# Parallel Volume Image Segmentation with Watershed Transformation

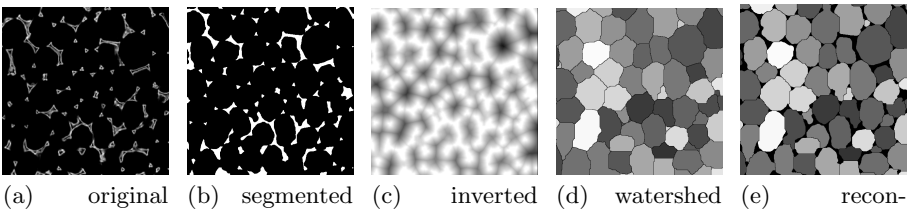
Björn Wagner[1], Andreas Dinges[2], Paul Müller[3], and Gundolf Haase[4]

[1] Fraunhofer ITWM, 67663 Kaiserslautern, Germany
bjoern.wagner@itwm.fraunhofer.de
[2] Fraunhofer ITWM, 67663 Kaiserslautern, Germany
[3] University Kaiserslautern, 67663 Kaiserslautern, Germany
[4] Karl-Franzens University Graz, A-8010 Graz, Austria

**Abstract.** We present a novel approach to parallel image segmentation of volume images on shared memory computer systems with watershed transformation by immersion. We use the domain decomposition method to break the sequential algorithm in multiple threads for parallel computation. The use of a chromatic ordering allows us to gain a correct segmentation without an examination of adjacent domains or a final relabeling. We will briefly discuss our approach and display results and speedup measurements of our implementation.

## 1 Introduction

The watershed transformation is a powerful region-based method for greyscale image segmentation introduced by H. Digabel and C. Lantuéjoul [2]. The greyvalues of an image are considered as the altitude of a topological relief. The segmentation is computed by a simulated immersion of this greyscale range. Each local minimum induces a new basin which grows during the flooding by iterative assigning adjacent pixels. If two basins clash the contact pixels are marked as watershed lines.



(a) original scan (b) segmented and closed edge system (c) inverted distance map of the background (d) watershed transformation of the distance map (e) reconstructed cells

**Fig. 1.** Cell reconstruction sequence of a metal foam

A.-B. Salberg, J.Y. Hardeberg, and R. Jenssen (Eds.): SCIA 2009, LNCS 5575, pp. 420–429, 2009.

In 3d image processing the watershed transformation can be used for object reconstruction. This is shown in figure 1 for the reconstruction of the cells of a metal foam[1] from a computer tomographic image. Due to the huge size of volume datasets the watershed transformation is a very computation intense task and the parallelization pays off.

The paper is organized as follows. Section 2 describes the sequential algorithm we used as a base for our parallel implementation. Section 3 gives a detailed description of our parallel approach and in section 4 we present some benchmarks and discuss the results.

## 2 The Sequential Watershed Algorithm

### 2.1 Preliminary Definitions

This section outlines some basic definitions, detailed in [6], [4] and [3].

A graph $G = (V, E)$ consists of a set $V$ of vertices and a finite set $E \subseteq V \times V$ of pairs defining the connectivity. If there is a pair $e = (p, q) \in E$ we call $p$ and $q$ neighbors, or we say $p$ and $q$ are adjacent. The set of neighbors $N(p)$ of a vertex $p$ is called the neighborhood of $p$.

A path $\pi = (v_0, v_1, \ldots, v_l)$ on a graph $G$ from vertex $p$ to vertex $q$ is a sequence of vertices where $v_0 = p, v_l = q$ and $(v_i, v_{i+1}) \in E$ with $i \in [0, \ldots, l)$. The length of a path is denoted with $length(\pi) = l + 1$.

The geodesic distance $d_G(p, q)$ is defined as the length of the shortest path among two vertices $p$ and $q$. The geodesic distance between a vertex $p$ and a subset of vertices $Q$ is defined by $d_G(p, Q) = \underset{q \in Q}{min}(d_G(p, q))$.

