

Parallel Image Component Labeling With Watershed Transformation

Alina N. Moga and Moncef Gabbouj, *Senior Member, IEEE*

Abstract—The parallel watershed transformation used in gray scale image segmentation is reconsidered in this paper on the basis of the component labeling problem. The main idea is to break the sequentiality of the watershed transformation and to correctly delimit the extent of all connected components locally, on each processor, simultaneously. The internal fragmentation of the catchment basins, due to domain decomposition, into smaller subcomponents is ulteriorly solved by employing a global connected components operator. Therefore, in a pyramidal structure of master-slave processors, internal contours of adjacent subcomponents within the same component are hierarchically removed. Global final connected areas are efficiently obtained in $\log_2 N$ steps on a logical grid of N processors. Timings and segmentation results of the algorithm built on top of the *Message Passing Interface* (MPI) and tested on the Cray T3D are brought forward to justify the superiority of the novel design solution compared against previous implementations.

Index Terms—Watersheds, connected components, image segmentation, parallel computing, efficient algorithms.

1 INTRODUCTION

THE watershed transformation [7], [13], [21], [22], [23], [33], [34] has been broadly studied in the frame of gray scale image segmentation. The method performs by labeling connected components—catchment basins—within an image. Watershed transformation has been used in several industrial, biomedical, and computer vision applications (see [5], [6], [17], [23], [31]) where large images or sets of data are not uncommon. Although several trials have been previously made to parallelize watersheds (see [25], [26], [27], [28], [29]), the task is far from being easy since the operation relies on the history of region growth. Various serial methodologies for computing catchment basins with zero-width watershed lines [7], [13], [21], [22], [23], [33] have been employed for the purpose of parallelization on MIMD computers (see [24]). In each parallel approach, the image is distributed to a virtual grid of processors and partial results are produced by any of the watershed labeling techniques referred above. Considering the recursive nature of the watershed transformation, some parallel implementations choose to ignore the missing nonlocal data, produce incorrect subresults, and correct them based on the subresults from neighboring processors. Other implementations exploit only the local data on which they can properly operate and resume the computation as soon as the needed remote information becomes available. Both paradigms rely however on migrating data only between nearest neighboring processors. Since the correct result in one processor may depend on data encapsulated by nonadjacent processors, several iterations of computation and communication are

performed until stabilization. Hence, data dependent parallel algorithms have emerged with limited control upon performance. However, the proposed algorithm reinforces the research focused on parallel watershed algorithms and improves upon both running time and scalability.

Although several software and hardware algorithms, experimental studies, and surveys on parallelization of various techniques for region growing and labeling of connected components in both binary and gray scale images can be found in the literature [1], [2], [3], [4], [8], [9], [11], [12], [14], [18], [19], [32], [35], the proposed new parallel watershed algorithm is developed by regarding the watershed transformation as both a labeling of connected components—catchment basins—method and a region growing technique.

The paper is organized as follows. Section 2 describes the serial watershed algorithm used as a basis for the parallel implementation proposed in Section 3. A complexity profile is sketched in Section 4. Finally, conclusions on the margin of timings collected for different images on the Cray T3D computer and comparisons with previous parallel implementations of watersheds are drawn in Section 5.

2 THE SEQUENTIAL WATERSHED ALGORITHM

The original serial watershed transformation for gray scale images (see [7], [13], [21], [23], [33]) is a sequential operation. The algorithm starts by detecting and labeling initial seeds, e.g., minima of the gradient, which characterize regions of interest. The latter are defined as connected plateaus of pixels in the gradient image which do not have neighboring pixels of lower gray level (plateaus of minima). Starting from minima, ordered region growing is then performed. Thus, nonlabeled pixels are assimilated into different components in an increasing order of gray levels. Inside flat areas which are not yet labeled, called plateaus of non-minima, components progress synchronously, such that

• The authors are with the Signal Processing Laboratory, Tampere University of Technology, P.O. Box 553, FIN-33101 Tampere, Finland.
E-mail: moncef@cs.tut.fi.

Manuscript received 24 May 1996; revised 18 Feb. 1997. Recommended for acceptance by R. Szeliski.

For information on obtaining reprints of this article, please send e-mail to: transpami@computer.org, and reference IEEECS Log Number 104670.

they incorporate equal extents within the plateau. Consequently, a pixel on such a plateau is labeled along the shortest path, completely included in the plateau, to a lower downward brim of the plateau.

The recursive label propagation (flooding) is performed using an ordered queue which is an array of H FIFO queues, one queue for each of the H gray levels in the image. The ordered queue is initialized with pixels within plateaus of minima, detected in the first stage, which have nonlabeled neighboring pixels. A pixel of gray level h is inserted in the h th FIFO queue. The ordered queue is then parsed from the lowest gray level queue to the highest one. Pixels removed from the current FIFO queue, one at a time, assign their label to nonlabeled neighboring pixels of higher altitude and which are in turn inserted in the proper FIFO queue of the ordered queue. The FIFO order of serving candidate pixels within the same connected plateau ensures the synchronous breadth-first propagation of labels coming from different minima inside a plateau. When all FIFO queues have been emptied, each pixel in the image was appended to a single connected component, and the region growing procedure stops. The set of nonoverlapping connected components is a complete partition of the image and their labels depict the output image. Notice that unlike the watershed algorithm in [34], the above flooding scheme does not construct watershed lines (zero-width watershed lines). Although several drawbacks of the algorithm have been pointed out in [13], [33], e.g., scanning order dependency, our aim is only to parallelize the serial algorithm and not to eliminate its shortcomings.

Flooding in a simple input image and its corresponding output labels are illustrated in Fig. 1. Bold pixels in both images represent the seeds for region growing (regional minima).

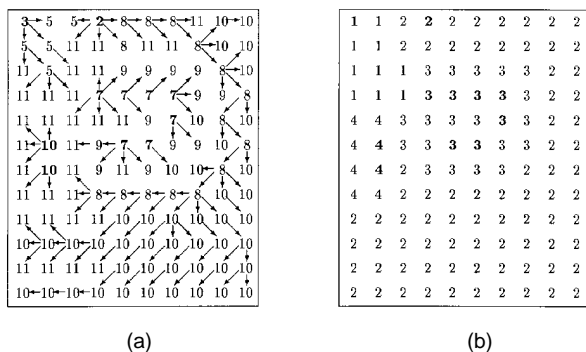


Fig. 1. Sequential watershed: (a) Flooding the input image. (b) Output image of labels.

3 THE PARALLEL WATERSHED ALGORITHM

From the brief description in the previous section one can observe that watershed transformation performs recursively and the overall data access pattern is global. Global operations are generally tackled in MIMD implementations in a *divide-and-conquer* fashion. As in [2], [14], [19], the image domain is divided into subdomains (blocks or slices) and intermediate labeling is performed within each subdomain. The boundary connectivity information between

subimages is locally stored in a graph [2] or in an equivalence table [14], [19]. More specifically, labels on each side of a common boundary between adjacent subimages and pertaining to the same component are retained as belonging to the same equivalence class. Final labels correspond to the connected components of the global graph obtained by combining all the local graphs. However, since the connected components operator is associative and without dependencies, that is, there is no order in which local results are computed, a reduction operation with a varying count of data from each processor and using the connected components operator solves the task efficiently [15], [16]. Thus, if N is the number of processors, the global labels will be available at the root processor in $\log_2 N$ steps, regardless of the data complexity.

