Segmentation of Colon Tissue Sample Images Using Multiple Graphics Accelerators

Sándor Szénási[a]

[a] *John von Neumann Faculty of Informatics, Óbuda University, 96/B Bécsi út, H-1034 Budapest, Hungary*

Abstract

Nowadays, processing medical images is increasingly done through using digital imagery and custom software solutions. The distributed algorithm presented in this paper is used to detect special tissue parts, the nuclei on haematoxylin and eosin stained colon tissue sample images. The main aim of this work is the development of a new data-parallel region growing algorithm that can be implemented even in an environment using multiple video accelerators. This new method has three levels of parallelism: a) the parallel region growing itself b) starting more region growing in the device c) using more than one accelerator. We use the split-and-merge technique based on our already existing data-parallel cell nuclei segmentation algorithm extended with a fast, backtracking-based, non-overlapping cell filter method. This extension does not cause significant degradation of the accuracy; the results are practically the same as those of the original sequential region growing method. However, as expected, using more devices usually means less time is needed to process the tissue image; in the case of the configuration of one central processing unit and two graphics cards, the average speed-up is about 4–6X. The implemented algorithm has the additional advantage of efficiently processing very large images with high memory requirements.

Keywords: Medical image segmentation, Cell nuclei detection, Data parallel algorithm, Distributed algorithm, GPGPU, CUDA
PACS: 42.30.Tz, 87.57.nm
2000 MSC: 65D18

URL: E-mail: szenasi.sandor@nik.uni-obuda.hu / Phone num.: +361666551 (Sándor Szénási)

1. Introduction

Nowadays, digital microscopes are becoming increasingly popular among pathologists. The processing of microscopic tissue images and the segmentation of tissue components are now done through digital imagery and special immunodiagnostic software products [1]. These are fast and accurate products and can serve several additional functions, like remote access, archiving [2, 3], searching and tagging [4], semi-automatic diagnostics [5, 6, 7], registration [8], computer-aided tissue engineering [9] etc. This kind of processing offers a very promising way of using different segmentation techniques with the images received; this way, the different components of the tissues can be separated effectively. Appropriately, precise recognition of the tissue components would provide a safe background for automated status analysis of the examined patients, or at least promote the work of pathologists with this pre-processing.

Our work focuses on the segmentation of images containing haematoxylin and eosin (HE) stained colon tissue samples. There are several procedures to identify the main structures in these images and many are based on a reliable cell nuclei detection method. There are several image processing algorithms for this purpose [10, 11, 12, 13], but some factors could increase the challenge. The size of the images can easily reach 100 megabytes; therefore, the image processing speed plays an important factor.

In this paper, after the presentation of the technical background (related work, evaluation method, etc.), we propose a new cell nuclei segmentation algorithm implemented in a heterogeneous environment. This method uses all the available GPUs of the system for the most computationally intensive tasks (data-parallel cell nuclei segmentation), and all the available CPU cores for the less computationally intensive additional tasks (splitting and merging images, and controlling the GPUs).

2. Evaluation of cell nuclei detection methods

2.1. Accuracy of nuclei segmentation algorithms

For comparison, we have to evaluate the accuracy of the different algorithms. We have 39 colon tissue sample images manually annotated by qualified pathologists (we will refer to these as the Gold Standard slides), therefore we can compare the outputs of the algorithms to these results. There are several available evaluation methods for this purpose, but most of them are not suitable for this task, therefore we designed a new methodology. We have to know the exact position and shape of cell nuclei for further diagnosis purposes, therefore the basic object-level comparison methods are not applicable (for example, just compare the number of cell nuclei, etc.); we need a pixel-level comparison method. The widely used confusion matrix gives very clear and easily understandable results, based on Equation 1.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

where

- TP: Number of true-positive pixels (the pixel is correctly classified as part of a nucleus in both the reference result set and in the test result set).
- TN: Number of true-negative pixels (the pixel is correctly classified as not part of a cell nucleus).

- FP: Number of false-positive pixels (in the test result, the pixel is classified as part of a nucleus, but in the reference result it is not).
- FN: Number of false-negative pixels (the pixel is incorrectly classified as not part of a cell nucleus).

However, this pixel-level evaluation itself will not give us perfect results, because during the segmentation our task is not only to determine whether a pixel belongs to a nucleus or not, we have to identify the closed nuclei objects themselves. For example, in the case of false-negative hits, the pixel-level evaluation cannot indicate how many nuclei the algorithm misses (for diagnostic purposes, it really matters whether we miss only one big nucleus, or a lot of small nuclei). Another problem can be when there are several small nuclei in the reference slide, but the algorithm identifies them as one large nucleus. In this case, the pixel-level comparison indicates relatively small errors; however, this can be very important information for detecting malicious cells.

Our specialized measurement number is not based only on the pixel-by-pixel comparison; instead it starts by matching the cell nuclei together in the reference and the test results. One cell nucleus from the reference result set can only have one matching cell nucleus in the test result set and vice versa. After matching the cell nuclei, we can compare the paired elements using the confusion matrix. There are some other improvements: for example, we use some weighting in the case of false-positive and false-negative pixels based on the distance from the nearest valid pixel, which is important for the appropriate results near the borders of the nuclei.

The implementation of this evaluation method raises several problems. The pairing of the test-reference nuclei is a very resource-consuming step (in the case of several overlapping nuclei, the number of valid pairings can be billions); therefore we use a backtracking-based method to find the optimal result [14]. In this paper, we will use this evaluation method for every task where we need to check the accuracy of the nuclei detection algorithm (evaluation of algorithms, testing, parameter optimizing, etc.).

2.2. Comparison of nuclei detection methods

The main purpose of these algorithms is the same: we have to select the pixels of the sample, which could belong to any nucleus. The first thought would be to use the colours of the pixels for this separation, but in practice, this causes many difficulties. In the case of a specific image, we can achieve good results because we can easily teach the program whether a given colour represents a nucleus pixel or not. However, our experiences show that the colours of the images pre-processed by different labs are significantly different. This problem can be solved with some profile files (one profile for each lab), since we can transform all images into a standardized colour space. However, in practice, it turned out that there are significant differences between results from the same laboratories as well. Even if the same tools and materials are used, a different amount of stain and processing time can cause different colours (in some cases, the nuclei are very strong dark areas, but in the case of some other images, these are significantly less contrasting).

There are various automatic threshold based techniques to solve this problem. Several papers deal with segmentations using the K-means procedure [15], which produces very quickly and with impressive results. The main limitation of this method is the insufficient accuracy [16]. Further options are the texture based methods [17] and colour clustering [18]. It is easy to achieve the quick results initially with moderate accuracy, but further development is generally

impossible. Nevertheless, it is worth considering these techniques as they are quite flexible in regards to various staining conditions. Therefore, these procedures can be used for fast pre-processing.

Region growing is a more sophisticated technique [19]. This is because we can exactly define and fine-tune the iteration steps by choosing an arbitrary fitness function and stopping condition. Both of these may consider the colour of the pixels, the environmental conditions, the size of the increased region, their position, etc. Another important advantage of the region growing approach is that it provides information not only about the individual pixels (whether a given pixel belongs to a cell nucleus or not), but it gives detailed information about the whole cell nuclei objects (the result of the region growing is a list of cell nuclei). This information is essential by itself for the diagnosis (number of nuclei, density of nuclei, etc.), and it is useful for the further segmentation of the image (glands, surface epithelium, etc.).

However, region growing has some disadvantages as well. First, the biggest problem is that this method is rather slow. The process is slow to the extent that practical use seems almost impossible, because the segmentation of large images (8192x8192 pixels size or even greater) containing a moderate number of nuclei may require up to one hour to complete. However, because the process offers good accuracy, it is definitely worth dealing with this drawback, though, we have tried to speed up the process as much as possible (without loss of accuracy). For the implementation, we use the graphics hardware, because it is used in similar projects with good results [20, 21].

3. Cell detection with data parallel region growing

3.1. Parallel region growing

The implemented region growing algorithm iterates the following three steps until one of the stopping conditions is met. Due to space limitations, this paper contains only a brief description of parallel region growing. Detailed introductions can be found in [22].

