Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

ELSEVIER

Information Sciences 149 (2003) 219–234

INFORMATION SCIENCES
AN INTERNATIONAL JOURNAL

www.elsevier.com/locate/ins

A parallel algorithm for generating chain code of objects in binary images

Tsorng-Lin Chia [a,*], Kuang-Bor Wang [a], Lay-Rong Chen [b], Zen Chen [b]

[a] *Department of Electrical Engineering, Chung Cheng Institute and Technology, Taoyuan 33509, Taiwan, ROC*

[b] *Institute of Computer Science and Information Engineering, National Chiao Tung University, Hsinchu, Taiwan, ROC*

Received 30 December 2000; received in revised form 23 January 2002; accepted 29 April 2002

Abstract

This paper addresses parallel execution of chain code generation on a linear array architecture. The contours in the proposed algorithm are viewed as a set of edges (or contour segments) that can be traced by a top-down contour tracing method to generate the chain codes for the outer and inner object contours. A parallel algorithm that contains the chain code generating rules and operations needed is also described, and the algorithm is mapped onto a one-dimensional systolic array containing $\lceil \frac{1}{2}(N+1) \rceil$ processing elements (PEs) to devise this architecture. The architecture extracts the contours of objects and quickly generates the corresponding chain codes after the image data in all rows are inputted in a linear fashion. The total processing time for generating the chain codes in an $N \times N$ image is $O(3N)$. By doing so, the real-time requirement is fulfilled and its execution time is independent of the image content. In addition, a partition method is developed to process an image when the parallel architecture has a fixed number of PEs; say two or more. The total execution time for an $N \times N$ image by employing a fixed number of PEs is $N(N+1)/M + 2(M-1)$, when M is the fixed number of PEs.

Keywords: Chain code; Parallel algorithm; Chain collection; Chain linking

* Corresponding author. Fax: +886-3-3801407.
E-mail address: tlchia@ccit.edu.tw (T.-L. Chia).

0020-0255/02/$ - see front matter
PII: S0020-0255(02)00267-0

1. Introduction

A prominent shape feature of an object in a binary image is its object contour. The object contour is useful in object analysis [1], pattern recognition [2], image restoration [3,4], and object features computation [5–7] such as the perimeter, area, and corner. The chain codes concisely represent the object contour, and several algorithms have been documented to generate the chain code [8–12].

The generation process of the object contour chain codes is a non-local operation since the object contour in the image can be an arbitrary shape. Consequently, most procedures for the chain code generation are sequential methods that trace the contour clockwise or counterclockwise, starting at an initial point and returning to the same point. However, a parallel method for chain code generation is necessary since these existing sequential algorithm usually do not meet the real-time requirement. Several parallel techniques for generating chain code have been developed in previous research. Schmidt [13], Cederburg [14], Chakravarty [15], and Zingaretti [16] employed a window operator to detect the contour codes in a raster scan fashion and store it into some data structure such as a linked list or a graph. The time complexity of these algorithm are generally $O(N^2)$, where $N \times N$ denotes the image size. Other methods derive the chain code from some other data structures such as run-length coding [16], pyramid [10], and quadtree [6], which also require a long processing time.

This paper presents a parallel architecture, which is an extension of the work in [2], to generate the chain code in real time. The parallel architecture requires $N + 1$ processing elements (PEs) for an $N \times N$ image, while the algorithm requires an execution time of $3N$ cycles regardless of the image content.

The rest of this paper is organized as follows. Section 2 introduces some definitions and contains the proposed chain code generation algorithm. The linking of chain codes belonging to the same object contour (inner and outer) is specified in Sections 3 and 4. The complexity and performance of the algorithm are analyzed in Section 5. A partition method is introduced in Section 6 to process a large size image when the architecture has a fixed number of PEs, while concluding remarks are provided in Section 7.

2. Chain code generation algorithm

2.1. Contour structure

An object of a region type can be represented by a set of the closed contours in a binary image, including an *outer contour* surrounding the object and, perhaps, some *inner contours* contained by the object [2]. A contour point is

denoted as a black (in the case of an outer contour) or white (in the case of an inner contour) pixel along the object boundary. A contour point can be classified into three types: transition pixels, contour starting point, and contour ending point. A black (or white) *transition* pixel is a pixel in the binary image where a transition from white to black (or black to white) occurs during a data row scan. An *outer contour starting point* (OCSP) is a black transition pixel whose neighbors in the upper, upper left and upper right are all white pixels. In contrast, an *inner contour starting point* (ICSP) is a white transition pixel whose neighbors in the upper, upper left and upper right are all black pixels. An outer or inner contour starting point can be viewed as a splitting point of two pieces of subcontours. A *contour ending point* is a merging point of two subcontours. For example, Fig. 1 illustrates the binary pattern of character "A" and its corresponding outer and inner contours. The solid lines represent the outer contour segments and the dash lines are the inner contour segments. The contour starting points of OCSP and ICSP are denoted as "◯" and "◎", respectively and the ending points are marked as a square "□".

