RMESH 구조에서 선형 사진트리의 윈도우 연산을 위한 상수시간 알고리즘

(Constant Time Algorithm for the Window Operation of Linear Quadtrees on RMESH)

김 경 훈 † 우 진 운 ††
(Kyung Hoon Kim) (Jin Woon Woo)

요 약 계층적 자료구조인 사진트리는 이진 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. 윈도우 연산은 영상에서부터 윈도우에 표시된 부분 영상을 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 이진 영상의 윈도우 연산을 수행하는 효율적인 알고리즘을 제안한다. 이 알고리즘은 $n\times n\times n$ RMESH의 계층구조에서 선형 사진트리의 위치코드들을 효율적으로 전송할 수 있는 기본적인 연산들을 이용함으로써 상수 시간의 시간복잡도를 갖는다.

키워드 : RMESH, 선형 사진트리, 위치코드, 윈도우 연산

Abstract Quadtree, which is a hierarchical data structure, is a very important data structure to represent binary 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 window operation is one of important geometry operations in image processing, which extracts a sub-image indicated by a window in the image. In this paper, we present an algorithm to perform the window operation of binary images represented by quadtrees, using three-dimensional n×n×n processors on RMESH(Reconfigurable MESH). This algorithm has constant-time complexity by using efficient basic operations to route the locational codes of quardtree on the hierarchical structure of n×n×n RMESH.

Key words : RMESH, linear quadtree, locational code, window operation

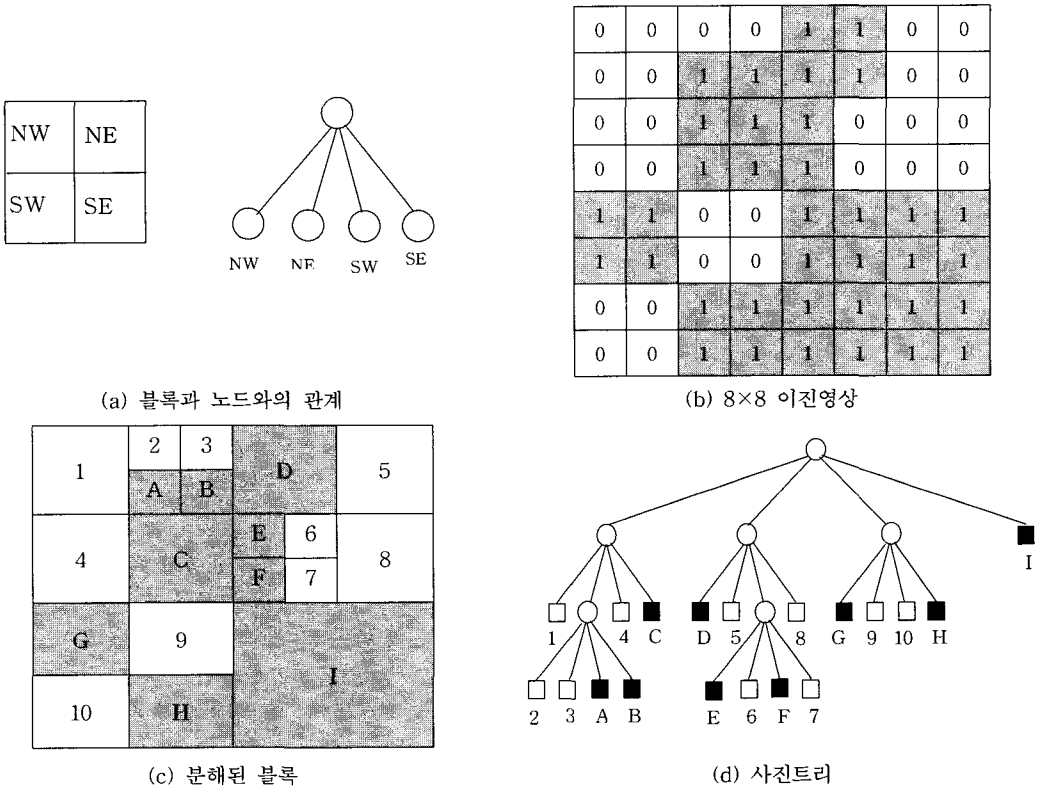
1. 서 론

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 분해(decomposition)하기 때문에 이진영상을 표현하는데 매우 유용한 자료구조이다[1, 2].

