GPU implementation of nonlinear anisotropic diffusion for medical image enhancement

Fernando Villalbazo, Juan J. Tapia, and Julio C. Rolón

CITEDI Research Center, Instituto Politécnico Nacional, Avenida del Parque 1310, Mesa de Otay, Tijuana, Baja California, México, 22510
fvillabazo@citedi.mx
{jtapiaa,jcrolon}@ipn.mx
Paper received on 11/29/13, Accepted on 01/19/14.

Abstract. In this work, the implementation of the diffusion process in a graphics processing unit (GPU) with application to medical image enhancement is presented. The method is implemented in the many-core architecture of the GPU and uses its resources of texture and shared memory. The diffusion equation is used to perform the enhancement of medical images. Due to the characteristics of the diffusion algorithm, it is possible to take advantage of the resources of the processor and reduce the image processing time.
A gain of up to 20 times in the execution time of the GPU implementation of anisotropic diffusion algorithm has been achieved, with respect to the implementation in a general purpose processor. An important noise reduction has also been accomplished by using the diffusion parameters found with the proposed fast search algorithm. A gain of up to three times in the GPU implementation of diffusion filtering execution time has been achieved using the fast search method instead of exhaustive search.

Keywords: GPU, Image enhancement, Nonlinear diffusion, Parallel algorithm

1 Introduction

In recent years, applications that combine general purpose processors with graphics processors have increased significantly, due to the high processing power of the GPU, their ability to perform massively parallel processing and their scaling performance that is achieved by increasing the number of graphics processors. For these reasons, a GPU is a suitable cost-effective solution to achieve high performance computing for general purpose applications. An important field of application is the medical image processing. The medical image applications are time consuming, because of the amount of data to be processed and the requirement of high spatial resolution [1].

Medical images contain certain amount of noise that is inherent to the acquisition process and affects the image quality. Noise reduction without loss of

Mireya García-Vázquez, Grigori Sidorov, Sunil Kumar (Eds.).
Advances in computing science, control and communications.
Research in Computing Science 69, 2014, pp. 131- 142

desired information in the image is an important challenge in the area of digital image processing. The diffusion process is a method that applies a selective smoothing filter and performs noise reduction inside the homogeneous regions. The noise is removed while the sharpness of the edges is preserved. The numerical algorithm is implemented with an explicit finite difference method, therefore there are not data dependencies and properly fits the architecture of GPUs, enabling the parallelization down to the pixel level.

The diffusion process has been widely used in image denoising and applied to several types of medical images [2–7]. In [8], a form to estimate the stopping time T for the diffusion is proposed. T is calculated by correlating the original and the enhanced images. In [9], a multigrid solver is proposed to solve the anisotropic diffusion equation and to estimate the diffusion parameters. However, the parameter localization through these methods is based on a estimation, not on the evaluation of the diffusion equation. The main objective of this work is the effective utilization of a GPU to reduce the processing time of the medical image enhancement through the diffusion process and the estimation of the diffusion parameters that ensure a minimal noise in the image, using a rapid search optimization method.

Section 2 summarizes the nonlinear diffusion method; section 3 describes the parallelization strategy we have followed; section 4 presents the experimental results for five synthetic MRI (Magnetic Resonance Imaging) images; finally, some conclusions and comments about future work are outlined.

2 Diffusion filtering

The diffusion process may be considered as a dispersion phenomena in which concentration of either mass, heat, or any other physical variable of interest moves from an area of high concentration to an area of low concentration, until the equilibrium is reached. The diffusion equation is

$$\frac{\partial u}{\partial t} = \nabla \cdot (\alpha \nabla u), \tag{1}$$

where $\nabla\cdot$ is the divergence operator and α is a diffusion parameter which defines the diffusion intensity. If α is constant in the medium, a linear diffusion is produced. If α is a function of some parameter of the medium, a nonlinear diffusion is produced.

The result of the diffusion when applied to images is a family $u(x, y, t)$ at different scales t, where

$$u(x, y, t+1) = G_\sigma(x, y) * u(x, y, t), \tag{2}$$

here $G_\sigma(x, y)$ is a Gaussian filter with variance σ [11].