1. It checks the four possible directions in which the contour can be expanded. In case of the first iteration, this means the four neighbours of the starting point (seed point), in the latter iterations the pixels around the lastly accepted contour point (see below). We can check all directions at the same time; therefore, four threads examine the different neighbours, whether they are suitable for further expansion or not.

2. In the next step, all contour points are evaluated to decide the direction in which the region should be expanded. The algorithm evaluates a fitness function for every point. Unfortunately, some parameters of this fitness function change at the insertion of every new point (centre of the region, average intensity of the region, etc.). So, they have to be re-calculated in every iteration for every contour point. However, this is a well parallelizable process; every thread calculates the fitness of a single contour point and all threads execute the same code on a different data. This is an ideal data-parallel task and is executable in the GPGPU (General-Purpose Computing on Graphics Processing Units [23]).

3. At the end of each iteration, the algorithm selects the contour point with the highest fitness value. It can use the ”atomicMax” function of the GPGPU to make the threads calculate the highest fitness. This point will expand the region in the next iteration.

The iteration goes until the region reaches the maximum size, which is given by a parameter. After each iteration, a region level fitness function is also evaluated based on the intensity differences between the inner and outer contour of the region, and the circularity of the candidate region. Finally, the result of the region growing function is the state where the maximum fitness was reached (after each iteration it has stored the actual region level fitness value in an array).

3.2. GPGPU implementation

As visible in the pseudo code (see Algorithm 1), the number of threads is equal to the size of the contour. This number is usually less than 500; therefore, if we want to utilize the whole processing power of the GPGPU, we need to start more threads. This is possible if we use a higher level of parallelism, and execute more than one region growing at the same time (in separate GPU blocks). The adjacent seed points can cause problems, since the parallelized search of those can result in overlapping cell nuclei, which is unacceptable. Fortunately, the maximum radius of a cell nucleus is known (it is an input parameter of the search procedure, called R_{MAX}). Hence, we can presume that searches started from two or more seed points (where distances between these points are more than $4 * R_{MAX}$) can be considered as independent searches and can be launched in a parallelized way. Using this technique, we can use thousands of threads, leading to a very high GPU performance.

The results of this new implementation are practically the same as for the original one, therefore the accuracy is the same. We ran several tests in different images using the CPU implementation (Intel Core-i5 2400 processor, with four cores) and the GPU implementation (Gigabyte GTX580, with 480 cores). The third and fourth columns of Table 1 show the run-times. These depend heavily on the attributes of the tissue sample image (size, number of nuclei, etc.), therefore it would be better to use a relative measurement unit. The unit we used was that we calculated how long it took relatively to process one pixel of the image. The next two columns show these relative values, and the last column contains the difference between the CPU and the GPU run-times. As can be seen, the data-parallel version is usually two to three times faster than the original one. As it is also visible, the GPU implementation gives better results for large images. When using images of 1024x1024, 2048x2048 and 4096x4096 pixels, the average relative run-times are 0.0154/0.009/0.008 ms/pixel, respectively; meanwhile, the same values with the CPU implementation are 0.025/0.034/0.029 ms/pixel, respectively.

3.3. Parameter optimization

The region growing algorithm prepared this way has several parameters, and it is very sensitive to the appropriate settings. Fine-tuning of these is as important as the previously mentioned speed increase. We have 27 mutually independent parameters (details of the different pre-processing image filters, maximal contrast length and cell nucleus candidate size, parameters of intensity contrast dimensions, etc.) with predefined target sets, and our aim is to search for a set of parameters that gives the best possible accuracy. Due to the large number of parameters and their reasonably large target set, defining the values manually seems hopeless, so we have developed an evolutionary algorithm to find the optimal values. This evolution-based algorithm was used to successfully determine a set of significantly better parameters than the manually adjusted ones.

Briefly, the main attributes of the genetic algorithm used were:

- Representation of chromosomes: We stored 27 genes inside a chromosome, where every gene actually represents one of the region growing parameters. Every parameter is

separately encoded and the functionality of the different genes does not depend on their location inside the chromosome.

- Initial generation of genetic algorithm: The first generation was generated by randomly generated chromosomes. Some of the parameters naturally have upper and lower bounds (cell nuclei size, cell nuclei radius, cell nuclei circularity, average intensity, etc.). To determine these limits, we have done some statistical analysis on the annotated cell nuclei of the already presented Gold Standard slides. For example, regarding the sizes, 99.5% of the cell nuclei are between 34 and 882 pixels. Based on these results, the actual values of genes were chosen using Gaussian distribution within these intervals. Some of the parameters are downright technical; in these cases we use pure random numbers between the technically feasible intervals.

- Implementing selection operator: For parent selection, we used the well-known roulette wheel method where every chromosome has a slot that is sized proportionally according to its fitness value, based on Equation 2 (where P_i is the probability of selecting the i^{th} chromosome, F_k is the fitness value for the k^{th} chromosome, and $Min(F)$ is the smallest fitness value for the generation).

$$P_i = \frac{F_i - Min(F)}{\sum_k (F_k - Min(F))} \tag{2}$$

- Implementing crossover operation: We used uniform crossover, where we combine whole genes from the previously selected parents. For every gene, a random number determines which parent's gene is inherited. Genes from the parent with higher fitness value have a proportionally higher probability of being inherited.

- Implementing mutation operator: The probability of a mutation is 10%, and the size of the mutation can be small, medium-sized or large (with a 60%, 30% or 10% chance, respectively). The exact values of these mutations cannot be defined in a general form, because the values of the parameters are very different. Therefore every parameter has its own mutation range.

We started the genetic algorithm with 3000 chromosomes in the initial generation, and 300 chromosomes in every following generation. Every parameter set was tested against 11 representative image tissues. After 440 generations (this took approx. 3948 working hours), the system cannot give better results, therefore we stopped the search. The best accuracy reached at 83.6%, which is significantly better than the accuracy of the already existing (manually optimized) parameter set (78.1%) [24, 25].

4. Using multiple GPGPUs

4.1. Naive implementation

To gain maximal performance, it would be better to use more than one graphics card [26]. We have developed two protocols for multi-GPGPU operations. The first is based on the requirement that we need the same result as with the single GPGPU version. This is possible when the main process itself remains unchanged. Only the previously mentioned independent parts are expanded, so that they are processed at the same time by not only one but all GPUs.

This raises many problems. Fundamentally, this is because the GPGPUs work in independent memory areas. The trivial solution for this problem would be a full synchronization after every processing step, which makes a full copy between the independent memory areas. The naive implementation works as follows:

1. The seed point search algorithm searches for the points from which the region growings can be started.
2. Because the region growings that start from these points are completely independent, they can be distributed in any way among the available GPGPUs.
3. We can start as many blocks in each GPGPU as the number of seed points that the given GPGPU has.
4. All GPGPUs copy the processed memory regions into the global memory. Thanks to the seed point selection, all nuclei currently found are quite far from each other. Therefore, there are not any overlapping areas in the different GPGPUs. Thus, we do not expect any memory transfer conflicts.
5. After each GPGPU is done, they have to wait on each other using a global synchronization.
6. All GPGPUs refresh their private memory region using the data from the global memory.
7. Then, the next iteration can be started.

This method is easy to implement, but the effectiveness raises a number of questions. The biggest problem is that quite a large amount of data exchange is necessary. This is because all of the GPGPUs store the whole image, so that during the update process, all of them have to transfer all changed data from the other devices. The copy itself cannot be solved efficiently, because the detected nuclei candidates are not placed in a single contiguous area (and the shapes of these nuclei are unspecified). Therefore, the best solution is to copy lists of detected points (additionally, it makes it more difficult that the region growing use several auxiliary images, so we have to copy more bytes in case of one pixel).

This method is easy to implement; however due to the large number of memory operations, it is expected to be correspondingly much slower than the next version (details can be found in our previous article [27]).

4.2. Split-and-merge method

To achieve the fastest possible solution, we have to provide as great independence for the GPGPUs as possible. To this end, another option may be to simply divide the whole image into smaller slices (tiles) and to distribute these between the devices. Once processing is complete, we have to concatenate the results. The well-known name of this procedure is the split-and-merge method [28]. There may be problems near the edges of images, but we can use several techniques to manage the overlaying parts.

