SUITABILITY OF TOPOLOGICAL DATA STRUCTURES FOR DATA PARALLEL OPERATIONS IN COMPUTER CARTOGRAPHY

Bin Li
Department of Geography
University of Miami
Coral Gables, FL 33124-2060
Internet: binli@umiami.miami.edu

ABSTRACT

High performance computing is rapidly evolving towards parallel processing. Among various issues in applying parallel processing to computer cartography, a major concern is the compatibility between parallel computational models and existing cartographic data structures as well as algorithms. This paper assesses the suitability of topological data structures for Data Parallel Processing in computer cartography. By mapping selected cartographic operations to the data parallel model, we found that topological data structures, with minor extensions, can support data parallel operations. The topological information stored in the cartographic database can be used as index vectors for segmented-scan and permutation. To demonstrate this approach, the paper describes data parallel implementations of three cartographic operations are described, including line generalization, point-in-polygon search, and connectivity matrix construction.

INTRODUCTION

The motivation to conduct this research is three-fold. First, recent studies have shown the great potential of data parallel processing in computer cartography and GIS (Mower, 1992, 1993; Mills et al., 1992a, 1992b; Li, 1993). While most studies have been conducted in the raster domain, data parallel processing in the vector space has received less attention. Due to their spatial irregularity, cartographic operations in the vector domain are more difficult to adopt to the data parallel computational model. To perform data parallel processing, the vector space must be transformed to a regular one through specific decompositions such as the *uniform grid* (Franklin et al., 1989; Fang and Piegl, 1993). Although the *uniform grid* is a very efficient data structure for many vector operations, one of its drawbacks is the overhead to convert the data from and to their original structures. The conversion is necessary in the real world situations because few software packages for GIS and computer cartography use the *uniform grid*. It is therefore worthwhile to study the suitability of existing vector data structures for data parallel processing. Topological data structures are selected in this project because they are the most commonly used data structures in GIS software packages.

Second, large scale applications of the parallel computing technology require "enabling technologies" that are based on existing hardware and software environment and allow flexible evolutions. Set aside other technical and economic conditions for using parallel computing, software developers would not adopt strategies that require fundamental changes in existing data structures. It is preferable to have solutions that are less costly but can greatly increase processing capacity and functionality. One of such solutions is to develop add-on software that enables the user to run large GIS applications on parallel computers. Such add-on software must provide interface between the core modules and the new parallel programs. The interface will be more efficient if the existing data structure is compatible with the parallel computational model.

Third, it is conceptually challenging to examine how topological representations of geographic space may facilitate data parallel processing in cartography. Understanding such relations may help developing new insights to solving cartographic problems.

This paper reports findings on three popular operations in computer cartography and GIS. They are line simplification, point-in-polygon search, and connectivity matrix construction. The paper describes how data parallel procedures can be implemented with topological data structure. It emphasizes on whether the selected cartographic algorithms can be expressed with data parallel operations without significant alternation on the original data structure.

The topological data structure used in the paper is described in ESRI's GIS training book (ESRI, 1990). Some good references on data parallel processing include the article by Hillis et al. (1986), the book by Blelloch (1991), the FORTRAN 90 Handbook by Admas et al. (1992), and a number of programming manuals from Thinking Machines Corporation (1991a, 1991b).

