An Optimal Parallel Algorithm for the Euclidean Distance Maps

Akihiro Fujiwara, Toshimitsu Masuzawa and Hideo Fujiwara

Graduate School of Information Science
Nara Institute of Science and Technology (NAIST)
8916-5 Takayama, Ikoma, Nara, 630-01 Japan

E-mail: akihir-f@is.aist-nara.ac.jp

Abstract: The *Euclidean distance map (EDM)* is a map where each element has the Euclidean distance between the correponding pixel and the nearest black pixel in an input binary image. The EDM plays an important role in the field of machine vision, pattern recognition, and robotics. In recent years, several $O(n^2)$ time sequential algorithms have been proposed for the EDM of an $n \times n$ binary image.

We present a PRAM algorithm for computing the EDM. This algorithm can be performed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log n/\log\log n)$ time using $n^2 \log\log n/\log n$ processors on the common CRCW PRAM, respectively. This algorithm is applicable to a wide class of distance maps as well as the EDM. The class contains, for example, cityblock, chessboad, octagonal and chamfer distance maps.

Key words: Parallel algorithms, image processing, distance maps, PRAM.

1 Introduction

The *Euclidean distance map (EDM)* of a black and white $n \times n$ binary image is the $n \times n$ map where each element has the Euclidean distance between the corresponding pixel and the nearest black pixel. The EDM plays an important role in machine vision, pattern recognition, and robotics[9].

Many algorithms have been proposed for computing the EDM. Yamada[10] presented an EDM algorithm that runs in $O(n)$ time using n^2 processors on an 8-neighbor connected mesh array. By the straight forward simulation of the algorithm, we obtain $O(n^3)$ time sequential algorithm. An $O(n^2 \log n)$ sequential algorithm for computing the EDM was presented by Kolountzakis and Kutulakos[8] . They also showed that their algorithm can be modified to run in $O(n^2 \log n/p)$ time using p processors $(1 \leq p \leq n)$ on the EREW PRAM.

In recent years, two $O(n^2)$ time sequential algorithms were presented for computing the EDM [2][6]. These algorithms are optimal since the trivial lower bound of the EDM problem is $\Omega(n^2)$. Chen and Chuang's algorithm[2] is based on the algorithm presented in [8]. They showed that their algorithm can be performed in $O(n^2/p + n \log p)$ time using p processors on the EREW PRAM . On the other hand, Hirata and Kato's algorithm[6] takes a geometrical approach. They also showed that their algorithm can be parallelized to run in $O(n^2/p)$ time using p processors $(1 \leq p \leq n)$ on the EREW PRAM.

In this paper, we present a parallel algorithm for computing the EDM on the PRAM. The algorithm can be performed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log n/\log\log n)$ time using $n^2 \log\log n/\log n$ processors on the common CRCW PRAM, respectively. The algorithm is optimal in the sense that the product of the time and the number of processors is equal to the lower bound of the sequential time for computing the EDM. Our algorithm is based on Hirata and Kato's algorithm[6], which is applicable to a wide class of distance maps as well as the Euclidean distance map. The class contains, for example, cityblock, chessboard, octagonal and chamfer distance maps. Our algorithm is also applicable to the same class of distance maps.

In Section 2, the definition and models of parallel computation which are used throughout this paper are given. Section 3 introduces the algorithm proposed by Hirata and Kato[6]. In Section 4, we present our parallel algorithm and describe complexities on the EREW PRAM and on the arbitrary CRCW PRAM. Section 5 discusses the class of distances that our parallel algorithm is applicable to.

2 Preliminaries

Given a black and white $n \times n$ binary image, let (i, j) denote each pixel $(1 \leq i, j \leq n)$. The index i stands for the row number, and the index j stands for the column number. We assume the pixel $(1, 1)$ is the one on the upper left corner in the image. Let BL denote the set of all black pixels. The *Euclidean distance map (EDM)* of the image is an $n \times n$ array ED, where each element ED_{ij} stores the Euclidean distance from pixel (i, j) to the nearest black pixel. For each i and j $(1 \leq i, j \leq n)$, ED_{ij} is defined as follows.

$$ED_{ij} = min_{(x,y)\in BL}\sqrt{(i-x)^2 + (j-y)^2}$$

An example of a binary image and its EDM is illustrated in Fig.1.

