Acceleration of Tear Film Map Definition on Multicore Systems

Jorge González-Domínguez[1], Beatriz Remeseiro[2], and María J. Martín[1]

[1] Grupo de Arquitectura de Computadores, Universidade da Coruña
Campus de Elviña s/n, 15071 A Coruña, Spain
jgonzalezd@udc.es, mariam@udc.es
[2] INESC TEC - INESC Technology and Science
Campus da FEUP, Rua Dr. Roberto Frias, 4200–465 Porto, Portugal
bremeseiro@fe.up.pt

Abstract
Dry eye syndrome is a public health problem, and one of the most common conditions seen by eye care specialists. Among the clinical tests for its diagnosis, the evaluation of the interference patterns observed in the tear film lipid layer is often employed. In this sense, tear film maps illustrate the spatial distribution of the patterns over the whole tear film and provide useful information to practitioners. However, the creation of a single map usually takes tens of minutes. Medical experts currently demand applications with lower response time in order to provide a faster diagnosis for their patients. In this work, we explore different parallel approaches to accelerate the definition of the tear film map by exploiting the power of today's ubiquitous multicore systems. They can be executed on any multicore system without special software or hardware requirements. The experimental evaluation determines the best approach (on-demand with dynamic seed distribution) and proves that it can significantly decrease the runtime. For instance, the average runtime of our experiments with 50 real-world images on a system with AMD Opteron processors is reduced from more than 20 minutes to one minute and 12 seconds.

Keywords: Parallel Programming, Multithreading, Image Segmentation, Dry Eye

1 Introduction

Dry eye syndrome is a prevalent disease characterized by symptoms of ocular discomfort, ocular surface damage, reduced tear film stability, tear hyperosmolarity, and inflammatory components[3]. These features can be identified by different types of diagnostic tests. Among them, the grading of the tear film lipid layer appearance is usually the first clinical observation made by the experts [4]. This clinical test consists of two steps [5]: 1) acquiring an input image

of the tear film lipid layer using the Tearscope Plus, an instrument which allows clinicians to rapidly assess the lipid layer thickness in a non-invasive way; and 2) categorizing this image into one of the five interference patterns defined for this purpose. This classification is a difficult clinical task, specially with thin layers which lack of color and/or morphological features. Consequently, there is a high degree of inter- and intra-observer variability.

In order to facilitate the task to the medical experts, by providing objective results and saving time, an automatic system for tear film classification was proposed in [10]. This research demonstrated that the interference patterns can be characterized by a feature vector composed of color and texture properties, and classified by means of machine learning algorithms. With the aim of reducing the computational requirements and allowing the classification system to work in real-time, feature selection was applied in [8]. That is, the feature vector can be now calculated in less than 1 second, while maintaining the accuracy over 96%.

Besides this global analysis, tear film maps were presented in [9] to represent the spatial heterogeneity of the tear film lipid layer and detect multiple interference patterns per image. This approach uses color and texture features as the global classification, and an adapted version of the seeded region growing algorithm to perform the segmentation of the image into the interference patterns. The proposed methodology is able to generate tear film maps with an accuracy over 90% in comparison with the manual annotations made by four experienced practitioners. However, there is still large room for improvement on processing time since the creation of tear film maps takes tens of minutes on a single CPU. Although the time needed to compute each feature vector is less than 1 second, the great number of feature vectors per single image leads the region growing step to a large processing time. This is nowadays a barrier for a wide adoption of this representation among experts, who usually expect shorter response time.

In this paper we present multithreaded approaches to accelerate the region growing step included in this algorithm by exploiting the computational power of multicore systems. Although parallel region growing implementations have been previously developed for multicore [6, 7, 11] and manycore systems [12, 14], the novelty of our work is three-fold: 1) we implement, for the first time, a parallel algorithm for the generation of tear film maps, which has different characteristics than the biomedical works presented on the state of the art; 2) we present a parallel version with dynamic seed distribution among threads that outperforms the static distributions applied in previous works, as will be shown in the experimental evaluation; and 3) up to our knowledge, this is the first work that provides an experimental evaluation of multithreaded region growing on multicore platforms with up to 64 cores.

The rest of the paper is organized as follows. Section 2 provides necessary background information. Section 3 describes the different parallel versions developed for tear film maps. Experimental results are presented in Section 4. Finally, Section 5 concludes the paper.

