GPU-Based Fuzzy C-Means Clustering Algorithm for Image Segmentation

Mishal Almazrooie[1], Mogana Vadiveloo[1], and Rosni Abdullah[*1,2]

[1]School of Computer Sciences, Universiti Sains Malaysia, 11800 USM, Pulau Pinang, Malaysia
[2]National Advanced IPv6 Center (NaV6), Universiti Sains Malaysia, 11800 USM, Pulau Pinang, Malaysia

January 7, 2016

Abstract

In this paper, a fast and practical GPU-based implementation of Fuzzy C-Means (FCM) clustering algorithm for image segmentation is proposed. First, an extensive analysis is conducted to study the dependency among the image pixels in the algorithm for parallelization. The proposed GPU-based FCM has been tested on digital brain simulated dataset to segment white matter(WM), gray matter(GM) and cerebrospinal fluid (CSF) soft tissue regions. The execution time of the sequential FCM is 2798 seconds for an image dataset with the size of 1MB. While the proposed GPU-based FCM requires only 4.2seconds for the similar size of image dataset. An estimated 674-fold superlinear speedup is measured for the data size of 700 KB on a CUDA device that has 448 processors.

Superlinear speedup, Fuzzy C-Means, Parallel algorithms, Graphic Processing Units (GPUs), CUDA

1 Introduction

Image segmentation has been one of the fundamental research areas in image processing. It is a process of partitioning a given image into desired regions according to the chosen image feature information such as intensity or texture. The segmentation is used with application in the field of medical imaging, tumors locating and diagnosis. Over the past few decades, as image segmentation has gained much interest, various segmentation techniques have been proposed, each of which uses different induction principle.

Clustering is one of the most popular techniques used in image segmentation. In clustering, the goal is to produce coherent clusters of pixels [1]. The pixels in a cluster are as similar as possible with respect to the selected image feature information. While the pixels belong in the adjacent clusters are significantly different with respect to the same selected image feature information [1]. There are variants of clustering algorithms have been used widely in image segmentation and they are K-Means [2], Fuzzy C-Means (FCM) [3], and ISODATA [4].

In the last decades, FCM has been very popularly used to solve the image segmentation problems [5]; [6]. It is a fuzzy clustering method that allows a single pixel to belong to two or more clusters. The introduction of fuzziness makes this algorithm to able to retain more information from the original image than the crisp or hard clustering algorithms [5]; [6]. However this sequential FCM becomes computationally intensive when segmenting large image datasets [6]. In such a case, the algorithm becomes very inefficient.

One-way to improve the performance of the FCM clustering algorithm is to use parallel computing methods. Initially, Graphic Processing Units (GPUs) were specific-purpose processors that only manipulate and accelerate the creation of images intended for output to a display. However, GPUs have recently shifted to general-purpose processors (GPGPUs) to solve general concerns, such as scientific and engineering problems. Data parallelism on a GPU is a powerful parallel model. In this paper, a fast and practical parallel FCM approach on GPGPU is presented and discussed.

This paper is organized as follows: Section 2 provides a background of FCM algorithm and the paral-

lel technology used. Section 3 presents related works. The proposed method is explained in detail in Section 4. The experimental results are presented and discussed in Section 5. Finally, Section 6 provides the conclusion and suggestions for future works.

2 Preliminaries

In the first sub-section, a brief introduction on Fuzzy C-Means (FCM) algorithm is presented. While in the following sub-section the parallel technology used in this work namely on General Purpose Computing on Graphics Processing Units (GPGPU) data parallelism is discussed.

2.1 Fuzzy C-Means Algorithm

Fuzzy C-Means was developed by [3]. It is an iterative optimization that minimizes the objective function defined in 1. The objective function consists of two main components u and v. u_{ij} is the membership function of a pixel, x_i. It represents the probability that x_i may belong to a cluster. The u_{ij} is dependent on the distance function, d_{ij}. d_{ij} is the Euclidean distance measure between the pixel x_i and each cluster center, v_j, $d_{ij} = ||x_i - v_j||$. m is a constant that represents the fuzziness value of the resulting clusters that are to be formed; $1 \leq m \leq \infty$.

$$J_i = \sum_{i=1}^{N} \sum_{j=1}^{c} u_{ij}^m ||x_i - v_j||^2 \qquad (1)$$

with respect to:

$$\sum_{j=1}^{c} u_{ij} = 1, 1 \leq i \leq n$$

$$0 < \sum_{i=1}^{n} u_{ij} < n, 1 \leq j \leq c \qquad (2)$$

$$\sum_{i=1}^{c} \sum_{i=1}^{n} u_{ij} = n.$$

In image clustering, the most commonly used feature is the grey level or intensity value of the image being segmented. Therefore, the objective function, j_m in 1 is minimized when higher membership value is assigned to pixels with intensity values close to a cluster center of the corresponding cluster, while lower membership value is assigned to pixels whose intensities are far from the cluster center.

$$v_i = \frac{\sum_{i=1}^{N} u_{ij}^m . x_i}{\sum_{i=1}^{N} u_{ij}^m} \qquad (3)$$

$$u_{ij} = \frac{1}{\sum_{k=1}^{c} \left[\frac{||x_i - v_j||}{||x_i - v_k||} \right]^{\frac{2}{m-1}}} \qquad (4)$$

Starting with random initialization of the membership values for each pixel from the manually selected clusters, the clusters are converged by recursively updating the cluster centers and membership function in 3 and 4. This is to minimize the objective function in 1. Convergence stops when the overall difference in the membership function between the current and previous iteration is smaller than a given epsilon value, ε. After the convergence, deffuzzifaction is applied. Each pixel is assigned to a specific cluster according to the maximal value of its membership function. The steps of the Fuzzy C-Means algorithm are illustrated in Algorithm 1.

