Experiments in the Identification of Terrain Features Using a PC-Based Parallel Computer

Demetrius-Kleanthis D. Rokos and Marc P. Armstrong

Abstract

This paper describes the process through which an existing serial algorithm that classifies terrain features in a digital elevation model (DEM) is recast into a version that executes in a parallel computer environment. The transformation process is guided by a formal analysis of dependency relationships that exist among the different steps in the algorithm. The resulting program is tested using a small parallel computer system. The results demonstrate that run times are reduced, and that the processors are used efficiently. The general approach to the development and implementation of parallel algorithms that is presented in this paper is extensible to a wide range of geographical computing problems.

Introduction

Researchers frequently use computer models to improve their understanding of spatial processes. Such models often start as relatively crude abstractions and are increasingly refined through the addition of enhanced procedures that are designed to capture new details of the processes being studied. This quest for realism, however, often comes with a penalty, because detailed models that handle numerous intricate interactions in spatial systems normally cause processing times to increase. Thus, the computational burden imposed by model improvements, ironically, can preclude interactive, exploratory modeling, thereby decreasing the usefulness of spatial models in research and decision-making.

One apparently simple way to overcome this problem is to use faster computers. While there is considerable room for debate, many researchers (e.g., Stone and Cocke, 1991; Flynn and Rudd, 1996) have suggested that the physical limit of processing speed for microprocessors is approaching. Consequently, computer manufacturers and researchers are turning to parallel architectures that integrate and harness the power of several inexpensive processors to provide computer environments that are designed to handle computationally intensive problems.

At present, most spatial data handling algorithms have been developed for sequential computing environments. Though the application of parallel programming to a problem may require the development of new algorithms and code, in many cases existing algorithms can be recoded and structured in a form that is compatible with the architecture of a parallel computer system. The purpose of this paper is to describe this process of translation, using as an illustration an existing sequential algorithm that extracts topographic features from a digital elevation matrix (Bennett and Armstrong, 1996). The general process of converting this algorithm to a parallel environment is extensible to a large number of models that operate on gridded spatial data.

The paper is structured as follows: First, basic principles of parallel processing and parallel algorithm design are briefly discussed. Next, the terrain feature extraction algorithm is described. The algorithm is implemented in a parallel processing environment and is evaluated along three dimensions that are important to the efficient use of parallel computer resources: the organization of the data required by each task, the size of the task allocated to each processor, and the number of processors used.

Parallel Processing

Algorithms that can be efficiently implemented in parallel are decomposable into relatively independent code segments that are allocated to different processors for execution. Because many spatial models treat geographic space as a collection of interacting spatial units, there is, *a priori*, a strong indication that they can be decomposed into relatively independent parts. In practice, this is not always easily accomplished, however, because the tasks identified in the problem decomposition process may exhibit dependencies that preclude parallel execution. In the following sections we examine several basic principles of parallel algorithm design. The discussion focuses on program granularity, problem decomposition strategies as developed in different parallel programming paradigms, and the main categories of parallel programming models.

Granularity of Parallelism

Carriero and Gelernter (1990: p. 1) assert that problem solving is naturally parallel and that purely sequential problems should be considered an "anomalous restriction." Typically, however, there is more than one way to decompose a problem, and the suitability of a particular strategy often hinges on the selection of an appropriate granularity of parallelism. Parallel algorithms can be classified into three categories with respect to the granularity of individual tasks (Polychronopoulos, 1988):

- *Fine* grained algorithms have individual tasks equivalent to basic code blocks (single instructions or small loops).

Department of Geography and Program in Applied Mathematical and Computational Sciences, The University of Iowa, 316 Jessup Hall, Iowa City, IA 52242 (marc-armstrong@uiowa.edu).

D.-K.D. Rokos is currently with the Hellenic Mapping Consortium (HEMCO), Tim. Vassou 11-13, 11521, Athens, Greece.

Photogrammetric Engineering & Remote Sensing, Vol. 64, No. 2, February 1998, pp. 135–142.

0099-1112/98/6402-135$3.00/0

- *Medium* grained algorithms have tasks larger than a basic code block but smaller than a subroutine.
- *Coarse* grained algorithms are comprised of large individual tasks that are equivalent to one or more subroutines.

The specification of the granularity of a decomposition strategy for a particular problem is often made after considering two general principles:

(1) The granularity of an algorithm must match the target computer architecture. Parallel computers with single-bit processors are not well-suited to coarse or medium grained parallelism, while computers with powerful processors are often used inefficiently if a fine or medium grained approach is adopted.
(2) Fine grained decomposition usually achieves more balanced distribution of a workload than medium or coarse decomposition, but it usually incurs a much greater communication overhead (Cok, 1991).

The granularity for a specific problem domain often is determined only after evaluating the trade-off between efficient workload balancing and communication overhead.

