アルゴリズム 54－2
（1996．10．17）

2値画像の重みつき距離変換を行なう並列アルゴリズム

藤原 暁宏　井上 美智子　増澤 利光　藤原 秀雄

奈良先端科学技術大学院大学 情報科学研究科
〒630-01 奈良県生駒市高山町 8916-5

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

白黒2値画像の距離変換とは，各画素について最も近い黒画素までの距離を求める処理である．また，白黒2値画像の最近点変換とは，各画素について最も近い黒画素の座標を求める処理である．本稿では，重みつき距離を用いて，距離変換及び最近点変換を行なう並列アルゴリズムを示す．このアルゴリズムは $n \times n$ の入力画像に対して，EREW PRAM 上では $O(\log n)$ 時間，$n^2/\log n$ プロセッサで実行でき，common CRCW PRAM 上では，$O(\log \log n)$ 時間，$n^2/\log \log n$ プロセッサで実行できる．また，p^2プロセッサのメッシュモデル，ハイパーキューブモデル上では $O(n^2/p^2+n)$ 時間，$O(n^2/p^2+(n\log p)/p)$ 時間でそれぞれ実行できる．これらの結果より，このアルゴリズムは PRAM，メッシュモデル $(1 \leq p \leq \sqrt{n})$，および，ハイパーキューブモデル $(1 \leq p \leq n/\log n)$ 上でコスト最適である．

キーワード： 並列アルゴリズム，距離変換，最近点変換，PRAM，メッシュ，ハイパーキューブ

A parallel algorithm for weighted distance transforms of binary images

Akihiro Fujiwara, Michiko Inoue, 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

The distance transform of a black and white binary image is an operation which computes, for each pixel, the distance to the nearest black pixel. The nearest feature transform of the image is also an operation which computes, for each pixel, the coordinates of the nearest black pixel. This paper presents a parallel algorithm for the weighted distance transform and the weighted nearest feature transform of an $n \times n$ binary image. We show that the algorithm runs 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 runs 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 $1 \leq p \leq n$). The algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leq p \leq \sqrt{n}$) and on the hypercube (for $1 \leq p \leq n/\log n$).

Key words: Parallel algorithm, distance transform, nearest feature transform, PRAM, mesh, hypercube

1 Introduction

The distance transform of a black and white binary image is an operation which computes, for each pixel, the distance to the nearest black pixel. The distance transform is first introduced by Rosenfeld and Pfaltz[16], and is used in many image processing operations, such as nearest neighbor interpolation, morphological processing (skeletonization, thinning) and pattern matching[13].

These distance transforms are based on many kinds of distances functions because the distance functions have different efficiency or usefulness. The distance transform based on the Euclidean distance is preferred in various applications, but the Euclidean distance transform is a time-consuming operation. Consequently, many approximations to the Euclidean distance have been considered, such as city block, chessboard and chamfer distances.

For distance transforms of $n \times n$ binary images based on these distances, many $O(n^2)$ time sequential algorithms are proposed*. For example, Borgefors[2] proposed a city block distance transform algorithm. The algorithm consists of only two scans; one is downward scan from left to right and the other is upward scan from right to left. Hirata[8] proposed an algorithm which is applicable to a wide class of distance transforms. The class contains almost all kinds of distance transforms for image processing operations.

For parallel computations, some algorithms are proposed. Schwarzkof[17] proposed an $O(\log n)$ time algorithm for the city block distance transform and an $O(\log^2 n)$ time algorithm for the chessboard distance transform on the mesh of trees with n^2 processors. Lee and Horng [11] proposed a chessboard distance transform algorithm. The algorithm runs in $O(\log n)$ time using $O(n^2/\log n)$ processors on the EREW PRAM, in $O(\log n/\log\log n)$ time using $O(n^2 \log\log n/\log n)$ processors on the CRCW PRAM and in $O(\log n)$ time on an $O(n^2)$ processor hypercube.

