1998 International Computer Symposium
Workshop on Image Processing and Character Recognition
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

PARALLEL EXECUTION OF DISTANCE TRANSFORMATION OPERATION ON A LINEAR ARRAY ARCHITECTURE

Kuang-Bor Wang, Mei-Jiun Ke**, Tsorng-Lin Chia*, and Zen Chen****

*Department of Electrical Engineering,
Chung Cheng Institute of Technology, Taoyuan, Taiwan, R.O.C.
**Taiwan Semiconductor Manufacturing Company, Hsinchu, Taiwan, R.O.C.
*** Institute of Computer Science and Information Engineering,
National Chiao Tung University, Hsinchu, Taiwan, R.O.C.

ABSTRACT

DT (Distance Transformation) has been widely used for image matching and shape analysis. In this paper, a parallel algorithm for computing distance transformation is presented. It is shown that the algorithm has an execution time of 6N-4 cycles, for an N×N image using a parallel architecture which requires $\left\lceil \frac{N}{2} \right\rceil$ parallel processors. In addition, the algorithm proposed can be used with various distance functions and its execution time is independent of the image content. Furthermore, we shall propose a partition method to process an image when the parallel architecture has a less number of PEs (Processing Elements). However, the total execution time is better than the four-pass algorithm with the same number of PEs.

I. INTRODUCTION

The DT (Distance Transformation) 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 the 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 a nearest background pixel.

The distance value depends on the type of the distance function used. There are three major types of distance functions : octagonal [8], weighted (chamfer) [9], and Euclidean [10]. Four main approaches are available for computation of DTs. The first commonly used approach is the 2-pass sequential raster scan method using chamfer distances [11]. This method is not dependent on image content. The second approach is a pipelining version of the 2-pass method [10]. The third approach is the "bucket-sort" method, where the pixels in the distance propagation edge is stored treated in order of size. Finally, the fourth approach is the massively parallel method [12], which can be extremely efficient if enough PEs (Processing Elements) are available [13].

The distance transformation operation can be implemented in either a sequential or a parallel manner. 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. Yamada [14], Borgrgors [15], Zhao and Daut [16], and Piper and Granum [17] developed a series of parallel algorithms based on the wave propagation scheme. Their algorithms perform the identical computation at all pixels simultaneously and iterate until no change in the distance values occurs. But, these algorithms (also called iteratively parallel local algorithm) require an architecture that allocates one processor to each pixel and end up with a total of N^2 processor elements. Besides, the number of iterations required is determined by the largest distance transformation value that can be obtained in the given image. Paglieroni [18] introduced a radically different algorithm that contains two scan operations: one parallel row scanning and one parallel column scanning. Typically, this type of parallel algorithm has a time complexity of order O(N) for an image of size $N \times N$, but the execution time is dependent on the image content. This is a serious disadvantage for the real applications.

On the other hand, the image size, $N \times N$, becomes very large if $N > 128$, so most parallel architectures for computing distance transformation need a huge number of PEs. Recently, Shih, King, and Pu [19] developed a systolic array using the sequential two-pass raster scan algorithm [20]. In their designs, the total execution time for two scans of the whole image is 12N-4 cycles. Chen and Yang [21] introduced a systolic array using the four-pass algorithm. The total execution time is 5N with N being the number of the PEs. If image is divided into m^2 subimages, whose size are $\frac{N}{m} \times \frac{N}{m}$, the total number of clocks increase to $4mN + \frac{N}{m}$.

In this paper, a parallel algorithm for computing distance transformation is presented. It will be shown that this algorithm has an execution time of 6N-4 cycles for an $N \times N$ image using a architecture containing $\left\lceil \frac{N}{2} \right\rceil$ parallel processors. In addition, the algorithm developed can work with various existing distance functions and its execution time is independent of the image content; thus it is quite flexible. Furthermore, we shall propose a partition method to process an image when the parallel architecture has a less number of processing elements (PEs). The total execution time is better than [21] with the same number of PEs.

The organization of the paper is as follows: In Section II, 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 III. In Section IV, a partition method is introduced to process a large size image when the architecture has a less number of PEs. Section V gives conclusions.

