Nieto et al. EURASIP Journal on Image and Video Processing 2011, 2011:10
http://jivp.eurasipjournals.com/content/2011/1/10

EURASIP Journal on Image and Video Processing
a SpringerOpen Journal

RESEARCH Open Access

Performance analysis of massively parallel embedded hardware architectures for retinal image processing

Alejandro Nieto[1*], Victor Brea[1], David L Vilariño[1] and Roberto R Osorio[2]

Abstract

This paper examines the implementation of a retinal vessel tree extraction technique on different hardware platforms and architectures. Retinal vessel tree extraction is a representative application of those found in the domain of medical image processing. The low signal-to-noise ratio of the images leads to a large amount of low-level tasks in order to meet the accuracy requirements. In some applications, this might compromise computing speed. This paper is focused on the assessment of the performance of a retinal vessel tree extraction method on different hardware platforms. In particular, the retinal vessel tree extraction method is mapped onto a massively parallel SIMD (MP-SIMD) chip, a massively parallel processor array (MPPA) and onto an field-programmable gate arrays (FPGA).

1 Introduction

Nowadays, medical experts have to deal with a huge volume of information hidden in medical images. Automated image analysis techniques play a central role in order to ease or even to remove manual analysis. The development of algorithms for medical image processing is one of the most active research areas in Computer Vision [1]. In particular, retinal blood vessel evaluation is one of the most used methods for early diagnosis to determine cardiovascular risk or to monitor the effectiveness of therapies [2]. A lot of effort has been devoted to the development of techniques that extract features from the retinal vessel tree and to measure parameters as the vessel diameter [3], tortuosity [4] or other geometrical or topological properties [5].

From the image processing point of view, special features of retinal images, such as noise, low contrast or gray-level variabilities along the vessel structures, make the extraction process highly complex. Different approaches to extract the retinal vessel tree or just some specific features with relevant information for the experts have been proposed [6-9].

In all the applications, accuracy is a requirement. However, in some of them, the computational effort is also the main issue. In this sense, a new technique was proposed in Alonso-Montes et al. [9]. This algorithm was designed specifically for its utilization in fine-grained SIMD architectures with the purpose of improving the computation time. It uses a set of active contours that fit the external boundaries of the vessels and support automatic initialization of the contours, avoiding human interaction in all the process. This solution provides reliable results because the active contours are initialized outside the vessels region, so narrow vessels can be extracted in an accurate way. The algorithm has been tested on a massively parallel processor, which features a correspondence of a processor-per-pixel. This solution provides the highest performance. However, when using real devices, we have to face certain limitations imposed by the technology (i.e. integration density, noise or accuracy), so the results are worse than expected [9]. At this point, other a priori less suitable solutions can provide similar or even better performance.

The algorithm can process the image quickly and efficiently, making it possible to operate online. This speeds up the work of the experts because it allows not only to have immediate results but also they can change parameters in real-time observation, improving the

* Correspondence: alejandro.nieto@usc.es
[1]University of Santiago de Compostela, Centro de Investigación en Tecnoloxías da Información (CITIUS), Santiago de Compostela, Spain
Full list of author information is available at the end of the article

diagnostic. It also reduces the cost of the infrastructure as it is not necessary to use workstations for processing. The algorithm can be integrated into a device with low cost, low form factor and low power consumption. This opens the possibility of using the algorithm outside the medical field, for example, in biometric systems [10].

Although the algorithm was designed for massively parallel SIMD (MP-SIMD) processors, it can be also migrated to other devices. DSPs or GPUs provide good results in common image-processing tasks. However, reconfigurable hardware or custom ASICs solutions permit to improve the matching between architecture and image-processing algorithms, exploiting the features of vision computing, and thus potentially leading to better performance solutions. In this study, we want to analyze devices that allow us to fully integrate the entire system on an embedded low power device. The algorithm described here is designed to operate on-line, immediately after the stage of image capture and integrated into the system, and not for off-line processing, so we select devices that allow this kind of integration. DSPs are a viable solution, but we cannot take advantage of the massively parallelism of the algorithm. On the other hand, the high power consumption of GPUs discards these for standalone systems.

Among the plethora of different platforms that today offer hardware reconfigurability, this paper focuses on the suitability of field- programmable gate arrays (FPGAs) and massively parallel processor arrays (MPPA) for computer vision. FPGAs are widely used as prototyping devices and even final solutions for image-processing tasks [11,12]. Their degree of parallelism is much lower than what an MP-SIMD provides, but they feature higher clock frequencies and flexible data representations, so comparable results are expected. Advances in the miniaturization of the transistors allow higher integration densities, so designers can include more and more features on their chips [13]. MPPAs are a clear example of this because until a few years ago, it was not possible to integrate several hundred microprocessors, even if they were very simple. These devices are characterized by a different computation paradigm, focusing on exploiting the task parallelism of the algorithms [14].

In this paper, the automatic method to extract the vessel tree from retinal images presented in Alonso-Montes et al. [9] was tested on an FPGA and an MPPA, and the results were compared with the native platform of the algorithm, an MP-SIMD processor.

The paper is organized as follows. Section 2 describes the retinal vessel tree extraction algorithm. Sections 3, 4 and 5 detail both the particular implementation of the algorithm and the architectures where it is implemented. Section 6 summarizes the results and conveys the main conclusions.

2 The retinal vessel tree extraction algorithm

The retinal vessel tree extraction algorithm was proposed by Alonso-Montes et al. [9]. This technique uses a set of active contours that fit the external boundaries of the vessels. This is an advantage against other active contour-based techniques which start the contour evolution from inside the vessels. This way, narrow vessels are segmented without breakpoints, providing better results. In addition, automatic initialization is more reliable, avoiding human interaction in the whole process. Figure 1 shows the result of applying the algorithm to a retinal image. Figure 2 summarizes the necessary steps to perform this task. It should be noted that although the images are represented in color only, the green channel is used, so the algorithm behaves as if it was processing gray-scale images.

An active contour (or snake) is defined by a set of connected curves which delimit the outline of an object [15]. It may be visualized as a rubber band that will be deformed by the influence of constraints and forces trying to get the contour as close as possible to the object boundaries. The contour model attempts to minimize the energy associated to the snake. This energy is the sum of different terms:

- The internal energy, which controls the shape and the curvature of the snake.
- The external energy, which controls the snake movement to fit the object position.

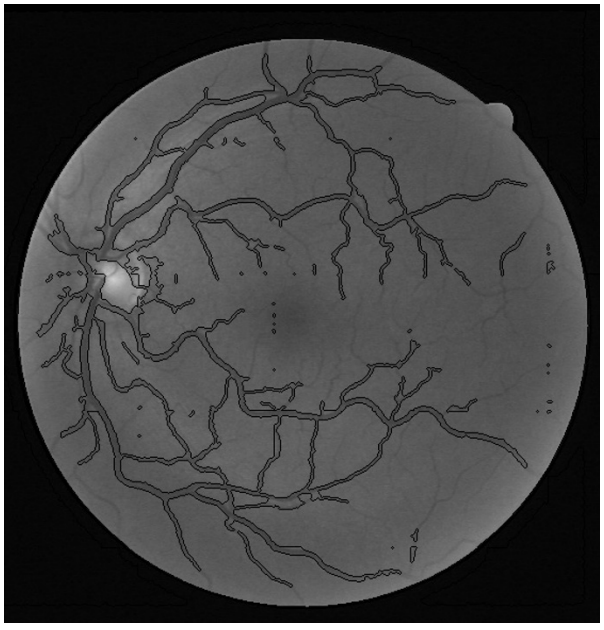

Figure 1 Retinal vessel tree extraction algorithm applied over a test image.

