

A PARALLEL MARKER BASED WATERSHED TRANSFORMATION

Alina N. Moga 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

The parallel watershed transformation used in grayscale image segmentation is here reconsidered on the basis of markers. The goal is to reduce the typical over segmentation by decreasing the number of catchment basins produced by flooding.

Assimilating the set of basins with a weighted neighborhood graph and computing the minimum spanning forest in which every tree is rooted at a marked vertex, all non-marked regions in each tree are incorporated in the root region of the tree.

A $\log_2 N$ distributed message passing algorithm performing the above stated goal on N processors is here presented. Two merits of the parallel algorithm are worth of mentioning: first, the local detection of the catchment basins conforming to the watershed principle (which strongly depends on the history of the region growth), with an extremely low communication traffic; and second, the parallel computation of the Borůvka like minimum spanning forest with the constraint that any tree contains exactly one marker.

Evaluation of a Cray T3D implementation under *Message Passing Interface* (MPI) is herein included.

1. INTRODUCTION

Watershed transformation [11] is a powerful tool for morphological image segmentation. When applied to real images, the transformation however produces over segmentation. A solution to cope with this problem was to introduce markers and flood the gradient image starting from these markers instead of regional minima. Furthermore, it was shown in [5] that representing the set of catchment basins as vertices in a neighborhood graph such that every two neighboring basins are linked by an edge whose weight is the minimum altitude along the common crest line, the construction of watersheds

from a set of markers on the neighborhood graph is tantamount to the computation of a minimum spanning forest (MSF) with each tree rooted at a marker.

Starting from the above ideas, a Single Program Multiple Data (SPMD) marker based watershed algorithm is here proposed to speedup both catchment basins labeling and minimum spanning forest computation. Although previous trials were made to parallelize the classical watershed algorithm [6, 7, 8, 9], the task is not easy because of the sequential recursive nature of the transformation. However, a strategy of labeling connected components like in [1] proves applicable for watersheds too.

The paper is organized as follows. Section 2 describes the parallel marker based watershed transformation. Complexity analysis of the parallel computation of the MSF is included in Section 3, while timings obtained on a Cray T3D for four test images are tabulated and graphically interpreted in Section 4. Finally, Section 5 concludes the paper.

2. DESCRIPTION OF THE ALGORITHM

Regular image decomposition method is used to uniformly split the $n \times n$ global gradient image in stripes or blocks of approximately $\frac{n}{N}$ size and each subimage is further assigned to one processor of a N logical grid.

The algorithm starts by determining the catchment basins associated with regional minima. The sequentiality of the serial flooding is split by exploiting the two dimensional precedence relation between a candidate pixel and its already flooded neighbor at whose region the candidate pixel will be appended. Thus, the predecessor of a pixel p is an immediate neighbor q of the lowest graylevel, if any. On plateaus surrounded by pixels of lower graylevel than the plateau (non-minima plateaus), the lower distance given by the length of the shortest path completely included on the plateau to a downward rim imposes the propagation order. If pixels on such plateaus adjacent to a lower graylevel location

¹This research has been supported in part by the Graduate School in Electronics Telecommunications and Automation

have the lower distance 0, then the predecessor of any pixel p inside the plateau with the lower distance $d > 1$ will be a neighbor q on the same plateau with the lower distance $d - 1$.

Because of the decomposition of the underlying domain of the global image into subdomains, flooding would enter a subdomain through its marginal pixels p which have their predecessors q in neighboring subdomains. Therefore, all these pixels p must be detected and labeled as when they would have been already reached by flooding. Consequently, ordered flooding originating in all regional minima and these artificial seeds introduced in the edges of a subdomain will correctly outline the catchment basins in $O(\frac{n^2}{N})$.

In order to detect the edge pixels which have non-local predecessors, the lower distance must be computed for non-minima plateaus which span over several subdomains. In an unbounded loop, neighboring processors exchange and update distance values of pixels in the edges of their subdomains and propagate the changes inside the afferent non-minima plateaus. The loop runs until stabilization (no change in all processors) executing a data dependent number of iterations. However, for the images we tested an upper bound of $n_{iterations} = 3$ was recorded. The complexity of this stage is $T_{comp} = O(\frac{n}{\sqrt{N}}) + O(n_{iterations} \max_{0 \leq i < N} \{n_{nmp_i}\})$, $T_{comm} = O(\tau n_{iterations} \frac{n}{\sqrt{N}})$, where τ is a latency factor, n_{nmp_i} is the total number of pixels within shared plateaus of non-minima in processor P_i and $1 \leq n_{iterations} \leq n \times (\sqrt{N} - 1)$.

