



ELSEVIER

Parallel Computing 24 (1998) 1981–2001

---

---

PARALLEL  
COMPUTING

---

---

Practical aspects

## Parallel watershed transformation algorithms for image segmentation

Alina N. Moga<sup>a,\*</sup>, Bogdan Cramariuc<sup>b,1</sup>, Moncef Gabbouj<sup>b,2</sup>

<sup>a</sup> *Albert-Ludwigs-Universität Freiburg, Institut für Informatik, Am Flughafen 17, D-79110 Freiburg, Germany*

<sup>b</sup> *Tampere University of Technology, Signal Processing Laboratory, P.O. Box 553, FIN-33101 Tampere, Finland*

Received 10 March 1997; received in revised form 15 March 1998

---

### Abstract

The watershed transformation is a mid-level operation used in morphological image segmentation. Techniques applied on large images, which must often complete fast, are usually computationally expensive and complex entailing efficient parallel algorithms. Two distributed approaches of the watershed transformation are introduced in this paper. The algorithms survey in a Single Program Multiple Data (SPMD) model both local and global connectivity properties of the morphological gradient of a gray-scale image to label connected components. The sequentiality of the serial algorithm is broken in the parallel versions by exploiting the ordering relation between two neighboring pixels successively incorporated in the same region. Thus, a path is traced, for every unlabeled pixel, down to its region of inclusion (whose label is then propagated backwards); in the second algorithm, regions grow independently around their seeds. In both cases only pixels which satisfy the ordering relation are incorporated in any region. This way, not only different regions are explored in a parallel fashion, but also different parts of the same region, when the latter extends to neighboring subdomains, are treated likewise. Running time and relative speedup evaluated on a Cray T3D parallel computer are used to appreciate the performance of both algorithms. © 1998 Elsevier Science B.V. All rights reserved.

*Keywords:* Watershed transformation; Morphological segmentation; Image processing; Rainfalling simulation; Hillclimbing simulation; Complexity analysis; Experimental results

---

---

\* Corresponding author. E-mail: moga@informatik.uni-freiburg.de

<sup>1</sup> E-mail: crama@cs.tut.fi

<sup>2</sup> E-mail: moncef@cs.tut.fi

## 1. Introduction

The watershed transformation applied to a gray-scale image detects and labels objects, which are connected components of similar gray-level. The classical fashion to describe the construction of watersheds identifies an image with a topographical surface. Starting by piercing holes in the regional minima (connected plateaus of constant altitude from which it is impossible to reach a location of lower altitude without having to climb) of the surface, and then slowly sinking it into a lake, water will progressively engulf the basins enclosing various minima. To prevent water from intermingling at the border between different basins, a hinder is set up. Once the surface is completely covered by water, the set of obstacles depicts the watershed image. Various definitions of watersheds have been proposed in the literature (see Refs. [2,8,9,21,23]) for both digital and continuous spaces.

The watershed transformation is prevalently used in industrial, biomedical, and computer vision applications. As concrete examples, we mention here the contribution of watersheds to an automatic system of analysis of images acquired during oil exploration in Ref. [22], road traffic analysis [4], and also to segmentation of electrophoresis gels [9], a moving heart (in nuclear medicine) [6], and 3D holographic images [3].

Initial efforts in the area of parallel implementation of watersheds came from the work in Refs. [2,5,8,9] since for large images the complexity of the analysis entails fast parallel algorithms. Previous experiments of parallelization of watersheds [11,16] using the classical sequential algorithm based on an ordered queue [2,9] resulted in not too efficient implementations. The reason is the highly sequential nature of the approach itself. By parallelizing it, the global ordered queue is divided into several local ordered queues. Therefore, if pixels in one queue in the global ordered queue are disseminated in different local ordered queues, a strong synchronization between the processors maintaining those particular local queues is required. In addition, due to the domain decomposition, repeated relabelings of the same pixels are performed to appropriately label parts of catchment basins which do not lie on the same subdomain as the regional minima from which the basins were generated.

Parallel realizations of watersheds based on image integration and sequential raster and antiraster scannings proved to be scalable, but still computationally expensive because of the repeated scannings [12,16,17].

Other parallel implementation of watersheds can be found in Refs. [1,10,18].

Due to the recursive nature of the watershed transformation, its parallelization is not a trivial task. In this paper, we aim to exploit a local property of the watershed flooding in order to break the sequentiality of the classical immersion simulation and increase the data locality. As in Ref. [8], a predecessor–successor flooding relation between two consecutive neighboring pixels incorporated into the same catchment basin is established. Further on, following these flooding connections between pixels, shortest paths between non-minima and minima, to which region the non-minima belong, are tracked and labeled. For this purpose, two SPMD algorithms are presented in this paper. Although the approaches are different, the goal remains the

same, namely, to split the morphological gradient of a gray-scale image, regarded as a topographic surface, into geodesic influence zones (see Ref. [2]).

Both algorithms set labels in separate stages for minima and non-minima, and differences occur in the latter stage. In the first algorithm, based on rainfalling simulation, every non-minimum pixel on a path a drop of water would slide toward a minimum is labeled, when the drop starts falling from that particular non-minimum point in the topographical relief. Equivalently, one can start and “walk” downward following a steepest slope line (a path  $P = \{p_0, \dots, p_s\}$ , where for any pair  $(p_i, p_{i+1})$ ,  $p_{i+1}$  is the lowest altitude neighboring pixel of  $p_i$ , or the next closer pixel to a lower gray-level pixel, in the case of a plateau) toward a minimum pixel, and propagate the label of the latter backward along the whole path (see Fig. 1); hence, the name “rainfalling simulation”.

Alternatively, in the second algorithm, water springs from regional minima and immerses higher neighborhoods which have not been already flooded (see Fig. 2). This bottom-up method is called “hillclimbing simulation”. However, neither of the methods constructs dams between neighboring areas of labels (0-width watersheds).

In every stage, local connectivity is first exploited to compute labels within each subimage, and then the results are merged to set labels for the whole image. Both methods are parallel by nature, solving a multiple origins and multiple destinations (non-minima/minima) connectivity problem [7]. Preliminary results of this work have been published in [13–15].

The paper is organized as follows. In Section 2, common definitions used in the paper are introduced. Section 3 presents two parallel watershed algorithms; the first is based on rainfalling simulation, and the second on hillclimbing simulation.

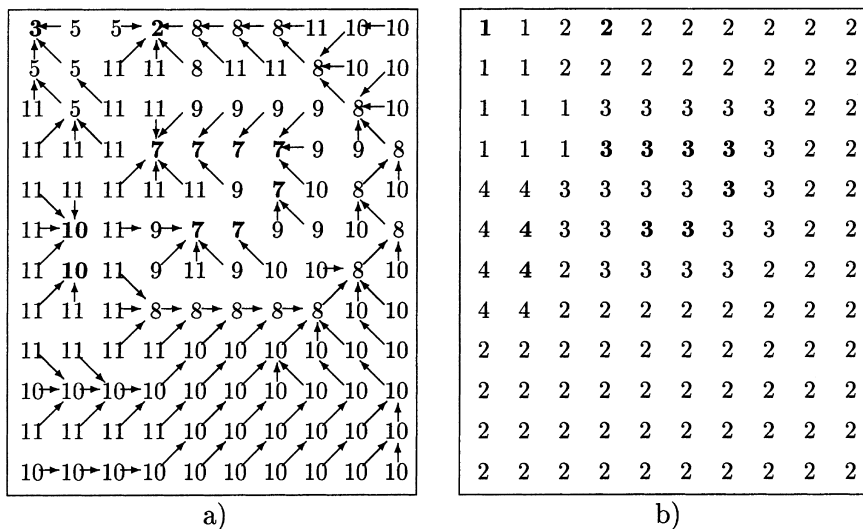


Fig. 1. (a) Rainfalling in the input image; (b) output image of labels.