LINE SIMPLIFICATION

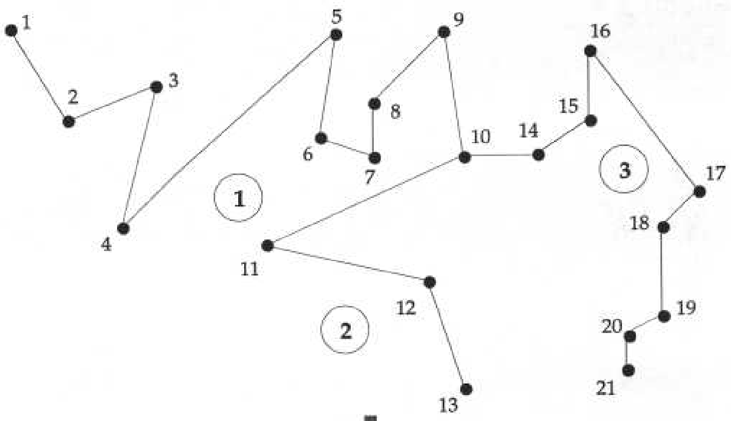
Line simplification is an important operation in cartographic generalization. A commonly used procedure is the Douglas-Peucker algorithm (Douglas and Peucker, 1973). Mower (1992) presented strategies to implement the algorithm for the data parallel (S-LINE) and the message passing (M-LINE) computational model. However, Mower's parallel procedures seem to apply to only a single line, which may not be applicable to a vector coverage that has more than one line. This section describes a data parallel procedure that executes the Douglas-Peucker algorithm on all the lines simultaneously.

The basic strategy is to store the coordinates of all the points on several long vectors, with two start_bit vectors defining the boundaries between line segments. *Prefix scan* can then be used to find the significant points in all segments. The process iterates until all points are either retained or eliminated.

The operation requires eight vectors:

- vector X and Y for the original x, y coordinates,
- vector Xs, Ys, Xe, and Ye for the start and end point coordinates,
- vector DIST to store the distance from each point to the line that links the two end points in each line segment,
- vector FLAG to record the status of each point on the line.

These vectors are segmented with two start-bit vectors, **S_up** and **S_down**. They specify the boundaries of line segments from the upward (from left to right) or the downward (from right to left) direction (Figure 1). Because new significant points are generated in each iteration, the length of these vectors are allocated dynamically.



ARC #	X, Y coordinates
1	(x1, y1) ... (x10, y10)
2	(x10, y10) ... (x13, y13)
3	(x10, y10) ... (x21, y21)

X

1	2	3	4	5	6	7	8	9	10	10	11	12	13	10	14	15	•••	20	21

S_up

0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	•••	0	0

S_down

0	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	•••	0	0

Figure 1. Mapping a line coverage to segmented vectors defined by two start-bits. Vector **X** stores the x coordinates of all the points. The numbers in **X** are the IDs of the points. Other vectors, including **Y, Xs, Ys, Xe, Ye, DIST,** and **FLAG**, have the same structures as **X**.

(1) Construct vector **X, Y, S_up,** and **S_down** from the arc-coordinate list. Generating vector **X** and **Y** requires a simple assignment but the start-bit vectors need to be constructed with parallel permutation. The parallel indices for the permutation are slight modifications of the running-totals of the vector "number-of-points" obtained from the arc-coordinate list. These

indices direct the assignment of scalar 1 to the start-bit vector **S_up** and **S_down** (Figure 2).

Number of Points	10	4	9
Running_total (up)	10	14	23
Running_total(down)	23	13	9

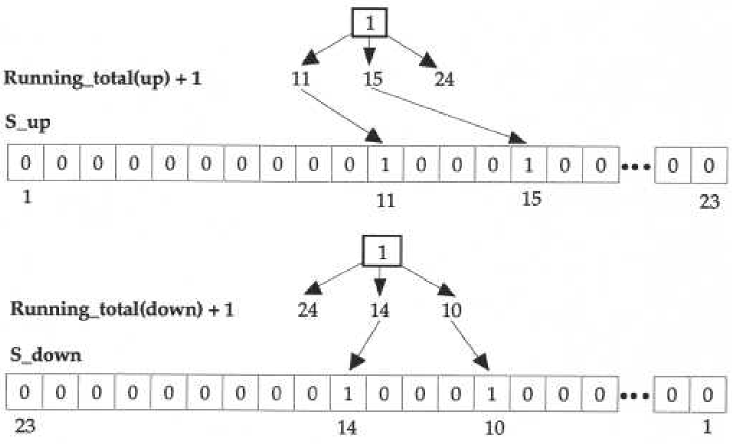


Figure 2. Constructing the start-bits from the "number-of-points" vector. Because *prefix-scan* with start-bit is direction dependent, two start-bit vectors are needed to define line segments.

