

# AN EFFICIENT WATERSHED SEGMENTATION ALGORITHM SUITABLE FOR PARALLEL IMPLEMENTATION

*Alina Moga, Bogdan Cramariuc, and Moncef Gabbouj*

Signal Processing Laboratory, Tampere University of Technology  
P.O. Box 553, FIN-33101 Tampere, Finland  
E-mail: moga@cs.tut.fi

## ABSTRACT

An important aspect of designing a parallel algorithm is exploitation of the data locality for minimization of the communication overhead. Aiming at this goal, we propose here a reformulation of a global image operation called the watershed transformation. The method lies among various approaches for image segmentation and performs by labeling connected components. Both serial and parallel programming models are presented and evaluated when running on SUN<sup>R</sup> and DEC<sup>R</sup> Alpha AXP workstations, and a Cray T3D, respectively.

## 1. INTRODUCTION

The watershed transformation was widely and successfully applied in different domains (e.g., in biomedicine, industry, and, generally in computer vision applications) as a powerful segmentation tool. The idea behind it is to split the morphological gradient of an original image, seen as a topographic surface, into geodesic influence zones [1, 6]. There are several watershed algorithms for digital space in the literature [?], but this paper focuses on a particular one based on the computation of the steepest slope lines, and produces zero width watersheds. Unlike the other methods the herein introduced algorithm has a higher degree of locality such that it can generate faster parallel implementations.

## 2. DESCRIPTION OF THE METHOD

The watershed algorithm based on rain falling simulation [2, 4] performs segmentation by labeling connected areas within the gradient of an image. Regarding the morphological gradient of the original image as a topographic surface, the rule of assigning labels can be derived from physics: a particle in free fall on a topographic surface will move due to gravity downward to

the deepest neighboring location. On flat areas, the rule is overloaded, such that the motion of the particle is directed toward the nearest brim of a downward slope, or it stops if the particle has reached a regional minimum. The task performed by the present algorithm is to trace a path for each non-minimum point on the surface (origin) to a minimum (destination), and to mark all pixels along the path with the label of the minimum. This path is a steepest slope line in a lower-complete image. The latter is the transformed gradient image such that any non-minimum pixel has a lower neighboring one. The result is a partition of the image which has the following properties: regions are connected, they do not overlap, and the partition is complete.

Segmentation is done in two main stages. In the first stage connected plateaus of pixels are explored. Plateaus of minima are uniquely labeled while the distance to a lower border for pixels within plateaus of non-minima is computed. This lower distance is further used to transform the gradient image into a lower-complete one. In the second stage a uniform rain over the topological surface is simulated. From each pixel a droplet starts sliding on the steepest slope line towards a minimum from which the label is propagated backwards along the whole path.

## 3. ADVANTAGE OF THE PROPOSED METHOD OVER THE WATERSHED BY IMMERSION

An important advantage of this watershed algorithm is its suitability for parallel implementation. While immersion is a global method (water arising from many sources progressively floods all the surface and interactions between waters coming from different sources are taken into consideration), raining can be denoted as a local method because each droplet follows on its own way regardless of neighboring droplets. In a parallel implementation rain falling simulation uses less

communication between processors, compared to immersion which has a highly global nature [3, 5].

#### 4. PARALLEL IMPLEMENTATION

A Single Program Multiple Data (SPMD) approach of the previously described algorithm is provided next. The image is regularly decomposed in blocks and adjacent blocks are placed in neighboring processes to minimize the communication cost. Local computations in any location within the distribution subdomain access a regular surrounding stencil of pixels. In order to perform near the edges of each subdomain, the latter is enlarged with an area of one grid point. Thus, an additional amount of memory is required for each processor to store the overlapping area, but the processor utilization is maximized, and the nonlocal memory reference is minimized.

The tile-by-tile structure of the serial algorithm is embedded in the parallel model. Both operations – detection and labeling of the minima and rain falling simulation – have a global character. When parallelizing each of them, the divide-and-conquer method is used. Thus, for every stage of the parallel algorithm four modules are provided: the *initialization* of the variables and data structures, the *computation* function, which is the serial operator applied on a subdomain to produce a partial result, the *linkage* unit to combine intermediate results from neighboring processes, and the *termination* of the stage.

In each subdomain merging is performed via message passing by swapping all its boundaries with the corresponding boundaries from neighboring subdomains. There is a regular grid-based point to point communication pattern between adjacent processes. Thus, at each step only neighboring sub-results can be mixed with the local one. Consequently, computation and linkage must be repeated until stabilization. A master process controls the termination moment, which occurs when no change has arisen in all the slave processes after the last iteration.

##### 4.1. Detection and Labeling of Minima

As emphasized in the serial algorithm, surfaces of constant altitude are tracked in the first stage. The above methodology applies here if there are surfaces which leave and possibly re-enter a subdomain. However, for a fine division of a global image, it is very likely that processes share plateaus.

In the *initialization* phase, all the pixels in all subdomains are considered as non-minima and labeled likewise.