Algorithm 1: FUZZY C-MEANS ALGORITHM

Assumptions: Image is transformed into feature space.

Step 1: Initialize the number of clusters c, $m = 2$, and $\varepsilon = 0.005$

Step 2: Initialize the membership function, u_{ij} randomly.

Step 3: **repeat**

   | Update the cluster center, v_i using Equation 3

   | Update the membership function u_{ij} using Equation 4

until $||u_{ij}^{k+1} - u_{ij}^k|| < \varepsilon$

2.2 Data Parallelism on GPGPU

Initially, GPU was a hardware equipped with a processor specifically designed to accelerate graphic processing. Eventually, GPU applications were extended to general-purpose computations. At present, GPGPU is used in many applications typically performed using a CPU, such as analytic, engineering, and scientific applications [7]. With the release of the massively parallel architecture called CUDA in 2007 from NVIDIA, GPUs have become widely accessible [8].

A GPU is a processor or a multiprocessor device that has hundreds or even thousands of cores called scalar processors (SPs), which are arranged in groups

named streaming multiprocessors (SMs), as shown in the left side of Fig. 1. Moreover, GPUs have different kinds of memories: global, local, texture, constant, shared, and register memories. Global, constant, and texture memories are accessible to all threads in the grid. Shared memory is visible to threads within one CUDA block. It is faster than the global memory but is limited by size. Register memory is visible to the thread that initialized the said memory and lasts for the lifetime of that thread.

CUDA is the parallel programming model used for NVIDIA GPGPUs. CUDA can increase the performance by harnessing the power of a GPU device. Thousands of threads can be executed concurrently using CUDA on GPGPU. The execution model of CUDA on NVIDIA devices is shown in Fig. 1.
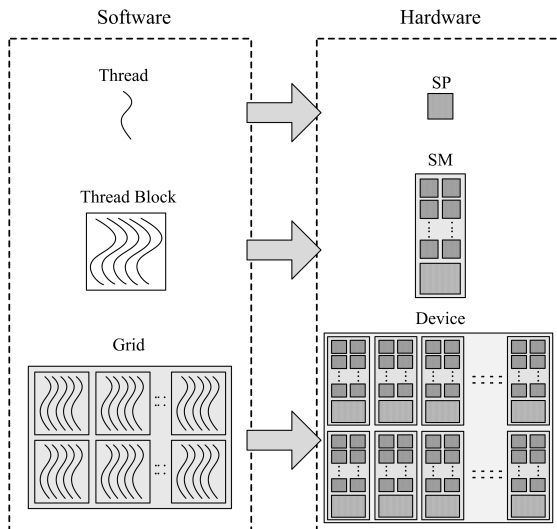

Figure 1: CUDA Execution Model on GPGPU

3   Related works

Li et al. proposed an Fuzzy C-Means (FCM) algorithm based on GPU [9]. They modified the sequential FCM algorithm, such that the calculations of the membership and cluster center matrices are not comparable to the sequential one. They have FCM on GPU using CUDA. The empirical results obtained by Li et al. showed that the proposed parallel FCM on GPU is more efficient than the sequential FCM. Instead of efficiency, they claimed that the proposed method exhibits improvement in the quality of the GPU segmented image. The authors achieved a 10-fold speedup with the proposed parallel FCM on NVIDIA GTX 260 device compared with the sequential FCM for natural images sized from 53kb to 101kb.

Mahmoud et al. presented a GPU-based br-FCM for medical images segmentation [10]. The br-FCM is a faster variant of the sequential FCM [11]. The GPU-based brFCM is implemented on different GPGPU cards. Mahmoud et al showed that the GPU-based brFCM has a significant improvement over the parallel FCM in [30]. The achieved speedup is up tp 23.42 fold faster than parallel FCM in [30] for medical images of 350x350 and 512x512 dimensions.

Shalom et al. proposed a scalable FCM based on graphic hardware [12]. On two different graphic cards, the results show that the proposed GPU-based FCM algorithm is more efficient and faster than the sequential FCM. The authors succeeded in reaching a 73-fold speedup on NVIDIA GeForce 8500 GT. Amazingly, a 140-fold speedup was achieved on NVIDIA GeForce 8800 GTX compared with sequential FCM for 65k yeast gene expression data set of 79 dimension.

Rowinska and Goclawski proposed a CUDA-based FCM algorithm to accelerate image-segmentation [13]. The proposed method has been tested on polyurethane foam with fungus color images and was compared with the sequential FCM implemented using C++ and MATLAB. The authors achieved a 10-fold speedup of their parallel proposal compared with the FCM implemented in C++ for object area of 310k pixels, and a 50- to 100-fold speedup compared with the FCM implemented in MATLAB for object area of 260k pixels. A comparison of our work and the previous related works is summarized in Table 1.

4   The Proposed Method

The sequential FCM algorithm has been subjected to extensive analysis in order to find out where the algorithm exhibits parallelism that we might exploit in the parallel design. The strongest data dependency in the FCM algorithm is the steps where the total summation calculation is required, as illustrated in step 3 in the sequential FCM (Algorithm 1). For instance, two sigma operations are needed to calculate the cluster centers as shown in Equation 3. Such a strong dependency makes parallelizing the sequential algorithm infeasible. According to Bernsteins conditions [14], this type of dependency is called output dependence. In parallel computing, the reduction method is an efficient approach to remove output dependence.

