Automated Feature and Name Placement on Parallel Computers

James E. Mower

ABSTRACT. *Implementations of general-purpose automated name-placement algorithms characteristically require extensive amounts of serial computing time to select names from large databases and place them onto small-scale maps. This paper presents a parallel algorithm for the automated selection of point features from a scale-independent database, and their placement on maps at a continuous range of presentation scales. The algorithm has been implemented and evaluated on a Connection Machine 2, a single-instruction-stream, multiple-data-stream computer. The execution performance evaluations presented here suggest that parallel computing environments offer cartographers and geographic information systems specialists fast and flexible alternatives to serial models of computation.*

KEYWORDS: *automated name placement, parallel computing, SIMD.*

Introduction

Few aspects of cartographic production are more time-consuming and less interesting to perform than labeling the map. Yet the automation of name placement has proven difficult to implement successfully. To function adequately, an automated name-placement system must select names from scale-independent databases, place them in a manner acceptably similar to that of a trained cartographer, and do so quickly.

The cost of performing automated name placement is a function of the constraints that cartographic production systems are required to satisfy. Primarily, names must be legible and they must not overlap other names or symbols of the same color. Of equal importance to the map user is that names refer unambiguously to their intended referent symbol. A lesser but still important constraint is that labeling conform to cartographic or artistic conventions; area labels should fill their areas; point-feature labels should align with the major horizontal axis of their composition, and so on (Imhof 1975; Robinson et al. 1984).

This paper presents a new name-placement procedure that executes on the Connection Machine 2 (CM-2), a massively parallel computer developed by Thinking Machines Inc. Parallel computers offer cartographers an exciting opportunity to execute computationally expensive procedures in a small portion of the time required to execute them on a fast workstation. Current high-performance workstations execute approximately 50 to 100 million floating point operations per second (MFLOPS). In comparison, current models of the CM-2 operate at 4,000 MFLOPS when fully configured with 65,536 processing units. Parallel computers have been used for image-processing applications in remote sensing since the early 1970s, but have only recently served as development platforms for work in cartography and geographic information systems (GISs). Areas of active research include the detection of line intersections in polygon overlays (Franklin et al. 1989; Hopkins, Healey, and Waugh 1992), and the generation of shortest paths through networks (Ding, Densham, and Armstrong 1992).

The CM-2 was selected as a development platform for this project because

1. The number of available processors is large enough to allow each name or feature in the project database to be assigned to a unique processor
2. Competition for map space can be managed through the interconnection of conflicting processors
3. The project can be implemented in a parallel variant of a common high-level computer language

Following an introduction to parallel computing hardware, this paper presents a parallel procedure for the automated placement of point-feature labels and discusses its implementation on a CM-2. The discussion is accompanied by a presentation of execution performance statistics for the implementation and a sample of its graphic output. The paper concludes with a statement on the applicability of parallel computing architectures for other aspects of automated cartographic production.

Approaches to Automated Name Placement

The procedures that cartographers apply to the labeling of point features vary with the objectives of the mapping context. In some contexts, a set of features is selected for placement before labels are applied. In others, the cartographer experiments with alternative selections of features and patterns of label placement. Features are added or deleted dynamically as they compete with one another for map space.

A general-purpose, name-placement procedure must be able to map arbitrary regions over a broad range of scales,

James E. Mower is an assistant professor in the Department of Geography and Planning and codirector of the Laboratory for Geographic Information Systems and Remote Sensing at the State University of New York at Albany, Albany, NY 12222.

extract features in the region and rank their importance, and resolve competitions for map space in favor of features of greater importance. At smaller map scales, insufficient space will exist to place all features and their labels in an acceptable manner. Ultimately, conflict resolution must end with the deletion of the least important features in crowded regions.

Since 1980, numerous articles on the design of automated name-placement procedures have appeared in cartographic journals and conference proceedings. Systems have been described that select and label point, line, and area features over a broad range of scales. Some of these designs have been directed toward the automation of name placement in production cartography environments, others toward its automation in a variety of end-user applications.

Zoraster (1991) categorized these procedures as either heuristic or optimization approaches. Heuristic approaches apply assumptions about the availability of map space during label placement and are capable of recovering, or backtracking, from incorrect intermediate solutions (Hirsch 1982; Doerschler and Freeman 1989; Ebinger and Goulette 1989; Freeman and Ahn 1984; Langran and Poiker 1986; Mower 1989). Optimization approaches solve a label-placement problem by formulating it as an integer programming problem. The goal of the integer programming problem is to minimize an objective function subject to constraints that prevent overlap between labels and between labels and feature symbols. The variables of the optimization problem represent label-placement options for each feature. The coefficients of the objective function are derived from rules for good label placement (Cromley 1985, 1986; Zoraster 1986, 1990; Zoraster and Bayer 1987).

Of all the procedures developed under one or the other approach, few have been implemented successfully in commercial mapping systems. The most important factor limiting the usefulness of automated name-placement systems, especially for real-time end-user applications, has been the excessive time required to complete labeling jobs on large data sets. The reduction of execution times must become a top priority for research in automated name-placement systems aimed at commercial or applied use.

All previous automated name-placement procedures have been developed for serial computing platforms, upon which a small number of processing units, controlled by a single central processing unit (CPU), execute instructions and store their results in random-access memory on silicon chips. Hillis (1985) argues, however, that up to 97% of the total silicon in a modern serial computer goes unused during any particular CPU cycle. As the area of silicon devoted to processing increases relative to the area devoted to memory, computational efficiency increases as well. Although the individual processors in a parallel computer are often less powerful than the CPU in a typical desktop microcomputer, current parallel computers with sufficiently large numbers of small processors can perform more floating point operations per second than any current serial computer. Given equivalent advances in processor design for serial and parallel architectures, future parallel computers will maintain processing advantages over future serial computers. Several manufacturers already employ standard microcomputer or workstation CPUs in parallel configurations.

MIMD and SIMD Architectures

Like the CPU in a conventional serial computer, the processors within a parallel computer execute programs as a stream of individual instructions. In a multiple-instruction, multiple-data (MIMD) machine, each processor is capable of receiving a unique instruction stream, thus allowing each processor to act as an autonomous computer (Figure 1). In a single-instruction, multiple-data (SIMD) machine, all processors listen to a single instruction stream broadcast from a single controlling CPU, frequently known as a front-end (Figure 2). Each processor becomes active or inactive by executing or ignoring the current instruction as determined by the state of its local data. The broadcast of a new instruction is delayed until every active processor has finished executing the previous instruction.

SIMD machines are useful for performing identical operations on large numbers of data elements. Many such tasks exist in cartography, remote sensing, and GISs (e.g., determining the direction of drainage for a cell in a digital elevation model [DEM], applying a filter over a pixel array, or detecting line intersections in a vector overlay procedure). For each of these problems, a single processor can be dedicated to a single problem element: an elevation, a pixel, or a line segment.

