PARALLEL THINNING ALGORITHMS AND THEIR IMPLEMENTATION
ON HYPERCUBE MACHINE

A Thesis Presented to
The Faculty of the College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

by
Yi-Chang Xu
June, 1991

TABLE OF CONTENTS

	page
CHAPTER 1 Introduction	1
CHAPTER 2 Parallel Thinning Algorithms and Their Features	9
2.1 Basic Terminologies and Concepts Used in Thinning Algorithms	9
2.2 The Effect of Size for the Windows Used in Thinning Algorithms	13
2.3 Subiterations in an Iteration	16
2.4 Three Parallel Thinning Algorithms	17
2.4.1 Zhang and Suen's Thinning Algorithm	17
2.4.2 One Pass Thinning Algorithm	25
2.4.3 Guo and Hall's Thinning Algorithm	27
CHAPTER 3 Implementation of Thinning Algorithm on Hypercube Machine	32
3.1 Hypercube Machine	33
3.1.1 Hypercube Structure	33
3.1.2 Hypercube Machine	36
3.1.3 Concurrent Programs	38
3.1.4 Communications in Concurrent System	39
3.2 Vision Algorithms and Topology Selection	41
3.2.1 Local Operation Feature of Vision Algorithms	41

3.2.2	Thinning Algorithms and Topology Selection	42
3.2.3	Ring Topology	44
3.2.4	Nearest Neighbor Topology (Mesh Topology)	48
3.2.5	Comparison of Two Topologies Used in Concurrent System for Thinning Algorithms	51
3.3	Changeable Value Label Thinning	59
CHAPTER 4	Comparison and Analysis of Results of Experiments	66
4.1	Analysis and Comparison for the Results of Simple Input Patterns	66
4.2	Analysis and Comparison for the Results of Real Images	71
4.2.1	Experimental Environment and Procedure	71
4.2.2	Topological Analysis	76
4.2.3	Implementation Speed Analysis	95
CHAPTER 5	Conclusion	101
REFERENCE		104
APPENDIX		108

CHAPTER 1

Introduction

Thinning techniques play an important role in image processing and computer vision. It is a fundamental preprocessing step in many applications of image processing, computer vision, and pattern recognition.

In many computer vision applications, the objects interested in a scene can be characterized by structures composed of line or arc patterns for shape analysis. For example, the recognition of handwriting or printed characters, fingerprints, printed circuits, chromosomes and biological cell structures, etc[1][2]. In these applications, the thickness of the pattern strokes does not contribute to the recognition process. That's why we need to get a thinned picture before we recognize it using shape analysis.

Thinning can be defined as a process in which outer layers of an object are successively removed until a skeleton of the object is obtained. From the definition of thinning, we can see two obvious features of thinning: Firstly, thinning is an iterative processing. In each step, only outermost layer can be peeled. Secondly, the time used for thinning depends on the size and the shape of the objects in an image.

One advantage of thinning is to reduce the data required to represent the topological structure of a object. Because thinning is preprocessing of image, it also reduces the processing time in later steps of image processing.

As an important step of preprocessing of images for pattern recognition, thinning algorithms have been studied intensively for the last two decades. A number of thinning algorithms have been proposed[3]-[24]. There are some commonalities between the algorithms, but no two algorithms are identical. They differ from each other in using different conditions or templates to classify the deleted pixels and preserved pixels during the processing.

According to the procedure of thinning, thinning algorithms can be divided into two categories: sequential and parallel thinning algorithms. In sequential thinning algorithms, the pixels of an image are processed one by one successively and the operation depends on preceding processed results. On the other hand, parallel thinning algorithms can process all the pixels at the same time. There is no a special order for processing in parallel algorithms. Since the whole task can be divided into several subtasks being carried out at the same time separately in parallel processing, this approach of processing is much faster than sequential processing.

Although there are many thinning algorithms, these algorithms follow some basic principles, which are summarized as below:

1. Connectivity should not be changed for both object and background. That is, the topology of an object should not be changed after thinning.
2. The skeleton should roughly lie along the Medial Axis[2] of the object. That is, the thinning procedure should not cause excessive erosion.
3. End points of the object should not be deleted.

Actually, the item 3 is an extension of item 2. If we delete the end points of a line, the item 1 is not violated since the topology of the image is not changed. However, if the processing continues, the stroke will become shorter and shorter until only a single point is remained. We still don't change the topology of the image, but we can not get any useful information from this result. This can be called as excessive erosion.

The principles described above are not very exact. The concept of thinning is not easy to define mathematically. There is no exact definition for the skeleton of an object. Besides, there is not a standard for the so called "excessive erosion". That situation may cause some difficulties in evaluating and comparing different thinning algorithms, but

that does not lead to many difficulties in practice. Many proposed thinning algorithms find applications in different fields of computer vision.

Usually we use 8-connectivity and 4-connectivity to define the connectivity of object and background respectively in an image or vice versa. It is because if we use the same connectivity for both object and background the contradiction will appear. This can be illustrated by fig 1.1. That is a 3*3 subimage. In this fig. 0 is used to express object and B for background. Actually, fig 1.1 is a cross with the center missing. If we only use 4-connectivity in the fig, both the object and background are disconnected. Contradiction appears at this time. We don't have a closed object line in the image but we get a separated background in the image! If we use only 8-connectivity in the image, that means both object and background are connected, so we get a closed object line with only one background. Both conditions are contradictory.

The solution for the contradiction above is to use different connectivity for object and background. If we use 8-connectivity for object and 4-connectivity for background the image can be explained without any contradiction. It's a closed line in an image.

Another approach to solve above problem is using 6-connectivity[19][24][25]. In this connectivity, all six pixels around the central pixel are neighbors of it. This is shown in fig 1.2. If 6-connectivity is used for both object and background in an image, there is no contradiction to explain the connectivity of an image.

For conventional scanning techniques, operation is on a line-by-line basis, this at once suggests rectangular array image will be produced. In this research, our discussion will be limited to rectangular array images.

Only the edge points will be processed during the thinning processing. A point which is an edge point must match the window in fig 1.3 or the windows which are obtained by rotating the window in fig 1.3 by 90° , 180° , or 270° . That is, it must have at least a zero neighbor pixel at the left, right, top, or bottom positions to the central pixel.

To get a one-pixel thick skeleton of an interested object is the objective of thinning. Since there is no exact definition of skeleton and different thinning algorithms use different conditions to preserve or to delete the pixels under processing. The skeletons obtained by various algorithms are not identical. But these skeletons are similar. They can be used for pattern recognition without

errors if the skeletons obtained by different algorithms with the distortion in a limited range.

Thinning procedure is a template operation. Most thinning algorithms use 3×3 windows for operation. The main reason to choose such windows is that they are simple and include sufficient information about the environment of the pixel under processing. Thinning algorithms are mainly used for binary image[2]. We pay much attentions to the results which express the topological features of the interested object. The topological results of the processing will not be influenced by the size of the windows for thinning except when we are processing two-pixel thick stroke. We will discuss this exception later.

For most applications of the computer vision in real time, the processing time is a decisive factor. How to speed up the processing time and getting a suitable skeleton to express the topology of the image become a focus of research for thinning algorithms.

Because parallel algorithms have advantage over sequential algorithms in the processing time, parallel thinning algorithms have been studied extensively in recent years. Most proposed thinning algorithms in recent years are parallel algorithms.

B	O	B
O	B	O
B	O	B

Fig 1.1 A 3x3 window Where, O's express object points;
B's express background points.

Fig 1.2 6 neighbors of the central pixel in 6-connectivity

X	X	X
0	1	X
X	X	X

Fig 1.3

A 3x3 window expressing a West edge point.
Where, 1's express object points; 0's for background
points; x's for don't care which may be 1 or 0.

In this research, three parallel thinning algorithms and their implementation on hypercube parallel computer are studied. All of these algorithms are proposed in 80's. Features, advantages, and disadvantages of these algorithms will be discussed based on the topological analysis, shapes of the skeletons, processing time, and so on. An error and some disadvantages of Zhang and Suen's algorithm are pointed out, corrected, and improved. The topics include problem decomposition, load balance, and topology selection. A labeling technique is proposed which can both save the memory space and improve the processing time. Experimental images are processed on the AMETEK S14 hypercube machine. The conclusions will be based on the theoretical analysis and the experimental results.

The remainder of this thesis is organized as follows. Chapter 2 introduce three parallel algorithms. The principles of thinning algorithms and their features are also discussed in this chapter. Chapter 3 presents the implementation of parallel algorithms on the hypercube machine. Topology selection, data flow analysis, and how to make the implementation more efficient are the contents of this chapter. A detailed analysis and a comparison of these algorithms are in chapter 4. The analysis and comparison are based on the results of both simple input patterns and real images. Finally, the conclusions are summarized in chapter 5

CHAPTER 2

Parallel Thinning Algorithms and Their Features

In this chapter, we will analyze and compare parallel thinning algorithms. Three parallel thinning algorithms will be introduced and discussed. They are listed as:

1. Zhang and Suen's Thinning Algorithm. (1984) [15]
2. One Pass Thinning Algorithm. (1986) [18]
3. Guo and Hall's Thinning Algorithm. (1989) [16]

In the following chapter, we will call the above algorithms as ZS, OPTA, and GH algorithm respectively for short.

Before we start our introduction to these three thinning algorithms. It is necessary to explain several basic terminologies and concepts of image processing and thinning algorithms in particular. That will make the discussion more clear.

2.1 Basic terminologies and concepts used in thinning algorithms

The following are some basic terminologies and concepts used in image processing and thinning algorithms.

1. Neighbors of the pixel P

For a 3*3 window in fig 2.1(a), all 8 pixels around the

central pixel P are neighbors of P , denoted as $N(P)$;

2. 4-Neighbors

In fig 2.1(b), the set of pixel P_2 , P_4 , P_6 , and P_8 are 4-neighbors of P , denoted as $N_4(P)$;

3. 8-Neighbors

In fig 2.1(c), all the pixels around P are 8-neighbors of P , denoted as $N_8(P)$;

4. 4-connectivity

If two pixels are 4-neighbors and their grey-level values are from a defined set V , it is said they are 4-connectivity;

5. 8-connectivity

If two pixels are 8-neighbors and their grey-level values are from a defined set V , it is said they are 8-connectivity.

6. 4-simple and 8-simple

P is 4-simple (8-simple) if we change it from 1 to 0 does not change the 4-connectivity (8-connectivity) of 1's in its neighbors.

For example, in fig 2.2(a), P is 4-simple. In fig 2.2(b), P is 8-simple. P is not 8-simple in fig 2.2(c). In fig 2.2(c), if P is changed from 1 to 0 the pixel at the left-up corner will be separated from other 1's elements in that image.

Three more basic concepts about the pixels which belong to skeletons of objects are defined as following:

7. Break Points;

A break points can be defined as such points if it is changed from 1 to 0, the connectivity of the objects will be broken.

8. End Points;

The end points can be defined as this: the points at the open extremes of a stroke. Here we use fuzzy definition again. It is not easy to define the end points mathematically. Some researchers defined the object points as the points which have only one 1 8-neighbor are end points. However, we will find that this definition is not correct under some special conditions and will lead to some errors in getting the skeletons.

9. Redundant Points

Redundant points can be defined as the points that belong to the skeletons obtained by using some thinning algorithms, but their deleting will not influence the connectivity of the skeletons.

All these different points are shown in fig 2.3. It is easy to understand the end and break points indicated in this fig. But for points 1a, 1b, and 1c , there is not a unique way to indicate the break and the redundant points.

(a)

(b)

(c)

Fig 2.1 (a). Neighboring pixels of central pixel P;
 (b). 4-neighbor pixels of P;
 (c). 8-neighbor pixels of P.

(a)

(b)

(c)

Fig 2.2 Examples of the 4-simple and 8-simple points.

Fig 2.3 Different kinds of points in a skeleton

For example, if 1a is considered as a break point in the skeleton, then 1b and 1c are redundant points because their deletion will not influence the connectivity of the image. If 1b is considered as break point, then either 1a or 1c can be thought as redundant point respectively.

Since redundant points do not make contribution to the connectivity to the redundant points as less as possible in the skeletons. Unfortunately, not every thinning algorithm can delete all the redundant points. The number of redundant points in a skeleton can be used to judge the performance of a thinning algorithm.

2.2 The effect of size of the window used in thinning algorithms

Usually, 3×3 windows are used in thinning processing. Under parallel processing, all the points can be processed simultaneously. There is not a special order in processing as in sequential processing. A problem appears at this step. If we use 3×3 windows only, some 2-pixel thick strokes will disappear after processing. Let's consider the subimage in fig 2.4, which is a part of a vertical 2-pixel wide line, there are two 3×3 windows with two column elements overlap. If points 1a and 1b are considered separately at the same time, either deletion of 1a or deletion of 1b will not break

the connectivity of 3×3 windows with 1a or 1b at the center. If a point is deleted based on these separative match, the two-pixel wide lines will disappear after thinning. Same thing will happen when two-pixel wide horizontal lines are processed.

