Decreasing Time Consumption of Microscopy Image Segmentation Through Parallel Processing on the GPU

Joris Roels[1,2(✉)], Jonas De Vylder[1], Yvan Saeys[2], Bart Goossens[1], and Wilfried Philips[1]

[1] Department of Telecommunications and Information Processing, Ghent University, Sint-Pietersnieuwstraat 41, 9000 Ghent, Belgium
Joris.Roels@telin.ugent.be

[2] Inflammation Research Center, Flanders Institute for Biotechnology, Technologiepark 927, Zwijnaarde, 9052 Ghent, Belgium

Abstract. The computational performance of graphical processing units (GPUs) has improved significantly. Achieving speedup factors of more than 50x compared to single-threaded CPU execution are not uncommon due to parallel processing. This makes their use for high throughput microscopy image analysis very appealing. Unfortunately, GPU programming is not straightforward and requires a lot of programming skills and effort. Additionally, the attainable speedup factor is hard to predict, since it depends on the type of algorithm, input data and the way in which the algorithm is implemented. In this paper, we identify the characteristic algorithm and data-dependent properties that significantly relate to the achievable GPU speedup. We find that the overall GPU speedup depends on three major factors: (1) the coarse-grained parallelism of the algorithm, (2) the size of the data and (3) the computation/memory transfer ratio. This is illustrated on two types of well-known segmentation methods that are extensively used in microscopy image analysis: SLIC superpixels and high-level geometric active contours. In particular, we find that our used geometric active contour segmentation algorithm is very suitable for parallel processing, resulting in acceleration factors of 50x for 0.1 megapixel images and 100x for 10 megapixel images.

Keywords: Microscopy · Image segmentation · GPGPU computing

1 Introduction

High throughput and resolution microscopy imaging has gained a lot of interest due to advanced acquisition development. Consequently, image analysis algorithms should be able to keep up with the increasing data stream. In practice, this seems to be a stumbling block, especially in the case of microscopy image segmentation: higher complexity algorithms typically yield more accurate results,

© Springer International Publishing AG 2016
J. Blanc-Talon et al. (Eds.): ACIVS 2016, LNCS 10016, pp. 147–159, 2016.
DOI: 10.1007/978-3-319-48680-2_14

but because of the increasing amount of data, they become less usable. A second issue results from the increasing interest in more complex (ultra)structural content. This requires more advanced segmentation algorithms that incorporate prior knowledge such as shape and texture characteristics, typically resulting into higher computational complexity. In some cases, inaccurate automated segmentation algorithms or challenging data sets force researchers to perform segmentation completely manual. For example, in [8], a team of 224 people annotated 950 thin neuron structures in a 1 million μm^3 electron microscopy (EM) dataset at nanometer resolution, leading to more than 20000 annotator hours.

On the one hand, it is possible to use computationally cheap segmentation algorithms, typically requiring a substantial amount of manual post-processing. On the other hand, higher quality algorithms exist that typically require more computational resources. The latter is a common reason why some segmentation algorithms are not used in practice, even if they guarantee high-quality results. A popular approach to mitigate this, is the use of hardware accelerators such as graphical processing units (GPUs). These devices allow us to exploit massive parallelism resulting in significant speedup factors of more than 50x and even real-time performance in microscopy applications [5,17].

However, GPU acceleration is not straightforward: it requires extreme care and programming expertise and the achievable speedup depends on the granularity and memory requirements of the algorithm, input data dimensions, hardware characteristics, etc. In this paper, we discuss how microscopy image segmentation algorithms can be accelerated through GPU processing and how the algorithm properties and input data influence the achievable speedup. We point out that the algorithm needs to exhibit coarse-grained parallelism, the processed data size needs to be sufficiently large to benefit from GPU parallelization and the computation/transfer ratio needs to sufficiently large so that the computational cost outweigh the memory transfer cost. In Sect. 2 we will discuss the conceptual idea of GPU processing compared to traditional CPU processing. Two different segmentation algorithms are described in Sect. 3: basic superpixel segmentation and high-level active contours. Next, we identify the key properties of the algorithm and input data in order to attain a higher speedup in Sect. 4. The paper is concluded in Sect. 5.