2.2. Contour tracing

The contour tracing method is implemented in three steps: (1) the contour tracing is initiated by detecting a starting point, (2) the tracing is continued through the transition pixels, and (3) the tracing ends at an ending point. These steps will be presented as follows:

1. Assuming that the system fetches the row data from right to left for more convenient to detect object characteristic. By definition, the existence of a starting point at coordinate (i, j) depends on the black/white information of pixels at coordinates $(i-1, j-1)$, $(i-1, j)$, and $(i-1, j+1)$. Therefore, these preceding neighboring pixels must be made ready for use by the current pixel by shifting (i.e., skewed) the input data flow of the preceding row two cycles ahead of the current row.
2. The contour tracing is divided into two subcontours upon detection of a starting point. Each subcontour segment will be traversed by passing down an edge token, which specifies the traversal path of the transition pixels.
3. Two contour tracings will end when both meet at a common pixel that is not a transition point. However, this merger will rupture if the above common pixel is a transition pixel instead of an ending point.

2.3. Parallel tracing of contours

A parallel chain code generation algorithm is developed according to the concept mentioned above. One PE is adopted for the pipeline operations performed on the pixels in each row: the required number of PE is N, if the image has N rows. An additional PE (i.e., the $(N+1)$th PE) is applied for two

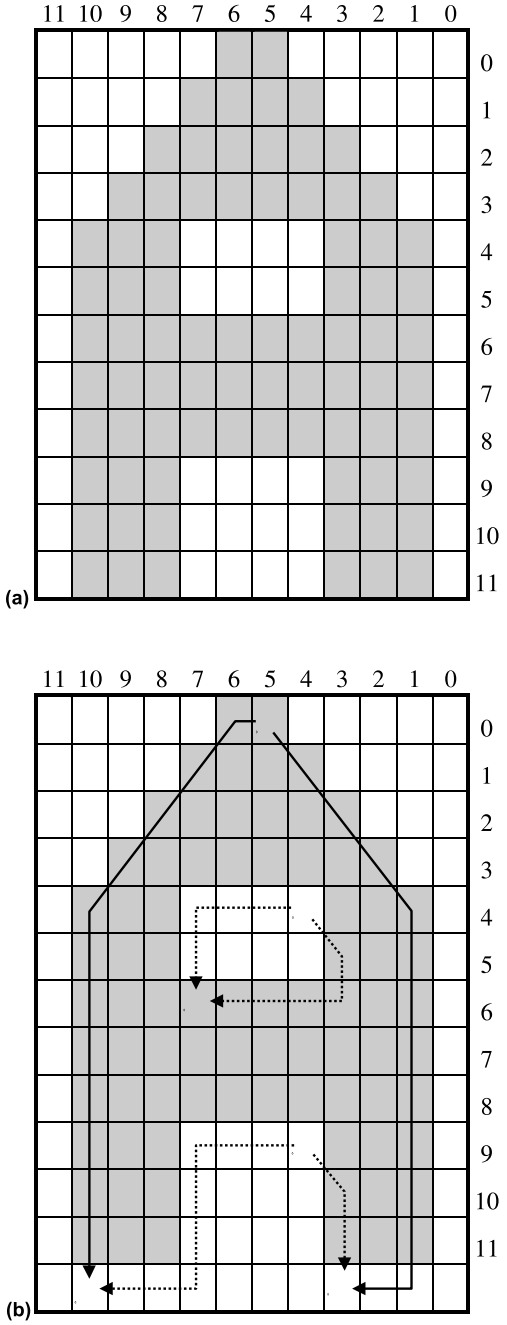


Fig. 1. (a) A binary pattern of the character "A". (b) The contour structure of the binary pattern.