The error mentioned above is not an exception if 3×3 sized window is used. Actually, if the edge points are processed from two opposite directions at the same time, such errors can not be prevented in parallel processing if only 3×3 sized window is used without taking additional measures explained later. We can get enough information about a single point by using 3×3 window. However, we don't get enough information about its neighbors by this sized window. When both neighboring points are edge points, the error will occur.

There are several approaches to prevent the error mentioned above. One is to use some additional various sized windows to check the two-pixel thick stroke. Another way is to divide one iteration into several subiterations. Third method is add some additional steps to check the environment of the neighboring points of the point under processing.

Since most thinning algorithms accept the second method, this approach will be discussed in detail. Before the

following discussion, it is necessary to define several terminologies used in thinning algorithms.

1. Iteration: An iteration is a procedure in which the edge points in a picture has been checked from all the directions in terms of the algorithm one time.

2. Scan: A scan is procedure in which all the points in a picture have been scanned one time. Sometimes the terminology "pass" is used instead of "scan".

The error generated in the above example is due to : a) the use of 3*3 sized window; and b) checking of edge points in the opposite directions simultaneously. Instead of detecting the edge points from all directions in one iteration, we can detect the edge points from some but not all directions during one scan. In this way, we divide one iteration into several subiterations. In each subiteration, edge points will be checked from some but not in the opposite directions. Once all the points in a picture have been checked from the directions chosen in this subiteration, the points which have been decided belonging to deleted points will be removed. Then another subiteration begins. In this following subiteration, the edge points will be detected from some other directions in which the opposite directions are excluded. Since all the points are checked in each subiteration, so we need one scan to complete a

subiteration. An iteration is completed at the end of all subiterations in an iteration are completed. This procedure will continue until no point can be removed from that picture.

2.3 Subiterations in an iteration

Backing to the example illustrated in fig 2.4, if we check the edge points from the East direction only instead of from both East and West directions at the same time, 1b will be deleted after this subiteration is completed since its deletion does not break the connectivity. In the following subiteration, the edge points will be checked from West direction. Since 1b has been deleted in the last subiteration, the 3*3 window with 1a in the center can not be deleted because if it is deleted the original connecting image will become two separate parts.

We can divide each iteration into several subiterations and check edge points from one direction only in each subiteration. Consider thinning is a point by point operation as well as an iterative process, it is easy to realize that large amount of time is consumed in this process. To improve the operation, we can check edge points from several directions in one subiteration. The only thing we should be careful is that two opposite directions can not

be in a same subiteration. Since the number of subiterations is reduced, the processing time will also be reduced efficiently.

The disadvantage of checking the edge points in several directions in one subiteration is that the skeletons obtained from thinning may not be totally symmetrical. Besides this, a little different skeleton will be obtained if we change the sequence of the subiterations. However, that problem does not influence the recognition procedure because the skeletons are similar and the topology is unchanged.

2.4 Three Parallel Thinning Algorithms

In this section we will introduce and analyze three parallel thinning algorithms. All these algorithms are proposed in 80's.

2.4.1 Zhang and Suen's Thinning Algorithm

In 1984, T. Y. Zhang and C. Y. Suen proposed a thinning algorithm[15] which they called fast thinning algorithm. 3×3 sized windows are used in this algorithm. The pixels in the windows are labeled as in fig 2.5(a). Each iteration is divided into two subiterations.

Let $B(P)$ be the number of 1's neighbors of P and $A(P)$ be the number of 0-1 patterns in an ordered set $P_2, P_3, \dots, P_8, P_1$;

In odd subiterations, if an edge point p satisfies :

$$a) \quad 2 \leq B(P) \leq 6;$$

$$b) \quad A(P) = 1;$$

$$c) \quad P_2 * P_4 * P_6 = 0;$$

$$d) \quad P_4 * P_6 * P_8 = 0$$

then P will be deleted from the image.

In even subiterations, the conditions (a) and (b) are the same. The condition (c) and (d) are changed to (c') and (d').

$$c') \quad P_2 * P_4 * P_8 = 0;$$

$$d') \quad P_2 * P_6 * P_8 = 0$$

Solving simultaneous equations c and d, we get,

$$P_2 = 0 \quad \text{and} \quad P_8 = 0; \quad \text{or}$$

$$P_4 = 0 \quad \text{or}$$

$$P_6 = 0$$

In the same way, we get the results from c' and d',

$$P_4 = 0 \quad \text{and} \quad P_6 = 0; \quad \text{or}$$

$$P_2 = 0; \quad \text{or}$$

$$P_8 = 0$$

The two groups of the solutions we obtained above show that edge points on the East, South, and Northwest directions

are checked in the odd subiterations; the edge points on the North, West, and Southeast directions are checked during the even iterations. These are shown in fig 2.5(b)-2.5(c).

The basic principles of thinning are presented in conditions (a) and (b). In condition (a), $B(P)=1$ and $B(P)=7$ are excluded, that prevent the deletion of the end points and the excessive erosion. Condition (b) is used to prevent the deletion of the break points. When $A(P)>1$, there are more than one group separated 1's around p. Under this condition, P can not be deleted otherwise the connected region will become two parts. This is shown in fig 2.6.

ZS algorithm is simple and clear. For many pictures, this algorithm works very well. However, there are some errors and disadvantage in this algorithm. It is necessary to make correction and modification to this algorithm.

In ZS algorithm, the end point is determined by $B(P)=1$. Of course, a point is an end point if it has only one 1 8-neighbor. But $B(P)=1$ is not the necessary and sufficient condition for an end point. For two-pixel wide 45° and 135° diagonal lines, such as in fig 2.7(a), or its mirror picture, the end points have two 1 8-neighbors.

Since $B(P)=1$ is the criterion to identify the end points

Fig 2.4 Two columns overlapping in two 3x3 windows

Fig 2.5

- (a). The labelled window in ZS thinning algorithm;
 (b). The directions of edge points checked in first subiteration;
 (c). The directions of edge points checked in second subiteration.

Fig 2.6

- (a). $A(P)=2$, so the central point P can not be deleted;
 (b). $A(P)=1$, so the central point P can be deleted.

in ZS algorithm, the real end points in fig 2.7(a) are not considered as end points because they have two 1's neighbors and will be deleted for they satisfy all the deletion conditions. Once these two points are deleted, the new exposed end points are checked with the deletion conditions and will be deleted too. As this checking and deletion procedure continues the two-pixel wide diagonal line will become one or two points at last as shown in fig 2.7(b). This error belongs to the excessive erosion. Correct information can not be obtained from these one or two points.

This error was pointed out by H. E. Lu and P. S. P. Wang in 1986[17]. Their suggestion is modifying condition (a) from $2 \leq B(P) \leq 6$ to $3 \leq B(P) \leq 6$. After this modification, the error mentioned above is prevented. However, no point will be deleted from these two-pixel wide diagonal lines.

The reduction of redundant points in a skeleton as many as possible is desirable. These points do not make contribution to the recognition procedure and will consume unnecessary processing time in the following procedure after thinning. Considering half of the points remained in the skeletons of the diagonal lines, it is significant to design some additional approaches to delete these points.

The reason of keeping two-pixel wide diagonal lines unchanged is due to the use of $A(P)=2$ in determining the break points. The condition is not correct for two-pixel wide diagonal lines. For two-pixel wide diagonal lines, all points have $A(P)=2$ except the end points. Since it is common practice to use 8-connect for objects and 4-connect for background, half points of such lines can be deleted without breaking the connectivity.

In order to improve the disadvantage in ZS algorithm and obtain a one-pixel wide skeleton of the object, some additional conditions are added to ZS algorithm which are based on there are two different forms of the diagonal lines (45° and 135°) and the different edge checking directions used in the odd iterations and even iterations. They are shown below:

In odd iterations, when $A(P)=2$, following conditions are checked.

1. $P_4 * P_6 = 1$ and $P_1 = 0$ or
2. $P_4 * P_2 = 1$ and $P_3' * P_7' * P_8' = 1$;

In even iterations, when $A(P)=2$, following conditions are checked.

- 1'. $P_2 * P_8 = 1$ and $P_5 = 0$ or
- 2'. $P_6 * P_8 = 1$ and $P_3' * P_4' * P_7' = 1$

Where "*" expresses logic "AND" operation.

The point under processing will be deleted when either condition (1) or condition (2) are satisfied in odd iterations. In even iterations, condition (1') or condition (2') will be used. Here P' expresses the complement of P .

Another error in this algorithm is the definition of the ordered set used for calculating $A(P)$. According to the paper of Zhang and Suen[15], this set is a set which forms a string composed of $P_2, P_3, \dots, P_1$ around P . For a window with $P_2=0$, correct answer can be obtained according to the ordered set defined in the paper as is shown in fig 2.8(a). However, if $P_2=1$, an error may be introduced. For example, in fig 2.8(b), since $A(P)=1$ and $B(P)=3$, so point P is deleted after the conditions a and b are satisfied. But it is obvious that P is a break point and can not be deleted.

The error discussed above can be corrected if the definition of the ordered set is changed to a closed ring composed of $P_2, P_3, \dots, P_8, P_1, P_2$. After this change, the correct results can be obtained whether P_2 is 0 or 1.

After putting all the modifications and the additional conditions described above to ZS algorithm, the errors and disadvantage of the original ZS algorithm will be prevented and improved.

2.4.2 One Pass Thinning Algorithm

One Pass Thinning Algorithm (OPTA) was proposed in 1986[18]. Which means in one pass all the outmost layer edge points will be deleted if they satisfy the thinning templates and do not match the restoring templates. These windows are illustrated in fig 2.9. There are no subiterations in each iteration.

As is discussed in section 2.2, it is impossible to thin all the edge points with 3×3 sized windows in one pass without errors. OPTA algorithm uses two kinds of different sized windows to prevent such an error. There are ten predefined windows in OPTA algorithm. In fig 2.9, windows a-h are used for thinning and windows i and j for preserving. The former are called thinning windows and the latter are called restoring windows. The edge points are checked in 8 directions in this algorithm as shown in fig 2.10.

During the processing, we use these windows to match each edge point and its neighbors. Three conditions may happen. 1). If the subimage matches any thinning window and does not match any of restoring windows, the central point in this subimage will be deleted; 2). If the subimage matches any of the restoring windows, the central point will be preserved; 3). If the subimage does not match any window

Fig 2.9

Ten windows used in OPTA algorithm.
 windows a -h used for thinning, windows i and j for
 restoring.

Fig 2.10

Eight directions used for check the edge points in OPTA
 algorithm.

of ten predefined windows, the central point will be kept unchanged in this iteration. It may be changed from 1 to 0 if the environment of it satisfies condition 1 in the following iterations.

The main advantage of OPTA algorithm is that only one pass is needed to delete all the outermost layer points on the boundary of the object. That will reduce the processing time efficiently according to the experimental results in this work. Like other thinning algorithms, the skeletons yielded by OPTA algorithm have some distortions. The skeletons are biased. One reason is that the even-number wide restoring windows are for restoring two-pixel wide horizontal and vertical subimages. Another reason is that this algorithm deletes edge points from convex corners faster than from concave corners. In fig 2.9, windows e-h delete edge points from convex corners, while windows a-d delete edge points from concave corners or flat lines. Since the first group of windows consist of more don't care items, it is easy to satisfy a 3×3 subimage than the latter, so they can be carried out faster than the latter group of windows.

2.4.3 Guo and R W Hall's Thinning Algorithm

Zicheng Guo and Richard W Hall proposed two thinning algorithms in 1989[16]. Only first one will be studied in

this research.

The edge points detected windows are shown in fig 2.11.

Using the same labels as in fig 2.5(a), Guo and Hall's algorithm is described below:

Let $C(P)$ be the number of distinct 8-connected components of 1's in P 's 8-neighborhood.

$$N(P) = \text{Min}(N_1(P), N_2(P))$$

$$N_1(P) = (P_1 \vee P_2) + (P_3 \vee P_4) + (P_5 \vee P_6) + (P_7 \vee P_8)$$

$$N_2(P) = (P_2 \vee P_3) + (P_4 \vee P_5) + (P_6 \vee P_7) + (P_8 \vee P_1)$$

An edge point will be deleted if it satisfies :

- a). $C(P)=1$;
- b). $2 \leq N(P) \leq 3$;
- c). apply one of the following:
 - 1. $(P_2 \vee P_3 \vee P_5) * P_4 = 0$ in odd iterations, or
 - 2. $(P_6 \vee P_7 \vee P_8) * P_8 = 0$ in even iterations.

Where " $\vee$ " expresses logic "OR" operation.

$C(P)=1$ means P is 8-simple. In other words, there is only one group of 8-connected 1's around P . Under this condition, P 's deletion will not break the connectivity of the elements in the 3×3 window under processing. Condition a guarantees P is not a break point.

GH algorithm is similar to ZS algorithm in some sense.

They all use 3×3 sized windows and each iteration is divided into two subiterations. Both of them use some conditions to check edge points but do not give the predefined windows like OPTA algorithms. GH algorithm has some advantages over ZS algorithm. We will study that in this section. The error in ZS algorithm is avoided in this algorithm.

