International Journal of Combined Research & Development (IJCRD)
eISSN:2321-225X;pISSN:2321-2241 Volume: 5; Issue: 6; June-2016

Accelerating Biomedical Imaging using parallel Fuzzy C-Means Algorithm

Vindhya D S
PG Student
Dept of CSE
Acharya Institute of Technology
Bengaluru, Karnataka, India
vindhyads@gmail.com

V Nagaveni
Assistant Professor
Dept of CSE
Acharya Institute of Technology
Bengaluru, Karnataka, India
nagaveni@acharya.ac.in

Abstract : The focus is mainly on the development, acquisition and image reconstruction strategies, using MRI, to accurately and quantitatively image physiology. Primary applications include functional brain imaging, structural brain imaging, and neuromuscular. The software used is OpenACC, to accelerate their advanced imaging model. OpenACC is a directive based programming model designed for scientists and researchers looking to tap into the computational power of accelerators without significant programming effort. This software provides the significant speed-ups using the new PGI compiled software on the NVIDIA GPU. Hence it is now able to develop some other application software that reduced the time it would normally take to reconstruct the MRI scan from 40 days down to a couple of hours, OpenACC also allowed to run on one of the fastest supercomputers in the world.

Keywords : OpenACC, MRI, FCM, GPU.

1. INTRODUCTION

In the past decades, medical image has been commonly used to facilitate the clinic diagnosis. Various imaging techniques such as X-rays, Ultrasounds, Computed Tomography scans (CT) or Magnetic Resonance Images (MRIs) have been used to sense the irregularities in human body. The physicians identify the tumors, tissues, and the anatomical structures according to all of these images. To detect abnormality in brain the brain MRI is useful medical imaging tool. In general, the brain MRI can be classified into three significant regions, such as matter (WM), grey matter (GM) and cerebrospinal fluid spaces (CSF). Many image-processing technologies have been used to copy with medical images; especially image segmentation technologies. The image segmentation is the process to split image data to a serial of non-overlapping homogeneous region. It has been used to analyze medical images for facilitating diagnosis and therapy. In addition, it can be used to reconstruct image, where it is useful to identify the abnormality in the brain. For the brain MRI, the image segmentation techniques are essential for clinic diagnosis, as they are used to classify WM, GM and CSF regions from observed image. The physicians can determine abnormality in the patient brain from these regions.

Clustering is one of the image segmentation techniques. Clustering is the process of classifying data into group of similarity. Some of clustering algorithms have been commonly adopted in computer, engineering and mathematics field. Similarly, the clustering algorithms have also been extended to medical fields. Clustering algorithms, such as K-means (KM) clustering, Moving K-means (MKM) and Fuzzy C-means, have been proposed to make the analysis of the brain MRI easier. Fuzzy C-means (FCM) algorithm [4] has been proved to achieve the better segmentation efficiency over the other clustering approaches. But the drawback of these clustering algorithms is the huge computational time required for convergence. In recent years, many high performance hardware and software technologies have been released, such as Intel and AMD multi-core systems, graphic processing units (GPU), OpenMP, OpenCL, CUDA and Hadoop. In these new technologies, the development of GPU is rapidly growing, and it has been used to accelerate computation-consuming applications. The GPU devices consist of up to hundreds cores per-chip, and it can issue the thousands of threads to fully utilize its computational power. GPU is not only adopted to develop graphic application but also utilized to solve general computing problem. The General-Purpose computing on Graphics Processing Units (GPGPU) such as Open Computing Language (OpenCL) and compute unified device architecture (CUDA), has successfully made supercomputing available to variety of applications. NVIDIA's new Kepler GK110 GPU raises the parallel computing bar considerably and will help solve the world's most difficult computing problems. By offering much higher processing power than the prior GPU generation and by providing new methods to optimize and increase parallel workload execution on the GPU, 2.Kepler GK110 - Extreme Performance, Extreme Efficiency Comprising 7.1 billion transistors, Kepler GK110 is not only the fastest, but also the most architecturally complex microprocessor ever built. Adding many new innovative features focused on compute performance, GK110 was designed to be a parallel processing powerhouse for Tesla® and the HPC market. Kepler GK110 will provide over 1 TFlop of double precision throughput with greater than 80% DGEMM efficiency versus 60-65% on the prior Fermi architecture. In addition to greatly improved performance, the Kepler architecture offers a huge leap forward in power efficiency, delivering up to 3x the performance per watt of Fermi. The following new features in Kepler GK110 enable increased GPU utilization, simplify parallel program design, and aid in the deployment of GPUs across the spectrum of compute environments ranging from personal workstations to supercomputers. .