MIMD machines are better choices for solving problems

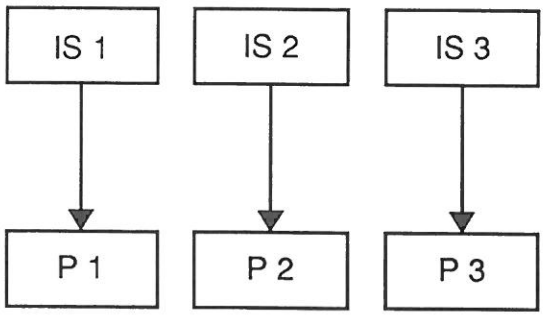

Figure 1. *Multiple instruction streams in an MIMD computer. Each processor P_i receives a unique instruction stream IS_i.*

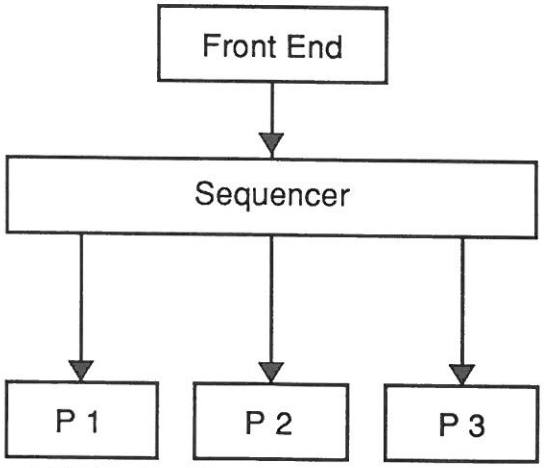

Figure 2. *The broadcast of instructions from a front-end computer to a network of SIMD processors. All processors* $\overset{3}{\underset{i-1}{P}}$ *receive an identical instruction simultaneously.*

with multiple, independent components. The simultaneous computation of a hill-shaded image and a drainage network for a grid cell DEM could be carried out on two large processors without requiring the processors to share the results of intermediate computations. When such requirements occur, however, the programmer must synchronize the processors to prevent the premature computation of a dependent component. Good synchronization in a MIMD program lowers the average amount of idle time for machine processors, increasing the overall efficiency of program execution. However, synchronization is generally more difficult to achieve on MIMD machines than on SIMD machines.

This project uses the CM-2, a SIMD computer, to implement a parallel procedure for automated name placement. The procedure assumes the existence of a communication network that allows the interconnection of any two processors at any location within the network. As an introduction to the computational requirements of the procedure, the following section provides a brief discussion of the processor and network architecture of the CM-2.

CM-2 Architecture

The CM-2 is a SIMD machine that can incorporate up to 64K processors. Unlike conventional desktop microprocessors that read or write 16 or 32 bits at a time, the CM-2 processors read or write only one bit at a time, making them relatively inexpensive to produce (Trew and Wilson 1991). The programmer can direct the CM-2 to emulate a larger parallel machine by establishing an array of virtual processors. Virtual processors are obtained by subdividing the available local memory for a given processor. With an increase in the number of virtual processors comes a corresponding decrease in the available memory for each.

Processors communicate with one another over multidimensional grids or through hypercube routing. Grid communication, though fast, is limited to message-passing between nearest neighbors along an n-dimensional grid. The hypercube router allows message-passing between any two processors, regardless of their address. The flexibility of hypercube routing comes at the expense of slower message handling.

A SIMD Procedure for Automated Placement of Point Features and Their Labels

The following SIMD procedure for the automated placement of point features and their labels stresses the notion of competition for map space through a one-to-one mapping of database features to processors. Processors, and hence features, communicate their position and the position of their labels to other processors via router links in the CM-2. The use of the router assures the interconnection of features with their neighbors, regardless of their relative addresses.

Following a procedure for automated name placement developed by Yoeli (1972), many recent name-placement procedures limit the positioning of point-feature labels to one of a small number of fixed, prioritized positions surrounding their referent symbols (Figure 3). Wu and Buttenfield (1991) report, however, that map publishers vary

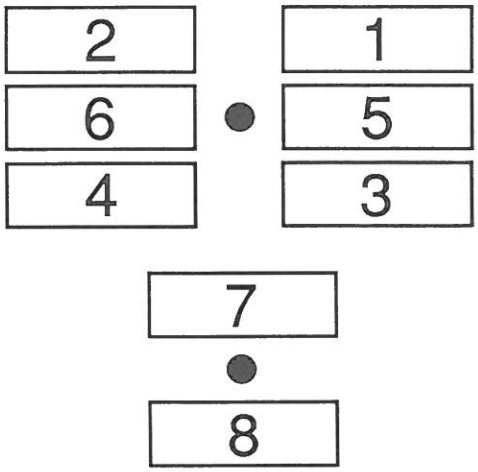

Figure 3. *Label suitability ranking for point features, after Yoeli (1972). Lower numbers signify higher rankings.*

in their preference for placing labels in one or more of these positions.

The procedure below places a point-feature label in one of eight possible locations surrounding a feature symbol without referencing fixed positional weights. Instead, the weights for the positions are made proportional to the number and the associated importance of the point-feature symbols that currently fall within the areas of the positions. Positions with lower weights are preferred over positions with higher weights.

Each top-level step in the procedure is labeled S or P to indicate whether the step and its "children" are performed as a serial operation on the front-end or as a parallel operation on the CM-2. Generally, steps that require input and output operations are performed on the front-end.

To place names on a map:

1 (S) Establish a geographic window onto the feature database. Read the name, geographic coordinates, and importance value for each feature. Place the label for each feature to the upper right of its point symbol.

2 (P) For each feature, create a list of the current feature's neighbors.

3 (P) Until no additions, deletions, or label modifications have been applied to the map, for each feature:

- 3.1) If the label of the current feature overlaps or crowds the point symbol of any of its neighbors' point symbols:
  - 3.1.1) Move the label of the current feature through the remaining positions in the sequence until it can be placed without interference.
  - 3.1.2) If no position works, execute the following deletion procedure (DL):
    - DL 1) Examine each remaining position for features of lesser importance than the current feature. Compute a weight for each position based on the importance of the features having point symbols within the position.
    - DL 2) If all remaining positions have fea-

tures of greater importance than the current feature:

DL 2.1) Delete the current feature.

DL 3) Otherwise, place the label of the current feature at the position holding the least number of interfering features (the lowest weight).

DL 4) If the current feature is overlapped by the label of another feature:

DL 4.1) Delete the current feature.

3.2) While the label of the current feature overlaps the label of a neighboring feature:

3.2.1) If the current feature is more important than the neighboring feature;

3.2.1.1) Do nothing.

3.2.2) Otherwise, move the label of the current feature to the next position in the sequence not occupied by a point-feature label. If there are no remaining positions in the sequence, follow procedure DL.

