Microprocessors and Microsystems 53 (2017) 190–201

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

GPU-based anisotropic diffusion algorithm for video image denoising

Amira Hadj Fredj*, Jihene Malek

Electronics and Micro-Electronics Laboratory, Faculty of Sciences, Monastir University, Monastir, Tunisia

ARTICLE INFO

Article history:
Received 1 February 2017
Revised 2 July 2017
Accepted 9 August 2017
Available online 10 August 2017

Keywords:
OSRAD filter
Graphic processor unit (GPU)
CUDA
Ultasound image
Real time implementation

ABSTRACT

Image filtering is the process of removing noise which perturbs image analysis methods. In some applications like segmentation, denoising is intended to smooth homogeneous areas while preserving the contours. Real-time denoising is required in a lot of applications like image-guided surgical interventions, video analysis and visual serving. This paper presents an anisotropic diffusion method named the Oriented Speckle Reducing Anisotropic Diffusion (OSRAD) filter. The OSRAD works very well for denoising images with speckle noise. However, this filter has a powerful computational complexity and is not suitable for real time implementation. The purpose of this study is to decrease the processing time implementation of the OSRAD filter using a parallel processor through the optimization of the graphics processor unit. The results show that the suggested method is very effective for real-time video processing. This implementation yields a denoising video rate of 25 frames per second for 128×128 pixels. The proposed model magnifies the acceleration of the image filtering to $30\times$ compared to the standard implementation of central processing units (CPU). A quantitative comparison measure is given by parameters like the mean structural similarity index, the peak signal-to-noise ratio and the figure of merit. The modified filter is faster than the conventional OSRAD and keeps a high image quality compared to the bilateral filter and the wavelet transformation.

1. Introduction

Image denoising is a needful part of image processing workflows. Denoising aims to decrease noise in homogeneous areas while preserving image contours. Denoising is important for pretreatment methods such as object recognition, segmentation [1], classification and pattern analysis [2]. Due to the special texture of ultrasound images, their denoising is particularly difficult [3]. Signal noise, frequently referencing to the speckle, is a granular noise detected within some types of coherent imaging systems as the example of ultrasound images and Synthetic Aperture Radar (SAR). Unfortunately, there has been a degradation of the optical ultrasound-image quality due to the existence of speckle noise. It has been hard to model such noise in real-time. An indefinite number of filtering methods have been put forward in the literature in the purpose of reducing the speckle noise. For instance, in [6] Achim et al. introduced a method that was novel and nonlinear. It was multiscale and homomorphic, in the objective of suppressing speckles in ultrasound images. The authors of the work of Coup et al. [7] studied a filter based on NonLocal (NL) means for ultrasound images by bringing the Pearson distance as a relevant standard for patch comparison. Some methods require wavelets in order to denoise ultrasound images [8,9]. Several medical-ultrasound-image applications have been devoted for the use of nonlinear diffusion filters. Unlike linear diffusion filtering, diffusivity is reduced by the anisotropic process at locations having a bigger probability of being edges by utilizing the stop edge function [4]. The mainly basic target of anisotropic diffusion is the incorporation of an adaptive smoothness constraint in the process of denoising. In other words, this smoothness is encouraged in a homogeneous area. Also, it is discouraged across boundaries, in the goal of preserving the image discontinuities. In the target of performing this task, Perona and Malik [10] gradually elaborated and created the Anisotropic Diffusion (PMAD) method. Indeed, the latter was a multi-scale smoothing and edge detection scheme, which was a powerful concept in image processing. It has been an adaptive edge-preserving smoothing technique that has been extensively utilized for image segmentation, image restoration, image smoothing, and edge and detail detection.

It was specifically important to denoise ultrasound images using the method suggested by Yu and Acton [11]. This method was called the Speckle Reducing Anisotropic Diffusion (SRAD). The SRAD filter was commonly utilized for reducing the speckle noise in particular. In this situation, the diffusion was controlled via the local image statistics. After that, for the enhancement of the quality of an image, Karl [12] applied on the image a directional filter

* Corresponding author.
E-mail addresses: amira.hadjfredj@gmail.com (A. Hadj Fredj), jihenemalek14@gmail.com (J. Malek).

http://dx.doi.org/10.1016/j.micpro.2017.08.003
0141-9331/

List of symbols

u	The noisy image.
u_0	The noisy image at $t = 0$.
div	divergence operator.
D	matrix diffusion.
∇u	gradient operator.
$v1$, $v2$	eigenvectors
1- k_{Kuan}	diffusion function based on Kuan filter.
1-k_{Lee}	diffusion function based on Lee filter.
c_{tang}	tangente constant.
v_f	The local spatial variance of u.
v_g	The local variance of u_0.
$\bar{g}$	The local mean of u_0.
$q(t)$	The instantaneous variation coefficient.
$q_0(t)$	The noise variation coefficient.
k	Iteration number.
η	neighborhood pixels.

through the addition of a non-scalar component to the SRAD algorithm, which was known as the Oriented Speckle Reducing Anisotropic Diffusion (OSRAD) filter. In fact, a similar kind of enhancement is the matrix anisotropic diffusion that adds to the standard scalar anisotropic diffusion. Practically, Karl [12] obtained results that are not suitable for real-time processing. It is clear that advances in ultrasound imaging keep bringing forth and producing larger body image datasets, higher resolutions, and more complex algorithms. Because of the fact that these advances grow faster than the Central Processing Unit (CPU) performance, the leveraging data-parallel techniques will be increasingly significant. In this way the Graphics Processing Units (GPUs) were used to provide a high performance platform for implementing applications in different areas. As there are many scientific problems that are governed by Partial Differential Equations (PDEs) and dummyTXdummy- require a fast implementation of the numerical methods used to obtain their solutions, it is important to efficiently implement those numeric solutions on GPUs. The use of a multi-core processor and GPUs accelerate the process of noise removal. As a result, it can be made while doing real-time filtering.

The contribution of this paper is the fast implementation of the OSRAD filter which is able to achieve a number of frames per seconds (fps) adapted for real-time applications by using a parallel computing model "CUDA". In order to reduce the speckle noise from ultrasound images, Karl in [12] smoothed the image by a convolution with a Gaussian filter in order to assess curvature directions computed on the smoothed image. Instead of using a Gaussian filter, a median filter was utilized for reducing speckle noise, which is more adapted in our case than a Gaussian function.

The proposed method is composed of two main parts: In the first time the diffusion coefficients are computed using both CPU and GPU processors, and in the second time the implementation of the numerical scheme utilizing the Jacobi method is done using a GPU processor. The kernels are implemented on a grid of $W/32 \times H/32$ thread blocks (where H & W are the image dimensions). Before doing any arithmetic operations, data accessed by each thread block are read first from the global memory. For the median filter a shared memory is used, $(BLOCK\_WIDTH + 2) \times (BLOCK\_HEIGHT + 2)$ in size. The latter has a higher latency than the global memory and a much higher bandwidth. The different steps of the algorithm appear in (Fig. 1). Compared with several denoised algorithms on the GPU, the modified version of the OSRAD filter presents more effective results in terms of processing time and quality of image.

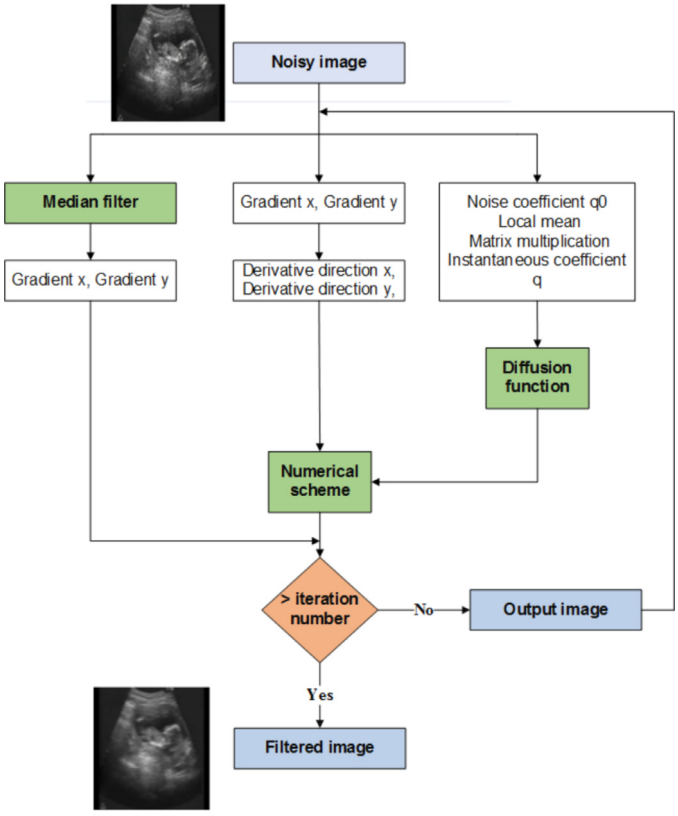


Fig. 1. Diagram organization of algorithm implemented.

