J Supercomput (2017) 73:1929–1951
DOI 10.1007/s11227-016-1897-2

Accelerating compute-intensive image segmentation algorithms using GPUs

Mohammed Shehab[1] · **Mahmoud Al-Ayyoub**[1] · **Yaser Jararweh**[1] · **Moath Jarrah**[2]

Published online: 22 October 2016

Abstract Image segmentation is an important process that facilitates image analysis such as in object detection. Because of its importance, many different algorithms were proposed in the last decade to enhance image segmentation techniques. Clustering algorithms are among the most popular in image segmentation. The proposed algorithms differ in their accuracy and computational efficiency. This paper studies the most famous and new clustering algorithms and provides an analysis on their feasibility for parallel implementation. We have studied four algorithms which are: fuzzy C-mean, type-2 fuzzy C-mean, interval type-2 fuzzy C-mean, and modified interval type-2 fuzzy C-mean. We have implemented them in a sequential (CPU only) and a parallel hybrid CPU–GPU version. Speedup gains of 6× to 20× were achieved in the parallel implementation over the sequential implementation. We detail in this paper our discoveries on the portions of the algorithms that are highly parallel so as to help the image processing community, especially if these algorithms are to be used in real-time processing where efficient computation is critical.

Keywords Image segmentation · GPUs · Performance evaluation · Fuzzy clustering algorithms

✉ Yaser Jararweh
yijararweh@just.edu.jo

Mohammed Shehab
mohammed_shihab@daad-alumni.de

Mahmoud Al-Ayyoub
maalshbool@just.edu.jo

Moath Jarrah
mjarrahl@just.edu.jo

[1] Department of Computer Science, Jordan University of Science and Technology, Irbid, Jordan

[2] Department of Computer Engineering, Jordan University of Science and Technology, Irbid, Jordan

1 Introduction

Medical image technologies are being used to help doctors and experts in the medical field. There exist many technologies such as magnetic resonance imaging (MRI), computed tomography (CT), and digital mammography. These technologies produce a huge number of data that can be used to analyze and diagnose tissues abnormalities. Researchers in information technology domain facilitate the analysis process by developing image processing techniques that are accurate, efficient, and fast [22,31].

Image denoising, image recognition, image rotation and image segmentation are some research areas that apply information technology to medical images [8,9,11,12]. Image segmentation is a method that is used to extract objects from image. It is also used to extract or determine an important region from medical images which is called the region of interest (ROI) [13,33]. Different techniques are used to segment an ROI such as threshold-based methods, compression-based methods, and histogram-based methods.

Also, there are other methods for determining the ROI which are based on data mining and machine learning. These include clustering methods and region-growing methods [23,40,43,44]. ROI helps medical experts to diagnose the disease and monitor the progress level of cancerous diseases. Doctors can automatically determine the type of a tumor using a classification technique where image segmentation is a key operation [18,47]. Because of image segmentation's importance in the field, for instance, computer vision [27–29], many different algorithms were proposed in the last decade to enhance image segmentation techniques. Clustering algorithms are among the most popular in image segmentation. The proposed algorithms differ in their accuracy and computational efficiency. This paper studies the most famous and new clustering algorithms and provides an analysis on their feasibility for parallel implementation. We have studied four algorithms which are: fuzzy C-mean, type-2 fuzzy C-mean, interval type-2 fuzzy C-mean, and modified interval type-2 fuzzy C-mean. We have implemented them in a sequential (CPU only) and a parallel hybrid CPU–GPU version.

In the past, parallel programming has been used widely in clusters and distributed systems. Heavy tasks are divided into smaller tasks that are executed by different machines that are connected through a network [15,16]. Optimization solutions were proposed in the literature to enhance the performance based on the network topology [10,17]. On the other hand, parallel programming using graphic processing units (GPUs) has emerged which improves the computational efficiency of the algorithms without loosing accuracy. In GPU and CUDA programming, the application can run simultaneously more than 512 threads, while modern CPUs can run up to 32 threads at the same time [6,7].

In this work, speedup gains of $6\times$ to $16\times$ were achieved in the parallel implementation over the sequential implementation. Experiments were conducted using medical MRI and digital mammography images [36]. A hybrid CPU–GPU parallel programming was used to achieve high speedup [22]. We detail in this paper our discoveries on the portions of the algorithms that are highly parallel so as to help the image processing community, especially if these algorithms are to be used in real-time processing where efficient computation is critical.

This paper is organized as follows. Section 2 provides a background and related work in the area of image segmentation and parallel programming. Section 3 details the four fuzzy C-mean (FCM)-based algorithms that we considered in this paper. In Sect. 3.2, our parallel implementation of the four algorithms is described. Section 4 shows the results and presents a discussion. Finally, we conclude our work in Sect. 5.

2 Background and related work

This section covers two topics. First is the segmentation in medical images and the different methods in extracting ROI from images. The second topic is the parallel programming using GPUs and and its use to enhance the computation of the segmentation process [21,41].

2.1 Image segmentation techniques

Many algorithms were proposed in the literature to improve image segmentation operation. Tan and Isa [42] proposed an algorithm to mix histogram-based with fuzzy C-mean technique (HTFCM). The authors compared their algorithm with the ant colony algorithm and showed better results. Also, Tang proposed a segmentation algorithm that uses region-growing methods [43]. The method is based on seed region growing algorithm which was proposed earlier by Adams and Bischof [3]. It selects a seed pixel using the watershed segmentation method that was proposed by Shih and Cheng [40].

Other researchers used clustering methods algorithms due to their high efficiency such as in [42]. Ji et al. proposed in [30] a new segmentation algorithm using Gaussian mixture model (GMM), exception maximization (EM) algorithm and FCM. A local minimum is the main drawback of the GMM which was solved using FCM. The dimension of an image that was used in the authors' experimental results was 176×218. They used a 3 % noise and 60 % intensity. The results produced a 39 % accuracy.

Icer [26] proposed another GMM and FCM method to increase the accuracy of image segmentation. Corpus callosum (CC) was used as the dataset in this research. MR brain and midsagittal section of CC were the focus of the effort. Semra Icer used these two methods in two main steps. The first step extracts the GMM to segment the image by a probability density function. The second step applies the FCM. The accuracy of this process was 97 %.

Wang et al. proposed a new modified algorithm for fuzzy C-mean (MFCM) using a filtering method [46]. The accuracy of the classification process is improved due to the filtering method to reduce the effect of the noise. The authors used a diffusion filter with multi-scale fuzzy C-means technique. MRI images for brain were used as the dataset. This effort produces an accuracy of 87 % for images of 12 % noise and 84 % for images of 15 % noise. The FCM and MFCM without filtering produced 77 % accuracy for images of 12 % noise and 70 % accuracy for images of 15 % noise. The multi-scale technique and filtering method improved the classification and robustness.

Ahmed et al. presented a new modified fuzzy C-mean algorithm named bias-corrected fuzzy C-mean (BCFCM) [4]. The inhomogeneity of neighborhood pixels is the main attribute that was used in this paper. They modified the objective function to include the measurement of inhomogeneity of neighborhood pixels. They made a comparison between BCFCM, FCM, and expectation maximization algorithms (EM) for MR brain images. [4,5] used the FCM version that was used in [46]. BCFCM achieved about 94 % accuracy. The images used in the experiment have a Gaussian noise of $\sigma = 6$. FCM achieved 78 % and EM achieved 85 % accuracy for the same MR images of the same Gaussian noise.

