Real-time image segmentation on a GPU

Alexey Abramov[1], Tomas Kulvicius[1,2], Florentin Wörgötter[1], and Babette Dellen[3,4]

[1] Georg-August University, Bernstein Center for Computational Neuroscience, Department for Computational Neuroscience , III Physikalisches Institut, Göttingen, Germany
{abramov,tomas,worgott}@bccn-goettingen.de,
[2] Department of Informatics Vytautas Magnus University, Kaunas, Lithuania
[3] Bernstein Center for Computational Neuroscience, Max-Planck-Institute for Dynamics and Self-Organization, Göttingen, Germany
[4] Institut de Robòtica i Informàtica Industrial (CSIC-UPC), Barcelona, Spain
bdellen@iri.upc.edu

Abstract. Efficient segmentation of color images is important for many applications in computer vision. Non-parametric solutions are required in situations where little or no prior knowledge about the data is available. In this paper, we present a novel parallel image segmentation algorithm which segments images in real-time in a non-parametric way. The algorithm finds the equilibrium states of a Potts model in the superparamagnetic phase of the system. Our method maps perfectly onto the Graphics Processing Unit (GPU) architecture and has been implemented using the framework NVIDIA Compute Unified Device Architecture (CUDA). For images of 256×320 pixels we obtained a frame rate of 30 Hz that demonstrates the applicability of the algorithm to video-processing tasks in real-time[1].

1 Introduction

Image segmentation, i.e. the partitioning of an image into disjoint parts based on some image characteristics, such as color information, intensity or texture is one of the most fundamental tasks in computer vision and image processing and of large importance for many kinds of applications, e.g., object tracking, classification and recognition [1]. As a consequence, many different approaches for image segmentation have been proposed in the last twenty years, e.g. methods based on homogeneity criteria inside objects of interest [2], clustering [3–5], region-based growing [1], graph cuts [6, 7] and mean shift segmentation [8]. We can distinguish between parametric (model-driven) [6, 7] and nonparametric (data-driven) techniques [1, 3–5, 8]. If little is known about the data being segmented, nonparametric methods have to be applied. The methods of superparamagnetic clustering is a nonparametric method which solves the segmentation problem by finding the equilibrium states of the energy function of a ferromagnetic Potts

[1] By real-time we understand processing of a full frame at 25Hz or faster.

model (without data term) in the superparamagnetic phase [9–11]. By contrast, methods which find solutions by computing the minimum of an energy function usually require a data term – otherwise only trivial solutions are obtained. Hence, the equilibrium-state approach to the image segmentation problem has to be considered as fundamentally different from approaches which find the minimum energy configuration of energy functions in Markov random fields [12].

The Potts model [9], which is a generalization of the Ising model [13], describes a system of interacting granular ferromagnets or spins that can be in q different states, characterizing the pointing direction of the respective spin vectors. Depending on the temperature, i.e. disorder introduced to the system, the spin system can be in the paramagnetic, the superparamagnetic, or the ferromagnetic phase. In the ferromagnetic phase, all spins are aligned, while in the paramagnetic phase the system is in a state of complete disorder. In the superparamagnetic phase regions of aligned spins coexist. Blatt et al. (1996) applied the Potts model to the image segmentation problems in a way that in the superparamagnetic phase regions of aligned spins correspond to a natural partition of the image data [11]. Finding the image partition corresponds to the computation of the equilibrium states of the Potts model.

The equilibrium states of the Potts model have been approximated in the past using the Metropolis-Hastings algorithm with annealing [14] and methods based on cluster updating, which are known to accelerate the equilibration of the system by shortening the correlation times between distant spins. Prominent algorithms are Swendsen-Wang [3], Wolff [4], and energy-based cluster updating (ECU) [5]. All of these methods obey detailed balance, ensuring convergence of the system to the equilibrium state. However, convergence has only been shown to be polynomial for special cases of the Potts model.

