VOL. 12, NO. 21, NOVEMBER 2017 ISSN 1819-6608
ARPN Journal of Engineering and Applied Sciences

www.arpnjournals.com

MODELLING THE BEHAVIOR OF THE CPU AND THE GPU VERSUS THE CLUSTERS NUMBER VARIATION FOR SEQUENTIAL AND PARALLEL IMPLEMENTATIONS OF BCFCM ALGORITHM

Noureddine Ait Ali[1], Bouchaib Cherradi[1, 2], Ahmed El Abbassi[3], Omar Bouattane[1] and Mohamed Youssfi[1]
[1]SSDIA laboratory, ENSET-Mohammedia, Hassan University Casablanca (UH2C), Morocco
[2]STICE Team, CRMEF-El Jadida and LaROSERI laboratory, FS-El Jadida, Chouaib Doukkali University (UCD), Morocco
[3]EPIM Team, FST-Errachidia, My Ismail University (MIU), Morocco
E-Mail:aitali_noureddine@yahoo.fr

ABSTRACT

Image segmentation plays a crucial role in the medical imaging analysis to diagnosis diseases. The FCM algorithm is a widely used technique in this field and still being under improvement by researchers either at accuracy or at time execution. BCFCM (bias field correction FCM) is a robust variant of FCM that segment and corrects the intensity inhomogeneity artifact on medical images. However the algorithm is always a time consuming problem because of the powerful treatment requirement. GPU-based parallelism is one of the used solutions to enhance his efficiency in terms of execution time. In this paper we have studied and modelled the behavior of CPU and GPU hardware against the sequential BCFCM and the parallel PBCFCM implementations. The modelled results for i7 3.5 Ghz CPU and GTX 760 GPU considering the clusters number variation show interesting behaviors and are in good accordance to experimental ones.

Keywords: BCFCM algorithm, image segmentation, parallel computing, GPU, modelling.

1. INTRODUCTION

Image segmentation is an essential preprocessing step in computer vision and medical image processing [1] that helps doctors for better visibility of brain diseases. The main objective of image segmentation technique is to split the concerned image (pathologic or normal) automatically or semi-automatically into various regions, like each region is homogeneous with respect to some characteristics [2]. Many studies have been done on medical image segmentation and many efforts have been made in the literature to propose effective and efficient segmentation methods [3]. The authors in [4] gave a survey about the frequently used MRI images segmentation algorithms with an emphasis on their characteristics, with detail study of advantages and disadvantages of these techniques.

A bias field is an artefact presented as the form of a low frequency signal that corrupts magnetic resonance images because of the inherent in homogeneities caused by the MRI machine. In fact, it exist two approaches that deal with bias field artefact. The first one can be used as a pre-processing step where the corrupted MRI image is restored by splitting it by an estimated bias field signal using a surface fitting approach [5]. The second approach propose to modify an algorithm as the fuzzy c-means algorithm [6] to be used to correct and segment an MRI image corrupted by a bias field signal [7, 8].

The authors in [9] presented a novel fuzzy c-means algorithm (RCLFCM) for segmentation and bias field correction of brain MR images. They have used a new gray-difference coefficient and design a new impact factor to measure the effect of neighbour pixels; so that the robustness of anti-noise can be enhanced. *Jeetashree Aparajeeta* [10] proposed algorithms that have successfully been tested with synthetic data with bias field of low and high spatial frequency. The images used were taken from Brain web database.

Nowadays, with the appearance of the new technology as the graphical processing unit used for general purpose computing (GPGPU) has encourage many researchers to exploit their performance to accelerate many algorithms of many fields including medical imaging [11-16], in order to make it more efficient. In this context, *Tian et al* [17] improved the sequential hard c-means [18] algorithm initial cluster centres in order to minimize the iterative procedures number; in addition they have proposed a parallel version of this algorithm on Single Instruction Multiple Data computational model to reduce its complexity. In the same context, the authors in [19] proposed BCEFCM to segment the image and correct the bias field simultaneously; the proposed framework is not only able to deal with noise and correct the bias field but it is also faster and more accurate than state-of-the-art methods.

In the goal to limit the execution time constraint, the authors in [20, 21] proposed a parallel implementation of bias field correction fuzzy c-means [8] on GPU architecture, this algorithm was tested using three different GPUs devices. The speedups reached depend on the devices performance. For GTX 580 they have attempt about 52x over the sequential implementation, 21x on GTX 760 and 12x on GT 740 for big images size. In the same context, the authors in [22] proposed parallel version of spatial fuzzy c-means SFCM clustering algorithm [23] that is robust variant of FCM against noise artefact by including neighbour spatial function in the process of membership matrix updating. In addition they have proposed mathematical models characterizing the behaviour of the hardware on which implemented their algorithms [24].

