IEICE TRANS. INF. & SYST., VOL. E79–D, NO. 8 AUGUST 1996

PAPER Special Issue on Architectures, Algorithms and Networks for Massively Parallel Computing

A Simple Parallel Algorithm for the Medial Axis Transform

Akihiro FUJIWARA†, *Student Member*, Michiko INOUE†, Toshimitsu MASUZAWA†, *and* Hideo FUJIWARA†, *Members*

SUMMARY The medial axis transform (MAT) is an image representation scheme. For a binary image, the MAT is defined as a set of upright maximal squares which consist of pixels of value 1 entirely. The MAT plays an important role in image understanding. This paper presents a parallel algorithm for computing the MAT of an $n \times n$ binary image. We show that the algorithm can be performed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log\log n)$ time using $n^2/\log\log n$ processors on the common CRCW PRAM. We also show that the algorithm can be performed in $O(n^2/p^2+n)$ time on a $p \times p$ mesh and in $O(n^2/p^2+(n\log p)/p)$ time on a p^2 processor hypercube (for $1 \leqq p \leqq n$). The algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leqq p \leqq \sqrt{n}$) and on the hypercube (for $1 \leqq p \leqq n/\log n$).

key words: *parallel algorithm, image processing, medial axis transform, PRAM, mesh, hypercube*

1. Introduction

The *medial axis transform* (*MAT*) is an image representation scheme proposed by Blum [2]. For a digital binary image, the MAT problem is defined as a problem to find all upright (also called rectilinear or iso-oriented) maximal squares, which consist of pixels of value 1 entirely*. The MAT plays an important role in image understanding [12].

For computing the MAT of an $n \times n$ binary image, an $O(kn^2)$ time sequential algorithm was presented by Vo [14], where k is the longest side length of upright maximal squares in the MAT. A simple $O(n^2)$ time sequential algorithm was presented by Stout [13], which is time optimal since the trivial lower bound of the MAT problem is $\Omega(n^2)$.

Many parallel algorithms were also presented. In general, the efficiency of parallel algorithms is measured by the running time and the number of processors. The *cost* of a parallel algorithm is defined as the product of the running time and the number of processors of the algorithm. Also a parallel algorithm is called *cost optimal* if its cost is equal to the lower bound of sequential time for the problem. Finding fast cost optimal parallel algorithms is a fundamental goal in parallel computation research.

Chandran and Mount [4] presented a parallel algorithm that runs in $O(\log n \log\log n)$ time using n^2 processors on the CREW PRAM. However Kim [8] pointed out that the algorithm has some errors. Chandran et al. [3] presented an algorithm that runs in $O(n)$ time using n processors on the CREW PRAM. Their algorithm is cost optimal. The algorithm can be modified easily to run in $O(n^2/p)$ time on the $p \times p$ mesh ($1 \leqq p \leqq n$). Jenq and Sahni [6] developed an algorithm that runs in $O(\log n)$ time using n^2 processors on the CREW PRAM. They also showed that their algorithm can be mapped on an n^2 processor hypercube so that it runs in $O(\log^2 n)$ time. Kim [8] presented a fast cost optimal algorithm. The algorithm runs in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM.

In this paper, we present a parallel algorithm for computing the MAT. We show that the algorithm can be performed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log\log n)$ time using $n^2/\log\log n$ processors on the common CRCW PRAM. We also show that the algorithm can be performed in $O(n^2/p^2+n)$ time on a $p \times p$ mesh and in $O(n^2/p^2+(n\log p)/p)$ time on a p^2 processor hypercube (for $1 \leqq p \leqq n$). The algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leqq p \leqq \sqrt{n}$) and on the hypercube (for $1 \leqq p \leqq n/\log n$).

Results of our algorithm are comparable to the result of Kim's algorithm [8] on the EREW PRAM. In Kim's algorithm, not only sorting, merging and prefix operations but an algorithm for the all tallest neighbors problem [7] are used. In our algorithm, we mainly use two prefix minima operations. Therefore our algorithm is much simpler. Furthermore, prefix operations are basic operations in parallel computations, and many simple and efficient algorithms are proposed for prefix operations on various parallel computation models. Using these proposed algorithms, our algorithm becomes portable.

In Sect. 2, we give definitions and models of parallel computation. In Sect. 3, we introduce an algorithm for computing the MAT. In Sect. 4, we present implementations of the algorithm on the EREW PRAM and

Manuscript received December 20, 1995.
Manuscript revised March 20, 1996.

†The authors are with the Graduate School of Information Science, Nara Institute of Science and Technology, Ikoma-shi, 630–01 Japan.

*In [12], side length of the square is restricted to a odd number. However, this restriction is unnecessary in practice, we do not restrict the length.

the common CRCW PRAM. We also present implementations on the mesh and the hypercube in Sect. 5. Section 6 is conclusion.

2. Preliminaries

2.1 The MAT Problem

Given an $n \times n$ binary image I, let $I[i,j] \in \{0,1\}$ denote a value for a pixel (i,j) $(0 \leqq i \leqq n-1, 0 \leqq j \leqq n-1)$, where i (resp. j) stands for the row (resp. column) index. We assume the pixel $(0,0)$ is on the top left corner in the image. We call a pixel of value 0 (resp. 1) a *0-pixel* (resp. *1-pixel*).