II RELATED WORK

1.THE QUADRO FX 5600 GRAPHICS CARD

The Quadro FX 5600 is a graphics card equipped with a G80 graphics processing unit (GPU). The Quadro has a large set of processor cores that can directly address a global memory. This architecture supports the single instruction, multiple-data (SIMD) programming model, which is more general and flexible than the programming models supported by previous generations of GPUs, and which allows developers to easily implement data-parallel algorithms. In this section we discuss NVIDIA's Compute United Device Architecture (CUDA) and the architectural features of the G80 that are most relevant to accelerating MRI reconstructions. More complete descriptions are found in from the application developer's perspective, the CUDA programming model consists of ANSI C supported by several keywords and constructs. CUDA treats the GPU as a coprocessor that executes data-parallel kernel functions. The developer supplies a single source program encompassing both host (CPU) and kernel (GPU) code. NVIDIA's compiler, nvcc, separates the host and kernel codes, which are then compiled by the host compiler and nvcc, respectively. The host code transfers data to and from the GPU's global memory via API calls, and initiates the kernel code by calling a function.

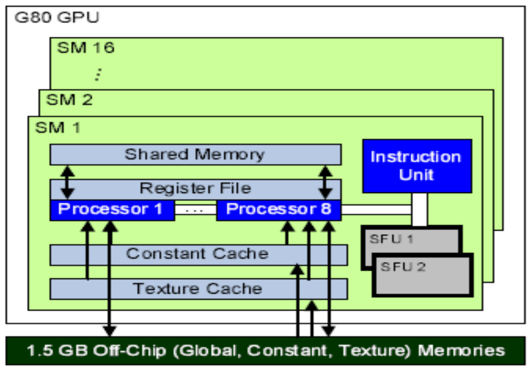


Figure1 Architecture of Quadro FX 5600

Figure 1 depicts the Quadro’s architecture. The G80 GPU consists of 16 streaming multiprocessors (SMs), each containing eight streaming processors (SPs), or processor cores, running at 1.35 GHz. Each SM has 8,192 registers that are shared among all threads assigned to the SM. The threads on a given SM's cores execute in SIMD (single-instruction, multiple-data) fashion, with the instruction unit broadcasting the current instruction to the eight cores. Each core has a single arithmetic unit that performs single-precision floating point arithmetic and 32-bit integer operations. Additionally, each SM has two special functional units (SFUs), which perform more complex FP operations such as the trigonometric functions with low latency. Both the arithmetic units and the SFUs are fully pipelined. Thus, each SM can perform 18 FLOPS per clock cycle (one multiply-add operation per SP and one complex operation per SFU), yielding 388.8 GFLOPS (16 SM * 18 FLOP/SM * 1.35 GHz) of peak theoretical performance for the GPU. The Quadro has 76.8 GB/s of bandwidth to its 1.5 GB, o_- chip, global memory. Nevertheless, with computational resources supporting nearly 400 GFLOPS and each multiply add instruction operating on up to 16 bytes of data, applications can easily saturate that bandwidth. Therefore, as depicted in Figure 2, the G80 has several on-chip memories that can exploit data locality and data sharing to reduce an application's demands for o_-chip memory bandwidth. For example, the Quadro has a 64 KB, o_-chip constant memory, and each SM has an 8 KB constant memory cache. Because the cache is single-ported, simultaneous accesses of direct addresses yield stalls. However, when multiple threads access the same address during the same cycle, the cache broadcasts that address's value to those threads with the same latency as a register access. This feature proves beneficial for the MRI reconstruction algorithm. In addition to the constant memory cache, each SM has a 16KB shared memory for data that is either written and reused or shared among threads. Finally, for read-only data that is shared by many threads but not necessarily accessed simultaneously by all threads, the o_-chip texture memory and the on-chip texture caches exploit 2D data locality to substantially reduce memory latency. Threads executing on the G80 are organized into a three level hierarchy. At the highest level, each kernel creates a single grid, which consists of many thread blocks. The maximum number of threads per block is 512. Each thread block is assigned to a single SM for the duration of its execution. Threads in the same block can share data through the shared memory and can perform barrier synchronization by invoking the sync threads primitive. Threads are otherwise independent, and synchronization across thread blocks is safely accomplished by terminating the kernel. Finally, threads within a block are organized into warps of 32 threads. Each warp executes in SIMD fashion, with the SM's instruction unit broadcasting the same instruction to the eight cores on four consecutive clock cycles. SMs can interleave warps on an instruction-by-instruction basis to hide the latency of global memory accesses and long latency arithmetic operations. When one warp stalls, the SM can quickly switch to a ready warp in the same thread block or in some other thread block assigned to the SM. The SM stalls only if there are no warps with all operands available. Tuning the performance of a CUDA kernel often involves a fundamental trade-o_ between the efficiency of individual threads and the thread-level parallelism (TLP) among all threads. This trade-o_ exists because many optimizations that improve the performance of an individual thread tend to increase the thread's use of limited resources that are shared among all threads assigned to an SM. For example, as each thread's register usage increases, the total number of threads that can simultaneously occupy the SM decreases. Because threads are assigned to an SM not individually, but in large thread blocks, a small increase in register usage can cause a correspondingly much larger decrease in SM occupancy.

