A GPU Accelerated Algorithm for Blood Detection in Wireless Capsule Endoscopy Images

Sunil Kumar, Isabel N. Figueiredo, Carlos Graca and Gabriel Falcao

Abstract Wireless capsule endoscopy (WCE) has emerged as a powerful tool in the diagnosis of small intestine diseases. One of the main limiting factors is that it produces a huge number of images, whose analysis, to be done by a doctor, is an extremely time consuming process. Recently, we proposed (Figueiredo et al. An automatic blood detection algorithm for wireless capsule endoscopy images. In: Computational Vision and Medical Image Processing IV: VIPIMAGE 2013, pp. 237–241. Madeira Island, Funchal, Portugal (2013)) a computer-aided diagnosis system for blood detection in WCE images. While the algorithm in (Figueiredo et al. An automatic blood detection algorithm for wireless capsule endoscopy images. In: Computational Vision and Medical Image Processing IV: VIPIMAGE 2013, pp. 237–241. Madeira Island, Funchal, Portugal (2013)) is very promising in classifying the WCE images, it still does not serve the purpose of doing the analysis within a very less stipulated amount of time; however, the algorithm can indeed profit from a parallelized implementation. In the algorithm we identified two crucial steps, segmentation (for discarding non-informative regions in the image that can interfere with the blood detection) and the construction of an appropriate blood detector function, as being responsible for taking most of the global processing time. In this work, a suitable GPU-based (graphics processing unit) framework is proposed for speeding up the segmentation and blood detection execution times. Experiments show that the accelerated procedure is on average 50 times faster than the original one, and is able of processing 72 frames per second.

S. Kumar (✉) · I. N. Figueiredo
CMUC, Department of Mathematics, Faculty of Science and Technology, University of Coimbra, Coimbra, Portugal
e-mail: isabelf@mat.uc.pt

C. Graca · G. Falcao
Instituto de Telecomunicações, Department of Electrical and Computer Engineering, Faculty of Science and Technology, University of Coimbra, Coimbra, Portugal

J. M. R. S. Tavares, R. Natal Jorge (eds.), *Developments in Medical Image Processing and Computational Vision,* Lecture Notes in Computational Vision and Biomechanics 19,
DOI 10.1007/978-3-319-13407-9_4

1 Introduction

Wireless capsule endoscopy (WCE), also called capsule endoscopy (CE), is a non-invasive endoscopic procedure which allows visualization of the small intestine, without sedation or anesthesia, which is difficult to reach by conventional endoscopies. As the name implies, capsule endoscopy makes use of a swallowable capsule that contains a miniature video camera, a light source, batteries, and a radio transmitter (see Fig. 1). This takes continual images during its passage down the small intestine. The images are transmitted to a recorder that is worn on a belt around the patient's waist. The whole procedure lasts 8 h, after which the data recorder is removed and the images are stored on a computer so that physicians can review them and analyze the potential source of diseases. Capsule endoscopy is useful for detecting small intestine bleeding, polyps, inflammatory bowel disease (Crohn's disease), ulcers, and tumors. It was first invented by Given Imaging in 2000 [12]. Since its approval by the FDA (U.S. Food and Drug Administration) in 2001, it has been widely used in hospitals.

Although capsule endoscopy demonstrates a great advantage over conventional examination procedures, some improvements remain to be done. One major issue with this new technology is that it generates approximately 56,000 images per examination for one patient, whose analysis is very time consuming. Furthermore, some abnormalities may be missed because of their size or distribution, due to visual fatigue. So, it is of great importance to design a real-time computerized method for the inspection of capsule endoscopic images. *Given Imaging Ltd.* has also developed the so called RAPID software for detecting abnormalities in CE images. But its sensitivity and specificity, respectively, were reported to be only 21.5 and 41.8 % [10], see also [19]. Recent years have witnessed some development on automatic inspection of CE images, see [1, 4–6, 7, 9, 14, 15, 18, 20].

The main indication for capsule endoscopy is obscure digestive bleeding [5, 9, 14, 18, 20]. In fact, in most of these cases, the source of the bleeding is located in the small bowel. However, often, these bleeding regions are not imaged by the capsule endoscopy. This is why the blood detection is so important when we are dealing with capsule endoscopy. The current work is an extension of the paper [8], where an automatic blood detection algorithm for CE images was proposed. Utilizing Ohta color channel (R+G+B)/3 (where R, G and B denote the red, green and blue channel, respectively, of the input image), we employed analysis of eigenvalues of the image Hessian matrix and multiscale image analysis approach for designing a function to discriminate between blood and normal frames. The experiments show that the algorithm is very promising in distinguishing between blood and normal frames. But, the algorithm is not able to process huge number of images produced by WCE examination of a patient, within a very less stipulated amount of time. However, the computations of the algorithm can indeed be parallelized, and thus, can process the huge number of images within a very less stipulated amount of time. In the algorithm we identified two crucial steps, segmentation (for discarding non-informative regions in the image that can interfere with the blood detection) and the construction of an

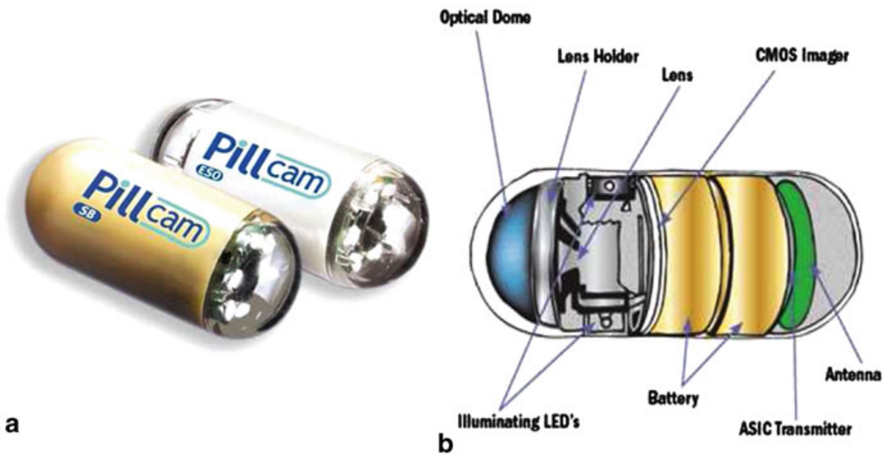


Fig. 1 **a** Image of the capsule. **b** Interior of the capsule

appropriate blood detector function, as being responsible for taking most of the global processing time. We propose a suitable GPU-based framework for speeding up the segmentation and blood detection execution times, and hence the global processing time. Experiments show that the accelerated procedure is on average 50 times faster than the original one, and is able of processing 72 frames per second.