Parallel Programming Paradigms

Carriero and Gelernter (1990) introduced the term parallel paradigm to describe parallelism in an implementation-independent context. There are three main parallel paradigms: event, algorithmic, and geometric parallelism. Carriero and Gelernter (1990: p. 13) noted that they "... aren't provably the only ones possible. But empirically they cover all examples we have encountered in the research literature." The feasibility of applying them depends not only on the problem's structure but also on the computer architecture and the available programming environment.

Event parallelism involves the identification of a main process (master task) and a set of specialized worker processes. The master task schedules a work plan, distributes data to the worker processes, and collects their results. Because a new set of data to be processed is sent whenever it is determined that a worker process is idle, event parallelism is inherently a load balancing method. Problems can arise, however, if one worker task requires considerably more computation than the others. Furthermore, event parallelism incurs a communication overhead penalty as the controller must continuously communicate (send and receive data) with worker processes. Consequently, it is best-suited to problems that require considerable processing but little communication (Cok, 1991).

Algorithmic parallelism involves the conceptualization of a problem as a network in which each node is a specialized process that operates on data passing through the network in a "production line" fashion. Although this paradigm often may be the most natural way of decomposing an algorithm, its efficient implementation depends heavily on the underlying computer architecture and the configuration of its interprocessor communication topology.

Geometric parallelism involves the decomposition of the problem space into subregions within which local operations are performed. In this paradigm, data often are evenly distributed to all processes, where each process usually is a complete program or procedure that performs all the required computations for each subset (Cok, 1991: p. 113). This is done for two-dimensional geographic data sets by dividing them into N equal parts, where N is the number of worker processors (e.g., Healy and Desa, 1989). Carriero and Gelernter (1990) noted that this is the most appropriate approach for decomposing problems that must produce a series of values with predictable organization and interdependencies. Thus, while geometric parallelism may be an intuitive and easy way to decompose problems, its efficient implementation also depends on the characteristics of the particular problem. A successful decomposition of a problem domain using geometric parallelism maximizes processing within a region and minimizes inter-region exchange of information, because interprocessor communication is the major factor that determines whether geometric parallelism can be efficiently implemented in a particular problem domain.

Process and Data Dependencies

Dependencies among program components can impair the performance of parallel programs. For example, if a process is completed on one processing element (PE) before another on which it depends for data, then the PE is forced to remain idle. In practice, parallel implementations of algorithms with complex process and data dependencies may not only perform poorly, but may even produce incorrect results if the correct sequence of execution is not guaranteed. Because of these problems, a formal analysis of dependencies enables a programmer to determine whether a problem can be efficiently implemented in parallel, and can be used to suggest appropriate decomposition strategies.

Williams (1990) identified several different types of dependency relationships that can exist between two processes (P_1 and P_2):

- **Prerequisite:** process P_1 must fetch what it requires before P_2 stores its results. P_1 must not require results produced by P_2, as there is no guarantee that P_1 will finish before P_2.
- **Conservative:** process P_1 must send its results to another process P_3 before P_3 receives the results of P_2. This suggests that both processes modify the same data structures and the results of process P_2 are required for subsequent processing by P_3. This type of relationship allows the unconstrained parallel execution of the two processes until the time that they return their results.
- **Commutative:** process P_1 may execute before or after P_2, but not at the same time. This occurs mainly when both processes modify the same memory locations during their execution.
- **Contemporary:** processes P_1 and P_2 are completely unrelated and may execute at the same time, without any constraints.
- **Consecutive:** process P_1 must send all or part of its results to process P_2. This implies that the results of P_1 are used by P_2, so the execution of the two processes is inherently sequential.

In most of the commercially viable parallel processing environments that are now available, a message-passing programming model can be used to ensure that the proper sequence of instructions is maintained. Though data are not shared among processes, they are connected through communication channels. Thus, whenever a process wants to receive data from, or send data to, another process, it uses a message. Using this approach, a dependency analysis is used to guide the derivation of a parallel message-passing version of the terrain feature extraction algorithm that is described in the next section.

Terrain Feature Extraction

Digital elevation models (DEMs) play an important role in many GIS-based analyses. Slope and aspect are often computed from DEMs, and, in some instances, other representations of terrain (e.g., TINs and drainage catchments) are derived from them. These derived terrain models help researchers to deal with the extreme abstraction of gridded terrain features by forming structures that more closely approximate their models of the world. Subcatchments and hillslopes, for example, provide an "unambiguous template for structuring models in geomorphology, hydrology, landscape ecology, and other fields" (Band, 1989a:151) and are used to delimit areas with distinct patterns of vegetation, soil, and microclimate (Band, 1989b).

Two basic topographic features are used to derive sub-

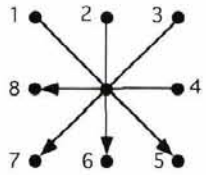

Figure 1. The 3 by 3 elevation window with the four symmetric bisecting transects (after Bennett and Armstrong, 1996).

catchments and hillslopes: drainage and divide networks. Drainage networks define the main flow patterns of water, sediment, nutrients, and pollutants in a watershed. Divide networks, on the other hand, define the extent and the shape of the runoff contributing area for each channel segment of a drainage network and bound relatively independent watershed partitions.

