An Hierarchical Labeling Technique for Interactive Computation of Watersheds

Kevin Bourgeois, Sébastien Limet, Sophie Robert, Victor Essayan

An Hierarchical Labeling Technique for Interactive Computation of Watersheds

Kevin Bourgeois
Univ.Orléans,
INSA Centre Val de Loire, LIFO EA4022,
Géo-Hyd (Antea Group) Orléans, France
Email: kevin.bourgeois@univ-orleans.fr

Sophie Robert
Univ.Orléans,
INSA Centre Val de Loire,
LIFO EA4022, Orléans, France
Email: sophie.robert@univ-orleans.fr

Sébastien Limet
Univ.Orléans,
INSA Centre Val de Loire,
LIFO EA4022, Orléans, France
Email: sebastien.limet@univ-orleans.fr

Victor Essayan
Géo-Hyd (Antea Group)
Orléans, France
Email: victor.essayan@anteagroup.com

***Abstract*—The watershed computation is a prevalent task in the geographical information systems. It is used, among other purposes, to forecast the pollutant concentration and its impact on the water quality. The algorithm to compute the watershed can be hard to parallelize and with the increasingly data growth, the need for parallel computation increases. In this paper we propose a new method to parallelize the watershed computation. Our algorithm is decomposed into two tasks, the parallel watershed segmentation into a hierarchy that allows in a second task to retrieve randomly large watersheds at run-time in interactive time.**

I. Introduction

The hydrology is a branch of the geosciences that is mostly interested in the water cycle, the water resources and the environmental watershed sustainability. In many countries, there is a duty to provide citizens with data on water which is the role of organizations like the BRGM (French Geological Survey) in France. Such a duty becomes a challenge with the progress in measuring devices that made amount of available data grow dramatically in the last decades. The work presented in this paper takes place in a project that aims at providing to all people a web site where it is possible to click on a map and to get instantly information on this point in particular the geographical area its drains (such area is called the watershed of the point). This computation may be very time consuming especially on big terrains and it is not easy to reach interactive time required by the targeted application. Indeed, the starting point of the watershed computation is a digital elevation model (DEM), i.e. a matrix that represents the terrain and where each cell contains the average elevation of the surface corresponding to the cell. From the DEM is computed the direction flow matrix that indicates for each cell the direction where a drop of water will flow when it falls on it following the steepest slope for example. Finally, when the user chooses a point on the map, its watershed is computed climbing back the flow directions until the ridgelines are reached. Of course a naive implementation of this computation would lead to run-times incompatible with interactive applications even when using parallel programs.

It is for this reason that we proposed a pre-processing chain of treatments that produces a set of sub-watersheds between some point of interests which correspond more or less to junctions of rivers. Figure I shows the points of interest and their associated sub-watershed. In this context a sub-watershed is the area drained by a point of interest that does not go beyond another point of interest. The idea is that when the user chooses a point on the map, the system computes at run-time the sub-watershed of this point up to points of interest. This area is a small one that can be quickly computed. Then the global watershed of the point is computed by adding the pre-computed sub-watersheds of the points of interest that flows to point chosen by the user. Such computation may be very fast when using a smart labeling of the sub-watersheds that encodes the dependencies between the points of interest as illustrated Figure I. Indeed it can be noticed that if the label of sub-watershed is the prefix of another sub-watershed, then the second one flows to the first one.

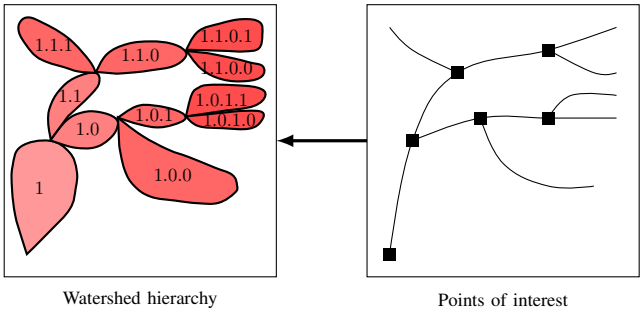


Fig. 1. The points of interest and the watershed hierarchy associated

The processing chain includes the computation of the direction flow as well as the accumulation flow that compute for each cell of the terrain how many cells flow to it. The

accumulation flow is used to determine the points of interest. These two computations are not detailed in this paper but have been efficiently parallelized using implicit parallel techniques in [1]. This paper focuses on the computation of the sub-watersheds coupled to their hierarchical labeling which is the most time consuming step of the processing chain and its parallelization is not trivial.

