Information Processing Letters 82 (2002) 73–81

Information Processing Letters

www.elsevier.com/locate/ipl

Parallel distance transforms on a linear array architecture

Tsorng-Lin Chia [a,*], Kuang-Bor Wang [b], Zen Chen [c], Der-Chyuan Lou [b]

[a] *Department of Information Management, Ming Chuan University, Taoyuan, Taiwan*
[b] *Department of Electrical Engineering, Chung Cheng Institute of Technology, Taoyuan, Taiwan*
[c] *Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan*

Received 30 November 2000; received in revised form 15 June 2001
Communicated by F.Y.L. Chin

Abstract

Distance transformation (DT) has been widely used for image matching and shape analysis. In this paper, a parallel algorithm for computing distance transformation is presented. First, it is shown that the algorithm has an execution time of $6N - 4$ cycles, for an $N \times N$ image using a parallel architecture that requires $\lceil N/2 \rceil$ parallel processors. By doing so, the real time requirement is fulfilled and its execution time is independent of the image contents. In addition, a partition method is developed to process an image when the parallel architecture has a fixed number of processing elements (PEs); say two or more. The total execution time for an $N \times N$ image by employing a fixed number of PEs is $2[N^2/M + 2(M - 1)]$, when M is the fixed number of PEs.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Distance transformation; Linear array; Parallel processing; Partition method

1. Introduction

The distance transformation (DT) is very useful in many applications such as thinning [1], medial axis transformation [2], convex hull extraction [3], robot path finding [4], skeletonization [5], Voronoi diagram extraction [3], planar tessellation [3], robot aerial image registration [4], and pattern matching [6,7]. DT is defined by a local operation based on a central pixel and the pixels in its neighborhood. The local operations are iterated until the distance values converge. Thus the operation produces a distance map for a given binary image containing object pixels and background pixels. For each object pixel in the image, its distance value is equal to the distance from it to the nearest background pixel.

There are three major types of distance transformation functions: octagonal [8], weighted (chamfer) [9], and Euclidean [10]. In general, they cannot be computed using similar algorithms. The Euclidean DT generates complete error-free Euclidean distance maps. But, computing the Euclidean distance is essentially a global operation that is very expensive and unsuitable for VLSI implement. Hence, it is only necessary to consider the local operation and give a reasonable approximation to the Euclidean distance.

An approximation method for computing the DT in only two passes over the image is the chamfer distance transformation. It has better accuracy than the others [9]. In this paper, we will propose a two-pass algorithm and design a hardware architecture for chamfer DT.

* Corresponding author.
E-mail address: tlchia@mcu.edu.tw (T.-L. Chia).

PII: S0020-0190(01)00250-2

Table 1
Comparison among several chamfer distance transformation algorithms

Methods	Author	Computation flow	#PEs	Time complexity	Sequential or parallel	Processing time
Type 1	Rosenfeld [11]	2-pass raster scan	1	$O(N^2)$	Sequential	Independent
	Verwer [12]	Bucket sort	1	$O(N^2)$	Sequential	Independent
	Rosenfeld [13]	2-pass raster scan	1	$O(N^2)$	Sequential	Independent
Type 2	Borgefors [14]	Pyramid	N^2	$O(N)$	Parallel	Dependent
	Piper [15]	MPP	N^2	Iterate until no change	Parallel	Dependent
	Paglieroni [16]	Row scan & col. scan	$2N$	$O(N)$	Parallel	Dependent
Type 3	Shih [17]	2-pass raster scan	$\lceil N/2\rceil$	$O(12N-4)$	Parallel	Independent
	Chen [18]	4-pass raster scan	N	$O(5N)$	Parallel	Independent
	Our method	2-pass raster scan	$\lceil N/2\rceil$	$O(6N-4)$	Parallel	Independent

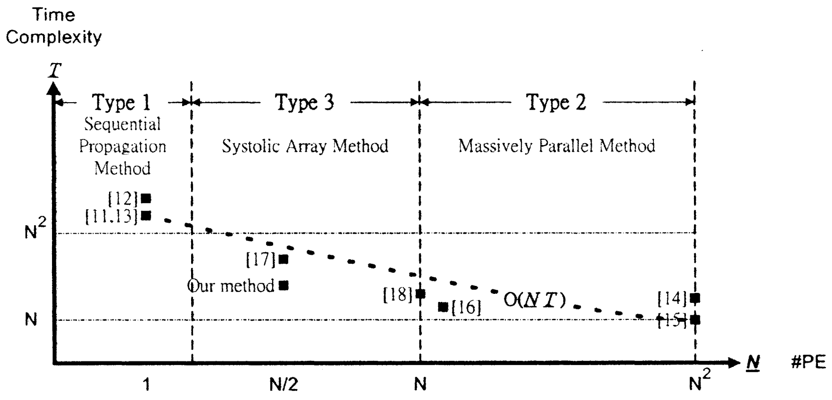


Fig. 1. The computational complexity diagram of the several chamfer DT algorithms.