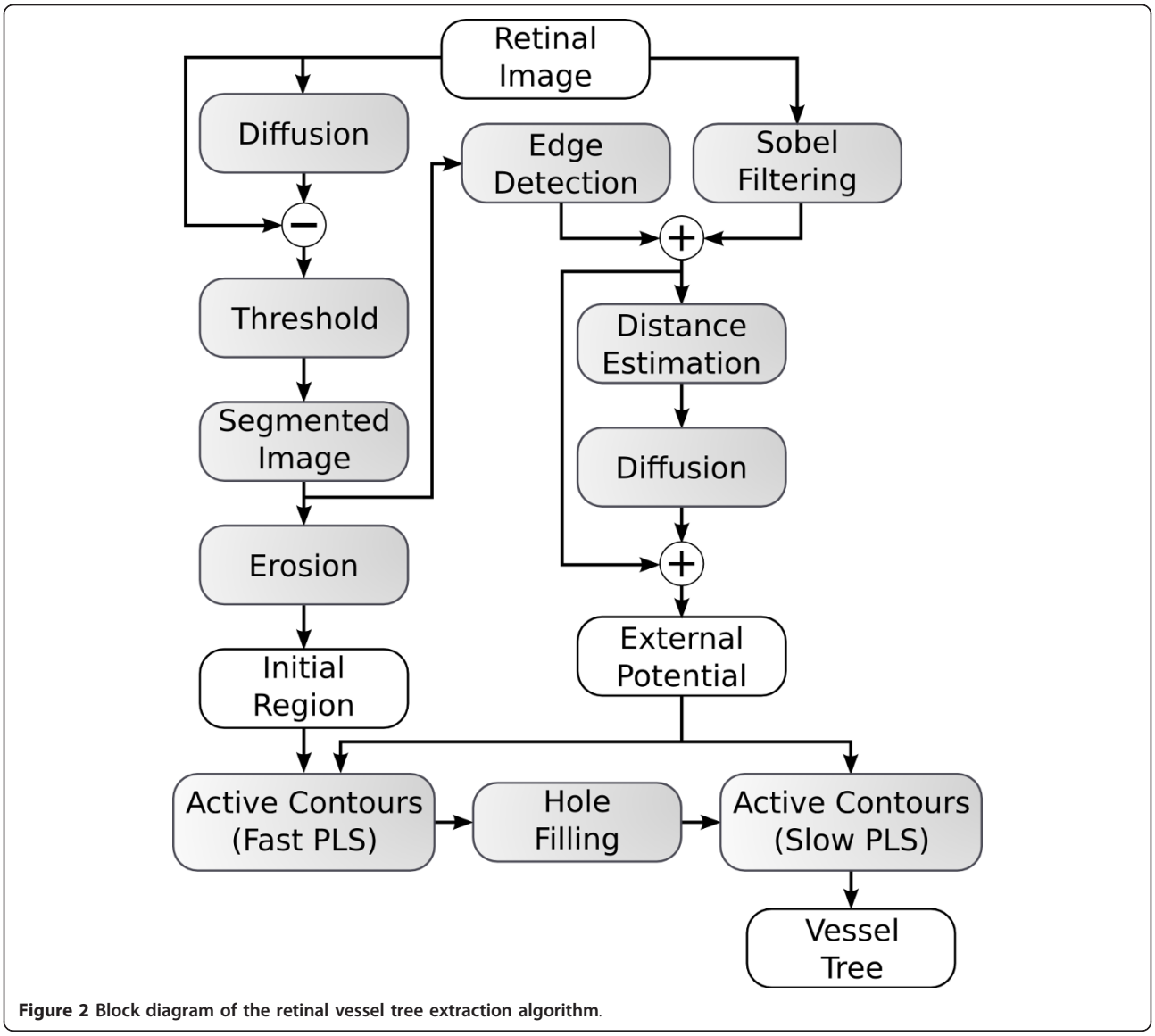

Figure 2 Block diagram of the retinal vessel tree extraction algorithm.

• Other energies included with the aim of increasing the robustness, derived from potentials (as the so-called inflated potential) or momenta (as the moment of inertia) [16].

The snake will reach the final position and shape when the sum of all these terms reaches a minimum. Several iterations are normally required to find this minimum. Each step is computationally expensive, so the global computational effort is quite high. Also, the placement of the initial contour is very important in order to reduce the number of intermediate steps (lower computational load) and to increase the accuracy (less likely to fall into a local minimum). Although they can fit to local minima of energy positions instead of the real contour location and an accurate convergence criteria requires longer computation times, such techniques are widely used in image-processing tasks. Snakes or active contours offer advantages as easy manipulation with external forces, autonomous and self-adapting and tracking of several objects at a time.

There are several active contour models. Among the plethora of different proposals, the so-called Pixel-Level Snakes (PLS) [17] was selected. This model represents the contour as a set of connected pixels instead of a higher-level representation. In addition, the energy minimization rules are defined taking into account local data. This way, it will perform well in massively parallel processors because of its inherent parallelism. The algorithm operation is divided into two main steps: (1) initialize the active contours from an initial estimation of the position of vessels and (2) evolve the contour to fit the vessels.

2.1 Active contours initialization and algorithm execution flow

One of the most important steps in active contours is initialization. As it was detailed before, two input images are needed: the initial contour from which the algorithm will evolve and the guiding information, i.e., the external potential. Figure 2 summarizes this process.

The first task is intended to reduce noise and pre-estimate the vessels boundaries, from which the initial contours will be calculated. In so-doing, adaptive segmentation is performed, subtracting a heavy diffused version of the retinal image itself followed by a threshold by a fixed value, obtaining a binary map. To ensure that we are outside of the vessels location, some erosions are applied. The final image contains the initial contours.

The second task is to determine the guiding information, i.e., the external potential. It is estimated from the original and the pre-estimation vessels location images (calculated in the previous task). An edge-map is obtained by combining the boundaries extracted from those images. Dilating several times this map, diffusing the result and combining it with the original boundaries estimation will produce the external potential. It actually represents a distance map to the actual vessels position.

These two tasks are done only once. External potential is a constant during all the process. Once the active contours image is obtained, it is updated during the evolution steps.

As Figure 2 shows, PLS is executed twice for this concrete application. During the *fast PLS*, topological transformations are enabled so the active contours can be merged or split. This operation is needed to improve accuracy to remove isolated regions generated by the erosions required for the initial contour estimation. In this stage, the inflated potential is the main responsible of the evolution because the contour is far from the real vessels location and the rest of potentials are too weak to carry out this task. The aim of this stage is to evolve the contour to get it close to the vessels. It is called *fast* because a small number of iterations are needed. During the second PLS iteration, the *slow PLS*, topological transformations are disabled. The external potential is now in charge of the guidance of the contour evolution and the internal potential prevents the evolution through small cavities or discontinuities in the vessels topology. The accuracy of the result depends deeply on this stage, so a higher number of iterations are needed (*slow evolution*). Between both stages, a hole-filling operation is included in order to meet greater accuracy, removing isolated holes inside the active contours.

2.2 Pixel-Level Snakes

It was commonly said that an active contour is represented as a spline. However, the approach selected here, the PLS, is a different technique. Instead of a high-level representation of the contour, this model uses a connected set of black pixels inside a binary image to represent the snake (see Figure 1). We must note that a *black pixel* means a pixel *activated*, i.e., an active pixel of the contour. The contours evolve through an activation and deactivation of the contour pixels through the guidance of potential fields. This evolution is controlled by simple local rules, so high performance can be achieved even in pure-software implementations. Its natural parallelism eases hardware implementations and it is one of its main advantages.

Figure 3 shows the main blocks of the PLS. First of all, the different potential fields must be computed.

- The external potential is application dependent, so it must be an external input. This was discussed previously in this section. It is constant during all the evolution.
- The internal potential is calculated from the current state of the contour. Then it is diffused several times to obtain a topographic map that helps avoid abrupt changes in the shape of the contour.
- The inflated potential simply uses the current active contour, without any change. It produces inflating forces to guide the contour when the other potentials are too weak as is the case when the boundaries are trapped in local minima.

The involved potentials are weighed, each one by an application-dependent parameter, and added to build the global potential field. Active contours evolve in four directions: north, east, west and south (NEWS). Next algorithm steps are dependent on the considered direction, so four iterations are needed to complete a single evolution step.

The next step is to calculate a collision mask. The collision detection module enables topographic changes when two or more active contours collide. Topographic changes imply contour merging and splitting. This module uses a combination of morphological hit-and-miss operations, so only local access to neighbors is needed. The obtained image that contains pixels are forbidden to be accessed in the current evolution.

During the guiding forces extraction step, a directional gradient is calculated from the global potential field. As this is a non-binary image, a thresholding operation is needed to obtain the pixels to which the contour will evolve. At this point, the mask obtained from the collision detection module is applied.

Input: Initial contour (C),
External potential (EP)
Output: Resulting contour (C)
C = `HoleFilling` (C)

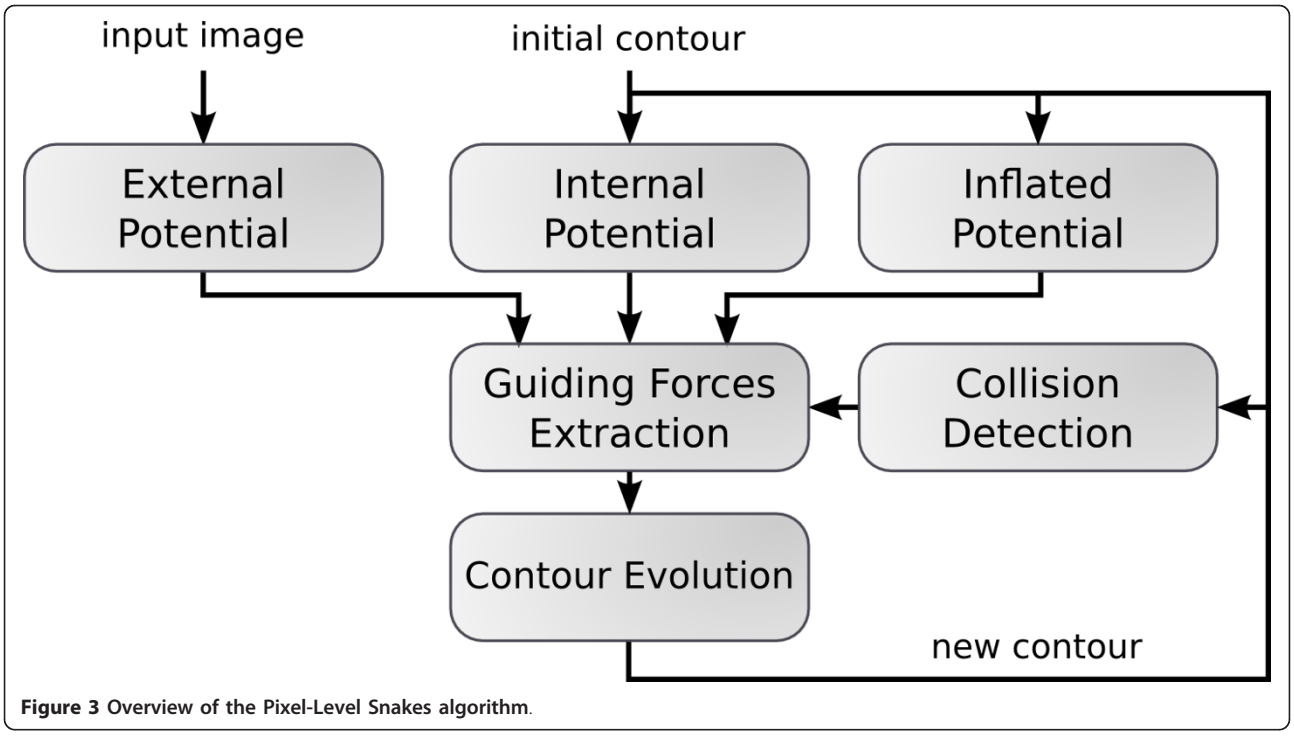

Figure 3 Overview of the Pixel-Level Snakes algorithm.

```
for i = 0...iterations do
  IP = InternalPotential (contour)
  foreach dir 2 (N, E, W, S) do
    IF = InflatedPotential (C)
    CD = CollisionDetection (C, dir)
    GF = GuidingForce (EP, IP, IF, CD, dir)
    C = ContourEvolution (GF, C, dir)
  end
end
C = BinaryEdges (C)
function InternalPotential (C)
aux = BinaryEdges (C)
IP = smooth (aux, times)
return IP
function InflatedPotential (C)
IF = C
return IF
function CollisionDetection (C, dir)
if enable then
  if dir = N then
    aux1 = shift (C, S) andnot C;
    aux2 = shift (aux1, E);
    aux3 = shift (aux1, W);
    CD = aux1 or aux2 or aux3;
  else
    % Other directions are equivalent
  end
else
  CD = zeros ()
end
return CD
function GuidingForce (EP, IP, IF, CD, dir)
aux1 = EP + IP + IF
aux2 = aux1 shift (aux1, dir)
aux3 = threshold (aux2, 0)
GF = aux3 andnot CD
return GF
function ContourEvolution (GF, C, dir)
aux = shift (C, dir) and GF
C = C or aux
return C
```

Algorithm 1: Pixel-Level Snakes. All variables are images. All operations are performed over all pixels of the image before the execution continues.

