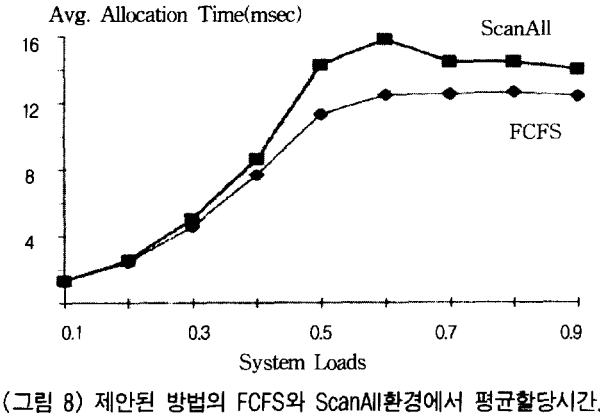
(그림 5) 제안된 방법과 [14]의 평균 할당시간.



(그림 6) 제안된 방법과 [14]의 프로세서 이용률.



(그림 7) 제안된 방법의 FCFS와 ScanAll환경에서 프로세서 이용률.



(그림 8) 제안된 방법의 FCFS와 ScanAll환경에서 평균할당시간.

안된 방법과 FCFS, Scan All에 의한 이용률과 평균 할당시간을 나타낸다. Scan All에서 이용률이 약간 FCFS보다 좋음을 알 수 있다. 그러나 평균 할당 시간은 더 나쁘다.

5. 결 론

본 논문은 완전 인식 능력을 가진 3차원 토러스 멀티컴퓨터 시스템을 위한 최초 적합 프로세서 할당 정책을 제안한다. 종합적인 컴퓨터 시뮬레이션은 입력 부하의 실제 범위하에서 제안된 정책의 평균 할당 시간이 최적 적합 전략에 기반한 기존 정책[14]보다 훨씬 적다는 것을 보여준다. 이것은 2차원 Coverage-Status-Table(CST)을 이용하여 자유 부분메쉬를 찾는 효율적인 검색 메커니즘에 의해서 실현된 것이다.

고차원 메쉬나 토러스 병렬 컴퓨터들을 위한 기존 대부분의 할당 정책은 FCFS가 불필요하게 차후의 작업을 블록화 시킬 수 있다 하더라도 FCFS의 단순성과 공정성 때문에 FCFS 스케줄링 정책을 사용한다. 또한 블록킹으로 인한 잠재적인 성능손실을 피하기 위해 다른 스케줄링 기반의 효과도 연구한다. 다양한 부하와 연산 조건 하에서 또다른 스케줄링 기법과 작업 할당 방법의 관계를 알아내는 연구가 진행되고 있다.

참 고 문 헌

[1] M. J. Quinn, *Parallel Computing Theory and Practice*, McGraw-Hill, New York, 1994.

[2] K. Hwang, *Advanced Computer Architecture : Parallelism, Scalability, Programmab-ility*, Mcgraw-Hill, New York, 1993.

[3] N 큐브 Corp., *N큐브/ten : An Overview*, Beaverton, OR, Nov. 1985.

[4] C. L. Seitz, "The cosmic 큐브," *Commun. ACM*, Vol 28, No. 1, pp.22-23, Jan. 1985.

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

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

[7] L. M. Ni and P. K. McKinley, "A survey of wormhole routing techniques is direct networks," *IEEE Computer*, pp.62-76, Feb. 1993.

[8] 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.

[9] S. M. Yoo and H. Y. Youn, "An efficient task allocation scheme for two-dimensional mesh connected systems," *Int'l Conf. on Dist. Comp. Systems*, pp.501-508, May. 1995.

[10] J. Ding and L. N. Bhuyan, "An adaptive submit allocation strategy for two-dimenssional mesh connected systems,"

Int'l Conf. on Parallel Processing, pp.II-193-200, Aug. 1993.

[11] P. J. Chuang and N. F. Tzeng, "An efficient submit allocation strategy for mesh computer systems," *Proc. Int'l Conf. on Dist. Comp. Systems*, pp.256-263, Aug. 1991.

[12] T. Lin, W-K. Huang, F. Lombardi, and L.N Bhuyan, "A submit allocation scheme for mesh-connected multiprocessor systems," *Int'l Conf. on Parallel Processing*, pp.II-159-163, Aug. 1995.

[13] R. E. Kessler and J. L. Schwarzmeier, "CRAY T3D : A new dimension for Cray research," *Proc. COMPCON*, pp. 176-182, Feb. 1993.

[14] W. Qiao and L. M. Li, "Efficient processor allocation for 3D tori," *IEEE Int'l Parallel Processing Symp.*, pp.466-471, Apr. 1995.

추 현 승

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

1988년 성균관대학교 수학과 졸업(학사)

1990년 텍사스 주립대(달라스) 전자계산학과(공학석사)

1996년 텍사스 주립대(알링턴) 전산공학과(공학박사)

1997년 특허청 심사4국 컴퓨터심사 담당관실 심사관(사무관)

1998년~현재 성균관대학교 전기전자 및 컴퓨터공학부 조교수

관심분야 : 병렬 및 분산 처리, ATM네트워크, 이동네트워크 등

윤 희 용

e-mail : youn@ece.skku.ac.kr

1977년 서울대학교 전기공학과 졸업(학사)

1979년 서울대학교 전기공학과 대학원(공학석사)

1988년 Univ. of Massachusetts at Amherst, 컴퓨터공학과 대학원(공학박사)

1988년~1991년 Univ. of North Texas, 전산학과 조교수

1991년~2000년 Univ. of Texas at Arlington, 컴퓨터공학과 부교수

1999년~2000년 한국정보통신대학원 대학교 교수

2000년~현재 성균관대학교 전기전자 및 컴퓨터공학부 교수(특별 채용 교수)

관심분야 : 병렬 및 분산 처리, 저장시스템, 이동컴퓨팅 등

박 경 린

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

1986년 중앙대학교 전자계산학과 졸업(학사)

1988년 중앙대학교 전자계산학과 대학원(공학석사)

1992년 텍사스 주립대(알링턴) 전산공학과 대학원(공학석사)

1997년 텍사스 주립대(알링턴) 전산공학과 대학원(공학박사)

1998년~현재 제주대학교 자연과학대학 전산통계학과 조교수

관심분야 : 분산/병렬 처리 시스템, 오류 허용 시스템, 성능평가 등