2 Background

This section presents the background of the problem, including a brief description of the classic algorithm for image segmentation known as seeded region growing. Additionally, its use to create tear film maps is also explained.

2.1 Seeded region growing

Given a set of initial points known as seeds, which can be manually or automatically selected, the algorithm finds a tessellation of the image into regions. The seeds are the first points of the regions, which are grown examining, through an iterative process, the neighboring points.

The algorithm performs the growing only if a homogeneity criterion is satisfied. The original method [1] was applied to gray-scale images.

Algorithm 1: Pseudo-code of the region growing algorithm.

```
Data: list of seeds L, growing threshold β
Result: matrix of regions R
1  initialize matrix of regions R := 0
2  initialize sequentially sorted list SSL := φ
3  for each seed s ∈ L do
4      i := getLabel(s)
5      R[s] := i
6      N = getNeighbors(s)
7      for each neighbor n ∈ N do
8          δ = |property(n) − mean[i]|
9          add(SSL, n, i, δ))
       end
   end
10 while notEmpty(SSL) do
11     y := pushFirst(SSL)
12     N = getLabeledNeighbors(y)
13     removeBoundaryNeighbors(N)
14     if sameLabel(N) then
15         i := getLabel(N)
16         δ := getDelta(y)
17         if δ < β then
18             R[y] := i
19             update(mean[i])
20             N = getNoLabeledNeighbors(y)
21             for each neighbor n ∈ N do
22                 δ = |property(n) − mean[i]|
23                 add(SSL, n, i, δ))
               end
           end
       else
24         R[y] := −1
       end
   end
```

Algorithm 1 illustrates the process of growing carried out to get the final regions from the seeds. Firstly, the pixels corresponding to the seeds are labeled in the matrix of regions R (see lines 4 and 5). Then, all the neighbors of the seeds are added to a sorted list SSL (see lines from 6 to 9). This list is sorted based on the homogeneity criterion δ, which represents the difference between the new element and the average of an existing region with regard to a particular property, which in the classic algorithm is the gray-level value of the pixels. Therefore, the first element in the list will be the one with the minimum δ value.

Following, the sorted list SSL is processed until it does not contain any element. Thus, the process subsequently described is applied to each element of the list. The first element is removed from the list, and its neighbors are analyzed (see lines from 11 to 13). If all the neighbors previously labeled have the same label, other than the boundary label (used to

separate two adjacent regions), then its δ value previously calculated is obtained and compared to the growing threshold β (homogeneity parameter, whose value depends on the problem to solve). If $\delta < \beta$, then the element is labeled with the same label than its neighbors, the average property of the region is updated, and all the neighbors of the element are added to the SSL list (see lines from 18 to 23). Otherwise, if the neighbors already labeled do not have the same label, then the element is labeled as a boundary (see line 24).

2.2 Tear film maps

An adapted version of the seeded region growing algorithm to process tear film images was proposed in [9] based on the probabilities provided by a soft classifier. The objective was to create tear film maps which represent the spatial distribution of the interference patterns. It is a tile-based procedure which consists in dividing the input image into a set of overlapping *tiles*; i.e., areas of the original image of equal size, and analyzing them independently. The whole process, from the input image to the tear film map, is subsequently explained.

1. **Region of interest.** Input images acquired with the Tearscope Plus include irrelevant areas, such as the sclera or the eyelids, and so a preprocessing step is needed to extract the region of interest (ROI) in which the further analysis will take place. Note that the ROI corresponds to the area around the pupil, and it can be located by means of image processing techniques through a completely automatic process [9].

2. **Feature extraction.** The tiles inside the ROI are analyzed, and a descriptor per tile is obtained as follows: *(a)* the input image in RGB is transformed to the L*a*b color space, *(b)* a quantitative vector is created for each L*a*b channel using the co-occurrence features technique for texture extraction [8], and *(c)* the individual vectors are concatenated into a single one. The resultant descriptor is composed of 23 features which can be computed in less than 1 second [2].

3. **Soft classification.** Partial class-membership probabilities are used in soft classification to model uncertain labeling and mixtures of classes. Thus, they will be applied as the homogeneity criterion of the seeded region growing algorithm subsequently presented. This step consists in computing these probabilities for each feature vector previously obtained from the ROI tiles using a *support vector machine* (SVM) [9].