Instead of using $B(P)$ to check the end points in ZS algorithm, $N(P)$ is adopted in GH algorithm. This condition has its advantage. No matter whether the end points have one or two 1's 8 neighbors, these points will be preserved by GH algorithm. A one-pixel wide skeletons for diagonal lines will be obtained according to GH algorithm. Fig 2.12 can illustrate that point clearly. It is intuitively clear that the central point P in this window can be deleted without violating the connectivity of the image. When applying ZS algorithm, P is reserved since $A(P)=2$. When OPTA algorithm is applied, P is kept by both windows i and j in fig 2.9. Suppose the same picture is processed by GH algorithm, since $C(P)=1$ and $N(P)=2$, p will be deleted. That is what we want for this subimage. Same function is realized by the improved ZS algorithm. This is done by adding some additional conditions to ZS algorithm.

Up to now, three parallel thinning algorithms have been introduced and their features are discussed. More detailed

Fig 2.11 Windows used in GH algorithm for edge points check
 Windows a and b are used in odd iterations;
 Windows c and d are used in even iterations.

Fig 2.12 The central point will be preserved by ZS or OPTA algorithms. However, this point is deleted by GH or improved ZS algorithms.

study and comparison about these algorithms are given in Chapter 4.

CHAPTER 3

Implementation of Thinning Algorithms on The Hypercube Machine

The development of parallel vision algorithms accompanied the development of the parallel computers. For many applied fields of image processing and computer vision, the processing time is a very important factor. We must consider this factor when applying a vision algorithm for real time problems. There were rapid developments in both software and hardware in last two decades because of the requirements of the industry and scientific research.

The most obvious advantage of the parallel processing is that it can divide the problem being processed into several parts and process different parts at the same time. The processing time of parallel processing is much faster than that of sequential processing.

Theoretically, if we divided the problem into N parts and assign each part to each node of parallel computer, then the processing time reduces to $1/N$ of the original processing time. That is true only if we can divide the problem into n unrelated parts. Unfortunately, we never have such an ideal problem which can be divided into N unrelated parts. There are always requirements for different parts to communicate

with each other in practical applications. Sometimes each node need to interchange the data for next processing. Sometimes some nodes need to wait the results of other nodes. These problems are referred to as problem decomposition, load balance.

To make a problem to be processed parallel, we need not only the parallel algorithms but also the support of the hardware. Of many hardware structures explored until now, the hypercube structure is one of most widely used architecture configuration[26]-[29].

In this chapter, we will introduce hypercube structure. The implementation of thinning algorithms on the hypercube machines and how to improve the implementation and the factors we should consider when parallel algorithms are used on the hypercube machine are the main topics of this chapter.

3.1 The Hypercube Machines

3.1.1 Hypercube architecture

A hypercube is a n -dimensional binary cube. Fig 3.1 shows a 3-dimensional hypercube. The relation between the number of nodes and the dimensions of a hypercube is expressed as:

$$N=2^n \quad \text{or} \quad n=\log_2 N \quad [3.1]$$

where, N : the number of nodes

n : the dimensions

A n -dimensional hypercube can be constructed from two $n-1$ dimensional hypercubes. To realize this, two $n-1$ dimensional cubes are created first, then the nodes are linked together in which the numbers to express them on two separate subcubes differ only in the most significant bit. Fig 3.2 shows a 4-dimensional hypercube structure which is composed of two 3-dimensional hypercubes.

A node in a hypercube is connected to n other nodes directly. Where n is the dimension of this hypercube. These $n+1$ nodes are called neighboring nodes. For example, in fig 3.1, nodes 100, 010, and 001 are neighboring nodes of node 000.

A notable characteristic of hypercube configuration is that many topologies such as ring, mesh, tree, and hypercube topologies can be embedded on the hypercube configuration. Fig 3.3 gives an example of ring topology embedded on a 3-dimensional hypercube.

The flexibility of hypercube structure for matching different topologies gives us great convenience when we are

Fig 3.1 A 3-dimensional hypercube

Fig 3.2 A 4-dimensional hypercube structure

Fig 3.3 Ring topology embedded on a 3-dimensional cube

handling various problems.

3.1.2 Hypercube Machine

A hypercube machine is a parallel computer which uses hypercube structure to organize multiprocessors in it. Since all the experiments in this research are implemented on the AMETEK S14 parallel computer, we will use it as an example to introduce the hypercube machine.

The AMETEK S14 is a concurrent computing system consisting of 2^n processors. Where n is in the range from 4 to 8 inclusively. Each processor is arranged at a position in the computer in the same way as each node is on a hypercube. In this research, we used 16-processor AMETEK s14 system.

At each node, there is a 8 MHz Intel iAPX286 processor which functions as CPU , a 8MHz iAPX287 numeric coprocessor which offers the float-point operation, and a iAPX186 which manages the communications between relative nodes.

Only neighboring nodes can communicate each other directly. Communications between undirected connected nodes must pass the messages through some intermediate nodes.

The AMETEK S14 needs a host computer to maintain its operation. The host computer provides an interface between users and the concurrent system. In this research, the host computer is VAX-11/750.

Only one processor in concurrent system can communicate with the host computer directly. This processor is called corner processor. That's really a bottle neck between the host and the concurrent system. It is desirable to reduce the communications between the host and the concurrent system as much as possible.

There are two ways to create a concurrent program on the AMETEK S14. One is the XOS routines, and another is the User Topology Interface routines. Both have their advantages and disadvantages.

The XOS routines provide flexibility for changing the topology within the same concurrent program. However, these routines are difficult to use. Since the methods for expressing float-point number used in the host computer and the concurrent system are different, we have to convert float-point numbers when transferring them between the host and the concurrent system.

The User Topology Interface routines are easy to use.

They do the float-point numbers format conversion automatically. While these routines restrict us to set one topology for a concurrent program. If we want to use another topology, a new program must be created. In this research, user topology interface routines are used.

3.1.3 The Concurrent Programs

A concurrent program always consists of two parts: the parent program and the child program. The parent program is run on the host computer and the child program is run on the concurrent system separately. Each part has different functions.

The functions of the parent program are:[29]

1. Download the child program to all of the nodes;
2. Coordinate communications between itself and child program in the nodes;
3. Provide the file I/O and other UNIX utilities;
4. Handle all user interface I/O with the nodes.

The functions of the child program are:

1. In each node, compute the operations specified by the application;
2. Coordinate communications between itself and the parent on the host, and between nodes.

Once a concurrent program is created, It is run first on the host computer for simulation. The host computer works as a simulator in this period. It uses its time-sharing feature to simulate the operations on the parallel processors.

After the necessary corrections are made to concurrent program and the program can run smoothly on the simulator, the program is ready to run on the AMETEK S14.

It is not necessary to run a program through above two steps. A program can be run directly on the concurrent system once it is created. However, it is very difficult to find the run time errors if the simulating step is skipped.

3.1.4 Communications in concurrent system

When a concurrent program is running, each processor is working at the same time. Since the task used to be carried out by a single CPU in sequential computer is divided to N parts and are carried out by N processors at the same time now, the $1/N$ processing time is expected. However, this ideal situation does not exist in practical applications.

Firstly, to decompose the problem to n equal parts which will take equal implementation time is almost impossible. Even if the amount of the data processed by each processor

can be arranged to be same amount, the processing time may be different because the magnitudes of the data are different which will take different times to process. Secondly, many application programs need to interchange the data during the processing. Both factors will influence the implementation time of the concurrent programs.

The first problem is related to how to decompose the task for each processor. In order to make the implementation more efficient, the task arranged for each processor should be as equal as possible. That will reduce the waiting time for processors which complete the subtask faster than other processors.

Second problem is related to communications in concurrent system. There are two kind of communications in the concurrent system: Communications between the host and the nodes and internodes communications. The communication time takes a considerable part in the whole processing time.

All the messages from the host to the processors or vice versa must pass through a special processor called corner processor. The time for communications between host and processors is much longer than that for internodes communications. If possible, substituting the first communications to the second communications is desirable if the same objective can be completed.

Only neighboring nodes communicate directly. When it is necessary for unneighborly nodes to communicate, the message is passed through some intermediate nodes. Once the message reaches to the neighboring nodes, the nodes which need the message can get this message directly. In this procedure, the topology set for the program should be considered carefully since suitable topology selection reduces the steps for indirect communications and the amount of interchanging data efficiently.

The key approach to improve the efficiency of a concurrent program is try to reduce the communications to the minimum in the program. In the following discussion, we will see that the communication in concurrent programs strongly depend on the topology selection. The load balance for each node is also very important. But the load balance is not difficult in image processing since most image preprocessing is point by point processing. Dividing the whole image into equal parts is suitable in most conditions.

3.2 Vision Algorithms and Topology Selection

3.2.1 Local operation feature of vision algorithms

Most preprocessing for the images are local operations. They differ from each other in the window size chosen and the

different objectives we demand.

The local operations can be further divided into two kinds. Point operation and template operation. Threshold operation is a typical point operation. Edge detection, smoothing, and thinning are some template operations. In template operations, the results depend on the gray-level of the point and its neighboring points under processing. 3*3 sized windows are most commonly used windows. Some other sized windows are also used in template operations[30].

For all local operations, the information we need is the values of the interested point and points near it. This feature is particularly suitable for us to carry out parallel processing on them.

3.2.2 Thinning Algorithms and Topology Selection

As a kind of preprocessing vision algorithm, thinning has many common characteristics similar to other vision algorithms. While, a notable difference between thinning and other algorithms is that thinning is an iterative process. This feature makes us pay much attention to the communications in creating a concurrent thinning program since the data interchanges are carried out repeatedly as the iterative processes continue. Once the algorithm is

chosen for the program, the communications arranged in the program become the main factor to influence the efficiency of the program. If we do not handle the communications in the program suitably, the accumulative time consumed due to inadequate communication arrangement is considerable. This influence is bigger to iterative thinning process than it is to other process because there is only one time data communication between nodes in many other vision algorithms such as smoothing, edge detection, and etc.

The main approach to reduce the communications is to choose a suitable topology for the algorithm used. In a concurrent system, the terminology "topology" means the logic linkage[29]. This logic linkage indicates which neighbors of a given node can be communicated directly with that node, and how these neighbors are logically placed in relation to that node. Different topologies lead to different performance.

Matching a parallel algorithm with the interconnection network structure is often a difficult task. Before choosing a topology, we must know the data flow requirements between processors for performing specified task on the input data.

Referring to thinning algorithm, the range of changed

size of windows is very narrow comparing to some other vision algorithms. It is easy to certify the amount of interchanging data of thinning algorithm in a concurrent program.

Intermediate steps needed for data transferring is another factor we must consider. The more steps needed, the more time will take.

Now we have seen two main principles for choosing the topology in a program. They are: 1. reduce the communications as many as possible; 2. reduce the intermediate communication steps as many as possible.

Ring topology and mesh topology (also called nearest neighbor topology) are two most common used topologies in creating a concurrent program for image processing. Both topologies can fit the local operations of vision algorithms very well. In the following sections, we will analyze these two topologies based on the communication requirements for thinning.

3.2.3 Ring Topology

In ring topology, all the nodes are connected in a one-dimensional nearest neighbor configuration. Each node,

that is, each processor can communicate with its left and right neighbors directly. Fig 3.4 shows a n-node ring configuration with nodes labeled from 0 to N-1.

The number of the neighbors of a processor in ring topology is small. There are only two neighbors can be connected directly for each processor. The boundary nodes 0 and N-1 satisfy periodic boundary condition. It is noted that if the problem under processing doesn't possess this periodic feature, the breaking of the connection of nodes 0 and N-1 is necessary in the program, otherwise, it will lead to wrong results. The communications between two distant nodes need at least $N/2$ steps.

Ring topology works very well for most low level vision algorithms. Its structure can match the local operations of these algorithms.

When we process a picture on a concurrent system, the whole picture is divided into N subimages which will be loaded to N processors separately. Because thinning is a template operation, when processing the points on the boundary of each subimage, we need not only the data in this subimage but also the data in the neighboring subimage. Figs 3.5 and 3.6 illustrate a 512*512 image partitioning and the data needed for processing in one subimage. From fig 3.6, it

Fig 3.4 Ring structure

a. original 512x512 image

Fig 3.5

A 512x512 image partition using ring topology
Each subimage is loaded to a processor in the concurrent system.

Fig 3.6 3x3 window operations at different positions in a subimage in ring topology

is seen that all the data needed for processing in one subimage can be obtained from the immediate neighbors in ring topology. This is an advantage of the ring topology.

For windows 1 and 3 in fig 3.6, the operations need the data from left and right neighbors. For window 2, all the data needed are in the subimage. It is noted that the data needed from the neighboring nodes depends on the size of the windows. When the window becomes bigger, more data is required from the neighboring nodes.

3.2.4 Nearest Neighbor Topology(Mesh Topology)

Nearest neighbor topology is also called mesh topology. In this topology, processors are arranged in a n-dimensional nearest neighbor fashion and only the processors which are adjacent can communicate directly. Figs 3.7 show a two-dimensional mesh topology structures. In two-dimensional mesh topology, a node can communicate with its E, S, W, and N direction neighbors directly.

The mesh topology structure has periodic feature. The boundary nodes along each axis are periodically connected. For example, node 7 in fig 3.7 can communicate directly not only nodes 3, 6, and 11, but also node 4.