Each processor assigns labels to pixels starting from 0. In order to have non-overlapping ranges of labels, a local offset into a global range of labels is computed based on the number of labels L_i used by each processor P_i . Thus, P_0 has a local offset 0 and P_s the accumulated sum $\sum_{i=0}^{s-1} L_i$. Consequently, any label shifted with the local offset will be unique.

In this circumstance, basins extending to more than one subimage will be fragmented, being assigned different labels. Nevertheless, translating the precedence flooding relation between edge pixels and their non-local parents in terms of their labels, a boundary connectivity graph is obtained in each processor.

Our goal now is to construct the weighted neighborhood graph of the catchment basins. This graph consists of three parts. First, edges separating internal regions are collected during flooding. The weight associated with the edges is given by the minimum graylevel along the common crest line. Second, edges between adjacent regions located on neighboring processors are also appended to the graph with the same definition of the weight. Third, the boundary connectivity graph is embedded into the neighborhood graph with all edges

weighted -1 . Remark that the latter edges are the lightest in the graph and therefore they will be first included in the MSF. Consequently, fragments of the same global catchment basin produced by the domain decomposition will be in the same connected component of the MSF and thus merged together.

The neighborhood graph is further condensed such that every remaining edge satisfies the constraint that its weight is either -1 , or at most one of the two ending vertices are marked. At this stage, every processor holds a neighborhood graph in which two types of edges are distinguishable. Edges with both ends internal nodes (labeling regions from the current processor) and edges with one external end (labeling a region from an adjacent processor).

The MSF is computed in a *divide-and-conquer* fashion. A sequential MSF algorithm is applied in each processor out of N and the partial results are merged tree-wise in $\log_2 N$ steps. The global result will be available at the root processor. But this method is appropriate when the sequential MSF operator is associative and without dependencies (there is no order in which the local results are merged). According to the historical survey on the minimum spanning tree problem in [3], the Borůvka like MSF algorithm has the required properties and it possesses the greatest parallel potential. Therefore, the round robin version of the algorithm (see [10]) is used in our application.

The neighborhood graph is represented by the list of its nodes and for every node the list of edges incident on it is sorted by weight. Note that an edge (u, v) will occur in both u 's and v 's edges list, unless u or v is an external node. In this case (u, v) appears only in the edges list of the internal node, while the external node has in the current processor an empty list of outgoing edges. Nodes are processed iteratively in FIFO (First-In-First-Out) order as follows.

1. An isolated node (without outgoing edges) is a solved node and it is not processed.

2. A marked node with the lightest incident edge with a positive weight is not further processed and stored as unsolved. Non-marked adjacent nodes will hook on it or its outgoing edges will be invalidated by appending their other ending nodes to different marked components. In the latter case, the current node becomes isolated, and consequently solved.

3. Otherwise, pick the lightest edge (u, v) satisfying the enounced constraint and find their corresponding roots r_u, r_v .

- 3.a) if r_u (r_v) is an external node, r_v (r_u) is stored as unsolved.

- 3.b) if both r_u, r_v are internal nodes, their trees are merged (see *Union, Find* operations with path com-

pression in [2]). The new root is $r_{uv} = \min(r_u, r_v)$. The edges lists of r_u and r_v are merged into a new sorted list, but internal edges to the new component and outgoing edges which do not satisfy the constraint are eliminated. In addition, every root of a tree stores its component nodes. Accordingly, all the nodes equivalent to r_u and r_v , respectively, will be stored as equivalent to the new root. Moreover, if either r_u or r_v has a marker or it is unsolved, the property will be inherited by the new root as well. Finally, if the current node did not become isolated or unsolved (case 2. and 3.a) after processing, it is enqueued again.

After the working queue has been depleted, slave processors pass on their masters the unsolved nodes. For every unsolved node, its marker, the list of outgoing edges, and the internal nodes of the component are sent along with the node.

Master processors insert all the unsolved nodes in the working queue along with the ones received from their slaves. The MSF operator is applied again as described above and the result is sent further up the tree hierarchy of processors. In this way, components astride a common subdomain boundary and sharing the lightest edge will coalesce.

