선형 사진트리의 선형이동을 위한 상수시간 RMESH 알고리즘

김 경 훈[†] · 우 진 운[††]

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. 선형이동은 영상을 주어진 거리만큼 이동시키는 연산으로, 영상 처리의 응용에서 중요하게 사용되는 연산에 속한다. 본 논문에서는 RMESH (Reconfigurable MESH) 구조에서 3-차원 $n\times n\times n$ 프로세서를 사용하여 선형 사진트리로 표현된 이진 영상의 선형 이동을 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 시간 복잡도를 갖는다.

Constant Time RMESH Algorithm for Linear Translation of Linear Quadtrees

Kyung Hoon Kim[†] · Jin Woon Woo[††]

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 linear translation is one of important operations in image processing, which moves the image by a given distance. In this paper, we present an algorithm to perform the linear translation of binary 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 route the locational codes of quardtree on the hierarchical structure of $n\times n\times n$ RMESH.

키워드 : RMESH, 선형 사진트리(Linear Quadtree), 위치코드(Locational Code), 선형이동(Linear Translation)

1. 서 론

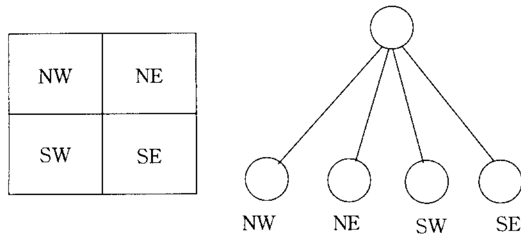
계층적 자료구조는 컴퓨터 그래픽, 영상처리, 지형처리, 패턴 인식 및 로보트 공학분야 등의 자료를 표현하는데 매우 적합한 기법이다. 특히 계층적 자료구조 중의 하나인 사진트리(quadtree)는 디지털 영상을 규칙적으로 분해(decomposition)하기 때문에 이진영상을 표현하는데 매우 유용한 자료구조이다[1, 2].

$n\times n$ 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 4개의 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 보자. 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 (그림 1)(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

※ 이 연구는 2003학년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 정 회 원 : 단국대학교 대학원 컴퓨터과학부 박사과정 수료
†† 종신회원 : 단국대학교 정보컴퓨터과학부 교수
논문접수 : 2003년 3월 24일, 심사완료 : 2003년 6월 17일

있음을 의미한다. 그리고 원형 노드는 내부 노드로서 GRAY 노드라 한다.



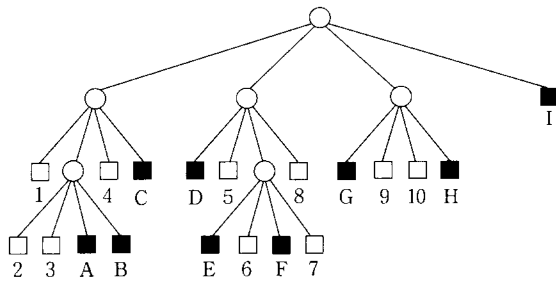
(a) 블록과 노드와의 관계

0	0	0	0	1	1	0	0
0	0	1	1	1	1	0	0
0	0	1	1	1	0	0	0
0	0	1	1	1	0	0	0
1	1	0	0	1	1	1	1
1	1	0	0	1	1	1	1
0	0	1	1	1	1	1	1
0	0	1	1	1	1	1	1

(b) 8×8 이진영상

1 2 3 A B D 5
4 C E 6 F 7 8
G 9 10 H I

(c) 분해된 블록



(d) 사진트리

(그림 1) 이진 영상과 사진트리와의 관계

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 인덱스 방법은 r행과 c열의 픽셀에 $r \times n + c$의 인덱스를 부여하고, columnmajor 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 $c_{i-1} \cdots c_1 c_0$, $(i = \log_2 n)$일때, 해당 픽셀에 이진수 표현으로 $r_{i-1} c_{i-1} \cdots r_1 c_1 r_0 c_0$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 저장함으로써 저장 공간을 최소화할 수 있는 장점이 있기 때문이다.