The *upright maximal square* of an image is defined as a set S of pixels that satisfies the following three conditions:

(c1) All pixels in S are 1-pixels.

(c2) There are two pixels (i_1, j_1), (i_2, j_2) in S such that $i_2 - i_1 = j_2 - j_1$, and a set of pixels $\{(i,j) | i_1 \leqq i \leqq i_2, j_1 \leqq j \leqq j_2\}$ is equal to S.

(c3) There exists no set S' of pixels that satisfies the above two conditions and $S \subset S'$.

The MAT problem is defined as a problem to find all upright maximal squares of an image. Formally, the MAT problem is a problem to find the following $n \times n$ arrays, M and T [6].

$$\begin{aligned} M[i,j] &= 0 \qquad\qquad (\text{if } I[i,j] = 0) \\ &= max\{k | I[i+g, j+h] = 1, \text{for all } g,h \\ &\qquad s.t.\ 0 \leqq g \leqq k-1, 0 \leqq h \leqq k-1\} \\ &\qquad\qquad (\text{if } I[i,j] = 1) \\ T[i,j] &= true \quad (\text{if } max\{M[i,j-1], M[i-1,j], \\ &\qquad M[i-1,j-1]\} \leqq M[i,j] \dotplus 0) \\ &= false \quad (\text{otherwise}) \end{aligned}$$

(We assume that $M[-1,k] = M[k,-1] = 0$ $(-1 \leqq k < n-1)$.)

The $M[i,j]$ stores the side length of the maximum square of 1-pixels whose top left pixel is (i,j). The $T[i,j]$ is true if (i,j) is the top left pixel of a maximal upright square. An example of a binary image and its M and T are illustrated in Fig. 1.

2.2 Parallel Computation Models

Parallel computation models used in this paper are the PRAM, the mesh and the hypercube. The PRAM employs processors which have the capability to access any memory cell in a shared memory synchronously. Several models of the PRAM have been proposed with regard to simultaneous reading and writing to a single memory cell [5]. In this paper, we use the EREW (exclusive read exclusive write) PRAM and the common CRCW (concurrent read concurrent write) PRAM. The EREW PRAM does not allow any simultaneous access to a single memory cell. Simultaneous accesses to the same cell for read or write instructions are allowed on the CRCW PRAM. In the case of concurrent writing, different assumptions are made to resolve the write conflict. On the common CRCW PRAM, processors are allowed to write values to the same memory cell only when they are attempting to write the same value.

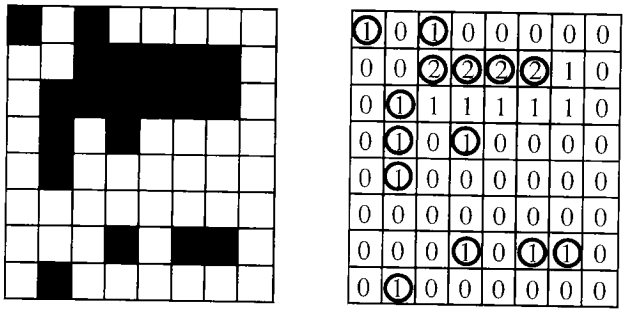


Fig. 1 An example of the MAT: (a) an input image I, and (b) the M and T of I. (Numbers in the figure are values of M, and the number is circled if T is true.)

The $p \times p$ (two-dimensional) mesh is a SIMD machine with p^2 processors arranged into $p \times p$ grid. Let $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$ represent a processor located in a row x and a column y. A processor $P_{x,y}$ is connected to processors $P_{x,y-1}$, $P_{x-1,y}$, $P_{x,y+1}$ and $P_{x+1,y}$, whenever they exist.

The p processor hypercube is also a SIMD machine, which consists of $p = 2^d$ processors interconnected into a d-dimensional Boolean cube. The d-dimensional Boolean cube is defined as follows. Let $z_{d-1}z_{d-2}\cdots z_0$ be the binary representation of z, where $0 \leqq z \leqq p-1$. Then processor P_z is connected to processors $P_{z^{(k)}}$ $(0 \leqq k \leqq d-1)$, where $z^{(k)} = z_{d-1}z_{d-2}\cdots\overline{z_k}\cdots z_0$[†]. In this paper, we assume a 2-D indexing scheme of the processors [11], that is, processor P_z is denoted by $P_{x,y}$, where $z_{d-1}z_{d-2}\cdots z_0 = x_{d'-1}x_{d'-2}\cdots x_0 y_{d'-1}y_{d'-2}\cdots y_0$ and $d = 2d'$.

On the mesh and the hypercube, each processor can send or receive a word to or from one of its connected processors in unit time. In this paper, we define that $A_{x_1,y_1} \leftarrow B_{x_2,y_2}$ denotes a transfer from a variable B of processor P_{x_2,y_2} to a variable A of processor P_{x_1,y_1}.