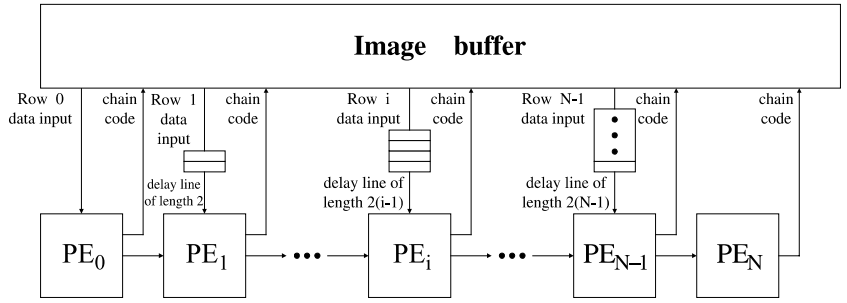


Fig. 2. The linear array architecture.

purposes: receiving the token output by the Nth PE, and detecting the ending points of contours. First, the linear array consisting of $N + 1$ PEs is devised to process a binary image with $N \times N$ pixels, as demonstrated in Fig. 2. The image buffer of the host computer initially stores the input binary image, while the buffer stores the chain code as the process continues. PE_i generally receives data from PE_{i-1} and derives the chain code of $P(i, j)$ in row i in every cycle. The input image in the image buffer is skewed so that the clock cycle of PE_i lags that of PE_{i-1} by two cycles because the data flow is shifted one pixel per cycle. This skewing is achieved by inputting the data in each row $(i = 0, 1, 2, \ldots, N - 1)$ through a delay line with a length of $2 \times i$. The space–time diagram of the image of character "A" in Fig. 1 after a two-cycle alteration illustrated in Fig. 3.

Each PE will execute one of the following three tasks or functions in each cycle:

Case 1. PE_i does not hold an edge token and has not received one from PE_{i-1}. The PE_i determines whether the current pixel is a transition pixel and

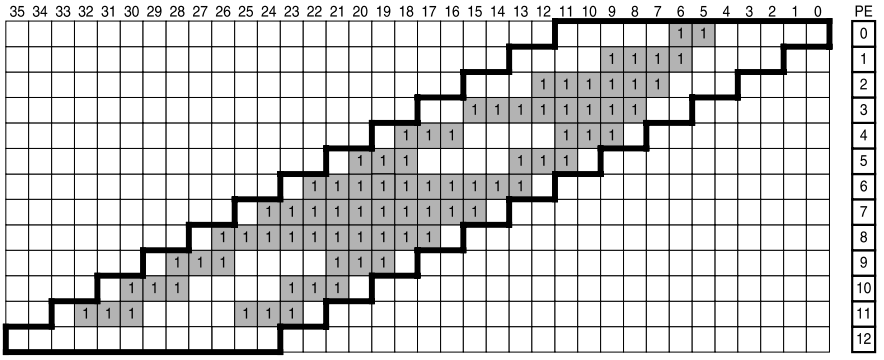


Fig. 3. The skewed image of Fig. 1.

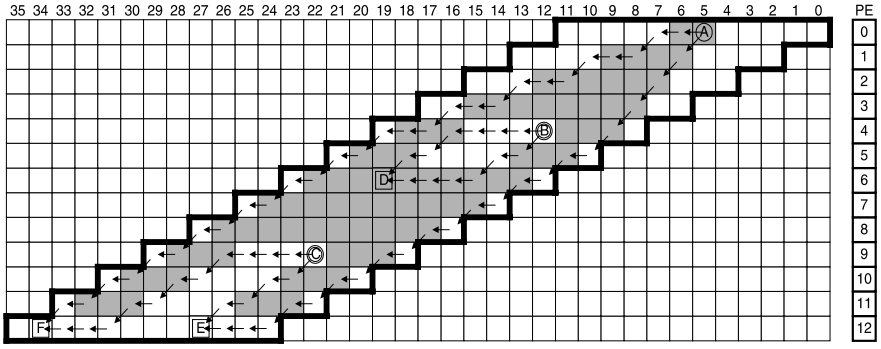


Fig. 4. Edge token transmitted during the contour tracing for the given pattern.

determines whether the transition pixel is an outer or inner contour starting point. PE_i will generate a right token and a left token if a starting point is detected. Each token contains the contour type and the starting point location information. The right token is passed down to PE_{i+1} and the left token is kept in the current PE_i and the contour tracing starts in the next cycle. The point in PE_0 at the cycle 5 is an OCSP, while the point in PE_4 at cycle 12 is an ICSP as depicted in Fig. 4.

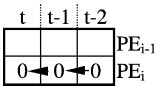
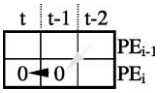
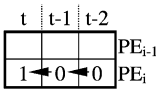
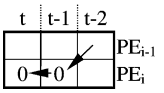
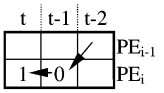
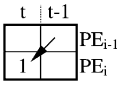
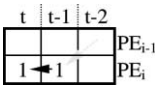
Case 2. PE_i contains (or receives) a token. The token is held in PE_i if the current pixel is black (white) and the contour type is outer-left (inner-left) or inner-right (outer-right). Otherwise, the token is passed down to PE_{i+1} in the next cycle. This situation is exemplified in Fig. 4, where the arrows indicate the motion of the edge tokens.