Earlier research in clustering technique used the classical FCM algorithm with some modifications or new added steps to improve its accuracy. Some other research suggests to improve the segmentation process by adding new mathematical methods such as the work done by Rhee et al., where the authors developed the FCM using a new equation to measure the membership function of data points with centroids of clusters [23]. After that, Hwang et al. developed an algorithm using hypothesizing upper and lower memberships for each data point [25]. Later, Qiu et al. modified on the algorithm in [25] resulting in a new algorithm that reduces the noise effect of older versions [36].

2.2 Segmentation using parallel programming

After the big improvement of accuracy in image segmentation algorithms, researchers used a new hardware technology to improve the computation time as well. One of the hardware technologies is the graphic processing unit (GPU). GPU is used in high-performance computing (HPC) to reduce the execution time through parallel implementation of algorithms [22,38].

Rowiska et al. implemented the FCM clustering algorithm in a parallel version [37]. CUDA was used as the programming language. The authors compared the sequential code of the FCM with the parallel version. They tested the two algorithm versions using different colored images with different size. The membership and centroids functions were executed on the GPU side, while the objective function and the termination condition were executed on the CPU side. The sequential implementation was implemented in C++ and MATLAB. The GPU parallel implementation was faster than the CPU implementation by 7× (7 times).

Walters et al. proposed a parallel technique for two segmentation algorithms which are: the Markov random fields (MRF) and HMMERs Viterbi [45]. The used dataset was medical images for liver. The hardware that was used in this paper is NVIDIA GPU 8800 GTX. In this research, the speedup of the parallel version was enhanced by 130× for MRF algorithm and 38.6× for the HMMER algorithm over the sequential CPU implementation.

Pan et al. in [34] parallelized the region-growing method (RGM) and the multi-level watershed method (MLWM) algorithms using GPU hardware. They used abdomen images and brain images as datasets. The hardware that was used in their work was the Geforce 8500 GT. The speedup was enhanced by 8× for the region-growing method and 2.3× for the multi-level watershed method.

Many researchers such as in [22,34,37,45] utilize the GPU capabilities to increase the computational performance of image processing algorithms. GPUs uses single instruction multiple data (SIMD) technique as the parallel paradigm. While both CPUs and GPUs can run and manage thousands of threads simultaneously via time-slicing, modern CPUs can run 4–12 threads simultaneously whereas GPUs can run a thousand of threads [20,22].

3 FCM-based algorithms for image segmentation

3.1 Serial version of four FCM algorithms versions

In this section, we present the sequential CPU implementations of the FCM and the type-2 fuzzy C-mean (T2FCM) algorithms. The programming language that was used in this study is C-sharp programming language [24].

3.1.1 Fuzzy C-mean algorithm

FCM algorithm is one of the most famous clustering algorithms that is used to segment data into N clusters [14]. The segmentation process is performed in three main steps. The first step calculates the centroid for each cluster (initially, these centroids are generated randomly). This operation uses Eq. 1.

$$V_j = \frac{\sum_{i=1}^{n} u_{ij}^{m} \cdot x_i}{\sum_{i=1}^{n} u_{ij}^{m}} \tag{1}$$

where m is the fuzziness factor, n is the number of points, v_j is the center of cluster j

The second step calculates the membership of each data point to all clusters' centroids. This step is carried out using Eq. 2

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_k\|} \right)^{\frac{2}{m-1}}} \tag{2}$$

where C is the number of clusters, x_i is the object point.

The third step calculates the distance between data points and clusters' centers. The three steps are repeated until the difference of the total distance between points and centers is less than or equal to some error threshold [37]. This step is called the objective function and is calculated using Eq. 3. Algorithm 1 shows the sequential FCM algorithm.

$$Jm = \sum_{i=1}^{n} \sum_{j=1}^{c} u_{ij}^{m} \left\| x_i - c_j \right\|^2, \quad 1 \le m < \infty \tag{3}$$

where m is the fuzziness factor, n is the number of points, c is the number of clusters.

Algorithm 1 FCM Algorithm

```
1: procedure FCM($c$, $m$, $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Initialize the membership matrix $U_{ij}$ according to Equation 2
5:     repeat
6:         Calculate the objective function $J^k$ according to Equation 3
7:         Set the loop counter $k = k + 1$
8:         Calculate the cluster center vectors $C_k = [C_i]$ according to Equation 1
9:         Calculate the membership matrix $U_{ij}$ according to Equation 2
10:        Calculate the objective function $J^(k + 1)$ according to Equation 3
11:    until
12:    $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
13:    Do segmentation (Alg. 5)
14: end procedure
```

Algorithm 2 T2FCM Algorithm

```
1: procedure T2FCM($c$, $m$, $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Initialize the membership matrix $U_{ij}$ according to Equation 2
5:     Calculate the membership matrix $a_{ij}$ according to Equation 4
6:     repeat
7:         Calculate the objective function $J^k$ according to Equation 3 and replace $U_{ij}$ with $a_{ij}$
8:         Set the loop counter $k = k + 1$
9:         Calculate the cluster center vectors $C_k$ according to Equation 1
10:        Calculate the membership matrix $U_{ij}$ according to Equation 2
11:        Calculate the membership matrix $a_{ij}$ according to Equation 4
12:        Calculate the objective function $J^{k+1}$ according to Equation 3 and replace $U_{ij}$ with $a_{ij}$
13:    until
14:    $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
15:    Do Segmentation (Alg. 5)
16: end procedure
```

3.1.2 Type-2 fuzzy C-mean algorithm

Rhee et al. in [23] presented the type-2 fuzzy C-mean algorithm (T2FCM) for data classification. It uses the same FCM steps and equations except for the membership function. T2FCM uses Eq. 4 which increases the accuracy of the membership value. Algorithm 2 shows the sequential T2FCM algorithm.

$$a_{ij} = u_{ij} - \frac{1 - u_{ij}}{2} \tag{4}$$

3.1.3 Interval type 2 fuzzy C-mean

Hwang et al. in [25] proposed the algorithm interval type-2 fuzzy C-mean (IT2FCM). This algorithm is based on the traditional FCM but with improvements that produced more accurate results. The algorithm uses two values of fuzziness to calculate the

Algorithm 3 IT2FCM Algorithm

```
1: procedure IT2FCM($c$, $m_1$,$m_2$, $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster' centers
4:     Sort data in an ascending order ($X_i - 1 < X_i < X_{i+1}$)
5:     Set the loop counter $k = k + 1$
6:     repeat
7:         Calculate the upper and lower memberships according to Equations 5 and 6
8:         Calculate the objective function $J_k$ according to Equation 7
9:         Get $index_C$ = index of a cluster center from the sorted list
10:        Calculate the cluster centers' vectors $C(k) = [C_i]$ according to Equation 10
11:        Defuzzification membership for a hard partition using Equation 11
12:     until
13:     $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
14:     Do Segmentation (Alg. 5)
15: end procedure
```

membership of data points. Two membership values are computed which are: the upper and lower membership for each cluster's center. These two membership values are calculated using Eq. 5 after sorting data points in an ascending order. The centroid values of clusters are then updated using these two memberships. The authors used two values for each center (V_{left} and V_{Right}). The average of the left and right centroid values is the value of the new cluster's center. The values (V_{left} and V_{Right}) are calculated as follows. First, find the index value of the old cluster's center K after sorting all data points in an ascending order. Then calculate the center using the same equation of FCM, which is Eq. 1. If index of a point K, then $u_{ij} = \overline{u_{ij}}$, otherwise $u_{ij} = \underline{u_{ij}}$. Equation 6 is used in the algorithm as shown in Algorithm 3.

