Efficient Parallel Algorithms for Euclidean Distance Transform*

Ling Chen[1,2], Yi Pan [3], Yixin Chen [4] and Xiao-hua Xu[1]

[1] Department of Computer Science, Yangzhou University, Yangzhou 225009

[2] National Key Lab of Novel Software Tech, Nanjing Univ. Nanjing 210093

[3] Department of Computer Science, Georgia State University, Atlanta, GA 30303, USA.

[4] Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, IL 61801, USA

Abstract The Euclidean Distance Transform (EDT) converts a binary image into one where each pixel has a value equal to its distance to the nearest foreground pixel. Two parallel algorithms for EDT transform on linear array with reconfigurable pipeline bus system (LARPBS) are presented. For an image with $n \times n$ pixels, the first algorithm can complete EDT transform in $O(\frac{\log n \log\log n}{\log\log\log n})$ time using n^2 processors. The second algorithm can compute the EDT in O(log n loglog n) time using $\frac{n^2}{\log\log n}$ processors.

Keywords: distance transform, parallel algorithm, image processing

---

* This research was supported in part by Chinese National Science Foundation under contract 60074013, Chinese National Foundation of High Performance Computing under contract 00219 and Science Foundation of Jiangsu Educational Commission , China and the U.S. National Science Foundation under Grants CCR-9211621, OSR-9350540, CCR-9503882, and ECS-0196569.

1. Introduction

A two-dimensional binary image is a function from the elements of an $n \times n$ array of pixels to {0,1}. Pixels of unit (respectively, zero) value are referred to as foreground (respectively, background) pixels of the image. The Euclidean distance transform of an image produces a distance map of the same size where the value of each pixel stands for the Euclidean distance to its nearest foreground pixel. The distance transform is a useful tool in digital picture processing. It has found a wide range of applications in image analysis, pattern recognition and robotics. A brute-force exhaustive search for nearest foreground pixel in an $n \times n$ image is a procedure with the inherent complexity $O(n^4)$. In recent years, many parallel algorithms have been proposed for computing the EDT. Chen and Chuang's algorithm can be performed in $O(n^2/p+n\log p)$ time using p processors [1] or in $O(n.\log n)$ time using $\frac{n}{\log n}$ processors[2] on the EREW PRAM. On the other hand, Fujiwara et al [3] presented algorithms in which one takes $O(\log n)$ time using $\frac{n^2}{\log n}$ procssors on the PRAM EREW and the other takes $O(\log n \ / \log\log n)$ time using in PRAM CRCW with $n^2 \log\log n / \log n$ processors. Also in PRAM CRCW, Hayashi et al. [4] presented a parallel EDT algorithm which runs in O(loglog n) time using $O\left(\frac{n^2}{\log\log n}\right)$ processors. Lee et al [5] presented an $O(\log^2 n)$ time algorithm on PRAM EREW using n^2 processors. Pavel and Akl [6] presented an algorithm running in $O(\log n)$ time on a PRAM using n^2 processors. Kolountzakis et al. [7] presented an $O\left(\frac{n^2 \log n}{r}\right)$ time algorithm on PRAM EREW with r ($r \leq n$) processors. Pan et al [8] presented an $O(1)$ time algorithm on a reconfigurable mesh (REMESH) with $O(n^4)$ processors. Also in REMESH, Datta [9] presented a constant-time algorithm using $O(n^3)$ processors. Recently, EDT algorithms on linear array with reconfigurable pipelined optical bus (LARPBS) [10] have been proposed by researchers. Pan et al [11] presented an $O(\log n.\log\log n)$ algorithm on LARPBS with n^2 processors, and an $O(\log\log n)$ time algorithm on this model

with n^3 processors. Datta et al [12] also presented an algorithm on LARPBS running in $O(\log\log n)$ time using $O(n^{2+\varepsilon})$ processors, here ε is a constant satisfying $0<\varepsilon<1$. In this paper, we present two parallel algorithms for EDT transform on LARPBS. For an image with $n \times n$ pixels, the first algorithm can complete EDT transform in $O(\frac{\log n \log\log n}{\log\log\log n})$ time using n^2 processors. The second algorithm can compute the EDT in O(log n loglog n) time using $\frac{n^2}{\log\log n}$ processors. As far as we know they are the most efficient EDT algorithms on the LARPBS model.

The rest of this paper is organized as follows. Section 2 illustrates the architecture and the primitive operations of LARPBS. In Section3, we describe some basic properties of Euclidean distance transform which are very important to the algorithms we present. In Sections 4 and 5, two parallel EDT algorithms on LAPBS are presented and their complexities are also analyzed. Section 6 concludes the paper.

2. LINEAR ARRAYS WITH RECONFIGURABLE PIPELINED BUS SYSTEMS

Recently, arrays with reconfigurable optical bus systems [10] have been proposed and have drawn much attention from the researchers. In these systems, messages can be transmitted concurrently on a bus in a pipelined fashion and the bus can be reconfigured dynamically under program control to support different algorithmic requirements. LARPBS is one such model where any processor involvement is not allowed during a bus cycle, except setting switches up at the beginning of a bus cycle. Hence, it can exploit the high bandwidth of optical buses used to connect processors there. Many algorithms have been designed for basic data movement operations, sorting and selection, computational geometry, and PRAM simulation on the LARPBS model [10].