Fig 3.7 2-dimensional mesh topology

Fig 3.8 N steps are needed to pass the message from the distant node to the host

Fig 3.9

A 512x512 image partitioning mesh topology
 Each subimage is loaded to a processor in concurrent system

Fig 3.10 3x3 sized window operations at different positions
 in a 128x128 subimage

For a N node square array fashion 2-dimension mesh topology, the most distant nodes will take $N^{1/2}$ steps to complete the communication to the host. This can be explained using fig 3.8. Since the boundary nodes are periodically connected, the biggest distance between the host and nodes in this structure is the distance between the central node and the corner node. It takes $N^{1/2}/2$ steps to pass the message from the central node to the boundary nodes. Same steps are needed to pass this message from there to the corner node. So the total steps are $N^{1/2}$.

Fig 3.9 shows a 512×512 image is partitioned into N equal squares using mesh topology.

Fig 3.10 illustrates 3×3 sized window operations at different positions in a subimage. Window 3 need the data from immediate neighbors. Window 2 need data from both immediate neighbor and indirectly connected neighbor. That means two steps are required for transferring the data when handling corner points of a subimage.

3.2.5 Comparison of two topologies used for thinning algorithms

In order to get high efficiency of implementation of an algorithm on a concurrent system, the topology selection

should be considered deliberately. Since this problem is related to the amount of the data interchanged and the steps needed for that interchange, so we will begin our discussion from these points.

The amount of the data interchanged in a program is decided by the window size used in the algorithms. As 3*3 window is most commonly used window, we will use this window as example to discuss the data interchange in ring and mesh topologies.

When using 3*3 windows in ring topology as is shown in figs 3.5 and 3.6, each processors between 0 to 15 need to get one row of 512 data from the left and right neighbors processor respectively. For processor 0, one row of 512 data is needed from its right processor. For processor 15, one row of 512 data is needed from its left processor. All these communications occur between immediate neighbors and can be completed in one step.

In a concurrent system, these data communications are carried out by processors at the same time. The data interchanged in a typical processor can be used to estimate and to analyze the data communication process. The amount of the data interchanged is:

$$512*2 = 1024$$

Applying same sized windows to mesh topology as is shown in figs 3.9 and 3.10, a typical processor, which is labeled as C processor in fig 3.10, need to get one row of 128 data from its N neighbor and S neighbor, one column of 128 data from its W neighbor and E neighbor respectively. Besides these, the processor need to get 1 datum from its NW, NE, SW, and SE neighbors respectively too. The total amount of data in this procedure is:

$$128*4 + 1*4 = 516$$

This amount is approximate half of the data needed to exchange in ring topology for a typical processor. However, not all the interchanged data can pass to C processor in one step in mesh topology. The data from NW, NE, SW, and SE need two steps to transfer to it.

It seems a little bit difficult to compare two topologies now. Ring topology need to transfer more data but all the interchanges are between immediate neighbors, while in mesh topology fewer data is needed to transfer but some data interchanges are between indirectly connected neighbors. Let's assume t is the time for a processor to send one data to the direct connected neighboring processor in both topologies. In the following discussion, we will find that mesh topology is more efficient in data communication if the procedure of this change is arranged suitably.

In ring topology, the total time for data interchange is $1024t$ since all the interchanges are between direct connected processors and the amount of data interchanged for a typical processor is 1024.

In mesh topology, like the C processor in fig 3.10, the four immediate neighbors of it also need to get the data from their 8 neighbors around them. Based on this, the data interchange can be done in the following manner.

Firstly, processors NW, W, and NE send one row of 128 data to their South neighbors and processors SW, S, and SE send one row data to their North neighbors as is shown in fig 3.11. Since all these interchanges can be carried out in each processor at the same time so only time $t_1=t_3=128t$ is needed for these interchanges. Notice that, 4 data in the processors which are at the diagonal positions of processor C are in C's immediate neighbors now after these procedures are completed.

Secondly, processors NE, W, And SW will send a $(128+2)*1$ array data to their West neighbors and processors NW, W, and SW will send a $(128+2)*1$ array data to their East neighbors. This procedure will take $t_2=t_4=130t$ to complete as shown in fig 3.12. When this step is completed, all the data necessary for calculation are in each processor. This is

shown in fig 3.12.

The above procedure of data interchange in mesh topology can be carried out by each processor step by step at the same time. The total time is $t_1+t_2+t_3+t_4=516t$. The actual time will be a little more than $516t$. It is because the data transferring is in a row fashion between two processors in AMETEK S14. When transferring a column data to the neighboring node, we must change this column data to a row data before sending them to other processors.

Actually, mesh topology can apply parallelism more efficiently than ring topology when parallel thinning algorithms are implemented on the hypercube structure. In a N nodes concurrent system, when interchanging a row of 512 data between the immediate neighbor, the procedure is carried out by one processor only in ring topology. While the same function is carried out by $N^{1/2}$ processors in mesh topology.

Although the problem of data transferring from the indirect connected processors to the central processor must be considered in mesh topology, since these data are very few and the procedure for data transferring can be arranged adequately as discussed above, that problem does not influence the efficiency of mesh topology so much.

Fig 3.11 The procedure of data transferring to processor C in step 1

Fig 3.12 The procedure of data transferring to processor C in step 2

It is easy to certify that the data transferred for a typical node in a N nodes system are: for ring topology, $2*512=1024$; for mesh topology, $4*512/N^{1/2} + 4$. It is noted that the amount of the data depends on the number N . With the increase of the number of nodes, fewer and fewer data are needed to transfer in mesh topology.

Another factor to influence the communication time is the steps to pass the message between most distant node in a selected topology. $N/2$ steps are needed to pass the message from the distant node to the host in ring topology. In mesh topology, $N^{1/2}$ steps are needed. It is easy to prove that $N/2$ is bigger than $N^{1/2}$ when N is bigger than 4.

The mesh topology is more efficient than ring topology for thinning algorithms. This conclusion is obtained based on the analyses of the amount of data interchange, the data interchange procedures in two topologies. That is because the mesh topology uses the parallelism more fully. Besides, since the steps needed to pass the message from the nodes to the host are less in the mesh topology, if communications are frequent in the processing, mesh topology is a better choice.

The above conclusion can not be extended to all other conditions. It was reminded before that one feature of

thinning algorithms is that the changing range for window size is very narrow. Only very little data is needed from the indirectly connected processors and this data can be arranged in a suitable way to pass to the destination without taking special time for this transfer. Some vision algorithms such as correlation use very big operation windows. When applying such algorithms on the concurrent system using mesh topology , a large number of data are needed from indirectly connected processors. Under this condition, both theoretical analyses and experiments may be needed to assess the merits of a topology.

3.3 Changeable Value Label Thinning

Saving memory space and implementation time are important in creating a program. In this research, a special label technique is proposed which can improve the processing time and save the memory space.

The parallel algorithms offer us the possibility to process all the points in an image simultaneously. However, the number of processors of a concurrent system is limited. Based on the parallelism of the parallel algorithms, we can divide the problem into N parts and handle them on N processors separately. It is not difficult to understand that the pixels in each subimage are processed sequentially since there is only one CPU in each processor. In this research, all the experiments are done on the AMETEK S14 concurrent system. There are 16 nodes in this system, so each image is divided into 16 subimages for processing in each node.

The problem of storing the intermediate processing results is due to the parallelism of parallel algorithms. There is not a special order in processing. The results obtained for each point depends on the values of the central point and its relative neighbors. If we change the value of the central point based on the result of processing, which is

just completed, wrong results may lead to when the neighboring points of the central point are processed. It is noted that in each iteration, we need the original values of all points in the last iteration. The processed results, that is, the new values of the points being processed, can be changed only after each scan is completed.

This can be explained more clearly by using fig 3.13. Assuming that OPTA algorithm is used, the correct result is in fig 3.14. On the other hand, an error of deleting point 1b will occur if we change the value of point 1a to zero when it matches a thinning window in OPTA algorithm before processing point 1b.

In order to prevent such errors, the thing we have to do in parallel thinning processing is to store the intermediate results during the processing. The original value must be kept until one scan is over. After each iteration, we transfer the stored intermediate results in the memory to the memory space used for keeping the original image data in the last iteration, then a new iteration will begin.

The procedure described above must be carried out repeatedly until the final results are obtained. The accumulative time for storing and transferring the intermediate results is considerable. Besides this, the

memory spaces for storing the intermediate results are equal to the memory spaces for original image. As the resolutions of an image become higher and higher, more and more memory spaces are needed.

Thinning processing is usually used for binary images[2]. What has been done in thinning is to change some 1's in original image to 0's until no more 1's can be changed. After each iteration, some "new 0"s appear in the image adding to the set of "old 0"s which present the background of the image in the last iteration. Actually, the reason we can't make the 1-0 change during the iteration is that the computer can not identify these "new 0"s from the "old 0"s in an image in the iteration which is being carried out now if both of them are presented as 0 in the image.

This problem gives us a hint. If we find a method which can identify "new 0" and "old 0" during the iteration, the changes for 1-0 during the iteration will be possible. It is obvious this problem can not be solved by using ordinary binary image whose possible value for each pixel are only "1" or "0". A special label approach is proposed in this research. A 0-M binary image will be used, where M is an integer which is larger than the number of the iterations used for thinning an image.

Instead "1" is used to present the object in the original image, M is used to present the object in the original image. During the i th processing, if a point satisfies the deletion conditions, it will be changed from M to i immediately. In order to avoid the errors discussed above, in i th iteration, the points whose values are greater or equal to i are treated as object points; for the points whose values are less than i , computer will treat them as background points.

This approach gives us the possibility to change the value of a point immediately once it is decided that it belongs to a flagged point. That is because although we change the value of a point during the iteration, the computer can treat these points of changed values as object in the same iteration. But these points are treated as background points in the next iteration. We are still processing a binary image, but a variable i is used to identify the object and background points in each iteration.

The key to this label approach is a changeable value " i " being used. This value become a threshold value in each iteration. In i th iteration, any point which should be deleted is changed to i that is considered as background point in next iteration, that is, $(i+1)$ th iteration. However, in i th iteration, this point will still be

considered as object point since its value satisfies the condition " equal to or bigger than i ".

Two benefits are obtained in this approach. First, the value of the point under processing can be changed once it is decided to be a deleted point. After the scan is over, the resulted image can be used for next thinning processing immediately. No data transfer procedure is necessary since all the changed values are in the image already. The time used for that procedure is saved. Secondly, the memory space for storing intermediate results is not required either.

The time saved by this method depends on the size of the image and the iteration number. When handling large sized picture and thick lines are in the picture, more time will be saved. The memory spaces saved depends on the size of the picture.

Fig 3.15 illustrates the thinning results of a 0-M label subimage. The resulted skeleton composed of points whose values are M is the same as the skeleton composed of points whose values are 1 in fig 3.13.

Increasing the implementation speed and saving the memory spaces are two main objectives when creating a computer program. Both objectives are realized here by using

this changeable value label method in thinning algorithms.

```

0 0 0 0 0 0
0 0 0 0 1 0
0 0 1a 1b 1 0
0 1 1 1 0 0
0 1 1 1 0 0
0 0 0 0 0 0

```

Fig 3.13 An original image

```

0 0 0 0 0 0
0 0 0 0 1 0
0 0 0 1b 0 0
0 1 1 0 0 0
0 0 0 0 0 0
0 0 0 0 0 0

```

Fig 3.14 The thinned result
of Fig 3.13

```

0 0 0 0 0 0
0 0 0 0 m 0
0 0 m m m 0
0 m m m 0 0
0 m m m 0 0
0 0 0 0 0 0

```

(a)

```

0 0 0 0 0 0
0 0 0 0 m 0
0 0 1 m 1 0
0 m m m 0 0
0 1 1 1 0 0
0 0 0 0 0 0

```

(b)

```

0 0 0 0 0 0
0 0 0 0 m 0
0 0 1 m 1 0
0 m m 2 0 0
0 1 1 1 0 0
0 0 0 0 0 0

```

(c)

Fig 3.15 The thinned procedure using 0-m label expression-1
for 0-1 label binary image in Fig 3.13

- (a). Fig 3.13 expressed in 0-m form
- (b). The resulted image after 1st thinning
- (c). The resulted image after 2nd thinning

Chapter 4

Comparison and Analysis of Results of Experiments

In chapter 2, we have discussed three thinning algorithms. Some important characteristics of thinning algorithms in general are also studied and discussed. More comparison and analysis are given in this chapter. The comparison and analysis are based on the results of both the simple input patterns and the real images.

4.1 Analysis and Comparison for the results of simple input patterns

Thinning is a preprocessing for high level image processing. The reduction of data to present the structure of a object is significant since that improves not only the time for the iterative thinning processing but also the time needed for the next high level image processing. It is desired that skeletons contain redundant points as fewer as possible. The number of redundant points in a skeleton can be used to judge the performance of a thinning algorithm.

During thinning process, the image processed is scanned repeatedly until the skeleton is formed. One difference between different algorithms is they will use different scan times to process an image. So scan times can be used to

estimate the efficiency of algorithms.

