Pergamon

0098-3004(94)00047-6

Computers & Geosciences Vol. 20, No. 9, pp. 1365–1378, 1994
Copyright © 1994 Elsevier Science Ltd
Printed in Great Britain. All rights reserved
0098-3004/94 $7.00 + 0.00

DATA-PARALLEL PROCEDURES FOR DRAINAGE BASIN ANALYSIS

JAMES E. MOWER
Department of Geography and Planning, SUNY at Albany, Albany, NY 12222, U.S.A.

(Received 16 December 1993; accepted 18 April 1994)

Abstract—Procedures for drainage basin delineation and modeling of drainage flow accumulation are provided for single instruction stream, multiple data stream (SIMD) computers. Performance statistics were gathered from a series of windows on a DEM and analyzed for an implementation of the procedures, written in C* and executed on a Thinking Machines CM-5 in SIMD mode. Results of the analysis show that the SIMD implementation completes execution in less than one-half the time of a comparable sequential implementation when both are run over the largest window. The results also show that execution times for the SIMD implementation initially increase linearly with the ratio of virtual processors to physical processors.

Key Words: Drainage basin delineation, Drainage flow accumulation modeling, SIMD, C*.

INTRODUCTION

Numerous hydrological modeling applications rely upon procedures that automatically extract drainage basin boundaries and drainage networks from grid cell DEMs. Some of these procedures, notably those that calculate slope, aspect, and cell drainage direction, operate locally on grid cells and their 8-adjacent neighbors. If each grid cell were represented by a unique processor on a sufficiently large parallel computer, the procedures in this group would execute simultaneously on all grid cells. Other procedures, such as those that perform drainage basin delineation and drainage flow accumulation, propagate values from starting cells to most or all of the other cells in the grid in iterative or recursive steps. Given the same parallel computer, these procedures also would execute simultaneously, but for a subset of the processors on any iteration. Naturally, the procedures in the first group achieve a higher degree of parallelism than those of the second group. Nonetheless, parallel implementations of the procedures in the second group also can achieve significant speed-ups over comparable sequential implementations.

This paper will introduce data-parallel procedures for performing drainage basin delineation and drainage flow accumulation on single instruction stream, multiple data stream (SIMD) computers. SIMD computers can apply identical operations synchronously over multiple spatial entities, providing elegant and efficient solutions to certain problems in drainage basin analysis. The author has tested these procedures on a CM-5 running in its synchronous mode as a SIMD computer. Performance statistics for the SIMD implementation and a comparable sequential implementation for a range of drainage analysis routines for a range of DEM sizes are included and discussed here.

The SIMD implementation was expected to attain large performance speed-ups for the sequential implementation within procedures that keep most of the processors active simultaneously throughout their execution. Speed-ups also were expected for the remaining SIMD procedures, written specifically to exploit the strengths of SIMD execution environments. Speed-ups were not expected for code invoking expensive parallel operators, including those that perform certain interprocessor communication and processor context setting tasks. To show the effects of such operations on execution times, two SIMD approaches to the drainage basin delineation problem are presented and tested. Both approaches are equivalent functionally but differ in the number of communication and processor context setting operations that they perform.

The SIMD and sequential implementations operate on standard USGS 1:24,000 DEMs and produce PostScript image files displaying analytical hill shading, drainage basin delineation, and drainage flow accumulation at a user-selected scale. An example from the source code for the CM-5 implementation is provided to illustrate key programming issues.

No parallel procedure for automated mapping will gain general acceptance if it must be performed in isolation from other mapping activities. Fortunately, developmental work on parallel algorithms for name placement (Mower, 1993), line simplification (Mower, 1994), polygon overlay (Franklin and others, 1989; Hopkins, Healey, and Waugh, 1992), location/allocation analysis (Ding, Densham, and

Armstrong, 1992), viewshed analysis (Mills, Fox, and Heimbach, 1992), and other applications is helping to stimulate interest in parallel computers as environments for solving problems in spatial data handling.

DELINEATING DRAINAGE BASIN BOUNDARIES AND MODELING DRAINAGE FLOW ACCUMULATION

The sequential approach

Marks, Dozier, and Frew (1984) developed a recursive, sequential procedure for delineating drainage basin boundaries from grid cell DEMs. Starting at a grid cell representing a basin outlet, the procedure looks at each of the cell's 8-adjacent neighbors in turn to see if any, or all are upstream from the outlet (Fig. 1). If so, the upstream cells are considered members of the drainage basin and the procedure applies itself recursively, searching for cells that are uphill from the new members. Recursion stops along a thread when no neighbor is determined to drain toward the current cell.

Until the threads encounter basin boundaries or the edge of the elevation matrix, the number of cells actively searching for neighbors at each level of recursion generally increases with successive levels. In the worse situation, from the center of a depression that curves monotonically upward in all directions, the number of active cells will increase by 8 at successive steps away from the center (Fig. 2). Although each active cell conducts its search for new members identically to and independently of every other cell, a sequential computer cannot conduct these searches simultaneously.

O'Callaghan and Mark (1984) developed a recursive, sequential procedure that models drainage flow accumulation within basis. Starting at a drainage basin outlet, the O'Callaghan and Mark procedure looks at each of the cell's 8-adjacent neighbors in turn to see if any, or all are upstream from the outlet. If so, the procedure calls itself recursively, searching from new upstream cells until a ridge or the edge of the matrix is discovered. Each call of the procedure returns the number of upstream cells for the current cell. If this value is greater than the threshold (a user-defined minimum number of upstream cells), the cell is considered to represent a stream channel.

The data-parallel approach

The drainage basin delineation and drainage flow accumulation procedures introduced here use a data-parallel approach that associates each grid cell in a DEM with a unique virtual processor (Fig. 3). The following sections describe this approach to problem solving and its application to these two procedures.

The data-parallel approach applies identical operations simultaneously to data elements distributed across an array of virtual processors (VPs) that, in turn, are mapped onto a set of parallel physical processors (PPs). If the number of VPs requested by a program is greater than the number of available PPs, each PP functions as VP/PP VPs, where VP is the number of requested VPs and PP is the number of available PPs. If $VP/PP \leqslant 1$, then each VP is represented by a unique PP. SIMD machines such as the Thinking Machines CM-2 employ large numbers of small processors (up to 65,536 bit-serial processors in the CM-2), providing lower VP/PP ratios than machines such as the CM-5 which provide smaller numbers of more powerful processors (up to 1024 Sun Microsystem SPARC processors in the CM-5). In preliminary tests of the implementations presented here, the author determined that the implementations require approximately the same time to run on a CM-2 with 8192 allocated bit-serial processors as they do on a CM-5 with 32 allocated SPARC processors. All of the timings reported in this paper refer to the CM-5 implementation.

During program execution, each VP can be active or inactive, depending upon the state of its local data or upon directives it receives from the controlling processor (the front end). If a VP is active, it executes the current instruction. If not, the instruction is ignored and the VP remains idle until the broadcast of the next instruction.

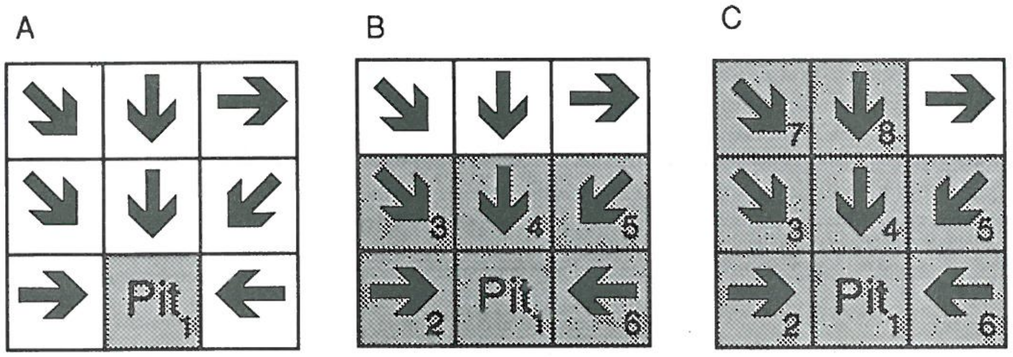

Figure 1. Marks, Dozier, and Frew (1984) procedure for determining basin membership. Arrows specify cell drainage direction. In A, pit cell searches for surrounding cells that drain toward itself. In B, pit determines that cells 2–6 are members of its own basin. In C, cells 2–6 continue recursive search for neighbors. Cell 4 determines that cells 7 and 8 drain toward itself, thus belonging to same basin. Cell in the upper-right corner drains to pit of neighboring basin.