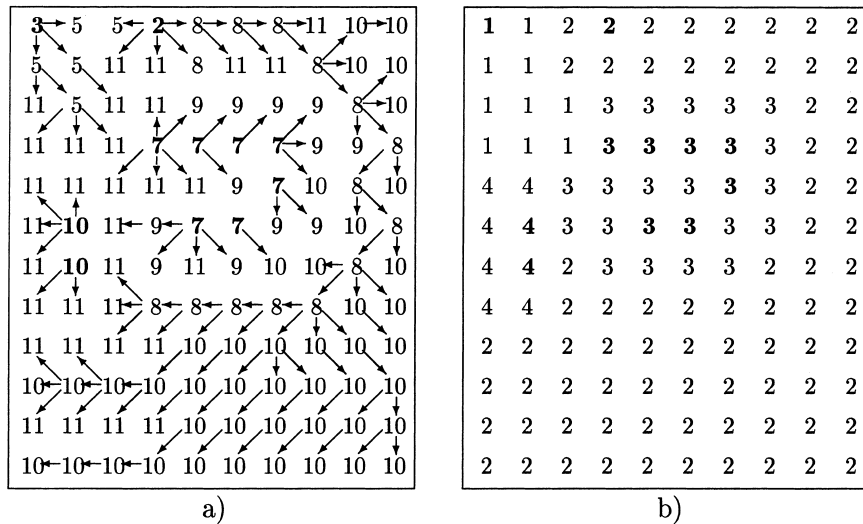


Fig. 2. (a) Hillclimbing in the input image; (b) output image of labels.

Section 4 summarizes some key insights into both implementations and their complexity along with experimental results. Finally, the paper is concluded in Section 5.

## 2. Preliminary definitions

Let  $D_f \subset \mathbf{Z}^2$  denote the domain of a two-dimensional (2-D) digital gray-scale image  $f$ . The underlying grid  $G$  of  $f$  is square using four- or eight-connectivity. Also let us denote by  $N_G(p) = \{p' \in \mathbf{Z}^2 | (p, p') \in G\}$  the set of neighboring pixels of a pixel  $p$  with respect to the grid  $G$  (see Ref. [23] pp. 584).

The parallelization strategy is based on the distribution of an image in a chess-board manner into  $P$  subimages, corresponding to a mapping of the problem to  $P$  processors. Thus, the global domain  $D_f$  is split to equal-sized and disjoint subdomains  $D_{f_i^d}$  – *distribution subdomains* – i.e.  $D_f = D_{f_0^d} \cup D_{f_1^d} \cup \dots \cup D_{f_{P-1}^d}$  and  $D_{f_i^d} \cap D_{f_j^d} = \emptyset, i \neq j$ . Local computations in any location of the distribution subdomain access a regular surrounding neighborhood of that location in four- or eight-connectivity. In order to perform near the edges of each distribution subdomain, the latter is enlarged with an area of one grid point width, such that the underlying subgrids are overlapping. Consequently,  $D_{f_i} = \{p \in D_f | (N_G(p) \cup \{p\}) \cap D_{f_i^d} \neq \emptyset\}$  defines the *overlapping subdomains*. The space enclosed by  $D_{f_i} \setminus D_{f_i^d}$  is called *extension area*. With this dispersion of the image  $f$  into subimages  $f_i$ , one can further define the *neighboring subimages* of a subimage  $f_i$  as the set  $N_G(f_i) = \{f_j | D_{f_i} \cap D_{f_j^d} \neq \emptyset, j \neq i\}$ . In addition, if  $f_i$  belongs to the workspace of processor  $P_i$ , then  $N_G(P_i) = \{P_j | f_j \in N_G(f_i)\}$  denotes the set of *neighboring processors* of  $P_i$ . Let us observe that the notation  $N_G(\cdot)$  has been used to designate the square

type neighborhood of a generic object in a grid topology – pixel, subimage, processor.

It can be noticed that an overlapping subimage  $f_i$  has two distinct parts: the distribution subdomain, in which computation is actually done, and the encompassing extension area. Let  $E(f_i, f_j) = D_{f_i^d} \cap D_{f_j}$  denote the *edge* of the distribution subimage comprised in the overlapping subimage  $f_i$ , neighboring the subimage  $f_j$ . Similarly, the *boundary* between two neighboring subimages  $f_i$  and  $f_j$  is given by  $B(f_i, f_j) = D_{f_i} \cap D_{f_j^d}$ .  $B(f_i, f_j)$  acts as a replica in  $P_i$ 's workspace of the edge  $E(f_j, f_i)$  of the subimage  $f_j$  stored in the adjacent processor  $P_j$ . A  $3 \times 3$  division of a  $6 \times 6$  image is illustrated in Fig. 3. The thick rectangle delimits an extended overlapping subdomain around its corresponding distribution subdomain, represented by the shaded area.

The herein introduced algorithms extensively use the terms of minimum and non-minimum. It can be observed in Fig. 1(a) that pixels within plateaus of minima (shown in boldface) have only neighboring pixels of higher or equal altitude. However, some pixels of altitude 8 belonging to the non-minima plateau starting at location (0,4) (relative to the (0,0) top left corner) have additionally neighboring pixels of lower altitude. These two categories of pixels within connected plateaus are next defined.

**Definition 1.** A pixel  $p \in D_f$  is an *inner* pixel of a connected plateau of altitude  $h$  in an image  $f$  if  $h = f(p) \leq f(p') \forall p' \in N_G(p)$ .

**Definition 2.** A pixel  $p \in D_f$  is an *outer* pixel of a connected plateau of altitude  $h$  in an image  $f$  if  $\exists p' \in N_G(p)$  such that  $h = f(p) = f(p')$  and  $p'$  is an inner pixel of the plateau, and  $\exists p'' \in N_G(p), f(p'') < f(p)$ .

**Definition 3.** A *plateau of minima* of altitude  $h$  within an image  $f$  is a connected plateau of inner pixels only. If the plateau contains a single pixel  $p$  such that  $h = f(p) < f(p'), \forall p' \in N_G(p)$ , it is also a plateau of one minimum.

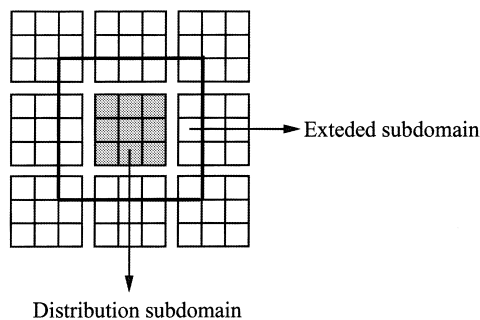


Fig. 3. Regular decomposition in overlapping subdomains.

**Definition 4.** A plateau of non-minima of altitude  $h$  within an image  $f$  is a connected plateau which has outer pixels.

Notice that the one-pixel wide plateau of altitude 9 starting at location (2,4) in Fig. 1(a) does not conform the definition of a non-minima plateau since it has no outer pixels. Recall that an outer pixel has not only a lower neighbor than itself, but also a neighbor being an inner pixel, which in the exemplified case does not exist; all pixels in the plateau have lower neighboring pixels.

Both algorithms rely on ordered flooding.  $p \succ q$  if  $p$  has higher altitude than  $q$ , or  $p$  and  $q$  have the same altitude, but  $q$  is closer to a lower border of its plateau than  $p$  is (see Ref. [9] pp. 27). The lower distance is a measure of closeness of a pixel within a plateau of non-minima to a lower border.

**Definition 5.** The lower distance  $d$  is defined as follows:  $d(p) = 0$  if  $p$  is a minimum; otherwise, it equals the length  $s$  of the shortest path  $\{p = p_0, p_1, \dots, p_s = q\}$  between pixels  $p$  and  $q$  such that  $\forall i \in \{1, 2, \dots, s\}$ ,  $(p_{i-1}, p_i) \in G$ ,  $f(q) < f(p)$ , and, if  $s > 1$ ,  $\forall i \in \{1, 2, \dots, s-1\}$ ,  $f(p_i) = f(p_{i-1}) (= f(p))$ .

The above mentioned 2-D ordering relation – in gray-level and lower distance – can be reduced to 1-D with the lower-complete transformation defined next.