A digital grid is a special kind of graph. For volume images usually the domain is defined by a cubic grid $D \subseteq \mathbb{Z}^3$, which is arranged as graph structure $G = (D, E)$. For $E$ a subset of $\mathbb{Z}^3 \times \mathbb{Z}^3$ defining the connectivity is chosen. Usual choices are the 6-Connectivity, where each vertex has edges to its horizontal, vertical, front and back neighbors, or the 26-Connectivity, where a point is connected to all its immediate neighbors. The vertices of a cubic digital grid are called voxels.

A greyscale volume image is a digital grid where the vertices are valued by a function $g : D \to [h_{min}..h_{max}]$ with $D \subseteq \mathbb{Z}^3$ the domain of the image and $h_{min}$ and $h_{max}$ the minimum and the maximum greyvalue.

A label volume image is a digital grid where the vertices are valued by a function $l : D \to \mathbb{N}$ with $D \subseteq \mathbb{Z}^3$ the domain of the image.

### 2.2 Overview of the Algorithm

Vicent and Soille [7] gave an algorithmic definition of a watershed transformation by simulated immersion. The sequential procedure our parallel algorithm is derived from is based on a modified version of their method.

[1] Chrome-nickel foam provided by Recemat International (RCM-NC-2733).

The input image is a greyvalue image $g : D \rightarrow [h_{min}..h_{max}]$, with $D$ the domain of the image and $h_{min}$ and $h_{max}$ are the minimum and maximum greyvalues respectively, and the output image $l : D \rightarrow \mathbb{N}$ is a label image containing the segmentation result.

The algorithm is performed in two parts. In the first part an ordered sequence $(L_{h_{min}}, \ldots, L_{h_{max}})$ of voxel lists is created, one list $L_h$ for each greylevel $h \in [h_{min}, \ldots, h_{max}]$ of the input image $g$. The lists are filled with voxels $p$ of the image domain $D$ so that $L_h$ contains all voxels $p \in D$ with $g(p) = h$. Moreover each voxel is tagged with the special label $\lambda_{INIT}$, indicating that this voxel has not been processed.

We have to use several particular labels to denote special states of a voxel. To distinguish them easily from the labels of the basins their value is always below the first basin label $\lambda_0$.

To assign a label $\lambda$ to a voxel $p$ the label image at coord $p$ is set to $\lambda$, $l(p) = \lambda$.

The second part of the sequence of lists is processed in iterative steps starting at the lowest greylevel of the input image $h_{min}$. For each greylevel $h$ new basins are created respectively to local minimas of the current level $h$ and get a distinct label $\lambda_i$ assigned. Further already existing basins, from former iteration steps, are expanded if they have adjoining pixels of the greyvalue $h$.

The expansion of the basins at greylevel $h$ is done before the initiation of new basins by using a breadth-first algorithm [1]. Therefore each voxel of $L_h$ is tagged with the special label $\lambda_{MASK}$, to denote it belongs to the current greylevel and has to be processed in this iteration step. This is also called masking level $h$. The set $M_h$ contains all voxels $p$ of level $h$ with $l(p) = \lambda_{MASK}$.

Each voxel $p$ which has at least one immediate neighbor $q$ that is already assigned to a basin, so $l(q) \geq \lambda_0$ is appended to a FIFO queue $Q_{ACTIVE}$. Further $p$ is tagged with the special label $\lambda_{QUEUE}$, indicating that it is in a queue.

Starting from these pixels the adjacent basins are propagated into the set of the of the masked pixels $M_h$. Each pixel of the active queue is processed sequential as follows:

- If a pixel has only one adjacent basin, it is labeled with the same label as the neighboring basin.
- If it is adjoining at least two different basins, it is labeled with the special label for Watersheds $\lambda_{WATERSHED}$.

All neighboring pixels which are marked with the label $\lambda_{MASK}$ are appended to a FIFO queue $Q_{NOMINEE}$ and are labeled with the label $\lambda_{QUEUE}$.

When the queue $Q_{ACTIVE}$ is empty the queue $Q_{NOMINEE}$ becomes the new $Q_{ACTIVE}$ and a new queue $Q_{NOMINEE}$ is created. The propagation of the basins stops when there are no more pixels in one of the queues.