The final step is to perform the evolution itself (contour evolution module). The active contour is dilated in the desired direction using the information from the guiding forces extraction module.

Except when the potentials are involved, all the operations imply only binary images, so computation uses only Boolean operations. The pseudocode in Algorithm 1 shows all the steps. We have to remark that all the variables (except the iterators) are images, two-dimensional arrays. Each operation has to applied over all the pixels of the image before continuing with the next operation.

This is an adapted version of the PLS for the retinal vessel tree extraction algorithm. There are only stages of

expansion. The way in which the contour is initialized ensures that alternating phases of expansion/contraction required in any other active contours method is not necessary here, which simplifies the code and increases performance. Further details of this active contours-based technique can be found in Vilariño and Rekeczky [17], Dudek et al. [18], Vilarino and Dudek [19].

2.3 Performance remarks

The inherent parallelism of the retinal vessel tree algorithm makes its hardware implementation simple with high performance. Finally, the image can be split into multiple subwindows and can be processed independently. In addition, the required precision for the data representation is low (see [9]), so the accuracy will not be seriously affected by this parameter. All these advantages will be exploited during the algorithm port to the hardware platforms of this review. One of the drawbacks of this extraction method is that it is hard to exploit temporal parallelism. However, the heavy computational effort comes from the PLS evolution, where each iteration directly depends on the previous one, forcing to execute all the steps serially.

Table 1 summarizes the type of operations present in the algorithm per pixel of the image and iteration of the given task. Table 2 sums up the total number of operations including the number of iterations per task and program flow-related tasks. The number of iterations was determined experimentally and agrees with the worst case of those studied to ensure the convergence of the contours. Considering that the input image is 768 × 584 px and that by each pixel 6846 operations must be performed, around 3 GOPs are required to process the entire image. The operations of this algorithm are very representative of image-processing operations which are part of the low- and mid-level stages. They comprise operations as filtering, basic arithmetics, logic operations, mask applications or basic program flow data dependencies. Any image-processing-oriented hardware must deal properly with all these tasks.

The retinal vessel tree extraction algorithm was tested employing a PC-based solution. It was developed using OpenCV and C++ on a computer equipped with an Intel Core i7 940 working at 2.93 GHz (4 physical cores running 8 threads) and 6 GB of DDR3 working at 1.6 GHz and in triple-channel configuration. To evaluate the efficiency of the implementation, the DRIVE database was used [20]. The retinal images were captured with a Canon CR5 non-mydriatric 3CCD. They are 8-bit three channel color images with a size of 768 × 584 px. Using this computer, each image requires more than 13 s to be processed. This implementation makes use of the native SSE support which OpenCV offers. To take advantage of the multi-core CPU, OpenMP was used to parallelize loops and some critical blocks of the algorithm which are implemented outside the OpenCV framework. This allows us to obtain around a 15% higher performance. Although it would be possible to do certain optimizations to further improve performance, it would be very difficult to achieve times under 10 s, which takes us away from our goal. This is because the algorithm is not designed to run on such architectures, not because of the algorithmic complexity, but due to the large number of memory accesses required and that are not present in focal-plane processors.

Even with a high-end computer, the result is not satisfactory in terms of speed. Candidate systems using this algorithm require a faster response. To address this and other problems associated with a conventional PC, such as size or power consumption, we propose three implementations on three specific image-processing devices: a Vision Chip, a custom architecture on FPGA and a MPPA. From the characteristics of the algorithm, it is extracted that (by its nature) an MP-SIMD architecture matches better. We also test the capabilities of the reconfigurable hardware on FPGAs, which increasingly provides more features. Finally, using the MPPA, we check whether exploiting the task parallelism instead of its massive data parallelism also provides good results.

Table 1 Type and number of operations per pixel per step of each task

	Initialization	Fast PLS	Hole filling	Slow PLS
Pixel-to-pixel				
Arithmetic	7	5	-	7
Boolean	1	4	-	9
Pixel-to-neighborhood				
2D filters	11	8	-	8
Binary masks	10	2	1	5

Table 2 Number of operations per pixel

	# iterations	Operations per px
Initialization	1	189
Fast PLS	6	697
Hole filling	18	199
Slow PLS	40	5,761
		6,846

Pixel-to-neighborhood operations shown in Table 1 are transformed to pixel-to-pixel operations

This includes program flow operations

3 Pixel-Parallel Processor Arrays

Conventional image-processing systems (which integrate a camera and a digital processor) have many issues for application in general- purpose consumer electronic products: cost, power consumption, size and complexity.

One of the main disadvantages is the data transmission bottlenecks between the camera, the processor and the memory. In addition, low-level image-processing operations have a high and inherent parallelism which only can be exploited if the access to data is not heavily restricted. Computer Vision is one of the most intensive data processing fields, and conventional systems do not provide any mechanism to address adequately this task, so this issue comes up as an important drawback.

Pixel-parallel processor arrays aim to be the natural platform for low-level image-processing and pixel-parallel algorithms. They are MP-SIMD processors laid down in a 2D grid with a processor-per-pixel correspondence and local connections among neighbors. Each processor includes an image sensor, so the I/O bottleneck between the sensor and the processor is eliminated, and the performance and power consumption are highly improved. This and their massively parallelism are the main benefits of these devices.

Pixel-parallel processor arrays operate in SIMD mode, where all the processors execute simultaneously the same instruction on their local set of data. To exchange information, they use a local interconnection, normally present only between the nearest processors to save silicon area. Concerning each processor, although with local memories, data I/O and sensing control to be self-contained, they are as simple as possible in order to reduce area requirements, but still powerful enough to be general purpose. The idea behind these devices is that the entire computing is done on-chip, so that input data are logged in through the sensors and the output data are a reduced and symbolic representation of the information, with low-bandwidth requirements.

One of the drawbacks of this approach is the reduced integration density. The size of the processors must be as small as possible because, for a 256 × 256px image, more than 65 k processors plus interconnections must be included in a reduced area. This is the reason why many approaches utilize analog or mixed-signal implementations, where the area can be heavily optimized. Nevertheless, accuracy is its main drawback because it is hard to achieve large data-word sizes. In addition, a careful design must be done, implying larger design periods and higher economic costs. The scalability with the technology is not straightforward because of the human intervention in all the process, which does not allow automation. The size of the arrays is also limited by capability to distribute the signals across the array. The effective size of the arrays forces us to use lowresolution images. Examples of mixed-mode focal-plane processors are the Eye-Ris vision system [21] or the programmable artificial retina [22].

Other approaches use digital implementations with the aim to solve the lack of functionality, programmability, precision and noise robustness. The ASPA processor [23] and the design proposed by Komuro et al. [24] are the examples of this kind of implementations.

As each processor includes a sensor, it should occupy a large proportion of the area to receive as much light as possible. However, this will reduce the integration density. New improvements in the semiconductor industry enables three-dimensional integration technology [25]. This introduces a new way to build visions system adding new degrees of freedom to the design process. For instance, [26] proposes a 3D analog processor with a structure similar to the eye retina (sensor, bipolar cells and ganglion cells layers, with vertical connections between them) and [27] presents a mixed-signal focal-plane processor array with digital processors, also segmented in layers.

As a representative device of this category, the SCAMP-3 Vision Chip was selected to map the retinal vessel tree extraction algorithm described in Section 2.

3.1 The SCAMP-3 processor

The SCAMP-3 Vision Chip prototype [28] is a 128 × 128 px cellular processor array. It includes a processor-per-pixel in a mixed-mode architecture. Each processor, an Analog Processing Element (APE), operates in the same manner as a common digital processor but working with analog data. It also includes a photo-sensor and the capability to communicate with others APEs across a fixed network. This network enables data sharing between the nearest neighbors of each APE: NEWS. All processors work simultaneously in SIMD manner.

Figure 4 shows its basics elements. Each APE includes a photo-sensor (Photo), an 8 analog register bank, an arithmetic and logic unit (ALU) and a network register (NEWS). A global bus connects all the modules. All APEs are connected through a NEWS network, but the array also includes row and column address decoders to access to the processors and extract the results. The data output is stored in a dedicated register (not shown).

Operations are done using switched-current memories, allowing arithmetic operation and enabling general-purpose computing. As current mode is used, many arithmetic operations can be done without extra hardware [29]. For example, to add two values, a simple node between two wires is needed (Kirchhoff's law).

