RMESH 구조에서 선형 사진트리의 영역 확장과 스케일링을 위한 상수시간 알고리즘

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. 영역 확장은 영상을 주어진 거리만큼 확장시키는 연산이고, 스케일링은 영상을 주어진 크기만큼 증폭시키는 연산으로 영상의 기하학적 연산에 속한다. 본 논문에서는 RMESH(Reconfigurable MESH) 구조에서 3-차원 $n\times n\times n$ 프로세서를 사용하여 선형 사진트리로 표현된 영상의 영역 확장과 스케일링을 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.

Constant Time Algorithms for Region Expansion and Scaling of Linear Quadtrees on RMESH

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 to quadtrees. The region expansion is an operation to expand images by a given distance and the scaling is an operation to scale images by a given scale factor. In this paper, we present algorithms to perform the region expansion and scaling of images represented by quadtrees, using three-dimensional $n\times n\times n$ processors on RMESH(Reconfigurable MESH). These algorithms have constant time complexities 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), 영역 확장(Region Expansion), 스케일링(Scaling)

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].

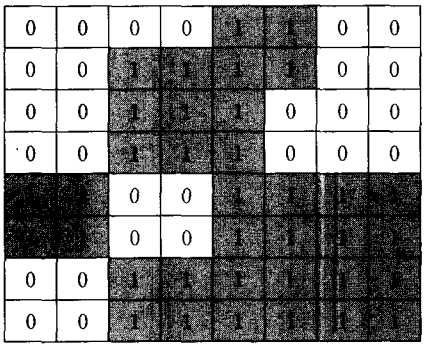
$n\times n$ 영상($n=2^k$, k는 양의 정수)에 대한 사진트리는 다음과 같이 정의된다. 사진트리의 루트(root) 노드는 전체 영상을 표현하는 것으로, 만약 영상의 모든 픽셀(pixel)들이 같은 색을 가진다면 루트 노드는 자식 노드를 갖지 않지만, 서로 다른 색을 가진다면 루트 노드는 4개의 자식 노드를 갖는다. 자식 노드는 왼쪽부터 각각 영상의 NW, NE, SW 및 SE 블록(block)의 색을 표현한다. 이와 같은 분해 과정은 노드가 표현하는 블록이 단지 하나의 공통된 색을 가지게 될 때까지 4개의 자식 노드에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

예를 들면, 8×8 이진 영상을 사진트리로 표현해 보자. (그림 1)(b)는 (그림 1)(a)를 분해한 최종 결과를 블록으로 나타낸 것이고, (그림 1)(c)는 (그림 1)(b)의 블록에 대해 사진트리로 표현한 것이다. (그림 1)(b)와 (그림 1)(c)에서 WHITE 블록은 숫자, BLACK 블록은 영문자로 구별하였다. (그림 1)(c)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.

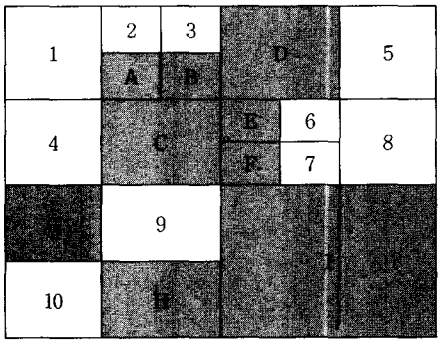
사진트리에서 레벨(level)은 루트 노드에서 임의의 노드

※ 이 연구는 2002학년도 단국대학교 대학연구비의 지원으로 연구되었음.
† 종신회원 : 단국대학교 정보컴퓨터학부 교수
논문접수 : 2003년 12월 5일, 심사완료 : 2004년 5월 17일

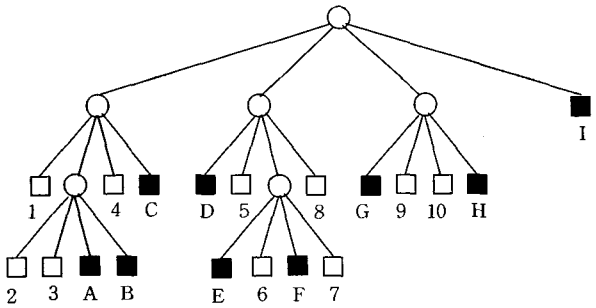
까지의 거리로 정의하며, 루트 노드의 레벨은 0으로 한다. 그리고 사진트리의 높이는 $\log_4(n\times n)$ 값으로 정의한다. 사진트리의 높이가 h일 때 레벨이 l인 노드의 블록 크기를 $2^{(h-l)}\times 2^{(h-l)}$로 계산할 수 있다. 다시 말해서 이 블록은 $4^{(h-l)}$개의 픽셀들의 모임을 표현한다. 예를 들면, (그림 1)(c)에서 노드 I는 4×4, 노드 D는 2×2, 노드 E는 1×1의 블록 크기를 갖는다.