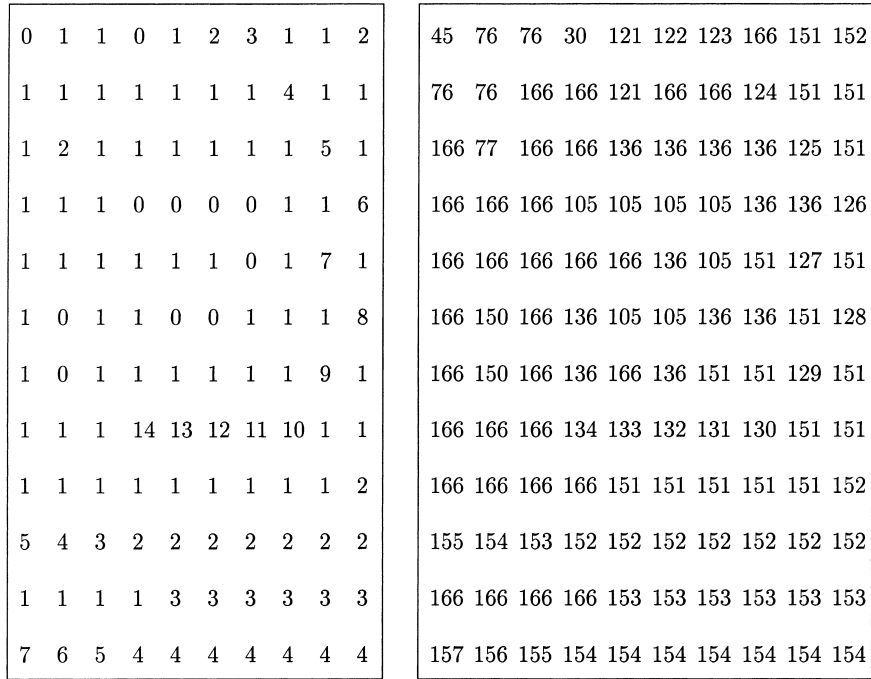
**Definition 6.** The mapping  $l(p) = \lambda \times f(p) + d(p)$ ,  $\forall p \in D_f$ , is called the lower-complete transformation of an image  $f$  based on the lower distance image  $d$ .  $\lambda$  is a constant larger than any lower distance value (for example  $\max_{p \in D_f} \{d(p)\} + 1$ ).

With the above mapping, any non-minimum pixel in the lower-complete image has a neighbor of lower altitude than itself (there exists no plateau, except the regional minima). Fig. 4(a) illustrates the lower distance image of the input image in Fig. 1(a), while in Fig. 4(b) the lower-complete transformed image can be observed ( $\lambda = 15$ ). As a result,  $p_1 \succ p_2$  iff  $l(p_1) > l(p_2)$ . One can easily proof that if  $f(p_1) < f(p_2)$  or, if  $f(p_1) = f(p_2)$  and  $d(p_1) < d(p_2)$ , then  $l(p_1) < l(p_2)$ ; and, reversely, if  $l(p_1) < l(p_2)$  then  $f(p_1) < f(p_2)$  or  $f(p_1) = f(p_2)$  and  $d(p_1) < d(p_2)$ ; otherwise, if  $l(p_1) = l(p_2)$  then  $f(p_1) = f(p_2)$  and  $d(p_1) = d(p_2)$ .

Consequently, by flooding in increasing order of the lower-complete value, a non-minimum pixel  $p$  is flooded from its steepest lower-complete neighbor, information which can be locally inquired. This is the key solution for splitting the sequentiality of the watershed transformation and allowing parallel threads to develop.

### 3. The parallel watershed algorithms

A divide-and-conquer parallel implementation of the watershed transformation based on rainfalling and respectively hillclimbing simulation in a lower-complete image is below described. First, minima are detected and labeled, lower distances



a) b)  
 Fig. 4. (a) Lower distance image; (b) lower-complete image.

inside non-minima plateaus computed, and the lower-complete image generated. Next, raffleing and hillclimbing of non-minima pixels are separately presented.

### 3.1. Detection and labeling of the minima pixels

In this stage, computation of the lower distance inside plateaus of non-minima pixels and detection and labeling of each regional minimum (a connected plateau of minima) with an unique label are performed at the same time. The pseudo-code of this stage along with explanations follows next.

**Algorithm 1.** *Detection of minima in each processor  $P_i$*

Initialization: /\* MAX\_LABEL < NARM < OUTBOARD \*/  
 $\forall p \in D_{o_i}^d, o_i(p) = \text{NARM}$  and  $\forall p \in D_{o_i} \setminus D_{o_i}^d, o_i(p) = \text{OUTBOARD}$   
 $\forall p \in D_{d_i}^d, d_i(p) = \text{MAX\_DIST}$  ( $= \infty$ ) and  $\forall p \in D_{d_i} \setminus D_{d_i}^d, d_i(p) = 0$   
 Input:  $f_i, o_i, d_i, \lambda_i = 1, \text{current\_label}_i = i \times \frac{\text{MAX\_LABEL}}{P}$ ;  
 Output:  $o_i, d_i, \lambda_i, \lambda$ .  
 (1) Raster scan( $p \in D_{f_i}^d$  not visited before) {  
     (1.1) Inquiry the neighborhood of  $p$

```

(1.2)if  $p$  is a regional minimum then
     $o_i(p) \leftarrow current\_label_i; current\_label_i \leftarrow current\_label_i + 1;$ 
(1.3)else if  $p$  is an inner pixel then explore the plateau originating in  $p$ 
}
(2) $state \leftarrow ON$ 
(3)while( $state = ON$ ) {
    (3.1) $state \leftarrow OFF$ 
    (3.2)for each( $P_j \in N_G(P_i)$ ) SendReceive( $E(o_i, o_j), E(d_i, d_j), B(o_i, o_j), B(d_i, d_j)$ )
    (3.3)for each( $P_j \in N_G(P_i)$ ) UpdateEdge( $j, f_i, o_i, d_i, state_i$ )
    (3.4)UpdateDistributionSubdomain( $f_i, o_i, d_i, \lambda_i$ )
    (3.5)All_Reduce( $state_i, state, OR$ )
}
(4)All_Reduce( $\lambda_i, \lambda, MAX$ )

```

Step (1) comprises the pseudocode for the serial execution of this stage. Thus, at step (1.1), by checking the graylevels of the neighboring pixels in four- or eight-connectivity,  $p$  can be classified as a local minimum (strictly higher neighborhood), case in which  $p$  is labeled with the current label at step (1.2), an inner pixel (higher or equal neighborhood), or a non-minimum (otherwise).

At step (1.3), a plateau is scanned in a breadth-first order, and visited pixels are labeled with the current label value. The examination always starts from an inner pixel which introduces in the list of candidates neighboring pixels of equal altitude (recall that an inner pixel which is not a local minimum has such a neighborhood). A currently investigated candidate pixel  $q$  may still satisfy the inner condition; otherwise, it has lower neighbors, and  $q$  is an outer pixel. In the latter case, according to the definition, the lower distance value is 1, and  $q$  is stored in a list. If, after scanning the plateau, the list of outer pixels is not empty, another breadth-first scan is performed to reset the label of the plateau to NARM, compute the lower distance function, and update the value  $\lambda_i = \max_{r \in D_i^d} \{d_i(r)\} + 1$ . For this purpose, a wave front, comprising at the beginning all outer pixels, drifts iteratively and exhaustively across the plateau. Pixels swept by a wave set the lower distance to the time stamp of the wave, which is initially 1 and is incremented at each iteration. The partial labels and lower distances inside non-minima plateaus for the image example in Fig. 1(a) are illustrated in Fig. 5, when four processors are used ('M' stands for MAX\_DIST and 'N' for NARM). Let us notice that for non-minima pixels which are not on a non-minima plateau, as well as for minima pixels, the lower distance remained MAX\_DIST.