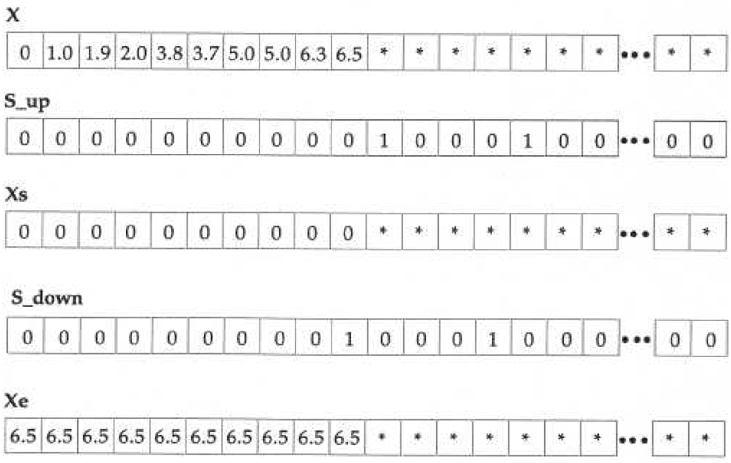


Figure 3. Illustration of the copy-scan operation. Only line segment 1 is shown here. Other line segments are marked with "*". Note that the x coordinate of the start point, 0, is distributed to all points in segment 1 of **Xs**; and the end point coordinate, 6.5, is copied to all locations in segment 1 of **Xe**.

(2) Assign 1 to positions in vector **FLAG** that correspond to the start and end points. By default, the start and the end points are significant and retained.

(3) Use *copy-scan* to distribute start/end point coordinates from vector **X** and **Y** to corresponding line segments in vector **Xs, Ys, Xe, Ye** (Figure 3).

(4) Calculate the distances from each point to the line that links the two end points of a corresponding line segment. This step involves element-wise operations among vector **X, Y, Xs, Ys, Xe,** and **Ye**. The results are stored in vector **DIST**.

(5) Use *maximum-scan* to find the longest distance in each segment in **DIST**. Select positions that are greater than the user-specified tolerance, record them as significant points in vector **FLAG**.

(6) Calculate the new lengthes of the vectors based on the number of new significant points found. Since a new significant point becomes a start point of one line segment as well as an end point of another, for each new significant point, one position must be added to the vectors. The calculation can be done with the selection procedure in step (5).

(7) Permute the original **X, Y** coordinates to the new vectors (Figure 4).

(8) Update the start-bit vectors so that locations corresponding to the new significant points are included as start-bits (Figure 4).

(9) Repeat (3) to (8) until all entries in vector **FLAG** become either 1 or 0. Retain points that have flag value 1.

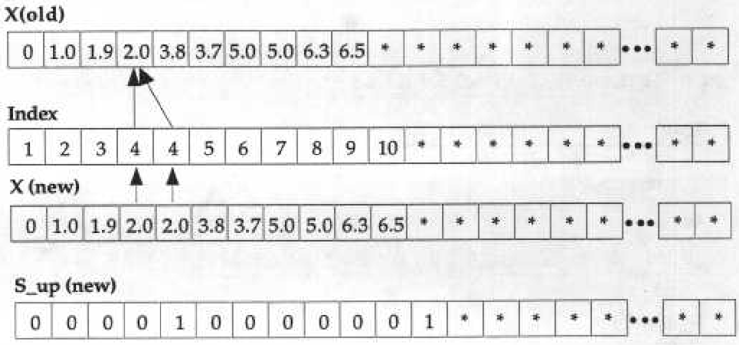
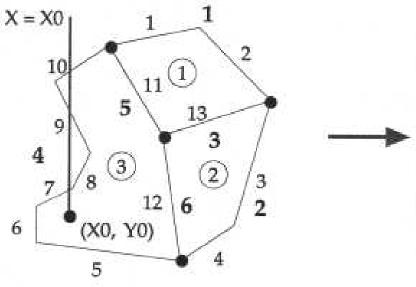


