Eleventh Annual Symposium on Geographic Information Systems

GIS 97

Vancouver
British Columbia
Canada

A Parallel Cartographic Modelling System: Design, Implementation, and Performance

Morin, Patrick R., M.C.S. Student, Carleton University, Ottawa
Dubrule, Diane E., Assistant Professor of Philosophy, Carleton University, Ottawa
Sack, Jörg-Rüdiger, ALMERCO-NSERC Chair and Professor of Computer Science, Carleton University, Ottawa

Abstract

We describe a parallel implementation of C. Dana Tomlin's map algebra (Tomlin 1990). The implementation adheres as closely as possible to Tomlin's specifications. It is written in C with the MPI ("Message-Passing Interface") library of parallel functions. The implementation is portable to a wide number of parallel computing platforms and has been tested on 3 different platforms ranging from a loosely coupled cluster of workstations to a tightly couple supercomputer.

1. Introduction

In recent years GIS and Cartography have gone through substantial changes with respect to users, problems, problem domains, and data. The user community is rapidly expanding to include users from different sectors of the economy; along with this come different demands regarding the type, required speed, scope and scale of applications. The effect of these changes is a rapid and huge increase in the computational demands placed on GIS and Cartographic modelling. To keep up with the computational demands without sacrificing accuracy of models or resolution, parallel computing appears to be the only solution.

In this paper, we describe the design and implementation of a parallel system for cartographic modelling. The remainder of the paper is organized as follows: Section 2 describes the Map Algebra specification. Section 3 gives a high level description of our implementation. Section 4 discusses design issues which arose during the implementation. Section 5 presents a simple application. Section 6 presents performance results for the system. Section 7 concludes with some open problems and ongoing projects.

2. Map Algebra

Each of the fifty-nine operations described by Tomlin (1990) operates on geographic data stored in one or more "map layers" and produces a single output layer. A layer consists of a two-dimensional array ("raster") of locations (each containing a single value), together with general facts about the layer, such as its dimensions and resolution. The functions

we have implemented include all of Tomlin's local, zonal and incremental functions and most of the focal functions.

The local map algebra functions perform operations on the values at a single location on one or more layers and assign the result of the calculation to the corresponding location in the output layer. For example, LocalMean computes a new raster in which the value of the raster at each location is the mean of the values at that location in the input rasters.

The zonal functions input two layers ("firstlayer" and "secondlayer") and compute a new value for a location in the output layer ("newlayer") as a function of firstlayer values in the location's second- layer zone. In Tomlin's terminology, a zone is a set of locations (not necessarily contiguous or having any special shape) that have the same value. It can be as small as a single location or as large as all the locations in a raster. Although the term is used with respect to all layers and types of functions, when referring to the zonal functions, the term signifies a group of secondlayer locations with the same value which determine the firstlayer locations to be used in computing newlayer values. For example, ZonalPercentage computes a newlayer in which the value at each location is the percentage of firstlayer locations within that location's secondlayer zone that have a value equal to the value at that location.

Focal map algebra functions treat locations as the foci of neighbourhoods, computing new values for a location on the basis of the values in the location's neighbourhood. Tomlin defines a neighbourhood as a set of locations each of which bears a specified distance and/or directional relationship to a particular location, as defined by a range of distances, ranges of directions or combinations of distance and direction. A FocalVariety operation, for example, computes a newlayer in which the value of the raster at each location is the number of distinct values in that location's neighbourhood.

The incremental functions are a second type of map algebra function that treats locations as the foci of neighbourhoods. All compute new values for a location on the basis of the values in the location's immediate (nine-location) neighbourhood, which is used as a way of defining the non-punctual cartographic form of a location, i.e. its lineal, areal and surficial characteristics. For example, the length of a road can be computed using Incremental Length, which looks at each location as the focus of an immediate neighbourhood: If the focus contains part of the road, it identifies its linear form, which depends on which of the immediately adjacent locations also contain parts of the road, then multiplies the length associated with that lineal form by the resolution of the raster, and assigns the result—this location's contribution to the road length—to the location in an intermediate layer. ZonalSum (with the intermediate layer as firstlayer and a layer with all locations set to the same value as secondlayer) is used to sum the contributions of all the locations, which the total length of the road. The areal group of incremental functions produce similar results with areas. The calculation of surficial features, such as slopes and drainage, depend on viewing the location as a grid square, which is bevelled by taking into consideration the relative heights of locations in the immediate neighbourhood. The heights, given in a surfacelayer, can also be considered in determining linear and areal features.

