RMESH에서 선형 사진트리의 블록 위치 계산을 위한 상수시간 알고리즘

한 선 미† · 우 진 운††

요 약

계층적 자료구조인 사진트리는 영상을 표현하는데 매우 중요한 자료구조이다. 사진트리를 메모리에 저장하는 방법 중 선형 사진트리 표현 방법은 다른 표현 방법과 비교할 때 저장 공간을 매우 효율적으로 절약할 수 있는 이점이 있기 때문에 사진트리와 관련된 연산의 수행을 위해 선형 사진트리를 사용하는 효율적인 알고리즘 개발에 많은 연구가 진행되어 왔다. 블록위치 계산은 영상에서부터 주어진 블록을 완전히 포함하는 컴포넌트를 추출하는 연산으로, 영상 처리의 응용에서 중요하게 사용되는 기하학적 연산에 속한다. 본 논문에서는 RMESH(Reconfigurable MESH) 구조에서 3-차원 $n \times n \times n$ 프로세서를 사용하여 선형 사진트리로 표현된 영상의 블록위치를 계산하는 상수시간 알고리즘을 제안한다. 이 알고리즘은 $n \times n \times n$ RMESH의 계층구조에서 선형 사진트리의 위치코드들을 효율적으로 처리하는 기본적인 연산들을 이용함으로써 상수시간의 시간복잡도를 갖는다.

키워드 : RMESH, 선형 사진트리, 위치코드, 블록위치

Constant Time Algorithm for Computing Block Location of Linear Quadtree on RMESH

Seon Mi Han† · Jin Woon Woo††

ABSTRACT

Quadtree, which is a hierarchical data structure, is a very important data structure to represent images. The linear quadtree representation as a way to store a quadtree is efficient to save space compared with other representations. Therefore, it has been widely studied to develop efficient algorithms to execute operations related with quadtrees. The computation of block location is one of important geometry operations in image processing, which extracts a component completely including a given block. In this paper, we present a constant time algorithm to compute the block location of images represented by quadtrees, using three-dimensional $n \times n \times n$ processors on RMESH(Reconfigurable MESH). This algorithm has constant-time complexity by using efficient basic operations to deal with the locational codes of quardtree on the hierarchical structure of $n \times n \times n$ RMESH.

Key Words : RMESH, Linear quadtree, Locational code, Block location

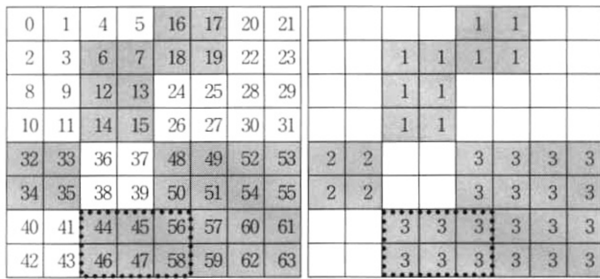
1. 서 론

계층적 자료구조는 컴퓨터 그래픽, 영상처리, 지형처리, 패턴 인식 및 로보트 공학분야 등의 자료를 표현하는데 매우 적합한 기법이다. 특히 계층적 자료구조 중의 하나인 사진트리(quadtree)는 디지털 영상을 규칙적으로 분해하기 때문에 이진영상을 표현하는데 매우 유용한 자료구조이다[1, 2].

지금까지 사진트리를 메모리에 저장하기 위한 여러 가지 방법들이 제안되었다. 그 중 트리 구조를 사용하는 방법은 각 노드가 자신의 자식 노드를 가리키는 포인터 값을 저장하는 공간을 필요로 하므로 사진트리를 구성하는 노드들의 수가 많을 경우 포인터를 기억하기 위한 많은 저장 공간을 필요로 하는 단점이 있다. 이러한 단점을 보완하기 위하여 선형 사진트리(linear quadtree) 표현 방법을 사용한다[1].

선형 사진트리 표현 방법은 사진트리의 BLACK 노드에 해당되는 블록의 위치와 크기에 관한 정보, 즉 (*Index*, *Level*)만을 저장하는 것이다. 이때 (*Index*, *Level*)을 위치코드(locational code)라 한다. 여기에서 *Index*는 사진트리 노드에 해당하는 블록의 맨위 왼쪽에 있는 픽셀의 shuffled 행-우선(row-major) 인덱스이다.

※이 연구는 2006학년도 단국대학교 대학연구비의 지원으로 연구되었음
† 준 회 원 : 단국대학교 대학원 컴퓨터과학전공 박사과정
†† 종신회원 : 단국대학교 정보컴퓨터학부 교수
논문접수 : 2006년 10월 13일, 심사완료 : 2007년 4월 9일



(a) 이진영상의 예 (b) 컴포넌트 레이블링 결과

구 분	위치코드와 레이블	블록위치
선형사진트리	(6, 3, 1) (7, 3, 1) (12, 2, 1) (16, 2, 1) (32, 2, 2) (44, 2, 3) (48, 1, 3)	
주어진 블록	(44, 2) (56, 3) (58, 3)	레이블 3

(c) 레이블을 가진 위치코드와 블록위치

(그림 1) 블록위치 계산의 예

블록위치 계산은 주어진 블록을 완전히 포함하는 컴포넌트를 결정하는 연산으로, 영상이 콤포넌트 레이블링 알고리즘에 의해 레이블이 부여된 상태에서 해당 블록을 포함하는 컴포넌트의 레이블을 출력하여야 한다[1].