The most desirable solution for a scalable parallel algorithm is based on reduction operations. Therefore, our efforts are further concentrated on making this parallelization strategy applicable to watersheds. The recursive nature of the serial algorithm is broken as in other parallel approaches of the watershed transformation based on the property that flooding is ruled by a two-dimensional ordering relation. Thus, a pixel p gets a label from an immediate neighbor q of lowest gray level, if any. On nonminima plateaus, the lower distance given by the length of the shortest path, completely included in the plateau, to a lower border of the plateau (see also [23]) imposes the propagation order. Accordingly, a pixel p within a plateau having the lower distance d' , $d' > 1$, receives its label from a neighboring pixel q with the lower distance $d' - 1$.

The flooding relation can be formulated in mathematical terms as follows. $p > q$ (to be read p is flooded from q) if

$$f(q) = \min_{r \in N_C(p)} \{f(r) \mid f(r) < f(p)\} \quad (1)$$

otherwise, $p > q$ if

$$f(p) = f(q) \text{ and } d(p) = d(q) + 1 \quad (2)$$

$f(\cdot)$ stands for gray level, $d(\cdot)$ for lower distance, and $N_C(\cdot)$ for the surrounding pattern of neighboring pixels.

Let us remark that the above relations render the dual ordering of the serial flooding as designed in [7], [21], [23]. The validity of the two rules can be checked based on the gray level image in Fig. 1a, the lower distance image in Fig. 2, and the output image in Fig. 1b. Note that in the lower distance image in Fig. 2 values are used for non-minima pixels which do not have lower neighboring pixels (except the marginal pixels of the plateaus they lie on and which have the lower distance 1). In Fig. 2, M stands for a maximum value MAX_DIST ($= \infty$) used at the initialization of the lower distance image.

Let us use the notation (*line, column, value*) to describe any pixel, where (*line, column*) is the pixel location (lines are indexed from 1 (topmost) downward and columns from 1 (leftmost) to the right) and *value* the gray level, e.g., (0 ... 255). Then, pixel (5, 3, 11) in the input image takes label three according to (1) from its northeastern neighbor (4, 4, 7), while pixel (3, 9, 8) of lower distance five receives label two conforming to (2) from its northwestern neighbor (2, 8, 8) with lower distance four.

M	1	1	M	1	2	3	M	1	2
1	1	M	M	1	M	M	4	1	1
M	2	M	M	M	M	M	M	5	1
M	M	M	M	M	M	M	M	M	6
M	M	M	M	M	M	M	1	7	1
M	M	M	M	M	M	M	M	1	8
M	M	M	M	M	M	1	1	9	1
M	M	M	14	13	12	11	10	1	1
M	M	M	M	1	1	1	1	1	2
5	4	3	2	2	2	2	2	2	2
M	M	M	M	3	3	3	3	3	3
7	6	5	4	4	4	4	4	4	4

Fig. 2. Lower distance image.

Consequently, the global propagation of labels would enter a subimage through pixels p , within the subdomain edges, which have a predecessor q , according to the flooding relation, only in a neighboring subdomain. If each such pixel p is assigned a unique label and inserted in the ordered queue as when it would have been already reached by the agglomeration process, flooding performs as in the global transformation, but more labels are manipulated. The difference appears in the components crossing the boundaries of a subdomain, which have different labels in different processors. However, as emphasized above, consistent labels are computed by the reduction connected components operation. Note that flooding of shared plateaus of nonminima differs from the serial case. Thus, in order to compensate for the early insertion in the queue of edge pixels with nonlocal predecessors, the time stamp given by the lower distance is used, such that every pixel is labeled by its predecessor, according to the global precedence relation in (2). This relation is however satisfied in the serial algorithm by the synchronous progression of the streams originating in different sources.

The novelty of the approach comes from two sources. The first is how local flooding in a processor can be performed to work independently from the rest of the processors, which increases also the data locality; and the second is how to combine the partial flooding results by a global reduction operation to considerably diminish the data dependency. In addition, the algorithm has a reduced software engineering cost. Indeed, it reuses the serial procedure of flooding based on the ordered queue and the sequential module for computing connected components.

As discussed above, the key problem is to explore each pixel p on the edges of a subdomain and determine its predecessor q according to the flooding relation. In order to check the local neighborhood for such marginal pixels, each subdomain is enlarged with a haul of one grid point—one line/column/point from the neighboring subdomain, if any. Otherwise, artificial pixels with arbitrarily chosen gray levels are introduced. Note that in the beginning, all pixels in the output subimages are labeled NARM (Not A Regional Minimum) and OUTBOARD in the haul. Thus, if a marginal pixel p has lower neighbors, checking whether it has a neighbor of the lowest gray level in the same subdomain is an easy task. If all these steepest neighbors are in adjacent subdomains, one of them, say q , is chosen as predecessor of p . The next available label is introduced for p , and the pair (p, q) is stored as being “the same” in the

local boundary connectivity graph. Note that each processor assigns labels starting from zero.

		0					1				
0		3	5	5	2	8	8	8	11	10	10
		5	5	11	11	8	11	11	8	10	10
		11	5	11	11	9	9	9	9	8	10
		11	11	11	7	7	7	7	9	9	8
		11	11	11	11	11	9	7	10	8	10
		11	10	11	9	7	7	9	9	10	8
1		11	10	11	9	11	9	10	10	8	10
		11	11	11	8	8	8	8	8	10	10
		11	11	11	11	10	10	10	10	10	10
		10	10	10	10	10	10	10	10	10	10
		11	11	11	11	10	10	10	10	10	10
		10	10	10	10	10	10	10	10	10	10

Fig. 3. Regular input image decomposition.

Let us consider a 2×2 division into equally sized blocks of the same input image as before, mapped onto a logical grid of processors with row major indexing (see Fig. 3). For simplicity pixels will still be addressed as $(line, column, value)$ where $(line, column)$ are the global coordinates and $value$ the gray level. The latter attribute will be omitted whenever the pixel location is the only significant information.

According to the above rule, pixel $(7, 4, 9)$ and $(7, 5, 11)$ in block $(1, 0)$ have as predecessor pixel $(6, 5, 7)$ in block $(0, 0)$. Therefore, $[(7, 4); (6, 5)]$ and $[(7, 5); (6, 5)]$ are stored and $(7, 4)$ and $(7, 5)$ get unique labels in block $(1, 0)$. Similarly, in block $(1, 1)$, pixel $(7, 6, 9)$ is preceded by $(6, 5, 7)$ in block $(0, 0)$ and $(7, 7, 10)$ by $(6, 6, 7)$ in block $(0, 1)$. Consequently, $[(7, 6); (6, 5)]$ and $[(7, 7); (6, 6)]$ are stored and $(7, 6)$ and $(7, 7)$ are individually labeled.