선형 이동은 x축과 y축 방향으로 주어진 거리만큼 평행 이동 시키는 연산으로, 영상이 위치코드로 주어져 있을 때 선형 이동의 결과는 이동된 영상을 위치코드로 출력하여야 한다.

예를 들어, (그림 2)(a)의 이진영상에 대하여 x축 왼쪽으로 2, y축 아래로 1만큼 선형 이동하는 연산을 살펴보자. 이때 모든 픽셀들이 왼쪽으로 2만큼, 아래쪽으로 1만큼 평행 이동되어야 한다. 즉, r행과 c열의 픽셀은 $r-2$행과 $c+1$열의 픽셀이 된다. 그리고 영상의 우측과 상단은 픽셀 값 0으로 채우며, 영상의 좌측과 하단에서 범위를 벗어나는 부분은 절단된다. 선형 이동한 이진영상을 나타내면 (그림 3)(a)와 같으며, 이를 위치코드로 나타내면 (그림 3)(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) 선형 이동 후의 이진영상

구 분	위 치 코 드
선형 이동 전	(6, 3) (7, 3) (12, 2) (16, 2) (24, 3) (26, 3) (32, 2) (44, 2) (48, 1)
선형 이동 후	(6, 3) (7, 3) (8, 2) (12, 3) (13, 3) (14, 3) (32, 3) (33, 3) (36, 3) (38, 3) (39, 3) (42, 3) (43, 3) (44, 2) (50, 3) (51, 3) (56, 2)

(b) 선형 이동 후의 위치코드

(그림 3) 선형 이동의 예

선형 이동은 영상 처리의 응용에서 중요하게 사용되는 연산에 속한다. 따라서 현재까지 단일 프로세서 뿐만 아니라 병렬 컴퓨터 구조에서 $n \times n$ 이진 영상의 선형 이동을 수행하는 알고리즘들이 제안되었다[1-4].

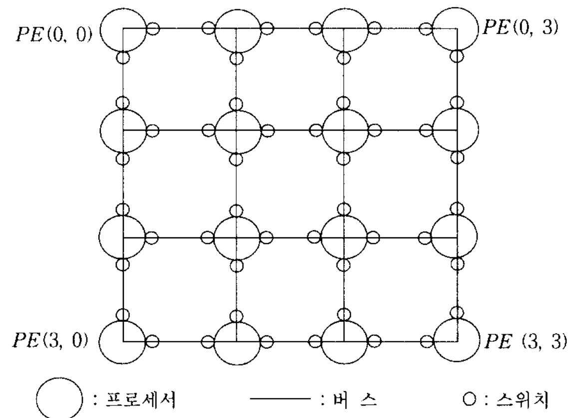
본 논문에서는 이진 영상의 선형 이동을 수행하는 상수 시간 알고리즘을 제안하며, 이 알고리즘은 $n \times n \times n$ RMESH에서 $O(1)$시간 복잡도를 갖는다. 지금까지 $n \times n \times n$ RMESH 상에서 상수 시간을 갖는 이진 영상 알고리즘들이 개발되었는데, 이진 영상과 선형 사진트리 사이의 상호 변환 알고리즘[5, 6], 윈도우 연산을 구하는 알고리즘[7] 등을 들 수 있다.

2. RMESH 구조

RMESH는 Reconfigurable MESH의 약어이다. 기존의 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[9-11]. 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 4)는 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

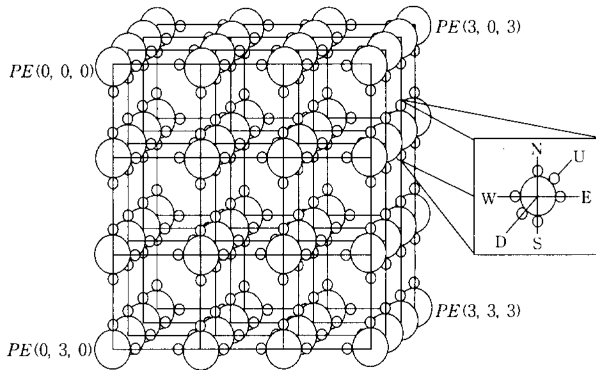
브로드캐스트 버스상의 통신 제어를 위하여 버스 스위치가 있다. 버스 스위치들은 각 프로세서의 상, 하, 좌, 우에 하나씩 존재하는데, 이를 각각 N(north), S(south), W(west), E(east)라 한다. 버스 스위치는 각 프로세서의 소프트웨어에 의하여 $O(1)$ 시간에 조작되며, 스위치의 개폐 여부에 따라 브로드캐스트 버스를 다수의 서브버스(subbus)들로 재구성이 가능하다. 예를 들어, 각 프로세서가 자신의 S와 N 스위치를 끊고 E와 W 스위치를 연결한다면 여러 개의 서브버스가 형성되는데, 이를 행 버스(row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

