Proceedings of the *FOSS/GRASS Users Conference 2004* - Bangkok, Thailand, 12-14 September 2004

Accelerating Raster Processing with Fine and Coarse Grain Parallelism in GRASS

Onil Nazra Persada*, Thierry Goubier*

* Curtin University of Technology, Sarawak, Malaysia onil.p / thierry.g@curtin.edu.my

1 Introduction

GIS technology is deployed across a wide range of applications, from relatively simple maps to applications that require huge computational power such as remote sensing data storage and processing, and natural phenomena such as fire, flooding, landslide, and pollution.

High quality modeling and data sources produce huge rasters and long processing times. Parallelizing such applications is a well researched field, with proven and efficient methods[8]. The dataset is usually large. As a consequence, raster processing algorithms are sensitive to memory limitations. Based on those observations, the use of a Beowulf cluster [10] for raster processing appears convincing. Each node in a cluster adds memory, cache, memory bandwidth, I/O bandwidth and a CPU, and communication overhead.

Fine grain parallelism and the use of massively parallel computation represent a promising solution. The remote sensing community has very early on used the power of parallel computers to process satellite and airborne data. But the number of cells to be processed is very high which by consequence requires an architecture containing a high number of processors. SIMD (Single Instruction Multiple Data) machines such as Connexion machine CM-2 and CM-200, and the MasPar machine MP-1 have been a good choice to process this kind of applications. But those machines, expensive and too specialized, have been abandoned.

Nowadays, current microprocessors integrate Multimedia units, which reproduce on a smaller scale this paradigm. This approach has been termed SWAR[2] (SIMD Within A Register) and allows fine grain parallelism on standard microprocessors. And the maturation of new computing devices such as FPGAs[1] and Reconfigurable Computing capable of efficient fine grain parallelism and tools offers another solution to fine grain parallelism[1].

Our approach is to used a hybrid Beowulf cluster where each node contains a microprocessor and a fine grain parallel processor. The coarse grain parallelism is implemented among nodes. The fine grain processor will run fine grain parallel tasks with a SIMD paradigm. Current microprocessors support SIMD processing through SWAR (SIMD Within A Register). They provide large registers that may be partitioned in smaller fields, and instructions that operate on all fields simultaneously.

The next section presents more aspects of raster based GIS applications, followed by a discussion on parallelizing this kind of applications. Further a beowulf cluster combined with fine grain processors will be presented. Some experiments and results will be discussed before concluding and discussing some future aspects of this work.

2 Raster Based GIS applications

Currently two data formats, vector and raster, are widely used in GIS applications. Vector based GIS applications manage irregular and complex spatial data, while in raster based applications, data are more regular. To limit our scope of work, this paper focuses only on raster spatial data.

[1] Field Programmable Gate Arrays

Raster processing algorithms use a regular dataset, a two-dimensional grid (that may be extended to three or more dimensions). Two types of raster processing express that regularity: map algebra (r.mapcalc) and cellular automata (CA) [11]. In map algebra, an expression will be applied to each cell of the input raster(s) to produce an output raster. In CA models, each cell of the raster is associated with a simple automaton, using it's current state and it's neighbors' states to update itself at each generation. CA have been used to model certain natural phenomena, such as forest fire, landslide, and sandpile [5, 7].

Remote sensing data falls into this raster category. The resolution of a raster depends on the real size on the ground represented by a pixel; the smaller the area, the higher the resolution is. High resolution data represents a huge number of pixels. By consequence, remote sensing generates huge amounts of data; a single band from an earth observation satellite can generate over 60MBytes per image, and this amount is bound to increase with each new satellite generation by the resolution of the data[2] and in the number of bands[3]. Raster data follows the same pattern, where the size is defined by the extent of the geographical region and the resolution of the data. Applications targeting large areas or long period of time will use low-resolution satellite data (500m or 1km on the ground).

Consequently raster algorithms are I/O intensive, Memory intensive and Computation intensive.

I/O intensive applications pose particular challenges. Storage speed has not increased at the same rate as memory speed has, making it more and more a challenge to extract high performance from files. Compression can lower the number of I/O operations by reducing the size of the file, at the expense of the compression/decompression computation in itself. Parallel I/O, or spreading the data over multiple storage units, is also a solution when the data distribution is done right.