If p is a local minimum (all neighbors are of higher altitude than p) it is just uniquely labeled. However, if p does not have any lower neighbors and even more, it has neighbors of the same gray level in adjacent subdomains, then p is on a shared plateau. After exploring the part of the plateau in the current subdomain, the plateau is classified as minimum if no lower surrounding pixels have been found. Otherwise, it is nonminimum and the lower distances were computed only with respect to the lower brims of the plateau within the same subdomain. Three cases of shared plateaus are illustrated in Fig. 4. First, if the plateau has been detected as minimum in every subdomain where it

spreads (the plateau in the middle in Fig. 4 or the plateau of altitude seven in Fig. 3), it was all over correctly shaped (see Fig. 4 and Fig. 5a), though not uniquely labeled (Y, R, S, T, D, L, M, N in Fig. 4). A final unique label would emerge at the end of the linkage phase. Second, if the plateau has been detected in at least one processor as nonminimum, all the other parts should be marked alike, and the correct lower distances must be computed (see the snake-like shaded plateau on the left of Fig. 4 or the plateau of altitude eight in Fig. 3). In the latter example, only the part of the plateau in block (0, 0) in Fig. 5a was appropriately detected as nonminimum. Finally, in the third case, the plateau was classified by all involved processors as nonminimum (the shaded elliptical plateau at the bottom of Fig. 4), but the lower distances were not computed according to the whole distributed set of lower surrounding pixels.

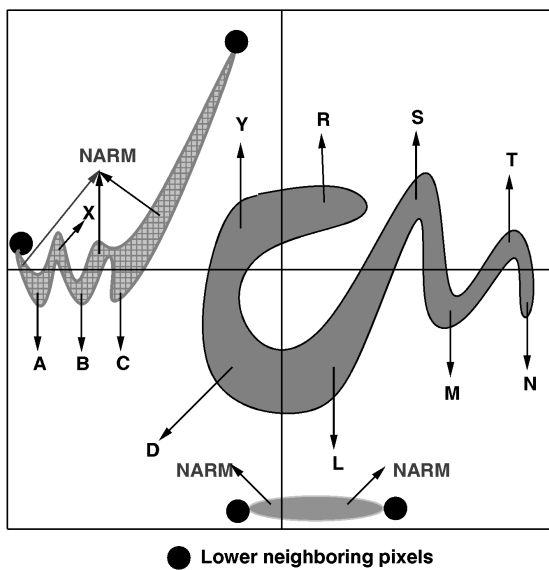


Fig. 4. Three cases of shared plateaus.

In conclusion, computation of the lower distances for shared nonminima plateaus must be solved. These values are usually generated in a breadth-first scanning of the plateau. Starting from all surrounding lower pixels, waves drift iteratively inside the plateau. Pixels swept by a wave will set the lower distance at the time stamp of the wave, which is one initially, and increases by one at each iteration. The wave propagation stops when the plateau has been exhaustively flooded. One may remark again the recursive nature of the operation. Unfortunately, there is no way to break down the sequentiality. When all parts of the shared nonminima plateau have been explored, merging lower distance values of pixels on the common border of two adjacent subdomains is not possible by applying simple local operators or other mathematical manipulations. A surface may leave and reenter a subdomain several times, such that the global connectivity is locally invisible. Only by nearest neighboring interprocess communication can the final global lower distances be computed.

A regular grid-based communication pattern is used to exchange the lower distances of pixels within edges with neighboring processors. Iteratively, lower distances in every subdomain are corrected based on values of non-minima pixels from adjacent subdomains. More specifically, pixels in the edges of a subdomain update their lower distance according to the following relation:

$$d(p) = \min_{r \in N_G^+(p)} \{d(p), d(r) + 1 \mid f(r) = f(p)\} \quad (3)$$

where N_G^+ stands for the nonlocal neighborhood replicated in the haul. In addition, if p turns into a nonminimum pixel due to the above correction ($d(p) < \text{MAX_DIST}$ and $\text{label}(p) \neq \text{NARM}$), its label is also set back to NARM. Every edge pixel which modified its lower distance propagates the change inside the subdomain by a breadth-first scanning. In a similar manner as above, $\forall q \in N_G(p)$, $f(q) = f(p)$, $\text{label}(q) \neq \text{OUTBOARD}$, and $d(q) > d(p) + 1$, $d(q) \leftarrow d(p) + 1$, and the wave of change propagation is extended with q . Furthermore, if $\text{label}(q) \neq \text{NARM}$, then $\text{label}(q) \leftarrow \text{NARM}$.

The loop of communication and updating terminates when no changes occur in all processors. To detect this moment, each processor sets a local flag to signal whether any modification has occurred in its subdomain since the last communication step. A global reduction OR is then performed over all processors' flag, such that the result of the operation is available to all processors. Therefore, termination would occur when the global flag is OFF (no changes). The number of iterations performed until a steady state is reached is data dependent, but for the images we tested this value did not exceed three. However, we shall see in Section 4 in which cases the number of iterations reaches extreme limits and what are these values.

The label subimages are illustrated in Fig. 5 before and after lower distance correction (N stands for NARM).

Labels of edge pixels which turned back to NARM after the lower distance computation are eliminated from the range of used labels. The remaining labels are realigned, such that only consecutive values are used for labeling. In our example, this operation is effective only in block (0, 1) where label one will be set to zero.

Further on, every edge pixel p which has its predecessor q , according to (2), in an adjacent subdomain is labeled and the pair (p, q) is stored in the local boundary connectivity graph. This is the case of pixels (1, 6) in block (0, 1), (7, 9) in block (1, 1), and (8, 5) in block (1, 0) (see Figs. 2, 3, and 6a). [(1, 6); (1, 5)], [(7, 9); (6, 10)], [(8, 5); (8, 6)] are stored in the corresponding local boundary connectivity graphs.

So far, only pixels on the edges and on shared plateaus have been analyzed. Minima are further detected among pixels not yet visited. For internal nonminima plateaus, the lower distance is not computed since in such cases the ordered flooding ensures correct propagation of labels. Label values before and after detecting minima inside each block are shown in Fig. 6.

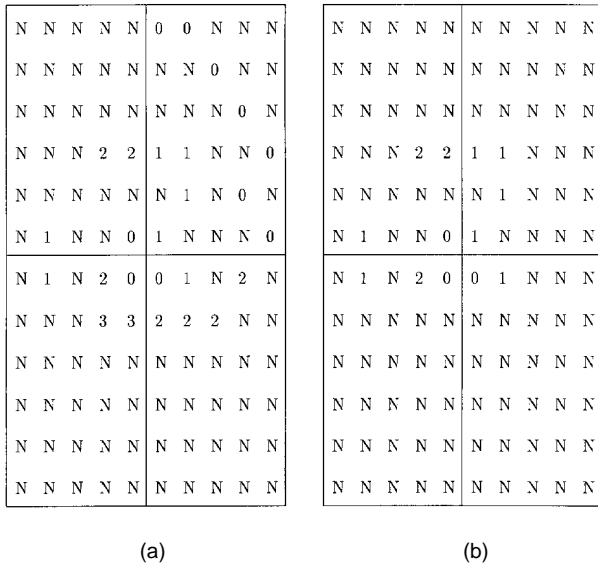


Fig. 5. Labels (a) before and (b) after distance correction.

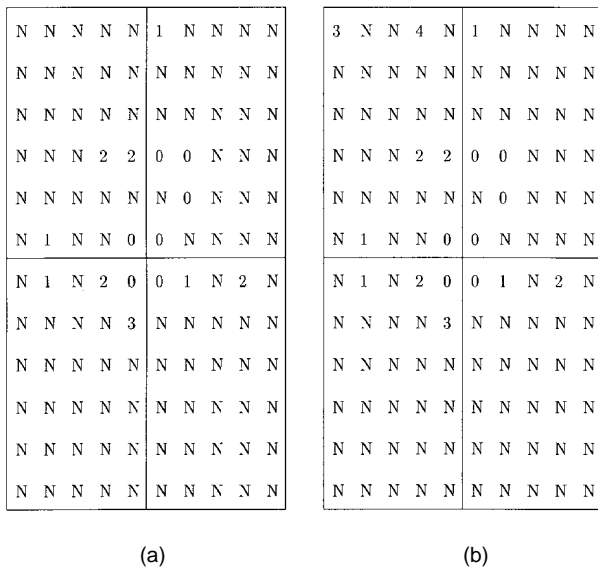


Fig. 6. Labels (a) before and (b) after detection of minima.