간에 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 5)는 4×4×4 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 서브 버스가 형성되는데, 이를 UD 버스라 한다.



(그림 5) 4×4×4 RMESH 구조

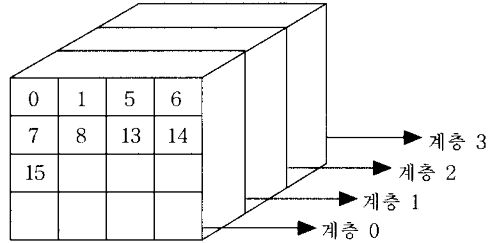
3. 기본적인 연산

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

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.

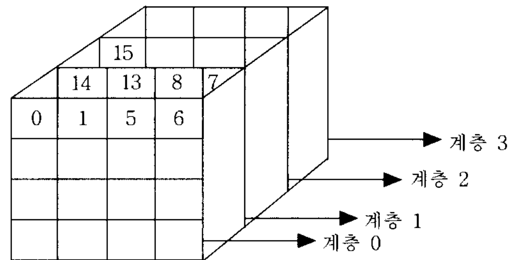
예를 들어, 4×4의 영상에서 9개의 위치코드들이 존재할 때, 이 위치코드들은 (그림 6)과 같이 4×4×4 RMESH의 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. (그림 6)에서는 편의상 *Index* 값만을 보여준다.



(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 permute 시킨다. 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 6)이 이러한 단계를 거치면 (그림 7)과 같게 된다.



(그림 7) 4×4×4 RMESH의 재구성 결과

3.2 위치 코드의 이동

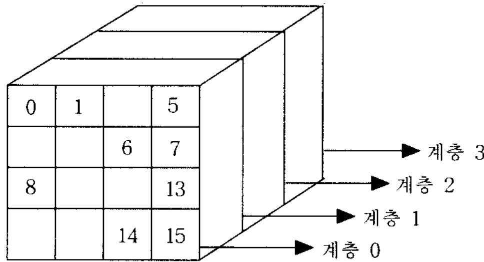
위치코드들이 $n \times n \times n$ RMESH의 계층 0에 속하는 프로세서에 하나씩 저장되어 있을 때, 위치 코드가 나타내는 shuffled row-major 인덱스인 *Index* 값을 대응하는 행과 열 번호 (i, j)로 변환한 후, 이 위치 코드를 계층 0의 프로세서 $PE(0,\ i,\ j)$로 이동한다.

예를 들어, (그림 6)에 주어진 위치 코드의 *Index*에 대해 대응하는 행과 열 번호를 구하면 (그림 8)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 8) 위치 코드의 *Index*를 행과 열번호로 바꾼 예

이와 같은 위치 코드의 이동은 [5]에 주어진 알고리즘에 의해 O(1) 시간에 수행될 수 있으며, 수행된 결과는 (그림 9)와 같이 이동된다.



(그림 9) 위치코드의 이동 결과

3.3 위치 코드의 분해

이 연산은 크기가 $s \times s(1 < s \leq n)$인 블록을 나타내는 위치 코드를 그 블록에 포함된 1×1 블록을 나타내는 위치 코드들로 분해하는 것이다. 예를 들어, 높이가 2인 사진 트리에서 위치 코드 (12, 1)은 2×2 크기의 블럭이므로 이 위치 코드는 (12, 2), (13, 2), (14, 2), (15, 2) 라는 4개의 위치 코드들로 분해되어야 한다.