3.2.3) Reset the label positions of neighbors of deleted features to their lowest-weighted positions.

4 (S) Create an output file containing the coordinates of the remaining features and their labels.

Description of the Procedure

The procedure begins sequentially in step 1, carrying out several data-processing operations on the front-end computer. An input module prompts the user to specify the minimum and maximum latitude and longitude of a region for placement, and to select a map scale. It scans the database for features within the region, comparing their positions against the user's window. For those features falling within the region, the module reads their names and importance values (currently assumed to be the population of a settlement), assigns the feature to a population class, sets the height of the lettering to the height attribute of the population class, and computes the length of its label as the sum of the widths of its individual characters (given in a font look-up table) scaled to the height of the lettering. After transforming the feature's position into rectangular coordinates, the module computes the extent of the feature's neighborhood, the area over which the feature can compete for map space. The neighborhood is defined as the minimum bounding rectangle (MBR) covering the eight possible positions for the feature's label (Figure 4). The module assigns the feature to the next available processor, places its label to the upper right of its point symbol, stores the feature's attributes in the processor's local memory, and continues scanning the database for additional features within the user's region.

When the data-input module has finished scanning the database, step 2 implements a parallel search for overlapping neighborhoods. Each feature simultaneously compares its neighborhood MBR with that of every other feature, recording features as neighbors if their neighborhoods intersect. The length of each feature's neighbor list is scale-dependent: At large map scales, few neighborhoods overlap; at very small scales, all neighborhoods overlap with each other (Figures 5 and 6). When features search for label overlaps, they need only consult with the features on their neighbor lists.

Step 3 initiates an iterative search for label overlaps. Each feature finds the MBR of its label at its current location and searches the features on its neighbor list, checking the position of its label against neighboring point symbols for overlaps. If a feature's label overlaps a point symbol, the feature attempts to move its label to the next of the eight label positions in a clockwise direction from the current position, ending with position 7 (step 3.1). If all the positions are occupied by other point symbols, it competes with its neighbors for a space to place its labels (procedure DL).

Competition for space between two features is always decided in favor of the feature with the greater importance. For this project, the importance of a feature is equated with its population. The feature searches each of its eight possible label positions for point symbols that belong to other features. It computes a weight for each position as the sum of the populations of other features claiming point-symbol space within it. The greater the weight, the lower the preference the feature has for placing its label in that position.

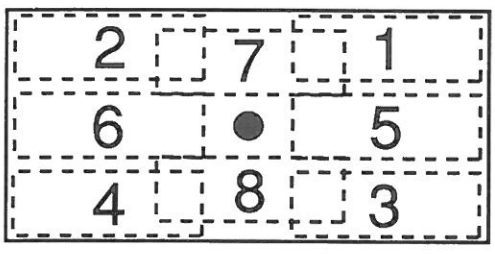

Figure 4. *The neighborhood of a feature defined as the minimum bounding rectangle (MBR) enclosing all possible label positions for a feature.*

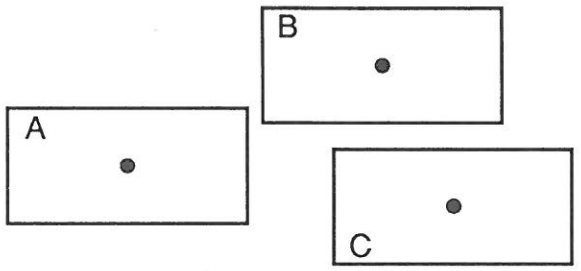

Figure 5. *Feature neighborhoods on a large-scale map. At this scale, the rectangles representing the neighborhoods for features A, B, and C do not intersect.*

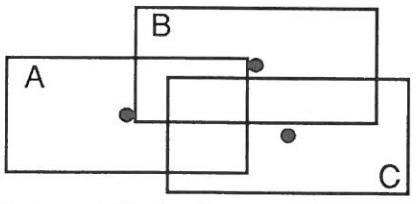

Figure 6. *Feature neighborhoods on a small-scale map. The features represented at a larger scale in Figure 5 now intersect one another at a smaller scale. Each feature recognizes the other two features as neighbors.*

If it finds that any of the features claiming point-symbol space in a position have a population greater than itself, it sets the weight to a maximum value, indicating that the position is invalid for placing its label. After the feature has completed its evaluation of the surrounding positions, it places its label in the position having the smallest weight. If all of the surrounding positions have been marked as invalid, the feature deletes itself. Currently, the procedure does not permit a deleted feature to be reconsidered for placement even if the circumstances that caused its deletion are removed in a later step. In some situations, this may produce a sparser placement than could be achieved manually.

Figure 7 portrays a sample feature, F, surrounded by its neighbors a through i. F cannot place its label to the upper right without overlapping point symbol c; therefore, it calculates weights for each of its label positions to determine the best alternate location for placing its label. Table 1 lists the populations of F and its neighbors and Table 2 summarizes the weights and rankings of each label position for feature F based upon the populations and positions of its neighbors. In this example, position 3 is ranked highest because the sum of the populations for the point symbols within its region is lower than that of any other position. Position 1 is marked I (invalid) to note that feature F cannot place its label over, and thereby delete, a feature of greater population than itself (feature c).

F places its label in position 3, overlapping point symbol i. All of the features, including F and its neighbors, then compare the position of their point symbols with the position of their neighbors' labels. Feature i finds that its point symbol is overlapped by the label of feature F, indicating that F is more important than i. Feature i subsequently deletes itself from the map. In this example, feature e will place its label over feature g, forcing its deletion as well (Figure 8).

At the end of step 3.1, no label will overlap a point symbol, but some labels may overlap other labels. Step 3.2 directs each feature to compare the MBR of its label at its current position with the MBRs of its neighbors' labels at their current positions. If a feature finds that its label overlaps a neighbor's label, it compares its importance with that of its neighbor. If the feature is more important than its neighbor, it does nothing. If it is less important, it moves its label to the next remaining position in its sorted list. If the remaining positions in the sequence have been marked as invalid, the feature deletes itself and its neighbors move their labels back to their lowest-weighted positions in an attempt to claim the vacated space. It is important to note that a feature cannot return the position of its label to a previously visited position unless one of its neighbors has been deleted. This restriction prevents features in a neighborhood from undergoing an endless cycle of label adjustment.

Every time a label is moved or a feature is deleted, the procedure notes that the map has changed and cycles through another round of label adjustment. After completing a cycle in which no changes are made, the procedure draws the map with the labels and point symbols at their current locations.

