ABSTRACT

PARALLELIZATION FOR IMAGE PROCESSING ALGORITHMS BASED ON CHAIN AND MID-CRACK CODES

by
Wai-Tak Wong

Freeman chain code is a widely-used description for a contour image. Another mid-crack code algorithm was proposed as a more precise method for image representation. We have developed a coding algorithm which is suitable to generate either chain code description or mid-crack code description by switching between two different tables. Since there is a strong urge to use parallel processing in image related problems, a parallel coding algorithm is implemented. This algorithm is developed on a *pyramid* architecture and a *N cube* architecture. Using link-list data structure and neighbor identification, the algorithm gains efficiency because no sorting or neighborhood pairing is needed.

In this dissertation, the local symmetry deficiency (LSD) computation to calculate the local k-symmetry is embedded in the coding algorithm. Therefore, we can finish the code extraction and the *LSD* computation in one pass. The embedding process is not limited to the k-symmetry algorithm and has the capability of parallelism.

An adaptive quadtree to chain code conversion algorithm is also presented. This algorithm is designed for constructing the chain codes of the resulting quadtree from the boolean operation of two quadtrees by using the chain codes of the original one. The algorithm has the parallelism and is ready to be implemented on a pyramid architecture.

Our parallel processing approach can be viewed as a parallelization paradigm - a template to embed image processing algorithms in the chain coding process and to implement them in a parallel approach.

PARALLELIZATION FOR IMAGE PROCESSING ALGORITHMS BASED ON CHAIN AND MID-CRACK CODES

by
Wai-Tak Wong

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy

Department of Computer and Information Science

January 1999

APPROVAL PAGE

PARALLELIZATION FOR IMAGE PROCESSING ALGORITHMS BASED ON CHAIN AND MID-CRACK CODES

Wai-Tak Wong

12/9/98

Dr. Frank Y. Shih, Dissertation Advisor — Date
Professor of Computer and Information Science, NJIT

12/10/98

Dr. James A. M. McHugh, Committee Member — Date
Chairperson and Professor of Computer and Information Science, NJIT

12/9/98

Dr. Daochaun D. Hung, Committee Member — Date
Associate Professor of Computer and Information Science, NJIT

12/9/98

Dr. Pengcheng Shi, Committee Member — Date
Assistant Professor of Computer and Information Science, NJIT

12/9/98

Dr. Edwin Hou, Committee Member — Date
Associate Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author:	Wai-Tak Wong
Degree:	Doctor of Philosophy
Date:	January 1999

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1999

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1992

- Bachelor of Science in Chemical Engineering,
  Nation Taiwan University, Taipei, Taiwan, Republic of China, 1986

Major: Computer and Information Science

Presentations and Publications:

Shih, F. Y., and W.-T. Wong. "A new single-pass algorithm for extracting the mid-crack codes of multiple regions." *Journal of Visual Commun. and Image Rep.*, vol. 3, no. 1, pp.217-224, March 1992.

Shih, F. Y., and W.-T. Wong. "Reconstruction of binary and gray-scale images from mid-crack code descriptions." *Journal of Visual Commun. and Image Rep.*, vol. 4, no. 2, pp. 121-129, June 1993.

Shih, F. Y., and W.-T. Wong. "An improved fast algorithm for the restoration of images based on chain codes description." *CVGIP: Graphical Models and Image Process.*, vol. 56, no. 4, pp. 348-351, July 1994.

Shih, F. Y., and W.-T. Wong. "Fully parallel thinning with tolerance to boundary noise." *Pattern Recognition*, vol. 27, no. 12, pp. 1677-1695, 1994.

Wong, W.-T., Y.-L. Chen, and F. Y. Shih. "A fully parallel algorithm for the extraction of chain and mid-crack codes of multiple contours." Conf. Proc. of Inter. Comput. Symp., HsinChu, Taiwan, R.O.C., pp. 565-570, 1994.

BIOGRAPHICAL SKETCH
(Continued)

Shih, F. Y., and W.-T. Wong. "A new safe-point thinning algorithm based on the mid-crack code tracing." *IEEE Trans. Syst. Man Cybern. (T-SMC)*, vol. 25, no. 5, pp. 370-378, February 1995.

Shih, F. Y. and W.-T. Wong. "A one-pass algorithm for local symmetry of contours from chain code." 1st *Inter. Workshop on Comput. Vision, Pattern Recogn. and Image Process.*, 1998.

Shih, F. Y. and W.-T. Wong. "A one-pass algorithm for local symmetry of contours from chain code." accepted by *Pattern Recognition*, 1998.

This thesis is dedicated to
our almighty GOD and Jesus Christ

ACKNOWLEDGMENT

First, I would like to thank our heavenly Father and our Lord Jesus Christ. I know it is all His grace.

Next, I would like to thank my wife for her unconditional love for being a mother of three young children, a housewife and giving all of her support to our family.

I would like to express my appreciation to my advisor Prof. Frank Y. Shih, without his research guidance, I think I could not accomplish my Ph. D. goals.

Special thanks are given to Prof. James A. M. McHugh, Prof. Daochuan Hung, Prof. Pengcheng Shi and Prof. Hou for serving as my committee members.

TABLE OF CONTENTS

Chapter Page

1 INTRODUCTION 1

1.1 Contour Representation by Chain/Crack/Mid-Crack Coding 1

1.2 A Single Pass Mid-Crack Coding Algorithm 4

1.2.1 Codes Generation Look-up Table and Move Table 4

1.2.2 Mid-Crack Coding Algorithm in Raster Scan Fashion 9

1.3 Applications for Chain Coding and Mid-Crack Coding 12

1.4 Organization of this Dissertation 13

2 A PARALLEL CHAIN/MID-CRACK CODING ALGORITHM ON PYRAMID ARCHITECTURE 15

2.1 Introduction 15

2.2 The Parallel Codes Extraction Algorithm 16

2.3 The Coding Stage 19

2.3.1 The similarities between Chain Coding and Mid-Crack Coding 19

2.3.2 Codes Generation Look-up Table and Move Table 19

2.3.3 Coding Mechanism 20

2.3.4 The Procedure of Coding Stage 22

2.4 The Merging Stage 23

2.5 Determination of Contour Types 25

2.6 Experimental Results 26

2.7 Complexity Evaluation 26

2.8 Summary 27

3 A PARALLEL CHAIN/MID-CRACK CODING ALGORITHM ON N CUBE ARCHITECTURE 31

3.1 Introduction 31

3.2 The New Parallel Algorithm for Chain and Mid-Crack Coding 32

3.2.1 The Coding Stage 32

TABLE OF CONTENTS
(Continued)

Chapter Page

3.2.2 The Merging Stage 40

3.3 Experimental Results 41

3.4 Time Complexity Evaluation 46

3.5 Summary 46

4 AN ONE-PASS ALGORITHM FOR LOCAL SYMMETRY OF CONTOURS FROM CHAIN CODES 47

4.1 Introduction 47

4.2 Proposed One-pass Algorithm for Chain Coding 50

4.3 The One-pass Algorithm for Local Symmetry 51

4.4 Parallelism of the One-pass K-Symmetry Algorithm 55

4.5 Experimental Results 56

4.6 Summary 58

5 AN ADAPTIVE CONVERSION ALGORITHM FROM QUADTREE TO CHAIN CODES 59

5.1 Introduction 59

5.2 Traversal Algorithm for Pointer-Based Quadtree 62

5.3 Chain Coding for Quadtree Traversal Algorithm 67

5.4 Adaptive Level Base Chain Coding Retrieval 73

5.5 Experimental Results 74

5.6 Analysis 75

5.7 Summary 76

6 SUMMARY AND FUTURE RESEARCH 77

6.1 Contribution and Summary of this Dissertation 77

6.2 Future Research 78

REFERENCES 79

LIST OF TABLES

Table Page

1.1 The Look-Up Table of Mid-Crack Coding 7

1.2 The Move Table of Mid-Crack Coding 8

2.1 The Look-Up Table of Chain Coding 21

2.2 The Move Table of Chain Coding 22

4.1 The result of external boundary in Figure 4.3 54

5.1 The tracking of the traversal algorithm for the 8-neigbors of the SE child of the most northwest quadrant in a 64×64 image 69

5.2 The generation rules of the pseudo parent type of a node 69

5.3 Classification of shared boundary ocurrance for the Head and Tail of a codelink 70

5.4 The particular cases of the two Heads generated on a pixel will not be in order 70

5.5 The result of the merging process for each node in Figure 5.1 75

LIST OF FIGURES

Figure Page

1.1 Silhouette with the inside and outside chain coded contours and the mid-crack coded contour 3

1.2 Freeman chain code and the mid-crack code on the vertical and the horizontal cracks 3

1.3 Examples of five types of code-links initialized from the mid-cracks around the central pixel 6

1.4 Headcode (dashed line) determination 6

1.5 Index value calculation in the window operation 8

1.6 The linked list in a connectedness structure array 9

1.7 The system flow-chart 11

2.1 An example of a pyramid architecture 17

2.2 An example of similarities between mid-crack code and chain code. Solid arrow denotes codelinks and dotted arrow denotes headcodes 20

2.3 Two cases of chain and mid-crack codes while the internal contours is closed 25

2.4 The input image with four objects and five contours 28

2.5 The result of Figure 2.4 by applying our algorithm with chain codes 29

2.6 The result of Figure 2.4 by applying our algorithm with mid-crack codes 30

3.1 (a) A 3 cube architecture 33

3.1 (b) A 3 cube recirculating network 33

3.2 The parallel chain coding algorithm in N cube architecture 34

3.3 The link-list structures for a chain set 35

3.4 The order of the codelink generation 36

3.5 The patterns that needs extra swapping procedure 37

3.6 An input image with three chains 37

3.7 (a) The coding result for the first two rows of the Figure 3.6 38

3.7 (b) The coding result for the second two rows of the Figure 3.6 39

LIST OF FIGURES
(Continued)

Figure Page

3.7 (c) The coding result for the fifth rows of the Figure 3.6 40

3.8 (a) The merging result for the first two rows of the Figure 3.6 42

3.8 (b) The merging result for the second two rows of the Figure 3.6 42

3.8 (c) The merging result for the row 0 to row 3 of the Figure 3.6 43

3.8 (d) The merging result of the Figure 3.6 43

3.9 (a) The input image which lists 3 digits 0, 9 and 6 horizontally 44

3.9 (b) The resulting chains 44

3.10 (a) The input image which lists 3 digits 0, 9 and 6 vertically 45

3.10 (b) The resulting chains 45

4.1 (a) Digital curve C 49

4.1 (b) Curve C^* obtained from reflecting the local region C^+ of length k after p_i in curve C 49

4.1 (c) Curve C' obtained from rotating curve C^* to align it with the local region C^- of length k before P_i in curve C 49

4.2 The link-list structures for a chain set 50

4.3 An input image with two chains 52

4.4 The coding result for Figure 4.3 53

4.5 An input image contains two digits, 9 and 6 56

4.6 Results of LSD computation for the external boundaries in Figure 4.5 57

5.1 An image and its quadtree representation 61

5.2 The link-list data structure for a chain set 71

5.3 An exmaple to illustrate the classification of the location for the Head and Tail 72

CHAPTER 1

INTRODUCTION

1.1 Contour Representation by Chain/Crack/Mid-Crack Coding

When a 2-dimensional digital image of a real scene is processed by a computer, it is often transformed into one or more binary regions by means of various image segmentation techniques. For further analysis, features are extracted and classified. In consideration of the computational time and memory size, it is desirable to convert the binary regions/images into a specific form which is more efficient and convenient to be processed. For this purpose, contour representation is utilized. The contour representation of a binary image is determined by specifying a starting point and a sequence of moves around the borders of each region. Current methods of contour tracing are based on Freeman chain code or crack code [1,2,3]. Some techniques can extract the contour image directly from a grayscale image instead of a binary image. They are based on the similar concept on edge operator.

Chain code and crack code are the popular coding techniques for binary images. They are developed not to produce efficient codings in the sense of minimizing the number of code bits required to describe a boundary, but rather to make certain manipulation operations convenient. The chain code moves along a sequence of the center of border points, while the crack code moves along a sequence of "cracks" between two adjacent border points. Typically, they are based on the 4- or 8-connectivity of the segments, where the direction of each segment is encoded by using a numbering scheme, such as 3-bit numbers $\{i \mid i = 0, 1, \cdots, 7\}$ denoting an angle of $45i°$ counter-clockwise from the positive x-axis for a chain code, or 2-bit numbers $\{i \mid i = 0, 1, 2, 3\}$ denoting an angle of $90i°$ for a crack code. The elementary idea of the chain or crack coding algorithm is to trace the border-pixels or cracks and sequentially generate codes by considering the neighborhood adjacency relationship. The chain and the crack codes can be viewed as a connected sequence of straight line segments with specified lengths and directions. An obvious disadvantage of the chain code is observed when we use it to compute the area and perimeter of an object. Referring to Figure 1.1, the inside chain code appears to underestimate the area and perimeter while the outside chain code overestimates them. The disadvantages in the crack code are that much more codes are generated and the perimeter is much overestimated. The mid-crack code [4], located in between, should make a more accurate computation of the geometric features.

The mid-crack code is a variation and an improvement of the traditional tracing methods between the chain code and the crack code. In contrast to Freeman chain code, which moves along the center of pixels, the mid-crack code moves along the edge mid-point of a pixel producing codes of links. For the horizontal and vertical moves, the length of a move is 1, and for diagonal moves, it is $\sqrt{2}/2$. If the crack is located in between two adjacent object pixels in the vertical direction, it is said to be on a vertical crack. Similarly, if the crack is located in between two adjacent object pixels in the horizontal direction, it is said to be on a horizontal crack.

Figure 1.2 shows the Freeman chain code and the mid-crack code on the vertical and the horizontal cracks. There are two restrictions on the moves in the mid-crack code. If a move is from the vertical crack, the codes 0 and 4 are not allowed. Similarly, the codes 2 and 6 are not allowed in moves from the horizontal crack. The experimental verification of the mid-crack code in area and perimeter computation is shown in [4] where the mean perimeter error value is -0.074% and the mean area error is -0.006%. Therefore the mid-crack code is a desirable alternative method in contour tracing with its benefit in accuracy. A disadvantage of the mid-crack code is that it is always longer than the Freeman chain code. A conversion algorithm between these two code sequence is described in [4] to complement the defects as a compression process.

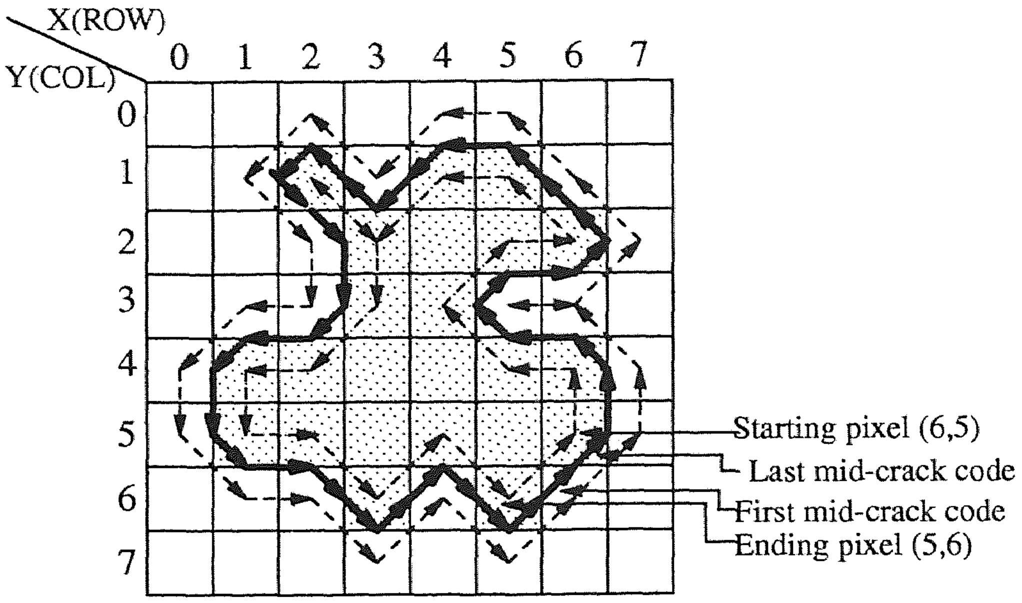


Figure 1.1 Silhouette with the inside and outside chain coded contours (dashed lines) and the mid-crack coded contour (solid line)

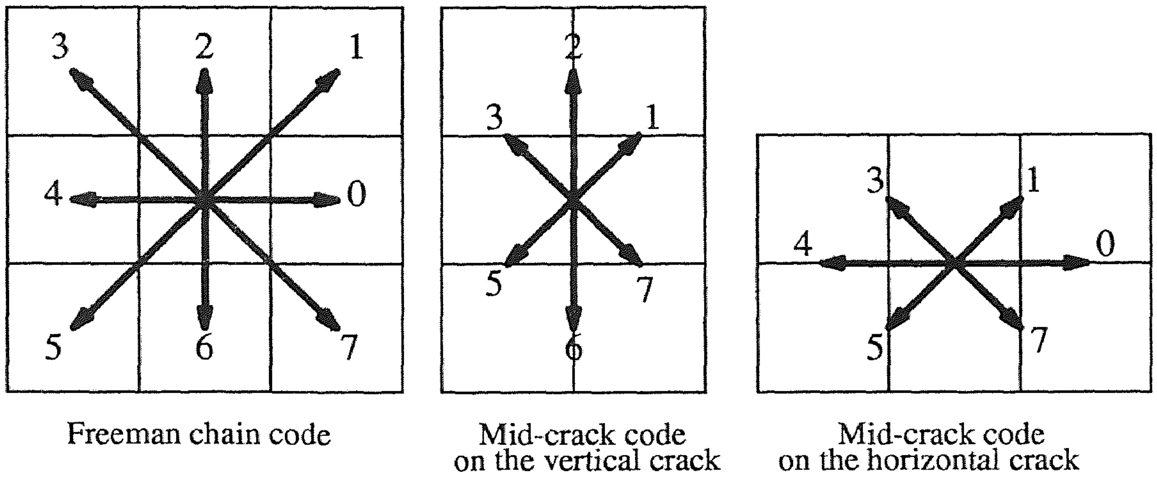

3 2 1 4 0 5 6 7
Freeman chain code
2 3 1 5 7 6
Mid-crack code on the vertical crack
3 1 4 0 5 7
Mid-crack code on the horizontal crack

Figure 1.2 Freeman chain code and the mid-crack code on the vertical and the horizontal cracks

1.2 A Single Pass Mid-Crack Coding Algorithm

This dissertation is based on our previous work, a single pass mid-crack coding algorithm [7]. That algorithm uses a 3×3 window for the codes extraction in a raster-scan fashion. It only requires a single row-by-row scan to generate all the code sequences for a complex binary image which is composed of many objects. Also, the algorithm has an advantage in detecting the spatial relationship between objects. Besides, the perimeter computation and area adjustment can be performed in parallel. Before we start to describe the algorithm, the terminology is introduced first for the better understanding the content of this chapter and later.

Terminology

Codelink	a link of the central pixel's connected codes in the 3×3 window
Boundary link	a link of connected codelinks for all border pixels being scanned. It is also called chain in later chapter
Thead	the first code of a link connected to the tail of the current boundary link or codelink
Headcode	the first code of the current boundary link or codelink

1.2.1 Codes Generation Look-up Table and Move Table

A set of 3×3 window masks containing every variety of mid-crack codelinks initialized from the mid-cracks around a central pixel is illustrated in Figure 1.3. We could summarize them into following five types of encoding based on the number of codelinks: no-code, one-code, two-code, three-code and four-code links. Each codelink is associated with a thead which is illustrated in Figure 1.4. According to the permutations, a look-up table is set up for code generation, as shown in Table 1.1. The mid-crack codes are based on 8-connectedness and counter-clockwise tracing for the external boundary and clockwise tracing for the internal boundary. The index value will be discussed next. The index value 0 reflecting a isolated pixel which is treated as a noise. The related information such as the total number of codelinks, all thead, and all codelinks can be obtained from drawing the mid-crack codes surrounding the central pixel, as illustrated in Figure 1.3.

A 3×3 window, which is incorporated with different weights at each element exploring the presence of eight neighboring locations, is shown in Figure 1.5 (a). If an object pixel occurs, the weighted window is convolved with the 3×3 neighborhood centered at that pixel. Assume that the binary image has the object pixel "1" and the background pixel "0." This convolution is performed to calculate the index value of the look-up table. An example of the window operation is illustrated in Figure 1.5 (b). The related information with respect to the mid-crack codes surrounding a pixel in Table 1.1 can be retrieved by the use of the index value. Then, a series of operations are applied to concatenate these individual codelinks to their suitable boundary links.

From a codelink, we can determine the relative move in column and row with respect to the current location. A move table listing the relative coordinates for all the moves, is shown in Table 1.2. For example, the code 0 indicates one-pixel move in x-axis (or column), and no move in y-axis (or row). The destination of moves acts as an important role as we search for a right link in the check-head or check-tail step which will be discussed in next section.