At the final master, the equivalence table between nodes and the roots of the trees which engulfed them lacks information about components solved earlier in the tree hierarchy. This data has not been passed on, but it still exists in the corresponding processors' tables. Since any time a node appended to a new component got a new root less than or equal to the current root, a global reduce operation with the minimum operator over all processors' tables will set in the final master's table the missing information. Further on, labels of all roots are reassigned new consecutive values starting from 0. The corresponding ranges of labels are then scattered back to processors and relabeling with the final labels is performed in every subimage.

3. COMPLEXITY ANALYSIS OF THE PARALLEL MSF ALGORITHM

If $G = (V, E)$ is the global neighborhood graph, then $|V| = L = \sum_{i=0}^{N-1} L_i$ and $|E| \leq 3L$ (planar graph). Suppose that each processor has $\frac{L}{N}$ nodes. Using the complexity of the sequential Borůvka MSF $O(m \log_2 n)$ for a graph with m edges and n vertices (see [3, 10]), the complexity of the first step is $T_{comp}^0(L, N) = O(\frac{L}{N} \log_2 \frac{L}{N})$. At each of the $\log_2 N$ steps, components across a common boundary between two neighboring subimages are merged by applying again the same MSF algorithm. If $\sqrt{\frac{L}{N}}$ is the number of nodes on the edge of a subim-

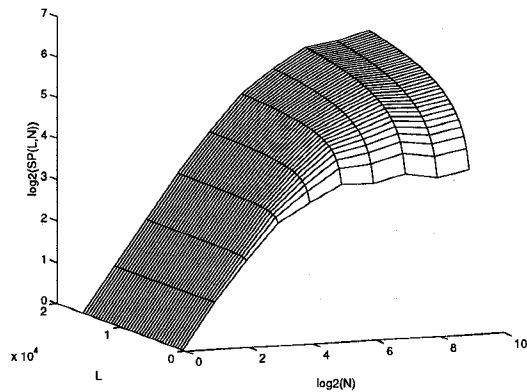


Figure 1: Theoretical speedup evaluation

age, in the k th step there are $2\sqrt{\frac{L}{N}} 2^{\lfloor \frac{k}{2} \rfloor}$ nodes and $\sqrt{\frac{L}{N}} 2^{\lfloor \frac{k}{2} \rfloor}$ edges. Thus, $T_{comp}^k(L, N) = O(\sqrt{\frac{L}{N}} 2^{\lfloor \frac{k}{2} \rfloor} \log_2(2\sqrt{\frac{L}{N}} 2^{\lfloor \frac{k}{2} \rfloor}))$ and $T_{comm}^k(L, N) = O(\tau \sqrt{\frac{L}{N}} 2^{\lfloor \frac{k}{2} \rfloor})$. The total complexity is $T(L, N) = T_{comp}^0(L, N) + \sum_{k=1}^{\log_2 N} (T_{comp}^k(L, N) + T_{comm}^k(L, N))$. Defining the relative speedup as the ratio $SP(L, N) = \frac{T(L, 1)}{T(L, N)}$ and expanding the formula, $SP(L, N) = \frac{1}{1/N + c(L, N)}$, where $c(L, N)$ is a decreasing function in both L and N . Thus, the higher the number of processors N and the number of nodes L , the higher the relative speedup (see Fig. 1).

4. EXPERIMENTAL RESULTS

Results of the algorithm implementation using MPI on a Cray T3D are presented in this section. The behavior of the algorithm was studied on images of different sizes (256×256 , 512×512 , and 1024×1024), but also on equal size images (512×512) and different complexities.

The marker images were obtained by collecting the regional maxima of the original image smoothed by a multiscale dilation with the scale $\sigma = 2.0$ (see [4]).

The execution time for all the test images (without image decomposition and retrieval) were collected up to $N = 128$ PEs and tabulated in Table 1. For comparison purpose, the number of regions of the oversegmented image ($\#regs_o$) are included along with the number of marked regions ($\#regs_m$).

Relative speedup is illustrated in Fig. 2 in logarithmic scale. As concluded in the previous section, a better performance is observed with an increasing number of processors and a higher number of regions, but also with a larger image size.

Due to the lack of space only the output of the test image *Peppers* is illustrated in Fig. 3.b to show the improvement upon the oversegmented image in Fig. 3.a.