Many methods have been proposed to parallelize the watershed delineation but, as far as we know none of them both assign consistent labels related to the hierarchy of the sub-watersheds and delineate the watersheds themselves (i.e. assigns the cells of a sub-watershed with the right label). The task is complex since the labels cannot be assigned independently by each process which may cause many communications. The parallelization proposed here, spares communications by the use of temporary labels and dependency graphs. These graphs are exchanged once and are used by each processor to compute the final labels. Experiments on big DEMs show that this method is efficient and scalable for huge datasets.

The rest of the paper is organized as follows. Section II presents some related work. The labeling method and its parallelization are presented Section III. We present and discuss the results of the parallelization against a naive version of the algorithm in Section IV. Finally, Section V concludes the paper and draw some perspectives.

II. Related Work

As the watershed labeling is a very prevalent task in geosciences and in image processing, many researches tried to increase the performance in order to be suitable for large datasets.

Historically, the watershed segmentation appeared in the GIS domain where topographic maps were used to delineate the drainage basins in order to forecast how the nutrients, the sediments, the pollutants are transported outside the basins and how it affects water quality. The watershed segmentation in image processing does not serve the same purpose, but both use the same algorithms by trying to find a dividing lines to categorize cells or pixels with common properties.

There are several approaches in image processing to find the watershed. In the flooding methods, the image is considered as a topographic relief. A water source is placed in each regional minimum of the terrain, and the water level is progressively increased and the meeting of two water sources constitutes an edge of the watershed. A great improvement of this method was proposed by Wand and Liu [2] in GIS domain with a method called priority-flood and later improved by R. Barnes [3] and G. Zhou et al [4].

Another method consists in simulating a rainfall over the surface. The drops that flow over a point of the terrain will follow the steepest path, down to the outlet, every points sharing the same outlet are in the same watershed. J-P. Thiran [5] proposed a fast rainfall based method that has been improved by J. De Bock et al. [6] by dividing the watershed labeling into low-complexity relabeling steps. H. Sun et al. [7] improved this method thanks to a chain-code that indicates for each pixel the steepest path.

All the papers cited above are about sequential segmentation. However the parallelization of this task can be difficult because of the processing order of the cells implies either a lot of communications or a strategy to divide the terrain. There are several successful attempts to parallelize this task. J. Roerdink and A. Meijster [8] listed several strategies to compute watershed in sequential and in parallel. Swiercz and Iwanowski [9] proposed a parallel version for distributed architectures. They tried to avoid communications by pre-labeling selected pixels of bands that separate the non-overlapping tiles distributed among the threads. This allows each thread to compute its tile independently. The algorithm proposed by H.-T. Do et al. [10] is designed to work on distributed architectures. They proposed to split the terrain into overlapping tiles and to assign label on each tile independently. The processes synchronized their tiles in order to create a local dependency graph (LDG) that indicates when a pixel has a temporary label because the process has not enough information. The LDGs are sent in all to all communications to create a global dependency graph (GDG) to assign the definitive label for each pixel. Finally R. Barnes proposed a parallel version of the priority-flood algorithm [11] based on the sequential version proposed by G. Zhou et al.

Our algorithm is designed to extract a watershed hierarchy between points of interest. The hierarchy is encoded in the labels used to identified the different sub-watersheds. This feature increases the difficulty of the parallelization wrt the methods cited above since it requires a global coherent labeling. The solution we propose uses a rain falling technique to identify and label points of interest. It uses dependency graphs to make the labeling coherent among the processes and path compression techniques to efficiently label all the cells of the DEM.

III. Parallel Watershed Hierarchy

This section provides some definitions and notations used to formalize the watershed labeling problem and then describes the parallel algorithm we have implemented.