Since in the parallel implementation, due to the domain decomposition, a plateau may extend over more than one subdomain, step (1) solely does not produce either a unique label, in the case of a plateau of minima (processors use disjoint ranges of labels) (see the plateau of graylevel 7 in Fig. 1(a)), or the lower distance is inaccurate because lower brims of the plateau exist in remote subdomains. Moreover, in the latter case, when such lower brims are located only remotely, the part of the plateau analyzed locally will be wrongly classified as a minima plateau (see the plateau of altitude 8 which has two NARM pixels in the top left subimage and minima labeled

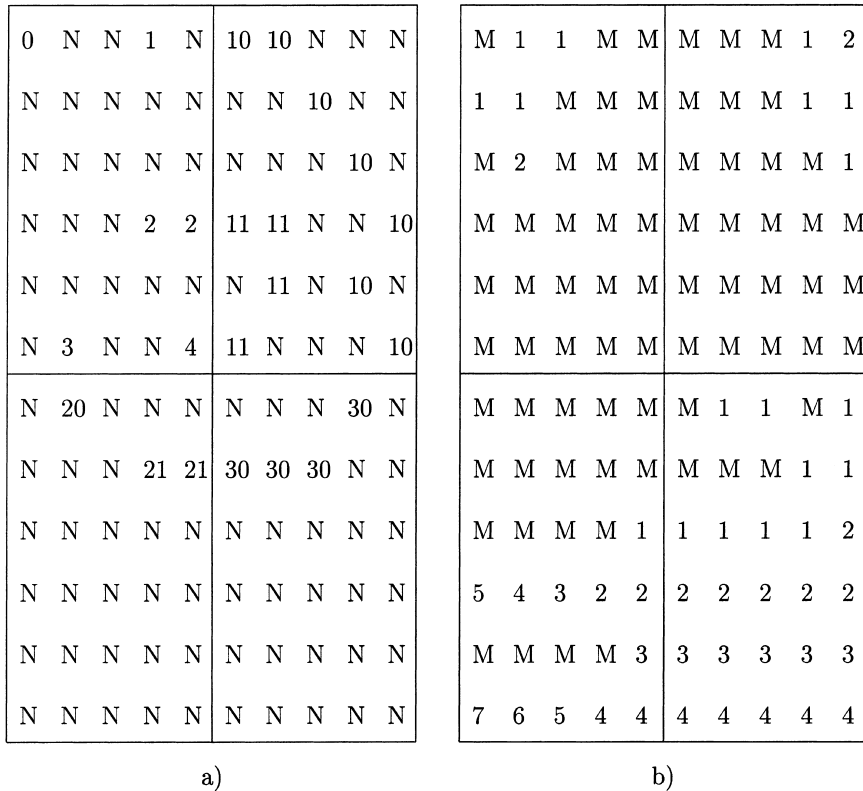


Fig. 5. (a) Local labels for minima plateaus; (b) local lower distances for non-minima plateaus.

pixels in the other subimages). Therefore, further steps are performed to update the partial results of step (1) and produce global values.

At step (3.2), the *SendReceive* routine combines in one call sending of the edge  $E(o_i, o_j)$  and  $E(d_i, d_j)$  to processor  $P_j$  and receiving of labels and distances in  $B(o_i, o_j)$  and  $B(d_i, d_j)$  from another processor  $P_j$ . A Message Passing Interface (MPI) function [20] is used, which actually shifts values across the regular grid topology of processors.

At step (3.3), parts of the same plateau from different subimages are merged by adjusting first the label and lower distance of pixels in each subimage edge based on the non-local neighborhood replicated by message passing in the associated boundaries. Thus, if neighboring pixels pertain to the same minima plateau, but have different labels, relabeling with the minimal label is performed. For example in Fig. 5(a), pixels (3,4) and (3,5) of altitude 7 labeled 2 and 11, respectively; consequently pixel (3,5) will be relabeled 2. Furthermore, the plateau of altitude 8 has been classified as minima in three subimages and, at the same time, it has NARM pixels in the top left subimage, pixel (0,4) and (1,4). Consequently, the latter pixels become

outer pixels in the top left subimage, and hence their lower distance equals 1, while pixel (0,5) resets its label to NARM and sets its lower distance to 2. Finally, in other situations corrections of the lower distance inside plateaus of non-minima are performed, such that the distances are computed according to the entire set of distributed outer pixels. Therefore, for each edge pixel  $p$ , the lower distance is update based on the values of its neighbors in the extension area:  $d(p) = \min_{r \in N_G(p) \cap B(o_i, o_j)} \{d(p), d(r) + 1 | f_i(r) = f_i(p)\}$ . Moreover, if  $o_i(r) = \text{NARM}$  and  $o_i(p) \neq \text{NARM}$ , then  $o_i(p) \leftarrow \text{NARM}$ . Changes in label and/or distance are then propagated in every subdomain by a breadth-first scan at step (3.4). Whenever modifications in the distance subimage occur, the value of  $\lambda_i$  is also updated. Modifications in both distance and label are marked by setting the variable  $state_i$  to ON. If the merging operation leaves all pixels unchanged,  $state_i$  remains unchanged, too.

Based on all local states, the moment of termination of the loop (3) is detected at step (3.5). The new state is computed by a global OR reduction operation such that the result is available at every processor:  $state = state_0 \text{ OR } state_1 \text{ OR } \dots \text{ OR } state_{p-1}$  (see Ref. [20]). Consequently, if  $state$  is ON in at least one processor, all processors keep exchanging messages; otherwise, the computation has stabilized and the processors stop intercommunicating. The global images of labels and lower distances inside non-minima plateaus can be observed in Fig. 6.

The global value  $\lambda = \max_{0 \leq i < P} \{\lambda_i\}$  is computed by a global reduction operation at step (4), using the MAX operator [20], such that the value of  $\lambda$  will be available at every processor (in our example  $\lambda = 15$ ). Each processor  $P_i$  applies next the rule to transform its subimage  $f_i$  into a lower-complete version  $l_i$  based on the lower distance subimage  $d_i$ :  $l_i(p) = \lambda \times f_i(p) + d'_i(p)$ , where  $d'_i(p) = 0$  if  $o_i(p) < \text{NARM}$ ,  $d'_i(p) = d_i(p)$  if  $d_i(p) < \text{MAX\_DIST}$  and  $o_i(p) = \text{NARM}$ , otherwise  $d'_i(p) = 1$ . The values in  $l_i$ ,  $0 \leq i < 4$  can be observed in Fig. 4(b). In the following, the algorithm acts on the graph representation of the image. Pixels within the image are vertices in the graph, and there is a directed arc from a pixel  $q$  to a neighboring pixel  $p$ , if  $l(q) > l(p)$ , and  $l(p) = \min\{l(t) | t \in N_G(q)\}$  ( $p$  is the steepest neighbor of  $q$ ). The directed graph thus defined is acyclic (it is not possible to return to the same point following only descending paths) and its arrows actually coincide with the ones in Fig. 1(a), but imposed on the lower-complete image in Fig. 4(b). Therefore, the graph is a forest whose trees contain non-minima as internal nodes and minima which have non-minima in their neighborhood as leaves.

### 3.2. Flooding of the non-minimum pixels

The labeling problem is restated as follows. Given a forest in which the leaves are labeled, it is desired to label every unlabeled (NARM) pixel in the forest with the label of a leaf with which it is connected by a path. Two algorithms are provided for generating such labelings: first, based on the top-down parsing (rainfalling simulation) of the paths, and second, based on the bottom-up flooding (hillclimbing simulation).