Figure 4. New vectors are created in each iteration. Vector **Index** directs **X(new)** to obtain coordinates in corresponding locations in **X(old)**. The start-bit vector is also updated. Note that the new vectors may have different length from the previous ones because line segments split at the new significant points.

POINT-IN-POLYGON SEARCH

Point-in-polygon search can be accomplished by the plumb-line algorithm—the search point (X0, Y0) is in polygon P if the number of intersects between the plumb-line X = X0 and P, and above the search point, is odd (Monmonier, 1982). Since intersects are calculated with each link (a line segment defined by two points), it is necessary to establish the arc-link topology (Figure 5).



ARC#	Link
1	1, 2
2	3, 4
3	13
4	5, 6, 7, 8, 9, 10
5	11
6	12

Figure 5. The polygon, the search point, and the arc-link topology.

Once the arc-link topology is built, the linkages between the links and the polygons also are established. We can first identify links that intersect with the plumb-line above the search point. Then with the topology vectors as parallel indices, the number of intersects can be permuted to arcs then to polygons.

(1) Find intersects between the links and the plumb-line. The following parallel structure is used to accommodate the calculations:

```
shape [N]vector;
struct segment {
            int    id;
            float  x1, y1;
            float  x2, y2;
            int    flag;
};
struct segment:vector     link;
```

illustrated as:

```
LINK        1  2  3  4  5  6  7  8  9  10  11  12  13
            X1 ...
            Y1 ...
            X2 ...
            Y2 ...
            flag ...
```

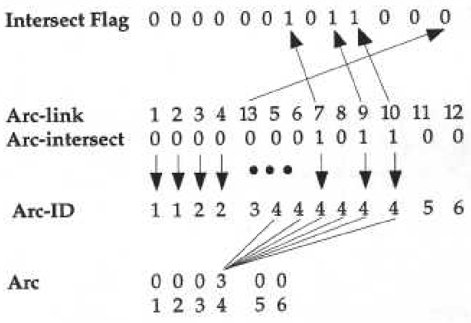
Each element in **LINK** stores the link ID, the coordinates of two endpoints, and a flag to indicate whether the link intersects with the plumb-line. To check intersects, the x, y coordinates of the search point are broadcasted to all locations in **LINK** and the calculations are performed simultaneously. Links that intersect with the plumb-line above the search point have flag value 1, others 0. For the example in figure 5, the intersect flags turn out as follows:

0 0 0 0 0 0 1 0 1 1 0 0 0

(2) Find the number of intersects with each arc (Figure 5).

• Use vector **Arc-link** as the parallel index, get the intersect flags to vector **Arc-intersect** so that the intersect information could be associated with each arc.

• Use vector **Arc-ID** as the parallel index, combine the number of intersects to vector **Arc**.



Intersect Flag 0 0 0 0 0 0 1 0 1 1 0 0 0
Arc-link 1 2 3 4 13 5 6 7 8 9 10 11 12
Arc-intersect 0 0 0 0 0 0 0 1 0 1 1 0 0
Arc-ID 1 1 2 2 3 4 4 4 4 4 4 5 6
Arc 0 0 0 3 0 0
1 2 3 4 5 6

Figure 6. Using parallel communications to find the number of intersects with each arc.