Drainage and divide networks are traditionally obtained by manual interpretation of topographic maps or aerial photographs. These tasks are quite tedious and time consuming for any but the smallest data sets and have "provided a strong restriction on the scale and the complexity of the watershed research that is generally attempted" (Band, 1989a: 151). To overcome this limitation and also deal with the increasing availability of topographic information in digital form, researchers have tried to automate the feature extraction process (see, for example, Peucker and Douglas (1975), Mark (1983), O'Callaghan and Mark (1984), Jenson (1985), Marks *et al.* (1984), Morris and Heerdegen (1988), and Bennett and Armstrong (1989; 1996)). These approaches differ in their complexity, in the way that they treat difficult or ambiguous classification problems, and also in their suitability for implementation in parallel processing environments.

In this research, we selected an approach to terrain feature extraction reported recently by Bennett and Armstrong (1989; 1996), and converted it to a parallel form. Their inductive terrain feature extraction (ITFE) algorithm uses a three-step procedure to extract hydrologically significant features from digital elevation models. In step one, ITFE (following Jenson (1985)) records, in a six-element Boolean array, the two-dimensional shape of the four directed, symmetrical transects that cross the center cell of a 3 by 3 elevation window (Figure 1). This information is used to classify each transect as ridge, valley, slope break, or flat by comparing it with prototypes that have been generated for each of the hydrologic categories (Table 1). Transects are then placed into the category with which they have the greatest similarity.

This first step has two characteristics that provide the user with control over the feature extraction process. Unlike most other feature extraction algorithms based on local operators, this approach is able to perform well in areas of low relief because it allows for "flexible windowing": when the elevation of the center cell is not significantly different from either end of a transect running through it, the window is recursively extended in the direction of uncertainty. As transects are classified, a topographic significance parameter is also used to define "the difference in elevation between the central point and a line bounded by two points on opposite sides of the window" (Bennett and Armstrong, 1989: p. 62). The consideration of topographic significance serves two purposes: it accounts for noise in the digital elevation model that could cause the erroneous classification of cells, and it is used to control the degree of detail represented in the derived feature networks.

In the second step, information derived from the shape of the four transects is used to classify the center cell of the elevation window into one of six hydrologic categories using the production rules described in Table 2. After each cell has been placed into a hydrologic category, the next step of the procedure creates an initial approximation of the basin's morphology by establishing preliminary topological relationships among classified cells. This is accomplished by linking adjacent cells that perform similar hydrologic functions (Table 3). The results of this step are stored in the form of an undirected graph. For each DEM cell, linkages with its neighbors are stored in two 8-bit records that describe upslope and downslope connectivity (Figure 2). Each element in the two records (*LikeNeighbors* and *activeLink*) corresponds to a linkage between a cell and one of its eight neighbors. Each element in the *LikeNeighbors* array is used to record whether there is a valid link between the center cell and the corresponding neighbor. If two cells are both topographically significant and classified in the same hydrologic category, then the link between the two cells is characterized as active (one cell drains to another) and the respective element in the *activeLink* array is set to true. The network is then constructed from the information stored in these records. The implementation of ITFE in a parallel environment is described in the following section.

TABLE 1. TRANSECT CLASSIFICATION RULES (AFTER BENNETT AND ARMSTRONG, 1996). A # INDICATES THAT A BIT POSITION IS NOT CONSIDERED FOR THAT CLASSIFICATION

Bit Mapping:		
1	true if center point is topographically significant	
2	true if first point of transect is higher than mid point	
3	true if last point of transect is higher than mid point	
4	true if the transect is extended in the direction of the first point	
5	true if the transect is extended in the direction of the last point	
6	true if extension of either line reaches a boundary.	
Line Classification:		
ridge	Both bounding points lower than middle	#FF##F
valley	Both bounding points higher than middle	#TT##F
slope break	One bounding point higher, the other lower than the middle;	#TF### or #FT### or
	One end of the line extended, one not	###FT# or ###TF #
flat	Both ends of the line extended	###TTT

TABLE 2. CELL CLASSIFICATION RULES (AFTER BENNETT AND ARMSTRONG, 1996)

Class	Criteria
Drainage	if transects classed as valleys>=1, peaks=0
Divide	if transects classed as ridges>=1, valleys=0
Pass	if transects classed as valleys>=1, ridges>=1
Pit	if transects classed as valleys=4
Slope break	none of the above: transects classed as slope breaks>=1
Plain	none of the above: transects classed as flat>=1

TABLE 3. VALID TOPOLOGICAL LINKS (AFTER BENNETT AND ARMSTRONG, 1996)

1. Drainage points to:	2. Divide points to:	3. Slope breaks to:
drainage points,	divides,	slope breaks.
passes,	passes.	
pits,		
plains.		

