TRANSPUTER BASED PARALLEL PROCESSING FOR GIS ANALYSIS: PROBLEMS AND POTENTIALITIES

Richard G. Healey and Ghazali B. Desa
Regional Research Laboratory, Scotland
Department of Geography
University of Edinburgh
Drummond Street
Edinburgh EH8 9XP
Scotland, U.K.

ABSTRACT

The availability of parallel processing computers based on large number of individual processing elements, offers the possibility of multiple orders of magnitude improvement in performance over the sequential processors currently used for GIS analysis. Before this potential can be realized, however, a number of problems must be addressed. These include assessment of the relative merits of different parallel architectures, choice of parallel programming languages and re-design of algorithms to allow effective distribution of the computational and i/o load between individual processors, so performance can be optimized. These problems are examined with particular reference to transputer-based parallel computers and some possible GIS application areas are discussed.

INTRODUCTION

The limitations of serial processors for handling computationally intensive problems in fields such as fluid dynamics, meteorological modelling and computational physics are well-known, but it is only comparatively recently that attention has been turned to this problem in the fields of remote sensing/image processing and GIS (Yalamanchi and Aggarwal 1985, Dangermond and Morehouse 1987). Parallel processing techniques, where one or many computational tasks are distributed across a number of processing elements, have been proposed as a solution to the problem (Verts and Thomson 1988). They offer the potential for orders of magnitude improvement in performance, which should allow real-time processing of very large datasets, with powerful modelling and visualization capabilities.

Since parallel processing hardware is still at an early stage of development and parallel programming methods are distinctly in their infancy, it is difficult to make firm statements about how GIS might avail itself of this new technology. More appropriate at this stage is an examination of several aspects of the overall problem. These aspects include evaluation of existing types of hardware and software for parallel processing and approaches to the re-design of GIS algorithms, so they can take advantage of these novel machine architectures. This paper addresses these issues with particular reference to parallel processing based on networks of transputers.

TYPES OF PARALLEL ARCHITECTURE

It is not the intention here to give an extended survey of parallel architectures, as several of these are already available (Bowler et al. 1987a, Treleaven 1988), so a brief outline of the major types will suffice to provide the context for the present discussion.

SIMD and MIMD parallelism

One major approach to the design of a parallel computer is to link processing elements (PEs) into a two-dimensional array. If a SIMD (single instruction, multiple data-stream) method of operation is used, each program instruction is despatched simultaneously to each PE which executes it on the data it has stored locally. Examples of such machines include the NASA Massively Parallel Processor, the Connection Machine and the AMT distributed array processor. MIMD (multiple instruction, multiple data-stream) machines have grown in importance of recent years because of the availability of cheap but powerful microprocessor chips. Programs running on these machines are executed by all the PEs, but at any moment each processor may be at a different stage of program execution.

Shared and distributed memory systems

MIMD machines can be sub-divided on the basis of how memory is allocated to individual microprocessors. If a number of these are connected to a number of memory modules by means of a switch, to form a common global memory, the machine is of the shared memory type. An example of this is the BBN Butterfly which utilizes Motorola 68000 or 68020/68881 chips. In a distributed memory system each PE comprises a microprocessor with its own local memory. Hardware switches or links connect the individual PEs. An example would be the CALTECH Mark III Hypercube, which is also based on up to 128 Motorola 68020/68881 chips with additional I/O processors and Weitek floating point units.

Fixed and reconfigurable architectures

A further sub-division of distributed memory MIMD machines can usefully be made, depending on whether the topology of the links between the individual PEs is fixed in the hardware or is electronically re-configurable. The latter introduces a much greater degree of flexibility into the ways the computing resource can be utilized for different applications. Examples of fixed and reconfigurable architectures are the Intel iPSC-VX, based on 80286/80287 processors, and the Meiko Computing Surface based on transputers.

TRANSPUTER-BASED PARALLEL PROCESSING

Since it is less widely known than the Motorola or Intel chip sets, it is useful to outline some of the particular features of the Transputer, which was designed with parallel processing in mind, before assessing some of the advantages and disadvantages of different parallel architectures.

The most recent version of the INMOS transputer, the T800 chip, contains a number of processing components. The first of these is a 10-MIPS 32-bit RISC processor linked at 80 MByte/Sec to 4K of on-chip RAM. In addition there is an integral 64-bit floating point unit capable of 1.5 Mflops. The chip has 4 20 MBit/sec INMOS links which can be directly connected to other transputers, together with an external memory interface with a bandwidth of 26.6 MByte/sec. The T800 is claimed to achieve more than five times the performance of the Motorola 68020/68881 combination on the Whetstone benchmark (Bowler et al. 1988).

The major vendor of transputer-based parallel computers is the UK firm Meiko Ltd. which has developed a modular, extensible and reconfigurable computer system called the 'Computing Surface'.