Since the real-time aspect is getting more and more important in image processing and especially in image segmentation, parallel hardware architectures and programming models for multicore computing have been developed to achieve acceleration [15]. In this paper, we investigate opportunities for achieving efficient performance of superparamagnetic clustering using the Metropolis algorithm with annealing [14], and propose a real-time implementation on graphics processing units (GPU). For images of size 256×320 pixels the Metropolis procedure on GPU is 160 times faster than on CPU. Furthermore, a novel short-cut, consistent with the relaxation procedure of the Metropolis algorithm, is introduced for fast cooling.

The remainder of the paper is organized as follows: Section 2 describes the proposed segmentation algorithm. In Section 3, segmentation results for several test images are presented and the respective processing times on GPU and CPU are reported. In Section 4, the results are discussed and directions for future work are given.

2 The segmentation algorithm

The overall algorithm consists of several major steps as illustrated in Fig. 1. First, a parallel Metropolis procedure is developed and used to partition the image into disjoint regions (see 2.1). To reduce the total number of required Metropolis iterations, we developed a parallel algorithm that distinguishes between true object boundaries and boundaries caused by domain fragmentation (this is: uniform areas are split into meaningless sub-segments, see 2.2). The corresponding true segments are then relabeled (see 2.3), and the Metropolis algorithm is reapplied (see 2.4) for another short relaxation process after which steady state is achieved.

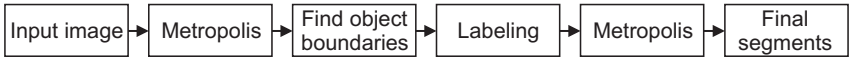

Fig. 1. Block diagram of the proposed segmentation method.

2.1 Metropolis algorithm

In the Potts model, a spin variable σ_k, which can take on q discrete values $v_1, v_2, \ldots, v_q$, called spin states, is assigned to each pixel of the image. The energy of the system is described by

$$E = -\sum_{<ij>} J_{ij}\delta_{ij} \quad , \tag{1}$$

with the Kronecker sign

$$\delta_{ij} = \begin{cases} 1 \text{ if } \sigma_i = \sigma_j, \\ 0 \text{ otherwise.} \end{cases} \tag{2}$$

where σ_i and σ_j are the respective spin variables of two neighboring pixels i and j. The function

$$J_{ij} = 1 - |\mathbf{g_i} - \mathbf{g_j}|/\overline{\Delta} \tag{3}$$

is a coupling constant, determining the interaction strength, where $\mathbf{g_i}$ and $\mathbf{g_j}$ are the respective color vectors of the pixels, and

$$\overline{\Delta} = \alpha \cdot (\sum_{<i,j>} |\mathbf{g_i} - \mathbf{g_j}| / \sum_{<i,j>} 1) \tag{4}$$

computes the averaged color vector difference of all neighbors $< i, j >$. The factor $\alpha \in [0, 10]$ is a system parameter.

The Metropolis algorithm allows generating spin configurations S which obey the Boltzmann probability distribution [16]

$$P(S) \sim \exp\left[-\beta E(S)\right] \quad , \tag{5}$$

where $\beta = 1/kT$, T is the temperature parameter, and k is the Boltzmann constant.

Initially, values are assigned randomly to all spin variables. According to the Metropolis algorithm, each spin-update procedure consists of the following steps [17]:

1. The system energy E_A of the current spin configuration S_A is computed according to Eq. 1.
2. A pixel i with spin variable σ_i in spin state v_l is selected and for each possible move to a new spin state $\sigma_i \neq v_l$ the energy E_B of the resulting new spin configuration S_B is computed according to Eq. 1. The number of possible moves is $(q-1)$.
3. Among all new possible configurations we find the configuration with the minimum energy

   $$E_{new} = \min(E_1, E_2, \ldots, E_{q-1}) \quad , \tag{6}$$

   and compute the respective change in energy

   $$\Delta E = E_{new} - E_A \quad . \tag{7}$$

4. If the total energy of the configuration is decreased by this move, i.e. $\Delta E < 0$, the move is always accepted.
5. If the energy increased, i.e. $\Delta E > 0$, the probability that the proposed move will be accepted is given by

   $$P_{A \to B} = \exp\left(-\frac{|\Delta E|}{kT_n}\right) \quad , \tag{8}$$

   and

   $$T_{n+1} = \gamma T_n \qquad \gamma < 1 \quad , \tag{9}$$

   where γ is the annealing coefficient. We draw a number ξ randomly from a uniform distribution in the range of $[0, 1]$. If $\xi < P_{A \to B}$, the move is accepted.

