ア ル ゴ リ ズ ム 84－3
（２００２．５．２３）

ユークリッド距離変換アルゴリズムのハードウェア化に関する研究

曾　培峰、平田　富夫
〒 464-8603 名古屋市千種区
不老町 1 番地
名古屋大学大学院
工学研究科
電話: +81-52-789-3440
FAX: +81-52-789-3089
Email: zengpf@ieee.org, hirata@nuee.nagoya-u.ac.jp

あらまし
バイナリ画像のユークリッド距離変換アルゴリズムに基づいて、サイズの $O(N^2)$ を持つハードウェアが提案された。In-place 計算アルゴリズムをハードウェア化することで、入力データ、中間結果や出力を同じ領域にセーブし、ハードウェア量を大幅に軽減した。また、乗算器の代わりにカウンタや加算器を用いることで計算速度を向上させた。

キーワード
ユークリッド距離変換、VLSI、画像処理、パタン認識

A Hardware Algorithm for Euclidean Distance Transform

PeiFeng Zeng and Tomio Hirata
Graduate School of Engineering
Nagoya University
Furo-cho, Chikusa-ku
Nagoya 464–8603, Japan
Telephone: +81-52-789-3440
Fax: +81-52-789-3089
Email: zengpf@ieee.org, hirata@nuee.nagoya-u.ac.jp

Abstract

Based on an efficient algorithm of Euclidean distance transform for binary images, a circuit of $O(N^2)$ size is proposed. By using in-place calculation, intermediate data storing and result output can use the same memory with the input data. This reduces the amount of memory largely. By replacing the multipliers with counters and adders, the size of the circuit is further reduced and the calculation speed is also improved.

Keywords

Distance transform, VLSI, image processing, pattern recognition

I. Introduction

For binary images, the distance map gives the distances from pixels to the nearest pixels of value zero, called 0-*elements* in the paper. Besides the Euclidean distance that belongs to L_2 distance, there are many kinds of distances such as city block (L_1), chessboard (L_∞), and octagonal distances. They have found many applications in image processing [1], pattern recognition, morphology and robotics, etc.

Euclidean distance transform could not find much applications when the approach with time complexity $O(N^3)$ was known for $N \times N$ images. It was improved in [2] to $O(N^2 \log N)$. The time complexity was further reduced to $O(N^2)$ in [3][4]. Hirata [4] proposed a simple $O(N^2)$ time algorithm for computing

Euclidean distance maps. These efforts make practical applications of Euclidean distance transform possible.

Since the emergence of digital camera, image size becomes larger and larger. Expanding applications of digital image raise the need to manipulate vast amounts of visual data efficiently. That is, the size of circuit should be reduced for processing images of a determined dimension.

Among algorithms for Euclidean distance transform with the time complexity of $O(N^2)$, the proposal in [4] does the distance transform in two separate steps. This enables a easy design of hardware. So the implementation of Euclidean distance transform circuit in this paper is based upon the algorithm.

Since the algorithm is software-oriented, some reforms should be applied in our implementation to improve the processing speed and reduce the size of the circuit. In this paper, the *in-place* technique is used and the input data, intermediate data are stored in the same memory with the output results. Without extra memory for storing a large quantity of intermediate data, the scale of circuit is reduced greatly, and the control circuit is simplified, too. Also, in our design, comparators and adders are used to implement the function of multipliers so as to speed up the calculation and reduce the circuit size.

This paper is organized as follows. Section II gives some definitions about distance transform and the implementation of Euclidean distance calculation that is used as the base of our design. In Section III and Section IV, the circuit for in-place Euclidean distance calculation is realized. In Section V, the size and performance of circuit is studied. Some conclusions are reached in Section VI.

II. Preliminaries

A. Definitions for Distances

In this paper, we denote by $\mathbf{B} = \{b_{i,j}\}$ a binary image with the size $N \times N$, and (i, j) the element in the ith row and jth column of the image. The *Euclidean distance map* $\mathbf{D} = \{d_{i,j}\}$ of an image $\mathbf{B} = \{b_{i,j}\}$ is defined as

$$d_{i,j} = \min_{0 \le p,q \le N-1} \left\{ \sqrt{(i-p)^2 + (j-q)^2} |_{b_{p,q}=0} \right\}. \tag{1}$$