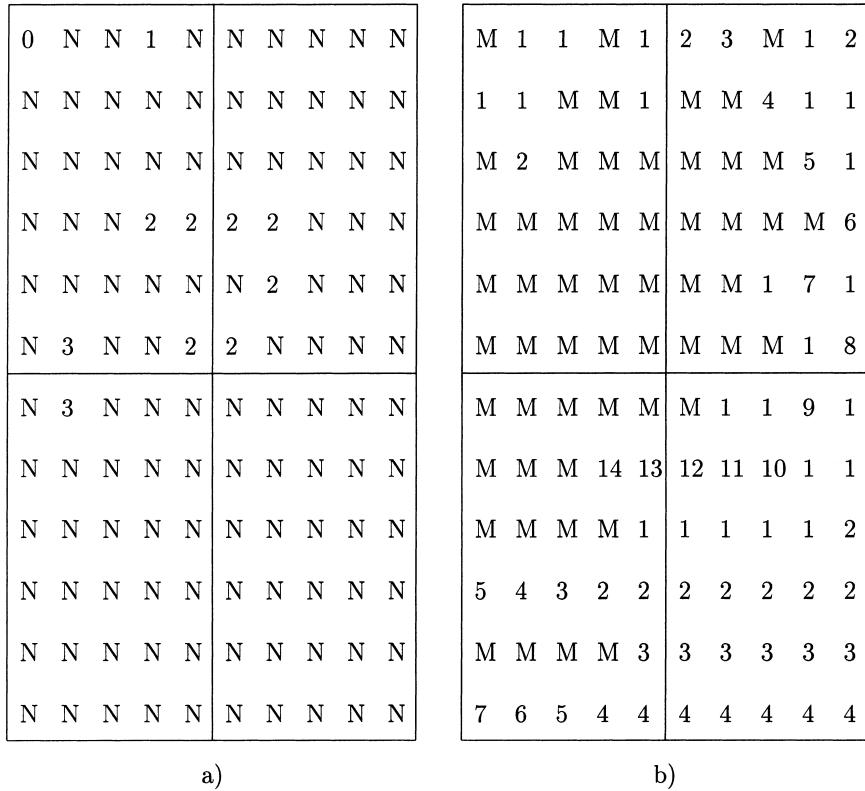


Fig. 6. (a) Global labels for minima plateaus; (b) global lower distances for non-minima plateaus.

### 3.2.1. Flooding based on the rainfalling simulation

Rainfalling technique is derived from a basic principle from physics: a drop in free fall on a descending topographic surface will move due to gravity downward to the deepest location until it reaches a minimum. Thus, by computing paths originating in non-minimum pixels (internal nodes in the forest) and ending in minima (leaves) rainfalling as in Fig. 1 is simulated.

**Algorithm 2.1** Labeling of non-minima by rainfalling simulation in processor  $P_i$

Input:  $o_i, l_i, next_i, stencil_i$ ;

Output:  $o_i$ .

- (1) Raster scan( $p \in D_{i_q}$  not visited before) {
  - (1.1)Depth-first scan from  $p$  to  $q$
  - (1.2)if  $o_i(q) < NARM$  then /\* resolved path \*/
    - for( $crt = p; crt \neq q; crt \leftarrow next_i(crt)$ )
      - $o_i(crt) \leftarrow o_i(q)$ ;
  - (1.3)else PUT\_QUEUE2( $p, q$ ) /\* unresolved path \*/

```

(2) while the queue is not empty {
  (2.1)for each( $P_j \in N_G(P_i)$  and  $stencil_i(P_j) = \text{TRUE}$ )
    Swap( $E(o_i, o_j), B(o_i, o_j), stencil_i(P_j)$ )
  (2.2)for each pair ( $(head, tail) = \text{GET\_QUEUE2}()$ )
    do steps (1.2),(1.3) with  $p = head$  and  $q = tail$ 
}

```

At step (1), each processor  $P_i$ , in a raster scan over its distribution subdomain  $I_i^d$ , initiates for every non-minimum pixel, which has not been already visited, a search for a minimum. The latter can be reached along a path by following successive arcs in the graph representation of the lower-complete image. An additional subimage  $next_i$  stores for each non-minimum pixel the following pixel in the path it lies on. Thus, at step (1.1), if  $p_{crt}$  is the terminal pixel in the path, the scan either stops, if  $p_{crt}$  already lies on another path or is a minimum; otherwise, it extends with a steepest lower-complete neighbor  $p_{next}$  of  $p_{crt}$ . In the first case  $q = p_{crt}$  and in the latter case  $next_i(p_{crt}) \leftarrow p_{next}$ ,  $p_{crt} \leftarrow p_{next}$ , and the scan continues.

If the whole path is comprised in  $I_i^d$ , all pixels along the path are labeled with the label of the ending pixel at step (1.2). Otherwise, the path extends to other subdomains and cannot be resolved by  $P_i$  solely. Therefore, the path is unresolved and the head and tail of the path are stored in a queue, awaiting to be resolved. Let us notice that a path is unresolved not only when it hits the edges of the distribution subdomain, but also when it intersects another unresolved path, analyzed before the current path. A FIFO queue stores the terminal pixels of an unresolved queue at step (1.3). PUT\_QUEUE2 and GET\_QUEUE2 are the two operators performing insertion and removal of a pair from the queue. The result of step (1) can be observed in Fig. 7(a). Thus, all pixels on resolved paths are labeled, while the other pixels are on unresolved paths and hence still NARM labeled.

Step (2) aims to resolve all unresolved paths through communication. A dynamic, unstructured communication pattern is used at step (2.1). Thus, processor  $P_i$  accounts the neighbors it communicates with based on an array of flags  $stencil_i$ . At the initialization time,  $\forall P_j \in N_G(P_i)$ ,  $stencil_i(P_j) = \text{TRUE}$ . If  $stencil_i(P_j) = \text{TRUE}$ , communication with  $P_j$  takes place; otherwise, not. If  $P_i$  receives in the boundary  $B(o_i, o_j)$  only labeled pixels, it will not swap again labels with processor  $P_j$  at the next iterations since the maximum information from  $P_j$  upon labels has been already received. Consequently,  $stencil_i(P_j) \leftarrow \text{FALSE}$ . Complementary, if  $P_i$  sends a fully labeled edge of pixels  $E(o_i, o_j)$  to  $P_j$ , it resets the flag  $stencil_i(P_j)$  since no further information is needed from  $P_j$ , or provided to  $P_j$ .

At step (2.2), for every pair ( $head, tail$ ), initially stored into the queue, the  $tail$  pixel is checked if it has been labeled. If  $o_i(tail) < \text{NARM}$ , then the whole path from  $head$  to  $tail$  is labeled with  $o_i(tail)$ . Otherwise, the pair is reinserted into the queue.

After the whole queue has been scanned, new changes of labels of edge pixels are recommunicated to the corresponding neighboring processors. A processor keeps communicating and completing paths as long as there are unresolved paths in its subdomain. When all processors become inactive, all pixels have been labeled and

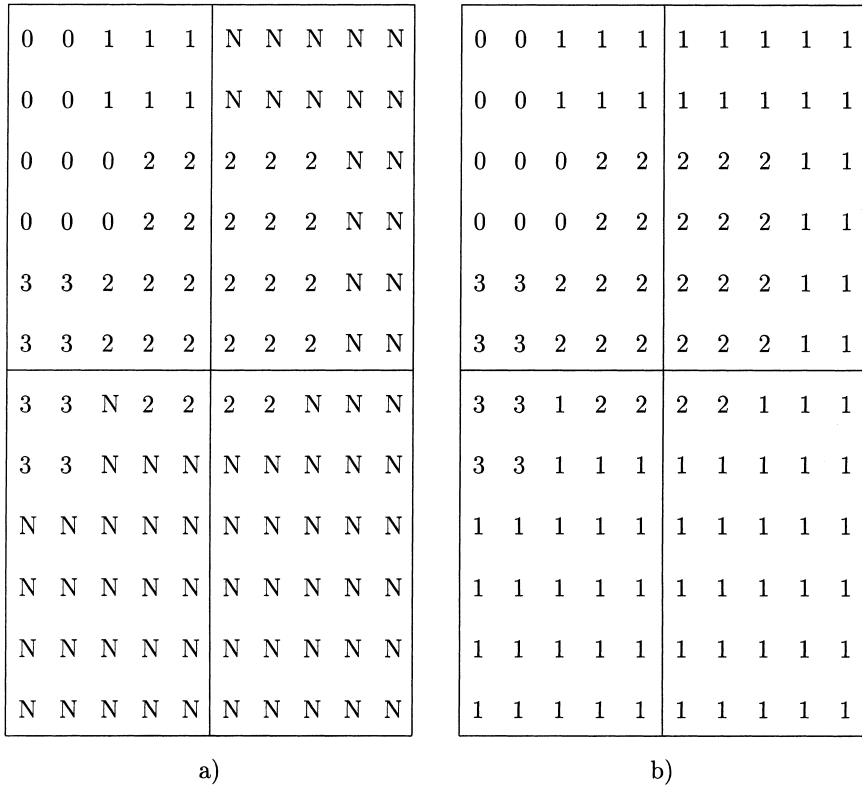


Fig. 7. (a) Labels after the local rainfalling simulation; (b) Global labels.

the algorithm stops. Unlike in the previous stage, a processor can decide locally when to terminate the communication loop. No global decision is needed, eliminating thus the overhead of a global reduction operation. Another positive aspect is that no relabelings and no synchronization between paths are required. To each NARM pixel, a single downward pixel from which it gets the label has been stored in  $next_i$ , such that there is no competition in setting a label to a non-minimum pixel. The global labels are illustrated in Fig. 7(b).