For each row i $(1 \leq i \leq n)$, we define BL_i as the set of all black pixels in the ith row or above, namely $BL_i = BL \cap \{(x, y)|x \leq i\}$. In this paper, we consider the computation of the Euclidean distance from pixel (i, j) to the nearest black pixel in BL_i: if we want to obtain the complete EDM, all we have to do is to turn image upside down, to apply the same algorithm and to compare the two Euclidean distances. The Euclidean distance from pixel (i, j) to the nearest black pixel in BL_i is denoted by d_{ij}.

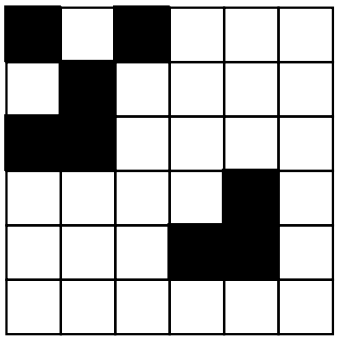
Since we treat some geometric problems in this paper, we present definitions of them. Let $F = \{f_1(x), f_2(x), \ldots, f_n(x)\}$ be a set of functions. The *lower envelope* of F is a function $h(x) = min_{1\leq k\leq n} f_k(x)$. We say $f_i(x)$ is *on the lower envelope* of F if $f_i(x') = h(x')$ holds for some x'.

The *convex hull* of point set P is the smallest convex polygon of which each point in P is either on the boundary or in the interior. A point of P is called an *extreme point* of P if it is on the boundary of the convex hull. Let p_l and p_r be the extreme points of P with the smallest and the largest x coordinates. Notice that p_l and p_r are uniquely determined in our application of the convex hull, since every point has a distinct x coordinate. By cutting the boundary of the convex hull at p_l and p_r, the boundary is divided into two chains, an *upper hull* and a *lower hull*. For completeness, we assume that p_l and p_r are contained in both the upper and the lower hull.

The parallel computation model used in this paper is the EREW (exclusive read exclusive write) PRAM and the common CRCW (concurrent read concurrent write) PRAM. The PRAM employs processors which have the capability to access any memory cell in the shared memory synchronously. Several models of the PRAM have been proposed with regard to simultaneous reading and writing to a single memory location[7].

The EREW PRAM does not allow any simultaneous access to a single memory loca-

tion. Simultaneous access to the same location for a read or a write instruction is allowed on the CRCW PRAM. In the case of concurrent writing, different assumptions are made about which processor's value is written into the memory location. On the common CRCW PRAM, processors are allowed to write to the same memory location only when they are attempting to write the same value. On the common CRCW PRAM, processors are allowed to write to the same memory location only when they are attempting to write the same value.



(a)

0	1	0	1	2	3
1	0	1	$\sqrt{2}$	2	$\sqrt{5}$
0	0	1	$\sqrt{2}$	1	$\sqrt{2}$
1	1	$\sqrt{2}$	1	0	1
2	2	1	0	0	1
3	$\sqrt{5}$	$\sqrt{2}$	1	1	$\sqrt{2}$

(b)

Figure 1: An example of the Euclidean distance map: (a) an input image I. (b) the Euclidean distance map of I.

3 Hirata and Kato's algorithm

Our parallel algorithm can be considered as a parallel implementation of the optimal sequential algorithm presented by Hirata and Kato[6]. We first briefly describe the sequential algorithm. The algorithm consists of three steps.

Hirata and Kato's algorithm [6]
step 1: For every pixel (i, j) $(1 \leq i, j \leq n)$, compute the Euclidean distance g_{ij} to the nearest black pixel in the same column:

$$g_{ij} = i - max_{(k,j) \in BL_i} k$$

If no black pixel is in $\{(1, j), (2, j), \ldots, (i, j)\}$, set $g_{ij} = +\infty$.
step 2: Let $f_{i,k}(j)$ be the Euclidean distance from (i, j) to the nearest black pixel in a column k, namely $f_{i,k}(j) = \sqrt{(j-k)^2 + g_{ik}^2}$. Since $d_{ij} = min_{1 \leq k \leq n} f_{i,k}(j)$, compute a function $h_i(x) =$

$min_{1 \leq k \leq n} f_{i,k}(x)$, for each row i $(1 \leq i \leq n)$. (Notice that the function $h_i(x)$ is the lower envelope of $F_i = \{f_{i,k}(x) | 1 \leq k \leq n\}$.)

step 3: Compute d_{ij} from $h_i(x)$ for each i, j $(1 \leq i, j \leq n)$. □

Fig.2. shows an input 6×6 image, the set of functions $F_5 = \{f_{5,1}, f_{5,2}, f_{5,3}, f_{5,4}, f_{5,5}\}$ and the lower envelope of F_5. The Euclidean distance d_{51} can be computed from $f_{5,1}$, d_{52} can be computed from $f_{5,2}$, d_{53} and d_{54} can be computed from $f_{5,4}$, and so on.