This chapter is structured as follows. A choice of the suitable color channel is made in Sect. 2.1 and segmentation of informative regions is done in Sect. 2.2. A blood detector function is introduced in Sect. 2.3. The outline of the the algorithm is given in Sect. 2.4. Validation of the algorithm on our current data set is provided in Sect. 3. The GPU procedure for speeding up the segmentation and blood detection is described in Sect. 4. Finally, the chapter ends with some conclusions in Sect. 5.

2 Blood Detection Algorithm

Notation Let Ω be an open subset of R^2, representing the image (or pixel) domain. For any scalar, smooth enough, function u defined on Ω, $\|u\|_{L^1(\Omega)}$ and $\|u\|_{L^\infty(\Omega)}$, respectively, denote the L^1 and L^∞ norms of u.

2.1 Color Space Selection

Color of an image carries much more information than the gray levels. In many computer vision applications, the additional information provided by color can aid image analysis. The Ohta color space [17] is a linear transformation of the RGB color space. Its color channels are defined by $A_1 = (R + G + B)/3$, $A_2 = R - B$, and

$A_3 = (2G - R - B)/2$. We observe that channel A_1 has the tendency of localizing quite well the blood regions, as is demonstrated in Fig. 3. The first row corresponds to the original WCE images with blood regions and the second row exhibits their respective A_1 channel images. We also observe that, before computing the A_1 channel of the images, we applied an automatic illumination correction scheme [22] to the original images, to reduce the effect of illumination.

2.2 Segmentation

Many WCE images contain uninformative regions such as bubbles, trash, dark regions and so on, which can interfere with the detection of blood. More information on uninformative regions can be found in [1]. We observe that the second component (which we call henceforth a-channel) of the CIE Lab color space has the tendency of separating these regions from the informative ones. More precisely, for better removal of the uninformative regions, we first decompose the a-channel into geometric and texture parts using the model described in [2, Sect. 2.3], and perform the two phase segmentation. This latter relies on a reformulation of the Chan and Vese variational model [2, 3], over the geometric part of the a-channel.

The segmentation method is described as follows: We first compute the constants c_1 and c_2 (representing the averages of I in a two-region image partition). We then solve the following minimization problem

$$\min_{u,v}\left\{TV_g(u) + \frac{1}{2\theta}\|u - v\|^2_{L^2(\Omega)} + \int_\Omega \big(\lambda\, r(I, c_1, c_2)\, v + \alpha\, \nu(v)\big)\, dx\, dy\right\} \tag{1}$$

where $TV_g(u) := \int_\Omega g(x, y)|\nabla u|\, dx\, dy$ is the total variation norm of the function u, weighted by a positive function g; $r(I, c_1, c_2)(x, y) := (c_1 - I(x, y))^2 - (c_2 - I(x, y))^2$ is the fitting term, $\theta > 0$ is a fixed small parameter, $\lambda > 0$ is a constant parameter weighting the fitting term, and $\alpha\, \nu(v)$ is a term resulting from a reformulation of the model as a convex unconstrained minimization problem (see [2, Theorem 3]). Here, u represents the two-phase segmentation and v is an auxiliary unknown. The segmentation curve, which divides the image into two disjoint parts, is a level set of u, $\{(x, y) \in \Omega : u(x, y) = \mu\}$, where in general $\mu = 0.5$ (but μ can be any number between 0 and 1, without changing the segmentation result, because u is very close to a binary function).

The above minimization problem is solved by minimizing u and v separately, and iterated until convergence. In short we consider the following two steps:

1. v being fixed, we look for u that solves

$$\min_{u}\left\{TV_g(u) + \frac{1}{2\theta}\|u - v\|^2_{L^2(\Omega)}\right\}. \tag{2}$$

2. u being fixed, we look for v that solves

$$\min_{v}\left\{\frac{1}{2\theta}\|u - v\|^2_{L^2(\Omega)} + \int_\Omega \big(\lambda\, r(I, c_1, c_2)\, v + \alpha\, \nu(v)\big)\, dx\, dy\right\}. \tag{3}$$

It is shown that the solution of (2) is ([2, Proposition 3])

$$u = v - \theta \mathrm{div} p,$$

where div represents the divergent operator, and $p = (p_1, p_2)$ solves

$$g\nabla(\theta \mathrm{div} p - v) - |\nabla(\theta \mathrm{div} p - v)|p = 0.$$

The problem for p can be solved using the following fixed point method

$$p^0 = 0,\ p^{n+1} = \frac{p^n + \delta t \nabla(\mathrm{div} p^n - v/\theta)}{1 + \frac{\delta t}{g}|\nabla(\mathrm{div} p^n - v/\theta)|}.$$

Again from [2, Proposition 4], we have

$$v = \min\{\max\{u - \theta\lambda r(I, c_1, c_2), 0\}, 1\}.$$

The segmentation results for some of the WCE images are shown in Fig. 2. The first row corresponds to the original images, the second row shows the segmentation masks, and the third row displays the segmentation curves superimposed on the original images.

In these experiments (and also in the tests performed in Sect. 3) the values chosen for the parameters involved in the definition of (1), are those used in [2], with g the following edge indicator function $g(\nabla u) = \frac{1}{1+\beta\|\nabla u\|^2}$ and $\beta = 10^{-3}$.