Two numbers r and s [16] are introduced here to judge the results of four algorithms.

The number r is defined as:

$$r = \frac{\text{number of redundant points in the skeleton}}{\text{number of total points in the skeleton}}$$

From the definition of r , we can see that smaller value of r yields thinner skeleton. That means the data to express the structure of the object has been reduced efficiently by thinning.

The number s is the times of scans to get the final skeleton. It is noted here the difference between the iteration number and the scan number. For thinning algorithms which do not have subiterations such as OPTA algorithm, the points in a picture are scanned only one time in each iteration, so the scan number is the same as the iteration number. For thinning algorithms possessing subiterations, in each iteration, the points are scanned several times which are equal to the number of subiterations in an iteration. For such algorithms, the scan number is equal to the number of iterations multiplies the number of subiterations in an iteration.

Some patterns are used as inputs to be processed by four

algorithms. The results are listed in tab 4.1.

In tab 4.1, ZS algorithm gets good results for 0° and 90° lines. But for patterns 1 and 2, this algorithm generates some errors. It makes serious erosions. The lines are deleted to 2 points.

OPTA algorithm gets good results for all patterns except pattern 1. The scan number is low. Since this algorithm uses even number wide restoring windows to avoid two-pixel strokes to be deleted, so the skeletons obtained from 45° and 135° lines are not similar. For 45° line, an ideal diagonal line is obtained. On the other hand, the skeleton for 135° line is expressed by stepped line. 40% redundant points are remained in the skeleton.

GH algorithm and improved ZS algorithm offers the better skeletons than that offered by the other two algorithms. The scan numbers are lower too. For GH algorithm, it is because more reasonable criterion used in this algorithm to decide end points and break points. The use of $C(P)$ and $N(P)$ are better than the correspondent criteria used in other algorithms. For improved ZS algorithm, since some additional conditions are added to the original ZS algorithm, the errors and drawbacks are prevented.

Table 4.1 Comparison of skeletons obtained by different thinning algorithms (x expresses no significance)

Algorithm used for thinning

* : the original image points
 • : the deleted points

Table 4.1 Comparison of skeletons obtained by different thinning algorithms (x expresses no significance)

Algorithm used for thinning

(continue)

Original pattern	GH	ZS	improved ZS	OPTA
3	* * * * * * * * * * * *	* * * * * * * * * * * *	* * * * * * * * * * * *	* * * * * * * * * * * *
	s=1 r=0	s=1 r=0	s=1 r=0	s=1 r=0
4	* *	* *	* *	* *
	s=1 r=0	s=1 r=0	s=1 r=0	s=1 r=0

* : the original image points

• : the deleted points

Generally speaking, scan number s can be used to estimate the efficiency of a thinning algorithm because the thinning process is a window matching process point by point through out the whole image. The bigger the scan number is, larger time is consumed during the processing. But since different algorithms use different conditions to decide the processed results and some conditions are simple to carry out and some are not, the best way to compare the processing efficiency may be comparing the CPU time used for different algorithms. We will discuss that later.

4.2 Comparison and Analysis of the results of real images

4.2.1 Experimental Environment and Procedure

Several 512*512 images are processed with the AMETEK s14 16-node hypercube machine in this research. The multiprocessors in the concurrent system are controlled by the host VAX 11/750 minicomputer under UNIX 4.3 BSD operating system.

The input images include an image of different font of characters, an image of a printed circuit and two images of different shaped objects. The grey-level of the original image is in the range from 0 to 256. The process of image

acquisition, digitization, and display is accomplished with the VICOM image processor, which is interfaced with an RCA vidicon camera and controlled by VAX computer. The block diagram of experimental environment is illustrated in fig 4.1.

Mesh topology is used in the concurrent program. The programs are developed under user interface routines.

In order to minimize the influence of the noise in image acquisition, the input images are smoothed first. The median filtering is used for smoothing. Thresholding value is chosen by the histogram of the image. After thresholding is completed, the resulted binary image is used as input image for thinning. For all the algorithms considered here, the flow chart of the concurrent programs(parent and child) are the same in this research. Figs 4.2 and 4.3 illustrate the flow charts of the parent and child programs.

The pictures of the input images are in fig 4.4(a), 4.5(a), 4.6 (a), 4.7(a), Which are thinned using GH, OPTA, and improved ZS algorithms.

Figs 4.4(b)- 4.4(d), 4.5(b)-4.5(d), 4.6(b)-4.6(d), and 4.7(b)-4.7(d) are the skeletons obtained from the corresponding images by using GH, OPTA, and improved ZS

Fig 4.1 The environment of the experiments

Fig 4.2 The Flowchart of Parent Program

Fig 4.3 The Flowchart of Child Program

algorithms respectively.

4.2.2. Topological Analysis

Comparing the original images with their skeletons obtained by three algorithms, we can see that the connectivity that represents the topology features of the images is kept unchanged after thinning.

Thinning is used for shape analysis in computer vision and pattern recognition. The main requirements on the thinning process are keeping topology unchanged and generating the skeleton that roughly lie on the MAT of the object. In other words, the skeletons should represent the shapes of objects. The following analyses are based on that point.

Fig 4.4 (a) is an image consisting of different fonts of characters. Figs 4.4(b)- 4.4(d) give the skeletons of this image by GH, OPTA, and improved ZS algorithms. All these skeletons keep the basic shapes of the corresponding characters. The skeletons of many characters are similar between improved ZS and GH algorithms. It appears that the skeletons generated by improved ZS algorithm are better than that of other two algorithms. The skeletons of OPTA algorithm have larger distortion than that of other

algorithms. It is very sensitive to noise in original images. Some twigs attached to the skeletons are very obvious fig 4.5(c). Comparatively speaking, GH and improved ZS algorithms generate more smoothed and less distortion thinning results of image 1.

Twigs are harmful for pattern recognition. Sometimes they will lead to misidentification. One reason for the twigs in fig 4.4(c) is that the thinning windows for vertical and horizontal lines of OPTA algorithm have strict requirements for the edges. These windows are in fig 2.9(a)-(d). If a single noise point attaches to the edge of such lines, that point can not be deleted by the thinning windows. A twig from that point to the skeleton will be formed. That is a disadvantage of OPTA algorithm.

It is noted that skeletons of characters B and D of some font may be thinned to the shape like Arabic number 8 and character 0 respectively. We should design pattern recognition system carefully to identify such characters since their skeletons are ambiguous.

Character recognition is a main application of pattern recognition. From thinning results of image 1(fig 4.4(a)), it is seen all three algorithms can satisfy the requirements for that application. The improved ZS and GH algorithms

generate better results even there is some noise in the original images. For OPTA algorithm, it is desired to acquire images in good conditions. If the requirement for pattern recognition is higher, some additional steps may be necessary for deleting the twigs.

Fig 4.5(a) is a printed circuit. The thinned images are shown in fig 4.5(b)-4.5(d). The skeletons keep the shape of the original circuit. The topology of the image is unchanged after thinning. The thinned results of GH and improved ZS algorithms are better than that of OPTA algorithm for this image. All lines and holes are very clear in the skeletons of first two algorithms.

There are two disadvantages in the result of OPTA algorithm for image 2(fig4.5(a)). One is that there are some twigs attached to the lines or holes. This disadvantage has been discussed previously. Another drawback is that some holes are not obvious after thinning. That is also because the edge point deletion conditions in OPTA algorithm are more strictly than that of other two algorithms.

For example, for an object with a 2×2 hole inside an object, the thinning procedure is carried out from both the hole edge and the outer layer of the object under GH and improved ZS algorithms since both edges satisfy the edge

detection conditions. After thinning, such holes will become very obvious. If OPTA algorithm is used, thinning procedure can only be carried out from the outer edge of the object since the subimage around that hole does not satisfy the deletion conditions. The resulted skeleton for the object will be around the original hole. As in fig 4.5(c), such holes are difficult to see. That feature of OPTA algorithm is not good for thinning. Firstly, that makes the results not obvious; secondly, it influence the processing speed because we can not thin the image from both inside and outside edges.

In figs 4.6(a)(image 3) and 4.7(a)(image 4), there are some special shaped objects including rectangles, squares, triangles, 45 and 135 diagonal strokes. Among triangles, there are right angle, right angle isosceles, equilateral, and arbitrary triangles.

Figs 4.6(b) - 4.6(d) show the skeletons of image 3(fig 4.6(a)) by using three algorithms respectively.

Fig 4.6(c) gives the result of OPTA algorithm for image 3. In this image, except the square and the triangle with the right angle at the low-left corner, all other objects obtained skeletons which represent their shapes.

Figs 4.6(b) and 4.6(d) are almost same. The skeletons

for rectangles and diagonal strokes are correct, but the square and diamond become one or several points after thinning. The skeleton for triangles with a right angle miss the part of it which should start from the vertex of the right angle.

The causes of above errors are due to the deletion conditions used in these different algorithms. It is easy to deduct that one layer of the square in image 3 will be peeled totally with all three algorithms. Since this object is too symmetrical, when thinning procedure is over, only one or several points are left.

For diamond object, same thing happens as in processing square object by using GH and improved ZS algorithms. On the other hand, the corner points of that diamond will not be deleted by OPTA algorithm. A cross is left after thinning by that algorithm.

The local operation feature of thinning means that the results of thinning are influenced only by data values of the points in a limited area. After we discussed the thinning results of squares and diamonds, it is easy to explain the skeletons of two triangles with a right angle in them in image 3. Actually, the top part of the triangle on low left of the image 3 is exact same as the top part of the diamond

in the same image; the low-left part of another triangle on the middle of the image 3 is same as the low-left part of the square in that image. By using OPTA algorithm, the correct skeleton is obtained for the first triangle and part of skeleton is lost for the second triangle. Under GH and improved ZS algorithms, the skeletons miss the part which should start from the right angles vertex.

The positions of some objects in image 3 are special. The medial lines of the rectangles, square, diamond, and one triangle are perpendicular to the x-axis. In fig 4.7(a)(image 4), such objects are rotated with the medial lines are not at the right angle to x-axis. An equilateral and an arbitrary triangles are added in this image with the positions at right-up and right-down respectively. Figs 4.7(b) - 4.7(d) are the thinned results of that image.

Unlike the results of image 3, all the skeletons of image 4 are correct to represent the original objects by three algorithms. Some interesting results can be seen from these images. The skeletons are not unique for same object in different orientations. We get straight lines for rectangles if their medial axes are vertical to x-axis. While such rectangles are rotated by an angle, "I" type of skeletons are obtained. For square and triangles with right angle in image 4, correct skeletons are obtained. While

processing same objects at the orientations in image 3, either erroneous results or incomplete results are obtained.

The causes of above phenomena can be explained as follows. Theoretically, the skeleton of an object should be orientation independent. However, it is very difficult, may be impossible to realize that. No matter what kind of arrays are used to represent an image, the image we get is digitized. The pixels in an image are quantized. Once the orientation of an object is changed, the contour of the object may change its shape. For example, assuming the contour of an object is composed of straight lines, then the contour can be expressed as straight lines only under the condition of such lines are vertical, horizontal, or slant at 45 or 135 to the x-axis in a rectangle array. Therefore, when we rotate the object with an arbitrary angle, the contour of the object becomes zig zag lines. The points on the contour may or may not be deleted(thinned) in this new location as compared to the old location. Hence, the resulted skeleton may be different.

Fig 4.11 gives a rectangle shaped object in different orientations. The variation of the contour is obvious. After rotation, the environment of the edge points is changed a lot.

In this research, the rectangular, triangular, and square objects rotated at different angles are experimented. The input pattern and the skeletons obtained are in figs 4.8, 4.9, and 4.10 respectively. In these pictures, the skeletons are arranged from left to right by GH, OPTA, and improved ZS algorithm.

From the images in figs 4.8, 4.9, and 4.10, it is found that GH algorithm is less influenced by the rotations. This algorithm can produce correct skeletons except that the object is at some special orientations. The distortion in the skeletons is acceptable. OPTA and improved ZS algorithms can also produce correct skeletons at some orientations, but the results are not as good as that of GH algorithm. These two algorithms are easy to be influenced by the unsmoothed edges which is common for most images.

In general, the experiments in this research show that: all three algorithms, GH, OPTA, and improved ZS algorithms can keep the topological feature of images unchanged after thinning. For character and printed circuit images, GH and improved ZS algorithms are better than OPTA algorithm in terms of skeletons. The edge deletion conditions of OPTA algorithm are more strict than that of other two algorithms. That brings some disadvantages which have been discussed in this section. It is desirable to have preprocessed images

for OPTA algorithm. If high accuracy is needed for recognition, it looks that GH and improved ZS algorithms are better.