Case 3. PE_i contains a token and receives another token from PE_{i-1}. The situation corresponds to the case in which two tokens do not belong to the same contour if the current pixel is a transition pixel. PE_i should swap the two tokens and forward the old token it is holding to PE_{i+1}. Thus, PE_i will hold the token it has just received from PE_{i-1}. An ending point is detected and two tokens in the PE_i will be deleted and the contour tracings ended if the current pixel is not a transition pixel. The point in PE_{12} at cycle 27 (Fig. 4) is an ending point where an outer contour token and an inner contour token meet and are deleted after merging.

2.4. Code generation rules

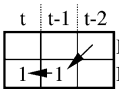
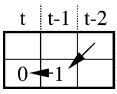
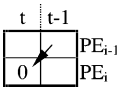
The chain code must be generated to represent the contour shape during the contour tracing. Each chain code reflects a specific spatial connection between the two transition pixels in the non-skewed image. The spatial connections also exist in the skewed image. The connection relations in eight possibilities after analysis are summarized in Table 1.

Table 1
The generation rules of a chain code in PE_i

Generated chain code	The edge token in the time–space diagram	Condition description
0		*Outer contour* 1. $PE_i(t)$ is a white pixel. 2. $PE_i(t)$ received an OR (outer-right) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an OR edge token from $PE_i(t-2)$.
		*Inner contour* 1. $PE_i(t)$ is a white pixel. 2. $PE_i(t)$ received an IL (inner-left) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ did not receive an IL edge token from $PE_{i-1}(t-2)$.
1		*Outer contour* 1. $PE_i(t)$ is a black pixel. 2. $PE_i(t)$ received an OR edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an OR edge token from $PE_i(t-2)$.
		*Inner contour* 1. $PE_i(t)$ is a white pixel. 2. $PE_i(t)$ received an IL edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an IL edge token from $PE_{i-1}(t-2)$.
2		1. $PE_i(t)$ is a black pixel. 2. $PE_i(t)$ received an OR (or IL) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an OR (or IL) edge token from $PE_{i-1}(t-2)$.
3		1. $PE_i(t)$ is a black pixel. 2. $PE_i(t)$ received an OR (or IL) edge token from $PE_{i-1}(t-1)$.
4		1. $PE_i(t)$ is a black pixel. 2. $PE_i(t)$ received an OL (outer-left) or IR (inner-right) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ did not receive an OL (or IR) edge token from $PE_{i-1}(t-2)$.

(*continued on next page*)

Table 1 (*continued*)

Generated chain code	The edge token in the time–space diagram	Condition description
5		1. $PE_i(t)$ is a black pixel. 2. $PE_i(t)$ received an OL (or IR) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an OL (or IR) edge token from $PE_{i-1}(t-2)$.
6		1. $PE_i(t)$ is a white pixel. 2. $PE_i(t)$ received an OL (or IR) edge token from $PE_i(t-1)$. 3. $PE_i(t-1)$ received an OL (or IR) edge token from $PE_{i-1}(t-2)$.
7		1. $PE_i(t)$ is a white pixel. 2. $PE_i(t)$ received an OL (or IR) edge token from $PE_{i-1}(t-1)$.

PE_i generates a chain code based on the information of the edge tokens received from itself and the PE_{i-1}. The relationships between the chain codes and the information of the edge tokens are also represented in Table 1. Each row in Table 1 stands for the generation rule of a specific kind of chain code. The generation rule specifies relations among PE's, clock cycles, pixel values, and edge tokens that can be illustrated in a time–space diagram. For example, when the PE_i receives three white pixels during the cycles from $t-2, t-1$, to t, and holds the same outer contour tokens during cycles $t-1$ and t, it generates a chain code "0" at cycle t for this outer contour. A complete illustration of the chain code generated for the pattern in Fig. 1 is demonstrated in Fig. 5.

There are two exceptions to the code generation rules presented in Table 1. The first exception is associated with the inner contour starting point because

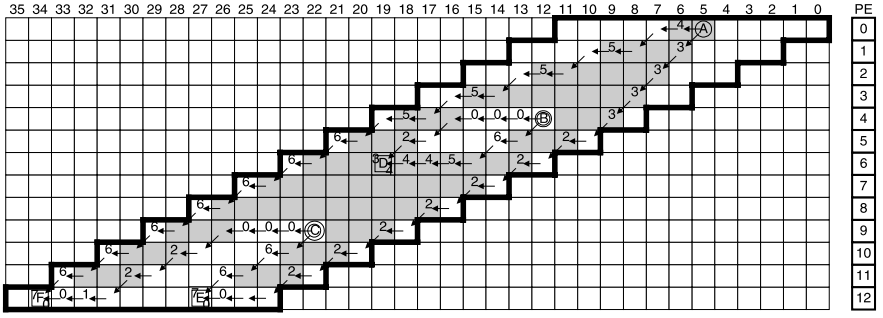


Fig. 5. The space–time diagram illustration of generated locations of chain codes.

