Distributed Region Growing Algorithm for Medical Image Segmentation

Sándor Szénási

***Abstract*—** Signal processing plays an important role in the work of pathologists; it is especially true for image processing software products. High-resolution digital images have taken over the role of traditional tissue slides on a glass plate. In addition to the direct effects of this advancement (sharing images, remote access, etc.), a new option appeared: the possibility of using image processing software for automatic (or semi-automatic) diagnostics. One of the most important tasks in this procedure is the segmentation of the tissue images; we have to identify the main components (in the case of colon tissue samples, these are the cell nuclei, glands and surface epithelium). There are several traditional image segmentation methods for this purpose, but none of them provides both acceptable accuracy and runtime. This paper presents a distributed region growing method implemented on CPUs and GPGPUs.

***Keywords*—**distributed algorithm, GPGPU, medical image segmentation, parallel algorithm

I. Introduction

NOWADAYS the digital microscope is becoming a more and more common device. In addition to several advantages of these devices, it is worth to mention that besides the suitable IT background the images (see Figure 1) gained this way can be subjected to numerous other processes: archivation, categorization [1], remote access, further post-processing [2,3,4,5], etc. One of the most promising improvements is the semi-automatic diagnosis based on the segmentation of the image [6,7,8,9,10].

Segmentation of Haematoxilin and Eosing stained colon tissue images means the detection of the following main components: cell nuclei, glands, and epithelium (see Figure 2). Cell nuclei detection is a crucial step in this process, because there are several gland and epithelium segmentation techniques based on the identified nuclei [11,12,13]. Therefore, we need an accurate and fast cell nuclei detection method, and fortunately there are several already existing implementations. One of the promising alternatives is the region growing approach [14], which consists of the following steps: 1) selection of some seed points; 2) examination of the neighbouring pixels of the actual region; and 3) selection of the next pixel (based on some fitness functions) to be added to the region. We have to iterate this process until some exit condition is met.

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.

S. Szénási is with the Óbuda University, Budapest, Hungary, (phone: 36-1-666-5579; fax: 36-1-666-5579; e-mail: szenasi.sandor@nik.uni-obuda.hu).

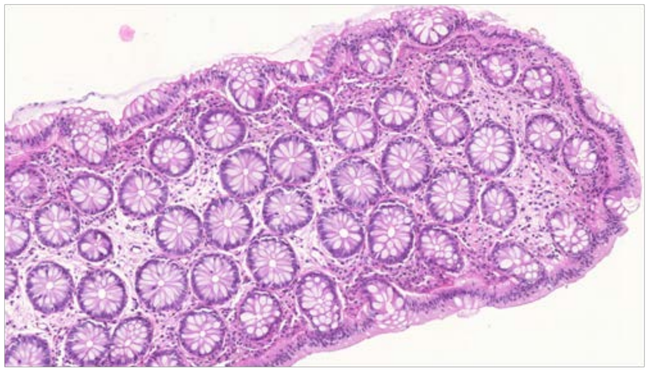


Fig. 1: Haematoxilin&Eosin stained colon tissue.

The region growing method has some limitations, which are mainly the high time and memory requirements. We have partially solved the speed problem by implementing a new data-parallel region growing algorithm [15]. This method already uses two levels of parallelization: 1) the region growing itself use hundreds of threads, each thread is responsible for the processing of one contour point, and 2) starting more than one region growing at the same time to utilize the full processing power of the devices. Region growing needs several parameters therefore we have to optimize these too [16,17].

II. GPGPU based region growing

A. Nvidia CUDA environment

GPGPU development has drastically evolved in the last few years. As GPUs became more powerful, software developers began enabling their applications to take advantage of this new massively parallel architecture [18,19]. The task of using the old standard graphics APIs (DirectX, OpenGL) for general-purpose computations poses several challenges. The use of these environments requires the mapping of all data and variables into graphics objects, while algorithms must be implemented as shared programs, pretending to perform transformations of graphical objects. These limitations make this kind of development hard and not widely spread.

The CUDA 1.0 software development environment (Compute Unified Device Architecture) introduced by Nvidia was a significant step forward, exposing a lot of hardware

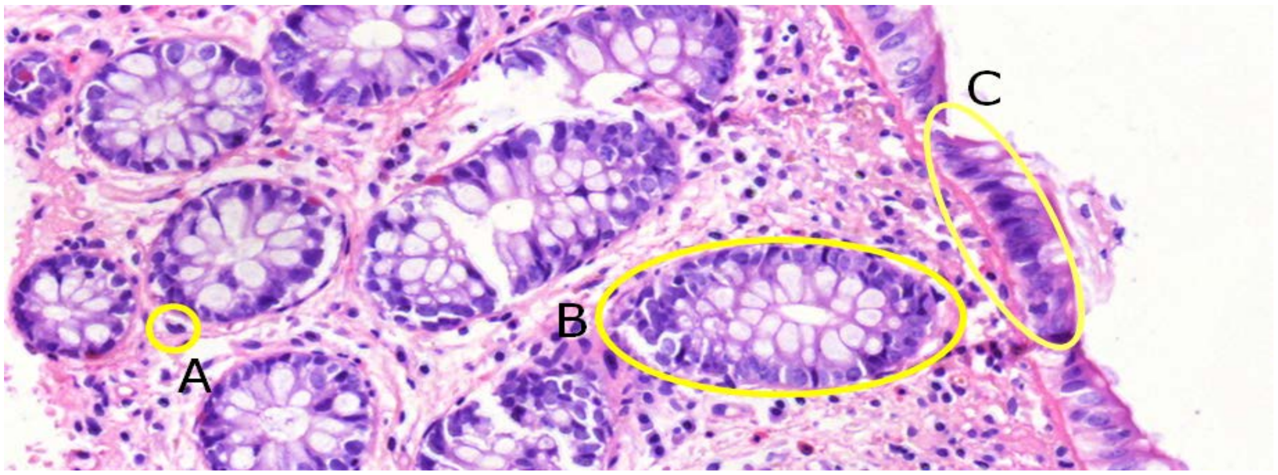


Fig. 2: Main components of a colon tissue. A – cell nucleus, B – gland, C – surface epithelium.

features that are not available via the original graphics-based application programming interfaces (DirectX, OpenGL). It has been widely deployed through thousands of applications and research papers.