The proposed parallel FCM design consists of two main parts: a sequential part executed on the CPU (host) and a parallel part executed on the GPU (device). Fig. 2 shows the block diagram of the proposed work. The following sub-sections discuss each stages of the block diagram in Fig. 2.

Table 1: Comparison of our work and previous related works

Work by	Method	Image dataset	Speedup
Li et al. [9]	Modified the original FCM algorithm and then parallelized it on GPGPU	Natural images (from 53kB to 101kB)	10x
Mahmoud et al. [10]	Parallelized br-FCM the variant of FCM algorithm on GPGPU	Medical images (Lung CT with the dimesion of 512x512; Knee MRI with the dimension of 350x350)	23x faster than in [30]
Shalom et al [12]	Proposed a scalable FCM GPU-based implementation	Yeast gene expression data set (79 dimension with 65K genes)	140x
Rowinska et al. [13]	Presented a CUDA-based FCM algorithm to accelerate image segmentation	Polyurethane foam with fungus color images (object area of 310k pixels)	10x
This paper	A parallel FCM approach on GPGPU using CUDA	Digital brain phantom simulated dataset (from 20kB to 1000kB)	Superlinear speedup up to 674x

4.1 Initialization and data transferring

As shown in Fig. 2, the first two steps are executed sequentially on the host. The membership is randomly initialized. The memories are allocated on the device global memory for the pixels of the image data, membership, and cluster centers. All the arrays are defined in a 1-D pattern.

After defining memories on the device, all the data are transferred from host to device, and then the main program loop is started. Subsequently, the parallel kernels are called concurrently to manipulate the image pixels on the device.

4.2 Calculating cluster centers from membership functions

The host calls four CUDA kernels one after another to calculate the cluster centers from memberships. The first CUDA kernel concurrently handles the heavy calculations, such as exponential, division, and multiplication of floating points for every pixel. At this step, the final summation is not included. The numerator and denominator of Equation 3 are calculated separately for every pixel, and the results are stored in two different arrays in the device global memory. The number of spawned CUDA threads in this kernel is defined to be equal to the number of image pixels, such that every thread will handle one pixel.

The second CUDA kernel at this phase is the reduction kernel, which computes the partial summation of the numerator of Equation 3. The reduction technique is an efficient method to break down the dependency among the data. The computation complexity of the sequential addition of n elements is $O(n)$. However, using parallel computing can significantly improve the computation complexity to $O(log\ n)$ [15][16]. Several CUDA reduction methods are available, such as in [17][18][15]. The CUDA reduction method used in this work is similar to [15] and is shown in Algorithm 2. First, a segment of the input is loaded into the device-shared memory. This device shared memory can facilitate fast access to the image pixels [19][20]. The reduction process is then performed over the shared memory. Each calculated partial sum of every segment stored in the shared memory is loaded to the output in the global memory. As illustrated in Algorithm 2, the CUDA block ID (blockIdx.x) is used as an index to store the partial sum from the device-shared memory to the
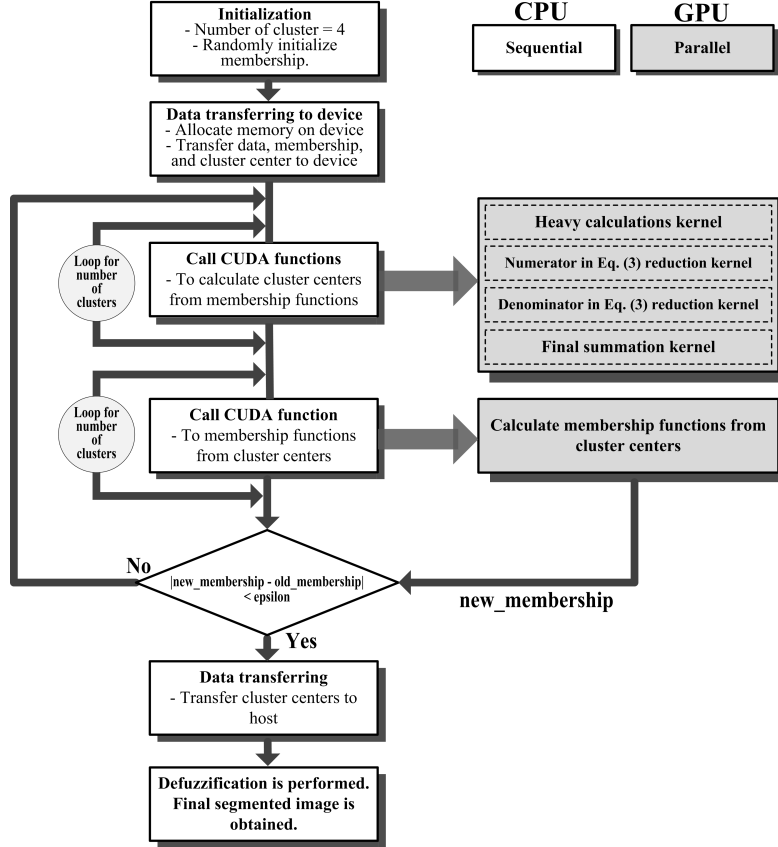
Figure 2: The Block Diagram of The Proposed Parallel Fuzzy C-Means

global memory. Fig. 3 demonstrates the reduction process performed on GPGPU using shared memory.

The actual reduction for the illustrated example in Fig. 3, reduces the addition operations from adding 16 elements to only 2 elements. Another example from the conducted experiments of this work is an image with a size of 1 MB (1048576 bytes) that was reduced to (1048576/128 << 1), which equals 4 KB (4096 bytes).