We should ensure that the region growings started at the edge of the tiles do not affect the results of the region growings started in different GPGPUs. Fortunately, this is easily met because we know the maximum radius of any cell nucleus (R_{MAX}). To determine this value, we can use the annotated cell nuclei of the Gold Standard images. We have done this earlier, in the phase of preparing initial generation for the parameter optimizing genetic algorithm. Based on the

processing of 7889, we set the value of R_{MAX} to 30 pixels (in fact, there was one larger nucleus, whose radius was 31.9 pixels, but using this parameter value increases the number of false-positive hits, therefore it is better to use the 30 pixels limit).

Hence, any two region growings can be started in parallel if the distance between the seed points is at least $4 * R_{MAX}$. Therefore we only have to split the entire image into smaller sub-images (as large as acceptable for the GPGPUs) using $4 * R_{MAX}$ pixels wide overlapping areas. This area will be important later in the merge phase, as this can ensure that no cell nuclei start from non-overlapping areas of one GPGPU that has pixels in the non-overlapping area of another GPGPU.

The main steps of the algorithm:

1. Division of the whole image based on the previously mentioned overlapping technique and sending these image parts to the GPGPUs.

2. Execution of the original region growing algorithm in all GPGPUs (processing all acceptable seed points).

3. Copying the processed memory areas into the global memory. This is not an independent task for the GPGPUs, because the overlapping areas can contain overlapping cell nuclei (and based on the processing order of the GPGPUs, these overlapping results may be different).

4. Filtering of the cell nuclei candidates. In the next step we have to select some of the remaining nuclei candidates, based on the following rules: 1) there must not be any overlapping nuclei in the result, and 2) the accuracy of this result must be the maximal available accuracy.

The disadvantage of the designed algorithm is that the results may no longer be identical to the results provided by the traditional sequential algorithm. During the selection of non-overlapping cell nuclei, we have to reject certain nuclei candidates. Moreover, the remaining accepted candidates may differ from the sequential result. However, it is important to note that this does not mean that the new procedure has less precision. This only means that the result may be another acceptable solution.

However, this approach promises several advantages. For our purposes, the most important benefit is the expected speed-up of processing. There are no synchronization steps between the GPGPUs. Therefore, all devices can work at their peak power. There are some additional costs (more than one GPGPU process the same overlapping area and the cost of the merge phase); however, these are not significant.

Another benefit of this method (and that is the reason why we implement this), is the possibility to process the input image by smaller distinct parts. It is important because the high memory requirement is a huge disadvantage of the region growing. The full sized source images are quite large (more than 200MB by image), and during the processing several copies are needed (modified by some filters). This all leads to the inability to use traditional region growing for these full-sized images. There is not enough memory for the entire image and the copies (especially in the GPGPU memory). Compared to the previous procedure, this method can also be used to solve this problem. As the processing of each of the sub-images is completely independent, these are executable in any order (one-by-one or parallel). As a result, in case of large images and limited resources, region growing is also applicable. For all of these reasons, we have implemented this version of parallel region growing and have done several tests with this application.

5. Implementation of the split-and-merge method

The method itself consists of two main parts, the division and the merge. In the first step, we have to decompose the whole image into smaller parts (with some overlapping areas), and we can process these through the GPGPUs. In the next step, we have to prepare the final solution by merging the results of the separate smaller images.

5.1. The split phase

The split phase contains two steps: the cutting up phase, and the cell nuclei detection phase. The CPU processes the cutting up. The input is the whole tissue sample image and the output represents the overlapping parts. This is a simple operation; the only question is the size of the sub images and the width of the overlapping section.

The size of the tiles must be as large as possible. We have some hardware limitations because we have to store about ten copies of the image (several variations of the same image using filters for the seed point search, the region growing, score calculation, etc.). We need fast access to the pixels; therefore, we cannot use a compression method. Hence, we can easily calculate the memory requirements. We did several tests and the 2048x2048 pixels size looks like the best choice. It is not too large, so we can store the images in older graphics cards (with 1GB internal memory or less), but it is large enough for optimal processing (the ratio of the overlapping and the non-overlapping part is ideal). If we use smaller tiles, then we can use more parallel processing units, but in practice, we usually work with input images 8192x8192 pixels, only two GPGPUs, and one CPU. Therefore, the tile size mentioned above is acceptable.

The next required parameter is the width of the overlapped area (see Figure 1). It is worth choosing it as narrow as possible because this will cause less overlapping cell nuclei candidates. We know the maximum radius of a cell nucleus ($R_{MAX} = 30$ pixels) and it is obvious that two region growings that start from at least a $4 * R_{MAX}$ distance cannot have any effect on each other. Therefore, we use 120 pixel wide overlapping areas.

The CPU creates all the required tiles and save them as separate files (in the future we can optimize this to use memory to memory transfers). The next step is the nuclei detection for all images. For this, we have to distribute the available tiles between the processing units. We have implemented a scheduler based on the common producer-consumer pattern [29]. This pattern is used in order to handle multiple client requests simultaneously. There are two main roles: those that produce data and those that consume the data produced. Data queues are used to communicate between the participants. In the current phase, we have only one producer. But, in the future, we would like to improve the implemented system to create a fully distributed architecture - a dedicated cell nuclei detection cluster (consumers) where the client applications can send the input images (producers).

Technically, the implementation goes as follows: after loading the input tissue image, an algorithm starts to partition it into smaller tiles according to the above and puts these slides into a processing queue. In the meantime, several processes are running (one for all GPUs and one for the CPU) and waiting for the input in the queue. Each process gets one task from the queue and runs the already implemented region growing. The scheduler listens to the task completion signals and when each is finished, it starts the next step - the merging of the intermediate results.

5.2. The merge phase

In the merge phase, we have to merge the intermediate results into the final result. The input is the set of the tiles and lists containing the detected cell nuclei candidates. There can be nuclei

in the overlapping area; therefore, we cannot merge the images directly. We have to handle all nuclei as separate objects and make a decision to keep the given nucleus or reject it. The rules of the merge algorithm are the following:

1. Processing the non-overlapping regions: We can accept those nuclei whose enlargement did not start from an overlapping area, without any further checks. It is obvious that the distance of the seed points must be at least $4 * R_{MAX}$ and therefore these nuclei cannot meet in the most unfortunate cases. So these nuclei can be handled as independent from the others. Hence, these are immediately acceptable (see "Type a" and "Type b" in Figure 1).

2. Processing overlapping regions: We need some further checks in these cases. Usually there are several nuclei which are in the overlapping area but after the nuclei detection these are not overlapped with others from other tiles (see "Type c" in Figure 1). In these cases, we can simply accept these nuclei. However, we need some additional calculations if there are some nuclei candidates from different tiles overlapping each other (see "Type d" in Figure 1). If the pixels of these nuclei are perfectly identical (this is the ideal case), then we have to accept one of them and reject the other. Unfortunately, in most cases these nuclei partially overlap each other and in these cases we have to decide which one to accept and which one to reject. This decision is complicated by the fact that in several cases we have not just two but three or more overlapping nuclei. In practice, these nuclei often form a long chain and it is difficult to decide which ones to accept.

In the latter case, there can be thousands of overlapping nuclei. Therefore, we have implemented a backtracking based solution. First, we have to collect all overlapping nuclei and execute a clustering procedure to find the sets of nuclei in which all nuclei have some effect on each other. We can represent this problem as a graph where the vertexes represent the nuclei and there is an edge between two vertexes if the corresponding nuclei overlap each other. In this problem space, our task is a classical component search. There are several clustering techniques; we use a modified version of the well-known Kruskal algorithm [30]. In the first step, we create several sets where each vertex in the graph is in a separate set. After this, we start a loop and in each iteration we look at the edges continuously. If there is any edge connecting two different components, we merge them. After each iteration, the remaining components represent sets of cell nuclei in which there is a way across the overlapping sections between every nucleus.

After that, we have to sort out one set of nuclei from each cluster in which there are not any overlapping cell nuclei and the accuracy of the set is maximal, where the accuracy of the set is the aggregate value of the accuracy of all nuclei in the set (see Figure 2).