More recently, the authors in [25] have proposed and compared two parallel implementations of BCFCM algorithm. The first implementation tries to exploit the performance of the GPU and avoid the data transfer latency, but the summations operations that need high synchronization of threads were done on GPU. The second method exploit the performance of the GPU to compute the parallel intensive portions and CPU to compute the low ones, however the data transfer is not avoided. The speedup attempted is about 12x faster than the serial version using GT 740m device.

In this paper we propose mathematical models that characterize the behaviour of CPU and GPU versus the variations of clusters number variable in two implementation versions of BCFCM algorithm. The sequential implementation was tested on i7 3.5 Ghz CPU while the parallel one was tested using GTX 760 GPU that has 1152 cores. The modelling results show that the speedup obtained is related to the clusters number by fixing the size of the image, more the clusters number is high more speedup is height.

The rest of this paper is organized as follows. In section 2, we will give a review of GPU Computing architecture and CUDA programming model (NVidia's GPU). Section 3 presents a review of the sequential version entitled bias field correction fuzzy c-means clustering algorithm BCFCM. In section 4, we present the stages of our parallel version PBCFCM. Section 5 present and discus our findings and results. Section 6, concludes the paper and gives some perspectives for this work.

2. GPU COMPUTING AND CUDA PROGRAMMING

Single instruction, multiple data (SIMD) architecture is one of the computing models that are used to accomplish processing and data parallelism. Indeed, in this architecture, multiple processors execute the same instructions on different data. Graphical processing Units (GPUs) that are components initially dedicated to image rendering with computing the values of each pixel to display on the screen, uses this type of architecture. Methods and algorithms for image processing are ideal candidates to massively parallel computation in a SIMD architecture because the independency between images pixels. In this paper we have choose, for our parallel implementation, one of the most used physical SIMD architectures that is the NVidia's GPU.

The NVidia's GPU architecture contains a great number of elementary processors that are composed of a large number of cores called streaming processors (SP) clustered into multiprocessor (MP) units. Each SP involves an arithmetic logical unit holding up integer and floating point operations. To adapt the GPUs to the processing, we need to write parallel functions called kernels with Compute Unified Device Architecture CUDA SDK [26].

The Figure-1 explains the Nvidia GPU architecture and the interactions possibilities with CPU (Host).

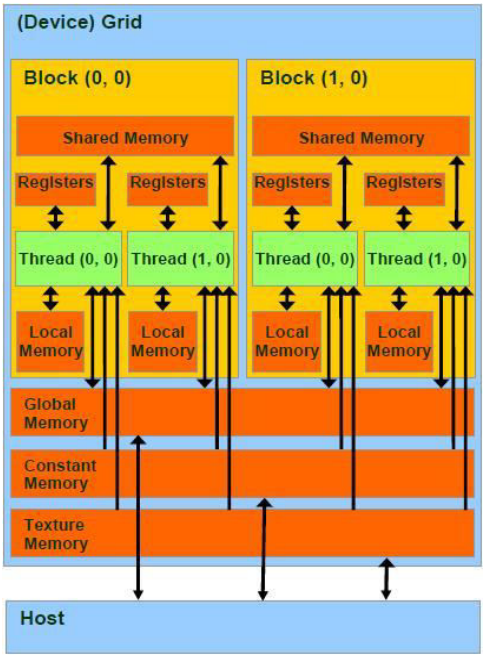


Figure-1. Typical NVidia's GPU architecture and the possible interactions with the Host (CPU).

NVidia developed CUDA (Compute Unified Development Architecture) that is a C library extension to provide a programming interface for users of his GPU devices. The main program is managed by the CPU (host) that is responsible for starting the program and executing serial code, while delegating parallel execution of compute-intensive tasks to the GPU device. To have a massively parallel version of a given algorithm with CUDA programming, we need to define C functions (kernels), which are executed in parallel by multiple GPU threads.

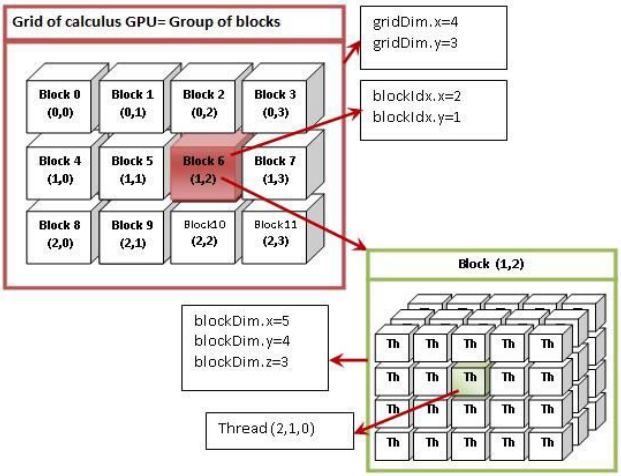


Figure-2. Execution model of a CUDA program on NVidia's GPU: Hierarchy grid, blocks and thread.