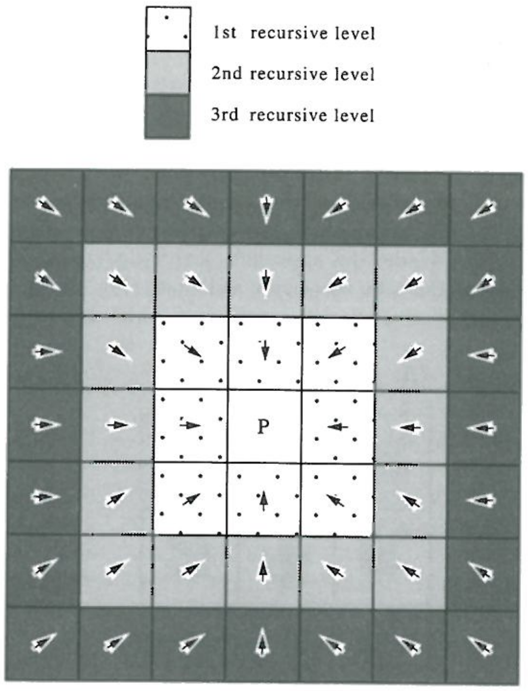

Figure 2. Worst-case performance for Marks, Dozier, and Frew (1984) procedure. In basin that curves monotonically upward in all directions from pit, number of cells actively searching for members increases by 8 on each level of recursion.

Two VPs can exchange data by passing a message over an interprocessor communication network. The time required to do this generally depends upon the number of paths that the message traverses between the two VPs. The CM-5 employs both grid and general communication networks. If the VPs are adjacent neighbors along a row, column, or diagonal in a simple grid, a message passed along the grid network requires one unit of time to travel between them. If the VPs are not adjacent, the message must travel throughout the CM-5 general communication network, a hypercube of dimension 12. On a hypercube of dimension n, messages may traverse a maximum of n steps from the source VP to the destination, requiring nt units of time, where t represents the average time required for a message to traverse a link in the hypercube. It is desirable, therefore, to pass messages over the grid network whenever possible. The data-parallel procedures described here use grid communication procedures for most of their message-passing operations.

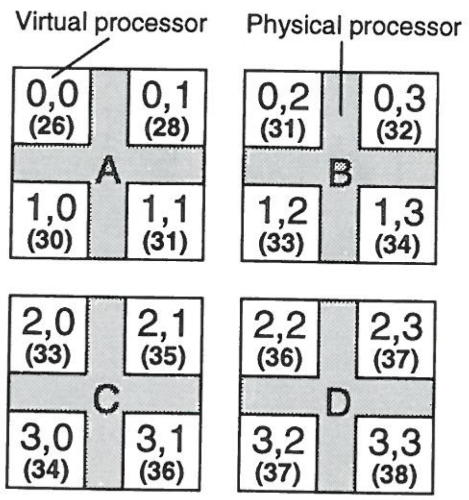

Figure 3. Mapping of VPs to PPs. In this example, 4 VPs (identified by their comma-separated row and column addresses) are mapped to each PP, identified by letters A–D, giving ratio $VP/PP = 4$. Elevation assigned to each VP is represented by number within parentheses.

A DATA-PARALLEL PROCEDURE FOR DRAINAGE BASIN BOUNDARY DELINEATION AND DRAINAGE FLOW ACCUMULATION MODELING

To delineate drainage basin boundaries and locate stream channels, the data-parallel procedure performs the following tasks in order:

(1) assignment of DEM elevation grid cells to VPs;
(2) removal of most false pits through elevation smoothing;
(3) calculation of drainage direction per grid cell;
(4) assignment of drainage basin membership to grid cells;
(5) removal of remaining false pits through a local flooding procedure;
(6) delineation of larger basin boundaries through the accumulation of smaller basins;
(7) location of stream channels through drainage flow accumulation; and
(8) calculation of hill shading values for the user window.

In Step 1, the front-end processor opens a USGS 1:24,000 series DEM and extracts grid cell elevations that fall within a user-supplied window, specified in DMS latitude and longitude values (Fig. 4). Each grid cell is mapped to a unique virtual processor in a 2-D shape, so that contiguity among adjacent grid cells is maintained among their corresponding VPs.

Step 2 applies the O'Callaghan and Mark (1984) smoothing operator to the elevation grid in a first attempt to remove false pits that arise through grid resampling procedures. To compute its smoothed elevation value, each interior cell locates and combines the elevations of its 8-adjacent neighbors with its own [Eq. (1), Fig. 5]:

$$\begin{aligned} z_{i,j} = {} & (z_{i,j} \times 0.25) \\ & + ((z_{i+1,j} + z_{i-1,j} + z_{i,j+1} + z_{i,j-1}) \times 0.125) \\ & + ((z_{i+1,j+1} + z_{i+1,j-1} + z_{i-1,j+1} + z_{i-1,j-1}) \\ & \times 0.0625). \end{aligned} \quad (1)$$

Cells on the edge of the matrix contribute values to the smoothing computations of interior cells but do

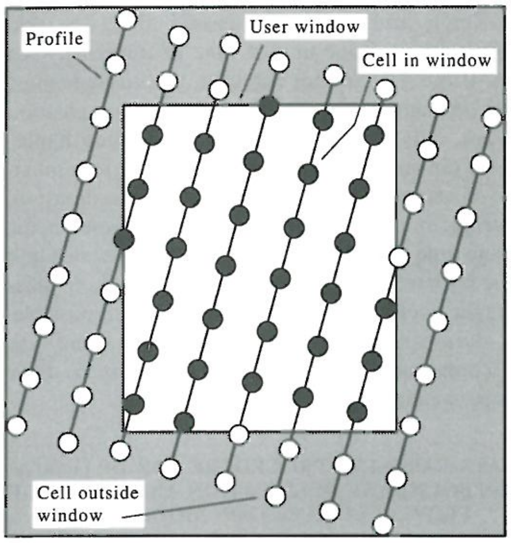

Figure 4. Extracting grid cell elevations for USGS 1:24,000 series DEM. User window, specified in latitude and longitude coordinates, defines clipping region for extracting elevations along profiles, aligned with UTM grid. Grid cells within user window are mapped to unique VPs.

not compute their own smoothed values. The user specifies the number of times to perform the smoothing operator on the matrix—about 10 iterations removes as many pits as can be removed by this technique. Because cells reference only the elevations of their adjacent neighbors, all of these references can be resolved through grid communication operations.

Step 3 determines the drainage direction of each grid cell by locating the steepest of the 8 paths from the cell to its adjacent neighbors. Slope is determined simply by dividing the elevation difference between the cell and its neighbor by their distance: 1 if the neighbor shares the same row or column; $\sqrt{2}$ if they are adjacent along a diagonal. Cells that do not have downhill neighbors are flagged as pits. In keeping with usual practice, interior pits are considered to be grid sampling artifacts and are removed in Step 5 through a local flooding procedure. Pits on the edge of the elevation matrix are assumed to drain to adjacent DEMs.

Step 4 implements an iterative search for grid cells that drain toward each pit. Two approaches were compared to highlight important efficiency issues in parallel computing. The first approach is a straightforward, but somewhat naive implementation of the Marks, Dozier, and Frew (1984) sequential algorithm in parallel. Under this approach, searching begins by activating the cells associated with pits (Fig. 6). On every iteration, each active cell queries the drainage

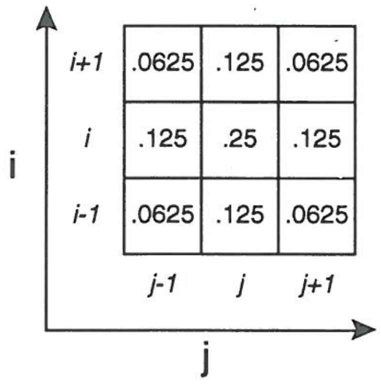

Figure 5. Weight matrix for smoothing operator in Equation (1).

