Parallel Computing of Catchment Basins in Large Digital Elevation Model

Hiep-Thuan Do*, Sébastien Limet, and Emmanuel Melin

LIFO – Univerité d'Orléans
Rue Léonard de Vinci, B.P. 6759 F-45067 ORLEANS Cedex 2
{hiep-thuan.do, sebastien.limet, emmanuel.melin}@univ-orleans.fr

Abstract. This paper presents a fast and flexible parallel implementation to compute catchment basins in the large digital elevation models (DEM for short). This algorithm aims at using all the specific properties of the problem to optimize local computations and to avoid useless communications or synchronizations. The algorithm has been implemented in MPI and the first benchmarks show the scalability of the method.

Key words: Parallel implementation, Digital elevation model, Catchment basin, Image Segmentation

1 Introduction

Geo-hydrology is an interdisciplinary domain dealing with the flow of water through aquifers and other shallow porous media. This domain is a branch of earth sciences relevant to many fields of soil science like agriculture or civil engineering. To study the flow of water one method consists in computing *catchment basins* only considering surface topography as if soil was impervious. The input information consists in a *Digital Elevation Model* (DEM).

Computing catchment basins is a problem very close to the well known *watershed transform* method in the field of mathematical morphology. This is a region-based segmentation approach widely studied in the area of *Image Analysis*. *Image Segmentation* is the process of partitioning the image into disjoint regions that are homogeneous with respect to some property such as gray value, altitude or texture. Indeed, a gray scale image can be viewed as a DEM where the lengths of the gradients between pixel values are considered as the altitude of the corresponding points. Catchment basins of the images denote the expected homogeneous regions of the images and ridges of land that separate catchment basins are called *watersheds*. As we see, the intuitive idea underlying watershed transform method comes from geography, indeed we propose to reverse this analogy by using watershed transform methods, wildly tuned for Image Analysis, to initial geographical catchment basin computing.

Watershed transform algorithms use two method classes based on flooding process [8, 1] or rain falling simulation [6, 5]. These methods are sequential and need an extensive use and a careful management of the memory. Required memory size is a priori unknown, and the data are addressed in unstructured manner, causing performance degradation on virtual memory.

* Hiep-Thuan Do's PHD Thesis is funded by Conseil General of Loiret (France)

The parallelization of classical sequential algorithms based on an ordered queue proposed in [2] requires strong synchronizations between the processors maintaining local queues and repeated labeling of the same pixels performed for appropriately labeling parts of catchment basins. To compute watershed, some other parallel methods [3, 4] have been proved scalable, but are still computationally expensive for large images.

The quality of the results of catchment basin computing depends on the quality of the DEM in the one hand and on the surface it covers on the other hand. Since several years, very large high resolution DEM issued from satellite or air plane LIDAR scanning, are available. Then it becomes crucial to focus on fast scalable parallel methods being able to perform catchment basin computation on very large of datasets.

The goal of this paper is to present a SPMD parallel implementation onto MIMD architectures to efficiently computes catchment basins of large DEM. This approach is mainly based on local data analysis. This makes possible a very efficient parallelization. We use a *dependency graph*, which encapsulates results of local computations and reduces drastically the cost of communications related to global analysis steps. This method overcomes synchronizations and communication overheads due to no local information needed by nodes to take local decisions. Moreover we adapt the *path compression* part of algorithm *Union-Find* [7] in order to, first, compute connected components, and secondly, resolve the global dependency graph. Further tests on different large DEM have been performed to evaluate the efficiency and the scalability of the method.

The paper is organized as follows. We describe the detailed implementation of our new algorithm in Section 2. Experimental results are sketched in Section 3. Finally, we close with a conclusion.

