Software Implementation and Hardware Acceleration of Retinal Vessel Segmentation for Diabetic Retinopathy Screening Tests

L. Cavinato, I. Fidone, M. Bacis, E. Del Sozzo, G. C. Durelli, M. D. Santambrogio
Politecnico di Milano, Milan, Italy
{lara.cavinato, irene.fidone, marco.bacis}@mail.polimi.it,
{emanuele.delsozzo, gianlucacarlo.durelli, marco.santambrogio}@polimi.it

***Abstract*— Screening tests are an effective tool for the diagnosis and prevention of several diseases. Unfortunately, in order to produce an early diagnosis, the huge number of collected samples has to be processed faster than before. In particular this issue concerns image processing procedures, as they require a high computational complexity, which is not satisfied by modern software architectures. To this end, Field Programmable Gate Arrays (FPGAs) can be used to accelerate partially or entirely the computation. In this work, we demonstrate that the use of FPGAs is suitable for biomedical application, by proposing a case of study concerning the implementation of a vessels segmentation algorithm. The experimental results, computed on DRIVE and STARE databases, show remarkable improvements in terms of both execution time and power efficiency (6X and 5.7X respectively) compared to the software implementation. On the other hand, the proposed hardware approach outperforms literature works (3X speedup) without affecting the overall accuracy and sensitivity measures.**

I. Introduction

Nowadays, several applications in the biomedical field have to guarantee different requirements. Indeed, while the accuracy remains the main requirement, performance, in terms of execution time, is becoming relevant as well. For instance, this aspect is critical when we consider applications, like screening tests, that require to elaborate a huge amount of data from a database. In this case, a faster execution could improve efficiency and produce earlier diagnoses.

In order to speed up the computation, a possible solution consists in employing hardware accelerators to increase the performance of this kind of applications. Indeed, since most of these algorithms are based on image processing techniques, they may drastically benefit from a hardware acceleration on devices like Field Programmable Gate Arrays (FPGAs). In particular, FPGAs are highly suitable for such applications, where they offer a better performance-per-watt ratio with respect to Central Processing Units (CPUs) and Graphics Processing Units (GPUs). Moreover, thanks to their reconfiguration capabilities, FPGAs result more flexible than Application-Specific Integrated Circuits (ASICs).

The purpose of this work is to emphasize the benefits of a hardware acceleration on FPGA for biomedical applications, in terms of both execution time and power consumption. To this end, we selected as case study a biomedical application that leverages *vessels* segmentation techniques. This technique is used along with preventive screening techniques to detect ocular diseases, such as diabetic retinopathy and degenerative maculopathy, and other pathologies. This kind of application is particularly suited for diseases as diabetic retinopathy which is a consequence of diabetes and it is caused by a metabolic decompensation. We aim to accelerate a vessels segmentation algorithm in order to streamline the preprocessing phase of a bigger screening algorithm, which can require a high time processing. Two public databases have been used to evaluate our work: DRIVE [1] and STARE [2]. Both databases are made up of 20 images (dimensioned respectively 768x584 pixels and 605x700 pixels). Each image has its corresponding processed one [3].

This work intends to make the following contributions:

- To demonstrate that the chosen platform is suitable for biomedical applications, embeddable within existing instrumentation or prone to be used in embedded applications, thanks to its performing features.
- To highlight the advantages of a hardware implementation for biomedical applications in term of computational speed, which becomes critical when dealing with large databases (where results are still needed in a reasonable time to make prompt decision) and for monitoring of patients (where the resolution is dependent on the speed of the computation in the chosen architecture).
- To use the vessel segmentation as a case study, with an accent on the screening for diabetic retinopathy, highlighting the advantages in an automatic monitoring both in public and private infrastructures.

The rest of the paper is organized as follows. Section II discusses the literature on software and hardware implementations of retinal vessel segmentation algorithms. Section III describes in details the proposed algorithm and methodology, while Section IV presents the experimental evaluation. Finally, Section V draws the conclusions.