(a) 8×8 이진영상



(b) 분해된 블록



(c) 사진트리

(그림 1) 영상과 사진트리와의 관계

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 $n\times n$ 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, 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 $r_{i-1}\,c_{i-1}\,\cdots\,r_1\,c_1\,r_0\,c_0$의 인덱스를 부여한다. 따라서 이러한 인덱스들 사이의 상호 변환이 가능하며, 인덱스의 위치를 나타내는 행과 열 번호도 쉽게 구할 수 있다.

(그림 1)(a)의 8×8 이진 영상에 shuffled row-major 인덱스를 부여하면 (그림 2)(a)와 같고, (그림 1)(c)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

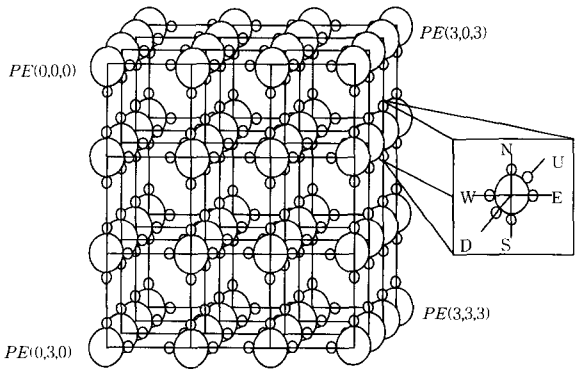
(그림 2)(b)와 같이 선형 사진트리의 표현에서 WHITE 노드에 대한 정보는 저장하지 않고 BLACK 노드에 대한 정보만을 저장하는 이유는 사진트리를 다시 구축하지 않고도 BLACK 노드에 대한 정보를 이용하여 WHITE 노드에 대한 정보를 쉽게 구할 수 있으며, 또한 BLACK 노드에 대한 정보만을 저장함으로써 저장 공간을 최소화할 수 있는 장점이 있기 때문이다.

지금까지 선형 사진트리와 관련된 영상 알고리즘들이 개

발되었는데, 영상 변환 알고리즘[3], 이진 영상의 축 변환 알고리즘[4] 등의 순차 알고리즘과 이진 영상과 선형 사진트리 사이의 상호 변환 알고리즘[5,6], 윈도우 연산을 수행하는 알고리즘[7] 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[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].

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 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.



(그림 3) 4×4×4 RMESH 구조

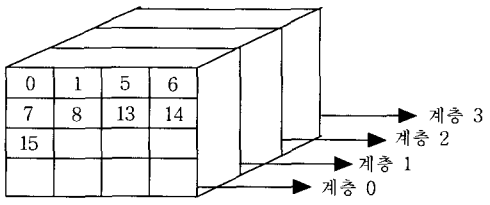
2. 3차원 RMESH의 기본 연산

여기서는 3차원 RMESH 구조에서 영역 확장과 스케일링을 효율적으로 수행하기 위해 필요한 기본적인 연산들을 알아본다.

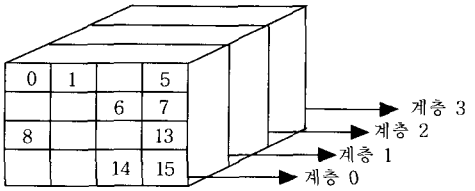
2.1 위치 코드의 이동

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.

(그림 4)는 위치 코드의 이동 예를 보여준다. (그림 4)(a)에서는 위치 코드가 계층 0에 속하는 프로세서에 하나씩 저장되어 있으며, 이 위치 코드의 *Index*에 대해 대응하는 행과 열 번호를 구하면 (그림 5)와 같게 된다.



(a) 위치 코드의 이동 전



(b) 위치 코드의 이동 후

(그림 4) 위치코드의 이동

위치 코드의 *Index* 값에 대응하는 행과 열 번호 (i,j)에 따라 이 위치 코드를 계층 0의 프로세서 $PE(0,i,j)$로 이동하는 연산은 [5]에 주어진 알고리즘에 의해 O(1) 시간에 수행될 수 있으며, 수행된 결과는 (그림 4)(b)와 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 5) 위치 코드의 Index를 행과 열 번호로 바꾼 예

2.2 위치 코드의 분해

이 연산은 크기가 $s \times s (1 < s \le n)$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 3단계로 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 $PE(l,i+u,j+v)$, $0 \le u < 2^{h-l}$, $0 \le v < 2^{h-l}$의 프로세서들과 연결되는 블록을 형성한다. 여기서 h는 사진 트리의 높이다. 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.