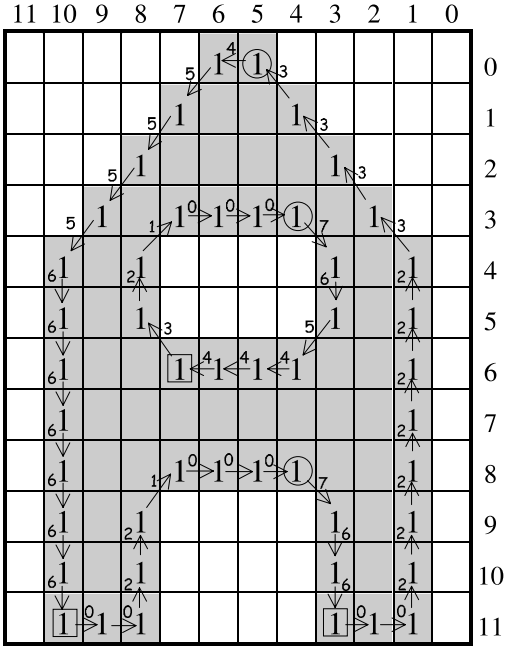


Fig. 6. The chain codes of the given pattern.

the inner contour tracing must be performed on the white transition pixels, whereas the actual inner contour tracing is finished at the black transition pixels. These two tracings differ at the two ends of the inner white row containing an inner contour starting point. Codes "1" and "7" are missing at the above two ends. These situations occur in PE_9 at cycles 22 and 26 as in Fig. 5. The correct chain code of the pattern in Fig. 1 is displayed in Fig. 6. The second exception is associated with the white ending points. In this case codes "1" and/or "7" are excessive codes. There situation occur in PE_{12} at cycles 27 and 34 as in Fig. 5.

3. Chain collection

Chain codes are outputted by each PE in a sequential timing sequence and stored in the image buffer after the chain code generation procedure is finished. The edge token label (i.e., the coordinates of the starting point) is employed to collect the chain codes belonging to the same contour segments. The chain codes generated with the same edge token label will be stored in the same location in the memory buffer of the image buffer. The chain codes belonging to contour segments $a, b, c, \ldots$, to f of pattern A are illustrated in Fig. 7(a).