The third kernel to be called in this phase (calculating cluster centers from the membership function phase) is another reduction kernel that calculates the partial sum of the denominator of Equation 3. Finally, the last CUDA kernel calculates both final summations from the previous two kernels and computes the final result. Only one thread is defined for this kernel. The reason for this one thread kernel is that instead of transferring the reduced arrays from the previous kernels to the host memory to calculate the final summations, in this proposed method the device is allowed to carry out the final summation only with one thread. Lastly, all the previous four CUDA kernels are called in iterative loops that are equal to the predefined number of clusters. This is to calculate the cluster centers from the membership functions as shown in Fig. 2.
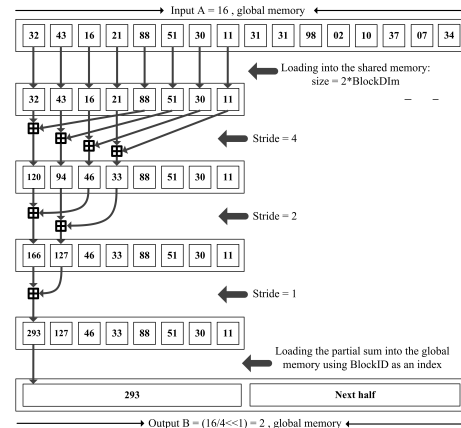

Figure 3: Sum Reduction Example on GPGPU. There are four CUDA block dimension in this example.

5

Algorithm 2: SUM REDUCTION ON GPGPU USING CUDA

Input: A large set $A = \{a_1, a_2, \ldots, a_n\}$ where $n = pixels$
Output: A reduced small set $B = \{b_1, b_2, \ldots, b_m\}$ where $m = n/blockDim << 1$

1 $global\_idx \leftarrow blockIdx.x * blockDim.x + threadIdx.x$
2 $local\_idx \leftarrow threadIdx.x$
3 $start \leftarrow 2 * blockIdx.x * blockDim.x$
4 $\_\_shared\_\_ \ partialSum[2 * MAX\_THREAD]$
5 //**Loading segment from the input into the shared memory:**
6 **if** $(start + local\_idx) < n$ **then**
7 $\quad\lfloor \ partialSum[local\_idx] = A[start + local\_idx]$
8 **else**
9 $\quad\lfloor \ partialSum[local\_idx] = 0.0$
10 **if** $(start + local\_idx + blockDim.x) < n$ **then**
11 $\quad\lfloor \ partialSum[local\_idx + blockDim.x] = A[start + local\_idx + blockDim.x]$
12 **else**
13 $\quad\lfloor \ partialSum[local\_idx + blockDim.x] = 0.0$
14 //**Reduction over the device shared memory:**
15 **for** $stride \leftarrow blockDim.x$ **to** 0 ; $stride/ = 2$ **do**
16 $\quad$ **if** $local\_idx < stride$ **then**
17 $\quad\quad\lfloor \ partialSum[local\_idx]+ = partialSum[local\_idx + stride]$
18 //**Storing the output into the device global memory:**
19 **if** $local\_idx == 0 \&\&(global\_idx * 2) < n$ **then**
20 $\quad\lfloor \ B[blockIdx.x] = partialSum[local\_idx]$

4.3 Calculating membership functions from cluster centers

Only one CUDA kernel is defined to compute membership functions from the cluster centers. Rather than defining CUDA threads and block dimensions, the implementation in this kernel is quite similar to the sequential algorithm. The spawned CUDA threads are defined equally to the image pixels, which implies fine-grained granularity. Thus, one thread will handle one pixel. In correspondence to the previous phase of the proposed work 2, this kernel will be called in an iterative loop equally to the predefined number of clusters. At this stage, the computed new membership function arrays will be transferred to the host. The host will determine if the new membership function satisfies the condition as shown in Fig. 2. If the condition is satisfied, finally the cluster center arrays will be transferred back to the host. Defuzzification is performed and the the final segmented image is obtained.

5 Implementation and Results

In this section, the implementation design of the proposed method is introduced in the first subsection. The functionality of the proposed method is proven using both qualitative and quantitative evaluations in the next subsections. The performance analysis is discussed in the final subsection.

5.1 Implementation

The proposed method was implemented using C language and CUDA. First, the sequential FCM algorithm was implemented in C. Our sequential C version was derived from a Java version available online at [21]. The sequential FCM in C was tested on Intel Core i5-480 CPU, Windows 7 Ultimate platform.

In the proposed parallel FCM, the image pixels, memberships, and cluster center arrays are defined in a 1-D pattern. The reason is to ensure coalesced memory transactions in the GPGPU. In addition, defining those input arrays in 1-D pattern will ease the number of CUDA block and grid sizes calculations. The CUDA block and grid sizes are consequently defined in 1-D patterns corresponding to the input arrays. Therefore, the form of the input has a significant effect on the performance of CUDA kernels because of the coalescing access [8][22]. Figure 4 illustrates examples on the indices of arrays are modified when converting multi-dimensional arrays to 1-D arrays. In this work, the image array was converted from 2-D to 1-D, and the membership array was con-

verted from 3-D to 1-D. The details of the parallel platform used in this experiment are shown in Table 2.
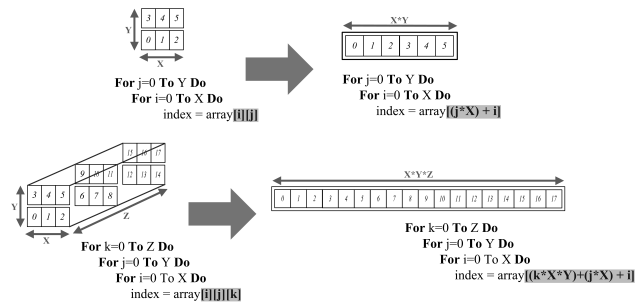

Figure 4: Converting Multidimensional Arrays to One Dimensional

Table 2: Platform of the experiments

CPU:	AMD Phenom(tm) II X4 810 Processor.
Kernel:	Linux x86_64 GNU.
GPU:	NVIDIA Tesla C2050.
CUDA:	CUDA compilation tools, release 5.0.