2.3 위치 코드의 합병

이 기본 연산은 앞 절에서 설명된 위치 코드의 분해와 반대되는 개념으로, 두개 이상의 위치 코드들이 합쳐서 하나의 큰 블록을 나타낼 수 있을 때 그 위치코드들을 하나의 위치코드가 되도록 조정하는 연산이다. 예를 들어, 높이가 2인 사진 트리에서 (12,2), (13,2), (14,2), (15,2)라는 4개의 위치 코드들은 각각 1×1 크기의 블록으로서 크기가 2×2인 하나의 위치 코드 (12,1)로 합병될 수 있다.

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

① 각 프로세서는 위치코드의 *Index* 값을 이용하여 그 인덱스가 대표할 수 있는 최대 블록의 크기를 결정한다.

② 연속적인 인덱스를 갖는 프로세서들을 세그먼트로 분리한 후, 각 프로세서는 자신이 속한 세그먼트 내의 첫 프로세서가 가진 인덱스에서부터 자신의 인덱스 사이의 거리와 자신이 속한 세그먼트의 길이를 계산한다.

③ 최대 블록의 크기와 세그먼트 내의 길이를 이용하여 합병된 새로운 위치 코드를 계산하고, 불필요한 위치 코드들을 제거한다.

2.4 정 렬

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

Jang과 Prasanna[14]는 $n \times n$ PARBUS에서 column sort 방법을 사용하여 $O(1)$ 시간에 정렬하는 알고리즘을 제안하였고, Nigam과 Sahni[15]는 column sort와 rotate sort 알고리 즘을 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 $PE(0,i,j)$ $(0 \le i,j < n)$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.

3. 영역 확장

영역 확장(region expansion)은 지리정보시스템, 영상처리, 컴퓨터 그래픽 등에서 유용하게 사용되는 연산이다. 이 연산은 주어진 이진 영상의 BLACK 영역에서부터 명시된 반경 내의 모든 픽셀들을 BLACK 픽셀로 만든다. 명시된 반경을 d라 할 때 BLACK 영역에 속하는 픽셀 $[i,j]$에 대해 확장되는 픽셀들을 정의하면 다음과 같다[1, 16].

$$E^d[i,j] = \{I[u,v] \mid 0 \le u < N,\ 0 \le v < N,\ \max\{|u-i|, |v-j|\} \le d\}$$

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

위치코드 : (12, 2)(26, 3)(48, 3)

(a) 영역 확장 전

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

위치코드 : (3, 3)(6, 3)(7, 3)(9, 3)(11, 3)
(12, 2)(18, 3)(24, 2)(33, 3)(36, 3)
(37, 3)(39, 3)(48, 2)

(b) 영역 확장 후

(그림 6) 영역 확장의 예

예를 들어, (그림 6)(a)의 영상에 대하여 $d=1$의 영역확장을 살펴보자. (그림 6)(a)에는 이진영상의 위치 코드가 주어지며, 영역 확장 후의 영상과 위치 코드를 (그림 6)(b)와 같이 얻을 수 있다.

영역 확장 연산에서 이진영상을 나타내는 $k(0<k\leq n^2)$개의 위치코드는 초기에 계층 0에 속하는 k개의 프로세서에 row-major 순서로 하나씩 저장되어 있으며, 확장 거리를 나타내는 거리 d의 값이 위치 코드와 함께 저장되어 있다고 가정한다. 영역 확장을 수행하는 RMESH 알고리즘은 (알고리즘 1)과 같이 10단계로 구성된다.

