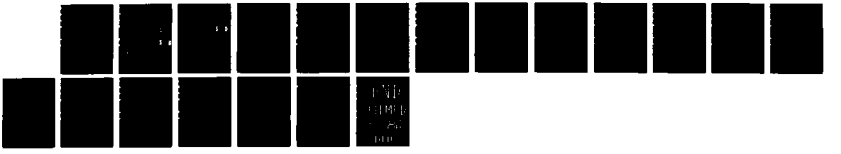
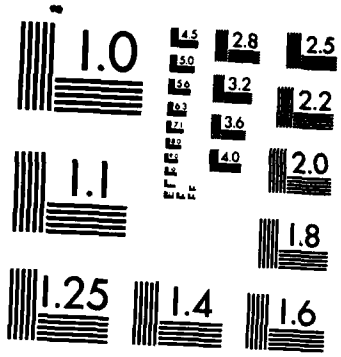
AD-A164 112 PARALLEL PROCESSING OF REGION BOUNDARIES(U) MARYLAND 1/1
UNIV COLLEGE PARK CENTER FOR AUTOMATION RESEARCH
A Y WU ET AL NOV 85 CAR-TR-159 AFOSR-TR-85-1216
UNCLASSIFIED F49620-85-K-0009 F/G 9/2 NL





CAR-TR-159
CS-TR-1573

F49620-85-K-0009
November 1985

PARALLEL PROCESSING OF REGION BOUNDARIES

Angela Y. Wu
Department of Mathematics, Statistics and Computer Science
American University
Washington, DC 20016

S.K. Bhaskar
Azriel Rosenfeld
Center for Automation Research
University of Maryland
College Park, MD 20742

COMPUTER VISION LABORATORY

CENTER FOR AUTOMATION RESEARCH

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

(2)

CAR-TR-159
CS-TR-1573

F49620-85-K-0009
November 1985

PARALLEL PROCESSING OF REGION BOUNDARIES

Angela Y. Wu
Department of Mathematics, Statistics and Computer Science
American University
Washington, DC 20016

S.K. Bhaskar
Azriel Rosenfeld
Center for Automation Research
University of Maryland
College Park, MD 20742

ABSTRACT

A region may be represented by specifying the curves that bound it. When p processors are available, typical operations on regions so represented can be performed much faster than using a single processor. This paper presents parallel algorithms to determine properties of regions, such as area; to perform operations on regions, such as union and intersection; to determine if a point lies inside a region; and to determine whether a given digital curve could be the boundary of a region. Some of the algorithms involve sorting, the time complexity of which depends on the particular model of parallel computation used.

The support of the U.S. Air Force Office of Scientific Research under Contract F49620-85-K-0009 is gratefully acknowledged, as is the help of Sandra German in preparing this paper.

1. Introduction

A region in a digital image is determined by specifying the interior of the region or its boundary. The interior (or exterior) of a region can be represented by various schemes such as run length codes, medial axis transforms, unions of rectangles, quadtrees, etc. The boundary of a region may be represented using chain codes or crack codes. For simple regions, these are usually compact representations and allow faster processing. See [1] for a detailed discussion.

Because of the importance of regions in digital image processing, and because of the wide availability of multiprocessor architectures, it is of interest to inquire if the availability of p processors results in a p-fold speed-up in the performance of operations on compact representations of regions. In this paper, we will consider this question for the case of the boundary representations.

Chain codes and crack codes are two ways to represent border, in which a starting point and a sequence of moves around the border are specified. The chain coding scheme moves along a sequence of border points belonging to the region. The 8-neighbor chain code goes from a point to one of its eight neighbors in directions that are multiples of 45°. The 4-neighbor chain code goes from a point to one of its four neighbors in directions that are multiples of 90°. In the crack code representation, the moves are along the cracks between the pixels belonging to the region and the adjacent pixels outside the region. Each move is either left, right, up or down. See Figure 1.

