RMESH 구조에서의 선형 사진트리 구축을 위한 상수 시간 알고리즘

공 헌 택[†] · 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 있으나, 이를 구축하기 위해서는 복잡하고 시간이 많이 걸린다. 본 논문에서는 RMESH 구조에서 3-차원 $n\times n\times n$ 프로세서를 사용하여 $n\times n$ 이진 영상을 $O(1)$ 시간에 선형 사진트리를 구축하는 알고리즘을 제안하였다. 제안한 알고리즘은 시간 복잡도 $O(1)$을 갖는 합병 알고리즘과 기존의 $O(1)$ 정렬 알고리즘을 사용함으로써 PARBUS 구조에서 제안된 알고리즘보다 간단하고 쉽게 이해할 수 있는 장점이 있다.

Constant Time Algorithm for Building the Linear Quadtree on RMESH

Heon Taek Kong[†] · Jin Woon Woo[††]

ABSTRACT

Quadtree, which is hierarchical data structure, is a very important data structure to represent binary images. Since a linear quadtree representation as is a way to store a quadtree is efficient to save space compared with other representations. It is, however, complicated and takes a large amount of time to build the linear quadtree. In this paper, we present $O(1)$ time a linear quadtree building algorithm for a $n\times n$ binary image using three-dimensional $n\times n\times n$ processors on RMESH structure. Our algorithm, by use of $O(1)$ time collapsing algorithm and reported $O(1)$ time sorting algorithm, is simpler and easier to understand than resently presented algorithm on PARBUS structure.

1. 서 론

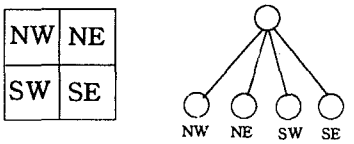
계층적 자료구조는 컴퓨터 그래픽, 영상처리, 지형처리, 패턴 인식 및 로보트 공학분야 등의 자료를 표현하는데 매우 적합한 기법이다. 특히 계층적 자료구조 중의 하나인 사진트리(quadtree)는 디지털 영상을 규칙적으로 분해(decomposition)하기 때문에 이진영상을 표현하는데 매우 유용한 자료구조이다[1, 2].

$n\times n$ 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 공

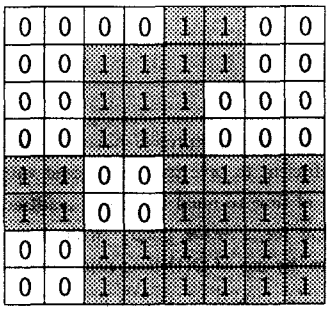
† 정 회 원:국립천안공업전문대학 전자계산과 부교수
†† 정 회 원:단국대학교 전산통계학과 부교수
논문접수:1997년 2월 11일, 심사완료:1997년 8월 12일

통된 색을 가지게 될 때까지 4 개의 자식 노드에 대해 순환적으로 적용된다.

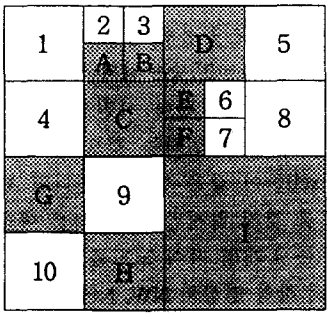
예를 들면, 8×8 이진 영상을 사진트리로 표현해 보자. 일반적으로 이진 영상에서는, (그림 1(b))와 같이 WHITE는 0으로 BLACK은 1로 표현한다. (그림 1(c))는 (그림 1(b))를 분해한 최종 결과를 블록으로 나타낸 것이고, (그림 1(d))는 (그림 1(c))의 블록에 대해 사진트리로 표현한 것이다. (그림 1(c))와 (d)에서 WHITE 블록은 숫자, BLACK 블록은 영문자로 구별하였다. (그림 1(d))에서 사각형 BLACK 노드는 블록 전체가 1로 구성되어 있음을 의미하며, 사각형 WHITE 노드는 블록 전체가 0으로 구성되어 있음을 의미한다. 그리고 원형 노드는 내부 노드로서 GRAY 노드라 한다.



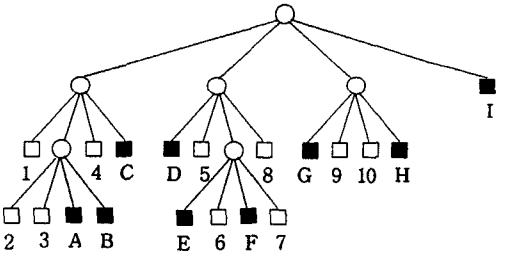
(a) 블록과 노드와의 관계



(b) 8×8 이진영상



(c) 분해된 블록



(d) 사진트리

(그림 1) 이진 영상과 사진트리와의 관계

(Fig. 1) Relationship between a binary image and its quadtree

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 가르키는 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*)를 위치 코드(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, snake-like 인덱스, 그리고 shuffled row-major 인덱스이다. Row-major 인덱스 방법은 r 행과 c 열의 픽셀에 $r\times n+c$의 인덱스를 부여되고, snake-like 인덱스 방법은 $r\times n+c'$