예를 들어, (그림 1(a))의 이진영상에 대하여 하단에 점선으로 표시된 블록의 위치 계산을 살펴보자. (그림 1(a))의 영상에는 3개의 컴포넌트들이 존재하며, 컴포넌트 레이블링 알고리즘에 의해 (그림 1(b))와 같이 3개의 컴포넌트로 레이블링될 수 있다. 이때 주어진 블록의 위치는 레이블 3인 컴포넌트에 완전히 포함되므로, 그 결과는 3이 되어야 한다.

이러한 블록위치 계산이 선형사진트리로 표현된 영상에서 수행되기 위해서는 주어진 블록도 위치코드로 표현되어야 한다. 예를 들어, (그림 1(a))의 영상을 위치코드로 나타내면 (그림 1(c))와 같으며, (그림 1(b))의 컴포넌트 레이블을 위치코드의 세번째 원소로 포함시켰다. 또한 주어진 블록의 위치코드를 구하면 (44, 2) (56, 3) (58, 3)이 되며, 레이블 3인 컴포넌트에 완전히 포함되므로 이 블록의 블록위치는 레이블 3으로 계산된다.

일반적으로 하나의 블록은 $n \times n$ 영상에서 $s \times t(1 \leq s, t \leq n)$ 크기로 주어지며, 블록의 왼쪽 상단이 i행과 j열일 때 $(i,\ j)$를 블록의 시작위치라 한다. 예를 들어, (그림 1(a))에서 점선으로 표시된 블록은 시작위치가 (6, 2)이고 크기가 2×3이다. 위치코드와 관련된 연산에서 주어진 블록 역시 위치코드로 표현되어야 한다. 즉, 시작위치가(i,j)인 크기 $s \times t(1 \leq s, t \leq n)$의 블록을 위치코드로 변환하는 것이다. 예를 들어, (그림 1(a))에서 주어진 블록은 시작위치가 (6, 2)이고 크기가 2×3 이므로, 이 블록을 위치코드로 변환하면 (그림 1(c))와 같이 (44, 2) (56, 3) (58, 3)로 된다.

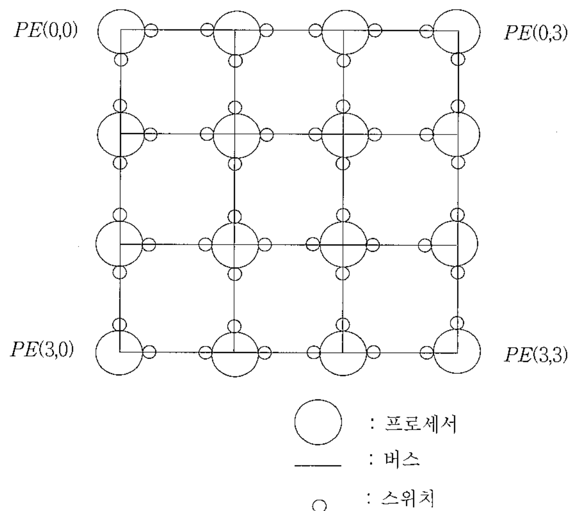
지금까지 선형 사진트리와 관련된 영상 알고리즘들이 개발되었는데, 영상 변환 알고리즘[3], 이진 영상의 축 변환 알고리즘[4] 등의 순차 알고리즘과 이진 영상과 선형 사진트리 사이의 상호 변환[5, 6], 비정렬(unaligned) 선형사진트리의 정렬(alignment)[7], 이웃 블록 찾기[8], 조직 맵을 이용한 클러스터링(clustering)[9] 등의 병렬 알고리즘을 들 수 있다.

RMESH는 Reconfigurable MESH의 약어이다. 기존의 메쉬(mesh) 구조에 동적으로 재구성 가능한 버스 시스템을 결합한 구조로서 Miller, Prasanna-Kumar, Reisis, Stout에 의하여 제안되었으며[10], 구조적인 장점 때문에 다양한 분야에서 연구되었고 효율적인 알고리즘들이 개발되었다[11, 12, 13]. 또한 버스 시스템의 재구성 방법 면에서 서로 차이를 갖는 PARBUS 구조와 MRN 구조가 제안되었다[14, 15].

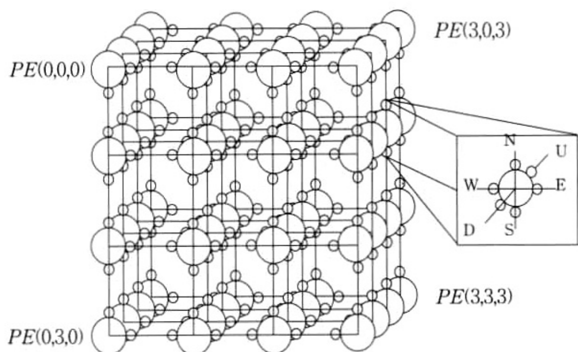
크기가 $n \times n$인 2-차원 RMESH의 기본 구조는 메쉬이며 프로세서들 사이의 통신을 위하여 브로드캐스트 버스(broadcast bus)가 존재한다. 예를 들어, (그림 2)는 4×4 RMESH 구조를 보여준다. 프로세서들을 식별하기 위해 각 프로세서에게 $PE\ (i, j)$ 를 부여한다. 이때 $0 \leq i,j < n$, i는 행의 인덱스이고, j는 열의 인덱스이다.