Memory intensive applications require large amounts of fast memory to perform well. Two bottlenecks exist: first, current processors are far faster than main memory is[4] and mat be stalled waiting for data from memory. Caches are used to alleviate this, but they are small compared to a raster. Second, the dataset has to fit into main memory. If not, then only the part of the dataset in main memory can be processed; the remain has to be loaded out of disk, incurring I/O operations. *Out of memory* algorithms are designed to make this as efficient as possible. If multiple iterations over the dataset are necessary, this approach is very difficult to implement efficiently. Some programming techniques may be used to reduce memory requirements and avoid out-of-memory algorithms such as in memory compression or packing. If a dataset fits into main memory, the performance may be dictated by the speed of the memory (which is one or two orders of magnitude slower than current CPUs).

Computation intensive applications are rather well taken care of by current microprocessors, which implement a large palette of techniques for accelerating processing: super-scalar, deep pipelining, multimedia processing units, high frequencies, large and fast caches. However, the current high performance model is best suited to a large number of iterations over a small data set; they are far less efficient when dealing with few iterations over a large data set. The ability to compress raster data by packing it (i.e. packing multiple cells in one register) is important.

It remains to be seen how the software of a GIS can be designed to accommodate those requirements. We will touch this subject in part in our experiments.

3 Parallelizing Raster Based GIS

Based on the I/O, memory and computation power needed for raster based applications, the use of a Beowulf cluster for raster processing appears convincing. Each node in a cluster adds memory, cache, memory bandwidth, I/O bandwidth and a CPU, and communication overhead.

Parallelism can be done using a coarse grain approach, where each processor execute a relatively big part of a program. This way, the dataset is decomposed into parts and one processor will execute each part. With further decomposition, a part could be as small as a cell and the operations on one cell can be done by one processor. This leads to fine grain parallelism.

Some interesting properties can be seen from the operations needed for each point on GIS rasters justify the use of fine grain parallelism:

[2] current resolutions varies from 30m to 1m in commercially available data

[3] From multi-spectral (7 bands) to hyper-spectral(128 bands and more)

[4] accessing main memory may cost in excess of 200 processor cycles

Regularity : some processing steps will apply the same operation on all cells of a raster (regularity), whereas others will apply different operations on different cells (irregular).

Iterative : some algorithms will apply the same operation (or sequence of operations) to the raster multiple times. The number of iterations may be fixed in advance, or depend on the processing results (i.e. based on the convergence of the results).

Cost : the operation needed for each cell could be inexpensive or costly in terms of computation.

Variables : an operation may need a number of inputs and outputs; the types of the inputs will determine the width of the input in bits, and the number and the types of the outputs will define the width of the output. The result of the execution of an operation can be an intermediate result or the final result.

Stochastic : the operation may need some random values, or a random element can be added to the data. Under some constraints, stochastic operations can be transformed to deterministic ones with additional inputs.

Neighbouring : the operation may require data from neighbouring cells; CAs will usually consider either 4 neighbours or 8 neighbours.

Result : the result of the operation done on a raster, can be a raster or other kind of data such as statistic, raster to vector conversion.

3.1 Hybrid Beowulf Cluster

The particularity of our Beowulf cluster is that each node is hybrid, containing a microprocessor and a fine grain processor as shown in Figure 1. A fine grain processor will run fine grain parallel tasks. It could be a reconfigurable computing platform using FPGA or other kinds of processors such as the multimedia extension of a microprocessor (MMX) and graphic cards.