COMPARISON OF PARALLEL ARCHITECTURES

Given the difficulties of comparing supposedly similar serial processors, it is not surprising that comparison of parallel architectures, where there are many more parameters to consider, is a rather inexact science. Nonetheless, some major points relating to the different categories described above can be identified.

With respect to SIMD and MIMD architectures, testing of distributed array processors against transputer networks indicates that the former operate best with strongly structured algorithms which do not have independently branching chains of instructions. Requirements for very fast I/O and data comparison operations also favour SIMD machines. The two types both perform well for sorting, but transputer networks are superior for 3-d graphics and modelling requirements (Roberts et al. 1988).

For shared and distributed memory systems the picture is less clear, because shortcomings of specific hardware configurations may be compensated, to varying degrees by the use of different operating systems and programing methods. Several points need to be considered, however. Firstly, access to local memory is on average three times faster than to a global or shared memory. Secondly, the speed of interprocessor communication and thirdly, the relationship between the computational performance of each PE and the speed of interprocessor communication need to be taken into account. The usefulness of a particular architecture will then depend on the extent to which a given problem can be decomposed into separate computational tasks, or requires access to a shared database of information (Ballie, 1988). The overall aims will be to minimize communications bottlenecks between processors, to get maximum utilization of each PE during the computation and to attain as nearly as possible a linear speed-up in processing time, as the number of processors used for a particular set of calculations is increased. Distributed memory systems, such as the transputer network, are proving popular because of fast memory access, but difficulties may arise for such systems in terms of communications overheads, if large amounts of data require to be transferred between processors.

Comparison between fixed and reconfigurable architectures is more straightforward, in that the latter is undoubtedly to be preferred, for two main reasons. Firstly, it allows the machine to 'mimic' other architectures, such as a SIMD or a hypercube, for comparative purposes. Secondly, it permits the overall computational resources to be divided into 'domains' of different sizes, so parts of the machine can be used for development while others are engaged in large scale computation.

Finally, the cost-effectiveness of parallel architectures involving large numbers of PEs, compared to single or multiple vector processors much be addressed. The parallel approach has an initial advantage because it is scalable from a small number of PEs, costing a few thousand dollars, to top-end Computing Surfaces costing several millions. In addition, using the unit comparison of megaflops/megadollar, a transputer-based machines seems to have approximately a five-fold improvement in cost effectiveness compared to a CRAY X-MP/48, with similar floating point performance (Bowler et al. 1987b).

These considerations indicate that reconfigurable, distributed memory MIMD machines such as the Meiko Computing Surface have a wide range of advantages, with potential limitations only in relation to interprocessor communication and extreme high-end reqirements for floating point performance. As a result, the Meiko machine was chosen as the basis for the Edinburgh Concurrent Supercomputer Project. The first phase of this has resulted in the installation of a Meiko machine with 200 transputers, each with 4 MB of local memory, together with a filestore and specialized graphics peripherals. Future project phases will allow the installation of large numbers of additional transputers, until the machine reaches its target size of at least 1024 processors, with 10,000 MIPS total processing power, 4 GBytes of distributed memory and an expected rating in excess of 1 Gflop. This will make it one of the most powerful supercomputers in Europe. The machine is jointly managed by the University Computing Service and the Department of Physics, but is in use by a variety of other research groups also.

PARALLEL PROGRAMMING LANGUAGES

Although parallel hardware has demonstrably reached the stage where a range of applications for large scale GIS processing can be envisaged, the position is less clear in terms of operating systems and programming languages.

Since parallel machines generate many new problems of system management, they have tended to be built with special purpose operating systems, which is a first major obstacle to software development activity of any kind! In the case of the Meiko this problem has now been resolved by the development of a UNIX System V compatible operating system for development work.

The second obstacle is the lack of support for parallel programming constructs in existing languages used for GIS software, specifically FORTRAN, C and to some extent PASCAL, although FORTRAN 8X is expected to include such facilities in the future. At present, there are three alternative methods of circumventing the problem:

i) Addition of new constructs into language compilers running on parallel machines

ii) Use of new languages, such as ADA or OCCAM, which support parallel programming directly

iii) Use of existing languages within a communications 'harness' provided by languages like OCCAM to allow access to parallel facilties.