### 3.2.2. Flooding based on hillclimbing simulation

An algorithm for computing bottom-up the steepest slope paths as in Fig. 2 is next described.

**Algorithm 2.2** Labeling of non-minima by hillclimbing simulation in processor  $P_i$

Input:  $o_i, l_i, slc_i, stencil_i$ ;

Output:  $o_i$ .

(1)Raster scan( $p \in D_{o_i^t}$ ) {

```

    if  $o_i(p) < \text{NARM}$  and  $\exists q \in N_G(p)$ ,  $o_i(q) = \text{NARM}$  then
      PUT_QUEUE( $p$ )
    else  $slc_i(p) \leftarrow \min\{l_i(q) | q \in N_G(p), l_i(q) < l_i(p)\}$ 
  }
(2)while the queue is not empty {
  (2.1) $p = \text{GET\_QUEUE}()$ 
  (2.2)for each  $q \in N_G(p) \cap D_{i^d}$ 
    if  $o_i(q) = \text{NARM}$  and  $l_i(p) = slc_i(q)$ 
       $o_i(q) \leftarrow o_i(p)$ ; PUT_QUEUE( $q$ )
  }
(3)while(TRUE) {
  (3.1)for each( $P_j \in N_G(P_i)$  and  $stencil_i(P_j) = \text{TRUE}$ )
    Swap( $E(o_i, o_j), B(o_i, o_j), stencil_i(P_j), stencil'_i(P_j)$ )
  (3.2)for each( $P_j \in N_G(P_i)$  and  $stencil_i(P_j) = \text{TRUE}$ )
    for each( $p \in B(o_i, o_j)$  and  $o_i(p) < \text{NARM}$ )
      for each( $q \in N_G(p) \cap E(o_i, o_j)$  and  $o_i(q) = \text{NARM}$  and  $slc_i(q) = l_i(p)$ )
         $o_i(q) \leftarrow o_i(p)$ ; PUT_QUEUE( $q$ )
  (3.3)if queue is empty break
  (3.4)do step (2)
  (3.5) $stencil_i \leftarrow stencil_i$  OR  $stencil'_i$ 
}

```

The data structure which stores the list of candidates for flooding is a FIFO queue initialized at step (1), in each processor, with minima which have non-minima in their neighborhood. During the same raster scan, for every non-minimum (NARM), the steepest lower-complete neighboring altitude is stored into the sub-image  $slc_i$ .

At step (2), flooding starting from the seeds previously accumulated into the FIFO queue is performed. Thus, a candidate  $p$  removed from the queue attributes its label  $o_i(p)$  to all its neighboring pixels  $q$ , if there is an arc from  $q$  to  $p$  in the forest, i.e.,  $slc_i(q) = l_i(p)$ , where  $l_i$  stands for the lower-complete subimage in processor  $P_i$ .  $q$  becomes a candidate and is inserted into the queue.

When the queue of candidates is emptied, either the computed paths hit the boundaries of the distribution subdomain, and they must extend in the neighboring subdomains, or some new candidates from adjacent subdomains must be received in the extension area. Therefore, at step (3.1), processors exchange repeatedly labels of pixels within the corresponding edges of the distribution subdomain. As in Algorithm 2.1, a dynamic unstructured communication pattern is used to avoid multiple transmissions of the same data. However, the  $stencil_i$  array is altered only at the end of the loop, and not within the *Swap* module. After the receipt of new labels in the extension area, at step (3.2), each unlabeled pixel  $q$ , in the edges of the distribution subdomain, is labeled from a pixel  $p$ , in the extension area, if  $p$  has a label and if  $p$  is the steepest lower-complete neighbor of  $q$ .  $q$  becomes a candidate for flooding and is

therefore stored in the FIFO queue. If no such new seeds for flooding have been found, the subimage  $o_i$  is completely labeled and the hillclimbing simulation terminates. Otherwise, a new relabeling like at step (2) is performed at step (3.4).

#### 4. Complexity analysis and experimental results

Both parallel watershed paradigms exploit parallelism in a tile-by-tile fashion. Apart from the lower-complete transformation, each stage parallelizes a global operation in a divide-and-conquer manner.

Regular grid-based distribution is used for a good load balance. Equal-sized block subimages imply roughly the same amount of computation and a faster communication. Two reasons support this statement. On one hand, the length of an edge whose pixels values are exchanged in the combination of partial results is small for a fine division and, on the other hand, processors reach the communication point almost at the same time.

Communication patterns are regular initially, only with the nearest neighborhood, but change when flooding, as processors become inactive. A processor exchanges with every neighbor the corresponding edge of labels and distances. Thus, for an image of size  $n \times n$  distributed to  $P$  processors, the communication time for each phase is  $O(n_{\text{iterations}} \times n/\sqrt{P})$ , where  $n/\sqrt{P}$  is the length of a subimage edge. The number of iterations  $n_{\text{iterations}}$  performed until the computation stabilizes is data dependent. The upper limit is  $n \times (\sqrt{P} - 1)$ , e.g. in the case of a snake or spiral shape communication thread crossing all processors. In the example in Fig. 8 is an

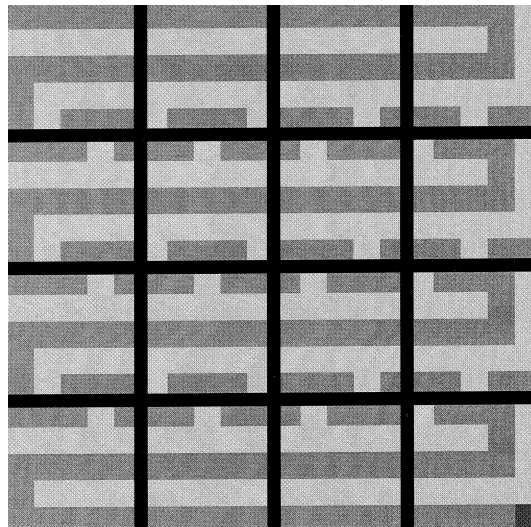


Fig. 8. Snake shape image.

image containing a snake-like plateau which has a lower neighbor only in the down right corner of the image. Thus, the correct lower distance function or the label of the minimum is propagated from processor to processor at each communication step. The communication thread involves a processor as many times the plateau crosses the subimage. The lower limit of  $n_{\text{iterations}}$  is 1 when no plateau crosses the boundaries of the subimage and non-minima pixels can be labeled locally. After a single iteration, the communication terminates. Note that while flooding, a processor becomes inactive as soon as all pixels within the edges are labeled. The above upper limit is still reached, but only by the top left processor; the others need less iterations.

Plateaus are tracked in a breadth-first order while paths are generated by a downward depth-first pass through the forest in Algorithm 2.1 (rainfalling simulation) and in an upward breadth-first scanning in Algorithm 2.2 (hillclimbing simulation).

The local detection and labeling time of plateaus is proportional to the number of pixels in the subimage  $n^2/P$ , but the complexity is larger in the presence of plateaus of non-minima, where the lower distance must be computed. Thus, pixels in non-minima plateaus are processed twice. In the linking phase, are reached only pixels within plateaus for which label and/or lower distance have been wrongly assigned. The global operation used for termination implies a  $O(\log P)$  complexity.

The lower-complete transformation is a point operation and the time consumed is also dependent on the number of pixels in the subimage,  $n^2/P$ .

