

An Efficient Fully Parallel Thinning Algorithm

N. H. Han, C. W. La, and P. K. Rhee,

Computer Science and Engineering, Dept. Inha University

Yong-Hyun Dong 253, Nam Ku, Incheon, Korea, E-mail: pkrhee@dragon.inha.ac.kr

Tel: +82-32-860-7448, Fax: +82-32-874-6682

Abstract

This paper addresses an efficient parallel thinning algorithm based on weight-values. The weight-value of a black pixel is calculated by observing neighboring pixels, and it gives us an efficient way to decide whether the pixel is deleted or not. Owing to weight-values, the proposed algorithm uses only 3×3 templates. Furthermore, it examines only the elimination conditions corresponding the weight-value of boundary pixels, and all elimination conditions will not be searched as most other parallel iterative thinning algorithms. Thus, the execution time can be reduced much comparing to that of previous approaches. The weight-value also allows us to deal with typical troublesome patterns efficiently. Without smoothing before thinning, the algorithm produces robust thinned images even in the presence of two pixel-width noises. We obtain encouraging results from extensive experiments.

Keywords: Parallel thinning algorithm, weight-value, elimination condition, connectivity preservation, image pattern, execution time

1. Introduction

Thinning is a very important technique extensively used in the areas of pattern recognition, visual inspection, character recognition, fingerprint recognition, etc. The objective of thinning is to reduce the amount of information in image patterns to the minimum needed for recognition. Thinned image helps the extraction of important features such as end points, junction points, and connections from image patterns. Thus, many thinning algorithms have been proposed until now[8]. Two major approaches of thinning digital patterns can be categorized into iterative boundary removal algorithms[1, 2, 3, 5, 9, 7] and distance transformation algorithms[12]. Iterative boundary removal algorithms delete pixels on the boundary of a pattern repeatedly until only unit

pixel-width thinned image remains. Distance transformation algorithms are not appropriate for general applications since they are not robust, especially for patterns with highly variable stroke directions and thicknesses. Thinning based on iterative boundary removal can be divided into sequential and parallel algorithms. In a sequential algorithm, the deletion of a pixel in the n -th iteration depends on the already processed result. Many parallel thinning algorithms are not fully parallel, i.e. 2 subiterations[4, 11] or 4 subiteration[2]. Frank's algorithm is fully parallel algorithm. However, it use two scanings and four scanings partially. Jang's algorithm does not use subiteration, but it needs four types of templates(3×3 , 4×3 , 3×4 , 5×5). Approaches using either subiterations or using various types of templates often fall down inefficiency in performance [2]. Using the concept of weight-values, the proposed algorithm produces shape-preserving thinned images, shows robustness to one or two pixel-width boundary noises, and executes faster than previously proposed approaches. The proposed thinning algorithm does not use subiteration, and thus it is a fully parallel thinning algorithm. It uses only 3×3 templates for elimination conditions and requires small number of templates to be examined. Thus, it requires smaller number of pixels to be investigated than other parallel algorithms, leading to faster execution. The algorithm produces consistent thinned images even when two images are similar in shape but different in boundary noises. Extensive experiments shows that the proposed algorithm is faster than other parallel thinning algorithms, especially then the pixel-width of image is thick.

2. Some Definitions and Preliminary Concepts

Input image is represented by black pixels and white pixels. Black pixels and white pixels are denoted as 1's and 0's, respectively. Eight adjacent pixels of a black pixel p denoted as pixels x_1 , x_2 , x_3 , x_4 , x_5 , x_6 , x_7 , and x_8 are defined as 8-neighbors of p (see Fig. 1), and denoted by $N(p)$. The weight-value of a black pixel p is defined as the number of black pixels in $N(p)$. The weight-value of p is denoted by

This work was supported in part by the grant of Inha University and Ministry of Information and Communication.

$W(p)$. The smallest weight-value of a black pixel might be 0, where no black pixel is found in its 8-neighbors. But, 9 is used to denote weight-value 0 since number 0 is already used to denote a white pixel.

X_1	X_2	X_3
X_8	p	X_4
X_7	X_6	X_5

Fig 1. 8-neighbors of a pixel p in 3×3 template.