를 부여된다. 이때 r이 짝수이면 c'는 c이고, 홀수이면 c'는 $(n-1)-c$이다. 그리고 shuffled row-major 인덱스 방법은 r과 c의 이진 표현이 $r_{i-1}\cdots r_1r_0$과 $c_{i-1}\cdots c_1c_0$, $(i=\log_2 n)$ 일 때, 해당 픽셀에 이진수 표현으로 $r_{i-1}c_{i-1}\cdots r_1c_1r_0c_0$의 인덱스를 부여한다. (그림 2)는 4×4 이진 영상에서 3 가지 방법에 따라 인덱스를 부여하는 예를 살펴본 것이다.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

(a) row-major

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

(b) snake-like

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

(c) shuffled row-major

(그림 2) 인덱스 부여 방법
(Fig. 2) Indexing method

(그림 1(b))의 8×8 이진 영상에 shuffled row-major 인덱스를 부여하면 (그림 3(a))와 같고, (그림 1(d))의 BLACK 노드들을 위치 코드를 이용하여 선형 사진트리 표현으로 나타내면 (그림 3(b))와 같다.

0	1	4	5			20	21
2	3					22	23
8	9			25	28	29	
10	11			27	30	31	
		36	37				
		38	39				
40	41						
42	43						

(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) 선형사진트리

(그림 3) 위치 코드를 이용한 선형 사진트리 표현
(Fig. 3) The linear quadtree representation using locational code

(그림 3(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.

현재까지 다양한 병렬 컴퓨터 구조에서 $n\times n$ 이진 영상을 사진트리로 변환하거나 사진트리에서 $n\times n$ 이진 영상으로 재변환하는 병렬 알고리즘들이 제안되었다[3, 4, 5, 6]. Hung과 Rosenfeld[3]는 $n\times n$ mesh 구조에서 $O(n)$ 시간 복잡도를 갖는 알고리즘을 제안하였으며, Ibarra와 Kim[4]은 n^2개의 프로세서를 갖는 SIMD-hypercube 구조에서 $O((\log n)$ 시간 복잡도를 갖는 알고리즘을 제안하였다. 그리고 [5]는 재구성가능 메쉬 중 PARBUS 구조에서 $n\times n\times n$ 프로세서를 사용하여 $O(1)$ 시간 복잡도를 갖는 알고리즘을 제안하였다.

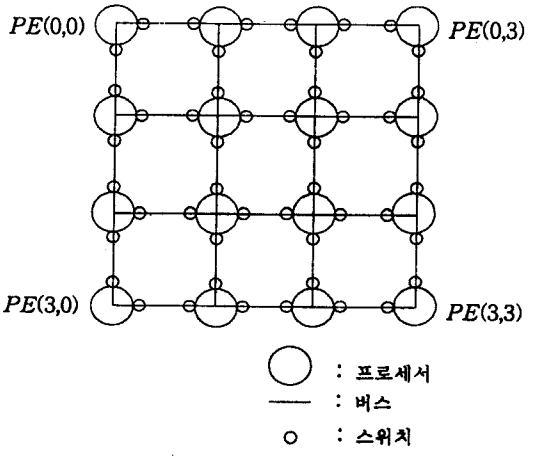
본 논문은 재구성가능 메쉬 중 RMESH 구조에서 $n\times n\times n$ 프로세서를 사용하여 $O(1)$ 시간 복잡도를 갖는 알고리즘을 제안한다. 본 논문에서 제안하는 알고리즘을 [5]와 비교할 때 상수 시간의 시간 복잡도를 갖는 점과 사용하는 프로세서들의 수는 같지만 사용하는 구조가 서로 다르다. 본 논문의 RMESH 구조는 PARBUS 구조와 서로 다른 특징을 가지며, 내부적인 데이터 이동 방법이 서로 다르다. 특히 본 알고리즘은 RMESH 구조의 상수 시간 정렬을 사용함으로써, 정렬을 전혀 사용하지 않는 [5]와 비교할 때 알고리즘 자체가 더 간결하다고 할 수 있다.

2. RMESH 구조와 정렬

RMESH는 기존의 메쉬(mesh) 구조에 동적으로 재구성 가능한 버스 시스템을 결합한 구조로서 Miller, Prasanna-Kumar, Reisis, Stout에 의하여 제안되었으며[7], 구조적인 장점 때문에 다양한 분야에서 연구되었고 효율적인 알고리즘들이 개발되었다[8, 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 4)는 4×4 RMESH 구조를 보여준다. 프로세서들을 식별하기 위해 각 프로세서에게 $PE(i, j)$를 부여한다. 이때 $0 \le i, j < n$는 행의 인덱스이고, j는 열의 인덱스이다.