2 CPU-GPU Parallel Processing

Traditionally, algorithm implementations are serially executed on the central processing unit (CPU). Using the GPU it is possible to exploit massive parallelism in algorithms resulting in significant speedups. However, there are several caveats linked to GPU processing:

– GPU programming in low-level programming approaches like CUDA/OpenCL requires specific knowledge on the GPU hardware architecture. One has to be aware of memory allocation, memory transfer between the CPU and GPU, memory type (local, global, shared, texture), thread synchronization, choosing GPU block sizes, etc. To alleviate these problems, recently several high-level

libraries have been developed (e.g. Thrust, HSA-Bolt, Vector, etc.) to be used from, e.g. C/C++. Alternatively, the GPU can be accessed from high-level languages like Python/Matlab. However, the program then needs to be specifically designed to run on (or take advantage of) a GPU, by using existing library functions that have been accelerated on the GPU. In many of these approaches, the programmer is still required to revert to low-level GPU programming for functionality that does not exist within these libraries.
- The GPU contains a large number of processing cores (called streaming processors) that are designed and optimized for parallel numerical computations. To take advantage of the processing abilities and memory architecture of the GPU, a relatively large number of cores (typically more than 256) needs to perform the same operation in parallel.
- The type of algorithm influences the achievable speedup using the GPU as well. Algorithms with the property that a large number of numerical operations can be performed independently, with limited dynamic control flow and limited recursion are more likely to have higher speedup factors. Memory accesses either need to be optimized for locality (shared/texture memory) or coalescing (global memory).
- Additionally, the type of data to process influences the achievable speedup using the GPU. Copying data from CPU to GPU memory and back requires time and introduces an overhead. For small data dimensions, the GPU will be scarcely occupied and acceleration due to parallel processing may not be as high as was hoped for. In these cases, parallel processing using the CPU might be more efficient (assuming the CPU is multi-core).

Recent programming languages such as Halide [15], Rust [9] and Quasar [6] (which we have used for our experiments) address the first issue by allowing the algorithm to be specified on a high level, automatic memory transfer, load balancing, scheduling, etc. The remaining points are more related to the algorithmic and data-depending influence on the attainable speedup. This is addressed in more detail in the following sections.

3 Microscopy Image Segmentation

One of the most fundamental and challenging image processing problems in microscopic imaging is segmentation. Typically, one defines this as isolating objects of interest in a given input image. In the next sections, we will describe two popular microscopy image segmentation approaches that are distinctive in terms of computational complexity and coarse-grained parallelism. We note that it is not within the scope of this manuscript to provide a detailed discussion of the techniques, for this we refer to the respective references.

3.1 Notations

We will describe a gray scale image as a function $f : \Omega \mapsto \mathbb{R}$, such that $f(\mathbf{x})$ corresponds with the pixel intensity of the image at spatial position $\mathbf{x} \in \Omega$. A

multichannel image is represented by a vector function $\mathbf{f} : \Omega \mapsto \mathbb{R}^C$ that consists of C gray scale images, one for each channel, e.g. $\mathbf{f}(\mathbf{x}) = [f_R(\mathbf{x}), f_G(\mathbf{x}), f_B(\mathbf{x})]$ for an RGB image.

A segmentation result involving K classes is defined by a classification function $u : \Omega \mapsto \mathcal{L}$ (where $\mathcal{L} = \{0, 1, \ldots, K-1\}$ denotes the set of K class labels) such that $u(\mathbf{x}) = i$ if the pixel located at position $\mathbf{x} \in \Omega$ belongs to segment class i. In the case of binary segmentation, this means that $u(\mathbf{x})$ will be a binary function evaluating to 1 if the pixel positioned at $\mathbf{x}$ belongs to the foreground segment and 0 otherwise. For the matter of readability, in the following, we will discard the spatial information unless this could lead to confusing expressions.

3.2 Superpixel Segmentation

Superpixel segmentation methods are essentially segmentation algorithms applied in over-segmentation mode. The obtained segments (or superpixels) should be connected regions of pixels with similar (intensity and/or texture) characteristics. Typically, they are used as a pre-processing step for complex image processing algorithms that are impractical on large data sets. As a result, superpixel techniques should require minimal computation time in order to avoid overhead. Superpixels have been applied intensively in electron microscopy applications because of its typical large-scale data sets [11,13,18].