A *boundary pixel* is a black pixel which has at least one white pixel in its 8-neighbors. That is, a black pixel whose weight value is between 1 and 7 is a boundary pixel. A group of pixels G is said to be 8-connected if for every pair of pixels p and q , there is a sequence of pixels $a_1(=p)$, $a_2, \dots, a_n(=q)$, where a_{k-1} is a 8-neighbor of a_k for $k > 1$ and $a_i \in G$ ($1 < i < n$). A pixel p is *deletable* if its removal does not change 8-connectivity of p . Parallel thinning algorithms based on iterative boundary removal use elimination conditions represented by $i \times j$ templates[3, 7] or/and elimination rules[4, 5]. Elimination conditions are used to determine whether a black pixel is removed (i.e. changed to white pixel) or not. Fig. 1 shows the form of a 3×3 template for pixel p . Many parallel iterative boundary approaches use 5×5 templates, 4×4 templates as well as 3×3 templates. The performance of an iterative boundary removal approach is dependent upon the size of templates used for elimination, and the number of templates to be searched for each removal. Owing to weight-values, the proposed algorithm needs only 3×3 templates. Furthermore, it examines only elimination conditions corresponding the weight-value of boundary pixels, and all elimination conditions will not be searched as most other iterative thinning algorithms. Thus, the time for searching templates can be reduced much, comparing to that of previous approaches.

3. The Proposed Thinning Algorithm

In this section, the proposed parallel thinning algorithm will be discussed. Elimination conditions are derived for each weight-value. Thinning is performed by deciding whether boundary pixels of image patterns are eliminated or not using the elimination conditions. The elimination conditions are given in Fig. 2. Each elimination condition consists of elimination templates or elimination rules, or both. A deletable pixel is determined by observing only elimination conditions corresponding the pixel's weight value. For example, a black pixel whose weight-value is 7 requires to search only 4 elimination conditions. Note that almost all efficient parallel thinning algorithms search all its elimination conditions for each removal. There are 7 groups of elimination conditions according to their weight-values. *ECG i*

(Elimination Condition Group i) is the group of elimination conditions the weight-values of whose central pixels are i , $1 \leq i \leq 7$. A pixel having

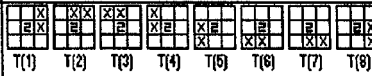

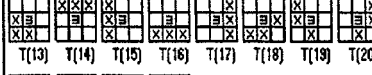
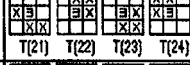

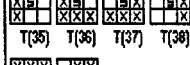
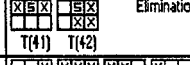
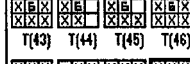
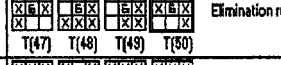
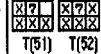

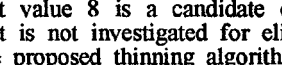
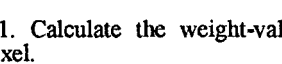
ECG1	Elimination rule: X is equal to or greater than 3.
ECG2	<div>  </div> <div>  </div> <div>Elimination rule: at least one X must be equal to or greater than 3.</div>
ECG3	<div>  </div> <div>  </div> <div>Elimination rule: at least one X must be equal to or greater than 7.</div>
ECG4	<div>  </div> <div>  </div> <div>  </div> <div>  </div>
ECG5	<div>  </div> <div>  </div> <div>Elimination rule: at least one X must be 8.</div>
ECG6	<div>  </div> <div>  </div> <div>Elimination rule: at least one X must be 8.</div>
ECG7	<div>  </div> <div>Elimination rule: at least two X must be 8.</div>

Fig. 2 Elimination conditions for each weight-value.

weight value 8 is a candidate of medial axis, and thus it is not investigated for elimination. The outline of the proposed thinning algorithm is as follows.

- Step 1. Calculate the weight-value for each black pixel.
- Step 2. For each boundary pixel having weight-value k , search through the elimination conditions for weight value k ($1 \leq k < 8$). If an elimination condition is matched to the boundary pixel, the pixel is deleted.
- Step 3. Repeat steps 1 and 2 repeatedly until there is no black pixel satisfying elimination conditions.

