Performance evaluation of spatial fuzzy C-means clustering algorithm on GPU for image segmentation

Noureddine Ait Ali 1 · Ahmed El Abbassi 1 · Omar Bouattane 2

Received: 19 March 2021 / Revised: 25 April 2022 / Accepted: 1 August 2022 / Published online: 10 August 2022
© The Author(s), under exclusive licence to Springer Science+Business Media, LLC, part of Springer Nature 2022

Abstract
Image processing by segmentation technique is an important phase in medical imaging such as MRI. Its objective is to analyze the different tissues in human body. In research area, Fuzzy set is one of the most successful techniques that guarantees a robust classification. Spatial FCM (SFCM); one of the fuzzy c-means variants; considers spatial information to deal with the noisy images. To reduce this iterative algorithm’s execution time, a hard SIMD architecture has been planted named the Graphical Processing Unit (GPU). In this work, a great contribution has been done to diagnose, confront and implement three different parallel implementations on GPU. A parallel implementations’ extensive study of SFCM entitled PSFCM using 3 × 3 window is presented, and the experiments illustrate a significant decrease in terms of running time of this algorithm known by its high complexity. The experimental results indicate that the parallel version’s execution time is about 9.46 times faster than the sequential implementation on image segmentation. This gain in terms of speed-up is achieved on the Nvidia GeForce GT 740 m GPU.

Keywords Fuzzy C-mean · SFCM · SIMD architecture · Clustering · GPU · CUDA

1 Introduction
The parallelism is in fact the trending of science, and the computing power that can offer to complex algorithms makes it an obligation not just a choice. The multi-core architecture birth at the processors level was a beginning to deal with such obligations. There are several techniques that rely heavily on multi-core. In [13], the authors suggested an
implementation of k-means [30] clustering algorithm on a parallel machine emulator [54] of the Reconfigurable Mesh Computer (RMC). This parallel implementation takes the SIMD architecture advantage from their proposed platform [54] to reduce its complexity. Otherwise, the open Multi Processing (OpenMP) application programming interface provides an efficient abstraction for implementing parallel algorithms for multi-core processors [9].

Many scholars worked on different technologies to speed up their software applications using regular multi-core CPUs [15, 23]. Other works have operated on the multiprocessors as a CPUs cluster and have implemented some clustering algorithms such as Fuzzy C-means (FCM) [11], Spatial Fuzzy C-means (SFCM) [18] and Gaussian Kernel-based FCM with spatial bias correction [34]. Yet, accelerating applications at the CPU level is still respectable compared to what is required in terms of computing. Therefore, many research fields were benefited from GPU acceleration techniques [10, 14, 22, 39, 40, 44].

In medical imaging field, image segmentation is considered as one of the most important steps to discriminate various body tissues for better diagnosis [21, 25, 31, 32]. However, the segmentation process suffers from many crucial drawbacks keeping it an active research area [24, 33, 46, 52, 55]. One of the difficulties facing segmentation algorithms is time-consuming issue due to an actual medical images’ high density.

To fix the mentioned matters, several clustering algorithms count on the parallel processing for execution time enhancement. Moreover their high computing capabilities Graphics Processing Units (GPU) are more adapted to image processing techniques [20, 29, 35]. Clustering algorithms not only can benefit from the parallel behavior of GPU processors but also can provide interesting performances [2, 6, 7, 49].

The present work deals with the execution time’s reduction, one of the most interesting variants of FCM algorithm that is SFCM [18]. This powerful algorithm demonstrates great results for its performance on image segmentation accuracy compared to FCM algorithm. Nevertheless, it is very time intensive consumer. In this study, the contribution is mainly focused on the SFCM algorithm parallelization strategies on massively parallel architecture (GPU). The key target is to overcome the high complexity’s drawback by reducing the execution time. In fact, the GPU computing performance’s advantage has been taken to implement three parallel versions that successfully hasten the SFCM. As it has already been mentioned that one of these implementations is considered as a direct application inspired from literature after being applied and compared to the two proposed implementations.

This work main contributions are summarized as follow:

– Taking advantage of the GPU massively parallel architecture, where a three parallel implementations comparison of the SFCM algorithm is suggested.
– To validate and illustrate the performance of the proposed parallel implementations, many experiments on different medical images segmentation, were conducted.
– Obtained results in terms of parallel versions’ speed up with respect to the sequential one shows an acceleration up to 9.46 times on the Nvidia GeForce GT 740 m GPU.
The remaining of the paper is structured as follows: In Section 2, some related works’ review. Section 3 briefly presents the SFCM algorithm sequential implementation. Section 4 Details the three studied parallel implementations of the SFCM algorithm. Section 5 presents the experimental results. Finally, the paper conclusion in Section 6.