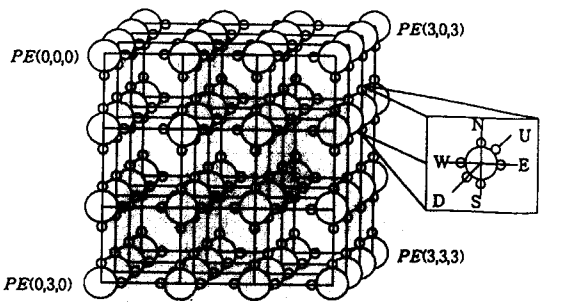
(그림 4) 4×4×4 RMESH 구조
(Fig. 4) 4×4×4 RMESH structure

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.

두 개의 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 5)는 4×4×4 RMESH를 보여준다. 버스 스위치들은 기본적으로 2-차원 RMESH와 같이 N, S, W, E 스위치가 존재하며, 추가적으로 각 프로세서마다 계층을 연결



(그림 5) 4×4×4 RMESH
(Fig. 5) 4×4×4 RMESH structure

하는 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.

2.3 정 렬

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

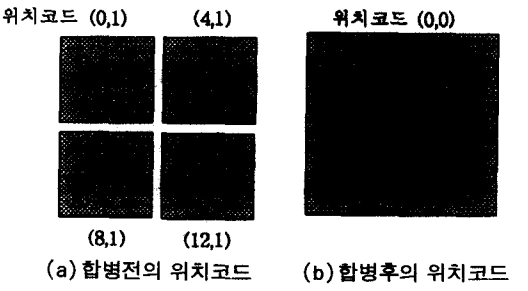
n 개의 데이터를 $O(1)$ 시간에 정렬하는 알고리즘을 개발하기 위해 초기에는 count sort 방법이 3-차원 $n\times n\times n$ RMESH[8, 9], PARBUS[14], MRN[13] 구조에 적용되었다. 그 후 사용되는 프로세서의 수를 줄이기 위한 노력이 계속되었는데, Jang과 Prasanna[15]는 $n\times n$ PARBUS에서 column sort 방법을 사용하여 $O(1)$ 시간에 정렬하는 알고리즘을 제안하였고, Nigam과 Sahni[16]는 column sort와 rotate sort 알고리즘을 각각 $n\times n$ RMESH에 적용하여 n 개의 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^2 개의 데이터를 $O(1)$ 시간에 정렬할 수 있는 알고리즘을 제안하였다. 이 알고리즘에서 초기의 n^2 개의 데이터는 계층 0에 속하는 $PE(0, i, j)$ $(0\le i, j<n)$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. 합 병

선형 사진트리의 표현을 사용하여 사진트리와 관련된 연산을 수행할 때 크게 2 가지 방법이 가능하다. 첫째 방법은 선형 사진트리를 원래의 이진 영상으로 변환한 후 연산을 수행하고 그 결과를 다시 선형 사진트리 표현으로 변환하는 것이다. 그러나 이 방법은 두번의 변환 과정을 거쳐야 하므로 시간을 많이 필요로 한다. 이러한 단점을 해결하기 위한 둘째 방법은 선형 사진트리의 위치 코드를 그대로 연산에 적용한 후 그 결과에 해당하는 위치 코드를 조정하여 정확한 선형 사진트리의 표현으로 만드는 것이다. 이와 같이 위치 코드를 조정하여 정확한 선형 사진트리 표현으로 만드는 것을 **합병(collapsing)**이라 한다.

합병이 요구되는 이유는 다음과 같다. (그림 6(a))와 같이 4 개의 블록에 대해 4 개의 위치 코드가 존재한다고 가정하자. 그러나 4 개의 블록이 모두 같은 색이므로 하나의 큰 블록으로 합병되어야 한다. 즉 (그림 6(b))와 같이 4 개의 위치 코드는 하나의 위치 코드로 만들어져야 한다.



(그림 6) 합병의 예
(Fig. 6) An example of collapsing

Shankar와 Ranka[17]는 하이퍼큐브 구조에서 이러한 합병을 사용하여 효율적인 이진 영상 알고리즘들을 제안하였다. 여기서는 [17]의 알고리즘을 3-차원 RMESH에 적합하도록 수정하여 $O(1)$ 시간에 합병할 수 있음을 보인다.

만약 k $(0<k\le n^2)$ 개의 위치 코드가 존재한다면, 초기에 위치 코드(*Index*, *Level*)들은 계층 0에 속하는 k 개의 프로세서에 row-major 순서로 하나씩 저장되어 있다고 가정한다. 합병 알고리즘은 알고리즘 1과 같이 8 단계로 구성된다.

[단계 1] 계층 0의 행 i에 있는 위치 코드들을 계층 i의 행 0으로 이동시킨후, 홀수 번째 계층에 있는 위치 코드를 역순으로 permute한다. 그리고 $n\times n\times n$ RMESH를 $n\times n^2$ RMESH의 개념을 가진 구조가 되도록 재구성한다.