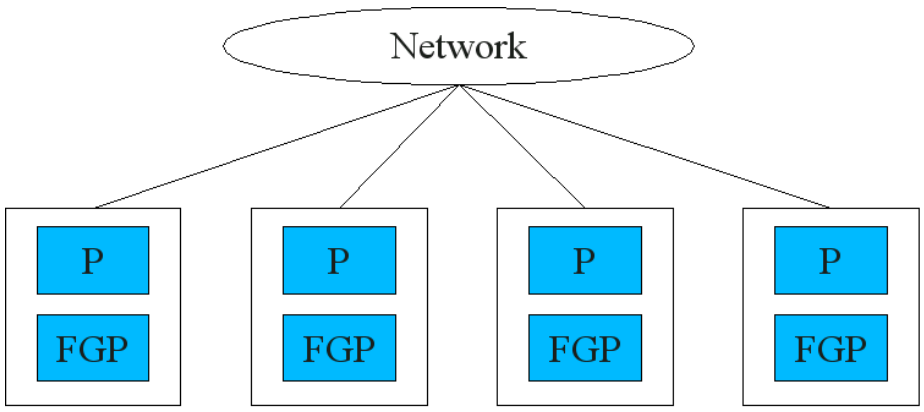

Figure 1: Hybrid Beowulf Cluster

The coarse grain parallelism is implemented among nodes. In the experiment, the hybrid Beowulf cluster is composed of 4 nodes, each node including a MMX functional unit used as a fine grain processor (Intel Pentium III with MMX/SSE extensions). The fine grain parallelism is obtained by implementing the core processing loop in SwarC[2]. The SwarC code is compiled to MMX assembler instruction with Scc, allowing parallel execution in the MMX unit.

The application studied is a cellular automaton to model hydrology rain run-off. The data is loaded from a GRASS database. The coarse grain parallelism on each node is obtained by partitioning the raster data map

in four strips. The cluster uses MPI (Message Passing Interface) to exchange the data between computing nodes and to time the processing.

3.2 SWAR

Raster data exhibit relevant characteristics for SIMD processing. Operations are often regular, require a limited neighbourhood, have few variables, have a narrow bit width (i.e. 8 bits for remote sensing; classified rasters such as land-cover may have 10 categories or less, requiring 4bits). They may be iterative (CA) or not (map algebra), and usually produce a raster as output. It is efficient and easy to map such operations on a SIMD architecture. Accordingly, mapping those operations to a microprocessor using SWAR is also efficient.

General purpose programming languages are unable to express SIMD operations. A common approach for commercial software is to hand-code the relevant parts of processing in MMX or SSE assembler instructions. Such hand-coded routines may also be found in high-performance math libraries. Another approach is to use a specific compiler and language to express those operations. Scc[2], the SWAR C Compiler from the University of Purdue, is one of them. It uses a form of C parallel, or a C like syntax with SIMD operators. It allows the expression of the operation in a common way, but generates SIMD code for standard microprocessors that may be integrated in a C program.

SWARC is a data parallel language with a C-like syntax. An operator such as = applied to an array is actually applied to all elements of the array, one by one or in one packed MMX instruction. Scc will compile the SWARC source code to an intermediate form of C and MMX assembler macros before calling to gcc for the remaining steps. This sample code illustrate the techniques used when dealing with SWAR: the data size is reduced as much as possible (going from 32bits int to 16bits) to allow Scc to pack more operations in one register. This allow us to pack 4 cells in 64bits (MMX; different extensions may use 128 bits registers), improving the efficiency of the code.

The peak performance of the multimedia unit of a microprocessor is highly implementation dependent, but we may estimate it in comparison with normal instructions under some hypothesis. We consider that the CPU can emit MMX or normal instructions on a one to one basis; that the latency is the same, and given that the register size is larger in MMX, the gain is then equivalent to the number of units that may be packed into a MMX register. But we have to be aware of the limitations.

- Packing the data into registers can prove complex and inefficient. The current processing chain in GRASS is loading a raster compressed from a file, extract it as an array of 32bits integers, and then pack it into a 8bits integer array before being processed by the SWAR code.
- SWAR allows to accelerate processing, but the microprocessor may already be limited by its memory interface. The gain of a SWAR solution may then be negligible. However, this may be used in a parallel system by using a smaller, cooler processor, we can increase the density of computation units.
- To really benefit from the gain of small bit widths in raster operations, it is necessary to compile dynamically the code. Dynamic compilation is still an area under development.
- Available tools are limited. Scc, the SWARC compiler, has not integrated SSE and SSE2 support; tools for reconfigurable computing are even farther away from production work.

4 Experiments

4.1 Rain Water Run-off simulation

The target problem we intend to solve is a cellular automaton simulating the flow of rain water fall-off on a landscape [4]. It takes a digital elevation model or Dem and a raster of initial water levels, and iterates. Each iteration computes a simple flow transfer equation between cells based on their height, that is elevation plus water level in the cell. It's a Von Neumann neighbourhood with four neighbours (north, south, east, west).