2 KEPLER GK110GPU ARCHITECTURE

NVIDIA’s new Kepler GK110 GPU raises the parallel computing bar considerably and will help solve the world’s most difficult computing problems. By offering much higher processing power than the prior GPU generation and by providing new methods to optimize and increase parallel workload execution on the GPU, 2.Kepler GK110 - Extreme Performance, Extreme Efficiency Comprising 7.1 billion transistors, Kepler GK110 is not only the fastest, but also the most architecturally complex microprocessor ever built. Adding many new innovative features focused on compute performance, GK110 was designed to be a parallel processing powerhouse for Tesla® and the HPC market. Kepler GK110 will provide over 1 TFlop of double precision throughput with greater than 80% DGEMM efficiency versus 60-65% on the prior Fermi architecture. In addition to greatly improved performance, the Kepler architecture offers a huge leap forward in power efficiency, delivering up to 3x the performance per watt of Fermi. The following new features in Kepler GK110 enable increased GPU utilization, simplify parallel program design, and aid in the deployment of GPUs across the spectrum of compute environments ranging from personal workstations to supercomputers:

Dynamic Parallelism – adds the capability for the GPU to generate new work for itself, synchronize on results, and control the scheduling of that work via dedicated, accelerated hardware paths, all without involving the CPU. By providing the flexibility to adapt to the amount and form of parallelism through the course of a program's execution, programmers can expose more varied kinds of parallel work and make the most efficient use the GPU as a computation evolves. This capability allows less structured, more complex tasks to run easily and effectively, enabling larger portions of an application to run entirely on the GPU. In addition, programs are easier to create, and the CPU is freed for other tasks.

Hyper-Q – Hyper-Q enables multiple CPU cores to launch work on a single GPU simultaneously, thereby dramatically increasing GPU utilization and significantly reducing CPU idle times. Hyper -Q increases the total number of connections (work queues) between the host and the GK110 GPU by allowing 32 simultaneous, hardware managed connections (compared to the single connection available with Fermi). Hyper-Q is a flexible solution that allows separate connections from multiple CUDA streams, from multiple Message Passing Interface (MPI) processes, or even from multiple threads within a process. Applications that previously encountered false serialization across tasks, thereby limiting achieved GPU utilization, can see up to dramatic performance increase without changing any existing code.

Grid Management Unit – Enabling Dynamic Parallelism requires an advanced, flexible grid management and dispatch control system. The new GK110 Grid Management Unit (GMU) manages and prioritizes grids to be executed on the GPU. The GMU can pause the dispatch of new grids and queue pending and suspended grids until they are ready to execute, providing the flexibility to enable powerful runtimes, such as Dynamic Parallelism. The GMU ensures both CPU and GPU generated workloads are properly managed and dispatched.

NVIDIA GPUDirect™– NVIDIA GPUDirect™ is a capability that enables GPUs within a single computer, or GPUs in different servers located across a network, to directly exchange data without needing to go to CPU/system memory. The RDMA feature in GPUDirect allows third party devices such as SSDs, NICs, and IB adapters to directly access memory on multiple GPUs within the same system, significantly decreasing the latency of MPI send and receive messages to/from GPU memory. It also reduces demands on system memory bandwidth and frees the GPU DMA engines for use by other CUDA tasks. Kepler GK110 also supports other GPUDirect features including Peer to Peer and GPUDirect for Video.

Macintosh, use the font named Times. Right margins should be justified, not ragged.

