Parallel Computing of Catchment Basin of Rivers in Large Digital Elevation Model

Hiep-Thuan Do
LIFO - Université d'Orléans
Rue Léonard de Vinci, BP 6759
F-45067 ORLEANS Cedex 2, FRANCE
Email: hiep-thuan.do@univ-orleans.fr

Sébastien Limet
LIFO - Université d'Orléans
Rue Léonard de Vinci, BP 6759
F-45067 ORLEANS Cedex 2, FRANCE
Email: sebastien.limet@univ-orleans.fr

Emmanuel Melin
LIFO - Université d'Orléans
Rue Léonard de Vinci, BP 6759
F-45067 ORLEANS Cedex 2, FRANCE
Email: emmanuel.melin@univ-orleans.fr

***Abstract*—This paper describes a new fast and scalable parallel algorithm to automatically determine catchment basin of rivers in large digital elevation models (DEM for short). This algorithms is based on the construction of a minimal spanning tree, via a hierarchy of graphs, modeling the water route on the DEM. It does not need any preprocessing like stream burning on the initial DEM and tends to make the most of incomplete DEM. Efficiency and scalability have been tested on very large DEM.**

I. Introduction

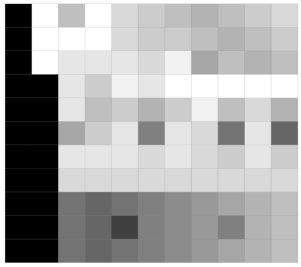
Geo-science reseach usually deals with very large datasets issued from satellite or air plane imagery now. Their interest is twofold, of course they can be directly used for visualization, but one can also extract informations from initial data, and then visualize outcomes and, if necessary, repeat the process to refine the results or obtain new ones. This analysis loop is convenient when each iteration is performed in a reasonable time i.e duration comparable to those usually observed on desktop computer for restricted datasets. A solution can be found with parallelism since one can use distant clusters to perform heavy computations and download results on its desktop. We propose to illustrate this scheme with an important issue in geo-hydrology: the catchment basin determination from digital elevation models (DEM for short).

In this paper we propose a method able to determine catchment basins of rivers from a DEM. This method combines the techniques used in different fields such as hydro-geology, image processing and graph theory in order to obtain an algorithm that both gives the most accurate results in term of geo-morphology and is efficient and scalable. Our method does not need any preprocessing like stream burning [1] on the initial DEM and we tend to exploit the most of DEM and avoid misleading inconsistencies they contain to extract catchment basins. Moreover we propose criterions to determine the most probable partition of catchment basin taking into account sea and border of the DEM which may belong to catchment basin of river outside of the datasets. This method is entirely parallel and scalable, it is a SPMD parallel implementation onto MIMD architecture, therefore it is possible to run it on a PC cluster to achieve reasonable computation time for very large DEM.

The paper is organized as follows. Section II presents the related work in geo-hydrology and we discuss analogy with similar problems issued from image segmentation including both sequential and parallel approaches. Section III describes our parallel algorithm. Experimental results are sketched in Section IV. Finally, we close with a conclusion.

0	20	15	20	17	16	15	14	15	16	17
0	20	20	20	17	16	16	15	14	15	16
0	20	18	18	18	17	19	13	15	14	15
0	0	18	16	19	18	20	20	20	20	20
0	0	18	15	16	14	16	19	15	17	14
0	0	13	16	18	10	18	17	9	18	8
0	0	18	18	18	17	18	17	16	18	16
0	0	17	17	17	17	17	17	17	17	17
0	0	9	8	9	10	11	12	13	14	15
0	0	9	8	5	10	11	12	10	14	15
0	0	9	8	9	10	11	12	13	14	15

(a) Example of DEM with a raster dataset



(b) Grayscale corresponding to the raster dataset

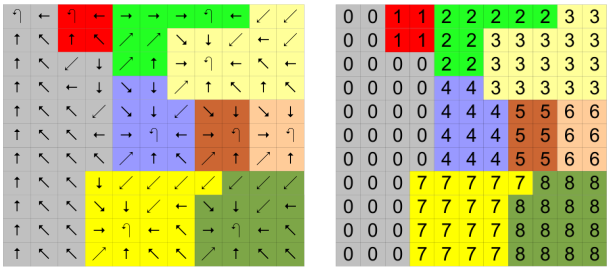
Fig. 1. Geological and image rasters

II. Catchment Basins from Different Points of View

Topography is often narrow linked to hydrology for example even before numerical era, geographers infer water presence from study of contour lines indicated onto maps. Numerical information greatly facilitates such analysis and a lot of methods have been proposed. The *raster* is a classical way to represent topography, it is a matrix of numerical values corresponding to heights of discrete points regularly positioned onto the terrain (Fig.1(a)). This relief map can be viewed as a digital gray scale image where the gray level of a pixel represents the altitude of that pixel (Fig.1(b)).

This information can be used to extract the *hydrographic network* making the assumption that rivers flow more probably into thalwegs and do not cross crests. Three steps are classically used:

1) Definition of a raster coding flow direction for each pixel. This produces a forest of disjoin trees those root is a local minimum (Fig. 2(a)).
2) Definition of a raster coding accumulated flows. This can be computed via the rain falling method.
3) Raster segmentation between river pixels and other ones, classically with the use of a fixed threshold.



