LAROB상에서 이진영상과 사진트리 사이의 효율적인 변환

김 명†

요 약

본 논문에서는 LAROB상에서 이진영상을 사진트리로 변환하거나, 사진트리를 이진영상으로 변환하는 효율적인 알고리즘들을 제안한다. 이 알고리즘들은 영상의 화소 개수가 $n \times n$일 때 n^2 프로세서를 사용하며, $\Theta(1)$의 시간 복잡도를 갖는다. 실행시간과 프로세서 수의 곱이 최적 순차 알고리즘의 실행시간인 $\Theta(n^2)$과 동일하므로, 여기서 제안한 알고리즘들은 최적 알고리즘이다.

Efficient Transformations Between Binary Images and Quadtrees on a Linear Array with Reconfigurable Optical Buses

Myung Kim†

ABSTRACT

We present efficient algorithms for transforming between binary images and quadtrees on the LAROB. For a binary image of size $n \times n$, both algorithms run in $\Theta(1)$ time using n^2 processors. These algorithms are optimal in the sense that the product of time and number of processors is asymptotically the same as the optimal sequential time which is $\Theta(n^2)$.

1. Introduction

Quadtrees are a hierarchical data structure which is useful to represent binary images. Quadtrees are attractive since they allow us to be able to perform a wide range of image operations directly on them. Thus, it has been of great interest to develop efficient parallel algorithms for transforming between binary images and quadtrees, and for performing image operations on quadtrees. In this paper, among these operations, we focus on the transformations between binary images and quadtrees.

Recently, a number of efficient parallel algorithms for such transformations were developed on architectures such as the mesh-connected computer[3], the SIMD hypercube[1,4], and the reconfigurable mesh[5]. For images of size $n \times n$, the algorithms in [1,3], and [4] use n^2 processors. They run in $\Theta(n)$ time, $\Theta(\log^2 n)$ time, $\Theta(\log n)$ time, respectively. The algorithm in [5] uses n^3 processors and

† 정 회 원 : 이화여자대학교 공과대학 컴퓨터학과 교수
논문접수 : 1999년 3월 24일, 심사완료 : 1999년 5월 25일

runs in $\Theta(1)$ time.

In this paper, we present efficient algorithms for the same transformations on an architecture called linear array with reconfigurable optical buses (LAROB) [8]. Unlike electronic buses, optical channels allow multiple messages to be transmitted during a single bus cycle. In addition to this advantage, parallel systems with optical interconnections resolve some limitations of electronic buses such as limited bandwidth, capacitive loading, and cross-talk. By fully utilizing the power of the optical buses, our algorithms take $\Theta(1)$ time using n^2 processors for images of size $n \times n$. These algorithms are optimal in the sense that the product of time and number of processors is asymptotically the same as the optimal sequential time which is $\Theta(n^2)$. Note that any sequential algorithm for transforming between binary images and quadtrees takes $\Omega(n^2)$ time, since all the n^2 pixels in the image must be visited during the transformation process. In fact, Samet[12] gave a $\Theta(n^2)$ time algorithm for the transformations.

The paper is organized as follows : In section 2, we describe the linear array with reconfigurable optical buses, and present some constant time operations that will be used in the algorithms. In section 3, we define the linear quadtree representation of a binary image, and give the quadtree building algorithm. Its converse algorithm is given in section 4. Section 5 concludes this paper.

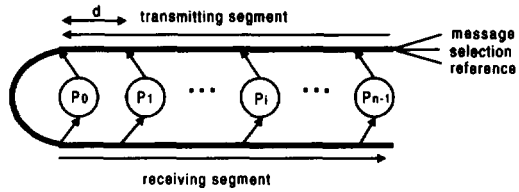
2. The LAROB model

An n processor LAROB is a 1-D reconfigurable mesh with optical pipelined buses. By setting up internal port connections properly, the LAROB can be divided into several pieces, each of which is called linear array processors with optical pipelined buses (APPB). Thus, we explain the APPB model followed by the LAROB. The detailed description of these two computing models can be found in [8].

2.1 The APPB model

Consider an n processor APPB[6,8,10]. Its processors, P_0, P_1, ... , P_{n-1}, are arranged in a row, and are connected to the optical bus. The bus is divided into two segments : upper segment and lower segment (see Figure 1). The processors write data to the upper segment, and read data from the lower segment. During a bus cycle, multiple processors can transmit their message by using different time slots of the bus. This is possible because optical buses are inherently directional, and have predictable delay per unit length.