Figure 5: Representative Results of The 101st, 91st and 96th Axial Slice of Brain Tissue Phantom Using Sequential Fuzzy C-Means and The Proposed GPGPU-Based Fuzzy C-Means.

5.2 Functionality Evaluation

The proposed GPGPU-based FCM is tested on digital brain phantom simulated dataset from the Brain Web MR Simulator [23] with the size of 20kB to segment white matter (WM), gray matter (GM) and cerebrospinal fluid (CSF) soft tissues regions. Skull stripping [24] has been carried out on the brain phantom images to remove skull and other non-brain soft tissues, so that only brain soft tissues are used in the proposed parallel Fuzzy C-Means (FCM) segmentation process. When applying the proposed FCM on the brain soft tissues, four clusters are manually selected to represent the WM, GM, and CSF soft tissues regions and the final cluster represent the background region. Therefore in the proposed parallel FCM, there are four cluster center values being associated with the aforementioned regions. The functionality of the proposed method is then proven using both qualitative and quantitative evaluations in the following subsections.

5.2.1 Qualitative evaluation

The qualitative evaluation is performed for both the segmented results of the proposed parallel FCM and the sequential FCM. This is to evaluate the similarity of the segmented result of the proposed parallel FCM with the segmented result produced by t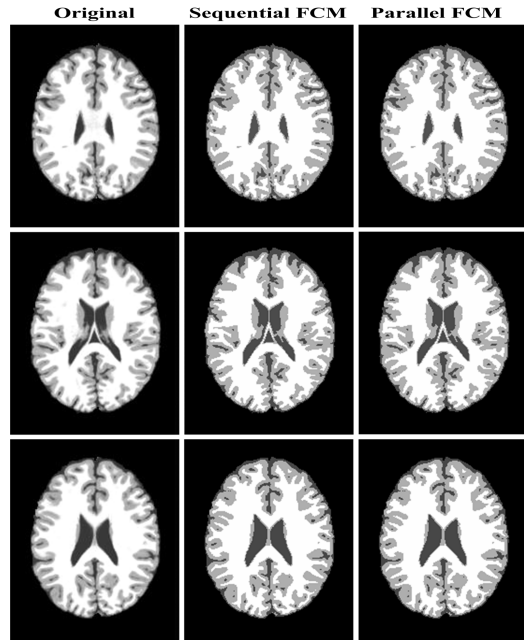he sequential FCM, visually. In Fig. 5, the experiment results are presented. It can be seen that the result of the proposed method is identical to the result of the sequential FCM.

5.2.2 Quantitative evaluation

The quantitative evaluation is used to compare the results of the proposed parallel Fuzzy C- FCM and sequential FCM. Evaluation metrics such as Dice Coefficient Similarity (DSC) [25] and performance analysis are used. DSC is used to evaluate if the accuracy of the segmented results of the proposed method is statistically similar to the segmented results of the sequential FCM based on the ground truth. While performance analysis is to compare the execution time and speed up of the proposed method with the sequential FCM. DSC is defined as in Equation 5.

$$DSC = \frac{2(PR \cap GT)}{PR + GT} \qquad (5)$$

Where PR is the segmented results of each method while GT is the ground truth provided with the dataset [23]. The DSC was implemented in C to be compatible with the implementation of the proposed method. An example of the ground truth is presented in Fig. 6e.
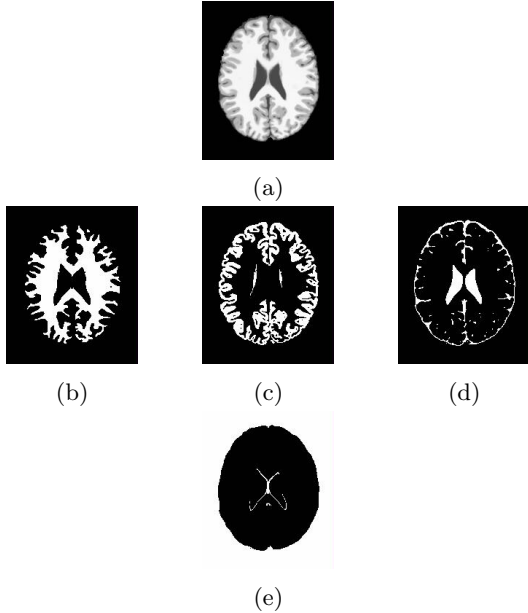
Figure 6: (a) The 96th Axial Slice of Brain Tissues Phantom and The Corresponding Ground Truth Images (b) White Matter (c) Gray Matter (d) Cerebrospinal Fluid (e) Background.

Fig. 7 illustrates the percentage of DSC of the proposed parallel FCM and sequential FCM for white matter (WM), gray matter (GM), cerebrospinal fluid (CSF) and background regions for 91th, 96th, 101th and 111th axial slices of brain tissues phantom. The accuracy of the segmented results of both the proposed method and sequential FCM are statistically similar.