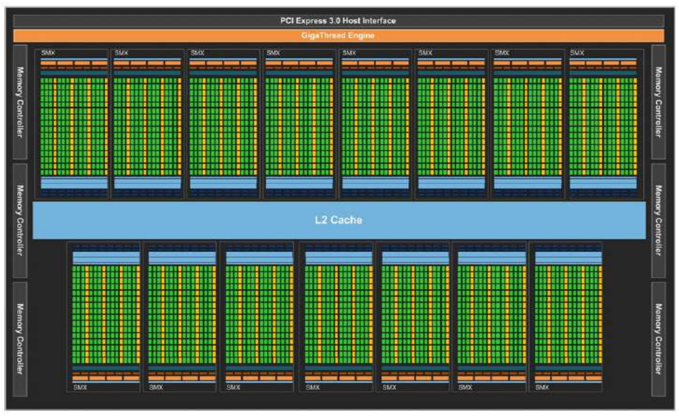


Figure 2 Overview of GK110/ Kepler architecture

An Overview of the GK110 Kepler Architecture Kepler GK110 was built first and foremost for Tesla, and its goal was to be the highest performing parallel computing microprocessor in the world. GK110 not only greatly exceeds the raw compute horsepower delivered by Fermi, but it does so efficiently, consuming significantly less power and generating much less heat output. A full Kepler GK110 implementation includes 15 SMX units and six 64-bit memory controllers. Different products will use different configurations of GK110. For example, some products may deploy 13 or 14 SMXs. Key features of the architecture that will be discussed below in more depth include:
1. The new SMX processor architecture
2. An enhanced memory subsystem, offering additional caching capabilities, more bandwidth at each level of the hierarchy, and a fully redesigned and substantially faster DRAM I/O implementation.
3. Hardware support throughout the design to enable new programming model capabilities.

PERFORMANCE PER WATT-A principal design goal for the Kepler architecture was improving power efficiency. When designing Kepler, NVIDIA engineers applied everything learned from Fermi to better optimize the Kepler architecture for highly efficient operation. TSMC's 28nm manufacturing process plays an important role in lowering power consumption, but many GPU architecture modifications were required to further reduce power consumption while maintaining great performance. Every hardware unit in Kepler was designed and scrubbed to provide outstanding performance per watt.

2.1 Streaming Multiprocessor (SMX) Architecture

Streaming Multiprocessor (SMX) Architecture Kepler GK110's new SMX introduces several architectural innovations that make it not only the most powerful multiprocessor we've built, but also the most programmable and power efficient. SMX: 192 single precision CUDA cores, 64 double precision units, 32 special function units

(SFU), and 32 load/store units (LD/ST). SMX Processing Core Architecture Each of the Kepler GK110 SMX units feature 192 single-precision CUDA cores, and each core has fully pipelined floating-point and integer arithmetic logic units. Kepler retains the full IEEE 754/2008 compliant single and double precision arithmetic introduced in Fermi, including the fused multiply add (FMA) operation. One of the design goals for the Kepler GK110 SMX was to significantly increase the GPU's delivered double precision performance, since double precision arithmetic is at the heart of many HPC applications. Kepler GK110's SMX also retains the special function units (SFUs) for fast approximate transcendental operations as in previous-generation GPUs, providing 8x the number of SFUs of the Fermi GF110 SM. Similar to GK104 SMX units, the cores within the new GK110 SMX units use the primary GPU clock rather than the 2x shader clock. Recall the 2x shader clock was introduced in the G80 Tesla-architecture GPU and used in all subsequent Tesla- and Fermi-architecture GPUs. Running execution units at a higher clock rate allows a chip to achieve a given target throughput with fewer copies of the execution units, which is essentially an area optimization, but the clocking logic for the faster cores is more power-hungry.

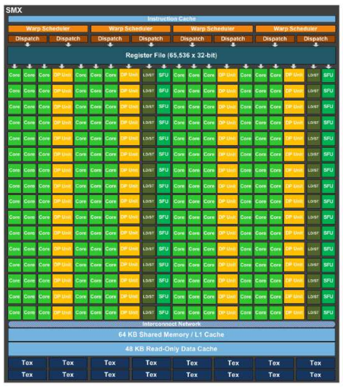

SMX
Instruction Cache
Warp Scheduler
Dispatch
Register File (65,536 x 32-bit)
Core
DP Unit
LD/ST
SFU
Interconnect Network
64 KB Shared Memory / L1 Cache
48 KB Read-Only Data Cache
Tex

Figure 3 Streaming Multiprocessor (Smx) Architecture

III PROPOSED ALGORITHM

- **Fuzzy C Means algorithm:** This algorithm works by assigning membership to each data point corresponding to each cluster center on the basis of distance between the cluster center and the data point. More the data is near to the cluster center more is its membership towards the particular cluster center. Clearly, summation of membership of each data point should be equal to one. After each iteration membership cluster centers are updated according to the formula:

$$\mu_{ij} = 1 / \sum_{k=1}^{c} (d_{ij} / d_{ik})^{(2/m-1)}$$

$$v_j = (\sum_{i=1}^{n} (\mu_{ij})^m x_i) / (\sum_{i=1}^{n} (\mu_{ij})^m), \forall j = 1, 2, ...c$$

Where,

'n' is the number of data points
'vj' represents the jth cluster center.
'm' is the fuzziness index m € [1,∞].
'c' represents the number of cluster center.
'µij' represents the membership of ith data to jth cluster center.
'dij' represents the Euclidean distance between ith data and jth cluster center.