Note that all elimination conditions will not be searched as most other iterative thinning algorithms[1, 2, 4, 7], but only elimination conditions corresponding the weight-value of a boundary pixel will be searched. Thus, the proposed algorithm can be executed faster than other parallel thinning algorithms.

4. Performance Evaluation

In this section, the proposed algorithm is evaluated and compared with other famous thinning approaches which show relatively higher performance than previous approaches. The most important criteria of thinning algorithms are connectivity preservation, quality of skeletons, robustness to boundary noise sensitivity, and execution speed. The proof of connectivity preservation will not be given due to the lack of space. Five algorithms that compared with the proposed algorithm are Guo's[4], Suen's[9], Lu's[11], Frank's[3], and Jang's[7]. Guo's, Zang's, and Lu's algorithms are two-subiteration thinning algorithms, and Frank's and Jang's are fully parallel thinning algorithms.

4.1 The quality of skeletons

The quality of skeleton of the proposed algorithm has been compared with other algorithms using images of typical test patterns.

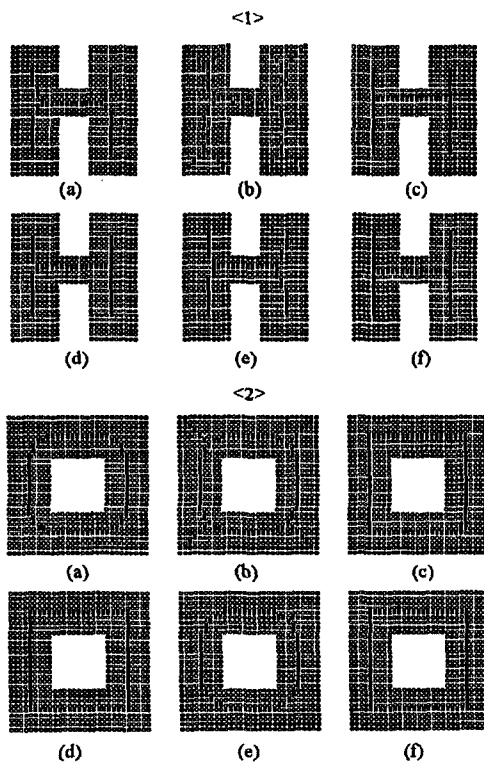


Fig 3. Comparison of skeleton in various images (a) Guo's algorithm, (b) Suen's algorithm, (c) Lu's algorithm, (d) Jang's algorithm, (e) Frank's algorithm, and (f) the proposed algorithm.

4.2 Robustness to noise sensitivity

The criterion of boundary noise sensitivity measures how sensitive is the thinned image to boundary noises. A thinning algorithm should not be sensitive to boundary noises. Most thinning algorithm cannot treat boundary noises properly, while the proposed algorithm can deal with boundary noises of one pixel-width, and two pixel-width. Fig. 4 shows an example. Boundary noise of one pixel-width can be removed (e.g. pixel (1,5),(6,6) of Fig. 4). If black pixels in the rectangle of Fig. 4(a) is a valuable data part, the pixel (3,5),(6,5),(6,6) as well as (1,5) must be treated as noise pixels. Most thinning algorithms can not treat such boundary noises efficiently [1, 4, 5, 6]. Therefore, the thinned image may be distorted after thinning. Some algorithms could treat one pixel-width boundary noise [7]. However, any of them can not treat two pixel-width boundary noises like (6,5) and (6,6). Some thinning algorithms could treat such noises, however, it need too much overhead because they need 3×6 and 6×3 templates to distinguish noises from data. The propose algorithm can treat the noise easily by the elimination conditions 1 and 3. Fig. 4(b) is the result of thinning Fig. 4(a) using Jang's algorithm and 12(c) is the result of thinning Fig. 4(a) using proposed algorithm. Jang's algorithm produces a distorted result, but the proposed algorithm gives a stable result.

