An Alternative Algorithm for Computing Watersheds on Shared Memory Parallel Computers

A. Meijster and J.B.T.M. Roerdink
University of Groningen,
Institute for Mathematics and Computing Science
P.O. Box 800, 9700 AV Groningen, The Netherlands
Email: arnold@cs.rug.nl roe@cs.rug.nl
Tel. +31-50-633931, Fax. +31-50-633800

Abstract

In this paper a parallel implementation of a watershed algorithm is proposed. The algorithm can easily be implemented on shared memory parallel computers. The watershed transform is generally considered to be inherently sequential since the discrete watershed of an image is defined using recursion, see [5]. However, recently a few research groups, see [2, 3, 4], have designed parallel algorithms for computing watersheds. Most of these parallel algorithms are based on splitting the source image in blocks, computing the watersheds of these blocks and merging the resulting images into the desired result. A disadvantage of this approach is that a lot of communication is necessary at the boundaries of the blocks. In this paper we show that it is possible to transform the computation of the discrete watershed into a sequence of three simple steps which are easier to execute in parallel than the original algorithm. In the first step the input image is transformed into a graph representation of the image. In the second step we compute the watershed of this graph and finally we transform the resulting graph back into the image domain.

1 Introduction

In [5] an algorithmic definition of the watershed of a digital gray scale image is given. In this section we will give a short summary of this definition.

A digital gray scale image is a function $f : D \longrightarrow \mathbb{N}$, where $D \subseteq \mathbb{Z}^2$ is the domain of the image (pixel coordinates) and for some $p \in D$ the value $f(p)$ denotes the gray value of this pixel. Gray scale images are looked upon as

topographic reliefs where $f(p)$ denotes the altitude of the surface at location p. Let G denote the underlying grid, i.e. G is a subset of $\mathbb{Z}^2 \times \mathbb{Z}^2$. A *path* P of length l between two pixels p and q is an $l+1$-tuple $(p_0, p_1, ..., p_{l-1}, p_l)$ such that $p_0 = p$, $p_l = q$ and $\forall i \in [0, l) : (p_i, p_{i+1}) \in G$. For a set of pixels M the predicate $conn(M)$ holds if and only if for every pair of pixels $p, q \in M$ there exists a path between p and q which only passes through pixels of M. The set M is called connected if $conn(M)$ holds. A *connected component* is a nonempty maximal connected set of pixels, i.e. if for a nonempty connected set M and for each connected set N we have $M \cap N = \emptyset \vee N \subseteq M$ then M is a connected component. A *regional minimum* (minimum, for short) of f at altitude h is a connected component of pixels p with $f(p) = h$ from which it is impossible to reach a point of lower altitude without having to climb. Now, suppose that pinholes are pierced in each minimum of the topographic surface and the surface is slowly immersed into a lake. Water will fill up the valleys of the surface creating basins. At the pixels where two or more basins would merge we build a "dam". The set of dams obtained at the end of this immersion process is called the *watershed* of the image f.
In order to define watersheds mathematically, we need a few definitions.

Definition 1. Let A be a set, and a, b two points in A. The *geodesic distance* $d_A(a, b)$ within A is the infimum of the lengths of all paths from a to b in A. With a little abuse of notation we write $d_A(a, B)$ for the geodesic distance between a point $a \in A$ and a set B, with $A \cap B \neq \emptyset$, which is the infimum of the lengths of all paths from a to any point in $A \cap B$.

Definition 2. Let A be some finite set of pixels. Let $B \subseteq A$ be partitioned in k connected components B_i. The *geodesic influence zone* of B_i within A, denoted $iz_A(B_i)$, is defined as the set

$$\{p \in A \mid \forall j \in [1..k] \backslash \{i\} : d_A(p, B_i) < d_A(p, B_j)\}$$

The set $IZ_A(B)$ is defined as the union of the influence zones of the connected components of B, i.e. $IZ_A(B) = \bigcup_{i=1}^{k} iz_A(B_i)$.

Definition 3. The complement of the set $IZ_A(B)$ within A is called the *skeleton by influence zones* of A, $SKIZ_A(B) = A \backslash IZ_A(B)$.

Definition 4. Let f be a gray level function. The set $T_h = \{p \in D \mid f(p) \leq h\}$ is called the *threshold set* of f at level h.

Let h_{min} and h_{max} respectively be the minimum and maximum gray level of the digital image. Let min_h denote the union of all regional minima at the height h.

Definition 5. Watershed definition Define the following recurrence for $h \in [h_{min}, h_{max})$:

$$X_{h_{min}} = T_{h_{min}} = \{p \in D \mid f(p) = h_{min}\}$$

$$X_{h+1} \quad = \quad X_h \cup min_{h+1} \cup (IZ_{T_{h+1}}(X_h) \backslash T_h)$$

The *watershed* of the image f is the complement of $X_{h_{max}}$ in D:

$$Wshed(f) = D \backslash X_{h_{max}}$$

Intuitively, one could interpret X_h as the set of pixels p, satisfying $f(p) \leq h$, that lie in some basin.