For each pixel $p \in Q_{ACTIVE}$ the distance $d_G(p, q)$ to the next pixel $q$ with a lower greyvalue is the same. That condition also holds for $Q_{NOMINEE}$. Further for all voxel $q \in Q_{NOMINEE}$ it is true $d(q) = d(p) + 1$ for all $p \in Q_{ACTIVE}$.

After the expansion the pixels of the current greylevel are scanned sequential a second time. If a voxel is still tagged with the label $\lambda_{MASK}$ a new basin is

created starting at this voxel. Therefore the pixel is labeled with a new distinct label and this label is propageted to all adjacent masked voxels, using a breadth-first algorithm [1] as in the flooding process. The propagation stops when no more pixels can be associated to the new basin. When there are still voxels with $l(p) = \lambda_{MASK}$ left, further basins are created in the same way until no more voxels with $\lambda_{MASK}$ label exist.

When all pixels of a greylevel are processed the algorithm continues with the following greylevel until the maximum greylevel $h_{max}$ has been processed.

Figure 3 shows a simplified example of a watershed transformation sequence on a two dimensional image.

## 3 The Parallel Watershed Algorithm

For the parallel watershed transformation we apply the divide and conquer principle. The image domain $D$ is divided into several non-overlapping subdomains $S \subseteq D$, usually into slices or blocks of a fixed size, on which the iterative steps of the transformation are performed concurrently. For each subdomain $S$ an own ordered sequence of pixel lists $(L^S_{h_{min}}, \ldots, L^S_{h_{max}})$ is created and initialized with the voxels of $S$ in the same way as for the sequential procedure. Further separate FIFO queues $Q^S_{ACTIVE}$ and $Q^S_{NOMINEE}$ are created for each $S$.

As in the serial case, the sequences are processed in iterative steps starting at the lowest greylevel of the image. For each greylevel the parallel algorithm expands existing basins and creates new basins for each subdomain concurrently. Due to the recursive nature of the algorithm we have to synchronize the performance between the subdomains to get correct results. The masking step, in which each voxel of the current greylevel is marked with the label $\lambda_{MASK}$ and the starting voxels for the label propagation are collected can be performed concurrently. The masking itself does not interact with any other subdomain. Further if a voxel of an adjacent subdomain must be checked whether it is already labeled there is also no problem with synchronization, because the relevant labels do not change during this step.

When all subdomains are masked, the algorithm can continue with the expansion of already detected basins. The algorithm implies a sequence of labeling events $\tau_p$ (read as labeling of pixel $p$), which is given by the greyvalue gradient of the input image, the ordering of the voxel lists $L^S_h$ and the scanning order of the used breadth-first algorithm. The order of labeling events was defined by sequential appending of the pixels to the queues. It can be said that if $q$ is appended to the queue after $p$ then follows $\tau_p \prec \tau_q$ (to be read p is labeled before q). Further for all $p \in Q^S_{ACTIVE}, \forall S$ and for all $q \in Q^S_{NOMINEE}, \forall S$ follows $\tau_p \prec \tau_q$. The label assigned to a voxel $p$ during the expansion depends on the labels of the already labeled voxels. The expansion relation can be formulated as follows:

$$l(p) = \begin{cases} c & \text{if } l(q) = c \ \forall q \in N^{\prec}(p) \\ \lambda_{WATERSHED} & \text{else} \end{cases} \tag{1}$$

where $N^{\prec}(p) = \{q \in N(p) : q \prec p \wedge l(q) \neq \lambda_{WATERSHED}$. If the sequence changes, for e.g. when the scanorder of the breadth-first algorithm is changed, the segmentation results also differ occasionally. Figure 2 shows an example for such a case for a simple example in one dimension. The pixels 1 and 2 are marked for labeling and are already appended to the queue $Q_{ACTIVE}$. In figure 2(a) pixel 1 will be labeled before pixel 2 and in figure 2(a) pixel 2 will be labeled before pixel 1.

As it can be seen the results of both sequences differ, because the labeling of the second pixel was influenced by the result of the first labeling. Thus it appears that we have to take care of the sequence of labeling events when performing a parallel expansion.


**Fig. 2.** Sequence dependend labeling

So if the concurrent performance does not follow the same sequence for each execution the results may be unpredictable. Therefore we introduce a further level of ordering of the labeling events.