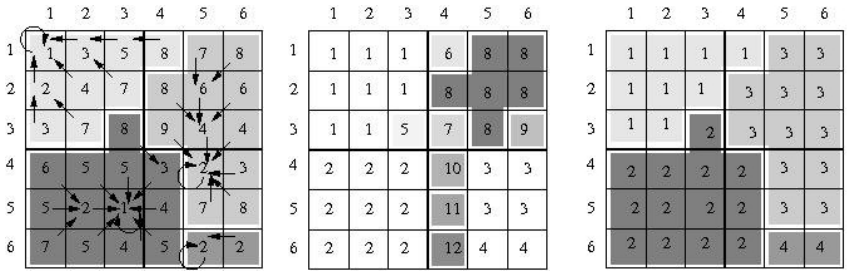
2 The parallel watershed algorithms

A *Digital Elevation Model* (DEM) is a regular square 2D grid of vertices representing geographic positions, with 4-connectivity or 8-connectivity where an altitude is associated to each vertex. Formally, the *grid* is a triple $G = (D, E, f)$, where (D, E) is a graph and f is the function giving the altitude of the nodes of D. The graph (D, E) consists of a set $D \subseteq Z^2$ of *vertices* (or *nodes*) and a set $E \subseteq D \times D$ of pairs of vertices describing the connectivity of the grid. Each node (x, y) represents a 2D geographic position. Each $p \in D$ has an altitude given by the function $f : D \longrightarrow \mathbb{R}$ assigning a float value to p. Therefore f denotes the topographic relief. We define a *neighboring* for each node p with respect to the grid G, denoted by $N_G(p) = \{p' \in D | (p, p') \in E\}$. In the case of 8-connectivity $E = \{((x, y), (x', y')) | (x, y) \neq (x', y')$ and $|x - x'| \leq 1$ and $|y - y'| \leq 1\}$.

Our method to delimit catchment basins of a DEM is in the class of arrowing technique. It consists in exploring node neighborhood, choosing the lower neighbor and iterate the process. We follow the path of steepest downslope through the grid until reaching a final point that has no lower neighbor. This point is a local minimum and is called a *well*. Vertices having the same final well belong to the same catchment basins. We formalize *the steepest downslope* via

the *parent* function. Each node $p \in D$, $parent(p)$ is the neighbor of lowest altitude. Note that $parent(p)$ may be p itself if all neighbors are upper. If two neighbors have the same altitude, we enforce an artificial strict order using the classical lexicographical order $\prec$ on pairs. This order eliminates *flat zone* without any specific processing. The practical results show that computed catchment basins are consistent with regard to hydro-geology. Figure 1(a) gives an example of a DEM, the arrows between nodes represents the function *parent*. In this DEM, the pixel $(6,5)$ and $(6,6)$ form a flat zone. Because of lexicographic order, $parent((6,6)) = parent((6,5)) = (6,5)$.

Formally, the parent q of the node p is the element of $N_G(p)$ that verifies: $\forall p' \in N_G(p) \cup \{p\}$ *such that* $p' \neq q$ *either* $f(q) < f(p')$ *or* $f(q) = f(p')$ *and* $q \prec p'$



(a) DEM (b) Local labeling (c) Global result

Fig. 1. Example of DEM

We call *flow graph*, the graph $Flow = (D, \{(p, parent(p)\})$. This graph is partitioned into connected components (see Figure 1(a)) that are almost trees (to be real trees, the edges of the form (p,p) have to be removed). In Figure 1(a), the four connected components have different backgrounds. The root of a tree in $Flow$ is such that $parent(p) = p$ and corresponds to a local minimum. In the example, the four wells are nodes $(1,1)$, $(4,5)$, $(5,3)$ and $(6,5)$. We define a *catchment basin* CB as a connected component in $Flow$. Then, a catchment basin contains all nodes of D that are in the same tree of $Flow$. The root rp of catchment basin CB is called *primary well*.

As in [4], our parallel approach maps data onto the processors of the parallel computer. The global domain D of a grid $G = (D, E, f)$ is partitioned into disjoint sub-domains D_i. In the example illustrated by Figure 1(a), the domain is split into four sub-domains symbolized by the thick lines in the grid. For example the sub-domain D_1 is the set $\{(x,y) \in Z^2 | 1 \leq x \leq 3 \; and \; 1 \leq y \leq 3\}$.

Each sub-domain D_i is extended with a so-called *extension area* denoted ${D_i}^+$. The extension area consists of all neighbors of nodes belonging to D_i that are not included in D_i. It is defined as follows: ${D_i}^+ = \{p \in D | p \notin D_i \; and \; \exists q \in D_i \; such \; that \; p \in N_G(q)\}$. In the example of Figure 1(a), $D_1^+ = \{(4,1),(4,2),(4,3),(4,4),(1,4),(2,4),(3,4)\}$.