[단계 1] : 위치 코드 <*Index, Level, d*>를 갖는 계층 0의 프로세서는 *Index* 값에 대응하는 행과 열 번호 (i,j)를 계산한다.
[단계 2] : 위치 코드를 계층 0의 프로세서 $PE(0,i,j)$로 이동한다.
[단계 3] : *Level* 값에 따라 위치 코드를 분해한다. 분해 과정에서 새로 생성되는 위치 코드에 확장 거리 d를 추가한다.
[단계 4] : 위치 코드를 갖는 계층 0의 프로세서는 자신과 인접한 프로세서를 비교하여, 인접한 프로세서들이 모두 위치 코드를 가지면 *Active* 필드에 0을, 그렇지 않으면 *Active* 필드에 1을 저장한다. 이때 인접한 프로세서가 위치 코드를 가지면 그 방향의 스위치를 끊는다.
[단계 5] : *Active* 값이 1인 계층 0의 프로세서 $PE(0,i,j)$는 W, N, E, S의 순서대로 인접 프로세서에게 $\langle i,j,d\rangle$를 전송하며, 이 신호를 받은 프로세서 $PE(0,i',j')$는 W 또는 E 스위치로 신호를 받은 경우 $|i'-i|\leq d$이면 *New* 값을 1로, N 또는 S 스위치로 신호를 받은 경우 $|j'-j|\leq d$이면 *New* 값을 1로 설정한다. 그렇지 않으면 *New* 값을 0으로 둔다. 신호를 받은 프로세서는 확장 거리 d와 신호를 받은 방향을 저장하며, 만약 하나의 프로세서가 두 개 이상의 신호를 받으면 가장 큰 확장 거리를 저장한다.
[단계 6] : *New* 값이 1인 계층 0의 프로세서 $PE(0,i',j')$는 인접한 프로세서가 위치코드를 갖거나 $New=1$이면 그 프로세서와의 스위치를 끊는다.
[단계 7] : *New* 값이 1인 계층 0의 프로세서 $PE(0,i',j')$는 저장된 신호의 방향에 따라 W, N, E, S의 순서대로 인접 프로세서에게 $1\langle i',j',d\rangle$를 전송하는데, 저장된 신호가 W 또는 E 이면 N과 S 방향으로, N 또는 S이면 W와 E 방향으로만 전송한다. 이 신호를 받은 프로세서 $PE(0,i'',j'')$는 W 또는 E 스위치로 신호를 받은 경우 $|i''-i'|\leq d$이면 *New* 값을 1로, N 또는 S 스위치로 신호를 받은 경우 $|j''-j'|\leq d$이면 *New* 값을 1로 설정한다.
[단계 8] : *New* 값이 1인 프로세서들은 자신의 shuffled-row-major 인덱스를 이용하여 새로운 위치 코드를 생성한다.
[단계 9] : 기존의 위치 코드와 새로운 위치 코드들을 정렬하여 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.
[단계 10] : 합병 알고리즘을 수행한다.

(알고리즘 1) 영역 확장을 수행하는 RMESH 알고리즘

영역 확장을 위한 RMESH 알고리즘이 수행되는 과정을 단계별로 살펴보자. 단계별 수행의 예를 들기 위해 (그림 6)(a)에 주어진 8×8 이진영상이 (그림 6)(b)와 같이 영역 확장하는 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, 위치 코드 (12, 2)의 경우, *Index* 값이 12이므로, 12의 이진수 표현은 001100 이다(전체가 8×8 영상이므로 3비트씩 모두 6비트로 표현된다). 여기서 행 번호는 001100의 밑줄 친 비트들에 해당하므로 010이고 열 번호는 001100의 밑줄 친 비트들에 해당하므로 010이 되어, 행과 열 번호가 (2, 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. 예를 들어, 위치 코드 (12, 2)는 $PE(0,2,2)$로, (26,3)은 $PE(0,3,4)$로, (48, 3)은 $PE(0,4,4)$로 이동된다. 이 과정은 2.1절의 기본적인 연산에 해당하며 $O(1)$ 시간에 수행된다.

[단계 3]에서는 위치 코드를 분해하여 새로운 위치 코드들을 생성한다. 이때 확장 거리 d는 새로운 위치 코드와 함께 저장된다. 예를 들어, (12,2)는 (12,3), (13,3), (14,3), (15, 3)의 위치 코드들로 분해된다. 이 과정은 2.2절의 기본적인 연산에 의해 $O(1)$시간에 수행 가능하다.

[단계 4]에서 위치 코드를 갖는 계층 0의 프로세서는 자신과 인접한 프로세서를 비교하여, 인접한 프로세서들이 모두 위치 코드를 가지면 *Active* 필드에 0을, 그렇지 않으면 *Active* 필드에 1을 저장한다. 이때 인접한 프로세서가 위치 코드를 가지면 그 방향의 스위치를 끊는다. 이 단계는 위치 코드가 나타내는 블록의 경계에 있는 프로세서들을 구별하기 위한 것으로 각 프로세서의 스위치 조작에 의해 수행될 수 있으므로 $O(1)$ 시간이 걸린다.