Lee and Horng also showed in [12] that computation of the chessboard distance transform is interchangeable with computation of the medial axis transform. For the medial axis transform, Fujiwara et al. [7] proposed an efficient parallel algorithm. By using the algorithm, the chessboard distance transform can be computed in $O(\log n)$ time using $n^2/\log n$ processors on the EREW PRAM, in $O(\log\log n)$ time using $n^2/\log\log n$ processors on the common CRCW PRAM, 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 $1 \leq p \leq n$).

Some algorithms are also proposed for real machines. Paglieroni[14] proposed a unified distance transform algorithm and its architecture. Embrechts and Roose proposed two algorithms, one is for the city block distance transform[5] and the other is for the chamfer3-4 distance transform[6]. They implemented their algorithms on a parallel computer iPSC/2.

In this paper, we present a parallel algorithm for the weighted distance transform. The weighted distance is a generalization of above mentioned distances, such as city block, chessboard and chamfer distances, which are approximations to the Euclidean distance. Complexities of our algorithm are,

- $O(\log n)$ time, $n^2/\log n$ processors (EREW PRAM)
- $O(\log\log n)$ time, $n^2/\log\log n$ processors (common CRCW PRAM)
- $O(n^2/p^2 + n)$ time, p^2 processors (mesh)
- $O(n^2/p^2 + (n\log p)/p)$ time, p^2 processors (hypercube)

where $1 \leq p \leq n$.

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 a lower bound of sequential time for the problem. Finding a cost optimal parallel algorithm is an important objective in parallel computation research. Our algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leq p \leq \sqrt{n}$) and on the hypercube (for $1 \leq p \leq n/\log n$).

Our algorithm is also applicable to the nearest feature transform[13] with the same complexities. The nearest feature transform

*Recently, some $O(n^2)$ time sequential algorithms [3][4][8] are also proposed for the Euclidean distance transform.

is an operation which computes, for each pixel, the coordinates of the nearest black pixel. From the result of the nearest feature transform, we can easily solve the nearest neighbor problem and the Voronoi diagram problem in binary images. Our algorithm first computes the nearest feature transform of an input image, and then computes the distance transform. Note that most of known algorithms for computing distance transforms cannot apply to the nearest feature transform.

This paper is organized as follows. In Section 2, we give definitions and models of parallel computation. In Section 3, we describe our algorithm, and in Section 4, we discuss complexities of the algorithm for each model. We conclude this paper in Section 5.

2 Preliminaries

2.1 Weighted distance, distance transform and nearest feature transform

Given an $n \times n$ image I, let $I[i,j] \in \{0,1\}$ denote a value for a pixel (i,j) of I $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$, where i (resp. j) stands for the row (resp. column) index. We assume the pixel $(0,0)$ is on the top left corner of the image. For clarity of description, we call a pixel (i,j) a black pixel if $I[i,j] = 1$, otherwise we call the pixel a white pixel.

The *weighted distance* $d_w(p_1,p_2)$, between a pixel $p_1 = (i_1,j_1)$ and $p_2 = (i_2,j_2)$, is defined by the following expression,

$$\begin{aligned} d_w(p_1,p_2) &= w_0|i_1-i_2| + w_1|j_1-j_2| \\ &\qquad (\text{if } |i_1-i_2| \leq |j_1-j_2|) \\ &= w_1|i_1-i_2| + w_0|j_1-j_2| \\ &\qquad (\text{otherwise}) \end{aligned}$$

where w_0 and w_1 are nonnegative constants. The weighted distance can be considered as a generalization of many distance functions. For example, d_w is

- a city block distance (if $(w_0,w_1) = (1,1)$)
- a chessboard distance (if $(w_0,w_1) = (1,0)$)
- an optimal chamfer distance (if $(w_0,w_1) = (1, 1/\sqrt{2}-1+\sqrt{\sqrt{2}-1})$)
- a chamfer2-3 distance (if $(w_0,w_1) = (2,3)$)
- a chamfer3-4 distance (if $(w_0,w_1) = (3,4)$)
- a quasi-Euclidean distance (if $(w_0,w_1) = (1,\sqrt{2}-1)$)

and so on[13].