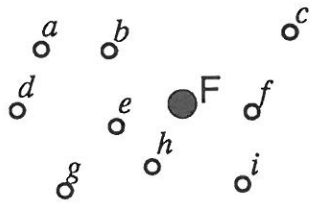

Figure 7. *The spatial distribution of a sample set of point features described in Table 2.*

Implementation Details

The data for this project were extracted from the U.S. Geological Survey (USGS) Geographic Names Information System (GNIS) national database for populated places (USGS 1987). Each record in the GNIS's populated places file contains, among many other attributes, fields for the alphanumeric name of a populated place, the location of its centroid in degrees, minutes, and seconds of latitude and longitude, and, for incorporated places, its population. These fields were included in the project database for records falling within New York state with nonblank population fields. Places were not extracted if they contained alternate name fields. The resulting database contains approximately 5,500 features for New York state.

The current implementation of the procedure was written in C* for execution on a CM-2 at the Northeast Parallel Architecture Center at Syracuse University. The C* programming language, an extension of ANSI C, is one of several high-level programming languages implemented on the CM-2 that include parallel operators for instruction flow control, interprocessor communication, and the manipulation of virtual processors. At the time that the current implementation was being tested, 32K processors were installed on the CM-2, far more than were necessary to allow the assignment of each feature to a unique physical processor.

The procedure itself does not prescribe an appropriate label density, but controls it through the adjustment of a

Table 1. Populations for the sample features represented in Figure 7.

	Feature									
	a	*b*	*c*	*d*	*e*	*f*	*g*	*h*	*i*	F
Pop.	14,302	28,922	99,027	21,331	34,587	16,692	12,845	11,349	4,326	98,402

Table 2. Rankings for the label positions of feature F in Figure 7 based on the populations of its surrounding features.

Position	Total Population (Weight)	Rank
1	99,027	I
2	43,224	6
3	4,326	1
4	12,845	2
5	16,692	4
6	55,918	7
7	28,922	5
8	15,675	3

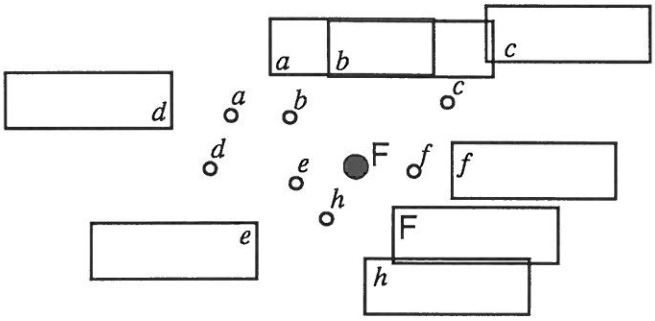

Figure 8. *The initial label placement of the point features in Figure 7 after resolving label/point-symbol overlaps. Note the deletion of point symbols g and i.*

variable width buffer surrounding each label. An increase in the width of the buffer beyond 0 extends the MBR of the label, effectively increasing the area over which its feature searches for overlapping labels or point-feature symbols. Except for the map in Figure 17, maps created for this project used a buffer width of 0. In accord with common practice, labels mask underlying line work and avoid intersecting the neat line. The size of a label and its referent point symbol are determined by the membership of the associated feature in one of five population classes, each having a unique assigned label height.

The user begins the execution of the program on the front-end computer by supplying the name of the feature database, the geographic coordinates of the area to be labeled, the scale of the output map, and the name of the graphic output file. Following the termination of the program, the user downloads the graphic output file to a graphic workstation.

To remain device-independent, the procedure does not specify a particular graphic environment for map production. The current implementation generates a PostScript language program that specifies the layout of the map. The implementation calculates label MBRs by referring to a character-width look-up table for the label's font. The current look-up table contains information for the Helvetica typeface on the Apple Laserwriter. Additional fonts can be supported as their look-up tables are created. Numerous PostScript interpreters have been used to display the maps produced for this project including Newsprint from Sun Microsystems Inc.; Ghostscript, an interactive PostScript interpreter; and the PostScript interpreter within an Apple Laserwriter, upon which the maps for this article were produced.

Analysis of the Procedure

If the number of places in the feature database is less than or equal to the number of processors, the running time of the procedure will be dominated by label-overlap detection and conflict resolution (step 3 of the procedure). Whenever a feature moves its own label or whenever one of its neighbors moves its label, the feature retrieves the positions of each of its neighbors' point symbols and labels and checks the position of its label against them for overlap. Because all SIMD processors receive the same instructions simultaneously, no processor can begin a new instruction until every feature has checked all of the neighbors on its list. The time that a feature requires to check all its neighbors increases as a constant value multiplied by the length of the list of its neighbors.

The average length of a neighborhood list is inversely related to map scale, as shown by Figures 5 and 6. At a sufficiently large scale, no neighborhood intersects any other neighborhood. As scale decreases, more neighborhoods intersect with one another until, at some sufficiently small scale, all neighborhoods intersect. Without optimization, the running time of the program is expected to increase linearly with the length of the longest neighborhood list. In practice, running time is reduced by removing references to deleted neighbors from neighborhood lists at the end of steps 3.1 and 3.2.

Whenever a label overlaps a point-feature symbol or another label, the conflict is resolved in favor of the feature of greater importance. At large map scales, few neighborhood intersections occur and most features are able to position their labels on the map. As scale decreases and more neighborhood intersections occur, the opportunities for the neighborhoods of features of lesser importance to intersect with those of greater importance increase. It is expected, then, that higher proportions of less important features will appear on larger-scale maps than on smaller-scale maps.

Benefits of Parallel Processing

SIMD computers provide a better platform for modeling competition for map space than do serial platforms. The parallel procedure does not require that features be ordered for placement onto the map in a sequential manner. Mower (1989) found that when features are applied to a map sequentially, features that are applied early in the sequence are more likely to be deleted through competition for map space than features applied late in the sequence. In that

Table 3. Percentages of features selected from each decile for maps in Series 1.

Scale Denominator	1	2	3	4	5	6	7	8	9	10
500,000	28.57	38.09	36.50	61.90	58.73	71.42	82.53	85.71	95.23	96.92
750,000	1.58	6.34	6.34	19.04	22.22	30.15	58.73	53.96	74.60	90.76
1,000,000	0.00	1.58	0.00	7.93	1.58	9.52	25.39	36.50	53.96	84.61
1,250,000	0.00	0.00	0.00	1.58	0.00	1.58	6.34	19.04	38.09	69.23
1,500,000	0.00	0.00	0.00	1.58	0.00	3.17	1.58	6.34	17.46	61.53
1,750,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	11.11	50.76
2,000,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	4.76	43.07
2,250,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	3.17	33.84
2,500,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	29.23
2,750,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	23.07
3,000,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.58	18.46
3,250,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	13.84
3,500,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	15.38
3,750,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	12.30
4,000,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.69
4,250,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.69
4,500,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.69
4,750,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.15
5,000,000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.61

study, small-scale maps that were produced from a list of features sorted by population often included a disproportionate selection of the most populous features and the least populous features. Throughout the sequential procedure, the most populous places would win space conflicts with less populous places. In the latter stages of the procedure, places with low population values were easily placed into the holes left by the removal of larger places. The parallel procedure considers all features simultaneously, thereby eliminating order-related effects.