The *computation* module performs as follows. Each process uses a disjoint interval of labels to label minima. If no lower neighboring pixels surround the currently explored plateau, the latter is uniquely labeled. Otherwise, for each pixel within the plateau, the shortest path to a lower border of the plateau following only pixels within the surface is generated. The length of this path is called the lower distance. Since minima do not have lower neighbors, their lower distance is 0 by default.

After exchanging labels and lower distances of pixels in the edges, the *linkage* of intermediate labelings is performed. Three cases may occur when merging two parts of a plateau from two adjacent subdomains. First, if both were classified and labeled as minima, their labels are in different ranges of values. By joining them, the minimal label is propagated in the part labeled with the higher value. Second, if lower neighbors enclose only one part, it will be properly diagnosed as a non-minimum while the other part as a minimum. Corrections are made in the latter by resetting the label and computing the lower distances. Finally, if both parts were detected as non-minima, the lower distances must be computed according to the entire set of lower borders. Therefore, lower distances are recalculated.

Naturally, modifications in one subdomain may reflect in neighboring subdomains too. Hence, processes must keep communicating and updating. A flag is set/reset in each process if changes occurred or not since the last communication. A master process is notified about all the flags. Based on these, a global flag is computed by a logical OR. *Termination* corresponds to a global flag OFF.

##### 4.2. Rain falling simulation

The *initialization* function of the second stage is called the lower-complete transformation and its role is next motivated. In the simulated immersion, a non-minimum pixel gets label from a steepest neighbor, if any; otherwise, from its predecessor which has the smallest lower distance among its neighboring pixels. The lower-complete transformation derives a new adjacency relation between neighboring pixels in the image based on a single metric, the lower-complete tone. This is a linear combination of the graylevel and the lower distance such that the ordering relation in both metrics is preserved.

The task of the *computation* module is to label each non-minimum pixel by "walking downward" on a steepest slope line towards a minimum. When the minimum is reached its label is assigned to all the pixels upward along the line. The algorithm computes paths with many origins and many destinations. Thus, in the lifetime of the parallel algorithm there are several active

paths in each process, starting from different origins. A steepest slope line is either solved by a single process, if the whole path belongs to the workspace of the same process, or remains unresolved if the path extends over several subimages.

Paths are first locally traced in each process and resolved if possible. Then, in the *linkage* step, unresolved subpaths are pasted with resolved ones from adjacent subdomains. Neighboring processes exchange labels of pixels within the edges to allow paths to extend in adjacent subdomains, if needed. Based on this additional data a new trial of labeling unresolved paths is initiated. A process keeps labeling, communicating, and completing paths while there are unresolved paths. When all the processes become inactive, all the pixels have been labeled and the algorithm stops.

Unlike the previous stage, *termination* is locally detected in each process. This reduces the communication traffic and the idle time in the processes, waiting for the global decision from the master. Another positive aspect is that no relabeling and no synchronizations between processes are needed. To each non-minimum pixel a single downward pixel from which it gets the label has been chosen, such that there is no race in labeling a pixel.

## 5. PERFORMANCE EVALUATION

Running times for the serial implementation of the algorithm measured on a SUN<sup>R</sup> workstation with 60 MHz CPU clock are tabulated next.

| Image<br>( <i>lin</i> × <i>col</i> ) | Trsh. | # regs | 1st scan | 2nd scan | Total time |
|--------------------------------------|-------|--------|----------|----------|------------|
| cermet<br>(256 × 256)                | 150   | 261    | 1.21     | 0.28     | 2.08       |
| body-section<br>(430 × 330)          | 30    | 766    | 3.48     | 0.81     | 5.48       |

The parallel watershed algorithm has been implemented and tested on a network of five DEC<sup>R</sup> Alpha AXP workstations. The software used is called PVM (Parallel Virtual Machine). The average running times for two images are shown below for a 2 × 2 division and number of machines used *N*.

| Image<br>( <i>lin</i> × <i>col</i> ) | Trsh. | # regs | <i>N</i> |      |      |      |      |
|--------------------------------------|-------|--------|----------|------|------|------|------|
|                                      |       |        | 1        | 2    | 3    | 4    | 5    |
| cermet<br>(256 × 256)                | 150   | 261    | 1.47     | 1.20 | 1.18 | 1.02 | 0.95 |
| body-section<br>(430 × 330)          | 30    | 766    | 1.63     | 1.50 | 1.41 | 1.25 | 1.10 |

A more realistic evaluation is obtained when the

same parallel algorithm is built on top of MPI (Message Passing Interface) and run on a Cray T3D computer. In the next table *N* is the logarithmic number of processors used. Results are shown starting from the minimal number *N* that the algorithm can handle. However, these are just empirical results drawing the scalable behavior of the parallel algorithm.

| Image<br>( <i>lin</i> × <i>col</i> ) | <i>N</i> |      |      |      |      |      |      |      |
|--------------------------------------|----------|------|------|------|------|------|------|------|
|                                      | 0        | 1    | 2    | 3    | 4    | 5    | 6    | 7    |
| Lenna<br>(512 × 512)                 | 6.12     | 3.35 | 1.93 | 1.10 | 0.90 | 0.54 | 0.54 | 0.34 |
| People<br>(1K × 1K)                  |          | 12.6 | 8.74 | 4.75 | 5.03 | 2.71 | 2.41 | 1.42 |
| Peppers<br>(2K × 2K)                 |          |      |      | 12.1 | 6.10 | 3.07 | 1.91 | 1.06 |

## 6. RESULTS

The images for which performance was presented above are illustrated here before and after applying the watershed transformation. In the output image, we show only the boundaries of the detected and labeled areas.