2.3 Detector Function

We now introduce the detector function that is designed to discriminate between blood and non-blood frames. We resort to the analysis of eigenvalues of the image Hessian matrix and multiscale image analysis approach. Based on the eigenvalues, both blob-like and tubular-like structures can be detected. For a scalar image $I : \Omega \subseteq \mathbb{R}^2 \to \mathbb{R}$, we define the Hessian matrix of one point (x, y), and at a scale s, by

$$H_s(x, y) = \begin{pmatrix} I^s_{xx} & I^s_{xy} \\ I^s_{xy} & I^s_{yy} \end{pmatrix},$$

where I^s_{xx}, I^s_{xy} and I^s_{yy} are the second-order partial derivatives of I and the scale s is involved in the calculation of these derivatives. The Hessian matrix describes the second order local image intensity variations around the selected point. Suppose $\lambda_{s,1}$ and $\lambda_{s,2}$ are two eigenvalues of the Hessian matrix H_s. Further, suppose that $|\lambda_{s,1}| \le |\lambda_{s,2}|$. Setting $F_s = \lambda^2_{s,1} + \lambda^2_{s,2}$, we define

$$F(x, y) = \max_{s_{min} \le s \le s_{max}} F_s(x, y), \tag{4}$$

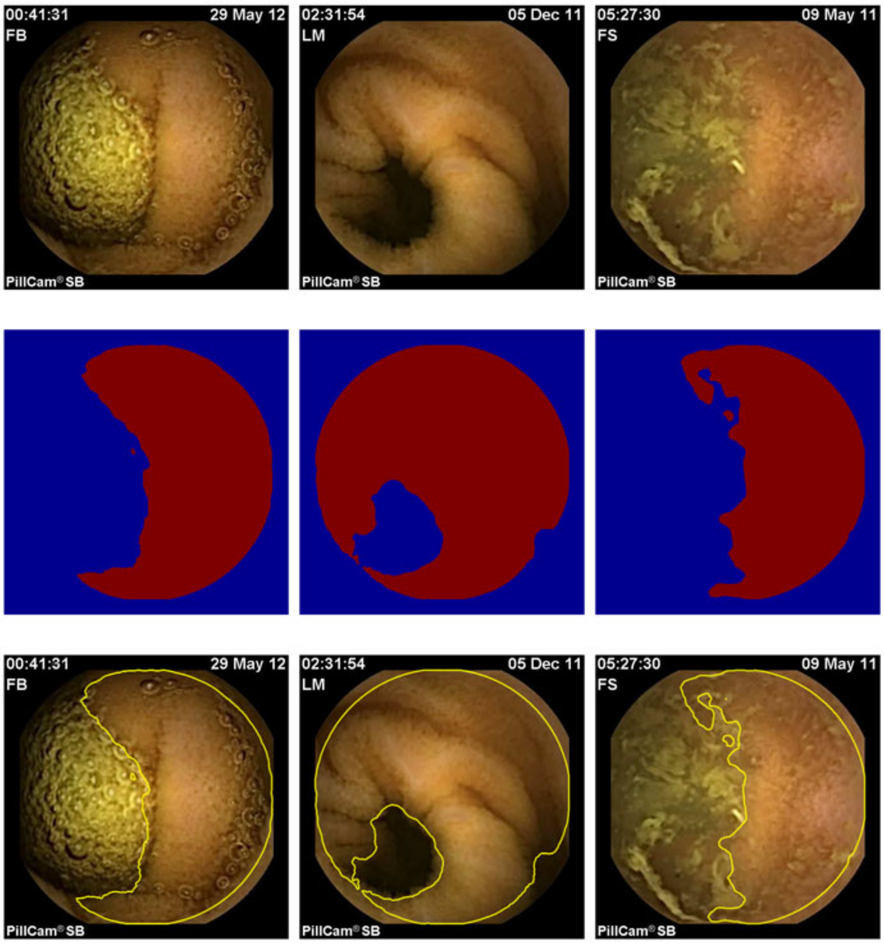


Fig. 2 *First row*: Original image. *Second row*: Segmentation mask. *Third row*: Original image with segmentation curve superimposed

where s_{min} and s_{max} are the minimum and maximum scales at which the blood regions are expected to be found. We remark that they can be chosen so that they cover the whole range of blood regions.

Setting now

$$f_1 = \exp\left(-\beta F_s^2\right) \quad \text{and} \quad f_2 = \left(1 - \exp\left(-\alpha\left(\frac{\lambda_{s,1}}{\lambda_{s,2}}\right)^2\right)\right),$$

and motivated from [11], we define the blob (B_s) and ridge (R_s) detectors (at each point of the domain)

$$B_s = \begin{cases} 0, & \text{if} \quad \lambda_{s,1}\lambda_{s,2} < 0 \quad \text{or} \quad |\lambda_{s,2} - \lambda_{s,1}| > \delta \\ (1 - f_1)f_2, & \text{otherwise}, \end{cases} \tag{5}$$

and

$$R_s = \begin{cases} 0, & \text{if} \quad \lambda_{s,2} > 0, \\ (1 - f_1)(1 - f_2), & \text{otherwise.} \end{cases} \tag{6}$$

Here α and β are the parameters which control the sensitivity of the functions and δ is an user chosen threshold. We then compute the maximum for each scale

$$B(x, y) = \max_{s_{min} \leq s \leq s_{max}} B_s(x, y) \quad \text{and} \quad R(x, y) = \max_{s_{min} \leq s \leq s_{max}} R_s(x, y),$$

In the computations, we take $s = 8, 10, 12, 14$. The results of the functions F and the sum $B + R$, for blood and non-blood images are displayed in Figs. 3 and 4, respectively.