The paper is organized as follows. The second section describes some related work. The third section gives a brief overview of the 2D OSRAD method. The fourth section discusses the GPU implementation. The fifth section presents the results. The advantages and limitations of the GPU implementation are presented in section six. Finally, the last section provides some conclusions and recommendations for future work.

2. Related work

In this section, we present some related work concerning image filtering and GPU implementation.

Wavelet transform: The wavelet transform is a multi-resolution analysis tool capable of transferring accurate temporal and spatial information. In the literature, various noise reduction techniques in relation to wavelet approaches have been put forward [13,14]. Li and Wang suggested in [15] a method based on wavelets. In their work, the authors decomposed a noisy image in the target of getting a different sub-band image. In addition to that, they maintained the coefficients of low-frequency wavelets unchanged. After that, and considering the relationship between vertical, horizontal and diagonal high-frequency wavelet coefficients, the writers compared the latter with the Donoho threshold in the purpose of achieving image denoising. In [13,16], the authors utilized the wavelet filters to denoise the speckle noise in optical coherence tomography data. Wavelets may be a good tool for image denoising owing to their energy density, thinness, and correlation properties. On the other hand, they are inadequate to a greater extent in their denoising performance as a result of simple thresholding methods. In order to improve efficiency when facing large data streams, such as from high resolution digital images or high definition videos, most of the Discrete Wavelet Transform (DWT) computations are performed on GPUs rather than CPUs. In [17] a GPGPU

framework was proposed for supporting wavelet-based denoising and process acceleration. This framework was designed to support wavelet-based denoising that would employ different forms of DWT and thresholding strategies through the use of GPU textures for updating the input parameters. In [18] the author explored the implementation of the 2D DWT on modern GPUs. In order to determine which algorithm was more suitable for such a platform, the author analyzed the performance of both the Filter Bank scheme (FBS) and the Lifting Scheme (LS).

Bilateral filter: Introduced by Tomasi and Manduchi [19], they are operators that preserve the edges. In a lot of computer vision and graphics tasks like denoising and texture editing, these operators have found widespread use. The idea of bilateral filtering is the computation of every pixel weight utilizing a spatial kernel and its multiplication using a function of influence in the intensity domain. This latter can decrease the pixel weight with large intensity differences. Nevertheless, under this form the filter cannot manipulate speckle noise. In addition, this filter tends to over smooth edges. A number of solutions have been already suggested to accelerate evaluating the bilateral filter [20,21]. In [22], a recursive implementation of the bilateral filter was proposed, in which computational and memory complexities were linear in both image size and dimensionality. On the other hand, its range filter kernel used pixel connectivity, and for that reason it could not be used directly for applications that in fact would ignore spatial relationships.

With the development of graphics processors, various implementations of bilateral filter have been proposed using the CUDA language. In [23], Maxime and François proposed a fast implementation of bilateral filter for the processing of large 3D volumes up to several GBs acquired by x-ray or electron tomography.

In [24] Howison presented an application of a rapid turnaround of MRI data smoothing over a range of filtering parameters. The idea was that between data acquisition and physical inspection there was a battery of filtering operations; the proposed method could minimize the time required for such a filtering battery, or could enable a real-time and interactive exploration of filter parameter choices. The CUDA programming language of NVIDIAs was used to take advantage of the high computational throughput of GPU accelerators. The author compared the performance of the CUDA implementations of bilateral and anisotropic diffusion filters for denoising 3D MRI datasets.

Anisotropic diffusion: Compared with other filters, anisotropic diffusion filters stand out thanks to their good speckle removal performance and better edge preservation quality. The PMAD filtering, being a very well known nonlinear technique [10] was developed from the heat diffusion equation through the introduction of a diffusion function dependent from the image gradient norm. As a matter of fact, the diffusion function can reduce high-gradient diffusion. However, the PMAD filter has two drawbacks. Firstly, it identically smooths data in all directions. Secondly, it is not easy to choose the threshold on the gradient norm necessary for the diffusion function. As a consequence, this technique will not be really automatic and cannot effectively remove noise. As a result, several studies have been done to improve the performance of anisotropic diffusion filter [25]. For example, Yu and Acton developed Speckle Reducing Anisotropic Diffusion (SRAD) for speckle reduction [11,26,27,5]. By exploiting an instantaneous coefficient of variation to adjust the weights of diffusion coefficients in iteration, it is proven to have a better performance in speckle removal and edge preservation than many other methods. The SRAD algorithm has been applied to 3D using CUDA, and the authors in [28] proved that the GPU implementation took less time, increased efficiency and fully met the needs of real time systems.

To adapt the SRAD correctly to multiplicative noise, a filter of Detail Preserving Anisotropic Diffusion (DPAD) was put forward by Aja-Fernandez and Alberola-Lopez [29]. This filter could estimate noise utilizing a whole-image local-statistics distribution mode. Jiang et al. [30] suggested a robust DPAD GPU implementation in order to improve the execution time by optimizing the logical judgment in a diffusion stop condition. However, the computation of the 3-D RDPAD algorithm required a huge amount of time to complete because it directly operated on the 3-D image and it had a logical judgment in every iteration. Recently, Karl et al. [12] combined the DPAD filter with an anisotropic diffusion matrix and presented an OSRAD. In [31] the authors described the implementation of the diffusion coefficient for the OSRAD method using the GPU. In the last few years, there have been some work that has exploited the speed capabilities of graphics hardware to accelerate the solution of diffusion equations. In [32], the paper described a novel GPU implementation of the Nonlinear Coherent Diffusion (NCD) algorithm for speckle reduction in ultrasound images. In [24], the author compared the performance of the CUDA implementations of bilateral and anisotropic diffusion filters for denoising 3D MRI datasets.

3. OSRAD algorithm

The OSRAD filter is a nonlinear filter that smooths an image while preserving major edges. To do so, Karl et al. [12] substituted a scalar diffusion function with a D matrix diffusion, which reoriented the diffusion on an orthonormal basis, $B = (v1^*, v2^*)$, related to the local image structures [31]. In addition to that, they utilized the eigenvalues of the D matrix to the same degree as a diffusion function associated with every basis vector depending on the local statistics and on the noise level of the image. Therefore, we may write the OSRAD filter as follows [31]:

$$\begin{cases} u(x,0) = u_0 = g \\ \dfrac{\partial u}{\partial t} = div(D\nabla u) \end{cases} \tag{1}$$

$$D = \begin{pmatrix} v1 | v2 \end{pmatrix} \begin{pmatrix} 1 - k_{Kuan} & 0 \\ 0 & c_{tang} \end{pmatrix} \begin{pmatrix} v1^T \\ v2^T \end{pmatrix} \tag{2}$$

D depends on the local u image structure, which will be diagonalized with the $\lambda 1$ and $\lambda 2$ eigenvalues as well as with the $v1$ and $v2$ eigenvectors, respectively. In such a case, $v1$ is the gradient direction and $v2$ the tangent curvature direction. The computation of these two is done on a smoothed Gaussian image. Added to that, ∇u is the gradient operator and div is the divergence operator. As a matter of fact, the filter should observe, along the $v1$ direction, the presence of an edge in the aim of smoothing or preserving the boundaries. Karl et al. [12] defined $\lambda 1$ as (1- k_{Kuan}) and $\lambda 2$ was fixed to the constant(c_{tang}). According to the definition of the OSRAD filter, the output can be calculated after a primordial process. Firstly, the diffusion function is based on the Kuan or the Lee filter. Secondly, a convolution with the Gaussian filter is required to compute the gradient directions of the derivative ones. It should be mentioned that the diffusion function can reduce high-gradient diffusion. There is no clearly evident diffusion function. As a result, this technique is not truly automatic and will not be able to effectively remove noise. Since the diffusion is defined as $1 - k = 1 - \frac{v_f}{v_g}$, we will use both the Lee and Kuan functions in order to compare them. They are both speckle filtering techniques but they present a small difference.