[단계 2] 각 프로세서는 위치 코드의 *Index* 값을 이용하여 그 인덱스가 대표할 수 있는 최대 블록의 크기를 결정하여 *NumPix* 레지스터에 저

장한다.

[단계 3] 연속적인 인덱스를 갖는 프로세서들을 segment로 분리한다.

[단계 4] 각 프로세서는 자신이 속한 segment 내의 첫 프로세서가 가진 인덱스에서부터 자신의 인덱스 사이의 거리를 계산하여 *Position* 레지스터에 저장한다. 그리고 자신이 속한 segment의 길이를 계산하여 *Length* 레지스터에 저장한다.

[단계 5] 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.

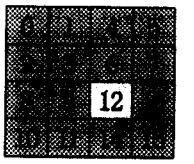
[단계 6] 각 프로세서는 min(*NumPix*, *Follow*1)의 값을 구하여 *MaxBlk* 레지스터에 저장한다. 그리고 *MaxBlk* 값을 이용하여 자신의 위치 코드의 새로운 레벨을 계산한 후 *Level*1 레지스터에 저장한다.

[단계 7] 각 프로세서는 새로운 위치 코드 (*Index*, *Level*1)을 이용하여, 사진트리에서 위치 코드에 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 위치 코드가 불필요한가를 검사한다. 만약 불필요하다면 *Redun* 레지스터에 0을, 그렇지 않다면 1을 저장한다.

[단계 8] 계층 i의 행 0에 있는 〈위치 코드, *Redun*〉들을 계층 0의 행 i로 이동시킨다.

(알고리즘 1) 합병 알고리즘
(Algorithm 1) Collapsing algorithm

예를 들어, (그림 7(a))의 이진 영상에 대하여 (그림 7(b))와 같이 위치 코드가 주어졌을 때, 위치 코드 (4, 2), (5,2), (6,2), (7,2)는 모두 같은 색을 가지므로 하나의 위치 코드 (4,1)로 합병되어야 한다. 따라서 (그림 7(b))의 위치 코드들은 합병 알고리즘을 통하여 (그림 7(c))와 같이 조정되어야 한다.



(a) 이진영상의 예

Index : 0 4 5 6 7 8 13 14 15
Level : 1 2 2 2 2 1 2 2 2

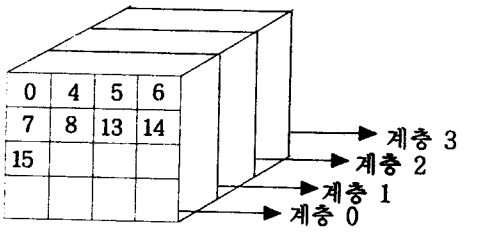
(b) 합병 전의 위치코드

Index : 0 4 8 13 14 15
*Level*1 : 1 1 1 2 2 2

(c) 합병 후의 위치코드

(그림 7) 합병 알고리즘의 적용 예
(Fig. 7) An example of appling the collapsing algorithm

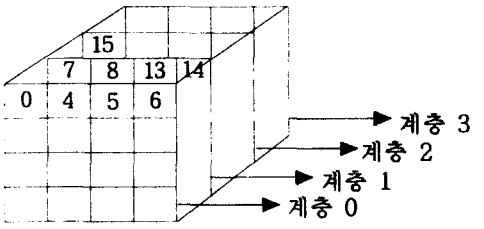
합병 알고리즘이 수행되는 과정을 단계별로 살펴보자. $n \times n$ 이진 영상에 대하여 3-차원 $n \times n \times n$ RMESH 구조를 사용한다. (그림 7(b))의 위치 코드는 (그림 8)과 같이 4×4×4 RMESH의 계층 0에 속하는 프로세서에 row-major 순서로 하나씩 저장된다. (그림 8)에서는 편의상 *Index* 값만을 보여준다.



(그림 8) 4×4×4 RMESH상의 계층 0의 초기 상태
(Fig. 8) The initial status of layer 0 on 4×4×4 RMESH

[단계 1]에서는 세가지 작업이 차례로 일어난다. 첫째, UD 버스를 이용하여 행 i에 있는 위치 코드들을 계층 i로 이동시킨 후, 각 계층에서 열 버스를 이용하여 행 0으로 이동시킨다. 이 경우 데이터 이동에 걸리는 시간은 $O(1)$이며, (그림 8)은 단계 1 후에 (그림 9)와 같게 된다.

둘째, 홀수 계층에 있는 위치 코드를 역순으로 permute시키는데, 그 과정은 다음과 같이 $O(1)$ 시간에



(그림 9) 계층 0의 행 i의 위치 코드를 계층 i의 행 0으로 이동

(Fig. 9) Moving the locational code in row i of layer 0 to row 0 of the i^{th} layer

수행된다.