A. Definitions

The *domain* associated to a terrain (i.e. the mesh) is denoted $\mathcal{D}^{\mathcal{T}}$ and is a subset of $\mathbb{N}\times\mathbb{N}$. All the data associated are stored into matrices of domain $\mathcal{D}^{\mathcal{T}}$. Therefore, a cell c of the domain $\mathcal{D}^{\mathcal{T}}$ is defined as a pair (i,j) and X_c designates the value of the cell c in the matrix X. Let R be the input matrix (i.e. the DEM of the terrain) where each cell represents the height of the terrain. Let $\mathcal{N}_c$ be the set representing the neighboring of a given cell. The neighborhood can be of different size and shape, for example, a neighborhood can be either a cross (Von Neumann neighborhood) or a box (Moore neighborhood). To compute the watershed, we use a Moore neighborhood of size one (a Tchebychev distance of one). At last, let S be the function to define the successor of a cell c as follows:

$$\forall c \in \mathcal{D}^{\mathcal{T}},\ S(c) = c'\ s.t.\ R(c') = \min_{\{c''\in\mathcal{N}_c\}} R(c'') \quad (1)$$

that means that the function S indicates for each cell, a cell in the neighborhood with the steepest slope. It can be the cell itself if it is the lowest point. This method is known as the D8 method in the GIS domain. In this paper, we consider that R is lower complete, i.e. for each cell $c \in \mathcal{D}^{\mathcal{T}}$, there is one and only one cell $c' \in \mathcal{N}_c$ such that $R(c') = \min_{\{c'' \in \mathcal{N}_c\}} R(c'')$.

A first labeling L can be defined as follows:

$$\forall c \in \mathcal{D}^{\mathcal{T}},\ L_c = \begin{cases} L_{c'} & \text{if } \exists c' \in \mathcal{D}^{\mathcal{T}},\ S(c) = c' \\ c & \text{otherwise} \end{cases} \quad (2)$$

The labeling process is recursive. A cell must have the same label as its successor and when a cell without successor is reached (i.e. the end of a stream) a label is assigned to it and carried over to all the upstream cells. This labeling allows to gather the cells into watersheds that are tight (i.e. no water can flow from one to another). Moreover, these watersheds can be numerous with some large ones corresponding to the main streams and small basins for the cells are close to the sea for example.

The last preliminary matrix for the hierarchy computation is the flow accumulation matrix A defined as follows:

$$\forall c \in \mathcal{D}^{\mathcal{T}},\ A_c = \sum_{\{c' | S(c') = c\}} A_{c'} \quad (3)$$

For each cell c A_c depends of how many cells c drains and with an appropriate threshold, the flow accumulation allows to extract the streams.

B. Hierarchy

In order to retrieve the watershed of an outlet dynamically chosen by the user, we define a watershed hierarchy with a special labeling. The labeling method L presented Section III-A, gives the same label to all the cells with a common outlet which defines large watersheds that correspond to those of each main stream. As our goal is to compute dynamically, the watershed defined by a cell clicked at runtime, we decompose the large watersheds into sub-watersheds with a specific numbering to memorize which sub-watershed flows into another.

This method is based on *points of interest*. These points are defined thanks to the function S, the matrix A and a threshold t as follows:

Definition 1. *c is a* point of interest *if it exists at least two points c' and c'' in the neighborhood $\mathcal{N}_c$ s.t.*

- *either $S(c') = S(c'') = c$ and $A_{c'} \geq t$ and $A_{c''} \geq t$*
- *or $S(c) = c$ and $A_c \geq t$*

In other words, a cell is a point of interest if there is at least two incoming cells whose accumulation flows are higher than the threshold (i.e. it is the junction of two rivers) or if the cell is the outlet of the stream. The set of points of interest is denoted $PoI(S, A, t)$ and the set of outlets is denoted $O(S, A, t)$. We consider that the points in $PoI(S, A, t)$ are numbered by a function ord.

The points of interest allow to divide the terrain into sub-watersheds. The number and the size of the sub-basins depends on the threshold t.

The watershed hierarchy is based on the information of which watershed flows into another. To express this information a new labeling is used. The objective is to find in the label the watershed order. For this purpose the labeling is based on the concatenation of the interest point labels. This new labeling called *hierarchical labeling* and denoted H is defined as follows:

$$\forall c \in \mathcal{D}^{\mathcal{T}},\ H_c = \begin{cases} ord(c) & \text{if } c \in O(S, A, t) \\ H_{c'}.ord(c') & \text{if } S(c) = c' \\ & \text{and } c' \in PoI(S, A, t) \\ H_{c'} & \text{if } S(c) = c' \\ & \text{and } c' \notin PoI(S, A, t) \end{cases} \quad (4)$$

With this labeling all the cells of a sub-watershed that flows into another get a label prefixed by the label of the outlet sub-watershed. Therefore the labeling H describes the hierarchy of the sub-watersheds and if the watershed of a point c is sought it is sufficient to find the upstream point of interest p and to extract from the mesh all the cells which labeling is prefixed by H_p. The set of cells that have the same label l according to H is denoted $W(l)$ and is the following

$$W(l) = \{c \in \mathcal{D}^{\mathcal{T}} | H_c = l\}$$

Then the watershed of a cell c of label l contains the union of all the $W(l')$ such that $l' = l.l''$ (i.e. l is a prefix of l').