Kuan filter: It is derived from the Minimum Mean Square Error (MMSE) criteria under the assumption of a nonstationary mean and a nonstationary variance. It is quite similar to the Lee filter in form. However, it is considered to be a little more accurate than the Lee filter due to the fact that no approximation is required in the total derivation. The diffusion Kuan function can be written as

follows:

$$1 - k_{Kuan} = \frac{1 + 1/q(t)}{1 + 1/q_0(t)^2} \quad [26] \tag{3}$$

where $q(t)$ presents the instantaneous variation coefficient. Otherwise, it is an edge detector in the presence of speckles.

$$q(t) = \sqrt{\frac{v_g}{\bar{g}^2}} \tag{4}$$

where v_g is the local variance of the degraded g image, $\bar{g}$ is its local mean, and $q_0(t)$ presents the noise variation coefficient. It can be calculated from the mean and the standard deviation of the region of interest of the constant intensity. At each iteration, this parameter should be estimated, as it drives the locally-applied diffusion quantity.

Lee filter: It was derived from the MMSE criteria by introducing the local statistics method in it. It was believed that it could reduce speckle noise while preserving the edges in the image. The Lee diffusion function can be written as follows:

$$1 - k_{Lee} = \frac{1}{1 + (q(t) - q_0(t)^2)/q_0(t)^2(1 + q_0(t)^2)} \tag{5}$$

Finally, we have chosen a semi-explicit numerical scheme for the implementation of equation (1). As proposed in [12] we desire a good compromise among implementation ease, speed and accuracy, so, we use the Jacobi scheme, which has the advantage of being symmetric and straightforward to parallelize using a multithreading approach. The Jacobi numerical scheme is written as follows:

$$u^{k+1}(x) = \frac{u^k(x) + dt \sum_{n\in\eta} D_n^k u^k(n)}{1 + dt \sum_{n\in\eta} D_n^k} \quad [26] \tag{6}$$

where the subscripts are the pixel coordinates and the superscript is the iteration step. For every x point, η is its neighborhood in every direction while we denote by dt the step time between two successive iterations. For the time step and the number of iterations, $dt = 0.005$ with 300 iterations used in the experiments.

4. Complexity analysis on CPU

In our implementations of both sequential and parallel anisotropic diffusion computing algorithms in this paper, we have used the GCC g++ 4.9.2 compiler to generate the executable codes, which run under the operating system Ubuntu Linux 14.04 64-bit. We have tested all the C++ codes on an HP Notebook with Intel Core i7-4510U CPU @ 2 GHz. This processor has dual cores where each one can run four logical threads simultaneously. The CPU implementation uses the dual cores of the processor, for the reason that the execution time increases by using a single core depending on the size of the image. Thus, on the computing platforms based on Intel architecture, the usage of the instructions of Intel Streaming SIMD Extensions (SSE) is essential for efficient implementations of filters and other image processing tasks. The Intel C++ SIMD classes provide parallelism, which is not easily implemented using typical mechanisms of C++. Therefore, we have employed C++ class libraries to use the SSE which is easier to use and portable, but limitedly supporting instructions.

The implementation of the algorithm in C++ on the CPU request distribution program in a set of sub-functions following the steps already described in Fig. 1. After generating the source code successfully using the Qt Creator tool, it is time to embed the main entrances to see the result of filtering. Before compiling the source code, this series is introduced into the principal project file. Once the source code is updated and saved it should go to the compilation. We subsequently detail the different images used to test the effectiveness of our algorithm. An evaluation of the execution time of this algorithm has been performed. Note that the execution time of this algorithm has been performed. Note that the execution time of the numerical scheme function is much larger than the other functions of the program. Indeed, the required time for the numerical scheme represents 70% of the global execution time. Thus, the Gaussian function requires a double access memory for 1D smoothing along x and then 1D smoothing along y which enables optimization to be considered on this algorithm part. In this context, we focus on the use of the GPU to improve our processing, and it is done by utilizing the CUDA programming tool.

5. CUDA-based modified OSRAD method

One type of the smoothing techniques is median filtering, which is linear Gaussian. As a matter of fact, all smoothing techniques are effective in what relates to removing noise in smooth patches or as an alternative to smooth regions of a signal. On the other hand, they adversely affect edges. During the reduction of noise in a signal, preserving the edges is significant. The latter are very important for visual image appearance. For the levels of noise varying from small to moderate, the median filter is obviously manner better than the Gaussian blur in the removal of noise while keeping edges for a fixed window size. Nevertheless, its performance for high noise levels is less than the Gaussian blur; while on the contrary, it is particularly effective for salt and pepper noise (impulsive noise) and for speckle noise. For that reason, the median filtering is very widely utilized in the processing of digital images. As a result, a median technique should change the Gaussian smoothing.

The modified OSRAD methodology is explained in this section. By exploiting the SIMD parallelism, the GPU-implementation execution depends for the most part on easily making a parallel adaption to the denoising function. Furthermore, it is explained how to implement a CUDA filter version. First, the process is highly parallel: Each pixel in the image requires a similar processing. Second, all data required for the actual processing can be stored in the GPU memory, thus reducing drastically the bus traffic and increasing the performance. In [12], Karl et al. computed the derivative direction in a smoothed image, which was convolved with a Gaussian function. The convolution with a Gaussian function aims to reduce Gaussian noise, whereas, the US images are contaminated with speckle noise which is reduced by applying the median filter. For this, replacing the Gaussian function with a median function is proposed.

The organization of the algorithm is summed up in Fig. 2. A CUDA program is divided into a host part and a device part. The host part runs on a CPU while the device part runs on GPU. The OSRAD algorithm is also divided into two main parts; the computation of a matrix coefficient (diffusion function) and the implementation of a numerical scheme. In order to perform this task, the number of blocks and threads, the iterations number and the time step are chosen in the first step. Then, to calculate the noise coefficient q_0, a region of interest is selected, where a homogenous area is manually chosen. The calculation of this coefficient is applied to a few pixels and not to the whole image; for this purpose the host, and not on the GPU is carried out in order not to lose the transfer time between CPU - GPU and GPU - CPU, as this factor must be calculated at each iteration to evaluate the noise variation in the image. The rest of the algorithm is performed on the GPU, so to control the diffusion function, the edges in the presence of speckles must be detected by computing the q coefficient. To do this, three kernels are launched respectively: *local_mean*, *matrix_multiplication* and *compute_q*. As mentioned above, the diffusion is performed using Eq. (3) or Eq. (5). In both cases, the coefficient q and q_0 are used to run the $diffusion\_function$ kernel. Since this is a directional filter, two kernels are performed to cal-