The SCAMP-3 was manufactured using 0.5-mum CMOS technology. It works at 1.25 MHz consuming 240 mW with a maximum computational power of 20 GOPS. Higher performance can be achieved by increasing the frequency, at the expense of a higher power consumption. Using this technology, a density of 410 APEs/mm^2 is reached (less than $50\mu m \times 50\mu m$ per APE).

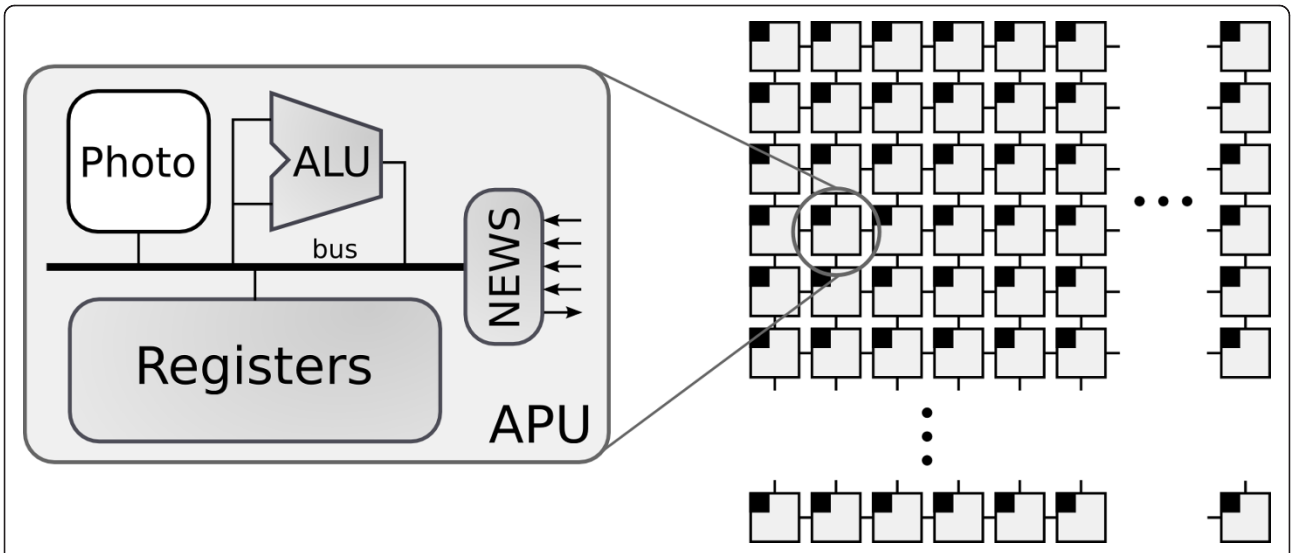

Figure 4 Overview of the SCAMP-3 main elements.

3.2 Implementation

The implementation of the retinal vessel tree extraction algorithm is straightforward. The selected algorithm, as well as other operations present in the low- and mid-level image-processing stages, matches well with this kind of architectures. The SCAMP features a specific programming language and a simulator to test the programs which speeds up the process. For instance, the simulator allows to select the accuracy level of the operations, allowing focusing first on the program functionality and then on the precision of the implementation. This is a necessary step to solve the problem caused by not so accurate memories. Specific details, specially those referred to current-mode arithmetic can be found in Dudek [29].

However, some modifications have to be added to the algorithm because of the particularities of the device. Some operations were added to increase the accuracy of the algorithm. The volatility and the errors due to mismatch effects during the manufacture of the memories must be taken into account and the SCAMP has methods to improve the results. The distance estimation during the external potential estimation and accumulated adding are operations that need carefully revision due to the looseness of the switched-current memories.

Other point to take into account is the data input. While for many applications the optical input is the best option, for other applications, where the images are high resolution or the photo-detectors are not adequate to sense the images, a mechanism to upload the image is needed. However, one of the greatest benefits of these devices is lost, the elimination of the bottleneck between the sensing and processing steps. For instance, to integrate the APEs with the sensors of the camera used for the retinal image capture (a Canon CR5 non-mydriatric 3CCD [20]) will provide better results.

It has to be noted that in the SCAMP-3, the size of the array is much lower than the size of the utilized images. The input images can be resized, but the result will be seriously affected. This forces us to split the image into several sub-images and process it independently. As it was mentioned in Section 2, this algorithm allows to consider the sub-images as independent without affecting the quality of the results. However, it affects to the performance and this is not generalizable and highlights the problems of these devices when their size is not easily scalable. More details of the implementation can be found in Alonso-Montes et al. [30].

4 Field-programmable Gate Arrays

An FPGA consists of a set of logical blocks connected through a dense network. These blocks can be programmed to configure its functionality. Combinational functions can be emulated and connected together to build more complex modules.

The great flexibility of the network, which includes a deep hierarchy where each level is optimized for certain tasks, made them very appropriate in all industrial fields and research areas. Nevertheless, it is also one of its drawbacks because much of the chip area is consumed in connections that are not always necessary, increasing cost and power consumption and reducing the working frequency. Certainly, GPUs are a tough competitor as they allow efficient designs in a short time. However, its scope is much more limited since they cannot be used in embedded or portable systems due to their high power consumption and their little suitability for standalone operation. In addition, FPGA vendors are working

hard to improve the high-level programming languages (like SystemC) and integrating modules for specific solutions (both softcore and hardcore) to facilitate the use and debugging of the FPGAs and to reduce design time.

Furthermore, the capabilities of its internal blocks are growing. Apart from dedicated memory blocks, multipliers or DSP units or embedded processors, they also include elements as PCI-Express endpoints, DDR3 SRAM interfaces or even high-performance multi-core processors [31]. The aim is not only to increase performance, but also to speed up the design process. FPGAs are the devices where traditionally ASICs are tested prior to manufacturing. They were selected because of its rapid prototyping and reconfiguration ability. Nevertheless, this is changing. New devices offer faster solutions and small area requirements, reducing the time-to-market with a low cost (compared with a custom design) because of the range of IP cores available in the market is very extensive. Code portability, scaling to higher-capacity FPGAs or migration to new families make them a device to be considered as a final solution and not only as a test platform.

One of the challenges on FPGA design is to develop a custom hardware to fit the application or algorithm. Therefore, apart from having a wide knowledge of the algorithm, some skills on both hardware and software design are required. In addition, the performance will depend on the particular implementation. FPGA industry is making a big effort to ease the design. C-like languages as SystemC or Impulse C or even high-level graphical programming as the enabled by LabVIEW [32] allow a programming closer to the way it is done in a traditional computer. Algorithms are easier to port than using HDL languages because they are intended to model the systems from the behavior point of view instead from a pure- hardware approach.

FPGAs are widely used as computer vision systems. The dense network enables low-, mid- and high-level image processing, adapting to the needs of each level (spatial and temporal parallelism with custom datapaths) [11]. There are many proposals in the literature, as SIMD accelerators, stream processing cores, MIMD units, directly implemented algorithms or other kind of processors that, using the dedicated resources, can lead to an adequate performance in many applications [33]. In addition, accuracy can be tuned to the real needs of the application, saving resources.

4.1 Custom architecture: Coarse-grain Processor Array

Taking into account that the higher performance of an active contour algorithm is achieved when most of the processing is done on-chip and that a parallelism degree as high as in the pixel-parallel processor array cannot be achieved due to the shortage of hardware resources, an alternative architecture was proposed Nieto et al. [34].

The purpose of this architecture is to exploit the large amount of on-chip memory to perform as much computation as possible without using external memories. The parallelism degree has to be reduced because a processor-per-pixel approach is unrealizable. The proposed architecture is the SIMD processor depicted in Figure 5.

The processing array is composed by a set of processing elements (PE) arranged in a matrix form. Local connections between them are included, forming a classical NEWS network (vertical and horizontal connections to the closest PE). A complex network that includes also diagonal connections was considered, but the increase of hardware resources (about a 2 × factor) made us to discard it.

As all PEs work in SIMD manner, a unique control unit is needed. This task is carried out by a simple micro-controller (uController). It includes a memory for program storage and controls the execution flow of the program. Some operations of the algorithm must be applied several times (as the PLS steps) and the micro-controller will help to tune up the algorithm easily. It also has to discern between control operations (loops and branches) and compute-intensive operations (to be driven to the processing array).

As Figure 5 shows, the two main modules of each PE are a Register File and an ALU. The Register File is made up of an embedded Dual-Port Block RAM and stores a sub-window of the image. To store partial results during algorithm execution, the Register File also has to store several and independent copies of the original sub-window. An example will clarify this: if the Block RAM size is 8 Kb, it can store up to 1024 8-bit words or, this is, 4 images of 16 × 16 px. The ALU implements a reduced set of mathematical operations. The instruction set covers both arithmetic (as addition, subtraction, multiplication or multiply-and-accumulate operations) and bitwise operations (common Boolean and bit shifts), featuring general-purpose low-level image processing. To reduce hardware requirements, the embedded multipliers or DSP units available are used. The Register File provides two operands but does not allow store operations at the same time so each operation is performed in two clock cycles (fetch-data and execution-store).

Concerning execution, once the image is loaded into the distributed memory of each PE, the uController starts processing. If the current instruction is a *control* operation, the processing array will halt. If not, it will be decoded and distributed to all the PEs. Each PE will execute this instruction over its own local data. For instance, if the size of the sub-window is 16 × 16 px, 256 iterations are needed to complete its execution.