The CUDA program execution model is based on the fact that all threads run the same kernel concurrently, and each one is associated with a unique thread ID. Threads are arranged into three-dimensional thread blocks. Threads belonging to the same block cooperate by sharing

data through a very interesting component that is shared memory and by synchronizing their execution through extremely fast barriers. In contrast, threads belonging to different blocks cannot perform barrier synchronizations with each other. The Figure-2 shows an execution model of a CUDA program based on the hierarchy grids, blocks and threads.

3. SEQUENTIAL BIAS FIELD CORRECTION FUZZY C-MEANS ALGORITHM

The standard FCM [6] objective function for partitioning an image containing $x_k\{k=1,...,N\}$ pixels into C clusters is given by:

$$J_{FCM} = \sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p}\|x_k - v_i\|^2 \quad (1)$$

u_{ik}: The degree of membership of pixel x_k in the cluster V_i,
V_i: The prototypes (or centre) of the cluster i,
N: The total number of pixels in the image
p: A weighting exponent parameter (p>1) on each fuzzy membership value, it determines the amount of fuzziness of the resulting classification.

The main step of this iterative algorithm is to update the membership matrix to determine in which cluster the pixel belongs by the equation:

$$u_{ik} = \frac{1}{\sum_{j=1}^{C}\left(\frac{\|v_i - x_k\|}{\|v_j - x_k\|}\right)^{2/(p-1)}} \quad (2)$$

The membership matrix is initialized and updated, at every iteration, according to the following conditions:

$$U\left\{u_{ik} \in [0,1]\ , \sum_{i=1}^{C} u_{ik} = 1\ \ \forall k\ \ , 0 < \sum_{k=1}^{N} u_{ik} < N,\ \ \forall i\right\} \quad (3)$$

In most works in the literature, the observed MRI signal is modelled as a product of the true signal generated by the underlying anatomy, and a spatially varying factor called the gain field.

$$Y_k = X_k G_k \quad (4)$$

Where X_k and Y_k are the true and observed intensities at the k^{th} pixel, respectively, G_k is the gain field at the k^{th} pixel. The application of a logarithmic transformation to the intensities allows the artefact to be modelled as an additive bias field.

$$y_k = x_k + \beta_k \quad (5)$$

Where x_k and y_k are the true and observed log-transformed intensities at the k^{th} pixel, respectively, and β_k is the bias field at the k^{th} pixel.

Ahmed *et al* [8] proposed a modification to (1) by introducing a term that allows the labelling of a pixel to be influenced by the labels in its immediate neighbourhood. The modified objective function is given by:

$$J = J_{BCFCM} = \sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p}\|y_k - \beta_k - v_i\|^2 + \frac{\alpha}{N_r}\sum_{i=1}^{C}\sum_{k=1}^{N} u_{ik}^{p}\left(\sum_{y_k \epsilon N_k}\|y_r - \beta_r - v_i\|^2\right) \quad (6)$$

Where:

N_k: Set of neighbour's pixels that exist in a window around x_k.
N_r: Cardinal of N_k.
α: Neighbours effect parameter

The new membership function is then given by:

$$u_{ik} = \frac{1}{\sum_{j=1}^{C}\left(\frac{D_{ik} + \frac{\alpha}{N_r}\gamma_i}{D_{jk} + \frac{\alpha}{N_r}\gamma_j}\right)^{1/(p-1)}} \quad (7)$$

Where:

$$\gamma_i = \left(\sum_{y_r \in N_k}\|y_r - \beta_r - v_i\|^2\right) \quad (8)$$

And

$$D_{ik} = \|y_k - \beta_k - v_i\|^2 \quad (9)$$

The cluster prototype (centroid) updating is done by the expression:

$$V_i = \frac{\sum_{k=1}^{N} u_{ik}^{p}\left((y_k - \beta_k) + \frac{\alpha}{N_r}\sum_{y_r \in N_k}(y_r - \beta_r)\right)}{(1+\alpha)\sum_{k=1}^{N} u_{ik}^{p}} \quad (10)$$

The estimated bias field is given by the expression:

$$\beta_k = y_k - \frac{\sum_{i=1}^{C} u_{ik}^{p} v_i}{\sum_{i=1}^{C} u_{ik}^{p}} \quad (11)$$

In the following, we present the main steps of this algorithm.