3. Algorithm

3.1 Overview of the Algorithm

To find the upright maximal square, we find the *maximal upper right triangle* and the *maximal lower left tri-*

[†] $\overline{x_k}$ is the complement of x_k.

(a) (b) (c)

Fig. 2 (a) A square of 1-pixels, (b) An upper right triangle of 1-pixels and (c) A lower left triangle of 1-pixels.

angle. The maximal upper right triangles and the maximal lower left triangles are denoted by the following two $n \times n$ arrays, UR and LL, respectively (See Fig. 2).

$$\begin{aligned} UR[i,j] &= 0 && \text{(if } I[i,j]=0\text{)} \\ &= max\{k | I[i+g, j+h] = 1, \text{for all } g,h \\ &\qquad s.t.\ 0 \leqq g \leqq h \leqq k-1\} \\ &\qquad \text{(if } I[i,j]=1\text{)} \\ LL[i,j] &= 0 && \text{(if } I[i,j]=0\text{)} \\ &= max\{k | I[i+g, j+h] = 1, \text{for all } g,h \\ &\qquad s.t.\ 0 \leqq h \leqq g \leqq k-1\} \\ &\qquad \text{(if } I[i,j]=1\text{)} \end{aligned}$$

Using these two arrays, we compute the MAT in the following four steps.

Step M1: Compute $UR[i,j]$.

Step M2: Compute $LL[i,j]$.

Step M3: Set $M[i,j] = min\{UR[i,j], LL[i,j]\}$.

Step M4: Set $T[i,j] = true$ if $max\{M[i,j], M[i-1,j], M[i-1,j-1]\} \leqq M[i,j]$ and $M[i,j] \neq 0$. Otherwise, set $T[i,j] = false$.

We can compute LL in a similar way to the computation of UR, and we can compute M easily in parallel once UR and LL are obtained. The computation of T is described in Sects. 4 and 5 because it depends on parallel models. In next subsection, we consider only the computation of UR.

3.2 Algorithm for Computing UR

To simplify the algorithm for computing UR, a row and a column of 0-pixels are added to an input image as the bottom row and the rightmost column. Added pixels are $(n,0), (n,1), \ldots, (n-1,n), (n,n)$ and $(0,n), (1,n), \ldots, (n-1,n)$.

We define a set of pixels, $D_{i,j}$, for each pixel (i,j) as follows.

$$\begin{aligned} D_{i,j} = \{(i+g, j+h) | 0 &\leqq g \leqq h \\ &\leqq min\{n-i, n-j\}\} \end{aligned}$$

(Examples of $D_{i,j}$ are illustrated in Fig. 3.)

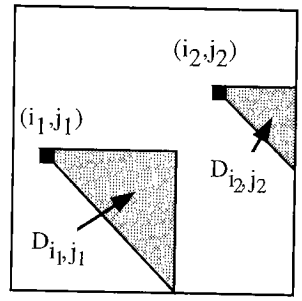


Fig. 3 Examples of $D_{i,j}$.

To compute $UR[i,j]$, all we have to do is to find the smallest column index of the 0-pixel in $D_{i,j}$. Let $S[i,j]$ be the smallest column index of the 0-pixel in $D_{i,j}$, that is, $S[i,j] = min\{j+h | I[i+g, j+h] = 0, 0 \leqq g \leqq h \leqq min\{n-i, n-j\}\}$. Then $UR[i,j] = S[i,j] - j$.

To compute $S[i,j]$ for all pixels (i,j) $(0 \leqq i \leqq n, 0 \leqq j \leqq n)$ efficiently in parallel, we use the prefix minima operation twice. The *prefix minima* of a sequence $(x_1, x_2, \ldots, x_n)$ is defined to be the sequence $(m_1, m_2, \ldots, m_n)$, such that $m_k = min\{x_h | 1 \leqq h \leqq k\}$.

First, we compute $MD[i,j]$, which defined as below, for each pixel (i,j).

$$\begin{aligned} MD[i,j] = min\{j+k | I[i+k, j+k] = 0, \\ 0 \leqq k \leqq min\{n-i, n-j\}\} \end{aligned}$$

MD can be computed from a prefix minima of column indexes of 0-pixels, which are arranged diagonally from lower right to upper left. It is easy to see that

$$\begin{aligned} S[i,j] = min\{MD[i, j+k] | \\ 0 \leqq k \leqq min\{n-i, n-j\}\}. \end{aligned}$$

Therefore, we can compute S from the prefix minima of MD, which are arranged horizontally (on each row) from right to left.

The algorithm we propose is presented in the following.

Algorithm for computing UR

Step U1: (Initialization)

For each pixel (i,j) $(0 \leqq i \leqq n, 0 \leqq j \leqq n)$, set $C[i,j] = j$ if $I[i,j] = 0$ else set $C[i,j] = n^{\dagger}$.