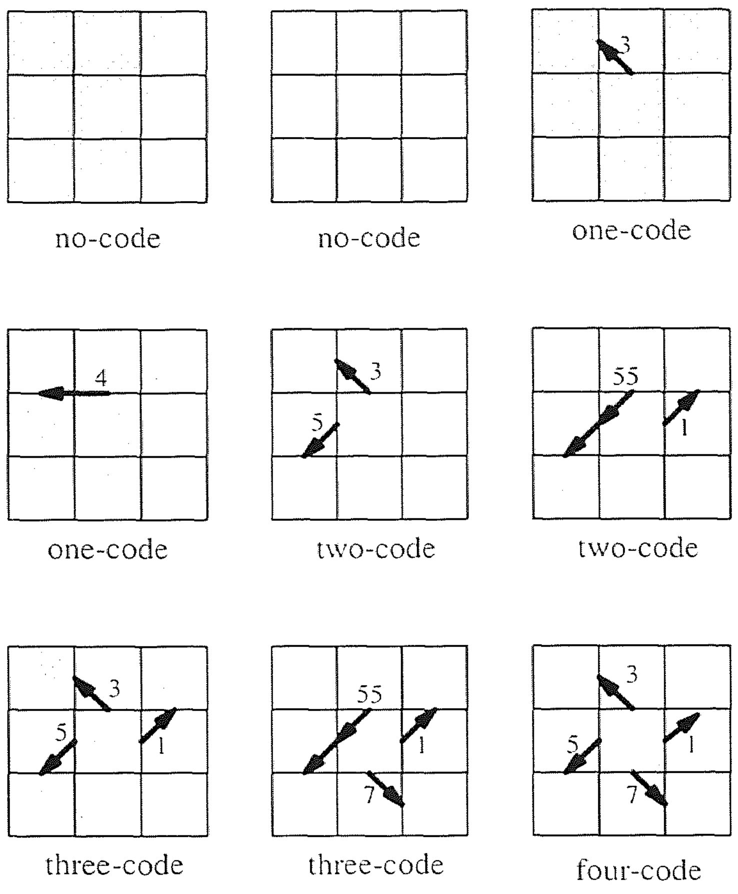


Figure 1.3 Examples of five types of code-links initialized from the mid-cracks around the central pixel

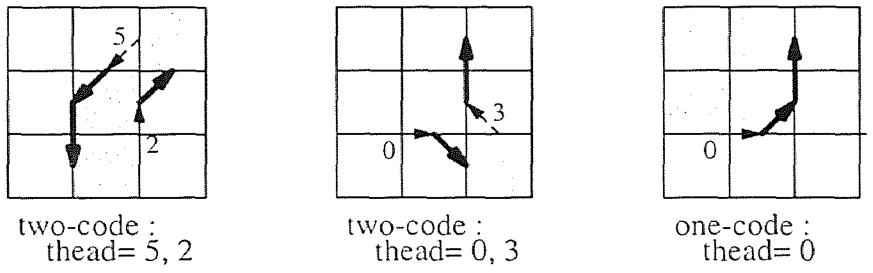


Figure 1.4 Thead (dashed line) determination

Table 1.1 The Look-Up Table of Mid-Crack Coding

Index	Number of Codelinks	Thead	Codelink
0	0	Nil	Nil
1	1	7	3317
2	1	6	217
3	1	7	217
4	1	5	1175
5	2	5,0	117, 3
...	...	...	...
130	2	6,3	77, 2
131	2	7,3	77, 2
132	2	5,3	775, 1
133	3	7,5,3	77, 3, 1
134	2	6,3	77, 1
...	...	...	...
251	0	Nil	Nil
252	1	5	4
253	1	5	3
254	0	Nil	Nil
255	0	Nil	Nil

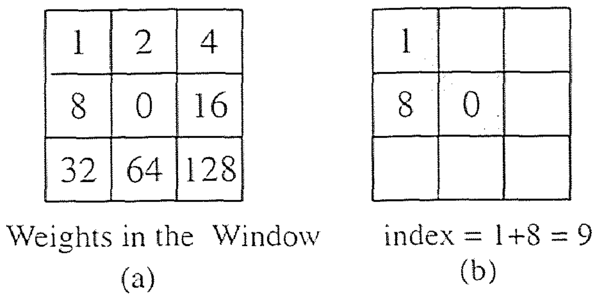


Figure 1.5 Index value calculation in the window operation

Table 1.2 The Move Table of Mid-Crack Coding

Move	Relative Coordinates	
	Column(dx)	Row(dy)
0	+1	0
1	+1	-1
2	0	-1
3	-1	-1
4	-1	0
5	-1	+1
6	0	+1
7	+1	+1

1.2.2 Mid-Crack Coding Algorithm in Raster Scan Fashion

When the first object pixel is fetched, we quickly obtain its codelinks simply by a look-up table. After that, we create a new linked list in the connectedness structure array shown in Figure 1.6. A list in the connectedness structure includes: head and tail coordinates, codelinks, thead, perimeter computation, and area adjustment. A list-typed data structure to store the temporary boundary links is employed in the representation. If the boundary link is connected with the codelink, we will not only increase the code sequence, but also change the coordinates of head and tail positions. Then in the process of scanning the next pixel, its codelinks are joined into the existing links by checking the conditions in the check-tail and check-head steps of the algorithm.

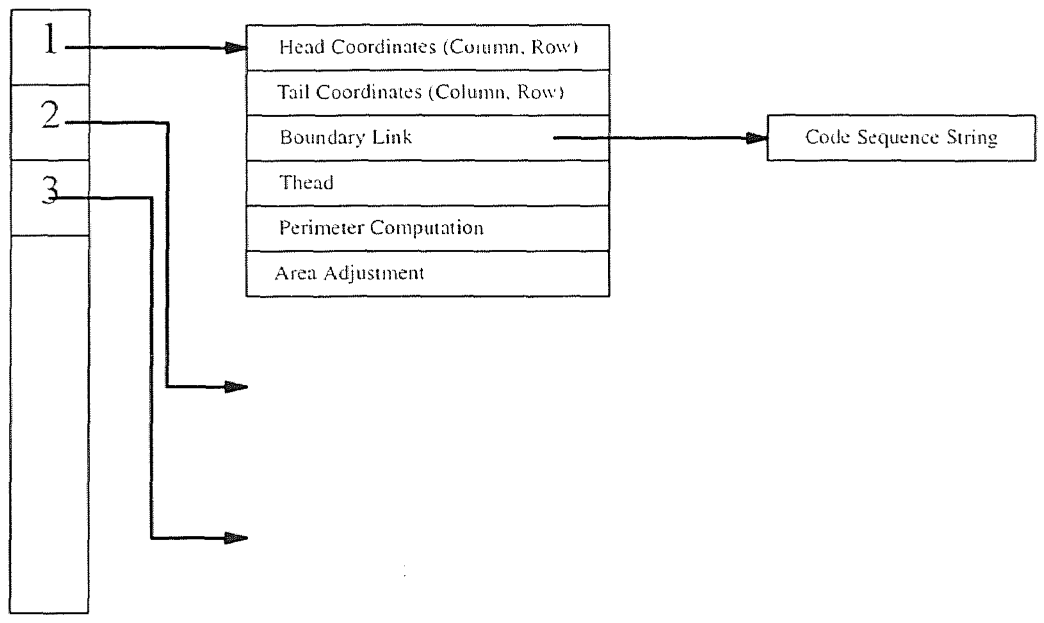


Figure 1.6 The linked list in a connectedness structure array

In the check-head step, we check the head coordinates and the headcode of the searched boundary link in the connectedness structure array and match with the tail coordinates and the thead of the codelink respectively. In the check-tail step, we check the tail coordinates and thead of the boundary link and match with the head coordinates and the headcode of the codelink. If the current codes have connectivity with the neighboring codes, there are two kinds of concatenation ways: head concatenation and tail concatenation. In the head concatenation case, we connect the codelink with the matched boundary link. the tail concatenation case, we connect the matched boundary link with the codelink.

If none of the links satisfies the condition, we create a new link in the connectedness structure array to store this code information. After the object pixels are scanned completely, each linked-list sequence in a block of the array represents the codes of an internal or external contour of an object. After the whole image is scanned, the boundary links of different objects exist in different blocks of the array. A system flow-chart is described in Figure 1.7 to assist the reader to understand the algorithm.

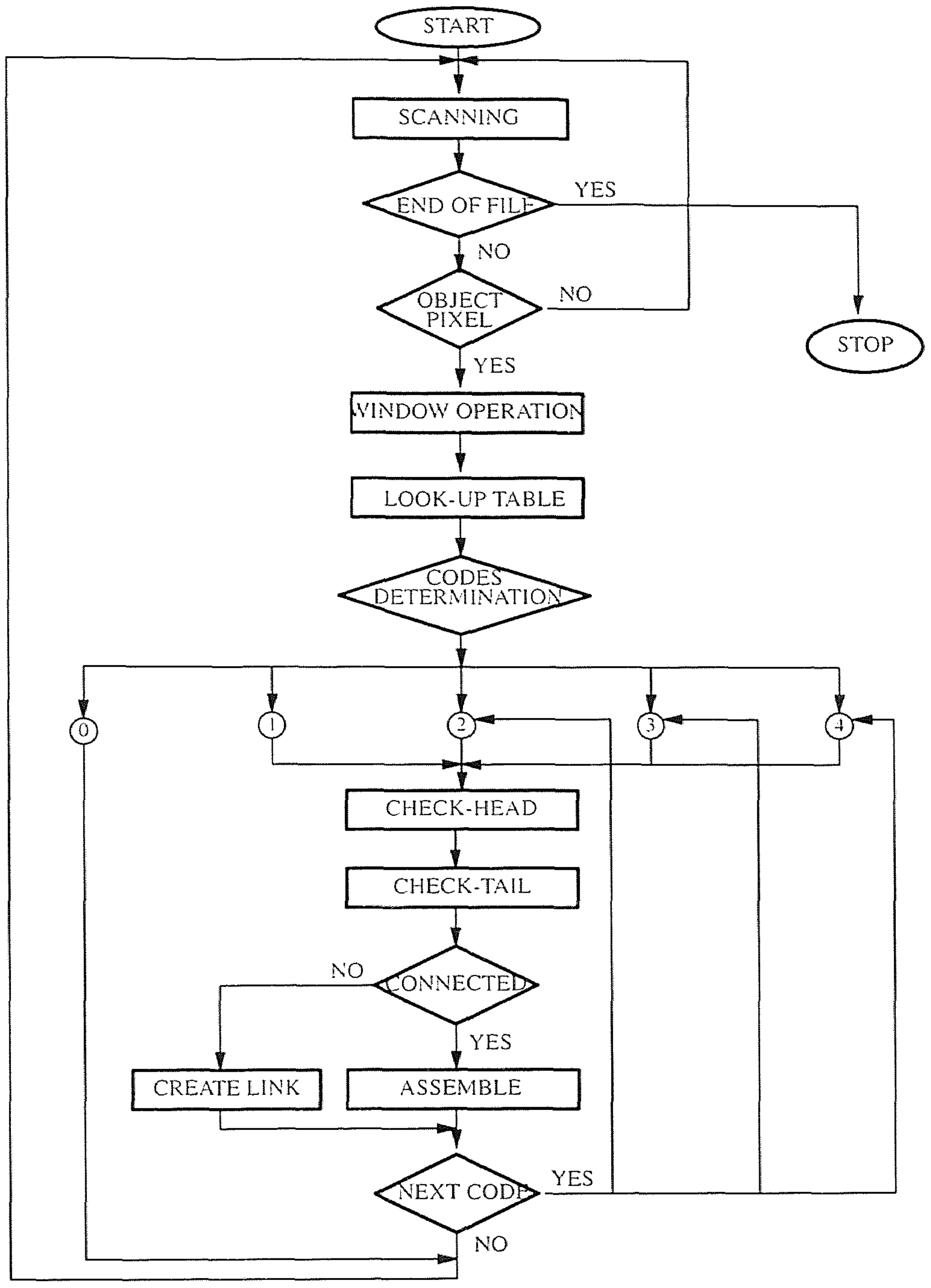


Figure 1.7 The system flow-chart

1.3 Applications for Chain Coding and Mid-Crack Coding

The applications for chain coding and mid-crack coding are in large varieties, from the computation of the geometric properties of an object to the skeletonization. Besides, they can be used for region filling, corner detection, local symmetry computation, line segment identification, etc.. The followings are the abstracts of our interested portions.

Chang and Leu [8] presented an algorithm to convert the chain codes description (boundary representation) to y-axis representation (region representation) [9]. Based on that, they derived the computation formulas of the point membership property, area, moments and centers, and eccentricity of an object. They also solved the problem for the intersection and union operations between two objects. A revision of that algorithm [8] was presented by Shih and Wong [10] as an improvement and correction. Shih and Wong [11] also presented a mid-crack codes version of that algorithm which was used as a method of region filling. This improved algorithm does not have the limitation such as (a) the boundary must be closed, (b) cannot handle the cases where the test line intersects the boundary tangentially, and (c) the closed boundary can not loop back on itself.

Koplowitz and Plante [12] defined a feature of the chain code links, the Straight Line Distance of a point, which is the maximum number of links that can produce a digital straight line centered about that point. The straight line distance is then used as an indicator of the curvature at a point on the chain coded curve. If the curvature is high enough, the point is considered as a corner point. Iñesta, Buendía, and Sarti [13] defined another measurement, named local symmetric deficiency, which evaluates the position of a local vicinity of any contour point. The lower this quantity is, the higher the symmetry is in the local region considered. This technique can also be used as a corner detector. Yuan and Suen [14] presented an optimal algorithm for identifying straight lines in chain codes description. The algorithm turns the complicated problem of computing the straightness of digital arcs into a simple task by constructing an passing area around the evaluating pixel and determining whether the straight line lies on it. The straight line is extended each time a new pixel is accepted until an unacceptable one is encountered.

Another application of chain codes and mid-crack codes is skeletonization or thinning. Thinning algorithms based on the contour generation method in chain codes were presented by Kwok [15], Vossepoel, Buys and Koelewijn [16], and Xu and Wang [17]. The essential idea is first to convert the input image into chain codes for each closed contour, and then to trace around the contour. If a boundary point is removed, the algorithm will generate a few new chain codes to replace the old one. The contour tracing will process each contour layer-by-layer iteratively until no further deletion occurs. Shih and Wong [18] presented a thinning algorithm which adopted the advantages of the simple safe-point testing [19] and the mid-crack code tracing [4]. This algorithm allows to thin multiple objects simultaneously. An improvement of this algorithm in parallel verison was also presented by Shih and Wong [20].

1.4 Organization of this Dissertation

The organization of this dissertation will be given in this section. The outline of this dissertation is as follows and the brief statement of each chapter is given later.

Chapter 1	Introduction.
Chapter 2	A fully parallel algorithm for the extraction of chain and mid-crack codes of multiple contours.
Chapter 3	A parallel chain and mid-crack coding algorithm on *N* cube architecture.
Chapter 4	A one-pass algorithm for local symmetry of contours from chain codes.
Chapter 5	An adaptive conversion algorithm from quadtree to chain codes.
Chapter 6	Summary and future research.

In Chapter 2, a fully parallel algorithm for chain and mid-crack codes extraction based on the table look-up approach is presented. By using a divide-and-conquer strategy, it is developed on a pyramid parallel architecture. The coding algorithm is composed of two stages, coding and merging. It is implemented on the parallel machine AP1000 with the parallel simulator casim. Experimental results demonstrate the correctness and the flexibility.

In Chapter 3, a chain and mid-crack coding algorithm using link-list data structure on *N cube* architecture is presented. This algorithm does not have any restrictions of existing algorithms. It is fast, flexible and extendable.

In Chapter 4, by adopting the coding technique in Chapter 3, we present a one-pass k-symmetry algorithm. We can finish the code extraction and the *LSD* computation in one pass. The algorithm is suitable for parallel implementation.

In Chapter 5, an adaptive algorithm is presented for converting a quadtree representation of an image to its chain code representation. This algorithm is adaptive because it is able to adjust the total number of the internal nodes to be stored and retrieved it later in the chain code construction for the new quadtree derived from the original one. This algorithm has the parallelism and ready to implement in a pyramid architecture for it is in a recursive form of calling four children of the current node.

Then, the summary of this dissertation is given and future research is stated in Chapter 6.

CHAPTER 2

A FULLY PARALLEL ALGORITHM FOR THE EXTRACTION OF CHAIN AND MID-CRACK CODES OF MULTIPLE CONTOURS

2.1 Introduction

From the literature review, we know that most chain coding algorithms are using the contour-tracing sequential approach or the raster-scan single-pass approach. The former traces the border pixels one-by-one and generates codes by considering neighborhood allocation, e.g. a mid-crack coding algorithm [4]. The latter runs in a raster-scan fashion to generate codes and then combines chains together, e.g. run-length coding to generate chain codes [21] and mid-crack codes [7].

Dinstein and Landau proposed a parallel contour extraction and coding algorithm [22] that runs on the Exclusive Read Exclusive Write Parallel Random Machine (EREW PRAM) architecture in $O(log\ N)$ time with $O(N^2/\log N)$ processors. Their approach consists of three steps: (1) detection of contour pixels and assignment of pointers indicating the contour direction, (2) contour labeling, and (3) generating of lists of contour coordinates or codes. This algorithm make an assumption to simplify the process. It is assumed that the image does not contain any one pixel wide regions or protrusion. Therefore, a smoothing preprocessing such as morphological operations prior to the coding algorithm is needed. However, a picture may contain multiple regions and an object may contain multiple internal boundaries in our real world. If additional preprocessing step must be used, it should be suitable for the desired parallel architecture and should not affect the overall time complexity. Once restrictions exist, the algorithm has limitation for any specific usage. For example, since the input image cannot contain any one pixel wide protrusion or junction point, the developed coding algorithm can not be used for skeletonized or edge pixels image.

In contrast to the the above approach, a parallel coding algorithm [23] which is well-suited for the generation of chain or crack codes by adopting only two corresponding look-up tables is presented in this chapter. There is no assumption so that it can apply to all types of binary image. It is a significant extension of our previous work in Section 1.2 and it inherits the advantages of simplicity, efficiency and flexibility.

This chapter is organized as follows. In Section 2.2, the parallel algorithm is introduced. In Section 2.3, the coding stage is described. In Section 2.4, the merging stage is presented. In Section 2.5, the determination of contour types is discussed. In Section 2.6, experimental results are shown. In Section 2.7, the computational complexity is evaluated. Finally in Section 2.8, summary is made.

2.2 The Parallel Codes Extraction Algorithm

The contour representation is accomplished by a starting point and a sequence of moves around the border pixels of an object, namely a *chain* or *boundary link* which is definitely closed. A chain consists of the following four elements:

1. *Tail coordinates*: the coordinates of the starting point.
2. *Head coordinates*: the coordinates after the tracing of the chain is done.
3. *Code sequence string*: the sequence of codes indicating the directions of movement.
4. *Contour type*: the type of contours being external or internal.

The proposed algorithm using a *divide-and-conquer* strategy [24] consists of the following two stages:

1. *Coding*. The input image is partitioned into sub-images, and a *chain set* is extracted from each sub-image.

2. *Merging*. The chain sets of the adjacent nodes are merged into a new chain set. This step is recursively applied as working in a pyramid structure from bottom to top.

A pyramid architecture [25] is used in the implementation of our algorithm. Suppose that there are N nodes in the first (lowest) layer of the pyramid architecture shown in Figure 2.1. The higher the layer, the less number of nodes is. On the first layer of the pyramid, the entire image is divided into N sub-images corresponding to N nodes. Then, objects' contours are encoded into chain sets in each sub-image. On each subsequent layer, chain sets of its immediately lower layer are concatenated. In other words, the coding stage is performed on the first layer and the merging stage is carried out on the subsequent layers.

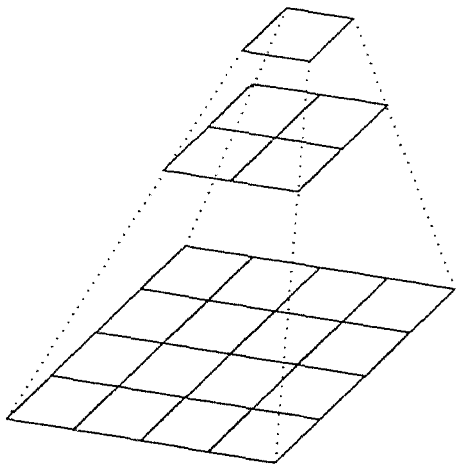


Figure 2.1 An example of a pyramid architecture

Two types of chains are defined as follows:

1. *Open chain*: if a chain crosses over two sub-images or more.
2. *Closed chain*: if a chain is located within one sub-image.

Assume that there is no object partially located in the whole image. The following properties are observed:

Property 1: On the subsequent layers, the closed chains remain closed. However, the open chains may become closed or remain open.

Property 2: The open chains are merged only when they touch on the shared boundaries.

Property 3: Eventually all chains after merging are closed.

