Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 00 (2011) 1–10

Procedia Computer Science

International Conference on Computational Science, ICCS 2011

# Parallel Computing Flow Accumulation in Large Digital Elevation Models

Hiep-Thuan Do, Sébastien Limet, Emmanuel Melin

*LIFO–Université d'Orléans (France)*

**Abstract**

This paper describes a new fast and scalable parallel algorithm to compute global flow accumulation for automatic drainage network extraction in large digital elevation models (DEM for short). Our method uses the D8 model to compute the flow directions for all pixels in the DEM (except NODATA and oceans). A parallel spanning tree algorithm is proposed to compute hierarchical catchment basins to model the flow of water from a sink (local minima) moving on DEM to its outlet (ocean, NODATA, or border of DEM). And finally, based on local flow accumulation and the hierarchical trees between sinks, we determinate entirely the global flow accumulation. From that, the drainage networks of DEM can be extracted. Our method does not need any preprocessing like stream burning on the initial DEM and tends to make the most of incomplete DEMs. Our algorithms are entirely parallel. Efficiency and scalability have been tested on different large DEMs.

*Keywords:* Parallelism, SPMD, Large Datasets, Digital Elevation Models, Minimum Spanning Tree

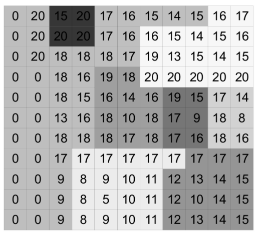
## 1. Introduction

Digital elevation models (DEM for short) are an important source of information in GIS applications. It has been widely used for modeling surface hydrology including the automatic delineation of catchment areas [1, 2], erosion modeling or automatic drainage network extraction [3]. All these computations are linked to the determination of flow direction [4] and then to the calculation of flow accumulation [5]. Moreover, flow accumulation is specially important to understand topographic controls on water, carbon, nutrient and sediment flows within and over full watersheds.

Geo-science research deals with increasingly large datasets coming from satellite or air plane LIDAR scans. Finer DEMs have the advantage to obtain more precise results when delimiting specific areas or running simulations. On one hand, data increase, on the other hand, sequential means of computation can not keep pace. Moreover, Geo-scientists need reasonable computation times compatible with an analysis loop. A solution is to parallelize these computations. This can be achieved if the algorithms are adapted to allow a good scalability to be able to take benefits of massive clusters available for scientists.

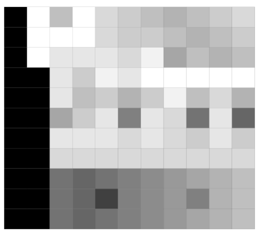
In this paper we propose a method able to compute in parallel the global flow accumulation for automatic extraction of drainage network from a large DEM. We use a totally parallel algorithm to compute global flow directions [6].

*Email addresses:* `hiep-thuan.do@univ-orleans.fr` (Hiep-Thuan Do), `sebastien.limet@univ-orleans.fr` (Sébastien Limet), `emmanuel.melin@univ-orleans.fr` (Emmanuel Melin)



| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 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) DEM as a raster



(b) Grayscale of the raster

Figure 1: Geological and image rasters

This algorithm combines techniques used in different fields such as hydro-geology, image processing and graph theory and is greatly scalable. From the resulting global flow directions, we propose another scalable parallel algorithm to compute flow accumulation. The results of the paper allow us to offer a more complete parallel tool to extract, modelize and interpret, hydrological information from large DEM.

This paper is organized as follows. In Section 2, we briefly present the related works for flow routing models in geo-hydrology and the solution we use. The section 3 describes steps of a parallel algorithm those intermediary data structure is crucial to our flow accumulation algorithm. This section is the opportunity to introduce important concepts necessary to understand the following. On this basis, Section 4 presents how to perform a parallel computation of flow accumulation on sinks which are homogeneous parts of the DEM. In Section 5, we present the parallel computation of the water flow through the global DEM. Experimental results are sketched in Section 6. Finally, we close with a conclusion.

## 2. Background and Previous work

Digital Elevation Model can be used to represent topography in GIS applications. Data of DEM is generally stored in one of the following data structures: (1) regular grid structure, (2) triangular irregular network (TIN for short) structure and (3) contour-based structure [7]. In this paper, we focus onto the grid-type DEMs i.e. 2D grids that store elevation data for each coordinate of the terrain as illustrated in Figure 1(a). Such a 2D grid is also called a *raster*, and a cell of this grid is called a *pixel* by analogy of the grayscale images (see Fig. 1(b)).

DEMs 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. The ***Step 1*** determines a raster coding flow direction for each pixel. This produces a forest of trees which root is a local minimum as in Figure 2(a). The ***Step 2*** determines a raster coding accumulated flows i.e. a raster where the value of a pixel $(x, y)$ represents the number of cells of the DEM that are drained to $(x, y)$. Finally the ***Step 3*** determines a raster segmentation between river pixels and other ones, classically with the use of a fixed threshold. In this paper, we focus on *Step 2* since a parallel solution for *Step 1* can be found in [6] and since a parallel solution for Step 3 is straightforward.