(a) Raster flows of the DEM Fig. 1(a) (b) Catchment basins labeling of DEM of Fig. 1(a)

Fig. 2. Catchment basins

From this computation it is also possible to extract catchment basins corresponding to rivers in the form of a partition of the entire DEM where areas are separated by watershed lines (see Fig. 2(b)). Watershed lines are ridges of land that separate catchment basins. In each catchment basin any cell has a path in the raster flow to a given cell belonging to a river.

In this paper we focus on catchment basins computation based on the first step of hydrographic network computation.

To obtain a raster coding flow direction for each pixel, two approaches are possible : mono-direction and multi-direction. In mono-direction approaches, the water flows from one pixel to one sole pixel on principle of steepest descent. Two algorithms are used according to connectivity choices, D4 for a perpendicular connectivity and D8 if we add oblique neighborhood [2]. In multi-direction approaches, water is distributed to several pixels with lower height according to geometrical considerations. In both cases, decisions are taken from local informations and potential parallelism is not affected by the choice of the method. Unfortunately real datasets are not perfect, they include many incoherences like no-data, plateaus and sinks. In fact, raw DEM like those provided by the USGS, consist of a multitude of small sinks and plateaus.

A plateau is a flat area with at least one spill-pixel (see Fig. 1(a)). Authors generally chose to assign flow direction such as all pixels belonging to the plateau will flow to the spill-pixel. A sink is an area without spill-points (see Fig. 1(a)). In this case the problem is that water will accumulate in one pixel called the pixel sink. In this case we have to choose a global direction allowing water to climb the hill and to escape the sink. Since in both cases, flow needs to be redirected, and since uphill flow is counter-intuitive, many authors [2], [3], [4] choose to modify initial datasets to obtain artifact-less raster called in image-computing *lower complete* [5]. In hydrology, modifications of data, like *stream burning* [1], to better reflect known hydrology can be useful but may compromise the ability to derive catchment parameters and conduct to a second terrain analysis [6].

A better choice may be keeping data unmodified and forcing flow direction uphill. In [7], the authors propose to compute flow direction locally to find sinks or plateaus, this can be done in $O(N)$ time. They propose a GIS framework allowing to handle very large DEM via an out of core method to fit capacity of a desktop computer. With this aim in view, they work on efficient I/O accesses but not, as far as we know, on parallel computing. They manage plateaus and sinks via a flooding algorithm onto the terrain [8]. This approach mixes flooding and rain-falling methods, indeed it first uniformly pours water onto cells, progressively all sinks or plateaus in the terrain are filled, merged until a steady-state is reached. At this point every drip will flow outside sinks and plateaus to the border of the DEM. In [7], authors show that this intuitive point of view can be formalized in a flow routing problem between cells. Moreover it is possible to transpose this algorithm from pixels to watersheds, this resolves the problem much more faster since this uses a much more lightweight data-structure compare to entire large DEM. They build a *watershed graph*, an undirected weighted graph with a node for each watershed and edges between adjacent watersheds. Each edge has a label with the lowest elevation that occurs along the boundary between the two watersheds. They add a virtual watershed to the graph, called the *outside watershed*, representing the outside of the terrain. Secondly they build a flow graph between watersheds taking into account adjacency and labels. Finally, they search a path from each watershed to the outside watershed. This approach may construct cycles which must be removed at the price of expensive computations [8], [9]. In [7], the authors propose a solution by simulating the way terrain flooding occurs in real-life if level of the sea rise. The water gradually fills watersheds enforcing an order to the discover of paths from the outside watershed to inner watersheds. This approach is typically sequential.

We have shown that the key element of flow computation algorithms is the partitioning of the terrain into watersheds. Indeed it 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 methods based on either flooding process [10], [11] or rain falling simulation [12], [13]. These methods are sequential and classically may 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. In [10] authors describe a

hierarchical segmentation of images. This method prevents over-segmentation into too many watersheds, which is very detrimental for the quality of image analysis. This work tries to suppress the irrelevant boundaries on the watershed transform. In [14], Meyer claims that diverse hierarchies can be produced via flooding algorithms in order to offer the best scale of contours, favoring the contrast of the regions, their color, or their size. He notes that through tailored flooding, it is possible to favor some regions considered most important than others. In [15] is presented an efficient algorithm based on minimum spanning tree (MST) implementing a waterfall algorithm constructing a hierarchical segmentation approach. In one hand, this work emphasizes the efficiency of algorithms storing information in a very condensed structure like MST or graphs but on another hand the structure of the information and the flooding algorithm seems to be quite intrinsically sequential.

The parallelization of classical sequential algorithms flooding based on an ordered queue proposed in [16] requires strong synchronizations between the processors maintaining local queues and repeated labeling of the same pixels performed for appropriately labeling parts of catchment basins. Fortunately, making some assumptions, for example giving up to detect watershed pixels, makes watershed transform a local concept. Therefore, it is possible to compute watershed with scalable parallel methods [17], [18], [19]. They break the sequentiality of the watershed transformation solving the problem independently on all sub-domains distributed onto processors. If it is impossible to rule about boundary data, local informations are computed and they are solved during another sequential step on a master processor. Moreover, it is possible to use path-compression from the union-find algorithm due to Tarjan [20] to accelerate the computation of connected components in images whether it is pixels or basins [21]. Note that these parallel approaches have a sequential part and do not compute hierarchical watersheds.