II. BASIC IDEAS AND THE PARALLEL ALGORITHM

Since the object contour can be in an arbitrary shape, the distance transformation is sequential or iterative. Rosenfeld and Kak [20] proved that the distance transformation can be completed in two scans of the image. 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 $P(i, j)$, $i, j \in [0, N-1]$ 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. 1). 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 :

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

P(i-1 , j-1)	P(i-1 , j)	P(i-1 , j+1)
P(i , j-1)	P(i , j)	

(a)

	P(i , j)	P(i , j+1)
P(i+1 , j-1)	P(i+1 , j)	P(i+1 , j+1)

(b)

Fig. 1 The adjacent pixels of pixel P(i, j)
(a) in the forward pass and,
(b) in the backward pass.

Based on the above concept mentioned, we shall propose a parallel distance transformation algorithm. In this algorithm, we use one PE for the pipeline operations performed on the pixels in one row. Thus, the required number of PEs is N. The algorithm is divided into two passes: forward and backward. In the forward pass all image rows are scanned and processed in parallel. 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. 2, 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 $\{P(i, j)\}$.
Output :  An $N \times N$ distance map $\{d(P(i, j))\}$.
Procedure:
/Initialization/
for $i = -1, 0, 1, ..., N$,   $j = -1, 0, 1, ..., N$
  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  $d(P(i, j)) = \infty$
for all $PE_i$, $i = 0, 1, \dots, N-1$,   $j = 0, 1, ..., N-1$
begin
step 1: forward pass
TD = $\mathrm{Min}(d(P(i-1, j-1)) + d_{n0}, d(P(i-1, j)) + d_{n1},$
       $d(P(i-1, j+1)) + d_{n2}, d(P(i, j-1)) + d_{n3})$,
$d(P(i, j)) = \mathrm{Min}(TD, d(P(i, j)))$,
output $d(P(i, j))$ to $PE_{i+1}$.
step 2: backward pass
TD = $\mathrm{Min}(d(P(N-i, j+1)) + d_{n0}, d(P(N-i, j)) + d_{n1},$
       $d(P(N-i, j-1)) + d_{n2}, d(P(N-i-1, j+1)) + d_{n3})$,
$d(P(N-i-1, j)) = \mathrm{Min}(TD, d(P(N-i-1, j)))$,
output $d(P(N-i-1, j))$ to $PE_{i+1}$.
end
```

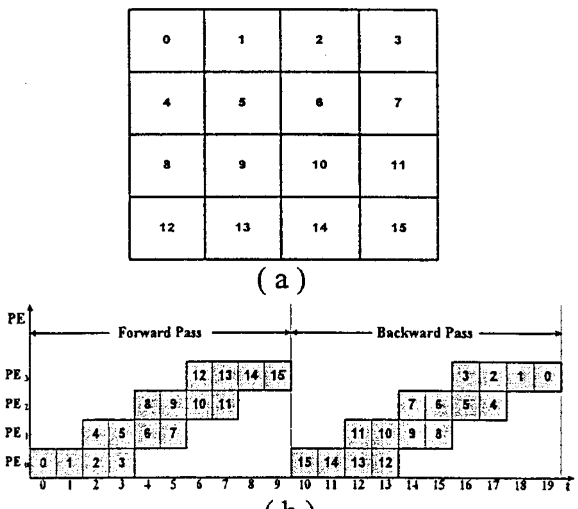


Fig. 2 (a) A 4×4 image. (b) The time-space diagram of the forward and backward passes.

The neighborhood distances d_{nk}, k = 0, 1, 2, and 3, are defined according to the selected distance function, as shown in Fig. 3. 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 chessboard function, the values of the neighborhood distances d_{n0}, d_{n1}, d_{n2}, d_{n3} are 1, 1, 1, and 1, respectively.

III. THE LINEAR ARRAY ARCHITECTURE