2 Review of some related work

Currently, many lightweight threads in a GPU encourage on properly designed parallel algorithms for a performance relatively faster than sequential implementation. In addition, Compute Unified Device Architecture (CUDA) programming environment provides an exceptional foundation for programmers to efficiently use NVIDIA(R) GPUs [41]. For instance, many image and signal processing applications, and heuristic optimization method such as (Smith-Waterman algorithm, genetic quantum algorithm, simulated annealing, particle swarm and the solution of differential equation) are few examples that have been implemented successfully using CUDA [4, 12, 37, 53].

Many image segmentation techniques try to discharge their computations to GPU hardware. Researchers in [26, 38] performed a level set medical image segmentation on GPU using efficiently CUDA programming environment. They have lowered significantly the execution time from about 8 s to less than 0.5 s for the level set method.

In [36], the authors implemented a deformable image registration algorithm on GPU. They applied this algorithm to segment the right ventricular in cardiac MRI images. The speedup improvements achieved were around 19 times faster over the sequential implementation.

Authors in [50] recommended a parallel nearest neighbor partitioning (NNP) method to accelerate NNP. In their method, blocks and threads are used to evaluate potential neural networks and to perform parallel subtasks. The speedup reached by PNNP is about 70 times faster than the NNP.

In [47], the authors proposed a parallel centerline extraction algorithm on GPU that withdraw tabular structures such as blood vessels from different medical imaging modalities. The results demonstrate that the method can extract airways and vessels in 3–5 s on a modern GPU.

Nitin Satpute et al. [42] proposed parallel seeded region growing for vessel segmentation using CT liver images. The gradient based fast parallel SRG (seeded region growing) for 2D vessel segmentation has reached an acceleration of 1.9 times faster compared to the state-of-the-art.

The earliest implementation of supervised algorithms is k-means on the Graphic Processor Unit (GPU) which was proposed in [27]. Many scholars accurately address K-means as general purpose clustering algorithm [51]. According to the previous literature, the first popular K-mean GPU implementation is suggested and described in [17].

One of the alternate powerful unsupervised classification technique is the fuzzy c-means algorithm [11] as several parallel implementations of this algorithm on GPUs have been proposed [2, 7, 8].
Cecilia et al. [16] proposed three different data clustering algorithms which are k-means, Fuzzy minimals (FM) and Fuzzy C-means (FCM). They applied their algorithms to IOT technology for immediate data analysis. The authors executed the algorithms on GPU peripheral and HPC (High performance Computing) cloud platforms suitable for cloud implementation. They obtained an acceleration factor of 11x for GPU over CPU, they also obtained a power saving of up to 150% for GPU and HPC devices compared to sequential execution.

The article published in [48] presents different steps to improve the conventional FCM algorithm’s performance in CPU. The suggested method provided equal segmentation accuracy compared with the existing methods but minimizes the segmentation time considerably up to seven times and iterations’ average number, after that they have compared their method’s performance to a GPU based implementation of FCM.

Author in [43] parallelized fuzzy c-mean for Brain tumor segmentation using FLAIR MRI sequences. The used method’s efficiency shows an acceleration of 17.6 times over the sequential implementation.

Li et al. [28] presented a modified version of FCM algorithm to improve the membership matrix calculation and the class centers updating. Their algorithm’s upgrade version was executed on GPU hardware using CUDA environment to reduce the execution time and to rise the visual including the segmentation efficiency. In their experiments, they used different images of different data sizes on a GTX 260 GPU and Intel Core 2 CPU. The authors achieved at least 10 times speed up improvements over the sequential FCM implementation.

In [5] authors have parallelized brFCM; a faster version of the standard FCM. They have compared their implementation to sequential brFCM and standard FCM, and they also have got a speedup of 2.24 and 23.42 times simultaneously. The GPUs used for their experiments are Tesla M2070 and Tesla K20m.

In [4], a parallel DFC algorithm was implemented and analyzed in GPU using two methods, the first one is called thread-based and the second one is called block-based. They have also used openMP to parallelize the DFC on fMRI data. As a result, a speedup ranging from 18.5 to 157 times on GPU and 7.7 times with openMP have been obtained respectively.

Otherwise, in [1, 3], the authors have proposed a bias field’s parallel implementation on GPU architecture for three different devices. They have reached a speedup of 52 times on GTX 580, 21 times on GTX 760 and 12 times on GT 740 m for big data on windows ×32 operating system. They have as well characterized a relevant way of their implementation’s behavior on the three different device cards for small, medium and large data size respectively.

### 3 Sequential version of the spatial fuzzy C-means algorithm

Applied in image segmentation, the SFCM fuzzy clustering algorithm [18] assigns image pixels to each class or category by using fuzzy memberships in addition to the neighbor’s effect. Let $X (x_1, x_2, ..., x_N)$ refers an image with $N$ pixels to be partitioned into $c$ clusters, where $x_i$ represents the grayscale feature. The algorithm is an iterative optimization that minimizes the cost function $J$ defined by Eq. 1:
\[ J = \sum_{j=1}^{N} \sum_{i=1}^{C} u_{ij}^m \|x_j - C_i\|^2 \]  

Where:

- \( C_i \) is the centroid of cluster \( i \);
- \( N \) is the number of pixels.
- \( m \) is a fuzzy weighting exponent.
- \( \|x_j - C_i\|^2 \) is a square of Euclidean distance between \( i \)th centroid \( C_i \) and \( j \)th data point \( x_j \).
- \( u_{ij} \) is the degree of membership of the pixel \( x_j \) for the cluster \( C_i \).
- \( C \) is the number of clusters that should respect the condition: \( 2 \leq c < N \).