In order to have nonoverlapping ranges of labels in each processor, a local offset is globally computed based on the number L_i of labels assigned in every processor. That is, for processor P_0 , the local label offset is zero, and for processor P_s it is equal to the accumulated sum

$$\sum_{i=0}^{s-1} L_i, \forall s = 1, 2, \dots, N$$

Gather and scatter operations are used to compute the pre-fixed sums at the master processor. In our example, the master gathers the local number of used labels $LocalNoLabels = \{5, 2, 4, 3\}$ (count in Fig. 6b) and computes and scatters down the local offsets $LocalOffsets = \{0, 5, 7, 11\}$. Relabeling is not performed in the image, but whenever label values need to be transferred to other processors correction with the local label offset is performed.

Before flooding, the ordered queue is initialized with all labeled pixels having nonlabeled pixels in their neighborhood. Flooding is then effected as in the sequential algorithm. Exceptions are those shared plateaus of nonminima, where flooding is performed such that a pixel with the lower distance d' passes on its label only to those neighboring nonlabeled pixels with the lower distance $d' + 1$. Otherwise, due to the early insertion of extra seeds, the global precedence relation given in (2) would not be satisfied. In the nonminima plateaus completely included in one subdomain, this relation is implicitly true due to the breadth-first scanning originating in all surrounding lower pixels of the plateau. The result of the flooding is shown in Fig. 7a, with Fig. 7b illustrating the image of absolute labels (the relative labels increased with the local label offset).

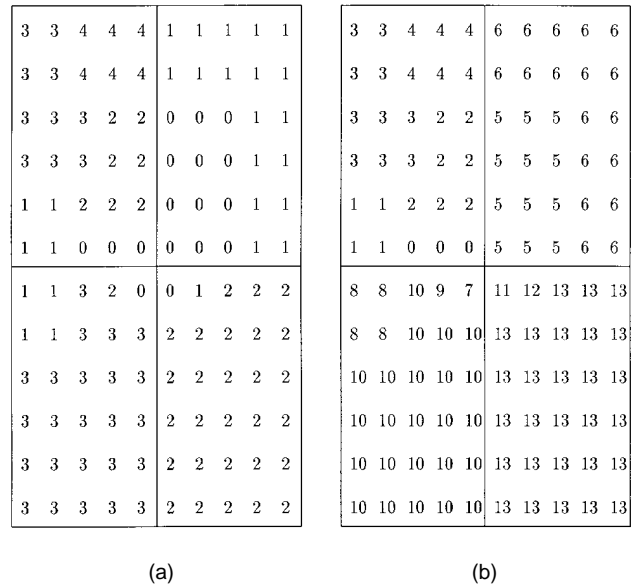


Fig. 7. (a) Relative and (b) absolute labels after flooding.

After all subimages have been labeled, neighboring processors exchange pixel labels within corresponding edges. The content of each local boundary connectivity graph is then modified such that each pixel is replaced by its absolute label. Moreover, labels of shared minima are also included in the graph.

Let G_i be the boundary connectivity graph of processor P_i . In our example, the pixel locations stored so far in each graph are:

- $G_0 = \emptyset$,
- $G_1 = \{(1, 6); (1, 5)\}$,
- $G_2 = \{(7, 4); (6, 5), [(7, 5); (6, 5)], [(8, 5); (8, 6)]\}$, and
- $G_3 = \{(7, 6); (6, 5), [(7, 7); (6, 6)], [(7, 9); (6, 10)]\}$.

Further on, by replacing in the graph pixel addresses with their labels and including also the connectivity information about shared minima (see Fig. 3 and Fig. 7b), one gets:

- $G_0 = \{[2, 5], [0, 5], [1, 8]\}$,
- $G_1 = \{[6, 4], [5, 2], [5, 0]\}$,

- $G_2 = \{[9, 0], [7, 0], [10, 13], [8, 1]\}$, and
- $G_3 = \{[11, 0], [12, 5], [13, 6]\}$.

Connected components are computed locally as in [10]. Let us consider the array of edges (u, v) , $u > v$ sorted in decreasing order of u . A forest of trees is grown from the initial set of trees in which every label is a one node tree. Let us introduce the array *GlobalParent* to store for each label its parent in the global forest. Initially, $GlobalParent[u] = u, \forall u$. If $Find(GlobalParent, u)$ is the operator which returns the parent of u with the path compression (nodes along the path are hooked directly on the root of the tree), and $Union(GlobalParent, u, v)$ merges the trees containing u and v , then the sequential algorithm for a graph G works as follows.

```

ConnectedComponents ( $G, GlobalParent$ )
  for each  $(u, v) \in G$ 
    if ( $Find(GlobalParent, u) \neq Find(GlobalParent, v)$ )
       $Union(GlobalParent, u, v)$ 

Find ( $GlobalParent, u$ )
  if ( $GlobalParent[u] \neq u$ )
     $GlobalParent[u] \leftarrow Find(GlobalParent, GlobalParent[u])$ ;
  return  $GlobalParent[u]$ ;

Union ( $GlobalParent, u, v$ )
   $Link(GlobalParent, Find(GlobalParent, u), Find(GlobalParent, v))$ 

Link ( $GlobalParent, u, v$ )
  if ( $u < v$ )  $GlobalParent[v] \leftarrow u$ ;
  else if ( $u > v$ )  $GlobalParent[u] \leftarrow v$ .

```

Thus, if $G = (V, E)$, then there are $|V|$ iterations to initialize the *GlobalParent* array, $2|E|$ *Find* operations, and at most $|V| - 1$ *Union* calls. It is shown in [10] that implementation with both path compression and union by rank can be viewed as linear in $(|V| + |E|)$. However, if each subdomain is $k \times k$ sized, then its boundary connectivity graph has at most $4k - 4$ vertices and $4k$ edges; and hence, the complexity is $O(k)$.

Local forests computed in the *GlobalParent* array are merged pairwise in $\log_2 N$ steps until the global forest results at the root processor (see [20, pp. 278–281]). When merging two forests, for example B into A , every edge $(u, v) \in B$ merges to A by $ConnectedComponents(B, GlobalParent_A)$. $GlobalParent_A$ is used to show that information only about forest A has been already stored in the *GlobalParent* array. Thus, if $L = \sum_{i=0}^{N-1} L_i$ is the total number of labels, the computational cost is $O(L \log_2 N)$. At each merging step, all edges in the forest are transmitted resulting in a communication complexity of $O(L \log_2 N)$.

