PARALLEL PARTITIONING: PATH REDUCING AND UNION–FIND BASED WATERSHED FOR THE GPU

Yeva Gabrielyan[1]*, Varduhi Yeghiazaryan*[1]*, Irina Voiculescu*[2]

[1]College of Science and Engineering, American University of Armenia, Armenia
[2]Department of Computer Science, University of Oxford, UK

ABSTRACT

The watershed transformation is a common step in different image processing tasks. With the fast development of general-purpose computing on GPUs, a number of parallel watershed algorithms have been introduced for that setup. We propose two novel parallel watershed algorithms: one intended for relatively larger images and another for smaller images. Our algorithms are based on the combination of the parallel path reduction technique, introduced in our previous paper on the topic, and the parallel version of the Union–Find algorithm. We show through multiple experiments, both in 2D and 3D, that our algorithms achieve unmatched execution times. On a typical gaming GPU, our algorithm processes an 800 megavoxel image in around 2.5 seconds.

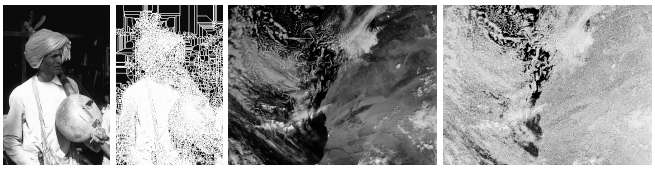
Index Terms— Watershed algorithms, GPU, CUDA, Path reduction, Union–Find

1. INTRODUCTION

The *watershed transformation* from mathematical morphology has been ordinarily used for decades in different image processing and general signal processing tasks. Reviews of definitions and sequential algorithms can be found in [1, 2, 3]. We have explained elsewhere [4, 5] the many and varied uses of this class of method. Suffice to say, for brevity, that these are used for partitioning images into regions with pixel similarity, as seen in Fig. 1. Whilst the input is typically expected to be greyscale, it is not hard to extend this work to RGB.

With the fast development of general-purpose computing on GPUs, a number of parallel watershed algorithms have been introduced for that setup [6, 7, 8, 9, 10, 4]. In our previous work [4] we presented experiments where our PRW algorithm [11] outperformed the DW algorithm [7, 12]—the state-of-the-art at the time.

We propose two novel parallel watershed algorithms designed for GPU implementation. The principal intended application is the partitioning of relatively *larger* images, typically several dozen megapixels/megavoxels or more. A few megapixels worked on a modest laptop in a fraction of a second (e.g. 4.9 megapixels in 25 milliseconds), and using more powerful GPUs allows for the processing of much larger volumes (800 megavoxels in around 2.5 seconds).



(a) (b) (c) (d)

Fig. 1. Illustration of watershed results. (a) and (c) original greyscale images; (b) and (d) original images overlaid with watershed lines in white. Individual regions are not semantically meaningful, but constitute input for other algorithms.

2. PARALLEL WATERSHED ALGORITHM

The main version of the novel algorithm, called PRUF, is presented in Alg. 1. It consists of four major steps: (S1) initialisation of pixel states and labels, (S2) resolution of non-minimal plateaux, (S3) label propagation, or path reduction, and (S4) merging of minimal plateau labels.

Step S1 categorises the pixels in the image into downhill (state 0)—pixels that lead to local minima; local minima (state 1); and plateaux (states 2 and 3)—neighbouring pixels that share the same greyscale value. The division of plateau pixels into states 2 and 3 is based on a row-major ordering of all pixels, thus identifying pixels that can lend their indices for labelling whole water catchment basins—disjoint regions separated by the watershed transform. For all pixels, a label is initialised as the index of the steepest downhill neighbour or a self-loop in case of state-1 or state-3 pixels. Step S2 further identifies non-minimal plateaux, changing the states of these pixels into 0. At this point, labels form downhill paths from every pixel to the minimum (state-1 or state-3 pixel) of the corresponding catchment basin. Step S3 reduces those paths into direct pointers to local minima. Note how the path reduction mechanism, based on a parameter called *reduction rate* (RR), has been developed to make use of synchronisations between the different threads and thus to speed up the process in a parallel implementation.