$$\overline{u_{ij}} = \begin{cases} \frac{1}{\sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right)^{\frac{2}{m_1 - 1}}}, & if \sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right) \leq P \\ \frac{1}{\sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right)^{\frac{2}{m_2 - 1}}} & \text{Otherwise} \end{cases} \tag{5}$$

$$\underline{u_{ij}} = \begin{cases} \frac{1}{\sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right)^{\frac{2}{m_1 - 1}}}, & if \sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right) \geq P \\ \frac{1}{\sum_{k=1}^{c}\left(\frac{\|X_i - C_j\|}{\|X_i - C_k\|}\right)^{\frac{2}{m_2 - 1}}} & \text{Otherwise} \end{cases} \tag{6}$$

where $P = \frac{j}{255}$, j is the index of a cluster, and i is the index of point

The objective function is calculated using Eq. 7. Clusters' centers are updated using Eqs. 8 and 9.

$$J_{\text{m}} = \sum_{k=1}^{C}\sum_{i=1}^{N} \frac{x_i}{\overline{u_{ik}}} + \frac{x_i}{\underline{u_{ik}}} \tag{7}$$

$$V_{\text{Left}} = \frac{\sum_{i=1}^{N} u_{ij}^{m} \times x_i}{\sum_{i=1}^{N} u_{ij}^{m}} \tag{8}$$

where $u_{ij} = \overline{u_{ij}}$ if the value of $i \le K$, and K is the index of center value after sorting the data points. Otherwise, $u_{ij} = \underline{u_{ij}}$.

$$V_{\text{Right}} = \frac{\sum_{i=1}^{N} u_{ij}^{m} \times x_i}{\sum_{i=1}^{N} u_{ij}^{m}}, \tag{9}$$

where $u_{ij} = \underline{u_{ij}}$ if the value of $i \le K$, and K is the index of center value after sorting the data points. Otherwise, $u_{ij} = \overline{u_{ij}}$. Finally, the new center values are calculated using Eq. 10:

$$C(k) = \frac{V_{\text{Left}} + V_{\text{Right}}}{2} \tag{10}$$

$$u_{ij} = \frac{\overline{u_{ij}} + \underline{u_{ij}}}{2} \tag{11}$$

3.1.4 Modified interval type-2 fuzzy C-mean

Pixels feathers in FCM and IT2FCM algorithms are supposed to be independent of each other. However, in medical images, pixels feathers are dependent, especially the neighboring pixels. Qiu et al. in [36] modified the IT2FCM algorithm in [25] by adding a feature of dependency for neighboring pixels, which produced better accuracy. The authors used a local spatial interaction between adjacent pixels. They used 3×3 local window to simplify the computation of dependency and named their algorithm modified interval type-2 fuzzy C-mean (MIT2FCM).

In MIT2FCM, the upper and lower memberships were used similar to [25]. The new step that was introduced is a local spatial of adjacent pixels. It is calculated using Eq. 12, where $u_{ij} = \frac{\overline{u_{ij}} + \underline{u_{ij}}}{2}$. Equations 13 and 14 are used in MIT2FCM as shown in Algorithm 4.

$$u_{i\text{spatial}} = \left(\sum_{l=1}^{N_k} \frac{1}{d^2(d_l, d_k)^{\frac{1}{m-1}}} \right)^{-1} \times \left(\sum_{a=1}^{N} \frac{u_{ij}}{d^2(x_a, x_k)^{\frac{1}{m-1}}} \right) \tag{12}$$

$$\overline{u_{ij}} = \begin{cases} \dfrac{1}{\sum_{k=1}^{c} \left(\frac{\|d_{ij}^*\|}{\|d_{ik}^*\|} \right)^{\frac{2}{m_1-1}}}, & \text{if } \sum_{k=1}^{c} \left(\frac{\|d_{ij}^*\|}{\|d_{ik}^*\|} \right) \le C \\ \dfrac{1}{\sum_{k=1}^{c} \left(\frac{\|d_{ij}^*\|}{\|d_{ik}^*\|} \right)^{\frac{2}{m_2-1}}} & \text{Otherwise} \end{cases} \tag{13}$$

Algorithm 4 MIT2FCM Algorithm

```
1: procedure MIT2FCM($c$, $m_1$,$m_2$ , $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Sort data in an ascending order ($X_i - 1 < X_i < X_{i+1}$)
5:     Set the loop counter $k = k + 1$
6:     repeat
7:         Calculate the upper and lower memberships according to Equations 5 and 6
8:         Calculate the spatial membership according to Equation 12
9:         Update the upper and lower memberships according to Equations 13 and 14
10:         Calculate the objective function $J_k$ according to Equation 7
11:         Get $index_C$ = index of cluster center from the sorted list
12:         Calculate the cluster centers' vectors $C(k) = [C_i]$ according to Equation 10
13:         Defuzzification membership for hard partition using Equation 11
14:     until
15:     $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
16:     Do Segmentation (Alg. 5)
17: end procedure
```

Algorithm 5 CPU version of the Do Segmentation Function

```
1: procedure DO SEGMENTATION(Points, Clusters)
2:     Create images same as the number of clusters
3:     for <All points> do
4:         for <All clusters> do
5:             if $Points.ClustersIndex == Clusters.ID$ then
6:                 Put pixel value on image.ID
7:             end if
8:         end for
9:     end for
10: end procedure
```

$$\underline{u_{ij}} = \begin{cases} \dfrac{1}{\sum_{k=1}^{c}\left(\dfrac{\left\|d_{ij}^*\right\|}{\left\|d_{ik}^*\right\|}\right)^{\frac{2}{m_1-1}}}, & \text{if } \sum_{k=1}^{c}\left(\dfrac{\left\|d_{ij}^*\right\|}{\left\|d_{ik}^*\right\|}\right) \geq C \\ \dfrac{1}{\sum_{k=1}^{c}\left(\dfrac{\left\|d_{ij}^*\right\|}{\left\|d_{ik}^*\right\|}\right)^{\frac{2}{m_2-1}}} & \text{Otherwise} \end{cases} \tag{14}$$

where $d_{ik}^* = \frac{d_{ik}^2}{u_{i\text{Spatial}}}$.

3.2 Parallel version of the FCM algorithms

This section presents the hybrid CPU–GPU implementation of the four segmentation algorithms: FCM, T2FCM, IT2FCM, and MIT2FCM. The hybrid strategy of using the CPU and GPU together provides a powerful tool for programmers to achieve efficient computation [35]. In the four cases, our parallel implementations improve the execution time over the sequential implementations.

The membership function is implemented in a parallel fashion to execute on the GPU card. Also, the *Do Segmentation Function* was parallelized to run on the GPU side as shown in Algorithm 6. This function is used to update and segment the image pixels based on the strength of the membership value compared to the center of a cluster.