II. Related work

Different implementations of retinal vessel segmentation algorithms taken from machine learning field are available in the state of the art. These algorithms can be clustered into few main categories [3]: *Pattern recognition techniques* focus on identifying regularities in data in order to classify retinal blood vessel features and other non vessel objects including background. *Tracking-based methods* found the extraction of vessels on statistical morphological operators. *Mathematical morphology techniques* adopt topological and geometrical continuous-space concepts. *Multi-scale approaches* are based

978-1-5090-2809-2/17/$31.00 ©2017 IEEE

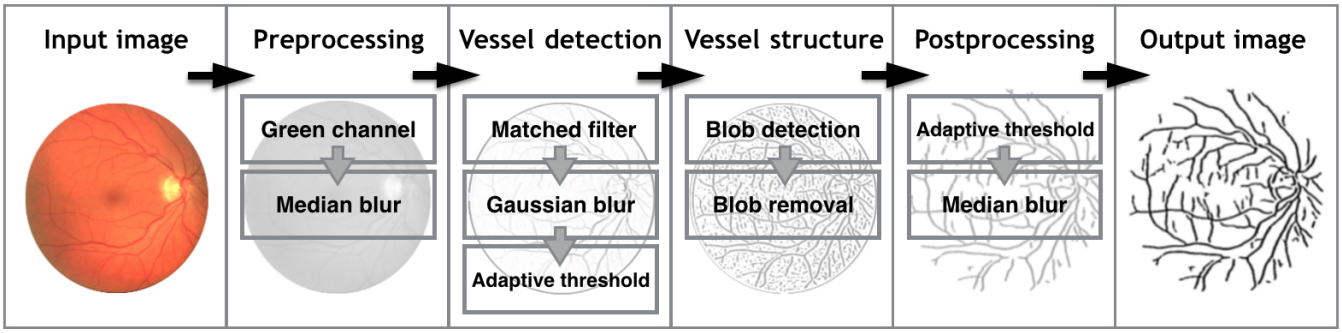

Fig. 1. Proposed algorithm steps and filters

on widths gradual decrease from the optic disc [4]. *Model based approaches* apply an explicit vessel models (classical or deformable) to extract the retinal vessels [5]. *Matched filters* involve the convolution of a 2D-kernel with the image, in order to indicate the presence of a given feature. *Neural networks* are a graph based approach, which detects the vessels structures as a human brain would do.

Our algorithm can be classified as a matched filter. This technique has been chosen due to the high performance it provides and its structure which is very suitable for an FPGA. Matched filter has already been exploited in several works. In [6], the authors apply a two-dimensional linear kernel with a Gaussian profile for the segmentation. Moreover, 12 different templates were used to search the shape of the vessel along the most likely directions. The authors of [7] propose a general framework for adaptive local thresholding based on a verification-based multi-threshold probing scheme. The application is based on a verification procedure that incorporates specific tests about blood vessel features. The work in [8] uses an exhaustive search optimization technique. Parameters such as matched filter size and threshold value were found by testing the algorithm on DRIVE database images. The algorithm proposed in [9] comprises a matched filter based on a zero-mean Gaussian function and its first-order derivative. The vessels are detected by thresholding the response to the matched filter, while the threshold value is adjusted using the image response to the first-order derivative. The work in [10] reports an hybrid model of matched filter and ant colony algorithm. The method proposed in [11] is based on the analysis of the phase congruency of the retinal image, which is a soft-classification of blood vessels. The phase congruency is computed using *Log-Gabor* wavelets [12]. The article in[13], a Gabor filter bank is used for the segmentation and textons are used for the filter responses. The corresponding functions are used as the framework for the classification into vessel and non-vessel classes. The work described in [14] is based on a matched filter technique as well. An illumination correction and contrast equalization are applied before the matched filter and followed by morphological operators.

While the works previously mentioned propose software implementations, other works focus on accelerating vessel segmentation on matched filters using hardware designs.

For instance, the work in [15] reports a hardware implementation on Xilinx Spartan 3 FPGA [16]. The presented algorithm, based on [17], deals with pixel-level snakes, a parallel active contour technique inspired by energy-based deformable models. The authors of [18] present an implementation, on a Xilinx Spartan 6 board [19], of an algorithm derived from the work in [6]. The implementation is done using several optimizations, such as parallelization and resource-sharing, in order to perform a real-time analysis.

III. Proposed Methodology

The proposed methodology consists in a pipeline of image processing techniques that process an input retinal image in order to detect the blood vessels. Following this methodology, we implemented an algorithm for retinal vessel segmentation which comprises four main steps: *preprocessing*, *vessel detection*, *vessel structure isolation* and *postprocessing*, as reported in Figure 1. Each of these main steps then involves a series of different convolutional and matched filters.