Name placement, of course, is one of many cartographic functions within a GIS and one that is rarely performed in isolation. Before parallel computers can become common platforms for the development and use of such systems, it must be demonstrated that a large number of the procedures that make up a GIS can be ported to parallel environments with performance and modeling improvements that justify their costs, currently on the order of $2 million for the CM-2. It is reasonable to assume that the cost of parallel computers will decrease as initial research and de-

Table 4. Percentages of features selected from each decile for maps in Series 2.

Scale Denom.	Dim. (°long x °lat)	1	2	3	4	5	6	7	8	9	10
1,000,000	1 x 1	0.00	5.00	0.00	2.50	2.50	15.00	15.00	30.00	40.00	81.81
2,000,000	3 x 3	0.00	0.00	0.00	0.40	0.40	0.00	1.22	2.86	9.42	38.49
3,000,000	5 x 5	0.00	0.00	0.00	0.20	0.00	0.00	0.20	0.41	4.52	17.82
4,000,000	7 x 5	0.00	0.00	0.00	0.00	0.00	0.00	0.72	0.00	1.08	11.44

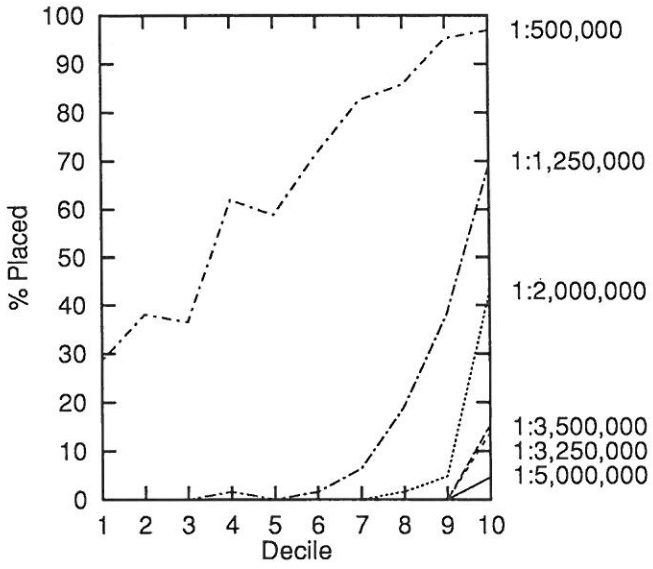

Figure 9. *Graph of the percentages of features in each decile that were placed on selected maps from Series 1.*

velopment costs are recovered and as production methods and components become standardized.

Fortunately, many of the problems encountered in geographic processing find natural expressions in SIMD or MIMD algorithms. SIMD computers have long been used for processing remotely sensed imagery using a processor-per-pixel model (Rohrbacher and Potter 1977). The author has shown that this model allows the computation of hill-shaded images for entire USGS 1:24,000 DEMs on a CM-2 in less than three seconds in real time (Mower 1992). In this case, fast execution times are attained by minimizing the number and complexity of interprocessor messages. Each processor calculates a slope, aspect, and illumination value for its pixel by acquiring only the elevations of its eight connected neighbors over a two-dimensional grid-network architecture. No message ever needs to travel more than one step in each grid dimension to reach its destination.

Not all geographic problems benefit by the application of SIMD models, however. Procedures that require extensive information-sharing among grid cells, that operate on only a few cells at a time, or that operate on levels of topological structure greater than the pixel or grid cell may underutilize SIMD machines or incur large interprocessor communication costs. Mower (1992) has found this to be true for aspects of drainage-basin analysis from grid cell DEMs and for applications of the Douglas simplification procedure (Douglas and Peucker 1973). Such procedures may find better expression under MIMD models if their data can be decomposed into autonomous units, such as topological pits or line segments. In the first case, each MIMD processor assigned to a unique pit computes its drainage basin boundary using standard sequential procedures. In the second case, each processor assigned to a line segment simultaneously applies a copy of the simplification procedure to it. Because neither case requires that information be shared across unit boundaries, both should attain reductions in execution times over sequential computers proportional to $\frac{n}{p}$, where n is the problem size and p is the number of available processors. The actual speed-up may be less than the predicted value if some of the processors are idle during program execution (Hansen 1990). The author is currently developing MIMD versions of the drainage-basin delineation problem and the Douglas line-simplification algorithm to test the validity of these assumptions for large data sets.

Testing the SIMD Name-Placement Procedure

To demonstrate the effectiveness of SIMD computers for automated name placement, a series of maps was created that establishes relationships among scale, execution time, and the distribution of names on the maps by importance. The series is composed of 19 maps of central New York state, a region characterized by an even distribution of features, bounded by 74° 45′ 0″ W, 76° 0′ 0″ W, 43° 30′ 30″ N, and 42° 15′ 0″ N (Series 1). The maps in the series vary in

Table 5. Timing statistics for Series 1 map production. Rows refer to individual maps in the series and include the map-scale denominator, maximum number of neighbors found for an individual feature at the start of processing, total running time in front-end CPU seconds, percentage increase in the maximum number of neighbors over the base scale of 1:500,000, and percentage increase in running time over the base scale of 1:500,000.

Scale Denominator	Number of Neighbors (original)	Running Time (CPU seconds)	% Increase Neighbors over 1:500,000	% Increase Time over 1:500,000
500,000	20	137.40	----------	----------
750,000	36	161.95	80	17.87
1,000,000	55	155.54	175	13.20
1,250,000	78	159.28	290	15.92
1,500,000	106	169.87	430	23.63
1,750,000	148	186.22	640	35.53
2,000,000	179	200.69	795	46.06
2,250,000	210	206.18	950	50.06
2,500,000	244	213.81	1120	55.61
2,750,000	283	231.33	1315	68.36
3,000,000	306	240.93	1430	75.35
3,250,000	327	249.28	1535	81.43
3,500,000	343	255.87	1615	86.22
3,750,000	360	260.99	1700	89.95
4,000,000	374	266.84	1770	94.21
4,250,000	390	273.75	1850	99.24
4,500,000	405	279.76	1925	103.61
4,750,000	420	288.58	2000	110.03
5,000,000	437	295.06	2085	114.75

scale from 1:500,000 to 1:5,000,000 in scale-denominator increments of 250,000. Table 3 summarizes the regions, scales, decile percentages, and execution times for each map in Series 1. For each map, the implementation classified the features within the region into population deciles, and recorded the percentage of features in each decile that appeared on the map. Load-independent execution times were recorded by the UNIX procedure gprof. It was expected