As for chamfer DT computation, three major types of approach are available, refer to Table 1 and Fig. 1. Type 1 [11–13] is the sequential propagation algorithm. Although the sequential algorithm is more suitable for VLSI implementation than the parallel algorithm, the total execution time is $O(N^2)$ for an $N \times N$ image which does not meet the requirements for real-time applications. The processing time of the above methods is in-dependent on the image contents. Type 2 [14–16] is the massively parallel method, which can be extremely efficient if enough processor elements (PEs) are available. The time complexity can be reduced to $O(N)$. However, this method is dependent on the image contents.

From the viewpoint of real-time application, most of the existing sequential algorithms for computing the distance transformations do not meet the real-time requirement. Hence, the use of a parallel algorithm is indispensable. Although adopting the algorithm of massively parallel method, whose total execution time is $O(N)$, meets the requirements for real-time applications. But the massively parallel algorithm needs $O(N^2)$ processor elements, which is unsuitable for VLSI implementation, and the execution time is dependent on the image contents. This is a serious disadvantage for the real applications.

The third one, Type 3 method, is the systolic array method. To make the linear array suitable for VLSI implementation, the number of PEs must not be too large. This means that the number of PEs, denoted as M, must be fixed and is usually smaller than the image size $N \times N$ ($N \geqslant 512$) in practical applications. The minimum number of PEs is 2, and the reasonable number must be suitable for VLSI implementation. So that we have: $2 \leqslant M < N$. This method is also independent of the image contents. Recently, Shih, King and Pu [17] developed a systolic array using the sequential two-pass raster scan algorithm [13]. In their design, the total execution time for two scans of the whole image is $12N - 4$ cycles. Chen and Yang [18] introduced a systolic array using the four-pass algorithm. The total execution time is $5N$ using N processing elements. If the image is divided into m^2 subimages, whose sizes are $(N/m) \times (N/m)$, the total number of clock cycles required increases to $4mN + N/m$.

In this paper, a parallel algorithm for computing distance transformation is presented. First, it will be shown that the algorithm has an execution time of $6N - 4$ cycles for an $N \times N$ image using a architecture containing $\lceil N/2 \rceil$ parallel processors. In addition, the algorithm developed can work with various existing chamfer distance functions and its execution time is independent of the image contents; thus it is quite flexible. Second, we shall propose a partition method to process an image when the parallel architecture has a fixed number of PEs. The total execution time is better than given in [18] with the same number of PEs.

The organization of the paper is as follows: In Section 2, a parallel distance transformation algorithm based on two scanning passes is presented. The corresponding linear array architecture is proposed and the time complexity of the approach is analyzed in Section 3. In Section 4, a partition method is introduced to process a large size image when the architecture has a fixed number of PEs. Section 5 gives the conclusions.

2. Basic ideas and the parallel algorithm

Traditionally, the distance transformation can be completed in two scans of the image [13]. In the first row-by-row scan (from left to right and top to bottom) of the $N \times N$ image, the temporary distance value $d_1(P(i,j))$ for pixel $P(i,j)$ is computed based on $P(i,j)$ and the four adjacent pixels $P(i-1,j-1)$, $P(i-1,j)$, $P(i-1,j+1)$, and $P(i,j-1)$ (refer to Fig. 2). Similarly, the temporary distance value $d_2(P(i,j))$ is computed based on $P(i,j)$ and $P(i,j+1)$, $P(i+1,j-1)$, $P(i+1,j)$, and $P(i+1,j+1)$ during the second row-by-row scan (from right to left and bottom to top). The final distance value of $P(i,j)$ in the distance map is given by a minimum operation:

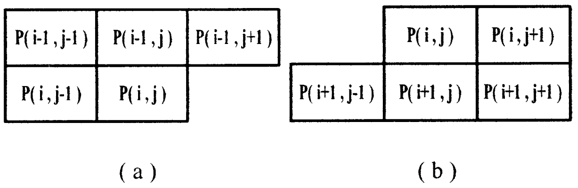


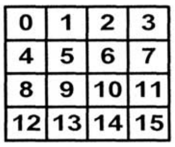
Fig. 2. The adjacent pixels of pixel $P(i,j)$: (a) in the forward pass, and (b) in the backward pass.

$$d\big(P(i,j)\big) = \mathrm{Min}\big(d_1\big(P(i,j)\big), d_2\big(P(i,j)\big)\big).$$