먼저 이 연산이 수행되기 전에, 분해될 위치 코드(a, b)는 계층 0의 프로세서 $PE(0, i, j)$에 저장되어 있는 것으로 가정한다. 여기서 i와 j는 각각 shuffled row-major 인덱스인 a에 대응하는 행 번호와 열 번호에 해당한다.

이 연산은 다음과 같이 3 단계로 수행될 수 있다.

① UD 버스를 이용하여 계층 0에 있는 위치 코드를 계층 b로 이동한다. 위치 코드 (12, 1)의 예를 들어보자. 먼저 이 위치 코드의 인덱스 12는 shuffled row-major 인덱스이므로 대응하는 행 번호 2와 열 번호 2를 구할 수 있다. 따라서 이 위치 코드는 $PE(0, 2, 2)$에 존재하게 되며 이 단계에서 $PE(1, 2, 2)$로 이동한다.

② 각 계층에서 위치 코드를 받은 $PE(l, i, j)$는 N, S, W, E 버스를 이용하여 $PE(l, i+u, j+v)$, $0 \leq u < 2^{h-l}$, $0 \leq v < 2^{h-l}$의 프로세서들과 연결되는 블록을 형성한다. 여기서 h는 사진 트리의 높이이다. 이때 형성되는 프로세서 블록은 해당 위치 코드의 크기와 일치한다. 그리고 하나의 신호를 블록내의 프로세서들에게 브로드캐스트하며, 신호를 받은 프로세서들은 자신의 행 번호와 열 번호를 이용하여 새로운 위치 코드를 만든다. 즉 $PE(l, i, j)$는 위치 코드$(n \times i + j, h)$을 만들어 낸다. 예를 들어, 위치 코드 (12, 1)을 전달받은 $PE(1, 2, 2)$는 $PE(1, 2, 3)$, $PE(1, 3, 2)$, $PE(1, 3, 3)$와 함께 블록을 형성하게 되고 각각 (12, 2), (13, 2), (14, 2), (15, 2)의 위치 코드들을 생성하게 된다.

③ UD 버스를 이용하여 새로 생성된 위치 코드들을 계층 0로 이동한다. 이 과정에서 모든 위치 코드들이 계층 0의 프로세서들로 이동하게 되며, 사진 트리의 위치 코드들은 서로 중복되는 부분이 없기 때문에 이 과정에서 중복된 위치 코드들이 생성될 수 없다.

3.4 정 렬

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^2개의 데이터는 계층 0에 속하는 $PE(0, i, j)(0 \leq i, j < n)$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. 선형 이동을 위한 상수 시간 알고리즘

선형 이동 연산에서 이진영상을 나타내는 $k(0 < k \leq n^2)$개의 위치코드는 초기에 계층 0에 속하는 k개의 프로세서에 row-major 순서로 하나씩 저장되어 있으며, 선형 이동을 위한 거리 x'과 y'의 값은 위치 코드와 함께 저장되어 있다고 가정한다. 선형 이동 연산을 수행하는 RMESH 알고리즘은 알고리즘 1과 같이 12 단계로 구성된다. 여기서 단계 8에서 11까지는 [17]의 알고리즘을 선형 이동을 위해 3-차원 RMESH에 적합하도록 수정하였다.

[단계 1] 위치 코드와 이동 거리 $\langle Index, Level, x', y' \rangle$를 갖는 계층 0의 프로세서는 $Index$ 값의 대응는 행과 열 번호 (i, j)를 계산한다.

[단계 2] 위치 코드를 계층 0의 프로세서 $PE(0, i, j)$로 이동한다.

[단계 3] 위치 코드를 분해하여, 새로운 위치 코드들을 생성한다. 새로운 위치 코드의 행과 열 번호(i', j')에 선형 이동 거리 x', y'을 더한 결과 $(i'+x', j'+y')$의 shuffled row major index를 계산하여 위치 코드의 $Index$에 저장한다. 이때 $(i+x', j'+y')$가 $n \times n$ 영상의 범위를 벗어나면, 위치코드의 $Index$ 값에 ∞을 저장한다.

[단계 4] 위치 코드의 $Index$ 값을 기준으로 정렬하며, $Index$ 값이 ∞인 위치 코드는 무시된다.