5.3   Performance analysis

Once the functionality of the parallel approach is confirmed, performance analysis in terms of execution time and speedup was performed. As mentioned in Section 4.3, fine-grained granularity is adopted in this work in which one CUDA thread is spawned to manipulate one pixel. The total number of the spawned concurrent threads are equal to the image size to be segmented which indicates to the design scalability. The execution time was measured in both sequential and parallel approach for the process of calculating clusters centers and memberships. The initialization process was excluded from the measurements in both approaches. The function gettimeofday() was used to measure the elapsed time. For the sake of verification, the cudaEventRecord() function from CUDA API was also used to test the execution time. Table 3 presents the execution time of both the sequential FCM and the proposed parallel FCM on GPGPU.

The results of the execution time listed in Table 3 and the corresponding speedup illustrated in Fig. 8, are the average execution time and speedup of 30 runs.

From Table 3, it is shown that we have conducted experiments on various sizes of dataset from 20KB up to 1MB. In order to evaluate the execution time of the proposed parallel FCM in larger size dataset, we have enlarged the original phantom dataset 6KB (the original dataset size) up to 1MB. This enlargement is done only on the basis to evaluate the execution time of the proposed method in a larger size dataset.

Table 3: The Execution Time of Sequential Fuzzy C-Means and The Proposed Parallel Fuzzy C-Means In Seconds.

Dataset Size (Byte)	Sequential FCM (sec)	Parallel FCM (sec)
20KB	57	0.102
40kB	114	0.195
60KB	177	0.321
80KB	231	0.505
100KB	287	0.632
120KB	341	0.864
140KB	394	0.977
160KB	446	0.986
180KB	503	1.22
200KB	558	1.45
300KB	845	2.18
500KB	1420	2.4
700KB	1955	2.9
1000KB	2798	4.2

Fig. 8 shows the speedup results of the proposed parallel FCM over the sequential FCM. The horizontal red line in the middle of the chart represents the number of processing elements in the GPGPU device (Tesla C2050) used in the experiments. Generally, when the speedup exceeds the number of the utilized processors in parallel computing then this speedup is referred to as superlinear speedup. In fact, the speedup may equal to the number of the parallel processors only in the ideal situation because of the external overheads, such as data transferring, synchronization, and scheduling [26]. However, superlinear speedup can be achieved in some circumstances, such as in low-level computations because of memory hierarchies and cache effect [27].

The superlinear speedup obtained in this work can be justified by the high speed of GPGPUs in manipulating floating points compared with CPUs. In fact, GPGPUs are highly optimized for floating-point computations. The Intel i5 CPU used in the experiments
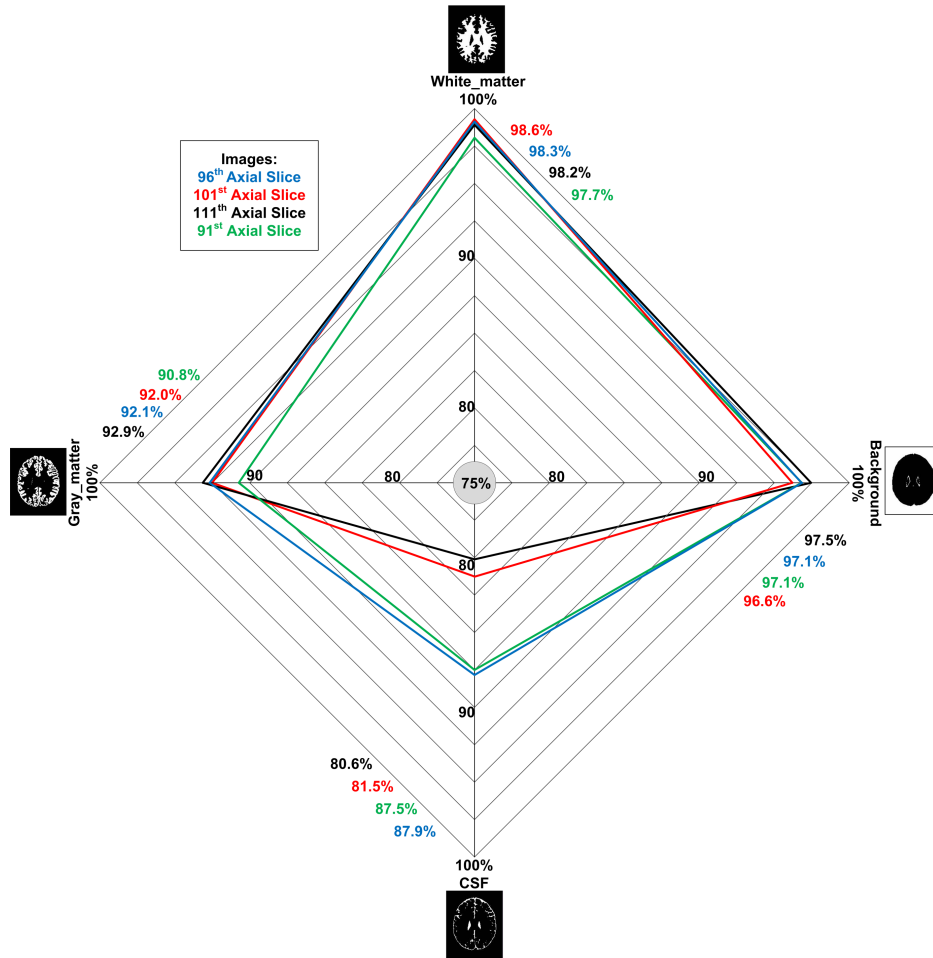
Figure 7: Percentage of Dice Similarity Coefficient for 91th, 96th, 101th, and 111th Axial Slices of Brain Tissues Phantom Images
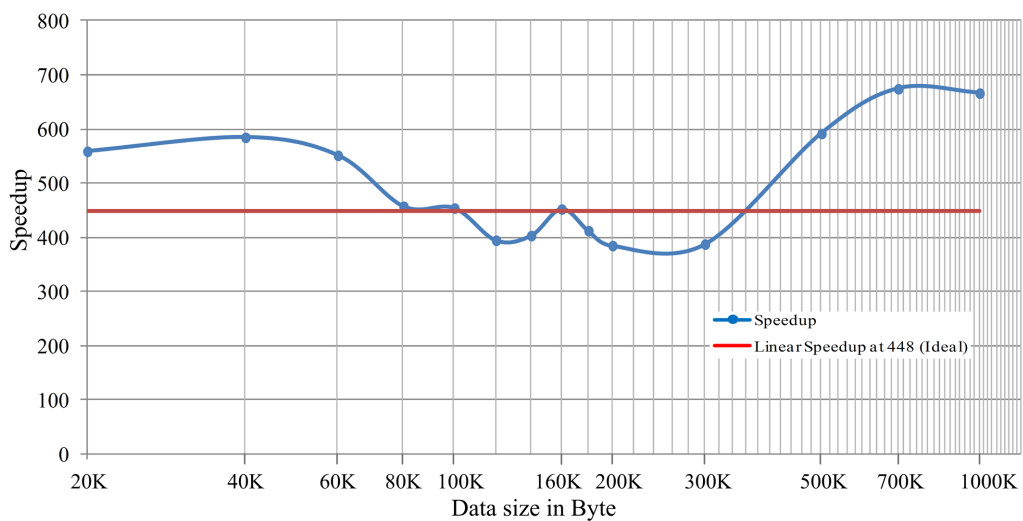