Let $\mathcal{S}$ be the set of all subdomains of the image domain $D$. Further $E : \mathcal{S} \rightarrow \mathfrak{P}(\mathcal{S}) = \{X | X \subseteq \mathcal{S}\}$ defines the environment of a subdomain with

$$E(S) = \{T | \exists p \in S \text{ with } \exists q \in N(p) \wedge q \in T\} \tag{2}$$

We define a coloring function $\Gamma : \mathcal{S} \rightarrow \mathcal{C}$ for the subdomains, with $\mathcal{C}$ an ordered set of colors, so that for a subdomain $S$ the condition

$$\forall U, V \in \mathcal{E}(S) \cup S : \Gamma(U) \neq \Gamma(V) \tag{3}$$

holds.

Further we define a coloring for the pixels $\gamma : D \rightarrow \mathcal{C}$ so that the condition

$$\forall p \in S : \gamma(p) = \Gamma(S) \tag{4}$$

holds.

The parallel expansion of the basins works as follows. For each color $c \in \mathcal{C}$ the propagation is performed for all voxels in the $Q^S_{ACTIVE}$ queues of all subdomains $S$ with $\Gamma(S) = c$. This is done in the sequence defined by the ordering of the colors. For two subdomains $U, V$ with $\Gamma(U) < \Gamma(V)$, U is processed before $V$.

Inside of a subdomain the propagation still performs sequential as depicted in section 2.2, but subdomains $S, T$ with $\Gamma(S) = \Gamma(T)$ can be performed concurrently.

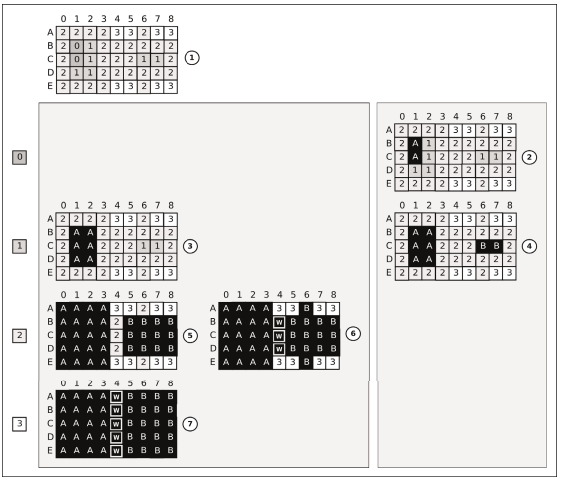


**Fig. 3.** Watershed transformation sequence

All neighboring pixels which are marked with the label $\lambda_{MASK}$ are appended to the FIFO queue $Q^S_{NOMINEE}$ of the subdomain they are element of and are labeled with the label $\lambda_{QUEUE}$.

After all colors have been processed the $Q^S_{NOMINEE}$ queues become the new $Q^S_{ACTIVE}$ queues and the propagation is continued until none of the queues of any subdomain contains any more voxels.

Due to the color depended performance of the expansion, it never happens that two voxels of adjacent subdomains are processed concurrently. So if voxel of adjacent subdomain have to be checked this can be performed without additional synchronization. Further for all pixels of any $Q^S_{ACTIVE}$ queue follows:

$$\forall p \in Q^S_{ACTIVE}, q \in Q^T_{ACTIVE}, S \neq T, \gamma(p) < \gamma(q) \rightsquigarrow p \prec q \tag{5}$$

So the results only depend on the domain decomposition of the image and the order of the assigned colors.

When the expansion has finished in all subdomains, the creation of new basins is performed. This can also be done concurrently in a similar way as by the expansion step. For each subdomain $S$ we create an own label counter $nextlabel^S$ which is initialized with the value $\lambda_{WATERSHED} + \mathcal{I}(S)$, where $\mathcal{I} : \mathcal{S} \rightarrow [1..\|\mathcal{S}\|]$ is a function assigning a distinct identifier to each subdomain. When a minimum is detected in a subdomain $S$, a new basin with the label $nextlabel^S$ is created and the counter is increased by $\|\mathcal{S}\|$. The increasing by $\|\mathcal{S}\|$ avoids duplicate labels in the subdomains.