When flooding locally by rainfalling, non-minima pixels on a resolved path are scanned twice. First downward to track the path, and second, upward to propagate the label which was found. A computational shortcut is used based on the property that if  $q$  is already on a path, any pixel  $p$  from which  $q$  can be reached along a steepest slope line is on the same class as  $q$ . However, the complexity is proportional to the number of non-minima in the subimage. The computational linkage part is on the other hand less expensive. Only the *tail* pixels of all unresolved paths are checked for label, and the propagation of the label is linear, given by the length of the path.

Flooding by hillclimbing has two sources of computational overhead. First, a raster scan is performed to initialize the queue of candidates (step (1) in Algorithm 2.2). Second, repeated scannings of the boundaries to propagate floodings coming from adjacent subdomains increase the complexity of the second stage.

The total running times of the parallel algorithm implemented on top of MPI [20] on the Cray T3D for images of different sizes have been measured and are here presented. The execution time  $T(P)$ , when  $P$  processors are used, is tabulated along with the number of iterations in each stage in Table 1, for the parallel rainfalling technique, and in Table 2, for the hillclimbing method. For both algorithms and all images, the running time drops significantly down from 1 processor to 128, achieving an ascendent relative speedup  $SP(P) = T(1)/T(P)$ . The relative speedup curves can be observed in logarithmic scale in Figs. 9 and 10. Finally one test image along with its segmented version are illustrated in Fig. 11.

Table 1  
Execution time of the rainfaling simulation (in seconds);

Image	Cermet		Lenna		Peppers		People	
Size	256 × 256		512 × 512		512 × 512		1024 × 1024	
#regs	418		5370		2795		14 684	
$P$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$
	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$
1	0.348505	1	1.015189	1	1.164825	1	5.471379	1
	0.554684	1	1.859499	1	1.871093	1	8.342664	1
2	0.272022	5	0.695771	3	1.182142	3	4.383623	4
	0.385878	1	1.145145	2	1.543666	2	5.957823	2
4	0.183724	5	0.539269	3	0.640967	4	3.473508	4
	0.253423	1	0.778327	2	0.830877	2	4.189793	2
8	0.126807	6	0.420088	4	0.469234	5	3.562227	5
	0.171933	1	0.549595	3	0.575644	2	3.995771	2
16	0.097571	9	0.355865	5	0.317673	5	2.686555	7
	0.131355	1	0.439758	3	0.392826	2	2.900425	2
32	0.088755	11	0.328644	8	0.317458	7	1.706675	7
	0.114394	2	0.375848	3	0.364172	3	1.841070	2
64	0.069634	12	0.298520	8	0.252327	8	1.586414	11
	0.081686	2	0.328875	3	0.275530	3	1.669039	2
128	0.075530	15	0.264362	16	0.218425	12	0.940380	13
	0.090181	2	0.283247	3	0.244203	3	0.981220	2

#regs refers to number of regions,  $T^1(\cdot)$  time after the first stage,  $T^{12}(\cdot)$  time after the second stage,  $n_{it}^1$  number of iterations in the first stage, and  $n_{it}^2$  number of iterations in the second stage.

Table 2  
Execution time of the hillclimbing simulation (in seconds)

Image	Cermet		Lenna		Peppers		People	
Size	256 × 256		512 × 512		512 × 512		1024 × 1024	
#regs	418		5370		2795		14 684	
$P$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$	$T^1(P)$	$n_{it}^1$
	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$	$T^{12}(P)$	$n_{it}^2$
1	0.345198	1	1.011121	1	1.152973	1	5.420266	1
	0.975523	1	3.471426	1	3.192832	1	13.326145	1
2	0.251056	5	0.657853	3	1.042509	3	4.039870	4
	0.579019	2	1.941959	4	2.111800	3	8.322568	3
4	0.164664	5	0.478272	3	0.562543	4	2.998173	4
	0.353288	2	1.190707	4	1.136389	4	5.119271	3
8	0.110548	6	0.355754	4	0.396535	5	2.933194	5
	0.220971	2	0.716239	4	0.696846	4	4.258827	3
16	0.084362	9	0.290430	5	0.261583	5	2.120576	7
	0.147138	3	0.495589	4	0.430034	4	2.760969	3
32	0.076255	11	0.261847	8	0.254780	7	1.330196	7
	0.115480	3	0.372468	4	0.352760	4	1.709962	3
64	0.060113	12	0.236042	8	0.202265	8	1.226880	11
	0.083646	3	0.296703	4	0.262258	4	1.415771	3
128	0.062646	15	0.212788	16	0.175073	12	0.724379	13
	0.080247	3	0.248033	4	0.218763	4	0.824115	3

#regs refers to number of regions,  $T^1(\cdot)$  time after the first stage,  $T^{12}(\cdot)$  time after the second stage,  $n_{it}^1$  number of iterations in the first stage, and  $n_{it}^2$  number of iterations in the second stage.

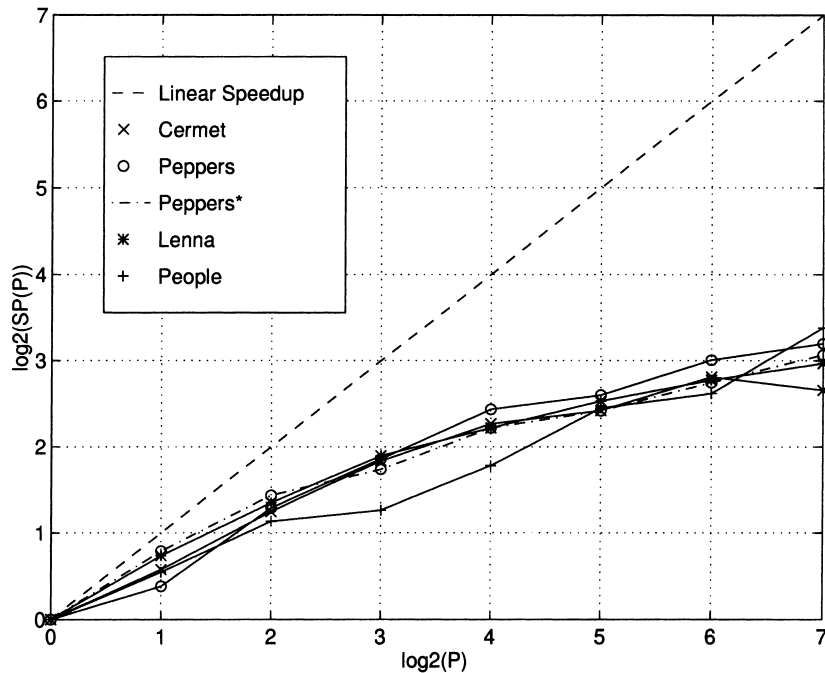


Fig. 9. Relative speedup of the rainfaling simulation.

## 5. Conclusions

Two parallel watershed algorithms were presented here to label connected components for image segmentation. Both algorithms start by detecting first the seeds of the regions, the regional minima of the gradient image. The image is then lower-complete transformed and represented as an acyclic directed graph or forest. Pixels are vertices and there is an arc between any two nodes if their lower-complete values satisfy the given ordering relation. Two methods of propagating the labels of minima, which are leaves, are proposed and compared. One is rainfaling simulation, in which every non-minimum pixel parses arcs downward to a leaf to get labeled; and the other is hillclimbing simulation, in which labels are propagated bottom-up along steepest slope lines.

