Waugh T. C. and Hopkins S. (1992) "An Algorithm for Polygon Overlay Using Cooperative Parallel Processing" to appear in International Journal of GIS.

Building a GIS for Parallel Computing Environments

James E. Mower
Department of Geography and Planning and Laboratory for Geographic Information Systems and Remote Sensing
State University of New York at Albany
147 Social Sciences, Albany, NY 12222
jmower@itchy.geog.albany.edu

Abstract

An introduction to the development of parallel algorithms is given through the introduction of a new procedure for labeling drainage basins from grid cell DEMs and two new procedures for the automated simplification of cartographic linework. Modeling and efficiency issues with regard to particular parallel computing platforms are stressed.

Introduction

Parallel computers offer cartographers and GIS professionals an exciting environment for the development of very fast display, analysis, and storage procedures. Whereas current high-performance workstations perform up to approximately 50 million floating point operations per second (MFLOPS), current massively parallel computers such as the Thinking Machines CM-200 perform approximately 4000 MFLOPS, about 4 times the number of MFLOPS performed by a Cray Y-MP/832 (Dongarra 1991). Research accounts on parallel computing platforms are available now for testing and developing algorithms.

Several investigators have already developed parallel algorithms for specific GIS procedures. Mower (1989, 1992) has written parallel automated name placement procedures for the Connection Machine, a fine-grained, single instruction stream, multiple data stream (SIMD) computer. The author has shown that his parallel procedure requires 1/10 of the processing steps required to execute an equivalent serial algorithm. Given processors of equivalent computational power, the parallel algorithm will always perform better than the serial algorithm.

Franklin and others have written parallel procedures for the automated detection of line intersections in polygon overlays using uniform grid techniques (Franklin and others 1989, Franklin, Kankanhalli, and Narayanaswami 1989). The authors have shown that the detection of line intersections on a 16 processor parallel machine executes 10 times faster than on a comparable machine with one processor.

These and other studies, such as one performed by Mills (1991) for the solution of intervisibility problems, have produced a small but interesting body of parallel algorithms for solving problems in automated cartography and GIS. To promote the further application of parallel computing to geographic research, this paper will

1) discuss strategies for envisioning GIS procedures as parallel procedures,

2) demonstrate examples of such procedures through the introduction of new parallel algorithms for the automated simplification of vector linework and for the extraction of drainage networks from digital elevation models, and

3) suggest directions for future research in parallel computing applications for GIS and automated cartography.

The author will use the Connection Machine as a software development model, but will discuss the relative advantages of other hardware platforms for the solution of specific problems in automated cartography and GIS.

Parallel Computing Architectures

Modern parallel computers vary a great deal in their underlying processor and network architectures. It is useful to classify them on the size of their processor arrays, the source of the processor instruction stream, and on the topology of the computer's interprocessor communication network.

Generally, a parallel computer is considered to be coarse-grained when the number of processors is small (anywhere from 2 to several hundred) but the computing power of individual processors is large. Modern coarse-grained computers often use off-the-shelf processors such as the Sun Microsystems SPARC chip or utilize their own proprietary designs.

A fine-grain computer typically contains anywhere from several hundred to many thousands of smaller processors. The processors may be as small as a bit-serial units which operate on one bit of information per processing cycle.

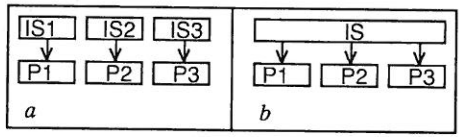

Table 1. Instruction streams in parallel computers. *a* represents a MIMD machine, *b* a SIMD machine.

In a parallel computing environment, two or more processors execute instructions simultaneously. In a single instruction stream, single data stream (SIMD) computer, a serial front-end computer broadcasts instructions to processing elements in the parallel machine. Each processor executes or ignores the instruction based upon the state of its local data. The CM2 is a SIMD computer that uses a Digital Equipment Corporation VAX or Sun Microsystems SPARCStation as a front-end computer. User programs are loaded and executed on the front-end; parallel code in the serial program is broadcasted to the CM2 processing elements.