In the first step, flow direction is the key point to perform an hydrological analysis onto DEM. It simulates the way the outflow from a given cell will be distributed to one or more neighbouring downslope cells. In the flow routing models, the potential flow directions are assigned to each cell. The flow directions are then used for modeling the direction where water flows to the cells of the terrain. Several flow routing models have been proposed such as the D8 based on slope gradient [2], the Rho8 developed by [3], the FD8 in [8], the D$\infty$ [4]. In all these 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 DEMs like those provided by the United States Geological Survey(USGS for short), consist of a multitude of small sinks and plateaus. A plateau is a flat area with at least one spill-pixel. Authors generally chose to assign flow direction such as all pixels belonging to the plateau will flow to one spill-pixel. A sink is an area without spill-points . In this case the problem is that water accumulates in the local minimum. In this case, it is needed to determine the global direction of the flow and allow water to climb up the hill and to escape the sink. Since in both cases (plateaus and sinks), flows need to be redirected, and since uphill flow is counter-intuitive, many authors choose to modify initial datasets to obtain artifact-less raster called in image-computing *lower complete* [2, 9, 10, 11]. In hydrology, modifications of data, like *stream burning* [12, 13], to better reflect known hydrology can be useful but may compromise the ability to derive catchment parameters and conduct to a second terrain analysis [14].

Once catchment basins of rivers are found, it remains to compute flow accumulation. Sequential approaches use a recursive approach to compute accumulations onto contributing area of each cell. This method is simple but it requires large amount of memory and is not well suited for large datasets. In [15], Wallis and al. propose to abandon recursive functions to a queue-based approach that can work concurrently on several partition of the initial terrain. This solution requires strong synchronizations between the processors to maintain boundary pixels shared by nodes of the parallel architecture. Nevertheless it makes possible to use all the memory of a PC cluster and this allows scalable flow computation for large datasets. TerraFlow offers a way to compute flow accumulation of large terrains onto a single computer via I/O and CPU efficient algorithms [16, 17]. For such an approach, the only way to reduce the computation time of a fixed dataset is to buy a more powerful computer since it does not take benefits from scalability of parallel architectures.

In this paper, we show how to use a hierarchical approach to compute in parallel the global flow accumulation of the DEM. Our algorithm is totally parallel. Pixels are not directly synchronized via communications. Instead, we only use light weight data structure in memory and we only communicate high level representation of data between nodes of the parallel architecture.

## 3. Water path computation

In this section, we briefly present the parallel watershed extraction of DEM, more details can be found in [6]. This method consists in constructing a minimal spanning tree (MST) onto a graph that represents the global water flow direction. The minimal spanning tree represents the easiest way to go from one sink of the DEM to the sea.

First, we introduce some definitions necessary in the following. 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 height 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 $max(h_{(x_1,y_1)}h_{(x_2,y_2)})\}$. 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. The connectivity in the grid may be either 4-connectivity, or 8-connectivity. We call this first graph $G_0$. Notice that $G_0$ 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 vertices. Other graphs are represented by classical adjacency lists.

We choose a block-distribution for vertices of $G_0$. The figure 2(b) gives an example of distribution. The DEM is distributed onto 6 processors which domains are delimited by the bold lines. The neighbors pixels are represented, they do not belong to the first processor but they are distributed on it to minimize communications.

The method consists in selecting for each vertex $v$ the lightest edge $(v, v', w)$ if and only if $h_{v'} < h_v$. If several edges may be selected, we choose $(v, v', w)$ such that $h_{v'}$ is minimal (water goes down the steeper slope). If we find several equal $h_{v'}$ value then $v'$ is the first in the lexicographical order $\prec$. 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). Enforcing that $h_{v'} < h_v$ models the rain falling, i.e. when the basin overflows, the water should fall down. The first graph, represented by the arrows of Figure 2(a), is call *initial flow direction graph*.

Next, the vertices are labelled to detect connected components of the graph restricted to the selected edges. This is implemented via a parallel approach, each processor computes locally its connected components, each component is identified by its root called *sink* (i.e. the minimal vertex in the component w.r.t. height values defined above). This connected component is the catchment basin ($CB$) of the sink. For the sake of conciseness, when there is non ambiguity, we name *sink* the catchment basin. One connected component may be distributed on several nodes. This problem is resolved via the construction of local graphs, illustrated in Figure 2(c), and updates of data via global exchanges phases to obtain a coherent global dependency graph distributed onto all processors.

We obtain a minor of the initial graph. Indeed, each connected component can be considered as a vertex of another graph which is a minor of the initial graph. Finally each processor broadcasts the connectivity of the new vertices of the minor that have external edges. In this way, each processor is able to know the whole connectivity of vertices it is in charge of.

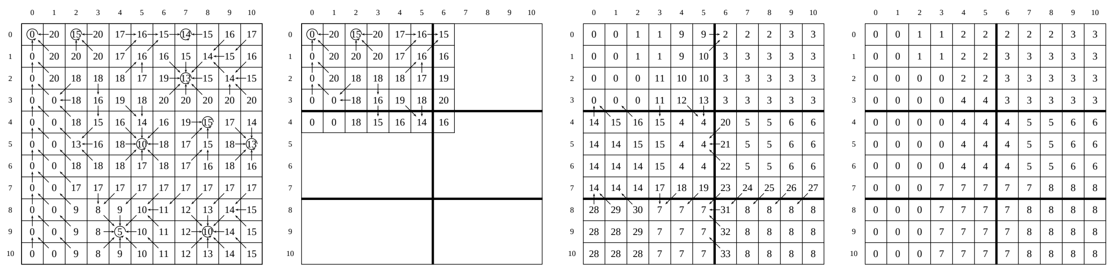
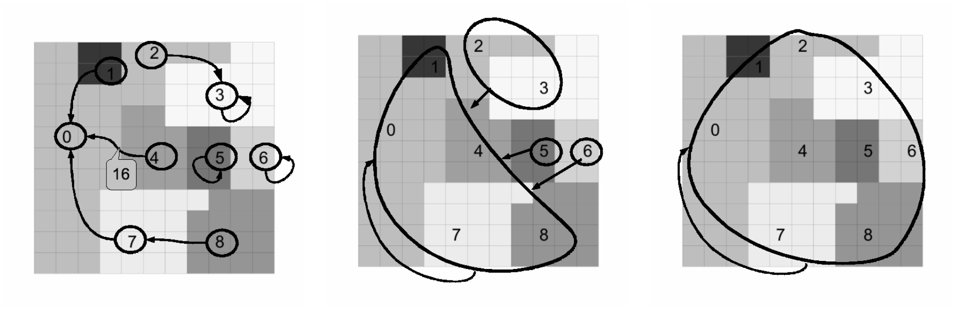