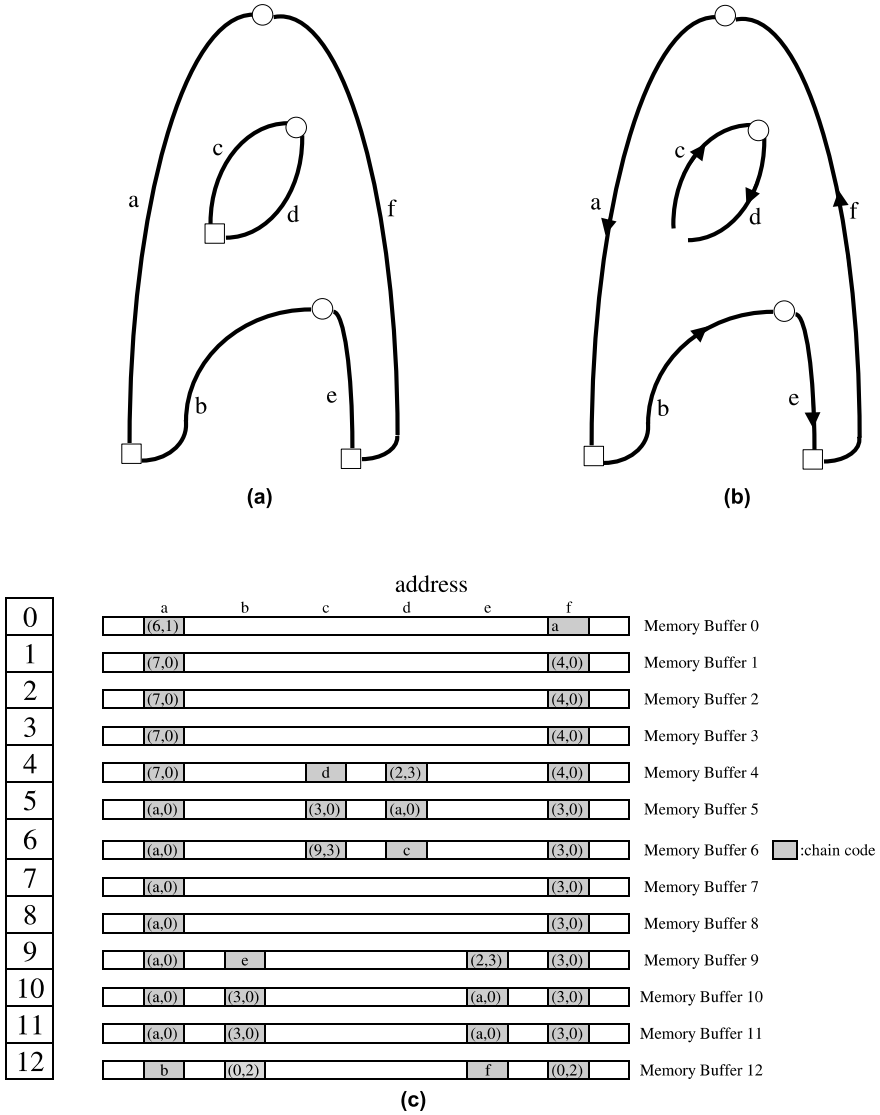


Fig. 7. (a) The contour structure of the pattern "A". (b) The linked contour structure. (c) The memory storage layout of the chain codes belonging to the same contour segment.

The PE for each contour segment may produce more than two chain codes, but only one location in the memory buffer can save these chain codes. Only thirteen kinds of the chain code sequences can be generated by the PE according to the spatial connection relationship between adjacent contour pixels. The chain code patterns and encoded chain code are summarized in Table 2.

Table 2
The chain code patterns and encoded codes

Chain code patterns	Encoded code
$\overline{0}$	$(0,n)$
$1+\overline{0}$	$(1,n)$
$1+\overline{0}+7$	$(2,n)$
2	$(3,0)$
$3+\overline{4}$	$(4,n)$
$3+\overline{4}+5+7$	$(5,n)$
$\overline{4}$	$(6,n)$
$\overline{4}+5$	$(7,n)$
$\overline{4}+5+7$	$(8,n)$
$5+\overline{4}+3$	$(9,n)$
6	$(a,0)$
$\overline{6}+7$	(b,n)
$7+\overline{0}$	(c,n)

The symbol "$-$" in the head of the chain code symbolizes the code that may be cycled. The code n represents the repeated number of the chain code with a symbol "$-$" in the head. Therefore, each PE collects all the chain codes that it created and encodes these chain codes into a 4-bit code with a repeated number. The encoded chain codes and the memory storage layout of the pattern in Fig. 1 are depicted in Fig. 7(c).

4. Chain linking

Two contour segments can be merged to obtain the complete chain codes of the object if they have the same starting point or ending point. This situation is exemplified by the pointer from f to a in Fig. 7(b). Two edge tokens are adjacent when they meet and an ending point is detected. A pointer is created in the ending point location of one token that points to the other end location of the chain as exemplified by the pointer from a to b in Fig. 7(b).

The PE will utilize a memory location that belongs to one contour segment to save the encoded chain code and the other to hold the pointer that indicates the location of the adjacent contour segment because the chain codes generated in the contour starting point contains one sequence at most. The encoded chain code for an OCSP (or ICSP) will be saved in the left (right) contour segment and the pointer is saved in the address of the right (left) contour segment. For example, the PE_0 in the OCSP A splits the object contour into two contour segments and generates the encoded chain code $(6,1)$. The addresses assigned to the left contour segment and the right contour segment are a and f, respectively. Thus, the encoded chain code will be stored at the address a in the

local memory buffer 0 and the pointer with point value a will be saved at the address f in the memory buffer 0.

The same cases exemplified in the starting points will also occur in the ending points. The encoded chain code will be saved in the address of the image buffer when two outer contour segments (or one outer contour segment with one inner contour segment) meet at an ending point. The pointer will be stored in the address of the other contour segment, whereas the opposite arrangement will occur if two joined contour segments are the inner contour segments.

A linked list with predecessor/successor relations in the memory is derived after the above manipulation is performed. The closed chain of the object can be extracted from a starting point of the contour segment and a top-down trace can be created at which the chain code has been stored in the same address among the memory buffers until the ending point appears. The chain will jump to the ending point of the next chain belonging to an adjacent contour segment based on the pointer in the former chain. A bottom-up trace is then made for the chain of the contour segment until the starting point arrives, after which the chain will jump to the next starting point of an adjacent contour segment. This sequential process will repeat until the chain is closed.