It consists of some extensions to the C and C++ languages to control the graphics card from the CPU code, and to start GPU kernels with thousands of threads.

A CUDA program consists of two main parts:

- A sequential program running on the CPU
- A data parallel program running on the GPU (called a kernel)

Each kernel is executed on the graphics card in one thread, and these threads are organized into blocks. These blocks are executed by the separate multiprocessors in parallel. Each multiprocessor consists of stream processors, operated in a SIMT (Single Instruction Multiple Threads) fashion.

CUDA provides additional functions for data exchange between the memory and the device memory. This means memory allocation, de-allocation, and data transfer between memory areas. Memory operations are performed through DMA (Direct Memory Access) to decrease the load of the CPU. However, this means data transfer over the PCI Express bus; therefore, it is significantly slower than the memory access of the CPU or the GPU.

```
Function RegionGrowing( image )
  result ← Ø
  While (NextSeedPoint( image ) ≠ Ø)
    region ← { NextSeedPoint( image ) }
    While ¬StoppingCondition( region )
      contour ← GenerateContour( region )
      For all pnt in contour
        fitness[pnt] ← FitnessFunction( pnt )
      End for
      maxindex ← MAX( fitness )
      region ← region ∪ contour[maxindex]
    End while
    If Acceptable(region)
      result ← result ∪ region
    End if
  End while
  return result
End function
```

Alg. 1: Region growing algorithm.

We have used CUDA 6.0 environment and Fermi based graphics cards (NVIDIA GeForce GTX 570). This allows us to start 1024 threads in one block, which is a big improvement based on the earlier generations with a maximum number of 512 threads in a block.

B. Region growing approach

There are several methods for cell nuclei detection, for example K-means based, or edge-detection based techniques [20,21]. One of the most promising methods is the region growing approach. This is a classical image segmentation method.

Seeded region growing performs a segmentation of an image with respect to a set of points, called seed points [22]. Initially, this point is the region candidate.

An iteration of the main loop means the addition of one pixel to the already existing region. For this, we have to generate the contour of the region candidate, and select the most promising point to extend the region.

We have to repeat this iteration until one of the exit conditions occur. There are several conditions; the most important for us is the maximum size of the region.

Algorithm 1 presents the sequential version of the region growing algorithm, using the following external functions:

- NextSeedPoint(image): The result of this function is the next available seed point from the image (in case of cell nuclei detection, this is the darkest pixel). The result is Ø if there are no available seed points.
- GenerateContour(region): Generates the contour of the given region. The result of the function is a list of the contour pixels.
- FitnessFunction(point): The result is the fitness value of the given contour point. This score value is as high as the pixel corresponds to the already existing cell nuclei candidate.

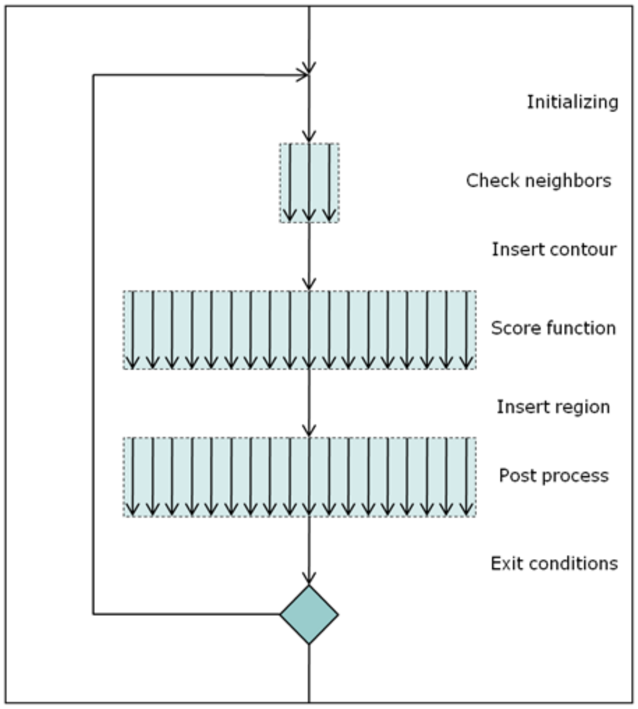


Fig. 3: Parallel parts of region growing algorithm.

- Acceptable(region): This function creates a post-checking using the size and colour of the built region candidate.

C. *Data parallel region growing*

Region growing has several advantages (it is one of the most accurate methods), but it has some disadvantages too. First, it is too slow. The processing of a full sized tissue image (8192x8192 pixels) requires about half an hour, which is unacceptable for practical usage.

Region growing is not a well parallelizable algorithm, but we can substitute some of its steps with a parallel one.