Algorithm 1: Sequential Bias field Correction fuzzy C-Means Algorithm (BCFCM)

1: Set the parameters *C, p,Nr* and α
2: Randomly Initialize the membership matrix $U^{(0)}$ according to Eq.(2)
3. Choose the stopping criterion: $\varepsilon$and initialize the loop counter $n=0$
4: Set the initial centroïds vector $V^{(0)}$ and the initial estimated bias field $\beta^{(0)}$
5: Calculate the initial objective function $J_{(0)}$ using Eq.(1)
6: Repeat
  7: Set the loop counter $n=n+1$
  8: Update the membership matrix $U^{(n+1)}$ using Eq. (7)
  9: Update the cluster center vector $V^{(n+1)}$ using Eq. (10)
  10: Update the bias field estimated matrix $\beta^{(n+1)}$ using Eq. (11)
  11:Update the objective function $J_{(n+1)}$ using equation (6)
12: Until $||J_{(n+1)}-J_{(n)}|| < \varepsilon$.
13: Do the image segmentation using the final membership matrix U, cluster center vector V and the image pixels $(X_1,...,X_N)$, by giving to the pixel x_k the i^{th} label of the cluster Vi which has the greatest membership value max (u_{ik}: k = 1, ..., C).

4. PARALLEL VERSION OF BIAS FIELD CORRECTION FUZZY C-MEANS ALGORITHM PBCFCM

In our previous work [20, 21], we have already proposed a parallel version of this algorithm on 3 Nvidia GPU and obtained interesting results on windows 7 (32 bits) platform. In this work, we propose an enhanced parallel implementation by using CUDA SDK on visual studio 2013 with windows 7 (64-bits) platform, we have implemented this algorithm on massively parallel architecture that is a graphical processing unit, widely used actually in GPGPU.

Note that our strategy is based on the principle that the stages putting negligible execution time are executed in Host (CPU). This strategy consists on the execution of the potions that need more execution time on GPU as CUDA kernels and the rest of the code is executed on CPU that manages the global application code.

The strengths of our parallel implementation for PBCFCM algorithm are:

a) The exploitation of the shared memory for data to be clustered and constant memory for centroids.

b) The use of local thread registers and new functionalities of CUDA SDK. This gave rise to more interesting results in terms of speed up as we will explain in the following section.

The following algorithm gives the details of our parallel PBCFCM implementation.

Algorithm 2: Parallel Bias field Correction fuzzy C-Means Algorithm Algorithm (PBCFCM)

1. Set the parameters *C*, *p*, *Nr* and α
2. Randomly Initialize the membership matrix $U^{(0)}$
3. Choose the stopping criterion: ε and set the loop counter $n=0$
4. Set the initial centroïds vector $V^{(0)}$ and the initial estimated bias field $\beta^{(0)}$
5.Transfer parameters, $V^{(0)}$, $\beta^{(0)}$ and image Pixels to GPU
6.Repeat
  7. Set the loop counter $n=n+1$
  8. Update the membership matrix $U^{(n+1)}$ on GPU. (Eq.4)
  9. Compute the numerator and denominator of cluster centers (Eq.7) on GPU then transfer the results to from GPU to CPU
10. Update the cluster centers vector $V^{(n+1)}$ on CPU
  11. transfer the new vector $V^{(n+1)}$ from to GPU
  12. Update the bias field estimated matrix $\beta^{(n+1)}$ on GPU. (Eq.8)
13.Until $\|V_{(n+1)}-V_{(n)}\| < \varepsilon$.
14. Do the image segmentation using the final membership matrix U, cluster centres vector V and the image pixels $(X_1,...,X_N)$, by giving to the pixel x_k the i^{th} label of the cluster Vi which has the greatest membership value max ($u_{ik:}$ k = 1, ..., C).

5. RESULTS AND DISCUSSIONS

Before presenting the main results of this work, we present in the following tow sub-sections, the hardware and software specifications in addition to the used database for validation and experiments.

5.1. Images database

Since our main objective in this study is the evaluation and modelling of the behaviour of the used devices (CPU and GPU) against our implementations parallel and sequential, we have chosen to build a database

of images from a test image often used as a reference in the field of digital image processing that is Lena image.

To evaluate the behaviour of the GPU device when executing PBCFCM algorithm against cluster number variation with different image sizes, we construct a bank of Lena images but with different densities that varies from 1024 pixels to about 10.2 million pixels (Mpixels).

Our experiments are done on the following images: 128x128 (img128), 512x512 (img512), 1024x1024 (img1024), 2048x2048 (img2048), 2708x2704 (img2708), 3000x3000 (img3000) and 3200x3200 (img3200).

Note that the effectiveness and accuracy of the proposed PBCFCM algorithm has been widely validated on T1-weighted and T2-weighted MRI image with different densities and values of additional bias field.

5.2. Software and Hardware specifications

Algorithm on CPU (sequential version) was implemented using Microsoft VC++ Toolkit and executed on Intel(R) Core(TM) i7-4770 8 cores 3.5GHz CPU to obtain reference runtimes.

Parallel portions of PBCFCM algorithm were implemented using CUDA SDK 6.5 and executed on GTX 760 GPU device, execution times results were carried out to give comparison with the equivalent portions in sequential version. Table-1 summarize the principal specifications of the used devices. Both sequential and parallel codes are compiled within Microsoft Visual Studio 2013 under Windows 7 (64-bit) operating system.