The aim was to compare the two approaches from the efficiency viewpoint and results. It can be observed that both algorithms are scalable, that is, the total execution time drops down, but not linear, with an increasing number of processors. However, increasing relative speedup is achieved until 64 and 128 processors. As expected, the hillclimbing based method is more time consuming than the rainfaling approach, but it scales better than the latter. Finally, the results of the segmentation are similar to those of the serial watershed algorithms which have been shown to produce good results. The parallel algorithms introduced in this paper do however

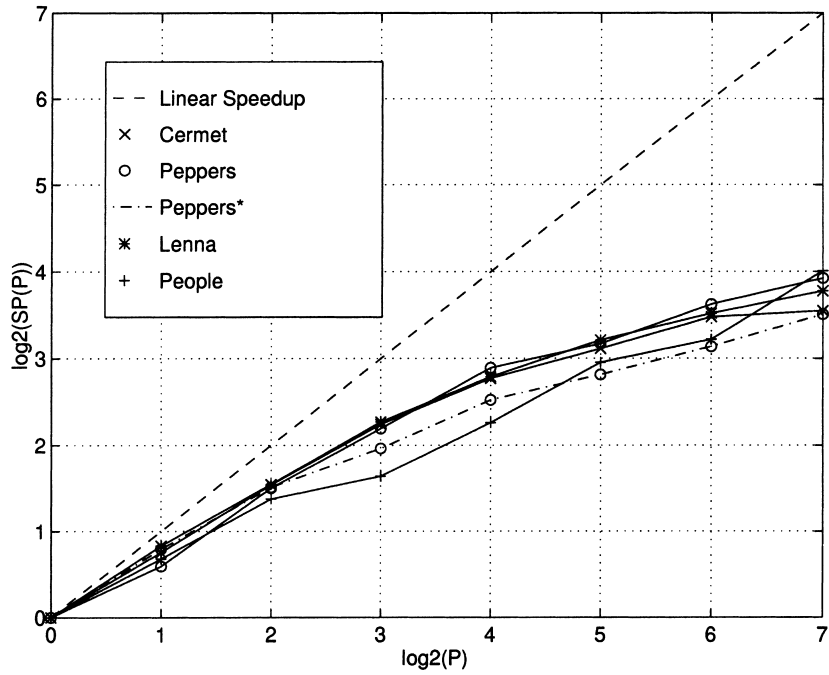


Fig. 10. Relative speedup of the hillclimbing simulation.

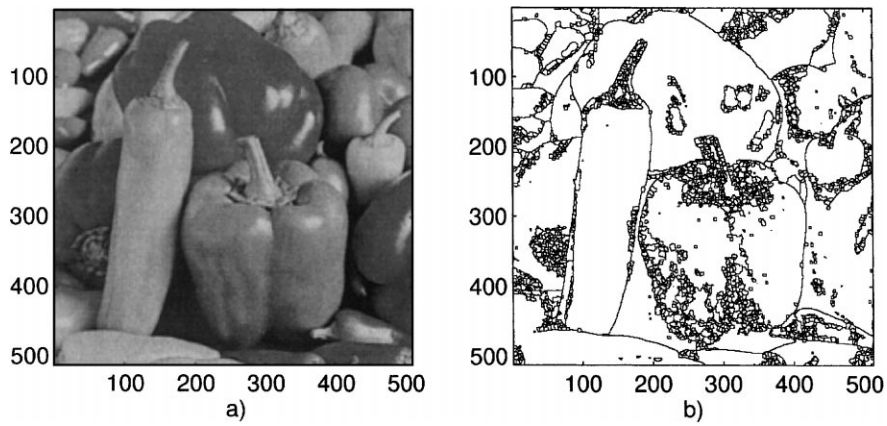


Fig. 11. *Peppers*: (a)Input Image; (b)Output Image.

suffer from the major shortcoming of traditional serial watershed algorithms, that is oversegmentation. A parallel design solution to this problem can be found in Ref. [19].

## Acknowledgements

This research has been supported by the Edinburgh Parallel Computing Centre in the Training and Research on Advanced Computing Systems program.

## References

- [1] A. Bieniek, H. Burkhardt, H. Marschner, M. Nölle, G. Schreiber, A parallel watershed algorithm, in: Proceedings of The Tenth Scandinavian Conference on Image Analysis, Lappeenranta, Finland, June 1997.
- [2] S. Beucher, F. Meyer, The morphological approach to segmentation: the watershed transformation, in: E.R. Dougherty (Ed.), *Mathematical Morphology in Image Processing*, Marcel Dekker, New York, 1993, pp. 433–481.
- [3] S. Beucher, *Segmentation d'images et morphologie mathématique*, Ph.D. Thesis, School of Mines, Paris, June 1990.
- [4] S. Beucher, J.M. Blosseville, F. Lenoir, Traffic spatial measurements using video image processing, in: Proceedings of SPIE, *Advances in Intelligent Robotics Systems*, Cambridge Symposium on Optical and Optoelectronic Engineering, Cambridge, Massachusetts, November 1987.
- [5] B.P. Dobrin, T. Viero, M. Gabbouj, Fast watershed algorithms: analysis and extensions, in: Proc. IS and T/SPIE Symposium on Electronic Imaging: Science and Technology, San Jose, California, February 1994.
- [6] F. Friedlander, *Le traitement morphologique d'images de cardiologie nucléaire*, Ph.D. Thesis, School of Mines, Paris, December 1989.
- [7] V. Kumar, A. Grama, A. Gupta, G. Karypis, *Introduction to Parallel Computing – Design and Analysis of Algorithms*, Benjamin/Cummings, Menlo Park, CA, 1994.
- [8] F. Meyer, Topographic distance and watershed lines, *Signal Processing* 38 (1) (1994) 113–125.
- [9] F. Meyer, S. Beucher, Morphological segmentation, *Journal of Visual Communication and Image Representation* 1 (1) (1990) 21–46.
- [10] A. Meijster, J.B.T.M. Roerdink, Computation of watersheds based on parallel graph algorithms, in: P. Maragos, R.W. Schafer, M.A. Butt (Eds.), *Mathematical Morphology and its Applications to Image and Signal Processing*, Kluwer Academic Publishers, Dordrecht, pp. 305–312.
- [11] A.N. Moga, T. Viero, B.P. Dobrin, M. Gabbouj, Implementation of a distributed watershed algorithm, in: *Mathematical Morphology and Its Applications to Image Processing*, Kluwer Academic Publishers, Dordrecht, 1994, pp. 281–288.
- [12] A.N. Moga, T. Viero, M. Gabbouj, M. Nölle, G. Schreiber, H. Burkhardt, Parallel watershed algorithm based on sequential scannings, in: Proc. IEEE Workshop on Nonlinear Signal and Image Processing, vol. II, Halkidiki, June 1995, pp. 991–994.
- [13] A. Moga, M. Gabbouj, A parallel watershed algorithm based on the shortest paths computation, in: *Parallel Programming and Applications*, IOS Press, Amsterdam, May 1995, pp. 316–324.
- [14] A.N. Moga, B. Cramariuc, M. Gabbouj, A parallel watershed algorithm based on rainfalling simulation, in: Proc. European Conference on Circuit Theory and Design, vol. 1, Istanbul, Turkey, August 1995, pp. 339–342.
- [15] A. Moga, B. Cramariuc, M. Gabbouj, An efficient watershed segmentation algorithm suitable for parallel implementation, in: Proc. IEEE International Conference on Image Processing, vol. II, Washington, DC, October 1995, pp. 101–104.
- [16] A. Moga, *Parallel watershed algorithms for image segmentation*, Ph.D. Thesis, Tampere University of Technology, February 1997.
- [17] A. Moga, M. Gabbouj, An optimal parallel watershed algorithm based on image integration and sequential scannings, in: Proc. SPIE Parallel and Distributed Methods for Image Processing, San Diego, California, 1997, pp. 104–115.
- [18] A. Moga, M. Gabbouj, Parallel image component labeling with watershed transformation, *IEEE Trans. Pattern Anal. Machine Intelligence* 19 (5) (1997) 441–450.

- [19] A. Moga, M. Gabbouj, Parallel marker-based image segmentation with watershed transformation, *J. Parallel Distributed Comput.* 25 (1) (1998) 27–45.
- [20] MPI: A Message Passing Interface Standard, Version 1.1, Message Passing Interface Forum, University of Tennessee, Knoxville, Tennessee, 12 June 1995.
- [21] L. Najman, M. Schmitt, Watershed of a continuous function, *Signal Processing* 38 (1) (1994) 99–112.
- [22] J.F. Rivest, *Analyse Automatique d’Images Geologiques: Application de la Morphologie Mathematique aux Images Diagraphiques*, Ph.D. Thesis, School of Mines, Paris, 1992.
- [23] L. Vincent, P. Soille, Watersheds in digital spaces: an efficient algorithm based on immersion simulations, *IEEE Trans. Pattern Anal. Machine Intelligence* 13 (6) (1991) 583–598.