[단계 5]에서 *Active* 값이 1인 계층 0의 프로세서 $PE(0,i,j)$는 W, N, E, S의 순서대로 인접 프로세서에게 $\langle i,j,d\rangle$를 전송하며, 이 신호를 받은 프로세서 $PE(0,i',j')$는 W 또는 E 스위치로 신호를 받은 경우 $|i'-i|\leq d$이면 *New* 값을 1로, N 또는 S 스위치로 신호를 받은 경우 $|j'-j|\leq d$이면 *New* 값을 1로 설정한다. 그렇지 않으면 *New* 값을 0으로 둔다. 이 단계에서는 블록의 경계에 있는 프로세서로부터 W, N, E, S 방향으로 확장 거리 d 내에 위치하는 프로세서들을 표시한다. 신호를 받은 프로세서는 확장 거리 d와 신호를 받은 방향을 저장하며, 만약 하나의 프로세서가 두 개 이상의 신호를 받으면 가장 큰 확장 거리를 저장한다. 예를 들어, (그림 6)(b)에서 인덱스가 6, 7, 9, 11, 24, 27, 36, 37, 49, 50의 위치에 있는 프로세서가 *New* 값을 1로 갖게 된다. 이 과정은 행과 열 버스의 조작만으로 가능하므로 $O(1)$ 시간이 걸린다.

[단계 6]에서는 [단계 5]에서 결정된 $New=1$을 갖는 프로세서들 중 경계 지역에 속하는 프로세서들을 구별한다. 이를 위해서 *New* 값이 1인 계층 0의 프로세서 $PE(0,i',j')$는 인접한 프로세서가 위치코드를 갖거나 $New=1$이면 그 프로세서와의 스위치를 끊는다.

[단계 7]에서는 주어진 영상의 블록 경계에 있는 프로세서로부터 대각선 방향으로 확장 거리 d 내에 위치하는 프로세서들을 표시한다. 이 과정을 위해서 단계 6에 의해 결정된 $New=1$을 갖는 프로세서를 확장 거리 d 만큼 W, N, E, S 방향으로 확장한다. 즉, *New* 값이 1인 계층 0의 프로세서 $PE(0,i',j')$는 저장된 신호의 방향에 따라 W, N, E, S의 순서대로 인접 프로세서에게 $\langle i',j',d\rangle$를 전송하는데,

저장된 신호가 W 또는 E 이면 N와 S 방향으로, N 또는 S 이면 W와 E 방향으로만 전송한다. 이 신호를 받은 프로세서 $PE(0,i'',j'')$는 W 또는 E 스위치로 신호를 받은 경우 $|i''-i'| \le d$이면 *New* 값을 1로, N 또는 S 스위치로 신호를 받은 경우 $|j''-j'| \le d$이면 *New* 값을 1로 설정한다. 예를 들어, (그림 6)(b)에서 인덱스가 3, 18, 25, 33, 39, 51의 위치에 있는 프로세서가 새로 *New* 값을 1로 설정하게 된다. 이 단계 역시 행과 열 버스의 조작만으로 가능하므로 $O(1)$ 시간이 걸린다.

[단계 8]에서 *New* 값이 1인 프로세서들은 자신의 shuffled-row-major 인덱스를 이용하여 새로운 위치 코드를 생성한다. 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. (그림 6)(b)의 예에서, (3,3) (6,3) (7,3) (9,3) (11,3) (18,3) (24,3) (25,3) (27,3) (33,3) (36,3) (37,3) (39,3) (49,3) (50,3) (51,3)의 위치 코드들이 생성된다. 이 단계는 인덱스 변환을 위한 계산만을 수행하므로 $O(1)$ 시간에 수행될 수 있다.

[단계 9]에서는 기존의 위치 코드와 새로운 위치 코드들을 정렬하여 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다. 이 과정은 정렬로서 2.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다.

[단계 10]에서 위치코드들을 합병하기 위해 2.3절의 합병 연산을 사용한다. 예를 들어, 앞 단계에서 생성된 위치코드들은 (3,3) (6,3) (7,3) (9,3) (11,3) (12,2) (18,3) (24,2) (33,3) (36,3) (37,3) (39,3) (48,2)의 위치코드로 합병된다. 합병 연산이 $O(1)$시간에 수행될 수 있으므로 이 단계 역시 상수 시간에 수행된다.