The bus consists of three identical waveguides : the message waveguide, the selection waveguide, and the reference waveguide. The message waveguide is used to transmit messages. The selection/reference waveguides are used to transmit address related information. All transmissions are synchronized. The bus length between two adjacent processors, d, is set so that the messages travelling on the bus do not overlap with each other. The end-to-end propagation time is $\sigma_b = 2n\tau$ seconds, where τ is the time for a message to travel through the optical distance d.



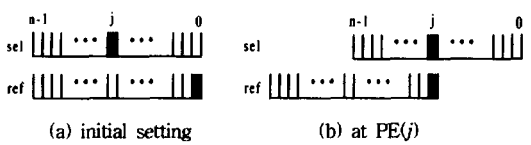
(Fig. 1) The APPB model

The APPB uses the time-division multiplexing techniques for message routing[8,10]. Time-division source-oriented multiplexing(TDSM) is used when the destination processors know the address of the source processor. In this scheme, each processor is assigned to a fixed time slot. The processors use their assigned time slots to send out their messages. The destination processors receive their messages when the time slots for the corresponding source

processors arrive through the lower segment of the bus. With this scheme, multicasting (broadcasting) is possible since all the messages, which were sent during a bus cycle, pass by every processor in the APPB.

Time-division destination-oriented multiplexing (TDDM) is used when the source processors know the address of the destination processor. Each processor is assigned to a fixed time slot. The source processors put their message to the time slot of the destination processor. All the messages are read by the destination processors simultaneously at the end of the bus cycle.

Address information can be encoded using the coincident pulse technique[8,10]. Suppose that processor P_i wants to send a message to processor P_j. P_i prepares its message and address related information before the bus cycle. P_i transmits its message when its time slot arrives during the next bus cycle. The message is put on the message waveguide. And the unary coded address information is put on the selection and reference waveguides as in Figure 2(a). The prepared information (or pulses) are transmitted synchronously on the upper segment of the bus without being changed. However, when the information passes by a processor on the lower segment of the bus, a unit delay is introduced on the message waveguide and the reference waveguide. Thus, the pulse on the selection waveguide coincide the pulse on the reference waveguide at the destination processor, P_j, as in Figure 2(b). Only a simple hardware mechanism is used for the destination processor to detect the coincidence of two pulses. By choosing multiple destinations, the source can send its message to several destinations during a single bus cycle.

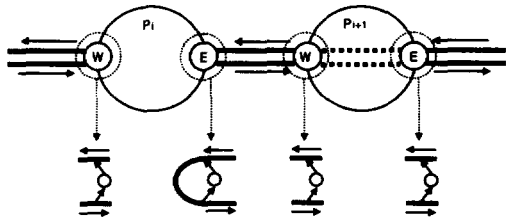


(a) initial setting (b) at PE(j)

(Fig. 2) Coincident pulse technique

2.2 The LAROB model

The array with reconfigurable optical buses (AROB) [8] is a 2-dimensional reconfigurable mesh with optical pipelined buses. The linear array with reconfigurable optical buses, LAROB, is a 1-dimensional version of it. Consider an n processor LAROB. The processors are labeled as P_0, P_1, ... , P_{n-1}. They are arranged in a row, and are connected to two unidirectional buses, as in Figure 3.



(Fig. 3) Two adjacent processors of a linear array with reconfigurable optical pipelined buses

As with the APPB, the upper bus is used as the transmitting segment and the lower bus is used as the receiving segment. As with a general reconfigurable network (RN), processors have two ports (E and W), and are allowed to reconfigure their internal port connections. Figure 3 shows the case when P_i's ports are disconnected. Note that the upper bus into the E port of P_i is connected to the lower bus going out of the same port. This is how several APPB's are realized in the LAROB. It is assumed that for an n processor LAROB, the bus can carry $\Theta(\log n)$ bits as is assumed for a reconfigurable mesh[7]. Each processor can perform arithmetic and logic operations on $\Theta(1)$ words in unit time.