The previous membership matrix \( u \) is given by the Eq. 2:

\[ u_{ij} = \frac{1}{\sum_{k=1}^{c} \left( \frac{d_{ik}}{d_{ij}} \right)^{2/(m-1)}} \]  

Where:

- \( u_{ij} \in [0, 1], \forall i, j \)
- \( \sum_{i=1}^{c} u_{ij} = 1, \forall j = 1, \ldots, N \)
- \( 0 < \sum_{j=1}^{N} u_{ij} < N, \forall i = 1, \ldots, C \)

The authors have included the neighborhood effect since the pixels are highly correlated. In other words, these neighboring pixels have similar feature’s values and the probability they belong to the same cluster is great.

To exploit the spatial information, a spatial function is defined as:

\[ h_{ij} = \sum_{k \in NB(x_j)} u_{ik} \]  

Where \( NB(x_j) \) represents a square window centered on pixel \( x_j \) in the spatial domain. A 3 × 3 windows were used throughout this work. The spatial function \( h_{ij} \) represents the probability where pixel \( x_j \) belongs to \( i \)th cluster. The spatial function of a pixel for a cluster is large if most of its neighborhood belongs to the same clusters. The spatial function is incorporated into new membership function as follows:

\[ u_{ij} = \frac{u_{ij}^p h_{ij}^q}{\sum_{k=1}^{c} u_{ik}^p h_{kj}^q} \]  

\( p \) and \( q \) are parameters to control the relative importance of both functions \( u \) and \( h \).

To update the centroids vector, the following function has been used:

\[ V_i = \frac{\sum_{j=1}^{N} u_{ij}^m x_j}{\sum_{j=1}^{N} u_{ij}^m} \]
The Algorithm 1 represents the sequential implementation’s different steps.

Algorithm 1 Sequential SFCM
1: Choose the cluster centers randomly
2: Initialize the objective function
3: Initialize the Membership functions
4: Repeat
5: Update the membership function using equation (2)
6: Update the spatial function using equation (3)
7: Update the new membership function using equation (4)
8: Update the cluster centers vectors using equation (5)
9: Calculate the objective function $J_{(N+1)}$ using equation (1)
10: Until $|J_{n+1} - J_n| < \varepsilon$

4 Parallel implementation of the SFCM algorithm

In this section, the algorithm’s different implementations will be introduced. Indeed, 3 hybrid methods are being studied, one of which is an application inspired by the literature and adopted [5, 45].

One of the major problems that increases the execution time of fuzzy C-mean and its variants is the power’s calculation. To overcome this crisis, the scalar product can be used so that the algorithm will only deal with special cases. As far as this paper is concerned, floating power has been used instead of double power since it is considered faster and produces the same results. This allows us to process the algorithm in a general way whatever the parameter $m$ ($\forall m \in \mathbb{R}$ and $m > 1$).

4.1 SFCM_PV1

In this first version, an advantage of the graphics processor’s power was taken to minimize the data transfer cost between CPU and GPU. This implementation’s steps are arranged as follows

1- Initialize the input data such as image, initial centroids, $m$, $p$, and $q$ parameters and membership matrices at CPU-level. Then, transfer the image to the GPU.
2- The second step intends to invoke the first Kernel which permits to calculate the membership matrix at the GPU level. As it is obvious, the calculation of the latter coincides perfectly with the GPU architecture, allowing it to well perform at this level.
3- The second Kernel consists in calculating the spatial function simultaneously for each pixel, then using this function to calculate the new membership matrix which also seems, in turn, very suitable to be calculated on the GPU.
4- For the Objective function $J = \sum_{i=1}^{c} \sum_{j=1}^{n} u_{ij}^{m} \|x_j - C_i\|$, the term $u_{ij}^{m} \|x_j - C_i\|$ was calculated in parallel where each thread calculates the term by iterating on the clusters. Whereas the final summation, a predefined function of the CUBLAS library was used to retrieve it at the CPU level. The same applies for the centroids’ updating of $V_i = \frac{\sum_{j=1}^{N} u_{ij}^{m} x_j}{\sum_{j=1}^{N} u_{ij}^{m}}$ each.
thread calculates the term $u_{ij}^m x_j$, the term $u_{ij}^m$ and keeps it on a 1D vector. While for the summation, the same function was applied as the one used at the objective function’s level so as the function returns the vector’s summation value corresponding to the centroid. Finally, the division is carried out at CPU level.