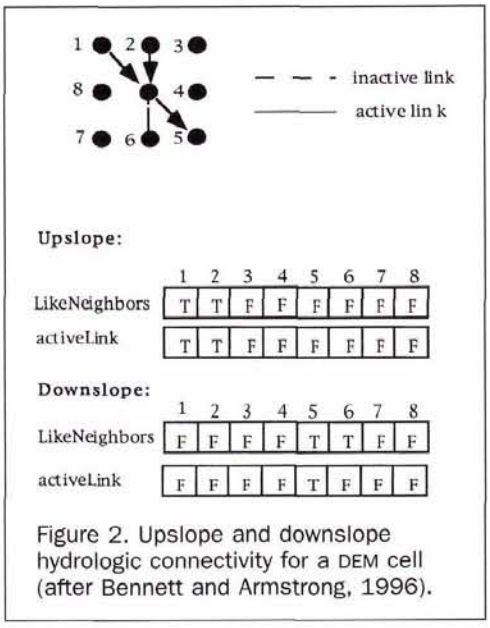

Figure 2. Upslope and downslope hydrologic connectivity for a DEM cell (after Bennett and Armstrong, 1996).

Problem Decomposition

When decomposing an algorithm for implementation, dependencies among processes and the relationships between processes and their data requirements must be identified. The goal of this analysis is to identify those segments of an algorithm that can be executed simultaneously, those that must be processed sequentially, and those that must be synchronized before additional processing can take place. The characteristics of the target computer architecture and parallel programming model are also considered. The result of this problem decomposition is a parallel inductive terrain feature extraction (PITFE) algorithm.

Data Dependencies

The three main steps of PITFE (transect classification, cell classification, and feature topology construction) are first considered as the basis of the dependency analysis. In the first step of the algorithm, each transect that bisects the center cell of a 3 by 3 elevation window is placed into a hydrologic category. Normally, this step requires the analysis of only three elevation values. In areas of low relief, however, where the determination of a transect's shape may require its recursive extension, the length of the transect extension cannot be predefined and processing load imbalances may occur. Because the number of points that must be considered for each transect classification is not known in advance, this presents a possible problem if each processor can only access data in its own local memory.

In the next step of PITFE, each DEM cell is placed into a hydrologic category. This is accomplished by examining the information produced from the classification of the four transects that bisect the cell under consideration. Because this information is produced only to classify the center cell of each window, data dependencies are not present in the cell classification task.

Finally, PITFE connects adjacent cells that perform similar hydrologic functions. This step requires the feature types of all nine cells of the DEM window to be known. Consequently, if we consider two windows A and B, where B is the window that is produced if we slide A one position (cell) to the right, the feature topology construction step for both windows will require access to the feature types of the six shared cells. Conventionally, the data set is scanned from left to right and from top to bottom. Each cell, therefore, is considered three times (Figure 3) when the algorithm is applied to (a) the preceding cells, (b) the current DEM window, and (c) the following cells.

Process Dependencies

Process dependencies in PITFE are identified not only between different processes, but also with respect to the portion of the data set to which they are applied. Using the classification scheme suggested by Williams (1990), the process dependencies of the algorithm are identified in Table 4. Given the consecutive dependencies also shown in Table 4 (see shaded cells), it is apparent that the three main processes in PITFE must be executed sequentially when applied to the same portion of the data set. Different processes applied to different portions of the data set, however, are unrelated and can be executed in parallel; they exhibit contemporary relationships. For example, cell classification within a window in the top-left of a data set, and feature topology construction in a bottom-left window, can execute in parallel without constraint, because information does not need to be exchanged.

The situation, however, is different when considering dependencies that are present in the feature topology construction process. In each window, the construction of the

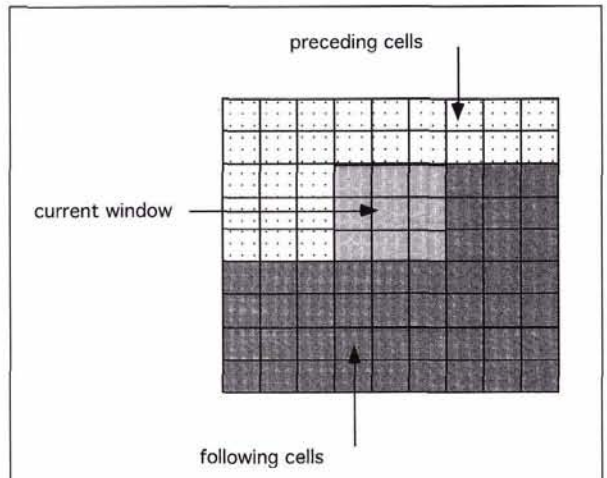

Figure 3. Each task is considered when applied to the preceding cells, the current window, and the following cells.

topological links between the center cell and its eight neighbors requires only the feature types of the window points. Therefore, feature topology construction for the three data-set subdivisions (preceding cells, the current window, and the following cells) is contemporary, because it is independent for each of these three instances. This approach is inefficient, however, because most topological links will be considered twice (Figure 4). Consequently, a more efficient version of PITFE (described in the next section) was also implemented to reduce the required number of evaluations.