The *distance transform* of a black and white binary image is an operation which computes, for each pixel, the distance to the nearest black pixel. In other words, the distance transform computes an array $DT[i,j]$ given by

$$DT[i,j] = min\{d(p,p_B) | p_B \in Black\},$$

where $d(p,p_B)$ is the distance from the pixel $p = (i,j)$ to a pixel p_B and $Black$ is a set of all black pixels in an input image[†]. If $d = d_w$, we call the distance transform the *weighted distance transform.*

The *nearest feature transform* is an operation which computes, for each pixel, the coordinates of the nearest black pixel. In other words, the nearest feature transform computes an array $NFT[i,j]$ of coordinates given by

$$NFT[i,j] = (x,y) \quad \text{s.t.} \quad d(p,p_B) = DT[i,j],$$

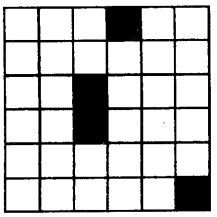
where $p = (i,j)$ and $p_B = (x,y) \in Black$.

From these definitions, the DT of an input image is uniquely defined, but the NFT may not be unique. It is clear that the DT can be derived from the NFT easily. Examples of the DT and the NFT are 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[9]. 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 concurrent access to a single memory cell. Concurrent accesses to the same cell for read or write instructions are allowed on the CRCW PRAM.

[†]We assume $Black \neq \phi$.



(a)

3	2	1	0	1	2
3	2	1	1	2	3
2	1	0	1	2	3
2	1	0	1	2	2
3	2	1	2	2	1
4	3	2	2	1	0

(b)

0,3	0,3	0,3	0,3	0,3	0,3
2,2	2,2	2,2	0,3	0,3	0,3
2,2	2,2	2,2	2,2	2,2	2,2
3,2	3,2	3,2	3,2	3,2	5,5
3,2	3,2	3,2	3,2	5,5	5,5
3,2	3,2	3,2	5,5	5,5	5,5

(c)

Figure 1: Examples of the DT and the NFT: (a) an input image I, (b) the DT of I and (c) the NFT of I. (The DT and the NFT are based on city block distance, i.e. $w_0 = w_1 = 1$.)

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.

The $p \times p$ (two-dimensional) mesh is a SIMD machine with p^2 processors arranged into $p \times p$ grid. Let $P_{x,y}$ $(0 \leq x \leq p-1, 0 \leq y \leq p-1)$ 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 $0 \leq z \leq p-1$. Then processor P_z is connected to processors $P_{z^{(k)}}$ $(0 \leq k \leq d-1)$, 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[15], 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 a unit time.

3 An algorithm for weighted distance transforms

3.1 Basic idea

First we introduce a basic idea of our algorithm.

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

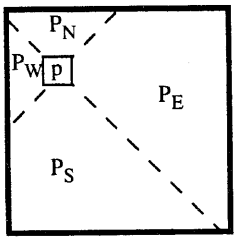


Figure 2: A pixel p and $P_N(i,j)$, $P_S(i,j)$, $P_E(i,j)$ and $P_W(i,j)$.

To compute the weighted distance transform and the nearest black pixel easily, we divide an input image into four set of pixels for each pixel (i,j), which are $P_N(i,j)$, $P_S(i,j)$, $P_E(i,j)$ and $P_W(i,j)$. These four sets are given by

$$\begin{aligned}
P_N(i,j) &= \{(i-g, j+h) | 0 \leq g \leq i, \\
&\quad \max\{-j,-g\} \leq h \leq \min\{n-1-j, g\}\} \\
P_S(i,j) &= \{(i+g, j+h) | 0 \leq g \leq n-1-i, \\
&\quad \max\{-j,-g\} \leq h \leq \min\{n-1-j, g\}\} \\
P_E(i,j) &= \{(i+g, j+h) | \max\{-i,-h\} \leq g \\
&\quad \leq \min\{n-1-i, h\}, 0 \leq h \leq n-1-j\}\} \\
P_W(i,j) &= \{(i+g, j-h) | \max\{-i,-h\} \leq g \\
&\quad \leq \min\{n-1-i, h\}, 0 \leq h \leq j\}\}
\end{aligned}$$

These sets are illustrated in Fig.2.