As in [15], we modelize our concrete hydrology problem as the computation of a particular MST. Research in graph theory has wildly explored minimum spanning tree computation problem.The best algorithm [22] known for MST has a quasi-linear complexity but it uses a tree structure that have to be parsed in specific order which is intrinsically sequential. Borůvka MST algorithm has a complexity of $m \times log(n)$ (m and n being respectively the number of edges and vertices of the graph) but it has the nice properties to built a hierarchical decomposition of the and to allow nice speed-up for parallel architecture [23], [24].

In this paper, we present a SPMD parallel implementation onto MIMD architectures to efficiently computes catchment basins of large DEM. This algorithm is a parallelization of Borůvka MST algorithm that directs the hierarchical construction in a such way that the second to last level gives exactly the catchment basins of the rivers in the DEM. This approach makes a connection between algorithms issued from hydrology and those issued from image analysis.

Our algorithm is mainly based on local data analysis. This

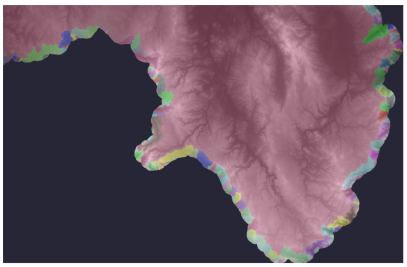


Fig. 3. Details of small catchment basins belonging to surrounded global catchment basins at the border of the DEM

makes possible a very efficient parallelization. All steps of the method are computed in parallel.

From hydrology:

- We evaluate a raster coding flow direction for each pixel with a D8 connectivity.
- We chose to solve plateaus with a simple lexicographic order which is a quick and good approximation when searching catchment basins in large DEM. More sophisticated approximations are also possible.
- We do not modify the dataset and we interpret sinks as watersheds partition of the datasets.
- We use a rain-falling approach onto pixels *and* onto watersheds to find flow direction
- We use a watershed graph enriched with a *outside watershed* to directs the merging of basins in a such way that catchment basins of each major rivers are isolated.

From Image Analysis:

- We use a hierarchical segmentation which construction is driven with the aim to build levels corresponding to catchment basins semantically consistent related to hydrology.
- We adapt the best parallel method using path-compression to build our hierarchy of catchment basins. For each level of the hierarchy we use the same parallel scheme.

From the Graph theory Point of view, we implement a parallel Borůvka MST algorithm. This kind of algorithm is based on a heriarchical construction of the MST using minors of the intial graph. The classical Borůvka algorithm may construct different hierarchies depending on some choices that are "don't care". In our method, we enforce the computation of one specific hierarchy where merged vertice are always included into global extension of a major river catchment basin to keep a hydrologic semantics. In this way, we obtain at the penultimate stage of the hierachy all the global extensions of major rivers of the DEM. This very desirable property for hydrologists does not break the correctness of the MST algorithm. The theoretical parallel runtime of our algorithm is in $O(n/p)$ where p is the number of processors and n the size of the DEM which garantees a good speed up for large DEM and the amount of communication is in $O(\sqrt{n/p})$.

III. The Parallel Watershed Algorithm

In this section, we first present the classic sequential Borůvka algorithm and its parallelization. Then we give some details on our implementation.

A. Sequential Borůvka algorithm

The graphs we deal with are *weighted graphs*. Let S be a set, possibly infinite, of values. A weighted graph G of type S consists of a set of *vertice* denoted $V(G)$ and a set of *weighted edges*, denoted $E(G)$, such that $E(G) \subseteq V(G) \times V(G) \times S$. For an edge (v_1, v_2, w) of $E(G)$, w denotes the *weight* of the edge. The *neighborhood* of a vertex v of G is the set $N_G(v) = \{v' \in V(G) | \exists (v, v', w) \ or \ (v', v, w) \in E(G)\}$. A graph H is a minor of the graph G if it is isomorphic to a graph obtained by zero or more edge contractions on a subgraph of G. An edge contraction removes an edge while simultaneously merging together the two vertices it used to connect. In our context, we are only interested in minors obtained by edge contractions on the graph G (and not a subgraph of G). To define, more formally the class of minors we deal with, let us introduce some few definitions. Let G be a graph, $\overline{V}$ a partition of $V(G)$. The set of weights of edges connecting $\overline{v_1}$ and $\overline{v_2}$, two elements of $\overline{V}$, is defined as $w_{G,\overline{V}}(\overline{v_1}, \overline{v_2}) = \{w | (\overline{v_1}, \overline{v_2}) \in \overline{V} \times \overline{V} \ and \ \exists v_1 \in \overline{v_1}, v_2 \in \overline{v_2} \ and \ (v_1, v_2, w) \in E(G)\}$. The minimal weight connecting two components $\overline{v_1}$ and $\overline{v_2}$ of $\overline{V}$ is $wMin_{G,\overline{V}}(\overline{v_1}, \overline{v_2}) = Min(w_{G,\overline{V}}(\overline{v_1}, \overline{v_2}))$ (we consider $Min(\emptyset) = \infty$). The *$wMin$-minor of G based on $\overline{V}$* is the graph which vertice are $\overline{V}$ and edges are the set $\{(\overline{v_1}, \overline{v_2}, w) | (\overline{v_1}, \overline{v_2}) \in \overline{V} \times \overline{V}, \overline{v_1} \neq \overline{v_2}, wMin_{G,\overline{V}}(\overline{v_1}, \overline{v_2}) = w \ and \ w \neq \infty\}$ i.e. all the vertice of a component of $\overline{V}$ are contracted and only the minimal edge between two components is preserved in the minor.