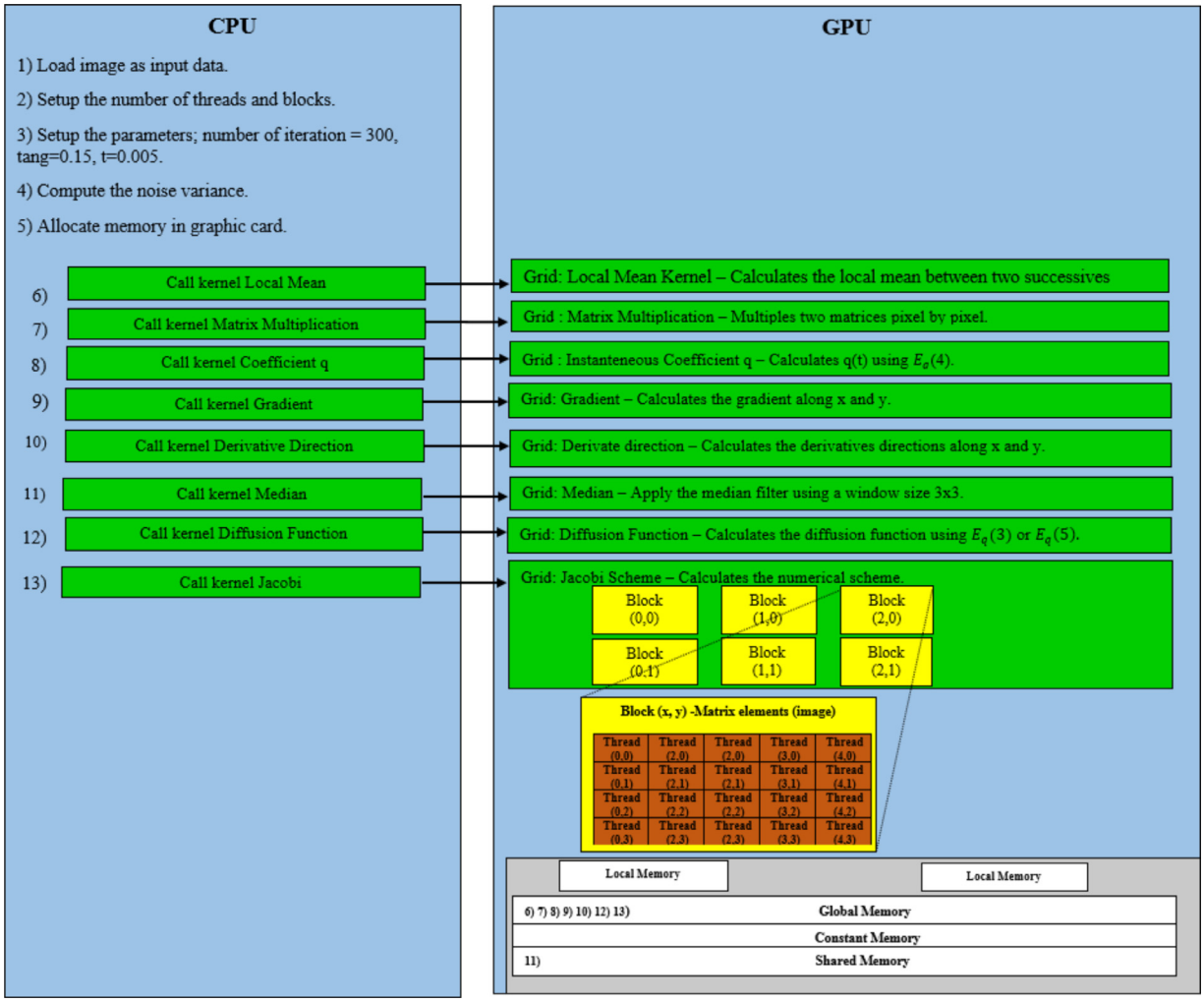


Fig. 2. Diagram block of the proposed GPU implementation.

culate the gradient along (x) and (y), *grad_x* and *grad_y* and two kernels to reach the derivative directions in the noisy and median filtered image.

The data transfer between the CPU and the GPU can be one of the major bottlenecks for achieving high performance. In our approach, data transfers are significantly reduced. Only the initial data are transferred from the host to the GPU. Intermediate data are built directly into the graphics memory.

6. Results

In this section, the performance evaluation of the OSRAD algorithm on both the CPU and the GPU is presented. As for the CPU implementation, the algorithm performed by Karl et al. [12] is tested in C++. The Opencv library (version 3.0.0) is used for image and video manipulation. Both CPU and CUDA implementations of the algorithm have been performed on a workstation whose characteristics are given in Table 1. In Table 2, the properties of NVIDIA Geforce 840M used in this work is provided.

Table 1
Test system main configuration.

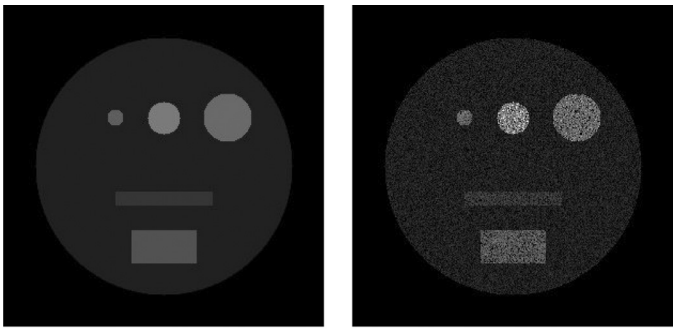
Parameters	Values
CPU	Intel Core i7-4510U
Number of cores	2
Frequency	2 GHz
RAM	8GB
GPU architecture	NVIDIA Geforce 840M
OS	Linux (Ubuntu 14.04)
CUDA	v7.5

Table 2
Properties of NVIDIA Geforce 840M.

Parameters	Values
Number of CUDA cores	384
Memory speed	900 MHz
Dedicated memory	4GB
Clock rate	1.12 GHz
Memory bandwidth	16 GB/s

6.1. Modified OSRAD versus original OSRAD

To improve the accuracy of the results and save time implementation, a median filter is used instead of a Gaussian function. With the median filter, a smoothed image is obtained in which the gradients are calculated in order to compute the derivative direction. A test with several median window sizes is done, so as to achieve a lower execution time and a higher PSNR. A synthetic image of size 300×300 (Fig. 3a) is utilized, which is corrupted with multiplicative speckle noise having a mean 1 and a standard deviation 0.5 [12] (Fig. 3b).



(a) (b)

Fig. 3. (a) Original synthetic 2D image, (b) Original image degraded with multiplicative speckle noise of mean 1 and standard deviation 0.5.

Table 3
Time and PSNR measures for different median window.

Window size	Time (ms)	PSNR (dB)
7×7	4.12	42.05
5×5	1.21	42.79
3×3	0.25	44.02

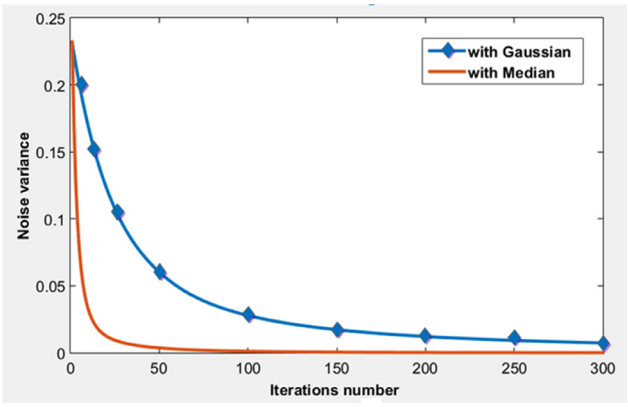


Fig. 4. Variation of the noise variance over time with both Gaussian and median function.

Table 3 shows the elapsed time for (3×3), (5×5) and (7×7) window sizes for one iteration and the PSNR measure for the denoise image where the parameters of the filter are mentioned in Table 4. It is observed from the figure that as the window size grows, the execution time also increases. With a large window size (7×7), the PSNR decreases compared to other window sizes, which is equal to 42.05 dB. In other words, the quality of the image is degraded due to the use of this window. Otherwise, a window of size (3×3) runs at only 0.25 ms allows a good performance of PSNR, which is equal to 44.02 dB. For the rest of the experiments a (3×3) window size is used.

To approve the use of the median filter, a comparison is made. Its noise variance is measured with both Gaussian and median functions. For the involved algorithm, the iteration number determines the denoising result. Therefore, the iteration number is changed and the variation of the noise variance over time is plotted (Fig. 4).

It is observed from the orange curve, that noise variance presents a drop compared to the blue curve, which means that the denoising process is much faster with the median function compared to the Gaussian function. In this case we can stop the filtering process when the noise variance is close to be constant, i.e.

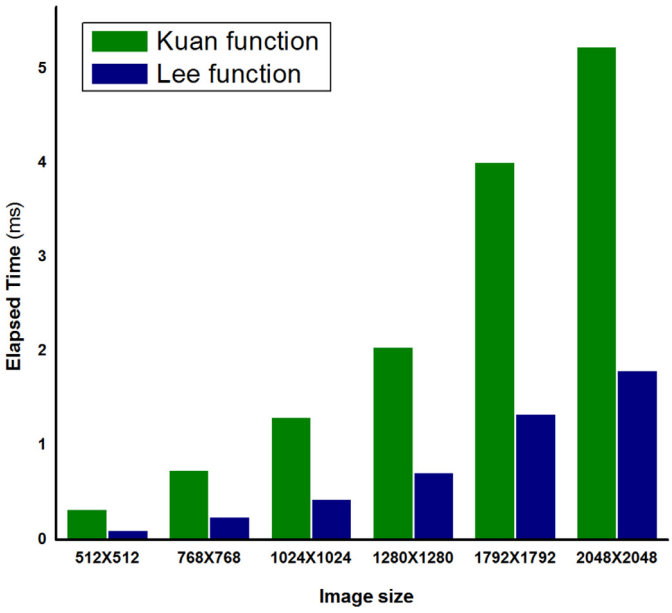


Fig. 5. Running time of Lee and Kuan diffusion function on GPU.

between 100 and 150 iterations. In this way, the number of iterations is reduced and consequently the processing time decreases.