3. A Parallel Map Algebra Implementation

Map Algebra forms the heart of a geographic information system based on a cartographic modelling approach (Tomlin 1990). Map algebra operations are concerned solely with data interpretation (as opposed to data preparation or presentation): they uncover facts, relationships and meanings implicit in existing data and express them explicitly as values assigned to locations in output layers. The map algebra operations must be complemented by facilities for preparing data for analysis and presenting results. The project described here makes use of features provided by NEMO (Hutchinson, et al. 1996), a system for parallel neighbourhood modelling. The interface with the data-preparation and presentation facilities is accomplished by a function that (1) builds a standard information structure from the partly processed map data and the parsed user requests it receives, (2) calls one or more map algebra functions and then (3) sends the result to the presentation facility.

Some of the map algebra functions are computationally very intensive; e.g., it takes 10 hours to process a raster of size 6000x6000 at 1000 cells/second. There are significant advantages in parallelizing the map algebra functions. Our work shows that processing time can be reduced linearly (in the number of processors) so that even a relatively simple map algebra operation that requires 104 seconds on a single processor machine can be carried out in 6.5 seconds on a sixteen processor machine.

Our implementation contains fifty-five of the fifty-nine operations and may be classified as a data parallel MIMD ("multiple instruction, multiple data") implementation. NEMO splits the raster that represents the map as a whole into subrasters. Each is sent to one of the processors together with the location of the subraster in the raster and other facts about the raster and the data it contains. Each processor uses its own program to compute as much as it can with the data it has. It passes messages to other processors to obtain additional data it requires to execute the program. Two types of communication are used: point-to-point communication, in which one processor receives data from another, and collective communication using MPI reduction functions, in which data is collected from all the processors, a predefined or user-defined reduction operation is performed on it, and the result is returned to all the processors.

A very different type of parallel implementation of the map algebra functions was carried out by Li (1992). He describes an SIMD ("single instruction, multiple data") implementation of map algebra functions of each type on a massively parallel Connection Machine (with 16,384 processors) in C*. A major advantage of a MIMD implementation is that general purpose MIMD hardware is readily available at a good price to performance ratio (as little as a cluster of 4 Intel Pentium processors), while SIMD hardware is typically very special purpose and expensive.

4. Design Issues

This section discusses some of the issues encountered in the design of the Map Algebra library and the design decisions made.

4.1. Zonal Operations

Zonal functions pose a problem for parallelism in that zones need not be contiguous, of equal size, or of a regular shape. Because of this, a single zone can span multiple subrasters which makes the parallel computation of, say, a ZonalMaximum non-trivial. The approach taken in our implementation is to have each processor compute values for its subraster, perform a global communication operation to combine the values for all zones, and then write out the appropriate values in its subraster. MPI provides efficient support for this type of operation in the form of the MPI_Allreduce

function. For example, we can compute the maximum value for each zone across all processors by performing an MPI_Allreduce operation with the parameter MPI_MAX. Another example is the calculation of a mean by performing two MPI_Allreduce operations, to sum the counts and values of each zone.

4.2. Focal Operations

The Focal family of Map Algebra functions implemented operate on a single raster using fixed shape neighbourhoods. The neighbourhoods can be specified either as an integer radius in which case all cells within the given radius are used in the computation, or as a list of (x,y) offsets which are taken as relative to the pixel being operated on. In this way, arbitrarily complex neighbourhoods can be specified. For the sake of efficiency, neighbourhoods specified as a radius are converted internally to a list of offsets.

The non-trivial aspect of implementing focal operations is that the neighbourhood data for a pixel may not be stored on the same processor that the pixel itself is stored. One way of handling this is to implement a demand-driven system in which each processor determines which pixels it needs from other processors and requests them from the other processors. The other processors then send the requested pixels.