(a) Flow graph onto the DEM (b) Local Domain (c) Local labeling for borders (d) Result

Figure 2: Parallel process



(a) Graph $F_1$ (b) Graph $F_2$ (c) Graph $F_3$

Figure 3: Different graphs computed at each step

.

Figure 2 illustrates the computation of one level of the hierarchy. Figure 2(a) gives the initial graph, the number in the cells are the elevation data and the arrows represent the flow direction. Figure 2(d) gives the result i.e. a labelling of the pixel that determine the different connected component of the graph restricted to the selected edges.

The process is repeated until obtaining a single component, this produces a hierarchy of minors of the initial graph. The final MST is the union of the edges selected at each iteration. Browsing this MST from the lower point, it is easy to orient it and obtain a hierarchical tree. Figure 3 illustrates the computation of the whole hierarchy (self arrows are to be ignored) on the DEM given Figure 1(a). We remark that the sink labelled 0 (which figures the sea) grows rapidly until encompassing the whole DEM. The resulting hierarchy does not fit our needs since we cannot distinguish catchment basins we are looking for. This is why the hierarchy construction is driven to avoid merging of river catchment basins. More details can be found in [6, 18].

## 4. Parallel computing of flow accumulation inside each sink

In this part we present our parallel algorithm to compute flow accumulation *local to each sink*. The result of this step is a 2D grid $FA$ called the *flow accumulation matrix* that gives for each pixel $p$ the flow accumulation taking into account only the internal accumulation of the sink it belongs to. More formally, given a DEM $D$ and its initial flow direction graph $FG$, the *flow accumulation of a pixel* $p$ is the number of pixels $p'$ such that there is a path from $p'$ to $p$ in $FG$ if $p$ is neither nodata nor belongs to the sea.

The data distribution is the same as in Section 3. The block affected to node $i$ is called the domain $D_i$ of $i$. We call *extern neighboring* the set of all pixels $p'$ such that there exists a pixel $p$ in the local domain and the edge $(p, p')$ is in the initial flow direction graph. For a processor $i$, the set of pixels having extern neighbors is denoted $B_i$ and is called the *border* of $D_i$. To optimize computations et communications, we distribute to each node its extern neighboring. This is called the extension area.

**Algorithm 1**: Computing local flow accumulation at $p \in D_i$

```
procedure computeFAL($p \in D_i$)
begin
    Stack = $\emptyset$;
    $p \rightarrow$ Stack;
    canCompute = true;

    while ((stack $\neq \emptyset$) and (canCompute)) do
        $p' \leftarrow$ Stack;
        if ($\forall c \in Children(p')$, $FA_i[c] = \infty$) then
            $FA_i[p] = 0$;
            for ($\forall c \in Children(p')$) do
                $FA_i[p'] = FA_i[p'] + FA_i[c] + 1$;
            end
        end
        else
            $p' \rightarrow$ Stack;
            for (($\forall c \in Children(p')$ s.t. $FA_i[c]=\infty$) and
            (canCompute)) do
                if ($leaf(c)$) then $FA_i[c] = 0$;
                else
                    if ($c \in D_i$) then $c \rightarrow$ Stack;
                    else canCompute = false;
                end
            end
        end
    end

    if (not canCompute) then Stack = $\emptyset$;
end
```

**Algorithm 2**: Computing local flow accumulation in $D_i$

```
procedure computeFAL($D_i$)
begin
    // Computing FAL at border $B_i$ of $D_i$
    repeat
        for ( $\forall b \in B_i$ ) do
            if ($FA[b] = \infty$) then
                computeFAL($b$) in Algorithm 1;
            end
        end
        Global synchronization;
        Global Exchange of FA[$B_i$] for all processors ;
    until ($FA[p] \neq \infty$, $\forall p \in B_i$) ;

    // computing FAL for all other pixels $q$ in $D_i$
    for ($\forall q \in D_i$ s.t. $FA_i[q] = \infty$) do
        computeFAL($q$) in Algorithm 1;
    end
end
```

Each processor initializes the flow accumulation value of all the pixels of $D_i$ to $\infty$. Then, they compute the initial flow direction graph using flow routing model D8 illustrated in [2]. For each processor $i$, we obtain a local flow direction graph denoted $FG_i$, using the method described in section 3 (see also [18]). Notice that $\bigcup_{0 \leq i \leq n} FG_i$ is exactly the flow direction graph of the whole DEM since all processors have all the information to determine the direction of the flow from each pixel of its local domain. For a pixel $p$ of $D$, $Children(p)$ is the set of pixels $p'$ such that $p'$ goes to $p$ in $FG$ and $leaf(p)$ is true iff $Children(p) = \emptyset$.