A popular superpixel segmentation technique is the Simple Linear Iterative Clustering (SLIC) algorithm [1]. SLIC superpixels are generated by applying the K-nearest neighboring algorithm on a multidimensional space incorporating intensity and spatial information (where K is the desired number of superpixels). Originally, it was described for color images. The multidimensional space would then correspond to the span of the CIELAB color space (because of its perceptual meaningfulness) and the 2D spatial domain (i.e. a 5-dimensional space). However, for general multichannel images, any kind of intensity space can be used. The distance d between two points of the joint $(C+2)$-dimensional space $[\mathbf{f}(\mathbf{x}), \mathbf{x}]$ and $[\mathbf{f}(\mathbf{x}'), \mathbf{x}']$, corresponding to spatial positions $\mathbf{x}$ and $\mathbf{x}'$, is then defined as a linear combination of the Euclidean color distance and spatial distance:

$$d = \|\mathbf{f}(\mathbf{x}) - \mathbf{f}(\mathbf{x}')\|_2 + \frac{m}{S} \|\mathbf{x} - \mathbf{x}'\|_2 , \tag{1}$$

where $\|\cdot\|_2$ denotes the Euclidean norm, m is a compactness parameter that allows a trade-off between the intensity and normalized spatial distance and $S = \left\lfloor \sqrt{N/K} \right\rfloor$ is the approximate superpixel size (where N is the number of pixels in the image). Note that we have discarded vector dependencies for notational simplicity. To enforce connectivity, all disjoint clusters are reassigned to their largest neighboring cluster at then end of the algorithm. The pseudocode of this technique is shown in Algorithm 1 and Fig. 1 shows the result of SLIC superpixels computed on an electron micrograph.

As superpixels are typically generated on large-scale data sets and their computing time should be minimized to avoid overhead, GPU acceleration would be helpful.

Algorithm 1. SLIC superpixels

Input: an image $\mathbf{f}(\mathbf{x})$, compactness parameter m, preferred superpixel size S, number of iterations $niter$
Output: a labeled image $u(\mathbf{x})$ where pixels with the same label belong to the same superpixel
Lines starting with '#' indicate comments

Initialize seeds
$\mathbf{c}_k = [\mathbf{f}(\mathbf{x}_{r_k}), \mathbf{x}_{r_k}]$ (cluster centers and corresponding intensities across a regular grid r_k)
$pixelDistances(\mathbf{x}) = +\infty$
$X_k = \emptyset$
for $i = 0 \ldots niter - 1$ **do**
    # Reassign pixels
    for all $\mathbf{c}_k$ **do**
        for all $\mathbf{x}$ in a $2S \times 2S$ region around $\mathbf{c}_k$ **do**
            $dist = d([\mathbf{f}(\mathbf{x}), \mathbf{x}], \mathbf{c}_k)$
            if $dist < pixelDistances(\mathbf{x})$ **then**
                $pixelDistances(\mathbf{x}) = dist$
                $u(\mathbf{x}) = k$
            end if
        end for
    end for
    # Recompute cluster centers
    for all $\mathbf{c}_k$ **do**
        $X_k = \{\mathbf{x} \in \Omega | u(\mathbf{x}) = k\}$
        $\mathbf{c}_k = \frac{1}{|X_k|} \sum_{\mathbf{x} \in X_k} [\mathbf{f}(\mathbf{x}), \mathbf{x}]$ ($|X|$ represents the cardinality of a set X)
    end for
end for
# Enforce connectivity
for all disjoint clusters Y_l of connected pixels with the same label l **do**
    X_k = largest neighbouring cluster of Y_l
    $u(\mathbf{x}) = k$ for all $\mathbf{x} \in Y_l$
end for

3.3 Active Contour Segmentation

Model-based approaches such as active contours isolate objects of interest by using stronger prior knowledge, i.e. by modeling motion, appearance, shape characteristics, etc. A specific energy function is minimized by moving and deforming an initial contour. This energy function should be minimal when the contour is delineating the object of interest. Active contours have been applied extensively in a broad range of microscopy applications such as phase contrast [10], confocal [14] and EM [12] due to the possibility of designing very application-specific energy functions.