Inside of a subdomain the propagation of a new label still performs sequential as depicted in section 2.2, but subdomains $S, T$ with $\Gamma(S) = \Gamma(T)$ can be performed concurrently, as in the expansion step. It may happen that a local minimum spreads over several subdomains and gets different labels in each subdomain. To merge the different labels the propagation overrides all labels with a

value lesser than their own. Therefore a pixel $p$ is labeled with the highest label of its neighborhood:

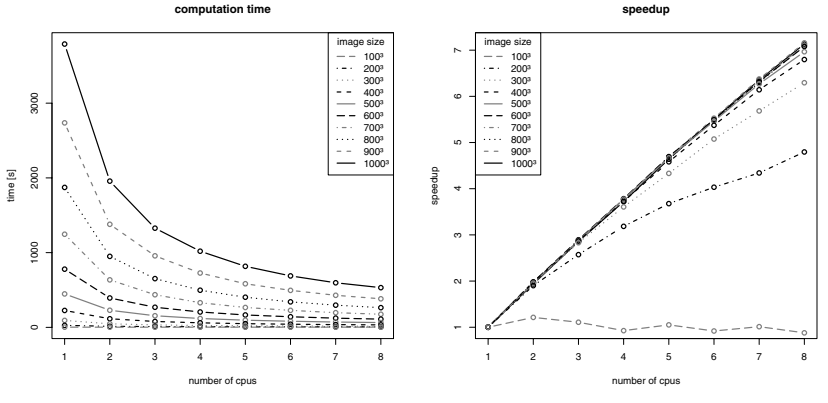
$$l(p) = \max_{q \in N(p)}(l(q)) \tag{6}$$

and this label is propageted to all adjacent voxels that are masked of have a label lower than $l(p)$. Due to the initial labeling of a new basin only affecting the pixels of minima, this simple approach doesn't interfere with other basins. The propagation stops when all voxels of the basin have the same label.

When all voxel of a greylevel have been labeled with the correct label the algorithm continues with the next greylevel until the maximum greylevel $h_{max}$ has been processed.

## 4 Results

To verify the efficiency of our algorithm we measured the speedup for datasets of different sizes[2], ranging from $100^3$ pixels to $1000^3$ pixels with cubic subdomains of a size of $32^3$ pixels on a usual shared memory machine[3]. We have chosen simulated data to be able to compare datasets of different sizes without clipping scanned datasets and influencing the results. As it is shown in figure 4(b) our algorithm scales well for image sizes above $200^3$ pixels. For images with $100^3$ and $200^3$ pixels there are not enough subdomains available for simultaneous computation to utilize the machine.




(a) computation time (b) speedup

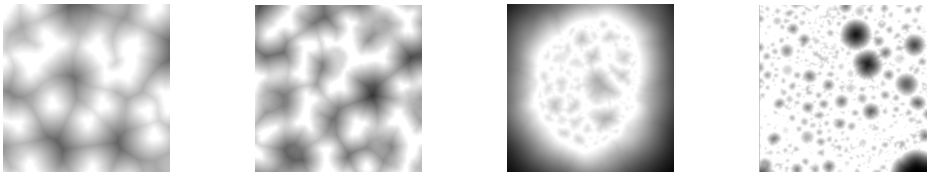
**Fig. 4.** Computation time and speedup or different image sizes

To prove the efficiency of our algorithm also for real volume datasets, we measured the speedup and the timing for the watershed transformation of the reconstruction pipeline mentioned in the introduction (see figure 1) for different

[2] Simulated foam structures.
[3] Dual Intel Xeon X5450@3.00GHz Quadcore.

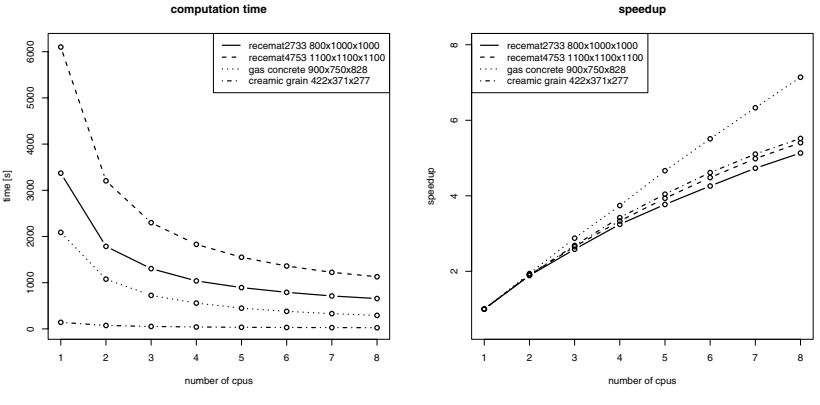
(a) recemat2733 (b) recemat4573 (c) ceramic grain (d) gas concrete