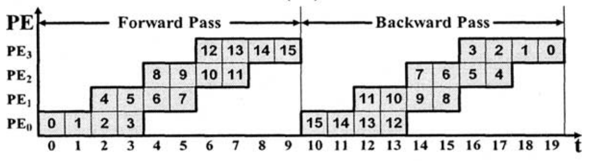
Based on the above concept, we shall propose a parallel distance transformation algorithm. In this algorithm, we use one processing element (PE) to do the pipeline operations on the pixels in each row. Thus, the required number of PEs is N. Later on, we shall propose a partition method using a fixed number of PEs that can be 2 or more. The algorithm is divided into two passes: forward and backward. In the forward pass all image rows are scanned and processed in parallel with each row in an assigned PE. Each row is scanned from left to right. However, the distance values of the preceding neighboring pixels in the row must be ready for use by the current pixel. Also, the input data flow of the preceding row must be two cycles ahead of the current row. In this scanning fashion, the $d_1(P(i,j))$ value for pixel $P(i,j)$ is computed in PE_i after it receives $d_1(P(i-1,j-1))$, $d_1(P(i-1,j))$, and $d_1(P(i-1,j+1))$ computed in PE_{i-1} during the last three cycles and $d_1(P(i,j-1))$ computed in PE_i during the last cycle. Similarly, in the backward pass $d_2(P(i,j))$ is computed in PE_{N-i-1}. Right after $d_2(P(i,j))$ is obtained, $d(P(i,j))$ is computed in PE_{N-i-1}. In Fig. 3, the time-space diagram for a 4×4 image is shown. The forward pass is executed at time instants 0 to 9; the backward pass is executed at time instants 10 to 19. Thus, the distance values of the required adjacent pixels are always ready before computing the distance value of the current pixel. The algorithm is given below.

```
Distance Transformation algorithm
Input: An $N \times N$ binary image $\{P(i,j)\}$.
  / $P(i,j) = 1$ is a background pixel and
    $P(i,j) = 0$ is an object pixel /
Output: An $N \times N$ distance map $D = \{d(P(i,j))\}$.
Procedure:
for $i = -1, 0, 1, \ldots, N,\ j = -1, 0, 1, \ldots, N$
  / Initialization /
  if ($i = -1$ or $j = -1$ or $i = N$ or $j = N$)
    then $d(P(i,j)) = \infty$   (a large number)
    elseif $P(i,j) = 1$
      then $d(P(i,j)) = 0$
      else $d(P(i,j)) = \infty$   (a large number)
for all $PE_i,\ i = 0, 1, \ldots, N-1,\ j = 0, 1, \ldots, N-1$
begin
Step 1: forward pass
$TD = \mathrm{Min}\big(d(P(i-1, j-1)) + d_{n0},$
          $d(P(i-1, j)) + d_{n1},$
          $d(P(i-1, j+1)) + d_{n2},$
          $d(P(i, j-1)) + d_{n3}\big),$
$d\big(P(i,j)\big) = \mathrm{Min}\big(TD, d(P(i,j))\big),$
output $d(P(i,j))$ to $PE_{i+1}$.
/ Wait until the forward pass of the $P(N-1, N-1)$
  finishes./
Step 2: backward pass
$TD = \mathrm{Min}\big(d(P(N-i, j+1)) + d_{n0},$
          $d(P(N-i, j)) + d_{n1},$
          $d(P(N-i, j-1)) + d_{n2},$
          $d(P(N-i-1, j+1)) + d_{n3}\big),$
$d\big(P(N-i-1, j)\big) = \mathrm{Min}\big(TD, d(P(N-i-1, j))\big),$
output $d(P(N-i-1, j))$ to $PE_{i+1}$.
end
```

The neighborhood distances d_{nk}, $k = 0, 1, 2, 3$, are defined according to the selected distance function, as shown in Fig. 4. For example, in the city block function, the values of the neighborhood distances d_{n0}, d_{n1}, d_{n2}, d_{n3} are 2, 1, 2, and 1, respectively; in the chamfer 3–4 distance function, the values of the neighborhood distances d_{n0}, d_{n1}, d_{n2}, d_{n3} are 4, 3, 4, and 3, respectively.

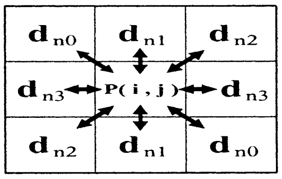


(a)

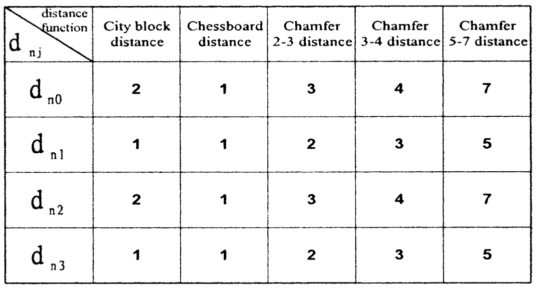


(b)

Fig. 3. (a) A 4×4 image. (b) The time-space diagram of the forward and backward passes; $0, 1, 2, \ldots, 15$ denote the pixel labels shown in (a).