A popular class of active contours, so-called geometric active contours, that benefits from a convex optimization problem represents the contour implicitly

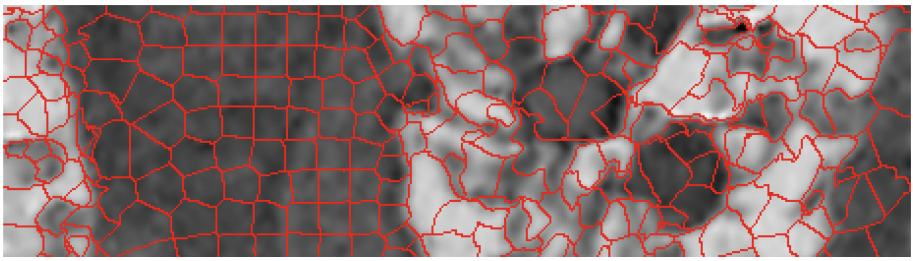


Fig. 1. SLIC superpixels computed on an electron microscopy image.

using a characteristic function $u : \Omega \mapsto [0, 1]$. This evaluates to 0 if the pixel does not belong to the segment, 1 elsewhere. We will focus on the segmentation method proposed in [4] where the contour is assumed to be smooth and forced to an intensity-based data-fit:

$$r(\mathbf{x}) = (\mu_1 - f(\mathbf{x}))^2 - (\mu_2 - f(\mathbf{x}))^2 \tag{2}$$

where μ_1 (respectively, μ_2) is the expected intensity inside (respectively, outside) of the contour. A gradient-based smoothness constraint suggests the following energy function that should be minimized:

$$E[u] = |\nabla u| + \beta \langle u, r \rangle + b(u), \tag{3}$$

where ∇ is the gradient operator, $|(w_1, w_2)| = \sqrt{\sum_{\mathbf{x}} w_1(\mathbf{x})^2 + w_2(\mathbf{x})^2}$ for images w_i, β a weighting parameter used to tune the influence of the data-fit term in relation to the total variation regularization and $\langle w, w' \rangle = \sum_{\mathbf{x}} w(\mathbf{x}) w'(\mathbf{x})$ for an image w. The function b is a convex potential function in order to constrain the minimal solution of Eq. 3 to the interval $[0, 1]$. In our experiments, we used:

$$b(x) = \min\left(\max\left(x, 0\right), 1\right). \tag{4}$$

The energy function in Eq. 3 can be efficiently minimized by introducing an additional variable v and computing the following iteration scheme [3]:

$$u^{(k+1)} = v^{(k)} - \frac{1}{\lambda} \nabla \cdot p \tag{5}$$

$$v^{(k+1)} = \min\left(\max\left(u^{(k+1)} - \frac{\beta}{\lambda} r, 0\right), 1\right), \tag{6}$$

where $\nabla \cdot$ denotes the divergence operator, $p(\mathbf{x})$ can be efficiently calculated using the following fixed point algorithm [2]:

$$p^{(0)} = (0, 0) \tag{7}$$

$$p^{(l+1)} = \frac{p^{(l)} + \delta t \, \nabla(\nabla \cdot p^{(l)} - \lambda v^{(k)})}{1 + \delta t \left| \nabla(\nabla \cdot p^{(l)} - \lambda v^{(k)}) \right|}, \tag{8}$$

Algorithm 2. Geometric active contours

Input: a grayscale image $f(\mathbf{x})$, expected foreground and background intensities μ_1 and μ_2, regularization parameters λ and β
Output: a binary image $u(\mathbf{x})$ representing the segmentation
Lines starting with '#' indicate comments

Initialization
$u(\mathbf{x}) = 0$, $v(\mathbf{x}) = 0$
$r(\mathbf{x}) = (\mu_1 - f(\mathbf{x}))^2 - (\mu_2 - f(\mathbf{x}))^2$
repeat
    # Estimate p
    $p(\mathbf{x}) = 0$
    repeat
        $divp(\mathbf{x}) = \nabla \cdot p(\mathbf{x})$
        $graddivp(\mathbf{x}) = \nabla(divp(\mathbf{x}) - \lambda v(\mathbf{x}))$
        $p(\mathbf{x}) = \dfrac{p(\mathbf{x}) + \delta t\, graddivp(\mathbf{x})}{1 + \delta t\, |graddivp(\mathbf{x})|}$
    until convergence criterium for $p(\mathbf{x})$ is satisfied
    # Update u
    $u(\mathbf{x}) = v(\mathbf{x}) - \frac{1}{\lambda} divp(\mathbf{x})$
    # Update v
    $v(\mathbf{x}) = b\left(u(\mathbf{x}) - \frac{\beta}{\lambda} r(\mathbf{x})\right)$
until convergence criterium for $u(\mathbf{x})$ is satisfied
Binarization
$u(\mathbf{x}) = u(\mathbf{x}) > 0.5$

where δt is the step size. The pseudocode of this technique is shown in Algorithm 2 and Fig. 2 shows the result of geometric active contours applied on a fluorescence microscopy image.