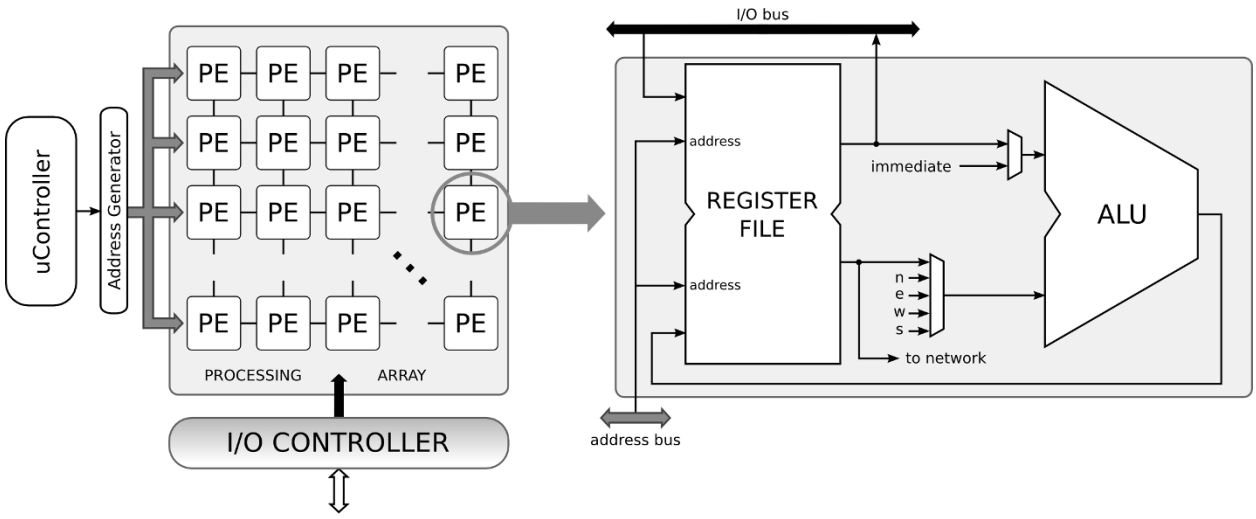

Figure 5 Overview of the Coarse Grain Processor Array architecture.

Then, the next instruction is processed. The following pseudo-code shows how the programming is done:

```
for (dir = 0; dir<4; dir++) {
  Im4 = not Im0
  if(dir = 0)//North
    Im5 = Im4 and shift(Im0, south, 1)
    Im6 = Im5 or shift(Im5, east, 1)
    Im4 = Im6 or shift(Im5, west, 1)
  else if(dir = 1)//East
    Im5 = Im4 and shift(Im0, west, 1)
    Im6 = Im5 or shift(Im5, north, 1)
    Im4 = Im6 or shift(Im5, south, 1)
  else if(dir = 2)//West
    Im5 = Im4 and shift(Im0, east, 1)
    Im6 = Im5 or shift(Im5, north, 1)
    Im4 = Im6 or shift(Im5, south, 1)
  else//South
    Im5 = Im4 and shift(Im0, north, 1)
    Im6 = Im5 or shift(Im5, east, 1)
    Im4 = Im6 or shift(Im5, west, 1)
}
```

Flow operations (*for* and *if*) are executed in the uController while the rest of operations are supplied to the Processor Array and executed over the whole sub-window before applying the next instruction. Each available sub-window is represented as Im[x]. Second operator supports variable shifts across the sub-window before operating in order to access to the neighborhood. To handle this characteristic, the Address Generator unit is included. It enables automatic network access if the data are in a neighbor PE, so human interaction is not needed and scaling through larger arrays is automatic, making the source code totally portable. More information about this feature can be found in Nieto et al. [34].

The I/O interface enables the communication between the computer host and the board. This module is directly connected to the processing array, and it will halt the uController when execution ends to do the data transfer.

4.2 Implementation

As the results are device-dependent, we opted not to select neither the highest performance nor the lowest cost FPGAs. We selected a representative device within the range of solutions for the consumer. The Xilinx Spartan-3 family was designed focusing on cost-sensitive and high volume consumer electronic applications. The device chosen for the algorithm implementation is an XEM3050 card from Opal Kelly with a Xilinx Spartan-3 FPGA, model SC3S4000-5 [35]. The most remarkable features of this FGPA are 6912 CLBs or 62208 equivalent logic cells (1 Logic Cell = 4-input LUT and a D flip-flop), 96 × 18 Kb embedded RAM blocks and 520 Kb of Distributed RAM, 96 dedicated 18-bit multipliers and a Speed Grade of -5. This FPGA uses 90 nm process technology. The board also includes a high-speed USB 2.0 interface and 2 × 32 MB SDRAM and 9 Mb of SSRAM. VHDL description and Xilinx ISE 10.1 tools were employed.

With this device, the following parameters for the coarse-grain processor array were chosen. Data width was set to 8-bits because the original implementation of the algorithm demonstrated that with this word-size sufficient precision is reached. Each sub-window is 16 × 16 px and up to 8 independent sub-windows per Block RAM can be used (each one features 18 Kb, where 2 Kb are parity bits -not used). The ALU contains an embedded multiplier. An USB 2.0 I/O controller is also

implemented. With these parameters, an array size of 9 × 10 PEs fills completely the FPGA. It is able to process images of 144 × 160 px at a clock frequency of 53 MHz.

The algorithm was implemented completely trustworthy to the original algorithm proposal. Instruction set and processors topology match algorithm operations. As it happens with the SCAMP implementation, it is still needed to split the input images into several subwindows. However, the architecture supports simple scaling to larger or newer FPGAs.

5 Massively Parallel Processor Array

MPPA provides hundreds or even thousands of processors. Each unit is encapsulated and works independently, so they have their own program and data memories. All units are connected to a programmable interconnection, enabling data exchange between them. These are usually point-to-point channels controlled by a message passing system which allows its synchronization. MPAAs also include distributed memories which are connected to the network using the same channels. This independent memories will help during the development process to store data when the local memory of each processor is not enough or to build FIFO queues, for instance.

The main differences between MPPA and multicore or manycore architectures are the number of processing units (which traditionally was much higher, though latest GPUs have increasingly computational units, making them comparable), their originally conceived general-purpose character and that they do not include a shared memory (as is the case of symmetric multiprocessing) [36].

They are focused on exploiting the functional and temporal parallelism of the algorithms instead of the spatial parallelism (as happened with the Pixel-parallel processor array). The idea is to split the original algorithm into several subtasks and match each with one or several processors of the array. Each processor executes sequential code, a module of the algorithm or the application. The different modules are connected together using the channels in a similar way of a flow diagram of an algorithm. This way they attempt to solve the bottleneck between processors and the external memory and avoid the need to load all data at a time while the functional units are stopped. Stream computing has been proved to be very efficient [14]. The parallelism is obtained running different modules in parallel. However, processors can include internal SIMD units making them even more powerful and flexible.

As they are encapsulated, higher working frequencies can be achieved, making them competitive despite the lower parallelism level if we compare them with a cellular processor, for example. The use of multiple computational units without explicitly managing allocation, synchronization or communication among those units are also one of its major advantages. This is one of the goals of MPPAs, to ease the development process. As all these processes can be done (commonly) through a friendly high-level programming language, some authors say that MP-PAs are the next evolution of FPGAs [37]. Designers are increasingly demanding high-performance units to address parts of the application which are difficult to map on a pure-hardware implementation. This is one of the reasons why future FPGAs will include high-end embedded microprocessors [31]. MP-PAs already provide this capability including a standard interface with the rest of modules of the system. Dedicated hardware as this will cut down power consumption and hardware resources while performance will be heavily increased. However, FPGAs are still faster for intensive computing applications [38].

As remarkable examples of MPPAs devices, we should cite the picoChip [39], the Tilera Processor [40] or the PARO-design system [41].

5.1 The Ambric Am2045 processor

The selected platform where to migrate the retinal vessel tree extraction algorithm is the parallel processor Am2045 from Ambric [37]. It is made up of a large set of fully encapsulated 360 32-bit RISC processors and 360 distributed memories. Each RISC processor runs its own code, a subtask of the complete algorithm. A set of internal interconnections allows data exchange between them. Synchronization between processors is done automatically through a flexible channel hierarchy. A simple handshake and local protocol between registers enables synchronization without intermediate logic, as it happens when using FIFOs. A chain of those registers is known as a *channel*. Figure 6 shows the main blocks of this architecture.

Ambric uses two kinds of processors, SR and SRD. Both are 32-bit RISC processors specially designed for streaming operations. SRD CPU enables instruction and data-level parallelism. Sub-word logical and integer operations as well as fixed point operations are also possible. SRD processors also include 3 ALUs, two of which work in parallel, and a 256-word local memory. SR CPUs are a simpler version of SRDs, specially designed to deal with simple tasks where DSP extensions are not needed. They only have an ALU and a 64-word local memory. A group of 2 SRD and 2 SR CPUs is known as a Compute Unit (CU) and includes a local interconnection to let direct access between them.

Distributed memory is organized in modules known as RAM Units (RU). Each RU has 4 banks of 256 words connected together through a dynamic configured interconnection. There are also direct links between the RU and the SRD CPUs (not shown in Figure 6) which

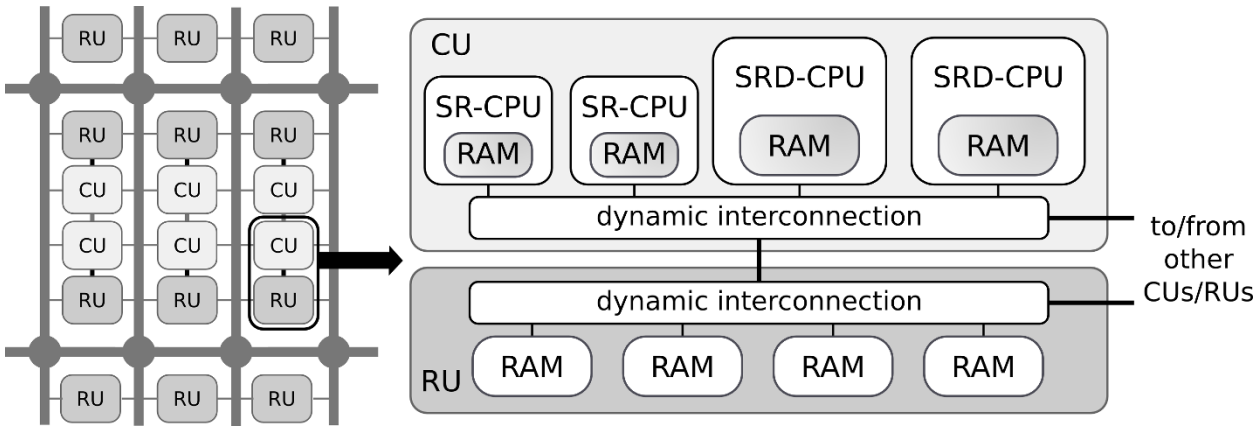

Figure 6 Overview of the Ambric architecture: Computational Units (CU) and RAM Units (RU).

provide random access or FIFO queues both for instructions and for data to increase performance and flexibility.