Fig 4.4(a). Image 1: Characters of different fonts

Fig 4.4(b). Skeleton of GH algorithm for Image 1

Fig 4.4(c). Skeleton of OPTA algorithm for Image 1

Fig 4.4(d). Skeleton of Improved ZS algorithm for Image 1

Fig 4.5(a). Image 2: A printed circuit

Fig 4.5(b). Skeletons of GH algorithm for Image 2

Fig 4.5(c). Skeletons of OPTA algorithm for Image 2

Fig 4.5(d). Skeletons of Improved ZS algorithm for Image 2

Fig 4.6(a). Image 3: Different shaped objects

Fig 4.6(b). Skeletons of GH algorithm for Image 3

Fig 4.6(c). Skeletons of OPTA algorithm for Image 3

Fig 4.6(d). Skeletons of Improved ZS algorithm for Image 3

Fig 4.7(a). Image 4: Different shaped objects

Fig 4.7(b). Skeletons of GH algorithm for Image 4

Fig 4.7(c). Skeletons of OPTA algorithm for Image 4

Fig 4.7(d). Skeletons of Improved ZS algorithm for Image 4

Fig 4.8. Skeletons of GH, OPTA, and Improved ZS algorithms for rectangular objects at different orientations

Fig 4.9. Skeletons of GH, OPTA, And Improved ZS algorithms for triangular objects at different orientations

Fig 4.10. Skeletons of GH, OPTA, and Improved ZS algorithms for square objects at different orientation

Fig 4.11. A rectangular shaped object in different orientations in a rectangular array

4.2.3 Implementation Speed Analyses

Table 4.2 lists the experimental results of image processing time for different input patterns and different algorithms. The time measurements include only thinning operation time in multiprocessors. The I/O download and upload time are not included in these measurements.

For the images of characters and printed circuit, OPTA algorithm is the fastest one. GH and improved ZS algorithms have almost the same processing time. On the average, OPTA algorithm is about 25% - 40% faster than other two algorithms.

For image 3, OPTA algorithm is still faster than the other two. Unlike the situation in first two images, there is a big difference in processing time between GH and improved ZS algorithms. Improved ZS algorithm is 33% faster in ordinary binary image thinning and 30% faster in changeable value binary image thinning than GH algorithm. While OPTA algorithm is 19% faster in ordinary binary image thinning and 14% faster in changeable value binary image thinning than improved ZS algorithm.

The reason for fast speed of OPTA algorithm is due to its lower scan number. For first three images, OPTA

Table 4.2 Record of processing time in experiments

input image	algorithm used for thinning	scan number	Image processing time (s)	$\frac{\text{processing time (C)}}{\text{processing time (O)}}$
image 1 (characters)	GH	45	28.355013 20.565266	72.5% [*]
	OPTA	31	20.796642 15.574083	74.9%
	Improved ZS	42	27.908751 19.563517	70.1%
image 2 (printed circuit)	GH	43	26.255461 18.386524	70.0%
	OPTA	22	15.273352 11.485356	75.2%
	Improved ZS	42	27.153877 19.351490	71.3%

* (O) expresses ordinary binary image is used for processing
(C) expresses changeable value binary image is used for processing

Table 4.2 Record of processing speed in experiments (continue)

input image	algorithm used for thinning	scan number	Image processing time (s)	$\frac{\text{processing time (C)}}{\text{processing time (O)}}$
image 3 (special shaped) icon	GH	89	50.912060	(O) *
			33.560844	(C)
	OPTA	45	28.088825	(O)
			20.011211	(C)
	Improved ZS	59	34.342991	(O)
			23.410864	(C)
image 4 (special shaped) icon	GH	74	41.377037	(O)
			27.453579	(C)
	OPTA	68	41.419308	(O)
			28.257797	(C)
	Improved ZS	48	27.133471	(O)
			20.068447	(C)

* (O) expresses ordinary binary image is used for processing

(C) expresses changeable value binary image is used for thinning

algorithm completes thinning in much lower scan numbers than that of other algorithms. It is because of its one pass thinning feature.

The most interested thing in experimental result of image 4 is that improved ZS algorithm achieves the fastest processing speed in this image and its scan number is lower than that of other algorithms. For many images, OPTA algorithm will use the lower scan times to thin them than GH and improved ZS algorithms since it possesses one pass feature. While the same function will take two times to complete by GH and improved ZS algorithms. However, the experiments for image 4 show that there are some exceptions.

The scan number is image dependent. For some images, OPTA algorithm may not have the lowest scan number.

One reason for that may be the strict requirements of OPTA algorithm in deleting edge points. An edge point must have at least three 0 neighbors and two or three 1 neighbors at special positions around it if it belongs to a deleted point(see fig 2.11). While the checking and deletion conditions of other algorithms are easier to satisfy. Among algorithms in this experiment, improved ZS algorithm has fewer requirements for deleted points. That is the reason that the improved ZS algorithm processes image 4 faster than other algorithms. In fig 4.11, all edge points of the

rectangle at the left position can be deleted according to OPTA or improved ZS algorithms. However, once it is rotated with an arbitrary angle to a new position such as it is at the right position in the fig, only six points satisfy the deletion conditions of OPTA algorithm; while all edge points still satisfy the deletion conditions of improved ZS algorithm.

The example above indicates that OPTA algorithm may lose its fast processing speed when objects being thinned are in some orientations in which a lot of edge points do not match the thinning windows of that algorithm.

Besides scan number, the processing time is also determined by the shape of objects. That is because only edge points are processed in thinning procedure. If two skeletons are obtained from two images with the same scan number by using the same algorithm, more time is needed for the image in which the total periphery of objects is longer than that of another image.

No matter what algorithm is used, the changeable value thinning approach described in chapter 3 improves processing time efficiently. All experiments show that fact. The processing time is about 15% -45% faster than that of using the same algorithm to process ordinary binary images in this

research. The saving time depends on the images and the algorithm used. Since this saving time is due to the saving of the procedure for storing and transferring the intermediate data which must be carried out when ordinary binary images thinning approach is used, so as the scan number increases, more time is saved.

From the experimental results, the following conclusion can be made. For most images, OPTA algorithm can process faster than other two algorithms in this research. The main factor which influences the processing time is the scan number. If the approach of dividing an iteration to subiterations is accepted in a thinning algorithm, the best choice is two subiterations.

CHAPTER 5

Conclusions

Parallel thinning algorithms and applying them on hypercube concurrent system are discussed and analyzed in this research.

As an important preprocessing in vision problems, thinning algorithms are researched intensively in recent years. The basic requirements for thinning is obtaining a skeleton of an object which represents the shape feature of the object with the distortion in an acceptable range. Besides these, the algorithms are desired to work as fast as possible.

GH, OPTA, and improved ZS algorithms discussed in this research can meet these requirements basically. For many images, OPTA one pass thinning algorithm is faster than GH and improved ZS algorithms. The disadvantage of this algorithm is its sensitivity to the noise on the boundary of the images and strict limitation for deleted points. GH algorithm produces better skeletons than that of other algorithms especially if there is some noise in the images. Its noise resistant ability is higher, but the processing speed is found to be lowest, at least in this study. Improved ZS algorithm improves and corrects the disadvantage

and errors in the original ZS algorithm and possesses some advantages like GH algorithm. The noise resistant ability of improved ZS algorithm is higher than that of OPTA algorithm but lower than that of GH algorithm. For some special images, the processing speed of improved ZS algorithm is even faster than that of OPTA algorithm. That phenomena indicates the processing speed is image dependent to some extent.

Since thinning technique handles only the edge points of the outermost layer of objects in each iteration, it is more suitable for recognition of the patterns whose lines are not very thick such as characters, numbers, printed circuits, finger prints, biological cells etc.

Because of the quantization of array to express the digital images, the contour of same object at different orientations may have different shapes. That may lead to the formation of skeletons with litter variations.

Hypercube concurrent system offers a flexible structure for us to choose different topologies according to different problems to be processed. Different topologies will lead to different efficiency. Since the topology selection is most important factor to influence the speed of implementation of algorithms on concurrent system, it need to be considered very carefully when creating a parallel program. Mesh

topology is selected to create the concurrent programs in this research. That is because this topology can make use of parallelism better than ring topology for thinning in hypercube concurrent system. Both the amount of data interchanged and the steps to communicate between distant nodes and nodes to the host computer are less in mesh topology than that of ring topology for implementation thinning algorithms on parallel computer.

The main way to increase the efficiency of a concurrent program is to reduce the data communications as much as possible. That can be realized by choosing the suitable topology which can make the amount and steps of data communication to minimum for the problem to be solved.

The proposed changeable value thinning approach described in chapter 3 improves the implementation considerably. This approach reduces both the implementation time and memory space.

REFERENCES

1. Fundamentals of Digital Image Processing , Anil K Jain, Prentice-Hall, 1989
2. Digital Image Processing , R. C. Gonzalez and P. Wintz, Addison-Wesley, 2nd ed., 1987
3. Some parallel thinning algorithms for digit pictures, R. Stefanelli and A. Rosenfeld, J. ACM, Vol 18 No.2, pp255-264
4. A systematic approach for designing 2-subcycle and pseudo parallel thinning algorithms , Yung-Sheng Chen, Pattern Recognition, No.3 1989, pp267-282
5. Methodologies in Pattern Recognition and Image analysis--a brief survey , J.Mantas, Pattern Recognition, No.1 1987, pp1-6
6. Computer processing of line image : a survey , R. W. Smith, Pattern Recognition, No 1 1987, pp7-16
7. A pattern adaptive thinning algorithm , V. K. Govindon and A. P. Shivaprasad, Pattern Recognition, No.6 1987, pp623-638
8. Binary picture thinning by an iterative parallel two-subcycle operation , Satoshi Suzuki and Keiichi Abe, Pattern Recognition, No.3 1987, pp297-308
9. A parallel symmetric thinning algorithm , N. G. Bourbakis, Pattern Recognition, No 4 1989, pp387-396
10. A one -pass Two operation process to detect the skeletal

- pixels on the 4-distance transform , C. Arcelli and G. Sanniti diBaja, IEEE Trans. on Pattern Analysis and Machine Intelligence, No.4 1989, pp411-413
11. Algorithms for Graphics and Image processing , T. Pavlidis, Rockville MD Computer Science Press, 1982
 12. A comparison of line thinning algorithms for digit geometry viewpoint , H. Tamura, Proc. 4th Int Joint Conf. on Pattern Recognition, 1978
 13. Fast parallel thinning algorithms: parallel speed and connectivity preservation , Hall Richard W, CACM32 Ja' 89, pp124-131
 14. A thinning algorithm by contour generation , Kwok. Paul C. K, CACM31 Nov. 88, pp1314-1324
 15. A fast parallel thinning algorithm for thinning digital pattern , Zhang T. Y and Suen C. Y, CACM27 Mr'84, pp236-239
 16. Parallel Thinning with Two-Subiteration Algorithms , Zicheng Guo and Richard W. Hall, CACM32 Mar'89, pp359-373
 17. A comment on A fast parallel algorithm for thinning digital pattern , H. E. Lu and P.S. P. Wang, CACM29 Mr'89, pp239-242
 18. A one-pass thinning algorithm and its parallel implementation , R. T. Chin, Hong-Khoon Wan, D. L. Stover, and R. D. Iverson, computer Vision, Graphics, and Image Processing 40 1987, pp30-40

19. Thinning algorithms on Rectangular, Hexagonal, and Triangular Arrays , E. S. Deutsch, CACM15 Sep. 72, pp827-837
20. An improved parallel algorithm , C. M. Holt, A. Stewart, M. Clint, and H. Perrott, CACM30 Feb. 87, pp156-160
21. Thinning Algorithms for Gray-Scale Picture , C.R. Dyer and A. Rosenfeld, IEEE Trans. on Pattern Analysis and Machine Intelligence, No.1 1979, pp88-89
22. The Ridge-Seeking Method for Obtaining the Skeleton of Digital Images , E. Salari and Pepesiy, IEEE Trans. on Pattern Analysis and Machine
23. Comparison of thinning algorithms on parallel processor, C. J. Hilditch, Image Vision Comput, Aug.1983, pp115-132
24. Hexagonal Parallel Pattern Transforms , M. J. E. Golay, IEEE Trans. on Pattern Analysis and Machine Intelligence, No.8. 1969, pp733-740
25. Robot Vision , Berthold Klaus Paul Horn, The MIT Press McGraw-Hill Book Company 1986
26. Efficient parallel algorithms for image template matching on hypercube SIMD machines , IEEE Trans. on Pattern Analysis and Machine Intelligence, No.6 1989, pp411-413
27. Vision algorithms for hypercube machines , T. N. Mudge and T.S. Abdel-Rahman, J. Parallel Distributed Comput. No.1 Feb.87, pp79-94
28. Parallel algorithms for image template matching on

- hypercube SIMD computer , IEEE Trans. on Pattern Analysis and Machine Intelligence, PAMI-9 Nov. 87, pp79-88
29. AMETEK system 14 User's Guide , AMETEK Computer Research Division 1986
30. An Introduction to Digital Image Processing , Wayne Niblack Prentice-Hall International (UK) Ltd. 1986

Appendix

```

/*****
*
*   This is a parent program for applying thinning
*       algorithm on the concurrent system
*       ( Mesh topology is used )
*
*       Developed by Yichang Xu
*       July 8, 1990
*
*****/