$n\times n$ 이진 영상($n=2^k$, k는 양의 정수)에 대한 사진트리는 다음과 같이 정의된다. 사진트리의 루트(root) 노드는 전체 영상을 표현하는 것으로, 만약 영상의 모든 픽셀(pixel)들이 같은 색을 가진다면 루트 노드는 자식 노드를 갖지 않지만, 서로 다른 색을 가진다면 루트 노드는 4 개의 자식 노드를 갖는다. 자식 노드는 왼쪽부터 각각 영상의 NW, NE, SW 및 SE 블록(block)의 색을 표현한다(그림 1(a) 참조). 이와 같은 분해 과정은 노드가 표현하는 블록이 단지 하나의 공통된 색을 가지게 될 때까지 4개의 자식 노드에 대해 순환적으로 적용된다.

예를 들면, 8×8 이진 영상을 사진트리로 표현해 보

· 이 연구는 2000학년도 단국대학교 대학연구비의 지원으로 연구되었음
† 종신회원 : 단국대학교 전산통계학과
khkim@kice.re.kr
†† 종신회원 : 단국대학교 전산통계학과 교수(연구책임자)
jwwoo@dankook.ac.kr
논문접수 : 2001년 9월 3일
심사완료 : 2002년 1월 14일



(a) 블록과 노드와의 관계 (b) 8×8 이진영상

(c) 분해된 블록 (d) 사진트리

그림 1 이진 영상과 사진트리와의 관계

자. 일반적으로 이진 영상에서는, 그림 1(b)와 같이 WHITE는 0으로 BLACK은 1로 표현한다. 그림 1(c)는 그림 1(b)를 분해한 최종 결과를 블록으로 나타낸 것이고, 그림 1(d)는 그림 1(c)의 블록에 대해 사진트리로 표현한 것이다. 그림 1(c)와 (d)에서 WHITE 블록은 숫자, BLACK 블록은 영문자로 구별하였다. 그림 1(d)에서 사각형 BLACK 노드는 블록 전체가 1로 구성되어 있음을 의미하며, 사각형 WHITE 노드는 블록 전체가 0으로 구성되어 있음을 의미한다. 그리고 원형 노드는 내부 노드로서 GRAY 노드라 한다.

사진트리에서 레벨(level)은 루트 노드에서 임의의 노드까지의 거리로 정의하며, 루트 노드의 레벨은 0으로 한다. 그리고 사진트리의 높이는 $\log_4(n\times n)$값으로 정의한다. 사진트리의 높이가 h일 때 레벨이 l인 노드의 블록 크기를 $2^{(h-l)}\times 2^{(h-l)}$로 계산할 수 있다. 다시 말해서 이 블록은 $4^{(h-l)}$ 개의 픽셀들의 모임을 표현한다. 예를 들면, 그림 1(d)에서 노드 I는 4×4, 노드 D는 2×2, 노드 E는 1×1의 블록 크기를 갖는다.

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 row-major 인덱스이다.

$n=2^i$, $i>0$인 $n\times n$ 이진 영상의 픽셀에 인덱스를 부여하는 방법은 여러 가지가 있으나, 그 중 가장 널리 사용되는 방법은 row-major 인덱스, column-major 인덱스, shuffled row-major 인덱스이다. Row-major 인덱

스 방법은 r 행과 c 열의 픽셀에 $r\times n+c$의 인덱스를 부여하고, column-major 인덱스 방법은 r행과 c열의 픽셀에 $c\times n+r$의 인덱스를 부여하며, shuffled row-major 인덱스 방법은 r과 c의 이진 표현이 $r_{i-1}\cdots r_1\ r_0$과 $c_{i-1}\cdots c_1\ c_0$, ($i=\log_2 n$)일 때, 해당 픽셀에 이진수 표현으로 $r_{i-1}\ c_{i-1}\cdots r_i\ c_1\ r_0\ c_0$의 인덱스를 부여한다. 따라서 이러한 인덱스들 사이의 상호 변환이 가능하며, 인덱스의 위치를 나타내는 행과 열 번호도 쉽게 구할 수 있다.