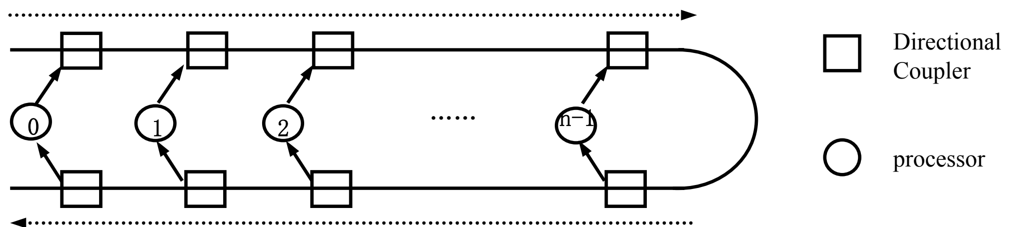

Fig.1 An linear array of n processors with an optical bus

A pipelined optical bus system uses optical waveguides instead of electrical signals to transfer messages among processors. Besides the high propagation speed of light, optical signal transmission on an optical bus has two other important characteristics： they are unidirectional propagation and predictable propagation delay. These advantages of using waveguides enable synchronized concurrent accesses of an optical bus in a pipelined fashion.

Fig 1 shows a linear array of n processors connected via an optical bus. Each processor is

connected to the bus with two directional couplers. Optical signals propagate unidirectionally from left to right on the upper segment and from right to left on the lower segment. An optical bus contains three identical waveguides, i.e., the massage waveguide for carrying data, the reference waveguide and the select waveguide for carrying address information, as shown in Fig. 2. To operate an optical bus in a pipelined fashion so that multiple data transfer can be performed in parallel, one unit delay ω (shown as a loop in Fig. 2) is added between two consecutive processors on the receiving segments of the reference waveguide.

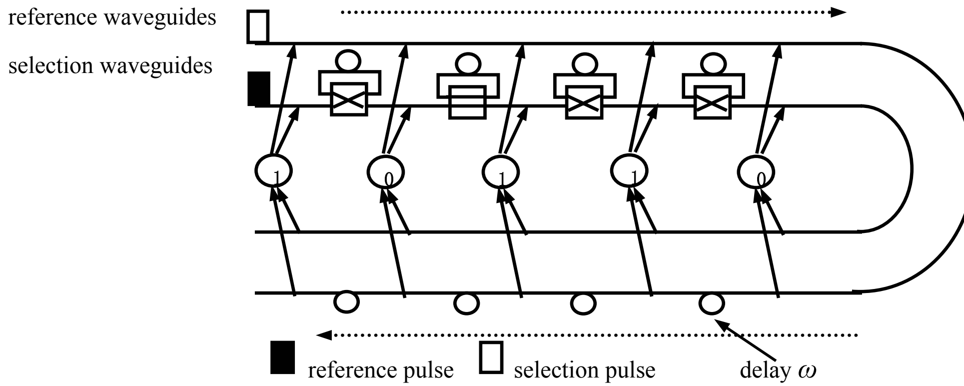

Fig. 2 An optical bus

The coincident pulse addressing technique [13, 14] can be applied on a optical bus system to implement the primitive operations of data transmission. Suppose that a source processor P_i wants to send a message to a destination processor P_j. Processor P_i first sends a reference waveguide at time t_{ref}, the beginning of a bus cycle, and a message frame at time t_{ref} on the message waveguide, which propagates synchronously with the reference pulse sent by P_i . Processor P_i also sends a select pulse at time $t_{sel}(j)$ on the select waveguide. Whenever processor P_j detects a coincidence of a reference pulse and a select pulse, it reads the message frame. Thus, in order for processor P_i to send a message to P_j , we need to have the two pulse coincide at P_j . This happens if and only if $t_{sel}(j)= t_{ref} +(n\text{-}j\text{-}1)\omega$, namely, the select pulse is delayed for $(n\text{-}j\text{-}1)\omega$ time relative to the reference pulse. Since the reference pulse goes through $n\text{-}j\text{-}1$ delays on the receiving segments, the two pulse will meet just at processor P_j .

Based on the coincident pulse addressing technique, the pipelined optical bus systems can support a massive volume of communications simultaneously and are particularly appropriate for applications that involve intensive communication operations such as broadcasting, one-to-one communication, multicasting, compression, split, and many irregular communication patterns. Here we describe implementation details of several primitive operations.

(1) One-to-one communication. One-to-one communication can be implemented in one bus cycle as follows. All the senders P_{ik} (k=1,2,…,r), which wants send a message to P_{jk} , send a reference pulse and a message frame at time t_{ref} and then send a select pulse at time $t_{sel}(j_k)$.

(2) Broadcasting Suppose the source processor P_i wants to broadcast a message to all the other processor in the array, P_i first sends a reference pulse at the beginning of its address frame. Then it sends n consecutive select pulses in its address frame on the select waveguide so that every processor on the bus can detect a double height pulse and thus reads the message.