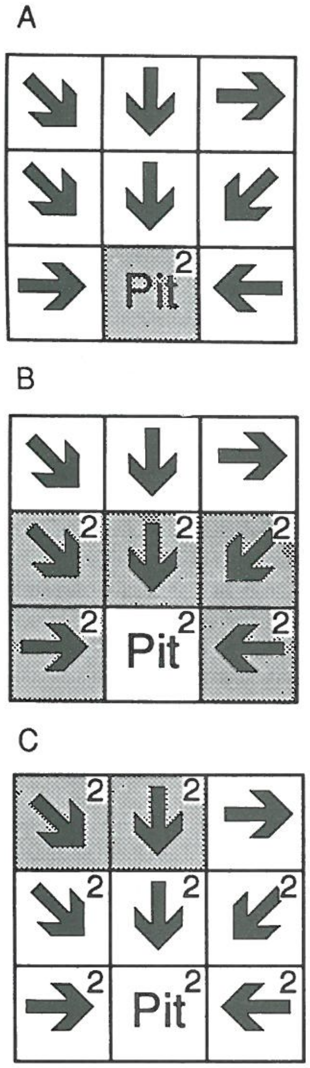

Figure 6. First approach to searching in parallel for drainage basin members. Shading indicates active cells; arrows indicate drainage direction. Basin label of pit (represented here as **2** in upper-right corner of cell) is initialized to combined value of its row and column indices. On first iteration, pit searches its adjacent neighbors for cells that drain toward itself (A), and copies its basin label to those that do. On second iteration, cells that drain into pit are activated and continue search, copying their basin labels to their uphill neighbors (B). On third iteration, cells discovered as members on second iteration now are active and continue search (C). Procedure will end when both active cells fail to locate uphill neighbors.

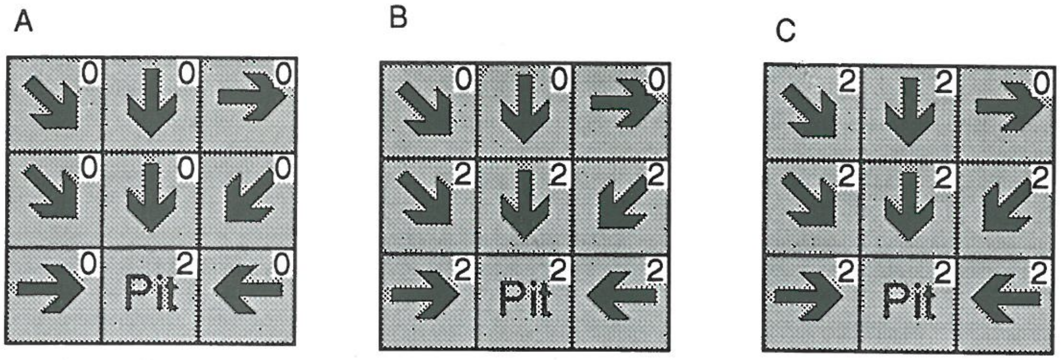

Figure 7. Second approach to searching in parallel for drainage basin members. All cells are active on each iteration. Pits are initialized with basin labels as in Figure 6; labels of all other cells are initialized to 0. On first iteration, each cell compares its label to that of its downhill neighbor, pointed to by its drainage direction (A). If label of downhill neighbor is nonzero, cell copies downhill label to itself. On second iteration, all cells remain active. On this iteration, the upper-left and upper-center cells are only remaining cells with 0 labels that locates downhill neighbors with nonzero labels (B). At end of third iteration, no cells with 0 labels remain and procedure stops (C).

direction of its 8-adjacent neighbors. Any cells that drain toward itself are in the basin and are activated. The cell that initiated the search copies its basin label (a hashed value of the row and column address of the pit) to the new members and then deactivates itself. The procedure iterates, spreading this pattern of activation across the processor array until all cells have been assigned to a drainage basin.

Two characteristics of the first approach are likely to slow the execution of a SIMD implementation: (1) each active cell must query all of its neighbors for their drainage directions although most cells will locate only one or two uphill neighbors; and (2) a new processor context must be determined on each iteration. This is an expensive operation on the CM-5.

Under the first approach, cells on the edge of an expanding basin "push" their labels uphill on each iteration. The second approach recasts the problem by having cells "pull" their labels up from their downhill neighbors (Fig. 7). In this approach, pits initialize their basin labels using the same procedure as the first approach. All other cells in the interior of the matrix initialize their basin labels to zero. On each iteration, every cell compares the basin label of its downhill neighbor (pointed to by its drainage direction) with its current basin label using a bitwise OR operator. If a cell determines that the basin label of its downhill neighbor is zero, the basin label of the cell does not change; otherwise, it takes on the label of the downhill cell. Nonzero labels propagate uphill by one cell on each iteration until all cells have nonzero labels.

The second approach has two important advantages over the first: (1) all cells (excluding pits) have one and only one downhill neighbor, reducing the number of neighbor queries by a factor of 8 over the first approach; and (2) the second approach keeps all cells active over the duration of the search, eliminating the need to set the processor context on each iteration. Although the second approach requires the same number of iterations to complete as the first approach, each iteration should take less time to execute.

The shape, as well as the size of the drainage basin determines the number of iterations required to complete Step 4. In a square basin with a central pit (such as the one depicted in Fig. 2), the expected number of iterations is proportional to $W/2$, where W is the width of the basin in grid cells. Figure 8 shows that a narrow basin having a small number of VPs active

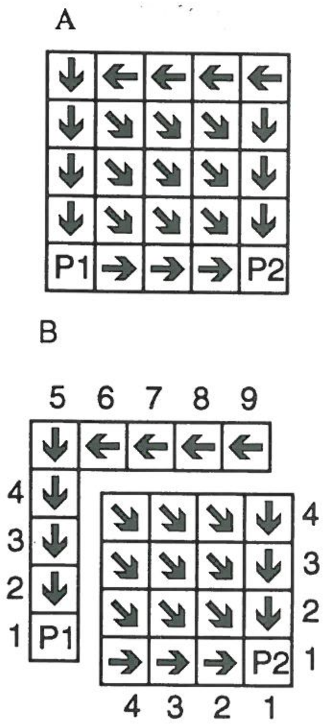

Figure 8. Sensitivity of SIMD drainage basin procedure to basin shape. Assuming drainage pattern in A, pit P1 will require 9 iterations to locate its members; pit P2 will require only 4 (B).

over each iteration may require a larger number of iterations to label than would a larger but more compact basin with many active VPs.

Using a procedure recommended by O'Callaghan and Mark (1984) to remove interior pits, Step 5 raises the elevation of each pit until it drains to a lower basin (Fig. 9). To do so, the pour point on the basin boundary first must be determined. Each cell within a single basin compares its basin label to its neighbor's. If the labels differ, then the cell is on the drainage basin boundary and is activated. Each active cell compares its elevation to the minimum boundary elevation determined by a scanning function. If its elevation is higher than the minimum or if the neighboring basin has a higher pit than its own, the cell deactivates itself. If more than one cell shares the minimum value, all cells but the one closest to the pit of the lower basin are arbitrarily deactivated. The procedure notes the position of the remaining active cell (the pour point) by copying its hashed row and column address to the other cells in the basin.