For the above parallel distance transformation algorithm, it can be implemented by a linear array architecture to meet the real time requirement. The linear array consists of N PEs to process a binary image with N×N pixels, as shown in Fig. 4. The PE's are indexed from left to right, beginning with PE_0. Initially, the image buffer 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 is at two cycles after that of PE_{i-1}. This is achieved by inputting the data in each row (i = 0, 1, 2, ..., N-1) through a delay line with a length of 2×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 d(P(i, j)) during the backward pass.

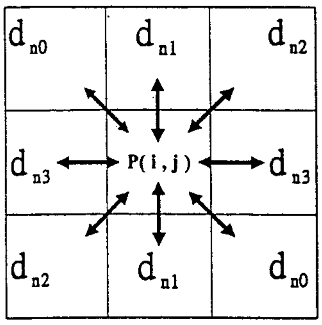
(A) The Processing Element:

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) Outputing the distance value of P(i, j) to PE_{i+1} and image buffer.

After taking all the necessary operations into consideration, the structure of PE_i shown in Fig. 5 is proposed. Two input ports are required to receive the distance values from PE_{i-1} and the image buffer. 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



(a)

distance function / d_{nj}	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. 3 (a) The notations of the neighboring distance constants, (b) the values of the distance constants of the various distance functions.

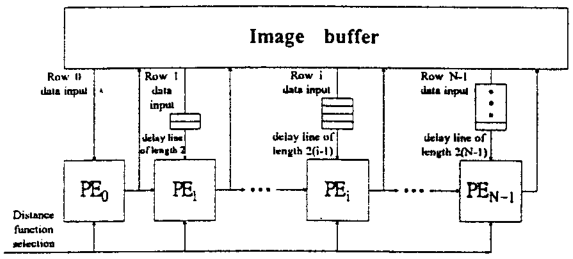


Fig. 4 The linear array architecture.

row i-1 are obtained in sequence through the port 2. These input distance values are inputted with two delay units. As a result, the needed input data from row i-1 are ready before starting to calculate 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 distance map memory.

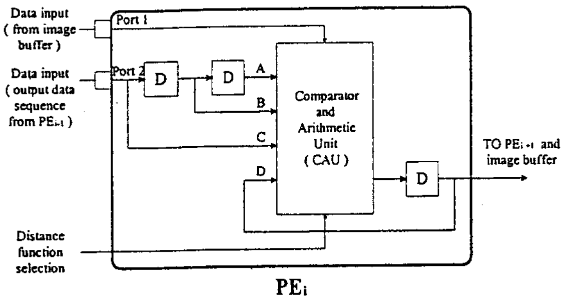

Data input (from image buffer)
Port 1
Data input (output data sequence from PE_{i-1})
Port 2
D
D
A
B
C
D
Comparator and Arithmetic Unit (CAU)
D
TO PE_{i+1} and image buffer
Distance function selection
PE_i

Fig. 5 The organization of a PE.

(B) 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. 6. 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 city block function, the distance value at P(i-1, j-1), P(i-1, j), P(i-1, j+1), and P(i, j-1) are 2, 1, 2, 1, respectively. So the input operand is 2 for adder A, 1 for B, 2 for C, and 1 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

3N-2 = (the length of the first row)
$\times$ (the execution cycle per pixel)
$+$ (the skewed delay per row)
$\times$ (the row number of image - 1)
$= N \times 1 + 2 \times (N - 1)$.

Hence the total execution time for both forward and backward passed is 6N-4 cycles, so it is of order O(N). Hence it meets the requirements for real-time applications.