Indeed, this algorithm has already been parallelized on the GPU in [2]. Moreover, since the pixels at the edges do not have a neighborhood window eliminating them was thoughtful. In this work, the pixels at the edges were taken into consideration so that each one has the characteristics of the closest pixel (Fig. 1). In addition to that, the execution mode previously used was debug mode, which is not as well performing as the release mode (visual studio). Small loops were used for more performance. Figure 1 explains the approach used in the 3 implementations.

To reduce the data transfer’s latency between CPU and GPU, a pinned memory was replaced by a pageable memory to slightly reduce the third implementation execution time. This led to a decrease between 0.5 s and 1 s since it requires a huge amount of data transfer. The speedup improved by 1 degree.
The Algorithm 2 represents the first parallel implementation’s different steps.

Algorithm 2 Parallel SFCM_PV1
1: Choose the cluster centers randomly
2: Initialize the objective function
3: Initialize the Membership functions
4: Transfer the image to GPU
5: Repeat
6: Transfer clusters to GPU constant memory
7: Update the membership function using equation (2) on GPU
8: Update the spatial function using equation (3) on GPU
9: Update the new membership function using equation (4) on GPU
10: Update the cluster centers vectors using equation (5) on GPU except the division is done on CPU
11: Calculate the objective function $J_{(N+1)}$ using equation (1) on GPU
12: Until $\|J_{n+1} - J_n\| < \varepsilon$ on CPU

4.2 SFCM_PV2

The second implementation follows the same previous reasoning except that at the 4th point, the vectors of Eqs. (1 and 5) have been transferred to the CPU to calculate their summation. The objective here is to exploit the CPU performance at this level to update the objective function and the new centroids.

The algorithm 3 represents the second parallel implementation’s different steps.

Algorithm 3 Parallel SFCM_PV2
1: Choose the cluster centers randomly
2: Initialize the objective function
3: Initialize the Membership functions
4: Transfer the image to GPU
5: Repeat
6: Transfer clusters to GPU constant memory
7: Update the membership function using equation (2) on GPU
8: Update the spatial function using equation (3) on GPU
9: Update the new membership function using equation (4) on GPU
10: Compute the cluster centers numerator and denominator on GPU without summation
11: Calculate the objective function $J_{(N+1)}$ using equation (1) without computing summation on GPU
12: Transfer the step 10 and 11 vectors to CPU
13: Update cluster centers on CPU
14: Update the objective function on CPU
15: Until $\|J_{n+1} - J_n\| < \varepsilon$ on CPU
4.3 SFCM_PV3

The third implementation has been widely used in literature [5, 45]. So, it was applied to this algorithm. The idea behind this version is:

1- Calculate the U membership matrix on the GPU
2- Calculate the spatial function and the new membership matrix $U'$ on the GPU and then transfer it to the CPU since the summation performs better at the CPU level, as does the objective function, which is calculated at the CPU level.

The Algorithm 4 represents the third parallel implementation’s different steps.

Algorithm 4 Parallel SFCM_PV3

1: Choose the cluster centers randomly
2: Initialize the objective function
3: Initialize the Membership functions
4: Transfer the image to GPU
5: Repeat
6: Transfer clusters to GPU constant memory
7: Update the membership function using equation (2) on GPU
8: Update the spatial function using equation (3) on GPU
9: Update the new membership function using equation (4) on GPU
10: Transfer the new membership to CPU.
11: Update the cluster centers vectors using equation (5) on CPU
12: Update the objective function $J_{(N+1)}$ using equation (1) on CPU
13: Until $\|J_{n+1} - J_n\| < \epsilon$ on CPU

5 Results and discussion

5.1 Experiment setup

Algorithms were implemented using Microsoft Visual Studio 2017 program and CUDA 9.2 libraries (https://developer.nvidia.com/cuda-92-download-archive). Sequential FCM algorithm is programmed using C language. Speed-up results were carried out on: Intel(R) Core(TM) i5-3230M 4 cores 2.6 GHz CPU and Nvidia GeForce GT 740 m GPU.

5.2 Image database

To validate the parallel implementations versus sequential ones, an extensive experiment was carried out on the four parallel versions of the clustering algorithm on different images with different densities, and where 16 medical images from a wide database were used. All these images are segmented into 5 clusters (Fig. 2).
5.3 Some GPU implementation specifications

In order to get an optimized implementation, GPU programming has many challenges to attain high performance improvements over the sequential one [19, 41].

The first challenge is the usage management of the memory’s different types on GPU. In fact, transferring data from the CPU side to the GPU and vice versa causes important delays. To overcome this issue, data transactions between CPU and GPU must be tracked and any useless transaction should be eliminated.

The second main challenge is the optimal GPU utilization. The threads’ best number run in single block of memory is 256 threads per block as shown Table 1. This value is used for our parallel implementations.