The calculations of the centroids and the objective functions are executed on the CPU side, because they need to calculate the summation as shown in Eqs. 1 and 3. The summation operation needs to run a number of iterations equal to the number of points multiplied by the number of clusters. If we parallelize this using threads, synchronization of all threads is needed. However, thread synchronization produces high delay on the GPU. Hence, performing this operation on the CPU is faster. Also, transferring data back and forth between the GPU and the CPU memories is avoided. In our previous work in [39], a parallel version without this CPU optimization achieved a speedup of 6×. However, in this work, we improved it by 9× after using optimization techniques as follows. In this work, we performed some mathematical operation in a different way of implementation. For example, the square function can be represented as a multiplication operation. Another optimization is to store the objective value using another variable. When the the operation needs the new value, we used the XOR

Algorithm 6 GPU version of the Do Segmentation Function

```
1: procedure DO SEGMENTATION(Points, Clusters)
2:     Create images as the number of clusters (Shared Memory)
3:     Pixel ID = Thread.ID
4:     for <All clusters> do
5:         if $Points.ClustersIndex == Clusters.ID$ then
6:             Put pixel value on image.ID
7:         end if
8:     end for
9: end procedure
```

Algorithm 7 Hybrid CPU–GPU FCM Algorithm

```
1: procedure FCM($c, m, \epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Initialize cluster centroids, data points and memberships matrix on GPU
5:     Initialize the membership matrix $U_{ij}$ according to Equation 2 (GPU)
6:     Transfer memberships from GPU
7:     repeat
8:         Calculate the objective function $J^k$ according to Equation 3
9:         Set the loop counter $k = k + 1$
10:         Calculate the cluster centers' vectors $C_k = [C_i]$ according to Equation 1
11:         Transfer centroids to GPU
12:         Initialize the membership matrix $U_{ij}$ according to Equation 2 (GPU)
13:         Transfer memberships from GPU
14:         Calculate the objective function $J^(k + 1)$ according to Equation 3
15:     until
16:     $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
17:     Do Segmentation on GPU (Alg. 6)
18: end procedure
```

operation to swap between the two values of the objective function. XOR operation increased the improvement by 1×. Multiplication operation increased the performance by 2×. Hence, the new version of parallel FCM is 9× faster than the sequential version. Those two optimization techniques were used for the four parallel versions presented in this paper.

As shown in Eq. 2, the membership function has a summation operation that iterates same as the number of clusters. Also, each pixel needs to calculate the Euclidean distance between itself and clusters' centroids. This can be done separately. Hence, we improved the T2FCM algorithm performance by calculating the membership U in Eq. 2, the new membership A in Eq. 4, and do segmentation function on the GPU side. The new parallel version of T2FCM is shown in Algorithm 8.

Algorithm 8 Hybrid CPU–GPU T2FCM Algorithm

```
1: procedure T2FCM($c$, $m$, $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Initialize the membership matrix $U_{ij}$ according to Equation 2 (GPU)
5:     Calculate the membership matrix $A_{ij}$ according to Equation 4 (GPU)
6:     Transfer memberships $A_{ij}$ from GPU
7:     Calculate the objective function $J^k$ according to Equation 3
8:     repeat
9:         Set the loop counter $k = k + 1$
10:        Calculate the cluster centers vectors $C_k$ according to Equation 1
11:        Transfer centroids to GPU
12:        Initialize the membership matrix $U_{ij}$ according to Equation 2 (GPU)
13:        Calculate the membership matrix $A_{ij}$ according to Equation 4 (GPU)
14:        Transfer memberships $A_{ij}$ from GPU
15:        Calculate the objective function $J^{k+1}$ according to Equation 3 and replace $U_{ij}$ with $a_{ij}$
16:     until
17:     $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
18:     Do Segmentation on GPU (Alg. 6)
19: end procedure
```

For the hybrid CPU–GPU implementation of the IT2FCM algorithm, all functions that calculate the membership values were converted to a parallel implementation. This algorithm needs to perform data sorting at the beginning of the code. We used a build-in sorting algorithm on the CPU side for two reasons: first, it runs only once at the beginning of the code. Second, the sorting is performed on a data structure type that contains many attributes such as: X-axis, Y-axis, Red, Green, Blue, and alpha values for each pixel point. Hence, if we are to create memory allocation on the GPU memory, it would have required to transfer more than one memory allocation which produced a high delay in time. For the aforementioned two reasons, the sorting algorithm is performed on the CPU side.

Also, at each iteration of the IT2FCM and MIT2FCM algorithms, a search for the index value of a cluster's center is needed. Thus, we used binary search algorithm which is fast for sorted data.

The MIT2FCM algorithm is similar to the IT2FCM algorithm except for the filtration operation. Hence, we used the same hybrid CPU–GPU implementation of

Algorithm 9 Hybrid CPU–GPU IT2FCM Algorithm

```
1: procedure IT2FCM($c$, $m_1$,$m_2$, $\epsilon$)
2:     Set the number of clusters $C$, the fuzziness parameter $m$ and the termination criterion $\epsilon$, set $k = 0$
3:     Initialize random cluster centers
4:     Sort data in ascending order ($X_i - 1 < X_i < X_{i+1}$).
5:     Set the loop counter $k = k + 1$
6:     repeat
7:         Calculate the Euclidean distance $d_{ij}$ (GPU)
8:         Calculate the upper and lower memberships according to Equations 5 and 6(GPU)
9:         Transfer the upper and lower memberships from GPU
10:         Calculate the objective function $J_k$ according to Equation 7
11:         Get $index_C$ = index of cluster center from the sorted list.
12:         Calculate the cluster centers vectors $C(k) = [C_i]$ according to Equation 10(Binary Search)
13:         Defuzzification membership for hard partition using Equation 11(GPU)
14:     until
15:     $\left\| J^{(k+1)} - J^k \right\| < \epsilon$
16:     Do Segmentation on GPU (Alg. 6)
17: end procedure
```

Table 1 Hybrid CPU–GPU functions

Algorithm	CPU-functions	GPU-functions
FCM	Calculate centroid	Calculate membership
	Calculate objective function	Do segmentation
T2FCM	Calculate centroid	Calculate membership
	Calculate objective function	Do Segmentation
		Calculate a_{ij} membership
IT2FCM	Calculate centroid	Calculate membership
	Calculate objective function	Do segmentation
	Build-in sort algorithm	Calculate upper and lower membership
IT2MFCM	Calculate centroid	Calculate membership
	Calculate objective function	Do segmentation
	Build-in sort algorithm	Calculate upper and lower membership
		Calculate spatial membership
		Filtering operation (mean filter)

IT2FCM with a parallel modified membership function of IT2FCM. This step is a filtering operation and the GPU can execute it faster than the CPU because it is parallelizable. The mean filter is used which updates the value of a pixel using the average of neighborhood pixels. Table 1 shows the list of functions of MIT2FCM and their locations whether on the GPU side or the CPU side.

In calculating the objective value for FCM, T2FCM, IT2FCM, and MIT2FCM, we compute the exponential function on the CPU side by performing multiplication if the power is two. Also, for IT2FCM and MIT2FCM, we calculate the value $\sum_{k=1}^{C} \frac{d_{ij}}{d_{ik}}$ in Eq. 5 only one time and then saved it in a memory array for later use when calculating

Table 2 Percentage of GPU utilization

Number of threads	1.0	1.1	1.2	1.3	2.0	2.1	3.0
64	67	67	50	50	33	33	50
96	100	100	75	75	50	50	75
128	100	100	100	100	67	67	100
192	100	100	94	94	100	100	94
256	100	100	100	100	100	100	100
384	100	100	75	75	100	100	94
512	67	67	100	100	100	100	100
768	N/A	N/A	N/A	N/A	100	100	75
1024	N/A	N/A	N/A	N/A	67	67	100

the upper and lower values in: $\sum_{k=1}^{C}\left(\frac{d_{ij}}{d_{ik}}\right)^{\frac{2}{m_1-1}}$. This optimization saved a large amount of time on the CPU side.