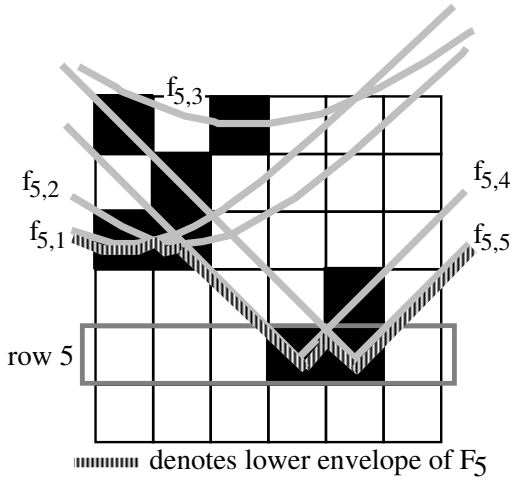


Figure 2: An example of the lower envelope. The function set F_5 is $\{f_{5,1}, f_{5,2}, f_{5,3}, f_{5,4}, f_{5,5}\}$.

The first step can be performed in $O(n^2)$ time by scanning each column downward. In the second step, we compute the lower envelope by scanning each row from left to right. Using the similar technique to Graham's scan for the convex hull problem[1], we can compute the lower envelope of F_i in $O(n)$ time: the total running time of the second step is $O(n^2)$. We use a stack to store the lower envelope found in the second step. The last step can be performed in $O(n^2)$ time by popping the stack and storing the each Euclidean distance to the element from right to left. Therefore, overall of the algorithm can be performed in $O(n^2)$ time sequentially. Hirata and Kato[6] also pointed out that the algorithm can be performed in $O(n^2/p)$ time using p processors $(1 \leq p \leq n)$ on the EREW PRAM by processing rows or columns in parallel.

4 A parallel algorithm

4.1 An overview of the algorithm

In this section, we present a parallel algorithm for computing the EDM. Our parallel algorithm is based on the algorithm described in the previous section, so our algorithm consists of the same three steps.

The first step can be easily parallelized. We now present an overview of the second and third step.

In the step 2, we compute a function $h_i(x) = min_{1 \leq k \leq n} f_{i,k}(x)$, namely the lower envelope of $F_i = \{f_{i,k}(x) \mid 1 \leq k \leq n\}$, where $f_{i,k}(x) = \sqrt{(x-k)^2 + g_{ik}^2}$.

Remark the following expression:

$$\sqrt{(x-a_1)^2 + b_1} \leq \sqrt{(x-a_2)^2 + b_2}$$

$$\Leftrightarrow -2a_1x + a_1^2 + b_1 \leq -2a_2x + a_2^2 + b_2$$

From this expression, we obtain the following lemma.

Lemma 1 *Let $F = \{f_k \mid f_k = \sqrt{(x-a_k)^2 + b_k}\ ,\ 1 \leq k \leq n\}$, and let π be the transform that maps a function $f_k = \sqrt{(x-a_k)^2 + b_k}$ into the line $\pi(f_k)$ defined by $y = -2a_kx + a_k^2 + b_k$. Then a function f_k is on the lower envelope of F if and only if $\pi(f_k)$ is on the lower envelope of $\pi(F)$, where $\pi(F)$ is the line set $\{\pi(f_k) \mid 1 \leq k \leq n\}$.* □

Another key observation is the following transform.

Definition 1 (Dual transform[1]) *A dual transform γ is the transform that maps a point $p = (a, b)$ into the line $\gamma(p)$ defined by $y = ax+b$, and that maps the line $l : y = cx+d$ into the point $\gamma(l) = (-c, d)$.* □

Property 1 [1] *Let γ be the dual transform. A point p is below a line l if and only if the line $\gamma(p)$ is below the point $\gamma(l)$.* □

Using this property, we can reduce the problem of computing the lower envelope of lines to the problem of computing the convex hull.