In a multiple instruction stream, multiple data stream (MIMD) computer, each processor executes a unique instruction stream (Figure 1). The Encore Computer Corporation Multimax 320 is a MIMD machine that runs a parallel variant of the Unix operating system, Umax, to coordinate parallel requests for system resources. MIMD programs typically subdivide a computing problem into a set of procedures, some of which execute simultaneously as forks. If a procedure depends upon another executing procedure for its input, the programmer must ensure that the procedure not begin execution until its data becomes available. MIMD programming languages on the Multimax establish program synchronization through the use of semaphores and monitors.

Some parallel computers offer both MIMD and simulated SIMD computation modes. The Thinking Machines, Inc. CM5 computer controls up to 1024 Sun Microsystems SPARC processors directly in MIMD mode or by mapping a number of virtual processors onto physical processors to simulate SIMD computations.

Parallel processing machines differ on their support of interprocessor communication techniques. Processors in SIMD machines are sometimes wired together as arrays with dimensions greater than or equal to 1 such that each processor in the array is connected to its adjacent neighbors along each dimension. Scanning functions perform operations on the data associated with processors along one or more of the array dimensions, typically through the computation of a running value. Scanning functions often provide the fastest method of interprocessor communication within SIMD machines.

Sometimes a processor needs to receive information from or send information to a processor that is not one of its directly connected neighbors. Assuming that the number of processors is large enough to make the interconnection of all processors impossible or impractical, the machine must provide a communication path between arbitrary processors. Such communication paths are often supported by hypercube networks of dimension n, where n refers to the number of communication paths emanating from each processor or router node. Full interconnection often comes at the expense of communication speed, especially for messages that must pass over several hypercube dimensions.

SIMD computers provide a natural programming environment for models that require the application of local processing operations over grid cell data structures. On a SIMD computer, a grid cell data structure maps onto an array of processing elements in the same way that it would map onto an array of silicon memory addresses on a serial computer. The difference, of course, is that SIMD computers process data locally. Interprocessor communication is handled through assignment operators. On a serial computer, an assignment operator reads the contents of a memory location (an rvalue) and writes it into another memory location (an lvalue). On a SIMD computer, both rvalues and lvalues can represent processor addresses.

Designing Parallel Algorithms

Besides raw speed, parallel computers offer cartographic and GIS software engineers an opportunity to rethink the way that they design algorithms. A first attempt at parallel design is often a liberating and frustrating experience. It is satisfying to be able

to perform a repetitive computation on thousands of data elements simultaneously but frustrating to synchronize the communication of their results. Essentially, designers have to convince themselves that the computations really are being performed simultaneously.

To illustrate the opportunities and problems associated with the design of parallel algorithms, three sample algorithms, parallel variants of two common analytical procedures, will be introduced. The first algorithm, PDRAIN, estimates the locations of stream channels from digital elevation data, drawing from several existing serial algorithms. For a comprehensive review of drainage network extraction algorithms, see Lammers and Band (1990). The second procedure, S-LINE, modifies the Douglas line simplification algorithm to operate on digitized linework in SIMD mode (Douglas and Peucker 1973). The third procedure applies Douglas line simplification in MIMD mode.

Peucker and Douglas (1975) introduced a series of parallel procedures, for the extraction of drainage networks from DEMs using a binary encoding of the elevation matrix as a set of topographic features. O'Callaghan and Mark (1984) extended their work with the introduction of methods for modeling stream channel flows through the accumulation of simulated runoff from uphill neighbors. Marks, Dozier, and Frew (1984) developed a recursive procedure for the delineation of drainage basin boundaries.

PDRAIN modifies the Marks, Dozier, and Frew basin labeling procedure for parallel computing environments. Most notably, it replaces recursion with the distributed execution of procedures across grid cells.

PDRAIN performs a one-to-one mapping of gridded elevation values onto a two-dimensional array of physical or virtual processors. Each processor in the array performs local processing on its grid cell value and on its 8 surrounding neighbors. Processing occurs in SIMD mode; that is, every active processor simultaneously performs the same operation on its local data.