Figure 8: Speed Up of The Proposed Parallel Fuzzy C-Means on Tesla C2050 of 448 Processing Elements

9

can achieve 23 GFLOPs (Giga Floating Point Operations per Second) [28]. Meanwhile, the Tesla C2050 GPU used in this work can achieve 1030 GFLOPs [29].

In Fig. 8, superlinear speedup is obtained when the data size varies from 20 KB to 80 KB. When the data size is larger than 80 KB up to 300 KB, the speedup varies from 400- to 448-fold. Superlinear speedup occurs again when the data size goes beyond 300 KB.

The results of the proposed parallel FCM shows unusual behavior of the speedup with respect to the number of the spawned threads even though the superlinear speedup provides outstanding performance. This behavior of the speedup poses some open questions. By referring to Fig. 8, we would like to list down the following open questions:

1. When the data size varies from 20KB to 100KB, superlinear speedup of the parallel FCM is achieved although the number of the spawned concurrent threads is considerably small?

2. Does the nature of FCM algorithm have any role to play in obtaining superlinear speedup on GPGPU?

3. When the data size varies from 100KB to approximately 360KB, the results show no superlinear speedup, why?

4. The superlinear speedup happened again when the data size exceeds 360KB and the achieved results are much better compared with the ones mentioned in question 1. What will the speedup behavior be when the data size exceeds 1MB?

5. The proposed parallel FCM was tested on NVIDIA Tesla C2050 device of 448 processing elements, will the superlinear speedup and speedup behavior with respect to the spawned threads number occur on other GPGPU devices of different specifications? Or, is the nature of FCM algorithm only fitting Tesla C2050 such that this outstanding performance is achieved?

6 Conclusion and Future Works

GPGPUs are vary practical parallel models because they are affordable and not expensive. In this work, we proposed an efficient GPU-based implementation for Fuzzy C-Means algorithm. The functionality of the proposed parallel FCM has been verified and proven by conducting qualitative and quantitative evaluations. The empirical results show that the parallel FCM works precisely as the traditional sequential FCM. In addition, high performance and superlinear speedup of approximately 674 folds have been achieved compared with sequential FCM.

In future, it would be interesting to explore the open questions mentioned in Section 5.3 and find some answers by using FCM algorithm as a case study on several GPGPU devices. Recently, new CUDA devices have been released featured with the capability of launching dynamic parallel kernels. Generally speaking, dynamic kernels or (nested kernels) enables to multiple levels reduction concurrently. It would be also an interesting topic in the future to implement FCM on such powerful devices.

References

[1] Chattopadhyay S., Pratihar D.K., and Sarkar, S.C.D *A Comparative Study of Fuzzy C-Means Algorithm and Entropy-Based Fuzzy Clustering Algorithms.* Computing and Informatics, Vol. 30, pp. 701-720, 2011

[2] Tou, J.T. and Gonzalez R.C. *Pattern Recognition Principles.* Addison-Wesley, London.1974

[3] Bezdek J.C. *Pattern Recognition with Fuzzy Objective Function Algorithms.* Kluwer Academic Publishers. 1981

[4] Ball G. and Hall D. *A Clustering Technique for Summarizing Multivariate Data.* Behav Sci 12, pp. 153-155, 1967

[5] Shen Y. and Li Y-L. *An automatic fuzzy c-means algorithm for image segmentation.* Springer-Verlag, Vol. 14, Number 2, pp. 123-128. 2009

[6] Vadiveloo M., Abdullah R., Rajeswari M. and Abu-Shareha A.A. *Image Segmentation With Cyclic Load Balanced Parallel Fuzzy C-Means Cluster Analysis.* IEEE International Conference on Imaging Systems and Techniques, Malaysia, 2011, pp. 124-129. 2011

[7] Farber Rob, *CUDA Application Design and Development.* Morgan Kaufmann Publishers Inc. ISPN 9780123884268, San Francisco, CA, USA, 1st edition, 2012.