As seen in previous section, the flow graph $FG_i$ is a forest of trees rooted by the sinks. Each tree being the catchment basin of a sink. Hence, the modeling of the watercourse in each catchment basin can be based on the $FG_i$. The parallel implementation for local flow accumulation computation is described in Algorithm 2. Each processor $i$ computes the local flow accumlation of $FA$ only for the pixels of $D_i$ ($FA_i$ denotes the corresponding part of $FA$). Note that due to data distribution onto processors, the catchment basin of a sink $s$, denoted $CB_s$, can be shared in several processors $i$. In this case, the local data at border $B_i$ of each processor $i$ must be exchanged with neighbouring processors. These exchange phases imply global barriers between all processors which may be very inefficient since it may need several exchanges depending on shape or size of the catchment basin and the number of processors. Therefore, in order to avoid useless waits, we do not mix computation for pixels $p \in B_i$ and others one. We first compute local flow accumulation for all pixels $p \in B_i$ of subdomains $D_i$ to concentrate the communication during this phase. After that, each processor can perform remaining flow accumulation computations using Algorithm 1 without any exchange nor synchronization.

For Algorithm 1, the flow accumulation of $p \in D_i$ is determinated if and only if all its children $c \in FG_i$ have been determinated. The flow accumulation of a leaf in flow graph $FG_i$ is equal to zero. The computation for all children $c$ is naturally recursively realized until flow accumulation of its children is determined. We used a stack of pixels, denoted $Stack$, to replace recursion with iteration in the implementation. Each processor uses its own $Stack$ for locally computing flow accumulation for a given $p$. In order to compute the local flow accumulation at $p$ in Algorithm 1, we use a $Stack$. The pixel $p$ is inserted into $Stack$. While flow accumulation of $p$ is still indeterninated and can be computed (not dependent of extern values), a pixel $p'$ is taken from $Stack$, if flow accumulation of all children of $p'$

is determined, the flow accumulation at $p'$ can be computed as the sum of children flow accumulations. Otherwise, $p'$ is inserted again into $Stack$. All children $c$ (flow accumulation equal $value_\infty$) of $p'$ in $FG_i$ belong in one of these three cases.

*Case 1:* The pixel $c$ is a leaf in the $FG_i$, the flow accumulation of $c$ is deternimated by zero. Our implementation doesn't need to insert all leaves into $Stack$.

*Case 2:* The pixel $c$ belongs to $D_i$, $c$ is then inserted into the $Stack$;

*Case 3:* The pixel $c$ is not in $D_i$, flow accumulation computing for $p$ cannot be continued. In this case, the $Stack$ is set as empty to optimize the use of memory. This is the case when the pixel belongs to a catchment basin that is shared by several processors. The computation for $c$ will be considered again after the exchanging of flow accumulation between neighbouring.

## 5. Global flow accumulation

In this section we describe how to obtain a global result covering all the DEM from the local computation of Section 4. The local flow accumulation was computed in each sink. Note that this result is local to a given sink but is not necessarily local to a processor. The main difficulty consists in determining the global flow of the entire DEM and therefore to drive flow uphill to reach progressively the sea. A very similar problem exists when we want to determine global catchment basins from DEM which are not lower-complete. In Section 3 we propose to construct a minimal spaning tree (called hierarchical tree or HT) linking sinks together and representing the most probable path of water (see also [6]). This is done taking into account the lower part of ridges between neighbor sinks. The hierarchical tree $HT$ obtained can be used for computing global flow accumulation. Our idea consists in computing flow accumulation between sinks following the hierarchical tree, then, to use this information to update accumulations flow of pixels in the entire DEM.

### *5.1. Flow in the hierarchical tree*

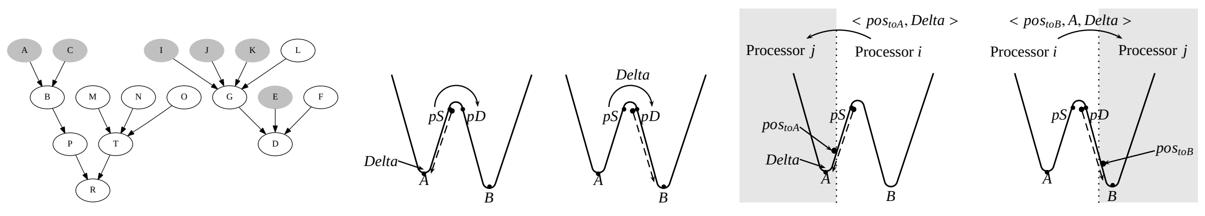
We turn now to describe the flow accumulation in the hierarchical tree $HT$. In Figure 4(a), we can see that the sink $A$ is a leaf of $HT$, the water would flow from $A$ into its parent $B$ in $HT$. The flow accumulation of the sink $A$ is added to the flow accumulation of $B$. The flow accumulation of $B$ is the sum of its local flow accumulation (the number of pixels belonging to the sink $B$) and the sum of the flow accumulation of all sinks children of $B$. The propagation is then done from sink $B$ to its parent $P$ in $HT$ until we reached it root $R$. The difficulty is to translate the transfert of the water accumulated in one sink to its parent in the flow accumulation matrix.