By using these four sets, we can compute the weighted distance transform in following six steps.

Algorithm for computing the weighted distance transform

(1) Find the nearest black pixel in P_N for each pixel.
(2) Find the nearest black pixel in P_S for each pixel.
(3) Find the nearest black pixel in P_E for each pixel.
(4) Find the nearest black pixel in P_W for each pixel.
(5) Compute the nearest feature transform, i.e. select the nearest black pixel among the above four pixels for each pixel (i,j) and store the coordinates to $NFT[i,j]$.
(6) Compute the weighted distance transform, i.e. compute weighted distance to $NFT[i,j]$ for each (i,j) and store the distance to $DT[i,j]$.

It is apparent that the first, second, third and fourth steps can be processed in a similar fashion. And in the fifth and sixth steps, NFT and DT can be easily computed in parallel: $NFT[i,j]$ and $DT[i,j]$ can be computed independently from the other elements of the arrays. Thus we describe how to find the nearest black pixel in P_E only, in the rest of this section.

We consider a pixel $p = (i,j)$ and $P_E(i,j)$. The weighted distance d_w from p to a pixel $(i_E, j_E) \in P_E(i,j)$ is given by

$$d_w = w_0|i_E - i| + w_1(j_E - j)$$

because $|i_E - i| \leq |j_E - j|$ and $j_E \geq j$ for any $(i_E, j_E) \in P_E(i,j)$.

From this property, we can obtain the following lemma.

Lemma 1 *Let $p = (i,j)$, p_1 and p_2 be three pixels such that $p_1, p_2 \in P_E(i,j)$ and $d_w(p,p_1) \leq d_w(p,p_2)$. Then $d_w(p',p_1) \leq d_w(p',p_2)$ holds for any pixel $p' = (i,k)$ $(0 \leq k \leq j)$.*

Proof
Let $p_1 = (i_1, j_1)$ and $p_2 = (i_2, j_2)$. From $d_w(p,p_1) \leq d_w(p,p_2)$,

$$\begin{aligned} & w_0|i_1 - i| + w_1(j_1 - j) \\ & \qquad \leq w_0|i_2 - i| + w_1(j_2 - j) \\ \Leftrightarrow\ & w_0|i_1 - i| + w_1(j_1 - k) + w_1(k - j) \\ & \qquad \leq w_0|i_2 - i| + w_1(j_2 - k) + w_1(k - j) \\ \Leftrightarrow\ & w_0|i_1 - i| + w_1(j_1 - k) \\ & \qquad \leq w_0|i_2 - i| + w_1(j_2 - k) \end{aligned}$$

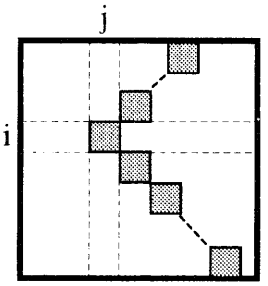


Figure 3: A pixel (i,j) and pixels in $DI(i,j)$

Since $p_1, p_2 \in P_E(i,k)$, $d_w(p',p_1) \leq d_w(p',p_2)$ holds. □

Let $NFT_E[i,j]$ denote the coordinates of the nearest black pixel in $P_E(i,j)$, $DI(i,j)$ denote the set of pixel $((P_E(i,j) - P_E(i,j+1))$ and $NFT_{ED}[i,j]$ denote coordinates of the nearest black pixel in $DI(i,j)$. ($DI(i,j)$ is illustrated in Fig.3.) The lemma 1 implies that $NFT_E[i,j]$ is equal to nearer one of $NFT_E[i,j+1]$ and $NFT_{ED}[i,j]$, i.e.

$$\begin{aligned} NFT_E[i,j] &= NFT_E[i,j+1] \\ (\text{If } d_w(p, NFT_E[i,j+1]) &\leq d_w(p, NFT_{ED}[i,j])) \\ &= NFT_{ED}[i,j] \quad \text{(otherwise)} \end{aligned}$$

(We assume $NFT_E[i,n] = (\infty,\infty)$, and also assume $NFT_{ED}[i,j] = (\infty,\infty)$ if no black pixel is in $DI(i,j)$.)