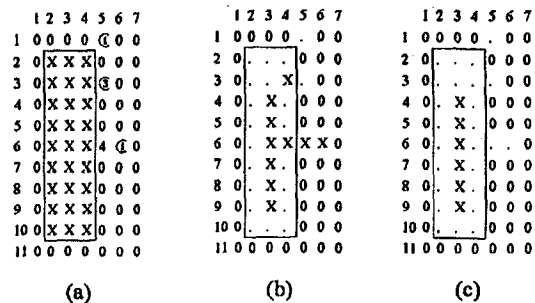


Fig. 4 Comparison of boundary noise removal capability: (a) an image with boundary noises (b) the result of thinning using Jang's algorithm, and (c) the result of the proposed algorithm.

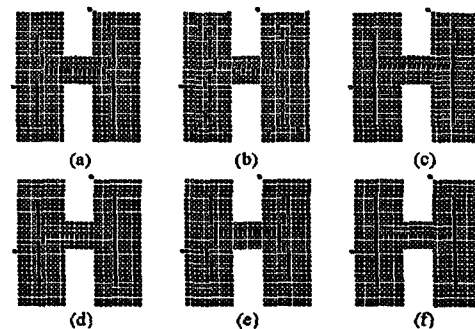


Fig. 5 Comparison of one pixel boundary noise

sensitivity: (a) Guo's algorithm, (b) Suen's algorithm, (c) Lu's algorithm, (d) Jang's algorithm, (e) Frank's algorithm, and (f) The proposed algorithm.

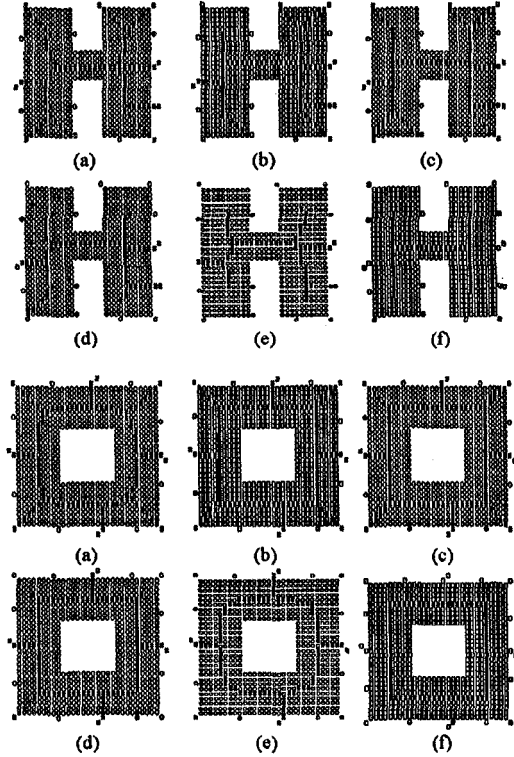


Fig. 6 Comparison of one or two pixel boundary noise sensitivity: (a) Guo's algorithm, (b) Suen's algorithm, (c) Lu's algorithm, (d) Jang's algorithm, (e) Frank's algorithm, and (f) the proposed algorithm.

4.3 Execution speed

The ultimate purpose of parallel thinning algorithms is the reduction of execution speed. However, there has been no systematical approach for comparing the execution speed of parallel thinning algorithms. Several rough comparisons have been done at the level of iteration numbers[4]. But, such comparisons may lead to wrong guides since the time required for iterations varies from one algorithm to the other algorithm. In this paper, we adopt the parallel computer model, PRAM(Parallel Random Access Machine), which is a generally accepted model of parallel computation. PRAM is essentially a collection of RAMs[12, 13, 14] all accessing the same memory. PRAM consists of processors, all of which have uniformly access to a shared memory. Processors share a common clock but may execute different instructions in each cycle. Therefore, a PRAM