Each spin update involves only the nearest neighbors of the considered pixel. Hence, spin variables of pixels that are not neighbors of each other can be updated simultaneously [18]. Therefore the Metropolis algorithm fits very well to the GPU architecture.

The energy function may contain many local minima in which the system can get trapped. This problem can be resolved by slow annealing of the spin system. An annealing schedule allows to simulate a cooling process by decreasing the temperature after each iteration (see Eq. 9). While slow cooling leads to an undesired increase in computation time, fast cooling faces the problem of domain fragmentation. In the next section, we present an algorithm for resolving the domain-fragmentation problem.

2.2 Resolving Domain Fragmentation

Domain fragmentation describes the fact that large uniform areas are being split into sub-segments despite high attractive forces within them [10]. It happens in the case of a too fast annealing process when the temperature decreases rapidly and the system arrives too early at the "frozen" state. For illustration, the spin configuration with $q = 6$ after 20 Metropolis iterations is presented for an example image (Fig. 2(a-b)). Large interaction forces within the apple and the background lead to the creation of domains that try to cover each other. This effect has its origin in the finite interaction range and local dynamics of the Metropolis algorithm. The fragmented domains, however, carry all the required information to resolve this problem. For this we consider the result after an initial fast cooling phase consisting of 20 Metropolis iterations only and find that domain-fragment boundaries are unstable and clear-cut whereas true segment boundaries are stable and characterized by a noisy local neighborhood (Fig. 2(b)). This holds true for real images due to their finite image gradient at true boundaries and it allows us to distinguish true segment boundaries from those caused by domain fragmentation.

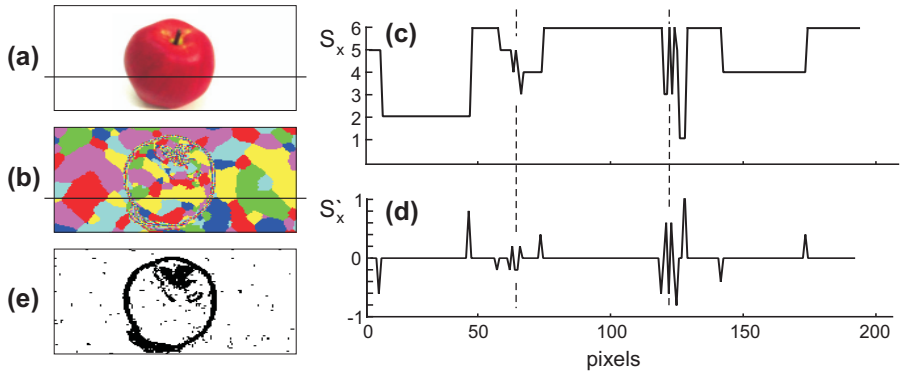

Fig. 2. Detection of real boundaries after using the Metropolis algorithm. (a) Input image. (b) Configuration of spin states after 20 Metropolis iterations. (c) Function of spin states for one image row as marked by a horizontal line in panels (a) and (b). (d) Changes of spin state for the same row where each peak represents a changing spin state. (e) Detected object boundaries.

The procedure works as follows. After a fixed small number of Metropolis iterations, we compute the spatial derivatives along the x and y direction of the spin-state configuration $S(x, y)$ according to

$$S'_x = \frac{\Delta S(x,y)}{\Delta x} \qquad \text{and} \qquad S'_y = \frac{\Delta S(x,y)}{\Delta y} \quad . \tag{10}$$