In relation to the first alternative, a 'parallel' C compiler has been announced for the transputer, but problems of standardization are likely to plague this approach. The second alternative offers promise because the transputer was designed to run OCCAM, a language for parallel programming based heavily on Hoare's work on communicating sequential processes (Jesshope 1988). The language has particular strengths in facilities for message passing along OCCAM channels, but is limited in its support for the variety of data structures found in existing sequential languages. ADA, by contrast, supports parallel programming through its tasking model, while having a wealth of data structures. It also provides constructs to support the use of sound software engineering techniques (Sommerville and Morrison 1987). An ADA compiler for the transputer is currently under development and this avenue for future work will be explored when the tools become available. The final alternative is the one which is most heavily used at present on the Meiko processor, particularly for FORTRAN in an OCCAM harness. The FORTRAN implementation allows block input and output of data arrays between FORTRAN programs running on different processors, across OCCAM channels communicating over the transputer hardware links. This approach is satisfactory in the short term for testing alternative parallel algorithms or re-using existing code, to take advantage of the substantial increase in processing power on a parallel machine. It is not, however, a suitable approach for the implementation of large programming projects, designed to produce reliable and maintainable software that makes effective use of parallel processing techniques.

The shortcomings in available facilities and standards for parallel programming languages require to be addressed urgently if progress in the use of the hardware is not be hampered. While these shortcomings make it difficult to port existing packages, the situation may not be entirely disadvantageous, as it allows attention to be focussed at present on research into methods of parallelization of algorithms. Development of effective approaches for different kinds of algorithms is fundamental to the proper utilization of parallel processing techniques.

PARALLELIZATION OF ALGORITHMS

It is apparent from the earlier discussion that algorithms appropriate for one kind of parallel architecture may be unsuitable for another. This examination of parallelization methods will be restricted to distributed memory MIMD machines.

Although architecture dependent at the level of specific implementation, several general points about the relationships between serial and parallel algorithms can still be made (Miklosko and Kotov 1984):

i) Effective serial algorithms may not contain any parallel elements

ii) Some apparently serial algorithms may contain significant hidden parallelism

iii) Non-effective serial algorithms can lead to effective parallel algorithms

Past experience and well-tried serial methods of programming may not therefore be a good guide to parellel algorithm design and the qualities of inventiveness and imagination may be of more value in developing new approaches to problem solving! There are, nonetheless, several broad approaches to algorithm parallelization, including

i) Event parallelism
ii) Geometric parallelism
iii) Algorithmic parallelism

Event parallelism

This is one of the most straightforward ways of exploiting parallel processing. The approach utilizes the concept of a master processor which distributes tasks to slave processors and assembles the computational results from each in turn. Such a configuration is usually termed a 'task farm' (Bowler et al. 1987b). In the simplest kinds of problem, each slave processor runs the same code against its own specific dataset. When the processor array is large and each processor individually very powerful, as on the Meiko machine, it can be difficult to achieve data input rates sufficiently high to keep the machine busy. Conversely, where the computational load is very high but the data input more restricted the Meiko delivers extremely high performance. A good example of this type of problem is ray tracing for the display of complex 3-d objects. Since each light ray being followed is independent, the algorithm is highly parallel. While such algorithms have important application for visualization problems in GIS, many other GIS processing requirements are not of this form.

Geometric parallelism

This is a very natural kind of parallelism for GIS algorithms, as it requires that the problem space be divisible into sub-regions, within which local operations are performed. Calculations performed on elements near the boundaries of sub-regions will generally require information from neighbouring sub-regions. This emphasis on algorithm localization matches an approach which can be found, for instance, in the Intergraph TIGRIS system, for interactive editing of topological data structures (Herring 1987) and in the inward spiral algorithm for TIN generation (McKenna 1987).

Since individual sub-regions will be handled by different processors, boundary data must be passed between them, introducing a communications overhead. It is therefore important, with the current level of development of distributed memory MIMD machines, to define sub-regions such that the amount of within region processing is maximized and the between region communication is minimized. It is also advisable to locate neighbouring regions on physically adjacent processors. (Bowler et al. 1988).

The four available hardware links on each transputer are sufficient for two-dimensional geometric parallelism, but even for the simplest three-dimensional case with sub-regions forming cubes, each sub-region will hve six boundary faces. This can be matched in the hardware by connecting multiple transputers as 'supernodes' to yield six or more links (Jesshope 1988).

For geometric parallelism it is also necessary to consider the way in which the overall processing operation and communication between processors is organized. If a tightly synchronous model is used, a master processor communicates with each sub-region processor to determine when all have completed their local computations for a given step. At this point an exchange of boundary information takes place before proceeding with the next computational step. This approach generally produces inefficient hardware utilization. The loosely synchronous model, where boundary information is exchanged as soon as one processor is ready to provide it and its neighbour is ready to receive it, is to be preferred if processor workloads are broadly similar. If workloads vary significantly, as might be expected for GIS applications, complex asynchronous behaviour may result, leading to unpredictable levels of inefficiency (Norman 1988). One solution to this is to recast the problem to allow improved dynamic balancing of the workload, perhaps by assigning non-contiguous portions of the overall space to individual processors. This approach has been used effectively for three-dimensional medical imaging (Stroud and Wilson 1987) and has immediate application for voxel-based processing of geosciences data (cf. Kavouras and Masry 1987).