그림 1(b)의 8×8 이진 영상에 shuffled row-major 인덱스를 부여하면 그림 2(a)와 같고, 그림 1(d)의 BLACK 노드들을 위치코드를 이용하여 선형 사진트리 표현으로 나타내면 그림 2(b)와 같다.

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

(a) 픽셀에 부여된 shuffled-row major 인덱스

BLACK

노드	A	B	C	D	E	F	G	H	I
Index	6	7	12	16	24	26	32	44	48
Level	3	3	2	2	3	3	2	2	1

(b) 선형 사진트리

그림 2 위치코드를 이용한 선형 사진트리 표현

그림 2(b)와 같이 선형 사진트리의 표현에서 WHITE 노드에 대한 정보는 저장하지 않고 BLACK 노드에 대한 정보만을 저장하는 이유는 사진트리를 다시 구축하지 않고도 BLACK 노드에 대한 정보를 이용하여 WHITE 노드에 대한 정보를 쉽게 구할 수 있으며, 또한 BLACK 노드에 대한 정보만을 저장하므로써 저장 공간을 최소화할 수 있는 장점이 있기 때문이다.

윈도우 연산은 영상에서부터 윈도우에 표시된 부분 영역을 추출하는 연산으로, 영상이 위치코드로 주어져 있을 때 윈도우 연산의 결과는 추출된 부분 영상을 위치코드로 출력하여야 한다.

예를 들어, 그림 2(a)의 이진영상에 대하여 그림 3(a)에서 점선으로 표시된 윈도우의 연산을 살펴보자. 이때 이진영상의 위치코드와 주어진 윈도우의 위치코드를 나타내면 그림 3(b)와 같으며, 윈도우 연산을 적용하게 되면 그림 3(c)와 같은 결과를 얻을 수 있다.

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

(a) 이진영상의 예

구 분	위치코드
선형사진트리	(6, 3) (7, 3) (12, 2) (16, 2) (24, 3) (26, 3) (32, 2) (44, 2) (48, 1)
윈도우	(12, 2) (24, 2) (36, 2) (48, 2)

(b) 선형사진트리와 윈도우의 위치코드

(12, 2), (24, 3), (26, 3), (48, 2)

(c) 윈도우 연산의 결과

그림 3 윈도우 연산의 예

윈도우 연산은 영상 처리의 응용에서 중요하게 사용되는 기하학적 연산에 속한다. 따라서 현재까지 단일 프로세서 뿐만 아니라 병렬 컴퓨터 구조에서 $n\times n$ 이진 영상의 윈도우 연산을 수행하는 알고리즘들이 제안되었다[1, 2, 3, 4]. 특히 기존의 알고리즘은 주로 윈도우의 크기를 $2^k\times 2^k$으로 제한하고 있으나 본 논문에서 제안하는 알고리즘에서는 윈도우의 크기를 제한하고 있지 않다.

본 논문에서는 이진 영상의 윈도우 연산을 수행하는 상수 시간 알고리즘을 제안하며, 이 알고리즘은 $n\times n\times n$ RMESH에서 $O(1)$시간 복잡도를 갖는다. 지금까지 $n\times n\times n$ RMESH 상에서 상수 시간을 갖는 이진 영상 알고리즘들이 개발되었는데, 이진 영상과 선형 사진트리 사이의 상호 변환 알고리즘[5, 6], 가시성 다각형을 구하는 알고리즘[7] 등을 들 수 있다.

2. RMESH 구조

RMESH는 Reconfigurable MESH의 약어이다. 기존

의 메쉬(mesh) 구조에 동적으로 재구성 가능한 버스 시스템을 결합한 구조로서 Miller, Prasanna-Kumar, Reisis, Stout에 의하여 제안되었으며[8], 구조적인 장점 때문에 다양한 분야에서 연구되었고 효율적인 알고리즘들이 개발되었다[9, 10, 11]. 또한 버스 시스템의 재구성 방법 면에서 서로 차이를 갖는 PARBUS 구조와 MRN 구조가 제안되었다[12, 13].