A. Preprocessing

The preprocessing step prepares the image by removing the noise before the detection step. At first, this step loads the image and splits it into its 3 color channels. The step just keeps the green channel, because it shows the best contrast features for vessel segmentation [20]. On the other hand, speckle noise, which is composed by pixels with a value distant from the near ones, may be present into the input image. In order to remove this kind of noise, the prepocessing step applies a 3x3 median blur filter.

B. Vessel Detection

The vessel detection step is composed of three different filters: matched, Gaussian and adaptive threshold filter.

At first, this step applies a particular version of the matched filter, derived from works in [6] and [18]. Such filter resembles an edge detection filter and convolves the retinal image by means of 13 different 16x16 kernels in order to keep the maximum response. More technically, each kernel derives from a basic one (defined in [6]) rotated by different degrees (from 0 to 180 in 15 degrees steps). The basic kernel is a matrix composed by a series of values, aligned in order to match the vertical direction of a vessel.

The different rotations of the kernel are necessary to match the image with different directions. Finally, the maximum response represents a merge of the different filters, in order to detect the vessels going in every possible direction.

Then, the vessel detection step applies a Gaussian blur filter, in order to reduce the image sharpness.

Finally, the step applies a 9x9 adaptive threshold filter to the maximum response, and produces a binary image as output. The threshold is computed for each pixel as the average of the neighboring pixels (in a 9x9 window). In this way, each pixel is classified as *background*, if its value is lower than the threshold, or *vessel* otherwise.

C. Vessel Structure Isolation

The resulting image from the vessel detection step has a large number of small marks/spots, which represent false positives. In the vessel structure isolation step the spots are removed by means of a *blob detection* algorithm, followed by a removal step. This algorithm works by detecting each group of connected vessel pixels. To detect and compute the area of each group, the algorithm analyzes each pixel upper and left neighbor. If one of these two neighbors belongs to an existing group, the current pixel is assigned to one of them. Otherwise, if there are no upper and left pixels, a new group is created and linked to the current pixel. In case of conflicts between the neighboring pixels groups, the groups are linked together.

Then, the *blob removal* step removes the groups with an area smaller than a threshold (chosen in order to maximize the accuracy in a given set of images).

D. Postprocessing

The postprocessing step purpose is to enhance the image resulting from the previous steps. At first, this step applies a 11x11 adaptive threshold filter to sharpen the details from the previous steps. Then, the postprocessing step applies a last 5x5 median blur in order to remove the remaining noise from the final image, composed of single-pixel errors.

For the software implementation, there are several computer vision libraries (such as *OpenCV* [21]) that can be used to rapidly develop the filters. As for the hardware development, each filter can be designed as a distinct hardware module. Then, in order to create a pipeline of computation, the modules are arranged in a chain, and the images are elaborated with a *streaming* oriented approach. This helps to reduce the required memory, and to speed up the execution.

IV. Experimental results

This section describes the experimental results. Section IV-A presents the implementation and experimental setup of both software and hardware versions of our algorithm; Section IV-B discusses the performance of these two implementations; Sections IV-C and IV-D compare our algorithm performance with software and hardware implementations in literature, respectively. The Key Performance Indicators (KPIs) used in the evaluation are the accuracy and the latency of the implementation. The accuracy is the ratio between the correctly classified pixels count and the total number of pixels in the image. The latency represents the average time to process a picture, and will be expressed both as a time value and as the achieved speed up with respect to an identical software implementation run on a high-performance CPU and to other implementations in the state of the art.

TABLE I

Comparison on execution time and power consumption between software and hardware proposed implementations

Implementation	Device	Execution time	Power consumption
Software	Intel Core i7	0.06806 *s*	26.929 *Watt*
Hardware	Zedboard	0.01041 *s*	4.749 *Watt*

TABLE II

Software implementations comparison (on DRIVE and STARE)

Work	Accuracy		Sensitivity		Time	
	DRIVE	STARE	DRIVE	STARE	DRIVE	STARE
[7]	–	0.9337	–	–	–	19 *s*
[8]	0.9535	–	–	–	11 *s*	–
[9]	0.9382	0.9484	0.7120	0.7177	–	10 *s*
[10]	0.9293	–	–	–	35 *s*	–
[11]	0.8900	0.9093	–	0.8035	2 *s*	–
[13]	0.9430	–	0.7673	–	–	
[14]	0.9340	0.9341	0.7060	0.7847	3.22 *s*	4.07 *s*
P.A.	0.9293	0.9030	0.6457	0.7291	0.068 *s*	0.073 *s*