Main objective of fuzzy c-means algorithm is to minimize:

$$J(U,V) = \sum_{i=1}^{n} \sum_{j=1}^{c} (\mu_{ij})^m \left\| x_i - v_j \right\|^2$$

where,

'$||x_i - v_j||$' is the Euclidean distance between i^{th} data and j^{th} cluster center.

ALGORITHMIC STEPS FOR FUZZY C-MEANS CLUSTERING

Let X = {x_1, x_2, x_3 ..., x_n} be the set of data points and V = {v_1, v_2, v_3 ..., v_c} be the set of centers.

1) Randomly select *'c'* cluster centers.
2) calculate the fuzzy membership 'μ_{ij}' using:

$$\mu_{ij} = 1 / \sum_{k=1}^{c} (d_{ij} / d_{ik})^{(2/m-1)}$$

3) compute the fuzzy centers 'v_j' using:

$$v_j = (\sum_{i=1}^{n} (\mu_{ij})^m x_i) / (\sum_{i=1}^{n} (\mu_{ij})^m), \forall j = 1, 2, ...c$$

4) Repeat step 2) and 3) until the minimum '*J*' value is achieved or $||U^{(k+1)} - U^{(k)}|| < \beta$.

where,

'k' is the iteration step.
'β' is the termination criterion between [0, 1].
'U = $(\mu_{ij})_{n*c}$' is the fuzzy membership matrix.
J' is the objective function.

ADVANTAGES

1) Gives best result for overlapped data set and comparatively better then k-means algorithm.
2) Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned membership to each cluster center as a result of which data point may belong to more than one cluster centers.

DISADVANTAGES

1) Apriori specification of the number of clusters.
2) With lower value of β we get the better result but at the expense of more number of iteration.
3) Euclidean distance measures can unequally weight underlying factors

IV FLOW OF GPU-BASED FCM CLUSTERING

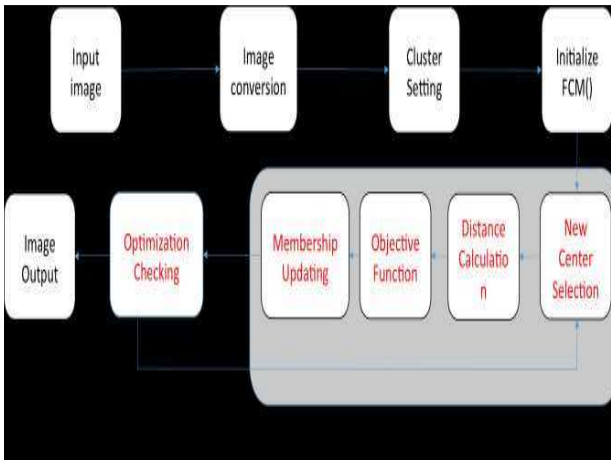


Figure 4 The process diagram of the proposed algorithm.

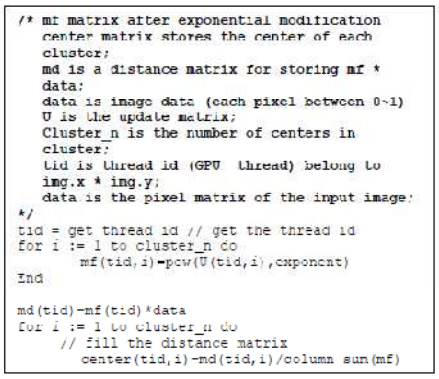
The kernel FCM part (gray box) is re-designed for execution on GPU as shown in Figure 3.. It includes following 9 steps:

1. Image Conversion
This step is to cover the original brain MRI to Grayscale image. Usually the format of the covered grayscale image is 8-bit. The input image is transferred into a gray-scale image that all the value of pixels are between 0 and 1.

2. Cluster Setting
This step is to set the number of clusters. The cluster number *c* is determined in FCM. The proper is the key to obtain the good result of FCM algorithm. In general, *c* is unknown, and *c* = {1, 2… *n*}. For the segmentation of brain MRI, *c* is set to 2.