Algorithm 1 illustrates how the labeling H can be computed. This algorithm relies on the well known path compression technique [12] used to compute connected components of graphs. This technique consists in recursively searching for a labeled point or an outlet following the successor function S. Once this point has been found the cells on the path followed from the initial point are labeled according to Equation 4. The ord function is simply implemented by a global counter.

Notice, Algorithm 1 does not label some outlets because of the threshold t. The advantage is to avoid to memorize very small watersheds which can be numerous. Such tiny watersheds can computed on-the-fly with a very small cost.

Figure 2 shows an example of labeling of two streams where the red points are the points of interest. The example shows clearly that if one wants to retrieve the watershed ending at the point of interest with the label 1.0, one just needs to query all the watersheds with a label beginning with 1.0.

As the algorithm is recursive the order in which the cells are accessed is not predictable. The parallelization is not easy and a naive implementation that only consists in splitting the data into equivalent tiles would not be efficient.

C. Parallelization

To be able to tackle large amounts of data in the context of watershed computation, we focus on distributed architecture. The parallelization is based on the distribution of the matrices needed for the computation on different processors. Each processor p manages a sub domain $\mathcal{D}_p^{\mathcal{T}} \subseteq \mathcal{D}^{\mathcal{T}}$. To optimize

Algorithm 1 The labeling algorithm

1: **procedure** SEQUENTIAL_LABELING($\mathcal{D}^{\mathcal{T}}$, H, A, t)
2: $n \leftarrow 0$
3: **for all** $c \in \mathcal{D}^{\mathcal{T}}$ **do**
4: **if** H_c is undefined **then**
5: $H_c \leftarrow$ PATH_COMP($\mathcal{D}^{\mathcal{T}}$,$H$, A, c, t, n)
6: **end if**
7: **end for**
8: **end procedure**

9: **function** PATH_COMP($\mathcal{D}^{\mathcal{T}}$,$H$,$A$,$c$,$t$,$n$)
10: $c' \leftarrow S(c)$
11: **if** $c' \neq None$ **then**
12: **if** c' is a point of interest **then**
13: **if** $H_{c'}$is undefined **then**
14: $h \leftarrow$ PATH_COMP($\mathcal{D}^{\mathcal{T}}$,$H$,$A$,$c'$,$t$,$n$)
15: $H_c \leftarrow h.n$; n++
16: **else**
17: $H_c \leftarrow H_{c'}.n$; n++
18: **end if**
19: **else**
20: **if** $H_{c'}$ is undefined **then**
21: $H_c \leftarrow$ PATH_COMP($\mathcal{D}^{\mathcal{T}}$,$H$,$A$,$c'$,$t$,$n$)
22: **else**
23: $H_c \leftarrow H_{c'}$
24: **end if**
25: **end if**
26: **else**
27: **if** $A_c \geq t$ **then**
28: $H_c \leftarrow n$; n++
29: **else**
30: $H_c \leftarrow NO\_LABEL$
31: **end if**
32: **end if**
33: **return** H_c
34: **end function**

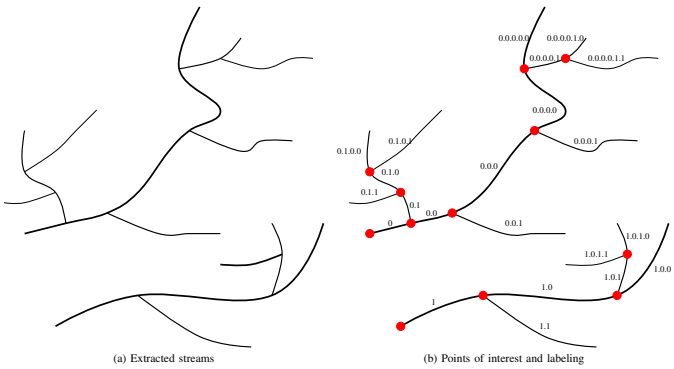


Fig. 2. Example of labeling of two streams

parallel computation, each mesh is divided with overlapping parts. As illustrated Figure 3 each processor owns additional cells called *ghost cells* and denoted by $\mathcal{G}$. Finally, we denote $\mathcal{O} \subset \mathcal{D}$ the cells which are shared with another processor.