2.1 2-차원 RMESH

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

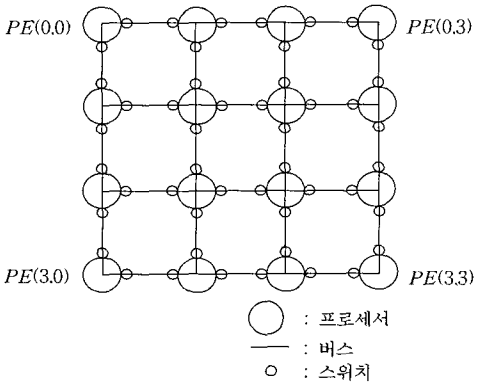


그림 4 4×4 RMESH 구조

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 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.

2.2 3-차원 RMESH

2-차원 RMESH를 확장하여 3-차원 RMESH를 구성할 수 있다. 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 l에서의 행과 열의 인덱스이다. 예를 들어, 그림 5는 4 RMESH를 보여준다. 버스 스위치들은 기본적으로 2-차원 RMESH와 같이 N, S, W, E 스위치가 존재하며, 추가적으로 각 프로세서마다 계층을 연결하는 U(up)와 D(down) 스위치가 존재한다. 그리고 모든 프로세서의 N, S, W, E 스위치를 끊고 U와 D 스위치를 연결하면 여러 개의 서브버스가 형성되는데, 이를 UD 버스라 한다.

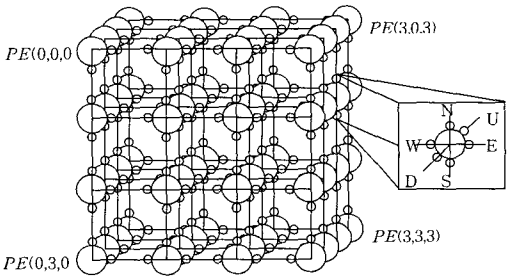


그림 5 4×4×4 RMESH 구조

3. 기본적인 연산

여기서는 3차원 RMESH 구조에서 윈도우 연산을 효율적으로 수행하기 위해 필요한 기본적인 연산들을 알아 본다.

3.1 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 row-major 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

permute함으로써 만들어진다.

예를 들어, 4×4의 영상에서 9개의 위치코드들이 존재할 때, 이 위치코드들은 그림 6과 같이 4×4×4 RMESH의 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. 그림 6에서는 편의상 *Index* 값만을 보여준다.

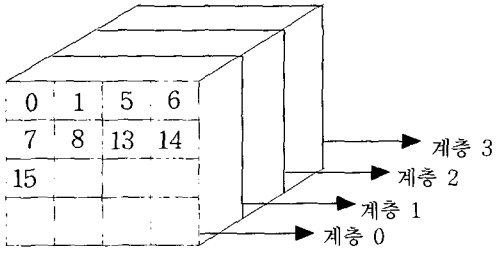


그림 6 4×4×4 RMESH상의 계층 0의 초기 상태

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 permute시킨다. 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 6이 이러한 단계를 거치면 그림 7과 같게 된다.

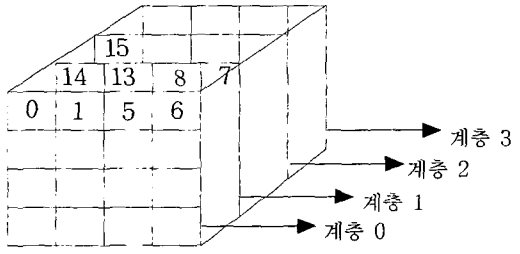
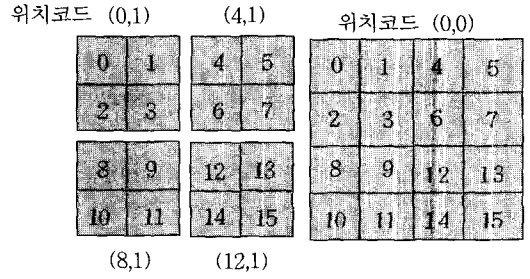


그림 7 4×4×4 RMESH의 재구성 결과

3.2 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 8(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 8(b)와 같이 4 개의 위치코드는 하나의 위치코드 (0,0)으로 합병되어야 한다.