5. Performance evaluation

The time complexity of the chain code generation algorithm can be easily adopting via the space–time diagram of an input image. If the input data rate is one pixel per cycle, all the chain codes will be obtained in $3N$ cycles where

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

Therefore, it meets the requirements for real-time applications since it is of order $\mathrm{O}(N)$ and independent of the image content.

6. The partitioning method

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

Assume an image is $N \times N$ size with a linear array that has M PEs. Let $N \geqslant M$. The image must first be divided into several subimages, $m = \lceil (N+1)/M \rceil$ each with M rows or less. This row partitioning starts from top to bottom (refer to Fig. 9(a)), hence the subimage S_j includes the part of the image from row $M \times j$ to row $M \times (j+1) - 1$, where $j = 0, 1, \ldots, \lceil (N+1)/M \rceil - 2$ and S_{m-1}, S_{m-1} includes the part of the image from row $M \times (m-1)$ to row N. These subimages are rendered in 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 of PE_0 (refer to Fig. 8).

According to the N and M values, the execution model is as follows:

Case 1: $M < N + 1 \leqslant 2M$

Although the element PE_0 executes the first row of the subimage S_1, it must wait until the corresponding output of PE_{M-1} belonging to subimage S_0 is computed. Therefore, the subimage S_1 must be delayed $2M - N$ clock cycles before it is inputted to port of PE_0. The programming delay line will be set to zero. The time–space diagram is illustrated in Fig. 9(b), and the total execution time is $3N$. The chain code of the image of size $N \times N$ can be obtained via

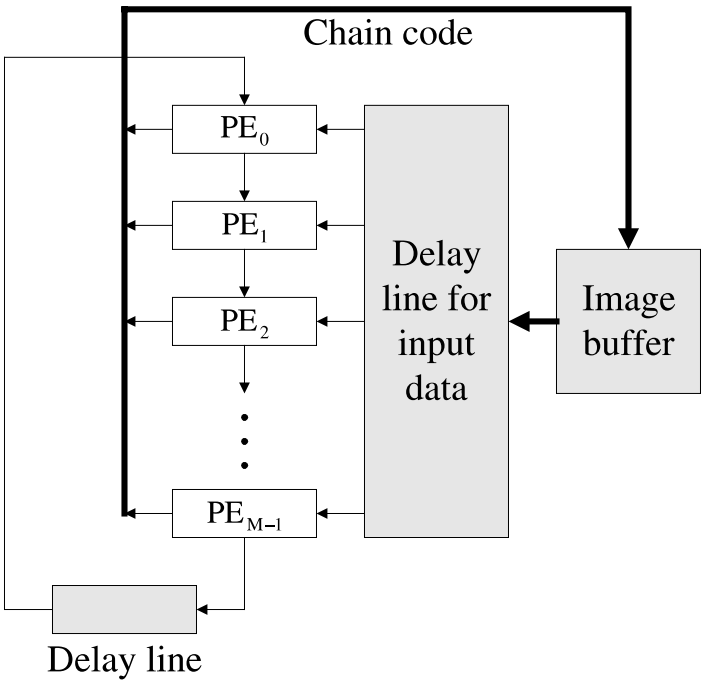
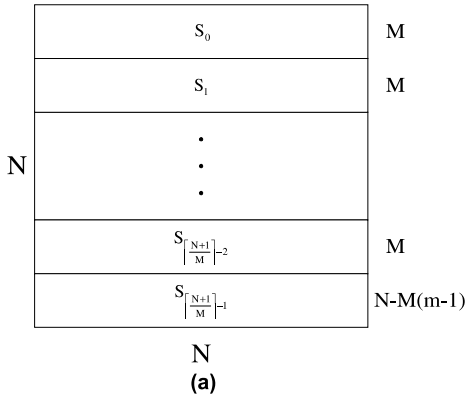
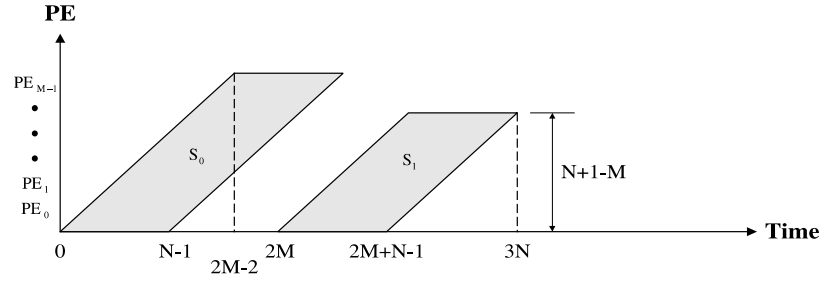