In Fig. 2(c,d) functions S_x and S'_x are depicted for one row of the original image. Each peak of S'_x represents a change in the spin state. Here we are interested only in the number of peaks rather than in the derivative values, because the Potts model does not penalize differences between certain spin states stronger than others (see Eq. 2). The frequency of peaks increases significantly at real boundaries (depicted by dashed lines). Thus considering couples of pixels in parallel we find boundaries

$$B(x_i, y_j) = \begin{cases} 1 \text{ if } S'(x_i, y_j) \neq 0 \text{ and } S'(x_{i-1}, y_j) \neq 0, \\ 1 \text{ if } S'(x_i, y_j) \neq 0 \text{ and } S'(x_i, y_{j-1}) \neq 0, \\ 0 \text{ otherwise.} \end{cases} \tag{11}$$

The result of this procedure is a binary image where objects are depicted by white and boundaries by black (see Fig. 2(e)). This step can also be implemented completely in parallel. Erroneous noisy speckles arising from this procedure are corrected by applying the Metropolis algorithm a second time for recovery (see 2.4). We used a fixed parameter $\alpha = 0.7$ for all images. For images which have not much texture a larger parameter $\alpha > 1$ can be used to obtain even better results.

Note, one cannot easily use a conventional edge detector (on the original image) for this. An edge detector would indeed find many segment boundaries, but it would also find others which are unrelated to the segments that come out from the Metropolis procedure. As we need to continue the relaxation process, we should do this using only "the correct" segments. Otherwise relaxation would have to undo all wrong segments to finally reach the minimum. Moreover the proposed procedure yields closed object boundaries while many edge detectors produce borders having gaps. The method of using the noisiness to distinguish real edges from domain edges is consistent within our algorithmic framework and, thus, allows continuation of the Metropolis procedure without problems.

2.3 Labeling of connected components

After resolving the domain fragmentation described in Sec. 2.2, all connected components, i.e. areas having a closed boundary, have to be labeled in order to get the spin states configuration back.

As our segmentation algorithm has to be sufficient for real-time applications, we decided to use a procedure suggested by He et al. (2009) which is, to our knowledge, among many algorithms proposed for the labeling of connected components in a binary image, the fastest labeling algorithm to date [19]. All steps of the employed labeling procedure are represented in Fig. 3.

The chosen algorithm completes labeling in two scans of an image: during the first scan it assigns provisional labels to object pixels (see Fig. 3(b)) and records label equivalences for labels, belonging to the same object. Label equivalences are being resolved during the first scan choosing one of the equivalent labels as a representative label. All representative labels are stored in the representative label table where provisional labels act as indexes. During the second scan,

all equivalent labels are replaced by their representative label obtained from the representative label table (see Fig. 3(c)). The detailed description of the algorithm and its optimizations can be found in [19].

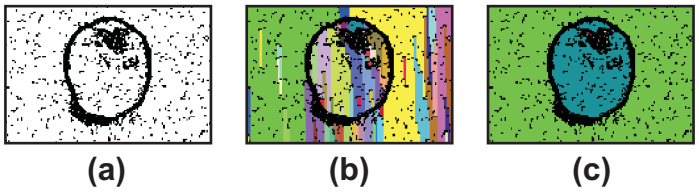

Fig. 3. Fast labeling of connected components. (a) Defined object boundaries. (b) Provisional labels after the first image scan. (c) Representative labels assigned after the second image scan.

Both image scans run on the CPU and are extremely fast for image sizes that are being used in our work. Especially the second scan can be accelerated on the GPU architecture, since representative labels can be assigned simultaneously to all pixels by independent parallel processing threads.

2.4 Employment of Metropolis for final relaxation

After the labeling of connected components we assign spin states to all pixels according to

$$\sigma(x_i, y_j) = L(x_i, y_j) \mod q \quad , \tag{12}$$

where mod means that the segment label $L(x_i, y_j)$ of the pixel is divided by the number of possible spin states q and the new spin state σ is the remainder of the division. After this assignment we apply five more Metropolis iterations to obtain the final spin configuration after which final segments can be extracted.

2.5 Experimental environment

As hardware platforms for testing of our segmentation algorithm we used

- NVIDIA card GeForce GTX 295 (using a single GPU) with 40 multiprocessors each having 8 cores, so 240 processor cores in total and 896 MB device memory.
- CPU 2.2GHz AMD Phenom Quad 9550 (using a single core) with 2 GB RAM.