(3) Multicasting. The implementation of this operation is similar to that of broadcasting. Suppose processor P_i (i=1,2,..g) wants to broadcast a message to $P_{ji,1}, P_{ji,2}$,…, $P_{ji,r}$. After sending a

reference pulse and a message frame at time t_{ref}, each processor P_i sends select pulses at times $t_{sel}(j_{i,1})$, $t_{sel}\,j_{i,2})$,…, $t_{sel}(j_{i,r})$.

The reader is referred to [10] for implementation details of other primitive operations on the LARPBS model. It has been shown[10] that by using the coincident pulse addressing technique, all the above primitive operations take O(1) bus cycles, where the bus cycle length is the end-to-end message transmission time over a bus [10]. To avoid controversy, let us emphasize that in this paper, by ``O(f(p)) time" we mean O(f(p)) bus cycles for communication plus O(f(p)) time for local computation. This approach of computation time measurement is commonly used in other computational models with optical connected bus systems such as AROB.

In addition to supporting fast communications, an optical bus itself can be used as a computing device for global aggregation. It was proven in [10] that by using n processors, the summation of n integers or real numbers with bounded magnitude and precision, the prefix sums of n binary values, the logical-or and logical-and of n Boolean values can be calculated in constant number of bus cycles. It was also shown in [10] that the minimum or maximum of n numbers can be found in O(loglog n) time on a LARPBS of size n.

In addition to the tremendous communication capabilities, a LARPBS can also be partitioned into several independent subarrays. The subarrays can operate as regular linear arrays with pipelined optical bus systems, and all subarrays can be used independently for different computations without interference. Hence, this architecture is very suitable for many divide-and-conquer problems. The basic communication, data movement, and aggregation operations provide an algorithmic view on parallel computing using optical buses, and also allow us to develop, specify, and analyze parallel algorithms by ignoring optical and engineering details. These powerful primitives that support massive parallel communications plus the reconfigurability of optical buses make the LARPBS computing model very attractive in solving problems that are both computation and communication intensive. Although LARPBS has great potential of computational ability, it is still a theoretical computational model at present. But unlike many other theoretical models, such like PRAM, the LARPBS model is implementable and practical using current optical technologies.

3. BASIC PROPERTIES OF EUCLIDEAN DISTANCE TRANSFORM

Let A=[$a(i,j)$] be an $n\times n$ binary image. We denote by $a(i,j)$ the pixel in the ith row and jth column. In the image the row indexes increase from the bottom to the top and the column indexes increase from left to right. The bottom-left pixel of the image is a(1,1). The Euclidean distance transformation of pixel a（i，j）is defined by $d_{ij} = \min_{(x,y)\in F} \sqrt{(i-x)^2 + (j-y)^2}$, where F is the set of foreground pixels. We denote the foreground pixel nearest to pixel $a(i,j)$ as $W(i,j)$. The distance from $a(i,j)$ to $W(i, j)$ is just d_{ij} which is the EDT at $a(i,j)$. We can shorten the search time for $W(i, j)$ by dividing the image into partitions and search the partitions in parallel. To divide each column of the image into partitions, some useful properties of EDT are used. These properties of EDT are described and proved in the lemmas below.

Lemma 1[7] Let A=（u，j） and B=（v，j）, be two pixels on the same column with A above B, i.e. $v<u$. Let $W(u, j) = (x, y)$ and $W(v, j) = (z, w)$, then $z\leq x$, namely, $W(u, j)$ is above

or on the same row as $W(v, j)$.

Proof: As shown in Fig.3, since $(y-j)^2+(x-u)^2 \leqslant (w-j)^2+(z-u)^2$ and $(w-j)^2+(z-v)^2 \leqslant (y-j)^2+(x-v)^2$, by adding these two inequalities we can get $(u-v)z \leqslant (u-v)x$. Since $v<u$, we can see $z \leqslant x$.

Q.E.D.

We can easily extent Lemma1 to more general cases as shown in Lemma 2 and Fig.4.

Lemma 2 Let $(i_1, j), (i_2, j) \ldots.. (i_r, j)$ be pixels on the same column and $i_1 < i_2 < \ldots. < i_r$. Suppose $W(i_k, j) = a(x_k, y_k)$, $(k=1, \ldots.., r)$, then $x_1 \leqslant x_2 \leqslant \ldots \leqslant x_r$.

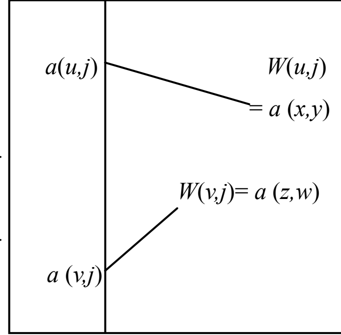

Fig. 3 An illustration of Lemma 1

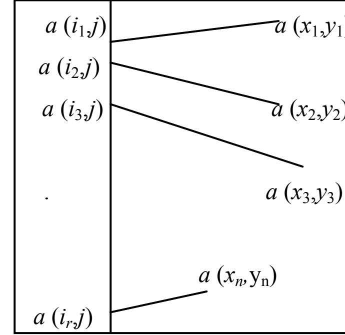

Fig. 4 An illustration of Lemma 2

If we denote the nearest foreground pixel in the ith row to pixel a(i, j) as $RW(i,j)$,we have the following lemma.