3. FCM Initializing
This step is to select the initial center of cluster. Typically, the performance of FCM depends on the initial cluster center and/or the initial membership matrix. If an initial cluster center that is close to the actual final cluster center, then FCM will converge in short.

4. New Center Selection
This step is to select the new centers of the clusters. This step is implemented in GPU. Each thread is response to calculate an element of *u*. The pseudo code is shown in Figure 5.

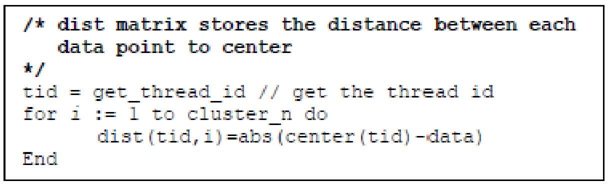


```
/* mf matrix after exponential modification
   center matrix stores the center of each
   cluster;
   md is a distance matrix for storing mf *
   data;
   data is image data (each pixel between 0~1)
   U is the update matrix;
   Cluster_n is the number of centers in
   cluster;
   tid is thread id (GPU  thread) belong to
   img.x * img.y;
   data is the pixel matrix of the input image;
*/
tid = get thread id // get the thread id
for i := 1 to cluster_n do
      mf(tid,i)=pow(U(tid,i),exponent)
End

md(tid)=mf(tid)*data
for i := 1 to cluster_n do
     // fill the distance matrix
       center(tid,i)=md(tid,i)/column sum(mf)
```

Figure 5 The pseudo code of step of the selection of new center

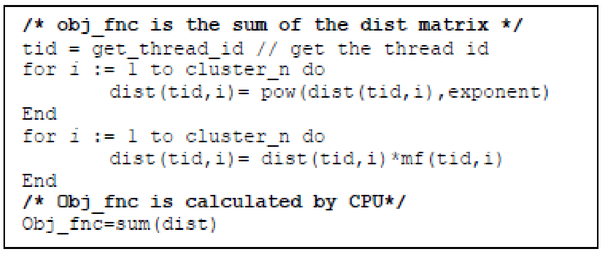
5. Distance Calculation
The distance between a data point and the cluster center in this step, *d* is calculated in this step. The thread i calculates *d* (*xi*, □*j*), *j* Σ{1, 2}. The pseudo code is shown in Figure 6.



```
/* dist matrix stores the distance between each
   data point to center
*/
tid = get_thread_id // get the thread id
for i := 1 to cluster_n do
      dist(tid,i)=abs(center(tid)-data)
End
```

Figure 6 .The pseudo code of step of calculation of distance between center and a data point

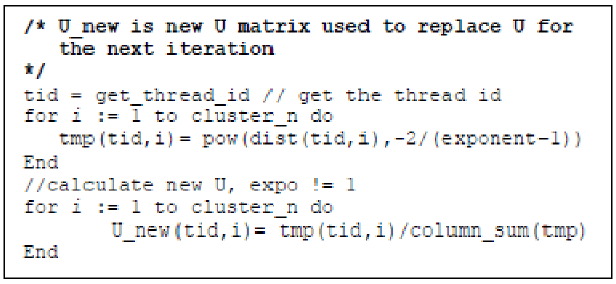
6. Objective Function Calculation
This step is to calculate the objective value by objective function; the distance matrix is recalculated on GPU in this step. The objective value *J* is calculated by CPU. The pseudo code is shown in Figure 7.



```
/* obj_fnc is the sum of the dist matrix */
tid = get_thread_id // get the thread id
for i := 1 to cluster_n do
      dist(tid,i)= pow(dist(tid,i),exponent)
End
for i := 1 to cluster_n do
      dist(tid,i)= dist(tid,i)*mf(tid,i)
End
/* Obj_fnc is calculated by CPU*/
Obj_fnc=sum(dist)
```

Figure 7.The pseudo code of step of objective value.

7. Membership Updating
This step is to calculate the new membership matrix *u* for next iteration, and this step is performed on GPU. The pseudo code is shown in Figure 8.



```
/* U_new is new U matrix used to replace U for
   the next iteration
*/
tid = get_thread_id // get the thread id
for i := 1 to cluster_n do
    tmp(tid,i)= pow(dist(tid,i),-2/(exponent-1))
End
//calculate new U, expo != 1
for i := 1 to cluster_n do
        U_new(tid,i)= tmp(tid,i)/column_sum(tmp)
End
```

Figure 8. The pseudo code of step of new *u* membership matrix

8. Optimization Checking
This step is to check whether the objective function is converged or not. If |*Jm-Jm-1*| ϵ, then FCM stops.ϵ□ is a small positive constant.