computer operates in a synchronous manner. PRAM can be divided into four types depending on how concurrent memory accesses are processed[14]: Exclusive-Read and Exclusive-Write (EREW), Exclusive-Read and Concurrent-Write (ERCW), Concurrent-Read and Exclusive-Write (CREW), and Concurrent-Read and Concurrent-Write (CRCW). The proposed algorithm and other five thinning algorithms are compared based on PRAM model. The model for estimating execution times(ET) of parallel algorithms will be discussed. In each iteration, boundary pixels are checked to be deletable or not, and deleted if they are deletable. Instructions operated in an iteration are categorized into read operations(RDs), decision operations(DSs), write operations(WRs), and miscellaneous operations(MSs). Time required for executing an iteration varies from one algorithm to other algorithm. Some algorithms require preliminary operations, which we call them Pre-iteration operations(Pis). For example, PI is the operation to be performed during thinning operation not before thinning operation. Frank's algorithm uses expansion operations to treat boundary noises at the first and second iterations. Time required for the i -th iteration for a processor k is denoted by $IT_{(k,i)}$. Thus, the execution time for i -th iteration processor k for a parallel thinning algorithm is represented by

$$IT_{(k,i)} = RD_{(k,i)} + DS_{(k,i)} + WR_{(k,i)} + MS_{(k,i)} \quad (1)$$

The proposed algorithm spends 8 RD operations if either EREW or ERCW model is used. It requires 1 RD operation if either CREW or CRCW model is used, and it spends 1 WR for all models. Note that the time for RD and WR operations is the same for all processors. It is not possible to calculate equation (1) exactly since a maximum time spent by a processor must be decided experimentally. It is assumed that with high probability at least one processor spends longest time which is required for DS operations, since there are sufficient boundary pixels in each iteration. Therefore, a worst case analysis is adopted here, and thus equation (2) is approximated by

$$ET_w = PI + \text{Longest}(IT_{(k)}) \times N \quad (2)$$

where $\text{Longest}(IT_{(k)})$ represents the time required for a processor which spends the longest time of each iteration. Average case analysis can be performed by estimating average execution time of iterations experimentally by the following equation.

$$ET_A = PI + \text{Average}(IT_{(k)}) \times N \quad (3)$$

$\text{Average}(IT_{(k)})$ is the average execute time of iterations. Note that in algorithms which use only elimination rules, $\text{Longest}(IT_{(k)})$ and $\text{Average}(IT_{(k)})$ are the same. Guo's algorithm is given in [4]. Guo's algorithm is a two-subiteration algorithm. Suen's algorithm is also a two subiteration parallel thinning algorithm. Lu's algorithm is algorithm improved from

Guob's algorithm. *Jang's algorithm* is a fully parallel thinning algorithm. The algorithm uses 20 elimination templates and 10 restoring templates. The elimination templates are twelve 3×3 , four 4×4 , and four 5×5 templates. The restoring templates are three 3×3 , three 4×3 , four 4×4 templates. *Frank's algorithm* uses index numbers ranging from 1 to 256. Frank's algorithm and the proposed algorithm need MS operations to get index number and to calculate weight-values respectively. Frank's algorithm calculates the index numbers before checking the elimination conditions in each iteration. The proposed algorithm also calculates weight-values in each iteration. For all parallel thinning algorithms, WR is 1 for both exclusive write and concurrency write models. We assume that all the execution time of instructions takes the same time and instructions are executed in sequential order in each processor. Guo's, Suen's and Lu's algorithms use elimination rules to check deletable pixels. Jang's algorithm uses only elimination templates. Frank's algorithm and the proposed algorithm use both elimination rule(s) and templates. Experimental results are given in the following tables.

Table 1. Comparison of the number of operations per iteration: worst case analysis.

Machine	Algorithm					
	Guo's	Suen's	Lu's	Jang's	Frank's	The proposed
EREW	PI = 0 RDs = 8 DSs = 40 WRs = 1 MS = 0	PI = 0 RDs = 8 DSs = 42 WRs = 1 MS = 0	PI = 0 RDs = 8 DSs = 42 WRs = 1 MS = 0	PI = 0 RDs = 24 DSs = 194 WRs = 1 MS = 0	PI = 52 RDs = 24 DSs = 34 WRs = 1 MS = 23	PI = 0 RDs = 8 DSs = 46 WRs = 1 MS = 16
CREW	PI = 0 RDs = 1 DSs = 40 WRs = 1 MS = 0	PI = 0 RDs = 1 DSs = 42 WRs = 1 MS = 0	PI = 0 RDs = 1 DSs = 42 WRs = 1 MS = 0	PI = 0 RDs = 1 DSs = 194 WRs = 1 MS = 0	PI = 20 RDs = 1 DSs = 34 WRs = 1 MS = 16	PI = 0 RDs = 1 DSs = 46 WRs = 1 MS = 9

Table 2 shows the number of iterations for each thinning algorithm when they are applied to the images of Fig. 3. Experiments also have been performed using 750 handwritten numeral data, and the results are given in Table 3.