Implementation

Because PITFE consists of several steps that exhibit different kinds of dependency relationships, several ways of organizing it for parallel processing could be devised. One way is to decompose it into the three steps described earlier. While the use of the algorithmic paradigm seems natural in this case, it creates relatively small tasks that generate a large number of messages (send data; receive results) and, in message passing environments, such communication overhead can degrade performance. Because the first two steps, transect and cell classification, are closely related, and the results of the first are required by the second, they were combined to form a single large-grained cell classification task. We refer to this as a *reduced* PITFE model. The remaining step of the algorithm, the feature topology construction task, is independent and executes after the cell classification task has been completed. To coordinate the execution of these two worker tasks, a master task was also designed to distribute data to the worker tasks, receive their results, and balance their workloads.

Though geometrical parallelism is a straightforward way to achieve performance improvements for many spatial problems (e.g., Armstrong and Marciano, 1994), its application to the first part of the PITFE algorithm (cell classification) presented obstacles to efficient implementation. Because transects are extended to determine their shape in flat areas,

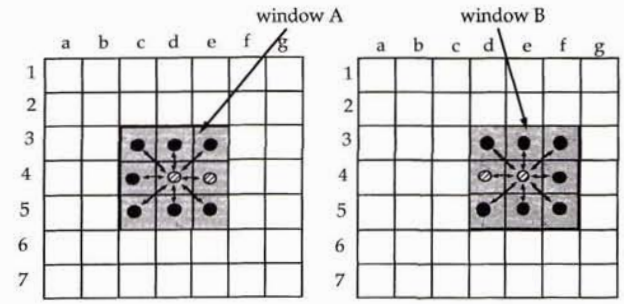

Figure 4. The topological links between cells (4,d) and (4,e) are established twice: first for window A and then for window B.

TABLE 4. THE PROCESS DEPENDENCIES OF THE INDUCTIVE TERRAIN FEATURE EXTRACTION ALGORITHM

	transect class. (current window)	transect class. (following windows)	cell class. (preceeding cells)	cell class. (current window)	cell class. (following windows)	feature topology construction (preceeding windows)	feature topology construction (current window)	feature topology construction (following windows)
transect class. (preceeding windows)	contemporary	contemporary	consecutive	contemporary	contemporary	consecutive	contemporary	contemporary
transect class. (current window)		contemporary	contemporary	consecutive	contemporary	contemporary	consecutive	contemporary
transect class. (following windows)			contemporary	contemporary	consecutive	contemporary	contemporary	consecutive
cell class. (preceeding windows)				contemporary	contemporary	consecutive	contemporary	contemporary
cell class. (current window)					contemporary	contemporary	consecutive	contemporary
cell class. (following windows)						contemporary	contemporary	consecutive
feature topology construction (preceeding windows)							conservative	conservative
feature topology construction (current window)								conservative

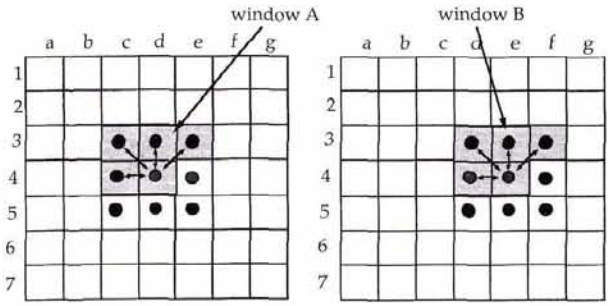

Figure 5. The topological link between (4,d) and (4,e) is established only once, when window B is considered.

elevation values in the direction of each extension must be made available to the cell classification task. However, because the length of a transect's extension cannot be known before the program is executed, the size of the elevation window required for the classification of a single cell cannot be predefined. There are several ways to treat this problem. One solution is for the master task to send the whole data set to each worker processor. Although this may be feasible for small data sets, it would degrade the algorithm's performance for large data sets because a large communication overhead penalty would be incurred. An alternative strategy was developed to reduce communication overhead: whenever a transect must be extended, the worker task encountering the problem sends all needed information to the master task which has access to the entire data set. The master task then extends the transect and returns its shape to the worker that requested it.

Geometric parallelism can cause an additional performance problem when uneven workloads occur. If a purely geometric decomposition were used, a worker processor that receives data representing a flat area would be required to perform many more transect extensions than other worker processors and would continue to process data after other workers had completed their tasks. Because of such problems, the cell classification task was implemented using event parallelism. This paradigm inherently balances processing loads and therefore, if one worker is required to do more processing than others, it will execute fewer times.