Consequently, once $NFT_{ED}[i,k]$ $(0 \leq k \leq n-1)$ are obtained, we can compute $NFT_E[i,k]$ $(0 \leq k \leq n-1)$ by a prefix minima operation[§] from right to left.

Now we describe how we compute $NFT_{ED}[i,j]$ for each (i,j) by showing the following lemma.

Lemma 2 *Let $p = (i,j)$, $p_1 = (i_1, j_1)$ and $p_2 = (i_2, j_2)$ be three pixels such that $p_1, p_2 \in DI(i,j)$ and $d(p,p_1) \leq d(p,p_2)$. Then $j_1 \leq j_2$ holds.*

Proof

[§]The *prefix minima* of a sequence $(x_0, x_1, \ldots, x_{n-1})$ is defined to be the sequence $(m_0, m_1, \ldots, m_{n-1})$ such that $m_k = min\{x_h \mid 0 \leq h \leq k\}$

Let $j_1 = j + k_1$ and $j_2 = j + k_2$. Then,

$$\begin{aligned} & d_w(p, p_1) \leq d_w(p, p_2) \\ \Leftrightarrow \quad & w_0 \cdot k_1 + w_1 \cdot k_1 \leq w_0 \cdot k_2 + w_1 \cdot k_2 \\ & \qquad (\text{Since } p_1, p_2 \in DI(i,j).) \\ \Leftrightarrow \quad & k_1(w_0 + w_1) \leq k_2(w_0 + w_1) \\ \Leftrightarrow \quad & k_1 \leq k_2 \end{aligned}$$

□

This lemma implies that the nearest black pixel to (i,j) has the smallest column index among all black pixels in $DI(i,j)$. By using the method described in the next subsection, we can find the pixel using two diagonal prefix minima operations.

As a consequence, we compute NFT_E as follows. First we compute the nearest black pixel in $DI(i,j)$ for each (i,j) by using two diagonal prefix minima operations. Second we compute NFT_E on each row by using horizontal prefix operations, which process from right to left.

3.2 The algorithm for computing the NFT_E

In this subsection, we describe details of our algorithm for computing the NFT_E.

The algorithm consists of three steps. In the first and second steps, we compute the nearest black pixel in $DI(i,j)$ for each pixel (i,j). In the first step, we compute the nearest black pixel, for each (i,j), in $DI_D(i,j) = \{(i+k, j+k) | (i+k, j+k) \in DI(i,j)\}$. From lemma 2, the pixel can be computed by prefix minima of column indeces on each diagonal sequence from lower right to upper left. We call this operation *UR-prefix*.

In the second step of the algorithm, we compute the nearest black pixel, for each (i,j), in $DI_U(i,j) = \{(i-k, j+k) | (i-k, j+k) \in DI(i,j)\}$ by computing prefix minima of indeces on each diagonal sequence from upper right to lower left, which we call *LL-prefix*. Since $DI_D(i,j) \cup DI_U(i,j) = DI(i,j)$, the nearest pixel to (i,j) in $DI(i,j)$ is one of the results of the first and the second step, which is the nearer one to (i,j).

In the third step, we compute the NFT_E. We assume that $NFT_{ED}[i,j]$ stores coordinates of the nearest black pixel to (i,j) in $DI(i,j)$. From lemma 1, $NFT_E[i,j]$ is equal to one of $NFT_{ED}[i,j]$, $NFT_{ED}[i,j+1]$, $\cdots$, $NFT_{ED}[i,n-1]$.