Table 2. The iteration number of each algorithm for various image.

algorithm image	Guo's	Suen's	Lu's	Jang's	Frank's	The proposed
Fig. 3 <1>	14	10	10	6	8	7
Fig. 3 <2>	14	9	9	6	6	5

Table 3. The iteration number of each algorithm for 750 numeral data.

Algorithm	Guo's	Suen's	Lu's	Jang's	Frank's	The proposed
Iteration number	9.25	6.89	6.60	5.25	5.63	5.45
ET _L	49	51	51	219	186	71
ET _A	49	51	51	158.35	186	59.55

REFERENCES

- [1] R. T. Chin, H. K. Wan, D. L. Stover, and R. D. Iverson, "A One-pass Thinning Algorithm and its Parallel Implementation," *Computer Vision Graphics Image Processing*, vol. 40, no. 1, pp.30-40, Oct. 1987.
- [2] A. Datta and S. K. Parui, "A Robust Parallel Thinning Algorithm for Binary Images," *Pattern Recognition*, vol. 27, no. 9, pp.1181-1192, 1994.
- [3] Frank Y. Shin and Wai-Tak Wong, "Fully Parallel Thinning with Tolerance to Boundary Noise," *Pattern Recognition*, vol. 27, no. 12, pp.1677-1695, 1994.
- [4] Z. Guo and R. Q. Hall, "Parallel Thinning with Two-subiteration Algorithms," *Commun. Assoc. Comput. Mach.*, vol. 32, no. 3, pp.359-373, Mar 1989.
- [5] R. W. Hall, "Fast Parallel Thinning Algorithms: Parallel Speed and Connectivity Preservation," *Commun. Assoc. Comput. Mach.*, vol. 32, no. 1, pp.124-131, Jan. 1989.
- [6] C. M. Holt, A. Stewart, M. Clint, and R. H. Pettott, "An Improved Parallel Thinning Algorithm," *Comm. of the ACM*, vol. 30, no. 2, 1987, pp.156-160.
- [7] Ben k. Jang and Roland T. Chin, "One-Pass Parallel Thinning: Analysis, Properties, and Quantitative Evaluation," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-14, no. 11, pp.1129-1149, Nov. 1992.
- [8] Louisa Lam and Seong-Whan Lee, "Thinning Methodologies-A Comprehensive Survey," *IEEE Trans. Patt. Anal. Machine Intell.*, vol. PAMI-14, no. 9, pp.869-885, Sep 1992.
- [9] Y.K. Chu and C.Y. Suen, "An Alternative Smoothing and Stripping Algorithm for Thinning Digital Binary Patterns," *Signal Processing*, vol. 11, no. 3, pp.207-222, 1986.
- [10] J. I. Toriwaki and S. Yokoi, "Distance Transformation and Skeleton Digitalized Pictures with Applications," *Progress in Pattern Recognition*, (L. N. Kandel and A. Rosenfeld, Eds), The Netherlands North-Holland, pp.187-265, 1981.
- [11] Lu.H.E. and Wang.P.S.P., "An improved fast parallel thinning algorithm for digital patterns", In *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, 1985, p.364-367.
- [12] George S. Almasi and Allan Gottlieb, "Highly Parallel Computing 2th edition", pp. 170-201, 1994.
- [13] Richard Neapolitan and Kumars Naiipour, "Foundations of Algorithms", pp.415-426, 1996.
- [14] J. E. Hopcroft, and J. D. Ullman, "The Design and Analysis of Computer Algorithms", pp.5-31,1974.