All cells in the basin with elevations below the pour point are reset to the pour point's elevation and their drainage directions are reset to an undefined value. The drainage directions of the "flooded" cells are then reoriented to the pour point using an iterative procedure. Each pour point tests the drainage directions of its 8 adjacent neighbors. Neighbors having the same basin label as the pour point and having an undefined drainage direction are reset to drain toward the pour point. These cells are activated

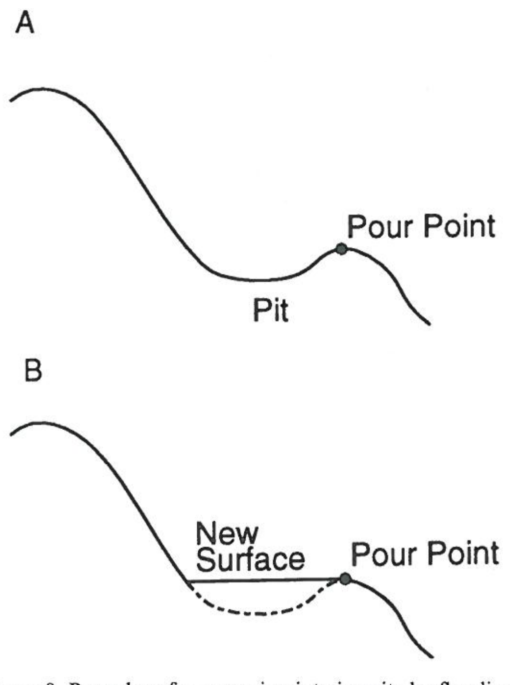

Figure 9. Procedure for removing interior pits by flooding: A, interior pit and its pour point (lowest point on basin's boundary with lower basin); B, elevations of all cells below pour point are made equal to pour point's elevation. Drainage directions of flooded cells are left undefined.

subsequently on the next iteration and examine their neighbors using the same criteria. The procedure stops if no cell has an undefined drainage direction.

The drainage redirection procedure employs a pattern of grid cell activation similar to that of the first approach to Step 4. Step 5 cannot emulate the second approach to Step 4 because no established drainage direction exists for each cell. It is expected, therefore, that Step 5 will incur the performance penalties inherent to the first approach to Step 4.

At the end of Step 5, the pour point is set to drain toward a neighboring cell such that the following criteria are met:

(1) the pour point must drain outside of its own basin;
(2) the exterior basin must have a pit lower than the current basin's pour point; and
(3) the pour point must drain to the lowest of its neighboring cells in the exterior basin.

Once the drainage direction of a pour point is established, it becomes a member of the basin into which it drains. Step 6 changes its basin label and those of its uphill neighbors to reflect this change. Relabeling begins with the lowest basin containing a redirected pour point and continues to the highest such basin. For each basin in turn, the pour point first acquires the basin label of its downhill neighbor. The other cells in the drainage basin of the pour point then are activated and copy the new basin label from the pour point to themselves (Fig. 10). Because most of the cells in the basin are not 8-adjacent neighbors of the pour point, copying is done over the hypercube network.

Step 7 locates stream channels through an iterative procedure that counts the number of units of "water" that drain through each cell (Fig. 11). All cells (except pits) are initialized with one unit. Emulating the second approach to Step 4, every nonpit cell sends all of its units to its downhill neighbor, pointed to by its drainage direction, on each iteration. The sending cell also increments the uphill cell counter of its downhill neighbor by the number of units that it sent. Processing stops when only the pit cells have nonzero water supplies, indicating that all of the water has drained through the system. At this point, the value of the uphill cell counter for each cell will equal the total number of units that have drained through it. The uphill cell values for all cells then are normalized to the range 0.0–1.0 and compared to the user's tolerance value, specified within the same range. A cell is considered to represent a stream channel if the normalized value of its uphill cell counter is greater than the user-specified threshold.

Step 8 computes a hill shaded image of the cells within the user window using an algorithm presented by Horn (1981). A cell calculates its slope and aspect solely with respect to its own elevation and those of its 8-adjacent neighbors. Then it compares its slope

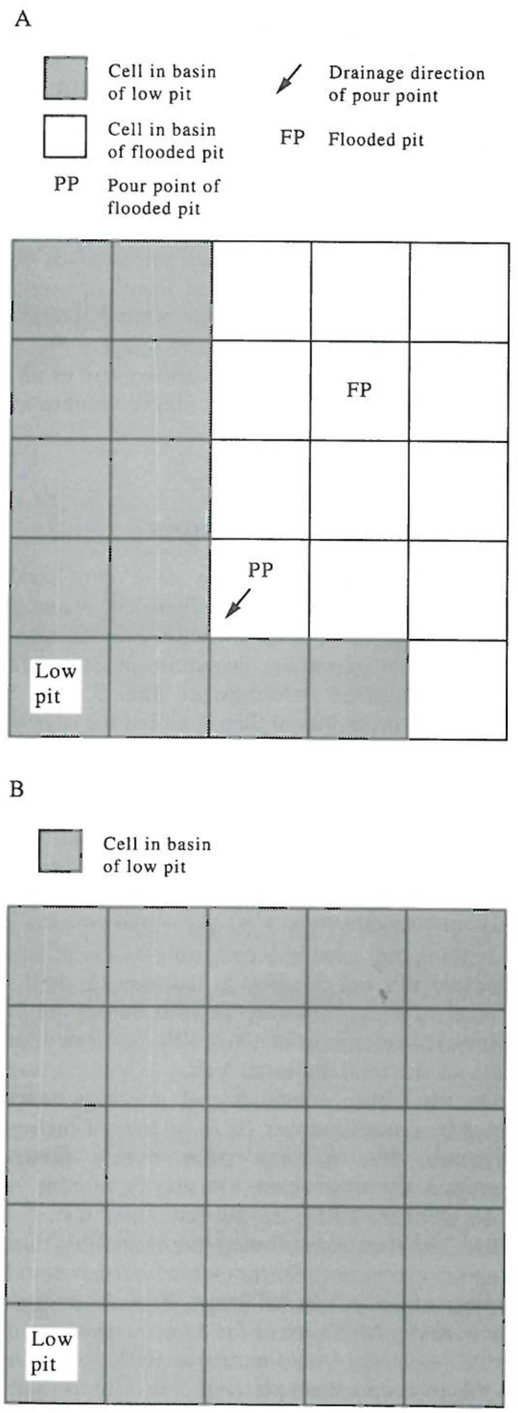
A
Cell in basin of low pit
Drainage direction of pour point
Cell in basin of flooded pit
FP Flooded pit
PP Pour point of flooded pit
FP
PP
Low pit
B
Cell in basin of low pit
Low pit

Figure 10. Merging drainage basins through pour points: A, pour point (PP) of basin of flooded pit acquires label of its downhill neighbor in basin of lower pit; B, new label of PP has been copied to all cells in its former basin.

and aspect to the azimuth and elevation of the illumination source (specified by the user) to calculate the illumination value for the cell (varying within the range 0–255). All of the cells (except those on the edge of the matrix) are active during the entire procedure, maintaining a high level of efficiency throughout its execution. The illumination values can be applied directly to create a hill-shaded map or as a value overlay to hues representing the basin delineation and drainage flow accumulation image layers, selected automatically by a map coloring subprocedure.