(a)



d_{nj} \ distance function	City block distance	Chessboard distance	Chamfer 2-3 distance	Chamfer 3-4 distance	Chamfer 5-7 distance
d_{n0}	2	1	3	4	7
d_{n1}	1	1	2	3	5
d_{n2}	2	1	3	4	7
d_{n3}	1	1	2	3	5

(b)

Fig. 4. (a) The notations of the neighboring distance constants. (b) The values of the distance constants for the various distance functions.

3. The linear array architecture

For the above parallel distance transformation algorithm, it can be implemented by a linear array archi-

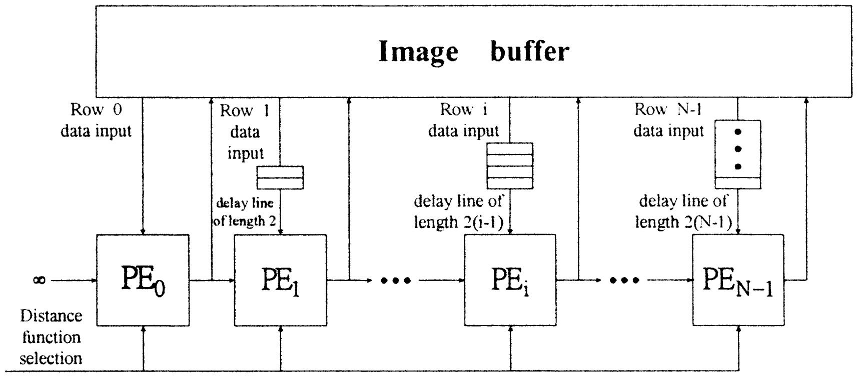


Fig. 5. The linear array architecture.

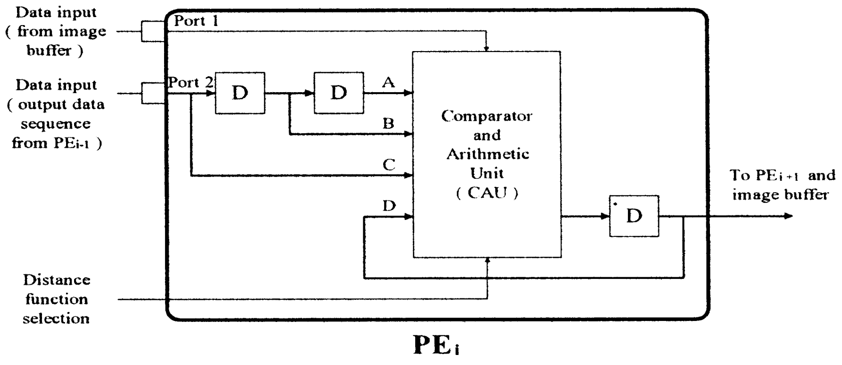

Data input (from image buffer)
Data input (output data sequence from PEi-1)
Distance function selection
Port 1
Port 2
D
D
A
B
C
D
Comparator and Arithmetic Unit (CAU)
D
To PEi+1 and image buffer
PEi

Fig. 6. The organization of a PE.

tecture to meet the real-time requirement. First, the linear array consisting of N processing elements is devised to process a binary image with $N \times N$ pixels, as shown in Fig. 5. The PEs are indexed from left to right, beginning with PE_0. Initially, the image buffer of the host computer stores the input binary image. As the process goes on, the buffer stores distance map or $d_1(P(i,j))$. In general, PE_i receives data from PE_{i-1} and computes $d_1(P(i,j))$ in row i in every cycle. Due to the fact that the data flow is shifted one pixel per cycle and that the distance values of four adjacent pixels must be completed beforehand, the input image in the image buffer is skewed so that the clock cycle of PE_i lags behind that of PE_{i-1} by two cycles. This is achieved by inputting the data in each row ($i = 0, 1, 2, \ldots, N-1$) through a delay line with a length of $2 \times i$. Hence, for the pixel $P(i,j)$ at time t, the distance values of $d_1(P(i-1, j-1))$, $d_1(P(i-1, j))$, and $d_1(P(i-1, j+1))$ are calculated and passed from PE_{i-1} to PE_i in the forward pass at time instants $t-3$, $t-2$, and $t-1$, respectively. The distance value of $d_1(P(i, j-1))$ is obtained and stored in PE_i at time $t-1$. Similarly, PE_{N-i-1} receives the distance values of $d_2(P(i+1, j+1))$, $d_2(P(i+1, j))$, $d_2(P(i+1, j-1))$, and $d_2(P(i, j+1))$ before it computes $d_2(P(i,j))$ and, then, $d(P(i,j))$ during the backward pass.