After the connected components of the global boundary connectivity graph are available at the master processor (see all the steps in Fig. 8) simplifications are again performed such that consecutive final numbers label the regions in the entire image. Concretely, in our example, the global forest is

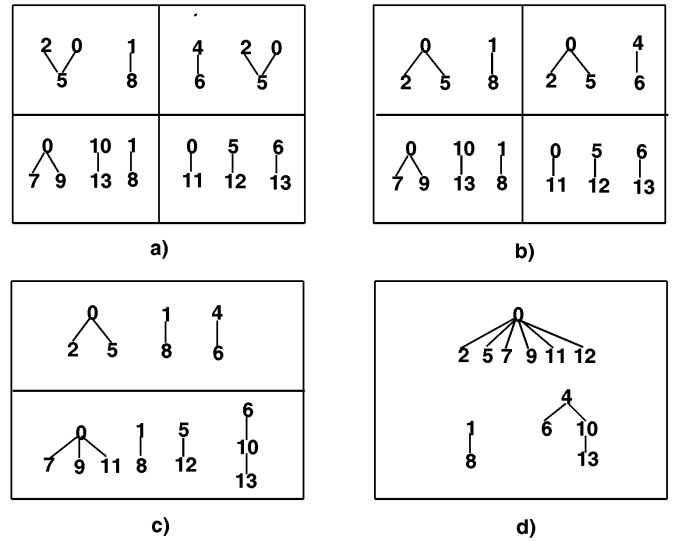


Fig. 8. (a) Local graphs. (b) Local forests; merging of forests. (c) Step 1. (d) Step 2.

$GlobalParent = \{0, 1, 0, 3, 4, 0, 4, 0, 1, 0, 4, 0, 0, 10\}$, or after label realignment (which also ensures path compression) $GlobalParent = \{0, 1, 0, 2, 3, 0, 3, 0, 1, 0, 3, 0, 0, 3\}$.

Ultimately, the master processor scatters down the corresponding range of labels to every processor based on the *LocalNoLabels* and *LocalOffset* arrays (processor P_i receives $LocalNoLabels[i]$ labels starting from $LocalOffset[i]$ in the *GlobalParent* array), and relabeling is performed in each subimage with the final global values.

At this moment, all connected components are correctly detected and labeled with consecutive natural numbers starting from zero. The label subimages before and after relabeling are shown in Fig. 9.

4 COMPLEXITY ANALYSIS

Let us consider a $n \times n$ image distributed to N processors. Each block size is $\frac{n}{\sqrt{N}} \times \frac{n}{\sqrt{N}}$. Pixels on the edges of a block are explored first and labeled if they are minima or if their predecessors, according to (1), are in neighboring processors. These are swept again to eliminate possible gaps in the range of used labels after the lower distance correction. Finally, edges are scanned once more to label pixels which are related to their predecessors, from adjacent processors, according to (2). Hence, $T_{comp}^1 = O\left(\frac{n}{\sqrt{N}}\right)$.

Each subimage is scanned twice, once for detecting and labeling the plateaus and a second time, during flooding. Thus, $T_{comp}^2 = O\left(\frac{n^2}{N}\right)$.

Shared plateaus of nonminima are swept $n_{iterations}$ times during the distance correction.

$$T_{comp}^3 = O\left(n_{iterations} \times \max_{0 \leq i < N} \{n_{nmp_i}\}\right)$$

where n_{nmp_i} is the total number of nonminima pixels within shared plateaus in processor P_i and

$$1 \leq n_{iterations} \leq n \times (\sqrt{N} - 1)$$

3 3 4 4 4	6 6 6 6 6	2 2 3 3 3	3 3 3 3 3
3 3 4 4 4	6 6 6 6 6	2 2 3 3 3	3 3 3 3 3
3 3 3 2 2	5 5 5 6 6	2 2 2 0 0	0 0 0 3 3
3 3 3 2 2	5 5 5 6 6	2 2 2 0 0	0 0 0 3 3
1 1 2 2 2	5 5 5 6 6	1 1 0 0 0	0 0 0 3 3
1 1 0 0 0	5 5 5 6 6	1 1 0 0 0	0 0 0 3 3
8 8 10 9 7	11 12 13 13 13	1 1 3 0 0	0 0 3 3 3
8 8 10 10 10	13 13 13 13 13	1 1 3 3 3	3 3 3 3 3
10 10 10 10 10	13 13 13 13 13	3 3 3 3 3	3 3 3 3 3
10 10 10 10 10	13 13 13 13 13	3 3 3 3 3	3 3 3 3 3
10 10 10 10 10	13 13 13 13 13	3 3 3 3 3	3 3 3 3 3
10 10 10 10 10	13 13 13 13 13	3 3 3 3 3	3 3 3 3 3

(a)
(b)

Fig. 9. (a) Local labels. (b) Global labels.

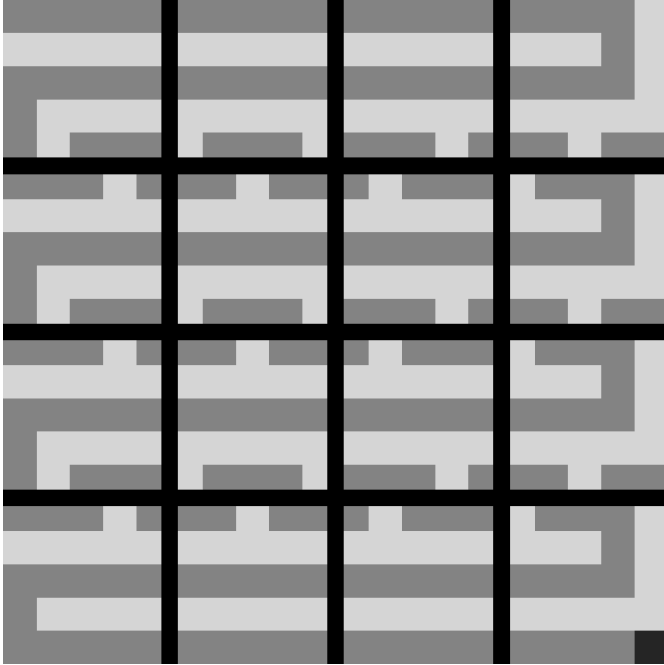


Fig. 10. Snake shape image.

Fig. 10 is an example where $n_{iterations}$ reaches the upper bound. In every iteration, edges are exchanged between neighboring processors, which entails a total communication cost

$$T_{comm}^1 = O\left(n_{iterations} \times \left(t_s + t_w \frac{n}{\sqrt{N}}\right)\right)$$

t_s stands for the startup time and t_w for the transfer time per word.

Computation of the local label offset by scatter and gather operations brings additional complexities.

$T_{comp}^4 = O(N)$ for computation of the prefixed sums and

$T_{comp}^2 = O(\log_2 N)$ for communication.

$T_{comp}^5 = O\left(\frac{n}{\sqrt{N}}\right)$ and $T_{comm}^3 = O\left(t_s + t_w \frac{n}{\sqrt{N}}\right)$ result from swapping the absolute labels between neighboring processors, while $T_{comp}^6 = O\left(\frac{n}{\sqrt{N}}\right)$ operations are needed for replacing each pixel in the boundary connectivity graph with its label.

Ultimately, the complexities of the connected components are

$$T_{comp}^7 = O(L \log_2 N) + O\left(\frac{n}{\sqrt{N}}\right)$$

and

$$T_{comm}^4 = O(\log_2 N \times (t_s + t_w L))$$

(L is the total number of labels used).