In the second step of the reduced PITFE algorithm (feature topology construction), geometric parallelism was used because each subset requires the same amount of processing, and data are equally distributed among worker tasks. Thus, the workload is well-balanced. Simple data dependencies exist between the boundaries of the subsets, however. In those cases, boundary data between two adjacent subsets are made available to both of them and they are treated differently from interior cells. This feature topology construction step presented yet another problem, however: double-processing of topological links. To overcome this problem, the topological links for a cell are constructed by first establishing the topological links between the center cell of a window and its four upper-left neighbors. Then, the topological links of the center cell with the rest of its neighbors are established when each is considered as the center cell of a DEM window (Figure 5). In this implementation, feature topology construction is conservative, as the record that holds the topological links for a cell is updated every time one of its lower-right neighbors is considered. Each link, however, is considered only once.

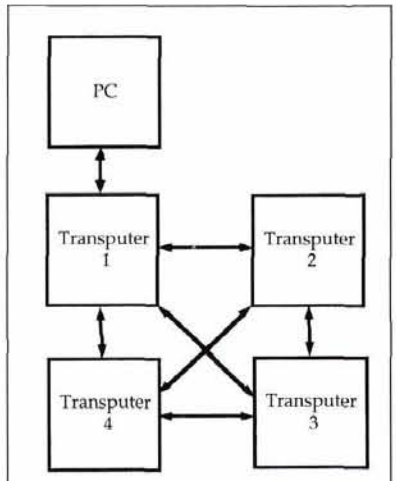

Figure 6. A schematic description of the target architecture.

Performance Analysis

The computer used to implement the PITFE algorithm is a microcomputer-based MIMD machine (Figure 6) consisting of a host PC that is serially linked to four fully interconnected T800 Transputers (see Ding and Densham, 1994). Because each Transputer is a complete computer with 1 megabyte of memory, the configuration can be described as a coarse-grained architecture. This indicates that the problem domain should be decomposed into relatively large tasks. Figure 7 depicts a schematic mapping of master and worker tasks to the Transputer architecture. The software was coded in Parallel Pascal (3L, 1989). This compiler is based on OCCAM, and implements a message passing programming model.

The data set used to examine the performance of the al-

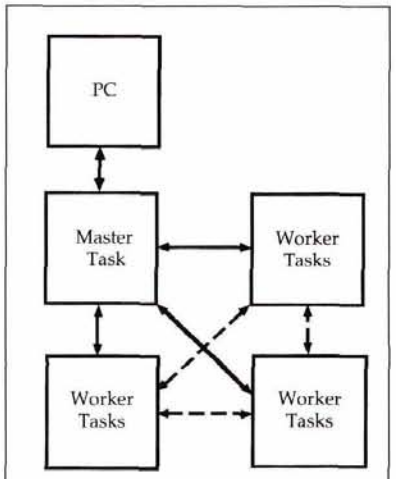

Figure 7. The identified tasks are mapped onto the target architecture. Worker tasks include both the cell classification and the feature topology construction tasks. Solid lines represent active and dashed lines inactive links for interprocessor communication.

TABLE 5. EXECUTION TIMING RESULTS USING 3 BY 3 AND 3 BY 31 WORKLOAD SIZES. ONE TICK IS EQUAL TO 64 MICROSECONDS

	3 by 31 workload size	3 by 3 workload size
Execution Time in Transputer ticks	4498	6259

gorithm is a 27 by 31 subset of the Mt. Baldy USGS DEM. This area includes a small watershed of the San Dimas river in southern California which is an area characterized by intense relief. The USGS characterized the Mt. Baldy DEM as a level one DEM which means that (a) no DEM point has an absolute error greater than 50 metres, and (b) the DEM does not possess an "array of points which encompass more than 49 contiguous deviations (an effective 7 by 7 array) wherein the relative integrity is not in error by more than 21 metres" (USGS, 1987: p. 11). The selected data set has a root-mean-square error of 5 metres based on a sample of 30 points.

Comparison of the Original and Reduced Models

The efficient implementation of a message-passing algorithm must use a minimum number of messages that each carries only needed information. By doing so, time is saved not only from reducing communication, but also from assembling smaller records. The reduced form of PITFE that was constructed by combining steps in the original algorithm was designed to operate in a message-parsimonious way and, as a consequence, it can be expected to yield increased levels of performance. To illustrate this effect, both versions (original and reduced) of PITFE were executed using one master and three worker processors. The resulting run time was reduced from 4498 to 3859 "ticks" (a time measure used in the Transputer environment, where each tick is equal to 64 microseconds; Microway, 1990), a 14.2 percent improvement.

Size of the Workload

Because the cell classification task is implemented using event parallelism, the data set is divided into uniform subsets (workloads) and distributed to the worker processors. Consequently, the specification of the size of the individual workload is an important issue. Because the parallel architecture used in this research is most efficiently applied to problems when the work local to each processor is maximized and interprocessor communication is minimized (Cok, 1991), increasing the workload size should make effective use of the available processors. To provide a simple demonstration of the importance of workload size, the cell classification step was run on the same data set with 3 by 3 and 3 by 31 workload sizes. As shown in Table 5, the decrease of communication overhead associated with the 3 by 31 workload yields a 28 percent performance improvement.