Algorithm 1 Path reducing watershed using UNION–FIND

```
1: for all p ∈ I in parallel do                ▷ Step S1: initialisation of labels & states
2:     q ← max{s ∈ N(p) | ∀t ∈ N(p), img(s) ≤ img(t)}
3:     if img(q) < img(p) then
4:         label(p) ← q; state(p) ← 0
5:     else if img(q) > img(p) then
6:         label(p) ← p; state(p) ← 1
7:     else if q > p then
8:         label(p) ← q; state(p) ← 2
9:     else if q < p then
10:        label(p) ← p; state(p) ← 3

11: repeat                                     ▷ Step S2: resolution of non-minimal plateaux
12:    for all p ∈ I in parallel do
13:        if state(p) ≥ 2 and ∃q ∈ N(p), state(q) = 0 ∧ img(q) =
           img(p) then
14:            label(p) ← q; newstate(p) ← 0
15:        else
16:            newstate(p) ← state(p)
17:    swap(state, newstate)
18: until ¬change

19: repeat                                     ▷ Step S3: label propagation, or path reduction
20:    for all p ∈ I in parallel do
21:        for i ← 1, RR and label(p) ≠ label(label(p)) do
22:            label(p) ← label(label(p))
23: until ¬change

24: for all p ∈ I in parallel do               ▷ Step S4: merging minimal plateau labels
25:    if state(p) ≥ 2 then
26:        for all q ∈ N(p), state(q) ≥ 2 do
27:            UNION(label, p, q)
28: for all p ∈ I in parallel do
29:    label(p) ← FIND(label, p)
```

After the execution of steps S1–S3, the labels of the pixels can be interpreted as a *disjoint-set* data structure. Every unique label forms a disjoint set of pixels pointing directly to the set representative. Every catchment basin in the watershed partitioning corresponds to one or a few disjoint sets. The case with a few labels—disjoint sets—covering the same catchment basin may only occur if the basin minimum is a plateau with more than one state-3 pixels (possible for 2D or higher-dimensional lattices). Thus, a parallel UNION–FIND implementation [13] is adopted for merging multiple disjoint sets denoting the same catchment basin. Our algorithm performs a UNION whenever neighbouring minimal-plateau pixels with varying labels are detected. This way, the different disjoint sets inside the same catchment basin are merged. Finally, the label of each pixel is replaced with the index of the set representative by performing a FIND.

The computational bottleneck of this algorithm is step S2 which, in the worst case, may require a linear number of iterations in the image size. We have previously shown [4] three implementation variants for this step: synchronous, block-asynchronous and balanced. The synchronous variant is the slowest due to all CUDA threads synchronising after every iteration. While the block-asynchronous variant is predictably the fastest, it does not ensure fair division of non-minimal plateaux among catchment basins. The balanced variant is computationally very close to the block-asynchronous variant while providing a fair, 'balanced', division of plateaux. Thus, the implementation of our novel algorithm adopts this balanced approach.

A slightly modified second version of the algorithm, called APRUF, has been devised for relatively *smaller* images (typically with a resolution below 10 megapixels/megavoxels). It modifies the implementation of step S3 by replacing lines 19–23 in Alg. 1 with a copy of lines 28–29. In smaller images, catchment basins are typically smaller, leading to shorter label paths to minima. In such cases, the costly global synchronisations outweigh the gain in execution time achieved from making updated labels visible to other threads/pixels.

The main contributions of our current work are twofold:

- a parallel UNION–FIND algorithm for the GPU [14, 15, 13] has been utilised to modify step S4, thus producing the PRUF,
- an alternative version of the algorithm, APRUF, has been devised to speed up step S3 for smaller images.