In image processing, the diffusion is the process through which clusters of high-energy pixels within an image are dispersed, which results in the softening of the image. Since the process evolves in time, at $t = 0$, the original image

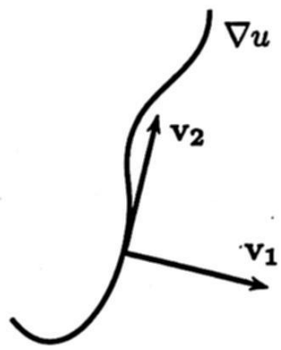

Fig. 1. Orientation of eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$

$u(x, y)$ is $u(x, y, 0) = u^0$. The Gaussian *scale* is $0 \leq t \leq T$, where T is the total number of scales of the process. Each new scale only depends on the image produced at the previous scale. The image at $u(x, y, t+1)$ is a smoother version of the image at $u(x, y, t)$. If $T >> 0$, all the image pixels will have the same gray level.

2.1 Nonlinear Diffusion

Introduced in [12], the nonlinear isotropic diffusion process makes use of Eq. (1) where the diffusion parameter α is defined as a function of the contour intensity. The contour estimator used is the image gradient ∇u.

In the nonlinear isotropic diffusion, the parameter α is a scalar value

$$\alpha(x, y, t) = g(||\nabla u(x, y, t)||), \tag{3}$$

where g is known as the diffusivity function. Conditions are imposed on g such that whenever the gradient is high, e.g. a contour in the image, diffusion is not applied; conversely, when gradient is low, e.g. an homogeneous region, the diffusion is maximized:

$$\lim_{||\nabla u|| \to \infty} g(||\nabla u||) \to 0, \quad \lim_{||\nabla u|| \to 0} g(||\nabla u||) \to 1. \tag{4}$$

In the nonlinear anisotropic diffusion, the smoothing depends on the gradient intensity and its direction [13]. To obtain information about the pixel neighbourhood, a structure tensor $\mathbf{J}$ is used [14], and it is calculated from the image gradient as

$$\mathbf{J}(\nabla u) = \nabla u \nabla u^T = \begin{bmatrix} J_{11} & J_{12} \\ J_{12} & J_{22} \end{bmatrix}. \tag{5}$$

The direction of the diffusion propagation is indicated by the eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$ of the structure tensor $\mathbf{J}$. The orientation of eigenvectors $\mathbf{v_1} \parallel \nabla u$ and $\mathbf{v_2} \perp \nabla u$ with respect to the image gradient is illustrated in Fig. 1. The diffusion parameter α from the Eq. (1) is defined as the diffusion tensor

$$\mathbf{D} = \begin{bmatrix} a & b \\ b & c \end{bmatrix} = \begin{bmatrix} \mathbf{v_1} & \mathbf{v_2} \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{bmatrix} \begin{bmatrix} \mathbf{v_1} \\ \mathbf{v_2} \end{bmatrix}, \tag{6}$$

where the parameters λ_1 and λ_2 define the diffusion intensity along the direction of the eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$, respectively.

The approach used in this work is the edge enhancing diffusion, which performs a controlled diffusion along the direction corresponding to the contour gradient, and a maximum diffusion in the direction normal to the contour gradient. The λ_1 parameter in the edge enhancing approach, is defined by a diffusivity function g. In the edge enhancing diffusion, the values of λ_1 and λ_2 are

$$\begin{aligned} \lambda_1 &= g(||\nabla u||) \\ \lambda_2 &= 1 \end{aligned} \tag{7}$$

The vector $\mathbf{v_1}$ can be represented as $\mathbf{v_1} = [\cos\theta, \sin\theta]^T$, where θ is defined as the angle of inclination of the image gradient [15]. The edge orientation can be computed as

$$\theta = arctan\left(\frac{2J_{12}}{J_{22} - J_{11}}\right) + \frac{\pi}{2}. \tag{8}$$

The vectors $\mathbf{v_1}$ and $\mathbf{v_2}$ are orthogonal, so the vector $\mathbf{v_2}$ can be computed as

$$\mathbf{v_2} = \left[-\sin\theta, \cos\theta\right]^T. \tag{9}$$

2.2 Discrete solution of the diffusion equation

The numerical implementation for the solution of the diffusion equation in Eq. (1) is obtained by finite differences centered in the space, and forward differences in the time discretization, which leads to an explicit scheme, where the image u^{t+1} is obtained from u^t with

$$u^{t+1} = u^t + \tau(\nabla \cdot (\alpha \nabla u)), \tag{10}$$

where τ is a parameter that keeps the numerical solution stable. In image processing, the parameter τ is chosen in the range of $0 < \tau < 1/4$. Each scale t represents one iteration of the numerical algorithm.