A group of 2 CU and 2 RU is called Brick. Bricks are connected using a distant reconfigurable channel network, which works at a fixed frequency.

The Am2045 chip uses 130-nm standard-cell technology and provides 360 32-bit processing elements and a total of 4.6 Mb of distributed RAM, working at a maximum frequency of 333 MHz and featuring a power consumption about 10 W. It also includes two DDR2-400 SDRAM interfaces and a 4-lane PCI-Express, to enable fast internal and external I/O transactions. This device does not feature shared memory, but it has an external memory that can only access certain elements that control the I/O to the chain of processors which map the algorithm. A more detailed description of the hardware can be found in Butts et al. [37].

5.2 Implementation

The computational paradigm in Ambric's device is completely opposed to the two former implementations addressed in this paper. While using cellular or coarse-grain processors, we were focusing on the characteristics of the algorithm that allowed to exploit its massive spatial parallelism, now this is not suitable. Although with stream processors it is possible to implement applications in a pure SIMD fashion, it is more appropriate to modify the algorithm and make certain concessions in order not to compromise performance. For instance, the iterative nature or operations as the hole-filling are very expensive in terms of both time and hardware resources (number of processors). This is one of the drawbacks of the computational paradigm used by this platform. Recursive operations are quite resource consuming because each iteration requires to replicate the set of processors which implements the operation. This is not a strict rule, and there are other approaches that can address this problem differently through a reorganization of operations and modules. However, most low-level and much of the mid-level operations have data dependencies that oblige to complete the previous operation over the whole or most part of the data set before applying the next operation. PLS is a clear example.

Figure 7 summarizes the algorithm mapped on the Am2045 device. 16-bit instead of 8-bit words are used to guarantee accuracy. The SIMD capabilities of the SRD processors are also used. This is specially advantageous for binary images because 32 pixels can be processed at a time, increasing greatly the performance during the PLS evolution. Many operations can be seen as 2D convolutions, split into vertical and horizontal filters. While for horizontal filtering the implementation is straightforward, for vertical filtering the local memories must be used to store the previous rows. FIFO queues balance the paths and avoid internal blockages during the processor communication. Although synchronization is automatic, it may occur that some paths were much faster than others. For example, consider a processor which supplies data to two processing paths, one much more slower than the other and a second processor where both paths are combined. When a processor tries to read from an empty channel, a stall occurs. FIFO queues avoid those stalls, but a bottleneck analysis is necessary.

Some operations of the algorithm were not implemented. The hole- filling step, located between slow and fast PLS evolutions and introduced to improve results (see Figure 2), was removed because of its heavy resource consumption. In the same way, some internal improvements in the PLS were eliminated, as the internal potential. This greatly simplifies the implementation of the PLS, leading to binary operations only. Each PLS step

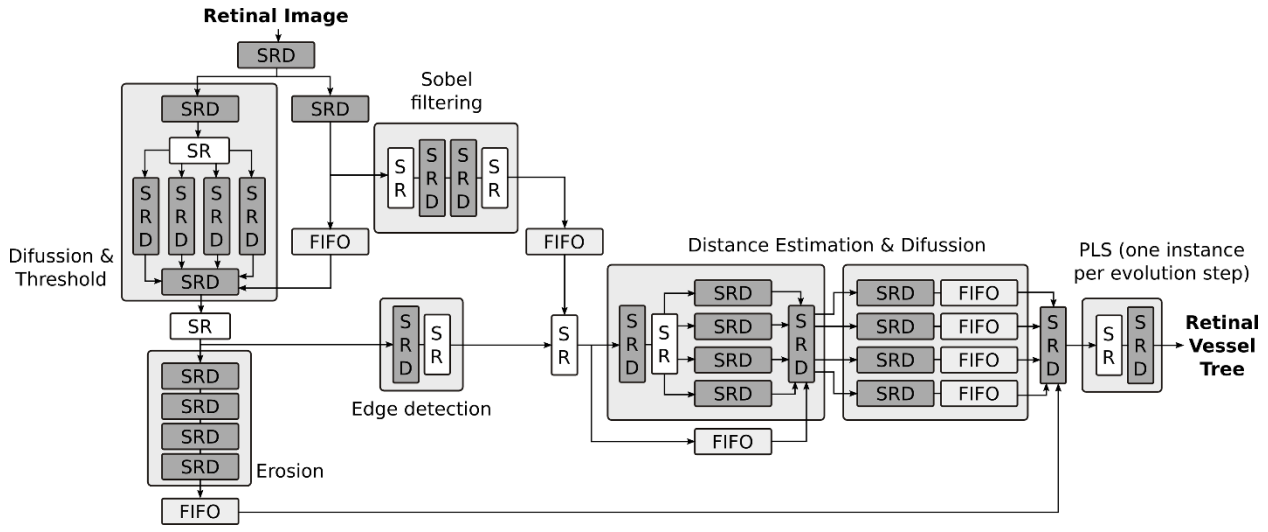

Figure 7 Retinal vessel tree extraction algorithm mapped in Ambric Am2045 device.

requires just one processor, and there is no need to store partial results, leading to a large increase in performance. Otherwise, PLS step will require more processors to be implemented. Although normally this is not a problem, when performing a large number of iterations, it will be something to consider. This results in an accuracy reduction in the results that, depending on the application, may make the results invalid. This point is discussed in depth in Section 6.

On the other hand, to tune up this kind of operations is tedious because the chain structure (see Figure 7) must be modified. Recursive operations over the same set of data make the implementation in stream processors much more complex than in cellular processors, and these operations are common during the first stages of image processing.

The following pseudo-code shows how SR CPUs work. It implements a 3 × 3 sharpening filter using only one processor. It will have a poor performance, but it illustrates how the programming is done and how these platforms speed up the development.

```
public void run(InputStream<Integer> in,
          OutputSteam<Integer> out) {
   //Sharpen filter implementation
   //(one processor used)
    px_1 = px_2; px_2 = px_3; px_3 = in.
readInt();
   px_4 = px_5; px_5=px_6; px_6=in.readInt
();
   px_7=px_8; px_8=px_9; px_9=in.readInt
();
   tmp = (-px_1-px_2-px_3);
   tmp = tmp + (-px_4+8*px_5-px_6);
   tmp = tmp + (-px_7-px_8-px_9);
   if (tmp < 0) tmp = 0;
   out.writeInt(tmp);
}
```

More details of the implementation can be found in Resco et al. [42].

6 Results and comparison

In this section, the results of the algorithm implementation are discussed. Table 3 summarizes the most relevant results of the different implementations. We can see that even with a next generation processor, the results are not satisfactory. Systems running this kind of algorithms usually are required for a rapid response, something that a PC hardly can provide.

As it was detailed in Sec. 2.3, it was developed using OpenCV/OpenMP and C++ on a computer equipped with a 4-core Intel Core i7 940. This way, compared with the initial straightforward MATLAB implementation (for test purposes) time execution drops from 41 to 13.7 s. As explained in the Introduction, this algorithm was designed to work on-line following the capture stage, so a PC is not a good choice. However, we will use it as a reference system for comparison.

When executing the retinal vessel tree extraction algorithm, we were not only seeking ways to reduce the processing time but also we wanted to test different platforms and determine their weaknesses. This way, SCAMP-3 implementation adds operations to improve accuracy (necessary when using analog memories), FPGA gets the most accurate results because it is possible to implement the original method trustworthy (at the cost of reducing performance) and Ambric's device provides the fastest results but reducing the accuracy (required hardware resources would be very high otherwise).

Table 3 Most relevant results of the retinal vessel tree extraction algorithm implementation on the different devices

	Intel Core i7 940	SCAMP-3	Spartan-3	Am2045
Window size (px)	-	128 × 128	144 × 160	-
Processors (used/available)	4/4	16,384/16,384	90/90	125/360
Working frequency (MHz)	2930	1.25	53	333
Window execution time (ms)	-	6.55	66.1	-
Required windows	1	30	20	1
Computation time (s)	13.6	0.193	1.323	0.008
Total execution time with I/O (s)	13.7	0.230	1.349	0.0087
Speed-up	1×	59.6×	10.2×	1574.7×*
Cycles-per-pixel (without I/O)	357,993	8,950	14,070	742*

* It should be noted that the Ambric Am2045 runs a simplified version of the algorithm. See Section 6 for details

It can be seen from Table 3 that Ambric's device gives the highest performance, lowering the total execution time and achieving a high speed-up. This is in part due to the simplifications we made in the algorithm as it was explained previously in Sec. 5. This implementation enables a performance not achievable by computers or FPGAs (at least using a low-cost device). We must emphasize that the simplification of the algorithm provides accurate and valid results, but they may not be suitable for certain applications. For instance, they are still valid to obtain the skeleton of the retinal tree but not to measure the vascular caliber. When migrating the algorithm to this platform, we have faced a trade-off between speed and validity of the results for any application and in this case and for comparison purposes, we give priority to the processing speed.