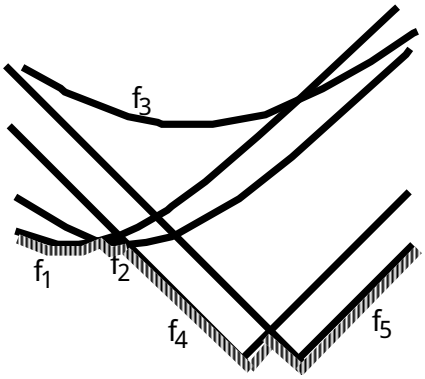
Lemma 2 [1] *Let $L = \{l_1, l_2, \ldots, l_n\}$ be a set of lines and γ be the dual transform. Then a line l_k is on the lower envelope of L if and only if $\gamma(l_k)$ is an extreme point on the lower hull of $\gamma(L)$, where $\gamma(L)$ is the point set $\{\gamma(l_k) \mid 1 \leq k \leq n\}$.* □

In consequence of Lemmas 1 and 2, we obtain the following lemma.

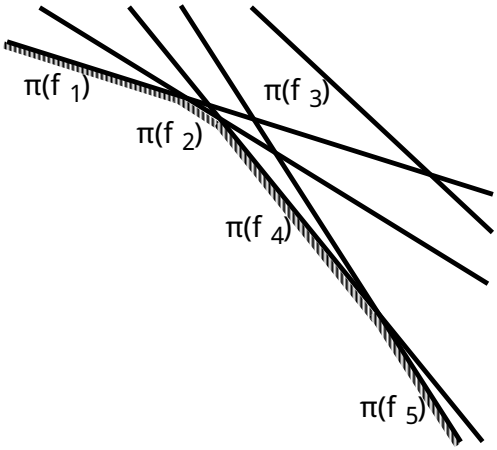
Lemma 3 *Let* $F = \{f_k \mid f_k = \sqrt{(x-a_k)^2 + b_k} \ \ , 1 \leq k \leq n\}$, *and* λ *be the transform that maps a function* $f_k = \sqrt{(x-a_k)^2 + b_k}$ *into the point* $\lambda(f_k) = (2a_k, a_k^2 + b_k)$[1]. *Then a function* f_k *is on the lower envelope of* F *if and only if* $\lambda(f_k)$ *is an extreme point on the lower hull of* $\lambda(F)$, *where* $\lambda(F)$ *is the point set* $\{\lambda(f_k) \mid 1 \leq k \leq n\}$. □

Fig.3 shows an example of these transforms. In this example, functions f_1, f_2, f_4, f_5 are on the lower envelope of $F = \{f_1, f_2, f_3, f_4, f_5\}$. Therefore, lines $\pi(f_1)$, $\pi(f_2)$, $\pi(f_4)$, $\pi(f_5)$ are on the lower envelope of $\pi(F)$. Also points $\lambda(f_1)$, $\lambda(f_2)$, $\lambda(f_4)$, $\lambda(f_5)$ are extreme points of the lower hull of $\lambda(F)$.

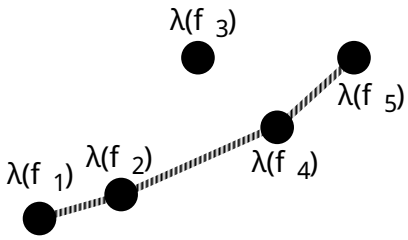
[1] $\lambda(f_k) = \gamma(\pi(f_k))$



(a)



(b)



(c)

Figure 3: An example of transforms: (a) A function set $F = \{f_1, f_2, f_3, f_4, f_5\}$ and its lower envelope. (b) The line set $\pi(F)$ and its lower envelope. (c) The point set $\lambda(F)$ and its lower hull.