The simulation has the following characteristics:

- An elevation map contains constant data and each cell consists of one elevation value.
- Each cell has an initial value of water level. This value is set to the same default for simplicity.
- The water level of a cell changes each discrete time step according to the following rules :
  - If the height of the cell is greater than the elevation plus the water level of a neighbouring cell, a portion of its water is drained into this neighbour.
    - If the elevation of the cell is bigger than the height of this neighbour, a part of the cell's water level will be drained into the neighbour, otherwise a part of the difference between the heights of the two cells will be drained into the neighbour.
  - If the height of the cell is less than the elevation plus the water level of a neighbouring cell, a portion of the water of the neighbour is drained into the cell.
    - If the elevation of the neighbouring cell is greater than the height of the cell, a part of the neighbouring cell's water level will be drained into the neighbour, otherwise a part of the difference between the heights of the two cells will be drained into the cell.

4.2 Parallelizing methods

The elevation map is decomposed into parts in strip and each processor works on its part. To calculate the state of the cells in the border of each part, the state of the neighbouring cells from the other parts are needed. The exchange of the states of these bordered cells is done by calls to the MPI library [6]. This implementation is similar to [9].

4.3 SWAR

For our SWAR experiments, we use the MPI timing functions to measure the processing time and the communication time.

This implements, on a single node, the cellular automaton defined in the previous section. The C code process the raster data with a moving window of 3 rows and 4 columns. The SWARC code is given the window and effectively computes the 2 middle cells. The results are shown in table 1.

```
void c_calc(int:16 [8] middleElev,
        int:16 [8] upElev,
        int:16 [8] downElev,
        int:16 [8] middleWater,
        int:16 [8] upWater,
        int:16 [8] downWater,
        int:16 [8] newWater)
{
        int:16 [8] tmp;
        int:16 [8] h;
        int:16 [8] min;
        int:16 [8] max;
        tmp = middleWater;
                /* For up */
        min = -(middleWater >> 2);
        max = upWater >> 2;
        h = (upElev - middleElev) * 25 + ((upWater - middleWater) >> 2);
        tmp += h > max ? max : (h < min ? min : h);
                /* For down, left and right ... */
                /* Shift to align */
        newWater = tmp[>>1];
}
```

4.4 Results

Three implementations are compared. The first one uses integers (C int). The second one uses floating point (C float). The third one uses Scc and code for 16bits integers (C short int). The dataset used is the Spearfish dataset on a GRASS 5.0.3 installation. The program is compiled with MPICH and the grass dynamic libraries.

The processing time is computed, excluding the time spend synchronizing and exchanging data between nodes. The time needed to exchange and synchronize is recorded in table 2. There is no load balancing between the nodes, the calculations being regularly distributed over the raster.

Number of Nodes	int	float	SWAR
1	32.09	25.02	7.49
2	17.20	13.41	3.75
4	8.90	6.95	1.87

Table 1: Run times of the CA in seconds

Number of Nodes	int	float	SWAR
1	0	0	0
2	0.63	1.45	0.07
4	1.52	2.06	0.23

Table 2: Communication times in seconds

The result of the cellular automaton after 100 generations is shown in figure 2.

4.5 Analysis

In this experiment, the communication cost can be further reduced by overlapping computation and exchange of the borders between the stripes. As long as the time needed to exchange the borders is smaller than the computation time for one stripe on each processor, then the communication cost can be negliged.

For GRASS practitioners, the question is whether a parallel version of main raster algorithms is worth. This depends on the efficiency of the parallel code, the ease of setup and the cost benefits of multiple nodes. Originally, beowulf clusters were developed as a low-cost alternative to high performance scientific workstations. Our results shows that a small scale cluster can provide a linear speedup; the Scc compiler is also portable to alternative platforms (MacOS X with Altivec, SPARC, MIPS and PA-RISC).

5 Conclusion and Future works

As a conclusion to these results, we can say that SWAR is an effective way of increasing the performance of raster programs by using fine grain parallelism. SWARC and Scc allow a programmer a simple, yet effective way to benefit from the added performance. In combination with coarse grain parallelism, they offer substantial speedup on raster operations.

Parallel I/O or compression techniques can be used if the volume of data to be processed is important.