Let *parent*(i) denote the parent of node i in the pyramid architecture. Consider a chain set in the $(k+1)$-th layer in node i, denoted as A_i^{k+1}, being merged from a number of chain sets in the k-th layer, A_j^k , $i = parent(j)$. Chain set A consists of two subsets, O and C. Subset O contains the open chains of set A and subset C contains the closed chains of set A. The parallel algorithm is stated as follows:

ALGORITHM

(*The Parallel Chain-Coding Algorithm in the Pyramid Architecture*)

```
input:   The $N \times N$ input image.
output:  The chain set $A^K$ on the top layer, $K$.
begin
        1. Coding stage:
           partition the input image into $N$ sub-images, $\{ s_j; 1 \le j \le N \}$. Each sub-
           image is in size $\sqrt{N} \times \sqrt{N}$.
           for each processor $j$, $1 \le j \le N$, pardo
                $A_j^1 = coding\ ( s_j )$
           end pardo
        2. Merging stage:
           for layer $k$, $k = 1$ to $K - 1$ do
                for each node $j$ in layer $k$ pardo
                     $A_i^{k+1} = merging\ (\{ A_j^k \ ;\ i = parent(j) \})$
                end pardo
           end do

end
```

2.3 The Coding Stage

Basically, the coding stage involves the codes generation and a series of operations which are applied to concatenate the new generated code links to the related chains by searching for a right link in the *check-head* or *check-tail* step as described in Section 1.2. We start with the discussion of similarities between chain coding and mid-crack coding.

2.3.1 The Similarities between Chain Coding and Mid-Crack Coding

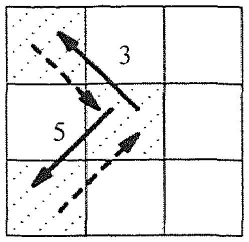
Three similarities are observed between the chain coding and the mid-crack coding:

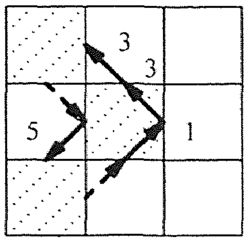
1. Their headcodes are identical.
2. Their codelinks point to the same pixel.
3. Their Theads are identical.

An example of the similarities is shown in Figure 2.2.

2.3.2 Codes Generation Look-up Table and Move Table

The codes generation table and move table of mid-crack codes have been described in Section 1.2.1. We will not repeat them here. In the same way, tables of chain codes are created. A chain code codes generation look-up table is shown in Table 2.1 and the move table is shown in Table 2.2.





Chain code :
codelink 1 = 3
Thead 1 = 1
codelink 2 = 5
Thead 2 = 7

Mid-crack code :
codelink 1 = 331
Thead 1 = 1
codelink 2 = 5
Thead 2 = 7

Figure 2.2 An example of similarities between mid-crack code and chain code. Solid arrows denote codelinks and dotted arrows denote Theads

2.3.3 Coding Mechanism

In this section, the usage of the look-up table and the move table is presented. For each sub-image s_j, A_j^1 is initialized as $\varnothing$, and is processed in a raster-scan fashion. If any object pixel is fetched, codelinks can be obtained quickly and simply using the look-up table. Then, the codelinks are joined to A_j^1. Three cases can occur during the process:

1. Merged to an existing chain.
2. Merged to two existing chains.
3. Creating a new chain.

Each case can be determined by checking the results from the check-tail and check-head steps which have been described in Section 1.2. The process is repeated until the raster-scan is completed. Because a chain may be open or closed in the intermediate step, three types of concatenation exist:

1. *Head concatenation* – concatenate the codelink with the matched chain and put the open chain into subset O.

2 *Tail concatenation* – concatenate the matched chain with the codelink and put the open chain into subset O.

3 *Head-and-tail concatenation* – concatenate the Tail's matched chain with the codelink, and with the Head's matched chain, then put the closed chain into subset C.

If none of the chains satisfies the above conditions, a new open chain is created in the chain set O.

Table 2.1 The Look-Up Table of Chain Coding

Index	Number of Codelinks	Thead	Codelink
0	0	Nil	Nil
1	1	7	3
2	1	6	2
3	1	7	2
4	1	5	1
5	2	5,0	1,3
...	...	...	...
130	2	6,3	7,2
131	2	7,3	7,2
132	2	5,3	7,1
133	3	7,5,3	7,3,1
134	2	6,3	7,1
...	...	...	...
251	0	Nil	Nil
252	1	5	4
253	1	5	3
254	0	Nil	Nil
255	0	Nil	Nil

Table 2.2 The Move Table of Chain Coding

Move	Relative Coordinates	
	Column	Row
0	+1	0
1	+1	-1
2	0	-1
3	-1	-1
4	-1	0
5	-1	+1
6	0	+1
7	+1	+1

2.3.4 The Procedure of Coding Stage

In each sub-image, s_i, the chain set, A_i^1, can be generated by the following procedure. Since the 3×3 window operation is applied everywhere including the image boundary, the sub-image is extended to have two more rows and columns of background elements.

```
PROCEDURE Coding
input:   The $\sqrt{N} \times \sqrt{N}$ sub-image $s_j$.
output:  The chain set $A_j$ consists of all chains in $s_j$.
begin
         while raster-scan each pixel on $s_j$ do
               if an object pixel then
                     1. Calculate the index value according to Figure 1.5(a).
                     2. Obtain codelinks and headcodes according to Table 2.1.
                     for each codelink do
                           • Obtain the destination coordinates according to Table 2.2.
                           • Apply the check-head step.
                           • Apply the check-tail step.
                           • Join to chain set $A_j$.
                     end do
               end if
         end do
```

end

2.4 The Merging Stage

After performing coding stage process at each sub-image, the extracted chain sets of each processor are merged together. The rules for merging two chains, c_i and c_j can be defined as follows:

1. The Head coordinates of c_i is matched to the Tail coordinates of c_j.
2. The headcode of c_i is matched to the thead of c_j.

Proposition 2.1: For two adjacent nodes i and j, the open chains of node i and j which lie on the shared boundary should have one-to-one correspondence, Head-To-Tail or Tail-To-Head.

Proof: Let A_i and A_j be two chain sets with a shared boundary, S_{ij}. The open chains of A_i with Heads or Tails which lie on S_{ij} are denoted as set L_i, and the open chains of A_j with Heads or Tails which lie on S_{ij} are denoted as set L_j. Then, the number of open chains in L_i is equal to the number of open chains in L_j, and all chains of L_i will be merged to those of L_j. Otherwise, some open chains are kept open until the processing is finished, since one open end (Tail or Head) has no chance to merge to other chains in the continuous processing. However, no partial objects in the input image is assumed. All chains should be closed after the merging process. By contradiction, the open chains on the shared boundary should have one-to-one correspondence. Moreover, only Head can be connected to Tail. Each pair of correspondence is either Head-To-Tail connection or Tail-To-Head connection.

Definition (union, $\bigcup$): For two chain sets A_i and A_j, let $A_i \bigcup A_j$ be defined as the result of merging A_i and A_j if these two chain sets satisfy the proposition above.

Let chain set A_l at layer $k+1$ which is the union of the chain sets A_i from the immediately lower layer k for all i, where $l = parent(i)$. Assume that the subset O_i of A_i in node i have been consistently sorted by positions of Heads and Tails into L_i for those Heads and Tails located at the shared boundary. If the shared boundary is in the vertical direction of $x = x_1$, only those chains whose Heads and Tails are at $x = x_1$ according to y coordinates are sorted. A sorted L_i contains only the chains corresponding to the neighbor's sorted L_j. A chain set $L_{ij} = L_i \bigcup L_j$ is defined by merging chains associated with L_i and L_j. Then, a union chain set of A_i and A_j is defined by

$$A_i \bigcup A_j = A_i - L_i + A_j - L_j + L_{ij}$$

The chain set A_l of node l is defined by

$$A_l = \bigcup_{l = parent(i)} A_i$$

The procedure for merging chain sets into a new chain set is expressed as follows:

```
PROCEDURE Merging
input:    The chain sets $A_i$ for any $i$ which parent is $A_l$.
output:   A new chain set $A_l$.
begin
          for each shared boundary $S_{ij}$ between $L_i$ and $L_j$ do
              1.  Sort Heads and Tails coordinates only for those chains which lie on
                  the boundary $S_{ij}$ .
              2.  Join the chains between $L_i$ and $L_j$ .
              3.  Move the new chain into chain set $C_l$ if the chain is closed. Other-
                  wise, move the new chain into chain set $O_l$ .
              4.  Move the chain set $C_i$ to the chain set $C_l$ for all $i$ which has the
                  same parent $l$
          end do
end
```

2.5 Determination of Contour Types

Methods for quickly determining the contour type are given. At both coding and merging stages, contour types are determined once the open chains become closed. In this section, the methods for determining the contour type are stated as follows.

After the check-head step and the check-tail step, the open status of a chain has already been known. Actually, only two cases exist and they are shown in Figure 2.3. If the first two codes of a chain are "45" or "35", it must be an internal contour. The other cases indicate that the chain is external. This determination is independent of coding-type, chain or mid-crack coding.

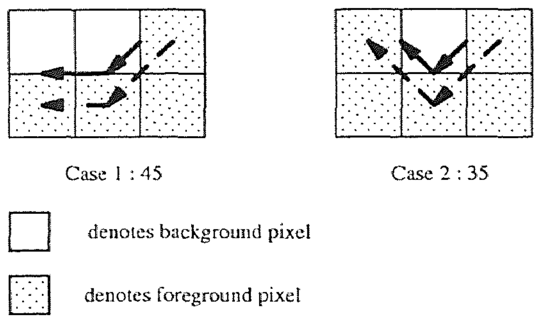


Figure 2.3 Two cases of chain and mid-crack codes while the internal contour is closed

In the merging stage, since Heads and Tails have been sorted by their coordinates, the contour type is checked only when two open chains become closed. The checking process is to compare the coordinates of the pair of Head and Tail. Suppose that merging process starts from small coordinates to large coordinates. From observation, if Head coordinates are larger than Tail coordinates, then the contour type is internal. Otherwise, the contour type is external.

2.6 Experimental Results

By simulating the pyramid architecture with four nodes in the first layer, the chain set of the final result in each node is shown. Multiple objects and the internal contour are concerned in the input image which is shown in Figure 2.4. The result of chain coding is illustrated in Figure 2.5 and that of mid-crack coding is shown in Figure 2.6. A *C* language function *times()* is embedded in the program to compute the used CPU time. For a 64×64 input image with 8 contours, it costs 0.1 user time and 0.1 system time in a SUN4M machine.

2.7 Complexity Evaluation

Since there is no other well known chain coding algorithm which can accept input image with multiple regions, the single pass coding algorithm [7] is used as a reference. It has worst case time complexity of $O(N^3)$. The following is the analysis. For an $N \times N$ input image, there may have $O(N^2)$ pixels to be processed. Each pixel may need $O(N)$ matching time to search the correct boundary link before concatenation if $O(N)$ open boundary links exist while the pixel is processed. So, the overall time complexity is $O(N^3)$.

The worst case of the time complexity for our proposed parallel algorithm is in $O(N^{3/2})$ time using $O(N)$ processors. The following is the analysis. The size of the input image is $O(N^2)$. If $O(N)$ processors is used, the size of each sub-image is $\sqrt{N} \times \sqrt{N}$. The height of the pyramid is $O(\log N)$. In coding stage, each object pixel only needs $O(1)$ to generate the chain codes. For each generated codelinks, it needs to search all the open chain in the chain set. The maximum number of the open chains existed will be of $O(\sqrt{N})$. The worst case is that $O(N)$ open chains exist in an sub-image in the merging step, By using $O(N \log N)$ sorting algorithm, each sub-image can be done in $O(N^{3/2})$ time. In merging stage, we have four sub-images to merge in each layer but the size of the sub-image increases from $\sqrt{N} \times \sqrt{N}$ to $\frac{N}{4} \times \frac{N}{4}$. We can write down the following equation for the time complexity of the merging stage.

$$O(Merging) = \sqrt{N}\log\sqrt{N} + 2\sqrt{N}\log(2\sqrt{N}) + \cdots + 2^{k-1}\sqrt{N}\log(2^{k-1}\sqrt{N})$$

$$+ 2\sqrt{N}\log(2\sqrt{N}) + 2^2\sqrt{N}\log(2^2\sqrt{N}) + \cdots + 2^k\sqrt{N}\log(2^k\sqrt{N})$$

$$= \sum_{k=1}^{\log\sqrt{N}} 2^{k-1}\sqrt{N}\log(2^{k-1}\sqrt{N}) + \sum_{k=1}^{\log\sqrt{N}} 2^k\sqrt{N}\log(2^k\sqrt{N})$$

$$< O(N^{3/2})$$

The overall time complexity of this algorithm is $O(N^{3/2})$.

2.8 Summary

We have presented a new Freeman chain-coding algorithm using parallel machine machines. Mid-crack coding can be similarly achieved by adopting a different look-up table. The algorithm is implemented using *CASIM*, a simulator of *AP*1000 parallel machine.

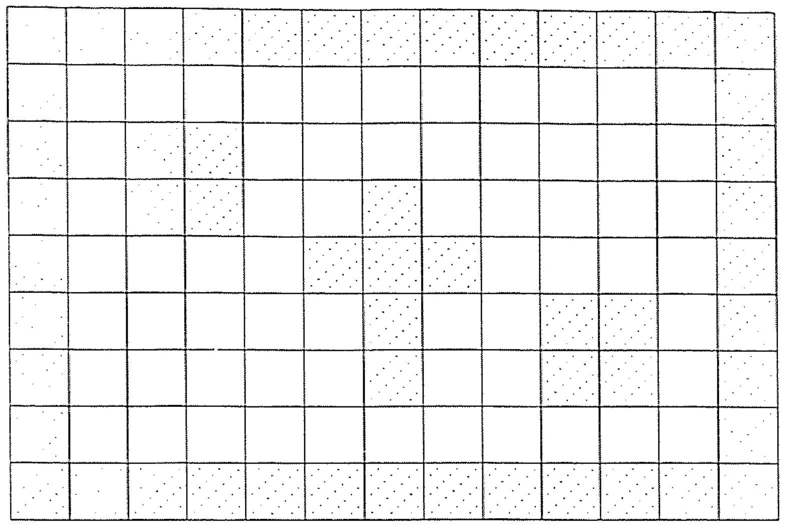


Figure 2.4 The input image with four objects and five contours

First Layer

Node 1

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(1.5)	(6.1)	4	Open	666644444
2	(7.1)	(1.4)	2	Open	00000122
3	(4.3)	(4.3)	0	Closed	2064

Node 2

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(6.1)	(13,4)	2	Open	444444222
2	(13.5)	(7.1)	0	Open	666700000
3	(6.5)	(7.4)	3	Open	5

Node 3

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(7.9)	(1.5)	6	Open	0000006666
2	(1.4)	(6.9)	4	Open	222234444
3	(7.6)	(6.5)	5	Open	7

Node 4

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(7.4)	(6.1)	7	Open	3126
2	(13.4)	(1.4)	0	Open	2222000000
3	(11.6)	(11.6)	0	Closed	2064
4	(6.9)	(13.5)	6	Open	4444445666

Second Layer

Node 5 (from Node 1 and 2)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(1.5)	(13.4)	2	Open	66664444444444444222
2	(13.5)	(1.4)	2	Open	66670000000000122
3	(4.3)	(4.3)	0	Closed	2064
4	(6.5)	(7.4)	3	Open	5

Node 6 (from Node 3 and 4)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(13.4)	(1.5)	6	Open	22220000000000006666
2	(1.4)	(13.5)	6	Open	22223444444444445666
3	(7.4)	(6.5)	5	Open	31267
4	(11.6)	(11.6)	0	Closed	2064

Top Layer

Node 7 (from Node 5 and 6)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(13.4)	(13.4)	6	Closed	2222000000000000066666664444444444444222
2	(1.4))	(1.4)	6	Closed	2222344444444444566666670000000000122
3	(4.3)	(4.3)	0	Closed	2064
4	(7.4)	(7.4)	5	Closed	312675
5	(11.6)	(11.6)	0	Closed	2064

Figure 2.5 The result of Figure 2.4 by applying our algorithm with chain codes

First Layer

Node 1

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(1,5)	(6,1)	4	Open	6666544444
2	(7,1)	(1,4)	2	Open	00000122
3	(4,3)	(4,3)	0	Closed	21076543

Node 2

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(6,1)	(13,4)	2	Open	4444443222
2	(13,5)	(7,1)	0	Open	666700000
3	(6,5)	(7,4)	0	Open	553

Node 3

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(7,9)	(1,5)	6	Open	00000076666
2	(1,4)	(6,9)	4	Open	222234444
3	(7,6)	(6,5)	5	Open	775

Node 4

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(7,4)	(7,6)	7	Open	33112176
2	(13,4)	(7,9)	0	Open	222221000000
3	(11,6)	(11,6)	0	Closed	21076543
4	(6,9)	(13,5)	6	Open	4444445666

Second Layer

Node 5 (from Node 1 and 2)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(1,5)	(13,4)	2	Open	66665444444444443222
2	(13,5)	(1,4)	2	Open	66670000000000122
3	(4,3)	(4,3)	0	Closed	21076543
4	(6,5)	(7,4)	3	Open	553

Node 6 (from Node 3 and 4)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(13,4)	(1,5)	6	Open	22222100000000000076666
2	(1,4)	(13,5)	6	Open	2222344444444445666
3	(7,4)	(6,5)	5	Open	33112176775
4	(11,6)	(11,6)	0	Closed	21076543

Top Layer

Node 7 (from Node 5 and 6)

No.	Head Coord.	Tail Coord.	Thead	Type	Code Sequence String
1	(13,4)	(13,4)	6	Closed	2222210000000000007666666665444444444443222
2	(1,4)	(1,4)	6	Closed	22223444444444456666667000000000012 2
3	(4,3)	(4,3)	0	Closed	21076543
4	(7,4)	(7,4)	5	Closed	33112176775553
5	(11,6)	(11,6)	0	Closed	21076543

Figure 2.6 The result of Figure 2.4 by applying our algorithm in mid-crack code

CHAPTER 3

A PARALLEL CHAIN AND MID-CRACK CODING ALGORITHM ON N CUBE ARCHITECTURE

3.1 Introduction

Most kernels for image processing operate on the object contour in order to obtain information for further analysis or computation. The motivation for the use of parallel processing in image related problems has been the strong urge to improve the performance of existing solutions in order to reach the maximal computing power. In the previous chapter, I have presented a pyramid parallel coding algorithm which run on $O(N)$ processors in time $O(N^{3/2})$. In this chapter, I am going to present a new parallel coding algorithm which can be implemented on N *cube* architecture. It inherits the flexibility from our previous algorithm which can generate the chain codes or mid-crack codes. It can also run on $O(N)$ processors in time $O(N \log N)$. This new algorithm is more efficient.

A three-dimensional cube is shown in Figure 3.1(a). Referring to [26], vertical lines connect vertices (processors) whose addresses differ in the most significant bit position. Vertices at both ends of the diagonal lines differ in the middle bit position. Horizontal lines differ in the least significant bit position. This unit-cube concept can be extended to an N –dimensional unit space, call an N *cube*, with n bits per vertex. A cube network for a SIMD machine with N processors corresponds to an N *cube* where $n = \log_2 N$. The binary sequence $A = (a_{n-1} \cdots a_1 a_0)_2$ is used to represent the vertex address for $0 \le A \le N - 1$. $\overline{a_i}$ is denoted as the complement of bit a_i for any $0 \le i \le n - 1$. The routing function is specified as follows:

$$C_i(a_{n-1} \cdots a_1 a_0) = a_{n-1} \cdots a_{i+1} \overline{a_i} a_{i-1} \cdots {}_0 \qquad \text{for } i = 0, 1, 2, \ldots, n-1$$

The routing function of a 3 *cube* is shown in Figure 3.1(b).

This chapter is organized as follows: In Section 3.2, the new parallel algorithm for chain or mid-crack coding is proposed. In Section 3.3, experimental results are given. In Section 3.4, time complexity of the algorithm is evaluated. Finally in Section 3.5, summary is made.

3.2 The New Parallel Algorithm for Chain or Mid-Crack Coding

A single instruction stream-multiple data stream (SIMD) design is used. Each row of the image feeds to the corresponding processor in the processor array. The new parallel coding algorithm also consists of two stages, *Coding stage* and *Merging stage*.

In this chapter, only chain code is used in the examples. Let the size of the input image be $N \times N$. Let s_j be the sub-image which contains the row j of the input image and its size is $1 \times N$. Let the number of processors used be N. Let A_j be the chain set of the processor j. Chain set A_j consists of two subsets, O_j and C_j. Subset O_j contains the open chains of set A_j and subset C_j contains the closed chains of set A_j. The parallel algorithm is stated and shown in Figure 3.2.