Table-1. CPU and GPU devices specifications.

Device	Property	Value
CPU	Processor	Intel Core i7-4770K
	Clock speed	3.5 GHz
	No. of Cores	4
	No. of Threads	8
	RAM	16 GB
	Operating system	Windows 7, 64 bits
GPU	Chipset	GeForce GTX 760
	Processor clock	1033 Mhz (GK104)
	Cudacores	1152
	Total MP	6
	Max Thread per Block	1024
	Shared Memory	64 KB
	Global Memory	2048 MB
	Memory bus width	256 bits
	Memory Bandwidth	192.2 GB/sec

5.3. Notations and definitions

In this paper, we will focus on the number of cluster that we want to extract from images (Cluster number variable), that we note *c*.

In the goal to have magnitudes that can perfectly reflect the behaviour of the experimented devices against the studied iterative clustering algorithms BCFCM and PBCFCM, evaluate and compare the performances of the used devices (CPU and GPU) against the execution of these algorithms, we define and consider the 2 following magnitudes:

The first one postpones the execution time reported to a single iteration, we call it "Execution time per iteration in seconds" and we note it *ETPI(s)*. This magnitude will allow us to make an evaluation of the algorithms by disregarding the number of iterations needed for convergence that depends on the initial conditions and the size of the images. Indeed, the convergence of the algorithm whatsoever for sequential or massively parallel implementation depend of these initial conditions.

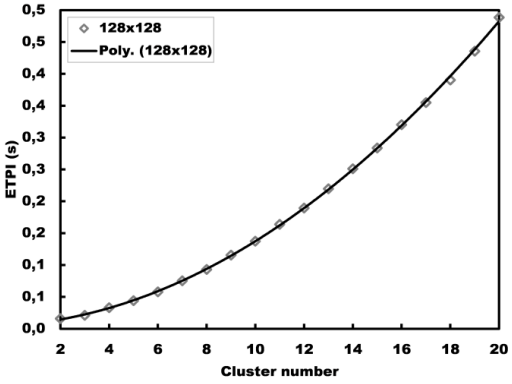
The second magnitude frequently used to evaluate the quality of a parallel implementation of an image processing algorithms compared to its sequential one is the ratio between the total execution time required for the convergence on CPU and the total execution time needed for convergence on GPU taking into account the same initial conditions. This ratio is called speed up GPU/CPU(x) and noted *SU(x)*.

5.4. Execution time per iteration variation modelling

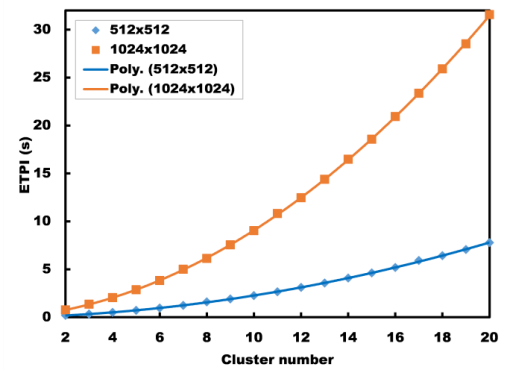
In this subsection, we present some interesting results and finding relative to one of the main ideas of this paper that is mathematically modelling of the variation of the *ETPI (s)* magnitude, a function of the variation of cluster number for both implementations (sequential and parallel). This will tell us about the behaviour of the used devices overlooked this variable.

5.4.1. CPU Execution time per iteration behaviour modelling

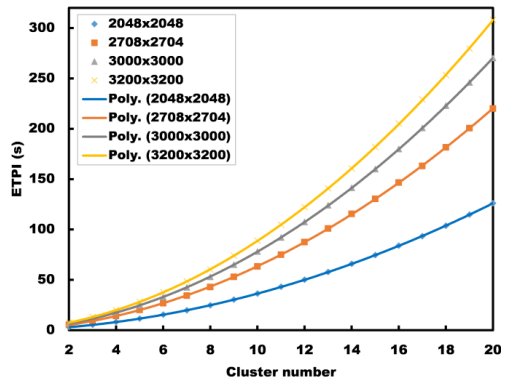
In figure 3we present the variation of execution time per iteration in seconds ($ETPI_{CPU/imgX(s)}$) for the sequential version of the studied algorithm with respect to the variation of cluster number *c*.In this figure we postponed the experimental results (square markers) and the fitting results (continuous line) of BCFCM algorithm, experimented on images with different size.



(a)



(b)



(c)

Figure-3. Experimental and modelled CPU execution time per iteration for BCFCM algorithm.