We have designed a score function to evaluate the nuclei candidates. The evaluation is based on the size (in pixels), radius, and circularity of the nucleus. To determine the appropriate weighting factors, we do an examination of the size, radius, and circularity of all nuclei in the already presented Gold Standard slides (colon tissue sample images manually annotated by pathologists). We have examined all nuclei in these images and calculated the values of size (Equation 3), radius (Equation 4) and circularity (Equation 5)) for each, and have drawn the distribution into 100 equally sized intervals where every interval has the following values:

$$T_{size}[i] = \frac{number\ of\ nuclei\ in\ the\ i^{th}\ size\ interval}{number\ of\ nuclei} \tag{3}$$

$$T_{radius}[i] = \frac{number\ of\ nuclei\ in\ the\ i^{th}\ radius\ interval}{number\ of\ nuclei} \quad (4)$$

$$T_{circularity}[i] = \frac{number\ of\ nuclei\ in\ the\ i^{th}\ circularity\ interval}{number\ of\ nuclei} \quad (5)$$

As expected, these values confirm the Gaussian distribution. We assume that the detected objects are as more like nucleus as many other nuclei exist with similar parameters. Therefore, we calculate this probability according to the following (Equation 6) (in the future we will replace this method with a fuzzy [31] based one):

$$Score(X) = W_{size} * T_{size}[X_{size}] + W_{radius} * T_{radius}[X_{radius}] + W_{circularity} * T_{circularity}[X_{circularity}] \quad (6)$$

where

- X:Nucleus candidate.
- $W_{size}, W_{radius}, W_{circularity}$: Weighting factors for size, radius and circularity.
- $T_{size}, T_{radius}, T_{circularity}$: Density tables for size, radius and circularity.
- $X_{size}, X_{radius}, X_{circularity}$: Size, radius and circularity of nucleus X.

We can use this score value to evaluate one nucleus and the sum of these score values to evaluate a set of nuclei. Our task is to find the best non-overlapping subset of the whole nuclei candidate set, where this summarized score value is maximal.

We have developed an algorithm based on the backtracking method [32] to find the best subset of non-overlapping nuclei. The number of subtasks equals the number of cell nuclei candidates in the examined cluster. Every subtask represents the decision that the corresponding nucleus is accepted or rejected. The backtrack search examines all potential solutions quite efficiently, and relies upon these search to select the combination of nuclei with the largest aggregate accuracy (see Algorithm 2).

In the case of overlapping nuclei, the outcome is always the best combination. We merge these sets with the already accepted nuclei and this leads to the final result of the whole merge task.

It is worth noting that in the clustering phase, we create several clusters of cell nuclei for further processing. There are not any overlapping cells in two different clusters. Therefore, we can consider these as independent groups. Thus, we can do the non-overlapping cell nuclei selection parallel for each group. It is hard to create a backtracking algorithm for the GPU; therefore, we have only a CPU implementation. However, based on the independence of the clusters, we can use the multi-core capability of the CPU to process more than one cluster at the same time. This can easily decrease the required processing time.

6. Evaluation of the algorithm

6.1. Accuracy test

Using the split-and-merge method does not always give the same result as the normal sequential region growing, but this does not necessarily mean that it is less accurate. We can use the already mentioned Gold Standard images to check the accuracy. Unfortunately, we do not

have any high-resolution annotated samples; therefore, we use smaller images with modified parameters: we set the tile size smaller than the recommended size (1024x1024 pixels instead the suggested 2048x2048 pixels). The widths of the overlapped regions are all the same (120 pixels).

In these tests, we use one CPU core to measure the "original accuracy" and one CPU + two GPUs to measure the split-and-merge accuracy. We have used the evaluation method described in our previous paper [16]. The results indicate that the new technique does not cause significant degradation of the accuracy (Table 2).

We do not have any large annotated images; therefore, we cannot measure the absolute value of accuracy in these cases. However, we can use the results of the already existing sequential CPU based application (verified by several tests and used by real-world applications for several years). As it is clearly visible in Table 3, the differences between the results are not significant and are usually less than 2%. It is worth noting that this does not mean lower accuracy. Sometimes the multi-GPU algorithm gives better results.

It has been noted that the split-and-merge method has only one limitation: we cannot guarantee that the result of this new algorithm will be exactly the same as the original sequential one. However, that posterior tests have good results, the difference is not significant and the speed-up is very impressive.

6.2. Speed test

Developing GPGPU codes is usually expensive and unconvincing; it is worth doing only if the GPGPU based application is spectacularly faster. We do several speed tests to check the speed-up and the results are very promising. The details of our runtime measuring method are as follows: we have done five independent tests on the given tissue images (as the standard deviations show the values are reliable). According to real-world conditions, we take into consideration all operations between the start and the end of the algorithm (load image from file, splitting, distribution, region growing, result reporting, merging). We use three devices in different configurations: CPU1 - Core-i5 2400 processor (4 cores, 3.1 GHz), GPU_1 and GPU_2 - Gigabyte GTX580 graphics cards (480 cores). It is hard to compare the speed of two algorithms running on different hardware, especially in this case since the CPU and GPU have drastically different architectures. We choose these two devices because (at the time of purchase) both represent the average quality/cost in their own market.

Table 3 also shows the detailed runtimes of the split-and-merge implementation using only CPU. The split part consists of the following: load the image from the hard disk, split it into smaller overlapping tiles, and save these onto the hard disk (most of the elapsed time is required by the file operations). The region growing part is the runtime of the already existing nuclei detection algorithm. The merge part consists of the following steps: loading the region growing result from the hard disk, collecting overlapping nuclei clusters, and finding the optimal subset of non-overlapping nuclei. As it is visible, the required additional time for the splitting and merging does not cause significant increases in the full execution time.

We have done some additional examinations, using multiple devices with the split-and-merge method. The tested configurations were as follows: only CPU, only GPU_1, CPU + GPU_1, GPU_1 + GPU_2, CPU + GPU_1 + GPU_2. We have run the image segmentation on 30 images of different resolutions (2048x2048, 4096x4096, and 8192x8192 pixels). The results (see Table 4) are promising. The processing time decreased drastically, and the CPU and the GPUs can work together very efficiently. Sometimes, in the case of 2048x2048 pixel size images, the CPU+GPU_1+GPU_2 version was slower than the GPU_1+GPU_2 version. However, this was only

caused by some scheduling problems: GPUs are more than twice as fast as the CPU; therefore, until CPU finished the segmentation of $Tile_1$, GPU_1 finished with $Tile_2$ and $Tile_4$, GPU_2 finished $Tile_3$, and the GPUs have to wait for the CPU.

As expected, configuration with more devices usually needs less time to process all the tiles. In the case of the GPUs, the speed-up is quite linear, which is not surprising because all the GPUs can work on the tiles independently. Therefore, these devices can use all of their processing power. This is not exactly true for the CPU, because it is slower than the GPUs. On the other hand, GPGPU implementations cause some load on the CPU too.

6.3. Tile size test

Our preliminary tests showed that the advantage of the GPU is greater in the case of large images. Therefore, we expect larger tile size to lead to better performance. To demonstrate this, we did some additional tests using different tile sizes (2048x2048 pixels and 4096x4096 pixels) on two full-sized tissue sample images (8192x8192 pixels). The tested configurations were the same as before: only CPU, only GPU_1, $CPU + GPU_1$, $GPU_1 + GPU_2$, $CPU + GPU_1 + GPU_2$. Table 5 shows the run-time values for both tile sizes, and the ratio of these values.

As can be seen, the CPU implementation was not able to benefit from the larger tile size, and it became even slower. In fact, the CPU code operates at the same speed, but some adverse circumstances degrade its performance. For example, there are a lot of empty areas in these full-sized images, and in the case of large tiles, the CPU has to rescan these areas (in the seed point search phase) more times than in the case of small tiles. Obviously, the GPUs also have to deal with this problem, but the seed point search is very parallelizable, therefore it is extremely fast in the graphics cards.

It can also be seen, that GPUs need less time to process the whole image using larger tiles. On the one hand, larger tiles mean more independently runnable region growing, fewer kernel launches, etc. On the other hand, the merge part is affected, because in the case of large tiles, the ratio of the overlapping areas is smaller. As expected, the GPU implementations are almost twice as fast in these cases. The best configurations are the 2GPU and the CPU+2GPU using the 2048x2048 pixels tile size (as was previously discussed, sometimes the CPU+2GPU version is a bit slower than the 2GPU version due to the unfortunate scheduling, but this is not general).