Fig. 2 Different Medical images used for experiments

| Image Name                      | Dimensions        |
|---------------------------------|-------------------|
| Cardiac-1                       | 1024x1024         |
| Brain-3                         | 1024x1024         |
| Brain-1                         | 1024x1024         |
| Breast-1                        | 1024x1024         |
| Breast-2                        | 1024x1024         |
| Brain-2                         | 1024x1296         |
| Brain-4                         | 946x1203          |
| Brain-5                         | 1024x1300         |
| Prostate-1                      | 1024x1024         |
| Tuberculosis                    | 1024x991          |
| Viral-Pneumo                    | 1133x1024         |
| COVID-19                        | 1024x1024         |
| Norm-X-Ray                      | 1024x840          |
| Prostate-2                      | 1024x1024         |
| Breast-3                        | 1024x806          |
| Cardiac-2                       | 1024x1024         |
5.4 Notation

The expression (6) is employed to calculate the speedup achieved by our parallel implementation’s versions over the sequential ones.

\[
\text{Speedup} = \frac{\text{Sequential Execution Time}}{\text{Parallel Execution Time}} \quad (6)
\]

5.5 Implementations performances of SFCM based implementations on MRI images

For the performance experiment different MRI images of different body organs were used such as brain, mammography, prostate and cardiac.

5.5.1 Execution time results

From Figs. 3, 4, 5 and 6 it shows the execution time evolution according to the algorithms implemented on the GPU and CPU cards respectively. The experimental tests were spread

| Thread per Block/ | 1.0  | 1.1  | 1.2  | 1.3  | 2.0  | 2.1  | 3.0  |
|------------------|------|------|------|------|------|------|------|
| Compute Capability | 64   | 67   | 50   | 50   | 33   | 33   | 50   |
|                  | 96   | 100  | 75   | 75   | 50   | 50   | 75   |
|                  | 128  | 100  | 100  | 100  | 67   | 67   | 100  |
|                  | 192  | 100  | 100  | 94   | 94   | 100  | 94   |
|                  | 256  | 100  | 100  | 100  | 100  | 100  | 100  |
|                  | 384  | 100  | 100  | 75   | 75   | 100  | 94   |
|                  | 512  | 67   | 67   | 100  | 100  | 100  | 94   |
|                  | 768  | N/A  | N/A  | N/A  | N/A  | 100  | 75   |
|                  | 1024 | N/A  | N/A  | N/A  | N/A  | 67   | 100  |

**Fig. 3** Execution time comparison between different implementations for Brain-5
over 4 different MRI brain images to prove and validate the parallel implementations’ performance’s credibility.

These results demonstrate that SFCM_PV1 is better performing among the three parallel algorithms and the sequential algorithm tested on the four different images. The minimum execution time achieved for SFCM_PV1, SFCM_PV2 and SFCM_PV3 is about 2.32 s, 2.5 s and 4.01 s respectively for the Brain-2 image (Fig. 4).

Figure 7 displays a summary histogram showing the run-time comparison’s results for different implementations and different images used above.

The results that come out obviously show how the three parallel implementations are better than the sequential ones. In addition, the two parallel implementations SFCM_PV1 and SFCM_PV2 are more efficient than SFCM_PV3 for all experimental test images.
To summarize the execution times’ overall results for the different images used, the table below (Table 2) has been drawn up. It summarizes the experimental validation’s measurements on different images and methods evaluated in seconds.

In order to broaden the test images’ base, the measurements have been extended to other medical images such as: Breast-1, Breast-2, Cardiac and Prostate. In fact, for Breast-1, the sequential execution time is around 54.52 s. This is considerably reduced to 6.13 s, 7.24 and 11.17 with our parallel implementations SFCM_PV1, SFCM_PV2 and SFCM-PV3 respectively.

Note that SFCM-PV1 implementation always proves its superiority over other parallel and obviously sequential implementations.

![Fig. 6](image1.png) Execution time comparison between different implementations for Brain-3

![Fig. 7](image2.png) Execution time comparison between different implementations and different images
5.5.2 Speedups results

The Fig. 8 represents the speed ups’ results achieved for the three parallel implementations validated on different medical images. It is apparent that the SFCM_PV1 algorithm performs better than the other two. Its speed up reaches about 9.5 for the Brain-3 image.

5.5.3 Image segmentation validation using SFCM_PV1

To validate the accuracy performance, experiments were conducted. Figure 9 represents a set of tests on the image segmentation quality based on parallel approaches. For this purpose, four

| Images/Implementations     | Sequential(s) | SFCM_PV1(s) | SFCM_PV2(s) | SFCM_PV3(s) |
|----------------------------|---------------|-------------|-------------|-------------|
| Brain-1                    | 50.45         | 6.15        | 7.24        | 11.16       |
| Brain-2                    | 19.51         | 2.32        | 2.50        | 4.01        |
| Brain-3                    | 64.83         | 6.85        | 8.23        | 13.21       |
| Brain-4                    | 47.25         | 5.23        | 6.24        | 9.89        |
| Brain-5                    | 99.34         | 10.68       | 13.21       | 20.72       |
| Breast-1                   | 54.52         | 6.13        | 7.24        | 11.17       |
| Breast-2                   | 45.17         | 5.07        | 5.90        | 9.46        |
| Breast-3                   | 37.41         | 4.5         | 5.08        | 7.84        |
| Cardiac-1                  | 49.02         | 5.59        | 6.49        | 10.04       |
| Prostate-1                 | 72.52         | 8.32        | 9.99        | 16.15       |
| Tuberculosis               | 27.79         | 3.30        | 3.57        | 5.82        |
| Norm-X-Ray                 | 55.69         | 6.50        | 7.33        | 12.25       |
| Viral- Pneumo              | 32.94         | 3.73        | 4.18        | 7.35        |
| COVID-19                   | 66.08         | 7.43        | 8.58        | 14.83       |
| Prostate-2                 | 95.26         | 10.45       | 12.14       | 20.7        |
| Cardiac-2                  | 47.96         | 5.45        | 6.14        | 10.33       |