브로드캐스트 버스상의 통신 제어를 위하여 버스 스위치가 있다. 버스 스위치들은 각 프로세서의 상, 하, 좌, 우에 하나씩 존재하는데, 이를 각각 N(north), S(south), W(west), E(east)라 한다. 버스 스위치는 각 프로세서의 소프트웨어에 의하여 $O(1)$ 시간에 조작되며, 스위치의 개폐 여부에 따라 브로드캐스트 버스를 다수의 서브버스(subbus)들로 재구성이 가능하다. 예를 들어, 각 프로세서가 자신의 S와 N 스위치를 끊고 E와 W 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 행 버스(row bus)라 하고, 자신의 E와 W 스위치를 끊고 S와 N 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 열 버스(column bus)라 한다.

임의의 두 프로세서들은 충돌이 없는 한 공통된 하나의 특정 스위치를 동시에 개폐할 수 있다. 버스상에는 특정 시간에 단 하나의 프로세서만이 데이터를 실을 수 있으며, 서브버스 위에 실린 데이터는 단위 시간에 그 버스에 연결된



(그림 2) 4×4 RMESH 구조



(그림 3) 4×4×4 RMESH 구조

모든 프로세서에게 전달될 수 있다. 만약 한 프로세서가 서브버스상에 있는 모든 프로세서에게 레지스터(register) X의 값을 브로드캐스트하려면 broadcast(X) 명령을 사용하고, 브로드캐스트 버스의 내용을 읽어 레지스터 R에 저장하려면 R := content(broadcast bus) 명령을 사용한다. 따라서 데이터 브로드캐스트는 $O(1)$ 시간에 수행된다.

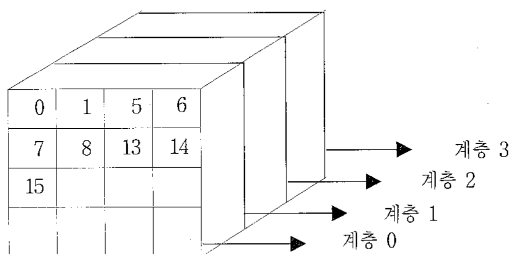
3-차원 RMESH에서는 각 프로세서에게 $PE(l,i,j)$를 부여한다. 이때 $0 \le l,i,j < n$, l은 각 프로세서가 위치한 계층(layer)이고, i와 j는 계층 l 에서의 행과 열의 인덱스이다. 예를 들어, (그림 3)은 4×4×4 RMESH를 보여준다. 프로세서들 사이에는 데이터 전달을 위해 브로드캐스트 버스가 존재하며 버스상의 통신 제어를 위하여 버스 스위치가 있다. 각 층에는 프로세서의 상, 하, 좌, 우에 스위치가 하나씩 존재하는데, 이를 각각 N(north), S(south), W(west), E(east)라 하고, 추가적으로 각 프로세서마다 계층을 연결하는 U(up)와 D(down) 스위치가 존재한다. 버스 스위치는 각 프로세서의 소프트웨어에 의하여 $O(1)$ 시간에 조작되며, 스위치의 개폐 여부에 따라 브로드캐스트 버스를 다수의 서브버스(subbus)들로 재구성이 가능하다. 예를 들어, 각 프로세서가 자신의 S와 N 스위치를 끊고 E와 W 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 행 버스(row bus)라 하고, 자신의 E와 W 스위치를 끊고 S와 N 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 열 버스(column bus)라 한다. 그리고 모든 프로세서의 N, S, W, E 스위치를 끊고 U와 D 스위치를 연결하면 여러 개의 서브버스가 형성되는데, 이를 UD 버스라 한다.

2. 기본적인 연산

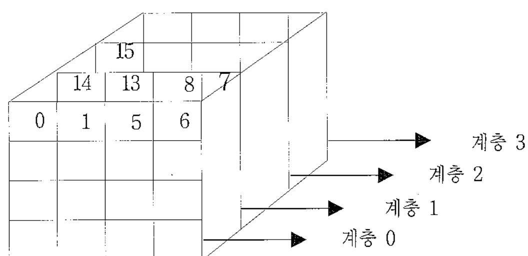
여기서는 3차원 RMESH 구조에서 블록위치 계산을 효율적으로 수행하기 위해 필요한 기본적인 연산들을 알아 본다.

2.1 3차원 RMESH의 재구성

3차원 $n \times n \times n$ RMESH 구조를 $n \times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다. 이 연산은 계층 0에 행-우선 순서로 저장된 위치코드들을 일련의 연속된 위치코드들로 재구성하기 위해 사용되며, 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층에 있는 위치코드를 역순으로 배치함으로써 만들어진다.



(그림 4) 4×4×4 RMESH상의 계층 0의 초기 상태



(그림 5) 4×4×4 RMESH의 재구성 결과

예를 들어, 4×4의 영상에서 9개의 위치코드들이 존재할 때, 이 위치코드들은 (그림 4)와 같이 4×4×4 RMESH의 계층 0에 속하는 프로세서에 행-우선 순서로 하나씩 저장된다. (그림 4)에서는 편의상 *Index* 값만을 보여준다.

이 연산을 위해서 다음과 같은 3단계 작업이 차례로 일어난다. 첫째, UD 버스를 이용하여 행 i에 있는 위치코드들을 계층 i로 이동시킨 후, 각 계층에서 열 버스를 이용하여 행 0으로 이동시킨다. 둘째, 홀수 계층에 있는 위치코드를 역순으로 배치시킨다. 셋째, $n \times n \times n$ RMESH를 $n \times n^2$ RMESH의 개념으로 재구성한다.