a) 합병전의 위치코드 (b) 합병후의 위치코드

그림 8 합병의 예

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

3.3 윈도우의 위치코드 변환

윈도우는 $n\times n$ 영상에서 $s\times t(1\le s,\ t\le n)$크기의 부분영상을 의미하는 것으로 왼쪽 상단이 i행과 j열일 때 (i, j)를 시작위치라 한다. 예를 들어, 그림 9에서 점선으로 표시된 윈도우는 시작위치가 (2, 2)이고 크기가 4×4이다. 위치코드와 관련된 연산에서 윈도우 역시 위치코드로 표현해야할 경우가 있다. 즉, 시작위치가 (i, j)인 크기 $s\times t(1\le s,\ t\le n)$의 윈도우를 위치코드로 변환하는 것이다. 예를 들어, 그림 9에서 윈도우는 시작위치가 (2, 2)이고 크기가 4×4이므로, 윈도우를 위치코드로 변환하면 (12, 2), (24, 2), (36, 2), (48, 2)로 된다.

0	1	4	5	16	17	20	21
2	3	6	7	18	19	22	23
8	9	12	13	24	25	28	29
10	11	14	15	26	27	30	31
32	33	36	37	48	49	52	53
34	35	38	39	50	51	54	55
40	41	44	45	56	57	60	61
42	43	46	47	58	59	62	63

그림 9 시작위치가 (2, 2)인 윈도우의 예

먼저 이 연산이 수행되기 전에, 시작 위치가 (i,j)이고 크기가 $s\times t(1\le s,\ t\le n)$인 윈도우가 계층 0의 프로세서 $PE(0,i,j)$에 저장되어 있는 것으로 가정한다.

이 연산은 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 윈도우의 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 row-major 인덱스이고 h는 $\log_4(n \times n)$이다. 예를 들어, 시작위치 (2, 2)인 크기 4×4인 윈도우는 $PE(0, 2, 2)$, ..., $PE(0, 2, 5)$, $PE(0, 3, 2)$, ..., $PE(0, 3, 5)$, $PE(0, 4, 2)$, ..., $PE(0, 4, 5)$, $PE(0, 5, 2)$, ..., $PE(0, 5, 5)$와 함께 블록을 형성하게 되고 각각 (12, 3), ..., (25, 3), (14, 3), ..., (27, 3), (36, 3), ..., (49, 3), (38, 3), ..., (51, 3)의 위치코드들을 생성하게 된다. 이 단계는 브로드캐스팅과 인덱스 변환을 위한 계산만을 수행하므로 $O(1)$ 시간에 수행될 수 있다.

둘째, 해당 윈도우를 나타내는 위치코드들을 합병하기 위해 3.2절의 합병 연산을 사용한다. 예를 들어, 앞 단계에서 생성된 위치코드들은 (12, 2), (24, 2), (36, 2), (48, 2)의 위치코드로 합병된다. 합병 연산이 $O(1)$시간에 수행될 수 있으므로 이 단계 역시 상수 시간에 수행된다.

따라서 윈도우의 위치코드 변환 연산을 $O(1)$ 시간에 수행될 수 있다.

3.4 정렬

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

n개의 데이터를 $O(1)$ 시간에 정렬하는 알고리즘을 개발하기 위해 초기에는 count sort 방법이 3-차원 $n \times n \times n$ RMESH[9, 10], MRN[13], PARBUS[14] 구조에 적용되었다. 그 후 사용되는 프로세서의 수를 줄이기 위한 노력이 계속되었는데, Jang과 Prasanna[15]는 $n \times n$ PARBUS에서 column sort 방법을 사용하여 $O(1)$ 시간에 정렬하는 알고리즘을 제안하였고, Nigam과 Sahni [16]는 column sort와 rotate sort 알고리즘을 각각 $n \times n$ RMESH에 적용하여 n개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다.

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

4. 상수 시간 RMESH 알고리즘

윈도우 연산에서 이진영상을 나타내는 $k(0 < k \le n^2)$ 개의 위치코드는 초기에 계층 0에 속하는 k 개의 프로세서에 row-major 순서로 하나씩 저장되어 있으며, 윈도우는 시작위치가 (i, j)일 때 크기와 함께 프로세서 $PE(0, i, j)$에 저장되어 있고 가정한다. 윈도우 연산을 수행하는 RMESH 알고리즘은 알고리즘 1과 같이 8 단계로 구성된다.

[단계 1] 주어진 윈도우를 위치코드로 변환하여, 계층 0의 프로세서에 저장한다.

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

[단계 3] 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층 에 있는 위치코드를 역순으로 permute한다. 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 해당하는가를 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 segment로 분리한다.