3 Experimental Results

3.1 Segmentation results

We applied the developed algorithm to a set of real images, i.e. *Cluttered scene*, *Lampshade* from the Middlebury dataset[2] and *Skier* from the Berkeley dataset[3] (see Fig. 4(a)). The results at the different stages of the algorithm are shown in Fig. 4(b-e).

In Fig. 4(b), the spin states after 10 Metropolis iterations are shown. Domain fragmentation is clearly visible, characterized by noisy boundaries, in all three images. The more textured an input image is, the more noisy entities are arising. Objects borders are found, resulting in a binary image (see Fig. 4(c)). Since the *Lampshade* and *Skier* images contain much more texture than *Cluttered scene*, more boundaries and consequently more boundary errors are visible at this stage (Fig. 4(c), middle and right panels). Experimentally it was determined that 10 Metropolis iterations are enough to obtain closed object boundaries which are acceptable for the labeling of connected components.

In Fig. 4(d) the results after the labeling of connected components are represented. Errors after the resolving domain fragmentation, resulting in noisy speckles, are removed by reapplying the Metropolis procedure for system relaxation. The respective final segments extracted after the final relaxation are shown in Fig. 4(e). Fig. 4(f) shows a comparison to a conventional segmentation algorithm.

3.2 Execution time

In Fig. 5(a), the dependence of the segmentation runtime on the number of pixels in an image for the GPU architecture is shown. We can see that the dependence is almost linear, as the Metropolis algorithm, resolving the domain fragmentation and the labeling procedure have almost the ideal linearity property versus image size (i.e., for $N \times N$ images, its complexity is $O(N^2)$).

Among all algorithmic steps only the runtime of the labeling depends on the structure of the input image, but deviations are in the range of two milliseconds for images up to 256×320 pixels and of ten milliseconds for images up to 1024×1240 pixels. For very textured images like *Skier* the labeling takes longer, since shapes of objects are more difficult and more provisional labels are being assigned, so more time is needed to solve label equivalences (see 2.3). The most time-consuming step is the Metropolis procedure, taking together with the relaxation process more than 90 percent of the total execution time (see Fig. 5(b)). Processing times of our segmentation algorithm on GPU and CPU are compared in Table 1. The comparison of processing times for the proposed GPU method and the efficient graph-based method on CPU of Felzenszwalb and Huttenlocher is shown in Table 2.

[2] available under *http://vision.middlebury.edu/stereo/*
[3] available under *http://www.eecs.berkeley.edu/Research/Projects/CS/vision/bsds/*