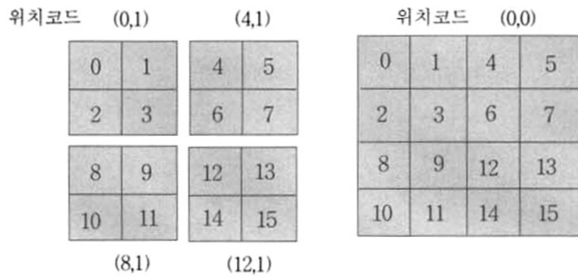
이러한 각각의 단계는 [6]에 의해 $O(1)$ 시간에 수행되며, (그림 4)가 이러한 단계를 거치면 (그림 5)와 같게 된다.

2.2 위치코드의 합병

선형 사진트리의 표현에서 두개 이상의 위치코드들이 합쳐서 하나의 큰 블록을 나타낼 수 있을 때 그 위치코드들을 하나의 위치코드가 되도록 조정해야 할 필요가 있는데, 이를 합병이라 한다.

예를 들어, (그림 6(a))와 같이 4 개의 블록에 대해 4 개의 위치코드 (0,1), (4,1), (8,1), (12,1)이 존재한다고 가정하자. 그러나 4 개의 블록이 모두 같은 색이므로 하나의 큰 블록으로 나타낼 수 있으며, (그림 6(b))와 같이 4 개의 위치코드는 하나의 위치코드 (0,0)으로 합병되어야 한다.

이와 같은 합병 연산은 [6]에 주어진 알고리즘에 의해 O(1) 시간에 수행될 수 있다.



(a) 합병전의 위치코드 (b) 합병후의 위치코드

(그림 6) 합병의 예

2.3 위치 코드의 이동

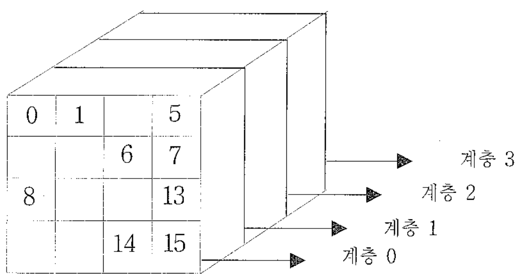
위치코드들이 $n \times n \times n$ RMESH의 계층 0에 속하는 프로세서에 하나씩 저장되어 있을 때, 위치 코드가 나타내는 *Index* 값을 행-우선 인덱스의 행과 열 번호 (i, j)로 변환한 후, 이 위치 코드를 계층 0의 프로세서 *PE(0, i, j)*로 이동한다.

예를 들어, (그림 4)에 주어진 위치 코드의 *Index*에 대해 행-우선 인덱스의 행과 열 번호를 구하면 (그림 7)과 같게 된다.

Index:	0	1	5	6	7	8	13	14	15
(i, j) :	(0,0)	(0,1)	(0,3)	(1,2)	(1,3)	(2,0)	(2,3)	(3,2)	(3,3)

(그림 7) 위치 코드의 Index를 행과 열번호로 바꾼 예

이와 같은 위치 코드의 이동은 [5]에 주어진 알고리즘에 의해 O(1) 시간에 수행될 수 있으며, 수행된 결과는 (그림 8)과 같이 이동된다.



(그림 8) 위치코드의 이동 결과

2.4 정렬

정렬은 컴퓨터와 관련된 응용에서 매우 중요한 알고리즘이므로 재구성 가능한 메쉬 구조에서도 효율적인 알고리즘의 개발에 많은 연구가 이루어져 왔다.

n 개의 데이터를 $O(1)$ 시간에 정렬하는 알고리즘을 개발하기 위해 초기에는 카우트 정렬(count sort) 방법이 3-차원 $n \times n \times n$ RMESH[11, 12], MRN[15], PARBUS[16] 구조에 적용되었다. 그 후 사용되는 프로세서의 수를 줄이기 위한 노력이 계속되었는데, Jang과 Prasanna[17]는 $n \times n$ PARBUS에서 카운트 정렬 방법을 사용하여 $O(1)$ 시간에 정렬하는 알고리즘을 제안하였고, Nigam과 Sahni[18]는 열 정렬(column sort)와 회전 정렬(rotate sort) 알고리즘을 각각 $n \times n$ RMESH에 적용하여 n 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다.

특히 Nigam과 Sahni는 회전 정렬 알고리즘을 3-차원 $n \times n \times n$ RMESH에 적용하여 n^2 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다. 이 알고리즘에서 초기의 n^2 개의 데이터는 계층 0에 속하는 $PE(0,i,j)$ $(0 \leq i,j < n)$에 존재하며, 정렬된 결과는 계층 0에 속하는 프로세서에 행-우선 순서로 하나씩 저장된다. 즉, $PE(0,0,0)$, $PE(0,0,1)$, $\cdot\cdot\cdot$, $PE(0,1,0)$, $PE(0,1,1)$, $\cdot\cdot\cdot$, $PE(0,n-1,n-1)$의 순서로 저장된다.