We denote by $\widetilde{\Omega}$, in the image domain, the segmented region of I, that is, $\widetilde{\Omega} = \Omega \cap \Omega_{seg}$, where Ω_{seg} is the segmented sub-domain of I containing the blood. We use the intensity and gradient information of the above functions for designing our detector function, DF, which is defined by

$$DF = \frac{||F||_{L^\infty(\widetilde{\Omega})} ||B + R||_{L^\infty(\widetilde{\Omega})}}{||B + R||_{L^1(\widetilde{\Omega})}}.$$

2.4 Algorithm Outline

For each WCE image the algorithm consists of the following four steps:

1. Firstly, we remove additional details (such as patient name, date and time) from the original image. For this purpose, we clip around the circular view of the original image. Next, we apply an automatic illumination correction scheme [22], for reducing the effect of illumination.
2. We then consider the Ohta color channel $(R + G + B)/3$ for the illumination corrected image.
3. We next apply the two-phase segmentation method [2] for removing uninformative regions (such as bubbles, trash, liquid, and so on) over the geometric part of the second component of the CIE Lab color space.
4. Finally, we compute the functions F, $B + R$ and the blood detector function DF.

3 Validation of the Algorithm

We test the performance of the algorithm on a data set prepared by medical the experts. *Given Imaging*'s Pillcam SB capsule was used to collect the videos in the University Hospital of Coimbra. To make the data set representative, the images

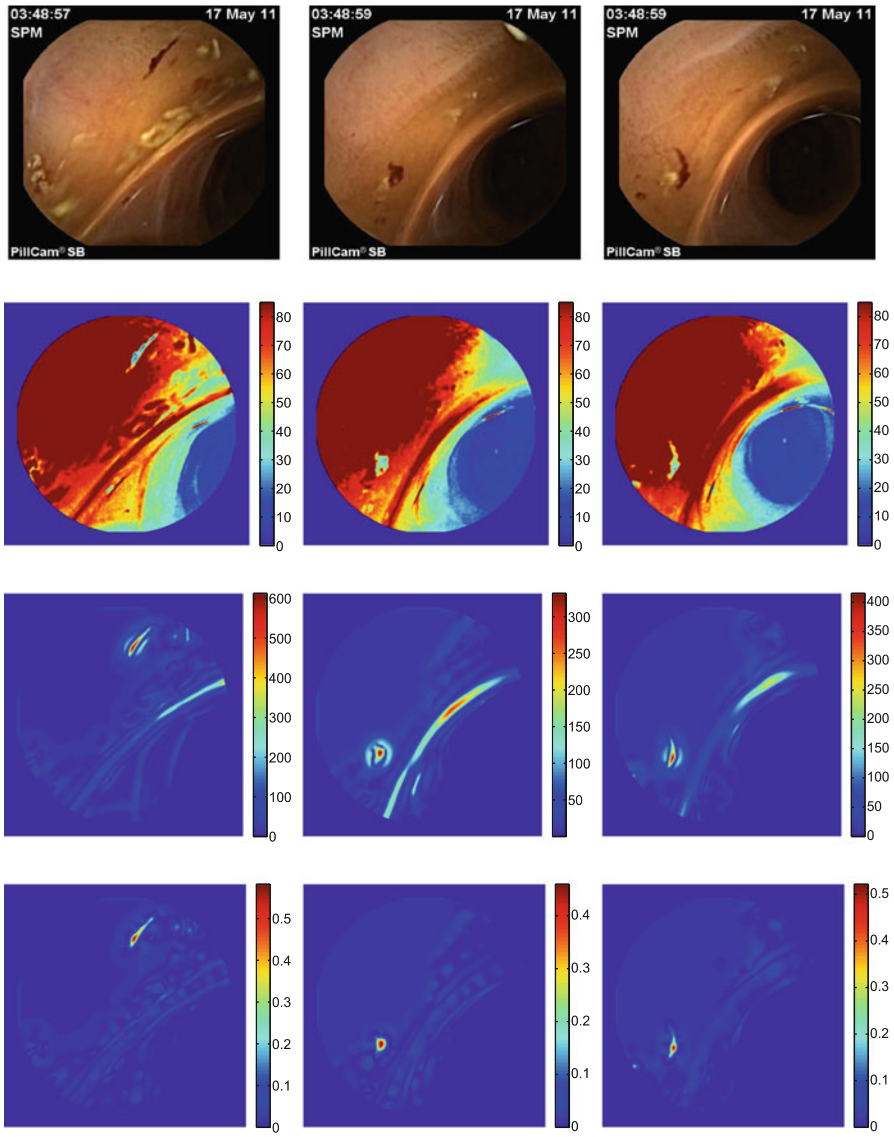


Fig. 3 *First row*: Original image with blood region. *Second row*: A_1 color channel. *Third row*: Function F. *Fourth row*: Function $B + R$

were collected from 4 patients video segments. The data set consists of 27 blood images and 663 normal images. We use standard performance measures: sensitivity, specificity and accuracy. These are defined as follows:

$$\text{Sensitivity} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \quad \text{Specificity} = \frac{\text{TN}}{\text{TN} + \text{FP}},$$