We chose to implement a solution more efficient in terms of communication, in which each processor determines beforehand which of its pixels will be needed by other processors and sends them these pixels. This is done by measuring the absolute value of the largest offset and padding the subrasters of the raster with data from neighbouring processors. Although this method may not always be optimal in terms of communication, its simplicity, coupled with the fact that most focal operation operate on circular neighbourhoods, makes it the most effective. This method has the additional advantage that it requires half as many communication operations as the previously described method.

Another important issue in the implementation of the focal operations was the question of whether to store extra neighbourhood data separately, or to create a new "padded" subraster which contains the pixels of the original subraster as well as those on the boundary which are needed as neighbourhood data. The second approach has the apparent disadvantage that a new area of memory must be allocated for all the pixels in the padded subraster, and all the pixels of the original subraster must be copied to this area. Both approaches were implemented and experimentally compared; the padded subraster approach proved to be more efficient. The reason for this is that, although there is some overhead involved in copying the pixels of the old subraster, pixel indexing becomes simpler (roughly six machine instructions per lookup). Since focal operations tend to look at many pixels, particularly when the neighbourhood size is large, this was the approach finally chosen.

4.3. FocalMajority, FocalMinority, and FocalVariety

Initially, a naive implementation of the FocalMajority, FocalMinority, and FocalVariety functions was implemented but was found to be unacceptably slow when the size of the neighbourhood was large. This is due to the fact that it performed a number of pixel operations quadratic in the size of the neighbourhood. Divide-and-conquer algorithms exist which use less than a quadratic number of pixel operations, but these methods have unacceptably high overheads when the neighbourhood size is small. After some performance tests, we designed and implemented a method which performs a linear number of pixel operations and uses a technique similar to that of the Counting Sort algorithm described in Knuth 1973.

This technique exploits the fact that there are only a finite number of possible values a pixel can have. It uses just two auxiliary arrays to keep track of the counts of each pixel type in the neighbourhood and the pixel types that have been encountered. In this way, the second array can be used as an index into the first to compute the majority, minority, or variety in time linear in the neighbourhood size. Also, the arrays need to be initialized only once, so that much of the overhead of the algorithm is incurred once for each processor rather than once for each pixel.

5. A Simple Application

Figure 1 shows the results of applying two map algebra operations in order to trace the shorelines of lakes. The first image is the origi-