2.3 Some constant time LAROB operations

Consider an LAROB which consists of n processors, P_0, P_1, $\cdots$, P_{n-1}. Assume that each processor has one datum (integer). The LAROB operations given in Observations 1-5 can be accomplished in constant number of steps, (i.e., $O(\sigma_b)$

time). Observations 1-3 can be easily obtained from the definitions of TDSM, TDDM, and the coincidental pulse technique. The proofs of Observations 4 and 5 can be found in [9,11].

Observation 1.

When an LAROB is divided into several APPBs, the processors in each APPB can compute its relative index inside the APPB in constant time. It can be done as follows : At the beginning of a bus cycle, the leader broadcasts its index to all the processors in the APPB. The other processors compute their relative indices by subtracting the index of the leader from their own indices.

Observation 2.

Let $\pi(0)$, $\pi(1)$, $\cdots$, $\pi(n-1)$ be a permutation of $0, 1, \cdots, n-1$. Suppose that for all i, $0 \le i < n$, P_i wants to send its data item to $P_{\pi(i)}$. If the addresses of the destination processors are known to the source processors (i.e., P_i knows the value of $\pi(i)$), this type of one-to-one communication can be done in one step.

Observation 3.

Let $\pi(0)$, $\pi(1)$, $\cdots$, $\pi(n-1)$ be a permutation of $0, 1, \cdots, n-1$. Suppose that for all i, $0 \le i < n$, P_i wants to receive a data item from $P_{\pi(i)}$. If the addresses of the source processors are known to the destination processors (i.e., P_i knows the value of $\pi(i)$), this type of one-to-one communication can be done in one step.

Observation 4.

The prefix sums of n integers V_i with $0 \le V_i \le n$, $0 \le i \le n-1$, and $\sum_{i=0}^{n-1} V_i \le n$, can be computed in constant number of steps.

Observation 5.

The sum of n integers, V_i, such that $V_i < n$ and sum of all V_i's is also less than n can be computed in constant number of steps.

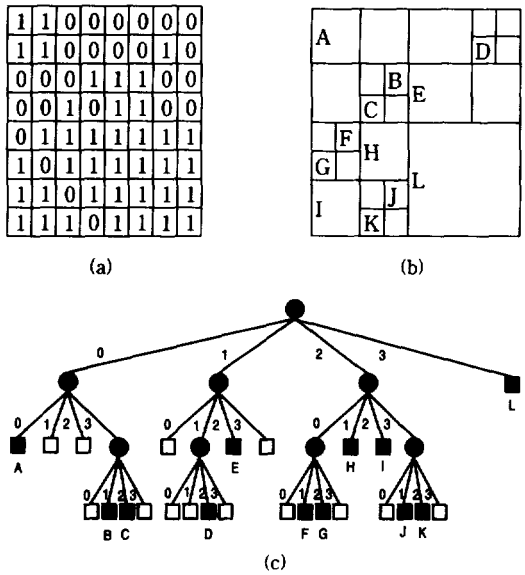
Let us define some terms that will be used throughout the paper. P_i, $0 \le i < n$, represents the i-th processor of the LAROB. When an LAROB is divided into several APPBs, the j-th processor of the i-th APPB is called as PE(i, j). We use these two notations interchangeably to give a clear presentation of the ideas. Pixel (r, c), $0 \le r, c < n$, is the pixel in row r and column c. $SRM(r, c)$ is the SRM index of pixel (r, c). For an $n \times n$ image, $SRM(r, c)$ is defined as follows : $SRM(r, c) = r_{d-1}c_{d-1}r_{d-2}c_{d-2}\cdots r_1c_1r_0c_0$, where $d = \log n$, $r = r_{d-1}r_{d-2}\cdots r_1r_0$, and $c = c_{d-1}c_{d-2}\cdots c_1c_0$. $SRM(r, c)$ is also represented as $SRM(i)$, where $i = r \times n + c$.

3. From a binary image to a quadtree

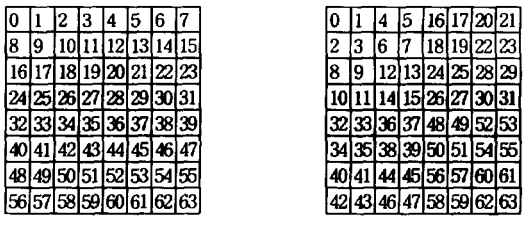
A quadtree of an $n \times n$ binary image is defined as follows : The root represents the entire image. If not all the pixels in the image are of the same color, the root has four sons, quadrants, which represent the NW, NE, SW, and SE blocks of the image, respectively. This decomposition is done until the block represented by the corresponding node consists of only one color. Figure 4(c) is an example of a quadtree representation of the 8×8 binary image given in Figure 4(a). The quadrants of each node are arranged in the order of NW, NE, SW, and SE from the left, and are labeled with 0, 1, 2, or 3 respectively as in the figure.