Lemma 3 If denote the distance from pixel a(i, j) to its nearest foreground pixel as d_{ij}, and denote the distance between a(i, j) and $RW(i, j)$ as $D[(i,j), RW(k,j)]$, then we have $d_{ij} = \min_{1\le k\le n} D[(i,j), RW(k,j)]$, namely, $W(i, j)$ can be selected from the set $\{RW(k, j) \mid 0<k \leqslant n\}$.

Proof: Suppose $W(i,j)$=a(x,y) and a$(x,y) \notin \{RW(k,j) \mid 0<k \leqslant n\}$. Let the nearest foreground pixel in the x-th row to pixel a(x, j) is a(x, z), i.e., $RW(x, j)$=a(x, z). Since a$(x,y) \notin RW(j)$, we have a$(x, y) \neq$a(x, z). From $RW(x, j)$=a(x, z), we get $|z-j| < |y-j|$, and hence $(i-x)^2+(j-y)^2 > (i-x)^2+(j-z)^2$. It is contradiction to the fact $W(i, j)$=a(x, y).

Q.E.D.

Based on Lemma 2 and Lemma 3 we can easily get the following lemma.

Lemma 4 Let a(i_1, j), a(i_2, j) ,…,a(i_r, j) are pixels in the ith column and $i_1 < i_2 < \ldots. < i_r$. Let $W(i_k, j) = RW(g_k, j)$, $(k=1, \ldots, r)$. Then we have (1) $g_1 \leqslant g_2 \leqslant \ldots. \leqslant g_r$ and (2) For every $i \in (i_{k-1}, i_k)$, $W(i, j)$ can be found in $RW(g_{k-1}, j)$, $RW(g_{k-1}+1, j)$, …..$RW(g_k, j)$.

Based on the above Lemmas, our algorithm consists of two phases: (1) For very column j $(j=1, \ldots..n)$, we first compute $RW(i, j)$ $(i=1, \ldots, n)$. (2) We choose some row indices i_1, i_2, ... i_r so that $1 \leqslant i_1 \leqslant i_2 \leqslant \ldots \leqslant. i_r \leqslant n$, and compute $W(i_k, j)$ $(k=1, \ldots..r)$ according to Lemma 3. From Lemma 4 we know that these pixels partition the jth column and their RW values into r parts. W and d

values of the pixels in each part can be selected in the corresponding part of RW values. Each part can be further partitioned in the same way. Since the processes of the parts are independent to each other, they can be carried out in parallel.

4. THE FIRST PARALLEL EDT ALGORITHM

4.1 The algorithm

To implement our algorithm, we set p, which is the number of processors in LARPBS, be equal to n^2 for an $n \times n$ image. The array is reconfigured into n subarrays each of which consists of n processors and will process one row or column of pixels. As stated above, our algorithm consists of two phases.

Phase 1: Compute $RW(i, j)$ for every pixel $a(i, j)$.

In this phase, we store the pixels on the ith row into the ith subarray, each pixel is stored in one processor. To compute the RW values for every pixel in the ith row is actually an 1-D EDT problem in the ith row. Suppose there are r foreground pixels in the ith row, which are $a(i,j_1)$, $a(i,j_2)$, …, $a(i,j_r)$, and the midpoints between foreground pixels $a(i,j_k)$ and $a(i,j_{k+1})$ are $m_k = \left\lceil \frac{j_k + j_{k+1}}{2} \right\rceil, (k = 1,....,r-1)$. Since for all pixels $a(i,j)$ with $j \in (m_k, m_{k+1}]$, their nearest foreground pixels in the ith row are all $a(i, j_{k+1})$, their RW values are also $a(i, j_{k+1})$. Therefore, we first compute the midpoints of each contiguous foreground pixels. Let $m_0 = 0$, $m_r = n-1$, then for all $a(i,j)$ with $j \in [m_k, m_{k+1})$, we have $RW(i, j) = a(i, j_{k+1})$ $k=1, 2…,r$.

To compute these midpoints m_k, we can first send the indices of the foreground pixels j_1, j_2, …. j_r to the r rightmost processors in the subarray by the LARPBS primitive operation of extraction and compression. Then these midpoints $m_k = \left\lceil \frac{j_k + j_{k+1}}{2} \right\rceil, (k = 1,....,r-1)$ can be obtained by data transforming in the neighbor processors. Then we can broadcast (i, j_{k+1}), for $k=1, 2…,r$, to the processors where $a(i, m_k)$, $a(i, m_k+1)$,…., $a(i, m_{k+1}-1)$ are stored. They are just the RW values of these pixels.

Phase 2: Compute $W(i, j)$ for every pixel $a(i, j)$. This phase consists of several steps as follows.

Step1 First the pixels $a(i, j)$ and their $RW(i, j)$ are rearranged in the array so that the pixels on the jth column and their RW values are stored into the jth subarray. After calculating the indices of the destination processor for each pixel, this data transformation can be carried out by LARPBS primitive operation of one-to-one communication.