Given a graph G, the Minimum Spanning Tree (MST for short) problem simply consists in finding a spanning tree that minimizes the sum of weights of its edges. Borůvka's MST algorithm builds progressively a hierarchy of minors of the initial graph. It is based on the property that given two connected components of a graph G, the lightest edge connecting these two component belongs to the MST of G.

Step 1: For each vertex of G, select one of the lightest edges incident on it. To avoid cycle, if a minimal edge of a vertex is already selected while visiting another vertex, no new edge is selected.

Step 2: Compute the connected components of the graph restricted to the selected edges (e.g. by labelling vertice).

Step 3: Construct the $wMin$-minor based of G on the connected components of *Step 2*.

The minor graph obtained after this iteration is the input of the next one. The algorithm finishes when the minor is reduced to one unique vertex. The final MST is the union of the edges selected in *Step 1* at each iteration. Note that the *Step 1* chooses randomly the vertices from which it find the lightest edge. The hierarchy of minors may be very different according to implementation choices but the complexity remains the same i.e. $O(mlog(n))$ where m is the number of edges in the graph.

B. Parallel Borůvka algorithm

The preceding section leads one to assume that Borůvka algorithm is a good candidate for parallel computing. We give the principles of the method used for example in [24].

Step 1: For each vertex, choose the lightest edge incident on it. This step may be carried out locally if each participating process owns all edges associated to its local vertices. Avoiding cycles locally is possible using a total order on the vertices.

Step 2: Compute the connected components of the graph restricted to the selected edges. During this step each component is affected to one process. This step classically needs communications between processes.

Step 3.1: Each process computes locally the $wMin$-minor based on the connected components of *Step 2* of sub-graph it owns. This step is purely local.

Step 3.2: Each process collects the edges incident to one of the components it has been assigned.

C. Parallel Borůvka algorithm adapted to catchment basin computation in DEM

A *digital square grid* G with *domain* $D \subseteq Z^2$ containing values of type S where S can be any set (typically S is $\mathbb{R}$ for DEM) can be considered as a special kind of graph, where the vertices are called *points* or *pixels*. The heigth values of each vertex v is denoted h_v. G can be endowed with a graph structure $G = (V, E)$ by taking for V the domain D, and for E the set $\{((x_1, y_1), (x_2, y_2), w) | \ (x_1, y_1)$ and (x_2, y_2) are connected and the weight of the edge w is $max(h_{(x_1,y_1)} h_{(x_2,y_2)})\}$. The connectivity in the grid may be either 4-connectivity, or 8-connectivity. We call this first graph G_0.

We denote $\prec$ the lexicographical order on pairs of integer, i.e $(i, j) \prec (k, l)$ iff $(j < l)$ or $((j = l)$ and $(i < k))$.

The order $\prec$ is used to choose one minimal edge when several are possible. This refers to the plateau problem described Section II. Our version of Parallel Borůvka algorithm distributes vertice amoung nodes of the parallel machine.

We choose a block-distribution for vertice of G_0. Note that this distribution minimizes the number of *extern* edges connecting two vertice belonging to different nodes. In a bloc of size i vertice we have $O(i)$ edges and only $O(\sqrt{i})$ extern edges. To minimize computations we add to each bloc the vertice of extern edges. The block affected to node p is called the domain of p and the vertice of extern edges, the extention aera.

Our implementation of *Step 1* consists in selecting for each vertex v the lightest edge (v, v', w) if and only if $h_{v'} \prec h_v$. If several edges may be selected, we choose (v, v', w) such that $h_{v'}$ is minimal (the water goes down the deeper slope). If we find several equal $h_{v'}$ value then v' is the first in the lexicographical order (the lexicographical makes sense even for vertice of minors since they are all labelled with one pixel of the initial DEM) . Note that our first step does not need communications to guarantee that we do not construct cycles since we impose $h_{v'} \prec h_v$.

The selected edges are considered as oriented for the *Step 2*. Selecting the lightest edge models the flooding process, i.e. when pouring the basin represented by v it will overflow via the lowest border (represented by the weight). Imposing that $h_{v'} \prec h_v$ models the rain falling, i.e. when the basin overflows, the water should fall down.

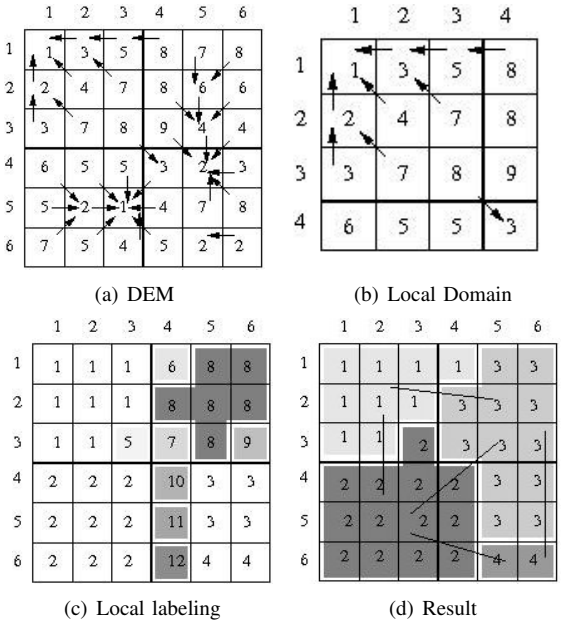
Step 2 consists in labelling the vertice to detect connected components of the graph restricted to the selected edges. This is implemented in three steps.