Step U2: (Compute the prefix minima diagonally)

Compute the prefix minima of C for each diagonal sequence $C[n,u], C[n-1,u-1], \ldots, C[n-u,0]$ and $C[v,n], C[v-1,n-1], \ldots, C[0,n-v]$ $(0 \leqq u \leqq n, 0 \leqq v \leqq n-1)$ and store the results in an $(n+1) \times (n+1)$ array MD, i.e. set

$$\begin{aligned} MD[i,j] = min\{C[i+k, j+k] | \\ 0 \leqq k \leqq min\{n-i, n-j\}\} \end{aligned}$$

†For our implementation, we set n, but it means ∞ in fact.

for each i, j $(0 \leqq i \leqq n, 0 \leqq j \leqq n)$.

Step U3: (Compute the prefix minima horizontally)

Compute the prefix minima of MD for each horizontal sequence $MD[i,n]$, $MD[i,n-1]$, ..., $MD[i,0]$ $(0 \leqq i \leqq n)$ and store the result in an $(n+1) \times (n+1)$ array S, i.e. set

$$S[i,j] = min\{MD[i,j+k] | 0 \leqq k \leqq n-j\}$$

for each i, j $(0 \leqq i \leqq n, 0 \leqq j \leqq n)$.

Step U4: (Compute UR)

For each pixel (i,j) $(0 \leqq i \leqq n, 0 \leqq j \leqq n)$, set $UR[i,j] = S[i,j] - j$. □

4. Implementation on the PRAM

We can compute LL in a similar way to the computation of UR. We can also compute M (resp. T) easily in $O(1)$ time using n^2 processors on any PRAM once UR and LL (resp. M) are obtained. Thus the complexity for the computation of UR dominates the complexity of the entire algorithm for computing the MAT.

In our algorithm for computing UR, step U1 and step U4 can be executed in $O(1)$ time using n^2 processors on any PRAM easily. In step U2 and step U3, prefix minima computations are performed. It is known that a prefix minima computation of m numbers can be performed in $O(\log m)$ time using $m/\log m$ processors on the EREW PRAM [10] and in $O(\log\log m)$ time using $m/\log\log m$ processors on the common CRCW PRAM [1]. Therefore, we can compute the prefix minima of our algorithm in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log\log n)$ time using $n^2/\log\log n$ processors on the common CRCW PRAM by applying this algorithm to each diagonal sequence or each horizontal sequence because both the length of each sequence and the number of the sequences are $O(n)$. Consequently we obtain the following theorem.

Theorem 1: The medial axis transform of an $n \times n$ binary image can be computed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log\log n)$ time using $n^2/\log\log n$ processors on the common CRCW PRAM. □

5. Implementation on the Mesh and the Hypercube

In this section, we describe an implementation of our algorithm. The implementation is used on both the mesh and the hypercube. We mention the implementation at first, and the complexity for each model next.

5.1 An Implementation

For simplicity, we assume that the number of processors is p^2 and that $n+1 = L \cdot p$ for some positive constant L. Each processor is denoted by $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$. Also we assume that an input image is partitioned into p^2 subsquares of the same size, and that each processor $P_{x,y}$ has a subsquare $I_{x,y}$ defined as follows.

$$I_{x,y}[g,h] = I[x*L+g, y*L+h] \quad (0 \leqq g \leqq L-1, 0 \leqq h \leqq L-1)$$

Arrays in our algorithm, C, MD, S and UR, are distributed similarly.

On this distribution, LL can be computed in a similar way to computation of UR, and M can be computed on each processor on the mesh and the hypercube once UR and LL are obtained. Once M is obtained, we can compute T by computations on each processor and transfers of L elements of M from $P_{x-1,y}$ to $P_{x,y}$ L times, from $P_{x,y-1}$ to $P_{x,y}$ L times and one element of M from $P_{x-1,y-1}$ to $P_{x,y}$. These transfers can be implemented by applying row and column shifts $O(n/p)$ times. (Note $L = O(n/p)$.) Each shift can be performed in $O(1)$ time on a $p \times p$ mesh and in $O(\log p)$ time on a p^2 hypercube [6]. The computation on each processor can be executed in $O(n^2/p^2)$ time easily. Therefore, T can be obtained in $O(n^2/p^2)$ time on the mesh and in $O(n^2/p^2 + (n\log p)/p)$ time on the hypercube. In the rest of this subsection, we consider only the computation of UR.

In the algorithm that computes UR, all computations can be locally performed on each processor except for two prefix minima computations; one is in step U2 and the other is in step U3. We implement these two prefix computations with the well-known divide and conquer approach. Let $a_0, a_1, \ldots, a_{m-1}$ be a sequence for which we compute the prefix minima. Assume that $m = k \cdot q$ for some positive integer k, and that this sequence is partitioned into the q subsequences of length k and stored on processors $P_0, P_1, \ldots, P_{q-1}$, where processor P_i stores a subsequence $a_{i*k}, a_{i*k+1}, \ldots, a_{(i+1)*k-1}$. We can compute a prefix minima of the sequence as follows. The result is stored as a sequence $d_0, d_1, \ldots, d_{m-1}$.

(1) Compute the minimum number of the subsequence on each processor, i.e. set

$$b_i = min\{a_j | i*k \leqq j \leqq (i+1)*k-1\}$$

on processor P_i $(0 \leqq i \leqq q-1)$.

(We call this minimum number *local minimum*.)