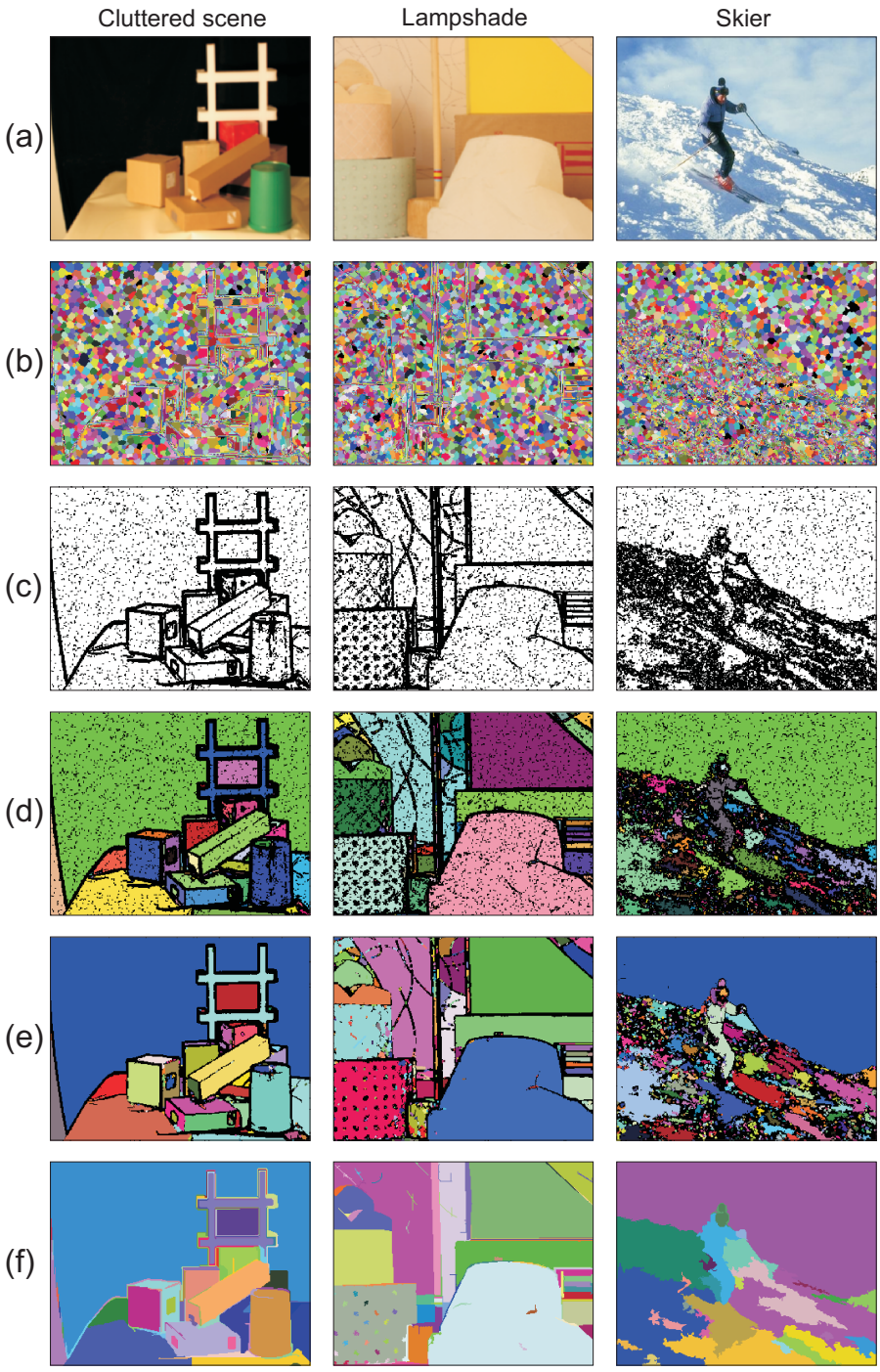

Fig. 4. Intermediate and final results of the segmentation algorithm for three example images. (a) Test images. (b) Results after 10 Metropolis iterations with $q = 256$. (c) Found objects boundaries. (d) Labeling of connected components. (e) Extracted segments after the final relaxation. (f) Results of graph-based image segmentation approach of Felzenszwalb and Huttenlocher [7].

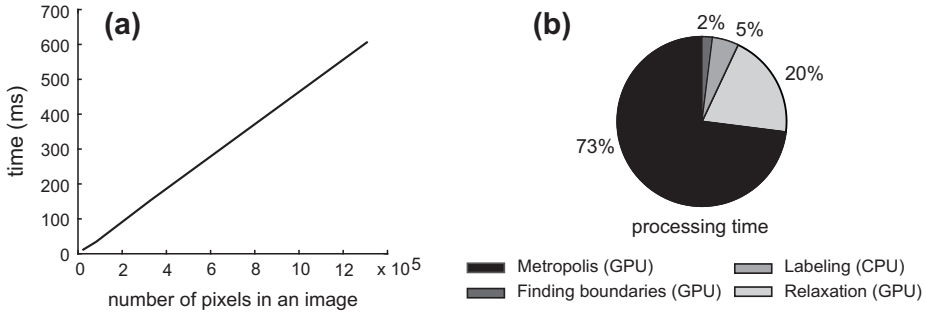

Fig. 5. Timing performances of the algorithm. (a) Execution time on GPU versus the number of pixels in an image. (b) Total computation time of all algorithmic steps in percentage.