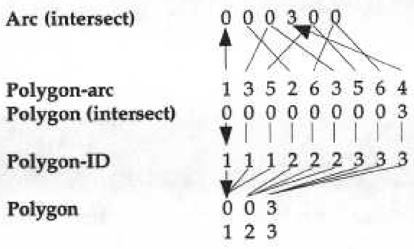
(3) Find the number of intersects with each polygon. Similar to the arc-link topology, the polygon-arc relation is mapped to vector **Polygon-arc** and **Polygon-ID** (Figure 7). Then the same procedures in step 2 are used to combine the total number of intersects with each polygon (Figure 8):

Polygon	**Arc**
1	1, 3, 5
2	2, 6, 3
3	5, 6, 4

$\longrightarrow$

```
Polygon-arc   1 3 5 2 6 3 5 6 4
Polygon-ID    1 1 1 2 2 2 3 3 3
```

Figure 7. Deriving parallel index vectors from the polygon-arc topology.



Arc (intersect) 0 0 0 3 0 0
Polygon-arc 1 3 5 2 6 3 5 6 4
Polygon (intersect) 0 0 0 0 0 0 0 0 3
Polygon-ID 1 1 1 2 2 2 3 3 3
Polygon 0 0 3
1 2 3

Figure 8. Using parallel communication to find the number of intersections with each polygon.

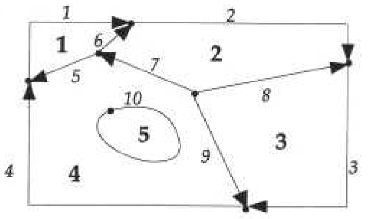
• Use vector **Polygon-arc** as the parallel index, identify which arc intersects with the plumb-line.

• Use vector **Polygon-id** as the parallel index, combine the number of intersects to vector **Polygon**.

(4) Select from vector **Polygon** the location that has the odd number of intersects.

CONNECTIVITY MATRIX CONSTRUCTION

Connectivity matrix is commonly used in spatial statistics. It is a numerical representation of spatial relations for one and two dimensional cartographic objects. With cartography evolving beyond its traditional scope of mapping geographic space and relations, spatial connectivity matrix becomes a necessary component for inferential cartographic analysis.



Arc #	Left	Right
1	0	1
2	0	2
3	0	3
4	0	4
5	4	1
6	1	2
7	4	2
8	2	3
9	3	4
10	4	5

(a)

	0	1	2	3	4	5
0	0	1	1	1	1	0
1	0	0	1	0	0	0
2	0	0	0	1	0	0
3	0	0	0	0	1	0
4	0	1	1	0	0	1
5	0	0	0	0	0	0

(b)

	0	1	2	3	4	5
0	0	1	1	1	1	0
1	1	0	1	0	1	0
2	1	1	0	1	1	0
3	1	0	1	0	1	0
4	1	1	1	1	0	1
5	0	0	0	0	1	0

(c)

Figure 9. Constructing a connectivity matrix from the left-right topology.

The following describes how the left-right topology can accommodate data parallel construction of the connectivity matrix for a polygon coverage. The procedure is straight-forward—the two polygons that share the same arc are connected. In addition, an arc cannot be shared by more than two polygons. Therefore, given vector LEFT and RIGHT, LEFT(i) and RIGHT(i) are neighboring polygons (Figure 9a). In other words, the LEFT and RIGHT vector actually serve as the indices for the two dimensional connectivity matrix. Using the FORALL construct in FORTRAN90, the relations between the connectivity matrix and the LEFT and RIGHT vector can be expressed as:

```
FORALL (i = 1:N)   C_mat(LEFT(i), RIGHT(i)) = 1
```

where **C_mat** is an N-by-N binary matrix, with 0 and 1 indicating the connectivity between two polygons (Figure 9b). A complete **C_mat** is obtained by combining with its transpose, i.e.,

C_mat = TRANSPOSE(C_mat) + C_mat

where TRANSPOSE is an intrinsic function in FORTRAN90 (Figure 9c).