Active contours are a more complex segmentation method and, due to the iterative implementation, significantly more computationally intensive.

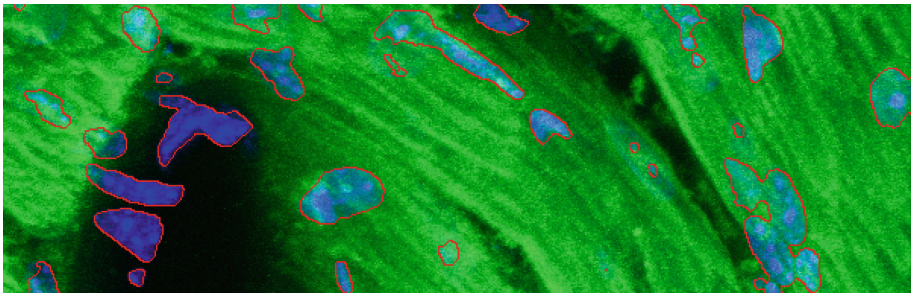


Fig. 2. Active contour segmentation result on the blue channel of a fluorescence micrograph. (Color figure online)

However, because of the high amount of pixel-wise image operations, parallel computing seems computationally interesting.

4 Accelerating Programs Using the GPU

Accelerating SLIC superpixel segmentation and active contour algorithms using the GPU has been studied in literature [7,16]. We stress that, in this paper, it is our goal to identify the general algorithmic and data-dependent properties that give rise to a higher potential speedup, such that GPU porting can be performed whenever it is likely to escribe. Firstly, it is worth noticing we can distinguish between two types of programs that have different acceleration properties as more computing resources are provided to the system: strongly and weakly scaled programs.

4.1 Strong and Weak Scaling

Intuitively, more computing resources result in faster computation, for programs assuming the workload remains constant (the program will not benefit in performance by increasing the workload). This type of programs is typically called *strongly* scaled, e.g. matrix operations such as addition, multiplication, etc. More specifically, the theoretically achievable speedup s by providing n times more computing resources to a subprogram that is responsible for a fraction q of the total (original) execution time is then given by Amdahl's law:

$$s = \frac{1}{1 - q + \frac{q}{n}}. \tag{9}$$

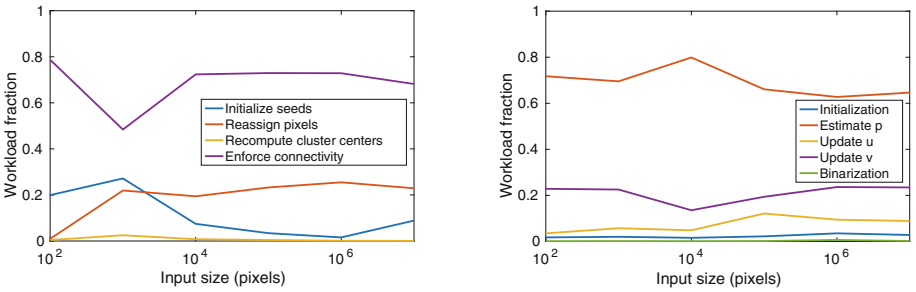
As $n \to +\infty$, the achievable speedup will be maximized to $\frac{1}{1-q}$. However, even in this case, a subprogram that requires a relatively small fraction of computing time ($q \to 0$) will still result in an insignificant global speedup. Clearly, in order to guarantee a high potential speedup, it is important to initially detect the subprograms that are responsible for the largest fraction q of computing time and focus on these parts of the program for parallelization.