Image size (px)	GPU / CPU (ms)		
	"Cluttered scene"	"Lampshade"	"Skier"
128×160	$9.55 / 1.4 \times 10^3$	$10.5 / 1.4 \times 10^3$	$11.0 / 1.5 \times 10^3$
256×320	$33.8 / 5.8 \times 10^3$	$34.3 / 5.9 \times 10^3$	$33.7 / 5.9 \times 10^3$
512×640	$150.5 / 24.3 \times 10^3$	$153.1 / 24.4 \times 10^3$	$154.6 / 24.3 \times 10^3$
1024×1280	$601.3 / 100.8 \times 10^3$	$612.2 / 102.2 \times 10^3$	$609.8 / 102.5 \times 10^3$

Table 1. Total computation times obtained for GPU and CPU for different sizes of the test images.

Image size (px)	GPU method / graph-based on CPU (ms)		
	"Cluttered scene"	"Lampshade"	"Skier"
128×160	9.55 / 10.0	10.5 / 10.0	11.0 / 10.0
256×320	33.8 / 75.0	34.3 / 75.0	33.7 / 75.0
512×640	150.5 / 510.0	153.1 / 500.0	154.6 / 470.0
1024×1280	601.3 / 3020.0	612.2 / 2950.0	609.8 / 2920.0

Table 2. Comparison of computation times obtained for the proposed method on GPU and graph-based method by Felzenszwalb and Huttenlocher on CPU [7].

4 Discussion

We introduced a novel parallel nonparametric image segmentation algorithm based on the method of superparamagnetic clustering. Using the highly parallel GPU architecture we obtained processing times which are sufficient for real-time applications. For images of size 256×320 pixels the algorithm can be used for real-time processing tasks and of size up to 512×620 for close to real-time applications. The algorithm has been adapted to fit the parallel architecture of GPUs, including a novel procedure to resolve the domain-fragmentation problem.

The proposed method has been applied to several real images. Obtained segmentation results for a single frame are comparable with conventional approaches. In Fig. 4(f) results of graph-based image segmentation proposed by Felzenszwalb and Huttenlocher (2004) are shown. We can see that our results (see Fig 4(e)) look very similar with the difference that our method yields more small segments for very textured images like *Skier*. This happens because our method takes into account only color information of interacting pixels. Therefore our algorithm has a better performance for large segments than for small ones, since the color segmentation works best for large uniform image regions. For textured areas, corresponding to small regions, the performance of our algorithm decreases, because the gray-value similarity of neighboring pixels is too low. Towards better results for very textured images, in the future texture segmentation can be incorporated into the algorithm. With respect to processing time our method outperforms the mentioned graph-based approach (see Table 2).

Also it is necessary to point out that the processing time is almost independent of image structure, number of segments, and image density, i.e. the relation between object pixels and boundary pixels during the labeling of connected components. The slowest part of the algorithm is Metropolis updating, since some annealing iterations have to be executed.

Before parallel hardware architectures became widespread, most image segmentation methods running on CPU either delivered precise segmentation results at low speed or real-time processing with relatively poor accuracy. Nowadays different types of parallel architectures are used for the real-time image segmentation: digital signal processors (DSP) with field programmable gate arrays (FPGA) and GPUs [15, 20, 21]. Using of DSPs and FPGAs makes it possible to achieve real-time processing [15] but requires far more development time than in the case of GPUs with CUDA. Furthermore, software developed for FPGAs is highly dependent on the used chip type and as a consequence has a limited portability while CUDA applications run on a wide range of GPUs without any problems.

After release of the framework CUDA by NVIDIA in 2007, some image segmentation algorithms were implemented on the GPU [20, 21]. A method proposed by Kim et al. (2009) segments cervicographic images using the spatially coherent deterministic annealing, but not in real-time. The real-time algorithm of Vineet and Narayanan (2008) performs a binary segmentation of the image into objects of interest and background, which is a different problem. In our

case, the whole image is segmented into similar regions according to a similarity criterion, here color.