3. RESULTS AND DISCUSSION

3.1. Experimental setup and dataset

The publicly available datasets used in the experiments include the C4L Image Dataset [16]—a set of various greyscale images grouped based on resolution—and the Berkeley Segmentation Data Set and Benchmarks 500 (BSDS500) [18, 17]. We converted the BSDS500 images to greyscale without any further processing before running the watersheds. In addition, we include three 3D images (128 megavoxels each) of resolutions $320\times320\times1250$, $500\times500\times512$, $4000\times4000\times8$, and a single 3D image (800 megavoxels) of resolution $4000\times4000\times50$, all constructed from the same microCT image [4]. The $4000\times4000\times8$ image is also interpreted, in separate experiments, as a set of eight 2D images.

The experiments were carried out on three different machines to include different processors and GPUs spanning multiple generations and computing capabilities: (a) Intel Core i7-6700HQ, NVIDIA GeForce GTX 960M with CUDA 10.1, (b) Intel Core i7-8700, NVIDIA GeForce GTX 1080 with CUDA 11.5, (c) AMD Ryzen 7 3700X, NVIDIA GeForce RTX 3060 Ti with CUDA 11.5. Every reported watershed execution time is a *minimum* of ten runs per image and an *average* across multiple images in the same dataset (where applicable). For a given image, the equivalence of the different watershed procedures is established by checking that they consistently produce the same number of regions. Much higher-performance GPUs are available in a variety of supercomputer setups, but our choice indicates that the proposed work is practical for mainstream computing hardware.

The two algorithms are based on two types of parameters: the size of the CUDA blocks and the reduction rate RR. The latter is only used in the first, main, version shown in Alg. 1. It is important to observe that these parameters only affect the

Table 1. Execution times (in ms) of the different watershed algorithms on 2D images. The best performance is in boldface.

GPU	source	C4L Image Dataset [16]						MicroCT	Berkeley [17]
	size	256×256	512×512	1024×1024	1920×1080	2560×1440	2560×1920	4000×4000	BSDS500
960M	DW	1.422	3.273	9.349	22.223	31.651	41.307	189.744	2.329
	PRW	0.617	2.066	5.948	13.630	20.944	27.684	131.971	1.494
	APRUF	**0.573**	**1.956**	6.489	**12.889**	**19.707**	**25.894**	321.317	**1.432**
	PRUF	0.602	2.042	**5.578**	13.225	20.434	26.594	**128.512**	1.473
1080	DW	0.546	1.015	2.559	5.081	7.742	9.951	45.809	0.716
	PRW	0.267	0.629	1.901	3.813	5.890	7.636	33.170	0.402
	APRUF	**0.228**	**0.578**	2.025	**3.600**	**5.451**	**7.157**	67.010	**0.362**
	PRUF	0.253	0.608	**1.837**	3.754	5.656	7.347	**30.036**	0.389
3060Ti	DW	1.422	2.052	3.996	7.697	11.544	14.620	59.241	1.636
	PRW	0.648	1.039	2.166	4.546	6.626	8.434	35.436	0.816
	APRUF	**0.466**	**0.806**	2.091	**4.149**	**6.010**	**7.791**	66.402	**0.627**
	PRUF	0.565	0.945	**2.037**	4.334	6.360	8.126	**33.311**	0.717

execution time of the algorithm; otherwise, the results of different setups are equivalent. We have established experimentally (part of the results reported in [4, 5]) that block sizes of 16×16 in 2D and 8×8×8 in 3D, and an RR value in *the range 5–10* lead to fastest execution. Thus, we use the specified CUDA block sizes and $RR = 6$.

3.2. 2D results

We compare the performance of our new algorithms PRUF and APRUF against PRW and DW. The execution times on all our 2D datasets are reported in Table 1. Due to a warp divergence issue, the implementation of the DW algorithm taken from [12] deadlocks on some of the images. To avoid this, for the DW algorithm six images (out of 500) were excluded from the BSDS500 dataset [17] and a single image (out of 10) from the 256×256 group of C4L Image Dataset [16]. Additionally, the number of watershed catchment basins, i.e. partitioned regions, produced with DW is occasionally different from its sequential implementation, whereas in case of PRW, PRUF and APRUF it is always a match.