3.1. The processing element

Based on the parallel algorithm, the operations performed in PE_i include:

(1) Comparing the distance values of four adjacent pixels in the forward or backward pass.
(2) Outputting the distance value of $P(i,j)$ to PE_{i+1} and image buffer.

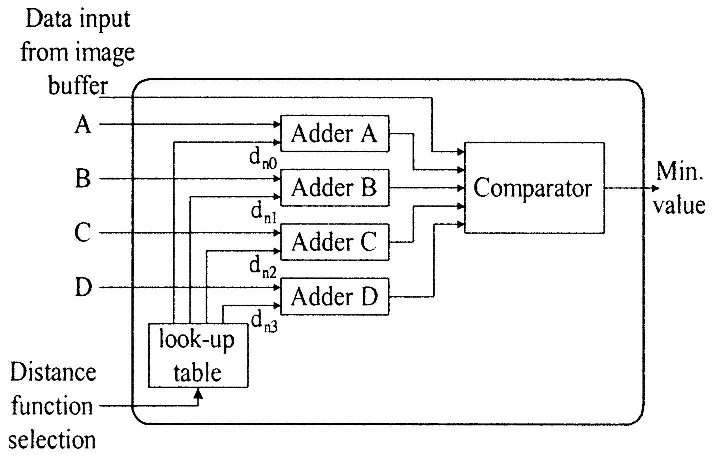


Fig. 7. The structure of a CAU unit.

After taking all the necessary operations into consideration, the structure of PE_i, shown in Fig. 6, is proposed. Two input ports are required to receive the distance values from PE_{i-1} and the image buffer of the host computer. Since the initial value of $d(P(i,j))$ in the image buffer is decided by the binary value of $P(i,j)$ in the initialization phase, the input data of port 1 is connected to the image buffer in the forward pass. The output value of $d_1(P(i,j))$ from PE_i will be sent back to image buffer. In the backward pass, the input data of PE_i needs $d_1(P(N-i-1,j))$, thus the input data of port 1 is also connected to the image buffer. For computing the distance value $d(P(i,j))$ the data path from PE_{i-1} is also needed so that the distance values of three adjacent pixels in row $i-1$ are obtained in sequence through port 2. These input distance values are input with two delay units. As a result, the needed input data from row $i-1$ are ready before starting calculating the distance value of $d(P(i,j))$ in the comparator and arithmetic unit (CAU). On the other hand, a feedback loop is connected from the output of CAU through a delay unit to the input of CAU. The path provides the distance value of $d(P(i,j-1))$ to CAU. Finally, the output is sent to PE_{i+1} and the image buffer.

3.2. The CAU

To find the minimum value of the four adjacent distance values and the value of $d(P(i,j))$ itself, a comparator is used in CAU, as shown in Fig. 7. The comparator determines the minimum of the distance values at the four adjacent pixels that depend on the distance function chosen, and the current distance value $d(P(i,j))$. The output of the comparator is defined as the distance value $d(P(i,j))$ which is also passed to PE_{i+1}.

The various distance functions can be selected by control signals. The distance values at adjacent pixels are stored in a look-up table, and are sent to the inputs of four adders. For example, in the chamfer 3–4 distance function, the distance value at $P(i-1,j-1)$, $P(i-1,j)$, $P(i-1,j+l)$, and $P(i,j-1)$ are 4, 3, 4, 3, respectively. So the input operand for adder are 4 for Adder A, 3 for B, 4 for C, and 3 for D.

Using the space-time diagram of the input image, the time complexity of the algorithm can be easily analyzed. If the input data rate is one pixel per cycle, the distance map will be obtained for either pass in $3N-2$ cycles where

$$\begin{aligned}3N-2 &= (\text{the length of the first row})\\ &\quad\times(\text{the execution cycle per pixel})\\ &\quad+(\text{the skewed delay per row})\\ &\quad\times(\text{the row number of image} - 1)\\ &= N\times 1+2\times(N-1).\end{aligned}$$

Hence the total execution time for both forward and backward passed is $6N-4$ cycles, so it is of order

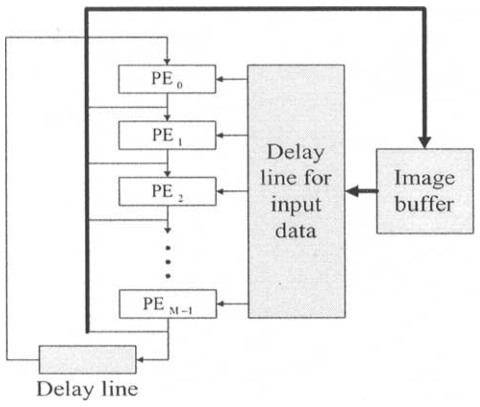


Fig. 8. The linear array architecture for executing the partitioned image.