Alternatively, increasing the workload may benefit the performance of an algorithm. This type of programs is also called *weakly* scaled, e.g. training-based algorithms. The fixed workload assumption is invalid and the theoretically achievable speedup s by providing n times more computing resources to a subprogram that is responsible for a fraction q of the total (original) execution time is then given by Gustafson's law:

$$s = 1 + (n - 1)q. \tag{10}$$

In this case, the achievable speedup is linearly related to the amount of computing resources and subprogram execution time fraction.

In general a program neither exhibits strong nor weak scalability, but rather a combination of both. The key message in the context of GPU processing is



(a) SLIC superpixels (b) Geometric active contours

Fig. 3. Execution time percentage of parts of the (a) SLIC superpixel and (b) active contour algorithm for variable input sizes (note that this axis is logarithmically scaled). The indicated subprograms are shown in comment in their corresponding pseudocode (Algorithms 1 and 2, respectively).

to parallelize the subprograms that are responsible for the largest fraction of computing time.

Algorithms 1 and 2 show the pseudocode of the discussed SLIC superpixel and geometric active contour algorithm, respectively. We have separated the algorithms in subprograms, indicated by the commented lines. Figure 3 shows the workload fraction of each part of the algorithms for variable input sizes. The most interesting function to parallelize in the SLIC algorithm is the connectivity enforcement. In this example, we have chosen for 10 iterations in the algorithm (which typically suffices). Obviously, a higher number of iterations will result into relatively more computing time reassigning the pixels and recomputing the cluster centers. The active contours algorithm spends most of the computing time in the estimation of p (fortunately, we typically have convergence after 1 iteration). As a consequence, this part of the algorithm is essential to parallelize in order to guarantee a higher speedup.

4.2 Estimating the Achievable Speedup

Figure 4 illustrates the achieved speedup by GPU acceleration of SLIC superpixel and active contour segmentation applied on a 0.001, 0.1 and 10 megapixel 8-bit grayscale image using Quasar. An important notice is that Quasar is designed for program execution on heterogeneous hardware and will decide at runtime whether to run a function on the host CPU or another specific device (usually a GPU) according to its own heuristics. The experiments were performed using an Intel Core i7 4720 2.60 GHz CPU and GeForce GTX 960M GPU.

Once the most time-requiring functions are detected, it is important to analyze their characteristics and the type of data that they will have to process. We provide a (non-exhaustive) list of properties that, according to our experiences, significantly impact the achievable speedup and illustrate them with examples of subprograms of the accelerated segmentation algorithms (see Fig. 4):