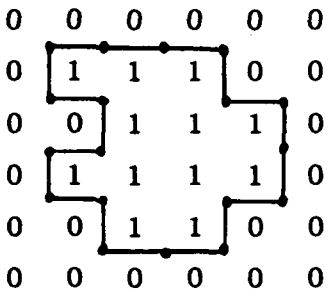


Figure 1: Boundary representations of a region, starting from its upper left corner

Crack code: 000303323221210121
4-neighbor chain code: 00303232120112
8-neighbor chain code: 007654313
Starting point is the northwest corner.

Key: In crack code and 4-neighbor chain code, $i = 90i\,^\circ$; in 8-neighbor chain code, $i = 45i\,^\circ$. All angles are measured counterclockwise from the positive x-axis.

From now on we consider only crack code, and we use the letters N, S, E, W to represent the move segments. Most of the results in the paper also apply to 4-neighbor chain code, with simple modifications.

To define the interior and exterior of a closed curve, we use the implicit "direction" associated with every crack code fragment, which is the direction in which a move is made. We adopt the convention that the interior of a closed curve always lies to the right of the curve when moving along the direction implied by the boundary code. With this convention, we may also represent holes in the regions. A hole is represented by the crack code of its boundary, but since a hole boundary and an outer boundary enclose areas on opposite sides of the boundary, a hole boundary must have a "direction sense" opposite to that of the outer boundary. We use the convention that the sense of an outer boundary

is specified by a boundary code in the clockwise direction while that of a hole boundary is specified by a boundary code in the anticlockwise direction. See Figure 2.

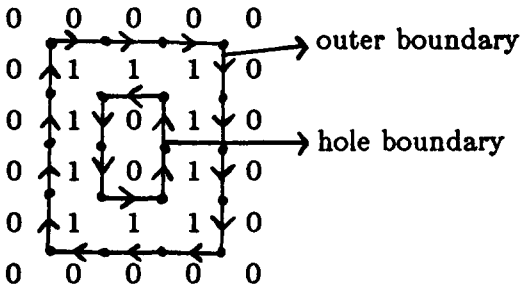


Figure 2

The purpose of this paper is to present a variety of algorithms which involve the crack code representation and which may be executed on suitable multiprocessor networks.

We assume that we have only a fixed number, p, of processors. In general $p \ll$ the number of boundary points. The parallel model of computation we use is a synchronous model. Each of the p processors has its own memory and its own address. At each time step, a processor can send out a message specifying the receiver's address; selective broadcasting is also allowed. At each time step, a processor can receive only one message. ZMOB [2] is an example of this model. This model of computation is strictly weaker than the shared memory models, which are currently unrealizable in practice. Therefore all the algorithms in this paper can be executed using any of the shared memory models. In fact, almost all the computations in our algorithms can be performed efficiently in a fixed interconnection tree network with 2-way communications along the links. The

only exception is a sorting step which enables us to find duplicates. A tree network does not perform this step very efficiently. Our algorithms can be executed efficiently on any network which has small (say log p) diameter and can perform sorting rapidly. If $S(n, p)$ denotes the time required to sort n numbers using p processors, we can express the time required by our algorithms in terms of $S(n, p)$. $S(n, p)$ will depend on the model of parallel computation that is used.

2. Boundary curves

A starting point and a sequence of (4-neighbor) moves can be regarded as a curve. The curve cannot be the boundary of a region unless it is closed and does not cross itself. In this section we present algorithms to decide if a given curve can be a region boundary.

First we describe an algorithm which obtains the absolute coordinates (with respect to the starting point) of all points along the curve. Such a step seems to be essential in most of the computations to be described, because the chain coding and crack coding schemes are "inherently sequential" and obtaining the absolute coordinates of every point allows parallel processing of the various segments of the code.

Let n be the number of points in the code of some curve. For simplicity let $2p+1$ be the number of processors, and assume p divides n. We assume that the $2p+1$ processors can be connected together as a binary tree with p processors as the leaves. These p processors each have $\frac{n}{p}$ moves of the boundary code.