The other main reason for the high performance of the Ambric's device is the high working frequency of the Am2045 device (333 MHz), which is considerably faster than those achieved by the SCAMP or the FPGA. Digital solutions provide higher clock frequencies but they are commonly limited by the interconnection between the processing units, as is the case of the FPGA. MPPAs architectures implement point to point connections with minor reconfigurable options than FPGAs so clock frequency does not depend on the algorithm which is being running. However, recent FPGAs families increase its computing capacity considerably. We have estimated that migrating the proposed design to a Virtex-6 model XC6VLX240T-1 [43], a considerable speed-up can be obtained, reaching 80 ms per image without compromissing accuracy in the original algorithm.

The cycles-per-pixel (CPP) metric measures the number of clock cycles required to execute all operations that result on each one of the pixels in the resulting image (see Table 2), normalizing the differences in frequency and number of processing cores. The above discussion is summarized using this value. It should be noted that the Ambric Am2045 runs a simplified version of the algorithm. When the number of operations is corrected, this leads us to an obvious reduction in performance, approximately multiplying by 3 the CPP value. Although the theoretical performance would remain higher than the other approaches, the problem we face is different: there is not enough processors and interconnections to fit the algorithm in the device. This was the main reason why it was decided to simplify it.

Analog computing increases greatly the density of integration. In the SCAMP-3 with a relatively old 0:5 *mu*m CMOS technology, each processing element occupies an area lower than 50μ m × 50μ m and it remains flexible enough to implement any kind of computation. However, accuracy must be considered due to the nature of the data representation (current mode) and the technology issues (mismatch, memory volatility, noise effects...). Analog computing allows to integrate a processor per pixel of the sensor, eliminating one of the most important bottlenecks in traditional architectures. Given the number of bits of the data representation, digital platforms guarantee accuracy independently of the technology process. While 7-8 bits are hard to reach using mixed-signal architectures [44], 32-bit or 64-bit architectures are common in digital devices.

The discussed algorithm is robust enough to run in platforms with short data representations as SCAMP-3. However, this is not only the unique factor which affects the final result. As it was discussed above, each device requires us to make certain concessions to guarantee its viability. Table 4 summarizes the maximum average accuracy (MAA) [45] of each implementation compared with the manual segmentation available in the DRIVE database [20]. We can see that using the Ambric device, the accuracy drops around a 10% when removing the mentioned operations. The main reason is the appearance of many false positives that now are not eliminated using the hole-filling operation. However, once skeletonized the vascular tree, they can be easily removed using hit-and-miss masks if the application requires it.

Concerning algorithm issues, a comparison with other approaches can be found in Alonso-Montes et al. [9].

Although digital designs consume more silicon area, the improvements in the semiconductor industry are lowering the silicon area requirements, providing higher rates of integration density. The benefits are an easily scaling and migration of the architectures, which it is possible to carry out with straightforward designs as a difference with analog computing. FPGAs and MPPAs are clear examples. Analog Cellular Processors have a large network that connects the processing elements. To upscale this network, keeping a high yield with different array sizes and manufacturing processes is extremely difficult. This is one of the main reasons why they are not able to process high-resolution images and why customers have more availability of digital devices. This way, MPPA devices are the most suitable platform to deal with big images because they have not restrictions in this sense (stream processing modules can also be implemented on FPGAs). Pure SIMD-matrix approaches offer good performance because they match the most common early vision operations but the image size is limited. In those cases, a sliding window system is a need to process bigger images.

Image size constrains the amount of RAM needed in the system, especially when working with high-resolution images. One advantage of this algorithm is that it needs a small amount off-chip memory and that it can take advantage of the embedded RAM to perform all computation, reducing IO. External RAM is mainly used to store the input image and the results so 4-8 MB are enough. The SCAMP-3 Vision Chip can store on-chip up to 8 128 × 128 px images (equivalent to 128 Kb, taking into account that the computation in done in analog mode), the architecture proposed for the Spartan-3 up to 8 144 × 160 px images (176 Kb) and the Ambric Am2045 can store up to 4.6 Mb of data. This is the main reason why PC performance is so low: the selected devices are capable of doing all the processing on-chip, accessing to the external memory just for loading the input image. On the contrary, the PC should make an intensive use of external memory to store partial results, making memory access a bottleneck. Explicit load/store operations are needed and this is why CPP is much higher than the other approaches.

Analog processing allows to integrate the processors with the image sensors. But this kind of processor distribution, although adequate for low and some steps of mid-level image processing, is not suitable for complex algorithms with more complex data dependencies. SCAMP-3 is very powerful for early vision tasks, but it lacks the flexibility to address higher-level operations. Using FPGAs, different architectures and computing paradigms can be easily emulated. Its dense network, although it consumes a large silicon area, provides this flexibility. MPPAs attempt to, reducing interconnection between processors, build more powerful computing elements. The fixed network of Cellular Processors limits its range of application. The programmable and dense network of FPGAs enables any kind of architecture emulation but reducing integration density and the working frequency. MPPAs are located in an intermediate position. Its programmable network provides enough flexibility to cover a wide range of applications, freeing up space to build more powerful processors, but limiting the kind of computations that can address. This is why recursive operations are hard to implement and the resource usage is so high. PLS had to be simplified to deal with this trade-off.

Table 4 Maximum Average Accuracy (MAA) for each implementation, including the manual segmentation by an expert

	Manual	Intel Core i7 940	SCAMP-3	Spartan-3	Am2045
MAA	0.9473	0.9202	0.9180	0.9192	0.8132

With respect to the algorithm development process, Time-To-Market (TTM) is key in industry. Ambric's platform offers the system with the lowest TTM. A complete SDK that provides a high-level language and tools for rapid profiling make the development much faster than in other platforms. This is one of the purposes of the platform, to offer a high-performance device keeping prototyping rapid and closer to software. Using HDL language to develop complex architectures or software-to-hardware implementations is much more expensive because they are closer to hardware than to software. This is specially true in the second case, where a high-level language (as SystemC or ImpulseC) is recommended.

Regarding portability, it is clear that a computer-based solution (even a mobile version) is not a valid solution because of size, power consumption or lack of integration and compactness between its components. For early vision tasks, a focal-plane processor (as SCAMP-3) is the best choice. The processors are integrated together with the sensors and their power consumption is very reduced. To accomplish complex operations, FPGAs offer reduced size and power consumption in its low-end products, allowing to build complex Systems-on-Chip and compacting all the processing on the same chip. For better performance, an MPPA or a larger FPGA is needed. Power consumption will be higher, specially for the high-end FPGAs, but the amount of processing units we can include is considerably higher.

Although the conclusions we have drawn in this section come from a specific algorithm, this has features common to most algorithms for low- and medium-level

image-processing tasks (see Sec. 2.3). We have seen that visual processors as SCAMP are excellent in early vision operations, but specific algorithms are needed to address the subsequent processing steps. MPPAs provide an environment closer to the programmer, with a great performance, limited by the low-level operations. FPGAs enable us to replicate any application with very acceptable results, although the development time is higher. Its internal network is both their greatest advantage and disadvantage.

7 Conclusions

In this paper, a comparison of different implementations of a retinal vessel tree extraction algorithm was made. The algorithm can operate online in a device with reduced size, cost and power consumption, opening new possibilities in other areas apart from the medical field. It was specifically designed focusing on performance, so an MP-SIMD processor array offers good results. However, the technology limitations, especially array size and a limited accuracy, make us consider other approaches.

FPGAs enable us to speed up most applications with acceptable results. They take advantage of its highly reconfigurable network to improve the matching between architecture and algorithm, exploiting its characteristics and potentially leading to better performance solutions. Advances in semiconductor industry enable to integrate more and more functional units in a reduced silicon area.

MPPAs take advantage of this integrating hundred of processors in the same chip. They focus on exploiting the task parallelism of the algorithms, and results prove that this approach provides remarkable performance. However, certain trade-offs must be done when dealing with low-level image processing not to compromise efficiency.

Results show that even using a high-end CPU, a significative gain can be achieved using hardware accelerators. A low-cost FPGA outperforms the Intel Core i7 940 by a factor of 10×. With a focal plane-processor, this factor reaches 60×. Using the selected MPPA, a factor of more than 1500× was reached, but we have to take into account that the algorithm was simplified in order to sort the limitations of the platform when dealing with low-level image processing. The accuracy drops about 10% which might compromise its suitability for some applications. First estimations using a Virtex-6 FPGA and the same architecture indicated here show a speed-up around 170×, even better than the focal-plane processor, which is the natural platform for this kind of algorithms.

The retinal vessel tree extraction algorithm presents common features to most of the low- and mid-level algorithms available in the literature. Except for high-level operations over complex data sets, where high precision is needed, the presented architectures perform adequately for low- and mid-level stages, where operations are simple and have to be applied over a large set of data. They are able to exploit the massive spatial parallelism of low-level vision, featuring general-purpose computation. Ambric's processor requires a special mention because, although it can exploit spatial parallelism of low-level vision, its throughput is very high when dealing with the mid-level stage, where task parallelism is clearly advantageous. However, SCAMP-3 is mainly restricted to the low-level stage where its low power consumption and form factor fit well. FPGAs are flexible enough to cover the complete application. Their major drawback is the time required to get the system ready.

Acknowledgements

This work is funded by Xunta de Galicia under the projects 10PXIB206168PR and 10PXIB206037PR and the program Maria Barbeito. Authors would also like to thank the reviewers for their helpful comments and suggestions.

Author details

[1]University of Santiago de Compostela, Centro de Investigación en Tecnoloxías da Información (CITIUS), Santiago de Compostela, Spain
[2]Department of Electronic and Systems, University of A Coruña, A Coruña, Spain

Competing interests

The authors declare that they have no competing interests.