[8] David B. Kirk and Wen-mei W. Hwu. *Programming Massively Parallel Processors: A Hands-on Approach.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2010.

[9] Haiyang Li, Zhaofeng Yang, Hongzhou He. *An Improved Image Segmentation Algorithm Based*

on GPU Parallel Computing. Journal of Software, Vol 9, No 8 (2014), 1985-1990, Aug 2014.

[10] Mahmoud Al-Ayyoub, AnsamM Abu-Dalo, Yaser Jararweh, Moath Jarrah, Mohammad Al Sa'd, *A GPU-based implementations of the fuzzy C-means algorithms for medical image segmentation.* `http://dx.doi.org/10.1007/s11227-015-1431-y`. The Journal of Supercomputing, DOI 10.1007/S11227-015-1431-Y, ISSN 0920-8542, Pages: 1-14, Springer US, 2015.

[11] Eschrich S., Jingwei Ke, Hall L.O., Goldgof D.B., *Fast accurate fuzzy clustering through data reduction.* `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1192702&isnumber=26743`. In Fuzzy Systems, IEEE Transactions on Vol.11, No.2, Pages: 262-270, DOI 10.1109/T-FUZZ.2003.809902, Apr 2003.

[12] Shalom, S.A.A., Dash, M. and Tue, M. *Graphics hardware based efficient and scalable fuzzy c-means clustering.* `http://dx.doi.org/10.1007/s11227-015-1431-y`. In Proceedings of the 7th Australasian data mining conference, volume 87, Australian Computer Society Inc, Darlinghurst, Australia, AusDM 08, pp 179186. 2008.

[13] Rowi ska Z, Gocawski J *Cuda based fuzzy c-means acceleration for the segmentation of n images with fungus grown in foam matrices.* Image Processing and Communications 17(4):191200. DOI:10.2478/ V10248-012-0046-7, 2012.

[14] Bernstein A. J. *Analysis of Programs for Parallel Processing.* Electronic Computers, IEEE Transactions on , VOL EC-15, NO.5, Pages:757-763, Oct. 1966.

[15] *CUDA Array Sum with Reduction,* `https://gist.github.com/wh5a/4424992`. Retrieved 10 May 2015.

[16] *CUDA Optimization,* `http://sbel.wisc.edu/Courses/ME964/2012/Lectures/lecture0313.pdf`. Retrieved 10 May 2015.

[17] *Faster Parallel Reductions on Kepler,* `http://devblogs.nvidia.com/parallelforall/faster-parallel-reductions-kepler/`. Retrieved 10 May 2015.

[18] *Optimizing Parallel Reduction in CUDA,* `http://developer.download.nvidia.com/compute/cuda/1.1-Beta/x86_website/projects/reduction/doc/reduction.pdf`. Retrieved 10 May 2015.

[19] *GPU Performance Analysis and Optimization,* `http://on-demand.gputechconf.com/gtc/2012/presentations/S0514-GTC2012-GPU-Performance-Analysis.pdf`. Retrieved 10 May 2015.

[20] *Using Shared Memory in CUDA C/C++,* `http://devblogs.nvidia.com/parallelforall/using-shared-memory-cuda-cc/`. Retrieved 10 May 2015.

[21] *Java Image Processing Cookbook,* `http://www.lac.inpe.br/JIPCookbook/Code/JAIStuff/algorithms/fuzzycmeans/FuzzyCMeansImageClustering.java.txt`. Retrieved 10 May 2015.

[22] *How to Access Global Memory Efficiently in CUDA C/C++ Kernels,* http://devblogs.nvidia.com/parallelforall/how-access-global-memory-efficiently-cuda-c-kernels. Retrieved 03 Mar 2015.

[23] D.L. Collins, A.P. Zijdenbos, V. Kollokian, J.G. Sled, N.J. Kabani, C.J. Holmes and A.C. Evans *Design and Construction of a Realistic Digital Brain Phantom.* IEEE Transactions on Medical Imaging, Vol. 17, No. 3, pp. 463-468, (1998)

[24] Dogdas B., Shattuck D.W., and Leahy R.M. *Segmentation of Skull and Scalp in 3D Human MRI Using Mathematical Morphology Human Brain Mapping.* Vol. 26, No.4, pp.273-285. 2005

[25] Zijdenbos A.P., Dawant B.M. Margolin R.A. and Palmer A.C. *Morphometric analysis of white matter lesions in MR images: Method and validation.* IEEE Transactions on Medical Imaging, 13(4), pp.716. 1994

[26] Gebali Fayez, *Algorithms and Parallel Computing.* Wiley Publishing, ISPN 0470902108, 1st edition, 2011.

[27] Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar, *Introduction to Parallel Computing.* Addison-Wesley, ISPN 0201648652, 2nd edition, 2003.

[28] *TESLA C2050 / C2070 GPU Computing Processor ,* `http://www.nvidia.com/docs/IO/43395/NV_DS_Tesla_C2050_C2070_jul10_lores.pdf`. Retrieved 10 May 2015.

[29] *Intel Core i5-400 Processor Series*, http://download.intel.com/support/processors/corei5/sb/core_i5-400.pdf. Retrieved 10 May 2015.

[30] *Soroosh 129/GPU-FCM*, https://github.com/Soroosh129/GPU-FCM. Retrieved 10 May 2015.