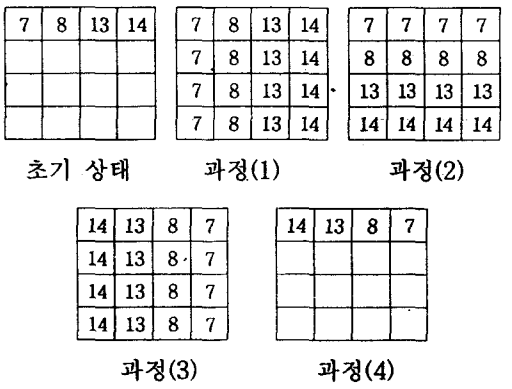
(1) 홀수 계층의 행 0에 있는 위치 코드를 열 버스를 이용하여 브로드캐스트한다.

(2) $PE(l, i, i)$, l=홀수, $0 \le i < n$에 있는 위치 코드를 행 버스를 이용하여 브로드캐스트한다.

(3) $PE(l, i, j)$, l=홀수, $i+j=n-1$, $0 \le i, j < n$에 있는 위치 코드를 열 버스를 이용하여 브로드캐스트한다.

(4) 행 0을 제외한 나머지 프로세서에 있는 위치 코드를 제거한다.

예를 들어, (그림 9)의 계층 1에서는 (그림 10)과 같은 과정을 통하여 permute된다.



(그림 10) RMESH상에서 permute하는 과정

(Fig. 10) The process of permuting on RMESH

셋째, $n \times n \times n$ RMESH를 $n \times n^2$ RMESH의 개념으로 재구성하는 것으로, 그 과정은 다음과 같이 관련된 프로세서들 사이의 스위치만을 조작하므로 $O(1)$ 시간에 수행된다.

(1) $PE(l, i, 0)$와 $PE(l, i, n-1)$, $0 \le l, i < n$의 UD 버스만을 연결한다.

(2) $PE(l, i, n-1)$, l=짝수의 D 버스를 끊고, $PE(l, i, n-1)$, l=홀수의 U 버스를 끊는다.

(3) $PE(l, i, 0)$, l=짝수의 D 버스를 끊고, $PE(l, i, 0)$, l=홀수의 U 버스를 끊는다.

이 과정을 통하여 각 계층의 프로세서들은 인접한 계층의 프로세서들과 맨 좌측 혹은 맨 우측에서 연결된 형태로 재구성된다. 이러한 구성은 n^2 개의 프로세서들이 n 번 연속적으로 연결된 형태이다.

이상과 같이 [단계 1]은 $O(1)$ 시간에 수행되며, (그림 8)의 초기 상태가 단계 1의 수행 후, 프로세서의 연결에 따라 계층들을 펼쳐 놓으면 (그림 11)과 같게 된다.

(그림 11) $n \times n^2$ RMESH

(Fig. 11) $n \times n^2$ RMESH

[단계 2]에서는 위치 코드의 *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, 4×4 영상의 경우, *Index* 0의 이진 표현이 0000이므로 $tzp(0)=2$, $NumPix=4^2=16$이고, *Index* 4는 이진 표현이 0100이므로 $tzp(4)=1$, $NumPix=4^1=4$이다. 이 단계에서는 위치코드를 가진 프로세서들만이 다음과 같은 과정을 수행하여 tzp(i)를 $O(1)$ 시간에 구한 후 NumPix를 계산하므로 소요 시간은 $O(1)$이다.

(1) (그림 11)에서 행 버스를 끊는다.

(2) $PE(0, b)$는 자신의 위치코드를 열 버스를 이용하여 브로드캐스트한다.

(3) 전달받은 프로세서는 위치코드를 이진수로 표현하여 자신의 행 번호에 해당하는 비트를 저장하고 검사하여 1이면 N 버스를 끊는다. 여기에

서 비트의 위치는 *Index* $i = b_{m-1} \cdots b_1 b_0$, ($m = 2\lfloor \log_2 n \rfloor$) 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 *a*를 *PE*(0, *b*)에 전달한다.

(6) *PE*(0, *b*)는 전달받은 행 번호 *a*를 이용하여 $tzp(i) = \lfloor \frac{(a+1)}{2} \rfloor$를 구한다.

[단계 3]에서 각 프로세서는 조건 ((*Index*-앞 프로세서의 마지막 픽셀의 인덱스)>1)을 검사하여 참이면 *Segment* 레지스터에 1을 저장하고, 거짓이면 0을 저장한다. 각 프로세서의 마지막 픽셀의 인덱스는 (*Index* +블록의 크기−1)이고, 블록의 크기는 $4^{(height-Level)}$이다. 그리고 (그림 11)에서 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 4]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 5]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 6]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 7]은 불필요한 위치 코드를 제거하기 위해 검사하는 단계로서 *Index*와 *Level*1의 값을 이용하여 *Lsib*과 *Rsib*를 구한다. 먼저 사진트리에서 위치 코드 (*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 Rsib을 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 0 값을 가진 프로세서의 위치 코드는 불필요한 것을 의미하므로 제거되어야 한다. 그리고 1을 가진 위치 코드의 레벨은 새로운 레벨인 *Level*1 값으로 변경되어야 한다. 이 단계도 프로세서 내에서 계산 시간만을 필요로 하므로 소요 시간은 *O*(1)이다.