9 Image output
The final value is calculated in the steps above. This value is utilized to verify the color of the pixels. If the value of a pixel is greater the final value of center, it is black, and vice versa. Final image is converted to binary image with black and white color of pixels.

V IMPLEMENTATION USING OPENACC

OpenACC directives are the fast, simple and portable way to accelerate your scientific code. With OpenACC, you insert compiler hints – in the form of OpenMP-like directives into the compute-intensive portions of even the largest, most complex FORTRAN or C application, and the compiler automatically maps that code to an accelerator – including NVIDIA GPUs for higher performance. (OpenACC is fully compatible and interoperates with OpenMP and MPI.) OpenACC compilers are:

- Portable: Future-proof your codes with this open Standard
- Fast: Straight forward, high-level, compiler driven Approach to parallel computing
- Powerful: Ideal for accelerating large, legacy Fortran Or C codes

2X IN 5 STEPS
Most developers who try OpenACC see speedups of from 2 to 10X, following five key steps:

1. Evaluate and plan
2. Add parallel directives
3. Add data movement directives
4. Tune data movement
5. Optimize parallel scheduling

VI EXPERIMENTAL RESULTS AND DISCUSSIONS

In this work, the proposed GPU based FCM algorithm is implemented on two different NVIDIA GPU devices, such as NVIDIA KEPLER GK110 with 192 single precision CUDA cores and 64 bit memory controller, and NVIDIA Quadro FX5600. The hosts (CPU) are Intel Xeon E3-1230 v2 3.30GHz and Intel Xeon E3-1231 v3 3.40GHz with 64GB RAM, respectively. The CUDA version is 6.5. The input brain MRIs are download from the brain web datasets.

Figure 9 shows the input images and Figure 10 shows the processed images. The experimental results shows that the proposed algorithm can obtain the same quality results as original FCM, and it can achieve significant speed up over the original FCM executed on very powerful CPU. The comparison of the performance between the proposed GPU based FCM algorithm and traditional FCM is show in Table 1. The Cost/Performance (CP) ratio shows that the proposed algorithm is valuable for analysis of MR brain image.

Table 1.Comparison OF Results obtained using Sequential,Parallel GPU(Quadro andKepler) Architectures

	CPU **Intel**	**CPU** **Intel**	**GPU** **NVIDIA**	**GPU** **NVIDIA Kepler**
MRI1a and b (1280*801)	17.57	15.0	11.26	9.6
MRI2(512*512)	4.89	4.34	3.56	3.0
MRI3(1150*1280)	25.18	23.89	16.92	15.6
MRI4(512*512)	5.08	5.03	3.52	3.1
MRI5(512*512)	5.13	4.48	3.41	3.0