Considering the execution times on the 2560×1920 dataset, the DW algorithm is the slowest (14.620ms), followed by PRW (8.434ms) and PRUF (8.126ms), making APRUF the fastest algorithm (7.791ms). The same pattern is present across nearly all 2D datasets. With the increase in the image resolution (4000×4000), PRUF (33.311ms) starts to significantly outperform APRUF (66.402ms). The results produced on the 1024×1024 dataset indicate that aside from the image size, the execution time is also influenced by the spatial properties of the image.

3.3. 3D results

Table 2 illustrates the executions times of PRW, PRUF and APRUF on our 3D datasets. The 4000×4000×50 is an 8×10^8 voxel image, too large for GTX 960M and GTX 1080 GPUs.

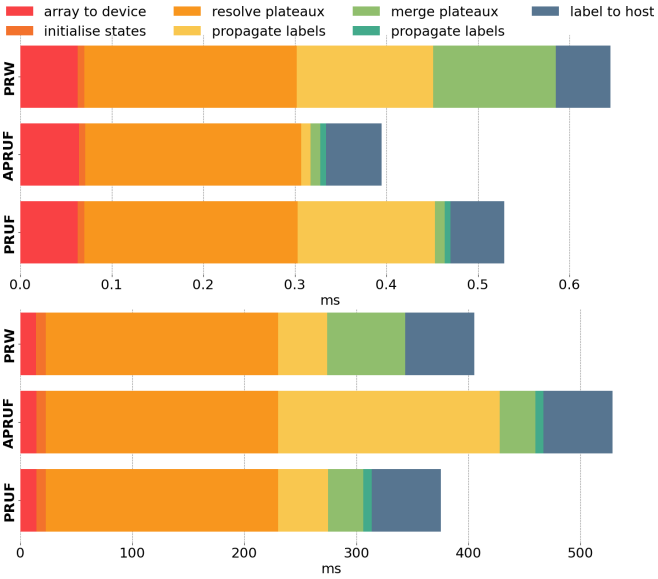


Fig. 2. Composition of the PRW, APRUF, PRUF execution times (in ms): data copy to the device (not in Alg. 1), initialise states (S1), resolve plateaux (S2), propagate labels (S3), merge plateaux and propagate labels (S4), data copy from the device (not in Alg. 1). Measured on 256×256 (top) and 500×500×512 (bottom) test images on the third GPU.

The execution time of APRUF (542.423ms) vs PRUF (385.062ms) on the 500×500×512 image volume clearly illustrates that the reduction rate on larger images is effective. Due to the parallel UNION–FIND implementation, PRUF (385.062ms) outperforms PRW (411.888ms) across all datasets. The varying numbers of partitioned regions—826280 in 320×320×1250, 908915 in 500×500×512, and 107134 in 4000×4000×8—are one of the factors explaining the noticeable differences in execution times of each algorithm across the 128 megavoxel images, e.g. 539.871ms vs

Table 2. Execution times (in ms) of the different watershed algorithms on 3D images. The best performance is in boldface.

GPU	# voxels	8.39×10^6	1.28×10^8			8×10^8
	volume	256×256×128	320×320×1250	500×500×512	4000×4000×8	4000×4000×50
960M	PRW	72.182	2242.272	1682.694	**1476.596**	
	APRUF	**65.704**	3200.780	2190.510	2959.418	GPU too small
	PRUF	67.157	**2117.745**	**1558.141**	1603.095	
1080	PRW	17.133	478.486	396.692	351.119	
	APRUF	**15.425**	695.130	483.507	629.769	GPU too small
	PRUF	15.932	**433.643**	**323.949**	**311.426**	
3060Ti	PRW	19.818	572.973	411.888	368.259	2596.140
	APRUF	**18.417**	866.459	542.423	623.693	4274.800
	PRUF	18.812	**539.871**	**385.062**	**356.605**	**2547.273**

385.062ms vs 356.605ms for PRUF. Unfortunately, the impact on the execution time of other factors, like the spatial extent of those regions, is more difficult to quantify.