SUMMARY

This preliminary study found that the topological data structure sufficiently supports data parallel implementation of three cartographic operations. First, the topologies stored in the vector cartographic database can be used as parallel indices to establish hierarchical relations among cartographic objects. For instance, using the arc-link topology as the parallel index, an arc vector can be easily constructed from the link vector. Similarly, a polygon vector can be built with the polygon-arc topology. The following C* statement express the hierarchical relations among cartographic objects:

```
ARC = [ARC_LINK] LINK;
POLYGON = [POLY_ARC]ARC;
```

Such linkages also channel information among cartographic objects from one level to another. The section on point-in-polygon search showed how the number of intersects can be accumulated from links to polygons.

Second, vectors that define segments for parallel prefix scan operations are also generated from topological information in the original database. Segmented scan makes it possible to execute accumulative operations simultaneously on all units of such cartographic objects as arc and polygon. The boundaries between cartographic objects are defined with two approaches, using the start-bit vector or the ID vector. Both are generated from the original topological information. With these vectors, many cartographic algorithms can be implemented with segmented *prefix-scan* which we used as the primary operation for line simplification.

Findings from this study should be applicable to data parallel implementations of other vector-oriented operations in analytical cartography, such as polygon overlay and network analysis. They also should be useful for assessing the technical feasibility of data parallel processing in the vector domain.

ACKNOWLEDGEMENT

The author wishes to thank the Northeast Parallel Architecture Center at Syracuse University for providing the computer hardware and technical supports.

REFERENCES

Adams, J., et al., 1992, FORTRAN 90 Handbook, Complete ANSI/ISO Reference, McGraw-Hill Book Company, New York.

Blelloch, G., 1991, Vector Models for Data-Parallel Computing, MIT Press, Cambridge, MA.

Douglas, D. H. and T. K. Peucker, 1973, Algorithms for the Reduction of the Number of Points Required to Represent a Digitized Line or its Caricature: The Canadian Cartographer, Vol. 10, No. 2, pp. 112-122.

ESRI, 1990, Understanding GIS, ESRI Inc., Redland, CA.

Fang, T. P., and L. Piegl, 1993, Delaunay Triangulation Using a Uniform Grid: IEEE Computer Graphics and Applications, Vol. 13, pp. 36-47.

Franklin, R., et al., 1989, Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines: Auto Carto 9, Proceedings, Ninth International Symposium on Computer-Assisted Cartography, Baltimore, MD, pp. 100-109.

Hillis, D., and G. L. Steele, Jr., 1986, Data Parallel Algorithms: Communications of ACM, Vol. 29, pp. 1170-1183.

Li, Bin, 1993, Opportunities and Challenges of Parallel Processing in Spatial Data Analysis: Initial Experiments with Data Parallel Map Analysis, Doctoral Dissertation, Department of Geography, Syracuse University, Syracuse, NY.

Mills, K., et al., 1992a, Implementing an Intervisibility Analysis Model on a Parallel Computing System: Computers & Geosciences, Vol. 18, pp. 1047-1054.

Mills, K., et al., 1992b, GORDIUS: A Data Parallel Algorithm for Spatial Data Conversion: SCCS-310, Syracuse University, Syracuse, NY.

Monmonier, M., 1982, Computer Assisted Cartography: Principles and Prospects, Prentice Hall, Englewood Cliffs, NJ.

Mower, J., 1992, Building a GIS for Parallel Computing Environment: Proceedings of the 5th International Symposium on Spatial Data Handling, Charleston, SC, pp. 219-229.

Mower, J., 1993, Automated Feature and Name Placement on Parallel Computers: Cartography and Geographic Information Systems, Vol. 20, No. 2, pp. 69-82.

Thinking Machines Corporation, 1991a, Programming in FORTRAN, Cambridge, MA.

Thinking Machines Corporation, 1991b, Programming in C*, Cambridge, MA.