Due to sampling error, DEMs frequently contain closed depressions (pits) on the order of one or two meters in depth over flat terrain. Since pits do not often occur naturally at this resolution in fluvially-eroded terrain, PDRAIN conforms to the common practice among serial drainage extraction algorithms of "flooding" pits in preprocessing operations.

A SIMD Procedure for Labeling Drainage Basins (PDRAIN)

In the following procedure, several preprocessing and postprocessing operations are performed in serial mode, usually whenever performing input or output operations. Each step of the procedure is marked with a (P) or an (S) to indicate parallel or serial processing, respectively.

To Label Drainage Basins:

1(S). Allow the user to specify a grid cell coordinate pair corresponding to a drainage outlet.
2(S). Read the elevations, assigning each to a unique processor.
3(P). Smooth the elevations with a local averaging procedure.

4(P). Find the slope and exposure of each cell using local finite differentiation of the surface. Identify the remaining pits, sort them from lowest to highest elevation, and find the drainage direction for each cell.
5(P). Assign each pit and the drainage outlet unique labels. Starting at each pit and at the drainage outlet, propagate their labels to all cells that drain into them.
6(P). Find the lowest point on the edge of each basin and label it as the pour point. Set its slope to 0. If any elevations in the basin fall below the elevation of the pour point, raise them to the level of the pour point. Set their slopes to 0 and their exposures to undefined.
7(P). Relabel the drainage basin, this time redirecting the drainage of the pour points back toward the location of the propagating neighbor.
8(S). Store the drainage values.

PDRAIN, like other drainage network models, spends a large part of its time identifying and removing false pits. O'Callaghan and Mark note that the application of a local smoothing operator over gridded elevations often removes a large proportion of the total pits discovered in raw elevation matrices. This project applies their 8 cell smoothing filter over the raw elevation data before explicit pit identification and extraction operations were performed (Step 3). Each processor averages its elevation with those of its 8 neighbors to compute a smoothed value for its grid cell.

Step 4 performs a parallel calculation of slope and exposure values for each grid cell using partial derivatives of the surface given by Marks, Dozier, and Frew. The partial derivatives in x were computed as the difference in elevation of the adjacent row neighbors. The partial derivatives in y performed the same computation on adjacent column neighbors.

Any pits that were not removed by the smoothing operation are identified as grid cells whose 8-connected neighbors have greater elevations than themselves. For each of the 8 directions, every grid cell simultaneously compares its elevation to its neighbor's in that direction. If all of its neighbors are higher than itself, it labels itself as a pit with a unique identifier.

Before pits can be removed, they must be associated with a drainage basin (Figure 2). Each pit propagates its label to each of its uphill neighbors in parallel, making the uphill neighbor "active," and itself "inactive." Each of the active neighbors then propagates its label in parallel to any of its uphill neighbors that have not been visited yet. This process continues until no active neighbor finds another uphill neighbor (Step 5).

At this point, some of the drainage basins that have been discovered are likely to be parts of larger basins. To coalesce the partial basins into larger basins, Step 6 uses a methodology suggested by Marks, Dozier, and Frew. It conducts a search for the pour point, or point of minimum elevation on each drainage basin boundary, making all elevations below the pour point equal to the pour point's elevation. The slopes of the pour point and the modified grid cells are set to 0.

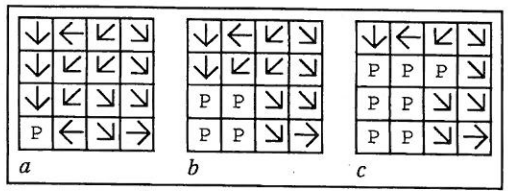

Figure 2. Propagating basin labels across a matrix of drainage directions. *a* represents drainage directions for each grid cell. P is a pit. In *b*, one propagation step has occurred and P's label has propagated uphill. In *c*, one more propagation step has occurred. P is unable to propagate its label into cells that face away from it.