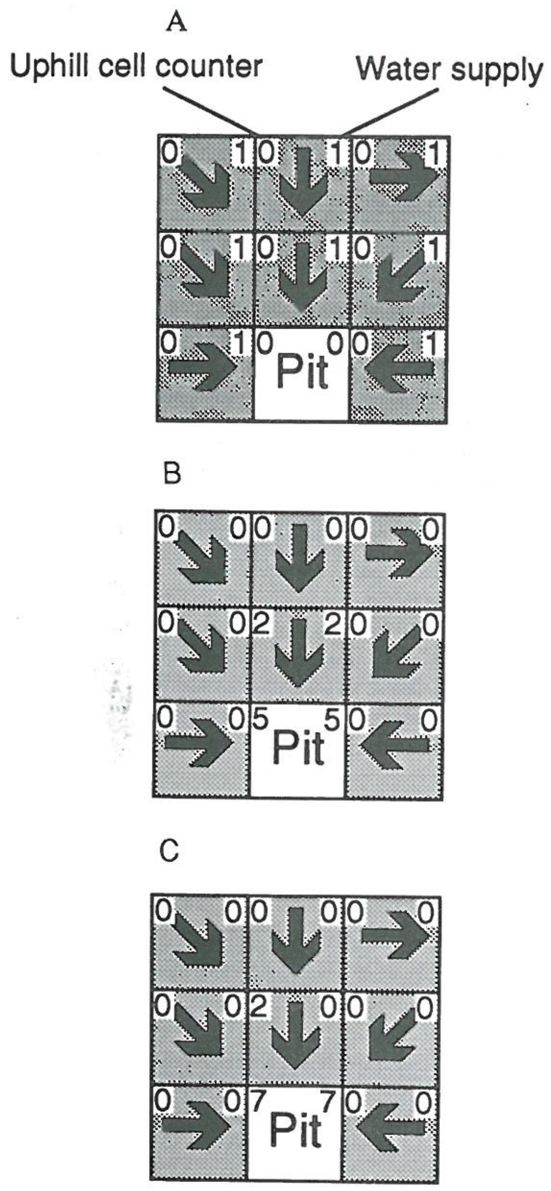

Figure 11. Modeling drainge flow accumulation. All cells (except pits) are active on each iteration and are initialized with 1 unit of water and 0 uphill neighbors. On every iteration, each nonpit cell sends its entire water supply to its downhill neighbor. Each cell (including the pit) then increments its uphill neighbor counter by its current water supply. On first iteration, every cell but pit sends 1 unit to its downhill neighbor (A). On second iteration, only center and pit cells have nonzero water supplies and uphill neighbor counters (B). Center cell drains its water supply to pit and increments pit's uphill cell counter by 2 (C). At beginning of third iteration, pit is only remaining cell with water and accumulation procedure ends. Any cell having normalized uphill cell counter greater than user's tolerance value is considered to represent stream channel.

IMPLEMENTATION DETAILS

The CM-5 is a product of Thinking Machines Inc. The current implementations were run on a CM-5 configured with 32 physical processors at the Northeast Parallel Architecture Center (NPAC) at Syracuse University.

The procedures described in this paper were implemented in the C* programming language, providing data-parallel extensions to ANSI C. ANSI C code will compile, link, and run under the C* programming environment without modification. The language supports both grid and general (hypercube) message-passing procedures through left-indexing, a syntactical construction that references VPs in multidimensional grids (from 1 to 31 dimensions) with array indices written within square brackets to the left of the processor array identifier and increasing in dimension toward the right. For the 2-D grids employed in this project, the left-most index represents the VP row address and the right-most represents the column address.

Example 1 uses C* left-addressing notation to implement the smoothing operator in Equation (1). In this notation, a dot is an alias for the index of a VP along the dimension referenced by the enclosing brackets. In Example 1, each active VP (represented by z on the left-hand side of the assignment operator) combines its local z value with those of its 8-adjacent neighbors (selected by adding to subtracting 1 to its index along one or both grid axes) and replaces the original local z value with the result:

Example 1,

$$\begin{aligned} z = {} & (z*0.25) + (([\cdot + 1][\cdot]z + [\cdot - 1][\cdot]z + [\cdot][\cdot + 1]z \\ & + [\cdot][\cdot - 1]z)*0.125) + (([\cdot + 1][1\cdot + 1]z \\ & + [\cdot + 1][\cdot - 1]z + [\cdot - 1][\cdot + 1]z \\ & + [\cdot - 1][\cdot - 1]z)*0.0625). \end{aligned}$$

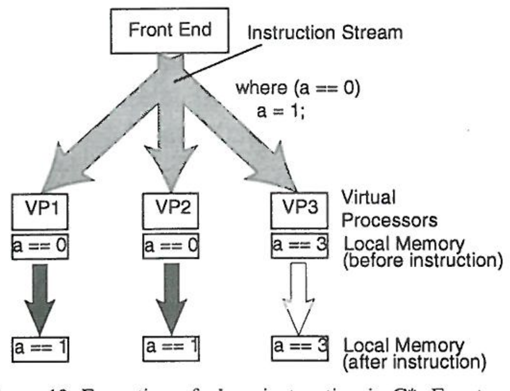

Figure 12. Execution of **where** instruction in C*. Front-end processor broadcasts instruction stream, carrying **where** instruction, to virtual processors (VP_n). Each VP sets its value of a to 1 if its current value of a is 0. In this example, processors VP_1 and VP_2 carry out instruction; VP_3 remains idle until broadcast of next instruction from front-end.

In C*, a "shape" specifies a set of processors that share a common data structure. The **where** operator changes the processor context within the shape based on the state of each processor's local (parallel) data. **where** operates similar to a parallel version of an **if** statement in standard C; code is executed on a VP if the expression for the **where** operator evaluates to true on its local data (Fig. 12).

The implementation generates performance statistics and produces encapsulated PostScript output files that represent the drainage basin boundaries, stream channels, and hill-shaded imagery for the window. The user can select to display any or all of these graphic layers within a single encapsulated PostScript image.

EFFICIENCY ISSUES

The computational efficiency of a data-parallel program on a CM-5 can be evaluated by examining the percentage of VPs that are active over its running time. For the smoothing procedure of Step 2, the drainage direction procedure of Step 3, and the hill-shading procedure of Step 8, all but the edge VPs are active at all times. In contrast, the first approach to basin delineation in Step 4 and the drainage redirection procedure of Step 5 have lower percentages of active VPs over their running times. When Step 4 begins, only pits are activated. Assuming a DEM of 300 columns and 400 rows mapped to a 2-D array of 120,000 VPs with 50 pits remaining after the smoothing operation in Step 2, only 0.04% of all the allocated VPs will be active at that time. Even if the number of active VPs rises to 1000 during the procedure, the percentage of active VPs increases to only 0.8% of the total allocated VPs.

On the CM-5, computational power is concentrated in a small number (1024 or less) of high-performance PPs. At any point during program execution, the set of active VPs may be running on a small number of PPs, leaving the others idle. If the active VPs were redistributed across the PPs during program execution, efficiency would be high until the number of active VPs fell below the number of PPs. For a small CM-5, such as the 32-node installation at NPAC, efficiency would remain at 100% for all steps in the preceding example (Fig. 13). Unfortunately, the current operating system software on the CM-5 does not provide dynamic load-balancing procedures. As a result, some of the physical processors are likely to remain idle at times during the execution of the SIMD implementation.

Each PP must cycle through its set of VPs sequentially. It is expected, therefore, that the running time of the entire implementation will increase linearly with the ratio VP/PP. This was tested by running the implementation for windows of increasing dimension to determine the relationship between execution time and the ratio of VP/PP.

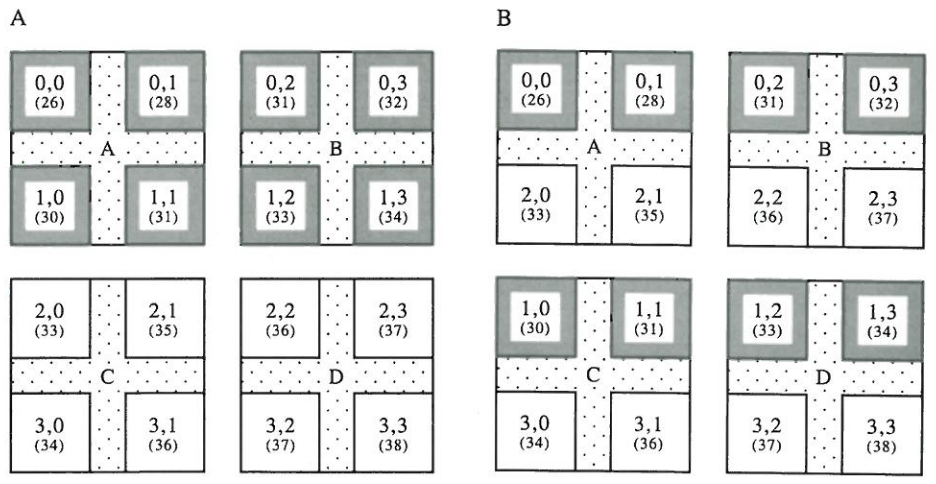

Figure 13. Balancing processing loads across PPs. Shaded VPs are active; unshaded VPs are inactive: A, PPs C and D are idle, where A and B each support 4 active VPs; B, active and inactive VPs have been redistributed evenly across PPs.