Step2 Divide the n processors in the jth subarray, where the pixels on the jth column and their RW values are stored, into $\log\log n$ groups each of which consists of $\frac{n}{\log\log n}$ processors. First we select $\log\log n$ pivot pixels, $a(k*\frac{n}{\log\log n}, j), (k=0,1,..,\log\log n-1)$, to compute their W

and d values. For each pixel $a(k*\frac{n}{\log\log n}, j)$, we use $\frac{n}{\log\log n}$ processors to search for its W value in the $RW(i,j)(i=0,1,..,n-1)$ in the following way:

(2.1) Divide n RW values, $RW(i,j)(i=0,1,..,n-1)$, into $\frac{n}{\log\log n}$ groups. Each group consists of $\log\log n$ elements and is stored in one processor.

(2.2) Each processor computes the distances from $a(k*\frac{n}{\log\log n}, j)$ to the $\log\log n$ RW values and finds the minimum one which is called local minimum.

(2.3) Using $\frac{n}{\log\log n}$ processors to find the minimum from the $\frac{n}{\log\log n}$ local minima obtained in (2.2). This minimum one is just the W and d value of pixel $a(k*\frac{n}{\log\log n}, j)$.

Since the W and d value of pivot pixels $a(k*\frac{n}{\log\log n}, j), (k=0,1,..,\log\log n-1)$ are obtained, by Lemma 4 the pixels of the jth column and their RW values are partitioned into $\log\log n$ sections by these pivot pixels as shown in Fig.5 (a) .

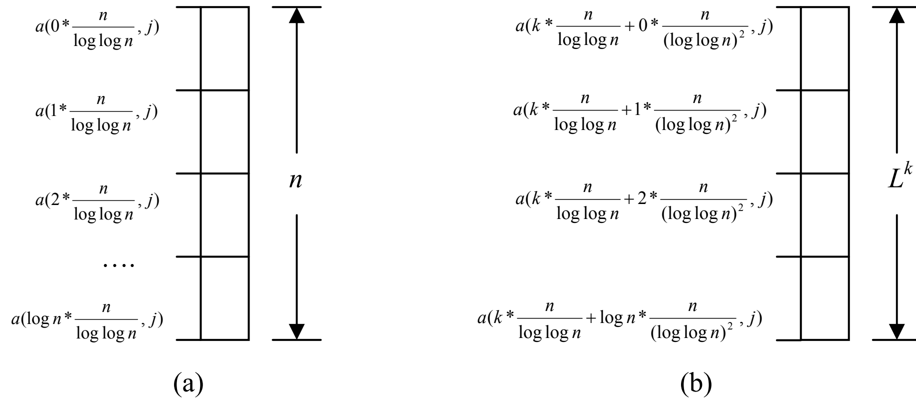

Fig. 5 (a) Partition of the jth column in step 2 of Phase 2
(b) Partition of the kth section of jth column in step 3 of Phase 2

Step 3 Suppose $x_{W(k*\frac{n}{\log\log n}, j)}$ is the row index of $W(k*\frac{n}{\log\log n}, j)$, which is the W

value of the pivot pixel $a(k*\frac{n}{\log\log n}, j)$ obtained in (2.3), then the number of the RW values corresponding to the pixels in the kth section is

$$L^k = \left| x_{W((k+1)*\frac{n}{\log\log n}, j)} - x_{W(k*\frac{n}{\log\log n}, j)} \right|.$$

It is obvious that $\sum_{k=0}^{\log\log -1} L^k \le n$. The jth subarray is partitioned into loglog n smaller subarrays. The kth such smaller subarray consists of L^k processors. The pixels of the kth section are assigned to the L^k processors of the kth smaller subarray.

(3.1) We further partition the L^k processors in the kth smaller subarray into $\log\log n$ groups each of which has $\frac{L^k}{\log\log n}$ processors. We also partition the $\frac{n}{\log\log n}$ pixels in the kth section into $\log\log n$ groups by the new pivot pixels $a(k*\frac{n}{\log\log n} + k_1 * \frac{n}{(\log\log n)^2}, j), (k, k_1 = 0,1,.., \log\log n - 1)$ as shown in Fig. 5(b). For each pixel $a(k*\frac{n}{\log\log n} + k_1 * \frac{n}{(\log\log n)^2}, j)$, we use $\frac{L^k}{\log\log n}$ processors to search for its W and d value in the L^k RW values of this section. We divide L^k RW values into $\frac{L^k}{\log\log n}$ groups. Each group consists of $\log\log n$ elements which are stored in one processor.

Each processor computes the distances from $a(k*\frac{n}{\log\log n} + k_1 * \frac{n}{(\log\log n)^2}, j)$ to the $\log\log n$ RW values and finds the local minimum.

(3.2) Using $\frac{L^k}{\log\log n}$ processors to find the minimum from the $\frac{L^k}{\log\log n}$ local minima obtained in (3.1). This minimum one is just the W and d value of pixel $a(k*\frac{n}{\log\log n} + k_1 * \frac{n}{(\log\log n)^2}, j)$.