4. **Seeded region growing.** This step is the target of our parallelization as it consumes most of the procedure time (more than 95%). It is composed of two main parts: the automatic search of the seeds, and the region growing from them. The seeds are obtained by analyzing non-overlapped tiles of the ROI in terms of feature vectors and class-membership probabilities [9]. Then, for each analyzed tile, its maximum probability p_{max} is compared with the seed threshold α. If $p_{max} > \alpha$, then the center of the tile becomes a seed and is added to the list L.

   The region growing process is presented in Algorithm 1, with the particularity that the homogeneity criterion here represents the difference between the class-membership probability of the new element and the average probability of the corresponding region [9]. That is, this criterion is defined as: $\delta = |p - mean[i]|$, where p is the probability of the new element of belonging to the class associated to the label i, and $mean[i]$ is the average probability of belonging to the same class calculated over the pixels already labeled as i. The value of the growing threshold β is defined based on previous research [9], and the tear film map is created by processing the matrix of regions. That is, using a set of colors

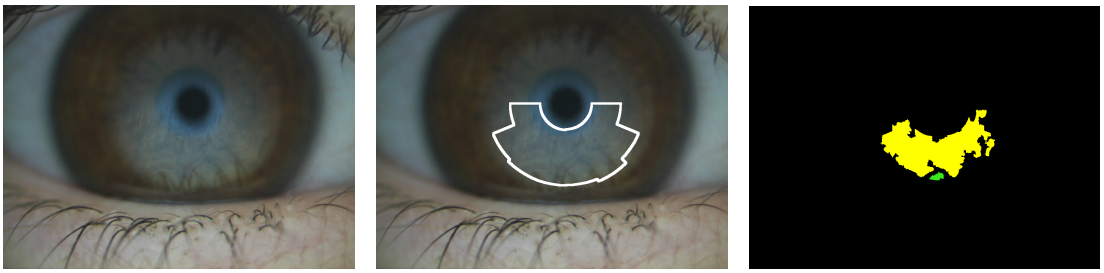

Figure 1: From left to right: input image of the tear film, location of the ROI, and output image obtained with our tear film mapping implementations.

to represent each interference pattern, tear film maps are generated by assigning colors to those elements which have a label different from the boundary label.

3 Parallel Implementation

Two different parallel versions of the region growing algorithm have been considered to create tear film maps. Both of them are implemented with the multithreading support available in C++11 standard. They receive as input a tear film image and the number of threads T, while they return the image with the tear film map that can support dry eye diagnosis. The two approaches provide tear film maps with the same accuracy as the original sequential code. Figure 1 shows an example of the images.

3.1 Parallelization I: full implementation

The first parallel proposal includes an additional initial step that processes the whole input image to calculate all the feature vectors and probabilities of each tile inside the ROI. The implementation of this first step avoids having to process the same tile more than once, which may happen if, for instance, a single tile is the neighbor of two different regions. Once finished this first step, it behaves like the classic version since the probabilities previously computed are simply accessed to evaluate the homogeneity criterion (see lines 8 and 22 in Algorithm 1).

As the vectors and probabilities are previously computed, the region growing step is very fast in this implementation, so we focused on optimizing the first step. The pixels of the input image are statically distributed among threads. As we work with a square tile of size $2 \cdot S$, $S-1$ pixels from each border of the image are not computed. Thus, in an execution with T threads and an image of $M \times N$ pixels, the number of pixels per thread is $\frac{(M-2 \cdot S+2) \cdot (N-2 \cdot S+2)}{T}$. Nevertheless, there exist many possible distributions of the pixels (see Figure 2). To decide the best distribution it must be taken into account the presence of several pixels that do not need any computation because they are not part of the ROI. If one thread has much more of those pixels assigned than other, this thread does not perform real computation, it is idle long time, the workload is not balanced, and the available resources are not efficiently exploited. As can be seen in Figure 1, most of pixels outside the ROI are located in the same areas of the image. Therefore, a block distribution (1D or 2D), where consecutive rows, columns or 2D blocks are assigned to the same thread (as the first two examples of Figure 2), will assign many pixels that do not need any computation to the same thread. To avoid this problem and obtain a good