[단계 6] 각 segment 내의 대표 프로세서는 자신의

*Index*와 *Level* 값을 같은 segment에 속하는 다른 프로세서들에게 브로드캐스트한다.

[단계 7] 각 segment 내에 있는 프로세서들은 자신이 대표 프로세서에 의해서 포함되는가를 검사한다. 만약 대표 프로세서에 의해서 포함된다면, 대표 프로세서의 위치코드를 제거하고, 그렇지 않으면 자신의 위치코드를 제거한다.

[단계 8] 위치코드들을 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.

알고리즘 1 윈도우 연산 알고리즘

윈도우 연산 알고리즘이 3-차원 RMESH에서 단계별로 수행되는 과정을 살펴보자. 단계별 수행의 예를 들기 위해 그림 3에 주어진 8×8 이진영상과 점선으로 표시된 윈도우를 사용한다.

[단계 1]은 주어진 윈도우를 위치코드로 변환하여, 계층 0의 프로세서에 저장하는 과정이다. 3.3절에 설명된 윈도우의 위치코드 변환 연산을 사용하여 $O(1)$ 시간에 수행할 수 있다. 그림 3에서 점선으로 표시된 윈도우는 (12, 2), (24, 2), (36, 2), (48, 2)의 위치코드로 변환되어 계층 0의 프로세서에 저장된다.

[단계 2]는 계층 0의 프로세서에 저장된 <*Index*, *Level*>에 따라 정렬하여 위치코드를 row-major 순서로 프로세서에 할당한다. 이 단계는 정렬로서 3.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다. 예를 들어, 그림 3(b)에 있는 이진영상과 윈도우의 위치코드들을 <*Index*, *Level*>에 따라 정렬하면 그림 10과 같다.

[단계 2] *Index* : 6 7 12 12 16 24 24 26 32 36 44 48 48

Level : 3 3 2 2 2 2 3 3 2 2 2 1 2

그림 10 위치코드의 정렬된 결과

[단계 3]은 3.1절에서 언급한 3-차원 RMESH의 재구성에 해당하며 $O(1)$ 시간이 걸린다. 그림 10의 정렬된 위치코드들이 [단계 3]의 수행 후, 프로세서의 연결에 따라 계층을 펼쳐 놓으면 그림 11과 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

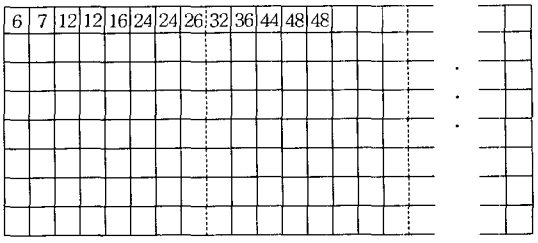
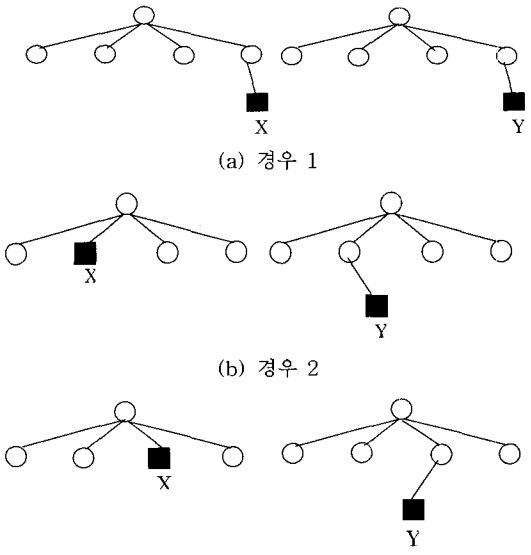


그림 11 $n \times n^2$ RMESH 구조



(a) 경우 1

(b) 경우 2

(c) 경우 3

그림 12 노드 X가 노드 Y를 포함하는 3 가지 경우의 예

포함한다면, 프로세서 X는 *Cover* 레지스터에 1, 그렇지 않으면 0을 저장한다. 그리고 그림 11에서 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 12(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 12(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 12(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 segment로 분리하는 과정으로서, 이 단계에서는 각 대표 프로세서가 스위치 조작만을 단순히 수행하므로 $O(1)$ 시간 걸린다.