Thus, the total computational time is

$$T_{comp} = \sum_{i=1}^7 T_{comp}^i =$$

$$O\left(\frac{n}{\sqrt{N}}\right) + O\left(\frac{n^2}{N}\right) + O\left(n_{iterations} \times \max_{0 \leq i < N} \{n_{nmp_i}\}\right)$$

$$+ O(N) + O(L \log_2 N)$$

$$\leq O\left(\frac{n^2}{N} + \frac{n}{\sqrt{N}} + n(\sqrt{N} - 1) \times \max_{0 \leq i < N} \{n_{nmp_i}\}\right) + N + L \log_2 N$$

and the total communication time

$$T_{comm} = \sum_{i=1}^4 T_{comm}^i =$$

$$O\left(n_{iterations} \times \left(t_s + t_w \frac{n}{\sqrt{N}}\right)\right) + O(\log_2 N) + O\left(t_s + t_w \frac{n}{\sqrt{N}}\right)$$

$$+ O(\log_2 N \times (t_s + t_w L))$$

$$\leq O\left(n^2 \left(1 - \frac{1}{\sqrt{N}}\right) + n \left(\sqrt{N} + \frac{1}{\sqrt{N}} - 1\right) + (L+1) \log_2(N)\right)$$

5 TIMINGS AND CONCLUSIONS

The algorithm described above has been implemented under *Message Passing Interface* (MPI) on a parallel computer of type Cray T3D at Edinburgh Parallel Computing Centre. Four natural images shown below (Figs. 13-16) were tested on up to 128 processors and the afferent running times in seconds are tabulated in Table 1. Additional results for some artificial images can be found in [24]. Notice that the measured times do not include data loading, distribution, coalescence, and saving.

The relative speedup computed as $SP(N) = \frac{T(1)}{T(N)}$ is plotted in Fig. 11, in logarithmic scale, against the ideal linear speedup. One may note the sharp ascension of the speedup with increasing number of processors for all images. The speedup tends however to saturate around 64-128 processors and the curve drops then down. Two images of 512×512 were tested to show the loose data dependency of the algorithm. Only the distance correction strongly relies on the data complexity. However, the two speedup curves grow very close to each other. Finally, better performances are obtained with larger image sizes. Thus, the lowest speedup corresponds to the 256×256 image while the highest speedup to the $1,024 \times 1,024$ image.

As mentioned in the introduction, the proposed algo-

TABLE 1
PARALLEL SEGMENTATION EXECUTION TIME (IN SECONDS)

N	<i>Cermet</i> 256 × 256	<i>Lenna</i> 512 × 512	<i>Peppers</i> 512 × 512	<i>People</i> 1024 × 1024
	418 regs	5370 regs	2795 regs	14684 regs
1	0.669816	2.566100	2.593566	11.883200
2	0.370638	1.550141	1.626408	7.603905
4	0.197451	0.762665	0.834887	4.204138
8	0.114612	0.427945	0.434503	2.123846
16	0.069887	0.248869	0.259074	1.008608
32	0.045889	0.140915	0.139553	0.552418
64	0.036216	0.087831	0.075441	0.278635
128	0.041284	0.069217	0.061846	0.171757

TABLE 2
PARALLEL SEGMENTATION EXECUTION TIMES
FOR VARIOUS ALGORITHMS (IN SECONDS)

Algorithm	Shortest Paths		Rainfalling		Connected Components	
	N	<i>Lenna</i>	<i>Peppers</i>	<i>Lenna</i>	<i>Peppers</i>	<i>Lenna</i>
1	6.420161	8.491929	6.128608	6.105197	2.566100	2.593566
2	3.482199	4.324643	3.356720	3.073354	1.550141	1.626408
4	1.947980	2.555786	1.935435	1.910854	0.762665	0.834887
8	1.480164	1.458012	1.105660	1.069720	0.427945	0.434503
16	1.048395	1.020210	0.907735	0.860031	0.248869	0.259074
32	0.610450	0.582911	0.542063	0.521104	0.140915	0.139553
64	0.550981	0.492310	0.549082	0.462122	0.087831	0.075441
128	0.348101	0.323131	0.341372	0.300032	0.069217	0.061846

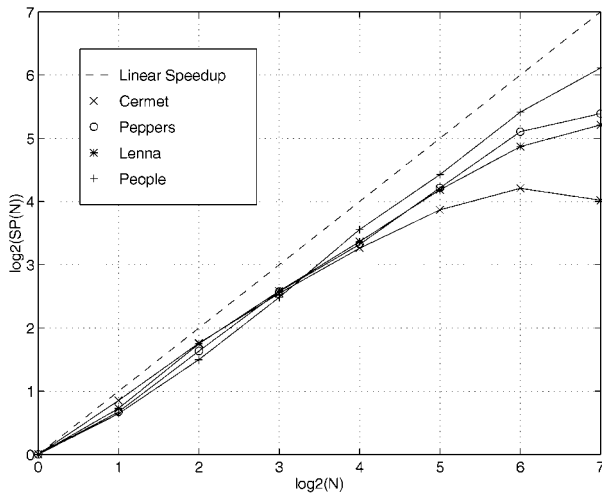


Fig. 11. Speedup versus number of processors.

algorithm outperforms the previous parallel implementations of the watershed transformation. For a brief comparison, additional running times are included in Table 2 for another two parallel watershed algorithms—based on the shortest paths computation [27] and the rainfalling simulation [25]—for image *Lenna* and *Peppers*. Both algorithms were implemented with *Parallel Utility Library-Regular Decomposition* (PUL-RD), which is a library built on top of MPI, and tested on the same Cray T3D computer as was the present algorithm.

Although the first two algorithms are data dependent, for these particular images it may be observed that the third algorithm based on the connected components runs much faster than the first two implementations. Moreover, let us compute the outperformance indicators $OP_{i3}(N) = T_i(N)/T_3(N)$, where $i = 1$ refers to the shortest paths based algorithm, $i = 2$ the rainfalling simulation, and $i = 3$ the present algorithm, to compare the last algorithm ($i = 3$) with the previous existing ones ($i = 1, 2$). OP_{13} and OP_{23} are illustrated for both images in Fig. 12. The outperformance curve is generally ascending, it starts from a value above two and reaches a sharp peak above six at 64 processors.

Another form of analyzing the outperformance is the ratio of the relative speedups:

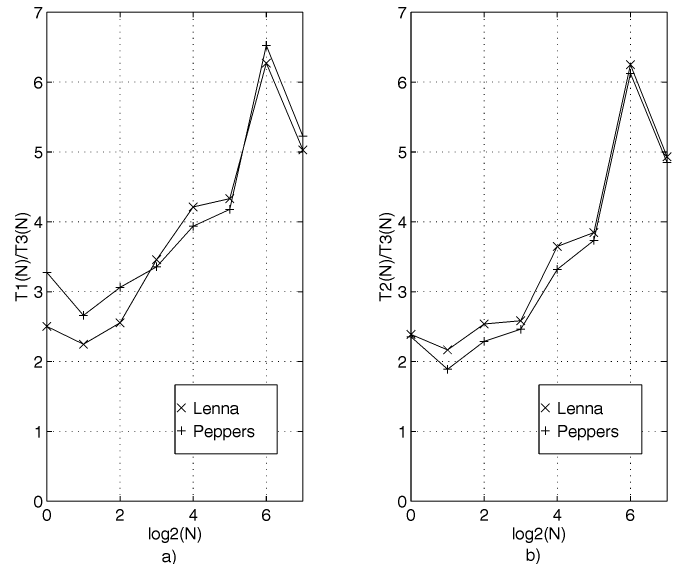


Fig. 12. Outperformance indicator. (a) Shortest paths—connected components. (b) Rainfalling—connected components.

$$SP_3(N) / SP_i(N) = \frac{T_3(1)}{T_3(N)} / \frac{T_i(1)}{T_i(N)} =$$

$$\frac{T_i(N)}{T_3(N)} / \frac{T_i(1)}{T_3(1)} = OP_{i3}(N) / OP_{i3}(1)$$

The ratio is greater than one, and it has a maximum of approximately three at 64 processors. This latter characteristic shows that the third algorithm scales better, up to 64 processors compared to 32 for the other two algorithms.