In image processing, there are also other kinds of distances defined for two pixels of an image. We denote by $p_i = (x_i, y_i)$ a pixel in an image. The *city block* distance is defined as

$$d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|. \tag{2}$$

The *chessboard* distance is defined as

$$d(p_1, p_2) = \max(|x_1 - x_2|, |y_1 - y_2|). \tag{3}$$

The *chamfer* distance is defined as

$$d(p_1, p_2) = \begin{cases} |x_1 - x_2| + (\sqrt{2} - 1)|y_1 - y_2|, & \text{if } |x_1 - x_2| > |y_1 - y_2|, \\ |y_1 - y_2| + (\sqrt{2} - 1)|x_1 - x_2|, & \text{elsewhere.} \end{cases} \tag{4}$$

The Euclidean, city block, and chessboard distance belong to L_2, L_1 and L_∞ distances, respectively [2]–[4].

B. The Unified Linear-time Scheme

The unified linear-time distance transform scheme is presented in [4] for calculating a wide class of distances including the Euclidean distance. It consists of two stages, i.e., the vertical scanning and the horizontal scanning.

B.1 Vertical scanning **T1**

In stage **T1**, the input image $\mathbf{B} = \{b_{i,j}\}$ is moved into the working area M and its *column distance* map $G = \{g_{i,j}\}$ is calculated for the latched data in $M = \{m_{i,j}\}$. The column distance is defined here as the distance from an element to the nearest 0-element in the same column. For each column j of $\mathbf{M} = \{m_{i,j}\}$, the column distance map $\mathbf{G} = \{g_{i,j}\}$ satisfies

$$g_{i,j} = \min_{0 \le p \le N-1} \{|i - p|_{b_{p,j}=0}\}, \tag{5}$$

where $g_{i,j} = \infty$ for a column j with no 0-element.

B.2 Horizontal scanning **T2**

In stage **T2**, the Euclidean distance map $D = \{d_{i,j}\}$ is calculated based upon the column distance map G. For each row i in $\mathbf{G} = \{g_{i,j}\}$, the distance from pixel (i, k) to the nearest 0-element in j column can be expressed as

$$f_k^i(j) = (j - k)^2 + g_{i,j}^2. \tag{6}$$

Thus the Euclidean distance from pixel (i, k) to the nearest 0-element turned out to be

$$d_{i,k}^2 = \min_{0 \le j \le N-1} f_k^i(j). \tag{7}$$

B.3 Improvement of **T2**

From the horizontal scanning, we know that the Euclidean distance from pixel (i, k) to the nearest 0-element is the smallest value of $f_k^i(j)$ for $0 \le j \le N - 1$. On the other hand, we can also express the distance for a row from the nearest 0-element in column j as a function $(j - k)^2 + g_{i,j}^2$. Such a function appears to be a curve when we draw it as a function of k, the column index of each element in a row. We can draw all such functions for a row by changing the value of j from 0 to $N - 1$. Since the Euclidean distance is defined as the shortest distance to 0-elements, it is the lower envelope formed by these functions. See Fig. 1.

In Fig. 1, every pair of curves intersect exactly once. So after calculating all intersections and using a stack, the lower envelope of all these curves can be determined [4]. The Euclidean distance is simply the lower envelope in Fig. 1.