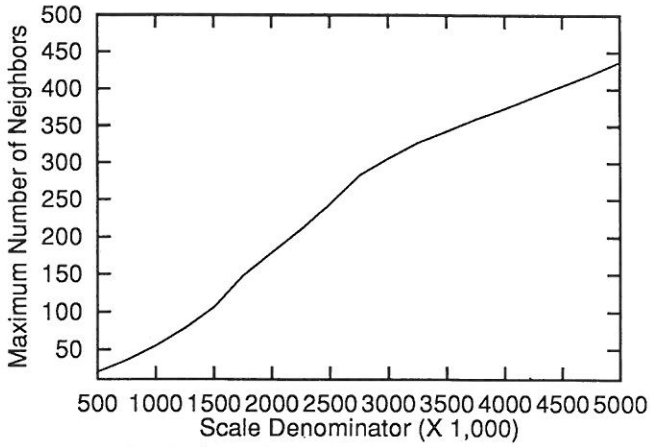

Figure 10. *Graph of the relationship between the maximum number of neighbors for an individual processor and map scale for all maps in Series 1.*

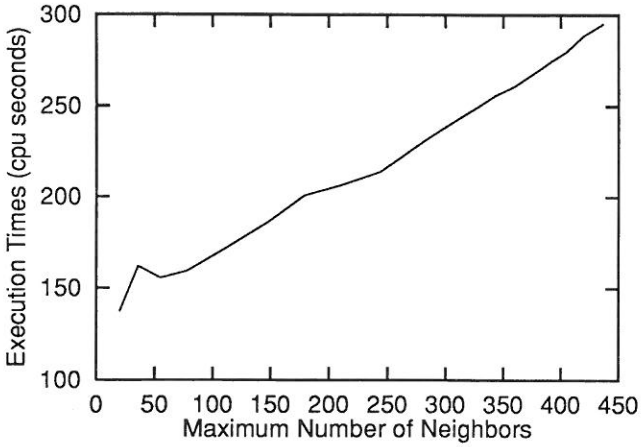

Figure 11. *Graph of the relationship between execution time and the maximum number of neighbors for a processor for all maps in Series 1.*

that running time would increase at a rate less than the increase of the length of the largest neighborhood list.

To demonstrate the capability of the program to make scale-independent selections of features over regions of varying dimensions, a second series of maps was prepared (Series 2) that varies in scale from 1:4,000,000 to 1:1,000,000 in scale-denominator increments of 1,000,000 and in the dimensions of their mapped regions from 1 degree longitude by 1 degree latitude to 7 degrees longitude by 5 degrees latitude. Table 4 summarizes the regions, scales, decile percentages, and execution times for each map in Series 2.

Analysis of Series 1 Test Results

Table 3 lists the percentage of features selected for placement at each scale for each map produced for Series 1. Figure 9 portrays the relationship between map scale and the number of features placed per decile for selected maps in this series. As the number of features selected for placement on each map decreases through scales of 1:500,000 to 1:5,000,000, the bulk of the reductions at each step are car-

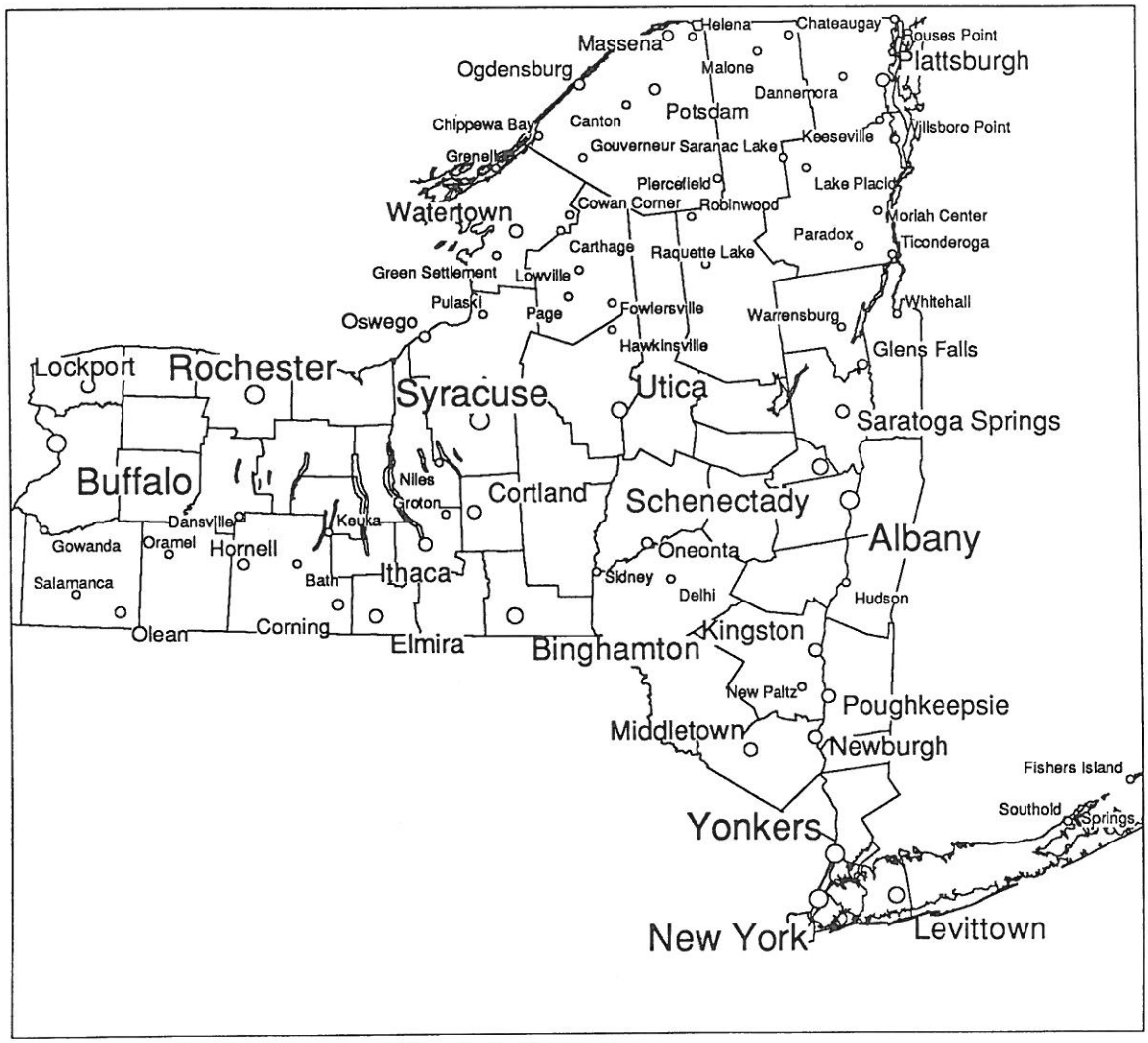

Figure 12. *Series 2, 1:4,000,000. Map covers 7° longitude, 5° latitude.*

ried by the lower population deciles. At 1:500,000, the percentage of features selected for the map from each decile increases monotonically from 28.57% of the features in decile 1 to 96.92% of the features in decile 10. At 1:1,250,000, no features remain from deciles 1, 2, 3, and 5; only 1.58% of the features from decile 4 and 6 remain. Features selected from deciles 7 through 10 now dominate the map, with values of 6.34%, 19.04%, 38.09%, and 69.23%, respectively. At 1:2,000,000, only deciles 8, 9, and 10 are represented by 1.58%, 4.76%, and 43.07%, respectively, of their original members. At 1:3,250,000, no members from deciles 1 through 9 have been selected, but 13.84% of the features from decile 10 remain. As scale decreases to 1:5,000,000 (with the exception of 1:3,500,000), the number of features selected from decile 10 remains the same or decreases with each step. Clearly, the parallel procedure inhibits the selection of low-population features on small-scale maps.