OPTIMIZATION DETAILS

The CM-5 is a relatively new computer and much of its system software, including the C* compiler, is under development. To gain acceptable performance values for the SIMD implementation, it was necessary to apply some optimization techniques that normally would be performed by a compiler. The techniques that are described here reduced execution times by about one-third for nonoptimized implementations.

Under the current release of the C* compiler, the object code produced from **where** instructions executes slowly. Examples 2 and 3 compare the structure

Table 1. Elapsed time in CPU sec for all steps in SIMD and sequential implementations for 4 user windows; both SIMD approaches for Step 4 are included

	Elapsed Time (CPU seconds)			
Step	1km X 1km	2km X 2km	4km X 4km	8km X 8km
1) Assign DEM grid cells to VPs				
SIMD	2	3	5	10
Sequential	2	3	5	10
2) Smooth elevations, 10 passes				
SIMD	1	1	1	1
Sequential	1	1	2	8
3) Calculate drainage direction				
SIMD	1	1	1	1
Sequential	1	1	1	3
4) Delineate basin boundaries				
SIMD, Approach 1	2	3	15	62
SIMD, Approach 2	1	1	1	2
Sequential	1	1	2	6
5) Remove pits through flooding				
SIMD	1	1	1	5
Sequential	1	1	2	5
6) Merge basins				
SIMD	1	1	1	1
Sequential	1	1	1	6
7) Accumulate drainage flow				
SIMD	1	1	1	2
Sequential	1	1	1	6
8) Calculate hill shading values				
SIMD	1	1	1	1
Sequential	1	1	1	3
Total Sequential Time:	**9**	**10**	**15**	**47**
Total SIMD Time (with Step 4, approach 1):	**10**	**12**	**26**	**83**
Total SIMD Time (with Step 4, approach 2):	**9**	**10**	**12**	**23**

Table 2. Overall execution times for SIMD and sequential implementations for all user windows; ratio *VP*/*PP* also is provided for each window, based upon 32 installed PPs in NPAC CM-5

	Window Size (km by km)							
	1 X 1	2 X 2	3 X 3	4 X 4	5 X 5	6 X 6	7 X 7	8 X 8
Rows	33	66	100	133	166	200	233	266
Columns	33	67	100	133	167	200	233	267
VPs	1,089	4,422	10,000	17,689	27,722	40,000	54,289	71,022
VP/PP	34.0	138.2	312.5	552.8	866.3	1,250.0	1,696.5	2,219.4
Total Time, Sequential	9	10	13	15	18	28	35	47
Total Time, Approach 1	10	12	16	26	35	44	54	83
Total Time, Approach 2	9	10	11	12	15	15	17	23

Slope of Least Squares Linear Trend of Time vs. VP/PP (SIMD Approach 2)	0.006 seconds increase per additional virtual processor

and result of a **where** instruction with those of a simple Boolean test. Both examples perform the same operations: for all processors having copies of *a* equal to *b*, set the copy of *a* to 0 and *b* to 1. Example 2 does this explicitly with the **where** operator; Example 3 uses a Boolean test. In Example 3, the comma-separated list of operations on the right-hand side of the logical AND will be performed only if the expression on the left-hand side evaluates to true. The author has determined that this method produces more efficient executable code than the first:

Example 2,

```
where (a == b){
    a = 0;
    b = 1;
}
```

and

Example 3,

$$(a == b)\&\&(a = 0, b = 1).$$

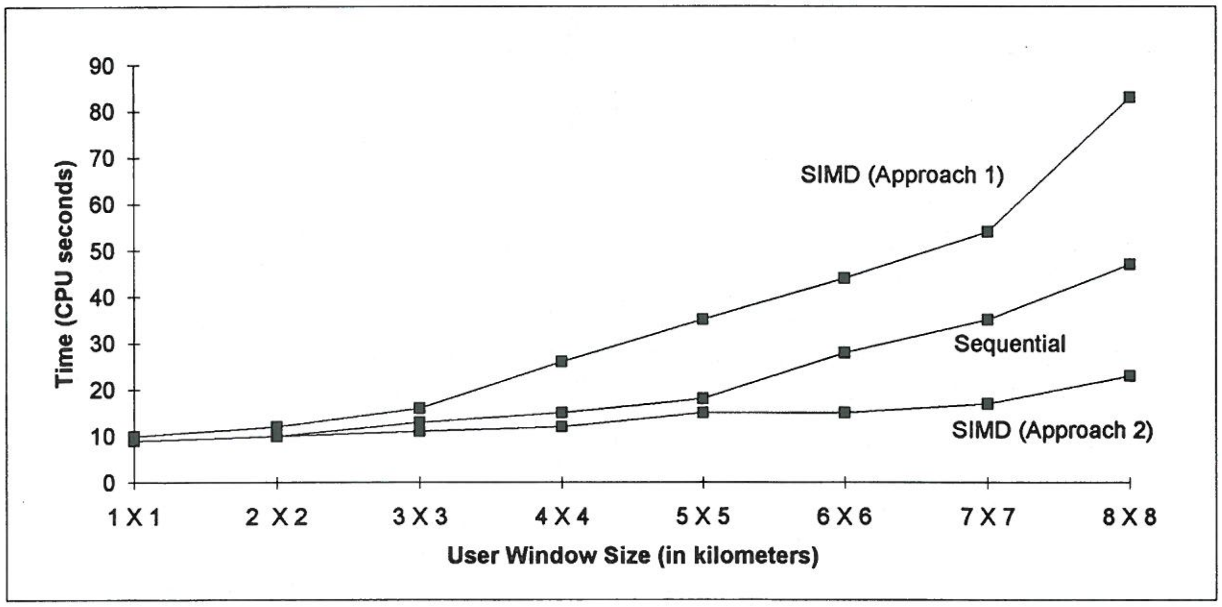

Figure 14. Performance curves for sequential and SIMD implementations. Two SIMD implementations are identified by their approach to Step 4.

(Figure 15 opposite)

Figure 15. Screen photograph of PostScript image of Ft. Douglas DEM showing hill-shading, drainage basin delineation, and predicted locations of stream channels for user window 10 km in easting × 12 km in northing. North is at top of image. Basins are designated by varying hues, hill-shading by value. Pixels on stream channels are dark blue. Scale of original PostScript image was 1:100,000.