In summary, the problem of finding the lower envelope of F_i is reduced to the problem of computing the lower hull of $\lambda(F_i)$. Since λ transforms a function $f_{i,k}(x) = \sqrt{(x-k)^2 + g_{ik}^2}$ to the point $(2k, k^2 + g_{ik}^2)$, the sequence of points $\lambda(f_{i,1}(x))$, $\lambda(f_{i,2}(x))$, $\ldots$, $\lambda(f_{i,n}(x))$ is sorted by x coordinate. The convex hull of n sorted points can be found in $O(\log n)$ time using $n/\log n$ processors on EREW PRAM[3] and in $O(\log\log n)$ time using $n/\log\log n$ processors on the common CRCW PRAM[5], respectively. Therefore the step 2 can be done efficiently.

In the step 3, we compute d_{ij} from the lower envelope of F_i. Let $LE = \{f_{i,j_1}(x), f_{i,j_2}(x), \ldots, f_{i,j_m}(x)\}$ be the set of the functions that are on the lower envelope, where $j_1 < j_2 < \ldots < j_m$. First, we compute intersection point (x_k, y_k) of $f_{i,j_k}(x)$ and $f_{i,j_{k+1}}(x)$ $(1 \leq k \leq m-1)$. Then $x_k < x_{k+1}$ and $h_i(x') = f_{i,j_{k+1}}(x')$ holds for any x' $(x_k \leq x' \leq x_{k+1})$. In other words, the Euclidean distances $d_{i\lceil x_k \rceil}$, $d_{i\lceil x_k \rceil + 1}$, $\ldots$, $d_{i\lfloor x_{k+1} \rfloor}$ can be computed from $f_{i,j_{k+1}}(x)$. Since $f_{i,j_{k+1}}(x) = \sqrt{(x-j_{k+1})^2 + g_{ij_{k+1}}^2}$, we broadcast j_{k+1} and $g_{ij_{k+1}}$ to the pixels $(i, \lceil x_k \rceil + 1), (i, \lceil x_k \rceil + 2), \ldots, (i, \lfloor x_{k+1} \rfloor)$ using prefix maxima operation and prefix sums operation, and compute the Euclidean distances for these pixels.

4.2 Details of the algorithm

In this subsection, we describe details of our parallel algorithm. In this algorithm, we use two operations, *prefix sums* and *prefix maxima*. The prefix sums of a sequence $(x_1, x_2, \ldots, x_n)$ is defined to be the sequence $(s_1, s_2, \ldots, s_n)$ such that $s_k = x_1 + x_2 + \ldots + x_k$. The prefix maxima of the same sequence is also defined to be the sequence $(m_1, m_2, \ldots, m_n)$ such that $m_k = max_{1 \leq l \leq k} x_l$.

The detail of the algorithm is described in the following. In the description, $f_{i,k}(x) = \sqrt{(x-k)^2 + g_{ik}^2}$, $F_i = \{f_{i,k}(x) \mid 1 \leq k \leq n\}$, and λ is the transform that maps a function $f_{i,k}(x)$ into the point $\lambda(f_{i,k}(x)) = (2k, k^2 + g_{ik}^2)$.

Parallel algorithm

step1: (Compute g_{ij} for every pixel (i,j))

For each column j $(1 \leq j \leq n)$, execute the following operations.

1. For each i $(1 \leq i \leq n)$, set $A_j[i] = i$ if pixel (i,j) is black, else set $A_j[i] = 0$. Compute the prefix maxima of A_j and store the result in array MA_j, i.e. set $MA_j[i] = max_{1 \leq k \leq i} A_j[k]$.

2. For each i $(1 \leq i \leq n)$, set $g_{ij} = +\infty$ if $MA_j[i] = 0$, else set $g_{ij} = i - MA_j[i]$.

step2: (Find the lower envelope of F_i)

For each row i $(1 \leq i \leq n)$, execute the following operations. (On the termination of step 2, the lower envelope of F_i is in the array LE_i.)

1. Compute the set of points $\{\lambda(f_{i,k}(x)) \mid 1 \leq k \leq n\}$, and find the convex hull of the point set.

2. For each j $(1 \leq j \leq n)$, set $B_i[j] = 1$ if $\lambda(f_{i,j}(x))$ is the extreme point on the lower hull, else set $B_i[j] = 0$.

3. Compute the prefix sums of B_i and store the result in SB_i, i.e. set $SB_i[j] = \sum_{k=1}^{j} B_i[k]$. For each j $(1 \leq j \leq n)$, set $LE_i[SB_i[j]] = j$ if $B_i[j] = 1$.