Step 7 conducts a second and modified search for drainage basins, starting at the pour point of the lowest basin. With the activation of the lowest pour point, labeling spreads uphill. This time, however, any neighbor with 0 slope, regardless of its exposure, is considered to be uphill. The procedure changes the drainage direction of points of 0 slope to point at the direction from which the label propagated. If a pour point of a higher basin is on the boundary of the current basin, it will become part of the lower basin, and then propagate the lower basin's label into its former sub-basin.

Table 1 compares the number of steps required to perform elements of the PDRAIN procedure on a SIMD computer to the number of steps that would be required to perform an equivalent function on a serial computer.

Procedure	Operations	SIMD Steps	Serial Steps
Smooth Elevations	Get 8 Z values	8	$8n$
	Smooth values	1	$1n$
Slope and Exposure	Get 4 Z values	4	$4n$
	Find 2 partial derivs.	2	$2n$
	Find Slope & exposure	2	$2n$
Label Basins	Propagate labels	(maximum$\sqrt{n}$)	maximum n
Find Pour Points	Find basin borders	k	$j \times k$
	Find minimum Z value	k	$j \times k$
	Flood basins	1	$m \times k$
Label Basins Again	Propagate labels	(maximum$\sqrt{n}$)	maximum n

Table 1. Comparison of the number of steps required to run PDRAIN on a parallel SIMD computer to the number of equivalent steps required to implement a similar procedure on a serial computer. In the table j is the average number of cells per drainage basin, k represents the number of drainage basins, m is the average number of points in a basin below the pour point and n is the total number of cells in the grid.

PDRAIN performs a number of diverse functions that serve to illustrate some desirable qualities of SIMD programming environments. Table 1 shows that PDRAIN performs several grid cell operations (e.g. smoothing and calculations of slopes and exposures) independent of the number of grid cells when the number of cells is less than the

number of available physical processors. When it is higher, portions of the elevation grid must be swapped in and out of the processor array or each processor must become a platform for two or more virtual processors. Whenever virtual processors are mapped onto physical processors, a certain amount of overhead is incurred. Specifically, the physical processor must subdivide its computing cycles and its random access memory between the two virtual processors.

The author is currently developing an implementation of PDRAIN that accommodates 65,536 elevations as a 256 by 256 grid. Since the number of physical processors on the development platform (a CM2) is 16,384, 4 virtual processors must be mapped to every physical processor.

The successful implementation of a parallel computer depends a great deal upon the nature of its communication network. In many SIMD machines, the communication network offers more than a data path between processors; it also offers an environment for applying symbolic operations over all or part of the processor array. Reduction operators perform calculations over a set of parallel variables and "reduce" their contents to serial variables or assign them to other parallel variables. Scanning operators generally perform arithmetic functions (e.g. running sums and products) over a range of parallel variables.

PDRAIN makes heavy use of parallel reduction operators. To find the pour point of a drainage basin, PDRAIN first determines which cells are on its boundary. It then uses a minimum reduction operator to find the lowest elevation among those cells.

Many of the procedures in PDRAIN use bitwise reduction operators to test truth conditions over a range of parallel processors. For example, the basin labeling procedure must be able to tell when all of the cells in the basin have stopped propagating their labels. It asks each processor to return FALSE (0) if it has not propagated a label in the previous cycle. If any processor returns TRUE (1), the logical OR of all the returned values from all processors must be 1 and the loop continues.

PDRAIN replaces the Marks, Dozier, and Frew recursive drainage basin labeling procedure with a parallel propagation procedure. In the worst case, starting from a pit at the corner of the grid, basin labels will propagate away from that point at the rate of one cell along each axis per iteration. Assuming an $n \times n$ grid, the labeling procedure will require at the most n steps.

Parallel Simplification of Lines

The next procedure, S-LINE, performs Douglas line simplification using different interprocessor communication functions than those used by PDRAIN. Each vertex along the line is assigned to a unique processor. Complex line segments between arbitrary vertices are grouped into a communication structure known as a scan set, defined as a portion of the total number of processors along one dimension of the processor array. In S-LINE, scan sets represent complex line segments between significant vertices on the original line. For a complete description of the Douglas procedure, see Douglas and Peucker (1973).