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

[정리 1] $k(0<k\le n^2)$ 개의 위치코드가 영역 확장을 위한 거리와 함께 $n\times n\times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 저장되어 있을 때, 영역 확장 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

4. 스케일링

스케일링(scaling)은 스케일 팩터 s, $s\ge 0$와 기준점 $[a,b]$가 주어질 때 $[i,j]$ 위치에 있는 픽셀을 다음과 같이 주어지는 $[i',j']$ 위치로 이동하는 연산이다[16].

$$i' = \lceil s\cdot i + a(1-s) \rceil$$
$$j' = \lceil s\cdot j + b(1-s) \rceil$$
$$0 \le i,\ j < N.$$

$i' \ge N$ 또는 $j' \ge N$인 경우에는 무시된다. 만약 두 개 이상의 픽셀이 같은 위치로 맵핑되면 하나의 픽셀만이 선택된다. 그리고 $s>1$인 경우, 영상의 경계를 이은 후에 영상의 내부를 채워야만 한다.

그러나 선형 사진트리를 사용할 경우, 스케일링은 다음과 같이 처리할 수 있다. 선형 사진트리를 나타내는 위치코드 <*Index, Level*>에서 *Index*가 나타내는 행 번호와 열 번호 $[i,j]$를 계산하여 위에서 주어진 공식에 따라 $[i',j']$을 계산한다. 그리고 *Level* 값을 이용하여 블록의 크기를 계산한다. *Level*이 나타내는 블록의 크기가 $b\times b$(여기서 $b=4^{(height-Level)}$)이므로, 새로운 블록의 크기는 $b'\times b'$(여기서 $b'=b\times s$)로 된다. 그 다음, 시작 위치 $[i',j']$를 갖고 $b'\times b'$의 크기로 스케일링된 블록에 대해 새로운 위치코드를 구한다.

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

위치코드 : (3,3)(6,3)(9,3)

(a) 스케일링 전

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

위치코드 : (15,3)(26,3)(27,3)(30,3)(31,3)
(37,3)(39,3)(45,3)(47,3)(48,2)
(52,2)(56,2)

(b) 스케일링 후

(그림 7) 스케일링의 예

예를 들어, (그림 7)(a)의 영상에 대하여 [0,0]을 기준점으로 $s=3$으로 스케일링 해보자. (그림 7)(a)를 나타내는 위치코드는 (3,3) (6,3) (9,3)으로서 각각의 위치코드에 대해 스케일링 공식을 적용한다. (3,3)의 행 번호와 열 번호를 구하면 [1, 1]이 되고, 주어진 공식에 따라 $[i',j']$를 구하면 [3, 3]이 된다. 그리고 블록의 크기를 나타내는 b는 $4^{(3-3)}=1$ 이므로, 스케일된 블록의 크기를 나타내는 b'은 $b\times s=1\times 3=3$이 된다. 따라서 이것은 행 번호, 열 번호 [3, 3]에서 시작하는 크기 3×3인 블록에 해당한다. 비슷한 방법으로 위치코드 (6,3)은 [6,3]에서 시작하는 크기 3×3인 블록에 해

당하고, 위치코드 (9, 2)는 [3, 6]에서 시작하는 크기 3×3인 블록에 해당한다. 이때 새로 생성된 블록들을 위치코드로 변환하면 (그림 7)(b)와 같이 된다. 새로 생성된 블록 중 $n\times n$ 영상 범위를 벗어난 부분은 제외된다.

스케일링 연산에서 이진영상을 나타내는 $k(0<k\leq n^2)$개의 위치코드는 초기에 계층 0에 속하는 k개의 프로세서에 row-major 순서로 하나씩 저장되어 있으며, 스케일 팩터를 나타내는 s와 기준점 $[a, b]$가 위치 코드와 함께 저장되어 있다고 가정한다. 스케일링 연산을 수행하는 RMESH 알고리즘은 (알고리즘 2)와 같이 7단계로 구성된다.

[단계 1] : 위치 코드 <*Index, Level, s, a, b*>를 갖는 계층 0의 프로세서는 *Index* 값에 대응하는 행과 열 번호 $[i, j]$를 계산한다.
[단계 2] : 스케일 팩터 s와 기준점 $[a, b]$를 사용하여 새로운 행과 열번호 $[i', j']$를 계산하며, *Level*을 이용하여 블록의 크기를 나타내는 $b=4^{(height-Level)}$를 계산하여, 새로운 블록의 크기를 나타내는 $b'=b\times s$를 계산한다.
[단계 3] : 시작 위치 $[i', j']$와 크기를 나타내는 b'값을 계층 0의 프로세서 $PE(0, i', j')$로 이동한다.
[단계 4] : 계층 0에서 $PE(0, i', j')$는 E, S 버스를 이용하여 $PE(0, i'+u, j'+v)$, $0\leq u<b'$, $0\leq v<b'$의 프로세서들과 연결되는 블록을 형성한다. 이때 $(i'+u, j'+v)$가 $n\times n$ 영상의 범위를 벗어나면 무시된다.
[단계 5] : 블록에 포함되는 계층 0의 프로세서들은 자신의 shuffled-row-major 인덱스를 이용하여 새로운 위치 코드를 생성한다.
[단계 6] : 새로운 위치 코드들을 정렬하여 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다.
[단계 7] : 합병 알고리즘을 수행한다.