TABLE III

Hardware implementations comparison

Work	Device	Accuracy	Execution time	Frequency
[15]	Spartan 3	0.9100	1.40000 *s*	53 *MHz*
[18]	Spartan 6	0.9007	0.03185 *s*	100 *MHz*
P.A.	Zedboard	0.9285	0.01041 *s*	100 *MHz*

A. Implementation and Experimental Setup

The Proposed Algorithm (P.A.) has been implemented both in software and in hardware. The software implementation has been written in C++, using OpenCV [21] libraries with CPU-specific optimizations. The test was run on a Intel Core i7-6700 CPU [22]. On the other hand, the target architecture for the hardware implementation is an *Avnet Zedboard* [23] powered by a Xilinx Zynq-7000 All Programmable System on Chip (APSoC). The chip includes both a hardwired Dual-Core ARM Cortex A9 [24] and a Xilinx Series-7 FPGA. This APSoC allows to have a CPU and FPGA collaboration, which means that the ARM processor is used for I/O operations while it delegates the filters computations to the FPGA chip. The hardware design has been developed using *Xilinx Vivado Design Suite* [25] (Vivado, Vivado HLS and Vivado SDK, version 2015.3). Finally, we measured power consumption by means of Intel *Power Gadget* [26] and *Energy Logger 4000* by Voltcraft for the Intel Core i7-6700 CPU and Zedboard platform, respectively.

B. Software and Hardware Implementation Analysis

Table I reports the performance of software and hardware implementations, in terms of power consumption of the target devices and execution time for a generic image from DRIVE database. The software implementation takes 0.06 seconds to compute each image, while the average power consumption is 26.929 Watt. On the other hand, the hardware implementation outperforms the software one, with an average execution time of 0.01 seconds and a chip-level (CPU and FPGA) average power consumption of 4.749 Watt. Therefore, the hardware implementation is 6 times faster than the same algorithm executed on a high-performance CPU, and 5.7 times more efficient in terms of power consumption.

C. Software Implementation Comparison

Table II reports the average values of accuracy and sensitivity, as well as the execution time of both related works and the software version of the proposed approach. The average accuracy of the proposed algorithm, for DRIVE and STARE databases, is respectively *0.9293* and *0.9030*, whereas the obtained sensitivity results are *0.6457* and *0.7291*. Even though the values of accuracy and sensitivity result few points lower than those obtained in the other works, the strength of our work is the speed of the computation, significantly faster than the works in the state of the art. In fact, accuracy values depend on the chosen algorithm, which can be better by means of optimization in future.

D. Hardware Implementation Comparison

Finally, Table III compares accuracy and hardware performance of the proposed implementation with respect to the hardware ones. Proposed hardware implementation outperforms [15] and [18] in terms of both accuracy (0.9285 vs 0.91 and 0.9) and execution time (0.01 vs 1.4 and 0.03).

V. Conclusions

Retinal vessel segmentation is a challenging task and the results of the extraction of the blood vessel structure are crucial in order to diagnose diseases like diabetes. To this end, this work presents an hardware implementation of a vessel segmentation algorithm, whose performance makes this work the one that can make way for using FPGAs in biomedical and screening applications. We managed to find a trade-off between accuracy and performance, in order to speed up the process without losing precision.

The possibility to have a very low computation time, allows to cut down on waiting times by providing a support for real-time diagnosis. Finally, in our opinion, we have demonstrated that hardware implementations and, in particular, FPGAs are suitable for biomedical applications as they can provide a high throughput in large databases and real time processing.

References

[1] M. Viergever, "DRIVE database." [Online]. Available: http://www.isi.uu.nl/Research/Databases/DRIVE/

[2] M. Goldbaum, "STARE database." [Online]. Available: http://www.ces.clemson.edu/~ahoover/stare/

[3] M. M. Fraz, P. Remagnino, A. Hoppe, B. Uyyanonvara, A. R. Rudnicka, C. G. Owen, and S. A. Barman, "Blood vessel segmentation methodologies in retinal images–a survey," *Computer methods and programs in biomedicine*, vol. 108, no. 1, pp. 407–433, 2012.

[4] Q. Li, J. You, L. Zhang, and P. Bhattacharya, "A multiscale approach to retinal vessel segmentation using gabor filters and scale multiplication," in *Systems, Man and Cybernetics, 2006. SMC'06. IEEE International Conference on*, vol. 4. IEEE, 2006, pp. 3521–3527.