$O(N)$. Hence it meets the requirements for real-time applications.

4. The partitioning method

To make the linear array suitable for VLSI implementation, the number of PEs must not be too large. This means that the number of PEs, denoted as M, must be fixed and is usually smaller than the image size N ($N \geqslant 512$) in practical applications. The minimum number of PEs is 2, and the reasonable number must be suitable for VLSI implementation. When $N > M$, the same parallel architecture presented previously will be used, but some extra hardware is needed to handle the image partition problem.

Assume an image whose size is $N \times N$, and a linear array that has M PEs. Let $N \geqslant M$. In the first place, the image must be divided into a number of subimages $m = \lceil N/M \rceil$ each with M or less rows. This row partitioning starts from top to bottom (refer to Fig. 9(a)), hence subimage S_j includes the part of image from row $M \times j$ to row $M \times (j+1) - 1$, where $j = 0, 1, \ldots, \lfloor N/M \rfloor - 1$ and S_{m-1} includes the part of image from row $M \times (m-1)$ to row $N-1$. These subimages are executed in an increasing order of the index j from 0 to $m-1$. The parallel architecture presented above will require an extra programmable delay line between the data output of PE_{M-1} and the input port 2 of PE_0 (refer to Fig. 8).

According to the values of N and M, we distinguish the following cases for the execution model:

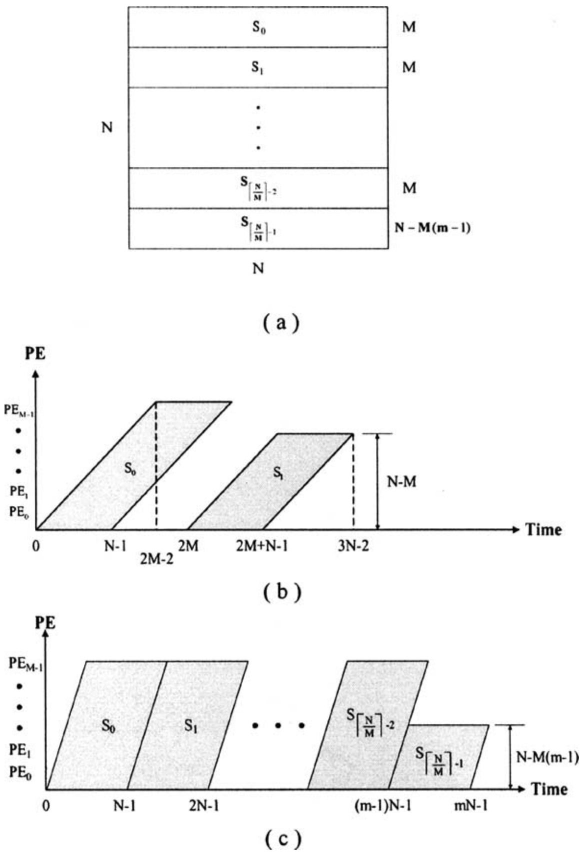


Fig. 9. The time-space diagram of the distance computation by our method for the case $N \geqslant M$: (a) an image is divided into subimages, (b) the diagram for $M \leqslant N \leqslant 2M$, (c) the diagram for $N \geqslant 2M$.

(1) *Case* 1: $M < N \leqslant 2M$.

In this case, the element PE_0 executes the first row of the subimage S_1, but it must wait until the corresponding output result of PE_{M-1} belonging to subimage S_0 has arrived. Therefore, the subimage S_1 must be delayed $2M - N$ clock cycles before it is input to the port 1 of PE_0. The programming delay line will be set to zero. The time-space diagram is shown in Fig. 9(b) and the total execution time is $2(3N-2)$. For comparison, the time-space diagram for the two-pass algorithm [17] is shown in Fig. 10(b), and the total execution time is $2[4N + 2(N-1)]$. Based on the analysis mentioned above, we compute the distance map of the image of size $N \times N$ using $\lceil N/2 \rceil$ PEs, and total execution time is the same if we use N PEs. In the case $N = 2M$, its speed is respectively about 2