(알고리즘 2) 스케일링을 수행하는 RMESH 알고리즘

스케일링 연산을 위한 RMESH 알고리즘이 수행되는 과정을 단계별로 살펴보자. 단계별 수행의 예를 들기 위해 (그림 7)(a)에 주어진 8×8 이진영상이 (그림 7)(b)와 같이 스케일링하는 과정을 사용한다.

[단계 1]에서 위치 코드 <*Index, Level, s, a, b*>를 갖는 계층 0의 프로세서는 *Index* 값에 대응하는 행과 열 번호 $[i, j]$를 계산한다. 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, (그림 7)(a)에서 위치 코드 (3, 3)의 행과 열 번호는 [1, 1], 위치 코드 (6, 3)의 행과 열 번호는 [1, 2], 위치 코드 (9, 3)의 행과 열 번호는 [2, 1]이 된다. 따라서 이 단계는 $O(1)$ 시간에 수행 가능하다.

[단계 2]에서는 스케일 팩터 s와 기준점 $[a, b]$를 사용하여 새로운 행과 열번호 $[i', j']$를 계산하며, *Level*을 이용하여 블록의 크기를 나타내는 $b=4^{(height-Level)}$를 계산하여, 새로운 블록의 크기를 나타내는 $b'=b\times s$를 계산한다. 이 과정은 기존의 위치 코드가 나타내는 블록을 주어진 스케일 팩터 s와 기준점 $[a, b]$에 따라 새로운 블록으로 계산하는 과정이다. 예를 들어, 단계 1의 예에서 얻어진 행과 열 번호 [1, 1]은 새로운 행과 열 번호 [3, 3], [1, 2]는 새로운 행과 열 번호 [3, 6], [2, 1]은 새로운 행과 열 번호 [6, 3]으로 계산되며, 각 블록의 크기는 3×3이 된다. 따라서 이 단계는 단순 계산만 필요하므로 $O(1)$ 시간에 수행 가능하다.

[단계 3]에서는 시작 위치 $[i', j']$와 크기를 나타내는 b' 값을 계층 0의 프로세서 $PE(0, i', j')$로 이동한다. 예를 들어, 단계 2에서 계산된 행과 열 번호 [3, 3]의 블록 정보는 $PE(0,3,3)$로, [3, 6]은 $PE(0,3,6)$으로, [6, 3]은 $PE(0,6,3)$로 이동된다. 이 과정은 2.1절의 기본적인 연산에 해당하며 $O(1)$ 시간에 수행된다.

[단계 4]에서는 단계 3에서 블록의 정보를 전달받은 계층 0의 프로세서 $PE(0, i', j')$가 E, S 버스를 이용하여 $PE(0, i'+u, j'+v)$, $0\leq u<b'$, $0\leq v<b'$의 프로세서들과 연결되는 블록을 형성한다. 이 과정은 다음과 같은 단계로 수행될 수 있다. 프로세서 $PE(0, i', j')$가 가진 블록 정보는 이 블록이 (i', j')의 위치에서 시작한다는 것을 의미하므로, $PE(0, i', j')$는 W, N 방향의 스위치를 끊는다. 그리고 E 방향으로 (i', j', b')을 브로드캐스트하여 이 신호를 받은 프로세서 $PE(0, i'', j'')$는 $|i''-i'|\leq b'$이면 *Active* 값을 1로 설정한다. 그 다음 S 방향으로 (i'', j'', b')을 브로드캐스트하여 $PE(0, i'', j'')$가 $|j''-j''|\leq b'$이면 *Active* 값을 1로 설정한다. 그렇지 않으면 *Active* 값을 0으로 둔다. 이 과정에서 블록의 우측과 좌측 경계에 위치한 프로세서들이 *Active* = 1을 갖게 된다. 마지막으로 *Active* = 1 인 프로세서들은 인접한 프로세서가 *Active* = 1인 값을 가지면 그 프로세서와의 스위치를 끊고, E와 S 방향의 순서대로 (i'', j'', b') 신호를 브로드캐스팅한다. 이때 양 방향에서 같은 신호를 받은 프로세서는 자신의 *Active* 값을 1로 설정한다. 이 과정에서 $(i''+u, j''+v)$가 $n\times n$ 영상의 범위를 벗어나면 무시된다. 따라서 이 과정은 프로세서의 조작과 브로드캐스팅에 의해 수행될 수 있으므로 $O(1)$ 시간이 걸린다.