3. 블록위치 계산을 위한 상수 시간 알고리즘

블록위치 계산에서 이진영상은 콤포넌트 레이블링 알고리즘에 의해 레이블(*Comp*)이 부여된 상태이며, 이 레이블은 위치코드와 함께 <*Index, Level, Comp*> 형태로 저장되어 있다고 가정한다. 또한 영상을 나타내는 k $(0 < k \leq n^2)$ 개의 위치코드는 초기에 계층 0에 속하는 k 개의 프로세서에 행-우선 순서로 하나씩 저장되어 있으며, 주어진 블록은 시작위치가 (i,j)일 때 크기 $s \times t (1 \leq s,t \leq n)$와 함께 프로세서 $PE(0,i,j)$에 저장되어 있다고 가정한다. 블록위치 계산을 수행하는 상수시간 알고리즘은 알고리즘 1과 같이 10 단계로 구성된다.

[단계 1] 주어진 블록을 위치코드로 변환하여, 계층 0의 프로세서에 <*Index, Level, Comp*> 형태로 저장한다. 이때 *Comp* 필드에는 ∞ 값을 저장한다.

[단계 2] 블록의 위치코드들을 포함하여 전체 위치코드들을 <*Index, Level Comp*>에 따라 정렬한다.

[단계 3] 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층 에 있는 위치코드를 역순으로 배치한다. 그리고 $n \times n \times n$ RMESH를 $n \times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다.

[단계 4] 각 프로세서 X는 인접한 바로 다음 프로세서 Y와의 관계가 다음의 어떤 경우에 해당하는 가를 검사한다.

경우 1: *Index*(X) = *Index*(Y), *Level*(X) = *Level*(Y), X가 Y를 포함하거나 Y가 X를 포함한다고 할 수 있다. 그러므로 X를 선택하여 "포함"이라 표시한다.

경우 2: *Index*(X) = *Index*(Y), *Level*(X) < *Level*(Y)일 때, X가 Y를 포함한다. 그러므로 X를 "포함"이라 표시한다.

경우 3: *Index*(X) < *Index*(Y)일 때, 만약 *Index*(Y) < *Index*(X) + size(X)를 만족한다면, X가 Y를 포함한다. 그러므로 X를 "포함"이라 표시한다.

[단계 5] "포함"으로 표시된 프로세서를 segment의 대표 프로세서로 하여 프로세서들을 세그먼트로 분리한다.

[단계 6] 각 세그먼트 내의 대표 프로세서는 자신의 *Index*와 *Level, Comp* 값을 같은 세그먼트에 속하는 다른 프로세서들에게 브로드캐스트한다.
[단계 7] 각 세그먼트 내에 있는 프로세서들은 자신이 대표 프로세서에 의해서 포함되는가를 검사한다. 만약 대표 프로세서에 의해서 포함된다면, *Select* = 1, 그렇지 않으면 *Select* = 0 으로 설정한다.
[단계 8] 위치코드들을 계층 0의 프로세서로 이동한 후, *Select* = 0 인 위치코드들을 제거한다.
[단계 9] 단계 1에서 구한 블록의 위치코드와 단계 8의 위치코드를 비교하여 같은 *Index*의 위치코드가 같은 *Level*을 갖는지 검사한다. 같은 경우일 때, 단계 8의 위치코드에 있는 *Comp*값을 단계 1의 위치코드의 *Comp* 필드에 복사한다.
[단계 10] 단계 9에서 갱신된 단계 1의 위치코드들을 이용하여 블록위치를 나타내는 컴포넌트 레이블을 결정한다.

(알고리즘 1) 블록위치 계산 알고리즘

블록위치 알고리즘이 3-차원 RMESH에서 단계별로 수행되는 과정을 살펴보자. 단계별 수행의 예를 들기 위해 (그림 1)에 주어진 8×8 이진영상과 점선으로 표시된 블록을 사용한다.

[단계 1]은 주어진 블록을 위치코드로 변환하여, 계층 0의 프로세서에 저장하는 과정으로, 다음과 같이 2단계로 수행될 수 있다.

첫째, 계층 0에서 $PE(0,i,j)$는 N, S, W, E 버스를 이용하여 $PE(0,i+u,j+v), 0 \le u < s, 0 \le v < t$의 프로세서들과 연결되는 블록을 형성한다. 여기서 형성되는 프로세서 블록은 해당 블록의 크기와 일치한다. 그리고 하나의 신호를 블록내의 프로세서들에게 브로드캐스트하며, 신호를 받은 프로세서들은 자신의 행 번호와 열 번호를 이용하여 하나의 위치코드를 만든다. 즉 $PE(0,i',j')$는 위치코드(k,h)을 만들어 낸다. 이때 k는 row-major 인덱스 (i',j')에 대응하는 shuffled 행-우선 인덱스이고 h는$\log_4(n \times n)$이다. 예를 들어, 시작위치 (6, 2)인 크기 2×3 인 블록은 $PE(0,6,2)$, $PE(0,6,3)$, $PE(0,6,4)$, $PE(0,7,2)$, $PE(0,7,3)$, $PE(0,7,4)$과 함께 프로세스 블록을 형성하게 되고 각각 (44, 3), (45, 3), (56, 3), (46, 3), (47, 3), (58, 3)의 위치코드들을 생성하게 된다. 이 단계는 브로드캐스팅과 인덱스 변환을 위한 계산만을 수행하므로 $O(1)$ 시간에 수행될 수 있다.