The recursion above is based upon the following case analysis [5], which is explained here in some detail in preparation of the parallel algorithm to follow.

For the recursive relation between X_h and X_{h+1} the threshold set T_{h+1} is considered. It is obvious that $X_h \subseteq X_{h+1} \subseteq T_{h+1}$. Let Y be a connected component of T_{h+1}. There are three possible relations between Y and X_h:

1. $Y \cap X_h = \emptyset$. In this case Y is a new minimum at level $h+1$ and thus (after piercing a hole in it) the starting set of a new basin. Clearly $Y \subseteq X_{h+1}$.

2. $Y \cap X_h \neq \emptyset$ and is connected. Clearly Y is an extension of the basin X_h, and thus $Y \subseteq X_{h+1}$.

3. $Y \cap X_h \neq \emptyset$ and is not connected. In this case Y contains two or more distinct minima of f. Let $Z_1, \ldots, Z_k$ be these minima. Then the basin X_h is expanded by computing the geodesic influence zone of Z_i within Y.

Most implementations of algorithms that compute the watershed of a digital gray scale function are direct translations of this recursive relation. The basic structure of these algorithms is a main loop in which h ranges from h_{min} to h_{max}. In every iteration the basins belonging to the minima are extended with their influence zones within the set T_{h+1}. The fact that X_h is needed to compute X_{h+1} clearly expresses the sequential nature of this algorithm.

2 Watershed of a Components-graph

Computing influence zones is not necessary if we can guarantee that no plateaus, clusters of neighbouring pixels that have the same gray-value, occur in the image. Of course, this is generally not true. Now, suppose that the image f does not contain plateaus, i.e.

$$\forall p, q \in D : (p, q) \in G \Rightarrow f(p) \neq f(q)$$

In this case every 'plateau' consists of exactly one pixel. We can artificially satisfy the condition above by transforming the image f into a directed valued graph $f^* = (F, E)$, called the *components graph of* f. Here F denotes the set of vertices of the graph and $E \subseteq F \times F$ the set of edges. The vertices of this graph are maximal connected sets of pixels which have the same gray-values. In the

remainder of this paper these sets are called *level components*. The set of level components at level h is defined as

$$L_h = \{C \subseteq T_h \backslash T_{h-1} \mid C \textit{ is a conn. component}\}$$

The set of vertices of the graph f^* is the collection of level components of f, i.e. $F = \bigcup_{h=h_{min}}^{h_{max}} L_h$. For level components v and w we have $(v, w) \in E$ iff. $\exists p \in v, q \in w : (p, q) \in G \wedge f(p) < f(q)$. By definition every directed path through this graph increases in altitude. With a little abuse of notation we denote the gray-value of a level component w by $f^*(w)$, which is the value $f(p)$ for some $p \in w$ if w is not a local minimum. If w is a minimum we define $f^*(w) = h_{min}$. Note that changing the gray value of a local minimum into the gray value of the absolute minimum does not change the catchment basin associated with such a minimum, but it avoids introducing new minima during the execution of the watershed algorithm. We denote the number of minima by N, such that we can index the minima M_1,..,M_N. Now, we can define the watershed of a components graph in a similar fashion as in the case of a gray level image.

Definition 6. Watershed of a components graph Define the following recurrence for $h \in [h_{min}, h_{max})$ and $i \in [1..N]$

$$X^i_{h_{min}} = \{M_i\}$$

$$\begin{aligned} X^i_{h+1} = X^i_h \cup \\ \{v \in F \mid f^*(v) = h+1 \wedge \\ (\exists w \in X^i_h : (w, v) \in E) \wedge \\ (\forall j \neq i, w \in X^j_h : (w, v) \notin E)\} \end{aligned}$$

The *watershed* of the components graph f^* is the complement of the union of the catchment basins in F:

$$Wshed(f^*) = F \backslash \bigcup_{i=1}^{N} X^i_{h_{max}}$$

Note that this definition closely resembles the definition of the watershed of a gray level image. In this definition we do not have to consider local minima at level $h+1$ since we changed the gray level of the local minima into h_{min}.

The expansion of catchment basins with their influence zones is now replaced by merging components at level $h+1$, that can be reached from exactly one catchment basin, to the corresponding basin. If a component can be reached from two different catchment basins then the node is defined to be a watershed node.