The hierarchical tree $HT$ imposes an order for the computation of flow accumulation of sinks. For example, consider the $HT$ of Figure 4(a). Assume that the two sinks $A$ and $C$ have finished to compute their flow accumulation and propagated it to sink $B$. Then, the sink $B$ can propagate its flows into its parent sink $P$. In the second connected component, sinks $I$, $J$ are marked as finished, but sink $L$ is still pending. This means that $G$ has not enough information to continue computation and therefore it is the same for its parent $D$.

In our implementation this work is performed at the same time as the step described thereafter and that consists in updating the flow accumulation matrix to translate water transfers in this matrix.

### *5.2. Update of flow accumulation of pixels*

We describe how we use flow accumulation at the sink level to update accumulations flow at pixels level. Let $A$ and $B$ be pixel sinks (i.e. local minima) and respectively $CB_A$ and $CB_B$ their catchment basins. The $HT$ tells that the water would flow from $A$ into its parent $B$. It remains to determine the most probable way the water would take to flow from A to B. In the $HT$ we have stored the lowest points of the watershed between $A$ and $B$ (they are denoted $pS$ and $pD$ respectively (see Figure 4). Note that these two points are on either sides of the ridge between $CB_A$ and $CB_B$. In our algorithm, we used both flooding under the $HT$ and flooding on the $DEM$ based on flow directions determined in the flow graph $FG_i$. In our modeling, to escape from $CB_A$ the water follows the path from $pS$ to $A$ in $FG$ (denoted $\pi_{[pS,A]}$) in the reverse direction, then it goes from $pS$ to $pD$ and finally follows the path from $pD$ to $B$ in $FG$ (denoted $\pi_{[pD,B]}$). The idea, is that the points of $\pi_{[pS,A]}$ carry all the accumulation of $A$ and the pixels of $\pi_{[pD,B]}$ accumulate



(a) The forest of hierarchical tree (b) $ToA$ on $\pi_{[pS,A]}$ (c) $ToB\_LOC$ on $\pi_{[pD,B]}$ (d) $ToA$ (e) $ToB\_LOC$

Figure 4: Propagation FA from sink $A$ to sink $B$

their own draining pixels plus the accumulation of $A$. So we have two types for propagations, called propagation $ToA$ (Fig. 4(b)) and propagation $ToB\_LOC$ (Fig. 4(c)).

In the propagation $ToA$ as described in Algorithm 3, the quantity $Delta$ is used to replace for flow accumulation of all pixels belonging to the descendant path between $pS$ and position of sink $A$, denoted $\pi_{[pS,A]}$. This is illustrated in Figure 4(b), where $Delta$, that is the flow accumulation of sink $A$, was determined during the steps of computing local flow accumulation and flow accumulation into the hierarchical tree. In our parallel implementation, the domain $D$ of DEM is partitioned into subdomains $D_i$ that are mapped by processors $i$. Therefore, the descendant path $\pi_{[pS,A]}$ may not entirely belong to the same subdomain $D_i$. In this case, the propagation $ToA$ on the descendant path $\pi_{[pS,A]}$ is broken when it reaches a neighbor pixel $pos_{toA} \notin D_i$ (see Figure 4(d)). To solve this problem, for each processor $i$, we use a data structure $ListSendA_i$ to communicate the propagation $ToA$ from neighbor $pos_{toA}$ to the subdomain $D_j$ in which the $pos_{toA}$ belongs to. At processor $i$, a pair $pair_A < pos_{toA}, Delta >$ is immediately inserted to $ListSendA_i$. The $ListSendA_i$ is then exchanged with others processors (see Figure 4(d)). To optimize the data exchange between neighboring processors, each processor $i$ only send to processor $j$ all $pair_A < pos_{toA}, Delta > \in ListSendA_i$ such that the pixel $pos_{toA}$ belongs to the subdomain $D_j$. And after that, the propagation $ToA$ on the descendant path $\pi_{[pS,A]}$ can be continued from $pos_{toA}$ with the quantity $Delta$ in $D_j$ in next iteration. The propagation $ToA$ can be repeated in several cycles local computation/global exchange, until the $ListSendA_i$ for all processors $i$ are entirely empty.

**Algorithm 3**: Propapation $ToA$ from $pos_{toA}$ with $Delta$ on the path $\pi_{[pS,A)}$ at Processor $i$

```
procedure ToA($PairA < pos_{toA}, Delta >$)
begin
    if ($pos_{toA} \notin D_i$) then
        $pair_A = < pos_{toA}, Delta >$;
        $pair_A \Rightarrow ListSendA_i$;
    end
    else
        $mFA_i[pos_{toA}]$ = Delta;
        $parent_{toA} = Parent_i[pos_{toA}]$;
        while ($pos_{toA} \neq parent_{toA}$) and ($parent_{toA} \in D_i$) do
            $FA_i[parent_{toA}]$ = Delta;
            $pos_{toA} = parent_{toA}$;
            $parent_{toA} = Parent_i[pos_{toA}]$;
        end
        if ($parent_{toA} \notin D_i$) then
            $pair_A = < parent_{toA}, Delta >$;
            $pair_A \Rightarrow ListSendA_i$;
        end
    end
end
```