A SIMD Procedure for Simplifying lines (S-LINE)

To simplify lines:

1(S) Assign each vertex to a unique processor and establish a user-defined tolerance.
2(P) Compute the equation of the straight line passing through the starting and ending points of the original line. Call it the base line.
3(P) For each vertex on the original line,
  3.1(P) Compute its perpendicular distance from the baseline.
4(P) If the distance from the baseline to the furthest vertex is greater than the tolerance, subdivide the line at that vertex, record it as significant, and repeat steps 2 through 4 on the two segments of the newly subdivided line.
5(S) Connect the significant vertices with arcs.

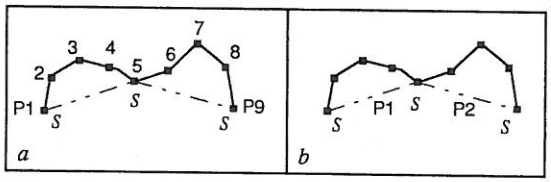

Figure 3. SIMD and MIMD views of the original line. In *a*, S-LINE assigns each vertex to a processor (P1-P9). Processors 1-5 are members of one scan set; processors 6-9 of another. In **b**, two processors, P1 and P2, represent the two original line segments subdivided at the central significant point *S*.

In S-LINE, each vertex is responsible for determining its own significance to the original line. By default, the two endpoints on the original line are flagged as significant. In step 2, significant vertices search for the next significant vertex along the line, proceeding from lower to higher processor addresses. On the first iteration, the first vertex will find the last on the line as significant and use its coordinates to calculate the equation of the line passing through both vertices. The equation is held in its local memory. Each vertex along the original line finds the first significant point below its address and uses the vertex's equation to calculate its distance from the baseline.

To find the furthest vertex from the baseline, the procedure divides the linear array of processors into scan sets. In S-LINE, a scan set represents a set of processors whose vertices fall between a pair of significant vertices. A maximum operator is passed over the scan set to find the vertex that is furthest from the baseline. If the distance is greater than the tolerance, the vertex is flagged as significant. The procedure is repeated over the two subdivisions on either side of the new significant point. The original line continues to be subdivided until no vertex is at a greater distance than the tolerance value from its baseline.

S-LINE performs a large number of interprocessor communication operations. Whenever a new significant vertex is discovered, each processor must redetermine its scan set by querying the processors below it for the nearest significant vertex. Once the

scan sets have been determined, the application of the maximum function over the scan sets must be performed.

Unfortunately, network communication operations on a parallel computer are often slow compared to other computational functions. Reductions in the use of interprocessor communication can lead to substantial reductions in execution speeds. A second procedure for simplifying lines on a MIMD machine will now be proposed. Unlike the SIMD procedure, the MIMD procedure requires little interprocessor communication. Instead of modeling processors as vertices, it models them as line segments. Figure 3 portrays the difference between the SIMD and MIMD views of the line.

A MIMD Procedure for Simplifying lines (M-LINE)

To simplify lines:

1(S) Read the vertices on the original line and store them in a serial array. Establish a user-defined tolerance.
2(S) Assign the original line to a processor. Flag the endpoints as significant.
3(P) Do the following steps:
  3.1(P) Calculate the equation of the baseline passing through the significant vertices.
  3.2(P) For each vertex between the significant vertices, calculate its distance from the baseline.
  3.3(P) Find the vertex that is furthest from the baseline and compare it to the tolerance for significance.
  3.4(P) If the vertex is significant, divide the line at the significant point and fork each segment to separate processors.
  until no vertex is at a greater distance from its baseline than the tolerance distance.

Table 2 compares the number of steps required to perform the major computational steps in S-LINE, M-LINE, and an equivalent serial procedure. Since each line segment can be simplified independent of the other segments, M-LINE does not require synchronization of the forked processes. Although the SIMD procedure requires the least number of time steps to perform the distance calculations, it requires the greatest number of communication operations.