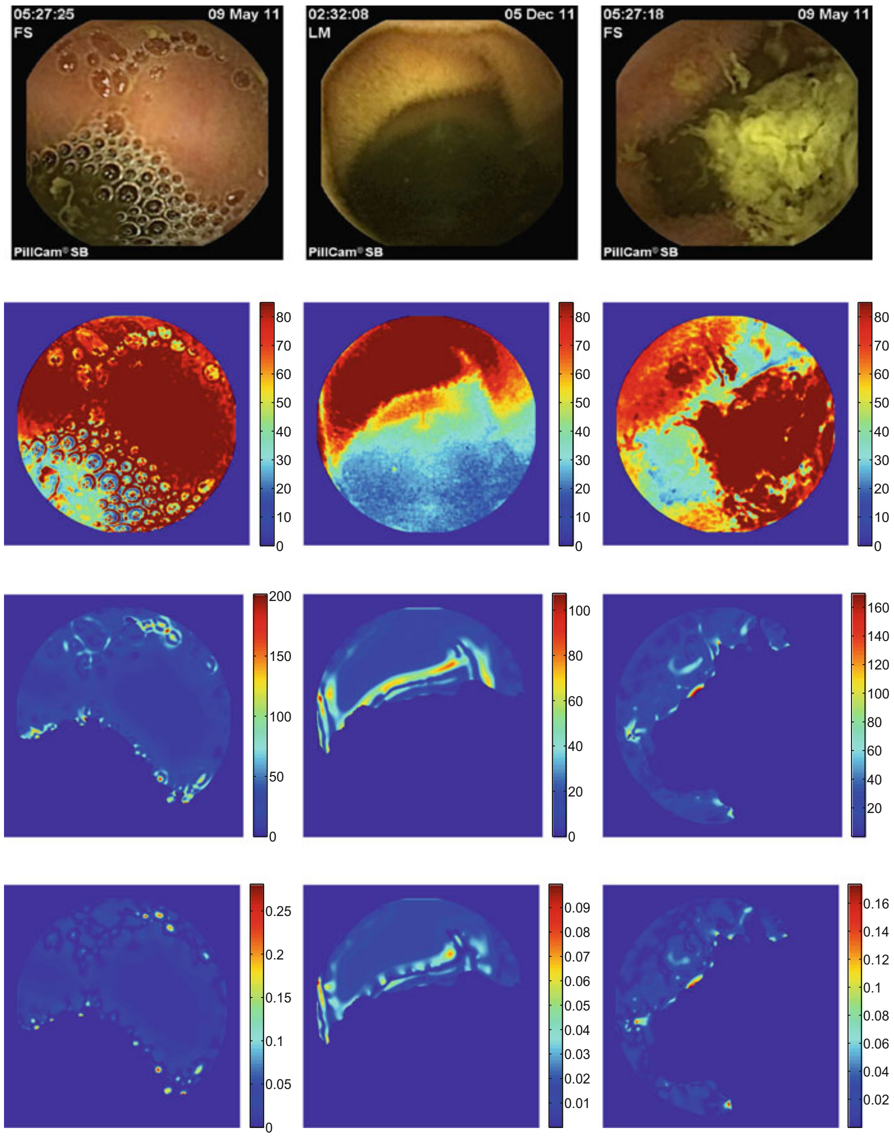


Fig. 4 *First row*: Original image without blood region. *Second row*: A_1 color channel. *Third row*: Function F. *Fourth row*: Function $B + R$

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TN} + \text{FP} + \text{TP} + \text{FN}},$$

where TP, FN, FP and TN represent the number of true positives, false negatives, false positives and true negatives, respectively. For a particular decision threshold T, if for an image frame J, $DF > T$, it is a positive frame; if $DF \leq T$, it is a negative

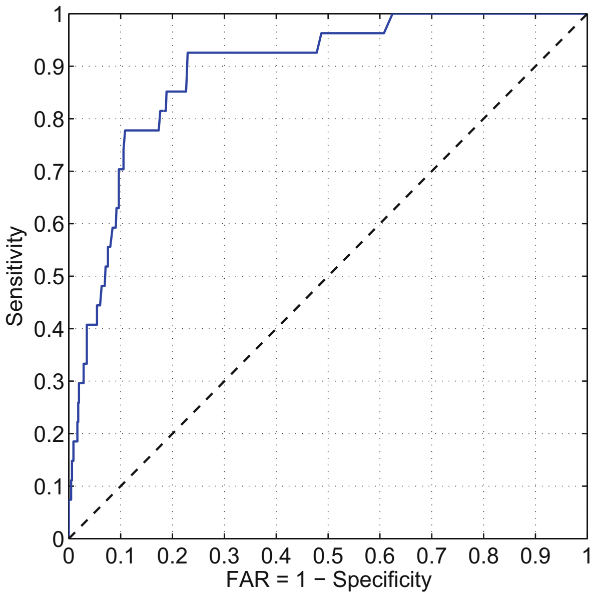


Fig. 5 ROC curve for function DF

frame. If J belongs to the class of blood image frames and it is classified as negative, it is counted as a false negative; if it is classified as positive, it is counted as a true positive. If J belongs to the class of non-blood image frames and it is classified as positive, it is counted as a false positive; if it is classified as negative, it is counted as a true negative.

Sensitivity represents the ability of the algorithm to correctly classify an image as a frame containing blood, while specificity represents the ability of the algorithm to correctly classify an image as a non-blood frame. The third measure, accuracy, is used to assess the overall performance of the algorithm. There is also another performance measure commonly used in the literature, false alarm rate (FAR). However, it can be computed from the specificity: FAR=1-Specificity.

Receiver operating characteristic (ROC) curve is a fundamental tool for detection evaluation. In a ROC curve sensitivity is plotted in function of FAR. Each point on the ROC curve represents a sensitivity/FAR pair corresponding to a particular decision threshold. It shows the tradeoff between sensitivity and specificity. Figure 5 represents the ROC curve with respect to the function DF. For FAR $\leq$ 10 %, the best sensitivity achieved is 70.37 %. In particular, the sensitivity, FAR and accuracy obtained are 70.37, 9.6 and 89.56 %, respectively, for the threshold $2.8928E + 007$. In summary, these results show that the presented algorithm is very promising for the detection of blood regions.

4 Speeding up the Segmentation and Detector Performance

In this section we describe general facts about the apparatus specifications. In particular, we detail the GPUs adopted and the underlying architectures. Finally, we address the parallelization of the algorithms proposed, namely by detailing the segmentation and blood detector parallelization procedures on the GPU, and reporting the results obtained for the current medical dataset.

The pipeline of the algorithm, described in Sect. 2, has been first implemented on a CPU Intel Core i7 950 CPU @ 3.07 GHz, with 12 GB of RAM, running a GNU/Linux kernel 3.8.0-31-generic. The C/C++ code was compiled using GCC-4.6.3.