Received: 1 March 2011 Accepted: 27 September 2011
Published: 27 September 2011

References

1. Bankman I: **Handbook of Medical Imaging: Processing and Analysis.** Academic Press, London; 2000.
2. Patton N, Aslam T, MacGillivray T, Deary I, Dhillon B, Eikelboom R, Yogesan K, Constable I: **Retinal image analysis: concepts, applications and potential.** *Prog Retin Eye Res* 2006, **25(1)**:99-127.
3. Lowell J, Hunter A, Steel D, Basu A, Ryder R, Kennedy R: **Measurement of retinal vessel widths from fundus images based on 2-D modeling.** *IEEE Trans Med Imaging* 2004, **23**:1196-1204.
4. Wilson C, Cocker K, Moseley M, Paterson C, Clay S, Schulenburg W, Mills M, Ells A, Parker K, G Quinn, *et al*: **Computerized analysis of retinal vessel width and tortuosity in premature infants.** *Investig Ophthalmol Vis Sci* 2008, **49(8)**:3577.
5. Ortiz D, Cubides M, Suarez A, Zequera M, Quiroga J, Gomez J, Arroyo N: **System for Measuring the Arterious Venous Rate (AVR) for the Diagnosis of Hypertensive Retinopathy.** *ANDESCON, 2010 IEEE* 2010, 1-4.
6. Salem N, Nandi A: **Unsupervised Segmentation of Retinal Blood Vessels Using a Single Parameter Vesselness Measure.** *Sixth Indian Conference on Computer Vision, Graphics Image Processing, 2008. ICVGIP '08* 2008, 528-534.
7. Lion N-X, Zagorodnov V, Tan Y-P: **Retinal Vessel Detection Using Self-Matched Filtering.** *IEEE International Conference on Image Processing, 2007. ICIP 2007* 2007, **6**:VI-33-VI-36.
8. Li Q, You J, Zhang L, Zhang D, Bhattacharya P: **A New Approach to Automated Retinal Vessel Segmentation Using Multiscale Analysis.** *18th International Conference on Pattern Recognition, 2006. ICPR 2006* 2006, **4**:77-80.
9. Alonso-Montes C, Vilarino D, Dudek P, Penedo M: **Fast retinal vessel tree extraction: A pixel parallel approach.** *Int J Circuit Theory Appl* 2008, **36(5-6)**:641-651.

10. Alonso-Montes C, Ortega M, Penedo M, Vilarino D: **Pixel Parallel Vessel Tree Extraction for a Personal Authentication System.** *IEEE International Symposium on Circuits and Systems. ISCAS 2008* 2008, 1596-1599.
11. MacLean W: **An evaluation of the suitability of FPGAs for embedded vision systems (IEEE).** *Computer Vision and Pattern Recognition-Workshops, 2005. CVP Workshops. IEEE Computer Society Conference on* 2005, 131.
12. Rode H, Chiddarwar A, Darak S: **Suitability of FPGA for computationally intensive image processing algorithms.** *IET Seminar Digests, 2009* 2009, 65-65.
13. Foty D: **Perspectives on Scaling Theory and CMOS Technology–Understanding the Past, Present, and Future.** *Proceedings of the 2004 11th IEEE International Conference on Electronics, Circuits and Systems, 2004. ICECS 2004* 2004, 631-637.
14. Dally W, Kapasi U, Khailany B, Ahn J, Das A: **Stream processors: Programmability and efficiency.** *Queue* 2004, **2(1)**:52-62.
15. Kass M, Witkin A, Terzopoulos D: **Snakes: Active contour models.** *Int J Comput Vis* 1998, **1(4)**:321-331.
16. Cohen L, Cohen I: **Finite-element methods for active contour models and balloons for 2-D and 3-D images.** *IEEE Trans Pattern Anal Mach Intell* 2002, **15(11)**:1131-1147.
17. Vilariño D, Rekeczky C: **Pixel-level snakes on the CNNUM: algorithm design, on-chip implementation and applications.** *Int J Circuit Theory Appl* 2005, **33(1)**:17-51.
18. Dudek P, Vilarino L: **A Cellular Active Contours Algorithm Based on Region Evolution (IEEE).** *10th International Workshop on Cellular Neural Networks and Their Applications, 2006. CNNA 2006* 2006, 1-6.
19. Vilarino D, Dudek P: **Evolution of Pixel Level Snakes Towards an Efficient Hardware implementation.** *IEEE International Symposium on Circuits and Systems, 2007. IS-CAS 2007* 2007, 2678-2681.
20. Staal J, Abramoff M, Niemeijer M, Viergever M, van Ginneken B: **Ridge based vessel segmentation in color images of the retina.** *IEEE Trans Med Imaging* 2004, **23(4)**:501-509.
21. Rodríguez-Vázquez Á, Domínguez-Castro R, Jiménez-Garrido F, Morillas S, Listán J, Alba L, Utrera C, Espejo S, Romay R: **The eye-RIS CMOS vision system.** *Analog Circuit Design* 2008, 15-32.
22. Paillet F, Mercier D, Bernard T: **Second Generation Programmable Artificial Retina.** *Proceedings of the Twelfth Annual IEEE International ASIC/SOC Conference, 1999* 1999, 304-309.
23. Lopich A, Dudek P: **Asynchronous cellular logic network as a co-processor for a general-purpose massively parallel array.** *Int J Circuit Theory Appl* 2010, **39**:963-972.
24. Komuro T, Ishii I, Ishikawa M, Yoshida A: **A digital vision chip specialized for high-speed target tracking.** *IEEE Trans Electron Devices* 2003, **50**:191-199.
25. Topol AW, Tulipe DCL, Shi L, Frank DJ, Bernstein K, Steen SE, Kumar A, Singco GU, Young AM, Guarini KW, Ieong M: **Three-dimensional integrated circuits.** *IBM J Res Dev* 2006, **50**:491-506.
26. Kurino H, Nakagawa M, Lee K, Nakamura T, Yamada Y, Park K, Koyanagi M: **Smart vision chip fabricated using three dimensional integration technology.** *Adv Neural Inf Process Syst* 2001, 720-726.
27. Foldesy P, Zarandy A, Rekeczky C, Roska T: **3D Integrated Scalable Focal-Plane Processor Array.** *18th European Conference on Circuit Theory and Design, 2007. ECCTD 2007* 2007, 954-957.
28. Dudek P: **Implementation of Simd Vision Chip with 128 × 128 Array of Analogue Processing Elements.** *ISCAS 2005 IEEE International Symposium on Circuits and Systems, 2005* 2005, **6**:5806-5809.
29. Dudek P: **A Processing Element for an Analogue SIMD Vision Chip.** *European Conference on Circuit Theory and Design. ECCTD 2003* 2003, **3**:221-224.
30. Alonso-Montes C, Dudek P, Vilarifio D, Penedo M: **On Chip Implementation of a Pixel-Parallel Approach for Retinal Vessel Tree Extraction (IEEE).** *18th European Conference on Circuit Theory and Design, 2007. ECCTD 2007* 2008, 511-514.
31. DeHaven K: **Extensible Processing Platform Ideal Solution for a Wide Range of Embedded Systems.** *Extensible Processing Platform Overview White Paper* 2010.
32. Curreri J, Koehler S, Holland B, George A: **Performance Analysis with High-Level Languages for High-Performance Reconfigurable Computing.** *16th International Symposium on Field-Programmable Custom Computing Machines, 2008. FCCM '08* 2008, 23-30.
33. Saegusa T, Maruyama T, Yamaguchi Y: **How Fast is an FPGA in Image Processing?** *International Conference on Field Programmable Logic and Applications, 2008. FPL 2008* 2008, 77-82.
34. Nieto A, Brea V, Vilarino D: **FPGA-Accelerated Retinal Vessel-Tree Extraction.** *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009* 2009, 485-488.
35. **Spartan-3 FPGA Family Data Sheet.** *Xilinx Product Specification DS099* 2009.
36. Schroder-Preikschat W, Snelting G: **Invasive Computing: An Overview.** *Multiprocessor System-on-Chip: Hardware Design and Tool Integration* 2010, 241.
37. Butts M, Jones A, Wasson P: **A Structural Object Programming Model, Architecture, Chip and Tools for Reconfigurable Computing.** *15th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, 2007. FCCM 2007* 2007, 55-64.
38. Hutchings B, Nelson B, West S, Curtis R: **Comparing Fine-Grained Performance on the Ambric MPPA Against an FPGA.** *International Conference on Field Programmable Logic and Applications, 2009. FPL 2009* 2009, 174-179.
39. Duller A, Panesar G, Towner D: **Parallel processing–the picoChip way!** *Commun Process Archit* 2003, 125-138.
40. Agarwal A: **The Tile Processor: A 64-core Multicore for Embedded Processing.** *Proceedings of HPEC Workshop* 2007.
41. Hannig F, Ruckdeschel H, Dutta H, Teich J: **Paro: Synthesis of hardware accelerators for multi-dimensional dataflow-intensive applications.** *Reconfig Comput Archit Tools Appl* 2008, 287-293.
42. Resco C, Nieto A, Osorio R, Brea V, Vilarino D: **A Digital Cellular-Based System for Retinal Vessel-Tree Extraction.** *European Conference on Circuit Theory and Design, 2009. ECCTD 2009* 2009, 835-838.
43. **Virtex-6 Family Overview.** In *DS099 White Paper* Edited by: bf I Xilinx 2010.
44. Montes CA: **Automatic Pixel-Parallel Extraction of the Retinal Vascular-Tree: Algorithm Design, On-Chip Implementation and Applications.** *PhD Thesis, Faculty of Informatics, University of A Coruna* 2008.
45. Niemeijer M, Staal J, van Ginneken B, Loog M, Abramoff M: **Comparative Study of Retinal Vessel Segmentation Methods on a New Publicly Available Database.** *Proceedings of SPIE* 2004, **5370**:648.

doi:10.1186/1687-5281-2011-10
Cite this article as: Nieto *et al.*: Performance analysis of massively parallel embedded hardware architectures for retinal image processing. *EURASIP Journal on Image and Video Processing* 2011 **2011**:10.