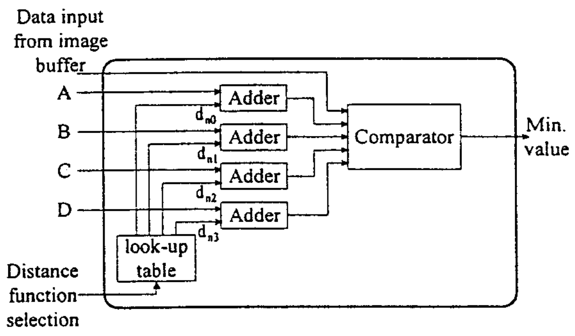


Fig. 6 The structure of a CAU unit.

IV. THE PARTITIONING METHOD

To make the linear array suitable for VLSI implementation, the number of PEs must be a reasonable number. This means that the number of PE, denoted as M, must be fixed and usually smaller than the image size N ($N \geq 512$) in practical applications. 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 $N \geq M$. In the first place, the image must be divided into a number of subimages $m = \left\lceil \frac{N}{M} \right\rceil$ each with M rows or less. This row partitioning starts from top to bottom (refer to Fig. 8.(a)), hence subimage S_j includes the part of image from row $M \times j$ to row $M \times (j+1) - 1$, where $j = 0, 1, \ldots, \left\lfloor \frac{N}{M} \right\rfloor -1$ 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. 7).

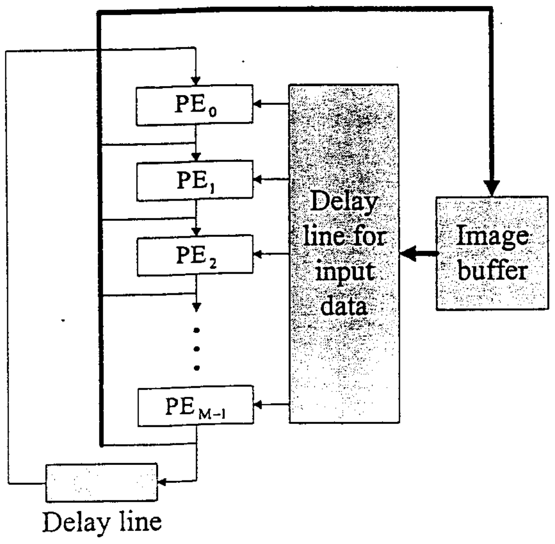


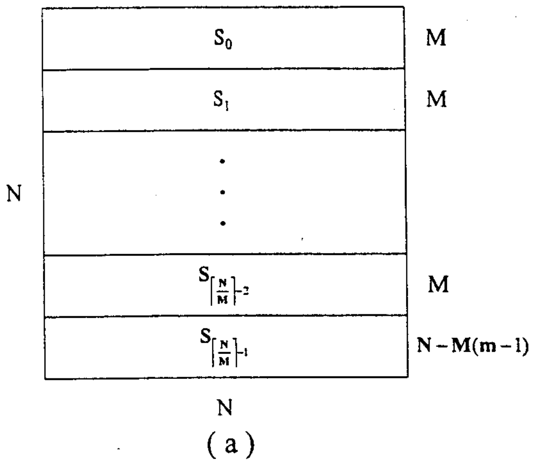
Fig. 7 The linear array architecture for executing the partitioned image.

According to the values of N and M, the execution model :

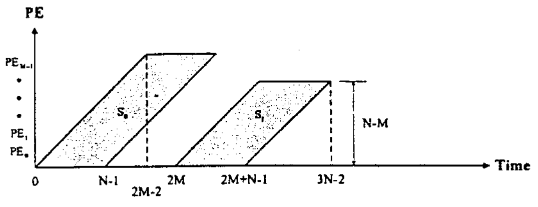
(1) Case 1: ($M<N\leq 2M$)

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 is arrived. Therefore, the subimage S_1 must be delayed 2M-N clocks before it is inputted to the port 1 of PE_0. The programming delay line will be set to zero. The time-space diagram is shown as Fig. 8.(b) and the total execution time are 2 (3N-2). For comparision, the time-space diagram for the two-pass algorithm [19] is shown as Fig. 9.(b), and the total execution time are 2 [4N + 2 (N-1)].

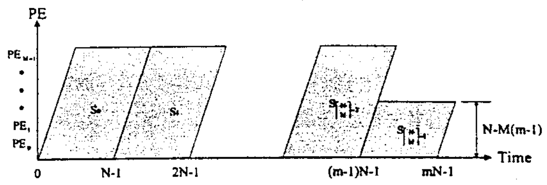
Based on the analysis mentioned above, we compute the distance map of the image of size $N\times N$ using $\left\lceil \frac{N}{2} \right\rceil$ PEs, and total execution time is the same if we use N PEs. In the case N=2M, its speed is respectively about 2 and 1.4 times faster than the two-pass and four-pass algorithms using the same number of PEs.


S_0
M
S_1
M
N
$S_{\lceil \frac{N}{M} \rceil -2}$
M
$S_{\lceil \frac{N}{M} \rceil -1}$
N - M(m - 1)
N

(a)



(b)



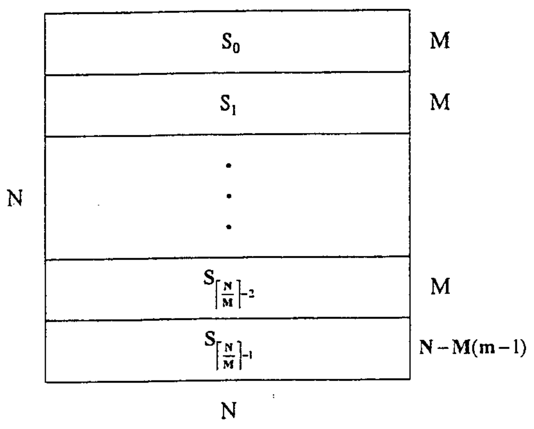
(c)

Fig. 8 The time-space diagram of the distance computation by our method for the case $N\geq M$, (a) an image is divided into subimages, (b) the diagram for $M\leq N\leq 2M$, (c) the diagram for $N\geq 2M$.

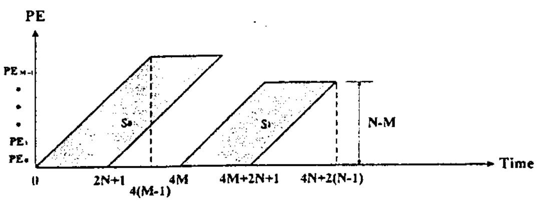
(2) Case 2: ($N\geq 2M$)