Since the W and d value of pivot pixels

$a(k*\frac{n}{\log\log n}+k_1*\frac{n}{(\log\log n)^2},j),k,k_1=0,1,..,\log\log n-1$ are obtained, the pixels of the jth column and their RW values are partitioned into $(\log\log n)^2$ sections by these pivot pixels.

Step 4 We process each section obtained in Step 3 recursively in the similar way as in Step3. The algorithm can be completed after $O(\log_{\log\log n} n)=O(\frac{\log n}{\log\log\log n})$ times of recursions.

4.2 Complexity of the algorithm

First we analyze the number of processors required by the algorithm. It is obvious that Phase 1, Step1, and Step 2 of Phase 2 use exactly n^2 processors.

In Step3 of Phase 3, the jth subarray , which is assigned to the jth column of pixels, is partitioned into $\log\log n$ smaller subarrays. The kth such smaller subarray consists of L^k processors. Since L^0, L^1, …, $L^{\log\log n}$ are the lengths of the sections of jth column, we have $\sum_{k=0}^{\log\log-1} L^k \le n$. Therefore the total number of processors used for the jth column in this step is not greater than n, and the total number of processors required in this step is not greater than n^2.

Similarly, each recurrence in Step4 uses at most n^2 processors. Therefore, the algorithm uses n^2 processors.

Now we analyze the time complexity of the algorithm.

Phase 1 only consists of LARPBS primitive operations of broadcasting, extraction and compression which can be completed in constant time. Therefore, time complexity of this phase is O(1).

In Phase 2, Step1 uses the primitive operation of one to one communication which requires O(1) time.

In Step2, (2.1) also consists of primitive operations of one to one communication. Since every processor receives loglog n RW values, it requires O(loglog n) time. In (2.2) every processor computes the distances from a pixel to $\log\log n$ RW values and finds the minimum one. Obviously this can be computed in O(loglog n) time. (2.3) searches the minimum from the $\frac{n}{\log\log n}$ local minima using $\frac{n}{\log\log n}$ processors. This requires $O(\log\log\frac{n}{\log\log n})$ time by LARPBS primitive operation of finding the minimum. Therefore time complexity of Step2 is O(loglog n)+ $O(\log\log\frac{n}{\log\log n})$ =O(loglog n).

In (3.1) of Step3, every processor computes the distances from a pixel to $\log\log n$ RW

values and finds the minimum one. Obviously this can be computed in O(loglog n) time. (3．2) searches the minimum from the $\frac{L^k}{\log\log n}$ local minima using $\frac{L^k}{\log\log n}$ processors. This requires $O(\log\log\frac{L^k}{\log\log n}) \le O(\log\log n)$ time by LARPBS primitive operation of finding the minimum. Therefore time complexity of Step3 is $O(\log\log n) + \max\{O(\log\log\frac{L^k}{\log\log n})\}$ $= O(\log\log n)$.

Similar to the time complexity analysis of Step 3, it is easy to see that the time complexity of each recurrence in Step 4 is also $O(\log\log n)$. Since the algorithm can be completed after $O(\frac{\log n}{\log\log\log n})$ times of recursions, the time complexity of Step4 is $O(\frac{\log n}{\log\log\log n}) * O(\log\log n)$。

From the analysis above, we can see that for an $n \times n$ image, the total time complexity of the algorithm is $O(\frac{\log n}{\log\log\log n}) * O(\log\log n) = O(\frac{\log n \log\log n}{\log\log\log n})$.

5. THE SECOND PARALLEL EDT ALGORITHM

Our second algorithm can compute the EDT for an $n \times n$ image in O(log n loglog n) time on an LARPBS with $\frac{n^2}{\log\log n}$ processors.

5.1 The algorithm

Similar to the first algorithm mentioned in Section 4, the second algorithm first reconfigures the $\frac{n^2}{\log\log n}$ processors in the array into n subarrays each of which consists of $\frac{n}{\log\log n}$ processors and processes one row or column of pixels. This algorithm also consists of two phases.

Phase 1: Compute $RW(i, j)$ for every pixel $a(i, j)$.

In this phase, we store the pixels on the ith row into the ith subarray. Since each subarray has

$\frac{n}{\log\log n}$ processors, *w*e divide the pixels on the *i*th row into $\frac{n}{\log\log n}$ groups each of which has *q*=loglog *n* pixels and is stored into one processor. The *k*th group of pixels, which consists of pixels *a*(*i,kq*), *a*(*i,kq*+1),…..*a*(*i,*(*k*+1)*q*-1), are stored into the *k*th processor of the *i*th subarray.

Let the leftmost foreground pixel (namely the foreground pixel with minimum column index) in the *i*th row is *a*（*i*，*lz*），then there are *lz*-1 background pixels on the left of *a*（*i*，*lz*）. Their *RW* values are just *a*（*i*，*lz*）. Let the rightmost foreground pixel in the *i*th row is *a*（*i*，*rz*），then there are *n-rz* background pixels on the right of *a*（*i*，*rz*）. Their *RW* values are just *a*（*i*，*rz*）.

Now we compute the *RW* values of the pixels between *a*（*i*，*lz*）and *a*（*i*，*rz*）.