These steps are (see Figure 3):

- After the region expansion, we have to check all possible directions in which the contour can be expanded. Our implementation uses four neighbourhoods; therefore, we have to check the four neighbouring pixels. A data parallel implementation can use four threads parallel to check these directions.
- One of the most time consuming parts is the score value calculation. We have to calculate the fitness function for all contour pixels after every iteration. We cannot speed up this calculation with cache memory usage, because the fitness function uses some values changing after every iteration (the mass-centre and the average intensity of the region candidate, etc.). However, with using GPUs, we can create a kernel to calculate this fitness value for one pixel, and we can run as many kernels parallel as many contour points as we have (see Figure 4). This technique can significantly speed-up the fitness value calculation; and it is ideal for GPU implementation.
- We can use the GPGPU implementation of the different filters [23] in the pre-processing and in the post-processing phases.

III. Distributed implementations

A. *Naive implementation*

To gain maximum speed, it would be better to create a third level of parallelization, and use more than one device at the same time. We have developed three protocols for this purpose. The advantage of the naive implementation and the synchronized compatible version is that these give exactly the same result as the original region growing algorithm. This is possible because the main process itself remains unchanged; only the independent region growings are randomly [24] distributed between the execution units; therefore, we can process these at the same time using several devices.

The main problem is that all devices (CPUs and GPGPUs) own independent memory areas. In the naive method, we have to keep syncing all data between these devices after all region growing. This technique 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 devices have to store the whole image, so that during the update process, all of them have to transfer all data from the others.

B. *Compatible, synchronized solution*

Instead of the above, it would probably be a better solution if the GPGPUs do not contain the entire image. Consequently,

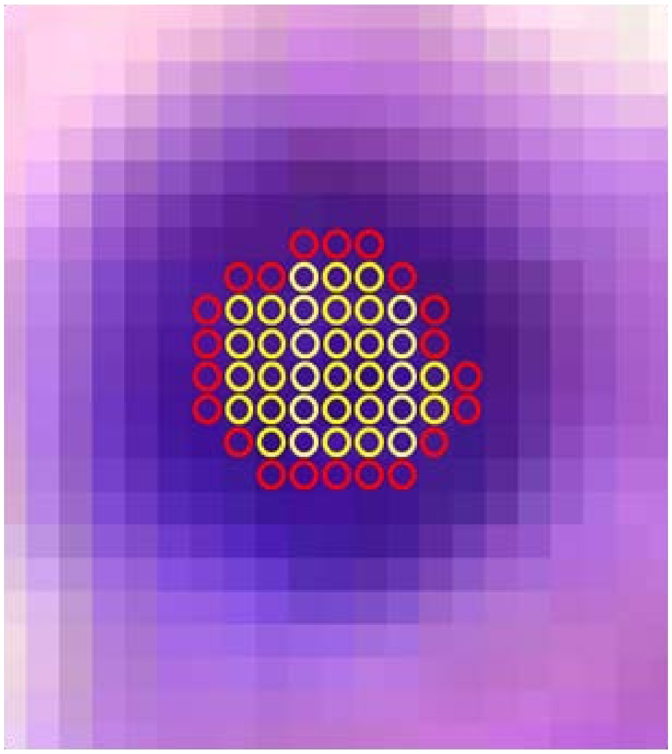
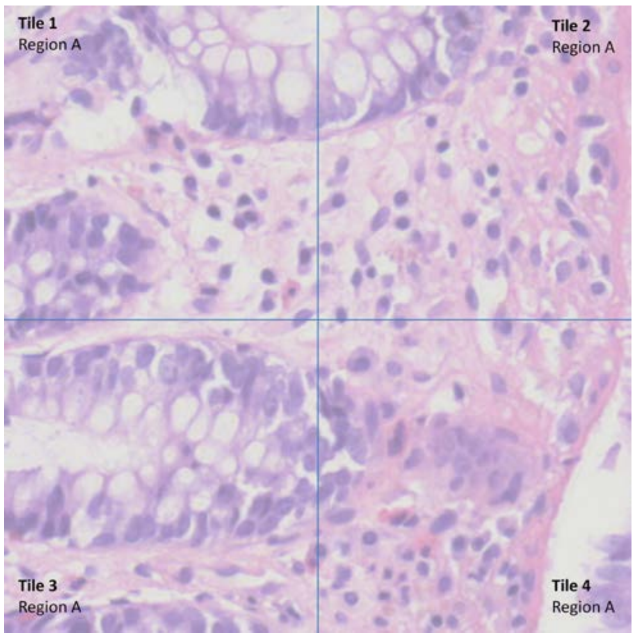
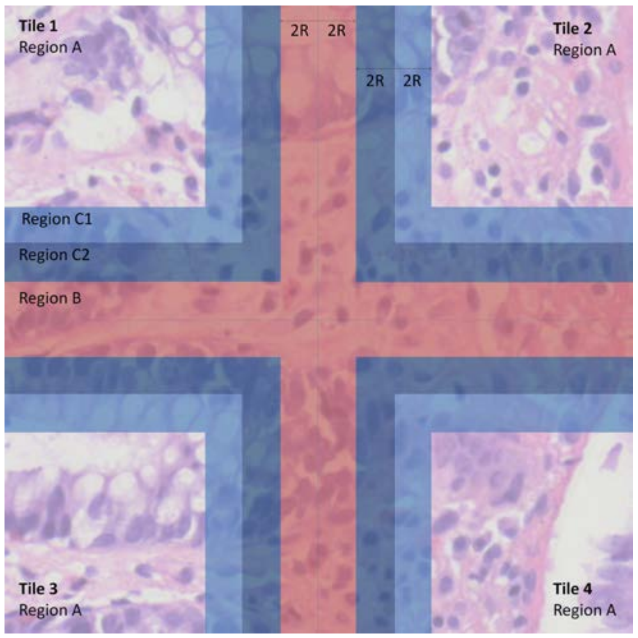


Fig. 4: Data parallel score calculation. Red circles represent the individual threads.





a) b)

Fig. 5: a) Image splitted into four Region A type areas. All areas are processed by different devices.
b) Red area: Region A; Blue area: Region C; Dark blue area: Region C_2; Light blue area: Region C_1.

it is not necessary to move the total amount of data between all devices. It would be better to split the complete picture into smaller regions, and distribute these between the available devices. Therefore, they are able to operate independently from each other.