Without optimization, it was expected that the running time of the program would increase linearly with the maximum number of neighbors for a feature. By collapsing the lengths of neighbor lists to account for deleted features, the actual running times were kept much lower than the predicted times. For each scale, Table 5 lists the running time and the maximum number of neighbors for features as well as the percentage increase in maximum neighbors and running time over the base scale of 1:500,000. Figure 10 shows the relationship between map scale and the maximum number of neighbors for map features. Figure 11 shows the relationship between the maximum number of neighbors and running time. From 1:500,000 to 1:3,000,000, the maximum number of neighbors for a feature increased by 1,430% (from 20 to 306), while running time increased only by 75.35% (from 137.40 to 240.93 CPU seconds). From 1:3,000,000 to 1:5,000,000, the maximum number of neighbors for a feature increased by 42.81% (from 306 to 437), while running time increased by 22.47% (from 240.93 to 295.06 CPU seconds).

To make a rough comparison of the execution times for the parallel implementation and a functionally equivalent serial procedure implemented on a Sun SPARCStation 2, the author ran the 1:3,000,000 Series 1 map on the SPARCStation. The SPARCStation completed the map in one hour and 25 minutes. The CM-2 completed the map in four minutes and one second.

Figure 13. *Series 2, 1:3,000,000. Map covers 5° longitude, 5° latitude.*

Analysis of Series 2 Test Results

The maps in Series 2 (Figures 12, 13, 14, and 15) represent a zoom from 1:4,000,000 to 1:1,000,000 centered roughly on Oneonta, New York. As scale increases and the field of view decreases, the percentages of features selected from each population decile increase steadily (excepting decile 7 from 1:4,000,000 to 1:3,000,000), as they did for increases in scale alone for the maps in Series 1 (Table 4, Figure 16).

As scale decreases from 1:1,000,000 to 1:4,000,000, few places of low population remain on the map, especially near large urban centers. In more remote areas, where places of low population generally compete with one another for placement, some of these places inevitably survive. Whether this is to be considered a bug or a feature of the procedure is best resolved with respect to the goal of the map. If the user wishes to fit as many places as possible onto the map, then their placement is appropriate. If not, then it is appropriate to change the measure of importance from population to some other index or to restrict the placement of features to those having a specific attribute. Neither of these

Figure 14. *Series 2, 1:2,000,000. Map covers 3° longitude, 3° latitude.*

changes would affect the overall performance of the procedure.

Because it refrains from forcing the placement of labels into some preferred slot, the procedure generally creates a balanced areal distribution of labels on each map. However, features that appear near the neat line are constrained from placing their labels in those positions that intersect the neat line. As a result, such places have fewer labeling options than other features within the region, and are subsequently less likely to appear on the map. Figure 12 includes Lockport but not Niagara Falls, although the latter city has a higher population. Unfortunately, the label for Niagara Falls can only fit in three positions: to the upper right of its point symbol, directly to the right, or to the lower right. Each of these positions interfere with the label or point symbol for Rochester, which has a higher population than Niagara Falls.

As the number of labeled features increases over a unit area, the likelihood that a map user will associate a given label with the incorrect point feature increases. By increasing the size of each feature's neighborhood, the programmer extends the search for overlaps beyond the edge of the feature's label. With extended neighborhoods, a feature will note that an overlap has occurred even when its label is merely "close" to another feature's label or point symbol. It will choose another location for its label or force the "overlapped" label to move. Unfortunately, an increase in the size of feature neighborhoods usually results in a decrease in the number of selected features.

To force the implementation to place the maximum num-

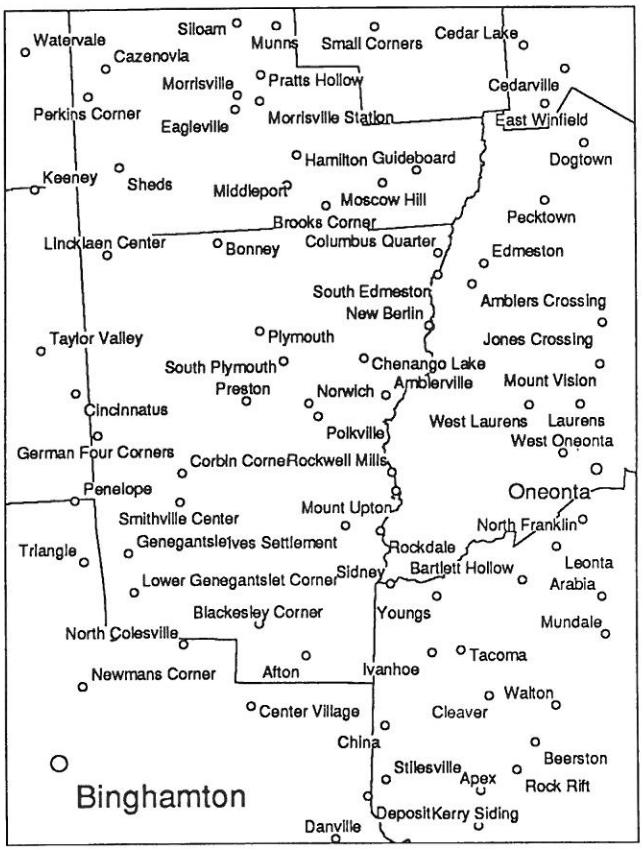

Figure 15. *Series 2, 1:1,000,000. Map covers 1° longitude, 1° latitude.*

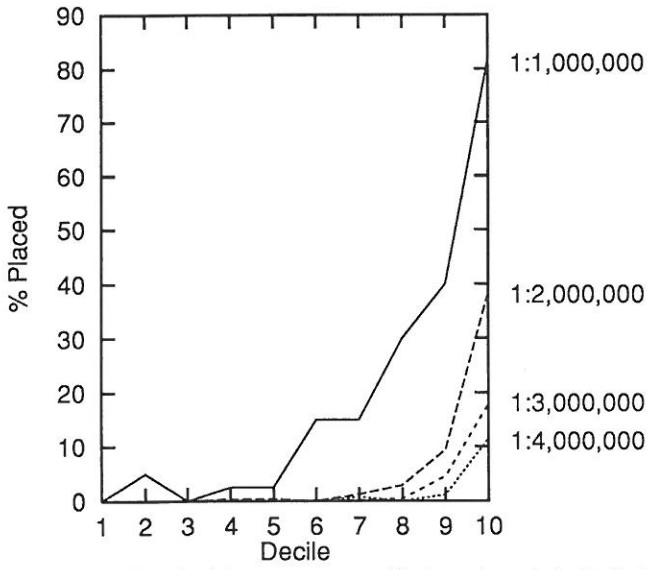

Figure 16. *Graph of the percentages of features in each decile that were placed on selected maps from Series 2.*

ber of features, Series 1 and Series 2 maps were created with neighborhood sizes set to their minimum values. As a result, some names appear too close to one another. Figure 15 contains several pairs of names that are too close: Genegantslet and Ives Settlement, Lower Genegantslet