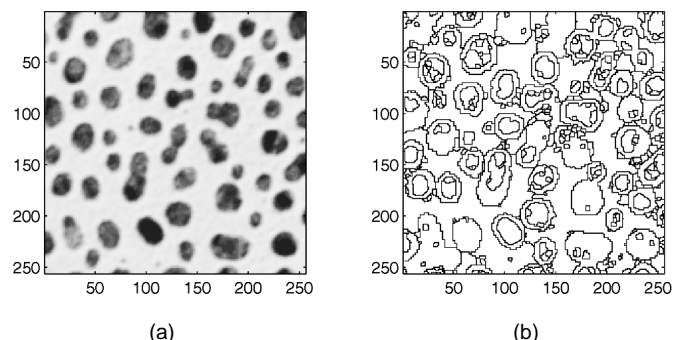


Fig. 13. Cermet. (a) Original image. (b) Segmented image (418 regions).

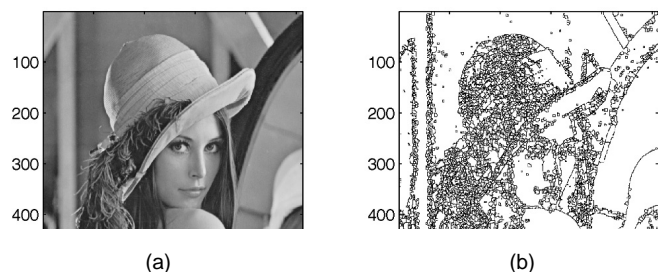


Fig. 14. Lenna. (a) Original image. (b) Segmented image (5,370 regions).

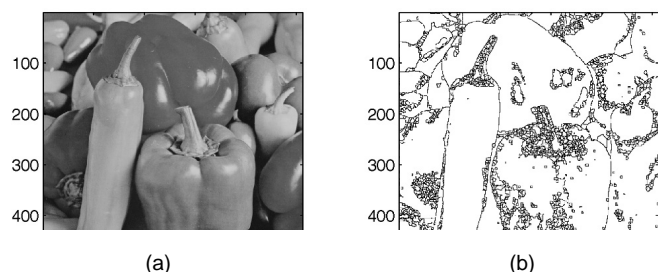


Fig. 15. Peppers. (a) Original image. (b) Segmented image (2,795 regions).

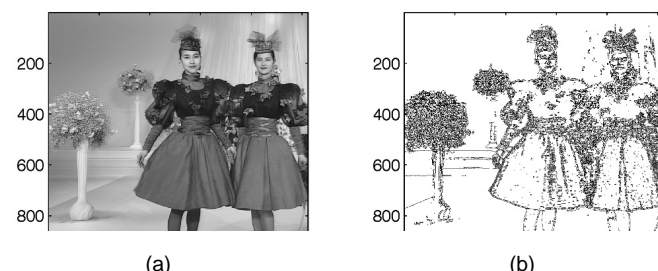


Fig. 16. People. (a) Original image. (b) Segmented image (14,684 regions).

In conclusion, the merit of the algorithm based on connected components is that it improves both running time and speedup and it is loosely data dependent. Moreover, the current approach permits an attractive and efficient extension of the parallel algorithm to perform with the aid of markers [28].

ACKNOWLEDGMENT

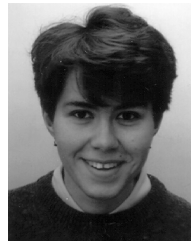
This research has been supported by the Graduate School in Electronics, Telecommunication and Automation (GETA) and Edinburgh Parallel Computing Centre in the Training and Research on Advanced Computing Systems (TRACS) programme.

REFERENCES

- [1] H.M. Alnuweiri and V.K. Prasanna Kumar, "Fast Image Labeling Using Local Operators on Mesh-Connected Computers," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 2, pp. 202–207, Feb. 1991.
- [2] H.M. Alnuweiri and V.K. Prasanna, "Parallel Architectures and Algorithms for Image Component Labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 10, pp. 1,014–1,034, Oct. 1992.
- [3] D.A. Bader and J. Jájá, "Parallel Algorithms for Image Histogramming and Connected Components With an Experimental Study," *J. Parallel and Distributed Computing*, vol. 35, no. 2, pp. 173–190, June 1996.
- [4] D.A. Bader, J. Jájá, D. Harwood, and L.S. Davis, "Parallel Algorithms for Image Enhancement and Segmentation by Region Growing With an Experimental Study," *J. Supercomputing*, vol. 10, no. 2, pp. 141–168, 1996.
- [5] S. Beucher, *Segmentation d'images et morphologie mathématique*. PhD thesis, School of Mines, Paris, June 1990.
- [6] S. Beucher, J. M. Blosseville, and F. Lenoir, "Traffic Spatial Measurements Using Video Image Processing," *Proc. SPIE, Advances in Intelligent Robotics Systems (Cambridge Symp. Optical and Optoelectronic Eng.)*, Cambridge, Mass., Nov. 1987.
- [7] S. Beucher and F. Meyer, "The Morphological Approach to Segmentation: The Watershed Transformation," E.R. Dougherty, ed., *Mathematical Morphology in Image Processing*. New York: Marcel Dekker Inc., 1993, pp. 433–481.
- [8] A. Choudhary and R. Thakur, "Connected Component Labeling on Coarse Grain Parallel Computers: An Experimental Study," *J. Parallel and Distributed Computing*, vol. 20, no. 1, pp. 78–83, 1994.
- [9] N. Coptý, S. Ranka, G. Fox, and R.V. Shankar, "A Data Parallel Algorithm for Solving the Region Growing Problem on the Connection Machine," *J. Parallel and Distributed Computing*, vol. 21, no. 1, pp. 160–168, 1994.
- [10] T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*. Cambridge, Mass.: MIT Press, 1990.
- [11] R.E. Cypher, J.L. Sanz, and L. Snyder, "An EREW PRAM Algorithm for Image Component Labeling," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 3, pp. 258–262, Mar. 1989.
- [12] R.E. Cypher, J.L. Sanz, and L. Snyder, "Algorithms for Image Component Labeling on SIMD Mesh-Connected Computers," *IEEE Transactions on Computers*, vol. 39, no. 2, pp. 276–281, Feb. 1990.
- [13] B.P. Dobrin, T. Viero, and M. Gabbouj, "Fast Watershed Algorithms: Analysis and Extensions," E.R. Dougherty, J. Astola, and H.G. Longbotham, eds., *Proc. SPIE Nonlinear Image Processing 5*, vol. 2,180, pp. 209–220, San Jose, Calif., Feb. 1994.
- [14] H. Embrechts, D. Roose, and P. Wambacq, "Component Labelling on a MIMD Multiprocessor," *CVGIP: Image Understanding*, vol. 57, no. 2, pp. 155–165, Mar. 1993.
- [15] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," technical report, Version 1.0, Univ. of Tennessee, Knoxville, May 1994.
- [16] Message Passing Interface Forum, "MPI: A Message-Passing Interface Standard," technical report, Version 1.1, Univ. of Tennessee, Knoxville, June 1995.
- [17] F. Friedlander, *Le traitement morphologique d'images de cardiologie nucléaire*. PhD thesis, School of Mines, Paris, Dec. 1989.
- [18] S. Hambrusch, X. He, and R. Miller, "Parallel Algorithms for Gray-Scale Digitized Picture Component Labeling on a Mesh-Connected Computer," *J. Parallel and Distributed Computing*, vol. 20, pp. 56–68, 1994.
- [19] T. Johansson, *Image Analysis Algorithms on General Purpose Parallel Architectures*, PhD thesis, Centre for Image Analysis, Univ. of Uppsala, Uppsala, Sweden, 1994.
- [20] V. Kumar, A. Grama, A. Gupta, and G. Karypis, *Introduction to Parallel Computing-Design and Analysis of Algorithms*. Benjamin/Cummings Publishing Company, Inc., 1994.
- [21] F. Meyer, "Un algorithme optimal de ligne de partage des eaux," *Proc. Eighth Congres Reconnaissance des Formes et Intelligence Artificielle*, pp. 847–857, Lyon, France, Nov. 1991.
- [22] F. Meyer, "Topographic Distance and Watershed Lines," *Signal Processing*, vol. 38, no. 1, pp. 113–125, July 1994.
- [23] F. Meyer and S. Beucher, "Morphological Segmentation," *J. Visual Communication and Image Representation*, vol. 1, no. 1, pp. 21–46, Sept. 1990.
- [24] A.N. Moga, *Parallel Watershed Algorithms for Image Segmentation*, PhD thesis, Tampere Univ. of Technology, Tampere, Finland, Feb. 1997.
- [25] A.N. Moga, B. Cramariuc, and M. Gabbouj, "An Efficient Watershed Segmentation Algorithm Suitable for Parallel Implementation," *Proc. IEEE Int'l Conf. Image Processing*, vol. 2, pp. 101–104, Washington, D.C., Oct. 1995. Los Alamitos, Calif.: IEEE CS Press, 1995.
- [26] A.N. Moga, B. Cramariuc, and M. Gabbouj, "A Parallel Watershed Algorithm Based on Rainfalling Simulation," *Proc. 12th*