For the nonlinear anisotropic diffusion, the spatial derivative $\nabla \cdot (\mathbf{D}\nabla u)$ is approximated by the discretized expression $\mathbf{A}u^t$

$$\nabla \cdot (\mathbf{D}\nabla u) \approx \mathbf{A}u^t \tag{11}$$

The expression $\mathbf{A}u^t$ is represented by a mask that is function of the diffusion tensor $\mathbf{D}$ [14, 16]

$$\begin{aligned} A_{i,j}u_{i,j} = {} & \frac{c_{i,j+1} + c_{i,j}}{2}u_{i,j+1} - \frac{b_{i-1,j} + b_{i,j+1}}{4}u_{i-1,j+1} + \frac{b_{i+1,j} + b_{i,j+1}}{4}u_{i+1,j+1} \\ & - \frac{a_{i-1,j} + 2a_{i,j} + a_{i+1,j} + c_{i-1,j} + 2c_{i,j} + c_{i+1,j}}{2}u_{i,j} \\ & + \frac{a_{i-1,j} + a_{i,j}}{2}u_{i-1,j} + \frac{a_{i+1,j} + a_{i,j}}{2}u_{i+1,j} + \frac{c_{i,j-1} + c_{i,j}}{2}u_{i,j-1} \\ & - \frac{b_{i+1,j} + b_{i,j-1}}{4}u_{i+1,j-1}. \end{aligned} \tag{12}$$

Algorithm 1 Pseudocode for the anisotropic process

```
u = noisy image
for t = 1 to T do
  Diff = 0
  compute J
  compute λ1, set λ2
  compute θ
  compute v1, v2
  compute D
  for i = 1 to M do
    for j = 1 to N do
      compute A_{i,j}
      compute Diff_{i,j} = Diff_{i,j} + A_{i,j}
    end for
  end for
  u = u + τ · Diff
end for
```

The sequential iterative process to obtain the enhanced image u from the noisy image, is shown in the Algorithm 1. The image in the scale t is computed from the data of the previous scale $t-1$ and then overwritten over the same memory space, therefore only the images in t and $t-1$ have to been stored in the entire process. The images are updated in each scale.

3 Parallel implementation of the nonlinear diffusion algorithm

The execution time of the anisotropic diffusion algorithm can be reduced by through a parallel implementation on the GPU and the efficient handling of the different types of memories. Parallelization of the complete diffusion process algorithm is performed at the pixel level, exploiting the fine grain level parallelism allowed by the GPU architecture. The CUDA programming model has been used instead of other models to obtain the best performance in GPU applications [17].

A graphics processor consists of a set of multiprocessors, where each of these has several cores that compute the same operation on different data, according to the SPMD (Single Program, Multiple Data) model [18, 19]. The work is distributed equally between the multiprocessors, in order to avoid overload or lack of work on some multiprocessors.

The management of different types of GPU memories enables the possibility to improve the algorithm performance. The global memory may be addressed by all the active threads of the GPU, with the disadvantage that when many threads need to access memory at the same time, some latency may arise that limits the performance of the GPU. The texture, constant and shared memories are on-chip memories, consequently, the access time for these memories is lower than the global memory.