**Algorithm 4**: Propapation $ToB\_LOC$ for $pos_{toB} \in \pi_{[pD,B)}$

```
procedure ToB_LOC($pair_B < pos_{toB}, A, Delta >$)
begin
    $FA_i[pos_{toB}] = FA_i[pos_{toB}]$ + Delta;
    $parent_{toB} = Parent[pos_{toB}]$;
    while (($parent_{toB} \neq pos_{toB}$) and ($parent_{toB} \in D_i$)) do
        $FA_i[parent_{toB}] = FA_i[parent_{toB}]$ + Delta;
        $pos_{toB} = parent_{toB}$;
        $parent_{toB} = Parent_i[pos_{toB}]$;
    end
    if (($parent_{toB} \notin D_i$)) then
        $pair_B = < parent_{toB}, A, Delta >$;
        $pair_B \Rightarrow ListSendB_i$;
        return true;
    end
    else
        PassedSinks_LOC[A] = finished;
        return false;
    end
end
```

We turn now to the propagation $ToB\_LOC$ from sink $A$ to its parent sink $B$. This propagation go through crossing point $pD \in CB_B$ (see Figure 4(c)), the quantity $Delta$ is used to increase for all pixels $pos_{toB}$ belonging to the descendant path $\pi_{[pD,B]}$. It is described in Algorithm 4. Firstly, we begin the $ToB\_LOC$ at the crossing point $pD$ of $B$. The propagation $ToB\_LOC$ is continuously realized for next pixel of $pD$ in $\pi_{[pD,B]}$. The final step depends on the state we obtain.

One one hand, the propagation $ToB\_LOC$, $<pstoB, A, Delta>$ has reached $B$. It means that the propagation between sink $A$ and sink $B$ on the path $\pi_{[pS,B]}$ is finished. The sink $A$ is then marked as finished for propagation to its parent $B$. This condition is used to decide that the propagation from sink $B$ can be continued for propagating to its root in $HT$.

One the other hand, the propagation reaches a neighbor pixel $pos_{toB} \in \pi_{[pD,B]}$, which does not belong to the domain of processor $i$ ($pos_{toB} \notin D_i$). This means that the propagation from sink $A$ into sink $B$ is suspended, it needs a communication. Like above, we used a data structure $ListSendB_i$ for each processor $i$ for solving the incompletion of the descendant path $\pi_{[pD,B]}$ in $D_i$ for propagation of flow accumulation (see Figure 4(e)). A $pair_B < pos_{toB}, A, Delta >$ is inserted into $ListSendB_i$. We send $ListSendB_i$ to processor $j$. And after that, the propagation is continued onto processors $j$.

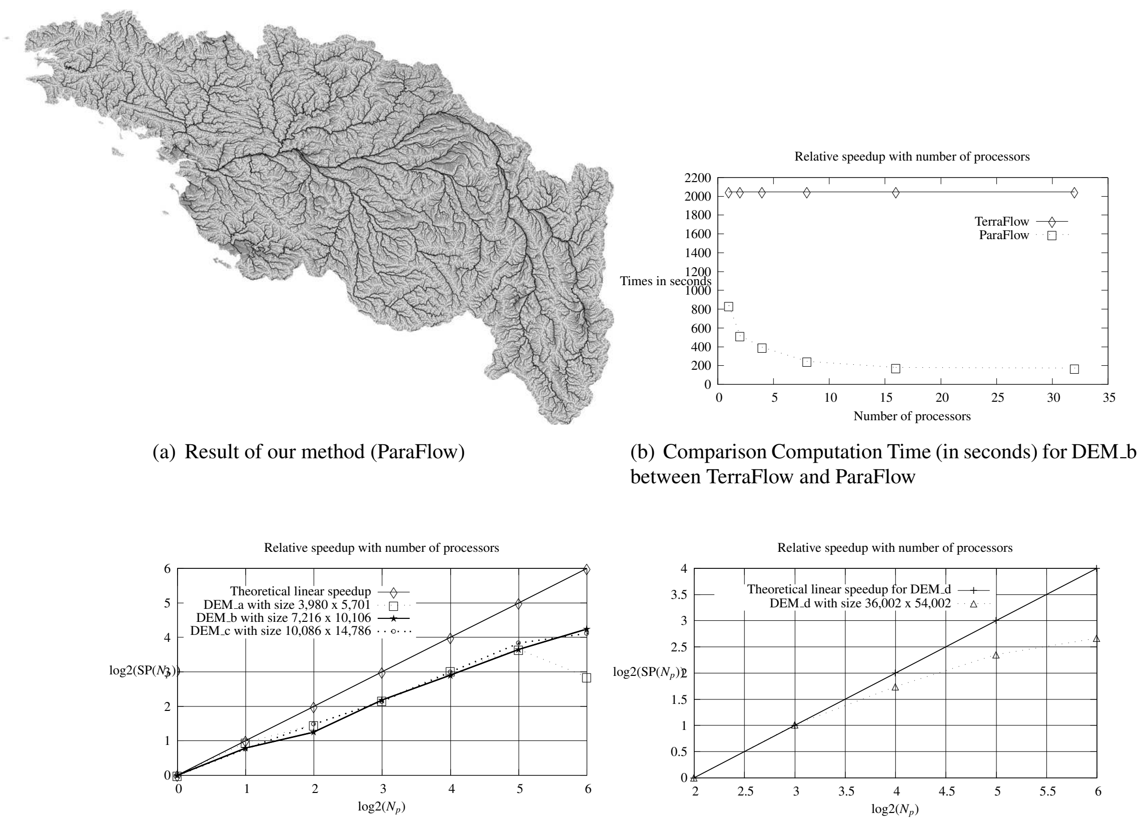
**Algorithm 5**: Propagation $ToB\_GLO$ with Delta from $pos \in CB_B$ to $sink_{root}$ $R$ of $B$ in $HT$

```
procedure ToB_GLO($pair_B < pos, A, Delta >$)
begin
    if ( $ToB\_LOC(pair_B)$) then
        $P$ = getParent($B$, $pS_{new}$, $pD_{new}$);
        propagable_LOC = true;
        ToRoot = PropagableToRoot($B$);
        while (($P \neq root$) and (propagable_LOC) and (ToRoot)) do
            newDelta = $FA_i[B]$;
            ToA($pS_{new}$, $newDelta$) in Algorithm 3;
            $pair_B = < pD_{new}, B, newDelta >$;
            if ($pD_{new} \notin D_i$) then
                $pair_B \Rightarrow ListSendB_i$;
                propagable_LOC = false;
            end
            else
                if ( $ToB\_LOC(pair_B)$) then
                    propagable_LOC = false;
                end
                else
                    $B = P$;
                    ToRoot = PropagableToRoot($B$);
                    $P$ = getParent($B$, $pS_{new}$,$pD_{new}$);
                    propagable_LOC = true;
                end
            end
        end
    end
end
```