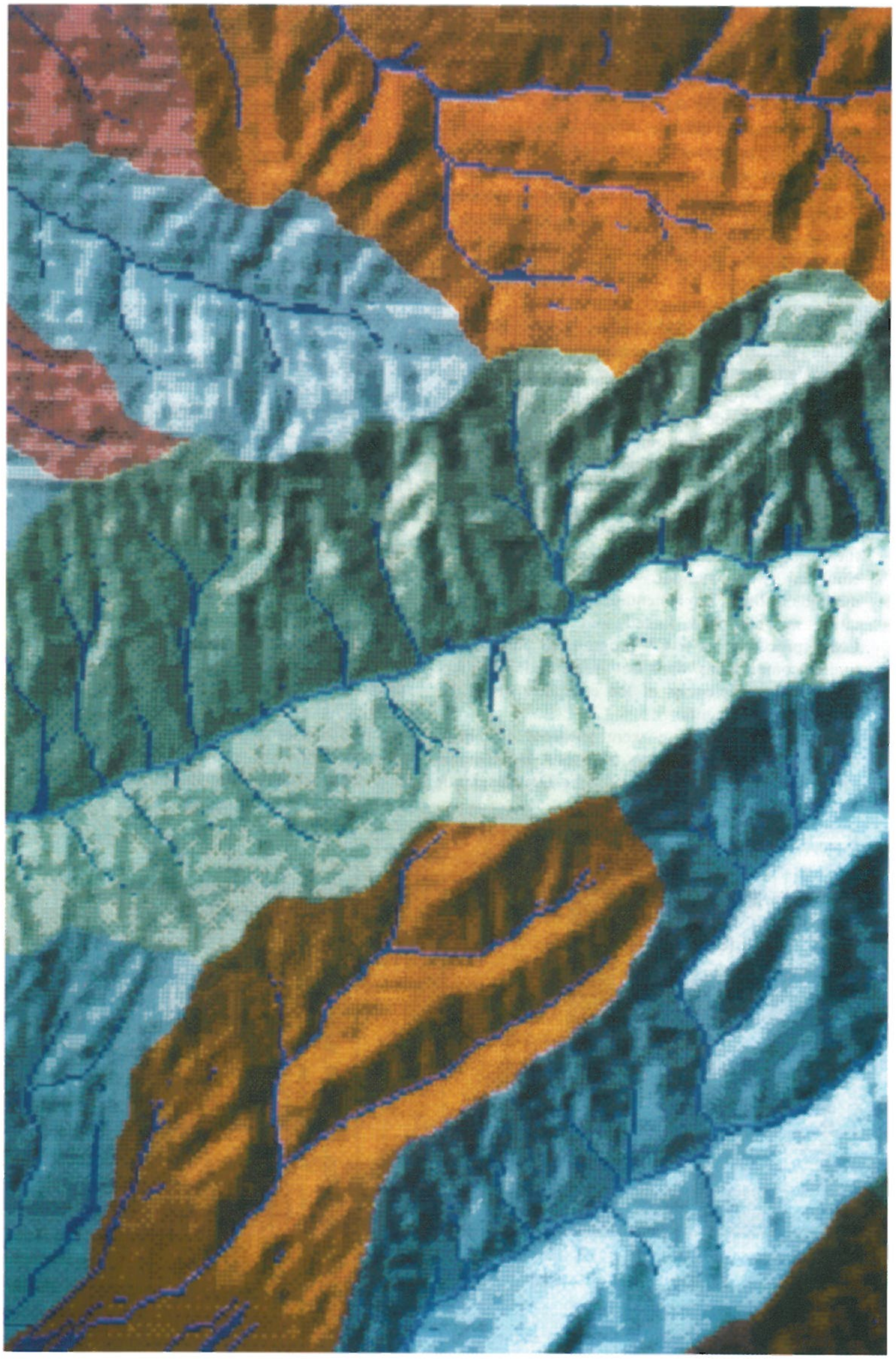

Figure 15—*caption opposite.*

It is important to allocate no more than the number of VPs actually required to represent the cells in the window. Even if a VP is inactive, the time required to check its state of activation is significant under the current release of the operating system. Fortunately, processor shapes can be declared with any number of elements along each dimension at runtime, much as memory is allocated dynamically with the malloc() family of procedures in standard C. The current implementation allocates a 2-D array of VPs at runtime based upon the size of the user's window.

RESULTS

The performance characteristics of the SIMD implementation were measured and compared to those of a sequential implementation built on the same source code base. The author ported the sequential implementation, written in ANSI C, directly from the C* implementation. Code fragments that do not make explicit use of parallel operators in C* (including all of the operations in Step 1) are shared by both implementations. Code fragments in the C* implementation that use parallel operators are emulated with standard iterative or recursive methods in the C implementation. The C implementation was compiled, linked, and run on a single CM-5 processor as a C* program to ensure that timing comparisons were not influenced by differences in the efficiencies of the compilers or of the runtime environments.

Performance statistics were gathered for both implementations for a series of 8 tests, based upon square user windows on the USGS Ft. Douglas, Utah 1:24,000 DEM, ranging from 1 × 1 km to 8 × 8 km in 1-km intervals in eastings and northings. Because working profiling procedures for the CM-5 were not available when this paper was written, both implementations generated their own performance statistics through calls to system level, load-independent timing calls. Although not as precise as profiling data (the smallest reported interval is the CPU sec), these statistics yet provide useful views of the performance behavior of the implementations. The window sizes, overall execution times, and detailed

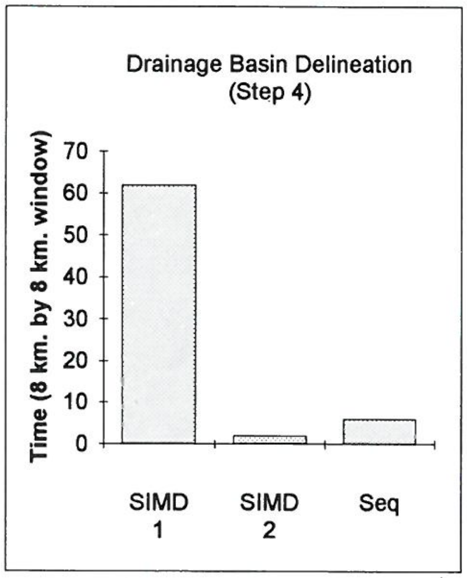

Figure 17. Performance statistics for sequential implementation and two SIMD implementations (approaches 1 and 2) of Step 4 for 8 × 8 km, user window.

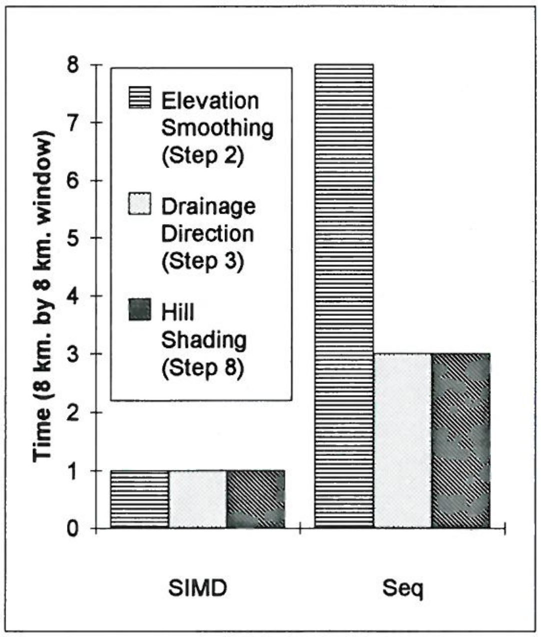

Figure 16. Performance statistics for SIMD and sequential implementations of Steps 2, 3, and 8 for 8 × 8 km user window.

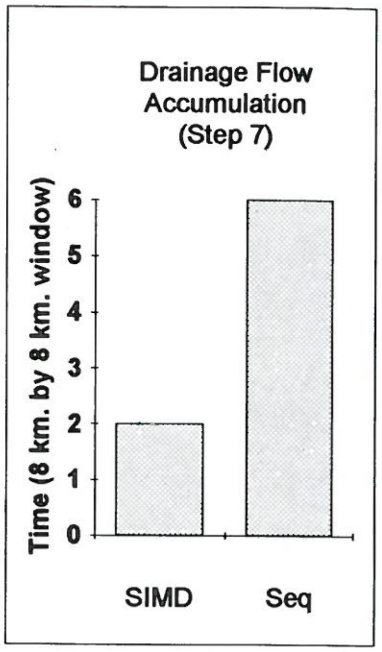

Figure 18. Performance statistics for SIMD and sequential implementations of Step 7 for 8 × 8 km user window.