A linear quadtree[2] is an efficient way of storing a quadtree. It is a collection of black leaf nodes, each of which is represented by a locational code : (S, L), where S is the shuffled row-major number of the top leftmost pixel of the block which is represented by the quadtree node, and L is the level on which the quadtree node is located (the root is on level 0). Note that the shuffled row-major indexing of an 8×8 mesh is given in Figure 5(b). The fact that each node is represented independently

makes the linear quadtree representation to be a good candidate for parallel processing. For instance, the linear quadtree of the binary image in Figure 4(a) is : (0, 2), (13, 3), (14, 3), (22, 3), (24, 2), (33, 3), (34, 3), (36, 2), (40, 2), (45, 3), (46, 3), (48, 1).



(Fig. 4) (a) 8×8 binary image, (b) Block decomposition of the binary image, (c) Quadtree representation



(Fig. 5) (a) Linear indexing, (b) Shuffled-row-major indexing

We now present an algorithm for transforming a binary image to a quadtree. For a binary image of size $n \times n$, we use n^2 processor LAROB. Pixel (r, c), $0 \le r, c < n$, of the image is initially stored in processor P_i such that $i = r \times n + c$. That is, pixels are stored in linear order, one pixel per processor. In addition to the color of the pixel, each processor knows the side length of the image which is n. From this input configuration, the algorithm generates a linear quadtree, and stores its locational codes in increasing order of their SRM indices, one code per processor. The locational codes are stored in the lowest numbered processors of the LAROB.

A brief description of the algorithm is given below. It consists of 7 phases. The background idea of the algorithm has been widely used in the literature [3,4]. Here we give an efficient way of implementing it on the LAROB. The detailed description of each phase follows the algorithm.

Algorithm ImageToQuadtree

1. Compute the SRM index of each pixel.
2. Sort the pixels in increasing order of their SRM indices.
3. Each processor computes the number of trailing black pixels of the run where its pixel belongs.
4. Each processor computes the size of the maximal block its SRM index can represent.
5. Each processor computes the size of the maximal black block it actually represents with the current run.
6. Choose the maximal black blocks.
7. The processors which represent a maximal black block generate the locational code of the block. All the locational codes are then packed to the lowest numbered processors, one code per processor.

Phase 1.

Pixel (r, c), $0 \le r, c < n$, is initially stored in processor P_i, $0 \le i < n^2$, such that $i = r \times n + c$. The purpose of this phase is to compute the SRM index of each pixel. If each processor P_i computes the SRM index of its own pixel independently, it takes $\Theta(\log n)$ steps. Here, we show how it can be done in $\Theta(1)$ time on the LAROB. Phase 1 consists of 5 steps as follows :

Step 1 : For all $0 \le r < n$,
$RH(r) \Leftarrow r_{d-1}0r_{d-2}0\ldots\ r_{\frac{d}{2}}0$

Step 2 : For all $0 \le r < n$,
$RL(r) \Leftarrow r_{\frac{d}{2}-1}0r_{\frac{d}{2}-2}0\cdots r_1 0 r_0 0$

Step 3 : For all $0 \le c < n$,
$CH(c) \Leftarrow 0c_{d-1}0c_{d-2}0\ \cdots\ 0c_{\frac{d}{2}}$

Step 4 : For all $0 \le c < n$,
$CL(c) \Leftarrow 0c_{\frac{d}{2}-1}0c_{\frac{d}{2}-2}\ \cdots 0c_1 0c_0$

Step 5 : For all $0 \le r,\ c < n$, $SRM(r, c) \Leftarrow$
$(RH(r)+CH(c))\times 2^{d}+(RL(r)+CL(c))$

First, the n^2 processors are grouped into n APPBs of n consecutive processors. The processors in each APPB r, $0 \le r < n$, cooperate to compute $RH(r)$. The first $d/2$ processors, PE(r, c), $0 \le c < d/2$, in each APPB r use a bit operation to take out the bit, r_{d-c-1}, from their r, compute $r_{d-c-1}\times 2^{d-2c-1}$, and store it into their local variable $V(r, c)$. $RH(r)$ is actually the sum of $V(r, c)$'s, for all $0 \le c < d/2$. Since each $RH(r)$, $0 \le r < n$, is less than n, we can use Observation 5 to add up $V(r, c)$'s in constant time. $RH(r)$ is then broadcasted to all the processors in APPB r. $RL(r)$ is computed similarly.