In order to process more frames per second, the segmentation and blood detector steps have been paralellized, for executing on GPU NVidia C2050 and NVidia GTX 680, compiled using NVIDIA Compute Unified Device Architecture (CUDA) driver 5.5 [21].

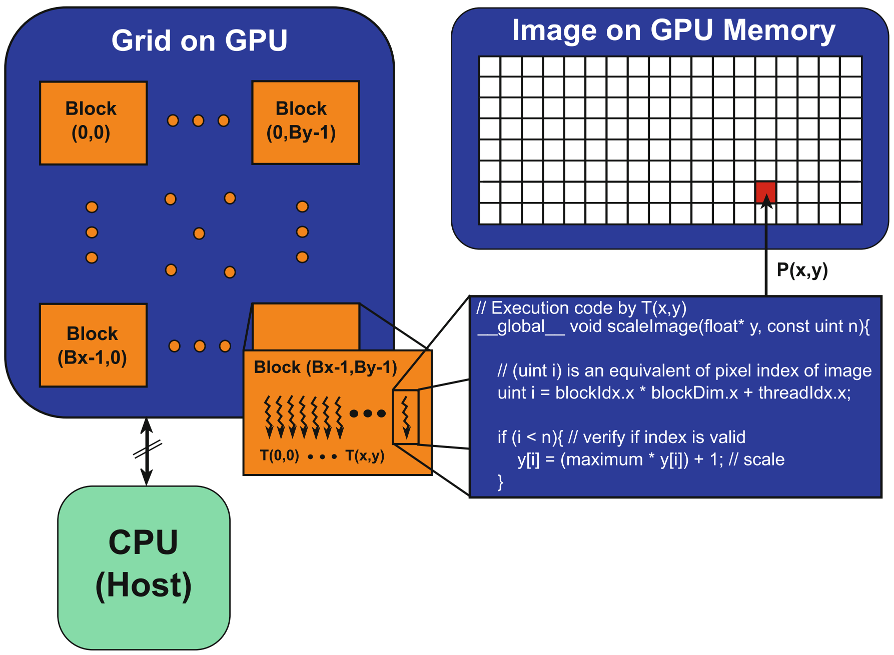
4.1 General Overview of the GPU Architecture

The host system usually consists of a CPU that orchestrates the entire processing by sending data and launching parallel kernels on the GPU device. At the end of processing, it collects computed data from the device and terminates execution. The parallelization of segmentation and blood detection procedures is carried out using the CUDA parallel programming model, by exploiting the massive use of thread- and data-parallelism on the GPU. CUDA allows the programmer to write in a transparent way, scalable parallel C code [21] on GPUs.

As shown in Fig. 6, each thread processes one pixel and thus multiple elements can be processed at the same time. This introduces a significant reduction in the global processing time of the proposed algorithm. When the host launches a parallel kernel, the GPU device executes a grid of thread blocks, where each block has a predefined number of threads executing the same code segment. Organized in groups of 32 threads (a warp), they execute synchronously and are time-sliced among the stream processors of each multiprocessor.

Figure 7 depicts a simplified overview of the GPU architecture. It shows that several multiprocessors contain a large number of stream processors (the number of stream processors and multiprocessors depends on the model and architecture of the GPU). In the present case, the NVidia GTX 680 GPU, which contains eight multiprocessors with each multiprocessor containing 192 stream processors, performing a total of 1536 CUDA cores, executes the algorithm faster.

Before processing starts on the GPU, data is uploaded to device memory. This process is typically slow and consists in transferring the information from the host CPU memory to the GPU global memory (device). At the end of the processing, results are transferred from the GPU device global memory to the host CPU RAM memory.



```
// Execution code by T(x,y)
__global__ void scaleImage(float* y, const uint n){

    // (uint i) is an equivalent of pixel index of image
    uint i = blockIdx.x * blockDim.x + threadIdx.x;

    if (i < n){ // verify if index is valid
        y[i] = (maximum * y[i]) + 1; // scale
    }
```

Fig. 6 Demonstration of the structure of a grid and thread blocks and how the same segment of code is executed by multiple threads. Each thread computes the result for one pixel

In the GPU, there are several memory types and they have different impacts on the throughput performance. We highlight two of them:

- Global memory accesses are time consuming operations with high latency and may represent a bottleneck in the desired system's performance. Instead, coalesced accesses should be performed whenever possible. They imply data in global memory to be contiguously aligned, so that all 32 threads within a warp can access the respective 32 data elements concurrently on the same clock cycle, with thread T(x,y) accessing pixel P(x,y), as depicted in Fig. 8.
- Also, modern GPUs have small and fast blocks of memory tightly coupled to the cores, which is shared by all threads within the same block. We can have several threads processing the same local data to optimize memory bandwidth (typically shared memory is faster than global memory when we need to share information among several threads), but shared memory is small in size. To maximize its use and performance, it is important to consider such size limitations. When large amounts of data have to be processed, data has to be partitioned in smaller blocks in order not to exceed the limits of shared memory. This action also represents penalties, since it increases the amount of data exchanges with global memory. Therefore, in the current work we use shared memory for calculating some procedures and global memory to perform the remaining functionalities, globally achieving an efficient memory usage as reported in later subsections.