It should be noted, however, that the amount of work required for each workload depends not only on the size of the data subset sent to it, but also on the relief of the area in which the algorithm is applied. In low relief areas, for example, transect classification will require extra processing (transect extension) to determine the shapes of transects. Consequently, by increasing the data subset size, the required work could increase disproportionately in low relief areas. Thus, the amount of processing required for a given workload is a function not only of the data set size, but also of local relief.

TABLE 6. TIMING MEASUREMENTS OF THE PITFE ALGORITHM WHEN 1, 2, AND 3 WORKER PROCESSORS ARE USED ON THE EXAMPLE DATA SET. UNITS ARE LOW PRIORITY TRANSPUTER CLOCK TICKS (1 TICK = 64 μSEC)

Number of Processors	Cell Classification	Feature Topology
1	9765	7390
2	5300	4700
3	3859	3275

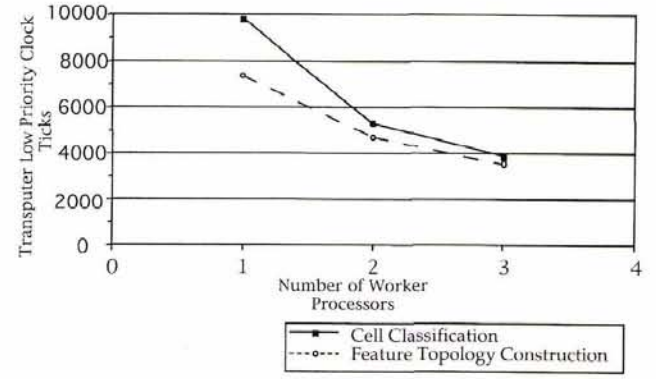

Figure 8. Execution times running the application with 1, 2, and 3 worker processors.

Increasing the Number of Processors Used

Because of the improvements demonstrated earlier, the effects of increasing the number of processors are reported only for the reduced form of PITFE. To evaluate the performance improvement of a parallel algorithm over an equivalent sequential version of the same algorithm, two measures are commonly used: speedup and efficiency. Speedup (S) is defined as the ratio of sequential run time (T_{seq}) to parallel run time (T_{par}): i.e.,

$$S = T_{seq}/T_{par}\,. \tag{1}$$

Efficiency (E) is the ratio of the speedup (S) to the number of processors (N) running in parallel: i .e.,

$$E = S/N. \tag{2}$$

The execution time of PITFE was measured when the application was run with one (sequential version), two, and three worker processors as shown in Table 6. It can be observed that the run time is reduced from 9765 "ticks" when one processor was used, to 3859 when three workers are applied to the cell classification step. Similar reductions are obtained for the feature topology step. Figure 8 shows the graph of the execution time of the cell classification and feature topology construction tasks against the number of worker processors used. It should be noted that, when the number of worker processors is increased, execution time is (theoretically) expected to decrease linearly up to a certain point, but will become asymptotically horizontal. This will occur because the increase in the communication overhead (caused by the increase of the amount of interprocessor communication required) will balance against the performance improvement which is anticipated when additional processors are applied to the computation of results.

The efficiency and speedup of each configuration was computed from the run times and are shown in Table 7. It is apparent from these results that, by increasing the number of worker processors, a considerable speedup of the algorithm is achieved and that in each case the processors are being used efficiently. Furthermore, for the feature topology construction step, an almost linear performance improvement can be observed, suggesting that this coarse-grained part of PITFE is scalable to a larger collection of processors. On the other hand, the performance improvement of cell classifica-

TABLE 7. MEASUREMENTS OF THE SPEEDUP AND EFFICIENCY OF THE PARALLEL VERSION OF THE PITFE ALGORITHM OVER ITS SEQUENTIAL COUNTERPART, USING 2 AND 3 WORKER PROCESSORS

Number of workers	Cell Classification		Feature Topology Construction	
	Speedup	Efficiency	Speedup	Efficiency
2	1.84	0.92	1.57	0.79
3	2.53	0.84	2.25	0.75

tion exhibits less linearity as the number of worker processors is increased. When the number of the processors involved is increased, additional interprocessor communication is required, and this overhead influences the timing results.

Summary and Conclusion

We have described several principles of parallel processing that were applied to the implementation of a parallel inductive terrain feature extraction algorithm. The PITFE algorithm resolves contradictions and inconsistencies in DEM terrain representations and derives feature networks even in low relief areas. Following the work of Williams (1990) and Carriero and Gelernter (1990), this application demonstrates how the terrain feature extraction problem can be decomposed into relatively independent tasks.