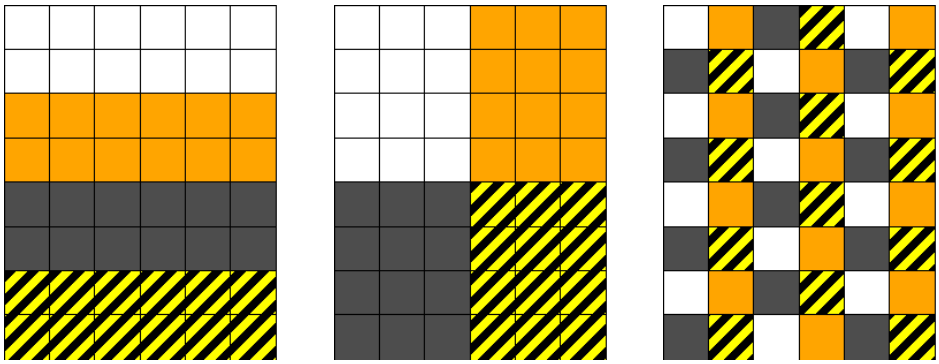

Figure 2: Examples of possible distributions of the pixels of an image among four threads, the pixels associated to each thread are represented by different patterns and colours. From left to right: 1D block distribution by rows, 2D block distribution, and cyclic distribution.

load balance among the threads, a cyclic distribution, where the pixels are assigned to threads in a round-robin way, is used (see the last image of Figure 2).

3.2 Parallelization II: on-demand implementation

Taking into account that the region growing only analyzes the neighbors of those elements which already belong to a region, the second alternative is based on analyzing the tiles on-demand. That is, there is no preliminary step to calculate the features vectors and class-membership probabilities of the whole image. They are calculated on-demand before evaluating the homogeneity criterion (see lines 8 and 22 in Algorithm 1) and only for those neighbors that are analyzed during the growing procedure. The parallelism is included in this approach by distributing the initial seeds among the threads. As the region growing procedure is independent for each seed, they can be analyzed in parallel. Two types of distribution were implemented:

- Static. It is the approach used by previous works to parallelize the region growing step of different applications [6, 7, 11]. The seeds associated to each thread are known in advance. We use a round-robin distribution that assigns similar number of seeds to each thread. The $i-th$ seed is analyzed by the thread j if and only if $j\%T = i$.
- Dynamic. The actual distribution of seeds to threads is initially unknown and it adapts to the size of each region. Each thread starts performing the region growing for one seed. Every time one thread finishes the computation of one seed, it looks for the next one that has not been analyzed yet by any thread. We use a shared variable to indicate the next seed to analyze. Threads must update this variable when starting the region growing of a new seed. These accesses are synchronized with a mutex in order to avoid simultaneous accesses from different threads.

The main advantage of the dynamic distribution is a better balanced workload. As the size of the regions is variable, the computation needed for each seed could present significant differences. The static distribution assigns the same number of seeds per thread, but it does not mean that the computation time of each thread is similar. The dynamic distribution adapts the workload to the real computational cost of the regions, i.e. threads that process small regions compute more seeds than threads associated to large regions. The drawback of the

dynamic distribution is the performance overhead due to the needed synchronization with the mutex. Ideally, the best performance would be obtained by a static distribution that assigns seeds with similar computational cost to each thread, as it would gather the advantages of both our static and dynamic distributions (good workload balance without synchronization overhead). However, as the computational cost of each seed is not known in advance, this type of distribution is not possible. Section 4 will provide a performance comparison between the two implemented distributions.

4 Experimental Results

Two platforms, with different characteristics, are used for evaluating the scalability of the two multithreaded implementations described in the previous section, as well as determining the impact of using static or dynamic seed distribution within the on-demand code:

- A platform with two 8-core Intel Xeon E5-2660 Sandy-Bridge processors (16 cores at 2.20 GHz in total). The memory hierarchy consists of 64 GB of main memory, 32 and 256 KB of private L1 and L2 caches, and 20 MB of shared L3 cache. The codes are compiled with GCC version 4.9.2.
- A system with four 16-core AMD Opteron 6272 processors (in total, 64 cores at 2.10 GHz). A private L1 cache of 16 KB is available for each core, while the 2 MB L2 and 8 MB L3 caches are shared among two and eight cores, respectively. Main memory size is 128 GB and the version of the available GCC compiler is 4.8.1.

Both GCC compilers support the C++11 standard, and all the experiments are compiled with the -O3 flag. Regarding the images, the experimentation was performed with the VOPTICAL_R dataset [13] which contains 50 images of the preocular tear film with a resolution of 1024×768 pixels. All the images were manually annotated by experts who delimited those regions associated with the five interference patterns. Note that the time needed to generate tear film maps is variable since the region-growing runtime depends on how much the regions grow. Furthermore, the runtime also depends on the ROI size, as all pixels outside the ROI are not processed. Thus, an image with smaller ROI is faster to analyze, regardless of the implementation considered.