- First, each process computes locally its connected components, each component is identified by its root called *well* (i.e. the minimal vertex in the component w.r.t. height values defined above). This is done using a method inspired from the *Union-Find* algorithm of [20] where Tarjan proposes a *Path Compression* technique to optimize this task. Unfortunately, a connected component of the whole graph may be distributed on several nodes. So each processor distinguishes primary wells (i.e. wells that are in its domain) and secondary wells (i.e. wells that are in its extention aera).
- Exchange the labels of the vertice of extension
- Compute a *local dependency graph* LDG which vertice are labels and which oriented edges are of the form (l_1, l_2) where l_1 is the label of a secondary well. Such an edge indicates that the component of l_1 and l_2 belongs to the same component of the whole graph. Each processor broadcasts its LDG to the others. After this communication phase, each processor constructs the *global dependency graph* GDG by merging all the LDGs. One can remark that GDG is a forest which roots are primary wells. Using for the second time Path Compression technique, each node computes the final label of their secondary wells.
- The final phase is purely parallel. It consists in a local linear scan of all vertice which label is a secondary well. We replace their label by the one found thanks to the GDG.

Step 3 is implemented as described in Section III-B. Each processor broadcasts the connectivity of the new vertice that have external edges. In this way, each processor is able to know the whole connectivity of vertice it is in charge of. In this phase, only the smallest edge between two vertice is kept. Note that we only broadcast one edge for a connectivity between *new* vertice and not all possible edges as in [24].

Notice that the initial graph G_0 is the DEM itself and is treated such as by our implementation since it is the most compact representation of this graph that may contains several millions of vertice. Other graphs are represented by classical adjacency lists.

Figure 4 illustrates the computation of one level of the hierarchy. Figure 4(a) gives the initial graph, the number in the cells are the elevation data. The DEM is distributed on 4 processors which domains are delimited by the bold lines. The extention aera of the first processor is given Figure 4(b). On the same figure are drawn the select edges and their orientation. The vertice of this domain belongs to two different connected



(a) DEM (b) Local Domain

(c) Local labeling (d) Result

Fig. 4. Parallel process

components of the whole graph. The node detects that the vertex $(1,1)$ is a primary well, while $(4,4)$ is a secondary one since it is in its extension aera. After exchanging labels of vertice of the extention aera, each processor computes the following local dependancy graphs. $LDG_1 = \{5 \rightarrow 10\}$, $LDG_2 = \{6 \rightarrow 1, 7 \rightarrow 3, 8 \rightarrow 3, 9 \rightarrow 3\}$, $LDG_3 = \emptyset$ and $LDG_4 = \{10 \rightarrow 2, 11 \rightarrow 2, 12 \rightarrow 2\}$. The global dependency graph allows to know the primary well of each secondary well, in particular that $5 \rightarrow 2$. Figure 4(d) gives the result after the relabeling of secondary wells. The connectivity of the resulting graph is represented by the lines. Notice that the domain of the first processor is now the vertex labelled 1 and its extention aera is vertice 2 and 3.

Figure 5 illustrates the computation of the whole hierarchy (self arrows are to be ignored) on the DEM given Figure 1(a). One can remark the well labelled 0 (which figures the sea) grows rapidely until uncompassing the whole DEM. The resulting hierarchy does not fit our needs since we cannot distinguish catchment bassins we are looking for. This is why we introduce next section new rules in the hierarchy construction to avoid merging of river catchment basins.

D. A tailored hierarchy construction

Since we rarely compute the global catchment basins at a continent scale, datasets involving major river catchment basins necessarily include part of seas, no-data and part of incomplete catchment basins corresponding to surrounding major river outside of the dataset (see Fig. 3). The hierarchical method must be driven to avoid misinterpretation of these secondary catchment basins.

Our hierarchical method states that basins are merged if they belongs to the same connected components of the graph of *Step 1*. This guarantees that these basins belong to a

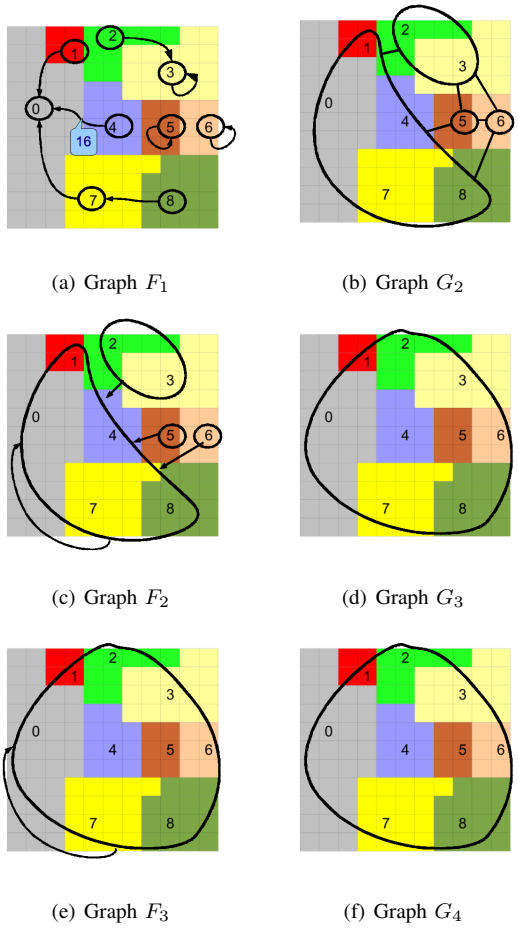


Fig. 5. Different graphs computed at each step

.