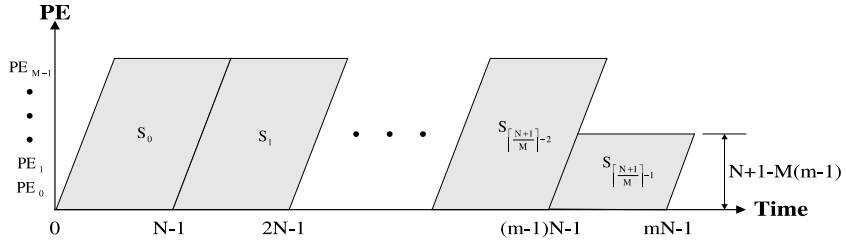
Fig. 8. The linear array architecture for executing the partitioned image.



(a)



(b)



(c)

Fig. 9. The time–space diagram of the chain code generation for the case $N+1>M$: (a) an image is divided into subimages; (b) the diagram for $M<N+1\leqslant 2M$; (c) the diagram for $N+1\geqslant 2M$.

$\lceil (N+1)/2 \rceil$ PEs, and the total execution time is the same if we employ $N+1$ PEs.

Case 2: $N+1\geqslant 2M$

The output of PE_{M-1} must be inputted through an $N-2M$ delayed line since the output results of the last row of S_j produced from PE_{M-1} are $N-2M$

clocks ahead of the input data of the first row of S_{j+1} to PE_0. 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 illustrated in Fig. 9(c), and the total execution time is $Nm + 2[N - M(m-1)]$.

7. Conclusion

This work has described parallel algorithm and its hardware architecture for generating a chain code. The proposed architecture can obtain the chain codes of the objects in an $N \times N$ binary image within $3N$ cycles employing an architecture of $\lceil (N+1)/2 \rceil$ PEs. The developed architecture also has commendable modularity, expandability, data flow regularity, and PE simplicity. It is also suitable for VLSI implementation because it has simple and modular properties. We also consider the partition problem when the image size is larger than the size of the PE array.

References

[1] J. Koplpwitz, J. Deleone, Hierarchical representation of chain-encoded binary image contours, Computer Vision and Image Understanding 63 (2) (1996) 344–352.
[2] H.F. Li, R. Jayakumar, M. Youssef, Parallel algorithms for recognizing handwritten characters using shape features, Pattern Recognition 22 (6) (1989) 641–652.
[3] L.W. Chang, K.L. Leu, A fast algorithm for restoration of images based on chain codes description and its applications, Computer Vision Graphics and Image Processing 50 (1990) 296–307.
[4] Z. Cai, Restoration of binary images using contour direction chain codes description, Computer Vision Graphics and Image Processing 41 (1988) 101–106.
[5] H. Samet, The quadtree and related hierarchical data structures, Computing Surveys 16 (1984) 187–260.
[6] L. Dorst, A.W.M. Smeuders, Length estimated for digitized contours, Computer Graphics and Image Processing 40 (1987) 317–333.
[7] J. Yuan, C.Y. Suen, An optimal O(n) algorithm for identifying line segments from a sequence of chain codes, Pattern Recognition 28 (5) (1995) 635–646.
[8] A. Rosedfeld, Algorithms for image/vector conversion, Computer Graphics 12 (1978) 135–139.
[9] I. Sobel, Neighborhood coding of binary images for fast contour following and general binary array processing, Computer Graphics and Image Processing 8 (1978) 127–135.
[10] P. Meer, C.A. Sher, A. Rosenfeld, The chain pyramid: hierarchical contour processing, IEEE Transactions on Pattern Analysis and Machine Intelligence 12 (1990) 363–376.
[11] F.Y. Shih, W.T. Wong, A new single-pass algorithm for extracting the mid-crack codes of multiple regions, Journal of Visual Communication and Image Representation 3 (3) (1992) 217–224.
[12] R.A. Schmidt, Fast generation of chain-code image descriptors, in: Proceedings of the IEEE International Conference on Robotics and Automation, St Louis, MO, 1985, pp. 868–872.
[13] R.T. Cederburg, Chain-link coding and segmentation for raster scan devices, Computer Graphics and Image Processing 10 (1979) 224–234.

[14] I. Chakravarty, A single-pass chain generating algorithm for region boundaries, Computer Graphics and Image Processing 15 (1981) 182–193.
[15] P. Zingaretti, M. Gasparroni, L. Vecci, Fast chain coding of region boundaries, IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (4) (1998) 407–415.
[16] S.D. Kim, J.H. Lee, J.K. Kim, A new chain-coding algorithm for binary images using run-length codes, Computer Vision Graphics and Image Processing 41 (1988) 114–128.