Let $p = (i,j)$, $NFT_{ED}[i,j] = (g,h)$, $NFT_{ED}[i,j_1] = (g_1,h_1)$ and $NFT_{ED}[i,j_2] = (g_2,h_2)$, and assume $j \leq j_1, j_2$ and $d_w(p, NFT_{ED}[i,j_1]) \leq d_w(p, NFT_{ED}[i,j_2])$. Letting function $f(i,j) = w_0|g-i| + w_1 \cdot h$, the followings hold.

$$\begin{aligned} & d_w(p, NFT_{ED}[i,j_1]) \leq d_w(p, NFT_{ED}[i,j_2]) \\ \Leftrightarrow \quad & w_0|g_1 - i| + w_1(h_1 - j) \\ & \qquad \leq w_0|g_2 - i| + w_1(h_2 - j) \\ \Leftrightarrow \quad & w_0|g_1 - i| + w_1 \cdot h_1 \leq w_0|g_2 - i| + w_1 \cdot h_2 \\ \Leftrightarrow \quad & f(i,j_1) \leq f(i,j_2) \end{aligned}$$

Since the function $f(i,j)$ is independent on j, we can compute the nearest pixels on row i by computing prefix minima of values of $f(i,k)$ $(0 \leq k \leq n-1)$. We call this prefix operation *L-prefix*.

The followings are details of the algorithm.

Algorithm for computing NFT_E

Step 1: (Computation of UL-prefix)

(1) For each pixel (i,j), set $A[i,j] = (i,j)$ if a pixel (i,j) is black, otherwise set $A[i,j] = (\infty, \infty)$.

(2) Compute the prefix minima of A for each diagonal sequence from lower right to upper left by comparing the column indeces, and store the result in ULP, i.e. set

$$\begin{aligned} ULP[i,j] &= A[i+m, j+m] = (i_m, j_m) \\ s.t. \quad j_m &= min\{j_k \mid A[i+k, j+k] = (i_k, j_k), \\ & \qquad 0 \leq k \leq min\{n-1-i, n-1-j\}\} \end{aligned}$$

for each (i,j) $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$.

Step 2: (Computation of LL-prefix and NFT_{ED})

(1) Compute the prefix minima of A for each diagonal sequence from upper right to lower left by comparing the column indeces, and store the result in LLP, i.e. set

$$\begin{aligned} LLP[i,j] &= A[i-m, j+m] = (i_m, j_m) \\ s.t. \quad j_m &= min\{j_k \mid A[i-k, j+k] = (i_k, j_k), \\ & \qquad 0 \leq k \leq min\{i, n-1-j\}\} \end{aligned}$$

for each (i,j) $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$.

(2) Set $NFT_{ED}[i,j]$ to the nearest one of the $ULP[i,j]$ and $LLP[i,j]$, i.e. set

$$NFT_{ED}[i,j] = ULP[i,j] \quad (\text{if } d_w((i,j),ULP[i,j]) \leq d_w((i,j),LLP[i,j]))$$
$$= LLP[i,j] \quad (\text{otherwise})$$

for each (i,j) $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$.

Step 3: (Computation of L–prefix and NFT_E)

(1) For each (i,j), set $B[i,j] = (w_0|g-i| + w_1 \cdot h,\ g,\ h)$ where $(g,h) = NFT_{ED}[i,j]$.

(2) Compute the prefix minima of B for each row from right to left by comparing the first indeces, and store the result in LP, i.e. set

$$LP[i,j] = B[i,j+m] = (f_m, g_m, h_m)$$
$$s.t.\ f_m = min\{f_k \mid B[i,j+k] = (f_k, g_k, h_k), 0 \leq k \leq n-1-j\}$$

for each (i,j) $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$.

(3) Set $NFT_E[i,j] = (b,c)$ such that $LP[i,j] = (a,b,c)$ for each (i,j) $(0 \leq i \leq n-1, 0 \leq j \leq n-1)$.

4 Complexities of the algorithm

By applying the algorithm which computes NFT_E to NFT_N, NFT_S and NFT_W, we can compute the weighted distance transform and the nearest feature transform. If we use the algorithm directly, eight diagonal prefix minima and four horizontal or vertical prefix minima operations are needed. However, by reusing the results of the diagonal prefix operations, we can compute the transforms using four diagonal prefix minima and four horizontal or vertical prefix minima operations.

For asymptotical complexity, it is apparent that the computation of NFT_E dominates the whole complexity. Thus we describe the complexity of the computation of NFT_E in the rest of this section.