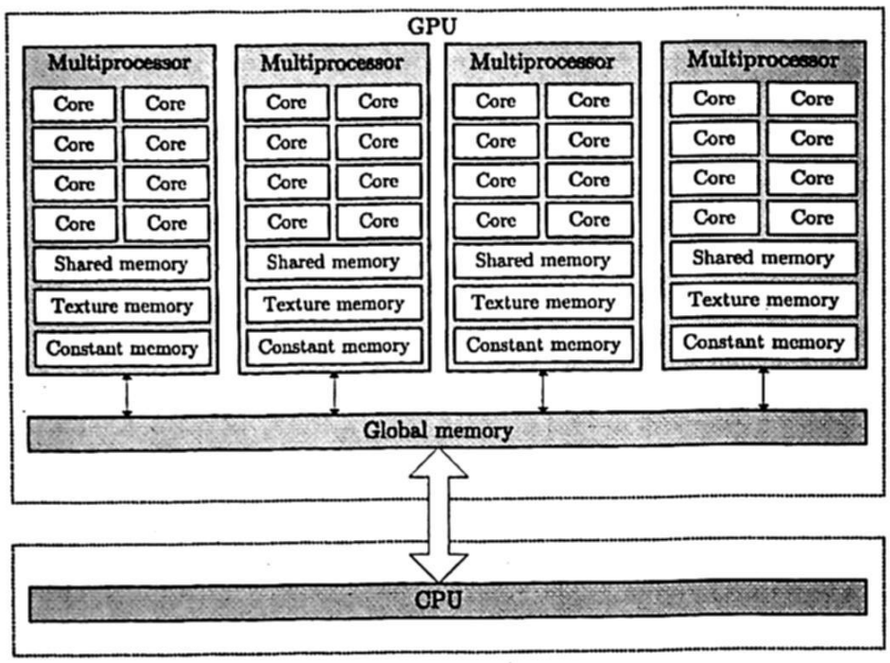

Fig. 2. Functional subdivision of the memory of a GPU device.

To transfer data from the CPU to the GPU on-chip memories, is necessary to transfer the data first to the GPU global memory and next to the corresponding memory locations. Texture memory is used to accelerate data reading from global memory, taking advantage of the two-dimensional image structure. Once written, the contents of these memories may only be transferred to the corresponding memory location of the actual processing unit, it is not possible for the processing unit to update them directly from the multi-thread cores. Only the host may instruct the GPU to transfer the contents of the global memory to the texture memory of the GPU. The GPU architecture and the distribution of the GPU memories into global, texture, constant and shared memories is shown in Fig. 2.

Synchronization points are added in the algorithm to prevent errors at the thread level. According to the explicit scheme, the image $u_{i,j}^{t+1}$ is obtained directly from $u_{i,j}^{t}$ and overwrites the image data on the global memory. In this process, image data are always kept inside the GPU and texture memory is used to store the images at time $u_{i,j}^{t+1}$. The gradient ∇u, the eigenvectors $\mathbf{v_1}$ and $\mathbf{v_2}$, the diffusion parameters λ_1, λ_2 and the tensors $\mathbf{J}, \mathbf{D}$ are calculated at the pixel level using the local memory of each thread.

The texture memory is read-only, consequently, the result of the nonlinear diffusion for each pixel, $u_{i,j}^{t+1}$ overwrites the value $u_{i,j}^{t}$ in the global memory. To begin a new iteration, the CPU instructs to the GPU to update the texture memory with the image data of the global memory.

Each function that is executed in parallel by the graphics processor is called kernel. The main program is executed in the CPU, and calls two kernel functions. Both kernels are two-dimensional, of size $[N/blocks, N/blocks]$, where *blocks* is the number of threads that can be executed by a block. Each thread computes

Algorithm 2 Pseudocode for the parallel anisotropic process

```
CPU memory allocation
GPU memory allocation
u = noisy image
MxN = image size
copy u from CPU to GPU
for t = 1 to T do
   kernel1 <<< (N/blocks, N/blocks), (blocks, blocks) >>> (u, D)
   update D in texture memory
   kernel2 <<< (N/blocks, N/blocks), (blocks, blocks) >>> (u, D)
   update u in texture memory
   copy u from GPU to CPU
   free CPU memory
   free GPU memory
end for
__global__ kernel1 (u, D)
{
  x = blockIdx.x * blockDim.x + threadIdx.x
  y = blockIdx.y * blockDim.y + threadIdx.y
  id = y * M + x
  compute J
  compute λ1, set λ2
  compute θ
  compute v1, v1
  compute D[id]
}
__global__ kernel2 (u, D)
{
  x = blockIdx.x * blockDim.x + threadIdx.x
  y = blockIdx.y * blockDim.y + threadIdx.y
  id = y * M + x
  compute Diff
  compute new u[id]
}
```

its identifier id, from their position in the kernel. The first kernel computes $\mathbf{J}$, λ_1, λ_2, θ, $\mathbf{v_1}$ and $\mathbf{v_2}$.