Fig. 8 Speedups times’ comparison between different implementations and different medical images
different images were used with different noise types. The first two Bain-front and Bain-side images are brought from the Brain web database\footnote{http://brainweb.bic.mni.mcgill.ca/brainweb/} with a noise level up to 9%.

The third image is subject to Gaussian white noise, while the fourth image is subject to salt noise. The obtained results prove the segmentation’s quality of the proposed implementation has the same original algorithm robustness. In addition, a contribution of another noise types, namely salt and paper noise and Gaussian noise. The results found prove that the power and the performance of the implementations are very satisfactory regarding noise and processing speed.

\begin{figure}[h]
\begin{center}
\includegraphics[width=\textwidth]{fig9.png}
\end{center}
\caption{Experiment results of segmentation comparison between FCM algorithm and SFCM_PV1 with different parameters and different noise}
\end{figure}
Figure 10 shows the experimental results of the image segmentation used above to evaluate the performance of the implemented parallel algorithm SFCM_PV1.

6 Conclusion

The SFCM algorithm’s study for the noisy image’s segmentation has revealed the power and the robustness of this classification method independently of the noise type. Furthermore, the parallelization success on a massively parallel architecture especially for large images is
another asset of the studied algorithm. Fact, thanks to the parallel implementations’ experiments going from a few dozen seconds to a few seconds was an accomplishment. This significant reduction in execution time is translated into a consistent acceleration of approximately 9.42 times compared to sequential execution.

Declarations

Conflicts of interests/competing interests  The Authors declare(s) that there is no conflict of interest. The authors whose names are listed immediately below certify that they have NO affiliations with or involvement in any organization or entity with any financial interest or non-financial interest in the subject matter or materials discussed in this manuscript.