European Conf. Circuit Theory and Design, vol. 1, pp. 339–342, Istanbul, Turkey, Aug. 1995.

- [27] A.N. Moga, B. Cramariuc, and M. Gabbouj, "A Parallel Watershed Algorithm Based on the Shortest Paths Computation," P. Fritzon and L. Finmo, eds., *Parallel Programming and Applications (Proc. Workshop Parallel Programming and Computation [ZEUS'95] and the Fourth Nordic Transputer Conf. [NTUG'95])*, pp. 316–324, Amsterdam, The Netherlands, May 1995. IOS Press Ohmsha.
- [28] A.N. Moga and M. Gabbouj, "A Parallel Marker Based Watershed Transformation," *Proc. IEEE Int'l Conf. Image Processing*, vol. 2, pp. 137–140, Lausanne, Switzerland, Sept. 1996. Los Alamitos, Calif.: IEEE CS Press, 1996.
- [29] A.N. Moga, T. Viero, B.P. Dobrin, and M. Gabbouj, "Implementation of a Distributed Watershed Algorithm," J. Serra and P. Soille, eds., *Computational Imaging and Vision—Mathematical Morphology and Its Applications to Image Processing (Proc. Int'l Symp. Mathematical Morphology, Fontainebleau, France)*, pp. 281–288, Dordrecht, The Netherlands, Sept. 1994. Kluwer Academic Publishers.
- [30] A.N. Moga, T. Viero, M. Gabbouj, M. Nölle, G. Schreiber, and H. Burkhardt, "Parallel Watershed Algorithm Based on Sequential Scannings," I. Pitas, ed., *Proc. 1995 IEEE Workshop on Nonlinear Signal and Image Processing*, vol. 2, pp. 991–994, Neos Marmaras, Halkidiki, Greece, June 1995.
- [31] J.F. Rivest, *Analyse Automatique d'Images Geologiques: Application de la Morphologie Mathematique aux Images Diagraphiques*, PhD thesis, School of Mines, Paris, 1992.
- [32] H. Samet and M. Tamminen, "Efficient Component Labeling of Images of Arbitrary Dimension Represented by Linear Bintreees," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 4, pp. 579–590, July 1988.
- [33] T. Viero, *Algorithms for Image Sequence Filtering, Coding and Image Segmentation*, PhD thesis, Tampere Univ. of Technology, Tampere, Finland, Jan. 1996.
- [34] L. Vincent and P. Soille, "Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 13, no. 6, pp. 583–598, June 1991.
- [35] M. Willebeek-LeMair and A.P. Reeves, "Solving Nonuniform Problems on SIMD Computers: Case Study on Region Growing," *J. Parallel and Distributed Computing*, vol. 8, pp. 135–149, 1990.



Alina N. Moga received the MSc degree in computer science from "Politehnica" University of Bucharest, Bucharest, Romania, in 1993 and the PhD degree in parallel image segmentation algorithms at Signal Processing Laboratory, Department of Information Technology, Tampere University of Technology, Tampere, Finland, in 1997. Dr. Moga is currently a research assistant with Albert-Ludwigs-Universitaet, Institut fuer Informatik, Freiburg, Germany. Her main research interests include parallel computing, efficient algorithms, and image segmentation.



Moncef Gabbouj (S'85, M'90, SM'95) received the BS degree in electrical engineering in 1985 from Oklahoma State University, Stillwater. He received the MS and PhD degrees in electrical engineering from Purdue University, West Lafayette, Indiana, in 1986 and 1989, respectively. Dr. Gabbouj is currently a professor with the Department of Information Technology-Pori, at Tampere University of Technology, Tampere, Finland. From 1994 to 1995, he was an associate professor with the Signal Processing Laboratory of Tampere University of Technology. From 1990 to 1993, he held the position of senior research scientist with the Research Institute for Information Technology, Tampere, Finland. His research interests include nonlinear signal and image processing and analysis, mathematical morphology, and neural networks.

Dr. Gabbouj is Associate Editor of the *IEEE Transactions on Image Processing* and served as Guest Editor of the European journal *Signal Processing* for its Special Issue on Nonlinear Digital Signal Processing (August 1994). He is a member and past Chairman, Chairman-Elect, and Secretary of the IEEE Circuits and Systems Society, Technical Committee on Digital Signal Processing, and past Chairman of the IEEE SP/CAS Chapter of the IEEE Finland Section. He was also the DSP Track Chair of the 1996 IEEE ISCAS and the Program Chair of NORSIG'96.

Dr. Gabbouj is the director of the International University Program in Digital Signal Processing in the Signal Processing Laboratory at Tampere University of Technology and Secretary of the Advisory Committee of Tampere International Center for Signal Processing. He is a member of Eta Kappa Nu, Phi Kappa Phi, IEEE SP, and CAS societies. Dr. Gabbouj is the Technical Program Chair of EUSIPCO 2000.

Dr. Gabbouj was corecipient of the Myril B. Reed Best Paper Award from the 32nd Midwest Symposium on Circuits and Systems. He was also corecipient of the NORSIG 94 Best Paper Award from the 1994 Nordic Signal Processing Symposium.