In steps 3 and 4, we use one APPB which consists of n^2 processors. For each $0 \le c < n$, $CH(c)$ and $CL(c)$ can be computed using $RH(c)$ and $RL(c)$. Note that $CH(c)$ is a half of $RH(c)$, and $CL(c)$ is a half of $RL(c)$. Thus, during the next two bus cycles, each P_i (= PE(r, c)) sends its $RH(r)$ and $RL(r)$ to P_k (= PE(c, r)) such that $k = c\times n + r$. It is a one-to-one communication and can be done in constant number of steps (by Observation 2). Each P_i (= PE(r, c)) now has four values, $RH(r)$, $RL(r)$, $CH(c)$, and $CL(c)$. Using these values, P_i computes its SRM index, $SRM(r, c)$.

Phase 2.

In this phase, the pixels are sorted in increasing order of their SRM indices. It can be easily done by having each P_i send its pixel to $P_{SRM(i)}$. Since the SRM indices are all different and their range is between 0 and n^2, this routing can be done in constant number of steps (Observation 2).

Phase 3.

Each P_i next computes the number of trailing black pixels of the run where its pixel belongs. Consider the image in Figure 4. The number of trailing black pixels that will be computed by P_i, TailSize(i), is defined as follows:

〈Table 1〉 The number of trailing pixels of a black run

P_i	...	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	...
SRM	...	32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	...
Color	...	0	1	1	0	1	1	1	1	1	1	1	1	0	1	1	0	...
Tail-Size(i)	...	0	2	1	0	8	7	6	5	4	3	2	1	0	2	1	0	...

The purpose of computing TailSize(i)'s is to use them in phases 4-6 to find the maximal black blocks. Phase 3 is divided into 4 steps. In step 1, each processor checks if its pixel represents the first black pixel of the run. This can be done by having P_{i-1} send the color of its pixel to P_i through the bus (i.e., the shift right operation). It is a one-to-one data routing. In step 2, the bus is reconfigured so that only those processors whose pixels belong to the same black run form an APPB. In step 3, in order to compute the number of trailing black pixels in the same run, all the processor with a black pixel set their flag to 1. The prefix sums of these binary values are then computed within each APPB (Observation 4). In step 4, the computed prefix sum of the end processor (i.e. total number of pixels in the run) of each APPB is broadcasted to all the processors of the corresponding APPB. The number of trailing black pixels in each PE is obtained from the broadcasted sum and its own prefix sum.

Phase 4.

As in Table 1, the pixels are currently stored in increasing order of their SRM indices. In this phase, each P_i computes the size of the maximal black block its SRM index can represent. Let MaxBlk1(s) be the size of the maximal block SRM s represents. MaxBlk1(s) is equal to 4^j, where j is the number of trailing zeros of s in base 4. For example, 32 can be written as 200_4. Since the number of trailing zeros is 2, MaxBlk1(32) = $4^2 = 16$.

As in the case of computing the SRM indices in phase 1, we divide n^2 processors into n APPBs of n consecutive processors. We use a similar method which was used for computing n^2 SRM indices. Since the SRM index in P_i is actually i, it can be said that the objective of each P_i is to compute the number of trailing zeros of its own index i. Each APPB r, $0 \le r < n$, is used to compute the number of trailing zeros of integer r, $Z(r)$, when r is represented in base 4. We will show later how to compute $Z(r)$ in constant time. Here, we assume that for each r, $0 \le r < n$, $Z(r)$ has already been computed and stored in PE(r, 0).

In order for each P_i to compute the number of trailing zeros of integer i, P_i needs to collect two values : $Z(r)$ and $Z(c)$, where $r = i$ div n and $c = i$ mod n. Note that $Z(i)$ is equal to $Z(c)$ if $c \neq 0$, $Z(c) + Z(r)$, otherwise. Since the processors that currently have $Z(r)$ and $Z(c)$ are $P_{i \text{ div } n}$ (= PE(r, 0)) and $P_{i \bmod n}$ (= PE(c, 0)), respectively, these two values can be picked up by P_i using two bus cycles.