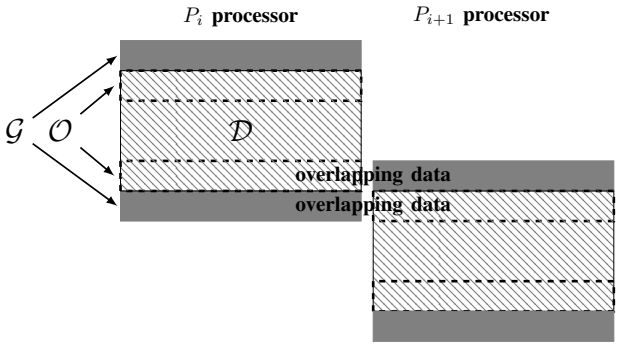


Fig. 3. Domain definitions for two processors with line bands

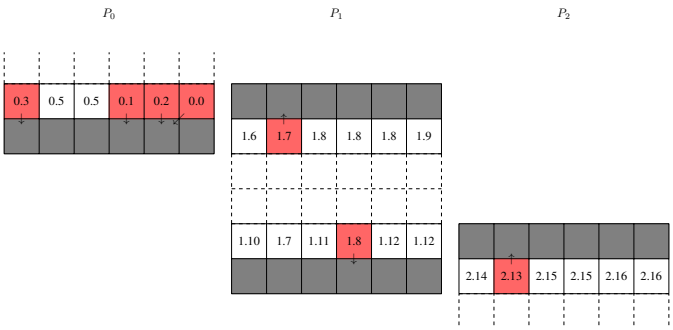


Fig. 4. First step of the parallel labeling (the p processor number as label prefix is omitted for reasons of clarity).

The naive method consists in labeling the cells in an iterative way. At each step the process computes, when it is possible, the labels of the cells of its local domain and exchanges with its neighbors the ghost data to propagate the label over other processes. The number of steps of this method is not known in advanced and it may involves a lot of communications. It can be really expensive when the number of processors grows.

Our method uses the similar scheme as the method implemented in Paraflow and described in [10]. However our algorithm requires to use the labeling H defined Equation 4 which complicates the task. The algorithm is split into three parts

The first part described Algorithm 2 consists in computing the cell labels of the local domain when it is possible, and in defining the *secondary cells* (line 21). A secondary cell is a cell whose successor is a ghost cell. For all secondary cells a temporary label is assigned. Then this label is assigned to the cell upstream thanks to the recursive algorithm. At last, to ensure the label uniqueness, all the local labels are prefixed by the processor number (line 25).

The second part begins with the ghost cell exchanges. Then a *local dependency graph* (LDG) is created to associate the temporary label of a secondary cell to the label of the ghost cell to which it depends. At this stage, the ghost cell labels may also be temporary labels. The LDG is described as a graph whose vertices are the labels and whose edges show how a temporary label need to be replaced. More precisely the edge $a \rightarrow b$ indicates that the label a is dependent of the

Algorithm 2 The local path compression in the parallel labeling algorithm

```
1: function PATH_COMP(D_p^T, H, A, c, t, n, SCells)
2:     c' ← S(c)
3:     if c' ≠ None then
4:         if c' ∈ D_p^T then
5:             if c' is a point of interest then
6:                 if H_c' is undefined then
7:                     h ← PATH_COMP(D_p^T, H, A, c', t, n)
8:                     H_c'.n; n++
9:                 else
10:                    H_c ← H_c'.n; n++
11:                end if
12:            else
13:                if H_c' is undefined then
14:                    H_c ←PATH_COMP(D_p^T, H, A, c', t, n)
15:                else
16:                    H_c = H_c'
17:                end if
18:            end if
19:        else
20:            H_c ← p.n; n++
21:            SCells ← SCells ∪ c
22:        end if
23:    else
24:        if A_c ≥ t then
25:            H_c ← p.n; n++
26:        else
27:            H_c ← NO_LABEL
28:        end if
29:    end if
30:    return H_c
31: end function
```

label b (or label a should be label b).