Summary

The aim of our research is developing a new data parallel region growing algorithm that can be implemented even in a GPGPU environment and which is capable of segmenting HE stained cell nuclei and identifying their exact locations and sizes. The new method has three levels of parallelization: 1) parallelization of the region growing method itself to use one thread for processing of each contour point, 2) starting more than one region growings in the GPGPU at the same time to utilize the processing power fully, and 3) using multiple GPGPUs based on the split-and-merge method.

As the results indicate, the split-and-merge technique does not cause significant degradation of the accuracy (the average difference between the accuracy of the original one-step processing method and the new method is about 0.03%). In the case of one CPU, the speed-loss caused by the split-and-merge technique is insignificant compared to the runtime of the original region growing.

The advantages of this technique are shown in the case of using multiple devices. As expected, a configuration with more devices usually needs less time to process all the tiles. In the case of the GPGPUs, this speed-up is quite linear, and additionally we can use the CPU for processing as well. Using the best configuration means about 6X speed-up (see Figure 3). Due to the lack of required hardware configurations, we cannot check configurations with more than two graphics cards, but it is expected that this trend continues for up to four GPGPUs (more GPUs can overload the CPU).

Another advantage of the new procedure is that we can process full sized colon tissue images. This was previously impossible due to the high memory requirement of the original region growing method. This is a very useful improvement for practical purposes.

The disadvantage of this technique is the special hardware requirement. Unfortunately, we cannot create a widely usable demo application because the system needs special hardware devices and Windows drivers (NVidia graphics cards with Fermi based cores). In practice, this is not an issue as these hardware elements are available for the end users.

References

[1] S. D. Olabarriaga, A. W. M. Smeulders, Interaction in the segmentation of medical images: A survey, Medical Image Analysis, vol. 5, no. 2, 2001, pp. 127–142.

[2] G. Placidi, Adaptive compression algorithm from projections: Application on medical greyscale images, Computers in Biology and Medicine, vol. 39, no. 11, 2009, pp. 993–999.

[3] P. Suapang, K. Dejhan, S. Yimmun, Medical image archiving, processing, analysis and communication system for teleradiology, in: IEEE Region 10 Conference (TENCON), 2010, pp. 339–345.

[4] A. Bogárdi-Mészöly, A. Rövid, H. Ishikawa, S. Yokoyama, Z. Vámossy, Tag and topic recommendation systems, Acta Polytechnica Hungarica, vol. 10, no. 6, 2013, pp. 171–191.

[5] M. Gurcan, J. Kong, O. Sertel, B. Cambazoglu, J. Saltz, U. Catalyurek, Computerized pathological image analysis for neuroblastoma prognosis, in: AMIA Annu Symp Proc, 2007, pp. 304–308.

[6] G. Valcz, I. Bándi, B. Wichmann, A. Patai, D. Szabó, G. Kiszler, M. Kozlovszky, B. Molnár, Z. Tulassay, Automated detection of epithelial changes in colorectal carcinoma, Zeitschrift für Gastroenterologie, vol. 50, no. 5, 2012, pp. 1–5.

[7] Z. Benyó, I. Benyó, M. Bolla, G. Lakatos, P. Nagy, L. Telegdi, J. Tick, Application of computational statistics to the investigation of tumors of the digestive system, in: Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 1991, pp. 1343–1344.

[8] A. Tanács, K. Palágyi, A. Kuba, Medical image registration based on interactively identified anatomical landmark points, Machine GRAPHICS & VISION, vol. 7, no. 1-2, 1998, pp. 151–158.

[9] W. Sun, P. Lal, Recent development on computer aided tissue engineering — a review, Computer Methods and Programs in Biomedicine, vol. 67, no. 2, 2002, pp. 85–103.

[10] M. Kowal, P. Filipczuk, A. Obuchowicz, J. Korbicz, R. Monczak, Computer-aided diagnosis of breast cancer based on fine needle biopsy microscopic images, Computers in Biology and Medicine, vol. 43, no. 10, 2013, pp. 1563–1572.

[11] M. El Adawy, Z. Shehab, H. Keshk, M. El Shourbagy, A fast algorithm for segmentation of microscopic cell images, in: ITI 4th International Conference on Information Communications Technology (ICICT), 2006, pp. 1–1.

[12] J. Hukkanen, A. Hategan, E. Sabo, I. Tabus, Segmentation of cell nuclei from histological images by ellipse fitting, in: 18th European Signal Processing Conference (EUSIPCO), 2010, pp. 1219–1223.

[13] R. Pohle, K. D. Toennies, Segmentation of medical images using adaptive region growing, in: M. Sonka, K. M. Hanson (Eds.), Medical Imaging 2001: Image Processing, Vol. 4322 of Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, 2001, pp. 1337–1346.

[14] S. Szénási, Z. Vámossy, M. Kozlovszky, Preparing initial population of genetic algorithm for region growing parameter optimization, in: 4th IEEE International Symposium on Logistics and Industrial Informatics (LINDI), 2012, pp. 47–54.

[15] K. Masood, N. Rajpoot, K. Rajpoot, H. Qureshi, Hyperspectral colon tissue classification using morphological analysis, in: International Conference on Emerging Technologies (ICET), 2006, pp. 735–741.

[16] S. Szénási, Z. Vámossy, M. Kozlovszky, Evaluation and comparison of cell nuclei detection algorithms, in: IEEE 16th International Conference on Intelligent Engineering Systems (INES), 2012, pp. 469–475.

[17] E. Gabriel, V. Venkatesan, S. Shah, Towards high performance cell segmentation in multispectral fine needle aspiration cytology of thyroid lesions, Computer Methods and Programs in Biomedicine, vol. 98, no. 3, 2010, pp. 231–240.

[18] S. Di Cataldo, E. Ficarra, A. Acquaviva, E. Macii, Automated segmentation of tissue images for computerized ihc analysis, Computer Methods and Programs in Biomedicine, vol. 100, no. 1, 2010, pp. 1–15.

[19] R. Adams, L. Bischof, Seeded region growing, IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 16, no. 6, 1994, pp. 641–647.

[20] O. Kutter, R. Shams, N. Navab, Visualization and GPU-accelerated simulation of medical ultrasound from CT images, Computer Methods and Programs in Biomedicine, vol. 94, no. 3, 2009, pp. 250–266.

[21] O. Fluck, C. Vetter, W. Wein, A. Kamen, B. Preim, R. Westermann, A survey of medical image registration on graphics hardware, Computer Methods and Programs in Biomedicine, vol. 104, no. 3, 2011, pp. 45–57.

[22] S. Szénási, Z. Vámossy, M. Kozlovszky, GPGPU-based data parallel region growing algorithm for cell nuclei detection, in: IEEE 12th International Symposium on Computational Intelligence and Informatics (CINTI), 2011, pp. 493–499.

[23] A. Eklund, P. Dufort, D. Forsberg, S. M. LaConte, Medical image processing on the GPU – past, present and future, Medical Image Analysis, vol. 17, no. 8, 2013, pp. 1073–1094.

[24] S. Szénási, Z. Vámossy, Implementation of a distributed genetic algorithm for parameter optimization in a cell nuclei detection project, Acta Polytechnica Hungarica, vol. 10, no. 4, 2013, pp. 89–86.

[25] S. Szénási, Z. Vámossy, Evolutionary algorithm for optimizing parameters of GPGPU-based image segmentation, Acta Polytechnica Hungarica, vol. 10, no. 5, 2013, pp. 7–28.