3.4. Further analysis

It is interesting to observe how much the different steps of the PRW, APRUF, PRUF algorithms contribute to the total execution times of these algorithms. The composition of the three algorithms for the 256×256 (average for 10 images) and 500×500×512 test images is visualised in Fig. 2. As expected, we can observe differences in steps S3 (label propagation) and S4 (merging minimal plateaux and another propagation). S4 in APRUF and PRUF (same in both algorithms) is significantly faster than in PRW for both examples (0.011ms vs 0.134ms, 31.788ms vs 69.717ms). S3 (same in PRW and PRUF) is slower than in APRUF on the smaller images of resolution 256×256 (0.149ms vs 0.01ms). However, on the large image of resolution 500×500×512, the original approach of PRW and PRUF proves faster than the modified approach of APRUF (43.849ms vs 197.79ms). This once again confirms that the choice of the algorithm—between APRUF and PRUF—should be made based on the image resolution (and expected sizes of watershed catchment basins).

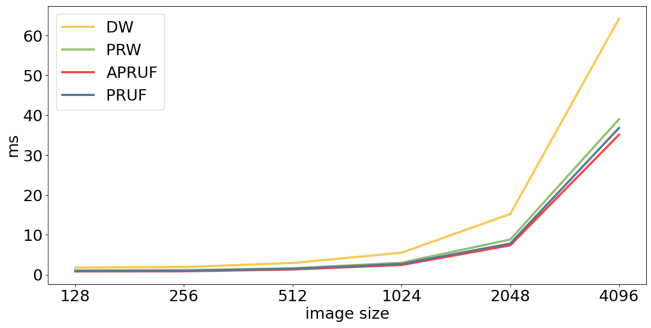


Fig. 3. Execution time (in ms) in terms of image resolution (128^2 to 4096^2), for each of DW, PRW, APRUF, and PRUF on the sample image [19] on the third GPU.

In order to analyse the dependence of the execution time on image resolution for the different watershed algorithms, we construct variants of the same image [19] in resolutions 128×128, 256×256, 512×512, 1024×1024, 2048×2048, and 4096×4096. The execution times for DW, PRW, APRUF and PRUF are illustrated in Fig. 3. As can be seen, the time growth rate for DW is much higher than that of our algorithms; and PRUF and APRUF have a lower growth rate in comparison to PRW.

Interestingly enough, the execution on the more powerful GPU, NVIDIA GeForce RTX 3060 Ti, is slower than on NVIDIA GeForce GTX 1080 across all datasets. It is likely that this results from differences in processor units (AMD and Intel, accordingly). Although both processors display similar statistics, Intel CPU cores have higher IPC (instructions per clock) and have their own heavily optimized compilers, making them generally faster on single-core executions. To support this claim we can observe that for the 500×500×512 image, PRUF on RTX 3060 Ti takes longer (14.261ms) to copy an array from host to device compared to GTX 1080 (12.07ms). Whereas label propagation, which is executed predominantly on the GPU, is much faster on RTX 3060 Ti (31.562ms) than on GTX 1080 (50.817ms).

4. CONCLUSIONS

With the advent of affordable GPU computing power, increasingly larger volumes of data (particularly images) have become easy to partition in the way we have shown. This is useful in and of itself for downstream tasks such as semantic segmentation of images, as well as for real-time signal processing tasks of other sorts (such as in voice recognition, for example). More complex algorithms that make use of image partitioning can also be envisaged, such as using the partitions (rather than individual pixels) for a variety of machine learning tasks.

5. REFERENCES

[1] Jos BTM Roerdink and Arnold Meijster, "The watershed transform: Definitions, algorithms and parallelization strategies," *Fundamenta informaticae*, vol. 41, pp. 187–228, 2000.