**Algorithm 6**: Parallel computing flow accumulation in $D_i$

```
procedure computeFAG($D_i$)
begin
    for ($\forall$ sink $A \in HT_i$) | ($A$ is leaf) do
        B = getParent($A$, $pS$, $pD$);
        Delta = $FA_i[A]$;
        ToA($< pS, Delta >$) in Algorithm 3 ;
        $pair_B = < pD, A, Delta >$;
        if ($pD \notin D_i$) then
            $pair_B \Rightarrow ListSendB_i$;
        end
        else ToB_GLO($pair_B$) in Algorithm 5;
    end
    $allFinished$ = false;
    while (not $allFinished$) do
        Global synchronization;
        $ListRecvA_i = \{pair_A < pS', Delta > \in ListeSendA_j \mid pS' \in D_i\}$;
        $ListRecvB_i = \{pair_B < pD', X, Delta > \in ListeSendB_j \mid pD' \in D_i\}$;
        $ListSendA_i = \emptyset$ ;
        $ListSendB_i = \emptyset$ ;
        for ($\forall$ $pair_A \in ListRecvA_i$) do
            ToA($pair_A$) in Algorithm 3;
        end
        $ListRecvA_i = \emptyset$;
        for ($\forall$ $pair_B \in ListRecvB_i$) do
            ToB_GLO($pair_B$) in Algorithm 5;
        end
        $ListRecvB_i = \emptyset$;
        $allFinished_i$ = ($ListSendA_i$=$\emptyset$) and ($ListSendB_i$=$\emptyset$);
        Global Synchronization;
        $allFinished = \sum_1^{numProcs} allFinished_k$;
    end
end
```

The global propagation for flow accumulation $ToB\_GLO$ from a pixel $pos \in CB_B$ into sink $root$ of sink $B$ is described in Algorithm 5. The function $PropagableToRoot(B)$ is used to verify that the sink $B$ can be propagated to its parent in the hierarchical tree $HT$, a sink $B$ is propagable to its parent in $HT$ if only if all its sinks children of $B$ are marked as finished for propagation. The parallel algorithm is finished when both $ListSendA_i$ and $ListSendB_i$ of all processors $i$ are empty. And, the result $FA_i$ partitioned in each processor $i$ is the global flow accumulation of catchment of rivers.



(a) Result of our method (ParaFlow)

(b) Comparison Computation Time (in seconds) for DEM_b between TerraFlow and ParaFlow

(c) Small DEMs

(d) Large DEMs

Figure 5: Relative speedup with number of processors ($N_p$)

## 6. Experimental results

The parallel algorithm described in this paper has been implemented in C++ using MPI library and tested on eight nodes linked with a Gigabit Ethernet network. Nodes are bi-pro with AMD Opteron Quad-Core 2376 2.3GHz with 16GB SDRAM DDR2-PC5300 EEC 667 MHz, and operating system is linux. Figure 5(a) illustrates the result of our method for the DEM_c. That is the drainage networks for Loire Bretagne region in France.

First of all, we compared the quality of the results of our algorithm with those obtained with TerraFlow and the main leader of commercial GIS software ArcGIS. The three methods gives very similar river networks (no more than 3% of pixels differ between them).

Let $N_p$ be number of processors used and $T(N_p)$ the running time. The results, given in Fig. 5(c) and in Fig. 5(d), exclude data loading, and saving. The running time displayed for each DEM is the avarage of five runs of our program on the DEM (The difference between several runs on the same DEM is less than 0.1%). Note that, due to memory size limitation, it is not possible to run our program for DEM_d (size 36,002×54,002) onto one sole node. In this case only, we choose to give a speedup relative to the execution time of our algorithm onto four processors. Then the speedup linear curve start at coordinate (2,0) and is parallel to the classical one (Fig. 5(d)). Note that this is another important benefit of our parallel algorithm, to allow computation onto large DEM whose size does not fit into the memory of one sole PC. Looking at Figure 5(c), for DEM_a and DEM_b, we remark that the relative speedup is close to linear speedup at the beginning of the curves. With larger DEM (for example DEM_d) speedup increases linearly (Figure 5(d)). This illustrates the good scalability of our approach.

We compared in Figure 5(b), the running time of TerraFlow/GRASS(Geographic Resources Analysis Support System) to our method to analyze hydrology of DEM_b (the results, here, include times for loading and save data). Note that TerraFlow is a sequential code so its time duration is represented by on line in Figure 5(b). Even with one processor our method is faster than TerraFlow on this DEM and with 8 processors our method is about ten times faster.