Since the output results of the last row of S_j produced from PE_{M-1} are N-2M clocks ahead the input data of the first row of S_{j+1} to PE_0, the output of PE_{M-1} must be inputted 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. 8.(c) and the total execution time are 2 { N m + 2 [N - M (m - 1) - 1] }. The time-space diagram of the two-pass algorithm is given in Fig. 9.(c) and the total execution time are 2 {(2m + 4) N - (m-1) 4M - 2}. The time-space diagram of the four-pass algorithm is given in Fig. 10.(b), and the total execution time are $4mN+\frac{N}{m}$.

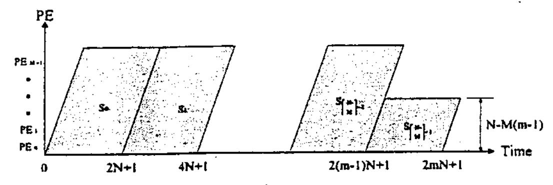
Consider that an image has a size of 1024×1024 and a linear array has M PEs with $M\leq1024$. When we compare the two-pass algorithm, the four-pass algorithm and our method, all using same number of PEs, the number of clock cycles required are different, as shown in Fig. 11. If N>>2M, our speed is nearly 2 times faster than the other two methods.


S_0
S_1
$S_{\lceil \frac{N}{M} \rceil -2}$
$S_{\lceil \frac{N}{M} \rceil -1}$
M
M
M
N-M(m-1)
N
N

(a)

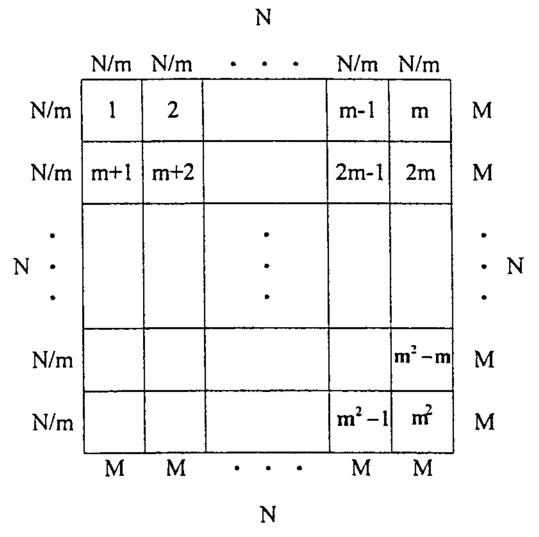


(b)

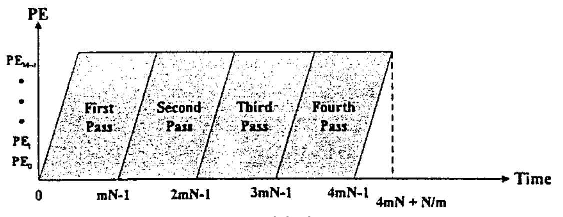


(c)

Fig. 9 The time-space diagram of the distance computation by the two-pass algorithm for the case $N\geq M$, (a) an image is divided into subimages, (b) the diagram for $M\leq N\leq 2M$, (c) the diagram for $N\geq 2M$.



(a)



(b)

Fig. 10 The time-space diagram of the distance computation by the four-pass algorithm for the case $N\geq M$, (a) an image is divided into subimages, (b) the diagram for $N\geq M$.

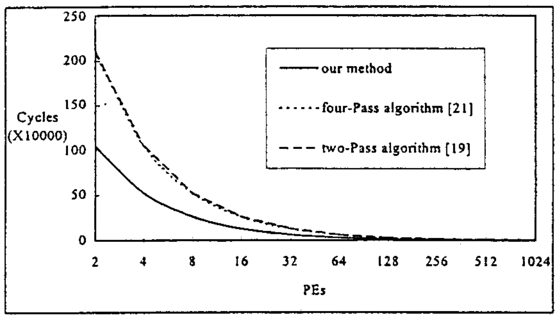


Fig. 11 Time comparison between two-pass algorithm, four-pass algorithm and our method.

V. 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 $\left\lceil \frac{N}{2} \right\rceil$ 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 and K. Ade, "Sequential thinning of object pictures using distance transformations," in *Proc. of the 8th Int. Joint Conf. on Pattern Recognition, Paris,* pp. 56-74, 1986.
2. H. Blum, "A transformation for extracting new descriptors of shape," in *Proc. Sym. on Models for Percep. of Speech and Visual Form,* pp. 362-390, 1967.
3. C. Arcelli and G. Sanniti di Baja, "Computing Voronoi diagrams in digital pictures," *Pattern Recognition Letters,* (4), pp. 383-389, 1986.
4. P. W. Verbeek, L. Dorst, B. J. H. Verwer, and F. C. A. Groen, "Collision avoidance and path finding through constrained distance transformation in robot state space," in *Proc. Int. Conf. On Intelligent Autonomous Systems,* pp. 634-671, 1986.