Figure 4 illustrates the end of the first step. The colored cells are secondary cells on each processor and their corresponding labels are temporary. Thus the LDGs built by each processor are the following:

$$LDG_{P_0} = \{0.3 \to 1.6, 0.1 \to 1.8, 0.2 \to 1.8, 0.0 \to 1.8\}$$
$$LDG_{P_1} = \{1.7 \to 0.5, 1.8 \to 2.15\}$$
$$LDG_{P_2} = \{2.13 \to 1.7\}$$

The LDG is not enough to finalize the labeling. For example, the secondary cells of label 0.0, 0.1 and 0.2 on P_0 depends of the 1.8 label of the P_1 processor which itself is temporary and depends on the 2.15 label of the P_2 processor.

This is why, the last step consists in computing a *Global Dependency Graph* (GDG) to end the labeling. The LDG are sent thanks to all to all communications of all the processors. The GDG construction as illustrated Algorithm 3, consists in reducing a path from a temporary label a to a label b such as there is no edge of the form $x \to b$ to the edge $a \to b$. Each process constructs its local part of the GDG from the union of all LDGs in order to finalize the secondary cell labeling. For the example illustrated Figure 4, the partial GDGs are the following

$$GDG_{P_0} = \{0.3 \to 1.6, 0.1 \to 2.15, 0.2 \to 2.15, 0 \to 15\}$$
$$GDG_{P_1} = \{1.7 \to 0.5, 1.8 \to 2.15\}$$
$$GDG_{P_2} = \{2.13 \to 0.5\}$$

Algorithm 3 The GDG construction and the final labeling

```
1: procedure FINAL_LABELING(H, SCells, LDGs, GDG)
2:     GDG = ∅
3:     for all Cell c ∈ SCells do
4:         h ← H_c
5:         x ← y s.t. (h → y) ∈ LDGs
6:         while ∃y s.t. (x → y) ∈ LDGs do
7:             x ← y
8:         end while
9:         GDG = GDG ∪ (h → x)
10:        while ∃y s.t. S(y) = c do
11:            H_y = x
12:            c = y
13:        end while
14:    end for
15: end procedure
```

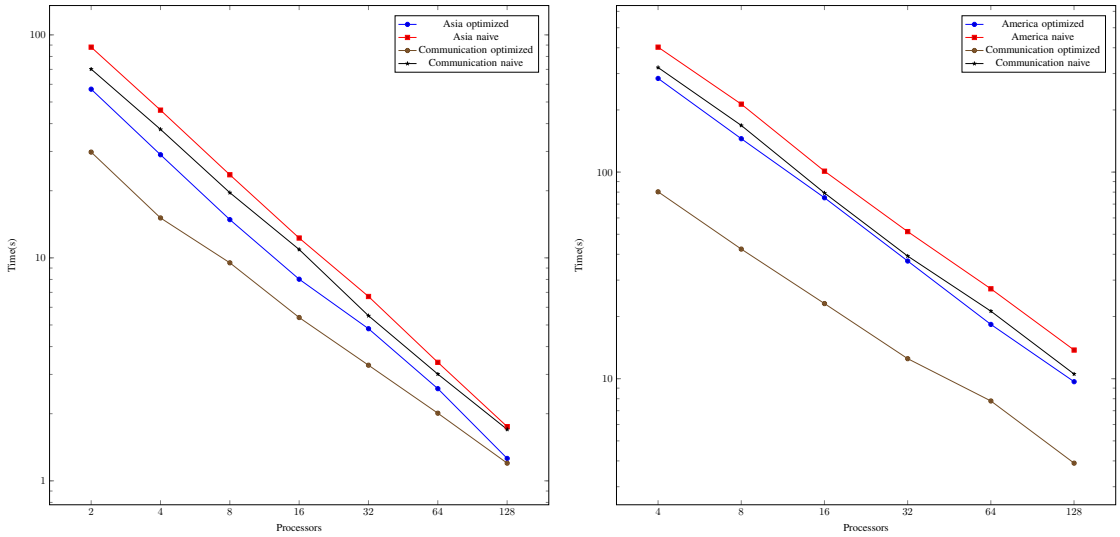
At last, to end the labeling, all the upstream cells of the secondary cells need to be updated as illustrated Algorithm 3 (line 10).

IV. Results

We used the compiler GCC 6.3.0 with optimization flag set to -O2 and OpenMPI 1.10.2. All tests have been done on the Centre de Calcul Scientifique en Région Centre (CCSC). The cluster runs on the Scientific Linux Release V6.6. It is composed of 48 nodes where each node hosts 20 CPU's Intel Xeon E5-2670 2.5Ghz, 64Gb of memory. The network is an InfiniBand 40G/s. We used 7 nodes for the tests with 128 processors. Every run has been done 8 times and averaged. The maximum deviation was less than 2%.