(2) Compute the prefix minima of the sequence of local minima $b_0, b_1, \ldots, b_{q-1}$ using communications. Let a sequence $c_0, c_1, \ldots, c_{q-1}$ be the result. For each i $(1 \leqq i \leqq q-1)$, P_i stores c_{i-1}. (c_{q-1} is ignored.) Set $c_{-1} = \infty$ on processor P_0.

(3) Compute the prefix minima on each processor, i.e. set

$$d_{i*k+h} = min\{c_{i-1}, min\{a_j | i*k \leqq j \leqq i*k+h\}\}$$

for all h $(0 \leqq h \leqq k-1)$ on processor P_i $(0 \leqq i \leqq q-1)$.

In the above computation, first and last steps are executed on each processor. Therefore the second step is a key for the implementation.

In the case of the prefix minima in step U3, it can be implemented easily as follows. This is because each sequence for which we compute the prefix minima is on processors arranged in a row, and they form a p processor array (on the mesh) or a p processor hypercube (on the hypercube). Efficient prefix minima algorithms are known on these parallel models.

Implementation of the prefix minima in step 3

1. (Compute the minimum numbers locally)

   Find a local minimum for each horizontal subsequence on each processor sequentially, i.e. set

   $$MS_{x,y}[g] = min\{MD_{x,y}[g,k] | 0 \leqq k \leqq L-1\}$$

   for each g $(0 \leqq g \leqq L-1)$ on processor $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$.

2. (Compute the prefix minima using communications)

   For each set of processors arranged in a row x, compute the prefix minima of the local minima, i.e. set

   $$MS_{x,y}[g] = min\{MS_{x,k}[g] | y \leqq k \leqq p-1\}$$

   for each g $(0 \leqq g \leqq L-1)$.

3. For each set of processors arranged in a row x, shift MS one processor left, i.e. set

   $$MS_{x,y-1}[g] \leftarrow MS_{x,y}[g]$$

   for each g $(0 \leqq g \leqq L-1)$. Also set $MS_{x,p-1}[g] = n$ $(0 \leqq g \leqq L-1)$ on processor $P_{x,p-1}$ $(0 \leqq x \leqq p-1)$.

4. (Compute the prefix minima on each processor)

   Compute the prefix minima for each horizontal subsequence on each processor sequentially, i.e. set

   $$S_{x,y}[g,h] = min\{MS_{x,y}[g], min\{MD_{x,y}[g,k] | h \leqq k \leqq L-1\}\}$$

   for each g, h $(0 \leqq g \leqq L-1, 0 \leqq h \leqq L-1)$ on processor $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$. □

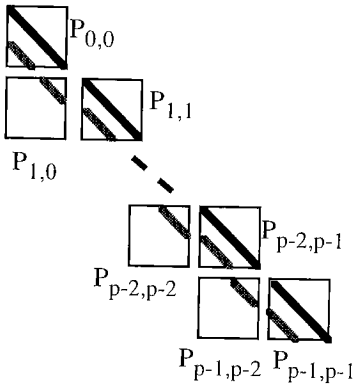


Fig. 4 An example of diagonal sequences on processors. (Two lines denote diagonal sequences.)

The prefix minima computation in step U2 can be implemented with the same idea, however, there is one remarkable difference; each diagonal sequence is on diagonally arranged processors. For example, a diagonal sequence $C[n,n]$, $C[n-1,n-1]$, ..., $C[0,0]$ is on processors $P_{p-1,p-1}$, $P_{p-2,p-2}$, ..., $P_{0,0}$, and $C[n,n-1]$, $C[n-1,n-2]$, ..., $C[1,0]$ is on $P_{p-1,p-1}$, $P_{p-1,p-2}$, $P_{p-2,p-2}$, $P_{p-2,p-3}$, ..., $P_{1,1}$, $P_{1,0}$, $P_{0,0}$ (See Fig. 4). These processors do not form a processor array or a hypercube. Thus, we cannot compute the prefix minima in a similar way to that in step U3.

To resolve this difficulty, we move the local minima to processors arranged in a row or a column. For example, let $M_{p-1,p-1}$, $M_{p-2,p-2}$, ..., $M_{0,0}$ be the local minima of one diagonal sequence on diagonally arranged processors, which are $P_{p-1,p-1}$, $P_{p-2,p-2}$, ..., $P_{0,0}$, respectively. We shift the local minimum $M_{k,k}$ $(p-1-k)$-processors right, that is, $M_{k,p-1} \leftarrow M_{k,k}$ $(0 \leqq k \leqq p-1)$. After shifting right, these local minima are on processors $P_{p-1,p-1}$, $P_{p-2,p-1}$, ..., $P_{0,p-1}$. For another example, let $M_{p-1,p-1}$, $M_{p-1,p-2}$, $M_{p-2,p-2}$, $M_{p-2,p-3}$, ..., $M_{1,1}$, $M_{1,0}$, $M_{0,0}$ be local minima. We shift the local minimum M_{k_1,k_1} $(p-1-k_1)$-processors right and the local minimum M_{k_2,k_2-1} $(p-k_2)$-processors right, that is, $M_{k_1,p-1} \leftarrow M_{k_1,k_1}$ $(0 \leqq k_1 \leqq p-1)$ and $M_{k_2,p-1} \leftarrow M_{k_2,k_2-1}$ $(1 \leqq k_2 \leqq p-1)$. (Figure 5 illustrates the idea.) This movement can be performed using $O(n/p)$ row or column shifts in parallel.