As mentioned above, the resolution of the diffusion equation depends for a large part on the computation of the diffusion coefficient "diffusion function kernel" (Fig. 2). This latter can use the Lee mode or the Kuan mode. In order to compare them, an implementation for both functions is done on the GPU. It can be seen (in Fig. 5) that the Lee function is approximately three times faster than the Kuan function. For an image size of 2048×2048, the Lee mode runs at 1.8 ms whereas the Kuan mode runs at 5.24 ms. On the other hand, with the Lee mode, OSRAD filtering presents a PSNR of approximately 44 dB, which is the same with Kuan.

Both function Kuan and Lee depend on q and q_0. $1 - k_{Kuan}$ tends to $+\infty$ as q tends to 0 which needs to know for each pixel the value of q. In the CUDA kernel, it must access memory for compare the q value. This memory access decreases the running time. While, using Lee function $1 - k_{Lee}$ is bounded by $1 + \frac{1}{q_0^2}$ with q_0^2 is a constant which is computed in advance for each iteration. For this reason the Lee function is faster than the Kuan function in execution.

$$\lim_{q \to 0^+} 1 - k_{Lee} = 1 + \frac{1}{q_0^2} \qquad \lim_{q \to +\infty} 1 - k_{Lee} = 0$$

$$\lim_{q \to 0^+} 1 - k_{Kuan} = +\infty \qquad \lim_{q \to +\infty} 1 - k_{Kuan} = \frac{q_0^2}{1 + q_0^2}$$

In order to demonstrate the denoised process with the modified OSRAD algorithm, some initial simulation experiments using a synthetic image has been conducted. In this regard, a comparison study of the devoted parallel implementation against the C++ reference implementation is made in terms of time performance and Peak Signal to Noise Ratio (PSNR), where the higher the PSNR the better the result. The Pratt's Figure Of Merit (FOM) metric, which is used to estimate the quality of the contours and which is between 0 and 1, is defined as follows [33]:

$$FOM = \frac{1}{max(N_I, N_B)} \sum_{i=1}^{N_B} \frac{1}{1 + \alpha . d_i^2} \tag{7}$$

where N_I and N_B are respectively the points of edges in the image and the ground truth image, d_i is the distance between an

Table 4
Parameters and results of the OSRAD filter on the GPU and the CPU for the synthetic 2D image of size 300×300.

Method	Δt	tang	iter	PSNR (dB)	FOM	Time processing (Second)
CPU version [12]	0.05	0.5	300	44.72	0.757	56.30
GPU version	0.005	0.15	300	44.02	0.881	1.75

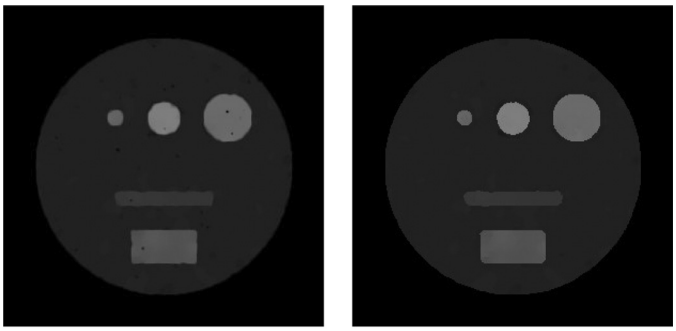


Fig. 6. Image denoising simulation, (a) CPU version [12], (b) GPU version.

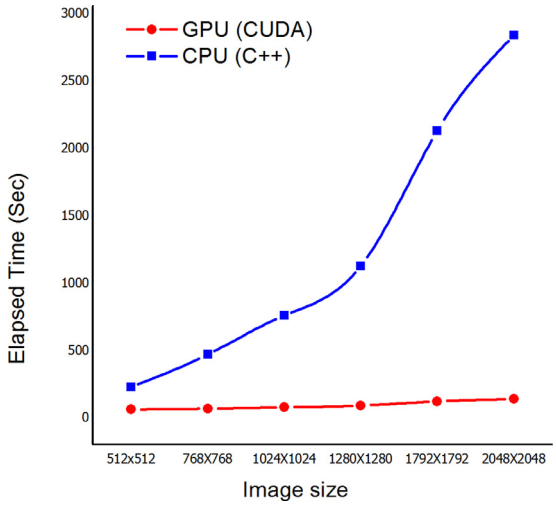


Fig. 7. Computation time versus image size for OSRAD filter on the CPU and the GPU. (For interpretation of the references to color in the text, the reader is referred to the web version of this article.)

edge pixel and the nearest edge pixel of the ground truth, and α is an empirical calibration constant $\alpha = 1/9$. In the experiments, we have used the Canny edge detector with a standard deviation of 0.1 and a threshold of 0.5.

In the first experiment, the synthetic image of (Fig. 3) is used. (Fig. 6a and b) respectively depict the filtering results using the CPU and the GPU. We notice that the GPU implementation is close the CPU one. The noise is reduced and the edges are well preserved.

The parameters of each filter are mentioned in Table 4, where the step time is denoted by Δt, the number of iterations is denoted by *iter* and the tangent is denoted by *tang*. In the proposed algorithm, the smoothing step time is set to 0.005, the *tang* is set to 0.15 and the denoising process runs adaptively with 300 iterations. In all experiments, the parameters of the filter are defined empirically.

The implementation made in this work shows good results for both the PSNR, which is equal to 44.02 dB, and the FOM score, which is equal to 0.881 (better than the CPU implementation). This means that the denoised image using CUDA produces smoother results, while the edges are well preserved. Having analyzed Table 4, one can deduce that the processing time required for implementing the anisotropic diffusion algorithm using CUDA is drastically reduced with the GPU. In particular, the parallel implementation of the OSRAD algorithm takes only 1.75 s for the 300×300 pixel image enhanced in contrast to 56.30 s required with the CPU implementation. Thus, the processing time of the proposed GPU-based algorithm implementation is approximately 32 times lower than the corresponding processing time achievable with the conventional CPU implementation. Since CUDA cores are responsible for dealing with all the data that move through a GPU, a graphics card with a high number of CUDA cores this can consistently improve performance. Titan X Pascal is a platform that have 3584 CUDA cores. Using the Titan X Pascal GPU card, the performance gain will be increase by 10.

From (Fig. 7), it is shown that the experiments have been done on various image sizes. In order to evaluate the execution time of the suggested parallel implementation in images of a larger size, the original synthetic image is enlarged (Fig. 3(a)) by up to 16 MB. This enlargement is done to evaluate the execution time of the proposed method in a larger size image.

It is observed from the curve of (Fig. 7), that the execution time of the parallel implementation is slower compared to the sequential implementation for different sizes. The processing time taken by the GPU (red curve) remains constant for the processing images with 512×512, 768×768, 1024×1024 and 1280×1280. Over this resolution, the processing times taken by the GPU tend to increase as the computation time rises. Either with a sequential computation (blue curve), the execution time will rise when the resolution increases. For the image of size 1024×1024, the GPU runs around $33\times$ faster than on the CPU.

6.2. Comparison of filtering techniques

6.2.1. Results for 2D synthetic Images

In order to verify the capability of edge-preservation, different smoothing effects with different methods using the GPU have been performed. (Fig. 8) shows a comparison of the modified OSRAD filter, the wavelet transformation and the bilateral filter. As it is well known, all these methods have been used for denoising the speckle noise. As illustrated in (Fig. 8a), a synthetic image of size 512×512 is artificially corrupted with speckle noise of a mean 1 and a standard deviation 0.5 (Fig. 8b). The parameters of each filter are mentioned in Table 5, where a DWT and DWT inverse transformation is done for the wavelet method with four levels. For the bilateral filter, a window size $\eta_x = 5 \times 5$ is used. In (Fig. 8d), it can be seen that the bilateral filter preserves the edge of the image, but the noise level cannot be efficiently reduced. It is clearly observed that the light regions (circle and rectangle) are still noisy. On the other hand, the wavelet can effectively reduce the noise, but also remove the edges (Fig. 8e). Contrarily, denoising with the OSRAD filter (Fig. 8c) is more effective than the other methods and the edges are preserved.