Acknowledging the limitations and results shown here, we have to improve the efficiency of the SWAR code generation, particularly its support for recent microprocessors as well as providing a more general framework for exploiting fine grain parallelism in a raster-based GIS.

References

[1] Mike Estlick, Miriam Leeser, James Theiler, and John J. Szymanski. Algorithmic transformations in the implementation of k- means clustering on reconfigurable hardware. In *Ninth international symposium on Field programmable gate arrays*, pages 103–110. ACM Press, 2001.

[2] Randall J. Fisher and Henry G. Dietz. Compiling for SIMD within a register. In *Languages and Compilers for Parallel Computing*, pages 290–304, 1998.

[3] Loic Lagadec and Bernard Pottier. A lut based high level synthesis framework for reconfigurable architectures. In *11th IEEE/ACM International Workshop on Logic & Synthesis*, pages 167–172, June 2002.

[4] Shapiro M. and Westervelt J. R.mapcalc: An algebra for gis and image processing. Technical report, US Army Construction Engineering Research Laboratory, USA, 1992.

[5] Bruce D Malamud and Donald L. Turcotte. Cellular-automata models applied to natural hazards. *Computing in Science and Engineering*, pages 42–51, May/June 2000.

[6] *MPI: A Message-Passing Interface Standard*, 1995.

[7] Goncalves P.P. and Diogo P.M. Geographic information systems and cellular automata: A new approach to forest fire simulation. In *EGIS*, 1994.

[8] Healey R., Dowers S., Gittings B., and Mineter M. *Parallel Processing Algorithms for GIS*. Taylor & Francis, London, 1998.

[9] William Emmanuel S. Yu Rafael P. Saldana, Winfer C. Tabares. Parallel implementation of cellular automata algorithms on the agila high performance computing system. In *Proceeding of the International Symposium on Parallel Architectures, Algorithms and Networks (ISPAN'02)*, 2002.

[10] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the power of parallelism in a pile-of-pcs. In *Proceedings, IEEE Aerospace.*, 1997.

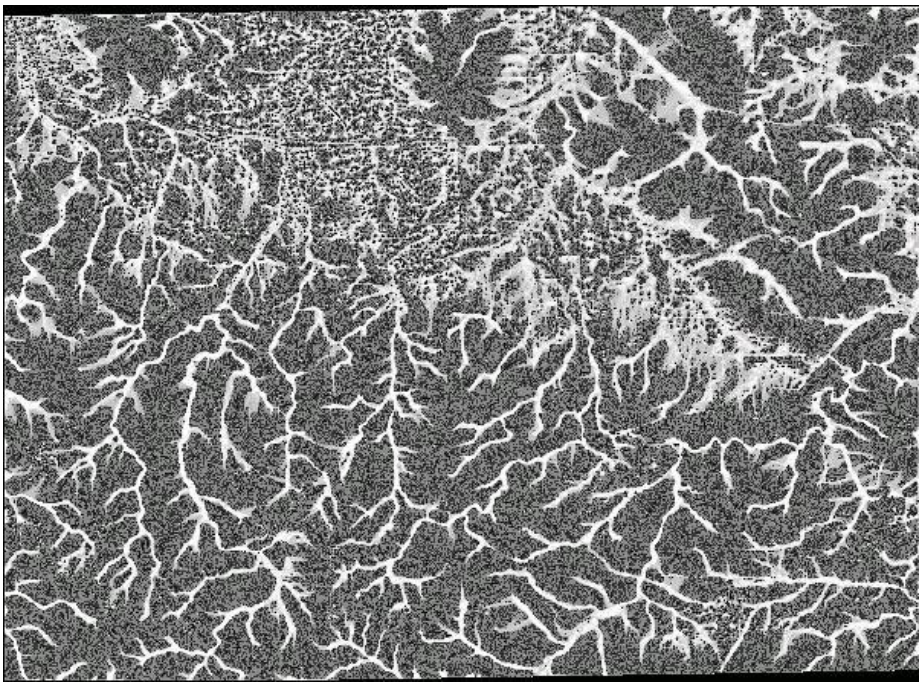

Figure 2: Result of the CA

[11] S. Wolfram. Computation theory of cellular automata. *Communication in Mathematical Physics*, 96:15–57, November 1984.