Once the local minima are on processors arranged in a column or a row, we can compute the prefix minima easily because these processors form a processor array or a hypercube. After computing the prefix minima of the local minima, we re-shift the result inverse to the first shift so as to restore them to required processors.

Details of the implementation are described below. For clarity of description, computations of shift and the prefix minima computation are performed in two stages. In former stage, we compute the prefix minima of lower left diagonal sequences, which are $C[n,u], C[n-1,u-1], \ldots, C[n-u,0]$ $(0 \leqq u \leqq n)$, and we compute the prefix minima of upper right sequences

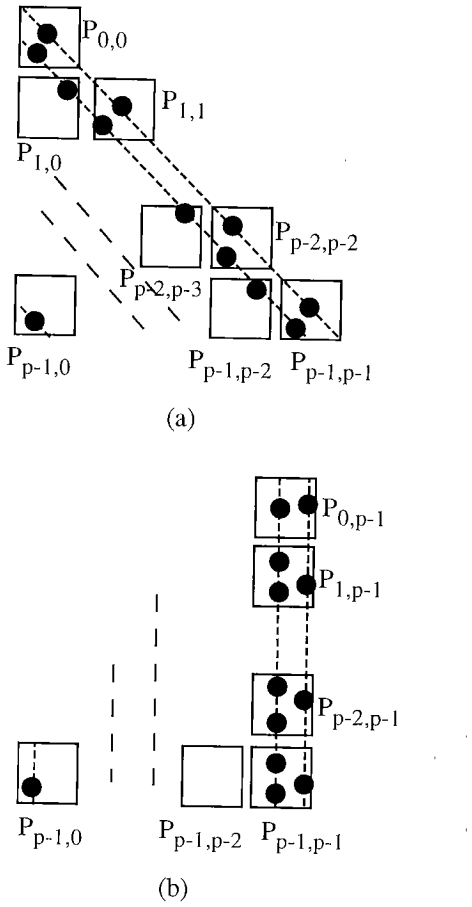


Fig. 5 An example of the shift (right): (a) diagonal sequences before shifting, and (b) diagonal sequences after shifted right. (Black circles denote local minima.)

in latter stage.

Implementation of the prefix minima in step 2

1. (Find the local minimum on each processor for each diagonal subsequence)

Find the local minimum for each diagonal subsequence on each processor sequentially, i.e. set

$$MD1_{x,y}[u] = min\{C_{x,y}[L-1-k, u-k] \mid 0 \leqq k \leqq u\}$$
$$MD2_{x,y}[v] = min\{C_{x,y}[v-k, L-1-k] \mid 0 \leqq k \leqq v\}$$

for each u, v $(0 \leqq u \leqq L-1, 0 \leqq v \leqq L-2)$ on a processor $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$.

(MD_1 and MD_2 denote the local minima of lower left diagonal subsequences and upper right subsequences on each processor, respectively.)

2. (Compute the prefix minima for lower left sequences)

(a) For each processor $P_{x,y}$ $(0 \leqq x \leqq p-1, 0 \leqq y \leqq p-1)$, set

$$MS1_{x,y}[k_1] = MD1_{x,y}[k_1] \quad (\text{if } y \leqq x)$$
$$= n \quad (\text{otherwise})$$
$$MS2_{x,y}[k_2] = MD2_{x,y}[k_2] \quad (\text{if } y < x)$$
$$= n \quad (\text{otherwise})$$

for each k_1, k_2 $(0 \leqq k_1 \leqq L-1, 0 \leqq k_2 \leqq L-2)$. Also set $MD2_{x,y}[-1] = n$.

(b) For each set of processors arranged in a row x $(0 \leqq x \leqq p-1)$, shift $MS1$ $((p-1)-x)$-processors right, and shift $MS2$ $((p-1)-x+1)$-processors right, i.e. set

$$MS1_{x,((y+(p-1)-x) mod (p-1))}[g_1] \leftarrow MS1_{x,y}[g_1]$$
$$MS2_{x,((y+(p-1)-x+1) mod (p-1))}[g_2] \leftarrow MS2_{x,y}[g_2]$$

for each g_1, g_2 $(0 \leqq g_1 \leqq L-1, -1 \leqq g_2 \leqq L-2)$.

(After this, each diagonal sequence is on processors arranged in a column.)

(c) For each set of processors arranged in a column y $(0 \leqq y \leqq p-1)$, compute prefix minima of the results, that is compute prefix minima of $MS1_{p-1,y}[k]$, $MS2_{p-1,y}[L-2-k]$, $MS1_{p-2,y}[k]$, $MS2_{p-2,y}[L-2-k]$, ..., $MS1_{1,y}[k]$, $MS2_{1,y}[L-2-k]$, $MS1_{0,y}[k]$ $(0 \leqq k \leqq L-1)$, and store the results to $MS1$ and $MS2$ again.