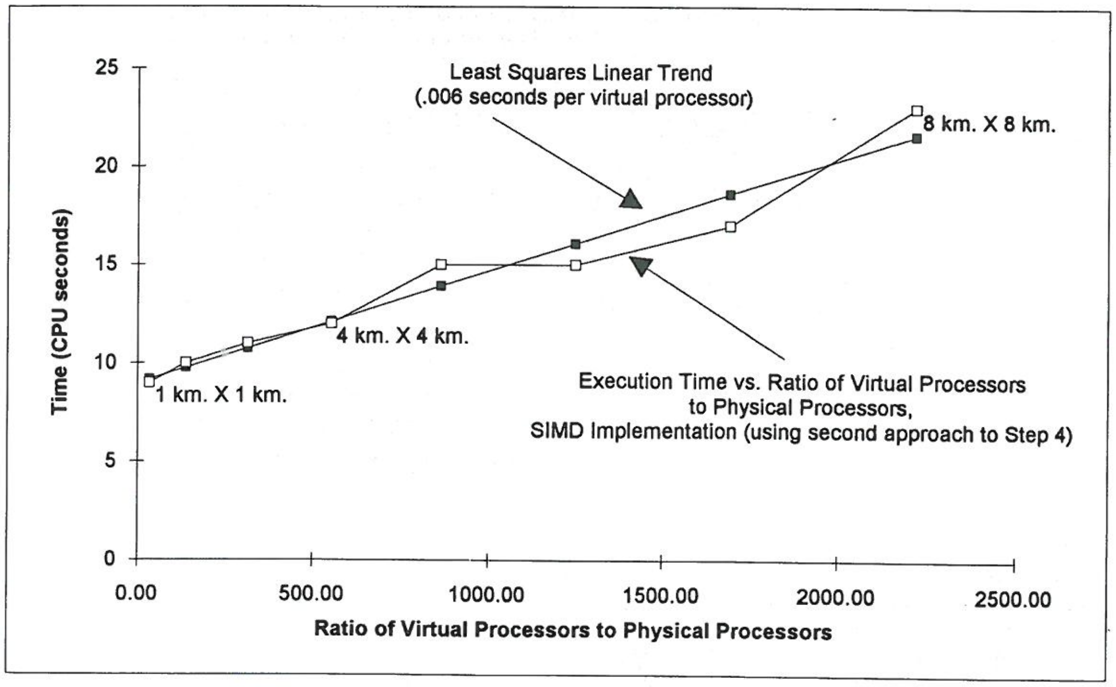

Figure 19. Relationship between ratio *VP/PP* and execution times for SIMD implementation, using second approach to Step 4. Least-squares linear trend for 8 user windows increases at rate of 0.006 CPU sec for each additional VP supported by PP.

performance statistics by procedural step (and by approach for Step 4) are summarized for selected windows in Table 1. Overall execution times for all user windows are included in Table 2 and summarized in Figure 14. Timing calls returning 0 sec were rounded upward to 1 sec in Table 1. The timings for Step 1 are identical for both implementations because they share the same source code.

Figure 15 shows the graphic output of the SIMD implementation for a 10 × 12 km window on the Ft. Douglas DEM with North at the top of the image. Drainage basins are represented by sets of contiguous pixels with constant hue. Variations in the value of each pixel represent the slope and aspect of its corresponding local surface relative to a light source illuminating the image from the northwest at an elevation of 45°. Pixels representing stream channels are dark blue. For all user windows, the graphic output of the sequential implementation is identical to that of the SIMD implementation.

As expected, steps that kept all of the interior VPs active for their duration (Steps 2, 3, and 8) had the lowest execution times for the SIMD implementation (Fig. 16). The granularity of the system timer was unable to capture execution times of < 1 sec, making it seem that the three SIMD steps required the same amount of time (rounded from 0 to 1 CPU sec for Table 1) to complete for all four windows. With regard to the rounded times in Table 1, the SIMD implementation of Step 2 seemed to attain the largest speed-up, executing 8 times faster than the sequential implementation for 10 smoothing passes on the 8 × 8 km window. It is likely, however, that all three steps attained speed-ups that were better than those suggested by the worst-case values reported in Table 1 for the SIMD implementation.

Figure 17 presents the execution times for the sequential implementation and the two SIMD implementations (approaches 1 and 2) of Step 4 running on the 8 × 8 km window. The implementation of the first SIMD approach to Step 4 takes over 10 times longer to complete than its sequential counterpart and about 30 times longer than the second approach. Clearly, by imitating the sequential procedure, the first SIMD approach performs an unnecessarily large number of interprocessor communication and context setting operations. Rewritten from the ground up as a SIMD procedure, the second SIMD approach attains a speed-up factor of 3 over the sequential approach. Using a similar execution strategy, the SIMD implementation of Step 7 achieved a speed-up factor of 3 over the sequential implementation (Fig. 18).

As expected, the SIMD implementation of Step 5, modeled on the first approach to Step 4, performed poorly, offering no speed-up over the sequential implementation for all but the 4 × 4 km user window. Step 5 required less time to complete than the first approach to Step 4 only because it operated on fewer cells in each basin (those that were below the pour point).

The SIMD implementation of Step 6, on the other hand, performed 6 times faster than the sequential implementation on the 8 × 8 km window, even though it processed only one basin on each iteration and utilized the hypercube network for message-passing operations. It achieved its speed-up by limiting each VP to receiving no more than one message during the entire step. This result further emphasizes the importance of minimizing interprocessor communication requirements in the design of SIMD procedures.

Table 2 and Figure 19 show that execution times for the entire SIMD implementation (using the second approach to Step 4) increased linearly at an overall rate of 0.006 CPU sec for each additional VP supported by a PP. With increasing numbers of PPs, the implementation should experience corresponding decreases in execution times.

DISCUSSION AND CONCLUSION

The parallel procedures for drainage basin analysis introduced here exhibit three important characteristics:

(1) SIMD implementations of the steps that operate locally on grid cells (Steps 2, 3, and 8) have the lowest execution times;

(2) SIMD implementations of the steps that propagate values across the VP array (Steps 6, 7, and the second approach to Step 4) have low execution times compared to their sequential counterparts; and

(3) execution times for the SIMD implementation (using the second approach to Step 4) increase linearly with the ratio of VP/PP.

To decrease execution times for these procedures further, it is necessary to increase the efficiency of Steps 4, 5, and 7. Each of these steps requires that values propagate away from a starting location in an iterative manner, limiting the number of contributing VPs on any iteration to a small percentage of the total set. Each step also allows imbalances to occur in the processing load carried by the PPs.

To distribute the load more evenly across the physical processors, the entire collection of procedures could be recast for a MIMD environment. Under this approach, drainage basins would be assigned to PPs as they became available. All of the PPs would run concurrent copies of the sequential implementation on their basins, requesting new basins upon their completion. Processors would remain busy until no unprocessed basins remained. At that point, processors would complete their final basins asynchronously and become inactive one by one. The overall efficiency of the MIMD procedures would increase with the ratio of drainage basins to PPs, thereby increasing the proportion of the time that the program executes on the full set of PPs.

The author currently is developing a MIMD implementation for the CM-5 and will compare its execution characteristics to those of the SIMD procedures described here. It is hoped that through the continued refinement of these and other parallel procedures for applications in cartography and GISs, along with improvements in parallel hardware and software environments, that parallel computers will become increasingly important platforms for development in these fields.

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

REFERENCES

Ding, Y., Densham, P. J., and Armstrong, M. P., 1992, Parallel processing for network analysis: decomposing shortest path algorithms for MIMD computers, *in* Proc. 5th Intern. Symp. Spatial Data Handling, Charleston, North Carolina, p. 682–691.

Franklin, W. R., Narayanaswami, C., Kankanhalli, M., Sun, D., Zhou, M., and Wu, P. YF., 1989, Uniform grids: a technique for intersection detection on serial and parallel machines: Proc. Autocarto 9, Baltimore, Maryland, p. 100–109.

Hopkins, S., Healey, R. G., and Waugh, T. C., 1992, Algorithm scalability for line intersection detection in parallel polygon overlay, *in* Proc. 5th Intern. Symp. Spatial Data Handling, Charleston, North Carolina, p. 210–218.

Horn, B. K. P., 1981, Hill shading and the reflectance map: Proc. IEEE, v. 69, no. 1, p. 14–47.

Marks, D., Dozier, J., and Frew, J., 1984, Automated basin delineation from digital elevation data: Geo-Processing, v. 2, no. 3, p. 299–311.

Mills, K., Fox, G., and Heimbach, R., 1992, Implementing an intervisibility analysis model on a parallel computing system: Computers & Geosciences, v. 18, no. 8, p. 1047–1054.

Mower, J., 1993, Automated feature and name placement on parallel computers: Cartography and Geographic Information Systems, v. 20, no. 2, p. 69–82.

Mower, J., 1994, Matching parallel algorithms and architectures: the case of line simplification: in preparation.

O'Callaghan, J. F., and Mark, D. M., 1984, The extraction of drainage networks from digital elevation data: Computer Vision, Graphics, and Image Processing, v. 28, no. 3, p. 323–344.