To evaluate the numerical accuracy, four quality indexes are used: the PSNR rate, the FOM metric, the Structural Similarity Index Measure (SSIM) [34], used to measure the similarity between

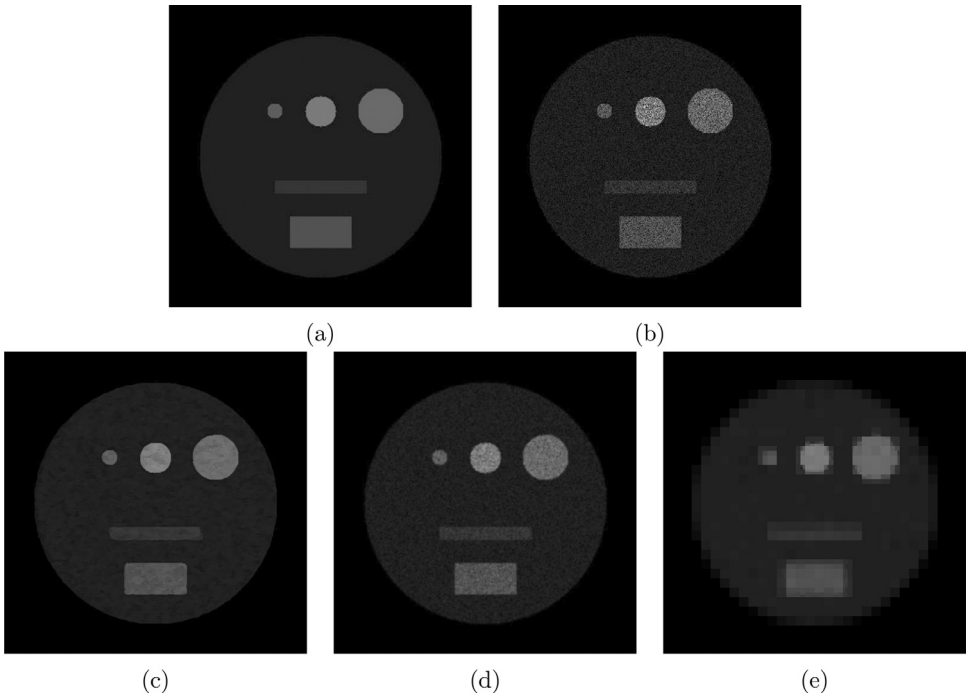


Fig. 8. (a) Original synthetic 2D image. (b) Original image with a Speckle noise with a 1 mean and standard deviation 0.5, (c) Results of OSRAD, (d) Bilateral, (e) Wavelet.

Table 5
Parameters and results of different filters on the GPU for the synthetic 2D image.

Image size	Filter	Δt	tang	iter	level	η_x	PSNR (dB)	SSIM	FOM	IQI	Time (seconds)
512×512	OSRAD	0.005	0.15	300	–	–	42.30	0.975	0.765	0.577	5.75
	Wavelet	–	–	–	4	–	38.32	0.954	0.566	0.770	13.61
	Bilateral	–	–	–	–	5×5	41	0.965	0.590	0.567	5.61
768×768	OSRAD	0.005	0.15	300	–	–	43.15	0.977	0.559	0.574	12.69
	Wavelet	–	–	–	4	–	40.30	0.971	0.457	0.795	15.99
	Bilateral	–	–	–	–	5×5	41.38	0.966	0.457	0.567	11.2
1024×1024	OSRAD	0.005	0.15	300	–	–	43.47	0.977	0.492	0.577	22.15
	Wavelet	–	–	–	4	–	41.75	0.978	0.407	0.828	27.19
	Bilateral	–	–	–	–	5×5	41.42	0.966	0.395	0.576	22.41
1280×1280	OSRAD	0.005	0.15	300	–	–	43.83	0.977	0.456	0.577	36.49
	Wavelet	–	–	–	4	–	42.85	0.983	0.425	0.827	40.79
	Bilateral	–	–	–	–	5×5	41.48	0.966	0.384	0.579	36.5
1792×1792	OSRAD	0.005	0.15	300	–	–	44.02	0.978	0.434	0.576	65.19
	Wavelet	–	–	–	4	–	44.39	0.978	0.421	0.832	68.11
	Bilateral	–	–	–	–	5×5	41.50	0.965	0.366	0.576	62.1
2048×2048	OSRAD	0.005	0.15	300	–	–	42.25	0.968	0.389	0.577	84.3
	Wavelet	–	–	–	4	–	44.90	0.988	0.356	0.849	91.38
	Bilateral	–	–	–	–	5×5	41.52	0.965	0.471	0.576	82.30

the noise-free image and the denoised one which is between 0 and 1, and the Image Quality Index (IQI), utilized to measure the similarity between the original image and the filtered one. The simulation results for different methods are presented in Table 5, where the anisotropic diffusion has a better behavior for the PSNR, SSIM and FOM measures for different resolutions. For an image size 512×512, the OSRAD filter has a good performance with the greatest PSNR value, which is equal to 42.30 dB, the highest MSSIM score which is equal to 0.975 (close to 1), and the highest FOM which is equal to 0.765. For the quality index, the wavelet has the best IQI, which is equal to 0.770. It can be seen from the table that the PSNR increases when the resolution goes up for different denoising methods. On the contrary, the FOM measure decreases with high image resolutions.

For the execution time, it can be observed that the OSRAD filter processing is faster than the wavelet denoising (5.75 s) for an image size 512×512 and close to the bilateral filter. As a conclusion, the anisotropic diffusion implementation on the GPU achieves a compromise for both quality and execution time. In (Fig. 9), the performance comparison of the modified OSRAD, the bilateral filter and the wavelet filter is shown. For different image sizes, the execution time is computed. The OSRAD GPU implementation takes significantly less time than the wavelet transformation, and it is close to the bilateral filter. To conclude, in the case of large image set computations like, 2048×2048, the proposed GPU implementation gives quick results and a more accurate denoising result compared to other methods.

6.2.2. Results for real ultrasound video images

Ultrasonography has become a very common tool for imaging physiological systems in the body. After X-ray imaging, ultrasonography is the most commonly performed procedure because it is real-time, non-radioactive and non-invasive. The most important parts for clinicians in the ultrasound image are the edges and local details existing between heterogeneous regions. Therefore, preserving and enhancing edges and local fine structures, and at the

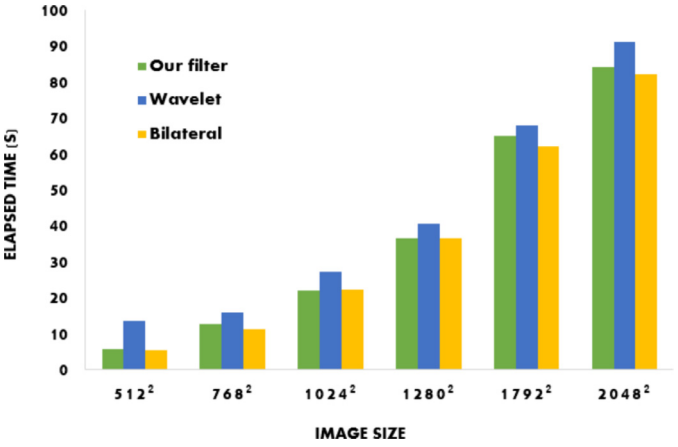


Fig. 9. Comparison results for different resolutions for the OSRAD, the Bilateral and the Wavelet filters.

Fig. 10. The original image and the image smoothed by the parallel implementation after 50 iterations, respectively.

same time reducing the noise, are very important. In this experiment, a real ultrasonography sequence composed of 240 frames of 128×128 pixels is used. The runtime for the real ultrasound video is 9.6 s, with 50 iterations for the parallel implementation in CUDA. Thus, the result locks the output frame rate at 25 fps, which is achieved in a real time solution. (Fig. 10) depicts an example of the smoothing of a video frame selected randomly from the tested video.