[단계 8]은 계층 *i*의 행 0에 있는 〈*Index*, *Level*1, *Redun*〉 값을 계층 0의 행 *i*로 이동시킴으로써 처음

[단계 1]	*Index*	:	0	4	5	6	7	8	13	14	15
	Level	:	1	2	2	2	2	1	2	2	2
[단계 2]	*NumPix*	:	16	4	1	1	1	4	1	1	1
[단계 3]	*Segment*	:	1	0	0	0	0	0	1	0	0
[단계 4]	*Position*	:	0	4	5	6	7	8	0	1	2
	Length	:	12	12	12	12	12	12	3	3	3
[단계 5]	*Follow*	:	12	8	7	6	5	4	3	2	1
	*Follow*1	:	4	4	4	4	4	4	1	1	1
[단계 6]	*MaxBlk*	:	4	4	1	1	1	4	1	1	1
	*Level*1	:	1	1	2	2	2	1	2	2	2
[단계 7]	*Lsib*	:	0	4	1	2	3	8	1	2	3
	Rsib	:	16	12	3	2	1	8	3	2	1
	Redun	:	1	1	0	0	0	1	1	1	1
[단계 8]	합병된 위치코드										
	Index	:	0	4	8	13	14	15			
	*Level*1	:	1	1	1	2	2	2			

(그림 12) 합병 알고리즘의 단계별 수행 과정

(Fig. 12) The process of each step in the collapsing algorithm

위치의 프로세서에게 보내는 과정이다. 이 단계는 [단계 1]의 역순으로 수행할 수 있으며, 소요 시간은 $O(1)$이다.

(그림 12)는 (그림 7(b))의 위치 코드들이 합병되는 과정을 단계별로 보여준다.

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 위치 코드가 $n\times n\times n$ RMESH의 계층 0에 row-major 순서로 각 프로세서에 하나씩 저장되어 있을 때, 합병 알고리즘은 $O(1)$ 시간에 수행된다.

4. 선형 사진트리의 구축

3-차원 $n\times n\times n$ RMESH 구조에서 $n\times n$ 이진 영상을 선형 사진트리로 구축하는 알고리즘을 살펴보자. 초기에 $n\times n$ 이진 영상은 계층 0의 n^2 개의 프로세서에 저장된다. 즉 픽셀 (i, j)는 $PE(0, i, j)$, $0\leq i, j<n$에 저장된다.

선형 사진트리를 구축하는 과정은 알고리즘 2와 같이 4 단계로 구성된다.

[단계 1] 계층 0의 프로세서들은 할당된 픽셀에 따라 위치 코드 (*Index*, *Level*)를 구한다.

[단계 2] 위치 코드의 *Index* 값에 따라 위치 코드를 오름차순으로 정렬한다.

[단계 3] 합병 알고리즘을 수행한다.

[단계 4] 계층 0의 프로세서들에서 불필요한 위치 코드들을 제거한다.

(알고리즘 2) 선형 사진트리 구축 알고리즘
(Algorithm 2) A linear quadtree building algorithm

알고리즘 2의 선형 사진트리 구축 알고리즘이 수행되는 과정을 단계별로 살펴보고 걸리는 시간을 알아보자. 단계별 수행 과정의 예를 들기 위해 (그림 13(a))의 8×8 이진 영상을 사용한다. 그리고 (그림 13(b))는 8×8×8 RMESH의 계층 0의 프로세서들의 shuffled row-major 순서를 나타낸다.

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

(a) 8×8 이진 영상

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

(b) shuffled row-major 인덱스

(그림 13) 이진 영상의 예와 인덱스
(Fig. 13) An example of a binary image and indexing

[단계 1]에서 계층 0의 프로세서들은 위치 코드 (*Index*, *Level*)를 구한다. 이때 할당된 픽셀의 값이 1이면 자신의 shuffled row-major 인덱스를 구하여 *Index*에 저장하고, 0이면 ∞을 *Index*에 저장한다. 그리고 *Level* 값은 $\log_4(n\times n)$로 초기화한다. (그림 13(a))에 대하여 Level 값은 모두 $\log_4(8\times 8)=3$이며, 단계 1의 결과는 (그림 14)와 같다. (그림 14)는 *Index* 값만을 보여준다.

∞	∞	4	5	∞	∞	∞	∞
2	∞	6	7	∞	∞	22	∞
∞	∞	12	13	∞	∞	28	∞
∞	∞	∞	∞	∞	∞	∞	∞
32	33	36	37	48	49	∞	∞
34	35	38	39	50	51	54	∞
40	41	44	45	∞	∞	∞	∞
42	43	46	47	∞	∞	∞	∞

(그림 14) 계층 0의 프로세서에 부여된 위치 코드의 *Index* 값
(Fig. 14) The *Index* value of locational code indexed on processors in layer 0

이 단계는 위치 코드를 만들기 위한 계산 시간만을 필요로 하므로 소요 시간은 $O(1)$이다.