Each of these terms are stored in the local memory of the GPU. The diffusion tensor $\mathbf{D}$ is calculated and stored in the global memory of the GPU, that is linked to the texture memory. The CPU must indicate to the GPU to update the diffusion tensors in texture memory from the global memory. The second kernel function calculates the amount of diffusion $Diff$ by reading the diffusion tensors and the image in the previous iteration from the texture memory. The new image is stored in global memory and then updated by the CPU for the next iteration. The parallel implementation of the anisotropic diffusion process is summarized in the Algorithm 2.

4 Experimental Results

The parallel algorithm for the nonlinear anisotropic diffusion was tested in a GPU with 336 cores (NVIDIA GeForce 460). All the calculations were made with single precision floating point arithmetic. Two implementations of the nonlinear anisotropic diffusion have been compared, through the computation of Eq. (10). The first implementation makes use of the global memory of the GPU, and the second uses the texture memory. In both cases, the execution time was measured using the GPU timer. The time measured includes the execution of the CPU and the GPU functions for the diffusion computation until the diffusion stopping time T is reached, it also includes the time spent in data transfers between both architectures, the texture memory allocation and memory updates. The experiment was conducted over 181 synthetic MRI images of 181×217 pixels with 8 bits per pixel [20]. Noise added into the process of image capture is modelled by a Gaussian distribution with zero mean and variance $\sigma = 0.002$.

The diffusivity function g used in the experiment is [12]

$$g(\nabla u) = \frac{1}{1 + \left(\frac{\|\nabla u\|}{K}\right)^2}, \tag{13}$$

where K is a contrast parameter which controls the amount of diffusion the function exerts.

Table 1. Average execution time in msec for the nonlinear anisotropic diffusion algorithm.

t	1	2	3	4	5
T_{CPU}	29.20	42.20	55.80	64.90	77.70
T_{global}	0.84	0.95	1.15	1.38	1.59
T_{tex}	0.36	0.53	0.71	0.89	1.07
T_{global}/T_{tex}	2.35	1.77	1.63	1.56	1.49
T_{CPU}/T_{tex}	20.33	19.79	19.73	18.33	18.24

The average execution time of the algorithm implemented in the CPU and in the GPU is shown in Table 1. The term T_{CPU} identifies the execution time in a CPU, T_{global} denotes the global memory approach in the GPU, and T_{tex} identifies the case of texture memory in the GPU. A total of $t = 5$ Gaussian scales were used. If more scales are used the error grows rapidly and the image becomes indistinguishable. A gain of up to 20 times in the algorithm execution time was achieved using the GPU architecture with the memory optimization, compared against the CPU implementation. In the GPU, the implementation with the memory optimization is 1.6 times faster than the implementation using global memory. A more detailed comparison between the CPU and the GPU is presented in [10], they use as metrics: execution time, occupancy and FLOPS.

The evolution of the nonlinear anisotropic diffusion process on an image for different scales is shown in Fig. 3, using the edge enhancing approach. It