Algorithm Boundary Coordinates:

1. For each of the p processors at the leaves do:

   Sequentially compute the relative coordinates (a, b) of the last point contained in that processor with respect to the first point (assumed to have coordinates (0,0)).

2. For $i := 0$ to $\log_2 p - 1$ do:

   Each processor p at level i gets the coordinates (a, b) and (c, d) stored by p's leftson and rightson respectively. p stores (a, b) in a field COUNT and $(a + c, b + d)$ as its coordinates.

3. For $i := 0$ to $\log_2 p - 1$ do:

   Each processor p at level i passes the value it received from its father to its own leftson. p adds its COUNT field to the value it received from its father and passes the sum to its rightson. The root, at level 0, passes its COUNT field to its rightson and the value (0,0) to its leftson.

4. Each of the p processors at the leaves now sequentially computes the absolute coordinates of all points stored within it, using the absolute coordinates of the first point (this is the value the processor has at the completion of step 3).

The first and last steps each take time $O\left(\frac{n}{p}\right)$. The two loops in steps 2,3 take time $O(\log p)$ each. Thus the overall complexity is $O\left(\frac{n}{p} + \log p\right)$, where n is the number of points in the boundary and $2p+1$ is the number of processors.

It is clear that at any given time only the processors at one level are active. Thus it is possible to maintain just p processors at the leaves, but to connect them in such a way as to simulate the tree connections. Such a scheme, while reducing the number of processors in half, leads to a logarithmic increase in the maximum degree of the interconnection network. In an addressable scheme, such as ZMOB, or in a shared memory model, no such penalty is incurred. If we are careful to distribute the boundary codes according to an inorder traversal of the binary tree with p processors, using the same principles as in the algorithm, the above algorithm can be modified in the obvious way without adding more links to the tree links.

To decide if a given curve is closed or not, we can use the first two steps of the above algorithm. When the root node of the tree obtains the coordinates of the last point in the curve with respect to the first point, the curve is closed iff it is the same as the starting point. This can be done in time $O\left(\frac{n}{p} + \log p\right)$.

Next, we show how to decide if a given chain code crosses itself. First, the coordinates of all the points on the curve are obtained as above. Now, we need to examine these coordinates for repeated values. One way to identify the duplicate points is to sort the coordinates and examine the sorted list for duplicate entries.

In $O\left(\frac{n}{p}\right)$ time we can decide whether this sorted list contains a repeated entry. If this is not the case the curve does not either touch or cross itself. If

repeated entries are found, then examine the directions of the associated segments. If the directions of the two segments incident on one point are E, E (or W, W) and those of the other are N, N (or S, S) then the curve crosses itself. Otherwise it touches itself. Hence the total time complexity is $O\left(\frac{n}{p} + \log p\right) + S(n, p) + o\left(\frac{n}{p}\right)$ where $S(n, p)$ is the time needed to sort n numbers with p processors. Using the shared memory model, $S(n, p) = O\left(\frac{n \log n}{p} + 2 \log p \log \frac{n \log p}{p}\right)$ [3]. We can determine if a crack code can be the boundary of a region since a boundary (whether outer or hole) must be closed and must not cross itself.

3. Region properties

In this section, we will assume that the crack code being considered always represents the boundary of a region with the direction conventions assumed in the Introduction. We present algorithms to solve the following problems:

(1) Given a boundary code, does it represent a hole or an outer boundary? Determine the location of a "corner", say the point with the least (y, x) key value, by simple comparisons in time $O\left(\frac{n}{p} + \log p\right)$. Then, observe the directions of the two segments incident with that point. If the direction is clockwise then the curve represents an outer boundary, otherwise it represents a hole.