[26] T. D. R. Hartley, U. V. Çatalyürek, A. Ruiz, F. D. Igual, R. Mayo, M. Ujaldon, Biomedical image analysis on a cooperative cluster of GPUs and multicores, in: P. Zhou (Ed.), ICS, ACM, 2008, pp. 15–25.
[27] S. Szénási, Distributed implementations of cell nuclei detection algorithm, in: Proceedings of the 1st International Conference on Image Processing and Pattern Recognition (IPPR), 2013, pp. 105–109.
[28] M. Sonka, V. Hlavac, R. Boyle, Image Processing, Analysis, and Machine Vision, 2nd Edition, Chapman & Hall, 1998.
[29] N. Wirth, Toward a discipline of real-time programming, Commun. ACM, vol. 20, no. 8, 1977, pp. 577–583.
[30] J. B. Kruskal, On the shortest spanning subtree of a graph and the traveling salesman problem, Proceedings of the American Mathematical Society, vol. 7, no. 1, 1956, pp. 48–50.
[31] E. Tóth-Laufer, M. Takács, I. Rudas, Neuro-fuzzy risk calculation model for physiological processes, in: IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY), 2012, pp. 255–258.
[32] S. S. Skiena, The Algorithm Design Manual, Springer, 2008.

Acknowledgements

This work makes use of results produced by the Hungarian National Technology Programme, A1, Life sciences, the “Development of integrated virtual microscopy technologies and reagents for diagnosing, therapeutical prediction and preventive screening of colon cancer “Hungarian National Technology Programme, A1, Life sciences, (3dhist08) project and the ÓE-RH 1104/2-2011 project.

Figures and Tables

Algorithm 1 Data parallel region growing

```
1:  function RegionGrowing(seedpoint)
2:      region ← {seedpoint}; last ← seedpoint; best ← ∅; contour ← ∅
3:      while ¬StopCondition(region, contour) do
4:          GetRegionProperties(region, ref I_region, ref R_MAX, ref center)
5:          parallel execution where threadIndex in (1,2,3,4)
6:              if InsideImage(last + D_threadIndex) then
7:                  new ← last + D_threadIndex
8:                  if new ∉ region ∪ contour and Processable(new) then
9:                      contour ← contour ∪ new                    ▷ Contour expansion
10:                 end if
11:             end if
12:         end parallel execution
13:         C_max ← 0
14:         parallel execution where threadIndex in (1, 2, .., |contour|)
15:             CP ← contour[threadIndex]
16:             C ← |I_CP − I_region| + α ∗ (d(CP, center)/R_MAX)   ▷ Score
17:             C_MAX ← AtomicMax(C, C_MAX)                        ▷ Maximal score race
18:         end parallel execution
19:         parallel execution where threadIndex in (1, 2, .., |contour|)
20:             if C = C_MAX then                                  ▷ Who wins?
21:                 last ← contour[threadIndex]
22:             end if
23:         end parallel execution
24:         region ← region ∪ last                                 ▷ Expand region
25:         contour ← contour \ last                               ▷ Reduct contour
26:         if best = ∅ or Score(region) > Score(best) then
27:             best ← region                                      ▷ First or better score
28:         end if
29:     end while
30:     return best
31: end function
```

Variables and functions of the pseudocode:

- I_{region}: The average intensity of the region.
- R_{MAX}: Maximal radius of the region.
- *center*: Mass center of the region.
- D: The 4 directions $D_1 = (0,1); D_2 = (1,0); D_3 = (0,-1); D_4 = (-1,0)$.
- I_P: Intensity of pixel P.
- $InsideImage(P)$: Result is true if P is in the picture.
- $Processable(P)$: Result is true if P is not part of any other regions.
- α: Weighting factor.
- $d(a,b)$: The (Euclidean) distance between points a and b.
- $Score(region)$: Fitness value of the region (based on size, intensity etc.).
- $StopCondition(region, contour)$: Checks the stopping conditions of the region growing procedure (size, available points, etc.).

Algorithm 2 Backtracking algorithm for non-overlapping nuclei filtering

```
1: procedure BACKTRACKING(level, ref R, ref Found, ref OPT)
2:     i ← 0                                          ▷ 0 - accept/1 - reject this nucleus
3:     while i ≤ 1 do
4:         j ← 1
5:         while (j < level) and (R_j = 0 or ¬Overlap(NC_level, NC_j) do
6:             j ← j + 1
7:         end while
8:         if j = level then                                  ▷ Acceptable together
9:             R_level ← i                                        ▷ Store this result
10:            if level = N then
11:                if ¬Found or Score(R) > Score(OPT) then
12:                    Found ← true                            ▷ First or better result
13:                    OPT ← R
14:                end if
15:            else
16:                Backtracking(level+1, ref R, ref Found, ref OPT);
17:            end if
18:        end if
19:        i ← i + 1
20:    end while
21: end procedure
```

Variables and functions of the pseudocode:

- N: Number of cell nuclei candidates in the given cluster.
- NC_i: The i^{th} cell nucleus candidate.
- *level*: Index of the currently examined nucleus candidate ($1 \leq level \leq N$). Initial value is 1.
- $Overlap(NC_1, NC_2)$: Result is true if NC_1 and NC_2 overlap each other.
- R: Vector of N integers. Contains the actual solution (R_i is 0 if the i^{th} cell nucleus is rejected and it is 1 if the i^{th} cell nucleus is accepted).
- $Found$: Becomes true when the algorithm finds the first valid solution. Initial value is false.
- OPT: Contains the optimal solution (format is the same as the format of R).

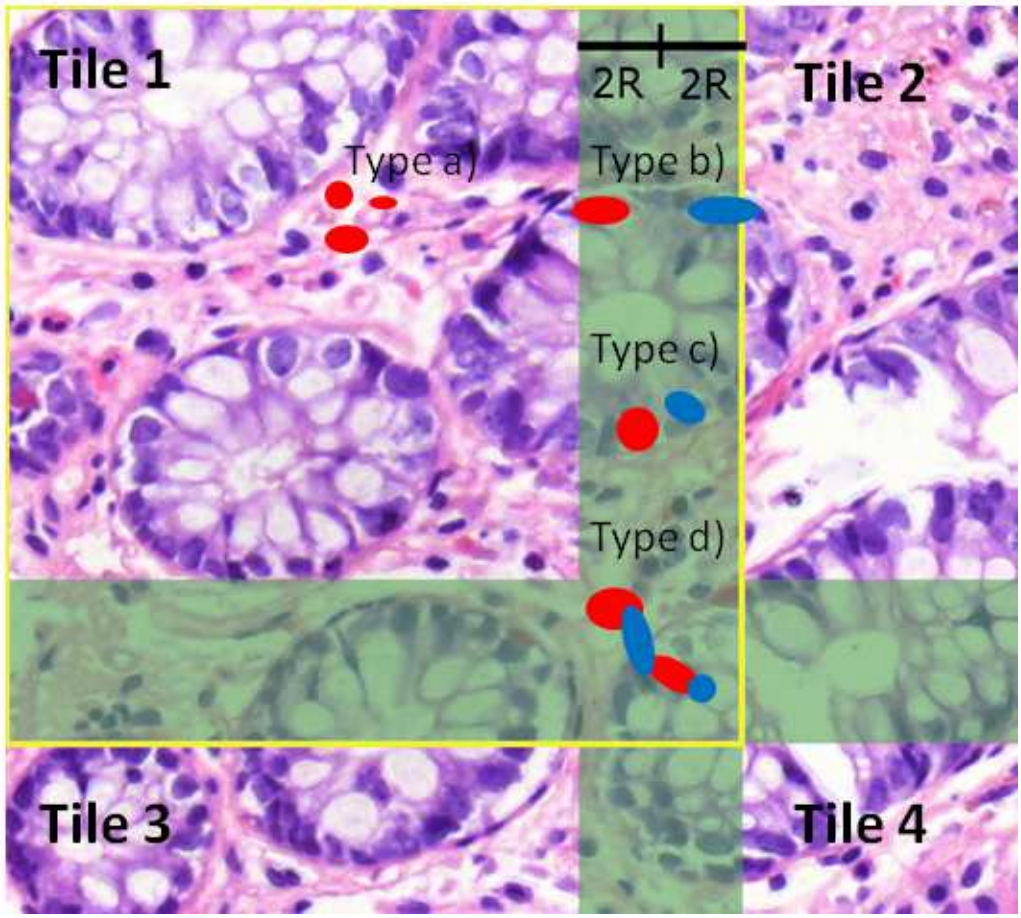

Figure 1: Splitted HE stained colon tissue image. Green area: overlapped area. Yellow rectangle: location of Tile 1. Red areas: cell nuclei candidates using a seed point from Tile 1. Blue areas: cell nuclei candidates using a seed point from Tile 2. Type a: cell nuclei candidates completely located in the non-overlapping area. Type b: cell nuclei candidates using seed points from the non-overlapping area but grown into the overlapping area. Type c: non-overlapping cell nuclei candidates in the overlapping area. Type d: overlapping cell nuclei candidates.