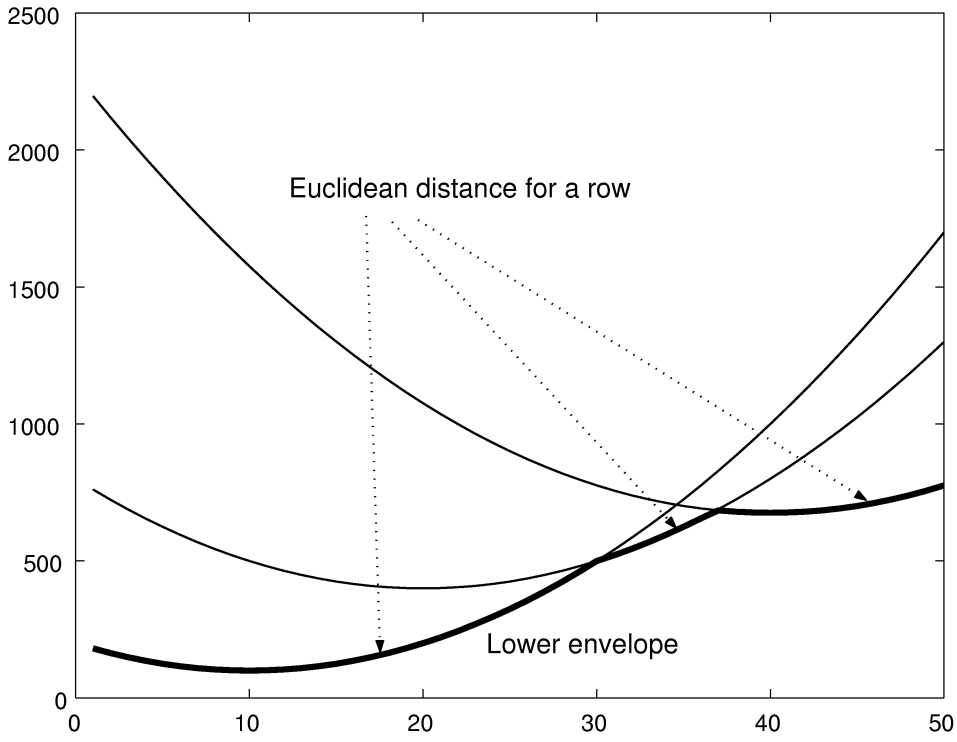

Fig. 1. Euclidean distance for a row in an image.

III. **T1** Implementation and Modification

From the discussion in Section II, we understand that in **T1**, column distance for all elements are to be calculated.

The conventional calculation is implemented by scanning the working area M twice. That is, the column distance is calculated by a forward scanning and a reverse scanning. They include the following steps.

Forward scanning:

1. Vector A of the length N is allocated for storing the temperary data, set $\{a_j = \infty|_{0\le j\le N-1}\}$ for all elements of A as the initialization[1].
2. Check the data in M from row 0 to row $N-1$ orderly. The elements in A will be incremented when the scanning is advanced to the next row. But $a_j = 0$ is set when $m_{i,j} = 0$ is encountered in scanning the row i.
3. Stop the scanning when the scanning for row $N-1$ is completed.

Reverse scanning:

1. Vector A of the length N is allocated for storing the temperary data, set $\{a_j = \infty|_{0\le j\le N-1}\}$ for all elements of A as the initialization.
2. Check the data in M from row $N-1$ to row 0 orderly. The elements in A will be incremented when the scanning is advanced to the next row. But $a_j = 0$ is set when $m_{i,j} = 0$ is encountered in scanning the row i.
3. Stop the scanning when the scanning for row 0 is completed.

We denote the results of the first scanning and the second scanning be $G' = \{g'_{i,j}\}$ and $G'' = \{g''_{i,j}\}$, respectively. The results of **T1** should be the minimum values of G' and G'', i.e., $G = \{\min(g'_{i,j}, g''_{i,j})\}$.

In the above calculation, a matrix G' is used to save the results of the first scanning that is of the same size as M.

To reduce the size of total memory used for **T1** calculation, the in-place technique is applied in this paper. The processing of **T1** is realized in two stages as follows.

1. *Data Input.* For input binary image $B = \{b_{i,j}|_{0\le i,j\le N-1}\}$, a working area is allocated with the size of $N \times N$. $\{b_{i,j}\}$ are latched into M row by row.

   In the data latching, the column distances $\{g_{0,j}|_{0\le j\le N-1}\}$ for every element in the first row of M to 0-element in the same column is calculated for each column.

   Thus, N times of data transferring are needed to move all data from B into M.
2. *Column Distance Calculation.* Calculate the shortest column distances for remaining elements in M to the 0-elements in the same columns $\{g_{i,j}|_{1\le i\le N-1}\}$ with the information $\{g_{i-1,j}|_{1\le i\le N-1}\}$, the previous calculation result. Store the results back into M.

In the following part of the section, the two stages of data input and column distance calculation in **T1** will be discussed in details.

A. Data Input

In the stage of data input, the input binary images are latched into the working area M. The corresponding unit of circuit is shown in Fig. 2. It works in the following steps:

1. All elements in the first row of M are initialized such that $\{m_{0,j} = \infty|_{0\le j\le N-1}\}$.
2. For processing the input data of row i, $\{b_{i,j}|_{0\le j\le N-1}\}$ are fed into a multiplexer to produce $F = \{f_j|_{0\le j\le N-1}\}$ that meet with
$$f_j = \begin{cases} 0, & b_{i,j} = 0 \\ \infty, & b_{i,j} = 1 \end{cases},$$
for $0 \le j \le N-1$.
3. A comparator bank is used to produce the result $\{\min(m_{0,j}, (f_j + i))|_{0\le j\le N-1}\}$ and send them to row 0 of M. The data are the column distance for row 0 in the range from row 0 to row i.
4. Meanwhile, the input data $\{b_{i,j}|_{0\le j\le N-1}\}$ are stored in the row i of M for $i > 0$.
5. Advance the processing to the next row, i.e., the row $i+1$. Repeat the operations from Step 2 to Step 4. The calculation will be finished after the last row is processed.