3.2.1 The Coding Stage

We recall from Section 1.2.1, the coding stage includes 1) the codes generation from the look up table and 2) the process of codelink. The set up of the look up table and the move table is very similar. The only difference is the sequence of the codelinks. Since the chains' Head or Tail coordinates of two merging chain sets must be in sequence in the merging stage, the order of the codelinks generation is significant. After the preliminary description, we will discuss the codelinks sequence.

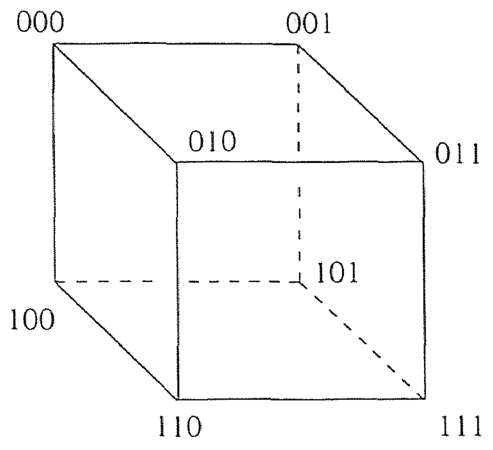


Figure 3.1(a) A 3 cube architecture

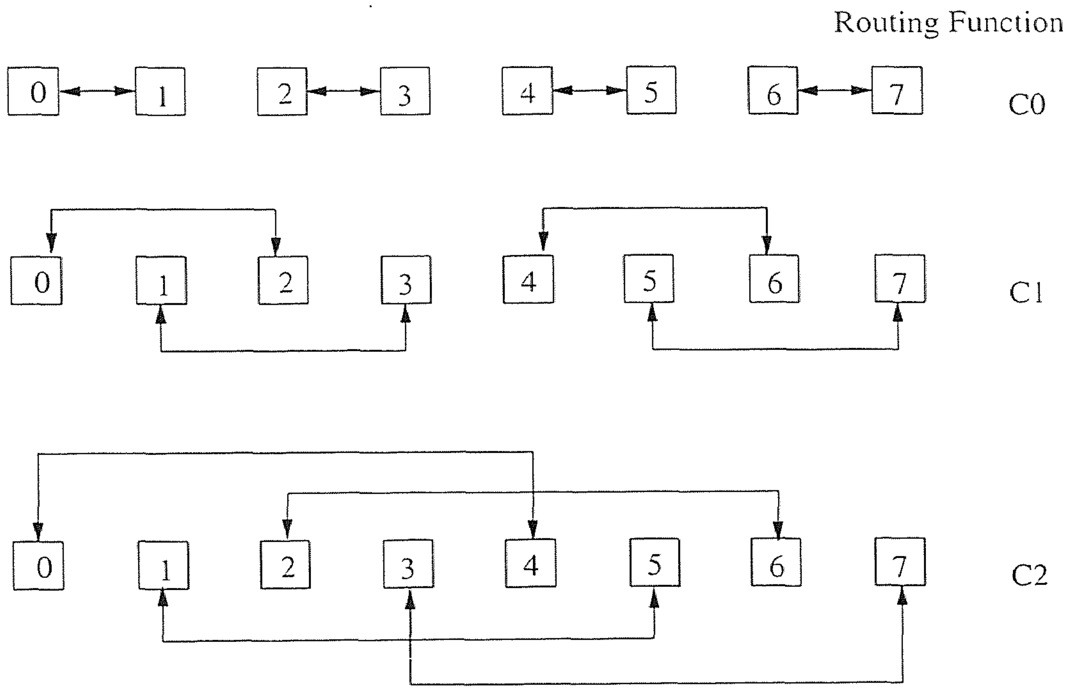


Figure 3.1(b) A 3 cube recirculating netowrk

```
                              ALGORITHM
input: The $N \times N$ input image.
output: The chain set $A_{N-1}$ on processor $N - 1$.
begin
    1. Coding stage:
        partition the input image into $N$ sub-images, $\{ s_j; 0 \le j \le N-1 \}$.
        for each processor $j$, $0 \le j \le N-1$, pardo
            $A_j = coding\ ( s_j )$
        end pardo
    2. Merging stage:
        for $k = 0$ to $\log N - 1$ do
            for each processor $y$, $y = 0$ to $N - 1$ pardo
                if $(y\ modulo\ 2^{k+1} = 2^k - 1)$ then
                    $A_x = merging\ (A_y, A_x)$
                    where $x = y + 2^k$ and $x \le N - 1$.
                endif
            end pardo
        end do
end
```

Figure 3.2 The Parallel Chain-Coding Algorithm in *N - Cube* Architecture

Link-list type data structures are adopted in our implementation. Seven link-lists as shown in Figure 3.3 are used to describe the open chain set which is in process. Because the partition of an input image is set row by row, except the first and the last processors, a sub-image in a processor will have two shared boundaries: one shares with the upper neighbor and the other one shares with the lower neighbor. Link-List H_u and H_l store the Head coordinates and the Headcode of a chain. Link-List T_u and T_l store the Tail coordinates and Thead of a chain. Subscript u and l are denoted as the upper shared boundary and the lower shared boundary. Link-list L_c is for code strings and information. Link-list *TempH* is the temporary storages for the Head coordinates and the Headcode of a chain. The last one, Link-list *TempT* is the temporary storage for the Tail coordinates and the Thead. When the Headcode of a chain is code 0 or Thead is code 4, it will connect to a codelink from the next processed pixel in the same row. When the Headcode of a chain is code 4 or Thead is code 0, it will connect to an existing chain from the same chain set.

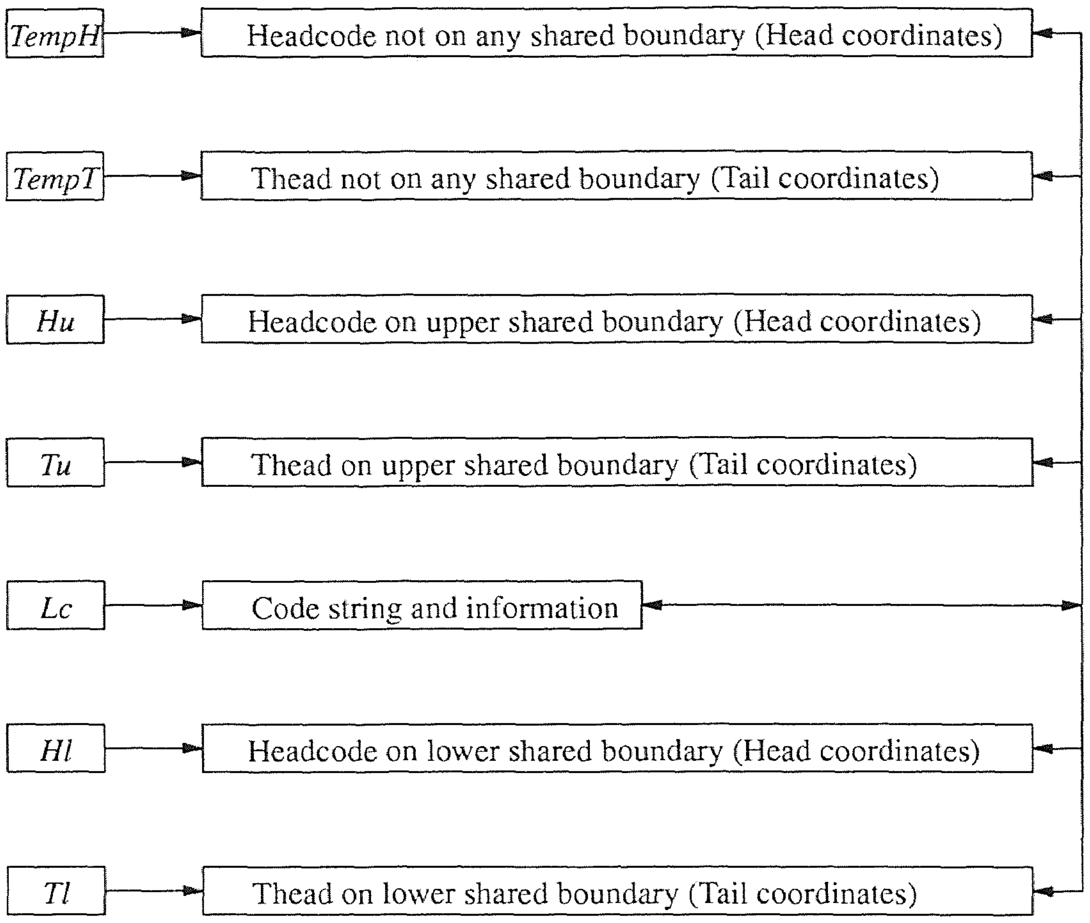

TempH
Headcode not on any shared boundary (Head coordinates)
TempT
Thead not on any shared boundary (Tail coordinates)
Hu
Headcode on upper shared boundary (Head coordinates)
Tu
Thead on upper shared boundary (Tail coordinates)
Lc
Code string and information
Hl
Headcode on lower shared boundary (Head coordinates)
Tl
Thead on lower shared boundary (Tail coordinates)

Figure 3.3 The link-list structures for a chain set

If the direction of the Headcode of a chain points upward, such as codes $1, 2, 3$, the Head coordinates will belong to the upper shared boundary L_u. On the other hand, if the direction of the Headcode of a chain points downward, such as codes $5, 6, 7$, the Head coordinates will belong to the lower shared boundary.

In the same way, if the direction of the Thead points upward, such as codes $1, 2, 3$, the Tail coordinates will belong to the lower shared boundary L_l. If the direction of the Thead points downward, such as codes $5, 6, 7$, the Head coordinates will belong to the upper shared boundary. Codes 0 and 4 do not belong to any shared boundaries because no vertical shared boundary exists.

In Figure 3.4, the order of the codelink generated is shown and the sequence will be preserved in the look-up table. From the observation, three cases would not produce the resulting Tail coordinates in order. These three special cases are shown in Figure 3.5. Extra swapping procedure is needed for the adjustment. The swapping procedure is to swap the two related Theads in link-list T_u or T_l to restore the sequence after these two codelinks are generated.

After a codelink is generated, we try to connect the codelink to the existed chain. If there is no connection, a new chain is created. If a connection exists, the Head or Tail coordinates will be replaced by those of the codelinks. Also, the code string is concatenated with the codelink. During coding stage, if a chain is closed, the closed chain is put into chain set C_i. Otherwise, the open chain is put into chain set O_i.

A 5×5 input image with all chain coded contours is shown in Figure 3.6 and the intermediate result of the coding stage for each row is shown in Figure 3.7. By using this approach, the generated chains and their Head or Tail coordinates will be in sequence. The overall time complexity $O(N \log N)$ can be achieved because we gain the advantage when two lists of ordinal chain set are merged in the merging stage by using link-list representation. The merging stage will be described briefly in the next section.

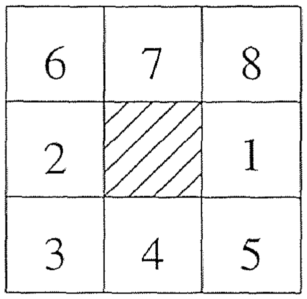
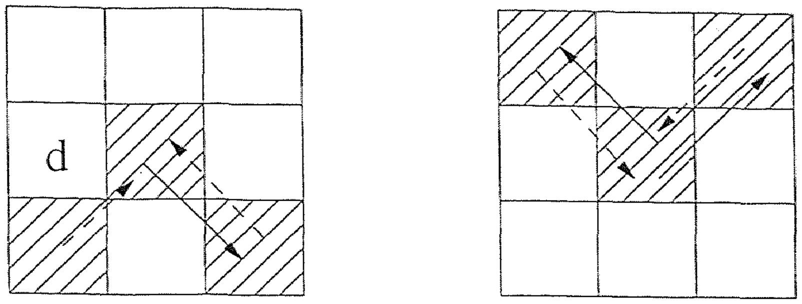


Figure 3.4 The order of the codelink generation



d: denotes don't care, object pixel or background pixel

Figure 3.5 The patterns that need extra swapping procedure

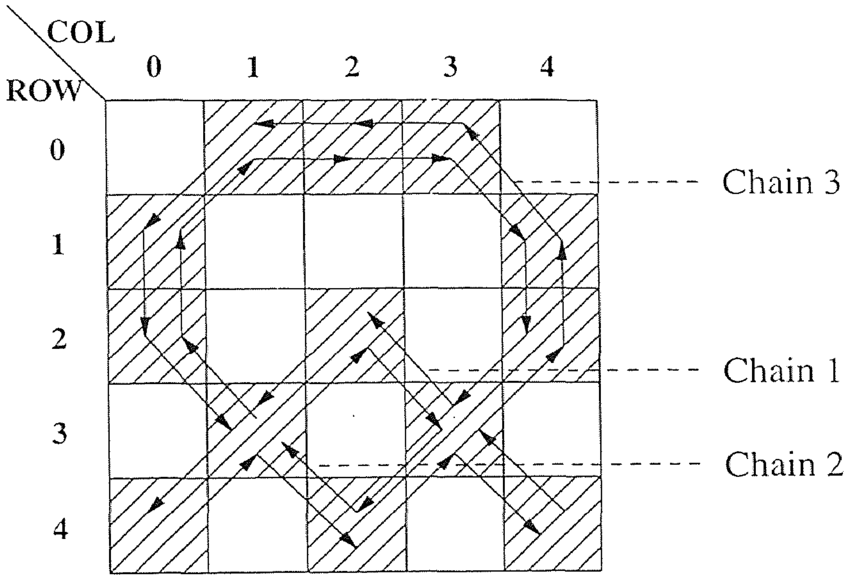


Figure 3.6 An input image with three chains

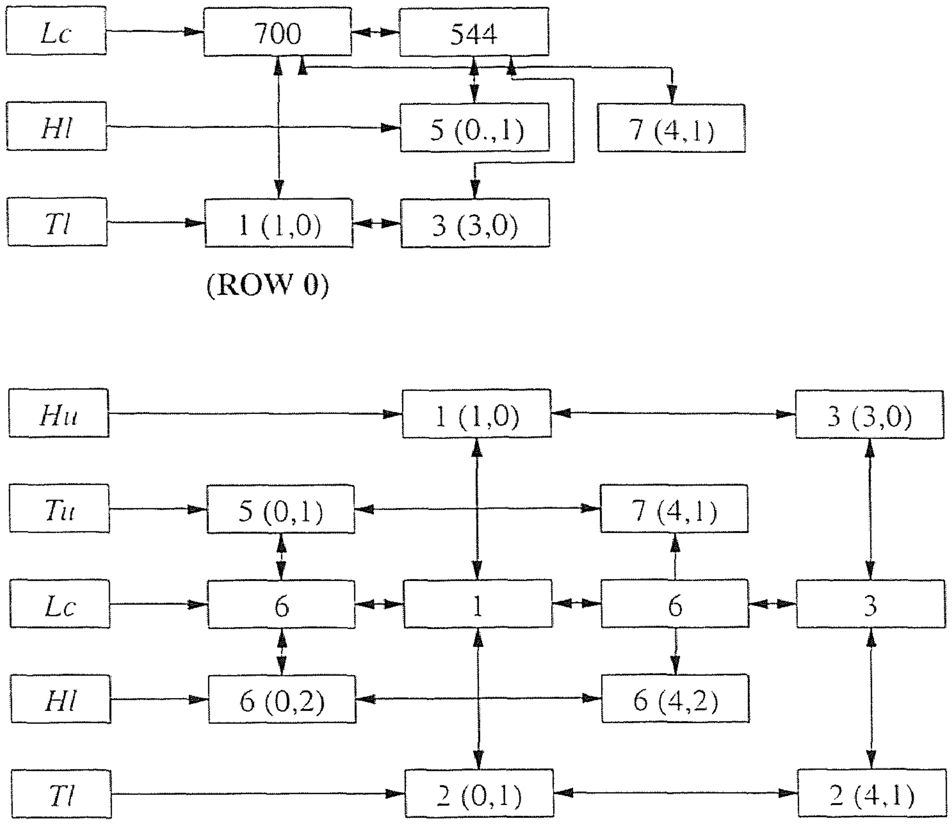
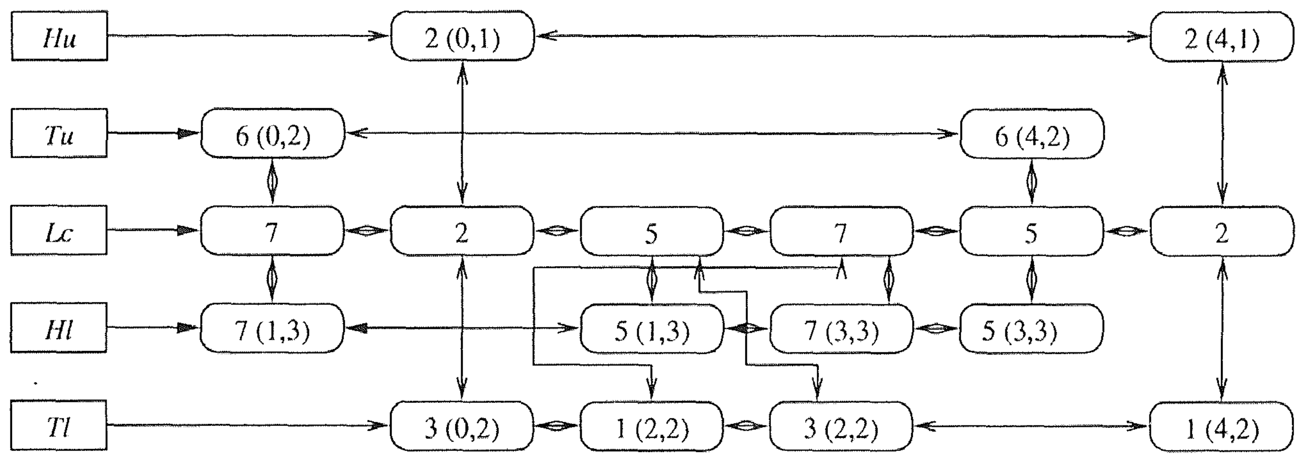
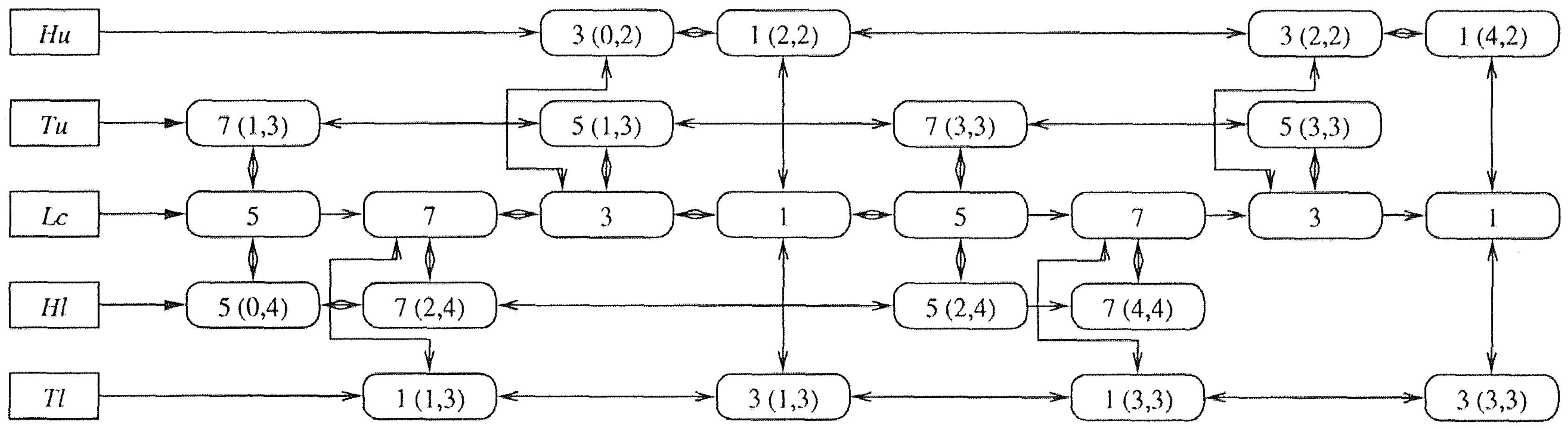


Figure 3.7(a) The coding result for the first two row of Figure 3.6

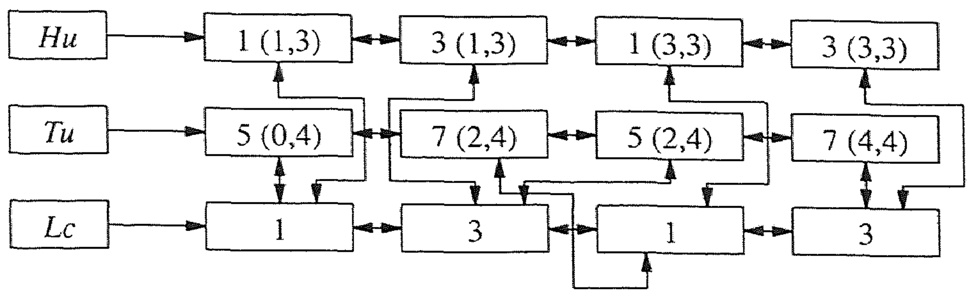


(ROW 2)



(ROW 3)

Figure 3.7(b) The coding result for the second two row of Figure 3.6.