Let the leftmost foreground pixel in the *k*th group be a（*i*，*lz*(*k*)）. In this group there are *ld*(*k*)=*lz*(*k*)-*kq*+1 background pixels on the left of *a*（*i*，*lz*(*k*)）. Let the rightmost foreground pixel in the *k*th group be a（*i*，*rz*(*k*)）. In this group there are *rd*(*k*)=（*k*+1）*q*-*rz*(*k*) background pixels on the right of *a*（*i*，*rz*）.

The *k*th processor, which denoted as PE（*k*）, sends *lz*（*k*），*ld*（*k*）to PE（*k*-1）and sends *rz*（*k*），*rd*（*k*）to PE（*k*+1）. At the same time it receives *rz*（*k*-1），*rd*（*k*-1）from PE（*k*-1） and *lz*（*k*+1），*ld*（*k*+1）from PE（*k*+1）. If *ld*（*k*）< *rd*（*k*-1），then for the pixels in PE（*k*）with column indices less than *lz*（*k*）, their *RW* values are all （*i*，*lz*（*k*））. Otherwise the *RW* value of the $\left\lfloor \frac{lz(k)-rz(k-1)}{2} \right\rfloor$ leftmost pixels is （*i*，*rz*（*k*-1））, and the *RW* value of the other pixels in this part is all （*i*，*lz*（*k*））. The *RW* values of the pixels whose column indices are greater than *rz*（*k*）can be computed in a similar way.

Suppose there are *r* foreground pixels in the pixels stored in PE(*k*), their column indices are *lz*(*k*)=j_1， j_2， …. j_r=*rz*(*k*). To compute the *RW* values for the other pixels, we first compute the midpoints of each contiguous foreground pixels: $m_l = (j_l + j_{l+1})/2, (l = 1,....,r-1)$ 。Let m_0= *lz*(*k*)，m_r=*rz*(*k*)，then for all *a*(*i,j*) with $j \in [m_l, m_{l+1})$, we have *RW*（*i*，*j*）=（*i*，j_{l+1}）*l*=1, 2…,*r*.

Phase 2: Compute *W*（*i*，*j*）for every pixel *a*（*i*，*j*）.

Step 1 First the pixels *a*（*i*，*j*）and their *RW*（*i*，*j*） are rearranged in the array so that the pixels on the *j*th column and their *RW* values are stored into the *j*th subarray. After calculating the indices of the destination processor for each pixel, this data transformation can be carried out by an LARPBS primitive operation of one-to-one communication. Since each processor receives loglog *n* pixels and their *RW* values, this requires loglog *n* such one-to-one communications.

Steps 2-4 are much similar to Step 2-4 in Phase 2 of the first algorithm. It consists of several recurrence steps each of which divides the pixels of the *j*th column into several sections. The difference is that in the first algorithm mentioned above the *j*th column is divided into loglog*n* sections in each recurrence step, while in this algorithm it is divided into two sections. Obviously, the algorithm can be completed after O(log *n*) times of recursions.

5.2 Complexity of the algorithm

First we analyze the number of processors required by the algorithm. It is obvious that Phase 1 uses exactly $\frac{n^2}{\log\log n}$ processors.

Phase 2 is a recurrence procedure. The first recurrence computes the *W* and *d* value of the midpoint pixel $a\left(\left\lceil \frac{n}{2} \right\rceil, j\right)$ of the *j*th column (*j*=1， …， *n*). Since *RW* values of pixels in the *j*th column are stored in $\frac{n}{\log\log n}$ processors each of which is assigned loglog *n* *RW* values, each column requires $\frac{n}{\log\log n}$ processors. It can easily be seen that this step also requires a total of $\frac{n^2}{\log\log n}$ processors. Suppose $x_{W(\lceil \frac{n}{2} \rceil, j)}$ is the row index of $W(\left\lceil \frac{n}{2} \right\rceil, j)$, then the numbers of the *RW* values corresponding to the pixels in the two sections are $L^1 = x_{W(\lceil \frac{n}{2} \rceil, j)}$ and $L^2 = n - x_{W(\lceil \frac{n}{2} \rceil, j)}$ respectively. In the second recurrence, two smaller subarrays which have $\frac{L^1}{\log\log n}$ and $\frac{L^2}{\log\log n}$ processors are required to process the two sections of the *j*th column. Since $L^1 + L^2 \leqslant n$, every column requires at most $\frac{n}{\log\log n}$ processors, and in this step of recurrence, at most a total of $\frac{n^2}{\log\log n}$ processors are used. Similarly, we can see that every step in the latter recurrence requires at most $\frac{n^2}{\log\log n}$ processors. Therefore, the algorithm uses $\frac{n^2}{\log\log n}$ processors.

Now we analyze the time complexity of the algorithm.

Phase 1 computes $RW(i, j)$ for every pixel $a(i, j)$. Since each processor computes *RW* values for loglog *n* pixels, this phase requires O(loglog*n*) time.

In Phase 2, Step1 consists of primitive operations of one to one communication. Since every processor receives loglog *n* *RW* values, it requires O(loglog *n*) time. In the first recurrence of Step 2, every processor first computes the distances from pixel $a\left(\left\lceil \frac{n}{2} \right\rceil, j\right)$ to $\log\log n$ *RW* values and finds the minimum one called local minimum. Obviously this can be computed in O(loglog *n*) time.