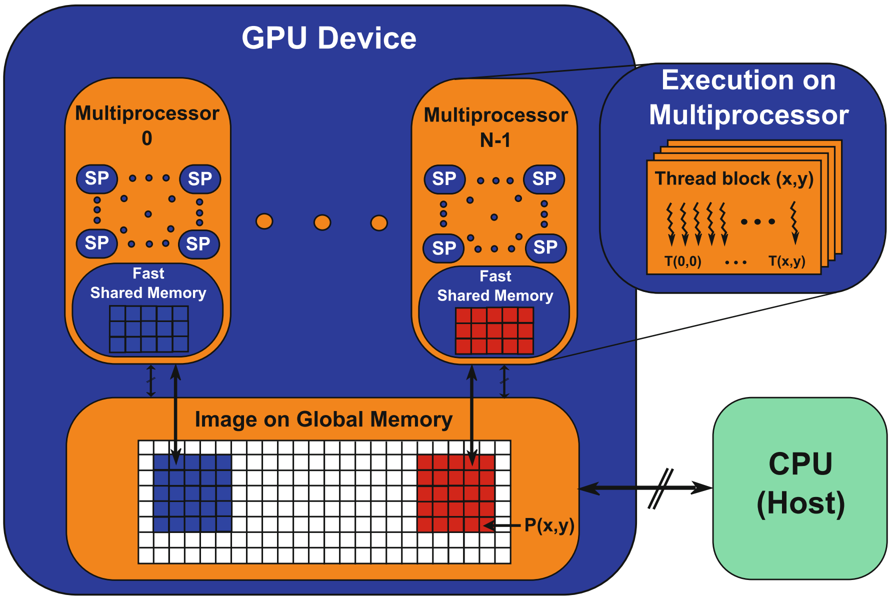


Fig. 7 Simplified GPU arquitecture. An example of how thread blocks are processed on GPU multiprocessors. A multiprocessor can execute more then one thread block concurrently

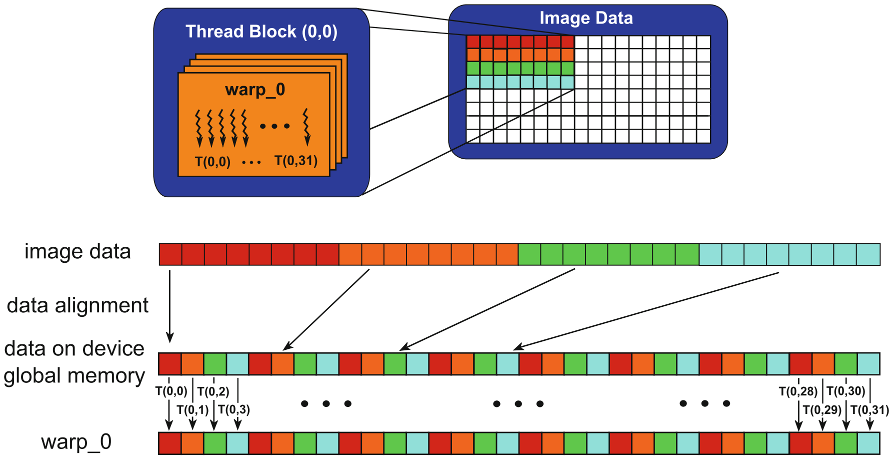


Fig. 8 Coalesced memory accesses illustrating a warp of 32 threads reading/writing the respective 32 data elements on a single clock cycle

4.2 Segmentation Parallelization

Some functions in the segmentation procedure, mentioned in Sect. 2.2, need to share image data between threads (e.g. neighboring pixels on the convolution procedure).

Table 1 Computation times in milliseconds (ms) for the segmentation procedure and throughput measured in frames per second (fps). The tests were performed on WCE images with 576 × 576 pixels

Processing Platform	Segmentation execution time (ms)	Segmentation (fps)
CPU Intel i7	240.0	4.2
GPU NVidia C2050	6.0	166.7
GPU NVidia GTX 680	4.8	208.3

Therefore, the use of shared memory is the best option to achieve a higher speedup (see [16] for a related work). These functions are: finding maximum and mean values, and 2D separable convolution [13]. All other functions perform slower if shared memory is used, because the total number of transactions to global memory will be greater.

The results of maximum and mean values are processed in two steps: the first step uses GPU grids with 256 × 256 block size; the second step uses 1 × 256 ; and in the 2D convolution, block sizes of dimension 16 × 16 are used.

The remaining functions in the segmentation step always use global memory and 1296 × 256 block sizes.

The computation times regarding the segmentation procedure are represented in Table 1, that shows the real speedups obtained using parallel computation on the GPU; as displayed, this procedure runs 40 times faster on GPU NVidia C2050 and 50 times faster on GPU NVidia GTX 680, when compared to an Intel i7 CPU.

4.3 *Blood Detector Parallelization*

For speeding up the blood detector procedure, described in Sect. 2.3, we only use one function that shares image data between threads: 2D separable convolution [13]. The remaining functions perform slower if we use shared memory because the total number of transactions to global memory would assume a higher impact. The results of 2D separable convolution are computed using block sizes of dimension 16 × 16 and 8 × 8 for the scale values $s = [8\ 10]$ and $s = [12\ 14]$ (see Sect. 2.3), respectively. All other functions always use global memory blocks with size 8 × 8.

The computation times of the blood detector procedure are presented in Table 2. We clearly see the speedup obtained using parallel computation on GPU. This algorithm runs 58.9 times faster on GPU NVidia C2050 and 59.5 times faster on GPU NVidia GTX 680, when compared to an Intel i7 CPU.

Table 2 Computation times in millisecons (ms) for the blood detector procedure and throughput measured in frames per second (fps). The tests were performed on WCE images with 576 × 576 pixels

Processing platform	Blood detector execution time (ms)	Blood detector (fps)
CPU Intel i7	529.9	1.9
GPU NVidia C2050	9.0	111.1
GPU NVidia GTX 680	8.9	112.4

Table 3 Throughput measured in fps and speedup archived to the complete algorithm (Segmentation and Blood Detector). Tests performed on WCE images with 576 × 576 pixels

Processing platform	Segmentation and blood detector (fps)	Speedup
CPU Intel i7	1.3	———
GPU NVidia C2050	66.7	51.3 times faster
GPU NVidia GTX 680	72.9	56.1 times faster

4.4 Speedup

Table 3 shows throughput measured in frames per second (fps) and the speedup of the full algorithm achieved. It can be seen that GPU NVidia GTX 680 is faster than NVidia C2050.

With the obtained speedup, the GPU NVidia GTX 680 shows to be able of processing 72 fps, which is equivalent to observe that the approximate total number of 56000 frames, generated by a complete WCE exam, can be computed in less than 13 min.