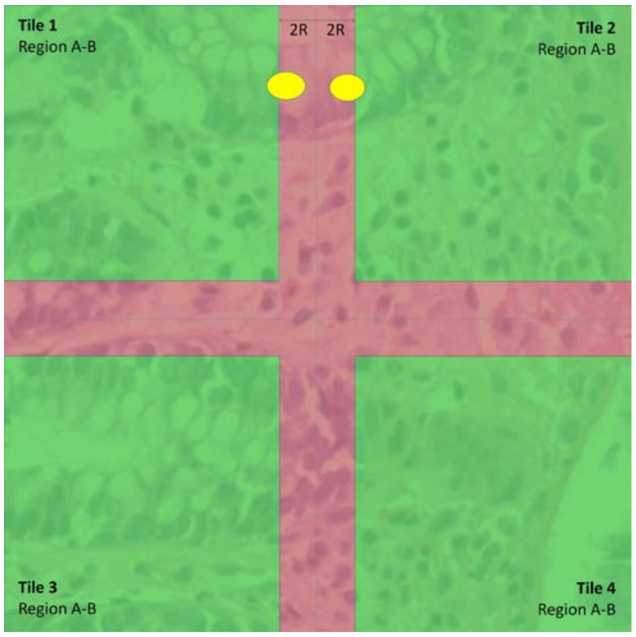
As usual, in case of similar distribution tasks, problems arise near the borders. It is possible that a nucleus is divided into two or more parts by the borders, and this can cause several problems. One of the GPGPUs may find the nucleus candidate, but the region cannot grow across the border; therefore, the algorithm will not be able to find the full shape. Two GPGPUs may find the same nucleus (two distinct parts of the same one). Therefore, in the result list, two nucleus candidates will appear instead of the correct one. The worst case is that due to the splitting method, too small parts have been placed in the memory regions of the two GPGPUs; therefore, none of them will identify it as a nucleus.

It would be good if the result of the multi-GPGPU process is exactly identical to the non-parallel version. It is important for the authorization process (the non-parallel version is already used in practice, and it is easier to obtain a permit for the new version, if the results of this are as similar as possible), and it is beneficial from the aspect of programming too (testing the application, etc.). Unfortunately, using completely individual GPGPU kernels for processing the image slices (which would be ideal for maximum performance) may cause several side effects as well. During the region growing, it was an important consideration that the processing order of the seed points was based on their score values (which is an integer value between 0 and 255). We have to process all seed points with higher score values, than the others with worse fitness value. It is possible that two or more seed points have the same score value, in fact the whole parallelization is based on this state. Because in this case, we can run these region growings in any order. Therefore, we can process these points parallel (if they are far enough from each other).

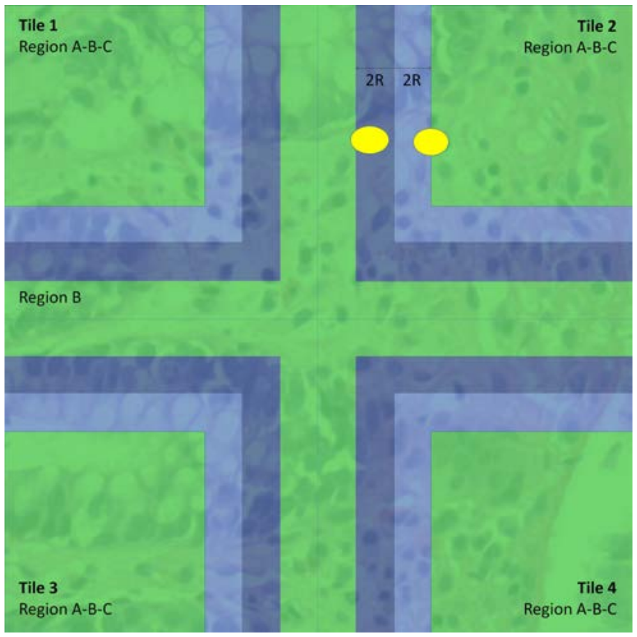
However, when there are several independent GPGPUs, we cannot guarantee this condition. It is possible that one of the GPGPUs have completed the processing of all seed points with a given score value and it starts processing points with lower fitness value, while at the same time the other GPGPUs work with a higher score value. It does not cause any problems inside the image slide, but it can be problematic in the overlapping areas. It is not acceptable that one of the GPGPUs finalizes a nuclei candidate with a lower score value, and because of this, another GPGPU cannot accept another (later found) overlapping nucleus candidate with a higher score value. Considering the above problems, the following algorithm should be used.

Before starting the algorithm, we have to split the tissue image into smaller parts. The following three areas are distinguishable:

- Area A: The GPGPU uses these areas for the region growing. The maximum size of these areas is based on the size the of the GPGPU memory. Another consideration is that these areas should be as large as possible, because this leads to higher parallelism (see Figure 5/a).
- Area B: Areas processed by the different GPGPUs are adjacent to each other; therefore, we cannot use all pixels of the regions as seed points. We should ensure that the region growings started at the edge of the picture tiles do not affect the results of the



a)



b)

Fig. 6: a) Green area: independent region growings started from A-B area. Yellow circles: maximum sized cell nuclei candidates. b) Green area: independent region growings started from A-B-C and B areas. Yellow circles: maximum sized cell nuclei candidates.

region growings started in different GPGPUs. Fortunately, this is easily met because we know the maximum radius of any cell nucleus (R). Hence, any two region growings can be started in parallel if the distance between the seed points is at least 4*R. We can ensure this constraint with the following technique: region growings can use the entire region A, but the seed points must be in the A-B region. The B region refers to all pixels that are farther from the nearest neighbour image tile than 2*R pixels (see Figure 5/a).