[5] M. M. Fraz, P. Remagnino, A. Hoppe, S. A. Barman, A. Rudnicka, C. Owen, and P. Whincup, "A model based approach for vessel caliber measurement in retinal images," in *Signal image technology and internet based systems (SITIS), 2012 eighth international conference on*. IEEE, 2012, pp. 129–136.

[6] S. Chaudhuri, S. Chatterjee, N. Katz, M. Nelson, and M. Goldbaum, "Detection of blood vessels in retinal images using two-dimensional matched filters," *IEEE Transactions on medical imaging*, vol. 8, no. 3, pp. 263–269, 1989.

[7] X. Jiang and D. Mojon, "Adaptive local thresholding by verification-based multithreshold probing with application to vessel detection in retinal images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 25, no. 1, pp. 131–137, 2003.

[8] M. Al-Rawi, M. Qutaishat, and M. Arrar, "An improved matched filter for blood vessel detection of digital retinal images," *Computers in Biology and Medicine*, vol. 37, no. 2, pp. 262–267, 2007.

[9] B. Zhang, L. Zhang, L. Zhang, and F. Karray, "Retinal vessel extraction by matched filter with first-order derivative of gaussian," *Computers in biology and medicine*, vol. 40, no. 4, pp. 438–445, 2010.

[10] M. G. Cinsdikici and D. Aydın, "Detection of blood vessels in ophthalmoscope images using mf/ant (matched filter/ant colony) algorithm," *Computer methods and programs in biomedicine*, vol. 96, no. 2, pp. 85–95, 2009.

[11] M. A. Amin and H. Yan, "High speed detection of retinal blood vessels in fundus image using phase congruency," *Soft Computing*, vol. 15, no. 6, pp. 1217–1230, 2011.

[12] S. Fischer, F. Šroubek, L. Perrinet, R. Redondo, and G. Cristóbal, "Self-invertible 2d log-gabor wavelets," *International Journal of Computer Vision*, vol. 75, no. 2, pp. 231–246, 2007.

[13] L. Zhang, M. Fisher, and W. Wang, "Retinal vessel segmentation using gabor filter and textons," *Medical Image Understanding and Analysis (MIUA 2014)*, pp. 155–160, 2014.

[14] J. Odstrcilik, R. Kolar, A. Budai, J. Hornegger, J. Jan, J. Gazarek, T. Kubena, P. Cernosek, O. Svoboda, and E. Angelopoulou, "Retinal vessel segmentation by improved matched filtering: evaluation on a new high-resolution fundus image database," *IET Image Processing*, vol. 7, no. 4, pp. 373–383, 2013.

[15] A. Nieto, V. M. Brea, and D. L. Vilariño, "Fpga-accelerated retinal vessel-tree extraction," in *Field Programmable Logic and Applications, 2009. FPL 2009. International Conference on*. IEEE, 2009, pp. 485–488.

[16] "Xilinx Spartan 3." [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/spartan-3.html

[17] C. Alonso-Montes, D. Vilarino, P. Dudek, and M. Penedo, "Fast retinal vessel tree extraction: A pixel parallel approach," *International Journal of Circuit Theory and Applications*, vol. 36, no. 5-6, pp. 641–651, 2008.

[18] D. Koukounis, C. Tttofis, and T. Theocharides, "Hardware acceleration of retinal blood vasculature segmentation," in *Proceedings of the 23rd ACM international conference on Great lakes symposium on VLSI*. ACM, 2013, pp. 113–118.

[19] "Xilinx Spartan 6." [Online]. Available: https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html

[20] M. Crittin, H. Schmidt, and C. E. Riva, "Hemoglobin oxygen saturation (so2) in the human ocular fundus measured by reflectance oximetry: preliminary data in retinal veins," *Klinische Monatsblätter für Augenheilkunde*, vol. 219, no. 04, pp. 289–291, 2002.

[21] "OpenCV library." [Online]. Available: http://opencv.org/

[22] "Intel Core i7-6700." [Online]. Available: http://ark.intel.com/it/products/88196/Intel-Core-i7-6700-Processor-8M-Cache-up-to-4_00-GHz

[23] "Zedboard." [Online]. Available: http://zedboard.org/product/zedboard

[24] "ARM Cortex A9." [Online]. Available: http://www.arm.com/cortex-a9.php

[25] Xilinx, "Vivado Suite." [Online]. Available: http://www.xilinx.com/products/design-tools/vivado.html

[26] "Intel Power Gadget." [Online]. Available: https://software.intel.com/en-us/articles/intel-power-gadget/