둘째, 해당 블록을 나타내는 위치코드들을 합병하기 위해 2.2절의 합병 연산을 사용한다. 예를 들어, 앞 단계에서 생성된 위치코드들은 (44, 2) (56, 3) (58, 3)의 위치코드로 합병되며, *Comp* 필드가 추가되어 (44, 2, ∞) (56, 3, ∞) (58, 3, ∞)의 위치코드로 변환되어 계층 0의 프로세서에 저장된다. 합병 연산이 $O(1)$시간에 수행될 수 있으므로 이 단계 역시 상수 시간에 수행된다.

따라서 단계 1에서 블록의 위치코드 변환 연산은 $O(1)$ 시간에 수행될 수 있다.

[단계 2]는 계층 0의 프로세서에 저장된 <*Index, Level, Comp*>에 따라 정렬하여 위치코드를 행-우선 순서로 프로세서에 할당한다. 이 단계는 정렬로서 2.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다. 예를 들어, (그림 10(c))에 있는 이진영상과 주어진 블록의 위치코드들을 <*Index, Level, Comp*>에 따라 정렬하면 (그림 9)와 같다.

Index	: 6	7	12	16	32	44	44	48	56	58
Level	: 3	3	2	2	2	2	2	1	3	3
Comp	: 1	1	1	1	2	3	∞	3	∞	∞

(그림 9) 위치코드의 정렬된 결과

[단계 3]은 2.1절에서 언급한 3-차원 RMESH의 재구성에 해당하며 $O(1)$ 시간이 걸린다. (그림 9)의 정렬된 위치코드들이 [단계 3]의 수행 후, 프로세서의 연결에 따라 계층을 펼쳐 놓으면 (그림 10)과 같이 저장된다.

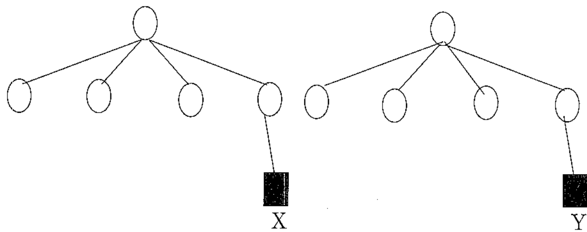
[단계 4]에서 각 프로세서는 바로 다음 프로세서와의 관계를 비교하여 경우 1, 2, 3 중 어떤 경우에 해당하는가를 검사한다. 포함 관계를 나타내기 위해 *Cover* 레지스터를 사용하는데, 만약 프로세서 X가 프로세서 Y를 포함한다면, 프로세서 X는 *Cover* 레지스터에 1, 그렇지 않으면 0을 저장한다. 그리고 (그림 10)에서 맨 마지막 프로세서의 *Cover*에는 무조건 0을 저장한다. 이 단계는 프로세서들이 바로 다음 프로세서만을 접근하여 계산하기 때문에 소요 시간은 $O(1)$이다. 단계 4의 3가지 포함 관계에 대해 살펴보자. 경우 1은 (그림 11(a))와 같이 노드 X와 노드 Y가 인덱스와 레벨이 모두 같은 경우로서, 서로 위치와 크기가 같은 블록이므로 어느 쪽이든 다른 쪽을 포함한다고 볼 수 있다. 경우 2는 (그림 11(b))와 같이 노드 X와 노드 Y가 인덱스는 서로 같으나 레벨이 서로 다른 경우로서, 레벨이 작은 위치코드의 블록이 더 크므로 정렬 후 앞에 있는 노드 X가 노드 Y를 포함하게 된다. 경우 3은 (그림 11(c))와 같이 노드 X와 노드 Y가 인덱스와 레벨이 모두 다르나 조건 *Index*(Y) < *Index*(X) + size(X)를 만족하는 경우로서 이 조건은 노드 Y의 블록이 노드 X의 블록의 일부분임을 의미한다. 이때 size(X)는 노드 X의 블록 크기를 나타낸다.

[단계 5]는 대표 프로세서, 즉 *Cover*의 값이 1인 프로세서가 자신의 좌측 스위치를 끊어 프로세서들을 세그먼트로 분리하는 과정으로서, 이 단계에서는 각 대표 프로세서가 스위치 조작만을 단순히 수행하므로 $O(1)$ 시간 걸린다.

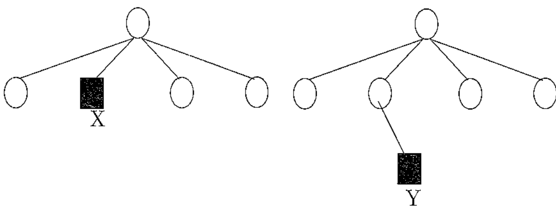
6 7 12 16 32 44 44 48 56 58

계층 1 계층 2

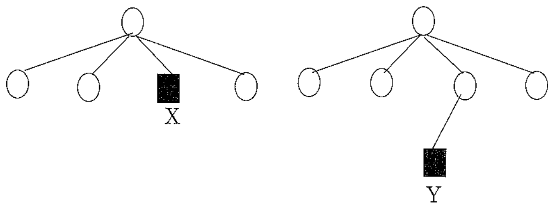
(그림 10) $n \times n^2$ RMESH 구조



(a) 경우 1



(b) 경우 2



(c) 경우 3

(그림 11) 노드 X가 노드 Y를 포함하는 3 가지 경우의 예

[단계 6]에서 각 세그먼트의 대표 프로세서는 자신의 *Index*와 *Level*, *Comp* 값을 같은 세그먼트에 속하는 다른 프로세서들에게 브로드캐스트한다. 이 단계에서는 같은 세그먼트에 속하는 프로세서 사이의 브로드캐스팅이 필요하므로 소요 시간은 $O(1)$이다.