hydrological unit since they have common frontiers and they all flow to the same local minimum. A problem may arise when a basin is at the border of the DEM since no information about its neigborhood outside the DEM is known. Without any special treatments, such basins will be integrated at a certain level to a basin inside the DEM which is wrong when their natural spill-point is not in the dataset. To overcome this problem, as in [7], we introduce an *outside watershed* called Ω. Intuitively it represents the last final sink were all rivers flow, that is sea in real life. One just have to decide that all pixels under 0 meter (or another arbitrary value) belong to Ω as well as virtual pixels in the neighborhood of the DEM. All these pixels receive a height value less than each pixel of the DEM. With this rule, a connected component at the border of the DEM has Ω component as neighbor so it can flow to Ω.

Figure 6 represents the partition graph with the *outside watershed* Ω. Pixels with height 0 belongs to Ω and all vertices at the border of the dataset are connected to Ω.

Ω may avoid a problematic merge of distinct hydrologic basins but it introduces a new problem. Indeed, since Ω is the lowest point of the DEM, the neighborhood of Ω tends to fuse to Ω at every level of hierarchy which is also unwanted since Ω

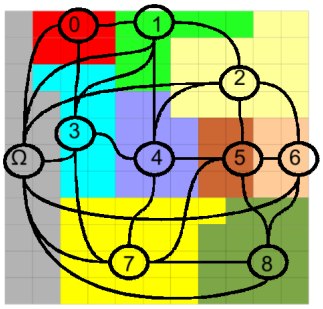


Fig. 6. First derived Connectivity Graph with *outside watershed* Ω with dataset of Fig. 1(a).

does not really belong to the DEM. This is why instead of the fusion to Ω, basins that should select an edge to Ω, do not but are marked. Such vertex is said ω marked or ω_m. In the *Step 1* of the algorithm, ω_m-vertice do not select any edges since it is known that the minimal one goes to Ω. When computing the minor graph, vertice of the new graph containing an ω_m vertex of the old graph is ω_m. With this simple rule, we direct the construction of the minimal spanning tree in such way that its root is Ω and its first children are the basins which spill points are Ω, i.e. either basins that really goes to the sea, or are at the border of the DEM. We compare the flow graph obtained with this rule (Fig. 7(a)) and without (Fig. 5(a)). In Figure 7(a), vertice 0, 1, 2, 6, 8, 7 and 3 are not merged with Ω since they are ω_m.

This method may lead to configurations, where a basin $\overline{\omega_m}$ is surrounded by basins higher than it. Usually this happens near quasi flat zones. In this case it is enough to impose that such an isolated basin selects the minimal edge to one ω_m neighbor. In our example, it happens for vertex 4 (Fig. 7(a)). Its height is 10 and its outlet to vertex 3 is 16 high and to vertex 5 is 17 high (see Fig. 1(a)). Vertex 3 is upper than vertex 4 and vertex 5 is lower than vertex 4. Figure 7(a) illustrates that we enforce the flow upper from vertex 4 to vertex 3 in this configuration. Finally, we find global coherent basins just before the upper stage in the hierarchy. In Figure 7(e) we obtain at the final stage the fusion of catchment 3, 4 and 5. Vertices 0, 6, 8, 7 are borders of this catchment basin. A part of a real case is presented in Figure 3, we have small catchment basins at the border of the DEM. In Fig. 8(a), we have the big catchment basin of the Loire River well distinguished, in purple, from other basins directly connected to the Atlantic ocean or at the border of the DEM. This catchment basin extracted from DEM, totally fits with the real catchment basin displayed in Figure 8(b)

Notice that these new rules do neither change the complexity of the algorithm nor break its parallelism.

IV. Experimental results

This parallel algorithm described in this paper has been implemented in C++ using Open MPI. It has been tested on