5 Conclusions

With the rapidly enhancing performances of graphics processors, improved programming support, and excellent price-to-performance ratio, GPUs have emerged as a competitive parallel computing platform for computationally expensive and demanding tasks in a wide range of medical image applications. We have proposed a GPU-based framework for blood detection in WCE images. The core of the algorithm lies in the definition of a good discriminator for blood and non-blood frames. This is accomplished by choosing a suitable color channel, image Hessian eigenvalue analysis and multiscale image analysis approach. Experimental results for our current dataset show that the proposed algorithm is effective, and achieves 89.56 % accuracy. Moreover, it is shown that the accelerated procedure is on average 50 times faster than the original one, and is able of processing 72 frames per second. This is achieved by parallelizing the two crucial steps, segmentation and blood detector functionalities in the algorithm, that were consuming most of the global processing time. To perform these steps more efficiently we now run parallel code on GPUs with an appropriate use of memory (shared and global). This novel approach allows processing multiple pixels of an image at the same time, thus sustaining the obtained throughput levels.

Acknowledgements This work was partially supported by the project PTDC/MATNAN/0593/2012, and also by CMUC and FCT (Portugal), through European program COMPETE/ FEDER and project PEst-C/MAT/UI0324/2011. The work of Gabriel Falcao was also partially supported by Instituto de Telecomunicações and by the project PEst-OE/EEI/LA0008/2013.

References

1. Bashar M, Kitasaka T, Suenaga Y, Mekada Y, Mori K (2010) Automatic detection of informative frames from wireless capsule endoscopy images. Med Image Anal 14:449–470
2. Bresson X, Esedoglu S, Vandergheynst P, Thiran JP, Osher S (2007) Fast global minimization of the active contour/snake model. J Math Imaging Vis 28:151–167
3. Chan TF, Vese LA (2001) Active contours without edges. IEEE Transac Image Process 10:266–277
4. Coimbra M, Cunha J (2006) MPEG-7 visual descriptors-contributions for automated feature extraction in capsule endoscopy. IEEE Transac Circuits Syst Video Technol 16:628–637
5. Cui L, Hu C, Zou Y, Meng MQH 2010) Bleeding detetction in wireless capsule endoscopy images by support vector classifier. In: Proceedings of the 2010 IEEE Conference on Information and Automation, pp. 1746–1751. Harbin, China, June 2010
6. Cunha JPS, Coimbra M, Campos P, Soares JM (2008) Automated topographic segmentation and transit time estimation in endoscopic capsule exams. IEEE Transac Med Imaging 27:19–27
7. Figueiredo IN, Kumar S, Figueiredo PN (2013) An intelligent system for polyp detection in wireless capsule endoscopy images. In: Computational Vision and Medical Image Processing IV: VIPIMAGE 2013, pp. 229–235. Madeira Island, Funchal, Portugal, 2013
8. Figueiredo IN, Kumar S, Leal C, Figueiredo PN (2013) An automatic blood detection algorithm for wireless capsule endoscopy images. In: Computational Vision and Medical Image Processing IV: VIPIMAGE 2013, pp. 237–241. Madeira Island, Funchal, Portugal, 2013
9. Figueiredo IN, Kumar S, Leal C, Figueiredo PN (2013) Computer-assisted bleeding detection in wireless capsule endoscopy images. Comput Meth Biomech Biomed Eng Imaging Visualization 1:198–210
10. Francis R (2004) Sensitivity and specificity of the red blood identification (RBIS) in video capsule endoscopy. In: 3rd international conference on capsule endoscopy. Miami, FL, USA, Feb 2004
11. Frangi AF, Niessen WJ, Vincken KL, Viergever MA (1998) Multiscale vessel enhancement filtering. In: Medical image computing and computer-assisted intervention, pp. 130–137. Cambridge, MA, USA, 1998
12. Idan G, Meron G, Glukhovsky A (2000) Wireless capsule endoscopy. Nature 405, 417–417
13. Lee H, Harris M, Young E, Podlozhnyuk V (2007) Image convolution with CUDA. NVIDIA Corporation
14. Li B, Q.-H-Meng M (2009) Computer-aided detection of bleeding regions for capsule endoscopy images. IEEE Transac Biomed Eng 56:1032–1039
15. Liedlgruber M, Uhl A (2011) Computer-aided decision support systems for endoscopy in the gastrointestinal tract: a review. IEEE Rev Biomed Eng 4:73–88
16. Martins M, Falcao G, Figueiredo IN (2013) Fast aberrant crypt foci segmentation on the GPU. In: ICASSP'13: Proceedings of the 36th IEEE International Conference on Acoustics, Speech and Signal Processing. IEEE

17. Ohta YI, Kanade T, Sakai T (1980) Color information for region segmentation. Comput Graphics Image Process 13:222–241
18. Pan G, Xu F, Chen J (2011) A novel algorithm for color similarity measurement and the application for bleeding detection in WCE. Int J Image Graphics Signal Process 5:1–7
19. Park SC, Chun HJ, Kim ES, Keum B, Seo YS, Kim YS, Jeen YT, Lee HS, Um SH, Kim CD, Ryu HS (2012) Sensitivity of the suspected blood indicator: an experimental study. World J Gastroenterol 18(31):4169–4174
20. Penna B, Tilloy T, Grangettoz M, Magli E, Olmo G (2009) A technique for blood detection in wireless capsule endoscopy images. In: 17th European signal processing conference (EUSIPCO 2009), pp. 1864–1868
21. Podlozhnyuk V, Harris M, Young E (2012) NVIDIA CUDA C programming guide. NVIDIA Corporation
22. Zheng Y, Yu J, Kang SB, Lin S, Kambhamettu C (2008) Single-image vignetting correction using radial gradient symmetry. In: Proceedings of the 26th IEEE conference on Computer Vision and Pattern Recognition (CVPR '08), pp. 1–8. Los Alamitos, California, USA, June 2008