References

1. Aitali N, Cherradi B, Bouattane O et al (2015) New fine-grained clustering algorithm on GPU architecture for bias field correction and MRI image segmentation. In: 2015 27th international conference on microelectronics (ICM). IEEE, Casablanca, pp 118–121
2. Aitali N, Cherradi B, El Abbassi A et al (2016) GPU based implementation of spatial fuzzy c-means algorithm for image segmentation. In: 2016 4th IEEE international colloquium on information science and technology (CiSt). IEEE, Tangier, pp 460–464
3. Aitali N, Cherradi B, El A, et al (2016) Parallel Implementation of Bias Field Correction Fuzzy C-Means Algorithm for Image Segmentation. ijacsa 7:. https://doi.org/10.14569/IJACSA.2016.070352
4. Akgün D, Sakoğlu Ü, Esquivel J, Adinoff B, Mete M (2015) GPU accelerated dynamic functional connectivity analysis for functional MRI data. Comput Med Imaging Graph 43:53–63. https://doi.org/10.1016/j.compmedimag.2015.02.009
5. Al-Ayyoub M, Abu-Dalo AM, Jararweh Y et al (2015) A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation. J Supercomput 71:3149–3162. https://doi.org/10.1007/s11227-015-1431-y
6. Ali NA, Cherradi B, El Abbassi A et al (2019) Parallel implementation and performance evaluation of some supervised clustering algorithms for MRI images segmentation. In: Proceedings of the 4th international conference on big data and internet of things. ACM, Rabat, pp 1–7
7. Ali NA, Abbassi AE, Cherradi B (2021) The performances of iterative type-2 fuzzy C-mean on GPU for image segmentation. J Supercomput 78:1583–1601. https://doi.org/10.1007/s11227-021-03928-9
8. Ali NA, Hamida S, Cherradi B et al (2022) A computational performance study of unsupervised data clustering algorithms on GPU. In: 2022 2nd international conference on innovative research in applied science, engineering and technology (IRASET). IEEE, Meknes, pp 1–6
9. Bak S, Bertoni C, Boehm S, Budiardja R, Chapman BM, Doerfert J, Eisenbach M, Finkel H, Hernandez O, Huber J, Iwasaki S, Kale V, Kent PRC, Kwack JH, Lin M, Luszczek P, Luo Y, Pham B, Pophale S, … Yeung PK (2022) OpenMP application experiences: porting to accelerated nodes. Parallel Comput 109:102856. https://doi.org/10.1016/j.parco.2021.102856
10. Baúto J, Canelas A, Neves R, Horta N (2018) Parallel SAX/GA for financial pattern matching using NVIDIA’s GPU. Expert Syst Appl 105:77–88. https://doi.org/10.1016/j.eswa.2018.03.026
11. Bezdek JC, Hathaway RJ, Sabin MJ, Tucker WT (1987) Convergence theory for fuzzy c-means: counter-examples and repairs. IEEE Trans Syst Man Cybern 17:873–877. https://doi.org/10.1109/TSMC.1987.6499296
12. Bharadwaj B, Saira Banu J, Madiajagan M, Ghalib MR, Castillo O, Shankar A (2021) GPU-accelerated implementation of a genetically optimized image encryption algorithm. Soft Comput 25:14413–14428. https://doi.org/10.1007/s00500-021-06225-y
13. Bouattane O, Cherradi B, Youssifi M, Bensalah MO (2011) Parallel c-means algorithm for image segmentation on a reconfigurable mesh computer. Parallel Comput 37:230–243. https://doi.org/10.1016/j.parco.2011.03.001
14. Bousselham A, Bouattane O, Youssifi M, Raihani A (2018) 3D brain tumor localization and parameter estimation using thermographic approach on GPU. J Therm Biol 71:52–61. https://doi.org/10.1016/j.jtherbio.2017.10.014
15. Cao P, Zhu X (2018) Speeding up exact algorithms for maximizing lifetime of WSNs using multiple cores. Wirel Commun Mob Comput 2018:1–12. https://doi.org/10.1155/2018/3830285
16. Cecilia JM, Cano J-C, Morales-García J, Llanes A, Imbermón B (2020) Evaluation of clustering algorithms on GPU-based edge computing platforms. Sensors 20:6335. https://doi.org/10.3390/s20216335
17. Che S, Boyer M, Meng J, Tarjan D, Sheaffer JW, Skadron K (2008) A performance study of general-purpose applications on graphics processors using CUDA. J Parall Distrib Comput 68:1370–1380. https://doi.org/10.1016/j.jpdc.2008.05.014
18. Chuang K-S, Tzeng H-L, Chen S, Wu J, Chen TJ (2006) Fuzzy c-means clustering with spatial information for image segmentation. Comput Med Imaging Graph 30:9–15. https://doi.org/10.1016/j.compmedimag.2005.10.001
19. Cook S (2013) CUDA programming: a developer’s guide to parallel computing with GPUs. Elsevier, MK, Amsterdam ; Boston
20. Eklund A, Dufort P, Forsberg D, LaConte SM (2013) Medical image processing on the GPU – past, present and future. Med Image Anal 17:1073–1094. https://doi.org/10.1016/j.media.2013.05.008
21. Elnakib A, Gimel’farb G, Suri JS, El-Baz A (2011) Medical image segmentation: a brief survey. In: El-Baz AS, Acharya UR, Laine AF, Suri JS (eds) Multi modality state-of-the-art medical image segmentation and registration methodologies. Springer New York, New York, NY, pp 1–39
22. Ferraz O, Silva V, Falcao G (2021) Hyperspectral parallel image compression on edge GPUs. Remote Sens 13:1077. https://doi.org/10.3390/rs13061077
23. Fritz F, Schmid M, Mottok J (2020) Accelerating real-time applications with predictable work-stealing. In: Brinkmann A, Karl W, Lankes S et al (eds) Architecture of computing systems – ARCS 2020. Springer International Publishing, Cham, pp 241–255
24. Li Y, Zhao K, Chu X, Liu J (2013) Speeding up k-means algorithm by GPUs. J Comput Syst Sci 79:216–229. https://doi.org/10.1016/j.jcss.2012.05.004
25. Li H, Yang Z, He H (2014) An improved image segmentation algorithm based on GPU parallel computing. JSW 9:1985–1990. https://doi.org/10.4304/jsw.9.8.1985-1990