Before summarizing, a comparison study is done for real ultrasound images between different denoising methods: OSRAD, bilateral and wavelet. (Fig. 11c) shows the effect of anisotropic diffusion, where the noise is smoothed in homogenous areas of the image, while preserving edges and enhancing them. However, with the bilateral filter, the noise is slightly decreased (Fig. 11d), and with DWT transformation, the edges are greatly affected and the image quality is degraded (Fig. 11e).

As it is mentioned above, the total number of threads in an allocated block should be determined according to the size of the image and the capacity of the GPU. We propose here to study the influence of the block size on the measurement of execution timer for the three algorithms: the OSRAD, the wavelet filter and the bilateral filter.

We define square blocks of size n in a range (8–512). The following figure presents the execution time depending on the block size n. As shown in Fig. 12, the lowest execution time is obtained for blocks 32×32 threads for the ultrasound image for three implementation when running on the GPU. The results demonstrate that the execution time improves further with the increase in the number of thread in block at start time. Nevertheless, there is an

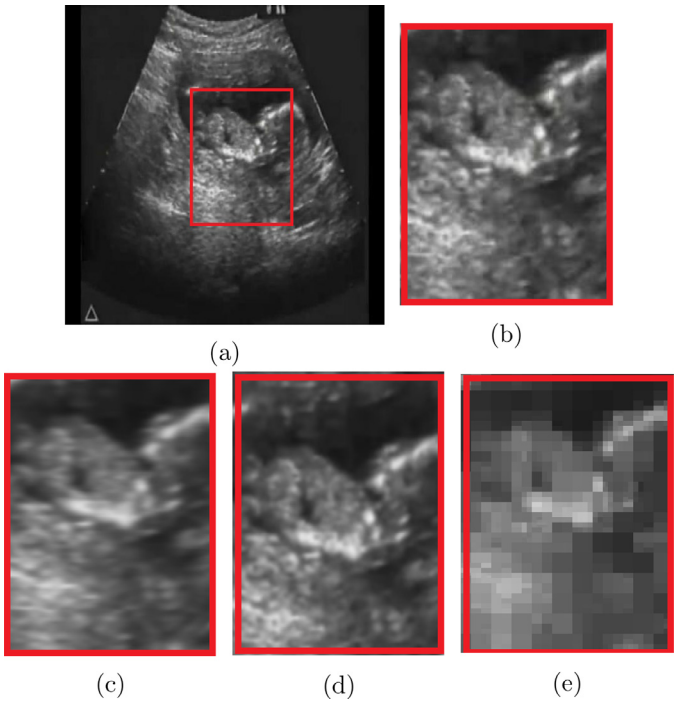


Fig. 11. (a) The 80th sample frame from the ultrasound video, (b) Sub image of the image (a), (c) Simulation results with OSRAD ($iter = 300$, $tang = 0.15$, $\Delta t = 0.005$), (d) Bilateral ($\eta = 5 \times 5$), (e) DWT ($Level = 4$).

Table 6
Parameters and results of Frames Per Second (FPS) rate obtained in the GPU for different denoising methods.

Filter	Δt	tang	iter	level	η_x	Total of FPS
OSRAD_GPU	0.005	0.15	50	–	–	25
Wavelet_GPU	–	–	–	4	–	10
Bilateral_GPU	–	–	–	–	5×5	7
OSRAD_CPU	0.05	0.5	300	–	–	8
DPAD_CPU	0.02	–	100	–	–	1.26
SRAD_CPU	0.025	–	40	–	–	2.19

upper limit to the performance improvement. As shown in Fig. 12, execution time becomes less obvious when the number of thread in block exceeds the maximum number of threads on our GPU card. The reason for this is linked to the occupation of the computing cores as recommended by NVidia strategy indicating that the more we defined large blocks, we obtain a more maximum occupancy of the GPU processing speeds. The suggested OSRAD filter presents the best performance in execution time, which is too close to the bilateral one. Yet, the wavelet function takes more time for different sizes of blocks. This issue is essentially due to the DWT and DWT inverse function.

From Table 6, the wavelet transformation achieves only 10 fps and the bilateral filter results in 7 fps, which is not suitable for medical applications in real time. For the CPU, we test the video sequence with the OSRAD, the DPAD and the SRAD. The result shows the output frame rate at 8, 1.26 and 2.19 fps, respectively.

Here, the performance gain of the parallel GPU-based implementation confirms the high processing capacity available in the CUDA architecture, with the video used in the experiment processed in real-time. Furthermore, the CUDA architecture as a computational infrastructure for image pre-processing is revealed to be viable and capable and it can be an alternative option to deliver high performance processing in a lot of applications. Moreover, it can even provide real-time processing at an affordable cost. Thus, the advantage of using GPU-based implementations can be totally

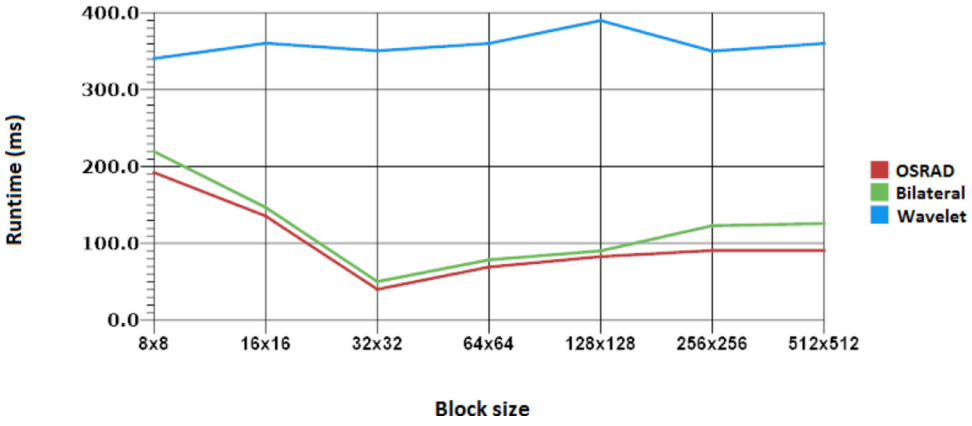


Fig. 12. Influence of block size on the GPU execution time.

warranted since the reduction in the execution time can minimize or even eliminate time restrictions. Such restrictions are common in many applications (such as in the medical field) which use image processing and analysis methods, requiring fast or real-time results for image-based diagnosis. However, when using the CUDA architecture in particular, an optimal implementation requires a maximum effort.

7. Advantages and limitations

The OSRAD method has been developed to remove speckle noise, a form of multiplicative noise, in imagery. This filter works very well in the CPU, but it requires more execution time, which is undesirable for real time applications like ultrasound echography. The main advantage of the proposed GPU implementation is the acceleration of processing time with respect to the CPU. Among the strong points in this implementation, the GPU memory is allocated just once at the beginning. Then the memory is accessed and changed in any kernel calls. At the end, the results are brought back from the GPU memory to the host just once, which increases the processing time significantly. Also, the parallel implementation produces better results for a single image with various sizes. After quality index measures, the suggested filter shows a very good performance both in edge preservation and noise cleaning. One disadvantage of using a GPU co-processor to accelerate computations is the cost of transferring data between the main memory on the host system and the GPU memory. Another limitation appears not in utilizing texture memory but in using shared memory only for the median kernel. At the same time, this can increase the performance because the access to these memories is faster.

8. Conclusion

A uniform framework enabling real time OSRAD filtering and parallel implementation is presented in this paper. The experimental results show that the proposed method outperforms other advanced methods for speed. The design optimization strategy is based on replacing the Gaussian function with a median function in order to enhance the quality of the image. The OSRAD filter has been successfully implemented using the NVIDIA Geforce 840M. The platform demonstrates that the modifiable OSRAD algorithm can operate at 25 frames per second for a video resolution 128×128. The suggested algorithm has been tested on several image types, like synthetic and real ultrasound images. In addition, the obtained results of the OSRAD filter have been compared to the bilateral filter and the wavelet transformation in terms of PSNR, MSSIM, FOM and IQI. Besides, it has been clear that the results obtained through the GPU implementation are higher than the ones got through the CPU implementation. As for future work, a hardware/software co-design for an FPGA platform will be developed for faster time-to-market solutions in the market of embedded system applications. The 3D algorithm operates directly on an image volume so that it greatly improves the performance of speckle reduction for 3D ultrasound images and keeps the advantages of the conventional anisotropic diffusion. We will extend our work to 3D filtering in a future research.