(2) Given a closed curve and a point we can determine the location of the point relative to the curve as follows:

a) Each processor is given the coordinates of the test point. If broadcasting is available this takes constant time. Otherwise, in a tree connection model, it takes time $O(\log p)$. Next, determine the coordinates of all points along the curve in time $O\left(\frac{n}{p} + \log p\right)$.

b) Each processor determines in time $O(n/p)$ if the test point coincides with any of the segments of the boundary contained in it. If this is the case, the point is on the boundary. Otherwise each processor does the following:

Determine which of the points has the greatest y-coordinate less than the y-coordinate of the test point, and such that:

(i) It has the same x-coordinate as the test point or

(ii) The x-coordinate of the test point falls between the x-coordinates of two adjacent points of the boundary. A processor needs to give the coordinate of its last boundary point to its next neighbor.

In either of these cases, the y-coordinate(s) of the point(s) are returned. If no such points are found, a null value is returned.

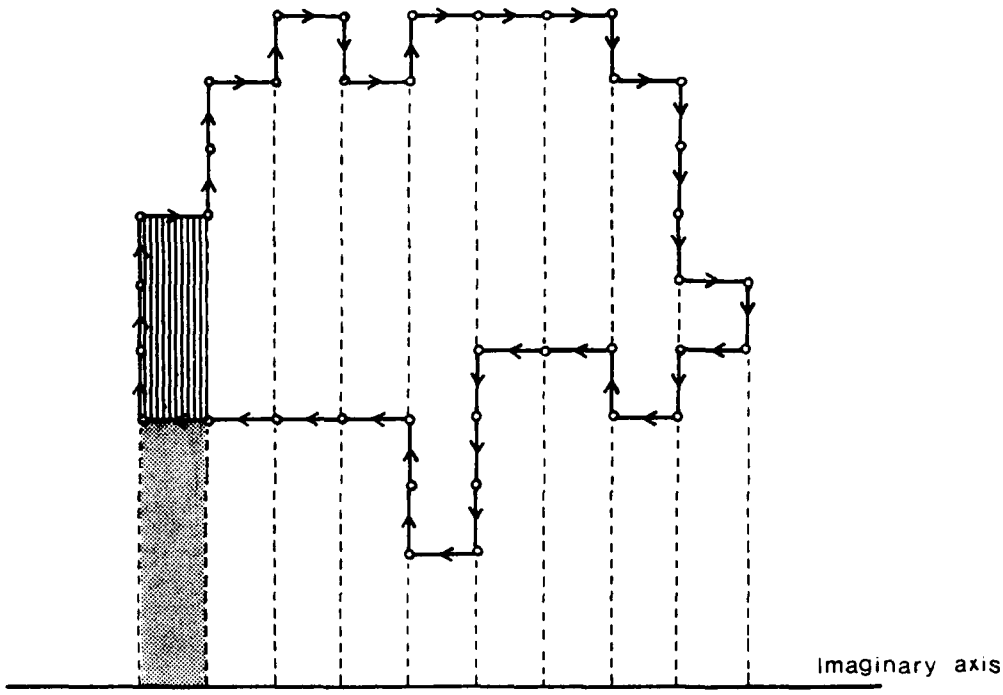
c) Using the tree connections, determine the largest y-coordinate, among all those passed up, which is less than the y-coordinate of the test point.

If no such y-coordinate exists, then the point is outside the curve. Otherwise, we observe the directions of the segments having that y-coordinate. If the test point lies to the right of the segments, it is an interior point, else it is an exterior point. This step only takes constant time.

(3) The boundary code of the complement of the area enclosed by a given set of boundary codes may be obtained as follows:

We assume that the given boundary codes are themselves properly enclosed in a frame, which is itself chain coded as an outer boundary. The complement can be obtained by reversing the direction of all segments in the given set of codes in time $O(n/p)$.