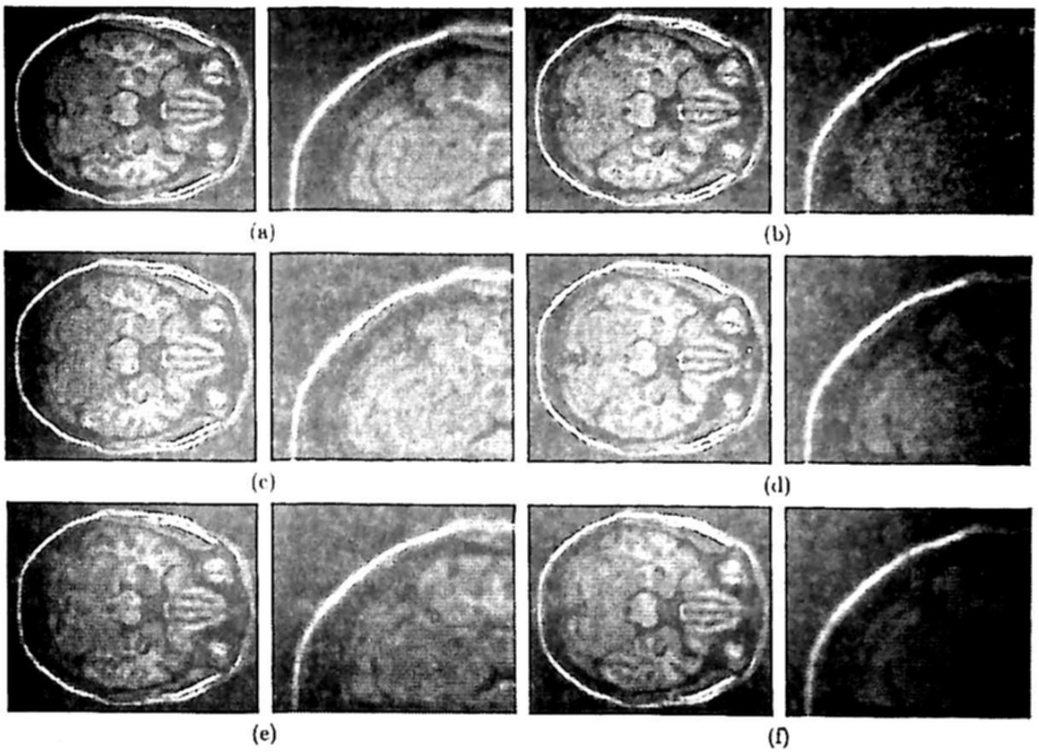

Fig. 3. Nonlinear anisotropic diffusion process evolution. (a) Noiseless image, (b) Noisy image. Diffusion (c) $t = 2$, (d) $t = 5$, (e) $t = 10$, (f) $t = 20$

is possible to see the smoothing effect of the diffusion filter and the gradual decreasing of noise in the images. If $T >> 0$, an undesirable effect is generated in the image edges, because of the extreme diffusion that is applied to those areas. The diffusion parameters are $\tau = 0.25$ and $K = 20$.

The quality of the enhanced image is measured by the mean squared error (MSE), which is given by

$$MSE = \frac{1}{M \times N} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} (u_{i,j}^0 - u_{i,j}^t)^2, \tag{14}$$

where $M \times N$ is the image size, u^0 and u^t are the original and enhanced images. The MSE error behavior of the anisotropic diffusion for values of the contrast parameter K, from $K = 0$ to 100 was calculated for the image of Fig. 3. The lowest MSE obtained is achieved with $K = 43$ and $t = 4$, which is $MSE = 27.33$. The error is reduced in the edges, improving the visualization of the image. The skull area keeps details through the diffusion scales, but the area inside the brain is smoothed after several diffusion scales. The test image is shown in Fig. 3b. We have found experimentally that the MSE error always has only one minimum. The error measurement is performed to optimize the diffusion parameters that produce the lowest error. The MSE calculation is performed by reading the pixel values from the texture memory and saving partial summations on GPU shared memory, combined with a reduction algorithm [19, 21]. The operation $u^0 - t^t$ is computed in parallel by the GPU and the summations of the difference are

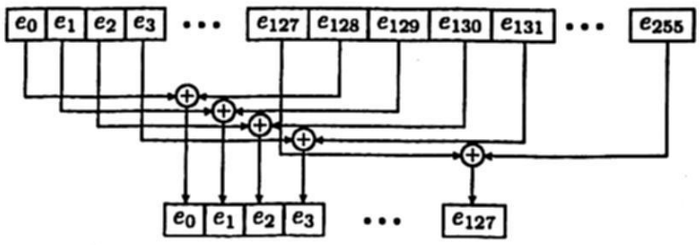

Fig. 4. First step of the reduction algorithm

stored in the shared memory of the corresponding multiprocessor. When all the summations are stored, these are used to compute the total error, which finally is transferred to the CPU. The summation is performed in parallel through the reduction algorithm, which allows to calculate the sum of the error with a reduced number of iterations. The error array is split in 2, and the elements i and $i + N/2$ are summed, where i is the element position, and N is the size array. When the calculation of a partial sum is completed, the array is split in 2 again and the summation is performed again. The algorithm continues until there is just one element, with the value of the sum of all elements. The first step of the reduction algorithm, with an array of 256 elements is shown in Fig. 4.