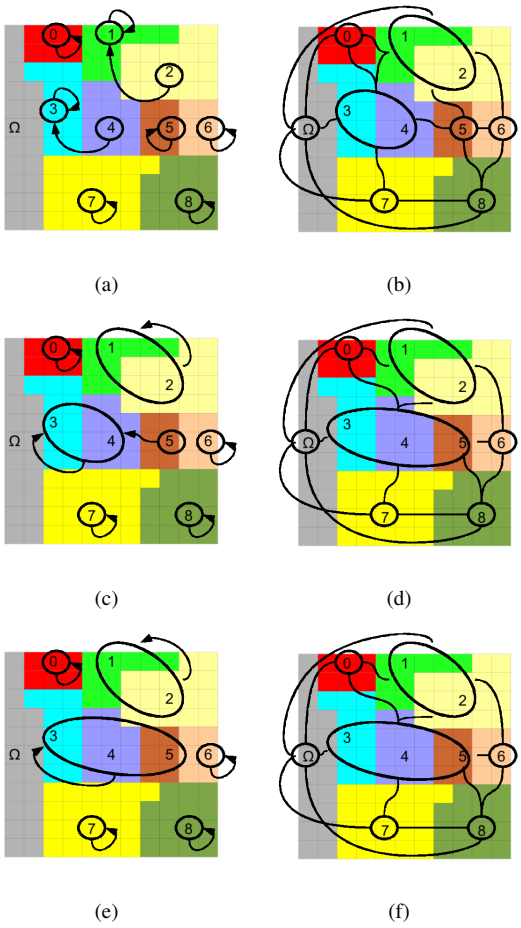
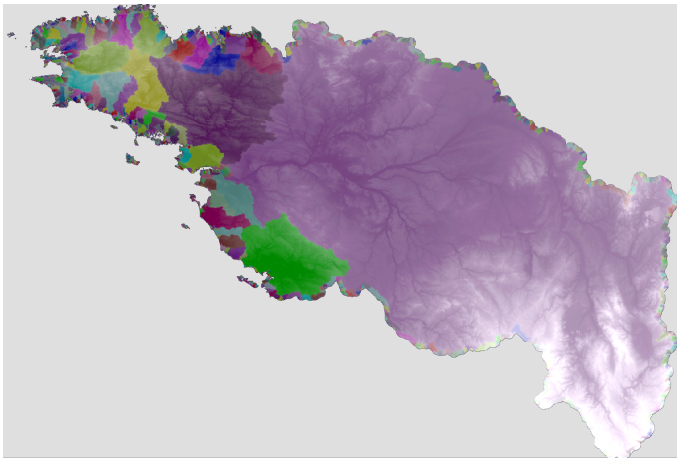
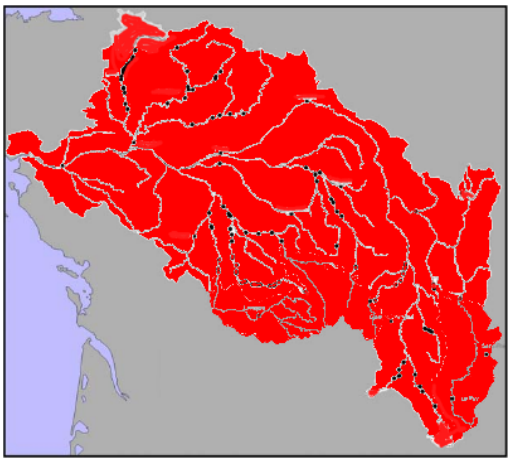


Fig. 7. Ω filtering



(a) Result of the method



(b) Real catchment basin

Fig. 8. Catchment basin of the Loire river (France)

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 5,701; 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.

Let $numProcs$ be number of processors used. The running time $T(numProcs)$ is the time elapsed between the moment that the first processor starts and the moment that the last processor finishes. The measured times with number of processors are tabulated in Table 1. It excludes data loading, and saving. The *relative speedup* of the parallel algorithm is measured by: $P(numProcs) = \frac{T_{min}}{T(numProcs)}$ 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 four processors. Then, in this case, the speedup linear curve start at coordinate (2,0) and is parallel to the classical speedup linear curve (Fig. 9).

Looking at Fig. 9, 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, such as MNT_a for 64 processors. With a larger DEM (1.9 Gpoints for MNT_c) speedup increases linearly (Fig. 9). This illustrates the good scalability of our approach. Note that MNT_c speedup is super linear at the beginning of the curve. A possible reason is the cache effect resulting from the different memory hierarchies.

TABLE I
PARALLEL COMPUTATION EXECUTION TIME (IN SECONDS)

	MNT_a	**MNT_b**	**MNT_c**
	3,980 x 5,701	10,086 x 14,086	36,002 x 54,002
	10 iterations	11 iterations	24 iterations
1	135.965	878.653	
2	70.591	508.436	
4	49.719	315.471	7,976.360
8	31.780	194.659	3,673.627
16	17.555	105.894	2,212.034
32	11.423	59.235	1,481.186
64	14.905	47.255	1,266.618

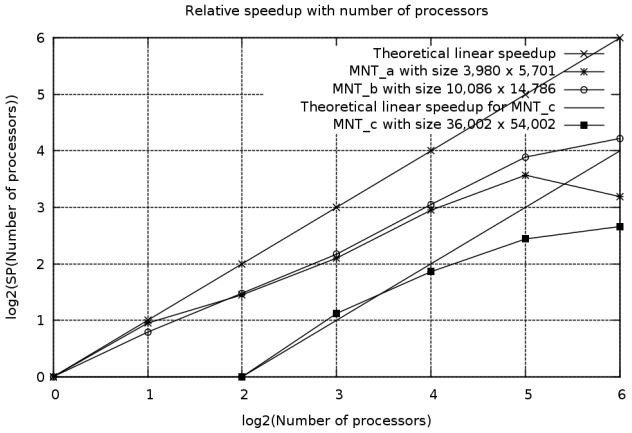


Fig. 9. Relative speedup with number of processors

V. Conclusion

We propose a formal definition of a parallel framework to construct a watershed hierarchy dedicated to DEM analysis. Of course, the method applies concepts derived from geo-hydrology, from image segmentation and from graph theory. The method tries to use the better of these areas *and* is fully parallel and does not use too complex data structures to alleviate memory need. We show that the hierarchy construction is generic. For example we propose parameters which allow to adapt the method to determine catchment basin of major rivers, taking into account sea and border of the DEM which may belong to catchment basin of river outside of the datasets.