After the above processing, the data in row 0 of M turn out to be the column distances for the elements in the first row. With them and the latched data in remaining rows of M, we can start the stage of column distance calculation.

B. Column Distance Calculation

With the column distances for the first row of M calculated in the previous step, the column distances for the rest rows are calculated in the parallel mode. That is, for each column, we calculate the column

[1]The ∞ is defined as a large value in our implementation, i.e., $N^2 + 1$, that is larger than any possible distance value for an $N \times N$ image.

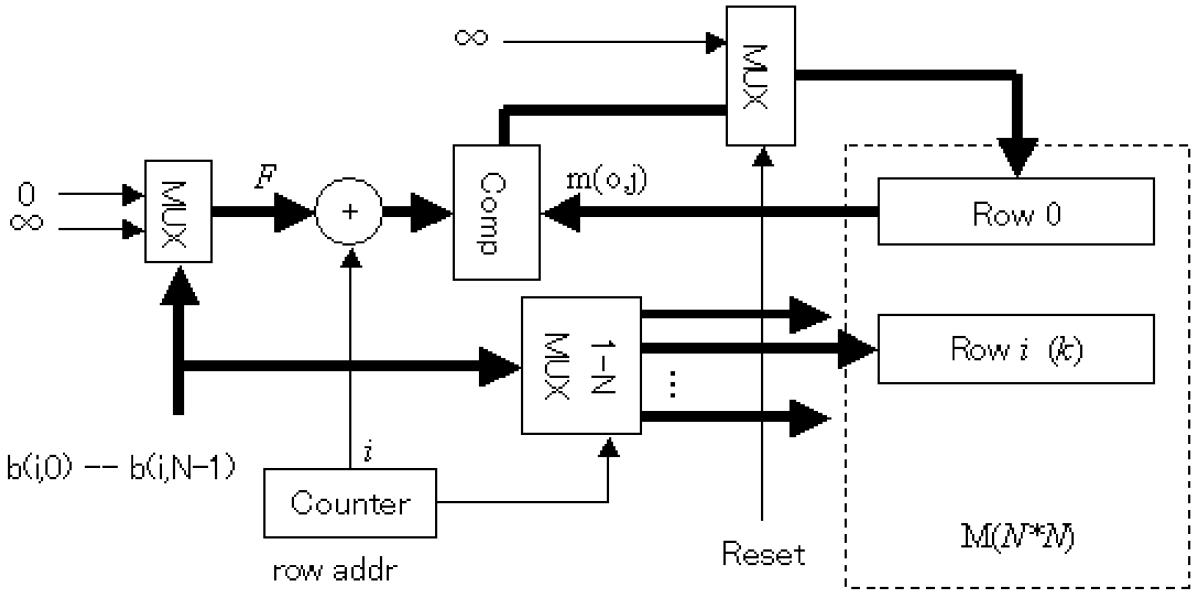

Fig. 2. Data input unit. Buses are of N bytes and drawn in thick line.

distance for the next row with the data of the current row at the same time. The implementation is shown in Fig. 3. To simplify the explanation, we discuss the calculation for column j.

1. A vector H with the length of N is allocated for column distance calculation. Its data are set to -1 as initialization. Thus, we have the initial value $h_j = -1$ for column j.
2. When the calculation for row i is finished, we have $g_{i,j}|_{0 \le i \le N-2}$, it indicates that the nearest 0-element in the forthcoming rows should not be appeared in the range $[i, i + g_{i,j})$.
3. When $g_{i,j} = 0$, the 0-element is at the exact row, the column distance for the following row should be incremented. Thus $h_j = +1$.
4. Calculate the column distance for row $i + 1$, i.e., $g_{i+1,j} = g_{i,j} + h_j$.
5. In the case that $h_j = +1$, the above calculation takes two elements at rows $(i + 1) + g_{i+1,j} - 1$ and $(i + 1) + g_{i+1,j}$ into the scope of consideration.
6. We set $h_j = -1$ when either of the two elements equals to 0.
7. Decrement the result when the element at row $(i + 1) + g_{i+1,j} - 1$ equals to 0.
8. Save the result into row $i + 1$ and advance the current row to $i + 1$.
9. Repeat the steps from 3 to 8 for all rows in the working area M.