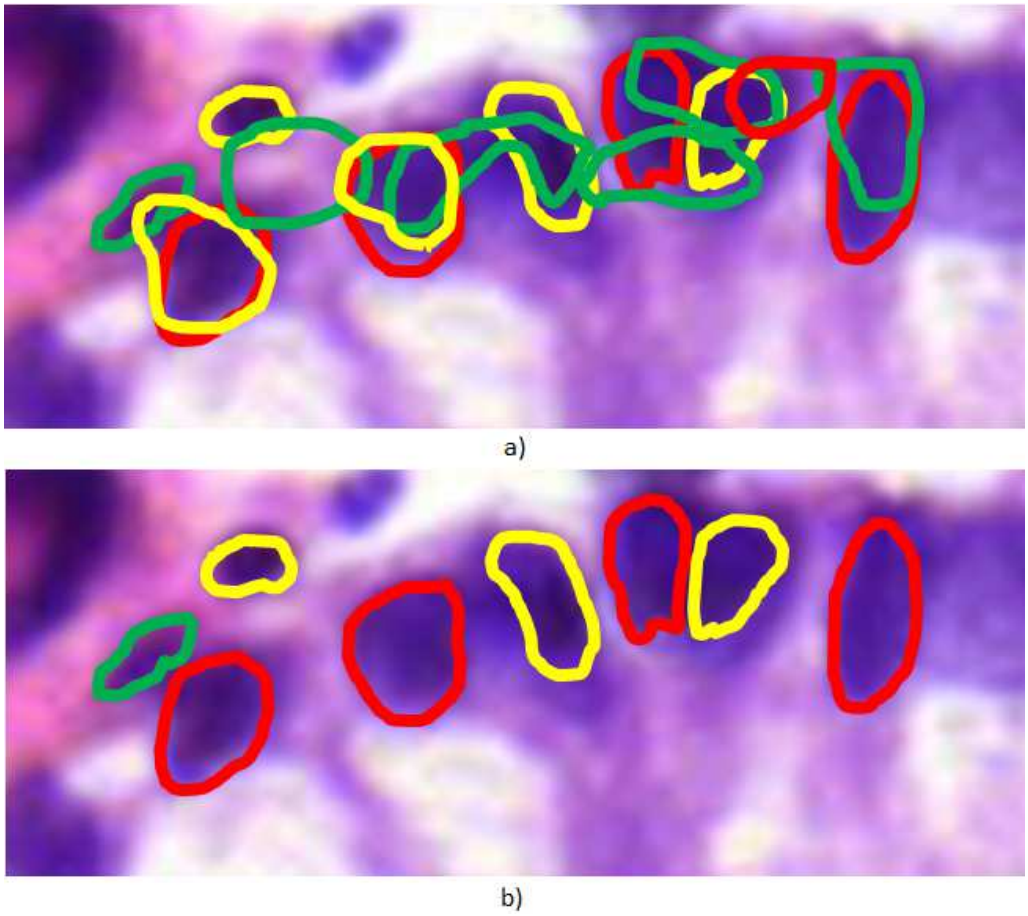

Figure 2: Result of the backtrack algorithm. a) Input: overlapping cell nuclei candidates (nuclei with different colours are from different GPGPUs) b) Output: the optimal solution. Non-overlapping cell nuclei candidates with maximum summarized score.

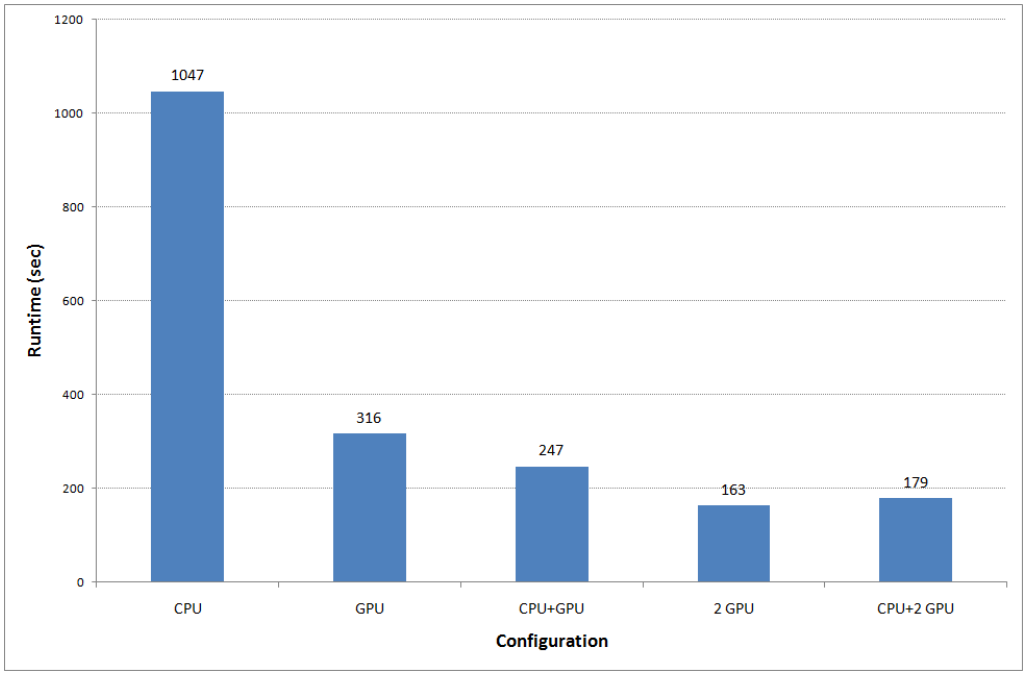

Figure 3: Average runtime for different configurations in case of full sized images (8192x8192 pixels) using recommended tile size (2048x2048 pixels).

Table 1: Runtimes of the CPU and the GPU implementations for different image dimensions. Relative runtime means how long it took to process one pixel. Speed-up is the ratio of these relative values.

Slide ID	Dim. (pixel)	Runtime (ms)		Rel. runtime (ms/pixels)		Speed-up
		CPU	GPU	CPU	GPU	
10359-04ep	1024	21584	15050	0.0206	0.0144	1.4
10393-04_ep	1024	24676	17204	0.0235	0.0164	1.4
1050-04IIadenomavill+dyspl	1024	18428	11950	0.0176	0.0114	1.5
1160-05CRCA-B	1024	31160	17298	0.0297	0.0165	1.8
11700-04CRCA-B	1024	21942	13764	0.0209	0.0131	1.6
12138-03Adenomavillosum	1024	20276	14996	0.0193	0.0143	1.4
12532-04CRCA-B	1024	29060	17916	0.0277	0.0171	1.6
2877-04IHyperpl	1024	34470	20240	0.0329	0.0193	1.7
6134-04p	1024	21830	14472	0.0208	0.0138	1.5
8658-04IHyperpl	1024	29402	18418	0.0280	0.0176	1.6
986604Chron	1024	27528	14714	0.0263	0.0140	1.9
986604Crohn	1024	25670	15790	0.0245	0.0151	1.6
9872-04_I_ep	1024	26718	15342	0.0255	0.0146	1.7
CRCA-B1	1024	35274	17132	0.0336	0.0163	2.1
rossz_1013-05CRC-B	1024	33498	19094	0.0319	0.0182	1.8
10359-04ep	2048	141256	41632	0.0337	0.0099	3.4
10393-04_ep	2048	140148	38392	0.0334	0.0092	3.7
1050-04IIadenomavill+dyspl	2048	106896	26298	0.0255	0.0063	4.1
1160-05CRCA-B	2048	201184	50144	0.0480	0.0120	4.0
11700-04CRCA-B	2048	131644	35624	0.0314	0.0085	3.7
12138-03Adenomavillosum	2048	85688	32134	0.0204	0.0077	2.7
12532-04CRCA-B	2048	153828	40556	0.0367	0.0097	3.8
2877-04IHyperpl	2048	151172	42892	0.0360	0.0102	3.5
6134-04p	2048	137362	37064	0.0327	0.0088	3.7
8658-04IHyperpl	2048	170760	44454	0.0407	0.0106	3.8
986604Chron	2048	164612	36846	0.0392	0.0088	4.5
986604Crohn	2048	143564	37500	0.0342	0.0089	3.8
9872-04_I_ep	2048	148648	40100	0.0354	0.0096	3.7
10359-04ep	4096	455175	101688	0.0271	0.0061	4.5
1050-04IIadenomavill+dyspl	4096	525724	140594	0.0313	0.0084	3.7
1160-05CRCA-B	4096	524166	139058	0.0312	0.0083	3.8
11700-04CRCA-B	4096	391826	152480	0.0234	0.0091	2.6
12138-03Adenomavillosum	4096	489434	133166	0.0292	0.0079	3.7
12532-04CRCA-B	4096	418242	134526	0.0249	0.0080	3.1
2877-04IHyperpl	4096	471368	171144	0.0281	0.0102	2.8
6134-04p	4096	909918	77446	0.0542	0.0046	11.7
8658-04IHyperpl	4096	429865	200300	0.0256	0.0119	2.1
986604Chron	4096	341888	98892	0.0204	0.0059	3.5
986604Crohn	4096	339073	139930	0.0202	0.0083	2.4
9872-04_I_ep	4096	607074	124704	0.0362	0.0074	4.9
CRCA-B1	4096	353276	123358	0.0211	0.0074	2.9