The best statistical trend of the variation in terms of execution time per iteration with respect to the variation of cluster number is represented by a polynomial functions with an practically $R^2 = 1$:

$$
\begin{aligned}
ETPI_{CPU/img128}(s) &= 0.0011*c^2 + 0.0026*c + 0.0052 \\
ETPI_{CPU/img512}(s) &= 0.0162*c^2 + 0.0662*c - 0.0131 \\
ETPI_{CPU/img1024}(s) &= 0.0668*c^2 + 0.2364*c + 0.0257 \\
ETPI_{CPU/img2048}(s) &= 0.2664*c^2 + 0.9747*c - 0.0151 \\
ETPI_{CPU/img2708}(s) &= 0.4661*c^2 + 1.6953*c - 0.0595 \\
ETPI_{CPU/img3000}(s) &= 0.5721*c^2 + 2.0759*c + 0.0266 \\
ETPI_{CPU/img3200}(s) &= 0.6536*c2 + 2.3085*c + 0.2434
\end{aligned}
\qquad (12)
$$

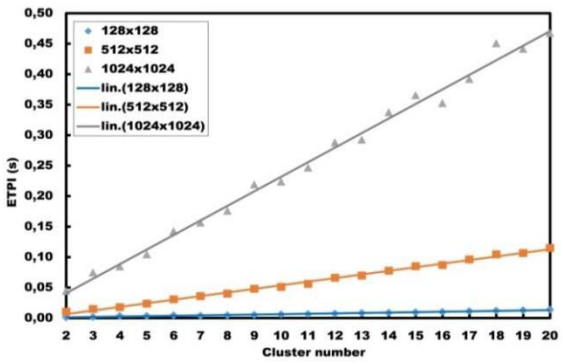
All these functions are polynomials of second order, which shows that the increase in execution time reported to a single iteration follows a growing law and this growth is even more pronounced as the number of cluster is more important. These statistical models are in good concordance with the experimental results and are only limited by the computational characteristics of the CPU and the amount of memory in our setup

5.4.2. GPU Execution time per iteration behaviour modelling

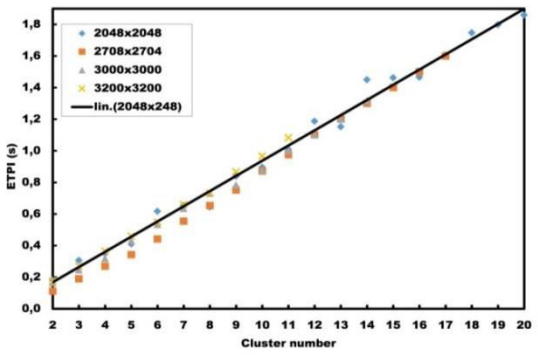
In this subsection, we intend to give from experimental executions of PBCFCM algorithm on GTX760 device, mathematical models that reflect as closely as possible the behaviour of this circuit when the number of clusters *c* that we want to extract from the image varies, keeping constant the variable size of the image.

As in the previous subsection, we present in Figure-4 executions time experimental results (square markers) and the fitting results (continuous line) for PBCFCM algorithm on GTX760 with the 7 test images used in the previous subsection.

In the following, we will present the results relative to mathematically modelling of the variation of the *ETPI (s)* a function of cluster number *c*. This will tell us about the behaviour of the used GPU device overlooked the variation of the cluster number *c* when executing PBCFCM with respect to the image data size parameter.



(a)



(b)

Figure-4. Experimental and modelled GPU execution time per iteration for PBCFCM algorithm on Nvidia GTX 760 GPU.

The best statistical trend of the variation in terms of execution time per iteration with respect to the variation of cluster number *c* is represented by a perfect linear functions with an practically $R^2=1$:

$$\begin{aligned}
&ETPI_{GTX760/img128}(s) = 0.0007*c - 0.0002\\
&ETPI_{GTX760/img512}(s) = 0.0061*c - 0.0066\\
&ETPI_{GTX760/img1024}(s) = 0.024*c - 0.0091\\
&ETPI_{GTX760/img2048}(s) = 0.0962*c - 0.0243\\
&ETPI_{GTX760/img27084}(s) = 0.1023*c - 0.1429\\
&ETPI_{GTX760/img3000}(s) = 0.0993*c - 0.0663\\
&ETPI_{GTX760/img3200}(s) = 0.0998*c - 0.0351
\end{aligned} \quad (13)$$

These models are in very good concordance with the experimental results and are only limited by the computational characteristics of the GPU.

The limitation is observed at the test image *img2708* for c=17, at the test image *img3000* for c=14 and at the test image *img3200* for c=11. This is due to insufficiency in terms of memory on the used GPU device.

Note that for reasons of clarity, we presented in Figure-4.b only the function modelling the variation of *ETPI(s)* variation for the *img2048* image. In fact, for images sizes greater than or equal to *2048x2048* we observe that the slope of the lines modelling the variation of the variable *ETPI(s)* function of *c* are too close.