The first part of the experimental evaluation consisted in a comparison of the output images provided by each parallel implementation. Although the outputs are not exactly the same, the difference is not appreciable for the practitioners. It means that there is no significant difference in the results accuracy when comparing them with manual annotations made by experts.

Tables 1 and 2 summarize the results of the performance evaluation on the two testbeds. We present results for three versions: full parallelization, as well as on-demand approach with static and dynamic seed distribution. We also show three values for each scenario (number of threads, parallel approach and system): the average, maximum and minimum execution time of the 50 images of the dataset.

The main conclusion that can be obtained from these tables is that parallel implementations significantly accelerate the generation of tear film maps. Among them, the on-demand version is faster than the full one for all combinations of images, systems and number of threads. Regarding the seed distribution, the dynamic approach is the fastest. For instance, the runtime is reduced by a factor of 9.95 (speedup) if we compare the parallel on-demand dynamic version to the sequential one on the Sandy-Bridge platform. This speedup is only 7.76 with the static distribution. On the AMD Opteron system the speedup values are also favourable to the dynamic

Table 1: Average, maximum and minimum runtime (in minutes) for the three multithreaded implementations working with the 50 images. We present results for different number of threads on the platform with two 8-core Sandy-Bridge processors.

	Full			On-demand static			On-demand dynamic		
Threads ↓	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
1	52.10	87.98	25.68	12.73	36.73	2.98	12.73	36.73	2.98
2	26.16	44.55	12.79	7.08	18.75	1.56	6.50	18.56	1.53
4	13.21	22.71	6.38	4.09	11.00	0.97	3.51	10.51	0.87
8	6.72	11.85	3.21	2.57	7.06	0.78	2.01	5.85	0.59
16	3.73	6.70	1.79	1.64	4.18	0.75	1.28	3.17	0.42

Table 2: Average, maximum and minimum runtime (in minutes) for the three multithreaded implementations working with the 50 images. We present results for different number of threads on the platform with four 16-core AMD Opteron processors.

	Full			On-demand static			On-demand dynamic		
Threads ↓	Avg	Max	Min	Avg	Max	Min	Avg	Max	Min
1	96.07	163.75	46.70	20.26	58.32	4.60	20.26	58.32	4.60
2	47.90	82.21	23.16	11.08	29.96	2.29	10.17	28.94	2.33
4	23.88	41.66	11.27	6.24	19.68	1.52	5.29	15.44	1.32
8	11.67	21.45	5.42	3.67	9.54	1.14	2.93	9.25	0.86
16	5.67	10.72	2.68	2.26	6.09	0.64	1.71	4.52	0.63
32	2.90	5.51	1.39	1.53	3.48	0.60	1.20	2.80	0.60
64	2.65	5.06	1.28	1.42	3.43	0.60	1.20	2.79	0.60

distribution: 16.88 to 14.27. As explained in Section 3.2, the dynamic distribution requires synchronization among threads to access the mutex. Despite the performance overhead provoked by this synchronization, it overcomes the static distribution thanks to a better workload balance: threads that compute faster regions analyze more seeds, instead of remaining idle.

Consequently, thanks to our work, the time to create the tear film maps is more suitable for experts. Without the parallel approach, the best sequential approach (on-demand) needs on average around 13 minutes on the Sandy-Bridge machine. The best parallel implementation (on-demand dynamic) reduces it to one minute and 17 seconds thanks to working on the available 16 cores. As the computational power of each AMD Opteron core is lower, the sequential time on the second system is higher (on average, around 20 minutes with the on-demand approach). However, as we can exploit the power of up to 64 cores, the average parallel time is similar to the Sandy-Bridge system: one minute and 12 seconds. Moreover, the impact is even more important if we take into account the most computationally expensive image (*Max* in Tables 1 and 2). In this case, the difference between the best sequential and parallel implementations is even higher: 33 and 55 minutes on the Sandy-Bridge and Opteron platforms, respectively.