[단계 2]에서는 위치 코드를 *Index* 값에 따라 정렬하여 위치 코드를 row-major 순서로 프로세서에게 할당한다. 그리고 *Index* 값이 ∞인 위치 코드를 제거한다. 이 단계의 주된 부분은 정렬로서 2.3절에서 언급된 정렬 알고리즘을 적용하면 $O(1)$ 시간에 수행된다. (그림 14)는 정렬 후 (그림 15)와 같게 된다.

2	4	5	6	7	12	13	22
28	32	33	34	35	36	37	38
39	40	41	42	43	44	45	46
47	48	49	50	51	54		

(그림 15) 정렬된 위치 코드
(Fig. 15) The sorted locational codes

[단계 3]에서는 정렬된 위치 코드에 대해 합병 알고리즘을 수행한다. 알고리즘 1을 사용하여 $O(1)$ 시간에 수행할 수 있다. (그림 15)의 위치 코드에 대해 합병 알고리즘을 단계별로 수행하면 (그림 16)과 같다.

[단계 4]에서는 *Redun*의 값이 1인 위치 코드들만을 계층 0의 프로세서에 row-major 순서로 저장한다. 이를 위해서 *Redun*의 값이 0인 프로세서는 위치 코드의 *Index*에 ∞을 저장한 후, 위치 코드를 *Index* 값에 따라 정렬한다. 그리고 *Index* 값이 ∞인 위치 코드를 제거한다. 이 단계의 주된 부분은 정렬로서 $O(1)$ 시간에 수행된다. (그림 16)에 단계 4를 적용하면 최종 결과는 (그림 17)과 같게 된다.

2	4	12	13	22	28	32	48
54	32						

(a) 계층 0의 프로세서에 저장된 위치 코드

Index : 2 4 12 13 22 28 32 48 54
Level : 3 2 3 3 3 3 1 2 3

(b) 선형 사진트리의 위치 코드

(그림 17) 구축된 선형 사진트리의 최종 결과
(Fig. 17) The final result of a linear quadtree

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

정리 2 $n\times n\times n$ RMESH상에서 $n\times n$ 이진 영상을 선형 사진트리로 구축하는 알고리즘은 $O(1)$ 시간 복

Index	:	2	4	5	6	7	12	13	22	28	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	54
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	3	3	3	3
NumPix	:	1	4	1	1	1	4	1	1	4	16	1	1	1	4	1	1	1	4	1	1	1	4	1	1	1	4	1	1	1	1
Segment	:	1	1	0	0	0	1	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Position	:	0	0	1	2	3	0	1	0	0	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	0
Length	:	1	4	4	4	4	2	2	1	1	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	1
Follow	:	1	4	3	2	1	2	1	1	1	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	1
Follow1	:	1	4	1	1	1	1	1	1	1	16	16	16	16	16	4	4	4	4	4	4	4	4	4	4	4	4	1	1	1	1
MaxBlk	:	1	4	1	1	1	1	1	1	1	16	1	1	1	4	1	1	1	4	1	1	1	4	1	1	1	4	1	1	1	1
Level1	:	3	2	3	3	3	3	3	3	3	1	3	3	3	2	3	3	3	2	3	3	3	2	3	3	3	2	3	3	3	3
Lsib	:	2	4	1	2	3	0	1	2	0	32	1	2	3	4	1	2	3	8	1	2	3	12	1	2	3	0	1	2	3	2
Rsib	:	2	12	3	2	1	4	3	2	4	32	3	2	1	12	3	2	1	8	3	2	1	4	3	2	1	16	3	2	1	2
Redun	:	1	1	0	0	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

(그림 16) 합병 알고리즘의 적용
(Fig. 16) The appling of collapsing algorithm

잡도를 갖는다.

5. 이진 영상으로 환원

3-차원 $n\times n\times n$ RMESH 구조에서 선형 사진트리로 표현된 위치 코드들을 원래의 이진 영상으로 환원하는 과정을 살펴본다. 여기서는 [5]에서 제안된 3-차원 PARBUS 구조의 알고리즘을 RMESH 구조에 적용한다.

초기에 선형 사진트리로 표현된 위치 코드들이 $n\times n\times n$ RMESH의 계층 0의 프로세서에 row-major 순서로 저장되어 있다. 이진 영상으로 환원하는 알고리즘은 알고리즘 3과 같이 3단계로 구성된다.

[단계 1] 위치 코드의 *Index*를 이용하여 row-major 순서의 행과 열 번호(i, j)를 계산한다.

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

[단계 3] 위치 코드의 *Level*을 이용하여 계층 0의 프로세서에 이진 영상을 구축한다.