The experiments are based on two datasets, the Asia and the North America DEMs from the Shuttle Radar Topography Mission (SRTM3) with a spatial resolution of 3 arc-second (around 90 meters). It corresponds to two matrices respectively 60001×90001 (around 20GB) 432001×61201 (around 98GB) of 32-bit integers.

Since we did not find any algorithm equivalent to ours, it was not possible to compare our results. However we implemented a naive version of the algorithm. This version starts the labeling from the outlet of the watershed and assigns directly a label to a cell by going up the stream. This method is iterative and at each step a process tries to compute as much cells as possible. When it finishes, the ghost cells are exchanged to compute new cells at the next iteration and so on until all processors have computed all the cells.



(a) Asia dataset

(b) North America dataset

Fig. 5. Execution time of the labeling

Nb processors	2	4	8	16	32	64	128
Optimized version	57.07	29.00	14.84	8.02	4.81	2.59	1.26
Naive version	88.09	45.97	23.58	12.27	6.71	3.40	1.75
Gain (%)	35.21	36.92	37.07	34.64	28.32	23.82	28

(a) Asia dataset

Nb processors	4	8	16	32	64	128
Optimized version	284.12	145.12	75.14	37.12	18.3	9.67
Naive version	402.66	213.34	101.06	51.53	27.23	13.76
Gain (%)	29.44	31.98	25.65	27.96	39.76	29.72

(b) North America dataset

Fig. 6. Execution times and gain

The execution times for the Asia dataset are illustrated Figure 5a for the naive version and our parallelization. The speed-up is linear for the two versions but the optimized version is between 25% and 35% more efficient. The data are distributed per band of lines of the matrix. So according to the number of processors P_0 receives the first band of lines and P_1 the second and so on. Another data partition could be a classical round robin one. It consists in cutting the domain in small bands of lines that are distributed in a round robin way on the processors. For our method this distribution leads to more communications without a better load-balancing. Indeed, the number of secondary cells and the number of LDGs increase with the number of bands. So the labeling depends more on the GDG whose construction is an expensive part of our method. Notice that small bands also penalize the naive version since it increases the number of communications.

The results depicted Figure 5b show the execution times over the North America. Notice that the size of the data does not allow us to perform the test for one and two processors. These results confirm that the speed-up is linear for the optimized and the naive version. The Figure 6b shows that the optimized version is between 25% and 40% better than the naive version which clearly shows that the number of communications of the naive version is impacts its performance. Notice that the time of the communications is almost four time smaller for the optimized version compared to the naive version.

The threshold t of the Definition 1 affects the number of sub-watersheds. The previous results were obtained with t set to 500000. In the case of the Asia dataset, almost 4100 sub-watersheds were built. If t is set to 50000 the number of watersheds increases up to 37000. But the execution time are very similar without relevant differences. Indeed, the execution time depends on the GDG construction. But if a cell is located at the border Asia dataset and the execution times are also very similar for these two thresholds.

The execution time to retrieve a watershed from any point of the DEM depends on the number of the sub-watersheds. If the threshold is small, the generated sub-watersheds are numerous. To build the watershed requested, a lot of sub-watersheds may be merged but the cost to find the first interest

t	hierarchy size	result size	execution time
500000	4100	522	40ms
50000	37000	5521	60ms

Fig. 7. Example of watershed reconstruction from a clicked point (Asia dataset)

point is low. If the threshold is high, the time spent to collect the cells until we reach a point of interest can be longer, but the time to query the sub-watersheds upstream is very fast. The Figure 7 illustrates an example for the Asia dataset. With the threshold $t = 50000$, 5521 sub-watersheds are merged in 60ms whereas for the threshold $t = 50000$ 522 sub-watersheds are required with 40ms to merge them. Depending on the DEM it is therefore important to choose a threshold which ensures a good compromise between the size of the sub-watersheds and their number.

Figure 8 shows an example of the sub-watersheds obtained with our method (in multicolor) and the watershed retrieves from a point clicked at run-time (in red). For the sake of readability we chose a small part of France used in the INSIDE project.