Table 2: Comparison of original region growing and split-and-merge method accuracy in the case of Gold Standard images.

Slide ID	**Accuracy**		**Difference**
	Original	**Split-and-merge**	
0259-ES-02	69,15%	69.70%	0.54%
0259-ES-03	76.29%	76.48%	0.18%
0259-PR-01	80.82%	80.95%	0.13%
0259-PR-02	83.06%	83.47%	0.42%
1031-ES-01	94.23%	94.11%	-0.11%
1031-ES-02	77.29%	77.52%	0.23%
1429-PR-02	70.27%	70.55%	0.27%
2167-ES-01	67.20%	67.15%	-0.04%
2167-ES-02	71.95%	71.96%	0.01%
2167-ES-03	75.83%	76.53%	0.70%
2224-PR-01	80.83%	80.71%	-0.12%
2224-PR-02	80.21%	80.58%	0.37%
2225-ES-01	84.22%	84.36%	0.14%
2225-ES-02	81.43%	81.46%	0.03%
2225-ES-03	80.52%	80.40%	-0.12%
2508-PR-01	83.60%	83.67%	0.07%
2508-PR-02	80.72%	79.83%	-0.89%
2819-ES-01	68.73%	68.70%	-0.03%
2819-ES-02	76.97%	77.17%	0.20%
2819-ES-03	63.79%	64.15%	0.36%
2819-PR-01	77.25%	78.31%	1.06%
2819-PR-02	80.31%	80.48%	0.17%
2819-PR-03	76.38%	76.58%	0.20%
2856-ES-01	67.80%	68.80%	1.00%
2856-ES-02	75.23%	75.98%	0.75%
2857-ES-01	92.47%	91.36%	-1.12%
2857-ES-02	81.86%	80.53%	-1.33%
2857-ES-03	83.10%	82.53%	-0.57%
2924-PR-01	89.14%	88.20%	-0.94%
2924-PR-02	83.20%	82.24%	-0.96%
2924-PR-03	75.72%	74.50%	-1.22%
3019-PR-01	82.67%	82.64%	-0.02%
3019-PR-02	80.91%	80.53%	-0.38%
3019-PR-03	93.06%	93.02%	-0.04%
3381-ES-01	81.39%	81.39%	0.00%
3381-ES-03	75.63%	75.63%	0.00%
3381-PR-01	65.32%	65.24%	-0.08%
3381-PR-02	72.38%	72.34%	-0.04%
3381-PR-03	70.68%	70.68%	0.00%
Average	**78.25%**	**78.22%**	**-0.03%**

Table 3: Results of processing 2048x2048 pixels images. Second column shows the difference of the results between the original CPU region growing and the new split-and-merge method (using the evaluation method described in [16], using the original result as the reference and the split-and-merge result as the test image). The following columns show the detailed and summarized runtime values. The last column contains the (runtime) ratio of the additional split and merge procedures to the full processing time.

Slide ID	Result diff.	Runtime (ms)				S&M ratio
		Split	Growing	Merge	Sum (ms)	
10359-04ep	1.61%	18	106033	0	106051	0.01%
10393-04_ep	2.35%	18	111103	0	111121	0.01%
1050-04IIadenomavill+dysp	1.82%	15	76976	0	76991	0.01%
1160-05CRCA-B	1.59%	21	157260	0	157281	0.01%
11700-04CRCA-B	1.33%	15	100046	0	100061	0.01%
12138-03Adenomavillosum	2.87%	18	66633	0	66651	0.02%
12532-04CRCA-B	2.00%	18	113923	0	113941	0.01%
2877-04IHyperpl	2.22%	18	126962	0	126980	0.01%
6134-04p	1.48%	18	100417	0	100435	0.01%
8658-04IHyperpl	1.76%	21	126534	0	126555	0.01%
986604Chron2048	1.29%	21	119499	0	119520	0.01%
986604Crohn4096	1.43%	18	113171	218	113407	0.20%
9872-04_I_ep	2.13%	21	117730	0	117751	0.01%

Table 4: Runtime of the split-and-merge method in case of different image sizes and different configurations.

#	Dimension (pixel)	Runtime (ms)				
		CPU	**GPU**	**CPU+GPU**	**2 GPU**	**CPU+2 GPU**
1	2048	106052	70362	54116	35695	33437
2	4096	220805	148044	92037	75841	60094
3	2048	111122	69557	55373	35340	33224
4	2048	76992	47040	32884	26195	24925
5	4096	424174	240474	160252	121555	107128
6	8192	1083125	660633	431933	337422	278292
7	2048	157282	76262	56325	38893	41673
8	4096	437465	241972	164046	122313	98545
9	2048	100061	62362	46538	31799	31964
10	4096	436925	273936	180174	143498	126996
11	2048	66652	54777	35009	33830	21571
12	4096	288475	245631	136297	124700	95004
13	2048	113942	70421	49835	36092	38700
14	4096	313304	209155	131199	112607	91188
15	2048	126981	75051	49218	39954	34104
16	4096	562331	326736	204569	168311	146081
17	2048	100436	63211	55295	33377	29708
18	4096	203343	143498	89572	72405	54752
19	2048	126556	75135	53888	39633	38457
20	4096	537592	330736	219273	169185	140135
21	2048	119521	62216	46126	31483	34610
22	4096	324692	189275	120909	95128	75794
23	2048	113409	68459	48952	35040	36048
24	4096	426888	258848	171849	134088	116544
25	2048	117752	70343	58902	35552	35015
26	4096	410396	251091	159968	126644	99918
27	2048	156156	85734	65036	47502	47177
28	4096	403251	227576	148390	113976	95378
29	4096	661035	381124	268560	197930	173550
30	8192	691642	483778	294366	248202	198931
	$\sum$	**9018357**	**5563437**	**3680891**	**2864190**	**2438943**

Table 5: Runtime of the split-and-merge method in case of different tile sizes and different configurations. The size of both images is 8192x8192 pixels. Difference means the runtime ratio for the larger tile size to the smaller tile size.

Tile Size (pixel)	Configuration	Runtime (ms)		
		Image 1	Image 2	Sum
1024	CPU	1083125	691642	1774767
	GPU	660633	483778	1144411
	CPU+GPU	431933	294366	726299
	2 GPU	337422	248202	585624
	CPU+2 GPU	278292	198931	477223
2048	CPU	1305257	789345	2094602
	GPU	360850	272111	632961
	CPU+GPU	299994	193892	493886
	2 GPU	187861	138384	326245
	CPU+2 GPU	188910	168561	357471
Difference	CPU	1.20	1.14	1.18
	GPU	0.54	0.56	0.55
	CPU+GPU	0.69	0.65	0.68
	2 GPU	0.55	0.55	0.55
	CPU+2 GPU	0.67	0.84	0.75