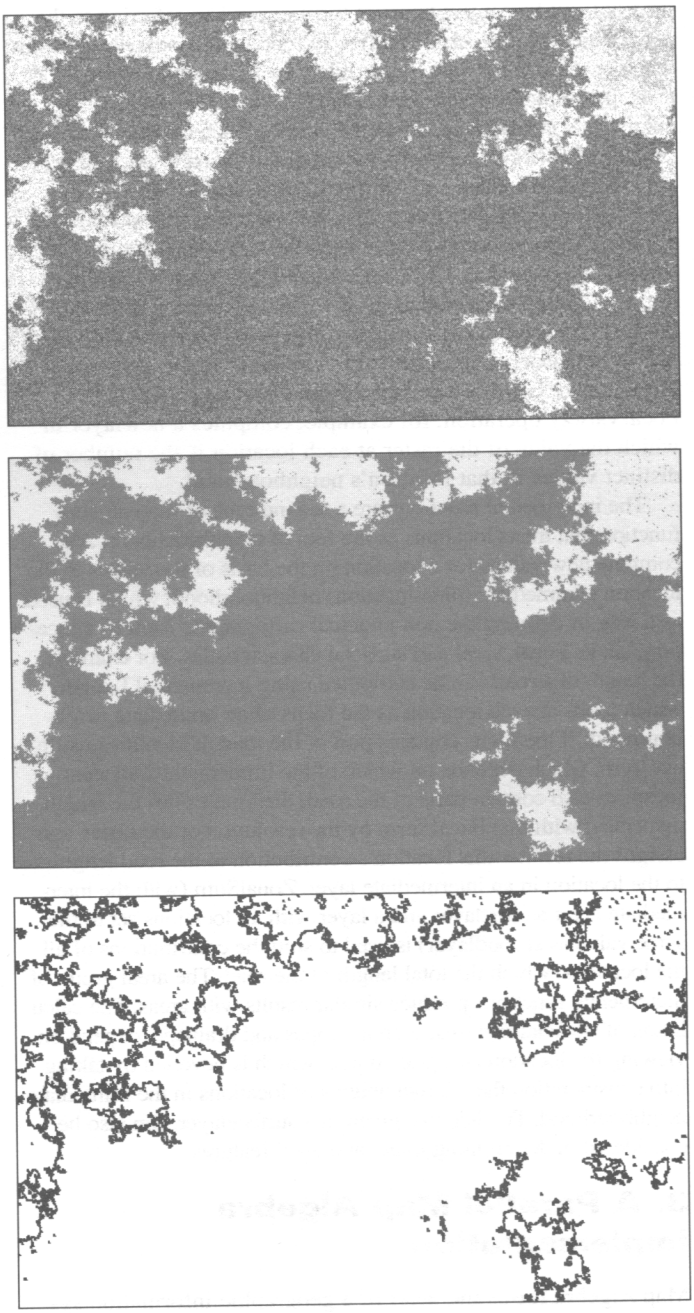

Figure 1. Shoreline tracing using the Map Algebra functions.

nal input raster, an aerial view of a forest and several lakes. The second image is the result of using a LocalRating operation to convert the image into two distinct parts, land and water. The third image is the result of applying the FocalVariety operation to the second image with a distance of 1. In this case, since there are only two types of pixels, land and water, the FocalVariety function sets a pixel to the value 1 (white) if it is not on a shoreline and 2 (black) if it is.

6. Experimental Results

This section presents performance results for the parallel Map Algebra implementation. The tests described herein were performed on a cluster of 16 166MHz Pentium processors connected by a 100MHz fast Ethernet switch. Each processor has 32 Mbytes of main memory and a 1GB SCSI hard disk. Results are presented for representative zonal and focal functions. (As far as implementation goes the focal and incremental functions are equivalent.)

For these tests we were interested in measuring the performance of our implementation with respect computing performance and not I/O. (Typically, a sequence of Map Algebra functions is executed on given input.) Towards this end, all tests loaded the input data, synchronized the processors (with a call to MPI_Barrier), performed the Map Algebra computations, and synchronized the processors again. The time measured was the time elapsed after the first synchronization until after the second.

Figure 2 shows the performance of the LocalMaximum function for between 1 and 16 processors on two input rasters of size 2048x2048. The performance is very near optimal for up to 6 processors and then drops off slightly. The reason for this

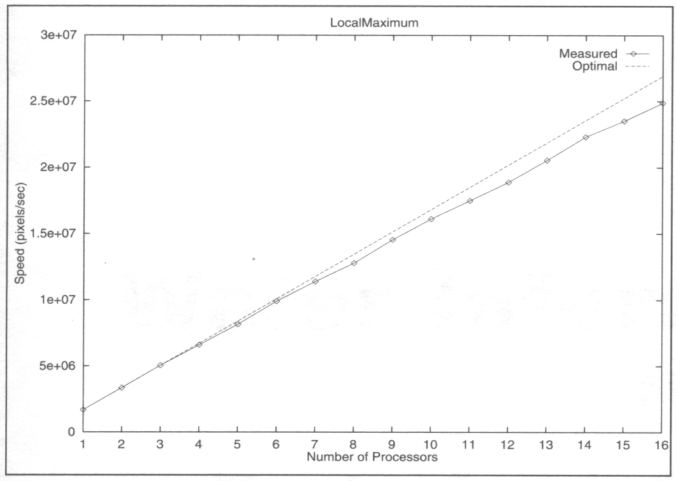

Figure 2: Performance of LocalMaximum.

is that at this point, the computation time is so small that the synchronization step begins to have a noticeable effect.

Figure 3 shows the performance of the FocalMaximum function on two rasters of size 2048x2048 with a neighbourhood of radius 5. Surprisingly, the speedup is better than optimal for some numbers of processors. This can be explained by the fact that as the number of processors increases, the number of pixels stored by each processor is smaller which allows the processor's on board cache to achieve better hit rates. This effect is only noticeable with focal functions since the local and zonal functions examine each pixel only once whereas the focal functions examine each pixel up to n times for a neighbourhood of size n. Similar, but more marked effects have been observed when the size of the raster exceeds the capacity of a processor's main memory and parts of the raster must be swapped to disk.