## 7. Conclusion

In this paper, we presented an efficient and scalable fully parallel algorithm to compute the global flow accumulation. This method allows rapid automatic drainage network extraction in very large DEM. The method does not use too complex data structures to alleviate memory need, moreover data are distributed onto the cluster. The method takes into account sea and border of the DEM which may belong to catchment basin of river outside of the datasets. The results we obtain, from hydrology point of view is very close to those computed by classical software.

We proposed a SPMD parallel implementation onto a PC cluster. We used it on a large DEM, and obtained good speedups and computation times for huge datasets. The running time for hudge DEMs on a pretty small cluster is of the same order than the classical GIS processing times for small datasets, with desktop computers. This shows the interests of such methods with regard to out-of-core algorithms.

The methodology we use to compute the global flow accumulation may apply on many classical processings on DEMs. Therefore we are working on giving a more general framework to ease implementation those processings in parallel. We are also working on improving the scalability of our framework when the user wants to store intermediate results. Each of these results is as big as the initial DEM and the data are distributed over the nodes of the cluster which needs to efficiently parallelize the I/O.

## Acknowledgments

The authors would like to thank the town of Orléans, the Conseil General du Loiret, the region Centre who granted this work. We would like also to thank the Compagny Géo-Hyd that provided the digital elevation models.

## References

[1] L. W. Martz, E. D. Jong, Catch: a fortran program for measuring catchment area from digital elevation models, Comput. Geosci. 14 (5) (1988) 627–640. doi:http://dx.doi.org/10.1016/0098-3004(88)90018-0.

[2] J. F. O'callaghan, D. M. Mark, The extraction of drainage networks from digital elevation data, Computer Vision, Graphics and Image Processing 28 (1984) 328–344. doi:http://dx.doi.org/10.1016/S0734-189X(84)80011-0.

[3] J. Fairfield, P. Leymarie, Drainage networks from grid digital elevation models, Water Resources Research 27 (5) (1991) 709–717.

[4] D. G. Tarboton, Terrain analysis using digital elevation models (TauDEM), Utah State University, Logan, UT, USA (2002).

[5] T. K. Peuker, D. H. Douglas, Detection of surface-specific points by local parallel processing of discrete terrain elevation data, Computer Graphics and Image Processing 4 (4) (1975) 375–387.

[6] H.-T. Do, S. Limet, E. Melin, Parallel computing of cachment basins of rivers in large digital elevation models, The International Conference on High Performance Computing and Simulation (HPCS-2010 Caen).

[7] M. V. Kreveld, Digital elevation models: overview and selected tin algorithms, in: M. van Kreveld, J. Nievergelt, T. Roos, P. Widmayer (Eds.), Algorithmic Foundations of GIS, Vol. 1340 of LNCS, Springer-Verlag, 1997.

[8] P. Quinn, K. Beven, P. Chevallier, O. Planchon, The prediction of hillslope flow paths for distributed hydrological modelling using digital terrain models, Hydrological Processes 5 (1991) 9–79.

[9] J. Garbrecht, L. W. Martz, The assignment of drainage direction over flat surfaces in raster digital elevation models, Journal of Hydrology 193 (1997) 204–213. doi:http://dx.doi.org/10.1016/S0022-1694(96)03138-1.

[10] A. Tribe, Automated recognition of valley lines and drainage networks from grid digital elevation models: A review and a new method, Journal of Hydrology 139 (1992) 263–293.

[11] J. B. T. M. Roerdink, A. Meijster, The watershed transform: Definitions, algorithms and parallelization strategies 41 (1-2) (2001) 187–228. URL `http://www.cs.rug.nl/ roe/publications/parwshed.pdf`

[12] M. F. Hutchinson, A new procedure for gridding elevation and stream line data with automatic removal of spurious pits, Journal of Hydrology 106 (3-4) (1989) 211–232. doi:http://dx.doi.org/10.1016/0022-1694(89)90073-5.

[13] O. Planchon, F. Darboux, A fast, simple and versatile algorithm to fill the depressions of digital elevation models, CATENA 46 (2–3) (2002) 159–176. doi:10.1016/S0341-8162(01)00164-3.

[14] J. N. Callow, K. P. V. Niel, G. S. Boggs, How does modifying a dem to reflect known hydrology affect subsequent terrain analysis?, Journal of Hydrology 332 (1-2) (2007) 30–39. doi:http://dx.doi.org/10.1016/j.jhydrol.2006.06.020.

[15] C. Wallis, D. Watson, D. Tarboton, R. Wallace, Parallel flow-direction and contributing area calculation for hydrology analysis in digital elevation models, in: PDPTA, 2009, pp. 467–472.

[16] L. Arge, J. S. Chase, P. Halpin, L. Toma, J. S. Vitter, D. Urban, R. Wickremesinghe, Efficient flow computation on massive grid terrain datasets, Geoinformatica 7 (4) (2003) 283–313. doi:http://dx.doi.org/10.1023/A:1025526421410.

[17] L. Arge, L. Toma, J. S. Vitter, I/o-efficient algorithms for problems on grid-based terrains, J. Exp. Algorithmics 6 (2001) 1.

[18] H.-T. Do, S. Limet, E. Melin, Parallel computing of catchment basins in large digital elevation model, in: HPCA (China), Vol. 5938 of LNCS - Lecture Notes in Computer Science, Springer-Verlag, Berlin Heidelberg, 2009, pp. 133–138.