(ROW 4)

Figure 3.7(c) The coding result for the fifth row of Figure 3.6

3.2.2 The Merging Stage

A chain set A_y at upper processor y will route to a lower processor x, where $x = y + 2^k$ based on the routing variable k. Two chain sets A_y and A_x will merge in processor x. We reuse the same notation A_x for this merged chain set because it is easier to describe the mathematic formula. From proposition 2.1, we know that the open chains of processor y and x which lie on the shared boundary should have one-to-one correspondence. Since the Head or Tail coordinates link-lists on L_{ly} has one-to-one correspondence to those link-lists on L_{ux} , it takes only time $O(N)$ to traverse the link-lists and merge them together. The resulting chain set will have the upper boundary L_{uy} and the lower boundary L_{lx}. The shared boundary will disappear after two processors are merged. The Head or Tail coordinates on the shared boundary will be deleted. If connection occurs, the code string of a chain in A_y is concatenated to the code string of a chain in A_x. During the merging stage, if a chain is closed, the closed chain will be put into chain set C_x. Open chains will be put into chain set O_x and are linked in the link-list L_c of processor x.

The rules for merging two chains are quite similar to those in Section 2.4. Let c_i be a chain on processors x and c_j be a chain on processors y. The head of c_i may connect to the tail of c_j and vice versa. The rules are defined as follows:

1. The Head coordinates H_{li} is equal to the Tail coordinates T_{uj}.

2. The Headcode of c_i is equal to the Thead of c_j.

or,

1. The Head coordinates H_{uj} is equal to the Tail coordinates T_{li}.
2. The Headcode of c_j is equal to the Thead of c_i.

An example is shown in Figure 3.8 to illustrate the intermediate and final results of the merging stage in Figure 3.6. After the first routing, the merged results of row $0-1$ and row $2-3$ of the input image are shown in Figure 3.8(a) and 3.8(b) respectively. Row 5 has no change. After the second routing, the merging result of row $0-3$ are shown in Figure 3.8(c). The row 5 still has no change. After the last routing, the final merged result is shown in Figure 3.8(d).

3.3 Experimental Results

Two input images and their results are shown in Figure 3.9 and Figure 3.10. The first image as shown in Figure 3.9(a) lists the three digits 0, 9 and 6 in horizontal direction. The second image as shown in Figure 3.10(a) lists them in vertical direction. The size and the shape of each digit are the same but the chain code sequences of the resulting chains are not all the same. It is due to the different partitions between two input images. Their results are shown in Figure 3.9(b) and 3.10(b) respectively.

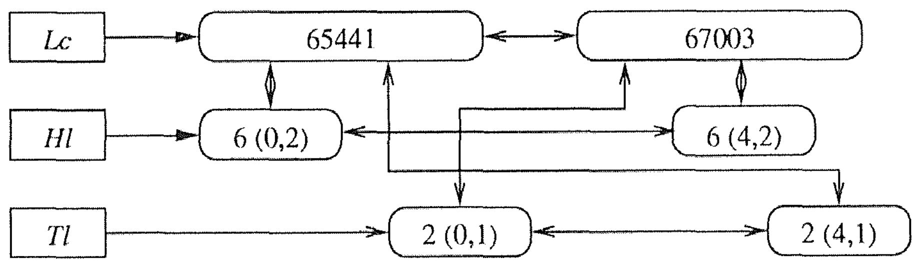


Figure 3.8 (a) The merging result for the first two row of Figure 3.6.

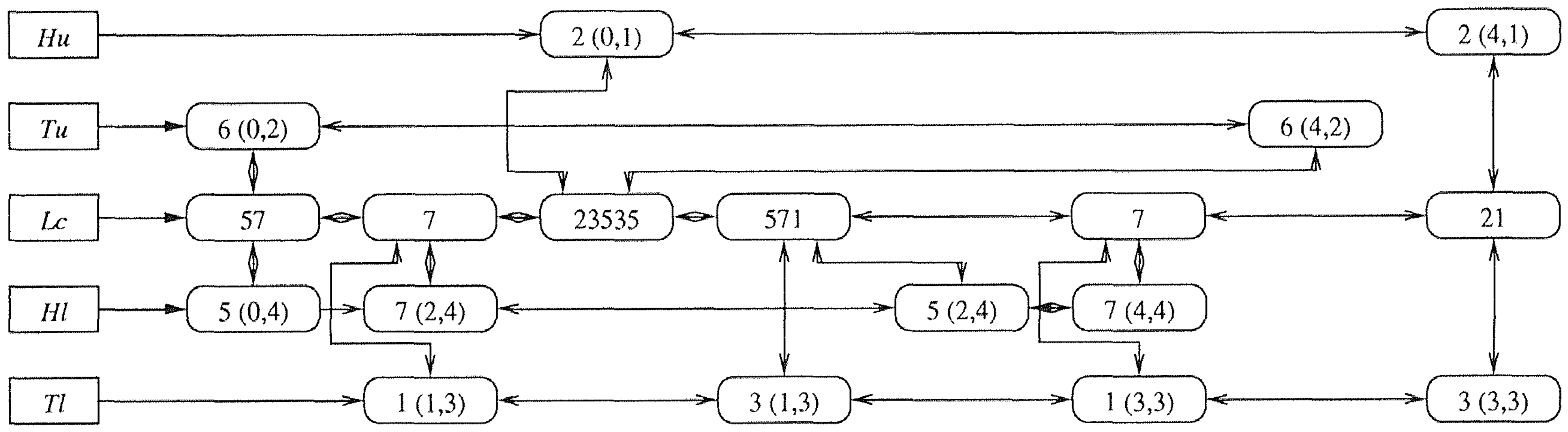


Figure 3.8 (b) The merging result for the second two row of Figure 3.6.

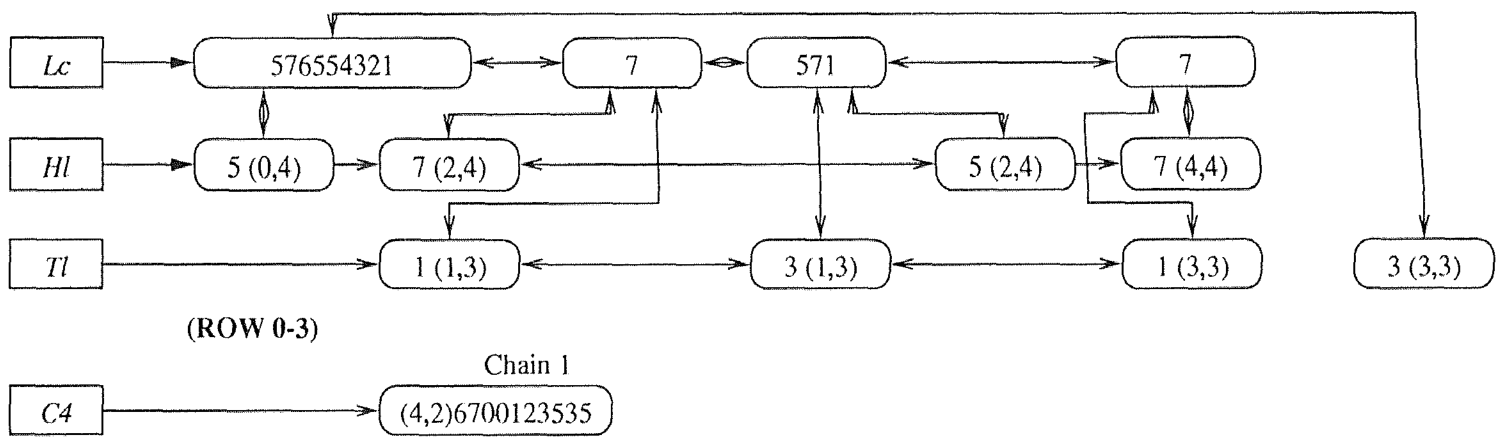


Figure 3.8 (c) The merging result for the row 0 to row 3 of Figure 3.6.

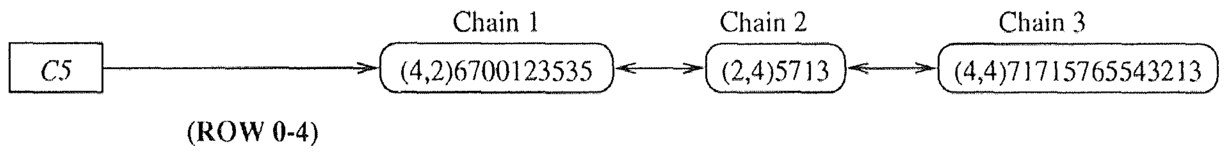
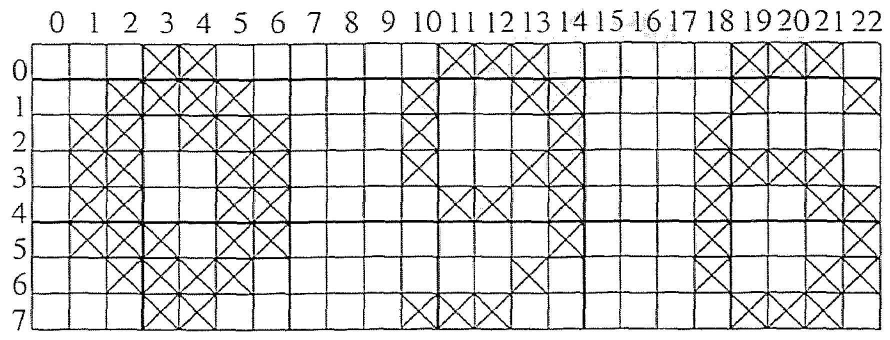


Figure 3.8 (d) The merging result of Figure 3.6.



(a)

```
number of closed chains: 6

Coordinate: (X,Y) = (1, 4)
code string: 665543322211 0776

Coordinate: (X,Y) = (5, 4)
code string: 6771223356

Coordinate: (X,Y) = (12, 4)
code string: 5577012234

Coordinate: (X,Y) = (14, 4)
code string: 7107665443222211 0044556

Coordinate: (X,Y) = (18, 4)
code string: 665644370113443221 00766

Coordinate: (X,Y) = (21, 4)
code string: 7012234557
```

(b)

Figure 3.9 (**a**) The input image which lists 3 digits 0, 9 and 6 horizontally (**b**) The resulting chains

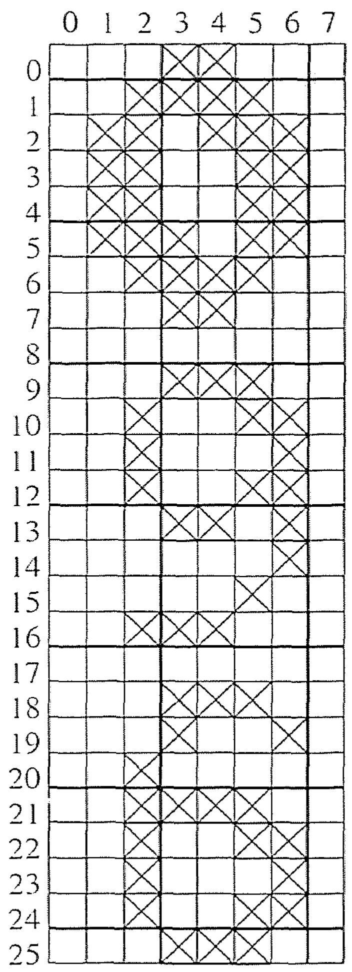


Figure 3.10 (a) The input image which lists 3 digits 0, 9 and 6 vertically

```
number of closed chains: 6

Coordinate: (X,Y) = (2, 24)
code string: 66665644370113443221007

Coordinate: (X,Y) = (5, 24)
code string: 5770122345

Coordinate: (X,Y) = (5, 12)
code string: 5770122345

Coordinate: (X,Y) = (1, 4)
code string: 6655433222110776

Coordinate: (X,Y) = (5, 4)
code string: 6771223356

Coordinate: (X,Y) = (4, 16)
code string: 5567107665443222110044
```

Figure 3.10 (b) The resulting chains

3.4 Time Complexity Evaluation

The proposed algorithm includes two major portions. One is the coding stage for each row of input image in each processor. The other is the merging stage between the chain sets of processors. In the coding stage, there are at most $O(N)$ object pixels for coding. Each pixel spends $O(1)$ time, therefore the computation time of coding stage is $O(N)$ in total. In the merging stage, we first traverse the Head Coordinates link-list H_u of the processor x and merge with the Tail Coordinates link-list T_l of the processor y. Then we traverse the Tail Coordinates link-list T_u of the processor x and merge with the Head Coordinates link-list H_l of the processor y. All link-list has at most $O(N)$ elements so that it takes $O(N)$ time to traverse a link-list. To merge two elements from two different link-lists takes $O(1)$ time. Therefore, the time it takes to merge the two chain sets will be $O(N)$. There are $O(\log N)$ routing in total. The overall time complexity for the merging stage is $O(N \log N)$. The overall time complexity for the whole algorithm is also $O(N \log N)$.

3.5 Summary

This chapter presents an efficient parallel coding algorithm for the binary image. Each processor reads the data for one corresponding row of image and generates an in-sequence coding result by using two lookup tables. Results are represented in a link-list type data structure. Then based on the routing variable, the chain set of the two corresponding processors are merged together. Since the intermediate results of the coding stage and the merging stage are all in sequence, it will only take $O(N)$ time to merge the two link lists. The overall time complexity will be $O(N \log N)$ for $N \times N$ input image with $O(N)$ processors. This algorithm can be applied to multi-resolution images easily.

CHAPTER 4

A ONE-PASS ALGORITHM FOR LOCAL SYMMETRY OF CONTOURS FROM CHAIN CODES

4.1 Introduction

Iñesta, Buendía, and Sarti [13] developed an algorithm to calculate the local symmetry deficiency (*LSD*) of scope k which reflects the degree of symmetry of a local zone for a curve. Their approach is based on the following concept originally derived by Owaga [27]: a curve segment arriving at a point p_i, forms a reflective symmetry at p_i with another curve segment coming out of it. This idea yields a measurement of the local symmetry up to where the two arms of a curve move away from a point. A size k neighborhood of p_i is determined to evaluate the amount of local symmetry at that point of the curve. The same procedure is carried out for all the points of the curve. We view it as a similarity rate relative to p_i in the "radius" of k points. Thus, the *LSD* of scope k, denoted as $\sum_k$, is defined to reflect the similarity of the curve. The *LSD* is calculated as the distances between those points traversed by the chain code in the k neighborhood on one side of p_i and the ones of a specular reflection, suitably rotated, of the curve on the other side. Due to the nature of this measurement, local symmetry will be maximum when $\sum_k$ is minimum. A curve will be perfectly symmetric at p_i in the vicinity of radius k when $\sum_k(p_i) = 0$.

The complexity of this algorithm is $O(kN)$, where N is the length of chain codes description. The radius of the support region is a parameter strongly dependent on the application desired for measuring *LSD*. Very small values of k will lack usefulness since they will not be able to detect useful symmetries in real objects. It is reasonable for k to assume values that will be small fractions of N.

Figure 4.1 outlines the local symmetry algorithm. There are three stages in this algorithm:

1. Reflection stage:

- Take the exiting curve C_k^+ with respect to p_i in the k segments following p_i.
- Reflect curve C_k^+ according to an axis formed by the first segment of C_k^+ producing C_k^*.

2. Rotation and direction adjustment stage:

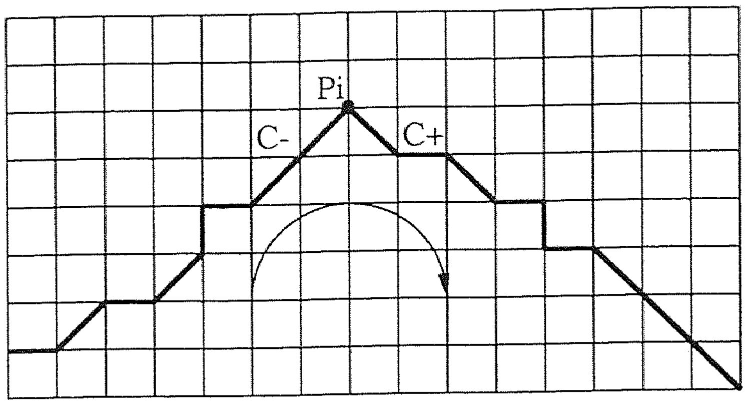
- Compute the 1-curvature at p_i.
- Rotate C_k^* with respect to the curvature and obtain $C_k^{'}$. $C_k^{'}$ matches the segment C_k^- which is the arriving curve at p_i.

3. Computing stage:

- Compute point-to-point distances between curve C_k^* and the corresponding part of the segment C_k^-.

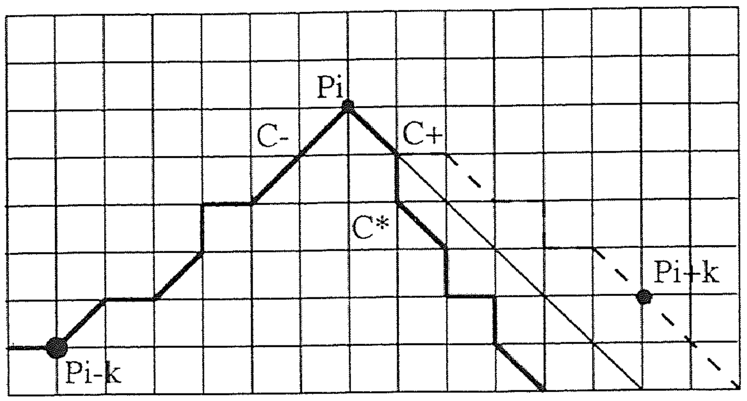
If the code length of the contour is $O(N)$, the complexity of the algorithm is $O(kN)$. However, if $k << N$, it could be considered to have a linear complexity. As the value of k decreases, the value of *LSD* becomes more local. It is reasonable for k to assume values to be small fractions of N. Thus, the algorithm is closer to linear complexity rather than quadratic one.

In this chapter, we embed the *LSD* computation in a new single pass chain coding algorithm. When an object pixel is coded and joined to a chain, the calculation of the k neighborhood for each stage (reflecting stage, rotation and direction adjustment stage, and computing stage) is proceeded. When a closed chain is traced back to the starting point, the *LSD* computation is completed. Therefore, the coding process and the symmetry calculation are simultaneously completed in one pass.



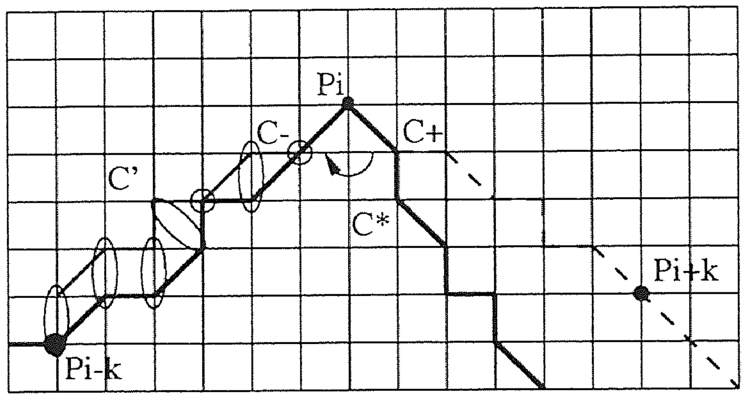
(a)

(k = 7)



(b)

(k = 7)



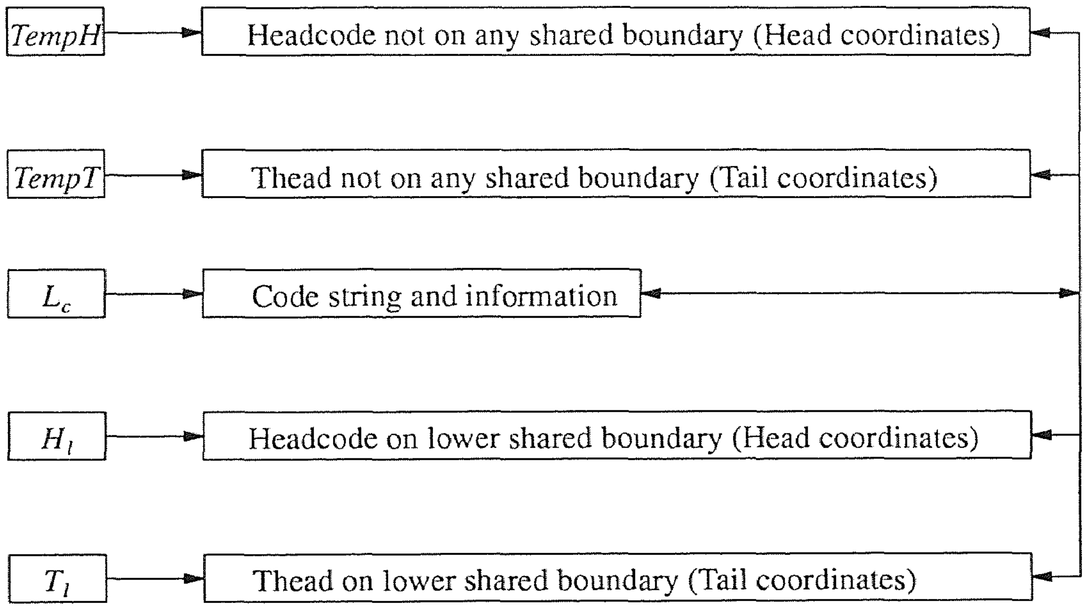
(c)

Figure 4.1 (a) Digital curve C. **(b)** Curve C* obtained from reflecting the local region C+ of length k after Pi in curve C. **(c)** Curve C' obtained from rotating curve C* to align it with the local region C- of length k before Pi in curve C.