[단계 5]에서 *Active* = 1인 계층 0의 프로세서들은 자신의 shuffled-row-major 인덱스를 이용하여 새로운 위치 코드를 생성한다. 즉 블록에 포함되는 프로세서 $PE(0, i, j)$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. (그림 7)(b)의 예에서, 블록에 포함되는 프로세서들에 의해 (15, 3) (26, 3) (27, 3) (30, 3) (31, 3) (37, 3) (39, 3) (45, 3) (47, 3) (48, 3) (49, 3) (50, 3) (51, 3) (52, 3) (53, 3) (54, 3) (55, 3) (56, 3) (57, 3) (58, 3) (59, 3)의 위치 코드들이 생성된다. 이 단계는 인덱스 변환을 위한 계산만을 수행하므로 $O(1)$ 시간에 수행될 수 있다.

[단계 6]에서는 새로운 위치 코드들을 정렬하여 계층 0의 프로세서에 row-major 순서로 하나씩 저장한다. 이 과정은 정렬로서 2.4절에서 언급한 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다.

[단계 7]에서 위치코드들을 합병하기 위해 2.3절의 합병 연산을 사용한다. 예를 들어, 앞 단계에서 생성된 위치코드들 중 (48, 3) (49, 3) (50, 3) (51, 3)은 (48, 2)로, (52, 3) (53, 3) (54, 3) (55, 3)은 (52, 2)로, (56, 3) (57, 3) (58, 3) (59, 3)은 (56,

2)로 합병된다. 이러한 연산의 결과로 (그림 7)(b)를 얻을 수 있다. 여기서 사용되는 합병 연산이 $O(1)$ 시간에 수행될 수 있으므로 이 단계 역시 상수 시간에 수행된다.

지금까지 (알고리즘 2)의 각 단계가 모두 $O(1)$ 시간에 수행될 수 있음을 설명하였으며, [정리 2]와 같이 요약할 수 있다.

[정리 2] 스케일링에서 $k(0 < k \leq n^2)$개의 위치 코드가 스케일 팩터 및 기준점과 함께 $n \times n \times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 저장되어 있을 때, 스케일링 알고리즘은 $O(1)$ 시간 복잡도를 갖는다.

5. 결 론

본 논문에서는 $k(0 < k \leq n^2)$개의 위치 코드가 $n \times n \times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 저장되어 있을 때, 상수 시간에 영역 확장과 스케일링을 수행하는 병렬 알고리즘을 제안하였다. 이 알고리즘들은 3차원 RMESH에서 상수 시간에 수행되는 위치 코드의 이동, 분해, 합병, 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 영역 확장 알고리즘은 10단계, 스케일링 알고리즘은 7단계로 구성되어 있으며, 각 단계는 효율적인 위치 코드 라우팅을 위하여 기본 연산들을 사용한다. 따라서 각 단계를 상수 시간에 수행할 수 있으므로, 본 알고리즘은 $O(1)$ 상수 시간 복잡도를 가진다. 특히 본 알고리즘은 위치코드를 이진 영상으로 변환하지 않고 직접 위치코드에 상수 시간 연산들을 적용한다.

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", 정보과학회논문지A, 제23-A권 제5호, pp.454-466, 1996.

[6] 공헌택, 우진운, "RMESH 구조에서의 선형 사진트리 구축을 위한 상수 시간 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.

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

[11] 김홍근, 조유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘", 정보학회논문지, 제20권 제11호, pp.1693-1701, 1993.

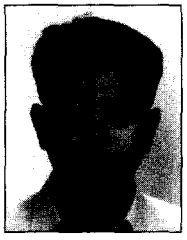
[12] J. Jang, H. Park and V. Prasanna, "A Fast Algorithm for Computing Histogram on a Reconfigurable Mesh," IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. 17, No.2, pp.97-106, 1995.

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

[15] 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.

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



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 : 알고리즘 및 병렬알고리즘, 인터넷 응용