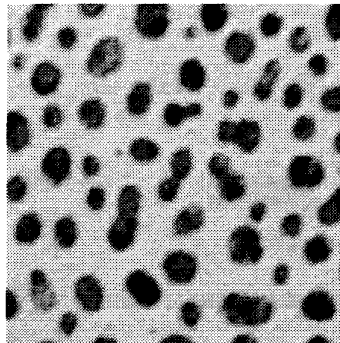


Figure 1: *Cermet* Input image

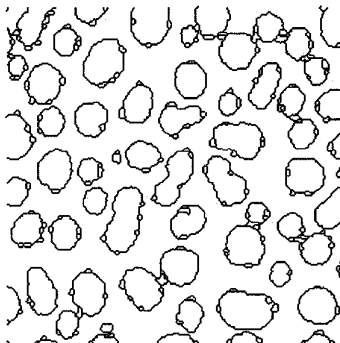


Figure 2: *Cermet* Output image

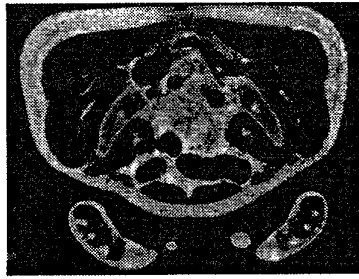


Figure 3: *Body-section* Input image

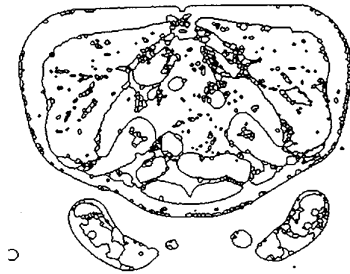


Figure 4: *Body-section* Output image

Because of lack of space only one of the images used for testing the parallel watershed implemented on the Cray T3D is shown below in the input and output form.



Figure 5: *People* Input image

## 7. ACKNOWLEDGMENT

We would like to thank to Prof. Irek Defee from Digital Media Institute in Tampere, Finland for providing us some test images which are part of the National Library of Medicine's Visible Human Program.

This work was partially supported by Edinburgh Parallel Computing Centre from University of Edin-

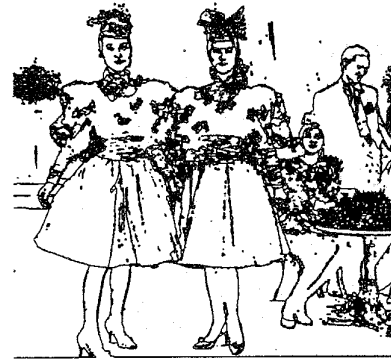


Figure 6: *People* Output image

burgh which allowed and assisted us to implement the application on a Cray T3D parallel computer using a library built on top of MPI called PUL-RD (Parallel Utility Library-Regular Decomposition).

## 8. REFERENCES

- [1] S. Beucher and F. Meyer, "The morphological approach to segmentation: The watershed transformation," in *Mathematical Morphology in Image Processing*, E. R. Dougherty, Editor, pp. 433-481, Marcel Dekker Inc., New York, 1993.
- [2] B. Cramariuc, A. Moga, and M. Gabbouj, "Image Segmentation by Component Labelling," accepted to *International Conference on Digital Signal Processing*, Limassol, Cyprus, June 1995.
- [3] A.N. Moga, T. Viero, B.P. Dobrin, and M. Gabbouj, "Implementation of a Distributed Watershed Algorithm," in *ISMM'94 Mathematical Morphology and Its Applications to Image Processing*, pp. 281-288, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.
- [4] A. N. Moga and M. Gabbouj, "A Parallel Watershed Algorithm Based on the Shortest Paths Computation," in *4th NTUG'95 & ZEUS'95 Parallel Programming and Applications*, P. Fritzon and L. Finmo, Editors, pp. 316-324, IOS Press, Linköping, Sweden, May 1995.
- [5] A. N. Moga, B. Cramariuc, and M. Gabbouj, "A Parallel Watershed Algorithm Based on Raining Simulation," accepted to *IEEE European Conference on Circuit Theory and Design*, Istanbul, Turkey, August 1995.
- [6] L. Vincent and P. Soille, "Watersheds in digital spaces: An efficient algorithm based on immersion simulations," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 13, no. 6, pp. 583-598, June 1991.