This chapter is organized as follows: In Section 4.2, the new one-pass chain coding algorithm is introduced. In Section 4.3, the one-pass algorithm for local symmetry is presented. In Section 4.4, parallelism of the one-pass k-symmetry algorithm is discussed. In Section 4.5, experimental results are given. Finally in Section 4.6, summary is made.

4.2 Proposed One-pass Algorithm for Chain Coding

The new one-pass algorithm [28, 29] for chain coding is an extension to our previous work in Section 1.2 with the using of the link-list structures presented in Section 3.2. Five link-lists shown in Figure 4.2 are used to describe the open chain set. Since the input image is processed in the raster-scan fashion, i.e. left-to-right and top-to-bottom, all the end points of open chains are located at the lower boundary. The coding operation and merging operation are the same as those in Section 3.2. We will not repeat them here.



TempH
Headcode not on any shared boundary (Head coordinates)
TempT
Thead not on any shared boundary (Tail coordinates)
L_c
Code string and information
H_l
Headcode on lower shared boundary (Head coordinates)
T_l
Thead on lower shared boundary (Tail coordinates)

Figure 4.2 The link-list structures for a chain set

4.3 The One-pass Algorithm for Local Symmetry

When a codelink is added to a chain, there are four cases of concatenation: it is (1) the first element of an open chain, (2) the last element of an open chain, (3) the joint of two open chains, or (4) the joint of an open chain to make it closed. According to the position of the new added codelink and the length of the chain, we can decompose and re-arrange the *LSD* computation.

The chain code description for the external boundary of the object in Figure 4.3 is $(3,3)$ 000666444222, where the coordinates $(column, row)$ denotes the location of the starting pixel and the posterior numbers express the chain codes of the contour. The chain codes are recorded from the end of the contour to the beginning when the contour is traversed clockwise for internal one and counterclockwise for external one. Therefore, if a codelink is added to an open chain as the first element, its chain code is the head of the chain. In the same way, if a codelink is added to an open chain as the last element, its chain code is the tail of the chain. Regarding to the k-symmetry computation, the exiting curve of a point is the leading (head) portion of a chain referring to that point. The arriving curve of a point is the following (tail) portion of a chain referring to that point. If a codelink is added as the first element of a chain, it is added to the head of an exiting curve. On the other hand, if a codelink is added as the last element of a chain, it is added to the tail of the arriving curve.

When a codelink is added to the head of the chain, we calculate the adjustment of first element's rotation and direction, the first reflected chain code (same as the chain code of the codelink), the first rotated chain code (same as the chain code of the second element) and its coordinates after rotation. Then, the first element is used to calculate the second element's reflection rate (1-curvature at the second element) and the second reflected chain code. If the rotation and direction adjustment are known (if the third element exists, this value is computed when the third element is added to the chain), its second rotated chain code and coordinates after rotation can be computed. If the fourth element exists, the point-to-point distance can be proceeded (based on the second element, the second reflected chain code is rotated and the rotated coordinates are used to compute the distance between it and the fourth element). The computation is continued until all the elements are processed or the scope k is reached.

When a codelink is added to the tail of the chain, we calculate the last element's reflection rate (1-curvature). Then, we compute all the reflected chain codes in the scope k of the last element. Then for all precedent (ancestor) elements in the scope k, we compute the n-th rotated chain code with respect to the last element. For example, we compute the second rotated chain code for the element next to the last one. The point-to-point distance can be proceeded at the same time.

When a codelink serves as a joint between two chains, the codelink is added to the tail of the first chain which serves as the head of the resulting chain. Then, the second chain is joined to the first chain. When two chains are joined together, the *LSD* computed for both the leading portion before the joint and the following portion after the joint. The final case can be derived in the same concept. The result of the external boundary in Figure 4.3 is shown in Table 4.1.

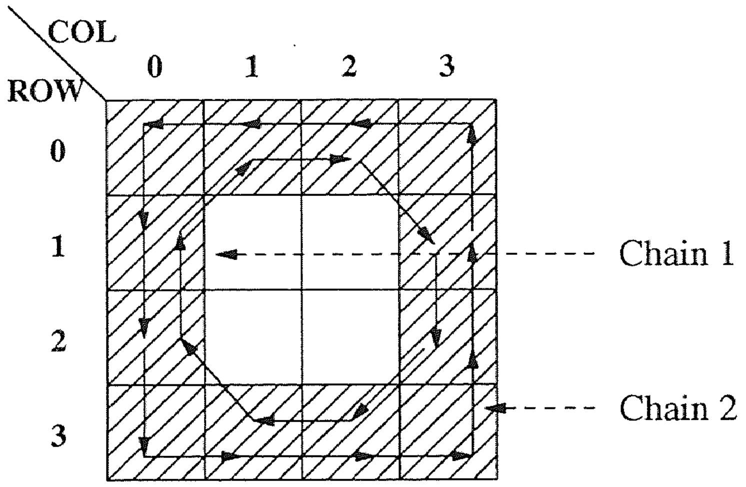
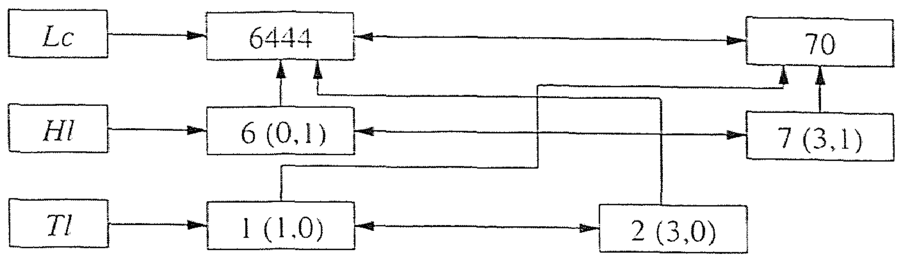
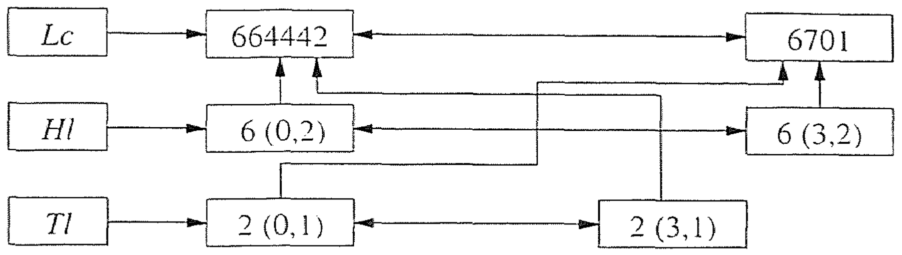


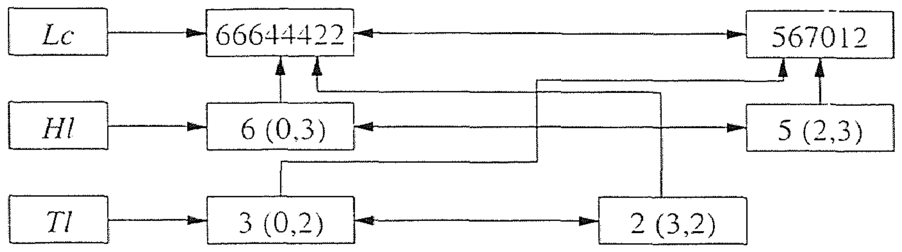
Figure 4.3 An input image with two chains



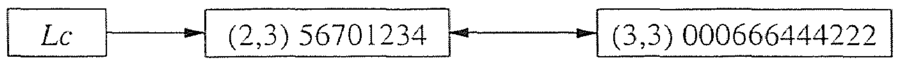
(After ROW 0 is processed)



(After ROW 1 is processed)



(After ROW 2 is processed)



(After ROW 3 is processed)

Figure 4.4 The coding result for Figure 4.3

Table 4.1 The result of external boundary in Figure 4.3

Row	Codes string	chain code	row	col	reflect	rotate	refl 0	refl 1	refl 2	rot 0	rot 1	rot 2	LSD $k = 3$
0	6444	6	0	0		2	6			0			0
		4	1	0	2	0	4	2		0	6		0
		4	2	0	0	0	4	4	2	0	0	0	0
		4	3	0	0		4	4	4				0
1	664442	6	0	1		0	6			2			0
		6	0	0	0	2	6	6		0	0		0
		4	1	0	2	0	4	2	2	0	6	6	2.83
		4	2	0	0	0	4	4	2	0	0		1.41
		4	3	0	0	2	4	4	4	6			0
		2	3	1	2		2	0	0				0
2	666444	6	0	2		0	6			2			0
	22	6	0	1	0	0	6	6		2	2		1.41
		6	0	0	0	2	6	6	6	0	0	0	0
		4	1	0	2	0	4	2	2	0	6	6	2.83
		4	2	0	0	0	4	4	2	0	0	6	2.83
		4	3	0	0	2	4	4	4	6	6	0	0
		2	3	1	2	0	2	0	0	6	0	0	0
		2	3	2	0		2	2	0				0
3	000666	0	2	3	2	0	0	6	6	4	2	2	2.83
	444222	0	1	3	0	0	0	0	6	4	4	2	2.83
		0	0	3	0	2	0	0	0	2	2	2	0
		6	0	2	2	0	6	4	4	2	0	0	2.83
		6	0	1	0	0	6	6	4	2	2	0	2.83
		6	0	0	0	2	6	6	6	0	0	0	0
		4	1	0	2	0	4	2	2	0	6	6	2.83
		4	2	0	0	0	4	4	2	0	0	6	2.83
		4	3	0	0	2	4	4	4	6	6	6	0
		2	3	1	2	0	2	0	0	6	4	4	2.83
		2	3	2	0	0	2	2	0	6	6	4	2.83
		2	3	3	0	2	2	2	2	4	4	4	0

row: row number processed

reflect: reflection adjustment (1st curvature) with respect to p_i for k segment

rotate: rotation adjustment with respect to P_i for k segments

refl 0,1,2: reflected chain code for $k = 0, 1, 2$ with respect to p_i

rotate 0,1,2: rotated chain code for $k = 0, 1, 2$ with respect to p_i

4.4 Parallelism of the One-pass K-Symmetry Algorithm

Following the concept in Chapter 3, we can implement the k-symmetry algorithm in parallel. Here, we present the parallelism for our one-pass k-symmetry algorithm in N-cube architecture. For example, an $n \times n$ input image is partitioned into n rows of $1 \times n$ sub-images. Except the sub-images in the first and the last row, others will have two shared boundaries: one shares with the upper neighbor and the other shares with the lower neighbor. To describe the open chain sets in processing, two more link-lists are enough. Link-List H_u stores the Head coordinates and the first code of a chain boundary, and link-list T_u stores the Tail coordinates and the last code of a chain. The subscript u denotes the link-list to be used for the upper shared boundary. As in Section 3.2, the algorithm contains two stages, coding stage and merging stage. Each processor takes care of one row of the image in the first stage. In the coding stage, each row is read in parallel and the chain code extraction is performed simultaneously. When codes are chained together, the *LSD* computation is performed. After each sub-image finishes the coding stage, we merge chains of each processor following the routing function described before.

The algorithm described above is just an example. Other parallel computation models such as mesh of trees networks and pyramid architectures are under investigation. Since the embedding process is only related to elements on the k-scope (k is far less than the size of the object contour), it will not affect the overall computation complexity of the chain coding process. If an optimal parallel chain coding algorithm is developed, we will obtain in the same way an optimal parallel k-symmetry algorithm.

4.5 Experimental Results

An image containing digits 9 and 6 is shown in Figure 4.5. The chain code description for the internal boundary (chain 1) and the external boundary (chain 2) for digit 9 are $(2,4)$ 55670012234 and $(2,7)$ 00445567107665443222211, respectively. For digit 6, that of the internal boundary (chain 3) is $(11,7)$ 566700122344 and that of the external boundary (chain 4) is $(11,7)$ 00766665644370113443221. The results of *LSD* computation with $k = 3$ are shown in Figure 4.6, where the horizontal axis is the code sequence from left to right and the vertical axis is the *LSD* value.

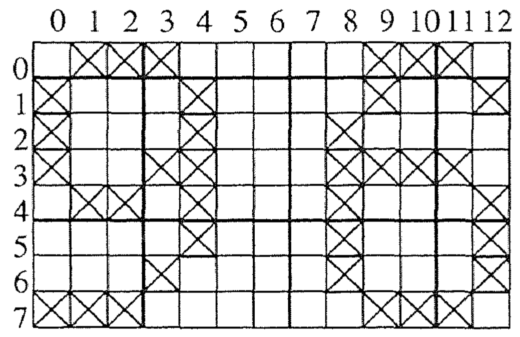
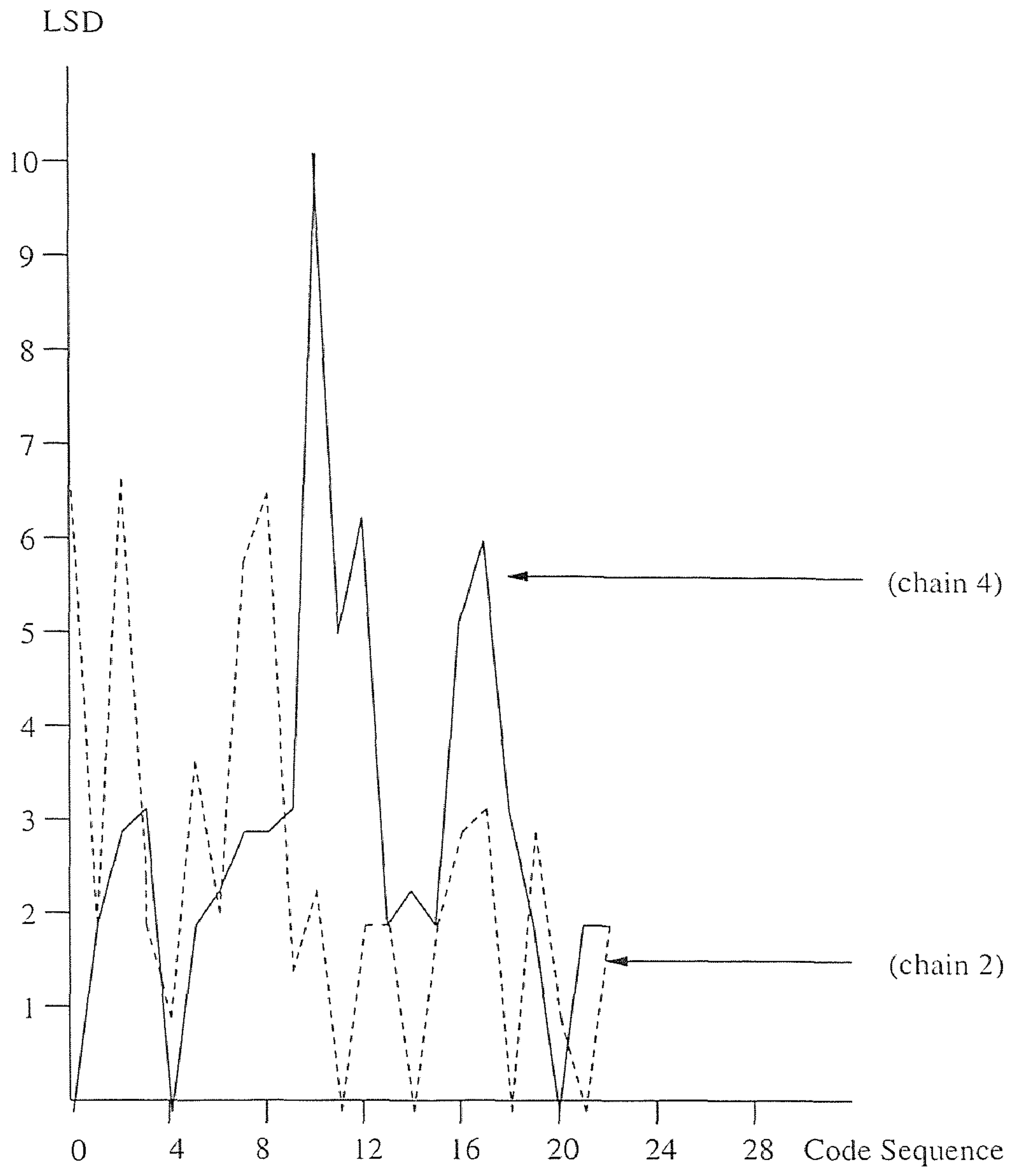


Figure 4.5 An input image contains two digits, 9 and 6



0044 5567 1076 6544 3222 211 (chain 2)

0076 6665 6443 7011 3443 211 (chain 4)

Figure 4.6 Result of LSD Computation for the external boundaries in Figure 4.5

4.6 Summary

A new one-pass chain coding algorithm is presented. A link-list data structure and *LSD* computation to calculate the local k-symmetry are adopted. We can finish the code extraction and the *LSD* computation in one pass. The new algorithm is suitable for parallel implementation. Since the *LSD* computation is only for k-scope, if the time complexity for the chain code extraction is dominated, the *LSD* computation will not affect the efficiency. Therefore, if we have an optimal parallel chain coding algorithm, we will have an optimal parallel k-symmetry algorithm. The embedding process is not limited to the k-symmetry algorithm. Currently, we are investigating the impeding of the corner detection algorithm [12] and the line segment identifying algorithm [14]. The proposed parallel processing approach can be viewed as a parallelization paradigm - a template to embed image processing algorithms in the chain coding process and to implement them in parallel approach.

CHAPTER 5

AN ADAPTIVE CONVERSION ALGORITHM FROM QUADTREE TO CHAIN CODES

5.1 Introduction

Representation and manipulation of digital images are two important issues in image processing, pattern recognition, pictorial database, computer graphics, geographic information systems, and other related applications. Quadtree is one of the compact hierarchical data structures for representing a binary image [30] . It is constructed by successively subdividing the image into four equal-size sub-images in the NW (northwest), NE (northeast), SW (southwest), and SE(southeast) quadrants. A homogeneously colored quadrant of the image is represented by a leaf node in the tree. Otherwise, the quadrant is represented by an internal node, and further divided into four sub-quadrants until each subquadrant has the homogeneous color. The leaf node with a black (white) color is called the black (white) node and the internal node is called the gray node. An example is shown in Figure 5.1. There are two widely used representations of quadtrees. A pointer-based quadtree uses the standard tree representation. A linear quadtree can be either a preorder traversal of the nodes of a quadtree or the sorted sequence (with respect to the preorder of the tree) of the quadtree's leaves.

The quadtree's data structure can be used as the representation of maps in the experimental Geographic Information Systems (GIS) successfully [31]. A fundamental operation for GIS is to overlay two maps. The results of the operation may be the union, intersection, or difference of the overlaid maps. Recently, researchers are interested in the parallel computational models of the quadtree algorithms [32, 33].

Another widely used method to represent digital images is contour representation. Freeman chain code [1] is one of the common coding techniques which we have discussed in previous chapters. Most recent article for the chain coding algorithm, by us [7, 23] which is to extract the chain codes first and link them together. Since the quadtree region representation and contour representation have different computational advantages, it is of interest to develop methods of converting from one representation to the other. For example, quadtree is convenience to represent object with region but not for linear object. However, chain coding is suitable for that. Previous algorithms for chain code to quadtree conversion are found in [35, 36]. The conversion from quadtree to chain code has also been reported by Dyer, et al. [37] and Kumar, et al. [38]. The latter one is the improvement of the earlier one. Both algorithms use the same approach which is shown as follows:

1. Find the initial starting node.
2. Extract the chain code of the current node.
3. Search the neighbor of the node in Step 2.
4. Mark the edge between two nodes in Step 2 and Step 3. If the edge has been visited, the process is terminated. Otherwise, go back to Step 2.

Our adaptive conversion algorithm from quadtree to chain codes is based on the 8-neighbor traversal algorithm presented by Fuhrman [39]. (A 4-neighbor traverse version can be found in the same article.) When the traversal algorithm reaches a node at the lowest level, we perform the chain coding process if the pixel is black. Chains are generated for each object pixel. When the traversal algorithm returns from the lowest level back to the root, we perform a merging process for chains that come from different nodes in each level. For example, an internal node will have four children and we merge all chains that come from them. The result of the merging process is a chain set. Then we return the resulting chain set to the parent of this internal node. Finally, we will have a chain set which represents the whole image when we reach the root node.