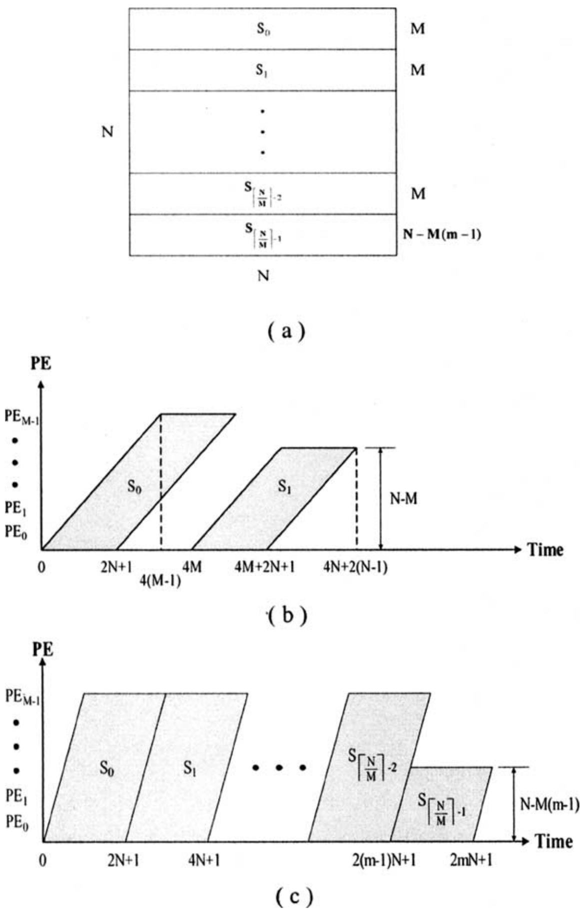


Fig. 10. The time-space diagram of the distance computation by the two-pass algorithm [17] for the case $N \geqslant M$: (a) an image is divided into subimages, (b) the diagram for $M \leqslant N \leqslant 2M$, (c) the diagram for $N \geqslant 2M$.

and 1.4 times faster than the two-pass and four-pass algorithms using the same number of PEs.

(2) *Case* 2: $N \geqslant 2M$.

Since the output results of the last row of S_j produced from PE_{M-1} are $N-2M$ clock cycles ahead of the input data of the first row of S_{j+1} to PE_0, the output of PE_{M-1} must be input through an $N-2M$ delay line. Thus, the side effect between S_j and S_{j+1}, can be reduced through the timing delay. The time-space diagram for this case is shown in Fig. 9(c) and the total execution time is $2\{Nm+2[N-M(m-1)-1]\}$. The time-space diagram of the two-pass algorithm is given in Fig. 10(c) and the total execution time is $2\{(2m+4)N-(m-1)4M-2\}$. The

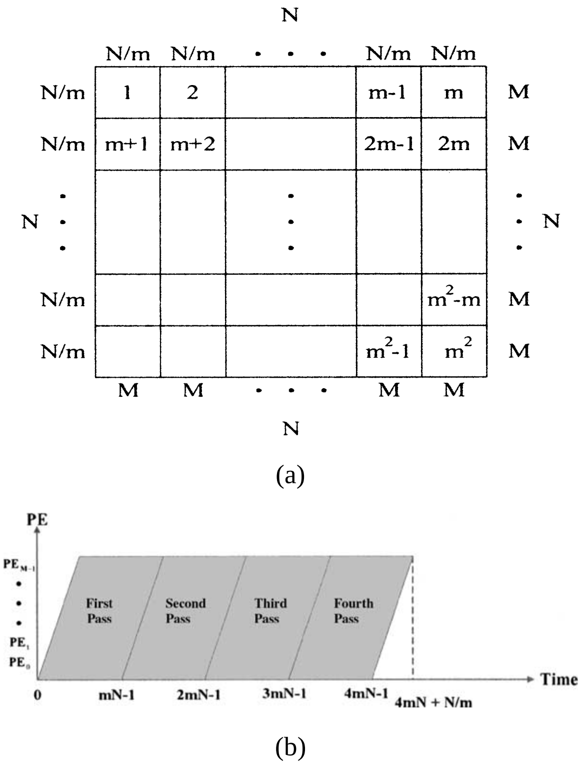


Fig. 11. The time-space diagram of the distance computation by the four-pass algorithm [18] for the case $N \geqslant M$: (a) an image is divided into subimages, (b) the diagram for $N \geqslant M$.