(Letting m_i be the number of functions on the lower envelope of F_i, $LE_i[k]$ $(1 \leq k \leq m_i)$ stores the index of the k^{th} function on the lower envelope. Let j_k be the index stored in $LE_i[k]$.)

step3: (Compute the Euclidean distance map)

For each row i $(1 \leq i \leq n)$, execute the following operations.

1. For each k $(1 \leq k \leq m_i - 1)$, compute the intersection point (x_k, y_k) of two functions $f_{i,j_k}(x)$ and $f_{i,j_{k+1}}(x)$.

2. For each j $(1 \leq j \leq n)$, set $C_i[j] = 0$. For each k $(1 \leq k \leq m_i - 1)$, if $\lceil x_k \rceil \neq \lceil x_{k+1} \rceil$ and $1 \leq x_k \leq n$, then set $C_i[\lceil x_k \rceil] = j_{k+1}$. If there exists $k'(1 \leq k' \leq m_i - 1)$ such that $x_{k'} < 1$ and $x_{k'+1} \geq 1$ then set $C_i[1] = j_{k'+1}$, else set $C_i[1] = j_1$. Compute the prefix maxima of C_i and store the result in MC_i, i.e. $MC_i[j] = max_{1 \leq k \leq j} C_i[k]$.

($MC_i[j]$ stores index j_k such that $f_{i,j_k}(j) = d_{ij}$.)

3. Set $MC_i[0] = 0$ and $g_{i\,0} = 0$. For each j $(1 \leq j \leq n)$, set $D_i[j] = g_{i\,MC_i[j]} - g_{i\,MC_i[j-1]}$ if $MC_i[j] \neq MC_i[j-1]$, else set $D_i[j] = 0$. Compute the prefix sums of D_i and store the result in SD_i, i.e. $SD_i[j] = \sum_{k=1}^{j} D_i[k]$.

($SD_i[j]$ stores $g_{i\,MC_i[j]}$.)

4. For each $j(1 \leq j \leq n)$, set $d_{ij} = \sqrt{(j - MC_i[j])^2 + SD_i[j]^2}$. □

4.3 Complexities on the EREW PRAM

In this subsection, we discuss complexities of the parallel algorithm on the EREW PRAM. As mentioned in the subsection 4.1, the extreme points of the lower hull of n sorted points can be found in $O(\log n)$ time using $n/\log n$ processors[3]. Both prefix sums and prefix maxima of n numbers can be computed in $O(\log n)$ time using $n/\log n$ processors[7]. The other operations in the algorithm can be performed with the same complexity obviously. Therefore, for each row or column, all steps can be performed in $O(\log n)$ time using $n/\log n$ processors. So the following theorem holds.

Theorem 1 *The Euclidean distance map of $n \times n$ binary image can be obtained in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM.* □

4.4 An implementation on the common CRCW PRAM

On the common CRCW PRAM, the extreme points of the lower hull of n sorted points can be found in $O(\log n/\log\log n)$ time using $n\log\log n/\log n$ processors[5]. Also the prefix sums of n numbers can be computed with the same complexities if each number is of $O(\log n)$ bits[4]. We adapt the prefix sums computations in our parallel algorithm, since every number is smaller than n in each prefix sums operation in our algorithm. Therefore, our parallel algorithm runs in $O(\log n/\log\log n)$ time using $n^2\log\log n/\log n$ processors on the common CRCW PRAM, if we can perform the prefix maxima operation with the same complexities. In what follows, we show that the prefix maxima "in our algorithm" can be computed with the complexities on the common CRCW PRAM.

In our parallel algorithm, prefix maxima operations are performed in steps 1 and 3. The sequences to which the prefix maxima operations are applied satisfy the following conditions.

(c1) Every element of the sequence is a non-negative integer of $O(\log n)$ bits.

(c2) Let the sequence be $S = (s_1, s_2, \ldots, s_n)$. If s_i and s_j $(i < j)$ are positive, then $s_i < s_j$ holds.

For the sequences satisfying the above conditions, the prefix maxima operation can be performed in $O(\log n/\log\log n)$ time using $n\log\log n/\log n$ processors on the common CRCW PRAM as follows. In the algorithm, we use the prefix sums computation presented in [4].

Computing the prefix maxima

Let $S = (s_1, s_2, \ldots, s_n)$ be a sequence satisfying the condition (c1) and (c2). The prefix maxima of S is denoted by $(max_1, max_2, \ldots, max_n)$.