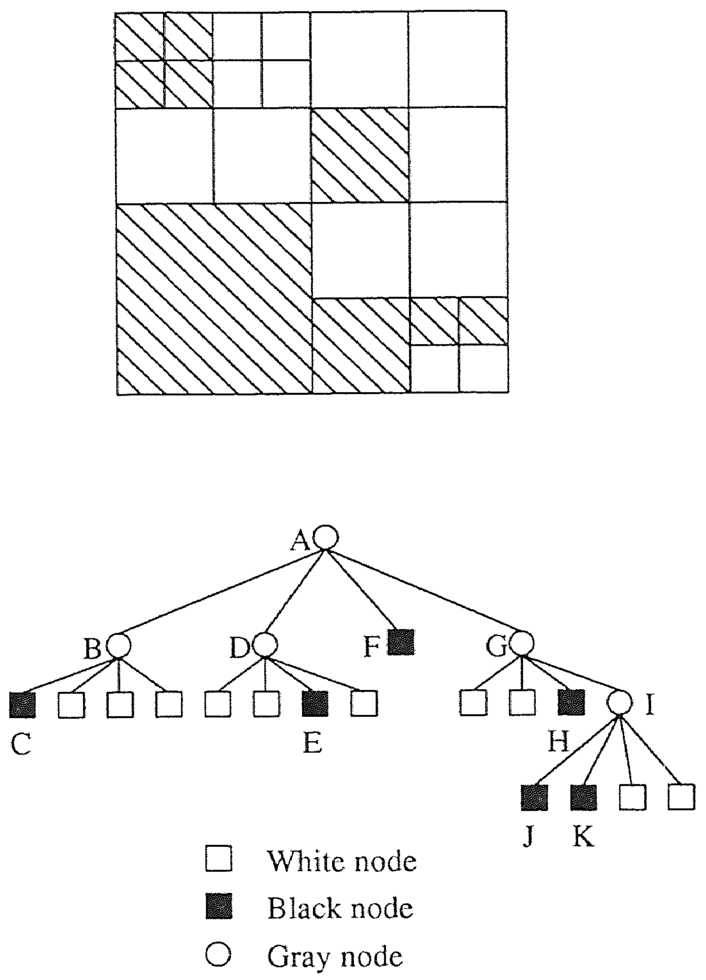


Figure 5.1 An image and its quadtree representation

Unlike the quadtree to chain code conversion algorithms proposed before, this new algorithm has the capability to process multiple regions of the input image without any extra effort. If we want to get the chain codes information of a result from the union, intersection, or difference of two similar quadtrees, this algorithm has the capability to reconstruct the new chain code information from the original one. This algorithm can be easily parallelized in pyramid architecture since it is based on recursive function call from one node to its 4 sub-quadrants.

In Section 5.2, we will present the traversal algorithm for the pointer-based quadtree. Chain coding for the traversal algorithm is briefly described in Section 5.3. In described in Section 5.3. In Section 5.4, we discuss the adaptive conversion algorithm based on the chain code contour reconstruction. An example is illustrated in Section 5.5. Analysis of the algorithm is given in Section 5.6. Finally, conclusions are made in Section 5.7.

5.2 Traversal Algorithm for Pointer-Based Quadtree

Samet [40] presented a generic top-down quadtree traversal algorithm in which each node and all 8 neighbors (orthogonally and diagonally adjacent) are visited in preorder. Fuhrmann presented a simple variation of Samet's algorithm which is written in C language style and shown as follows.

```
struct NODE {
    char color;
    char NODE *nw, *ne, *se, *sw;
};

traverse(root,w,n,e,s,nwc,nec,sec,swc)
struct NODE *root, *w, *n, *e, *nwc, *nec, *sec, *swc;
{
 if (root->color == GRAY) {
   /* traverse the northwest son of root */
   traverse(root->nw, w->ne, n->sw, root->ne, root->sw,
              nwc->se, n->se, root->se, w->se);

   /* traverse the northeast son of root */
   traverse(root->ne, root->nw, n->se, e->nw, root->se,
              n->sw, nec->sw, e->sw, root->sw);

   /* traverse the southwest son of root */
   traverse(root->sw, w->se, root->nw, root->se, s->nw,
              w->ne, root->ne, s->ne, swc->ne);

   /* traverse the southeast son of root */
   traverse(root->se, root->sw, root->ne, e->sw, s->ne,
              root->nw, e->nw, sec->nw, s->nw);
 }
 else
   functioncall();
}
```

This algorithm descends the quadtree by recursively calling the function traverse with 9 parameters which are the current node pointer (*root*) and the pointers of its 8 neighbors (*w*, *n*, *e*, *s*, *nwc*, *nec*, *sec*, *swc* ; where *w* is west, *n* is north, *e* is east, *s* is south and *c* is a mnemonic for corner). Once the traversal step reaches the leaf node, the procedure *functioncall()* is performed. The tracking for the 8 neighbors of the SE child of the most north-west quadrant in a 32 × 32 image is shown in Table 5.1. Here, we revise the quadtree traversal algorithm described above and the result is suitable for the chain coding process.

```
/* return a chain set */
CHAIN *traverse(
        /* the left upper corner and the right lower corner coordinates */
    int x1, y1, x2, y2,
        /* parent, pseudo parent and child type */
        ptype, pptype, ctype,
        /* grand parent node's color, current level and tree level */
        grandparentcolor, current_level, tree_level,
        /* current root node and its 8-neighbors */
    NODE root,w,n,e,s,nwc,nec,sec,swc)
{
 /* new node pointers for the next traverse */
 NODE *rootp, *wp, *np, *ep, *nwcp, *necp, *secp, *swcp;
 /* chain sets for the four children of the current root node
   and the resulting chain set from the merging or chain coding. */
 CHAIN *link_nw, *link_ne, *link_sw, *link_se, *flink;

 /* If it is a white node, no need to go further */
 if (root->color == WHITE) {
  return(DUMMY_EMPTY_CHAIN);
 }

 /* If the current level is not equal to the tree level, continue
   the traverse */
 if (current_level != tree_level) {
  /* pseudo parent type assignment as described in Table 5.2 */
  if ((ptype == NW_TYPE || pptype == NW_TYPE) && ctype == SW_TYPE)
    pptype=NW_TYPE;
  else if ((ptype == SW_TYPE || pptype == SW_TYPE) && ctype == NW_TYPE)
    pptype=SW_TYPE;
  else if ((ptype == NE_TYPE || pptype == NE_TYPE) && ctype == SE_TYPE)
    pptype=NE_TYPE;
  else if ((ptype == SE_TYPE || pptype == SE_TYPE) && ctype == NE_TYPE)
    pptype=SE_TYPE;
  else
    pptype=ctype;

  /* traverse the NW child */
  if (gpcolor == BLACK && ctype == SE_TYPE) {
   /* If its grandparent is a black node, its parent is SE_TYPE,
     and it is a NW child, then no contour will be generated.
     No need to go further */
   link_nw=DUMMY_EMPTY_CHAIN;
  }
  else {
   /* assign new node pointers for the NW child */
   nw_child_assign(root, w, n, e, s, nwc, nec, sec, swc,
     &rootp, &wp, &np, &ep, &sp, &nwcp, &necp, &secp, &swcp);

   link_nw=traverse($x_1$, $y_1$, ($x_1$ + $x_2$)/2, ($y_1$ + $y_2$)/2,
               ctype, pptype, NW_TYPE,
               current_level+1, tree_level,
               rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }

  /* traverse the NE child */
  if (gpcolor == BLACK && ctype == SW_TYPE) {
   link_ne=DUMMY_EMPTY_CHAIN;
  }
  else {
   ne_child_assign(root, w, n, e, s, nwc, nec, sec, swc,
     &rootp, &wp, &np, &ep, &sp, &nwcp, &necp, &secp, &swcp);

   link_ne=traverse(($x_1$ + $x_2$)/2, $y_1$, $x_2$, ($y_1$ + $y_2$)/2,
               ctype, pptype, NE_TYPE,
               current_level+1, tree_level,
               rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }

  /* traverse the SW child */
  if (gpcolor == BLACK && ctype == NE_TYPE) {
   link_sw=DUMMY_EMPTY_CHAIN;
  }
  else {
   sw_child_assign(root, w, n, e, s, nwc, nec, sec, swc,
     &rootp, &wp, &np, &ep, &sp, &nwcp, &necp, &secp, &swcp);

  link_sw=traverse($x_1$, ($y_1$ + $y_2$)/2, ($x_1$ + $x_2$)/2, $y_2$,
              ctype, pptype, SW_TYPE,
              current_level+1, tree_level,
              rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
  }


  /* traverse the SW child */
  if (gpcolor == BLACK && ctype == NE_TYPE) {
   link_sw=DUMMY_EMPTY_CHAIN;
  }
  else {
   se_child_assign(root, w, n, e, s, nwc, nec, sec, swc,
     &rootp, &wp, &np, &ep, &sp, &nwcp, &necp, &secp, &swcp);

   link_se=traverse(($x_1$ + $x_2$)/2, ($y_1$ + $y_2$)/2, $x_2$, $y_2$,
              ctype, pptype, SE_TYPE,
                current_level+1, tree_level,
                rootp, wp, np, ep, sp, nwcp, necp, secp, swcp);
    }

    /* merge the chain sets from the four children */
    flink=merging(link_nw, link_ne, link_sw, link_se);
   }
   else {
    /* perform the chain coding */
    flink=chain_coding(root, w, n, e, s, nwc, nec, sec, swc, x2, y2);
   }

   return(flink);
  }

/* follow Fuhrmann's traverse algorithm to assign 8-neighbor
  to the NW child */
void nw_child_assign(
       /* input nodes */
   NODE *root, *w, *n, *e, *s, *nwc, *nec, *sec, *swc,
       /* output nodes */
       *rootp, *wp, *np, *ep, *sp, *nwcp, *necp, *secp, *swcp;
{
 if (root->color != GRAY)
  *rootp=*ep=*sp=*secp=root;
 else {
  *rootp=root->nw;
  *ep=root->ne;
  *sp=root->sw;
  *secp=root->se;
 }

 if (w == NULL)
  *wp=*swcp=DUMMY_WHITE_NODE;
 else {
  if (w->color != GRAY)
   *wp=*swcp=w;
  else {
   *wp=w->ne;
   *swcp=w->se;
  }
 }

 if (n == NULL)
  *np=*necp=DUMMY_WHITE_NODE;
 else {
  if (n->color != GRAY)
   *np=*necp=n;
  else {
   *np=n->sw;
   *necp=n->se;
  }
 }
 if (nwc == NULL)
   *nwcp=DUMMY_WHITE_NODE;
 else {
   if (nwc->color != GRAY)
     *nwcp=nwc;
   else
     *nwcp=nec->se;
 }
}
```

The revised version of the quadtree traversal algorithm use the "virtual" leaves of the tree by "pretending" that the child of a leaf is a leaf of the same color. Even though a node does not have a neighbor, this algorithm provides a virtual neighbor which preserves the correct color. For example, a leaf node which represents a pixel in the southern boundary of an image has no south neighborhood. This traversal algorithm will pretend that it has "white" color neighbors in s, swc and sec. The assignment for a node's 8 neighbors is done in the *xx_child_assign* procedure, where xx is nw, ne, sw and se. If the color of a neighbor node is not gray, we use the same node instead for there is no descendants of that node. During the traverse, if it is a white node, the further traverse will be skipped for there is no contour generation. If it is a black node and its grandparent's color is black, then this black node may not generate contour. The criteria is quite simple and describe as follows:

1. The black node is NW_TYPE and its parent is SE_TYPE.
2. The black node is NE_TYPE and its parent is SW_TYPE.
3. The black node is SW_TYPE and its parent is NE_TYPE.
4. The black node is SE_TYPE and its parent is NW_TYPE.

Therefore, this algorithm will only traverse the black nodes which lie on the boundary of objects. When the traverse reaches the deepest tree level, the *chain_coding* procedure is performed. The location (coordinates) of the pixel is computed and passed down from each traversal step. The X-coordinate is equal to x_2 and the Y-coordinate is equal to y_2. The coordinates start from left to right and top to bottom. The coding procedure will be performed only on the leaf nodes including all virtual leaf nodes of the quadtree at the tree level (deepest one). For any other node, the *merging* procedure is performed. The input of the *merging* procedure is the coding result or the merging result of the subtree of that node.

5.3 Chain Coding for Quadtree Traversal Algorithm

To convert from the quadtree representation to chain codes, the link-list typed data structures which are convenient in the merging process are used. The 9 link-list data structures shown in Figure 5.2 are used to describe a chain set. There are four shared boundaries for a sub-image in a quadrant: north, south, east and west boundaries. Link-List H_n, H_s, H_e and H_w store the Head coordinates and the Headcode of a chain. Link-List T_n, T_s, T_e, and T_w store the Tail coordinates and the Thead of a chain. The subscripts n, s, e and w denotes the north (upper), south (lower), east (right), and west (left) boundaries. The last link-list L_c is for the code strings and information. The subscript c denotes the chain.

When a codelink is created, we must find out which shared boundary its Head and Tail are located on. Later, the merging procedure can be performed correctly. When the codelink is 0, it points to the east direction. It will connect to a chain whose Thead is located on its east shared boundary. Therefore, the Head coordinates of the codelink are added to H_e. When the codelink is 4, it points to the west direction. It will connect to a chain whose Thead is located on its west shared boundary. Therefore, its Head coordinates are added to H_w. From the similar concept, if the codelink is 2, we add them to H_n. If the codelink is 6, we add them to H_s. When the codelink is 1, 3, 5, or 7, it points to the northeast, northwest, southwest and southeast direction, respectively. The shared boundary type where the connection occurs will depend on its pixel location. Before an example illustration, we need to describe the order in the merging procedure to make it easier to understand the assignment of the Head coordinates. As aforementioned, a non-leaf node has four children. When a non-leaf node is merged, we first merge the sub-trees of its *nw* and *ne* children. The merging direction is the x-direction. Then, it follows by the merging the sub-trees of its *sw* and *se* children. The merging direction is the y-direction.

If the codelink is 1, it points to the northeast direction. If the pixel which generates the codelink is a *SE* child in a quadrant, the codelink will connect on the eastern shared boundary. Therefore, its Head coordinates are added to H_e. If the pixel which generates the codelink is a *NW* or *SW* child in a quadrant, the codelink will connect on the northern shared boundary. Therefore, its Head coordinates are added to H_n. If the pixel which generates the codelink is a *NE* child in a quadrant, the codelink will connect whether on the eastern shared boundary or the northern boundary. The exact shared boundary type where the connection will occur depends on the pairs which are the parent type or the pseudo parent type and the type of the node. The pseudo parent type is an inheritance which is based on some special combinations of the node and its parent. For example, if the type of a node is *SW* and that of its parent is *NE* but its pseudo parent type is *NW*, then it will pass down the parent type *SW* and pseudo parent type *NW* to its child. If the situation is not in these rules, the pseudo parent type is equal to the parent type. These rules are shown in Table 5.2. From the similar deduction, the shared boundaries where the Tail of codelinks are located on are classified. Both of them are shown in Table 5.3. Codelink "1" is used as an example to illustrate the classification in Figure 5.3. In the figure, we only show the *NW* portion (16×16) of a 32×32 input image. All other cases can be derived by the same concept.

The order of the codelink creation on a pixel has been shown in Figure 3.4. The order granted to each codelink generated on a pixel will make the Head and the Tail in order in the link list as many as it could be. From the observation, some particular cases would not produce the resulting H_e, H_w, T_n or T_s in order. An extra swapping procedure is needed for the adjustment. The swapping procedure is to swap the two related Heads or Tails after the codelinks creation. It restores the sequence to make it in order. H_e or H_w

will not be in order when a pixel generates two codelinks and both Heads are on the same link list H_e or H_w. The reason is that we grant the creation of codelinks 5 and 7 the higher priority than 1 and 3. When 5 and 3 are assigned to H_w, or 7 and 1 are assigned to H_e for a pixel, the sequence of the Head coordinates will not be in order in the vertical direction. Therefore, the swapping procedure is necessary. We show all the exception cases in this category in Table 5.4. The other category is for T_n and T_s. The patterns which needs the extra swapping on T_n or T_s have been shown in Figure 3.5.

Table 5.1 The tracking of the traversal algorithm for the 8-neigbors of the SE child of the most northwest quadrant in a 32×32 image

Neighbor type	1st Stage	2nd Stage	3rd Stage	4th Stage
Central pixel	root->nw	root->nw->nw	root->nw->nw->nw	root->nw->nw->nw->se
West	Null	Null	Null	root->nw->nw->nw->sw
North	Null	Null	Null	root->nw->nw->nw->ne
East	root->nw	root->nw->ne	root->nw->nw->ne	root->nw->nw->ne->sw
South	root->sw	root->nw->sw	root->nw->nw->sw	root->nw->nw->sw->ne
NW Corner	Null	Null	Null	root->nw->nw->nw->nw
NE Corner	Null	Null	Null	root->nw->nw->ne->nw
SE Corner	root->se	root->nw->se	root->nw->nw->se	root->nw->nw->se->nw
SW Corner	Null	Null	Null	root->nw->nw->sw->nw

Table 5.2 The generation rules of the pseudo parent type of a node

Parent type or Pseudo parent type	Node Type	Pseudo parent type passed down to child
NW	SW	NW
NE	SE	NE
SW	NW	SW
SE	NE	SE

Table 5.3 Classification of shared boundary ocurrance for the Head and Tail of a codelink

Headcode:Thead	Child type in a quadrant	Parent or pseudo parent type	Add to Shared boundary
0;0	NW,NE,SW,SE	Don't care	$H_e;T_w$
1;5	NW,SW	Don't care	$H_n;T_n$
	NE	NW,NE,SW	$H_n;T_n$
	NE	SE	$H_e;T_e$
	SE	Don't care	$H_e;T_e$
2;2	NW,SW,SW,SE	Don't care	$H_n;T_s$
3;7	NE,SE	Don't care	$H_n;T_n$
	NW	NW,NE,SE	$H_n;T_n$
	NW	SW	$H_w;T_w$
	SW	Don't care	$H_w;T_w$
4;4	NW,NE,SW,SE	Don't care	$H_w;T_e$
5;1	NE,SE	Don't care	$H_s;T_s$
	SW	NE,SW,SE	$H_s;T_s$
	SW	NW	$H_w;T_w$
	NW	Don't care	$H_w;T_w$
6;6	NW,NE,SW,SE	Don't care	$H_s;T_n$
7;3	NW,SW	Don't care	$H_s;T_s$
	SE	NW,SW,SE	$H_s;T_s$
	SE	NE	$H_e;T_e$
	NE	Don't care	$H_e;T_e$

Table 5.4 The particular cases of the two Heads generated on a pixel will not be in order

Headcode	Child type in a quadrant	Parent or pseudo parent type
3 & 5	NW	SW
	SW	NW
1 & 7	NE	SE
	SE	NE