4.1 Fast search method to optimize the diffusion parameters

To optimize the diffusion parameters, an exhaustive search of the whole space should provide the answer. However, this method is computationally intensive. In order to reduce the computational load generated by the exhaustive search, a fast search method is implemented, which is a variant of the three-step search method [22]. The algorithm consists in dividing the search space in 4 regions of the same size. The parameters that correspond to the locations of the centroids of each region are evaluated. The region with the lower error is chosen to split in the next iteration. The algorithm continues until a certain number of iterations is reached. This method has the advantage of reducing the computational cost of the search method, when the search space is very large. However, as with any optimization method that approximates the solution, the parameters not always match the real minimum with the approximate one.

Table 2. Minimum MSE found by the exhaustive and fast search algorithm.

Image	MSE_{ES}	MSE_{FS}	MSE_{dif}	$(K,t)_{ES}$	$(K,t)_{FS}$
10	25.9229	25.9364	0.0134	(43,4)	(45,5)
40	27.3342	27.3367	0.0025	(43,4)	(41,5)
70	28.1007	28.1033	0.0026	(43,4)	(41,5)
100	27.0664	27.1540	0.0198	(43,4)	(41,5)
130	26.8103	26.9351	0.1247	(41,4)	(39,5)

The fast search method was applied to five MRI images, T1 weighted. The image modality was chosen due to its high contrast. The minima MSE calculated using the exhaustive and fast search method are defined by the MSE_{ES} and the MSE_{FS}, respectively. The difference of the MSE between both methods is $MSE_{dif} < 1$, which does not represent a significant difference in the resulting images.

Table 2 shows the diffusion parameters and the error (MSE) that results from applying the exhaustive and fast search method on a search space $1 \leq K \leq 100$ and $1 \leq t \leq 30$. Furthermore, the diffusion parameters (K, t) localized with both methods, for comparison.

Average execution time of exhaustive search algorithm T_{ES} and fast search algorithm T_{FS} is: $T_{ES} = 113.13$ msec. and $T_{FS} = 31.19$ msec, for $1 \leq K \leq 100$ and $1 \leq t \leq 30$ for 181 MRI images. Processing the complete set of 181 images and calculating the error by using the fast search algorithm, a gain $G = 3.62$ of the execution time was achieved in the algorithm. For the search interval proposed in the experiment, using the exhaustive search needs to perform 3000 iterations of the algorithm, whereas with the fast search algorithm with 140 iterations the minimum is reached.

5 Conclusions

It has been demonstrated that the use of the GPU in-chip memories significantly decreases the processing time of anisotropic diffusion algorithm and error computation, because the algorithm is naturally adapted to the GPU architecture. Texture memory aim to fast read pixel values and shared memory is useful to fast error calculation, through the parallel reduction algorithm. In the case of medical image datasets, this reduction may lead to an improvement in the global performance of the system. The error computation is performed totally in the GPU, in a way to minimize the data transference between both architectures and to improve the performance of the implementation.

A fast search of the parameters that minimize the MSE in the enhancement process was implemented. The results of the MSE obtained with the fast search algorithm are very close to those obtained with the exhaustive search algorithm. The differences are negligible.

6 Acknowledgments

This work has been partially supported by COFAA-IPN, and by grants IPN-SIP-20120606 and IPN-SIP-20130489.