Algorithmic parallelism

With this approach, each individual processor performs a specific task on blocks of data which pass through the processor network in a production line fashion. Though attractive as a concept, algorithmic parallelism encounters difficulties at the implementation stage which tend to reduce its efficiency. These include communication and configuration problems. In the former case, each processor has to receive different initialization instructions for its specific portion of the computational task. In the latter case, one configuration of inter-processor linkages may be appropriate for some stages of the work and inappropriate for others. This will remain a problem until the technology for dynamic reconfiguration of processor arrays becomes available. While quantitative comparisons are very hard to find, experiments at Southampton University on Monte Carlo simulations suggest that geometric parallelism gives a much better approximation to linear speed-up as the number of processors is increased, than does algorithmic parallelism (Bowler et al. 1987a). Whether a similar conclusion would apply to parallel implementations of topological map processing must be left as a subject for future investigation!

CONCLUSIONS

Parallel processing hardware is now reaching the stage where extremely high performance computers can be built in a modular and cost effective way, particularly if powerful processing chips with good communications links, such as the transputer, are used. As usual the pace of algorithm research and software development is lagging significantly behind the hardware. Of the recognised approaches to algorithm construction, geometric parallelism, possibly combined with limited algorithmic parallelism, offers the most promising route for initial work in parallel GIS processing. Early areas of investigation using the Meiko Computing Surface include computationally intensive three didmensional GIS problems and in future, polygon overlay algorithms. Beyond these a whole range of novel and indeed exciting possibilities present themselves for parallel search on large spatial databases, real-time data structure conversion, and new approaches to visualization and user interfacing.

REFERENCES

Baillie, C.F. 1988, Comparing Shared and Distributed Memory Computers: Parallel Computing, Vol. 8, pp. 101-110.

Bowler, K.C., Kenway, R.D., Pawley, G.S. and Roweth, D. 1987a, An Introduction to OCCAM 2 Programming, Chartwell-Bratt, Lund.

Bowler, K.C., Bruce, A.D., Kenway, R.D., Pawley, G.S. and Wallace, D.J. 1987b, Exploiting Highly Concurrent Computers for Physics : Physics Today, October Edition.

Bowler, K.C., Kenway, R.D., Pawley, G.S. and Roweth D. 1988, OCCAM 2 Programming Course Notes, Publications, Dept. of Physics, University of Edinburgh.

Dangermond, J. and Morehouse, S. 1987, Trends in Hardware for Geographic Information Systems : Proceedings Auto-Carto 8, pp. 380-385, ASPRS/ACSM, Falls Church, Va.

Herring, J. 1988, TIGRIS : Topologically Integrated Geographic Information System : Proceedings Auto-Carto 8, pp. 282-291, ASPRS/ACSM, Falls Church, Va.

Jesshope, C. 1988, Transputers and Switches as Objects in OCCAM : Parallel Computing, Vol. 8, pp. 19-30.

Kavouras, M. and Masry S. 1987, An Information System for Geosciences : Design Considerations : Proceedings Auto-Carto 8, pp. 336-345, ASPRS/ACSM, Falls Church, Va.

McKenna, D. 1987, The Inward Spiral Method: An Improved TIN Generation Technique and Data Structure for Land Planning Applications : Proceedings Auto-Carto 8, pp. 670-679, ASPRS/ACSM, Falls Church, Va.

Miklosko, J. and Kotov, V.WE. (Eds.) 1984, Algorithms, Software and Hardware of Parallel Computers, Springer-Verlag, Berlin.

Norman, M. 1988, Asynchronous Communication Course Notes, Dept. of Physics, University of Edinburgh.

Roberts, J.B.G., Harp, J.G., Merryfield, B.C., Palmer, K.J., Simpson P., Ward, J.S. and Webber H.C. 1988, Evaluating Parallel Processors for Real-time Applications : Parallel Computing, Vol. 8, pp. 245-254.

Sommerville, I. and Morrison, R. 1987, Software Development with ADA, Addison-Wesley, Reading. Mass.

Stroud, N. and Wilson, G. (Eds.) 1987, Edinburgh Concurrent Supercomputer Project Newsletter No. 3, Edinburgh

University Computing Service.

Treleaven, P.C. 1988, Parallel Architecture Overview : Parallel Computing, Vol. 8, pp. 59-70.

Verts. W.T. amd Lee, C. 1988. Parallel Architectures for Geographic Information Systems, Technical Papers ACSM-ASPRS Annual Convention, Vol. 5, pp. 101-107, ACSM/ASPRS, Falls Church, Va.

Yalamanchi S. and Aggarwal, J.K. 1985, Analysis of a Model for Parallel Image Processing : Pattern Recognition, Vol. 18, pp. 1-16.