Memory optimization is another challenge that we faced in our parallel implementation. As mentioned in [20], the usage of memory is critical in GPU programming and can result in slow execution time. Hence, we calculate the memory size to be used using Eq. 15. This equation optimizes memory size usage and speeds up GPU computations. Also, large number of threads can slow down GPU computations when many threads try to read and/or write to the memory [19]. Also, since many models of NVIDIA GPU hardwares cannot support 1024 threads, we used a fixed number of threads of 256. We have used different numbers of threads as shown in Table 2. 256 threads produced the best GPU utilization as shown in the table.

$$\text{Number of blocks} = \text{ceil}\left(\frac{\text{size of data}}{\text{number of threads}}\right) \tag{15}$$

4 Results and discussion

This section presents the execution time comparison between the hybrid CPU–GPU implementation and the sequential implementation. We calculate the speedup using Eq. 16.

$$\text{Speedup} = \frac{\text{Sequential CPU}_{\text{TIME}}}{\text{Hybrid CPU GPU}_{\text{TIME}}} \tag{16}$$

1. Dataset

We have used one MR image and one mammography image for both FCM and T2FCM algorithms as in [32] and [1]. The MR is a brain image with size of 512 × 512. The Mammography images is also of size 512 × 512 and both images are gray-scale type.

2. Results

The GPU card that was used in this experiment is NVIDIA GT 740M with 2GB memory. The CPU is Intel core I7 with 6GB RAM. The platform is Windows 8.1 with C# and CUDA libraries installed. An integration library with Visual Studio 2013 was used to run C# on GPU [2]. Our implementations did not reduce the image segmentation accuracy in all cases. However, the speedup is improved. The number of clusters that was used is five.

(a) Results of MR and mammography images dataset:

Table 3 shows the experiments of sequential FCM and our hybrid FCM. We ran our experiments for ten times and the average CPU time is 167.4988 s (sequential version). However, the average execution time for our hybrid CPU–GPU is 16.8915 s for MR images. The results of the two FCM versions are the same with respect to the segmented image and accuracy. Table 4 shows the results for T2FCM algorithm versions using the same image that was used in the FCM experiment. The average CPU time is 134.5872 s and the average for hybrid CPU–GPU time is 21.993 s.

As shown in Tables 5 and 6, IT2FCM algorithm, using MR images, produced an average CPU time of 428.1371 s for the sequential version and 51.4126 s in the case of the hybrid CPU–GPU. The MIT2FCM results show 316.8554 s for the CPU average execution time and 24.0313 s for the hybrid CPU–GPU with a speedup of more than 13×. This is because of the filtering operation in MIT2FCM which improves the classification operation [46]. Figure 1 summarizes these results.

For the mammography images case, Tables 7, 8, 9, and 10 show the results for the four algorithms. The average FCM CPU time and average hybrid CPU–GPU time are 172.8874 and 20.3478 s, respectively. The average T2FCM CPU time and average hybrid CPU–GPU time are 166.3168 and 24.5534 s, respectively. The average IT2FCM CPU time and average hybrid CPU–GPU time are 461.6416 and 49.6946 s, respectively. The average MIT2FCM CPU time and average hybrid CPU–GPU time are 2427.1017 and 117.778 s, respectively with a speedup of more than 20×. Figure 2 summarizes these results.

Table 3 10 runs of FCM versions for the MR image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
25	533.017	24.765	0	0	25
17	154.603	13.323	0	0	17
26	152.734	20.159	0	0	26
16	94.303	12.589	0	0	16
15	88.463	11.801	0	0	15
15	88.137	11.734	0	0	15
18	105.179	14.1	0	0	18
20	117.364	15.651	0	0	20
26	151.85	20.24	0	0	26
32	189.338	24.553	0	0	32

Table 4 10 runs of T2FCM versions for the MR image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
29	169.557	27.946	0	0	29
31	182.443	29.802	0	0	31
31	181.731	29.706	0	0	31
18	106.219	17.494	0	0	18
32	188.842	30.379	0	0	32
28	165.616	26.938	0	0	28
20	119.872	19.359	0	0	20
18	106.349	17.522	0	0	18
4	25.185	4.263	0	0	4
17	100.058	16.521	0	0	17

Table 5 10 runs of IT2FCM versions for the MR image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
19	104.682	13.139	0	0	19
100	546.691	65.520	224.3910	224.391	100
100	551.144	65.459	224.391	224.391	100
100	552.454	65.829	104.003	104.003	100
100	543.722	65.967	175.404	175.404	100
100	548.036	65.310	220.402	220.402	100
100	542.912	66.152	316.695	316.695	100
100	554.082	65.423	257.285	257.285	100
27	151.697	18.208	4.197	4.197	27
34	185.951	23.119	4.197	4.197	34

Table 6 10 runs of MIT2FCM versions for the MR image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
10	212.611	16.416	0	0	10
14	292.634	21.797	0	0	14
7	154.956	11.639	0	0	7
25	502.081	37.78	3070.485	3070.485	25
20	404.757	31.21	3070.485	3070.485	20
21	425.397	31.965	3070.485	3070.485	21
20	406.137	31.243	3071.822	3071.822	20
16	327.118	24.825	0	0	16
12	249.774	18.934	0	0	12
9	193.089	14.504	0	0	9

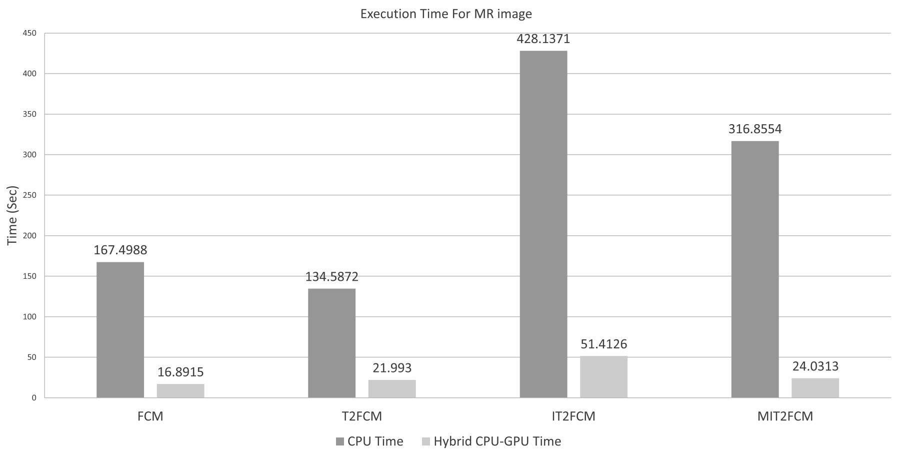


Fig. 1 Execution time of CPU and hybrid CPU–GPU for MR

Table 7 10 runs of FCM for the mammography image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
23	183.691	21.625	0	0	23
18	108.668	16.901	0	0	18
33	266.265	29.717	0	0	33
19	154.588	17.931	0	0	19
27	216.489	25.184	0	0	27
27	218.112	25.175	0	0	27
20	161.713	18.127	0	0	20
14	112.856	13.272	0	0	14
11	90.096	10.525	0	0	11
27	216.396	25.021	0	0	27