```

```

#include <stdio.h>
#include "nnp.h"
#include "timerp.c"

/*   Global variable declaration */

static unsigned char  d1[512][512], trans[262144];

user_main()
{   /*   Local variables declaration */

    int          i, j;
    static int    t, l, f[16];
    FILE          *f1, *f2;
    static float  sec;
    char          infile[20];

    /*   Input the image for thinning */

    printf("Enter the input image name:\n");
    scanf("%s", infile);
    if ((f1=fopen(infile, "r"))==NULL)
        { printf("can not open file %s \n", infile);
          exit(1);
        }
    fread(d1, sizeof(char), 512*512, f1);
    fclose(f1);

    printf("Complete reading the input image\n");

    /*   Transfer the data of a rectangular array of the
        input image to a string for downloading */

    array_to_string(d1,trans);

    printf("Complete rearranging the data of input
        image\n");

```

```

/*      Initialize the Mesh Topology      */
      nn(4,SAME,1,"xulc.exe");
      printf("Complete initialization\n");

/*      Download the data to multiprocessors      */
      dnldac(trans, 128*128);
      printf("Complete the download procedure\n");

/*      Start the time record procedure      */
      zero_timer(0);

/*      Receive the signal from the children to
      check whether or not thinning procedure should
      continue      */
s1:  upldi(f);
      for (i=0; i<16; i++)
          if (f[i]!=0)
              { sndi(f[i],ALL); I++;
                goto s1;
              }

/*      Record the time for thinning and inform
      multiprocessors to stop thinning      */
      read_timer(&sec);
      sndi(t,ALL);
      printf("iteration num : %d\n",I);
      printf("processing time : %f\n",sec);
      printf("\n");

/*      Upload the processed data from children      */
      upldac(trans,128*128);

      printf("Complete the upload of the thinned
              image\n");

/*      Transfer the data from the form of a string to the
      form of a rectangle array      */
      string_to_array(trans,d1);

      printf("Complete rearrange");

/*      Write the output data into a file      */

```



```

        f2=fopen("thin.img", "w");
        fwrite(d1, sizeof (char), 512*512, f2);
        fclose(f2);

    }
/* End of the main program */

/* Subroutines used in the main function of Parent
   program */

string_to_array(in , out)
unsigned char    in[262144], out[512][512];
{ int    i, j, l, m, f;
  f=0;
  for (l=0; l<4; l++)
      { for (m=0; m<4; m++)
          { for (i=512-(l+1)*128; i<512-l*128; i++)
              { for (j=m*128; j<m*128+128; j++)
                  { out[i][j]=in[f];
                    f++;
                  }
              }
          }
      }
  }

array_to_string(in, out)
unsigned char    in[512][512], out[262144];
{ int    i, j, l, m, f;
  f=0;
  for (l=0; l<4; l++)
      { for (m=0; m<4; m++)
          { for (i=512-(l+1)*128; i<512-l*128; i++)
              { for (j=m*128; j<m*128+128; j++)
                  { out[f]=in[i][j];
                    f++;
                  }
              }
          }
      }
  }
}

```

```

/*****
*
*   This is a child program for thinning the input image
*           by using GH algorithm
*
*           Developed by Yichang Xu
*           July 8, 1990
*****/

#include "nnc.h"
#include "interchangel"

/*   Global variable declaration                                */

static unsigned char  d1[128][128], d2[130][130];
static int            cp1, np1;

void user_main()
{
/*   Local variables declaration                                */

int            i, j;
static int     s, f, t;

/*   Receive the subimage for each processor                    */

dnldac(d1, 128*128);

/*   Interchange the data between the processors                */

c1:  for (i=0; i<128; i++)
      for (j=0; j<128; j++)
          d2[i+1][j+1]=d1[i][j]
          interchangel(d2);

/*   Start the thinning procedure                                */

f=0; I++;
if (I%2==1)

/*   Odd iteration procedure                                    */

{  for (i=1; i<129; i++)
    for (j=1; j<129; j++)
        if (d2[i][j]==99)
            {  if (d2[i][j+1]==0 || (d2[i-1][j]==0 &&
                d2[i-1][j+1]==0 &&
                d2[i+1][j+1]==99))
                {  cp(i,j);
                    if (cp1==1)
                        {  np(i,j);
                            if (np1==3 || np1==2)

```

```

                                { d1[i-1][j-1]=0; f++;
                                }
                            }
                    }

/*  Send and Receive signals to and from parent to check
    whether or not the thinning procedure continues */

    upldi(f);
    rcvi(t,PAR);
    if (t!=0)
        goto c1;
        goto c2;
    }

/*  Even iteration procedure */

    { for (i=1; i<129; i++)
        for (j=1; j<129; j++)
            if (d2[i][j]==99)
                { if (d2[i][j-1]==0 || (d2[i+1][j]==0 &&
                    d2[i+1][j-1]==0 &&
                    d2[i-1][j-1]==99))
                    { cp(i,j);
                        if (cpl==1)
                            { np(i,j);
                                if (npl==3 || npl==2)
                                    { d1[i-1][j-1]=0; f++;
                                    }
                                }
                            }
                    }
            }
        }

/*  Send and Receive signals to and from parent to check
    whether or not the thinning procedure continues */

    upldi(f);
    rcvi(t,PAR);
    if (t!=0) goto c1;
    }

/*  Stop thinning procedure and upload the output image to
    Parent */

c2:  upldac(d1,128*128)
    }

/*  End of the main program */

```

```

/*      Subroutines used in the main program      */
cp(i,j)
int i,j;
{ cp1= ( !d2[i-1][j] && (d2[i-1][j+1] || d2[i][j+1]
)) + ( !d2[i][j+1] && (d2[i+1][j+1] ||
d2[i+1][j] )) + ( !d2[i+1][j] && (d2[i+1][j-1]
|| d2[i][j-1] )) + ( !d2[i][j-1] &&
(d2[i-1][j-1] || d2[i-1][j] ));
}

np(i,j)
int i,j;
{ int n2p;
  np1= ( d2[i-1][j-1] || d2[i-1][j] )+
        ( d2[i-1][j+1] || d2[i][j+1] )+
        ( d2[i+1][j+1] || d2[i+1][j] )+
        ( d2[i+1][j-1] || d2[i][j-1] );

  n2p= ( d2[i-1][j] || d2[i-1][j+1] )+
        ( d2[i][j+1] || d2[i+1][j+1] )+
        ( d2[i+1][j] || d2[i+1][j-1] )+
        ( d2[i][j-1] || d2[i-1][j-1] );
  if ( n2p<np1 )
    np1=n2p;
}

```



```

        if (cpl==1)
        { np(i,j);
          if (npl==3 || npl==2)
          { d2[i][j]=I; f++;
            }
          }
        }
    }

/*  Send and Receive signals to and from parent to
    check whether or not the thinning procedure
    contines */

    upldi(f);
    rcvi(t,PAR);
    if(t!=0)
    goto c1;
    goto c2;
}

/*  Even iteration procedure */

{ for (i=1; i<129; i++)
  for (j=1; j<129; j++)
    if ( d2[i][j]==99 )
      { if ( d2[i][j-1]<I || ( d2[i+1][j]<I &&
        d2[i+1][j-1]<I && d2[i-1][j-1]>=I
        ))
        { cp(i,j);
          if (cpl==1)
          { np(i,j);
            if (npl==3 || npl==2)
            { d2[i][j]=I; f++;
              }
            }
          }
        }
    }

/*  Send and Receive signals to and from parent to
    check whether or not the thinning procedure
    contines */

    upldi(f);
    rcvi(t,PAR);
    if (t!=0) goto c1;
}

/*  Stop thinning procedure and upload the output
    image to Parent */

```

```

c2:      for (i=0; i<128; i++)
          for (j=0; j<128; j++)
              d1[i][j]=d2[i+1][j+1];
          upldac(d1,128*128);
    }

/*      End of the main program                                */

/*      Subroutines used in the main program                    */

cp(i,j)
int i,j;
{
    cp1=0;
    if ( d2[i-1][j]<I && (d2[i-1][j+1]>=I ||
        d2[i][j+1]>=I )) cp1++;
    if ( d2[i][j+1]<I && (d2[i+1][j+1]>=I ||
        d2[i+1][j]>=I )) cp1++;
    if ( d2[i+1][j]<I && (d2[i+1][j-1]>=I ||
        d2[i][j-1]>=I )) cp1++;
    if ( d2[i][j-1]<I && (d2[i-1][j-1]>=I ||
        d2[i-1][j]>=I )) cp1++;
}

np(i,j)
int i,j;
{
    int n2p;
    n2p=0; np1=0;
    if ( d2[i-1][j-1]>=I || d2[i-1][j]>=I )    np1++;
    if ( d2[i-1][j+1]>=I || d2[i][j+1]>=I )    np1++;
    if ( d2[i+1][j+1]>=I || d2[i+1][j]>=I )    np1++;
    if ( d2[i+1][j-1]>=I || d2[i][j-1]>=I )    np1++;

    if ( d2[i-1][j]>=I || d2[i-1][j+1]>=I )    n2p++;
    if ( d2[i][j+1]>=I || d2[i+1][j+1]>=I )    n2p++;
    if ( d2[i+1][j]>=I || d2[i+1][j-1]>=I )    n2p++;
    if ( d2[i][j-1]>=I || d2[i-1][j-1]>=I )    n2p++;
    if ( n2p<np1 )
        np1=n2p;
}

```

```

/*****
*
*   This is a child program for thinning the input image
*           by using OPTA algorithm
*
*           Developed by Yichang Xu
*           July 8, 1990
*****/

```

```

#include "nnc.h"
#include "interchange2"

```

```

/* Global variables declaration */

static unsigned char d1[128][128], d2[131][131];

void user_main( )
{ /* Local variables declaration */

    int i, j;
    static int f, t;

    /* Receive the subimage for each proceessor */

    dnldac(d1,128*128);

c1: /* Interchange the data between the processors */
    for (i=0; i<128; i++)
        for (j=0; j<128; j++)
            d2[i+1][j+1]=d1[i][j];
            interchange2(d2);

    /* Start the thinning procedure */

    f=0;
    for (i=1; i<129;i++)
        { for (j=1; j<129; j++)
            if( d2[i][j]==99 )
                { if (d2[i][j-1]==0)
                    { if (d2[i][j+1]==99 &&
                        d2[i][j+2]==0)
                        goto c2;
                    if (d2[i-1][j]==0 &&
                        d2[i+2][j]==0 &&
                        d2[i+1][j]==99)
                        goto c2;
                    if (d2[i-1][j-1]==0 &&
                        d2[i-1][j]==0 &&
                        d2[i][j+1]==99 &&
                        d2[i+1][j]==99)
                        { d1[i-1][j-1]=0; f++;

```



```
        if (t!=0)
        goto c1;
        upldac(d1,128*128);
    }
```

```

/*****
*
*   This is a child program for thinning the input image
*   by using OPTA algorithm
*   (0-99 binary image changeable value thinning)
*
*           Developed by Yichang Xu
*           July 8, 1990
*****/

```

```

#include "nnc.h"
#include "interchange2"

```

```

/* Global variable declaration */

static unsigned char d1[128][128], d2[131][131];

void user_main()
{ /* Local variables declaration */
    int i, j;
    static int f, t, I;

    /* Receive the subimage for each processor */

    dnldac(d1, 128*128);

    /* Interchange the data between the processors */
    for (i=0; i<128; i++)
        for (j=0; j<128; j++)
            d2[i+1][j+1]=d1[i][j];
c1:    interchange2(d2);

    /* Start the thinning procedure */

    f=0; I++;
    for (i=1; i<129; i++)
        { for (j=1; j<129; j++)
            { if (d2[i][j]==99 )
                { if (d2[i][j-1]<I)
                    { if (d2[i][j+1]>=I && d2[i][j+2]<I)
                        goto c2;
                    if (d2[i-1][j]<I && d2[i+2][j]<I &&
                        d2[i+1][j]>=I) goto c2;
                    if (d2[i-1][j-1]<I && d2[i-1][j]<I
                        &&
                        d2[i][j+1]>=I && d2[i+1][j]>=I)
                        { d2[i][j]=I; f++; goto c2; }
                    if (d2[i+1][j-1]<I && d2[i+1][j]<I
                        &&
                        d2[i-1][j]>=I && d2[i][j+1]>=I)
                        { d2[i][j]=I; f++; goto c2; }
                }
            }
        }
    }

```

```

        if (d2[i-1][j-1]<I &&
            d2[i+1][j-1]<I &&
            d2[i-1][j]>=I &&
            d2[i+1][j]>=I &&
            d2[i][j+1]>=I)
        { d2[i][j]=I; f++; goto c2;}
        goto c2;
    }

    if (d2[i-1][j]<I)
    { if (d2[i+2][j]<I && d2[i+1][j]>=I) goto c2;
      if (d2[i+1][j]>=I && d2[i][j+1]<I &&
          d2[i-1][j+1]<I)
        {d2[i][j]=I; f++; goto c2;}
      if (d2[i-1][j-1]<I && d2[i-1][j+1]<I &&
          d2[i][j+1]>=I && d2[i+1][j]>=I)
        {d2[i][j]=I; f++; goto c2;}
      goto c2;
    }

    if (d2[i][j+1]<I)
    { if (d2[i+1][j]<I && d2[i+1][j+1]<I)
        { d2[i][j]=I; f++; goto c2;}

        if (d2[i+1][j]>=I && d2[i+1][j+1]<I &&
            d2[i-1][j+1]<I)
        { d2[i][j]=I; f++; goto c2;}
        goto c2;
    }

    if (d2[i+1][j]<I)
    { if (d2[i+1][j-1]<I && d2[i+1][j+1]<I)
        {d2[i][j]=I; f++; goto c2;}
        goto c2;
    }
}

c2:  ;}
    }

/*  Send  and  Receive  signals  to  and  from  parent  to
    check  whether  or  not  the  thinning  procedure
    contines  */

upldi(f); rcv(t,PAR);
if (t!=0)
goto c1;

/*  Stop  thinning  procedure  and  upload  the  output  image
    to  Parent  */

for (i=0; i<128; i++)
    for (j=0; j<128; j++)
        d1[i][j]=d2[i+1][j+1];