Next, to search the minimum from the $\frac{n}{\log\log n}$ local minima using $\frac{n}{\log\log n}$ processors requires $O(\log\log\frac{n}{\log\log n})$ time by LARPBS primitive operation of finding the minimum. Therefore time complexity of the first recurrence is O(loglog n)+ $O(\log\log\frac{n}{\log\log n})$ = O(loglog n).

Similar to the time complexity analysis of the first recurrence, it is easy to see that the time complexity of each of the latter recurrence is also $O(\log\log n)$. Since the algorithm can be completed after O(log$n$) times of recursions, time complexity of Phase 2 is O(log$n$ loglog$n$).

From the analysis above, we can see that for an $n \times n$ image, the total time complexity of this algorithm is O(log$n$ loglog$n$). Even though the LARPBS can simulate an EREC PRAM in O(1) time, but the constant factor is usually large [15]. Hence, if we directly cite results on EREC PRAM model from [4] and use simulation results, the time complexity may be smaller, but will have a big constant factor there. We implement our algorithms directly on LARPBS to avoid the big constant factor and the extra factor is only O(log log n) which is very small for practical size of images. Hence, we believe that our 2nd algorithm is still valuable and comparable, in certain sense, to existing results on other PRAM models.

6. CONCLUSION

Two efficient parallel algorithms for Euclidean distance transform on linear array with reconfigurable pipeline bus system (LARPBS) are presented. For an image with $n \times n$ pixels, the first algorithm can complete EDT transform in $O(\frac{\log n \log\log n}{\log\log\log n})$ time using n^2 processors. The second algorithm can compute the EDT in O(log n loglog n) time using $\frac{n^2}{\log\log n}$ processors. Our algorithms take advantage of many important merits of LARPBS such as its high communication bandwidth, the versatile communication patterns it supports, and its ability of utilizing communication reconfigurability as an integral part of a parallel computation. It is shown that LARPBS is a powerful architecture for exploiting large degree of parallelism in a computational problem that most other machine models cannot achieve.

REFERENCES

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

[2] Chen L. (1995) An optimal algorithm for complete Euclidean distance transform. *Journal of Computer*, 18,

611-616.
［3］ Fujiwara, A. Masuzawa, T. and Fujiwara, H. (1995) An optimal parallel algorithm for the Euclidean distance maps of 2-D binary images. *Information Processing Letters*, 54, 277-282.
［4］ Hayashi, T. .Nakano, K and Olariu, S. (1998) Optimal parallel algorithm for finding proximate points with applications. *IEEE Transactions on Parallel and Distributed Systems*, 9, 1153-1166.
［5］ Lee, Y.H. Horng, S.J. and Kao, T.W. *et al* (1996) Parallel computation of exact Euclidean distance transform. *Parallel Computation*, 22, 311-325.
［6］ Pavel, S. and Akl, S.G. (1996) Efficient algorithms for Euclidean distance transform. *Parallel Processing Letters*, 5, 205-212.
［7］ Kolountzakis, M.N. and Kuatulakos, K.N. (1992) Fast computation of the Euclidean distance maps for the binary image. *Information Processing Letters,* 43, 181-184.
［8］ Pan, Y. and Li, K. (1999) Constant-time algorithm for computing the Euclidean distance maps of binary images on 2D meshes with reconfigurable buses. *Information Science*, 120, 209-221.
［9］ Datta, A. and Soundaralakshmi, S. (2001) Constant-time algorithm for the Euclidean distance transform on reconfigurable meshes. *Journal of Parallel and Distributed Computing*, 61, 1439-1455.
［10］ Li, K. and Zheng, S. Q. (1998) Parallel Computing Using Optical Interconnections, *Kluwer Academic Publishers,* Boston USA.
［11］ Pan, Y. Li, Y. and Li, J. *et al*(2000) Computing distance maps efficiently using an optical bus. Proceedings of 2000 Workshop on Parallel and Distributed Processing, Cancun, Mexico, 1-5 May, LNCS1800, pp178-185 Springer-Verlag, Heidelberg.
［12］ Datta, A. and Soundaralakshmi, S. (2001)Fast and scalable algorithm for the Euclidean distance transform on the LARPBS. Proceedings of 15th International Symposium on Parallel and Distributed Processing, San Francisco, California, USA, 23- 27 April, pp1393-1404, IEEE Computer Society Press, Los Alamitos, California.
［13］ Qiao, C. and Melhem, R .(1993) Time division optical communications in multiprocessor arrays. *IEEE Trans. Computers*, 42, 577-590.
[14] Levitan, S. and Melhem, R . (1990) Coincident pulse techniques for multiprocessor interconnection structures. *Applied Optics,* 29, 2024-2039.
[15] K. Li, Y. Pan, and S.-Q. Zheng, ``Efficient Deterministic and Probabilistic Simulations of PRAMs on Linear Arrays with Reconfigurable Pipelined Bus Systems," **The Journal of Supercomputing,** vol. 15, no. 2, pp. 163-181, February 2000.