5. C. Wayne and B. Philip, "Generation skeletons and centerlines from the distance transform," *CVGIP: Graphic Model and Iimage Processing,* Vol. 54, (5), pp. 420-437, 1992.
6. H. C. Liu and M. D. Srinath, "Partial shape classification using contour matching in distance transformation," *IEEE Trans. Pattern Anal. Machine Intell.,* Vol. 12, (11), pp. 1072-1079, Nov. 1990.
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.,* Vol. 24, (1), pp. 111-115, Jan. 1994.
8. P.P. Das , "Best simple octagonal distances in digital geometry, " *Journal of Approximation Theory,* Vol. 68, pp. 155-174, 1992.
9. G. Borgefors , "Distance Transformations in Digital Images, " *Computer Vision, Graphics, and Image Processing,* Vol. 34, pp. 344-371, 1986.
10. I. Ragnemalm, "The Euclidean distance transform in arbitrary dimensions," *Pattern Recognition Letters,* Vol. 14, pp. 883-888, 1993.
11. A. Rosenfeld and J. L. Pfalz, "Distance functions on digital pictures," *Pattern Recognition,* Vol. 1, pp. 33-61, 1968.
12. A. Rosenfeld and J. L. Pfalz, "Sequential operations in digital picture processing," *Journal of the ACM,* Vol. 13, (4), pp. 471-494, Oct. 1966.
13. G. Borgefors, T. Hartmann, and S. L. Tanimoto, "Parallel Distance Transforms on Pyramid Machines: Theory and Implementation," *Signal Processing,* Vol. 21, (1), pp. 61-86, Sept. 1990.
14. H. Yamada, "Complete Euclidean distance transform by parallel operation," in *Proc. on the 7th Int. Conf. on Pattern Recognition,* Montreal Canada, pp. 69-71, 1984.
15. G. Borgefors, "Hierarchical chamer matching: a parametric edge matching algorithm, " *IEEE Trans. Pattern Anal. Machine Intell.,* Vol. 10, (6), pp. 849-865, 1988.
16. D. Zhao and D. G. Daut, "A real-time column array processor architecture for images," *IEEE Trans. Circuits Syst. Video Tech.,* CSVT-2, (1), pp. 38-48, 1992.
17. J. Piper and E. Granum, "Computing distance transformations in convex and non-convex domains," *Pattern Recognition,* Vol. 20, (6), pp. 599-615, 1987.
18. D. W. Paglieroni, "A unified distance transformation algorithm and architecture," *Machine Vision and Applications,* Vol. 5, pp. 47-55, 1992.
19. F. Y. Shih, C. T. King, and C. C. Pu, "Pipeline architectures for recursive morphological operations," *IEEE Trans. Image Processing,* IP-4, (1), pp. 11-18, 1993.
20. A. Rosenfeld and A. C. Kak, *Digital picture processing:* Vol. 2, 2nd edition, Academic Press, New York, U.S.A., 1982.
21. C. H. Chen and D. L. Yang, "Fast algorithm and its systolic realization for distance transformation," *IEE Proc. Comput. Digit. Tech.,* Vol. 143, (3), pp. 168-173, May 1996.