(d) For each set of processors arranged in a column y $(0 \leqq y \leqq p-1)$, shift the results of the prefix minima one-processor up, i.e. set

$$MS2_{x,y}[L-2-k_1] = MS1_{x,y}[k_1]$$
$$MS1_{x-1,y}[L-2-k_2] \leftarrow MS2_{x,y}[k_2]$$

for each k_1, k_2 $(0 \leqq k_1 \leqq L-1, -1 \leqq k_2 \leqq L-2)$.

(e) For each set of processors arranged in a row x $(0 \leqq x \leqq p-1)$, shift $MS1$ $((p-1)-x)$-processors left, and shift $MS2$ $((p-1)-x+1)$-processors left, i.e. set

$$MS1_{x,((y-(p-1)+x) mod (p-1))}[g_1] \leftarrow MS1_{x,y}[g_1]$$
$$MS2_{x,((y-(p-1)+x-1) mod (p-1))}[g_2] \leftarrow MS2_{x,y}[g_2]$$

for each g_1, g_2 $(0 \leqq g_1 \leqq L-1, 0 \leqq g_2 \leqq L-2)$.

(f) For each processor $P_{x,y}$ $(0 \leqq y \leqq x \leqq p-1)$, set

$$C_{x,y}[L-1, u] = min\{C_{x,y}[L-1, u], MD1_{x,y}[u]\} \quad (\text{if } y = x)$$
$$C_{x,y}[v, L-1] = min\{C_{x,y}[v, L-1], MD2_{x,y}[v]\} \quad (\text{if } y < x).$$

for each u, v $(0 \leqq u \leqq L-1, 0 \leqq v \leqq L-2)$.

(g) Compute the prefix minima for each diagonal subsequence of local image sequentially, i.e. set

$$MD_{x,y}[g, h] = min\{C_{x,y}[g+k, h+k] \mid 0 \leqq k \leqq min\{L-1-g, L-1-h\}\}$$

for each g, h $(0 \leqq g \leqq L-1, 0 \leqq h \leqq L-1)$ on a processor $P_{x,y}$ $(0 \leqq y < x \leqq p-1)$ and for each g, h $(0 \leqq h \leqq g \leqq L-1)$ on a processor $P_{x,y}$ $(0 \leqq x \leqq p-1, y = x)$.

3. (Compute the prefix minima for upper right sequences)

Compute in a similar way to 2. □

5.2 Complexities

Finding local minima and prefix minima computations on each processor can be performed sequentially in $O(n^2/p^2)$ time on both the mesh and the hypercube models.

Next we consider the complexity on the mesh. Computation of shifts by rows or columns can be performed in $O(p \times n/p) = O(n)$ time on the mesh because one row or column shift can be performed in $O(p)$ time and the number of values to be shifted on each processor is $O(n/p)$. For the same reason, prefix minima for each row and column can be computed in $O(n)$ time.

Finally, we consider the complexity on the hypercube. It is known that shift of m numbers on an m processor hypercube can be performed in $O(\log m)$ time [9]. Because processors in a row or column form a complete p processor hypercube, we can shift the sequence in $O(\log p)$ time. Since the number of diagonal sequences on each processor is $O(n/p)$, we can perform the whole shifts in $O((n \log p)/p)$ time. For the prefix minima computation, it is also known that the prefix minima of m number can be computed in $O(\log m)$ times on a m processor hypercube [9]. Therefore, applying the algorithm to compute the prefix minima on each row or column, the complexity of the prefix computation becomes $O((n \log p)/p)$ time.

In consequence, we obtain the following theorem.

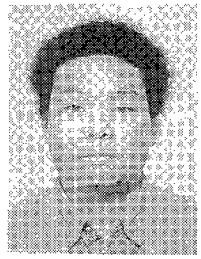
Theorem 2: The medial axis transform of an $n \times n$ binary image can be computed in $O(n^2/p^2 + n)$ time on a $p \times p$ mesh and in $O(n^2/p^2 + (n \log p)/p)$ time on a p^2 processor hypercube (for $1 \leqq p \leqq n$). □

6. Conclusion

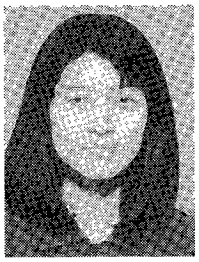
In this paper, we presented a parallel algorithm for computing the MAT. The algorithm has the smaller number of operations than any other known parallel algorithms for the MAT. The algorithm can be performed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log \log n)$ time using $n^2/\log \log n$ processors on the common CRCW PRAM. We also showed that the algorithm can be performed in $O(n^2/p^2 + n)$ time on a $p \times p$ mesh and in $O(n^2/p^2 + (n \log p)/p)$ time on a p^2 processor hypercube (for $1 \leqq p \leqq n$). Consequently the algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leqq p \leqq \sqrt{n}$) and on the hypercube (for $1 \leqq p \leqq n/\log n$).