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

[단계 7]은 각 segment에 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, 그외의 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, 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 row-major 순서로 저장하는 단계로서, 계층 i의 행 0에 있는 <*Index*, *Level*> 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*=0 인 프로세서는 위치코드의 *Index*에 ∞값을 저장한 후, 위치코드를 *Index* 값에 따라 정렬하고, *Index* 값이 ∞인 위치코드를 제거한다. 이 과정은 정렬로서 3.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$시간에 수행된다.

그림 13은 그림 3의 선형 사진트리와 윈도우에 대한 윈도우 연산이 수행되는 과정을 단계별로 보여준다.

[단계 3] *Index* : 6 7 12 12 16 24 24 26 32 36 44 48 48
Level : 3 3 2 2 2 2 3 3 2 2 2 1 2
[단계 4] *Cover* : 0 0 1 0 0 1 0 0 0 0 0 1 0
[단계 7] *Select* : 0 0 0 1 0 0 1 1 0 0 0 0 1
[단계 8] *Index* : ∞ ∞ ∞ 12 ∞ ∞ 24 26 ∞ ∞ ∞ ∞ 48
결과 : (12, 2) (24, 3) (26, 3) (48, 2)

그림 13 윈도우 연산 알고리즘의 단계별 수행 과정

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

정리 1: $k(0<k\leq n^2)$개의 위치코드가 $n\times n\times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 저장되어 있으며, 윈도우를 나타내는 시작위치와 크기가 주어질 때, 윈도우 연산 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

5. 결론

본 논문에서는 3-차원 $n\times n\times n$ RMESH 구조에서 선형 사진트리로 표현된 이진 영상의 윈도우 연산을 수행하는 알고리즘을 제안하였다. 이 알고리즘은 3차원 RMESH의 재구성, 위치코드의 합병, 윈도우의 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 윈도우 연산 알고리즘은 8 단계로 구성되어 있으며, 각 단계는 효율적인 위치 코드 라우팅을 위하여 기본 연산들을 사용한다. 따라서 각 단계를 상수 시간에 수행할 수 있으며, 본 알고리즘은 $O(1)$ 상수 시간 복잡도를 가진다. 특히 이 알고리즘에서는 위치코드를 이진 영상으로 변환하지 않고 직접 위치코드에 기본 연산들을 적용함으로써 시간 지체를 줄인다.

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] D. Rogers, Procedural Elements for Computer Graphics, McGraw-Hill, New York, 1985.
[4] H. Samet and M. Tamminen, "Computing Geometric Properties of Images Represented by Linear Quadtrees," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-7, No. 2, March

1985.
[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] 김수환, "구멍이 있는 다각형에서 가시성 다각형을 구하는 상수 시간 RMESH 알고리즘," 정보과학 2000년 가을학술발표논문집, 2000.
[8] 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.
[9] 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.
[10] J. Jenq and S. Sahni, "Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching," Proceedings 5th International Parallel Processing Symposium, pp.208-215, 1991.
[11] 김 홍근, 조 유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘," 정보학회 논문지, 제20권 11호, pp.1693-1701, 1993.
[12] 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.
[13] 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.
[14] B. Wang, G. Chen, and F. Lin, "Constant Time Sorting on a Processor Array with a Reconfigurable Bus System," Information Processing Letters, 34, 4, pp.187-190, 1990.
[15] J. Jang and V. Prasanna, "An Optimal Sorting Algorithm on Reconfigurable Meshes," Proceedings 6th International Parallel Processing Symposium, 1992.
[16] 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.

김 경 훈
1988년 숭실대학교 전산공학과(학사). 1993년 한양대학교 교육대학원 전산교육(석사). 1998년 ~ 현재 한국교육과정 평가원 전산부장. 2000년 ~ 현재 단국대학교 전산통계학과 박사 과정(수료). 관심분야는 알고리즘 및 병렬 알고리즘, 컴퓨터 교육

우 진 운
1980년 서울대학교 수학교육과(학사). 1989년 미국 University of Minnesota 전산학과(박사). 1980년 ~ 1983년 대한항공 및 국토개발연구원 전산실 근무. 1989년 ~ 현재 단국대학교 전산통계학과 교수. 관심분야 : 알고리즘 및 병렬알고리즘, 인터넷 응용