Another interesting aspect of Figure 3 is the manner in which the performance fluctuates with the number of processors. At first it may appear that these fluctuations are caused by measurement errors, but in fact they are related to the factorization of the number of processors and its effect on communication. To see this, consider 16 processors which are arranged as a 4x4 mesh. These processors must communicate roughly 2048x5x2x6=12280 pixels in all. If we consider 13 processors arranged as a 1x13 mesh, we would expect the amount of communication to be less since there are fewer processors, but in fact we see that roughly 2048x5x2x12=24560 pixels must be communicated between processors. The worst configurations occur when the number of processors is a prime number, and the best when it is a perfect square. In our system, this is not a serious issue since it consists of a high bandwidth network and the computing power gained by the extra processors makes up for the extra communication overhead, but in more loosely coupled networks it is conceivable that it may be wise to not use some processors in order to avoid this effect.

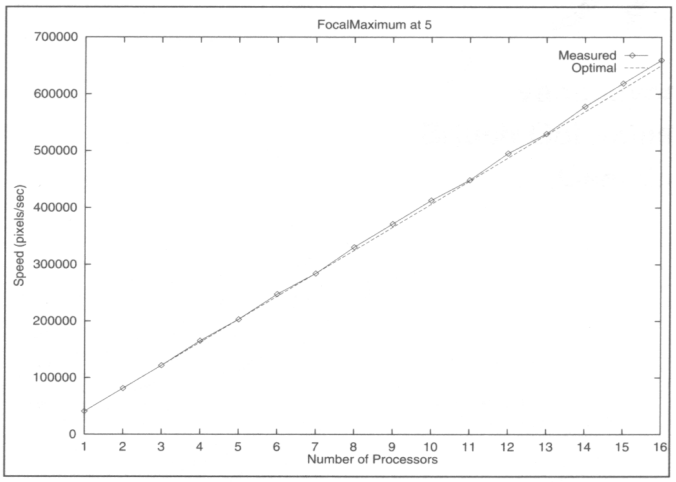

Figure 3: Performance of FocalMaximum.

Another method would be to write the number of processors as a sum of squares and treat each square as an individual mesh. The disadvantage of this approach is that it causes irregular communication patterns, increasing the complexity of the message passing code.

7. Conclusion

Continuing work on this project includes using the built-in NEMO facilities to help implement the remaining four Map Algebra operations involving spreading and radiating clauses of the focal functions. Other ongoing projects include a graphical Map Algebra specification language and a full featured Map Algebra parser and execution platform.

References

Message Passing Interface Forum. 1995. MPI: *A Message-Passing Interface Standard*. Knoxville, TN: University of Tennessee.

Hutchinson, David, Lars Küttner, Mark Lanthier, Anil Maheshwari, Doron Nussbaum, David Roytenberg, and Jörg-Rüdiger Sack. 1996. "Parallel Neighbourhood Modelling: Research Summary", *SPAA '96 Proceedings*, pp. 204-207; conference version in Proceedings of the 4th *ACM Workshop on Advances in GIS*, Rockville, MD, pp. 25-34.

Li, Bin. 1992. "Opportunities and Challenges of Parallel Processing in Spatial Data Analysis: Initial Experiments with Data Parallel Map Analysis." *GIS/LIS Proceedings*, 2:445-458.

Tomlin, C. Dana. 1990. *Geographic Information Systems and Cartographic Modeling*. Englewood Cliffs, NJ: Prentice Hall.

Knuth, Donald E. 1973. *Searching and Sorting*, volume 3 of *The Art of Computer Programming*. Addison-Wesley.

Biographies

Diane Dubrule holds degrees in Philosophy from Cornell (A.B.), and Toronto (M.A., Ph.D.) and is an Assistant Professor in the Philosophy Department and a fourth year undergraduate in the School of Computer Science at Carleton University.

Pat Morin holds a B.C.S. degree (Highest Honours) from Carleton University and is currently an M.C.S student at Carleton.

Jörg-Rüdiger Sack holds degrees in Computer Science from Bonn (Diplom), Germany and McGill, Montréal (Ph.D.), holds an ALMERCO-NSERC Chair and is a Professor in the School of Computer Science at Carleton University.