- Area C: There can be several seed points in the previously mentioned type B areas, which we have to include in the search process. These areas will be processed in a further step to simplify the parallelization. In summary, area C contains the pixels of A, where the distance from the nearest image tile being more than 2*R but less than 6*R. It is obvious that we can start region growing simultaneously from the B and A-B-C regions, because the minimum distance of the seed points will be at least 4*R. Therefore, the region growings will not meet (see Figure 5/b).
- Areas C_1 and C_2: The previously defined C area is further divided into two parts. C_1 is the set of pixels, which has a distance from the nearest neighbour tile of more than 4*R and less than 6*R. C_2 is the set of pixels with a distance from the nearest neighbour tile of more than 2*R and no less than 4*R (see Figure 5/b).

We have to take into account some additional parameters. At the edges of the original tissue sample, some neighbouring tiles can be found missing. Therefore, the B and C regions do not exist. We can simply handle these areas as type A.

Theoretically, the size of the tiles can be different. But, for simplicity (and faster memory transfers), we use unified resolutions. It would be worth not using square, but instead long rectangular areas, whose width equals the full image tissue width. In this case, all image parts have only one or two neighbours (on the top and the bottom). This can reduce the dependences, and increase the data transfer rate (rows one above the other can be moved by one sequential memory copy).

The algorithm is based on the followings:

1. Choosing the actual seed point limit and selecting all seed points with this fitness value.
2. Selecting seed points in the A-B areas in all GPGPUs, where the distance between these are more than 4*R (see Figure 6/a).
3. Starting region growings parallel in all GPGPUs from the previously selected seed points.
4. After region growings, all GPGPUs copy the B memory area into the host memory. The region growings and the memory copies are all independent. Therefore, we can run these procedures parallel.
5. Synchronization. All devices have to wait for the last one to complete the previous tasks.
6. Selecting seed points from the B or from the A-B-C area, where the fitness limit is equal to the previously selected value. Sending seed points positioned in the A-B-C area to the appropriate GPGPUs. Seed points in B area can be processed by the CPU (see Figure 6/b). We can use one of the

GPGPUs to process these starting points, but it needs too much memory copies, it is worth to do this directly in the host memory.

7. Starting region growings parallel in all GPGPUs and CPUs. These procedures can run parallel.
8. There may be changes in area C_1 in the GPGPUs, because the region growings from A-B-C can reach these pixels. There may also be changes in area C_2 in the CPU, because region growings from Area B can reach these pixels. Accordingly, after the previously mentioned region growings, the GPGPUs have to start the memory copies: the C_1 areas from GPGPUs to the host memory, and the C_2 areas from the host memory to the corresponding GPGPUs. Of course, this can be optimized based on whether there were any changes in these areas.
   The region growings and the memory copies are all independent. Therefore, we can run these procedures parallel.
9. Synchronization. All devices have to wait for the last one to complete the previous tasks.
10. If there are more seed points with the selected fitness limit, restart the iteration from step 2.
11. If there are not any more seed points with the selected fitness limit, restart the iteration from the first step.

After each iteration, we have to decrease the fitness limit until it reaches a minimum value. Seed points with fitness values less than this limit are not acceptable.

The biggest advantage of this method is that the results will be the same as the single GPGPU execution (which is the same as the traditional sequential results). In some cases, this can be critical (although in practice it turned out that there are several valid solutions with the same precision).

The drawback of this method is the synchronization requirements and the large data movement (although, it is still more manageable than the naive implementation).

C. *Split-and-merge method*

To achieve maximum performance, we have to develop an algorithm that permits as much independence for the devices as possible. Our third option is to simply divide the image into smaller tiles and process these separately. After this, we have to concatenate these results (this is the well-known split-and-merge method [25]). At the edges of these tiles, there may be several problems we have to handle.

In the split part, we have to split the entire image into smaller subimages (as large as acceptable for all devices). We use some overlapping using $OVER_{size}$ pixels width, where $OVER_{size}$ is a constant parameter. We can calculate the value