3 Parallel Computation of the watershed of a components graph

The definition of the watershed of a components graph given in the previous section suggests a simple algorithm for computing the watershed of a components graph. The idea is to compute the catchment basins $(CM_i)_{i\in[1..N]}$ by computing all possible paths that start in the minima M_i. The sets CM_i can easily be computed using standard breadth first graph algorithms. After computing these basins the algorithm determines which components are contained in two or more basins. These nodes are the watershed components. The nodes that are contained in exactly one basin are non-watershed nodes.

The time required to compute the catchment basin of one minimum is proportional with the number of nodes in the components graph. Let us say that C is the number of components in the graph, i.e. $C =\mid F \mid$. Computing all the basins one after another has complexity of the order $C \times N$. Since, for typical gray scale images, C and N are very large, the computation of the watershed in this way is very expensive. An alternative is to start computing all the catchment basins of the minima in parallel such that we can stop expanding a basin in a particular direction as soon as we have discovered that in that direction two or more basins have come together.

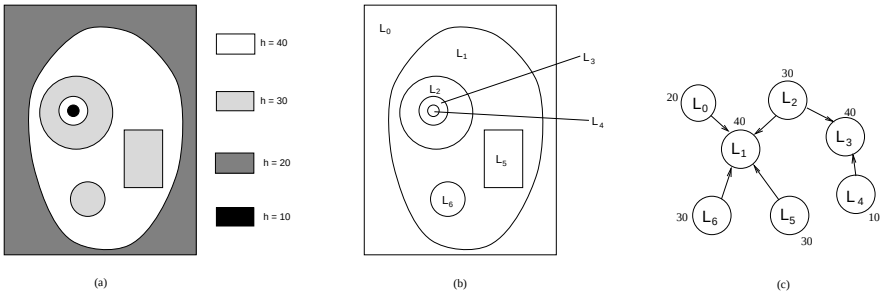

Figure 1: (a) original image f. (b) labeled level components. (c) components graph f^*. (d) watershed of f^*

Suppose we have a shared memory system with N processors. Each procesor i is assigned the task to label the components belonging to catchment basin X_h^i with the value i, unless this component is discovered to be a watershed component in which case it is labeled with the value $N+1$. We introduce an array $wsh[1..C]$ which is indexed by components. This array can be accessed and modified by each processor. In this array the labeling of the components is stored. Initially, all components are labeled with the (invalid) label 0. In the parallel algorithm each processor i changes the value of the corresponding

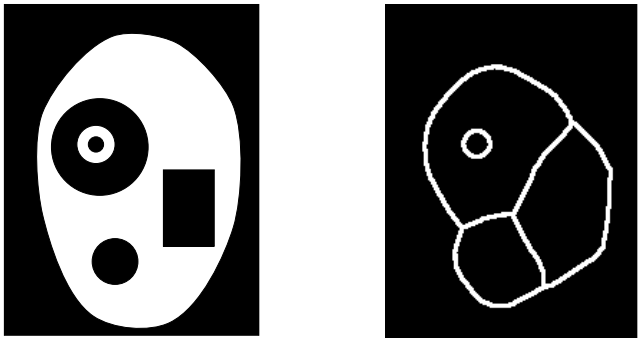

Figure 2: (left) thick watershed of f; (right) skeleton of thick watershed of f

initially: N is the number of minima;
$(\forall v \in F : wsh[v] = 0)$

do parallel for $i \in [1..N]$
begin $wsh[M_i] := i$;
$s := \{M_i\}$;
while $s \neq \emptyset$ **do**
begin choose $v \in s$;
$s := s \backslash \{v\}$;
$n := Neighbours(v)$;
$(* \; n = \{w \in F \mid (v, w) \in E\} \; *)$
forall $w \in n$ **do**
begin $P(w)$;
if $wsh[w] = 0$ **then**
begin $wsh[w] := wsh[v]$;
$V(w)$;
$s := s \cup \{w\}$
end else
if $wsh[w] \neq wsh[v]$ **then**
begin $wsh[w] := N + 1$;
$V(w)$;
$s := s \cup \{w\}$
end else $V(w)$
end
end
end;

Figure 3: Parallel watershed algorithm on a components graph.

minimum M_i into the label i. Since a minimum can never be reached from any other component in the graph, we do not have to worry that this value ever gets changed by any other processor.

During the expansion process, each processor expands its corresponding catchment basin iteratively. In each iteration the neighbours of the components that were added in the previous iteration are computed. The label of each neighbour is inspected inside a critical section. A critical section is a part of the program that can be executed by exactly one processor at the same time. This is necessary in order to avoid that values get overwritten when two or more processors try to change the value of a variable. A standard technique for solving this problem is to use semaphores, see [1]. A (binary) semaphore can be regarded as a special kind of boolean variable that can be changed by exactly one processor at the same time, using the operations P and V. A part of a program that is surrounded by a P and a V operation on the same semaphore is called a critical section. Semaphores are initalized with the value `true`. If s is a semaphore, with $s =$ `true`, then $P(s)$ changes the value of s into `false` and control is returned to the calling process immediately. If $s =$ `false` then each process that calls $P(s)$ is blocked until s becomes `true` again and one of the processes can enter the critical section and set the value of s to `false` again. A process that has passed a P-operation, and thus has blocked all other processors on the corresponding semaphore, can unblock the semaphore with the operation $V(s)$ which sets the value of s to `true`. For a complete description of semaphores the reader is referred to [1].