Regarding the implementation that computes the whole image (full approach), it obtains good speedups over its sequential counterpart (13.97 and 36.25 on average on the Intel and the AMD platforms, respectively), but it is always slower than the two on-demand versions. Figure 3 shows, for the three parallel versions, the speedups obtained for the average runtimes as well as for the slowest and fastest images. The speedups of the full approach are calculated with respect to its own sequential time and the best sequential time (on-demand approach). The speedups for the on-demand versions always use this last sequential runtime as reference. The

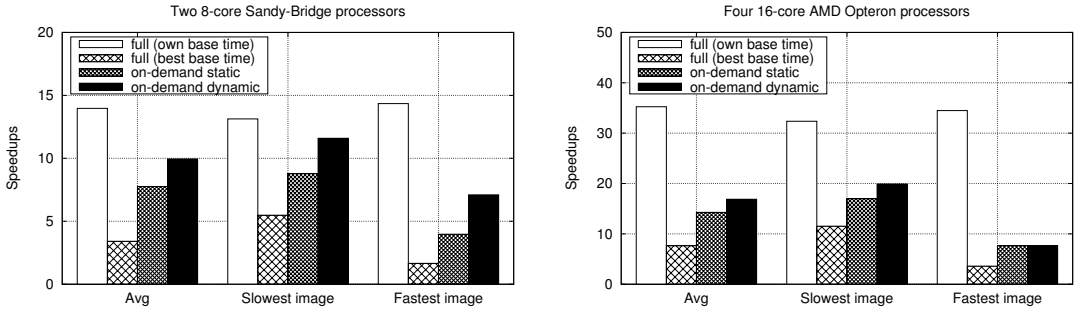

Figure 3: Speedups for the three approaches. Average speedups, as well as the values for the slowest and fastest images are presented. Two values are shown for the full approach: one that uses its runtime with one thread as reference and one with the best sequential time as baseline. Sequential time adopted for the on-demand versions is always the best for each system.

speedups of the full implementation over its own sequential runtime are always the highest. It proves that the problem of this implementation is not an inefficient parallelization, but the long time needed to calculate all feature vectors, which is much more computationally expensive than only analyzing the neighbouring pixels of the growing regions. The second box shows the speedups using the same baseline as the on-demand approach, and it provides the same conclusion as Tables 1 and 2: the good scalability of the full implementation is not able to compensate the difference of sequential runtime compared to the on-demand version.

5 Conclusions

The analysis of the interference patterns on the tear film lipid layer is a useful clinical test to diagnose dry eye syndrome. This task can be greatly simplified by means of the so-called tear film maps. However, the time required by the existing applications to generate them prevents a wider acceptance of this method. A performance analysis of this application shows that the bottleneck is gathered in one single step: the seeded region growing. In this paper we explore different multithreaded approaches to accelerate this step by exploiting the multicore capabilities of current machines. Two approaches were implemented. The full approach calculates in parallel the feature vectors and class-membership probabilities of all the pixels in the image, and then applies region growing using them. The on-demand algorithm starts the region growing from some initial seeds without having calculated all the information, and only computes the features and probabilities associated to those pixels which are in the neighborhood of the regions. In this case the parallelism consists in different threads exploring different seeds. Two seed distributions are available in the on-demand approach: static (the seeds are initially distributed among threads in a round-robin way), and dynamic (every time that one thread finishes the growing of one seed, it looks for the next one that has not been computed yet).

Experimental evaluation of the parallel implementations was performed on two machines with different characteristics based on the Intel Sandy-Bridge and the AMD Opteron processors, with 16 and 64 cores, respectively. A dataset with 50 tear film images was used in the evaluation. The experimental results show that all implementations provide high scalability. Moreover, the evaluation determines that the on-demand approach with dynamic seed distribution is the fastest on both systems. Note that the dynamic distribution is a novel approach

that significantly outperforms the static seed distribution used in the parallel region growing algorithms available in the state of the art. On average, it reduces the runtime from 12.73 to 1.28 minutes on the Sandy-Bridge system, and from 20.26 to 1.20 minutes on the platform with Opteron processors. Regarding the results for the image with the highest runtime, we can assert that in the worst case our multithreaded implementation just needs 3.17 and 2.79 minutes on the Sandy-Bridge and Opteron systems, respectively. The sequential implementation needs up to 36.73 and 58.32 minutes in the same systems.

As both experimental test platforms are NUMA systems, as future work we will analyze the impact on performance of applying different mapping policies (i.e., assignment of threads to concrete cores). We will also develop a CUDA implementation of the algorithm to generate tear film maps on NVIDIA GPUs, with thousands of cores and fast memory.