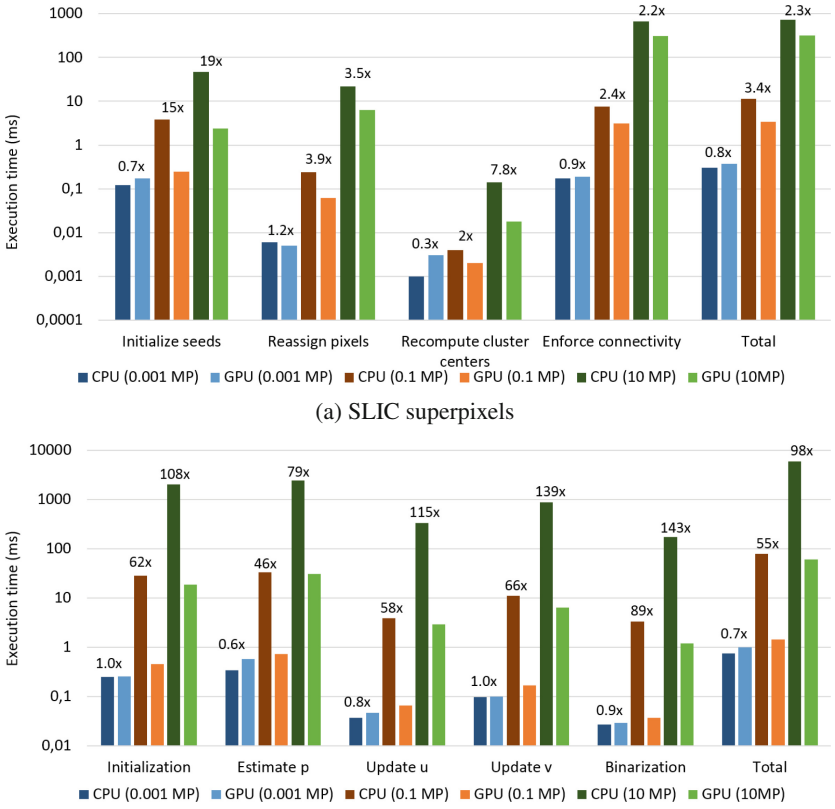


Fig. 4. Execution timing distribution (in ms) of the (a) SLIC superpixel and (b) geometric active contours algorithm applied on 0.001, 0.1 and 10 megapixel 8-bit images using multi threaded CPU and GPU processing. For each part of the program, the achieved speedup is indicated on top of the bars. Note that the execution time axis is logarithmically scaled.

- **Coarse-grained parallelism of the function:** Functions that consist of many computations that are mutually independent are typically called functions with coarse-grained parallelism. This naturally translates to parallel computing and is therefore an indicator for a high or low potential speedup. For example, all the functions in the active contour algorithm are pixel-wise image operations resulting in significant speedups.
- **Size of the data to process:** This is a consequence of concealing memory latency while accessing it. Larger amounts of data typically allow more operations to be performed in parallel, resulting in higher speedups, compared to small amounts of data. For example, we establish higher speedups in the initialization of the seeds and the cluster center recomputing of the SLIC algorithm. However, since the amount of superpixels is usually much smaller compared to

the number of pixels, and the pixel reassigning and cluster center recomputation iterate over the cluster centers, their corresponding speedup is relatively smaller. The active contours computation requires sufficiently enough input data in order to efficiently execute a number of iterations and conceal memory latency. This can be seen by the attained speedups as the input data size increases.

– **Computational complexity:** The operational complexity should justify the cost of transferring data to and from the device, i.e. maximize the computation/memory transfer ratio. Note that in many cases, data remains on the GPU memory: in this case global memory reads/writes should be used in order to compute the computation/transfer ratio. As an example, assume two $N \times N$ matrices have to be added using the GPU: this requires N^2 operations and $3N^2$ data reads/writes to global memory, resulting in a computation/transfer ratio of $\frac{1}{3}$. Alternatively, the case of matrix multiplication would require N^3 operations and $3N^2$ elements to be transferred, resulting in a computation/transfer ratio of $\frac{N}{3}$. In this case, the algorithm would benefit from larger matrix sizes. Similarly, we denote in the parts of the active contour algorithm where u and v are being updated, that the update for v is computationally (slightly) more intense than the update on u. Hence, the corresponding speedups are increasing faster as the input data size increases.

In practice, the achievable speedup is determined by a combination of the previous and other functional and data-dependent properties. Even under perfect function and data circumstances, the achievable speedup may still rely on implementation-dependent factors such as memory transfer, data type and alignment, thread divergence, etc. Nevertheless, the coarse-grained parallelism, data size and computation/memory transfer ratio give a good indication whether an algorithm is suitable to GPU acceleration.

5 Conclusion

Image segmentation remains one of the most challenging problems in microscopy analysis. Typically, the user is obligated to find an optimal balance between algorithm complexity on the one hand, which is heavily correlated with computational complexity, and manual post-processing on the other hand. High computational costs are a common reason for the impracticality of many high-quality segmentation algorithms and can be mitigated through recent developments in GPU accelerated computing. However, accelerating algorithms using the GPU is a costly operation because of the required programming expertise. Additionally, predicting the attainable speedup is difficult in practice because of the large amount of influencing factors. In this paper, we identify which algorithm and data characteristics significantly relate to the achievable GPU speedup. In particular, we have found that (1) the algorithm needs to exhibit coarse-grained parallelism, (2) the data size needs to be sufficiently large to benefit from GPU parallelization and (3) the computation/memory transfer ratio needs to sufficiently large so that the computational cost outweigh the memory transfer cost.

This is illustrated on two types of well-known segmentation methods that are extensively used in microscopy image analysis: superpixels and active contours.