[단계 5] 계층 0의 행 i에 있는 위치코드들을 계층 i의 행 0으로 이동시킨 후, 홀수 번째 계층에 있는 위치코드를 역순으로 permute한다. 그리고 $n \times n \times n$ RMESH를 $n \times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다.

[단계 6] 각 프로세서는 위치코드의 *Index* 값을 이용하여 그 인덱스가 대표할 수 있는 최대 블록의 크기를 결정하여 *NumPix* 레지스터에 저장한다.
[단계 7] 연속적인 인덱스를 갖는 프로세서들을 segment로 분리한다.
[단계 8] 각 프로세서는 자신이 속한 segment 내의 첫 프로세서가 가진 인덱스에서부터 자신의 인덱스 사이의 거리를 계산하여 *Position* 레지스터에 저장한다. 그리고 자신이 속한segment의 길이를 계산하여 *Length* 레지스터에 저장한다.
[단계 9] 각 프로세서는 *Length*와 *Position*의 차를 계산하여 *Follow* 레지스터에 저장한다. 그리고 (*largest power of* $4 \le$ *Follow*)의 조건을 만족하는 largest power의 값을 계산하여 *Follow*1 레지스터에 저장한다.
[단계 10] 각 프로세서는 min(*NumPix*, *Follow*1)의 값을 구하여 *MaxBlk* 레지스터에 저장한다. 그리고 *MaxBlk* 값을 이용하여 자신의 위치코드의 새로운 레벨을 계산한 후 *Level*1 레지스터에 저장한다.
[단계 11] 각 프로세서는 새로운 위치코드 (*Index*, *Level*1)을 이용하여, 사진트리에서 위치코드에 해당하는 노드보다 낮은 인덱스를 갖는 형제노드의 수를 계산하여 *Lsib* 레지스터에 저장하고, 높은 인덱스를 갖는 형제노드의 개수를 계산하여 *Rsib* 레지스터에 저장한다. 그리고 *Position*과 *Lsib*, *Follow*와 *Rsib*를 비교하여 위치코드가 불필요한가를 검사한다. 만약 불필요하다면 *Redun* 레지스터에 0을, 그렇지 않다면 1을 저장한다.
[단계 12] 필요한 위치코드들을 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.

(알고리즘 1) 선형 이동 알고리즘

선형 이동을 위한 RMESH 알고리즘이 수행되는 과정을 단계별로 살펴보자. 단계별 수행의 예를 들기 위해 (그림 2)에 주어진 8×8 이진영상이 (그림 3)과 같이 선형 이동하는 과정을 사용한다.

[단계 1]에서 위치 코드의 *Index* 값은 shuffled row-major 인덱스이므로 대응하는 행과 열 번호 (i, j)를 단순 계산에 의해 구할 수 있음을 앞 절에서 보았다. 예를 들어, 위치코드 (6, 3)의 경우, *Index* 값이 6이므로, 6의 이진수 표현은 000110이다(전체가 8×8 영상이므로 3비트씩 모두 6비트로 표현된다). 여기서 행 번호는 000110의 밑줄 친 비트들에 해당하므로 001이고 열 번호는 000110의 밑줄 친 비트들에 해당하므로 010이 되어, 행과 열 번호가 (1, 2) 됨을 계산할 수 있다. 따라서 이 단계는 $O(1)$ 시간에 수행 가능하다.

[단계 2]에서 위치 코드는 단계 1에서 계산된 i, j에 따라 계층 0의 프로세서 $PE(0, i, j)$로 이동한다. 이 과정은 3.2절의 기본적인 연산에 해당하며 $O(1)$ 시간에 수행된다.

[단계 3]에서는 위치 코드를 분해하여 새로운 위치 코드들을 생성한다. 이때 선형 이동 거리는 새로운 위치 코드와 함께 저장된다. 이 과정은 3.3절의 기본적인 연산에 의해 $O(1)$시간에 수행 가능하다. 새로운 위치 코드를 갖는 계층 0의 프로세서는 해당 위치 코드의 행과 열 번호 (i', j')에 선형 이동 거리 x, y'을 더한 결과 $(i'+x', j'+y')$의 shuffled row-major index를 계산하여 위치 코드의 *Index*에 저장한다. $i'+x'<0$, $i'+x' \ge n$, $j'+y'<0$, 또는 $j'+y' \ge n$ 이때 $(i+x', j+y')$가 $n \times n$ 영상의 범위를 벗어나면, 즉, 이면, 위치코드의 *Index* 값에 ∞을 저장한다.