Table 8 10 runs of T2FCM for the mammography image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
18	146.017	20.962	0	0	18
29	210.226	33.526	0	0	29
17	139.108	20.343	0	0	17
24	193.153	28.261	0	0	24
13	105.753	15.841	0	0	13
25	201.85	29.632	0	0	25
25	202.837	29.711	0	0	25
16	130.534	19.146	0	0	16
30	243.734	34.574	0	0	30
11	89.956	13.538	0	0	11

Table 9 10 runs of IT2FCM for the mammography image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
15	121.762	13.375	0	0	15
67	519.677	57.952	0	0	67
49	383.201	42.231	0	0	49
100	755.311	85.037	266.288	266.288	100
42	290.659	36.326	75.804	75.804	42
34	264.648	29.455	0	0	34
56	427.841	43.428	0.788	0.788	56
100	767.241	75.542	0.730	0.730	100
48	335.846	36.719	413.882	413.882	48
100	750.23	76.881	41.761	41.761	100

Table 10 10 runs of MIT2FCM for the mammography image

CPU iterations	CPU time	GPU time	CPU precision	GPU precision	GPU iterations
100	2791.866	137.136	467.726	467.726	100
100	2769.488	122.189	74.329	74.329	100
100	2787.901	140.427	314.631	314.631	100
100	2802.546	135.223	75.424	75.424	100
100	2804.152	136.328	74.078	74.078	100
24	692.593	34.527	75.583	75.583	24
100	2805.223	138.944	74.673	74.673	100
100	2799.686	135.742	0.500	0.500	100
43	1218.326	60.627	75.583	75.583	43
100	2799.236	136.637	73.407	73.407	100

(b) Speedup discussion

From the execution time shown previously, we can see that the speedup improvement for the parallel FCM is almost 10× for the MR image and more than 8× for the mammogram image compared to the sequential algorithm. We have used optimization techniques to speed up the execution time by reducing the data transfer rate from and to the GPU memory. For example, we transfer cluster index values to the GPU one way, and we copy it when the code terminates to visualize the results in our application GUI. Also, T2FCM speedup is more than 6× by moving special functions to the GPU and keeping others on the CPU side where they execute faster. The IT2FCM algorithm achieved a speedup of more than 8× for the MR image and more than 9× in the case of mammogram image. Finally, MIT2FCM achieved more than 13× for the MR image and more than 20× for the mammogram image. Figure 3 summarizes these results.

The filter step increases the image segmentation accuracy. However, it incurs a high execution-time overhead of about 53.5 s. Since the filtering operation can be

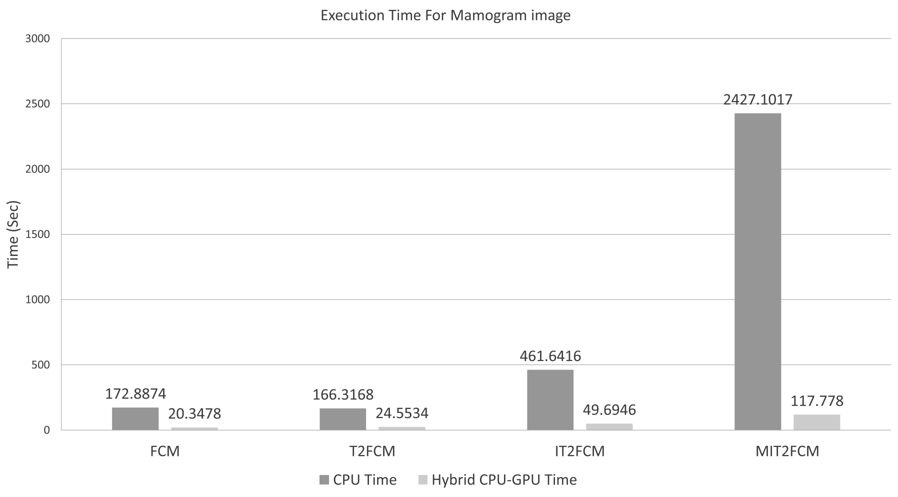


Fig. 2 Execution time of CPU and hybrid CPU–GPU for mamogram

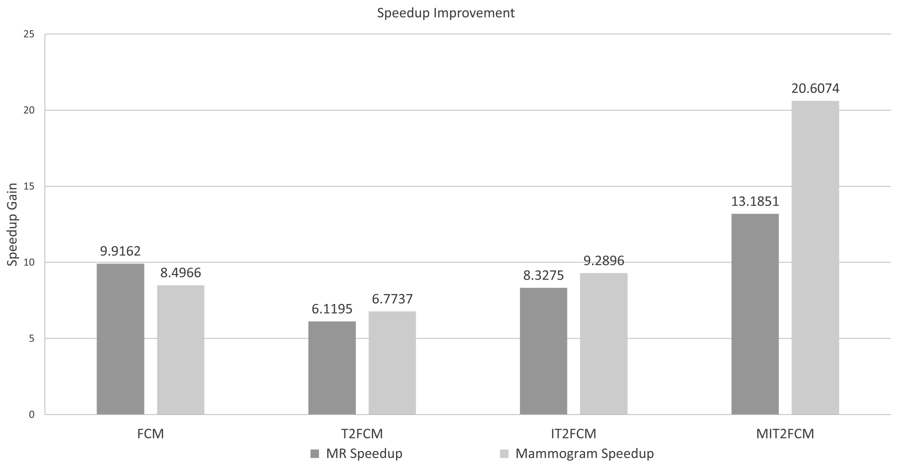


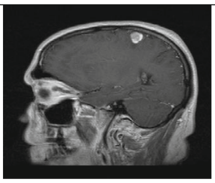
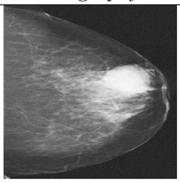
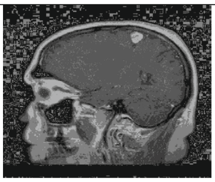
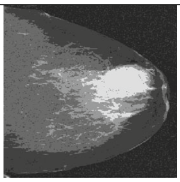
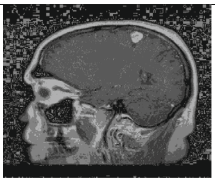
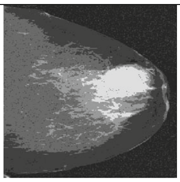
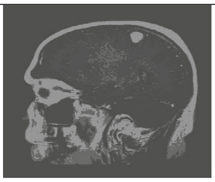
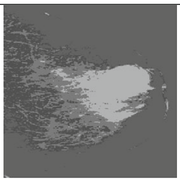
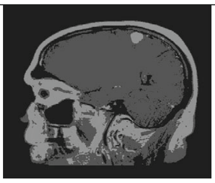
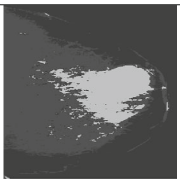
Fig. 3 Speedup gain of FCM algorithms

parallelized perfectly, we can move it to the GPU side which results in an overhead of only 0.38 s for the hybrid CPU–GPU versions.

Also, IT2FCM algorithm achieved higher speedup than the T2FCM because it uses two membership matrices (upper and lower memberships). Using one membership matrix in T2FCM results in more dependency for the data points. Consequently, calculating the memberships on the GPU side is better than the CPU side because of the low dependency between the elements of the matrices. The GPU can run the upper and lower matrices in parallel, so that each matrix is calculated separately.

