Distributed Implementations of Cell Nuclei Detection Algorithm

SÁNDOR SZÉNÁSI
Óbuda University
John von Neumann Faculty of Informatics
Bécsi Street 96/B, H-1034 Budapest
Hungary
szenasi.sandor@nik.uni-obuda.hu

Abstract: One of the most promising methods for cell nuclei detection in colon tissue images is region growing, but it has several limitations. The process is slow to the extent that practical use seems almost impossible since the segmentation of large images that contain many nuclei may require up to 40-60 minutes to complete. However, the method is very promising, it offers very good accuracy; therefore, it is definitely worth dealing with this drawback. However, we have tried to speed up the process based on distributed execution using some novel techniques: a naive implementation, a compatible synchronized version, and an implementation based on the split-and-merge technique. This paper presents the “compatible synchronized implementation” in detail.

Key–Words: medical image segmentation, cell nuclei detection, HE stained colon tissue, distributed algorithm

1 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 gained this way can be subjected to numerous other processes: archivation, categorization [1], remote access, further post-processing [2, 3] etc. One of the most promising improvements is the semi-automatic diagnosis based on the segmentation of the image [4].

Segmentation of Haematoxilin and Eosing stained colon tissue images means the detection of the following main components: cell nuclei, glands, and epithelium. Cell nuclei detection is a crucial step in this process, because there are several gland and epithelium segmentation techniques based on the identified nuclei [5, 6]. 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 [7], 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.

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 [8]. 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 [9].

2 Distributed implementations

2.1 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 [10] 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.

2.2 Compatible, synchronized solution

Instead of the above, it would probably be a better solution if the GPGPUs do not contain the entire image. Consequently, 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, but the candidate 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 has 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 (Figure 1(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 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 (Figure 1(b)).

- **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, 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 (Figure 1(b)).

- **Areas C₁ and C₂**: The previously defined C area is further divided into two parts. C₁ is the set of pixels, which has a distance from the nearest neighbour tile of more than 4*R and less than 6*R. C₂ is the set of pixels with a distance from the nearest neighbour tile of more than 2*R and no less than 4*R (Figure 1(b)).

We have to take into account some additional param-
eters. 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 han-
dle these areas as type A.

Theoretically, the size of the tiles can be differ-
ent. But, for simplicity (and faster memory transfers),
we use unified resolutions. It would be worth not us-
ing 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 depen-
dences, and increase the data transfer rate (rows one
above the other can be moved by one sequential mem-
ory copy).

The algorithm is based on the followings:

1. Choosing the actual seed point limit and select-
ing 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 (Figure 2(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 shared memory. The re-
gion growings and the memory copies are all in-
dependent. Therefore, we can run these proce-
dures 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 (Figure 2(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 shared 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 global mem-
ory, and the $C_2$ areas from the global 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. There-
fore, 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).

2.3 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 [11]). 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 sub images (as large as acceptable for all devices). We use some overlapping using \( \text{OVER}_{\text{size}} \) pixels width, where \( \text{OVER}_{\text{size}} \) is a constant parameter. We can calculate the value 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 [12]). We have developed a backtracking based [13] algorithm to solve this problem efficiently. The details of this method are in our previous paper [14].

3 Conclusion

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 naive method has not been implemented, since the preliminary tests show that its memory and runtime requirements are too high.

We have implemented 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.

Therefore, we have developed the third method, which can solve this problem too. Our estimations show that the runtime will be higher in this case; therefore, we should do some further optimization before the implementation.

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 ÖE-RH 1104/2-2011 project.

References:


Figure 2: Location of seed points in different steps.