[단계 4]는 계층 0의 프로세서에 새로운 위치 코드들을 *Index* 값에 따라 정렬하여 위치코드를 row-major 순서로 프로세서에 할당한다. 이 단계는 정렬로서 3.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다. 정렬 후, *Index* 값이 ∞인 위치 코드는 제거된다. 예를 들어, (그림 2)(b)에 있는 위치 코드들이 이 단계를 거치면 (그림 10)과 같게 된다.

[단계 5]는 3.1절에서 언급한 3-차원 RMESH의 재구성에 해당하며 $O(1)$ 시간이 걸린다.

[단계 6]에서는 위치코드의 *Index* 값을 이용하여 인덱스가 대표할 수 있는 최대 블록의 크기를 구하여 *NumPix* 레지스터에 저장하는 것으로, *Index* i가 대표할 수 있는 최대 크기는 $4^{tzp(i)}$이다. 이때 $tzp(i)$는 i를 이진수로 표현했을 때 trailing zero-pair들의 수이다. 예를 들면, 8×8 이진영상의 경우, *Index* 0의 이진 표현이 000000 이므로 $tzp(0)=3$, $NumPix = 4^3 = 64$이고, *Index* 4는 이진 표현이 000100이므로 $tzp(4)=1$, $NumPix = 4^1 = 4$이다. 이 단계에서는 위치코드를 가진 프로세서들만이 다음과 같은 과정을 수행하여 $tzp(i)$를 $O(1)$ 시간에 구한 후 *NumPix*를 계산하므로 소요시간은 $O(1)$이다. 이 값을 계산하기 위해 재구성된 RMESH에서 다음과 같이 수행한다.

① 단계 5의 재구성된 RMESH에서 행 버스를 끊는다.
② $PE(0, b)$는 자신의 위치코드를 열 버스를 이용하여 브로드캐스트한다.
③ 전달받은 프로세서는 위치코드를 이진 표현하여 자신의 행 번호에 해당하는 비트를 저장하고 검사하여 1이면 N 버스를 끊는다. 여기에서 비트의 위치는 *Index* $i = b_{m-1} \cdots b_1 b_0$, $m = 2\log_2 n$로 정의한다.
④ $PE(0, b)$는 신호(signal)를 보낸다.
⑤ 신호를 전달받은 프로세서 중 맨 마지막 프로세서, 즉 S 버스가 연결되어 있지 않은 프로세서는 자신의 행 번호 a를 $PE(0, b)$에 전달한다.
⑥ $PE(0, b)$는 전달받은 행 번호 a를 이용하여 $tzp(i) = \left\lfloor \frac{(a+1)}{2} \right\rfloor$를 구한다.

[단계 7]에서 각 프로세서는 조건 ((*Index* - 앞 프로세서의 마지막 픽셀의 인덱스) > 1)을 검사하여 참이면 *Segment* 레지스터에 1을 저장하고, 거짓이면 0을 저장한다. 각 프로

Index	6	7	8	9	10	11	12	13	14	32	33	36	38	39	42	43	44	45	46	47	50	51	56	57	58	59
Index	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3

(그림 10) 위치코드의 정렬된 결과

세서의 위치코드가 표현하는 마지막 픽셀의 인덱스는 (*Index* +블록의 크기−1)이고, 블록의 크기는 $4^{(height-Level)}$이다. 그리고 재구성된 RMESH에서 맨 앞 프로세서의 *Segment*에는 무조건 1로 저장한다. 여기서 *Segment*가 1인 프로세서는 하나의 segment의 시작을 의미한다. 그러므로 segment로 분리하기 위해 *Segment*의 값이 1인 프로세서의 W 스위치를 끊는다. 이 단계에서는 프로세서들이 앞 프로세서만을 접근하여 계산을 하기 때문에 소요 시간은 $O(1)$이다.