The importance of task size was demonstrated by running the cell classification task using 3- by 3- and 3- by 31-cell workloads. As expected, given that the Transputer architecture tends to perform best when allocated coarse-grained tasks, the application that used the larger workload size was 28 percent faster than the smaller. It should be noted, however, that the amount of processing required for performing cell classification with a given workload depends on the relief of the area represented. Consequently, a workload size that optimizes the performance of cell classification for a given data set, may be suboptimal not only for a different data set, but also for the same data set if a different decomposition strategy is used (e.g., dividing a data set into horizontal strips versus vertical strips).

The results illustrate the performance benefits of the parallel implementation over the traditional sequential approach. By using three worker processors running in parallel, the execution of PITFE was speeded up by 2.53 times with respect to an equivalent sequential version of the algorithm (one worker processor). Future research, however, must be conducted using a larger complement of processors to determine if there are practical limits to the scalability of PITFE.

This paper has illustrated how parallel processing principles can be applied to a geographical problem that contains considerable complexity in its data and process interdependencies. Based on these results, we expect that other geographical problems, which treat space in a similar way, can also be implemented efficiently in parallel environments by following the general approach described in this paper. The processing power provided by parallel environments enables researchers to develop more realistic process models, because they will be able to include more accurate and detailed representations of physical systems and more complex element interactions, while at the same time they retain their ability to interact with, and explore, the nature of geographical problems.

References

Armstrong, M.P., and R. Marciano, 1994. Inverse distance weighted spatial interpolation using parallel supercomputers, *Photogrammetric Engineering & Remote Sensing*, 60(9):1097–1103.

Band, L.E., 1989a. A terrain-based watershed information system, *Hydrological Processes*, 3:151–162.

———, 1989b. Spatial aggregation of complex terrain, *Geographical Analysis*, 21(4):279–294.

Bennett, D.A., and M.P. Armstrong, 1989. An inductive bit-mapped classification scheme for terrain feature extraction, *Proceedings GIS/LIS '89*, Orlando, Florida.

———, 1996. An inductive knowledge-based approach to topographic feature extraction, *Cartography and Geographic Information Systems*, 23(1):3–19.

Carriero, N., and D. Gelernter, 1990. *How to Write Parallel Programs: A First Course*, The MIT Press, Cambridge, Massachusetts.

Cok, R.S., 1991. *Parallel Programs for the Transputer*, Prentice Hall Inc., Englewood Cliffs, New Jersey.

Ding, Y., and P.J. Densham, 1994. A loosely synchronous, parallel algorithm for hill shading digital elevation models, *Cartography and Geographic Information Systems*, 21(1):5–14.

Flynn, M.J., and K.W. Rudd, 1996. Parallel architectures, *ACM Computing Surveys*, 28(1):67–70.

Healy, R.G., and G.B. Desa, 1989. Transputer based parallel processing for GIS analysis: Problems and potentialities, *Proceedings of the Ninth International Symposium on Computer-Assisted Cartography, AUTO-CARTO 9*, Bethesda, Maryland, American Congress on Surveying and Mapping, pp. 90–99.

Jenson, S.K., 1985. Automated derivation of hydrologic basin characteristics from digital elevation model data, *Proceedings of Auto-Carto 7*, pp. 301–310.

Mark, D.M., 1983. Automated detection of drainage networks from digital elevation models, *Proceedings of Auto-Carto 6*, Ottawa, Canada, pp. 288–298.

Marks, D., J. Dozier, and J. Frew, 1984. Automated basin delineation from digital elevation data, *Geoprocessing*, 2:299–311.

Microway Inc., 1990. *Quadputer Owner's Manual*, Microway Inc, Kingston, Massachusetts.

Morris, D.G., and R.G. Heerdegen, 1988. Automatically derived catchment boundaries and channel networks and their hydrological applications, *Geomorphology*, Volume 1, Elsevier Science Publishers B.V., Amsterdam, pp. 131–141.

O'Callaghan, J.F., and D.M. Mark, 1984. The extraction of drainage networks from digital elevation data, *Computer Vision, Graphics, and Image Processing*, 28:323–344.

Peucker, T.K., and D.H. Douglas, 1975. Detection of surface-specific points by local parallel processing of discrete terrain elevation data, *Computer Graphics and Image Processing*, 4:375–387.

Polychronopoulos, C.D., 1988. *Parallel Programming and Compilers*, Kluwer Academic, Boston.

Stone, H.S., and J. Cocke, 1991. Computer architectures in the 1990s, *IEEE Computer*, 24:30–38.

3L Ltd, 1989. *Parallel Pascal User Guide*, 3L Ltd.

USGS, 1987. *Digital Elevation Models- Data Users Guide*, The United States Department of Interior, U.S. Geological Survey, Reston, Virginia.

Williams, S.A., 1990. *Programming Models for Parallel Systems*, John Wiley & Sons, Chichester, England.

(Received 26 August 1996; accepted 25 January 1997; revised 09 April 1997)