5.5. Speedup GPU/CPU(x) variation modelling

In this part we focus on another a magnitude frequently used to evaluate the quality of a parallel implementation of an image processing algorithms, compared to its sequential implementations. It is about the ratio *speed up GPU/CPU(x)* between the execution time required for the convergence on CPU and execution time needed for convergence on GPU taking into account the same initial conditions. For these experiments, we used the same Lena image bank, each sample image is segmented by BCFCM and PBCFCM into a Cluster number *c* between 2 and 20 as is done in the previous subsection.

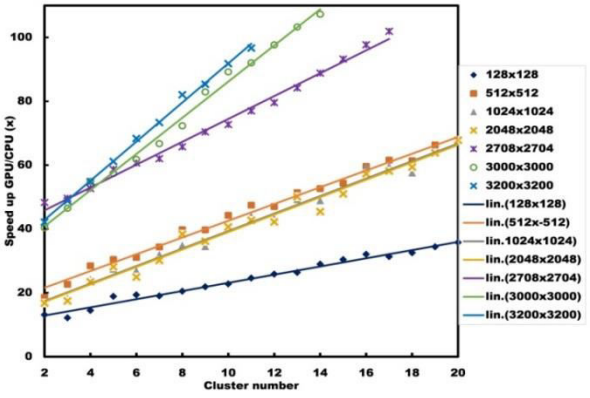


Figure-5. Experimental and modeled GPU/CPU (x) speed ups.

In Figure-5, we postponed the experimental results (markers) and the statistical fitting results (continuous lines) for the different images used in experiments.

Theoretical statistical fitting models show a perfect logarithmic behaviour of the variation in speed up versus cluster number c with an R^2 varying between 0.974 and 0.997 (eq.14).

$$\begin{aligned}
&img128 \quad SU_{GTX760/CPU}(x) = 1.2809*c + 10.336\\
&img512 \quad SU_{GTX760/CPU}(x) = 2.6234*c + 16.354\\
&img1024 \quad SU_{GTX760/CPU}(x) = 2.7416*c + 11.949\\
&img2048 \quad SU_{GTX760/CPU}(x) = 2.7327*c + 11.706\\
&img2708 \quad SU_{GTX760/CPU}(x) = 3.5811*c + 38.656\\
&img3000 \quad SU_{GTX760/CPU}(x) = 5.6658*c + 29.496\\
&img3200 \quad SU_{GTX760/CPU}(x) = 6.1297*c + 30.582
\end{aligned} \quad (14)$$

All these functions present linear behaviour, which shows that the increase in speed up*SU(x)* follows a growing law and this growth is even more pronounced as the number of cluster is more important. This confirms that the use of the parallel version of the algorithm is more desirable when the number of cluster is greater to benefit from the computational capability of the GPU. But as is mentioned in the previous section for the study on execution time per iteration variable, the problem of limitation is observed from the test image *img2708*.

6. CONCLUSIONS

Bias field artefact correction is still a challenging problem in image processing both at accuracy level as on the speed level; this is justified by the amount of recent work in the literature on this subject. After proposing massively parallel algorithm implementing one of the most popular technique to correct and segment images BCFCM and exploiting the performance offered by the modern Graphical Processing Units (GPU), we are focused in this work on the characterization and modelling of the behaviour of CPU and GPU devices against our implementations (sequential and parallel) to provide information that could guide researchers and users of this algorithms to the optimal situation that offer the best performances.

REFERENCES

[1] Y. Wu, C. He. 2015. A convex variational level set model for image segmentation. Signal Processing. 106: 123-133.

[2] D.E. Ilea, P.F. Whelan. 2011. Image segmentation based on the integration of colour-texture descriptors-a review. Pattern Recognition. 44(10): 2479-2501.

[3] M.A. Balafar, A.R. Ramli, M.I. Saripan, S. Mashohor. 2010. Review of brain MRI image segmentation methods. Artif. Intell. Rev. 33(3): 261-274.

[4] Sepideh Yazdani, Rubiyah Yusof, Alireza Karimian, Mohsen Pashna & Amirshahram Hematian. 2015.

Image Segmentation Methods and Applications in MRI Brain Images. IETE Technical Review. 32(6): 413-427.

[5] Gabor Szekely *et al*. 2000. Parametric estimate of intensity in homogeneities applied to MRI. IEEE Trans. on Medical Imaging. 19(3): 153-165.

[6] J. C. Bezdek, R. Ehrlich & W. Full. 1984. FCM: The fuzzy c-means clustering algorithm. Computers & Geosciences. 10(2): 191-203.

[7] L. Dzung Pham, L.P. Jerry. 1999. An adaptative fuzzy c means algorithm for image segmentation in the presence of intensity inhomogeneities, Pattern Recognition Letters. 20: 57-68.