(4) Given a boundary, to determine the area enclosed by the curve we first obtain the absolute coordinates of all points. Then for each segment in the boundary, the following computation is performed. If the segment is vertical (i.e., has direction N or S), it is ignored. Otherwise, if the segment has direction E (W) and its y-coordinate is y, the processor containing the segment adds (subtracts) y to an aggregate. All such computations can be completed in time $O(n/p)$. The aggregates in the p processors can then be summed together in time $O(\log p)$, giving an overall time complexity of $O\left(\log p + \frac{n}{p}\right)$. The computation is equivalent to adding each of the areas shown hatched vertically, and subtracting each area hatched horizontally.



4. Operations on pairs of regions

In this section we consider tasks involving two crack codes, both of which represent closed curves.

(1) Relative position of two regions:

Given two boundary codes of lengths m and n, and the absolute coordinates of their origins, we can determine if the regions (i.e., the curves) intersect, are touching (i.e., the curves are tangential), or are disjoint, and also if one region lies completely within the other.

We first obtain the absolute coordinates of all points in both curves in time $O\left(\frac{m+n}{p} + \log p\right)$. The points appearing on both boundaries are

found as was done in the case of just one curve, except that when we examine two repeated entries, we consider only points not both belonging to the same curve. This can be done by sorting the $m + n$ points on the key (x, y). If the two curves are disjoint we can determine if one lies inside the other by picking an arbitrary point of one curve and solving the problem of determining if this point lies inside the other curve, and if not, we try the same problem with the curves exchanged.

(2) Given two boundary codes, to determine the boundary codes of the union and intersection of the areas enclosed by them we have basically two cases to consider: Either the curves are disjoint, or they share points in common. This can be determined as discussed earlier.

Assume first that the curves are disjoint. Then, one may either lie inside the other, or not. If one does lie inside the other:

a) If both curves are outer boundaries: the intersection is the inner one, while the union is the outer one.

b) If both are holes: the intersection is the outer one, while the union is the inner one.

c) If one is a hole and the other an outer boundary: if the inner one is the hole boundary nothing need be done for the intersection. Otherwise the intersection is empty. If the inner one is the hole boundary, the union is the hole boundary, otherwise nothing need be done for the union.

We now consider the cases when the curves share common points.

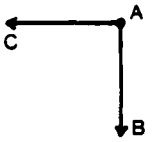
First, we determine all points common to both curves by the procedures described previously. Each such point can find the address of the processor(s) it appears in. Each processor containing a shared point now executes the following code:

a) If the directions to the next point in the two curves are opposite, then ignore the point.

b) If the two curves coincide at that point and go to the same next point (i.e., the next move directions from that point are the same in both curves), then nothing need be done.

c) If the next moves are different in the two curves, then we need to determine the direction of the boundary code from this point. We take either the intersection or the union of the regions enclosed by the segments and pick the appropriate one. Three basic cases arise:

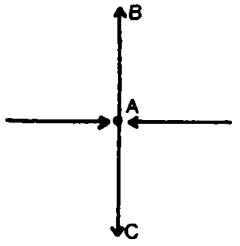
Case (1): If A is a common point, and in one curve, the next move from A is W, while in the other it is S: the region enclosed by the curve AC belongs to includes the area above AC, while that of AB includes the area to the left of AB. The intersection of the regions is the one included by AC while the union is the one included by AB. Thus, C is the next point if the intersection was desired, while B is the next point if the union was desired.



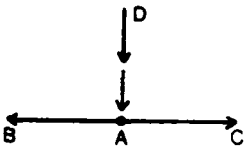
Once the successor is picked, a flag is put at the intersection

point to indicate that the resulting curve has a successor in another processor, the successor of this point is also stored.

Case (2): Here, the two curves abut at A. In this case both B and C are successors of A on the two curves. Analyzing the areas belonging to the regions, we find that the processor containing A on curve 1 needs to make A's successor be A's successor on curve 2 and the processor containing A on curve 2 needs to make A's successor be A's successor on curve 1 for both the union and intersection operations. This is accomplished by setting flags and address as in case (1).