For the sake of completeness, we show how to compute the number of trailing zeros in integer r, $0 \le r < n$, using APPB r. First, PE(r, c), $0 \le r < n$ and $0 \le c < d-1$, takes out the c-th least significant bit from r (i.e., bit r_c). If it is 0, then it sets a local variable $V(r, c)$ to be 1, 0 otherwise. The prefix sums of $V(r, c)$ are then computed inside APPB r. PE(r, c) has the leading bit of the trailing zeros of r if $\sum_{k=0}^{c} V(r, c) = c+1 = \sum_{k=0}^{c+1} V(r, c)$. This condition is met only when bit $c+1$ is the first least significant bit 1. This condition can be easily checked using one shift left operation. If PE(r, c) has the leading bit 0, then $\lceil c \text{ div } 2 \rceil$ is the number of trailing zeros of r in base 4.

Phase 5.

The objective of each P_i, $0 \le i < n$, is to compute the size of the maximal black block it actually represents with the current run. In order to do so, each P_i with a black pixel first computes the number which is the largest power of four and is less than or equal to TailSize(i). Let MaxBlk2(i) be this number. This number indicates that P_i cannot represent a quadtree block of size larger than MaxBlk2(i), if there are only TailSize(i) number of pixels following it. For instance, if there are only 12 pixels including its pixel, P_i cannot represent a quadtree block of size larger than 4. The size of the maximal quadtree block which is represented by P_i can be computed using Maxblk1(i), MaxBlk2(i). It is RealBlk(i) = min{Maxblk1(i), MaxBlk2(i)}.

The computation of MaxBlk2(i) is very similar to that for MaxBlk1(i). The only difference is that APPB r is used to locate the bit position of the leading 1 of integer r, instead of computing the number of trailing zeros of r. Let x be the bit position of the leading 1 of i. Then, Maxblk2(i) = $4^{\lfloor x/2 \rfloor}$.

Phase 6.

The objective of this phase is to discard the black blocks that are not maximal. In order to do so, each P_i which represents a black block participates in the process of eliminating its quadrant blocks. Suppose that P_i represents a black block of size x. Let $y = x/4$. The processors that represent the quadrant black block of P_i are : P_i, P_{i+y}, P_{i+2y}, P_{i+3y}. So, P_i notifies its block size x to these 3 processors (excluding itself). The

processors which get to know that their blocks are included into a larger block mark themselves as "to be deleted".

Phase 7.

Each P_i, which represents a maximal black block, generates its locational code (i, RealBlk(i)). In order to pack the locational codes to the lowest numbered processors, each processor with a locational code sets its flag to 1. The prefix sums of these values are next computed. The prefix sum, $\sum_{k=0}^{i}$ **flag**, in P_i becomes the index of the destination processor. Finally, the locational codes are moved to their destination processors during the next bus cycle.

Theorem 1.

The above algorithm transforms an $n \times n$ binary image to the corresponding linear quadtree in $\Theta(1)$ time using n^2 processors on the LAROB.

4. From a quadtree to binary image

Here, we present the converse algorithm. The I/O configuration of the algorithm is the reverse of that in section 3. The image reconstruction algorithm can be briefly described as follows. It consists of 3 phases. The detailed description follows.

Algorithm QuadtreeToImage

1. Move each locational code (S, L) to processor P_S.
2. Generate the pixels, and store them into the processors in increasing order of their SRM indices.
3. Sort the pixels in increasing order of their linear indices.

Phase 1.

The input locational codes are initially stored in the lowest numbered processors of the LAROB. These codes are unpacked so that code (S, L) is moved to processor P_S. This data movement can be done in $\Theta(1)$ time, since it is a one-to-one routing and the source processors know the address of the destination processors.

Phase 2.

What has to be done in this phase is to generate 4^L black pixels from each locational code (S, L) and store them in processors P_S, P_{S+1}, P_{S+2}, $\cdots$, P_{S+4^L-1}. In order to do so, each processor with a locational code becomes the leader of an APPB. Next, each leader processor broadcasts its (S, L) to the rest of the processors in the same APPB. Processor P_i, $0 \le i < n^2$, generates a black pixel if $i < S+4^L$. The processors that did not generate a black pixel in this phase set the color of their pixel to be white.