26. Liu L, Zhang Z, Li S, Ma K, Zheng Y (2021) S-CUDA: self-cleansing unsupervised domain adaptation for medical image segmentation. Med Image Anal 74:102214. https://doi.org/10.1016/j.media.2021.102214
27. MacQueen J (1967) Some methods for classification and analysis of multivariate observations. Proceedings of the 5th Berkeley Symposium on Mathematical Statistics and Probability, vol 1. pp 281–296
28. Merjulah R, Chandra J (2017) Segmentation technique for medical image processing: a survey. In: 2017 international conference on inventive computing and informatics (ICICI). IEEE, Coimbatore, pp 1055–1061
29. Moujahid H, Cherradi B, Al-Sarem M et al (2022) Combining CNN and grad-cam for COVID-19 disease prediction and visual explanation. Intell Autom Soft Comput 32:723–745. https://doi.org/10.32604/iasc.2022.022179
30. Peng H, Gong W, Beckmann CF, Vedaldi A, Smith SM (2021) Accurate brain age prediction with lightweight deep neural networks. Med Image Anal 68:101871. https://doi.org/10.1016/jメディア.2020.101871
31. Pratx G, Xing L (2011) GPU computing in medical physics: a review. Proc SPIE 8183, High-Performance Computing in Remote Sensing, 818302. https://doi.org/10.1117/12.898281
32. Qu J, Liu X, Sun M, Qi F (2017) GPU-based parallel particle swarm optimization methods for graph drawing. Discret Dyn Nat Soc 2017:1–15. https://doi.org/10.1155/2017/2013673
33. Reska D, Kretowski M (2021) GPU-accelerated image segmentation based on level sets and multiple texture features. Multimed Tools Appl 80:5087–5109. https://doi.org/10.1007/s11042-020-09911-5
34. Rivera C, Chen J, Xiong N, Zhang J, Song SL, Tao D (2021) TSM2X: high-performance tall-and-skinny matrix–matrix multiplication on GPUs. J Parall Distrib Comput 151:70–85. https://doi.org/10.1016/j.jpdc.2021.02.013
40. Rodriguez D, Gomez D, Alvarez D, Rivera S (2021) A review of parallel heterogeneous computing algorithms in power systems. Algorithms 14:275. https://doi.org/10.3390/a14100275
41. Sanders J, Kandrot E (2011) CUDA by example: an introduction to general-purpose GPU programming. Addison-Wesley, Upper Saddle River, NJ
42. Satpute N, Naseem R, Palomar R, Zachariadis O, Gómez-Luna J, Cheikh FA, Olivares J (2020) Fast parallel vessel segmentation. Comput Methods Prog Biomed 192:105430. https://doi.org/10.1016/j.cmpb.2020.105430
43. Saxena S, Shama S (2018) Brain tumor segmentation by texture feature extraction with the parallel implementation of fuzzy C-means using CUDA on GPU. In: 2018 fifth international conference on parallel, distributed and grid computing (PDGC). IEEE, Solan Himachal Pradesh, pp 580–585
44. Sharma DK, Malikov V, Parygin D, Golubev A, Lozenitsina A, Sadovnikova N (2020) GPU-card performance research in satellite imagery classification problems using machine learning. Procedia Comput Sci 178:55–64. https://doi.org/10.1016/j.procs.2020.11.007
45. Shehab M, Al-Ayyoub M, Jararweh Y, Jarrah M (2017) Accelerating compute-intensive image segmentation algorithms using GPUs. J Supercomput 73:1929–1951. https://doi.org/10.1007/s11227-016-1897-2
46. Singh P, Bose SS (2021) A quantum-clustering optimization method for COVID-19 CT scan image segmentation. Expert Syst Appl 185:115637. https://doi.org/10.1016/j.eswa.2021.115637
47. Smidist E, Elster AC, Lindseth F (2014) GPU accelerated segmentation and centerline extraction of tubular structures from medical images. Int J CARS 9:561–575
48. Valsalan P, Srimakrishnan P, Sridhar S, Latha GCP, Priya A, Ramkumar S, Singh AR, Rajendran T (2020) Knowledge based fuzzy c-means method for rapid brain tissues segmentation of magnetic resonance imaging scans with CUDA enabled GPU machine. J Ambient Intell Humaniz Comput. https://doi.org/10.1007/s12652-020-02132-6
49. Wang H, Peng H, Chang Y, Liang D (2018) A survey of GPU-based acceleration techniques in MRI reconstructions. Quant Imaging Med Surg 8:196–208. https://doi.org/10.21037/qims.2018.03.07
50. Wang L, Zhu X, Yang B, Guo J, Liu S, Li M, Zhu J, Abraham A (2018) Accelerating nearest neighbor partitioning neural network classifier based on CUDA. Eng Appl Artif Intell 68:53–62. https://doi.org/10.1016/j.engappai.2017.10.023
51. Wasić MK, Narayanan PJ (2011) Scalable clustering using multiple GPUs. In: 2011 18th international conference on high performance computing. IEEE, Bengaluru, pp 1–10
52. Xu H, Liu L, Lei X, Mandal M, Lu C (2021) An unsupervised method for histological image segmentation based on tissue cluster level graph cut. Comput Med Imaging Graph 93:101974. https://doi.org/10.1016/j.compmedimag.2021.101974
53. Yang K, Li H, Huang Y et al (2017) GPU-accelerated particle swarm optimization for selective harmonic elimination in multilevel converters with unequal DC levels. In: IECON 2017 - 43rd annual conference of the IEEE industrial electronics society. IEEE, Beijing, pp 1186–1191
54. Youssfi M, Bouattane O, Bensalah MO (2010) A massively parallel re-configurable mesh computer emulator: design, modeling and realization. ISEA 03:11–26. https://doi.org/10.4236/isea.2010.31002
55. Zhang X, Wang H, Zhang Y, Gao X, Wang G, Zhang C (2021) Improved fuzzy clustering for image segmentation based on a low-rank prior. Comp Visual Media 7:513–528. https://doi.org/10.1007/s41095-021-0239-3

Publisher’s note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Springer Nature or its licensor holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.