(알고리즘 3) 이진 영상의 환원 알고리즘
(Algorithm 3) A binary image rebuilding algorithm

[단계 1]에서 위치 코드 (*Index*, *Level*)의 *Index* 값은 shuffled row-major 인덱스이므로 1장에서 설명된 방법으로 row-major 인덱스로 바꿀 수 있다. 즉, *Index* 값을 이진수로 나타내었을 때 홀수 번째 비트들을 선택하여 행 번호 i를 구하고, 짝수 번째 비트들을 선택하여 열 번호 j를 구한다. 예를 들면, *Index* 값이 6일 경우, 이진수 표현이 0110이므로 행 번호는 01, 즉 십진수 1 이고, 열 번호는 10, 즉 십진수 2이다. 따라서 (1,2)를 얻게 되어 단계 2에서 $PE(0, 1, 2)$로 위치 코드가 이동하게 된다.

[단계 2]에서는 단계 1에서 구해진 (i, j)에 따라 위치 코드가 UD 버스를 통하여 계층 i에 있는 프로세서에게 전달되고, 그 다음 $PE(i, i, j)$로 이동되며 최종적으로 $PE(0, i, j)$로 이동된다.

[단계 3]에서 계층 0의 프로세서가 전달받은 위치 코드는 크기가 $2^{(height-level)}\times 2^{(height-level)}$인 블록을 대표하므로 이러한 크기의 블록으로 환원되어야 한다. 위치 코드의 *Level* 값을 이용하여 [5]에서 적용하는 서브버스 형성을 통하여 이진 영상의 블록을 만들 수 있으며, 전체 위치 코드에 대해 이 과정이 완료되면 원래의 이진 영상을 얻을 수 있다.

알고리즘 3은 [5]의 방법을 큰 변형없이 RMESH 구조에 적용시킨 것으로 $O(1)$ 시간이 걸린다.

6. 결 론

본 논문에서는 3-차원 $n\times n\times n$ RMESH 구조에서 선형 사진트리 변환을 위한 상수 시간 알고리즘을 제안하였다. $n\times n$ 이진 영상을 선형 사진트리로 구축하는 과정은 합병 알고리즘과 정렬 알고리즘을 이용하였고, 선형 사진트리를 $n\times n$ 이진 영상으로 환원하는 과정은 기존의 PARBUS 구조 알고리즘을 이용하였다.

특히 선형 사진트리의 구축은 복잡하고 시간이 많이 걸리는 작업으로 알려져 있다. 이를 위하여 본 논문에서는 시간 복잡도 $O(1)$을 갖는 합병 알고리즘을 제안하였으며, 기존의 $O(1)$ 정렬 알고리즘을 함께 사용함으로써 상수 시간에 선형 사진트리를 구축할 수 있음을 보여주었다. 여기서 제안한 선형 사진트리 구축 알고리즘은 PARBUS 구조의 알고리즘보다 간단하여 쉽게 이해할 수 있는 장점을 가진다.

참 고 문 헌

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

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

[3] Y. Hung and A. Rosenfeld, "Parallel Processing of Linear Quadtrees on a Mesh-Connected Computer," Journal of Parallel and Distributed Computing, Vol. 7, pp. 1-27, 1989.

[4] O. Ibarra and M. Kim, "Quadtree Building Algorithms on an SIMD Hypercube," Journal of Parallel and Distributed Computing, Vol. 18, pp. 71-76, 1993.

[5] 김 명, 장주욱, "재구성가능 매쉬에서 $O(1)$ 시간 복잡도를 갖는 이진영상/사진트리 변환 알고리

즘," 정보과학회논문지(A), 제23권, 제5호, pp. 454-466, 1996.

[6] F. Dehne, A. Ferreira, and A. Rau-Chaplin, "Efficient Parallel Construction and Manipulation of Quadtrees," Proceedings of International Conference on Parallel Processing, Vol. Ⅲ, pp. 255-262, 1991.

[7] 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.

[8] 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.

[9] 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.

[10] 김 홍근, 조 유근, "단순다각형의 내부점 가시도를 위한 효율적인 RMESH 알고리즘," 정보과학회 논문지, 제20권 11호, pp. 1693-1701, 1993.

[11] 원 태연, 우 진운, "이진영상 레이블링을 위한 RMESH 알고리즘," 병렬처리시스템 학술발표회 논문집, 제3권 2호, pp. 5-20, 1992.

[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.

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

공 헌 택

1984년 미국 NMSU대학교 전자계산학과(학사)

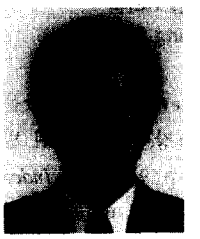
1987년 미국 Utah State University 전자계산학과(석사)

1992년~현재 단국대학교 대학원 전산통계학과(박사과정 수료)

1988년~1990년 한국국방연구원 전산체계연구부 근무

1990년~현재 국립천안공업전문대학 전자계산과 부교수

관심분야: 병렬알고리즘, 병렬처리, 데이터베이스



우 진 운

1980년 서울대학교 수학교육과(학사)

1989년 미국 University of Minesota 전산학과(박사)

1980년~1983년 대한항공 및 국토개발연구원 전산실 근무

1989년~현재 단국대학교 전산통계학과 부교수

관심분야: 알고리즘, 병렬처리 및 분산처리