After above calculations, the data in M turn out to be the column distance of the input image.

IV. **T2** Implementation and Modification

After the column distances for all elements have been calculated in previous section, the step of **T2** calculation can be executed and the Euclidean distance map D will be obtained. The algorithm in [4] does the **T2** calculation by searching the lower envelope of all distance function in Fig. 1. By calculating the intersections and using a stack, the lower envelope is calculated with the time complexity of $O(N^2)$.

While in hardware-oriented implementation, it is difficult to calculate the intersections and manipulate them efficiently. Instead, we have to calculate the distance values for all elements in a row, compare their values and select the smallest one. If only all Euclidean distances $\{(k - j)^2 + g^2_{i,j}|_{0 \le k \le N-1}\}$ should be calculated and compared, can we obtain Euclidean distance for element (i, j). That will deteriorate the time complexity to $O(N^3)$.

So one of the main objectives in our hardware-oriented design of **T2** is to find other solutions that implement the above calculation without so much time. We attack the problem by replacing multipliers with adders and comparators. That scheme can speed-up the total calculation by replacing multipliers with adders and comparators. Furthermore, that can also reduce the size of our circuit. The implementation of **T2** follows these steps as shown in Fig. 4:

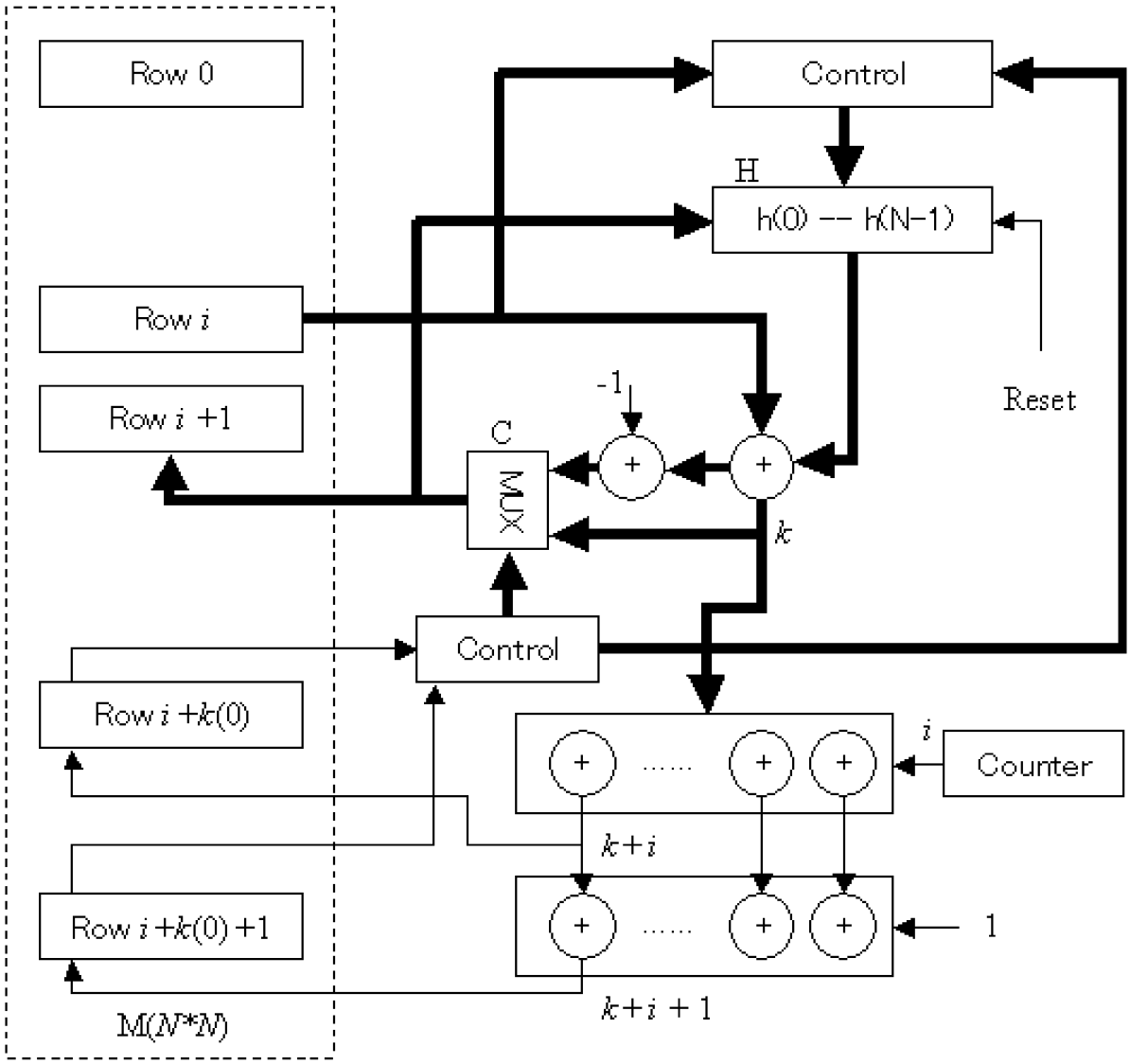