References

1. Achanta, R., Shaji, A., Smith, K., Lucchi, A.: SLIC superpixels compared to state-of-the-art superpixel methods. IEEE Trans. Pattern Anal. Mach. Intell. **34**(11), 2274–2281 (2012)
2. Aujol, J.F., Chambolle, A.: Dual norms and image decomposition models. Int. J. Comput. Vision **63**(1), 85–104 (2005)
3. Bresson, X., Esedoglu, S., Vandergheynst, P., Thiran, J.P., Osher, S.: Fast global minimization of the active contour/snake model. J. Math. Imaging Vision **28**(2), 151–167 (2007)
4. Chan, T.F., Esedoglu, S., Nikolova, M.: Algorithms for finding global minimizers of image segmentation and denoising models. SIAM J. Appl. Math. **66**(5), 1632–1648 (2006)
5. Crookes, D., Miller, P., Gribben, H., Gillan, C., McCaughey, D.: GPU Implementation of MAP-MRF for microscopy imagery segmentation. In: Proceedings - 2009 IEEE International Symposium on Biomedical Imaging: From Nano to Macro, ISBI 2009, pp. 526–529 (2009)
6. Goossens, B., De Vylder, J., Philips, W.: Quasar: a new heterogeneous programming framework for image and video processing algorithms on CPU and GPU. In: Proceedings of the IEEE International Conference on Image Processing, pp. 2183–2185 (2014)
7. He, Z., Kuester, F.: GPU-based active contour segmentation using gradient vector flow. In: Bebis, G., et al. (eds.) ISVC 2006. LNCS, vol. 4291, pp. 191–201. Springer, Heidelberg (2006). doi:10.1007/11919476_20
8. Helmstaedter, M., Briggman, K.L., Turaga, S.C., Jain, V., Seung, H.S., Denk, W.: Connectomic reconstruction of the inner plexiform layer in the mouse retina. Nature **500**(7461), 168–174 (2013). http://www.ncbi.nlm.nih.gov/pubmed/23925239
9. Holk, E., Pathirage, M., Chauhan, A., Lumsdaine, A., Matsakis, N.D.: GPU programming in rust: implementing high-level abstractions in a systems-level language. In: Proceedings - IEEE 27th International Parallel and Distributed Processing Symposium Workshops and PhD Forum, IPDPSW 2013, pp. 315–324 (2013)
10. Huang, Y., Liu, Z.: Segmentation and tracking of lymphocytes based on modified active contour models in phase contrast microscopy images. Comput. Math. Methods Med. **2015**, 1–9 (2015)
11. Jain, V., Turaga, S.C., Briggman, K.L., Helmstaedter, M.N., Denk, W., Seung, H.S.: Learning to agglomerate superpixel hierarchies. In: Advances in Neural Information Processing Systems, pp. 1–9 (2011)
12. Jorstad, A., Fua, P.: Refining mitochondria segmentation in electron microscopy imagery with active surfaces. In: Agapito, L., Bronstein, M.M., Rother, C. (eds.) ECCV 2014. LNCS, vol. 8928, pp. 367–379. Springer, Heidelberg (2015). doi:10.1007/978-3-319-16220-1_26
13. Lucchi, A., Smith, K., Achanta, R., Knott, G., Fua, P.: Supervoxel-based segmentation of mitochondria in EM image stacks with learned shape features. IEEE Trans. Med. Imaging **31**, 474–486 (2012)

14. Meziou, L., Histace, A., Precioso, F., Matuszewski, B.J., Murphy, M.F.: Confocal microscopy segmentation using active contour based on alpha (α)-divergence. In: Proceedings of the International Conference on Image Processing, pp. 3077–3080 (2011)
15. Ragan-Kelley, J., Adams, A., Paris, S., Durand, F., Barnes, C., Amarasinghe, S.: Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In: Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation, pp. 519–530 (2013)
16. Ren, C.Y., Reid, I.: gSLIC: a real-time implementation of SLIC superpixel segmentation. University of Oxford, Department of Engineering Science, pp. 1–6 (2011)
17. Stegmaier, J., Amat, F., Lemon, W.C., McDole, K., Wan, Y., Teodoro, G., Mikut, R., Keller, P.J.: Real-time three-dimensional cell segmentation in large-scale microscopy data of developing embryos. Dev. Cell **36**(2), 225–240 (2016)
18. Wang, S., Cao, G., Wei, B., Yin, Y., Yang, G., Li, C.: Hierarchical level features based trainable segmentation for electron microscopy images. BioMed. Eng. OnLine **12**, 59 (2013). http://www.biomedical-engineering-online.com/content/12/1/59