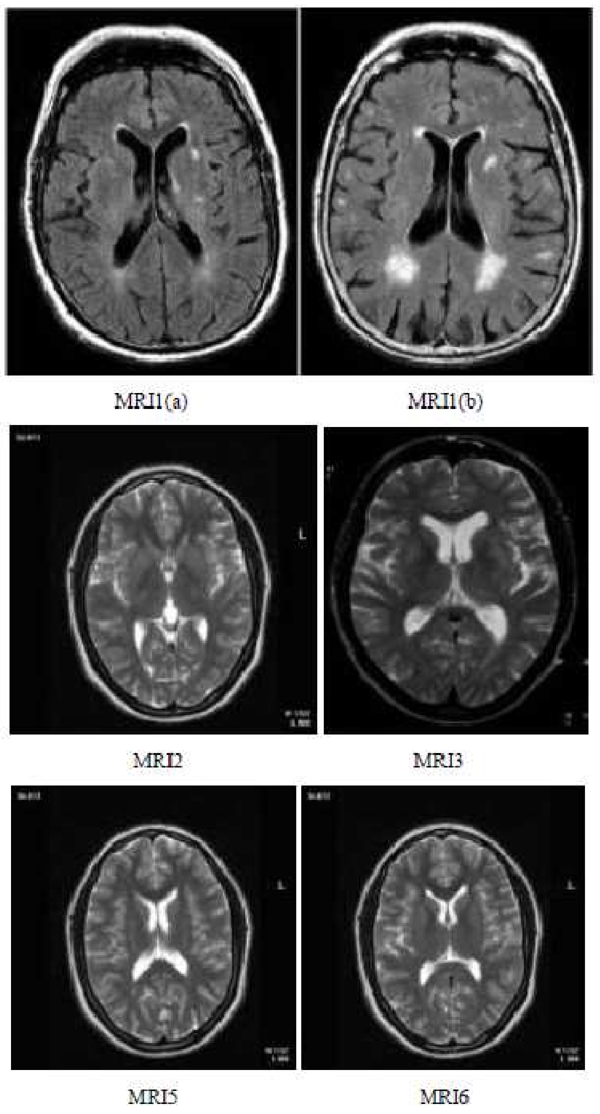


Figure 9 The original brain MR image

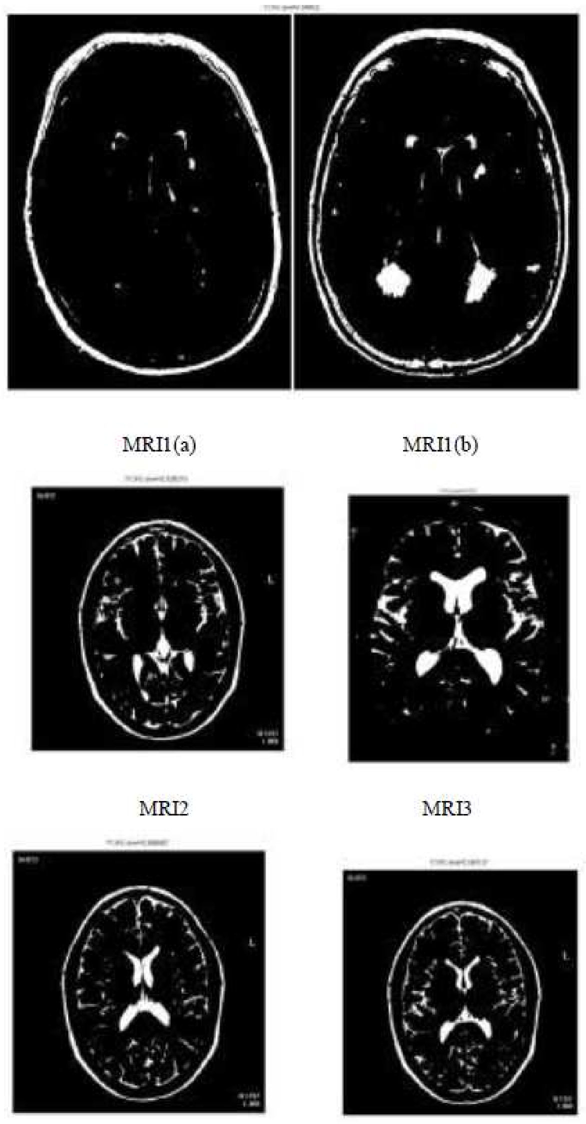


Figure 10 The processed brain mr image by GPU based on FCM algorithm

VII .CONCLUSION

Fuzzy C- Means algorithm gives best result for overlapped data set and comparatively better than k-means algorithm. Unlike k-means where data point must exclusively belong to one cluster center here data point is assigned membership to each cluster center as a result of which data point may belong to more than one cluster center. For the segmentation of brain MRI, FCM is a commonly used and efficient algorithm. However, it is computational-consuming algorithm. A Parallel FCM algorithm based on GPU to enhance the computation performance. The significant speed-ups using their new PGI compiled software OpenACC on the NVIDIA GPU. 2.Kepler GK110 - Extreme Performance, Extreme Efficiency Comprising 7.1 billion transistors, Kepler

GK110 is not only the fastest, but also the most architecturally complex microprocessor ever built. Kepler GK110 will provide over 1 TFlop of double precision throughput with greater than 80% DGEMM efficiency versus 60-65% on the prior Fermi architecture. In addition to greatly improved performance, the Kepler architecture offers a huge leap forward in power efficiency, delivering up to 3x the performance per watt of Fermi.

VIII REFERENCES

[1]2015 NVIDIA Corporation, UNIVERSITY OF ILLINOIS, "Accelerating
Biomedical Imaging"

[2]Che-Lun Hung, Yuan-Huai Wu, Yaw-Ling Lin, Yu-Chen Hu, Jieh-Shan Yeh, Chia-Chen Lin " GPU -based Fuzzy C Means Clustering Model For Brain MR Image" Proc. of the Third Intl. Conf. Advances in Computing, Communication and Information Technology- CCIT 2015

[3]Whitepaper ,NVIDIA's Next Generation CUDATM Compute Architecture: Kepler TM GK110

[4] M. Rakesh, amd T. Ravi, "Image Segmentation and Detection of
Tumor Objects in MR Brain Images Using Fuzzy C-means (FCM)
Algorithm," International Journal of engineering Research and application,vol.2,2012.pp.2088-2094.

[5]NCBIhttps://en.wikipedia.org/.../National_Center_for_Biotechnology_Information.