Figure 17. *Map covering the same region and displayed at the same scale as Figure 15. The feature neighborhoods have been extended to increase the minimum distance between adjacent labels.*

Corner and Sidney, Deposit and Kerry Siding, and Corbin Corner and Rockwell Mills. The map in Figure 17 covers the same region at the same scale as does Figure 15, but extends the feature neighborhoods to increase the distance between names on the map. Since labels are spread further apart on Figure 17 than on Figure 15, fewer places appear on Figure 17.

Summary

It has been the intent of this paper to demonstrate the power and flexibility that parallel processing can bring to automated name placement. Test results of the parallel name-placement procedure introduced in this paper show that

1. Features and places can be selected from a single database over varying windows and map scales
2. The execution performance of the procedure degrades slowly as scale decreases
3. If the number of names in the map database window is less than the total number of processors, increases in execution time are related primarily to increases in feature density, not to increases in the overall number of names in the map database window
4. Feature selection is limited to the most important places as scale decreases
5. The selection of unimportant features is suppressed in areas of high feature density

Although much more work is required to produce a fully functional system with provisions for line and area names or with the capabilities to produce thematic compositions, the application of the current generation of parallel computers to automated name placement can result in enormous reductions in computing time over serial-computer implementations. It is also worth restating that parallel computing architectures offer the designers of cartographic algorithms the freedom to model their computational environments around their problems, rather than force them to fit their problems to fixed architectures. It is hoped that with the increasing availability of parallel computers will come a matching interest in the exploration of these new and exciting resources.

ACKNOWLEDGMENT

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

REFERENCES

Cromley, R.G. 1985. "An LP Relaxation Procedure for Annotating Point Features Using Interactive Graphics." *Proceedings of Auto-Carto 7,* pp. 127-132.

___. 1986. "A Spatial Allocation Analysis of the Point Annotation Problem." *Proceedings of the Second International Symposium on Spatial Data Handling,* pp. 38-49.

Ding, Y., P.J. Densham, and M.P. Armstrong. 1992. "Parallel Processing for Network Analysis: Decomposing Shortest Path Algorithms for MIMD Computers." *Proceedings of the Fifth International Symposium on Spatial Data Handling,* pp. 682-691.

Doerschler, J., and H. Freeman. 1989. "An Expert System for Dense-Map Name Placement." *Proceedings of Auto-Carto 9,* pp. 215-224.

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.

Ebinger, L.R., and A.M. Goulette. 1989. "Automated Names Placement in a Non-Interactive Environment." *Proceedings of Auto-Carto 9,* pp. 205-214.

Franklin, R., C. Narayanaswami, M. Kankanhalli, D. Sun, M. Zhou, and P.Y.F. Wu. 1989. "Uniform Grids: A Technique for Intersection Detection on Serial and Parallel Machines." *Proceedings of Auto-Carto 9,* pp. 100-109.

Freeman, H., and J. Ahn. 1984. "AUTONAP: An Expert System for Automatic Map Name Placement." *Proceedings of the First International Symposium on Spatial Data Handling,* pp. 544-571.

Hansen, P.B. 1990. "The Nature of Parallel Programming." *Natural and Artificial Parallel Computation,* M.A. Arbib and J.A. Robinson (eds.). Cambridge, Massachusetts: The MIT Press. Hillis, W.D. 1985. *The Connection Machine.* Cambridge, Massachusetts: The MIT Press.

Hirsch, S.A. 1982. "An Algorithm for Automatic Name Placement Around Point Data." *The American Cartographer,* vol. 9, no. 1, pp. 5-17.

Hopkins, S., R.G. Healey, and T.C. Waugh. 1992. "Algorithm Scalability for Line Intersection Detection in Parallel Polygon Overlay." *Proceedings of the Fifth International Symposium on Spatial Data Handling,* pp. 210-218.

Imhof, E. 1975. "Positioning Names on Maps." *The American Cartographer,* vol. 2, no. 2, pp. 128-144.

Langran, G.E., and T.K. Poiker. 1986. "Integration of Name Selection and Name Placement." *Proceedings of the Second International Symposium on Spatial Data Handling,* pp. 50-64.

Mower, J.E. 1989. "The Selection, Implementation, and Evaluation of Heuristics for Automated Name Placement." Doctoral dissertation, State University of New York at Buffalo. Available through University Microfilms International, Ann Arbor, Michigan.

___. 1992. "Building a GIS for Parallel Computing Environments." *Proceedings of the Fifth International Symposium on Spatial Data Handling,* pp. 219-229.

Robinson A.H., R.D. Sale, J.L. Morrison, and P.C. Muehrcke. 1984. *Elements of Cartography.* New York: John Wiley and Sons.

Rohrbacher, D., and J.L. Potter. 1977. "Image Processing with the Staran Parallel Computer." *Computer,* vol. 10, no. 8, pp. 54-59.

Trew, A., and G. Wilson. 1991. *Past, Present, Parallel: A Survey of Available Parallel Computer Systems.* London: Springer-Verlag.

U.S. Geological Survey. 1987. *Geographic Names Information System.* Reston, Virginia: U.S. Geological Survey.

Wu, C.V., and B. Buttenfield. 1991. "Reconsidering Rules for Point-Feature Name Placement." *Cartographica,* vol. 28, no. 1, pp. 10-27.

Yoeli, P. 1972. "The Logic of Automated Map Lettering." *The Cartographic Journal,* vol. 9, no. 2, pp. 99-108.

Zoraster, S. 1986. "Integer Programming Applied to the Map Label Placement Problem." *Cartographica,* vol. 22, no. 3, pp. 16-27.

___. 1990. "The Solution of Large 0-1 Integer Programming Problems Encountered in Automated Cartography." *Operations Research,* vol. 38, no. 5, pp. 752-759.

___. 1991. "Expert Systems and the Map Label Placement Problem." *Cartographica,* vol. 28, no. 1, pp. 1-9.

Zoraster, S., and S. Bayer. 1987. "Practical Experience with a Map Label Placement Algorithm." *Proceedings of Auto-Carto 8,* pp. 701-708.