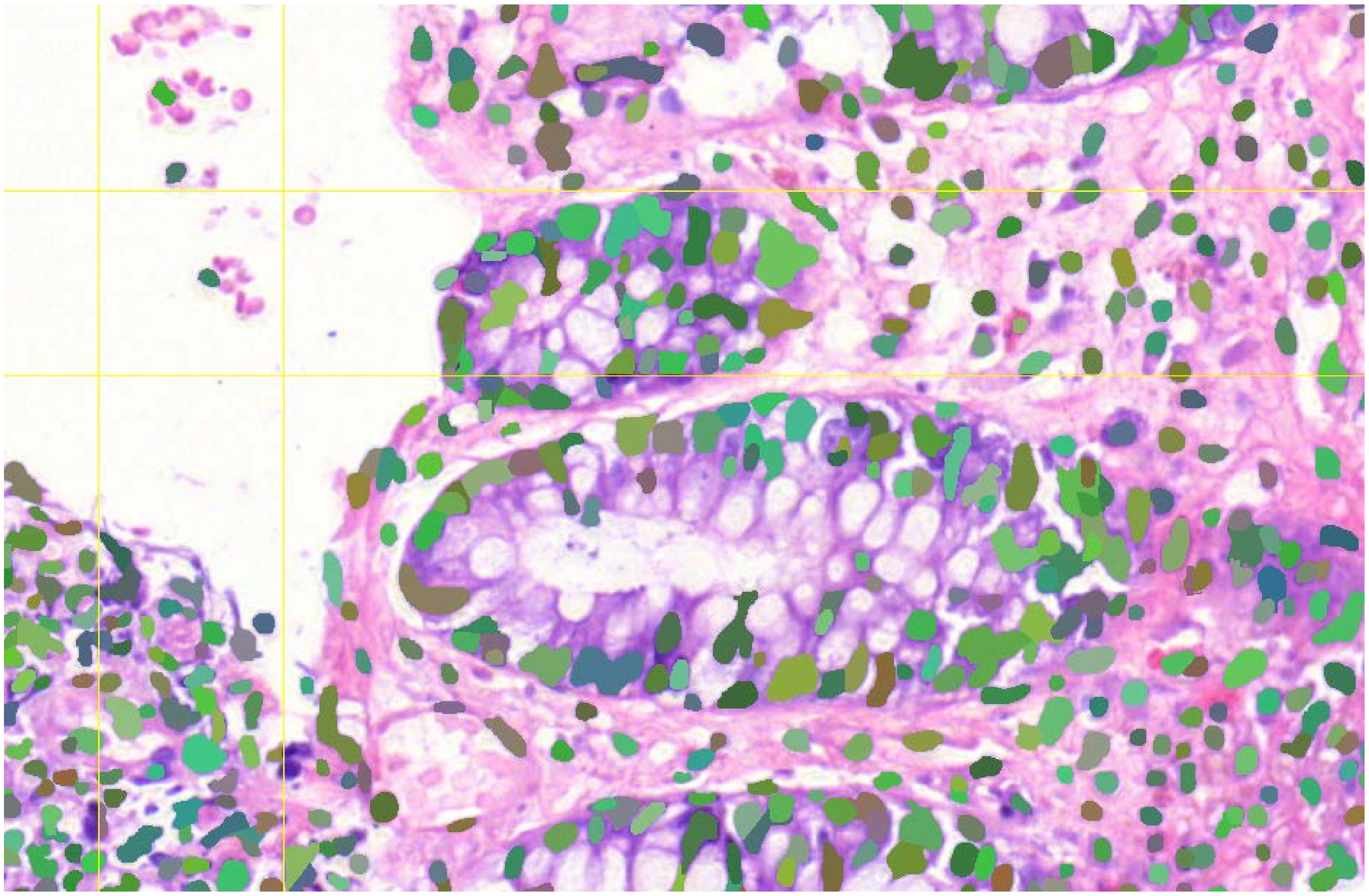


Fig. 7: Result of cell nuclei detection. The yellow lines show the border of the overlapping areas.
The green objects are the detected cell nuclei (different color represents individual cell nuclei candidates).

of this parameter since we know the maximum radius of any cell nucleus (R pixels). We use 4*R pixel width overlapping areas; therefore, there cannot be any two or more region growings started from the non-overlapping areas of different devices, which have shared pixels.

In the merging section, we concatenate the results. There may be several overlapping cells in these overlapping areas, and we have to select a non-overlapping subset of them (using some scoring function or a fuzzy approach [26,27,28]). We have developed a backtracking based [29] algorithm to solve this problem efficiently. The details of this method are in our previous paper [30].

IV. Results

A. Accuracy

This chapter contains the evaluation of the split and merge method, because this is the most promising of the presented three alternatives.

Our implementation uses the CPU for the split and for the merge phases, and uses the CPU and the available GPUs for the region growing section. The split and merge methods are implemented using C# (Visual Studio 2008), while the region growing methods are implemented using C++ and CUDA 6.0 (Visual Studio 2008). For testing purposes, we use the following hardware configuration: AMD64 Family 16 Model 4 CPU, 8 Gb RAM, Nvidia GeForce GTX 570 graphics card.

We used 15 pieces of 2048x2048 pixel-sized HE stained colon tissue images to analyse the application. In the first phase, we ran the original sequential CPU-based region growing algorithm on these images, and after that, we ran the new split-and-merge based application. As it is visible from the results, the new technique does not cause significant degradation of the accuracy. The average difference between the original one-step processing and the new split-and-merge method is less than 1%. In the case of larger images (full tissue image size is 8192x8192 pixels), the ratio of the overlapping and non-overlapping areas is more ideal; therefore, we could expect better results. Unfortunately, we cannot try this out, because the original sequential one-step CPU based region growing cannot process such large images due to its high memory requirements.

Table I contains the details about the accuracy test. As it is visible, the accuracy of the new method is usually the same as the original one.