Currently, we are investigating whether alternative approaches for computation of equilibrium states of the Potts model can be parallelized efficiently as well [3–5]. In the future, we will apply the proposed algorithm to the problem of image-sequence segmentation with the aim to track image segments in real time in a model-free way [22].

5 Acknowledgment

The work has received support from the German Ministry for Education and Research (BMBF) via the Bernstein Center for Computational Neuroscience (BCCN) Göttingen and the EU Project PACO-PLUS. B.D. also acknowledges support from Spanish Ministry for Science and Innovation via a Ramon y Cajal fellowship. We thank Karl Pauwels for valuable discussion.

References

1. Shapiro, L.G., Stockman, G.C.: Computer Vision. Prentice Hall (2001)
2. Sahoo, P.K., Soltani, S., Wong, A.K.C., Chen, Y.: A survey of thresholding techniques. Computer Vision, Graphics and Image Processing **41**(2) (1988) 233–260
3. Swendsen, R.H., Wang, S.: Nonuniversal critical dynamics in Monte Carlo simulations. Physical Review Letters **76**(18) (1987) 86–88
4. Wolff, U.: Collective Monte Carlo updating for spin systems. Physical Review Letters **62**(4) (1989) 361–364
5. von Ferber, C., Wörgötter, F.: Cluster update algorithm and recognition. Physical Review E **62**(2) (2000) 1461–1464
6. Boykov, Y., Funka-Lea, G.: Graph cuts and efficient N-D image segmentation. International Journal of Computer Vision **70**(2) (2006) 109–131
7. Felzenszwalb, P.F., Huttenlocher, D.P.: Efficient graph-based image segmentation. International Journal of Computer Vision **59**(2) (2004) 167–181
8. Comaniciu, D., Meer, P.: Mean shift: a robust approach toward feature space analysis. Pattern Analysis and Machine Intelligence **24**(5) (2002) 603–619
9. Potts, R.B.: Some generalized order-disorder transformations. Proc. Cambridge Philos. Soc. **48** (1952) 106–109
10. Eckes, C., Vorbrüggen, J.C.: Combining data-driven and model-based cues for segmentation of video sequences. In: Proc. of World Congress on Neural Networks. (1996) 868–875
11. Blatt, M., Wiseman, S., Domany, E.: Superparamagnetic clustering of data. Physical Review Letters **76**(18) (1996) 3251–3254
12. Boykov, Y., Kolmogorov, V.: An experimental comparison of min-cut/max-flow algorithms for energy minmization in vision. Pattern Analysis and Machine Intelligence **9** (2004) 1124–1137
13. Ising, E.: Beitrag zur Theorie des Ferromagnetismus. Z. Phys. **31** (1925) 253–258
14. Geman, D., Geman, S.: Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. Pattern Analysis and Machine Intelligence **6** (1984) 721–741

15. Meribout, M., Nakanishi, M.: A new real time object segmentation and tracking algorithm and its parallel architecture. Journal of VLSI Signal Processing **39**(3) (2005) 249–266
16. Carnevali, P., Coletti, L., Patarnello, S.: Image processing by simulated annealing. IBM Journal of Research and Development **29**(6) (1985) 569–579
17. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equation of state calculations by fast computing machines. J. of Chem. Phys. **21**(11) (1953) 1087–1091
18. Barkema, G.T., MacFarland, T.: Parallel simulation of the ising model. Physical Review E **50**(2) (1994) 1623–1628
19. He, L., Chao, Y., Suzuki, K., Wu, K.: Fast connected-component labeling. Pattern Recognition **42** (2009) 1977–1987
20. Kim, E., Wang, W., Li, H., Huang, X.: A parallel annealing method for automatic color cervigram image segmentation. In: Medical Image Computing and Computer Assisted Intervention, MICCAI-GRID'09 HPC Workshop. (2009)
21. Vineet, V., Narayanan, P.J.: CUDA cuts: fast graph cuts on the GPU. In: Proc. CVPRW'08. (2008) 1–8
22. Dellen, B., Aksoy, E.E., Wörgötter, F.: Segment tracking via a spatiotemporal linking process including feedback stabilization in an n-d lattice model. Sensors **9**(11) (2009) 9355–9379