References

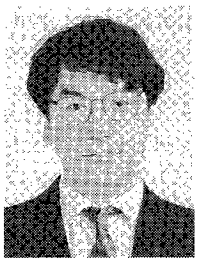
[1] O. Berkman, B. Schieber, and U. Vishkin, "Optimal doubly logarithmic optimal parallel algorithm based on finding all nearest smaller values," Journal of Algorithms, vol.14, pp.344–370, 1993.
[2] H. Blum, "Models for the perception of speech and visual form," vol.11, pp.362–380, MIT Press, 1967.
[3] S. Chandran, S.K. Kim, and D.M. Mount, "Parallel computational geometry of rectangles," Algorithmica, vol.7, no.1, pp.25–49, 1992.
[4] S. Chandran and D. Mount, "Shared memory algorithms and the medial axis transform," Proc. IEEE Workshop on Computer Architecture for PAMI, pp.44–50, 1987.
[5] J. JáJá, "An Introduction to Parallel Algorithms," Addison-Wesley Publishing Company, 1992.
[6] J-F. Jenq and S. Sahni, "Serial and parallel algorithms for the medial axis transform," IEEE Trans. Pattern Analysis and Machine Intelligence, vol.14, no.12, pp.1218–1224, 1992.
[7] S.K. Kim, "Optimal parallel algorithms on sorted intervals," Proc. 27th Annual Allerton Conference Comm. Control and Comput., Allerton, IL, pp.766–775, 1989.
[8] S.K. Kim, "Optimal parallel algorithms for region labeling and medial axis transform of binary images," SIAM Journal of Discrete Mathematics, vol.4, no.3, pp.385–396, 1991.
[9] V. Kumar, A. Grama, A. Gupta, and G. Karypis, "Introduction to Parallel Computing, Design and Analysis of Algorithms," The Benjamin/Cummings Publishing Company, Inc., 1994.
[10] R.E. Lander and M.J. Fisher, "Parallel prefix computation," Journal of ACM, vol.27, pp.831–838, 1980.
[11] S. Ranka and S. Sahni, "Hypercube algorithms with applications to image processing and pattern recognition," Springer-Verlag, New York, 1990.
[12] A. Rosenfeld and A.C. Kak, "Digital Picture Processing," Academic Press, 1982.
[13] D.F. Stout, "Prob 80-4: Improved solution," Journal of Algorithms, vol.4, pp.176–177, 1983.
[14] K. Vo, Prob 80-4, "Journal of Algorithms," vol.4, pp.366–367, 1982.



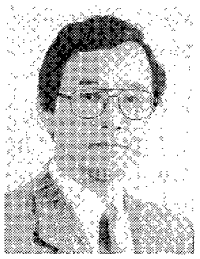
Akihiro Fujiwara received the B.E. degree from Osaka University in 1993 and the M.E. degree from Nara Institute of Science and Technology (NAIST) in 1995. He is currently a Ph.D. student at NAIST. His main research interests are design and analysis of parallel algorithms. He is a member of IEEE and IPSJ.



Michiko Inoue received her B.E., M.E. and Ph.D. degrees from Osaka university in 1987, 1989, and 1995 respectively. She is a research associate of graduate school of information science, Nara institute of science and technology. Her research interests are distributed algorithms, parallel algorithms, graph theory and design and synthesis of digital systems. She is a member of IPSJ and JSAI.



Toshimitsu Masuzawa received the B.E., M.E. and D.E. degrees in computer science from Osaka University in 1982, 1984 and 1987. He had worked at Education Center for Information Processing, Osaka University between 1987–1990, and had worked at Faculty of Engineering Science, Osaka University between 1990–1994. He is now an associate professor of Graduate School of Information Science, Nara Institute of Science and Technology (NAIST). He was also a visiting associate professor of Department of Computer Science, Cornell University between 1993–1994. His research interests include distributed algorithms, parallel algorithms and graph theory. He is a member of ACM, IEEE, EATCS and the Information Processing Society of Japan.



Hideo Fujiwara received the B.E., M.E. and Ph.D. degrees in electronic engineering from Osaka University, Osaka, Japan, in 1969, 1971, and 1974, respectively. He was with Osaka University from 1974 to 1985 and Meiji University from 1985 to 1993, and joined Nara Institute of Science and Technology in 1993. In 1981 he was a Visiting Research Assistant Professor at the University of Waterloo, and in 1984 he was a Visiting Associate Professor at McGill University, Canada. Presently he is a Professor at the Graduate School of Information Science, Nara Institute of Science and Technology, Nara, Japan. His research interests are logic design, fault-tolerant computing and design automation, including design for testability, built-in self-test, test pattern generation, parallel processing, and computational complexity. He is the author of Logic Testing and Design for Testability (MIT Press, 1985). He received the IECE Young Engineer Award in 1977, IEEE Computer Society Certificate of Appreciation Award in 1991 and Okawa Prize for Publication in 1994. He was an editor of IEEE Design and Test of Computers, IEICE Trans. on Information and Systems, and now an editor of J. Electronic Testing, J. Circuits, Systems and Computers, J. VLSI Design and others. Dr. Fujiwara is a fellow of the IEEE as well as a member of Information Processing Society of Japan.