[2] André Körbes and Roberto de Alencar Lotufo, "Analysis of the watershed algorithms based on the breadth-first and depth-first exploring methods," in *2009 XXII Brazilian Symposium on Computer Graphics and Image Processing (SIBGRAPI)*. IEEE, Oct. 2009, pp. 133–140.

[3] R Romero-Zaliz and JF Reinoso-Gordo, "An updated review on watershed algorithms," in *Soft Computing for Sustainability Science*, pp. 235–258. Springer, 2018.

[4] Varduhi Yeghiazaryan and Irina Voiculescu, "Path reducing watershed for the GPU," in *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2018, pp. 577–585.

[5] Varduhi Yeghiazaryan, *Parallel Front Propagation in Medical Image Segmentation*, DPhil thesis, University of Oxford, 2018.

[6] Claude Kauffmann and Nicolas Piche, "Cellular automaton for ultra-fast watershed transform on GPU.," in *ICPR*, 2008, pp. 1–4.

[7] Giovani Bernardes Vitor, Janito Vaqueiro Ferreira, and André Körbes, "Fast image segmentation by watershed transform on graphical hardware," in *XXX Iberian Latin American Congress on Computational Methods in Engineering - CILAMCE 2009*, Nov. 2009, vol. 1, pp. 1–14.

[8] Michal Hučko and Miloš Šrámek, "Streamed watershed transform on GPU for processing of large volume data," in *Proceedings of the 28th Spring Conference on Computer Graphics*, New York, NY, USA, 2012, SCCG '12, pp. 137–141, ACM.

[9] Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello, "Efficient GPU asynchronous implementation of a watershed algorithm based on cellular automata," in *2012 IEEE 10th International Symposium on Parallel and Distributed Processing with Applications*, July 2012, pp. 79–86.

[10] Pablo Quesada-Barriuso, Dora B. Heras, and Francisco Argüello, "Efficient 2D and 3D watershed on graphics processing unit: block-asynchronous approaches based on cellular automata," *Computers and Electrical Engineering*, vol. 39, no. 8, pp. 2638–2655, 2013.

[11] Varduhi Yeghiazaryan and Irina Voiculescu, "Path Reducing Watershed," `www.cs.ox.ac.uk/projects/segmentation/`, Accessed: 1/2/2022.

[12] "DW," `adessowiki.fee.unicamp.br/adesso/wiki/watershed/ismm2011_dw/view/`, Accessed: 4/9/2017.

[13] Stefano Allegretti, Federico Bolelli, and Costantino Grana, "Optimized block-based algorithms to label connected components on GPUs," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 2, pp. 423–438, 2020.

[14] Costantino Grana, Federico Bolelli, Lorenzo Baraldi, and Roberto Vezzani, "YACCLAB - yet another connected components labeling benchmark," in *2016 23rd International Conference on Pattern Recognition (ICPR)*, 2016, pp. 3109–3114.

[15] Federico Bolelli, Michele Cancilla, Lorenzo Baraldi, and Costantino Grana, "Toward reliable experiments on the performance of connected components labeling algorithms," *Journal of Real-Time Image Processing*, vol. 17, no. 2, pp. 229–244, 2020.

[16] Jeffrey Bush, "C4L image dataset," `https://ieee-dataport.org/documents/c4l-image-dataset`, 2021, Accessed: 1/2/2022.

[17] "BSDS500," `www2.eecs.berkeley.edu/Research/Projects/CS/vision/grouping/resources.html`, Accessed: 1/2/2022.

[18] Pablo Arbeláez, Michael Maire, Charless Fowlkes, and Jitendra Malik, "Contour detection and hierarchical image segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 898–916, 2011.

[19] "Nighttime swirls," NASA Earth Observatory. `https://earthobservatory.nasa.gov/images/145471/nighttime-swirls`, Accessed: 11/2/2022.