Fig. 3. Column distance calculation of **T1**. Buses are of N bytes and drawn in thick line.

A vector E with the length of N is used to store the intermediate result of the distance values $\{d_j|_{0\le j\le N-1}\}$.

In the Euclidean distance calculation for row i, the circuit executes the following steps.

1. Set the initial values in E to ∞ before the calculation.
2. Reset the column address counter J so the first element $g_{i,0}$ will be selected for calculation.
3. Repeat the steps from 4 to 6 for $0 \le j \le N-1$.
4. Calculate $g_{i,j}^2$ and j^2 simultaneously. The two operation takes one multiplication cycle.
5. Calculate $q_0 = g_{i,j}^2 + j^2$, the distance for column 0. For each column k $(0 < k \le N-1)$, calculate $q_k = q_{k-1} - 2j$. At the same time, another operation is executed for $d'_{k-1} = q_{k-1} + k^2$, the distance result [2] for column $k-1$. In the calculation, the data k^2 are buried in the circuit as constants. The calculation takes one cycle of addition when the k increments. So the total operations for the calculation of the data $\{d'_k|_{0\le k\le N-1}\}$ will be $N+1$ cycles of addition.
6. Compare the data $\{d_k\}$ in E and the $\{d'_k\}$. The results $\{\min(d_k, d'_k)|_{0\le k\le N-1}\}$ are selected by the comparator bank P and sent back to E.

[2] $d'_k = g_{i,j}^2 + (k-j)^2 = (g_{i,j}^2 + j^2) - \sum_k 2j + k^2 = q_{k-1} - 2j + k^2$.

7. When the above calculation is executed for $0 \leq j \leq N-1$, the data in E are stored back to row i in M.

Repeat the above operations for all rows to calculate Euclidean distance for a whole input image.