Acknowledgments

This research has been partially funded by the Ministry of Economy and Competitiveness of Spain and FEDER funds of the EU (Project TIN2013-42148-P), by the Galician Government and FEDER funds of the EU (ref. GRC2013/055), and by the FCT - Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) within project UID/EEA/50014/2013 and research grant 3018/BPD_B3A/2015.

We would also like to thank the Optometry Service from the University of Santiago de Compostela (Spain) and the Center of Physics from the University of Minho (Portugal) for providing us with the annotated dataset.

References

[1] R. Adams and L. Bischof. Seeded Region Growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, 1994.

[2] V. Bolón-Canedo, D. Peteiro-Barral, B. Remeseiro, A. Alonso-Betanzos, B. Guijarro-Berdiñas, A. Mosquera, M.G. Penedo, and N. Sánchez-Maroño. Interferential Tear Film Lipid Layer Classification: an Automatic Dry Eye Test. In *24th IEEE Intl. Conf. on Tools with Artificial Intelligence (ICTAI'12)*, pages 359–366, Athens, Greece, 2012.

[3] A. J. Bron. Diagnosis of Dry Eye. *Survey of Ophthalmology*, 45(2):S221–S226, 2001.

[4] J. P. Craig and A. Tomlinson. Importance of the Lipid Layer in Human Tear Film Stability and Evaporation. *Optometry & Vision Science*, 74:8–13, 1997.

[5] J. P. Guillon. Non-Invasive Tearscope plus Routine for Contact Lens Fitting. *Contact Lens & Anterior Eye*, 21 Suppl 1:31–40, 1998.

[6] P. N. Happ, R. S. Ferreira, C. Bentes, G. A. Costa, and R. Q. Feitosa. Multiresolution Segmentation: a Parallel Approach for High Resolution Image Segmentation in Multicore Architectures. In *3rd Intl. Conf. on Geographic Object-Based Image Analysis (GEOBIA'10)*, Ghent, Belgium, 2010.

[7] H. C. Kang, C. Choi, J. Shin, J. Lee, and Y. Shin. Fast and Accurate Semiautomatic Segmentation of Individual Teeth from Dental CT Images. *Computational and Mathematical Methods in Medicine*, 2014 (art. 810796), 2014.

[8] B. Remeseiro, V. Bolón-Canedo, D. Peteiro-Barral, A. Alonso-Betanzos, B. Guijarro-Berdiñas, A. Mosquera, M. G. Penedo, and Noelia Sánchez-Maroño. A Methodology for Improving Tear Film Lipid Layer Classification. *IEEE Journal of Biomedical and Health Informatics*, 18(4):1485–1493, 2014.

[9] B. Remeseiro, A. Mosquera, and M. G. Penedo. CASDES: a Computer-Aided System to Support Dry Eye Diagnosis Based on Tear Film Maps. *IEEE Journal of Biomedical and Health Informatics*, 2015. DOI: 10.1109/JBHI.2015.2419316.

[10] B. Remeseiro, M. Penas, N. Barreira, A. Mosquera, J. Novo, and C. García-Resúa. Automatic Classification of the Interferential Tear Film Lipid Layer Using Colour Texture Analysis. *Computer Methods and Programs in Biomedicine*, 111:93–103, 2013.

[11] S. Saxena, N. Sharma, and S. Sharma. An Intelligent System for Segmenting an Abdominal Image in Multi core Architecture. In *10th Intl. Conf. on Emerging Technologies for a Smarter World (CEWIT'13)*, New York, USA, 2013.

[12] S. Szenasi, Z. Vamossy, and M. Kozlovszky. GPGPU-based Data Parallel Region Growing Algorithm for Cell Nuclei Detection. In *12th IEEE Intl. Symp. on Computational Intelligence and Informatics (CINTI'11)*, pages 493–499, Budapest, Hungary, 2011.

[13] VOPTICAL_R, VARPA optical dataset acquired and annotated by optometrists from the Optometry Service of the University of Santiago de Compostela (Spain) and the Physics Center of the University of Minho (Portugal), 2015. [Online] Available: `http://www.varpa.es/voptical_r.html`.

[14] A. M. Westhoff. Hybrid Parallelization of a Seeded Region Growing Segmentation of Brain Images for a GPU Cluster. In *27th GI/ITG Intl. Conf. on Architecture of Computing Systems (ARCS'14)*, Luebeck, Germany, 2014.