[단계 7]은 각 세그먼트에 있는 프로세서들이 대표 프로세서에 포함되는가를 검사하는 과정으로서, 초기에 모든 프로세서들의 *Select* 레지스터에 0을 저장한다. 여기에서 0은 프로세서가 다른 프로세서에 의해서 포함되지 않았음을 의미한다. 대표 프로세서를 X, 그 외의 프로세서를 Y라 할 때, 조건 $Index(X) = Index(Y) - Index(Y) \bmod 4^{(height - Level(X))}$을 검사한다. 만약 조건을 만족한다면, 대표 프로세서 X가 프로세서 Y를 포함하므로 프로세서 Y는 *Select*의 값을 1로 변경하며, 대표 프로세서의 *Comp* 값을 자신의 *Comp* 필드에 복사한다. 단계 8에서 *Select*의 값이 1인 위치코드만 선택된다. 이 단계에서는 각 프로세서가 *Select*를 위한 계산과 선택만을 필요로 하므로 소요 시간은 $O(1)$이다.

[단계 8]은 선택된 위치코드들을 계층 0의 프로세서에 저장하는 단계로서, 계층 i의 행 0에 있는 <*Index*, *Level*, *Comp*> 값을 계층 0의 행 i로 이동시켜 처음 위치의 프로세서로 보낸다. 이 과정은 [단계 3]의 과정을 역순으로 수행할 수 있으며, 소요시간은 $O(1)$이다. 그 다음 *Select* = 0인 프로세서는 위치코드의 *Index*에 ∞ 값을 저장한다. 이 과정은 *Select* 값에 따라 위치코드를 제거하는 것으로 *Index* 값만 변경하므로$O(1)$ 시간에 수행된다.

[단계 9]는 단계 1에서 구한 블록의 위치코드와 단계 8의 위치코드를 비교하여 같은 *Index*의 위치코드가 같은 *Level*을 갖는지 검사한다. 이 과정을 위해 먼저 2.3절의 위치코드의 이동 연산을 이용하여 단계 1과 단계 8의 위치코드를 *Index*에 해당하는 프로세서로 이동한다. 그 다음, 두 개의 위치코드를 받은 프로세서는 두 위치코드의 *Level* 만을 비교한다. 일치하는 경우에만 단계 1의 위치코드의 *Comp* 필드에 단계 8의 *Comp* 값을 복사한다. 따라서 일치하지 않는 경우에는 단계 1의 위치코드의 *Comp* 필드에는 ∞ 값이 저장되어 있다. 이 과정에서는 2.3절의 이동 연산과 비교 연산만을 수행하므로 소요시간은 $O(1)$이다.

[단계 10]에서는 단계 9에서 갱신된 단계 1의 위치코드들을 이용하여 블록위치를 나타내는 컴포넌트 레이블을 결정한다. 먼저 단계 9에서 갱신된 위치코드들을 *Index* 값에 따라 정렬하면 계층 0의 프로세서에 행-우선 순서로 하나씩 저장된다. 이 과정은 정렬로서 2.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다. 그 다음, 단계 3과 같이 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층에 있는 위치코드를 역순으로 배치한다. 이 과정은 3-차원 RMESH의 재구성에 해당하며 $O(1)$ 시간이 걸린다. 컴포넌트 레이블을 결정하기 위해 각 프로세서는 자신의 *Comp* 값과 바로 다음 프로세서의 *Comp* 값을 비교하여 같으면 스위치를 연결하고 그렇지 않으면 스위치를 끊는다. 이러한 스위치 연결 상태는 세그먼트를 형성하게 되며 주어진 블록의 블록위치가 결정되기 위해서는 하나의 segment만을 형성해야 한다. 각 세그먼트의 끝에 있는 프로세서는 자신의 id와 *Comp* 값을 브로드캐스팅하며, 맨 마지막 프로세서의 id와 *Comp* 값가 맨 처음 프로세서에게 전달되어 *Comp* 값이 일치하면 그 값이 주어진 블록의 컴포넌트 레이블에 해당한다. 그렇지 않으면 레이블은 결정되지 않는다. 이 과정은 스위치의 개폐와 브로드캐스팅만을 필요로 하므로 $O(1)$에 수행될 수 있다. 따라서 이 단계는 모두 상수 시간에 수행될 수 있으므로 $O(1)$ 시간 걸린다.

(그림 12)는 (그림 9)의 다음 단계들을 차례로 수행한 결과를 단계별로 보여준다.

지금까지 알고리즘 1의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 설명하였으며, 다음과 같이 요약할 수 있다.