References

[1] M. Ben Abdallah, J. Malek, K. Krissian, R. Tourki, An automated vessel segmentation of retinal images using multiscale vesselness, in: In Systems, Signals and Devices (SSD), 8th International Multi-Conference on, Sousse, Tunisia, IEEE, March 2011, pp. 1–6.
[2] J. Malek, A.T. Azar, R. Tourki, Impact of retinal vascular tortuosity on retinal circulation, Neural Computing and Applications 26 (1) (2015) 25–40 Springer, doi:10.1007/s00521-014-1657-2.
[3] M. Ben Abdallah, J. Malek, R. Tourki, J.E. Monreal, K. Krissian, Automatic estimation of the noise model in fundus images, in: In Systems, Signals & Devices (SSD), 2013 10th International Multi-Conference on, Hammamet , Tunisia, IEEE, 2013, pp. 1–5.
[4] M. Ben Abdallah, J. Malek, R. Tourki, K. Krissian, Restoration of retinal images using anisotropic diffusion like algorithms, in: Computer Vision in Remote Sensing (CVRS), 2012 International Conference on, Xiamen, China, IEEE, Dec 2012, pp. 116–121.
[5] A.H. Fredj, M. Ben Abdallah, J. Malek, A.T. Azar, Fundus image denoising using FPGA hardware architecture, Int. J. Computer Applications in Technology 54 (1) (2016) 1–13.
[6] A. Achim, A. Bezerianos, P. Tsakalides, Novel bayesian multiscale method for speckle removal in medical ultrasound images, IEEE Trans. Med. Imaging 20 (2001) 772–783.
[7] P. Coup, P. Hellier, C. Kervrann, C. Barillot, Nonlocal means-based speckle filtering for ultrasound images, IEEE Trans. Image Process. 18 (2009) 2221–2228.
[8] S. Gupta, R. Chauhan, S. Sexana, Wavelet-based statistical approach for speckle reduction in medical ultrasound images, Med. Biol. Eng. Comput. (2004) 189–192.
[9] A. Thakur, R.S. Anand, Image quality based comparative evaluation of wavelet filters in ultrasound speckle reduction, Digit. Signal Process. (2005) 455–465.
[10] P. Perona, J. Malik, Scale-space and edge detection using anisotropic diffusion, IEEE Trans. Pattern Anal. Mach. Intell. 12 (1990) 629–638.
[11] Y. Yu, S.T. Acton, Speckle reducing anisotropic diffusion, IEEE Trans. Image Process. 11 (11) (2002) 1260–1270.
[12] K. Krissian, C.F. Westin., R. Kikinis, K. Vosburgh, Oriented speckle reducing anisotropic diffusion, IEEE Trans. Image Process. (2007) 1412–1424.
[13] W. Habib, A.M. Siddiqui, I. Touqir, Wavelet based despeckling of multiframe optical coherence tomography data using similarity measure and anisotropic diffusion filtering, in: 2013 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). Shanghai, 2013, pp. 330–333.
[14] S. Mallat, W.L. Hwang, Singularity detection and processing with wavelets., IEEE Trans. Inf. Theory 38 (2) (1992) 617–643.
[15] H. Li, S. Wang, A new image denoising method using wavelet transform., in: International forum on information technology and applications, IFITA 09. In Chengdu, 2009, pp. 111–114.
[16] M.A. Mayer, A. Borsdorf, M. Wagner, J. Hornegger, C.Y. Mardin, R.P. Tornow, Wavelet denoising of multiframe optical coherence tomography data, Biomed. Opt. Express 3 (3) (2012) 572–589.
[17] S.U. Yang, X.U. Zhijie, Parallel implementation of wavelet-based image denoising on programmable PC-grade graphics hardware., Signal Process. 90 (8) (2010) 2396–2411.

[18] T. Christian, S. Javier, P. Manuel, P. Luis, T. Francisco, Parallel implementation of the 2d discrete wavelet transform on graphics processing units: filter bank versus lifting., in: IEEE Transactions on Parallel and Distributed Systems, vol. 19, 2008, pp. 299–310.
[19] C. Tomasi, R. Manduchi, Bilateral filtering for gray and color images, in: IEEE International Conference on Computer Vision, ICCV, 2008, pp. 839–846.
[20] S. Paris, F. Durand, A fast approximation of the bilateral filter using a signal processing approach, Int. J. Comput. Vis. 81 (2009) 24–52.
[21] B. Weiss, Fast median and bilateral filtering, ACM Trans. Graph. 25 (2006) 519–526.
[22] Q. Yang, Recursive approximation of the bilateral filter, IEEE Trans. Image Process. 24 (2015) 1919–1927.
[23] M. Maxime, C. François, flowing bilateral filter: definition and implementations, Image Anal. Stereol. 34 (2015) 101–110.
[24] M. Howison, Comparing GPU implementations of bilateral and anisotropic diffusion filters for 3d biomedical datasets, Report LBNL-3425E, Lawrence Berkeley National Laboratory, Berkeley, CA, USA, 2010.
[25] A.H. Fredj, J. Malek, E.B. Bourennane, Fast oriented anisotropic diffusion filter, 2016 11th International Design & Test Symposium (IDT), Hammamet (2016) 308–312.
[26] M. Ben Abdallah, J. Malek, A. Taher Azar, H. Belmabrouk, J. Esclarin Monreal, K. Krissian, Adaptive noise-reducing anisotropic diffusion, Neural Comput. Appl. 27 (2016) 1273–1300.
[27] J. Malek, A.T. Azar, B. Nasralli, M. Tekari, H. Kamoun, R. Tourki, Computational analysis of blood flow in the retinal arteries and veins using fundus image, Comput. Math Appl. 69 (2015) 101–116.
[28] H. Zhu, Y. Chen, J. Wu, J. Gu, K. Eguchi, Implementation of 3d SRAD algorithm on CUDA, in: International Conference on Intelligent Networks and Intelligent Systems, 2011, pp. 97–100.
[29] S. Aja-Fernández, C. Alberola-López, On the estimation of the coefficient of variation for anisotropic diffusion speckle filtering, IEEE Trans. Image Process. 15 (2006) 2694–2701.
[30] H. Jiang, J. Liu, K. Luo, W. Hu, X. Liu, Implementation of 3-d RDPAD algorithm on follicle images based CUDA, 11th International Conference, ICIC 2015 (2015) 177–184.
[31] A.H. Fredj, J. Malek, Real time ultrasound image denoising using NVIDIA CUDA, in: 2016 2nd International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), Monastir, 2016, pp. 136–140.
[32] M. Nieniewski, P. Zajczkowski, Real-time speckle reduction in ultrasound images by means of nonlinear coherent diffusion using GPU, in: ICCVG 2014, in: LNCS, vol. 8671, Springer, Heidelberg, 2014, pp. 462–469.
[33] I.A. Abdou, W. Pratt, Quantitative design and evaluation of enhancement/thresholding edge detectors, in: Proceedings of the IEEE., vol. 67, 2005, pp. 753–766.
[34] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, QImage quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (2004) 600–612.

Amira Hadj Fredj got her fundamental license and MS degree in Microelectronics from the Higher Institute of Informatics and Mathematics of Monastir, Tunisia, in 2011 and 2014, respectively. Currently, she is preparing her PhD degree in Electronics and Microelectronics in the Faculty of Sciences of Monastir. Her main research includes SOC implementation, parallel architecture and medical image processing.

Jihene Malek received her BS in Physics from Monastir University, in 1994, and her MS and PhD in Electronics and Microelectronics from the Faculty of Sciences of Monastir in 1996 and 2002, respectively. Since 2015, she has been a senior lecturer in Electronics and Microprocessors with the Electronics Department, The Higher Institute of Applied Sciences and Technology Sousse. Her current research interests include classification, classifiers combination, features extraction, high dimension classification problems, neural network fuzzy sets, biomedical engineering, medical image processing and visualization, 3D reconstruction, hardware implementation, and computational fluid dynamics.