This algorithm is scalable. We propose a SPMD parallel implementation onto a PC cluster. We use it on a large DEM and we obtain good speedup and time computations for huge datasets compatible with classical GIS processing times for small datasets, with desktop computers.

We plan to use theses results to imagine a parallel computation of hydrographic networks from DEM datasets. We think that this framework might be useful in all situations where a hierarchical analysis is relevant to extract information from a GIS.

Acknowledgment

The authors would like to tkanks the town of Orleans Conseil General du Loiret, the Region Centre who granted this work.

References

[1] M. F. Hutchinson, "A new procedure for gridding elevation and stream line data with automatic removal of spurious pits," *Journal of Hydrology*, vol. 106 (3-4), pp. 211–232, 1989.

[2] J. F. O'callaghan and D. M. Mark, "The extraction of drainage networksfrom digital elevation data," *Computer Vision, Graphics and Image Processing*, vol. 28, pp. 328–344, 1984.

[3] G. J. and L. W. Martz, "The assignment of drainage direction over flat surfaces in raster digital elevation models," *Journal of Hydrology*, vol. 193, pp. 204–213, 1997.

[4] T. A., "Automated recognition of valley lines and drainage networks from grid digital elevation models: a review and a new method," *Journal of Hydrology*, vol. 139, pp. 263–293, 1992.

[5] J. B. T. M. Roerdink and A. Meijster, "The watershed transform: Definitions, algorithms and parallelization strategies," vol. 41, no. 6, pp. 187–228, January 2001.

[6] K. P. V. N. John Nikolaus Callow and G. S. Boggs, "How does modifying a dem to reflect known hydrology affect subsequent terrain analysis?" *Journal of Hydrology*, vol. 332, no. 1-2, pp. 30–39, January 2007.

[7] L. Arge, J. S. Chase, P. Halpin, L. Toma, J. S. Vitter, D. Urban, and R. Wickremesinghe, "Flow computation on massive grid terrains," *GeoInformatica*, vol. 7, p. 2003, 2001.

[8] S. Jenson and J. Domingue, "Extracting topographic structure from digital elevation data for geographic information system analysis." *Photogrammetric Engineering and Remote Sensing*, vol. 54(11), pp. 1593–1600, 1988.

[9] S. D. Peckham, "River network extraction from large dems," Ph.D. dissertation, 1995.

[10] S. Beucher and F. Meyer, "The morphological approach to segmentation: The watershed transformation," in *Mathematical morphology in image processing*, ser. Optical Engineering, E. Dougherty, Ed. New York: Marcel Dekker, 1993, vol. 34, ch. 12, pp. 433–481.

[11] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, June 1991.

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

[13] H. Sun, J. Yang, and M. Ren, "A fast watershed algorithm based on chain code and its application in image segmentation," *Pattern Recogn. Lett.*, vol. 26, no. 9, pp. 1266–1274, 2005.

[14] F. Meyer, "Hierarchies of partitions and morphological segmentation," in *Scale-Space '01: Proceedings of the Third International Conference on Scale-Space and Morphology in Computer Vision*. London, UK: Springer-Verlag, 2001, pp. 161–182.

[15] S. Beucher and B. Marcotegui, "Fast implementation of waterfall based on graphs," ser. Computational Imaging and Vision, C. Ronse, L. Najman, and E. Decencire, Eds. Dordrecht: Springer-Verlag, 2005, vol. 30, pp. 177–186.

[16] A. N. Moga, "Parallel watershed algorithms for image segmentation," Ph.D. dissertation, Tampere University Technology, February 1997.

[17] A. N. Moga and M. Gabbouj, "A parallel marker based watershed transformation," in *Proceedings International Conference on Image Processing*, vol. 2. Lausanne, Switzerland: IEEE Computer Society Press, September 1996, pp. 137–140.

[18] A. Bieniek, H. Burkhardt, H. Marschner, and M. Nöelle, "A parallel watershed algorithm," in *Proceedings of 10th Scandinavian Conference on Image Analysis*, Lappeenranta, Finland, June 1997, pp. 237–244.

[19] A. N. Moga and M. Gabbouj, "Parallel image component labeling with watershed transformation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 19, no. 5, pp. 441–450, May 1997. [Online]. Available: citeseer.ist.psu.edu/moga97parallel.html

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

[21] A. Meijster and J. B. T. M. Roerdink, "A disjoint set algorithm for the watershed transform," in *Proceedings IX European Signal Processing Conference (EUSIPCO'98)*, S. Theodoridis, I. A. Stouraitis, and N. Kalouptsidis, Eds., Rhodes, Greece, 08- 11 September 1998, pp. 1665–1668.

[22] B. Chazelle, "A minimum spanning tree algorithm with inverse-ackermann type complexity," *J. ACM*, vol. 47, no. 6, pp. 1028–1047, 2000.

[23] F. K. H. A. Dehne and S. Götz, "Practical parallel algorithms for minimum spanning trees," in *SRDS*, 1998, pp. 366–371.

[24] S. Chung and A. Condon, "Parallel implementation of borvka's minimum spanning tree algorithm," in *IPPS '96: Proceedings of the 10th International Parallel Processing Symposium*. Washington, DC, USA: IEEE Computer Society, 1996, pp. 302–308.