4.1 Complexities on the PRAM

In the algorithm for computing the NFT_E, we mainly use three prefix minima operations, UL–prefix, LL–prefix and L–prefix. The other computations can be processed $O(1)$ time using n^2 processors on any PRAM obviously. Thus, the complexity of these prefix minima operations is asymptotically equal to the entire complexity.

It is well 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 process the prefix minima operations 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 these known algorithms to each diagonal sequence or each row, 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 weighted distance 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.* □

4.2 Complexities on the mesh and the hypercube

First we describe partition of an input image. For simplicity, we assume that the number of processors is p^2 and that $n = L \cdot p$ for some positive constant L. We also assume that $P_{x,y}$ $(0 \leq x \leq p-1, 0 \leq y \leq p-1)$ denotes each processor. 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.

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

In the similar manner as the PRAM, each processor can independently process all computations except for three prefix minima operations. Thus the complexity of the prefix

minima operations dominates the entire complexity.

In [7], Fujiwara et al. showed that these prefix minima operations can be processed 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, using basic divide and conquer method and row and column shifts. Thus, from the result, we can process the prefix minima operations in the same complexities.

In consequence, we obtain the following theorem.

Theorem 2 *The weighted distance 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 $1 \leq p \leq n$).* □

5 Conclusion

In this paper, we presented a parallel algorithm for computing the weighted distance transform. The algorithm can be also applied to the nearest feature transform. The algorithm runs 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 runs 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 $1 \leq p \leq n$). Consequently the algorithm is cost optimal on the PRAMs, on the mesh (for $1 \leq p \leq \sqrt{n}$) and on the hypercube (for $1 \leq p \leq n/\log n$).

References

[1] O. Berkman, B. Schieber, and U. Vishkin. Optimal doubly logarithmic parallel algorithms based on finding all nearest smaller values. *Journal of Algorithms*, 14:344–370, 1993.

[2] G. Borgefors. Distance transformations in arbitrary dimensions. *Computer Vision, Graphics and Image Processing*, 27:321–345, 1984.

[3] H. Breu, J. Gil, D. Kirkpatrick, and M. Werman. Linear time Euclidean distance transform algorithms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17:529–533, 1995.

[4] 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.

[5] H. Embrechts and D. Roose. MIMD divide–and–conquer algorithms for the distance transformation. Part I: City Block distance. *Parallel computing*, 21:1051–1076, 1995.

[6] H. Embrechts and D. Roose. MIMD divide–and–conquer algorithms for the distance transformation. Part II: Chamfer3–4 distance. *Parallel computing*, 21:1077–1096, 1995.

[7] A. Fujiwara, M. Inoue, T. Masuzawa, and H. Fujiwara. A simple parallel algorithm for the medial axis transform of binary images. In *Proc. IEEE Second International Conference on Algorithms and Architecture for Parallel Processing*, pages 1–8, 1996.

[8] T. Hirata. A unified linear-time algorithm for computing distance maps. *Information Processing Letters*, 58:129–133, 1996.

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

[10] R. E. Lander and M. J. Fisher. Parallel prefix computation. *Journal of ACM*, 27:831–838, 1980.

[11] Y.-H. Lee and S.-J. Horng. Fast parallel chessboard distance transform algorithms. In *Proc. 1996 International Conference on Parallel and Distributed Systems*, pages 488–493, 1992.

[12] Y.-H. Lee and S.-J. Horng. The chessboard distance transform and the medial axis transform are interchangeable. In *Proc. 10th International Parallel Processing Symposium*, pages 424–428, 1996.

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

[14] D. W. Paglieroni. A unified distance transform algorithm and architecture. *Machine Vision and Applications*, 5:47–55, 1992.

[15] S. Ranka and S. Sahni. *Hypercube algorithms with applications to image processing and pattern recognition*. Springer-Verlag, New York, 1990.

[16] A. Rosenfeld and J. L. Pfalz. Sequential operations in digital picture processing. *Journal of the ACM*, 13:471–494, 1966.

[17] O. Schwarzkopf. Parallel computation of distance transform. *Algorithmica*, 6:685–697, 1991.