This data distribution has consequences when we want to locally compute catchment basins since the nodes of one basin can be distributed on several sub-domains as it can be seen in Figure 1(a). In such cases, it becomes impossible to compute the real tree without global data access. This kind of access is time consuming, so to avoid them our method computes a *partial tree* and postpones the problem to another phase of our algorithm. Among the local wells computed in a sub-domain, it is possible to locally detect primary wells of the global domain thanks to the extension area. Indeed a local well that has a parent in the extension area is not a primary well, it is then called *secondary well*. In Figure 1(b), the upper right sub-domain has four local wells $(1,4)$, $(3,4)$, $(3,5)$ and $(3,6)$, but all are secondary since their parents are in the extension area.

At this point, labels are locally given to both primary and secondary wells in a such way that labels of each sub-domains are disjoint. The method gives to all the nodes of each catchment basin the label of the local well. Figure 1(b) gives the result of this phase on the DEM of Figure 1(a). Nodes labeled with secondary well have gray background.

Until now, all computations can be performed without any communication between processors of the parallel computer. This is optimum for scalability and speed up. Moreover the local computations use a path compression technique inspired of the *Union-Find* algorithm [7]. Tarjan had shown that the time complexity of this step for an input image of size N_i, is quasi $O(N_i)$. So, in practice, this algorithm can be regarded to run in linear time with respect to its input. It does not need any extra memory space.

Next phase in our method leads to relabeling of secondary wells taking into account no local informations. During this phase each secondary well is coalesced into a primary well. We start with one to one communications in order to update pixel labels of the extension area. After this step we locally know in which catchment basin, pixels of the extension area belongs to. Then we introduce the *parent well* notion. A well pw is called *parent well* of a secondary well sw if $parent(sw)$ belongs to the catchment basin corresponding to well pw. We create a local dependency graph LDG_i which contains dependency informations between local catchment basins. We locally associate any secondary wells sw with its parent $parent(sw)$. In Figure 1(b), we get $LDG_1 = \{5 \rightarrow 10\}$, $LDG_2 = \{6 \rightarrow 1, 7 \rightarrow 3, 8 \rightarrow 3, 9 \rightarrow 3\}$, $LDG_3 = \emptyset$ and $LDG_4 = \{10 \rightarrow 2, 11 \rightarrow 2, 12 \rightarrow 2\}$.

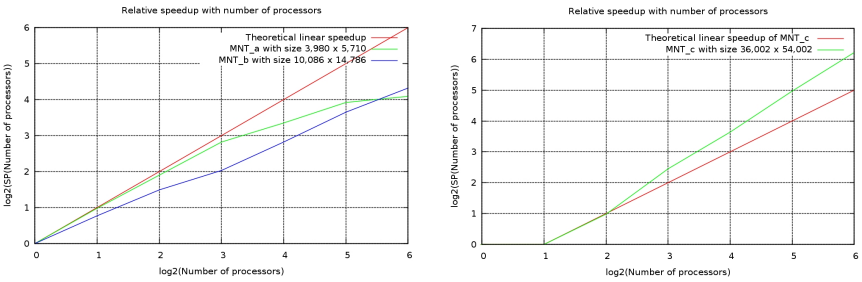
The local dependency graphs are then broadcasted to all other processors for building of *global dependency graph* GDG=$\bigcup_1^{P_{nb}} LDG_i$, where P_{nb} is the number of processors. The GDG is a graph which connected components are trees rooted by primary wells. On each processor P_i, catchment basin CB_{sw} corresponding to secondary well $sw \in LDG_i$ is merged into the catchment basin CB_{rw} corresponding to root well rw. This is called the union of wells. The union of wells is then propagated on the path between the root well rw and the secondary well sw in the GDG. This step is continuously repeated until all secondary wells are labelled with the identifier of their their roots (primary wells). Note that this process is parallel since each processor works on resolution

only for its own local secondary wells. Moreover we need only one communication phase involving data structures much more reduced compared to initial data size.

Finally, labels of nodes in the catchment basins corresponding to secondary well sw are replaced by label of its root well rw in GDG by the linear scan for all nodes in sub-domain D_i. This phase of our method is purely parallel without any communications. The global labeling of DEM is illustrated in Figure 1(c).