Phase 3.

The pixels of the image are currently stored in increasing order of their SRM indices. They are unshuffled in this phase. First, each processor P_i, $0 \le i < n^2$, computes SRM(i) (same as Phase 1 of the quadtree building algorithm). P_i then sends its tuple (i, SRM(i)) to processor $P_{SRM(i)}$. In order to unshuffle the pixels, each processor $P_{SRM(i)}$ uses the first integer in the received tuple as the address of the destination processor of its pixel. The output binary image gets reconstructed through the pixel unshuffling step.

Theorem 2.

The above algorithm transforms a linear quadtree to the corresponding $n \times n$ binary image in $\Theta(1)$ time using n^2 processors on the LAROB.

5. Conclusions

We have presented algorithms for transforming

between a binary image and a linear quadtree on the LAROB. By fully utilizing the power of the optical buses, these algorithms take $\Theta(1)$ time using n^2 processors for images of size $n \times n$. These algorithms are optimal. Compared to the previously known quadtree bulding algorithms on other computing models, such as mesh, hypercube, and reconfigurable mesh, our algorithms are simpler and efficient. These algorithms show that there is a good possibility that quadtree operations can be efficiently implemented on the LAROB model.

References

[1] F. Dehne, A. G. Ferreira and A. Rau-Chaplin, "Efficient Parallel Construction and Manipulation of Quadtrees," in *International Conference on Parallel Processing*, pp.255-262, 1991.

[2] I. Gargantini, "An Effective Way to Represent Quadtrees," *Communications of the ACM*, Vol.25, No.12, pp.905-910, 1982.

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

[5] M. Kim and J. Jang, "Constant Time Transformation Between Binary Images and Quadtrees on a Reconfigurable Mesh," *Journal of KISS (A) : Computer Systems and Theory*, Vol.23, No.5, pp.454-466, May 1996.

[6] K. Li, Y. Pan, and S.-Q. Zheng, *Parallel Computing Using Optical Interconnections*, Kluwer Academic Publishers, 1998.

[7] R. Miller, V. K. Prasanna Kumar, D. I. Reisis and Q. F. Stout, "Meshes with Reconfigurable Buses," *Proc. 5th MIT Conference On Advanced Research in VLSI*, pp.163-178, 1988.

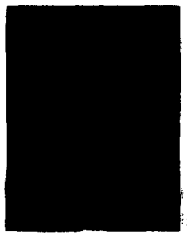
[8] S. Pavel and S. G. Akl, "On the Power of Arrays with Reconfigurable Optical Buses," *Proc. Int'l Conf. Parallel and Distributed Processing Techniques and Applications*, Vol.III, pp.1443-1454, Aug. 1996.

[9] S. Pavel and S. G. Akl, "Integer Sorting and Routing in Arrays with Reconfigurable Optical Buses," *Proc. Int'l Conf. on Parallel Processing*, Vol.2, pp.90-94, 1996.

[10] C. Qiao and R. G. Melhem, "Time-Division Optical Communications in Multiprocessor Arrays," *IEEE Trans. on Computers*, Vol.42, No.5, pp.577-590, May 1993.

[11] S. Rajasekaran and S. Sahni, "Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses," *IEEE Trans. on Parallel and Distributed Systems*, Vol.8, No.11, pp.1123-1132, Nov. 1997.

[12] H. Samet, "Region Representation : Quadtrees from Binary Arrays," *Computer Graphics and Image Processing*, Vol.13, pp.88-93, 1980.



김 명

e-mail : mkim@mm.ewha.ac.kr

1981년 이화여자대학교 수학과(학사)

1983년 서울대학교 계산통계학과(석사)

1990년 미네소타대학교 컴퓨터학과(석사)

1993년 캘리포니아 주립대학교(산타바바라 소재) 컴퓨터학과(박사)

1993년~1994년 캘리포니아 주립대학교(산타바바라 소재) 컴퓨터학과 강사, Postdoc

1995년~현재 이화여자대학교 컴퓨터학과 조교수

관심분야 : 병렬/분산 알고리즘, 광상호연결망 컴퓨팅, 네트워크/클러스터 기반 컴퓨팅