In the last segmentation algorithm which is the MIT2FCM, the authors of the algorithm improved the accuracy of the segmentation process by adding a filtering step. However, this step increases the execution time. With the parallel implementation

Table 11 Examples of the accuracy for each FCM algorithm

Algorithm	MRI	Mammography
Original image		
FCM		
T2FCM		
IT2FCM		
MIT2FCM		

of the filtering operation on the GPU, we solved its problem of having high execution-time overhead. Our aforementioned discoveries of optimization techniques help the image processing community, especially if these algorithms are to be used in real-time processing where efficient computation is critical.

Table 11 shows sample examples of the images that were used in our experiments and the accuracy of each segmentation algorithm. Also, Tables 12 and 13 show sample clusters that were used in the experiments.

Table 12 5 clusters of MR images for FCM algorithms

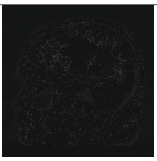
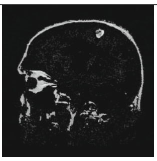
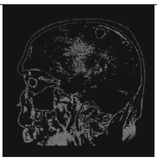
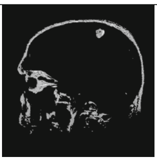
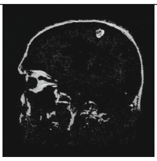
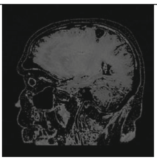
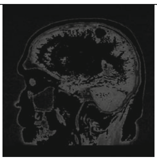
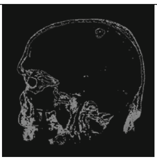
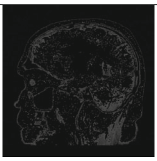
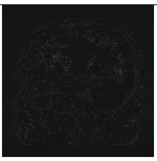
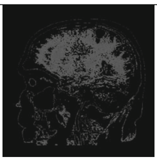
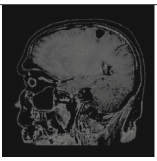
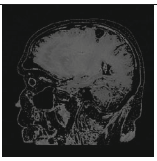
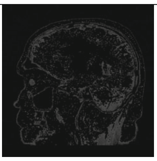
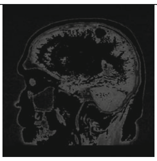
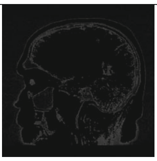
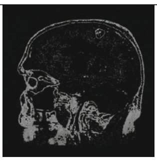
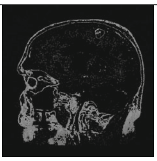
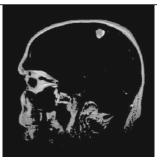
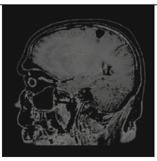
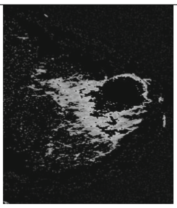
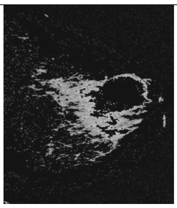
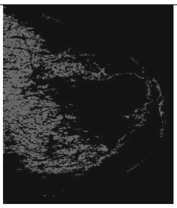
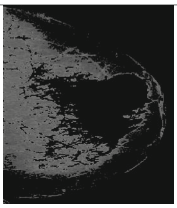
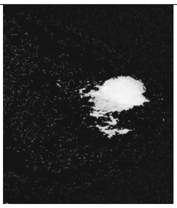
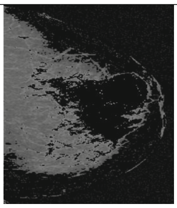
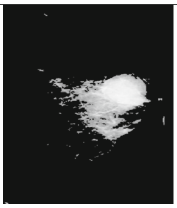
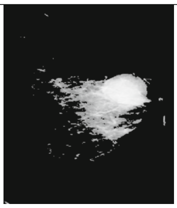
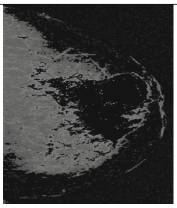
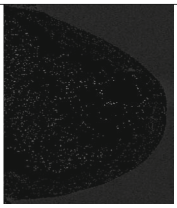
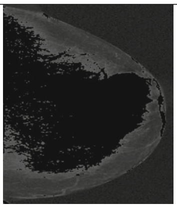
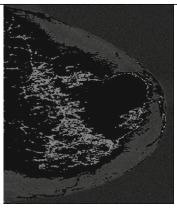
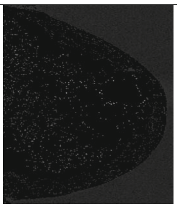
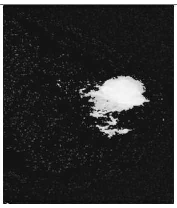
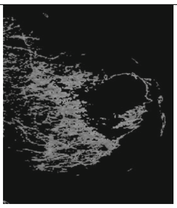
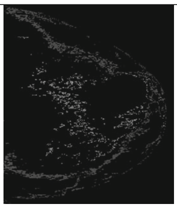
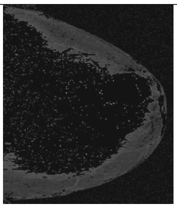
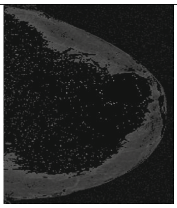
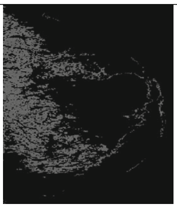
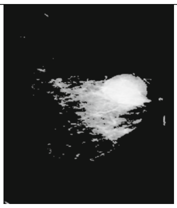
Algorithm type	FCM	T2FCM	IT2FCM	MIT2FCM
Cluster1				
Cluster2				
Cluster3				
Cluster4				
Cluster5				

Table 13 5 clusters of mammography images for FCM algorithms

Algorithm type	FCM	T2FCM	IT2FCM	MIT2FCM
Cluster1				
Cluster2				
Cluster3				
Cluster4				
Cluster5				

5 Conclusion

Image segmentation process is used to extract objects from images. Because of its importance, many methods were proposed in the literature to improve the segmentation process. In this paper, we studied four important clustering algorithms which are: FCM, T2FCM, IT2FCM, and MIT2FCM. Parallel implementations of the algorithms were developed and investigated to improve their execution time without penalizing the accuracy. GPU hardware was used to execute the parallel implementations and compared with sequential CPU-only implementations. We used MR and mammography images in conducting our experiments. Results show that speedup of 6× to 20× can be achieved with parallel hybrid CPU–GPU implementations. Also, we have discussed optimization steps in the algorithms to enhance their execution time. This is critical for real-time processing where efficient calculations are needed.

Acknowledgements This work is funded by Jordan University of Science and Technology (JO) (Grant No. 20160081).