References

1. Pratx, G., Xing, L.: GPU computing in medical physics: A review. Med. Phys. **38**(5) (2011) 2685–2697

2. Fernandez, J.J., Li, S.: Anisotropic Nonlinear Filtering of Cellular Structures in Cryoelectron Tomography. Comput. Sci. Eng. **7**(5) (2005) 54–61
3. Gerig, G., Kubler, O., Kikinis, R., Jolesz, F.A.: Nonlinear Anisotropic Filtering of MRI Data. IEEE Trans. Med. Imag. **11**(2) (June 1992) 221–232
4. Harry, M.S., Fernández, D.C.: Comparison of PDE-Based Nonlinear Diffusion Approaches for Image Enhancement and Denoising in Optical Coherence Tomography. IEEE Trans. Med. Imag. **26**(6) (june 2007) 761–771
5. Guo, S., Xiaoming, W., Renjing, C., Jing, Z.: An approach to suppress speckle noise and enhance edge. J. Electron. **23**(2) (2006) 225–230
6. Xuejun, S., Land, W., Samala, R.: Deblurring of Tomosynthesis Images Using 3D Anisotropic Diffusion Filters. In: Proc. SPIE. Volume 6512., San Diego, USA (2007)
7. Sun, Q., Hossack, J.A., Tang, J., Acton, S.T.: Speckle reducing anisotropic diffusion for 3D ultrasound images. Comput. Med. Imaging Graphics **28** (2004) 461–470
8. Mrazek, P., Navara, M.: Selection of optimal stopping time for nonlinear diffusion filtering. In: Int. J. Comp. Vision, Netherlands (2003) 189–203
9. Chen, D., MacLachlan, S., Kilmer, M.: Iterative parameter-choice and multigrid methods for anisotropic diffusion denoising. SIAM J. Sci. Comput. **33**(5) (October 2011) 2972–2994
10. Alvarado, R., Tapia, J.J., Rolón, J.C.: Medical image segmentation with deformable models on graphics processing units. J. Supercomput. DOI 10.1007/s11227-013-1042-4 (Published online: 17 december 2013)
11. Black, M.J., Sapiro, G., Marimont, D., Heeger, D.: Robust Anisotropic Diffusion. IEEE Trans. Image Proc. **7**(3) (March 1998) 421–424
12. Perona, P., Malik, J.: Scale-Space and Edge Detection Using Anisotropic Diffusion. IEEE Trans. Pattern Anal. Mach. Intell. **12**(7) (July 1990) 629–639
13. Weickert, J., Hagen, H.: Visualization and Processing of Tensor Fields. Springer (2006)
14. Weickert, J.: Anisotropic Diffusion in Image Processing. Teubner-Verlag (1998)
15. Terebes, R., Borda, M., Germain, C., Lavialle, O., Pop, S.: Asymmetric Directional Diffusion Based Image Filtering and Enhancement. In: Proc. of the IEEE Intl. Conf. on Automation, Quality and Testing, Robotics. Volume 3., Washington, DC, USA (2008) 413–418
16. Weickert, J., Zuiderveld, K., ter Haar Romeny, B., Niessen, W.: Parallel Implementations of AOS Schemes: A Fast Way of Nonlinear Diffusion Filtering. In: Proc. of IEEE International Conference on Image Processing, Santa Barbara, CA (Oct 1997) 396–399
17. Malik, M., et al.: Productivity of GPUs under different programming paradigms. Concurrency and Computation: Practice and Experience **24**(2) (2012) 179–191
18. Kirk, D.B., Hwu, W.m.W.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann (February 2010)
19. Sanders, J. and Kandrot, E.: CUDA by Example: An Introduction to General-Purpose GPU Programming. Addison-Wesley (July 2010)
20. Cocosco, C.A., Kollokian, V., Kwan, R.K.S., Pike, G.B., Evans, A.C.: BrainWeb: Online Interface to a 3D MRI Simulated Brain Database. In: Third Int. Conf. on Functional Mapping of the Human Brain, Copenhagen, Denmark (May 1997) 425
21. Sanderson, A.R., Meyer, M.D., Kirby, R.M., Johnson, C.R.: A framework for exploring numerical solutions of advection-reaction-diffusion equations using a GPU-based approach. Comput. Vis. Sci. **12**(4) (March 2009) 155–170
22. Mitchell, J.L., Pennebaker, W.B., Fogg, C.E., Legall, D.J.: MPEG Video Compression Standard. Chapman & Hall (1996)