**Fig. 5.** Segmented datasets



(a) recemat2733 (b) recemat4573 (c) ceramic grain (d) gas concrete

**Fig. 6.** Distance maps



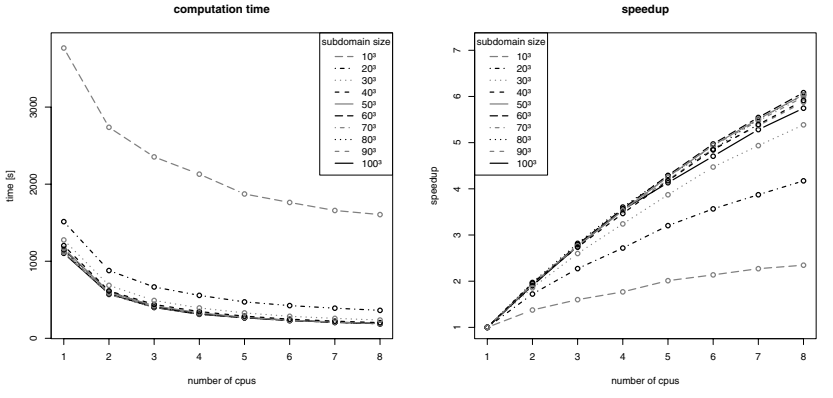

(a) computation time (b) speedup

**Fig. 7.** Computation time and speedup for different volume datasets

datasets. Figure 5 shows crosssections of the used datasets. In figure 5(a) and figure 5(b) segmentations of two different chrome-nickel foam provided by Recemat International are depicted, figure 5(c) shows a segmented ceramic grain and figure 5(d) displays the pores of a gas concrete sample. The corresponding distance maps are shown in figure 6.

As it can be seen in figure 7 our algorithm scales the same way for real datasets as for the simulated datasets.

We also measured the timing and speedup for different subdomain sizes ranging from $10^3$ to $100^3$ pixels for a sample of $1000^3$ pixel. As it is shown in figure 8 there is an impact for very small block sizes. We assume that this results from the large number of context switches in combination with very short computation times for one subdomain.




(a) computation time (b) speedup

**Fig. 8.** Computation time for different subdomain sizes

We have presented an algorithm study in order to efficiently parallelize a watershed segmentation algorithm. Our approach leads to a significant segmentation speedup for volume datasets and produces deterministic results. It still has the disadvantage that the segmentation depends on the domain decomposition. Our future work will research the impact of the domain decomposition on the segmentation results.

## References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, 2nd edn. MIT Press, Cambridge (2001)
2. Digabel, H., Lantuejoul, C.: Iterative algorithms. In: Actes du second symposium europeen d'analyse quantitative des microstructures en sciences des materiaux, biologie et medecine (1977)
3. Klette, R., Rosenfeld, A.: Digital Geometry: Geometric Methods for Digital Image Analysis. The Morgan Kaufmann Series in Computer Graphics. Morgan Kaufmann, San Francisco (2004)
4. Lohmann, G.: Volumetric Image Processing. John Wiley & Sons, B.G. Teubner Publishers, Chichester (1998)

5. Moga, A.N., Gabbouj, M.: Parallel image component labeling with watershed transformation. IEEE Transactions on Pattern Analysis and Machine Intelligence 19, 441–450 (1997)
6. Roerdink, J.B.T.M., Meijster, A.: Ios press the watershed transform: Definitions, algorithms and parallelization strategies
7. Vincent, L., Soille, P.: Watersheds in digital spaces: An efficient algorithm based on immersion simulations. IEEE Trans. Pattern Anal. Mach. Intell. 13(6), 583–598 (1991)