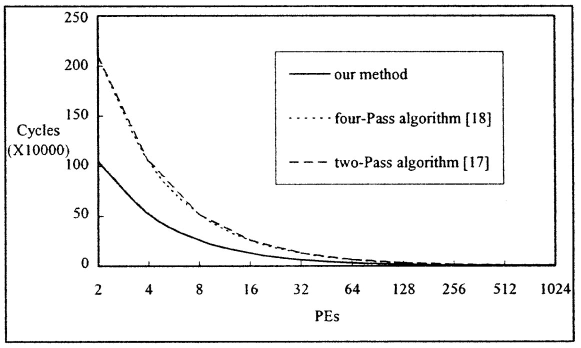


Fig. 12. Time comparison between two-pass algorithm, four-pass algorithm and our method.

time-space diagram of the four-pass algorithm [18] is given in Fig. 11(b), and the total execution time is $4mN+N/m$.

Consider that an image has a size of 1024×1024 and a linear array has M PEs with $M \leqslant 1024$. When we compare the two-pass algorithm, the four-pass algorithm and our method, all using the same number of PEs, the number of clock cycles required are different, as shown in Fig. 12. If $N \gg 2M$, our speed is nearly 2 times faster than the other two methods.

5. Conclusions

In this paper, a parallel algorithm and its hardware architecture for computing the distance transformation have been described. With our proposed architecture, the distance map of an $N \times N$ binary image can be obtained within $6N - 4$ cycles using an architecture of $\lceil N/2 \rceil$ PEs. Other advantages of our architecture include modularity, expandability, regularity of data flow, and hardware simplicity. These properties are highly desirable for VLSI implementations. We also consider the partition problem when the image size is larger than the size of the PE array. In the future, the distance transformation subject to some imposed constraints will be studied with a modified version of the current architecture.

References

[1] S. Suzuki, K. Ade, Sequential thinning of object pictures using distance transformations, in: Proc. 8th Internat. Joint Conf. on Pattern Recognition, Paris, 1986, pp. 56–74.

[2] H. Blum, A transformation for extracting new descriptors of shape, in: Proc. Symp. on Models for Perception of Speech, and Visual Form, 1967, pp. 362–390.

[3] C. Arcelli, G. Sanniti di Baja, Computing Voronoi diagrams in digital pictures, Pattern Recogn. Lett. 4 (1986) 383–389.

[4] P.W. Verbeek, L. Dorst, B.J.H. Verwer, F.C.A. Groen, Collision avoidance and path finding through constrained distance transformation in robot state space, in: Proc. Internat. Conf. on Intelligent Autonomous Systems, 1986, pp. 634–671.

[5] C. Wayne, B. Philip, Generation skeletons and centerlines from the distance transform, CVGIP: Graphic Model and Image Processing 54 (5) (1992) 420–437.

[6] H.C. Liu, M.D. Srinath, Partial shape classification using contour matching in distance transformation, IEEE Trans. Pattern Anal. Machine Intell. 12 (11) (1990) 1072–1079.

[7] R.L. Brown, The fringe distance measure: An easily calculated image distance measure with recognition results comparable to Gaussian blurring, IEEE Trans. Syst. Man Cybern. 24 (1) (1994) 111–115.

[8] P.P. Das, Best simple octagonal distances in digital geometry, J. Approximation Theory 68 (1992) 155–174.

[9] G. Borgefors, Distance transformations in digital images, Comput. Vision Graphics Image Process. 34 (1986) 344–371.

[10] I. Ragnemalm, The Euclidean distance transform in arbitrary dimensions, Pattern Recogn. Lett. 14 (1993) 883–888.

[11] A. Rosenfeld, J.L. Pfalz, Distance functions on digital pictures, Pattern Recogn. 1 (1968) 33–61.

[12] B.J.H. Verwer, P.W. Verbeek, S.T. Dekker, An efficient uniform cost algorithm applied to distance transforms, IEEE Trans. Pattern Anal. Machine Intell. 11 (4) (1989) 425–429.

[13] A. Rosenfeld, A.C. Kak, Digital Picture Processing, Vol. 2, 2nd edn., Academic Press, New York, 1982.

[14] G. Borgefors, T. Hartmann, S.L. Tanimoto, Parallel distance transforms on pyramid machines: Theory and implementation, Signal Process. 21 (1) (1990) 61–86.

[15] J. Piper, E. Granum, Computing distance transformations in convex and non-convex domains, Pattern Recogn. 20 (6) (1987) 599–615.

[16] D.W. Paglieroni, A unified distance transformation algorithm and architecture, Machine Vision Appl. 5 (1992) 47–55.

[17] F.Y. Shih, C.T. King, C.C. Pu, Pipeline architectures for recursive morphological operations, IEEE Trans. Image Process. 4 (1) (1993) 11–18.

[18] C.H. Chen, D.L. Yang, Fast algorithm and its systolic realization for distance transformation, IEE Proc. Comput. Digit. Tech. 143 (3) (1996) 168–173.