[단계 8]에서 먼저 각 프로세서는 자신이 속한 segment 내의 첫 프로세서가 가진 인덱스에서부터 자신의 인덱스 사이의 거리를 계산하여 *Position* 레지스터에 저장하는 것으로, 다음과 같이 수행한다.

① *Segment*의 값이 1인 프로세서는 자신의 *Index*를 브로드캐스트한다.

② 각 프로세서는 (*Index* −전달받은 *Index*)를 계산하여 *Position*에 저장한다.

그 다음은 프로세서가 자신이 속한 segment의 길이를 구하는 것으로, segment의 마지막에 위치한 프로세서가 ($Position+4^{(height-Level)}$)를 계산하여 segment 내의 다른 프로세서에 브로드캐스트하면, 각 프로세서는 전달된 값을 *Length* 레지스터에 저장한다. 이 단계에서는 같은 segment에 속하는 프로세서들 사이의 브로드캐스팅이 필요하고 전달받은 값을 이용하여 계산만을 수행하므로 소요 시간은 $O(1)$이다.

[단계 9]에서는 단계 4에서 계산된 *Length*와 *Position*의 차를 계산하여 *Follow* 레지스터에 저장한다. 그리고 (*largest power of* $4 \le Follow$)의 조건을 만족하는 값을 구하기 위해 $4^{\lfloor \log_4 Follow \rfloor}$을 계산하여 *Follow*1 레지스터에 저장한다. 이 단계는 *Follow*와 *Follow*1의 계산만을 수행하므로 $O(1)$ 시간 걸린다.

[단계 10]에서 각 프로세서는 자신의 *NumPix*의 값과 *Follow*1의 값을 비교하여 작은 값을 *MaxBlk* 레지스터에 저장하고, 새로운 레벨을 구하기 위해 ($height-\log_4 MaxBlk$)을 계산하여 *Level*1 레지스터에 저장한다. 이 단계 역시 계산만을 수행하므로 $O(1)$ 시간 걸린다.

[단계 11]은 불필요한 위치코드를 제거하기 위해 검사하는 단계로서 *Index*와 *Level*1의 값을 이용하여 *Lsib*과 *Rsib*를 구한다. 먼저 사진트리에서 위치코드 (*Index, Level*1)에 해당하는 노드의 형제 노드들 중 맨 왼쪽 노드의 인덱스 $i = \lfloor Index/d \rfloor \cdot d$, $d = 4^{(height-Level1+1)}$을 구한 후, $Lsib = Index - i$를 계산한다. 그리고 *Rsib*을 구하기 위해 $Rsib = d - Lsib$을 계산하여 저장한다. 그 다음, 조건 ($Lsib \le Position$ && $Rsib \le Follow$)을 검사하여 참이면 0, 거짓이면 1을 *Redun* 레지스터에 저장한다. 이때 0값을 가진 프로세서의 위치코드는 불필요한 것을 의미하므로 제거되어야 한다. 그리고 1을 가진 위치코드의 레벨은 새로운 레벨인 *Level*1 값으로 변경되어야 한다. 이 단계도 프로세서 내에서 계산 시간만을 필요로 하므로 소요 시간은 $O(1)$이다.

[단계 12]는 *Redun*의 값이 1인 위치코드들만을 계층 0의 프로세서에 row-major 순서로 저장하는 단계로서, 먼저 계층 i의 행 0에 있는 <*Index, Level*1, *Redun*> 값을 계층 0의 행 i로 이동시킴으로써 처음 위치의 프로세서에게 보낸다. 이 과정은 [단계 5]의 과정을 역순으로 수행할 수 있으며, 소요 시간은 $O(1)$이다. 그 다음 *Redun*의 값이 0인 프로세서는 위치코드의 *Index* 값에 ∞을 저장한 후, 위치코드를 *Index* 값에 따라 정렬하고, *Index* 값이 ∞인 위치코드를 제거한다. 이 과정의 주된 부분은 정렬로서 3.3절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다.

지금까지 알고리즘 1의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 보았으며, (그림 11)은 단계 6부터 단계 12까지의 과정을 보여준다.