1. For each k $(1 \leq k \leq n)$, set $A[k] = 1$ if $s_k \neq 0$, else set $A[k] = 0$. Compute the prefix sums of A and store the result in array SA, i.e. $SA[k] = \sum_{l=1}^{k} A[l]$.

2. For each k $(1 \leq k \leq n)$, set $B[SA[k]] = s_k$ if $s_k \neq 0$.

(By the operation, $B[i]$ stores the i^{th} positive element if exists.)

3. For each k $(1 \leq k \leq n)$, set $max_k = B[SA[k]]$ if $SA[k] \neq 0$, else $max_k = 0$. □

It is clear that the above prefix maxima operation can be performed in $O(\log n / \log\log n)$ time using $n \log\log n / \log n$ processors. Therefore, the following theorem holds.

Theorem 2 *The Euclidean distance map of $n \times n$ binary image can be obtained in $O(\log n / \log\log n)$ time using $n^2 \log\log n / \log n$ processors on the common CRCW PRAM.* □

5 The class of distances

Hirata and Kato pointed out in [6] that their algorithm can be applied to the computation of the distance map for distances other than the Euclidean distance with a little modification if the distance satisfies the following two conditions. (We denote the distance from $p_i = (x_i, y_i)$ to $p_j = (x_j, y_j)$ as $d(p_i, p_j)$.)

(1) Let $f(x, y)$ be a function such that $d(p_1, p_2) = f(|x_1 - x_2|, |y_1 - y_2|)$. Then $f(x, y)$ is a monotone increasing function for x and y, respectively. In other words, $f(x, y)$ satisfies the following two conditions.

$$\forall x_1, x_2, y \ \ [|x_1| < |x_2| \Rightarrow f(|x_1|, |y|) \leq f(|x_2|, |y|)]$$

$$\forall x, y_1, y_2 \ \ [|y_1| < |y_2| \Rightarrow f(|x|, |y_1|) \leq f(|x|, |y_2|)]$$

(2) If we assume that p_1, p_2, p_3 is on the same line, then $d(p_1, p_2) + d(p_2, p_3) = d(p_1, p_3)$.

Many distances (e.g. the Euclidean, cityblock, chessboard, octagonal and chamfer distances) satisfy these two conditions.

Since our algorithm is based on their algorithm, our algorithm can be also applicable to the same class of distance maps with a little modification.

6 Conclusions

In this paper, we presented an optimal parallel algorithm for computing the EDM of a $n \times n$ binary image. The algorithm computes the EDM in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM and in $O(\log n/\log\log n)$ time using $n^2 \log\log n/\log n$ processors on the common CRCW PRAM, respectively. The algorithm is applicable to a wide class of distance maps.

References

[1] T. Asano. *Computational Geometry.* Asakura-Syoten, 1990, in Japanese.

[2] L. Chen and H. Y. H. Chuang. A fast algorithm for Euclidean distance maps of a 2-D binary image. *Information Processing Letters*, 51:25–29, 1994.

[3] W. Chen. *Parallel Algorithm and Data Structures for Geometric Problems.* PhD thesis, Osaka University, 1993.

[4] R. Cole and U. Vishkin. Faster optimal parallel prefix sums and list ranking. *Information and computation*, 81:334–352, 1989.

[5] P. Fjällström, J. Katahainen, C. Levcopoulos, and O. Petersson. A sublogarithmic convex hull algorithm. *Bit*, 30:378–384, 1990.

[6] T. Hirata and T. Kato. An algorithm for Euclidean distance transformation. Technical Report AL41-4, IPSJ, 1994, in Japanese.

[7] J. JáJá. *An Introduction to Parallel Algorithms.* Addison-Wesley Publishing Company, 1992.

[8] M. M. Kulountzakis and K. N. Kutulakos. Fast computaiton of the Euclidian distance maps for binary images. *Information Processing Letters*, 43:181–184, 1992.

[9] D. W. Paglieroni. Distance transforms: Properties and machine vision applications. *CVGIP: Graphical Models and Image Processing*, 54:56–74, 1992.

[10] H. Yamada. Complete Euclidean distance transformation by parallel operation. In *Proc. 7th Conference on Pattern Recognition*, pages 69–71, 1984.