We use the common definition for accuracy [31], as

$$\text{Accuracy} = (TP+TN) / (TP+TN+FP+FN)). \quad (1)$$

Where

- TP: Number of True Positive pixels
- TN: Number of True Negative pixels
- FP: Number of False Positive pixels
- FN: Number of False Negative pixels

Table I: Accuracy test results. Where SlideID is the name of the processed tissue image. TP: number of True Positive pixels, TN: number of True Negative pixels, FP: number of False Positive pixels, FN: number of False Negative Pixels, Acc: Accuracy

	Pixel count (pixel)				
Slide ID	**TP**	**TN**	**FP**	**FN**	**Acc.**
10359-04ep	1082553	3080466	31285	0	0.9925
10393-04_ep	866131	3279280	48893	0	0.9883
1050-04IIade..	407115	3771061	16128	0	0.9962
1160-05CRCA-B	1022119	3130233	41952	0	0.9900
11700-04CRCA-B	945042	3218642	30620	0	0.9927
12138-03Aden..	766437	3390233	37634	0	0.9910
12532-04CRCA-B	842072	3321478	30754	0	0.9927
2877-04IHyperpl	817697	3349391	27216	0	0.9935
6134-04p	807429	3358960	27915	0	0.9933
8658-04IHyperpl	895711	3267490	31103	0	0.9926
986604Chron	761663	3412256	20385	0	0.9951
986604Crohn	981186	3180491	32627	0	0.9922
9872-04_I_ep	842033	3308238	44033	0	0.9895
				Average	0.9923

B. Speed-up

The main improvement of this method is that it makes it possible to run more than one region growings parallel. Generating the picture tiles (split part) and the merge of the nuclei candidates (merge part) are both very resource-demanding procedures. We have implemented a well parallelizable method for both; therefore, we can run these methods in multi-core environments.

Processing of the picture tiles (the region growing operation itself) is obviously parallelizable. We can run the region growing method in the different hardware devices (CPU cores and GPUs) at the same time. This can speed up the processing; the execution time is significantly less using more devices.

Table II contains the details about the speed test.

We have also examined the speed loss caused by the additional split and the merge procedures. Where the split section means the following steps: loading the image, splitting it into smaller parts, and saving these images into files (most of the time is required by file load/store operations). The region growing section refers to the following steps: find available seed points, run region growings from these seed points, and finally segment the cell nuclei candidates. The merge section consists of the following steps: collecting cell nuclei candidates from each device, and finding the optimal non-overlapping subset of them.

Table III contains the details about these processing time values. As it is clearly visible, the split and merge runtime is not a significant drawback.

Table II: Speed test results

Slide ID	Processig time (ms)		
	Original	Split&merge	S&M/O
10359-04ep	257073	63713	0.25
10393-04_ep	259569	63221	0.24
1050-04IIade..	197715	47028	0.24
1160-05CRCA-B	365415	78265	0.21
11700-04CRCA-B	244016	59592	0.24
12138-03Aden..	163098	61745	0.38
12532-04CRCA-B	284529	74225	0.26
2877-04IHyperpl	276495	69985	0.25
6134-04p	248493	57686	0.23
8658-04IHyperpl	316088	71857	0.23
986604Chron	301767	64575	0.21
986604Crohn	263500	69523	0.26
9872-04_I_ep	270224	66138	0.24
		Average	0.25

Table III: Processing time of substeps

Slide ID	Processig time (ms)			S+M/ S+R+G
	Split	R.Grow	Merge	
10359-04ep	492	59283	3937	0.0695
10393-04_ep	608	58403	4209	0.0762
1050-04IIade..	437	44451	2140	0.0548
1160-05CRCA-B	296	74047	3922	0.0539
11700-04CRCA-B	406	54787	4399	0.0806
12138-03Aden..	484	57152	4109	0.0744
12532-04CRCA-B	593	68640	4992	0.0752
2877-04IHyperpl	811	64397	4777	0.0798
6134-04p	484	52622	4580	0.0878
8658-04IHyperpl	312	68038	3507	0.0531
986604Chron	452	60937	3186	0.0563
986604Crohn	499	62996	6028	0.0939
9872-04_I_ep	530	61935	3672	0.0635
			Average	0.0707

As it is visible, the processing time is significantly less than the processing time of the original algorithm. This means about a 4x speed up using the distributed implementation. We can expect that using more than one GPU will give us faster execution with similar accuracy.

V. Conclusions

We have developed a data parallel region growing algorithm, and we are searching for the improvement to use it on a distributed environment. This paper contains three possible solutions. The first, the naïve method has not been implemented, since the preliminary tests show that its memory and runtime requirements are too high.

We have implemented and analysed the split-and-merge method, and the results are very promising. The speed-up and the memory requirement is acceptable; it is usually 4-5X faster than the original algorithm. However, it has one disadvantage: the result of this implementation is not always the same as of the original.

VI. Future plans

If the further decrease in the runtime is considered the main purpose of the further developments, then it is necessary to create a version that supports more than one GPUs. The algorithm is already available; we need to run the test.

Another way of further development is the segmentation of the further tissue components. Detection of cell nuclei is only the first main step of the whole tissue segmentation procedure. We should find the glands and the surface epithelium as well.

References

[1] A. Bogardi-Meszoly, A. Rovid, H. Ishikawa, S. Yokoyama, and Z. Vamossy, "Tag and topic recommendation systems," *in Acta Polytechnica Hungarica*, vol. 10, no. 6, pp. 171–191, 2013.

[2] Sz. Sergyan, "Useful and effective feature descriptors in content-based image retrieval of thermal images," *in 4th IEEE International Symposium on Logistics and Industrial Informatics*, Smolenice, Slovakia, pp. 227–231, 2012.

[3] J. Tick, Cs. Imreh, and Z. Kovacs, "Business process modeling and the robust pns problem," *in Acta Polytechnica Hungarica*, vol. 10, no. 6, pp. 193–204, 2013.

[4] Sz. Sergyan, "Precision improvement of content-based image retrieval using dominant color histogram descriptor," *in 1st WSEAS International Conference on Image Processing and Pattern Recognition (IPPR '13)*, Budapest, Hungary, pp. 197-203, 2013.