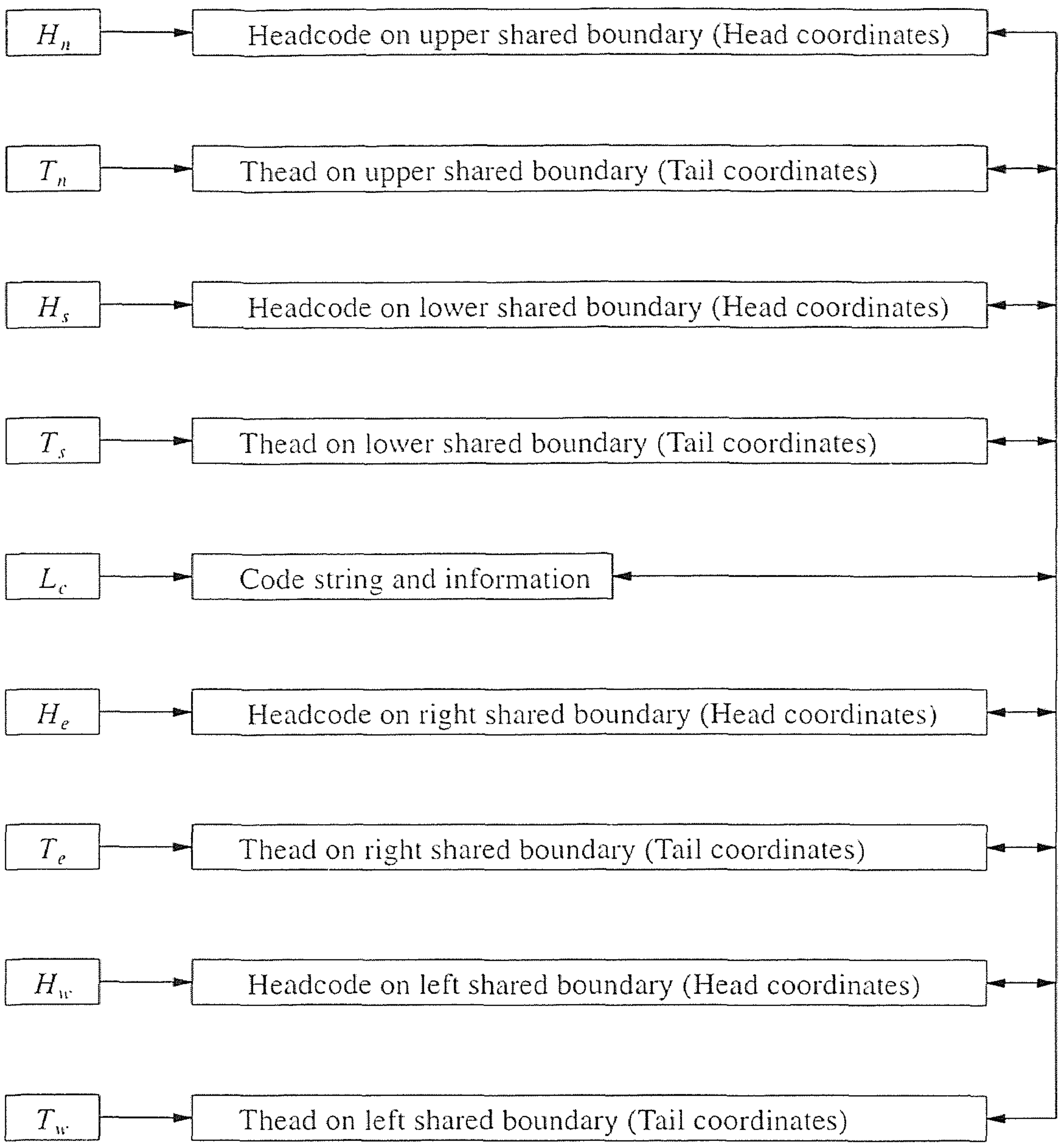
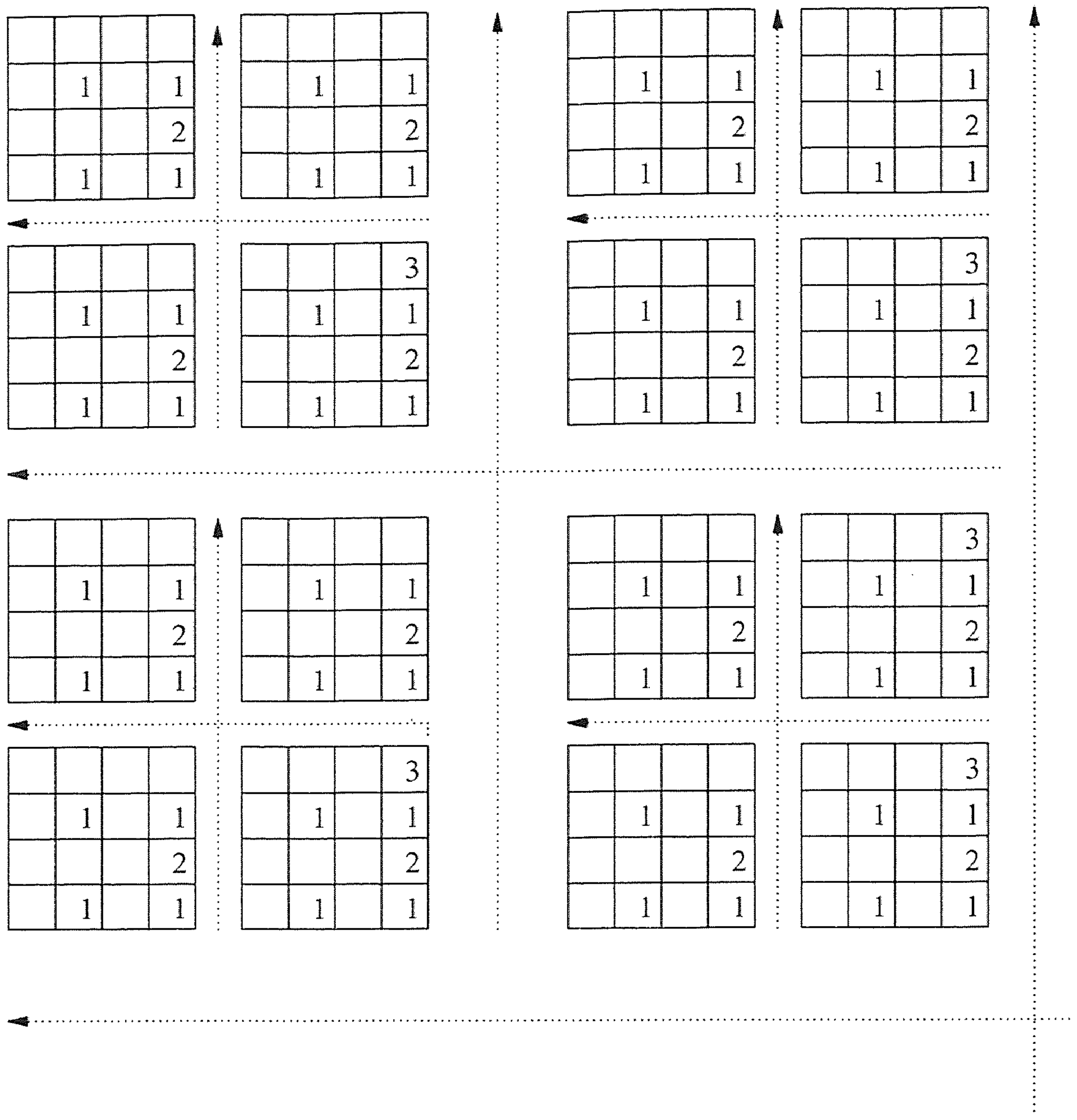

H_n
Headcode on upper shared boundary (Head coordinates)
T_n
Thead on upper shared boundary (Tail coordinates)
H_s
Headcode on lower shared boundary (Head coordinates)
T_s
Thead on lower shared boundary (Tail coordinates)
L_c
Code string and information
H_e
Headcode on right shared boundary (Head coordinates)
T_e
Thead on right shared boundary (Tail coordinates)
H_w
Headcode on left shared boundary (Head coordinates)
T_w
Thead on left shared boundary (Tail coordinates)

Figure 5.2 The link-list data structure for a chain set



1: derived from the node type which is equal to SE.
2: derived from the node type which is NE and its parent type which is SE
3: derived from the node type which is NE and its pseudo parent type which is SE

if a node is marked 1, 2 or 3 and the codelink is 1, the head of codelink will be He
Otherwise, it will be Hn.

Figure 5.3 An exmaple to illustrate the classification of the location for the Head and Tail of codelink 1

By using this approach, the Head and Tail coordinates of the open chains are in sequence in the link-list structure. When the quadtree traversal algorithm in Section 5.2 reaches the leaf nodes, all 8 neighbors are known. From their color we can calculate the index value of the look-up table for that pixel (leaf node). Then the chain codes are extracted and placed into a chain set. When the chain coding procedures for all the children of the same parent are finished, we merge the chain sets together. Basically, when two subtrees, named A and B, are merged together, we will merge the Head of subtree A to that of B. Then it follows by the Tail of subtree A to that of B. If a chain is closed after merging, it is added into the chain set C of the link-list L_c. Otherwise, the open chain is added into the chain set O of the link-list L_c. The same process is performed recursively from the leaf nodes back to the root node.

5.4 Adaptive Level Based Chain Coding Retrieval

As aforementioned, the boolean operation of two quadtrees is a fundamental operation for GIS. If we need the chain coding information of the resulting quadtree, we must perform the chain coding procedure for the new constructed quadtree. If we compare the new quadtree and the old quadtree, it may not have a great difference. This task can be simplified by constructing the chain coding information of the new quadtree from that of the old quadtree.

The link-list typed data structure which we use to describe a subtree of a node is simple and easy to maintain. During the merging procedure, we can save the intermediate result of all the nodes which are smaller than a given level. This information will be retrieved when we want to find the chain code contour of the new constructed quadtree. This step will avoid lots of unnecessary re-coding processes. During the boolean operation of two quadtrees, if the color of a node in the original quadtree is changed, we will mark it and all its ancestors to be changed. After the boolean operation is finished, we perform the chain coding algorithm. If a node is not marked, we need not go further.

Instead, we retrieve the link-list information from what we have saved before. If a node is marked, we track its four children to see that if any of them is marked. This process is performed recursively until the leaf node is reached or all the four sons of the node are not marked.

Since it is impossible to save all the nodes' information, we can save a portion of nodes instead. By given a level number, we can save all the information of nodes which are less than or equal to that level number. During the chain coding of the new quadtree, if a node is not marked and its level number is less than or equal to the given level, we can retrieve their chain codes directly. If its level number is larger than the given level, we must track all its descendant. This approach is adaptive. If the hardware has plenty of memory, we should use a small level number. Otherwise we can use a bigger number.

5.5 Experimental Results

We use Figure 5.1 as an input image to apply our conversion algorithm. Recall that the chain coding process is performed on leaf nodes or virtue leaf nodes in a quadtree. The result of the merging process for each internal node and leaf with black color is shown in Table 5.5. C, E, F, J and K are leaves with black color but only J and K are individual pixels in the input image. B, D, G, H and I are internal nodes. Root node is marked as A which have two closed chains. They are represented as $(0,1)\,0246$ and $(3,4)\,444666000001044432102465$, where (x, y) are the coordinates of the starting pixel in the chain code description and the code sequence of the contour follows them. The code sequence is read from the left to right.

Table 5.5 The result of the merging process for each node in Figure 5.1

Node	Head	Tail	Starting Pixel	Chain Code Sequence
C			(0,1)	0246
B			(0,1)	0246
E	H_s (3,4)	T_s (4,3)		02465
D	H_s (3,4)	T_s (4,3)		02465
F	H_e (4,7) H_n (4,3)	T_n (3,4) T_e (3,5)		444666000 21
H	H_w (3,5) H_e (6,6)	T_e (5,6) T_w (4,7)		43 01
J	H_e (7,6) H_w (5,6)	T_w (6,6) T_e (6,6)		0 4
K	H_w (6,6)	T_w (7,6)		4
I	H_w (5,6)	T_w (6,6)		044
G	H_w (3,5)	T_w (4,7)		0104443
A			(0,1) (3,4)	0246 444666000001044432102465

5.6 Analysis

Let the size of the image be $N \times N$ and let the number of boundary pixels be $O(B)$. The height of the quadtree (include virtual leaves) is H, where $N = 2^H$. Each chain coding and merging operation costs time O(1). The time required to merge the chains of two quadrants is dependent on the number of joints in those two quadrants. There is no need to sort or search the two pairs of the joint (the boundary pixel) since the Heads and Tails are in order. In sequential mode, the total number of joints in the image is in the same order of boundary pixels and that is the time needed in the traverse of nodes. The complexity of our algorithm is dependent on the height of the traverse plus the operation of the boundary pixels. Therefore, it is $O(H + B)$.

For a pyramid architecture with $O(N)$ processors and each processor handles a sub quadrant from its parent, the traverse of the tree needs time $O(H)$ in parallel. The chain coding operation is $O(1)$. The merging operation in a processor of level l requires time which is the same order of the number of open chains. The worst case occurs when the order of open chains' number is equal to the order of the side length of the quadrant in level l. If the number of the pixels of a quadrant in level l is $n \times n$, the time for the merging operation will require $O(n)$. If we sum up the merging time for all levels in parallel which is $2^H + 2^{H-1} + \cdots + 2^2$, we get $O(2^H)$ in total and that is $O(N)$.

5.7 Summary

In this chapter, an algorithm for converting quadtree representation to chain code representation is given. Different from other algorithms which search the neighbor of the current node, this algorithm is based on traversing the quadtree. Because of this reason, it is recursive and has the parallelism. It is ready to implement in a pyramid architecture. Besides, we introduce an adaptive method to construct the chain codes for the resulting quadtree of the boolean operation of two quadtrees by re-using the original chain codes. This adaptive method speeds up the conversion in this kind of applications.

CHAPTER 6

SUMMARY AND FUTURE RESEARCH

This dissertation is aimed to investigate the parallel coding algorithm and its applications for binary image and multi-resolution image. In this chapter we summarize the contributions of this research and briefly describe our direction of future research.

6.1 Contribution and Summary of this Dissertation

In this dissertation, we improve the applicability, flexibility and efficiency of the parallel chain/mid-crack coding process. The importance of this dissertation is that we develop a simple approach which is suitable for the parallelization of image processing algorithms. This approach can be treated as a template for such parallelization tasks.

Three different algorithms for two parallel architectures are investigated. The time computation complexity of the *pyramid* architecture in Chapter 2 is $O(N^{3/2})$, that of N *cube* architecture in Chapter 3 is $O(N \log N)$ and that of the algorithm for *pyramid* architecture using the link-list structure in Chapter 5 is $O(N)$, where $O(N)$ processors are used for $N \times N$) input images. The communication time is not considered in this dissertation because it is varied with different parallel computer.

A one-pass algorithm for local symmetry of contours from the chain code representation is developed. It is an example of the parallelization paradigm for the presented parallel coding algorithm.

Finally, an algorithm for converting quadtree representation to chain code representation is given. It is ready to implement in a pyramid architecture.

6.2 Future Research

First, the parallel coding algorithm should demonstrate real time performance not only theoretically but practically.

Next, optimality should be concerned in the parallelism and the adaptive conversion algorithm.

Finally, the integration of the embedding processes for the parallel coding algorithm and the conversion algorithm is an interesting part to investigate.

REFERENCES

1. Freeman, H. "Computer processing of line drawing images." *Comput. Surveys*, vol. 6, no. 1, pp. 57-97, March 1974.

2. Freeman, H., and J. M. Glass. "On the quantization of line-drawing data." *IEEE Trans. Syst. Man Cybernet. (T-SMC)*, vol. 5, pp. 70-79, 1969.

3. Rosenfeld, A., and A. C. Kak. *Digital Picture Processing*, Vol. 2, Academic Press, San Diego, California, 1982.

4. Dunkelberger, K. A., and O. R. Mitchell. "Contour tracing for precision measurement." *Proc. IEEE Inter. Conf. Robotics and Automation*, St. Louis, pp. 22-27, 1985.

5. Koplowitz, J. "On the performance of chain codes for quantization of line drawings." *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, vol. 3, no. 2, pp. 180-185, March 1981.

6. Saghri, J. A., and H. Freeman. "Analysis of the precision of generalized chain codes for the representation of planar curves." *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, vol. 3, no. 5, pp. 533-539, September 1981.

7. Shih, F. Y., and W.-T. Wong. "A new single-pass algorithm for extracting the mid-crack codes of multiple regions." *Journal of Visual Commun. and Image Rep.*, vol. 3, no. 1, pp.217-224, March 1992.

8. Chang, L.-W., and K.-L. Leu. "A fast algorithm for the restoration of images based on chain codes description and its applications." *Comput. Vision, Graphics, and Image Process. (CVGIP)*, vol. 50, pp. 296-307, 1990.

9. Merrill, R. D. "Representation of Contours and Regions for efficient Computer Search." *Commun. of ACM*, vol. 16, no. 2, pp. 69-82, 1973.

10. Shih, F. Y., and W.-T. Wong. "An improved fast algorithm for the restoration of images based on chain codes description." *CVGIP: Graphical Models and Image Process.*, vol. 56, no. 4, pp. 348-351, July 1994.

11. Shih, F. Y., and W.-T. Wong. "Reconstruction of binary and gray-scale images from mid-crack code descriptions." *Journal of Visual Commun. and Image Rep.*, vol. 4, no. 2, pp. 121-129, June 1993.

12. Koplowitz, J. and S. Plante. "Corner detection for chain coded curves." *Pattern Recognition*, vol. 28, no. 6, pp. 843-852, 1995.

13. Iñesta, J. M., M. Buendĩa, and M. Ã. Sarti. "Local symmetries of digital contours from their chain codes." *Pattern Recognition*, vol. 29, no. 10, pp. 1737-1749, 1996.

14. Yuan, J., and C. Y. Suen. "An optimal *O*(*n*) algorithm for identifying Line segments from a sequence of Chain Codes." *Pattern Recognition*, vol. 28, no. 5, pp. 635-645, 1995.

15. Kwok, P. C. K. "A thinning algorithm by contour generation." *Comm. ACM*, vol. 31, no. 11, pp. 1314-1324, November 1988.

16. Vossepoel, A. M., Buys, J. P., and G. Koelewijn. "Skeletons from chain-coded contours" *Proc. IEEE Inter. Conf. Pattern Recognition*, pp. 70-73, Altantic City, New Jersey, 1990.

17. Xu, W., and C. Wang. "CGT : A fast thinning algorithm implemented on a sequential computer." *IEEE Trans. Syst. Man Cybern. (T-SMC)*, vol. 17, no. 5, pp. 847-851, September/October 1987.

18. Shih, F. Y., and W.-T. Wong. "A new safe-point thinning algorithm based on the mid-crack code tracing." *IEEE Trans. Syst. Man Cybern. (T-SMC)*, vol. 25, no. 5, pp. 370-378, February 1995.

19. Naccache, N. J., and R. Shinghal. "SPTA : A proposed algorithm for thinning binary patterns." *IEEE Trans. Syst. Man Cybern. (T-SMC)*, vol. 14, no. 3, pp. 409-418, May/June 1984.

20. Shih, F. Y., and W.-T. Wong. "Fully parallel thinning with tolerance to boundary noise." *Pattern Recognition*, vol. 27, no. 12, pp. 1677-1695, 1994.

21. Kim, S.-D., Lee, J.-H., and J.-K. Kim. "A new chain-coding algorithm for binary images using run-length codes." *Comput. Vision, Graphics, and Image Process. (CVGIP)*, vol. 41, pp. 114-128, 1988.

22. Dinstein, I., and G. M. Landau. "Parallel alogrithms for contour extraction and coding on EREW PRAM computer." *Pattern Recognition Letters*, vol. 11, no. 2, pp. 87-93, February 1990.

23. Wong, W.-T., Y.-L. Chen, and F. Y. Shih. "A fully parallel algorithm for the extraction of chain and mid-crack codes of multiple contours." Conf. Proc. of Inter. Comput. Symp., HsinChu, Taiwan, R.O.C., pp. 565-570, 1994.

24. JaJa, J. *An Introduction to Parallel Algorithms*, Addison Wesley, Reading, Massachusetts, 1992.

25. Leighton, F. T. *Introduction to Parallel Algorithms and Architectures: Arrays • Trees • Hypercubes*, Morgan Kaufmann, San Mateo, California, 1992.

26. Hwang K. and F. A. Briggs. *Computer Architecture and Parallel Processing*, Mc Graw Hill, New York City, New York, 1984.

27. Ogawa H. "Corner detection on digital curves based on local symmetry of the shape." *Pattern Recognition*, vol. 22, no. 4, pp. 351-356, 1989.

28. Shih, F. Y. and W.-T. Wong. "A one-pass algorithm for local symmetry of contours from chain code." 1st *Inter. Workshop on Comput. Vision, Pattern Recogn. and Image Process.*, 1998.

29. Shih, F. Y. and W.-T. Wong. "A one-pass algorithm for local symmetry of contours from chain code." accepted by *Pattern Recognition*, 1998.

30. Samet H. *Applications of Spatial Data Structures - Computer Graphics, Image Processing, and GIS*, Addison Wesley, Reading, Massachusetts, 1990.

31. T.-W. Lin. "Set operations on constant bit-length linear quadtrees." *Pattern Recognition*, vol. 30, no. 7, pp. 1239-1249, 1997.

32. Dehne F., A. Rau-Chaplin and A. G. Ferreira. "Hypercube algorithms for parallel processing of pointer-based quadtree." *Comput. Vision and Image Understanding*, vol. 62, no. 1, pp. 1-10, July 1995.

33. Y. Hung and A. Rosenfeld. "Parallel processing of linear quadtree on a mesh-connected computer," *J. Parallel Distrib. Comput.*, vol. 7, pp. 1-27, 1989.

34. Zingaretti, P., M. Gasparroni, and L. Vecci. "Fast Chain Coding of Region Boundaries." *IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI)*, vol. 20, no. 4, pp. 407-415, April 1998.

35. Samet H., "Region Representation: quadtree from boundary codes," *Commun. ACM*, vol. 23, no. 3, pp. 163-170, March 1980.

36. Lattanzi M. R. and C. A. Shaffer, "An optimal boundary to quadtree conversion algorithm," *CVGIP: Image Understanding*, vol. 53, no. 3, pp. 303-312, May 1991.

37. Dyer, C. R., A. Rosenfeld and H. Samet, "Region representation: boundary codes from quadtrees," *Commun. ACM*, vol. 23, no. 3, pp. 171-179, March 1980.

38. Kumar, G. N. and N. Nandhakumar, "Efficient object contour tracing in a quadtree encoded image," *SPIE Application of Artificial Intell. IX*, vol. 1468, pp. 884-895, 1991.

39. D. R. Fuhrmann. "Quadtree traversal algorithms for pointer-based and depth-first representation." IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI), vol. 10, no. 6, pp. 955-960, November 1988.

40. H. Samet. "A top-down quadtree traversal algorithm." IEEE Trans. Pattern Anal. Mach. Intell. (T-PAMI), vol. 7, no. 1, pp. 94-98, January 1985.