[8] M.N. Ahmed, N.A. Mohamed, A.A. Farag, T. Moriarty. 2002. A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data. IEEE Trans. Med. Imaging. 21: 193-199.

[9] Deng W. Q., Li X. M., Gao X. & Zhang C. M. 2016. A Modified Fuzzy C-Means Algorithm for Brain MR Image Segmentation and Bias Field Correction. Journal of Computer Science and Technology. 31(3): 501-511.

[10] Aparajeeta J., Nanda P. K. & Das N. 2016. Modified possibilistic fuzzy C-means algorithms for segmentation of magnetic resonance image. Applied Soft Computing. 41: 104-119.

[11] Eklund A., Dufort P., Forsberg D. &LaConte S. M. 2013.Medical image processing on the GPU-Past, present and future. Medical image analysis. 17(8): 1073-1094.

[12] Pratx G. & Xing L. 2011. GPU computing in medical physics: A review. Medical physics. 38(5): 2685-2697.

[13] Erik Smistad, Thomas L Falch, Mohammad Mehdi Bozorgi, Anne C Elster, Frank Lindseth. 2015. Medical image segmentation on GPUs-A comprehensive review. Medical image analysis. 20(1): 1-18.

[14] T. Ivanovska R., Laqua L., Wang H. Volzke and K. Hegenscheid. 2013. Fast Implementations of the Levelset Segmentation Method with Bias Field Correction in MR Images: Full Domain and Mask-Based Versions, chapter book in: Pattern Recognition and Image Analysis. Volume 7887 of the series Lecture Notes in Computer Science (Berlin: Springer Heidelberg. pp. 674-681.

[15] Yin Kui-Ying, Sun Fa-Long, Zhou Sheng-Hua& Zhang Changchun. 2014. PAR Model SAR Image Interpolation Algorithm on GPU with CUDA. IETE Technical Review. 31(4): 297-306.

[16] Lin Y.-S.; Lee C.-L.; Chen Y.-C. 2017. Length-Bounded Hybrid CPU/GPU Pattern Matching Algorithm for Deep Packet Inspection. Algorithms. 10: 16.

[17] J. Tian, L. Zhu, S. Zhang, L. Liu. 2005.Improvement and parallelism of k-means clustering algorithm, Tsinghua Science and Technology 10(3): 277-281. ISSN 1007-0214, 01/21.

[18] J. MacQueen. 1967, June. Some methods for classification and analysis of multivariate observations. Proceedings of the 5th Berkeley symposium on mathematical statistics and probability. 1: 281-297.

[19] Chaolu F., Dazhe Z. & Min H. 2016. Image segmentation using CUDA accelerated non-local means denoising and bias correction embedded fuzzy c-means (BCEFCM)[J]. Signal Processing. 122(5): 164-189.

[20] N. AitAli, B. Cherradi, A. El Abbassi, O. Bouattane and M. Youssfi. 2016. Parallel Implementation of Bias Field Correction Fuzzy C-Means Algorithm for Image Segmentation. International Journal of Advanced Computer Science and Applications (IJACSA). 7(3): 367-374.

[21] Ait Ali N., Cherradi B., Bouattane O., Youssfi M. &Raihani A. 2015.New fine-grained clustering algorithm on GPU architecture for bias field correction and MRI image segmentation. In: the Procceding of the 27th IEEE International Conference on Microelectronics (ICM2015) pp.118-121.

[22] N. Ait Ali, B. Cherradi, A. El Abbassi, O. Bouattane and M. Youssfi. 2016. GPU based Implementation of Spatial Fuzzy C-means Algorithm for Image Segmentation, In: the Procceding of the 4th Edition of the IEEE International Conference on Information Science and Technology (CiSt'16), 24-26 October, Tangier, Morocco.

[23] Chuang K. S., Tzeng H. L., Chen S., Wu J. & Chen T. J. 2006. Fuzzy c-means clustering with spatial

information for image segmentation. Computerized medical imaging and graphics. 30(1): 9-15.

[24] N. AitAli, B. Cherradi, A. El Abbassi, O. Bouattane and M. Youssfi, Fuzzy Spatial Clustering Algorithm on GPU: Characterization and Behavior Modeling. In: the Procceding of the 2nd Edition of the IEEE International Conference on Electrical Sciences and Technologies in Maghreb (CISTEM'16), 26-28 October 2016, Marrakesh, Morocco.

[25] N. AitAli, B. Cherradi, A. El Abbassi, O. Bouattane and M. Youssfi, Fuzzy Spatial Clustering Algorithm on GPU: Characterization and Behavior Modeling. 3rd Edition of the IEEE International Conference on Advanced Technologies for Signal and Image Processing (ATSIP'2017), 22-24 Mai 2017, Fez, Morocco.

[26] 2015. https://developer.nvidia.com/cuda-zone.