```

```
        upldac(d1,128*128);  
    }
```

```

/*****
*
*   This is a child program for thinning the input image
*   by using improved ZS algorithm
*
*           Developed by Yichang Xu
*           July 8, 1990
*****/

#include "nnc.h"
#include "interchange2"
/* Global variable declaration */

static unsigned char d1[128][128], d2[130][130];

void user_main( )
{ /* Local variables declaration */
    int i, j, Ap, Bp;
    static int I, f, t;

    /* Receive the subimage for each proceess */

    dnldac(d1,128*128);

    /* Interchange the data between the processors */

c1:    for (i=0; i<128; i++)
        for (j=0; j<128; j++)
            d2[i+1][j+1]=d1[i][j];
            interchange1(d2);

    /* Start the thinning procedure */

    f=0; I++;

    /* Odd iteration procedure */

    if (I%2==1)
        { for (i=1; i<129; i++)
            for (j=1; j<129; j++)
                if (d2[i][j]==99)
                    {if (d2[i][j+1]==0 || d2[i+1][j]==0 ||
                        (d2[i][j-1]==0 && d2[i-1][j]==0 ))
                        { Bp=0;
                            if (d2[i-1][j-1]==99) Bp++;
                            if (d2[i-1][j]==99) Bp++;
                            if (d2[i-1][j+1]==99) Bp++;
                            if (d2[i][j+1]==99) Bp++;
                            if (d2[i+1][j+1]==99) Bp++;
                            if (d2[i+1][j]==99) Bp++;
                            if (d2[i+1][j-1]==99) Bp++;
                            if (d2[i][j-1]==99) Bp++;
                        }
                    }
            }
}

```

```

        if (Bp>=3 && Bp<=6)
        { Ap=0;
          if (d2[i-1][j-1]==0 &&
              d2[i-1][j]==99) Ap++;
          if (d2[i-1][j]==0 &&
              d2[i-1][j+1]==99)
              Ap++;
          if (d2[i-1][j+1]==0 &&
              d2[i][j+1]==99) Ap++;
          if (d2[i][j+1]==0 &&
              d2[i+1][j+1]==99)
              Ap++;
          if (d2[i+1][j+1]==0 &&
              d2[i+1][j]==99) Ap++;
          if (d2[i+1][j]==0 &&
              d2[i+1][j-1]==99) Ap++;
          if (d2[i+1][j-1]==0 &&
              d2[i][j-1]==99) Ap++;
          if (d2[i][j-1]==0 &&
              d2[i-1][j-1]==99) Ap++;

          if (Ap==1)
          { d1[i-1][j-1]=0; f++;
            goto c2;}

        if (Ap==2)
        { if (d2[i][j+1]==99)
          { if (d2[i+1][j]==99)
            if (d2[i-1][j-1]==0)
            { d1[i-1][j-1]=0;
              f++; goto c2;
            }
          }

        if (d2[i-1][j]==99)
        if (d2[i+1][j-1]==0)
        if (d2[i][j-1]==0)
        if (d2[i-1][j+1]==0)
        { d1[i-1][j-1]=0;
          f++;
        }
        }
        }
        }
        }
c2:
        ;
    }
}

/* Send and Receive signals to and from parent to
   check whether or not the thinning procedure
   continues */

upldi(f); rcvi(t,PAR);
if (t!=0)

```



```

                                if (d2[i+1][j-1]==0)
                                    if (d2[i][j+1]==0)
                                        { d1[i-1][j-1]=0;
                                          f++;
                                        }
                                    }
                                }
c3:                               ;
                                }
                                }
                                }

/*    Send    and    Receive signals to and from parent to
check whether or not the thinning procedure continues */

    upldi(f); rcvi(t,PAR);
    if (t!=0) goto c1;
}

/*    Stop thinning procedure and upload the output image
to Parent */
c4:   upldac(d1,128*128);
}
```

```

/*****
*
*   This is a child program for thinning the input image
*   by using improved ZS algorithm
*   ( 0-99 binary image changeable value thinning )
*
*           Developed by Yichang Xu
*           July 8, 1990
*****/

#include "nnc.h"
#include "interchange1"

/* Global variable declaration */

static unsigned char d1[128][128], d2[130][130];

void user_main()
{ /* Local variables declaration */

    int i, j, Ap, Bp;
    static int I, f, t;

    /* Receive the subimage for each processor */

    dnldac(d1, 128*128);

    /* Interchange the data between the processors */

    for (i=0; i<128; i++)
    for (j=0; j<128; j++)
        d2[i+1][j+1]=d1[i][j];
c1:    interchange1(d2);

    /* Start the thinning procedure */

    f=0; I++;
    if (I%2==1)

    /* Odd iteration procedure */

    { for (i=1; i<129; i++)
      for (j=1; j<129; j++)
        if (d2[i][j]==99)
          {if (d2[i][j+1]<I || d2[i+1][j]<I ||
              (d2[i][j-1]<I && d2[i-1][j]<I))
              { Bp=0;
                if (d2[i-1][j-1]>=I) Bp++;
                if (d2[i-1][j]>=I) Bp++;
                if (d2[i-1][j+1]>=I) Bp++;
                if (d2[i][j+1]>=I) Bp++;
                if (d2[i+1][j+1]>=I) Bp++;

```

```

if (d2[i+1][j]>=I)      Bp++;
if (d2[i+1][j-1]>=I)    Bp++;
if (d2[i][j-1]>=I)      Bp++;

if (Bp>=3 && Bp<=6)
{ Ap=0;
  if (d2[i-1][j-1]<I &&
      d2[i-1][j]>=I)      Ap++;
  if (d2[i-1][j]<I &&
      d2[i-1][j+1]>=I)    Ap++;
  if (d2[i-1][j+1]<I &&
      d2[i][j+1]>=I)      Ap++;
  if (d2[i][j+1]<I &&
      d2[i+1][j+1]>=I)    Ap++;
  if (d2[i+1][j+1]<I &&
      d2[i+1][j]>=I)      Ap++;
  if (d2[i+1][j]<I &&
      d2[i+1][j-1]>=I)    Ap++;
  if (d2[i+1][j-1]<I &&
      d2[i][j-1]>=I)      Ap++;
  if (d2[i][j-1]<I &&
      d2[i-1][j-1]>=I)    Ap++;

  if (Ap==1)
  { d2[i][j]=I; f++; goto c2;}
  if (Ap==2)
  { if (d2[i][j+1]>=I)
    { if (d2[i+1][j]>=I)
      if (d2[i-1][j-1]<I)
      { d2[i][j]=I;
        f++; goto c2;
      }
    }

    if (d2[i-1][j]>=I)
    if (d2[i+1][j-1]<I)
    if (d2[i][j-1]<I)
    if (d2[i-1][j+1]<I)
    { d2[i][j]=I; f++;
    }
  }
}
}
}

c2:
;

}

}

}

/* Send and Receive signals to and from parent to
   check whether or not the thinning procedure
   contines */
upldi(f); rcvi(t,PAR);
if (t!=0)
goto c1;

```

```

        goto c4;
    }

/*    Even iteration procedure                                */

{ for (i=1; i<129; i++)
  for (j=1; j<129; j++)
    if (d2[i][j]==99)
      { if (d2[i][j-1]<I || d2[i-1][j]<I ||
          (d2[i+1][j]<I && d2[i][j+1]<I))
        { Bp=0;
          if (d2[i-1][j-1]>=I)   Bp++;
          if (d2[i-1][j]>=I)     Bp++;
          if (d2[i-1][j+1]>=I)   Bp++;
          if (d2[i][j+1]>=I)     Bp++;
          if (d2[i+1][j+1]>=I)   Bp++;
          if (d2[i+1][j]>=I)     Bp++;
          if (d2[i+1][j-1]>=I)   Bp++;
          if (d2[i][j-1]>=I)     Bp++;

          if (Bp>=3 && Bp<=6)
            { Ap=0;
              if (d2[i-1][j-1]<I &&
                  d2[i-1][j]>=I)   Ap++;
              if (d2[i-1][j]<I &&
                  d2[i-1][j+1]>=I) Ap++;
              if (d2[i-1][j+1]<I &&
                  d2[i][j+1]>=I)   Ap++;
              if (d2[i][j+1]<I &&
                  d2[i+1][j+1]>=I) Ap++;
              if (d2[i+1][j+1]<I &&
                  d2[i+1][j]>=I)   Ap++;
              if (d2[i+1][j]<I &&
                  d2[i+1][j-1]>=I) Ap++;
              if (d2[i+1][j-1]<I &&
                  d2[i][j-1]>=I)   Ap++;
              if (d2[i][j-1]<I &&
                  d2[i-1][j-1]>=I) Ap++;

              if (Ap==1)
                { d2[i][j]=I;  f++;
                  goto c3;}
              if (Ap==2)
                {if (d2[i][j-1]>=I)
                  { if (d2[i-1][j]>=I)
                      if (d2[i+1][j+1]<I)
                        { d2[i][j]=I;  f++;
                          goto c3;
                        }
                    }
                }

              if (d2[i+1][j]>=I)
                if (d2[i-1][j+1]<I)

```

```

                                if (d2[i+1][j-1]<I)
                                    if (d2[i][j+1]<I)
                                        { d2[i][j]=I; f++;
                                        }
                                    }
                                }
                                ;
c3:                            }
                                }
                                }
                                }

/*  Send and Receive signals to and from parent to
    check whether or not the thinning procedure
    contines */

    upldi(f);  rcvi(t,PAR);
    if (t!=0)
        goto c1;
}

/*  Stop thinning procedure and upload the output image
    to Parent */
c4:  for (i=0; i<128; i++)
      for (j=0; j<128; j++)
          d1[i][j]=d2[i+1][j+1];
          upldac(d1,128*128);
}

```

```
/* subroutines used in the child program
```

```
*/
```

```
interchange1(d2)
unsigned char d2[130][130];
{ int i;
  static unsigned char sbuf[130], rbuf[130];

  sndac(&d2[1][1], 128, NORTH);
  rcvac(&d2[129][1], 128, SOUTH);
  sndac(&d2[128][1], 128, SOUTH);
  rcvac(&d2[0][1], 128, NORTH);

  for (i=0; i<130; i++)
    sbuf[i]=d2[i][128];
  sndac(&sbuf, 130, EAST);
  rcvac(&rbuf, 130, WEST);

  for (i=0; i<130; i++)
    d2[i][0]=rbuf[i];

  if (childnum %4 ==3)
    for (i=0; i<130; i++)
      rbuf[i]=0;

  for (i=0; i<130; i++)
    sbuf[i]=d2[i][1];

  sndac(&sbuf, 130, WEST);
  rcvac(&rbuf, 130, EAST);

  for (i=0; i<130; i++)
    d2[i][129]=rbuf[i];

  for (i=0; i<130; i++)
    { sbuf[i]=0; rbuf[i]=0; }

}
```

```
interchange2(d2)
unsigned char d2[131][131];
{ int i;
  static unsigned char sbuf[130], rbuf[130];

  sndac(&d2[1][1], 128, NORTH);
  rcvac(&d2[129][1], 128, SOUTH);

  sndac(&d2[2][1], 128, NORTH);
  rcvac(&d2[130][1], 128, SOUTH);

  sndac(&d2[128][1], 128, SOUTH);
  rcvac(&d2[0][1], 128, NORTH);
```

```

    for (i=0; i<130; i++)
        sbuf[i]=d2[i][128];

    sndac(&sbuf, 130, EAST);
    rcvac(&rbuf, 130, WEST);

    for (i=0; i<130; i++)
        d2[i][0]=rbuf[i];

    if (childnum %4 ==3)
        for (i=0; i<130; i++)
            rbuf[i]=0;

    for (i=0; i<130; i++)
        sbuf[i]=d2[i][1];

    sndac(&sbuf, 130, WEST);
    rcvac(&rbuf, 130, EAST);

    for (i=0; i<130; i++)
        d2[i][129]=rbuf[i];

    for (i=0; i<128; i++)
        sbuf[i]=d2[i+1][2];

    sndac(&sbuf, 128, WEST);
    rcvac(&rbuf, 128, EAST);

    for (i=0; i<128; i++)
        d2[i+1][130]=rbuf[i];

    for (i=0; i<130; i++)
        { sbuf[i]=0; rbuf[i]=0; }

}

```