References

1. Auntminnie (2016). http://www.auntminnie.com/index.aspx?sec=def
2. Cudafy.net (2016). https://cudafy.codeplex.com/
3. Adams R, Bischof L (1994) Seeded region growing. IEEE Trans Pattern Anal Mach Intell 16(6):641–647
4. Ahmed MN, Yamany SM, Mohamed N, Farag AA, Moriarty T (2002) A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data. IEEE Trans Med Imaging 21(3):193–199
5. Al-Ayyoub M, Abu-Dalo AM, Jararweh Y, Jarrah M, Al Sad M (2015) A GPU-based implementations of the fuzzy c-means algorithms for medical image segmentation. J Supercomput 71(8):3149–3162
6. Al-Ayyoub M, Qussai Y, Shehab MA, Jararweh Y, Albalas F (2016) Accelerating clustering algorithms using GPUs. In: Conference: 2016 IEEE High Performance Extreme Computing Conference (HPEC-2016), p 1. IEEE
7. Alsmirat MA, Jararweh Y, Al-Ayyoub M, Shehab MA, Gupta BB (2016) Accelerating compute intensive medical imaging segmentation algorithms using hybrid CPU–GPU implementations. Multimed Tools Appl. doi:10.1007/s11042-016-3884-2
8. Arabnia H, Oliver M (1987) Arbitrary rotation of raster images with SIMD machine architectures. Comput Graph Forum 6(1):3–11
9. Arabnia HR (1990) A parallel algorithm for the arbitrary rotation of digitized images using process-and-data-decomposition approach. J Parallel Distrib Comput 10(2):188–192
10. Arabnia HR, Bhandarkar SM (1996) Parallel stereocorrelation on a reconfigurable multi-ring network. J Supercomput 10(3):243–269
11. Arabnia HR, Oliver MA (1986) Fast operations on raster images with SIMD machine architectures. Comput Graph Forum 5(3):179–188
12. Arabnia HR, Oliver MA (1987) A transputer network for the arbitrary rotation of digitised images. Comput J 30(5):425–432
13. Begum SA, Devi OM (2012) A rough type-2 fuzzy clustering algorithm for mr image segmentation. Int J Comput Appl 54(4):4–11
14. Bezdek JC, Ehrlich R, Full W (1984) FCM: the fuzzy c-means clustering algorithm. Comput Geosci 10(2–3):191–203
15. Bhandarkar S, Arabnia H (1995) The Hough transform on a reconfigurable multi-ring network. J Parallel Distrib Comput 24(1):107–114
16. Bhandarkar SM, Arabnia HR (1995) The refine multiprocessor theoretical properties and algorithms. Parallel Comput 21(11):1783–1805
17. Bhandarkar SM, Arabnia HR, Smith JW (1995) A reconfigurable architecture for image processing and computer vision. Int J Pattern Recogn Artif Intell 09(02):201–229
18. Cheng H, Shan J, Ju W, Guo Y, Zhang L (2010) Automated breast cancer detection and classification using ultrasound images: a survey. Pattern Recogn 43(1):299–317
19. Cheng J, Grossman M, McKercher T (2014) Professional CUDA C programming. Wiley, New York
20. Cook S (2012) CUDA programming: a developer's guide to parallel computing with GPUs. Morgan Kaufmann, Newnes
21. Doi K (2005) Current status and future potential of computer-aided diagnosis in medical imaging. Br J Radiol 78(suppl_1):s3–s19
22. Eklund A, Paul Dufort DF, LaConte SM (2013) Medical image processing on the GPU past, present and future. Med Image Anal 17(8):01–22

23. Rhee FCH, Hwang C (2001) A type-2 fuzzy c-means clustering algorithm. In: IFSA World Congress and 20th NAFIPS International Conference, 2001. Joint 9th, vol 4, pp 1926–1929
24. Gauge C (2016) Fuzzy c-mean algorithm (2016). http://www.codeproject.com/Articles/91675/Computer-Vision-Applications-with-C-Fuzzy-C-means
25. Hwang C, Rhee FCH (2007) Uncertain fuzzy clustering: interval type-2 fuzzy approach to c-means. IEEE Trans Fuzzy Syst 15(1):107–120
26. İçer S (2013) Automatic segmentation of corpus collasum using Gaussian mixture modeling and fuzzy c means methods. Comput Methods Progr Biomed 112(1):38–46
27. Jafri R, Ali SA, Arabnia HR (2013) Computer vision-based object recognition for the visually impaired using visual tags. In: Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), p 1. The Steering Committee of The World Congress in Computer Science, Computer Engineering and Applied Computing (WorldComp)
28. Jafri R, Ali SA, Arabnia HR, Fatima S (2014) Computer vision-based object recognition for the visually impaired in an indoors environment: a survey. Vis Comput 30(11):1197–1222
29. Jafri R, Arabnia HR (2008) Fusion of face and gait for automatic human recognition. In: 5th International Conference on Information Technology: New Generations, 2008, ITNG 2008, pp 167–173. IEEE
30. Ji Z, Xia Y, Sun Q, Chen Q, Feng D (2014) Adaptive scale fuzzy local Gaussian mixture model for brain MR image segmentation. Neurocomputing 134:60–69
31. McAuliffe MJ, Lalonde FM, McGarry D, Gandler W, Csaky K, Trus BL (2001) Medical image processing, analysis and visualization in clinical research. In: 14th IEEE Symposium on Computer-Based Medical Systems 2001. CBMS 2001. Proceedings, pp 381–386
32. Michel K (2016) Parasitology research (2016). https://www.k-state.edu/parasitology/
33. Olabarriaga S, Smeulders A (2001) Interaction in the segmentation of medical images: a survey. Med Image Anal 5(2):127–142
34. Pan L, Gu L, Xu J (2008) Implementation of medical image segmentation in cuda. In: 2008 International Conference on Information Technology and Applications in Biomedicine, pp 82–85. IEEE
35. Papadrakakis M, Stavroulakis G, Karatarakis A (2011) A new era in scientific computing: domain decomposition methods in hybrid CPU–GPU architectures. Comput Methods Appl Mech Eng 200(13):1490–1508
36. Qiu C, Xiao J, Yu L, Han L, Iqbal MN (2013) A modified interval type-2 fuzzy c-means algorithm with application in MR image segmentation. Pattern Recogn Lett 34(12):1329–1338
37. Rowińska Z, Gocławski J (2012) Cuda based fuzzy c-means acceleration for the segmentation of images with fungus grown in foam matrices. Image Process Commun 17(4):191–200
38. Severance C (2010) High performance computing, an open textbook
39. Shehab MA, Al-Ayyoub M, Jararweh Y (2015) Improving fcm and T2FCM algorithms performance using GPUS for medical images segmentation. In: 2015 6th International Conference on Information and Communication Systems (ICICS), pp 130–135. IEEE
40. Shih FY, Cheng S (2005) Automatic seeded region growing for color image segmentation. Image Vis Comput 23(10):877–886
41. Sonka M, Hlavac V, Boyle R (2014) Image processing, analysis, and machine vision. Cengage Learning. ISBN-10: 1133593607
42. Tan KS, Isa NAM (2011) Color image segmentation using histogram thresholding fuzzy c-means hybrid approach. Pattern Recogn 44(1):1–15
43. Tang J (2010) A color image segmentation algorithm based on region growing. In: 2010 2nd International Conference on Computer Engineering and Technology (ICCET), vol 6, pp V6–634. IEEE
44. Ugarriza LG, Saber E, Vantaram SR, Amuso V, Shaw M, Bhaskar R (2009) Automatic image segmentation by dynamic region growth and multiresolution merging. IEEE Trans Image Process 18(10):2275–2288
45. Walters JP, Balu V, Kompalli S, Chaudhary V (2009) Evaluating the use of gpus in liver image segmentation and hmmer database searches. In: IEEE International Symposium on Parallel Distributed Processing, 2009. IPDPS 2009, pp 1–12. IEEE
46. Wang H, Fei B (2009) A modified fuzzy c-means classification method using a multiscale diffusion filtering scheme. Med Image Anal 13(2):193–202
47. Wani MA, Arabnia HR (2003) Parallel edge-region-based segmentation algorithm targeted at reconfigurable multiring network. J Supercomput 25(1):43–62