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

[단계 5]	*Indes* :	6	7	8	9	10	11	12	13	14	32	33	36	38	39	42	43	44	45	46	47	50	51	56	57	58	59
	Level :	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
[단계 6]	*NumPix* :	1	1	4	1	1	1	4	1	1	16	1	4	1	1	1	1	4	1	1	1	1	1	4	1	1	1
[단계 7]	*SegMent* :	1	0	0	0	0	0	0	0	0	1	0	1	1	0	1	0	0	0	0	0	1	0	1	0	0	0
[단계 8]	*Position* :	0	1	2	3	4	5	6	7	8	0	1	0	0	1	0	1	2	3	4	5	0	1	0	1	2	2
	Length	9	9	9	9	9	9	9	9	9	2	2	1	2	2	6	6	6	6	6	6	2	2	4	4	4	4
[단계 9]	*Follow*	9	8	7	6	5	4	3	2	1	2	1	1	2	1	6	5	4	3	2	1	2	1	4	3	2	2
	*Follow*1	4	4	4	4	4	4	1	1	1	1	1	1	1	1	4	4	4	1	1	1	1	1	4	1	1	1
[단계 10]	*MaxBlk*	1	1	4	1	1	1	1	1	1	1	1	1	1	1	1	1	4	1	1	1	1	1	4	1	1	1
	*Level*1	3	3	2	3	3	3	3	3	3	3	3	3	3	3	3	3	2	3	3	3	3	3	2	3	3	3
[단계 11]	*Lsib*	2	3	8	1	2	3	0	1	2	0	1	0	2	3	2	3	12	1	2	3	2	3	8	1	2	2
	Rsib	2	1	8	3	2	1	4	3	2	4	3	4	2	1	2	1	4	3	2	1	2	1	8	3	2	2
	Redun	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	0	0	0	1	1	1	0	0	0
[단계 12]	최종 위치코드들																										
	Index	6	7	8	12	13	14	32	33	36	38	39	42	43	44	50	51	56									
	*Level*1	3	3	2	3	3	3	3	3	3	3	3	3	3	2	3	3	2									

(그림 11) 단계별 수행 과정의 예

정리 1 : $k(0 < k \leq n^2)$개의 위치코드가 선형 이동을 위한 거리와 함께 $n \times n \times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 저장되어 있을 때, 선형 이동 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

5. 결 론

본 논문에서는 3-차원 $n \times n \times n$ RMESH 구조에서 선형 사진트리로 표현된 이진 영상의 선형 이동을 수행하는 알고리즘을 제안하였다. 이 알고리즘은 3차원 RMESH의 재구성, 위치 코드의 이동, 분해, 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 선형 이동 알고리즘은 12단계로 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 적용하였다.

참 고 문 헌

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

[2] 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.

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

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

[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 상수시간 알고리즘", 정보처리논문지, 제4권 제9호, pp.2247-2258, 1997.

[7] 김경훈, 우진운, "RMESH 구조에서 선형 사진트리의 윈도우 연산을 위한 상수시간 알고리즘", 정보과학회논문지, 제29권 제3·4호, pp.134-142, 2002.

[9] 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] 김수환, "구멍이 있는 다각형에서 가시성 다각형을 구하는 상수 시간 RMESH 알고리즘", 정보과학 2000년 가을학술 발표논문집, 2000.

[11] 김홍근, 조유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘", 정보학회논문지, 제20권 제11호, 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, Vol.34, No.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.

[17] R. Shankar and S. Ranka, "Hypercube Algorithms for Operations on Quadtree," Pattern Recognition, Vol.25, No.7, pp.741-747, 1992.

김 경 훈

e-mail : khkim@kice.re.kr

1988년 숭실대학교 전산공학과(학사)

1993년 한양대학교 교육대학원 전산교육(석사)

1998년~현재 한국교육과정 평가원 부연구위원

2000년~현재 단국대학교 컴퓨터과학부 박사 과정(수료)

관심분야 : 알고리즘 및 병렬 알고리즘, 컴퓨터 교육

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 정보컴퓨터과학부 교수

관심분야 : 알고리즘 및 병렬알고리즘, 인터넷 응용