Table 1: Execution Time (in seconds)

Image Size	Cermet 256 × 256	Lenna 512 × 512	Peppers 512 × 512	People 1024 × 1024
#regs _o	418	5370	2795	17118
#regs _m	156	709	264	1255
PEs	T(PEs)	T(PEs)	T(PEs)	T(PEs)
1	1.687556	16.853183	10.979316	73.834827
2	0.932992	9.326801	5.982528	49.075540
4	0.538162	5.946073	3.146247	24.516883
8	0.350871	2.999686	1.885601	15.267019
16	0.247727	2.052497	1.244604	7.925995
32	0.203052	1.317905	0.998566	5.207170
64	0.183916	1.052167	0.774909	3.092522
128	0.228555	1.027984	0.732203	2.341060

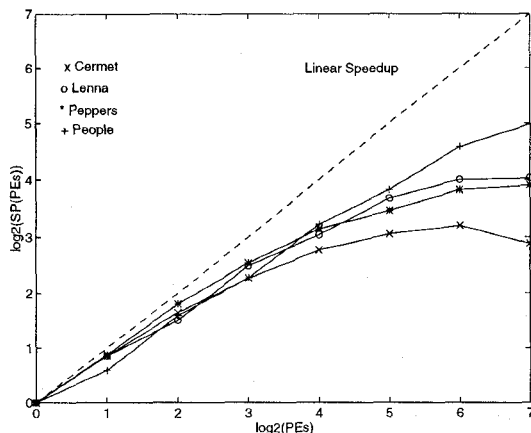


Figure 2: Speedup versus number of processors

5. CONCLUSIONS

The parallel marker based watershed algorithm bears an efficient method of computing distributedly the catchment basins and to merge these regions using a Borůvka like MSF operation with a marker based constraint.

Good performance is proved by both theoretical and experimental results, while the oversegmentation is significantly reduced.

6. ACKNOWLEDGMENT

We acknowledge the use of the Cray T3D provided by Edinburgh Parallel Computing Centre under the Training and Research on Advanced Computing Systems program.

7. REFERENCES

[1] H. M. Alnuweiri and V. K. Prasanna, "Parallel Architectures and Algorithms for Image Component Label-

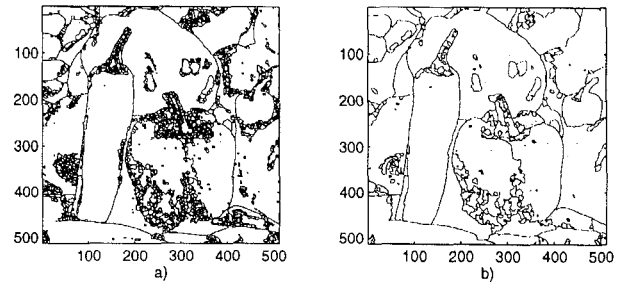


Figure 3: Output image: a) watershed algorithm; b) marker based watershed algorithm

ing," *IEEE Trans. Patt. Anal. Mach. Intell.*, vol. 14, no. 10, pp. 1014-1034, October 1992.

- [2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, "Introduction to algorithms," MIT Press, Cambridge, 1990.
- [3] R. L. Graham and P. Hell, "On the History of the Minimum Spanning Tree Problem," *Annals of the History of Computing*, vol. 7, no. 1, pp. 43-57, January 1985.
- [4] P. Jackway, "Gradient Watersheds in Morphological Scale-Space," *Proc. IEEE Workshop on Nonlinear Signal and Image Processing*, Halkidiki, Greece, June 1995.
- [5] F. Meyer, "Minimum Spanning Forests for Morphological Segmentation," *Mathematical Morphology and Its Applications to Image Processing*, pp. 77-84, Kluwer Academic Publishers, Dordrecht, The Netherlands, 1994.
- [6] A.N. Moga, T. Viero, and M. Gabbouj, "Parallel Watershed Algorithm Based on Sequential Scannings," *Proc. IEEE Workshop on Nonlinear Signal and Image Processing*, Halkidiki, Greece, June 1995.
- [7] A. Moga and M. Gabbouj, "A Parallel Watershed Algorithm Based on the Shortest Paths Computation," *Parallel Programming and Applications*, pp. 316-324, IOS Press, Amsterdam, The Netherlands, May 1995.
- [8] A.N. Moga, B. Cramariuc, and M. Gabbouj, "A Parallel Watershed Algorithm Based on Rainfalling Simulation," *Proc. European Conference on Circuit Theory and Design*, Istanbul, Turkey, August 1995.
- [9] A. Moga, B. Cramariuc, and M. Gabbouj, "An Efficient Watershed Segmentation Algorithm Suitable for Parallel Implementation," *Proc. IEEE International Conference on Image Processing*, Washington, D. C., October 1995.
- [10] R. Tarjan *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, Pennsylvania 19103, 1983.
- [11] 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.