Case (3): In this case the curves abut along AD. If the intersection is desired, A's successor is B while if the union is desired, A's successor is C.



Once the successors are picked, a sequential tour through the links, starting from each intersection point, gives the boundary code of the union or intersection.

Once the common points are obtained in time $O\left(\frac{n}{p} + \log p + S(n, p)\right)$, the links which determine the union or intersection can be found in time $O\left(\frac{n}{p}\right)$.

5. Concluding remarks

Chain codes and crack codes are compact representation for many types of regions, in particular, for simply connected regions. In this paper, we have shown that using p processors, each processor can handle a piece of the boundary and then combine the results. This can be done quite efficiently as long as the processors can be interconnected so that the p processors are not at more than distance $\log p$ apart—for example, as in a tree network. In some of the algorithms, we need to find the intersection points, and the sorting speed becomes an important factor in the time complexity.

From the boundary code representation, it is easy to convert to the run length code, which is a compact representation of the interior of a curve. Given the boundary code, we obtain the coordinates of all the points on the boundary and sort them on the key (y, x). Then, the number of 1's at a given y-level is just the difference between the greatest and least x's having that particular y in the sorted list. If the region contains holes, then the same type of difference gives the length of the hole at that y-level.

Another kind of region representation is based on specifying the vertices of a boundary polygon, where the edges are not constrained to be rectilinear. If the intersection points between two polygonal regions were to be determined by some other method, we can determine the boundary codes of the union and intersection by the same methods described here.

References

[1] A. Rosenfeld and A.C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.

[2] C. Rieger, J. Bane and R. Trigg, "ZMOB: A highly parallel multiprocessor", CS–TR–911, 1980.

[3] I. Valiant, "Parallelism in comparison problems", *SIAM J. on Computing* 4, 1975, 348–355.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS N/A
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) CAR-TR-159 CS-TR-1573	5. MONITORING ORGANIZATION REPORT NUMBER(S) AFOSR-TR- 85-1216 N/A
6a. NAME OF PERFORMING ORGANIZATION University of Maryland 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION Air Force Office of Scientific Research
6c. ADDRESS (City, State and ZIP Code) Center for Automation Research College Park, MD 20742	7b. ADDRESS (City, State and ZIP Code) Bolling Air Force BAse Washington, DC 20332
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER F49620-85-K-0009
8c. ADDRESS (City, State and ZIP Code)	10. SOURCE OF FUNDING NOS. PROGRAM ELEMENT NO.: 61102F; PROJECT NO.: 2304; TASK NO.: A7; WORK UNIT NO:
11. TITLE (Include Security Classification) Parallel Processing of Region Boundaries	
12. PERSONAL AUTHOR(S) Angela Y. Wu, S.K. Bhaskar, and Azriel Rosenfeld	
13a. TYPE Technical 13b. TIME COVERED FROM _____ TO N/A	14. DATE OF REPORT (Yr., Mo., Day) November 1985 15. PAGE COUNT 17
16. SUPPLEMENTARY NOTATION	
17. COSATI CODES FIELD / GROUP / SUB. GR.	18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A region may be represented by specifying the curves that bound it. When p processors are available, typical operations on regions so represented can be performed much faster than using a single processor. This paper presents parallel algorithms to determine properties of regions, such as area; to perform operations on regions, such as union and intersection; to determine if a point lies inside a region; and to determine whether a given digital curve could be the boundary of a region. Some of the algorithms involve sorting, the time complexity of which depends on the particular model of parallel computation used.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. ☐ DTIC USERS ☐	21. ABSTRACT SECURITY CLASSIFICATION UNCLASSIFIED
22a. NAME OF RESPONSIBLE INDIVIDUAL Dr. Buchal	22b. TELEPHONE NUMBER (Include Area Code) 767-4937 22c. OFFICE SYMBOL NM

DD FORM 1473, 83 APR EDITION OF 1 JAN 73 IS OBSOLETE.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

END

FILMED

3 -86

DTIC