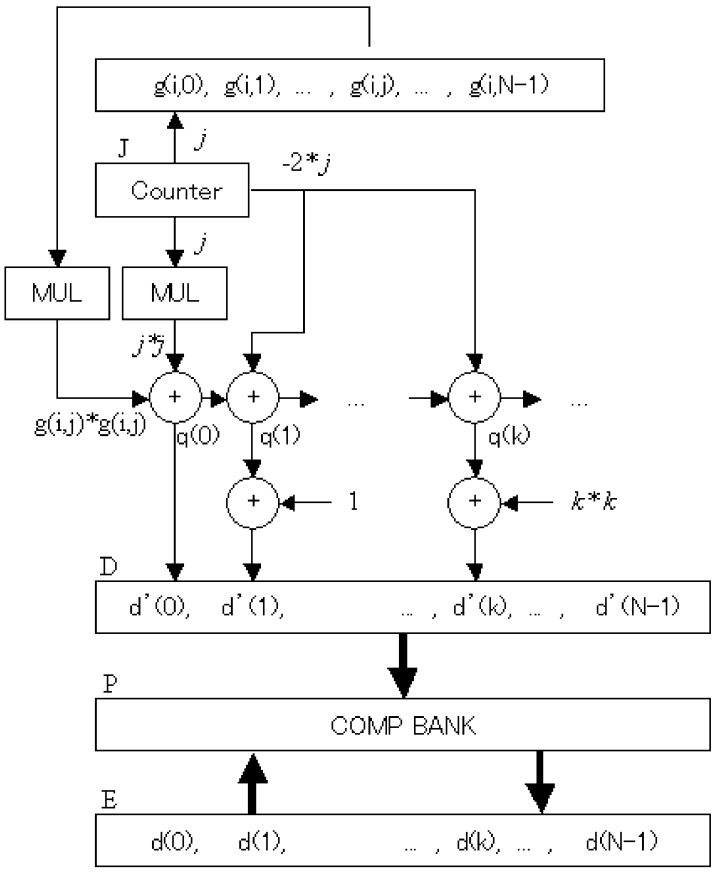

Fig. 4. The calculation of **T2**. Buses are of N bytes and drawn in thick line.

V. Circuit Size and Calculation complexity

In order to reduce the scale of circuit, or increase the processing ability in limited space, we must avoid the use of complicated components in our design. Meanwhile, there is a relation between the size of circuit and the computation complexity. The size of circuit can be further reduced on sacrificing the processing speed. So we must well determine the trade-off between the size of circuit and the processing speed.

A. Circuit Size

In this paper, we judge the size of our circuit upon the number of multipliers, comparators, adders, and buffers used. The center part of our circuit is the working area M that is of the size $N \times N$. Besides M, the number of multipliers, comparators, adders, and buffers for intermediate data storing also have a considerable contribution on the size of circuit. The numbers of various components in each unit of circuit are listed in Table I.

Moreover, there are some controlling units in our design. They will take additional space.

B. Calculation Complexity

Because the three units in our design work in cascade mode, the total calculation time will be the sum of all the processing time of the three parts.

In the data input unit, N data are processed in parallel, data are transferred into working space M row-by-row. N clocks are needed for total data transferring for the size of $N \times N$. In every clock there

TABLE I

CIRCUIT COMPONENTS USED IN EACH UNIT OF CIRCUIT FOR AN INPUT IMAGE WITH THE SIZE OF $N \times N$

Name	Data input	**T1**	**T2**
Multipliers	—	—	3
Comparators	N	—	N
Adders	N	$4N$	$2N-1$
Buffers	—	N	$2N$
Multiplexers	2N	N	—

include four steps of data processing, i.e., one step of multiplex switching, one step of data addition, one step of data comparison, and one clock of data saving.

In **T1** calculation unit, N data are also processed in the parallel mode. All the $\{g_{i,j}|_{1\leq i\leq N-1}\}$ can be calculated in $N-1$ calculation cycles. In every cycle, there include four adding operations (they take three steps of addition since one adder is operated simultaneously with other adders), two compare operations that can be executed simultaneously, one multiplex operation, and one data saving clock.

In **T2** calculation unit, to calculate the Euclidean distance for row i, all distances based upon $\{g_{i,j}|_{0\leq j\leq N-1}\}$ should be calculated. From Fig. 4, we understand that two multiplications can be performed simultaneously so they take one multiplication cycle. $2N-1$ adders are divided in two groups and they take $N+1$ steps of addition. The comparator bank do the data comparison in parallel mode so N data comparison only takes one operation cycle. So we can conclude that for an input image with the size of $N \times N$, **T2** calculation takes N^2 cycles of Multiplication, $N^2(N+1)$ cycles of addition, and N^2 cycles of comparison.

VI. CONCLUSIONS

In this paper, A hardware-oriented designing for 2D Euclidean distance calculation is discussed. The circuit executes the calculation in the same way as the scheme in [4] and does the column scanning for an input image to calculate the column distance. Then it does the row scanning to obtain the Euclidean distance based upon the column distance. In the row scanning, the scheme in [4] calculates the intersections for curves and sorts these intersections based upon the position of them. But it is difficult to realize the scheme in hardware. In this paper, we use adders and comparators to replace the multipliers to speedup the calculation and reduce the size of circuit. The circuit complexity of our design is $N(O^2)$. In addition, the in-place scheme is applied in both the column scanning and the row scanning. This greatly reduces the scale of our design with no deterioration upon calculation performance.

REFERENCES

[1] P.F. Zeng and T. Hirata. Distance map based image enhancement. *IEEE Trans. IP*, 2002. (submitted for publication).

[2] M. N. Kolountzakis and K. N. Kutulakos. Fast computation of Euclidean distance maps for binary images. *Information processing letters*, 43:181–184, 1992.

[3] 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.

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