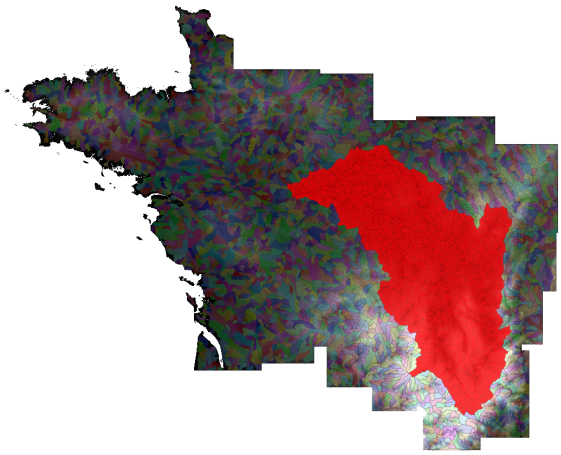


Fig. 8. Generated sub-watersheds hierarchy (in multicolor) and the resulting watershed of a point (in red)

V. Conclusion

In this paper, we presented a new method to compute a hierarchy of small sub-watersheds thanks to a labeling method that keeps trace of the order the water flows through them. We proposed an optimized parallelization that limits the communications but also the number of times the DEM is scanned. We showed its efficiency and good scalability on large datasets against a naive version.

This work is part of a more general project which aims at providing to hydrogeologists programming tools that ease them the use of parallel computers to process large datasets. In [1], we showed that on very large classes of algorithms used in this field, we can provide implicit parallel patterns that allow the user to program in a sequential way and obtain an efficient parallel program. The algorithm presented in this paper cannot be easily generalized since the optimizations we proposed are specific to the problem. The naive version could be implemented with implicit patterns which would lead to an overhead of about 25% to 35% compared to our method. Therefore, in a framework that targets to hide parallelism to non specialists, there are two solutions for such specific time consuming computations: either let the user implement a not optimized version with the available patterns or include such optimized algorithm as a function of the framework. Unless the overhead, the first solution allows the user to be independent of the presence or not of a specific function. On the other hand, since there are many specific problems to optimize in any science, the second could lead to a library containing numerous optimized specific functions which could be at the end difficult to use. Our framework should find a good balance between generality of the patterns and efficiency of the resulting programs.

Acknowledgment

We want to thank the BRGM and the INSIDE project for their support for this work as well as the Centre de Calcul Scientifique en région Centre (CCSC) which allowed us to perform the performance experiments.

References

[1] K. Bourgeois, S. Robert, V. Essayan, and S. Limet, "Efficient implicit parallel patterns for gis," in *International Conference on Computational Science*, 2017, to appear.

[2] L. Wang and H. Liu, "An efficient method for identifying and filling surface depressions in digital elevation models for hydrologic analysis and modelling," *International Journal of Geographical Information Science*, vol. 20, no. 2, pp. 193–213, 2006.

[3] R. Barnes, C. Lehman, and D. Mulla, "Priority-Flood: An Optimal Depression-Filling and Watershed-Labeling Algorithm for Digital Elevation Models," *Computers & Geosciences*, vol. 62, pp. 117–127, 2015.

[4] G. Zhou, Z. Sun, and S. Fu, "An efficient variant of the Priority-Flood algorithm for filling depressions in raster digital elevation models," *Computers & Geosciences*, vol. 90, pp. 87–96, 2016.

[5] J. Thiran, V. Warscotte, and B. Macq, "A queue-based region growing algorithm for accurate segmentation of multi-dimensional digital images," *Signal Processing*, vol. 60, no. 1, pp. 1–10, 1997.

[6] J. de Bock, P. de Smet, and W. Philips, "A Fast Sequential Rainfalling Watershed Segmentation Algorithm," in *Advanced Concepts for Intelligent Vision Systems, 7th International Conference*, 2005, pp. 476–482.

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

[8] J. Roerdink and A. Meijster, "The Watershed Transform: Definitions, Algorithms and Parallelization Strategies," *Fundamenta Informaticae*, vol. 41, no. 1-2, pp. 187–228, 2000.

[9] M. Swiercz and M. Iwanowski, "Fast, Parallel Watershed Algorithm Based on Path Tracing," in *Computer Vision and Graphics*, 2010, pp. 317–324.

[10] H. Do, S. Limet, and E. Melin, "Parallel computing of catchment basins of rivers in large digital elevation models," in *International Conference on High Performance*, 2010, pp. 39–47.

[11] R. Barnes, "Parallel priority-flood depression filling for trillion cell digital elevation models on desktops or clusters," *Computers & Geosciences*, vol. 96, pp. 56–68, 2016.

[12] R. E. Tarjan, "Data structure and network algorithms," *SIAM - Society for Industrial and Applied Mathematics*, 1983.