[5] K. Lamár, Gy. Morva, "Hardware and Software Functions of Standalone Field Data Acquisition Devices for the Low Voltage Power Distribution Grid", Carpathian Journal of Electronic and Computer Engineerng, vol. 6, no. 1, pp. 22-27, 2013

[6] B. Kiss, J. Sápi, L. Kovács, "*Imaging method for model-based control of tumor diseases*", in IEEE 11th International Symposium on Intelligent Systems and Informatics, pp. 271-275, 2013

[7] G. Valcz, I. Bandi, B. Wichmann, A. Patai, D. Szabo, G. Kiszler, M. Kozlovszky, B. Molnar, and Z. Tulassay, "Automated detection of epithelial changes in colorectal carcinoma," *in Zeitschrift fur Gastroenterologie*, vol. 50, no. 5, 2012.

[8] M. Lascu and D. Lascu, "Graphical programming based biomedical signal acquisition and processing," *in International Journal of Circuits, Systems and Signal Processing*, vol. 1, no. 4, pp. 317-326, 2007.

[9] T. Koga, S. Furukawa, E. Uchino, and N. Suetake, "High-speed calculation for tissue characterization of coronary plaque by employing parallel computing techniques," *in International Journal of Circuits, Systems and Signal Processing*, vol. 5, no. 4, pp. 435-442, 2011.

[10] J. Dong, K. Inthavong, J. Tu, "Image-based computational hemodynamics evaluation of atherosclerotic carotid bifurcation models", *Computers in Biology and Medicine*, vol. 43, no. 10, pp. 1353-1362, 2013

[11] M. El Adawy, Z. Shehab, H. Keshk, and M. El Shourbagy, "A fast algorithm for segmentation of microscopic cell images," *in ITI 4th*

International Conference on Information&Communications Technology, pp. 1–1, 2006.

[12] P. Kádár, M. Karacsi, "Requirements of island mode controller for microCHP in micro grid", *in 11th International Symposium on Intelligent Systems and Informatics*, pp. 83-86, 2013

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

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

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

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

[17] Sz. Sergyan and L. Csink, "Automatic parametrization of region finding algorithms in gray images", *in 4th International Symposium on Applied Computational Intelligence and Informatics*, pp. 199–202, 2007.

[18] K. Kwon, E-S. Lee, B-S. Shin, "GPU-accelerated 3D mipmap for real-time visualization of ultrasound volume data", *Computers in Biology and Medicine*, vol. 43, no. 10, pp. 1382-1389, 2013

[19] H-G. Lee, M-K. Choi, B-S. Shin, S-C. Lee, "Reducing redundancy in wireless capsule endoscopy videos", *Computers in Biology and Medicine*, vol. 43, no. 6, pp. 670-682, 2013

[20] A. Nomura, M. Ichikawa, K. Okada, H. Miike, and T. Sakurai, "Edge detection algorithm inspired by pattern formation processes of reaction-diffusion systems," *in International Journal of Circuits, Systems and Signal Processing*, vol. 5, no. 2, pp. 105-115, 2011.

[21] S. Suhaila and T. Shimamura, "Image restoration based on edgemap and wiener filter for preserving fine details and edges," *in International Journal of Circuits, Systems and Signal Processing*, vol. 5, no. 6, pp. 618-626, 2011.

[22] R. Adams, L. Bischof, "Seeded region growing," *in IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 6, pp. 641-647, 1994.

[23] V. Jimenez-Fernandez, D. Martinez-Navarrete, C. Ventura-Arizmendi, Z. Hernandez-Paxtian, and J. Ramirez-Rodriguez, "Digital circuit architecture for a median filter of grayscale images based on sorting network," *in International Journal of Circuits, Systems and Signal Processing*, vol. 5, no. 3, pp. 297-304, 2011.

[24] Gy. Gyorok and M. Toth, "On random numbers in practice and their generating principles and methods," *in International Symposium on Applied Informatics and Related Areas: AIS 2010*, pp. 1–6, 2010.

[25] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, Chapman & Hall, 2 edition, 1998.

[26] E. Toth-Laufer, M. Takacs, and I. J. Rudas, "Neuro-fuzzy risk calculation model for physiological processes," *in IEEE 10th Jubilee International Symposium on Intelligent Systems and Informatics (SISY)*, pp. 255–258, 2012.

[27] T. A. Várkonyi, "Fuzzyfied Robust Fixed Point Transformations", *in 16th International Conference on Intelligent Engineering Systems*, pp. 457-462, 2012

[28] C. Pozna, R-E. Precup, "Applications of signatures to expert systems modelling", Acta Polytechnica Hungarica, vol. 11, no 2, pp. 21-39, 2014

[29] M. L. Ginsberg, "Dynamic backtracking", *CoRR*, cs.AI/9308101, 1993.

[30] S. Szenasi, "Medical image segmentation with split-and-merge method," *in 5th IEEE International Symposium on Logistics and Industrial Informatics (LINDI 2013)*, pp.137-140, 2013.

[31] R. Kohavi and F. Provost, "Glossary of terms," *in Machine Learning*, vol. 30, no. 2, pp. 271-274, 1998.

Sándor Szénási received M.Sc. degree in 2004 and Ph.D. degree in 2013 from Doctoral School of Applied Informatics (GSAI) of Óbuda University in Budapest.

He is an associate professor in the Institute of Software Technology of the John von Neumann Faculty of Informatics, Óbuda University, Budapest. He specializes in problems of parallel algorithms, GPGPU programming and medical image processing. He engages both in theoretical fundamentals and in algorithmic issues with respect to realization of practical requirements and given constraints.

Dr. Szénási is a member of the John von Neumann Computer Society and IEEE. He is a reviewer of several WSEAS conferences and journals.

For more information, please visit: http://www.uni-obuda.hu/sanyo/gpgpu