정리: k $(0 < k \leq n^2)$ 개의 위치코드가 $n \times n \times n$ RMESH의 계층 0에 행-우선 순서로 각 프로세서에 저장되어 있으며, 주어진 블록을 나타내는 시작위치와 크기가 주어질 때,

단계	항목											
[단계 3]	*Index*	:	6	7	12	16	32	44	44	48	56	58
	Level	:	3	3	2	2	2	2	2	1	3	3
	Comp	:	1	1	1	1	2	3	∞	3	∞	∞
[단계 4]	*Cover*	:	0	0	0	0	0	1	0	1	0	0
[단계 7]	*Select*	:	0	0	0	0	0	0	1	0	1	1
	Comp	:	-	-	-	-	-	-	3	-	3	3
[단계 8]	*Index*	:	∞	∞	∞	∞	∞	∞	44	∞	56	58
[단계 9]			(44, 2, 3) (56, 3, 3) (58, 3, 3)									
[단계 10]	블록위치	:	레이블 3									

(그림 12) 블록위치 알고리즘의 단계별 수행 과정

블록위치 계산 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

4. 결 론

본 논문에서는 3-차원 $n \times n \times n$ RMESH 구조에서 선형 사진트리로 표현된 영상에서 주어진 블록의 위치를 계산하는 알고리즘을 제안하였다. 이 알고리즘은 3차원 RMESH의 재구성, 위치코드의 합병 및 이동, 블록의 위치코드 변환, 정렬 등의 기본적인 연산들을 사용한다. 이러한 연산들은 모두 $n \times n \times n$ RMESH의 계층적 구조를 효율적으로 이용함으로써 $O(1)$ 상수 시간 복잡도를 가진다.

본 논문에서 제안하는 블록위치 알고리즘은 10 단계로 구성되어 있으며, 콤포넌트 레이블링 알고리즘에 의해 레이블이 부여된 상태에서 시작한다. 각 단계는 효율적인 위치 코드 라우팅을 위하여 기본 연산들을 사용하여 각 단계를 상수 시간에 수행할 수 있으며, 제안한 알고리즘은 $O(1)$시간 복잡도를 가진다. 특히 이 알고리즘은 위치코드를 영상으로 변환하지 않고 직접 위치코드에 기본 연산들을 적용함으로써 오버헤드를 줄일 수 있으므로, 영상 처리의 신속한 처리에 기여할 수 있다.

참 고 문 헌

[1] H. Samet, Application of Spatial Data Structures, Computer Graphics, Image Processing, and GIS. Addison-Wesley, 1990.

[2] H. Samet, The Design and Analysis of Spatial Data Structures, Addison-Wesley, 1990.

[3] I. Gargantini, "Translation, rotation, and superposition of linear quadtrees," International Journal of Man-Machine Studies Vol.18, No.3, pp.253-263, 1985.

[4] T.R. Walsh, "Efficient axis-trnaslation of binary digital pictures by blocks in linear quadtree representation," Computer Vision, Graphics and Image Processing Vol.41, No.3, pp.282-292, 1988.

[5] 김 명, 장주욱, "재구성가능 매쉬에서 O(1) 시간 복잡도를 갖는 이진영상/사진트리 변환 알고리즘," 정보과학회논문지(A), 제23권, 제5호, pp.454-466, 1996.

[6] 공헌택, 우진운, "RMESH 구조에서의 선형 사진트리 구축을 위한 상수 시간 알고리즘," 정보처리 논문지, 제4권, 제9호, pp. 2247-2258, 1997.

[7] 김경훈, 우진운, "RMESH 구조에서 unaligned 선형사진트리의 alignment를 위한 상수시간 알고리즘," 정보과학회논문지, 제31권 1,2호, pp.10-18, 2004.

[8] G. Kim, et. al, "Finding Neighbor Blocks on 3D RMESH in Constant Time without Condition," Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications, pp.2225-2231, 2001.

[9] J. Jenq, "Clustering Using Self Organization Map on Rmesh," Proceedings of International Conference on Computers and Their Applications, pp.91-96, 2005.

[10] R. Miller, V. Prasanna-Kumar, D. Reisis, and Q. Stout, "Parallel Computation on Reconfigurable Meshes," IEEE Transactions on Computers, Vol.42, No.6, pp.678-692, 1993.

[11] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for The Hough Transform," Proceedings of International Conference on Parallel Processing, Vol.III, pp.34-41, 1991.

[12] 김수환, "구멍이 있는 다각형에서 가시성 다각형을 구하는 상수 시간 RMESH 알고리즘," 정보과학 2000년 가을학술발표논문집, 2000.

[13] 김홍근, 조유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘," 정보학회 논문지, 제20권 11호, pp.1693-1701, 1993.

[14] J. Jang, H. Park, and V. Prasanna, "A Fast Algorithm for Computing Histogram on a Reconfigurable Mesh," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, No.2, pp.97-106, 1995.

[15] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration," Journal of Parallel and Distributed Computing, 13, pp.139-153, 1991.

[16] J. Jang and V. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Meshes," Proceedings 6th International Parallel Processing Symposium, 1992.

[17] M. Nigam and S. Sahni, "Sorting n Numbers On $n \times n$

Reconfigurable Meshes With Buses," Proceedings 7th International Parallel Processing Symposium, pp.174-181, 1993.

[18] Sanjay Ranka and Sartaj Sahni, Hypercube algorithms with applications to image processing and pattern recognition, Springer-Verlag, New York, 1990.

한 선 미

e-mail: hseonmi@dankook.ac.kr

1998년 한국방송통신대학교 경영학과 학사

2002년 단국대학교 컴퓨터과학전공 석사

2006.3~현재 단국대학교 컴퓨터과학전공 박사과정

관심분야 : 병렬알고리즘, 분산 및 병렬처리, 컴퓨터 이론

우 진 운

e-mail: jwwoo@dankook.ac.kr

1980년 서울대학교 수학교육과 학사

1989년 미국 University of Minnesota 전산학과 박사

1980년~1983년 대한항공 및 국토개발연구원 전산실 근무

1989년~현재 단국대학교 정보컴퓨터학부 교수

관심분야 : 병렬알고리즘, 분산 및 병렬처리, 인터넷 응용