3 Experimental results

This algorithm described in this paper has been implemented in C++ using Open MPI. It has been tested on the **MIREV platform** consisting of eight nodes linked with a gigabit ethenet network. Each node is a bi-pro with AMD Opteron Quad-Core 2376 2.3Ghz with 16G SDRAM DDR2-PC5300 EEC 667 Mhz. We tested up to 64 processors on different *Digital Elevation Models* (MNT_a with size 3,980 x 5,710; MNT_b with size 10,086 x 14,786; and MNT_c with size 36,002 x 54,002) which are provided by the Company Géo-Hyd in Orléans, France.



(a) MNT_a and MNT_b Speedup (b) MNT_c Speedup

Fig. 2. Relative speedup with number of processors

Let P_{nb} be number of processors used. The running time T(P) is the time elapsed between the moment that the first processor starts and the moment that the last processor finishes. The measured times excludes data loading, distribution, coalescence, and saving. The *relative speedup* of the parallel algorithm is measured by: $SP(P_{nb}) = \frac{T_{min}}{T(P_{nb})}$ where T_{min} is the execution time of our algorithm onto one processor. Note that, due to memory size limitation, it is not possible to run our program for MNT_c onto one sole node. In this case only, we choose to fix T_{min} as the execution time of our algorithm onto two processors. Then, in this case, the speedup linear curve start at coordinate (1,0) and is parallel to the classical speedup linear curve (Figure 2(b)).

Looking at Figure 2(a), for MNT_a and MNT_b, we remark that the relative speedup is close to linear speedup at the beginning of the curves. Of course, when number of processors grows, the speedup partially decreases, since cost of communications becomes more important compared to local computation gain. With

a larger DEM (1.9 Gpoints for MNT_c) speedup increases linearly (Figure 2(b)) and the computation takes 92 seconds using 64 processors. This illustrates the good scalability of our approach. Note that MNT_c speedup is super linear. A possible reason is the cache effect resulting from the different memory hierarchies.

4 Conclusion

We presented an efficient and scalable parallel algorithm to compute catchment basins in large DEM. All computations are parallel and no complex data structures are constructed. Moreover, all data structures are distributed onto cluster nodes. Synchronizations and communications are localized between super steps of the algorithm to avoid time consuming during the core of the computation.

The algorithm described in this paper computes all catchment basins with regard to the DEM. Some of them are irrelevant from the hydro-geology point of view because they are too small or result of approximation of the elevation values. As in the area of *Image Segmentation* it is possible to create a hierarchy of catchment basins that would help to eliminate such problems.

Since principles of catchment basins computing in the field of geography is close to watershed transform method in the field of *Mathematical Morphology*, we think that our parallelizing method can be directly adapted to *Image Segmentation*. This would speed up segmentation for high resolution images.

References

[1] S. Beucher and F. Meyer. The morphological approach to segmentation: The watershed transformation. In E.R. Dougherty, editor, *Mathematical morphology in image processing*, volume 34 of *Optical Engineering*, chapter 12, pages 433–481. Marcel Dekker, New York, 1993.

[2] A.N. Moga. *Parallel Watershed algorithms for Image Segmentation*. PhD thesis, Tampere University Technology, February 1997.

[3] A.N. Moga, B. Cramariuc, and M. Gabbouj. Parallel watershed transformation algorithms for image segmentation. *Parallel Computing*, 24(14):1981–2001, 1998.

[4] A.N. Moga and M. Gabbouj. Parallel image component labeling with watershed transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(5):441–450, May 1997.

[5] S.L. Stoev. RaFSi - A Fast Watershed Algorithm Based on Rainfalling Simulation. In *WSCG*, 2000.

[6] H. Sun, J. Yang, and M. Ren. A fast watershed algorithm based on chain code and its application in image segmentation. *Pattern Recogn. Lett.*, 26(9):1266–1274, 2005.

[7] R.E. Tarjan. *Data structure and Network Algorithms*. SIAM - Society for Industrial and Applied Mathematics, 1983.

[8] L. Vincent and P. Soille. Watersheds in digital spaces: an efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, June 1991.