Procedure	S-LINE Steps	M-LINE Steps	Serial Steps
Calculate Baseline Equation	$log_2 l+1$	$log_2 l+1$	l
Distance to Baseline (worst case)	$n/2$	appx. $\sum_{i=1}^{n} \frac{n}{i}$	$\sum_{i=1}^{((int)\log 2n)+1} n+2(i-1)$
Scan for Nearest Sig. Vertex	$log_2 l+1$	—	—

Table 2. Comparison of times for performing the major computational elements of S-LINE, M-LINE, and an equivalent serial procedure. Steps are based on a worst-case tolerance value of 0.

M-LINE preserves the recursive nature of the Douglas algorithm and its inherent simplicity of implementation better than S-LINE does. Given MIMD processors of equivalent power to serial processors and a forking mechanism that does not require more overhead than a recursive call, M-LINE will always perform better on a MIMD computer than a functionally equivalent procedure will perform on a serial computer. This is not necessarily true for S-LINE, however. If the network communication operations are slow and the individual processors are small compared to standard microcomputer or workstation processors, S-LINE may have little or no processing advantage over them. This illustrates the importance of selecting an appropriate parallel computing model for the implementation of a particular algorithm.

Modeling Other Geographic Processes on Parallel Computers

The solution of some geographic problems on a parallel computer require different network communication services than those described for the drainage network and line simplification problems. If each processor requires a communication path to any other processor, it may be necessary to use hypercube or other higher-level message routing schemes. The author's automated name placement procedure utilizes hypercube communications to negotiate competition for map space among point symbols and their labels. Although the hypercube network on the CM2 provides slower message handling than the grid network provides, it is especially relevant to the modeling of geographic processes based on network data structures such as TIN, DLG, TIGER, and others that are not well suited to modeling over a fixed grid. By assigning each node in a street network to a SIMD processor, a line topology can be built with arcs that represent communication links between their nodes.

Conclusion

This paper has intended to show the flexibility and power that parallel processing machines lend to the modeling of geographic processes and to the design of their associated data structures. As its cost decreases, parallel processing will find numerous applications among GIS end-users as well. Many opportunities exits to adapt our current processing techniques to these new platforms.

Acknowledgement

This work was conducted using the computational resources of the Northeast Parallel Architectures Center (NPAC) at Syracuse University.

References

Dongarra, J. J. 1991. "Performance of Various Computers Using Standard Linear Equations Software," technical report CS-89-85, Computer Science Department, University of Tennessee, Knoxville Tennessee.

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.

Franklin, W. R., M. Kankanhalli, and C. Narayanaswami 1989. "Efficient Primitive Geometric Operations on Large Databases", Proceedings, GIS National Conference 1989, Ottawa.

Franklin, W. R., C. Narayanaswami, M. Kankanhalli, D Sun, M. Zhou, P. YF. Wu. 1989. "Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines," Proceedings, Autocarto 9, Baltimore.

Lammers, R.B. and L.E. Band 1990. "Automating Object Representation of Drainage Basins," Computers and Geosciences, vol. 16, no. 6:787-810.

Marks, D, J. Dozier, and J. Frew 1984. "Automated Basin Delineation from Digital Elevation Data," Geo-Processing, vol. 2:299-311.

Mills, K., 1991. "A Pilot Study of Intervisibility Analysis on the Connection Machine," Parallel Computing News, vol. 4, no. 3, pp. 4-7.

Mower, J. E., 1989. The Selection, Implementation, and Evaluation of Heuristics for Automated Cartographic Name Placement. Ph.D. dissertation, State University of New York at Buffalo.

Mower, J.E., 1992. "Automated Name Placement on Parallel Computers," manuscript submitted for review.

O'Callaghan, J.F. and D.M. Mark 1984. "The Extraction of Drainage Networks from Digital Elevation Data," Computer Vision, Graphics, and Image Processing, vol. 28:323-344.

Peucker, T.K. and D.H. Douglas 1975. "Detection of Surface Specific Points by Local Parallel Processing of Discrete Terrain Elevation Data," Computer Graphics and Image Processing, vol. 4:375,387.