If a neighbouring component w has not been labeled with a valid label yet, i.e. $wsh[w] = 0$, then w is labeled with the label of the component from which it has been reached, and thus w is merged with the basin. If w was already assigned a label that differs from the label of the component from which it was reached then the node is contained in some other basin, and thus it can be reached from at least two different minima. In this case w is labeled with the label $N + 1$ which means that w is a watershed component. If another process reaches this watershed component it can stop tracking all the paths via this component because it knows that all components that are reached along these paths have already been labeled by some other process, or they will be labeled during the execution of the rest of the algorithm. Because of this fact each component of the graph is labeled at most twice, and each node that has been labeled twice will not be visited again during the execution of the algorithm. At each visit a component is assigned a label. This means that this algorithm executes in time that is linear in the number of nodes in the graph, which is much better than time complexity $C \times N$ in the sequential case.

In general, the number of minima in the graph will exceed the number of available processors. This problem can be solved by creating virtual processors by running more than one process on a single processor. This kind of pseudo-parallelism does not affect the execution of the algorithm.

4 Computation of the watershed of a gray scale image

The computation of the watershed of a gray
scale image according the algorithm given in [5] is much more complex than the algorithm given in the previous section. This is a result of the fact that it is impossible to determine whether a pixel is a watershed pixel using the gray value of its neighbouring pixels, since a pixel can be part of a (very large) plateau. This fact makes it hard to compute the watershed of a gray scale image at the pixel level. We propose that the computation of the watershed of a gray scale image is performed in three consecutive steps. In the first step the level sets of the image are computed and the components graph is built. Computing level sets is a fast and simple operation, which can be parallelized but it usually is not worth the burden of doing this.

In the second step of the algorithm we compute the watershed of the components graph that we computed in the first step. This can be done using the algorithm given in the previous section. Finally the image is transformed back into the image domain. This step can be performed sequentially or in parallel. Both algorithms are evident. The result of transforming level nodes of the graph back into sets of pixels is that we end up with thick watershed plateaus, which is usually undesirable. In that case we can decide to use some skeletonization algorithm to obtain thin watersheds. Note that this is perfectly acceptable, since we can choose the watershed lines within a plateau arbitrarily.

5 Conclusions

In this paper we have shown that it is possible to compute the watershed transform of a gray scale image in parallel by splitting the computation in three consecutive stages. In theory all these stages can be implemented in parallel, but in practice it is only worth the bureden to implement the second stage in parallel.
In the first stage of the algorithm the input image is transformed into a directed components graph. In the second stage of the algorithm the watershed of this graph is computed by a breadth first colouring algorithm. The decision which colour to assign to a certain node can be made by examining the colours assigned to its neighbouring nodes. This locality property makes it possible to perform this stage in parallel, in contrast with the classical watershed algorithm. In the final stage of the algorithm the flooded graph is transformed back into the image domain. Pixels belonging to watershed nodes of the graph are coloured white, while pixels belonging to non-watershed nodes are coloured black. The resulting watersheds are "thick". "Thin" watersheds can be obtained by performing some skeletonization algorithm on the output image. The choice which skeletonization algorithm to use is arbitrary.

References

[1] E.W. Dijkstra. Co-operating Sequential Processes. In F. Genuys (ed.), *Programming Languages*, Academic Press, London, 1968, pp.43-112

[2] A. Meijster and J.B.T.M. Roerdink. A Proposal for the Implementation of a Parallel Watershed Algorithm. In Proceedings of CAIP'95, Springer Verlag, 1995.

[3] A.N. Moga, T. Viero, B.P. Dobrin, M. Gabbouj. Implementation of a distributed watershed algorithm. In J. Serra and P. Soille (Eds.), *Mathematical Morphology and Its Applications to Image Processing*, Kluwer, 1994, pp. 281-288.

[4] A.N. Moga, T. Viero, M. Gabbouj. Parallel Watershed Algorithm Based on Sequential Scanning. In I. Pitas (Ed.), *1995 IEEE Workshop on Nonlinear Signal and Image Processing*, June 20-22, Neos Marmaras, Halkidiki, Greece, pp. 991-994.

[5] L. Vincent and P. Soille, Watersheds in Digital Spaces: An Efficient Algorithm Based on Immersion Simulations. IEEE Transactions on Pattern Analysis and Machine Intelligence, **13**, no. 6, pp 583-598, june 1991.