ISSN:2277-1581
01 Feb. 2017

An Empirical Analysis of Image Compression with Huffman Coding Technique on GPGPU

E.Sudarshan[1] , Dr. S. Jumlesha[2] , Ch. Satyanarayana[3], C. Srinivasa Kumar[4], K. Seena Naik[5]
[2]Department of Computer Science and Engineering
[1] VITAE, Hyderabad;[3] JNTUCE, Kakinada ;[4] VITS, Hyderabad ;[5] SREC, Warangal

Abstract : Processing the volume of medical images may involve high computations and complexity, so the GPU (Graphics Processing Unit) technology provides the remarkable acceleration in the graphics computations and also in general purpose computations due to its highly threaded parallelism support. The General Purpose Graphics Processing Unit (GPGPU or GP^{2}U) is accomplished a superior performance and efficiency with the support of Compute Unified Device Architecture (CUDA) programming model. This is the preliminary motivation to adapt the GPGPU technology in the medical field to process the volume of images like ultrasound (US), computed tomography (CT), magnetic resonance imaging (MRI) to diagnose the diseases. The Central Processing Unit (CPU) may have to pay more attention to process the volume of medical images on vital computations other than the CPU administration tasks. To overwhelm the CPU's overhead, the coprocessor GPU effectively can operate as a turbo booster on images and graphical data. This review analyzes the medical image compression techniques in the GPU environment and it is showing the profound speedup to compute task in a snap. This paper presents the comparative study of traditional compression techniques Huffman coding with respect to computational performance when they implemented in CPU and GPU environments. The results shown that most of the GPU's compression techniques are computationally faster by 3.9 times on an average when compare to CPU processing.

Keywords: GPGPU, CUDA, MLP, GFLOPS, PRAM.

I. INTRODUCTION

Over the years great advancement takes place in the field of image and video compression techniques that have been made by huge demand for the storage and transmission of visual information. Especially image compression techniques enormously using in the medical field because each slice of CT image may consist up to 150MB or above on a data set, it would be 200MB to 400MB on an average [1]. So we must require sophisticated compression techniques to overcome the storage and transmission problem. Image compression is the process of identifying the redundancies of an image or in the volume of images to remove them without degrading the quality of an image even after decompression. This process can refer as an "*image compression*".

Basically image compression categories are two; one is *lossy* compression technique and a second lossless compression technique which will be used according to their appliance is described in [2]. Medical images may have a great impact to diagnose the diseases and volume of images may obtained from the 3D imaging equipments such as CT, US or MRI methods, that will used to diagnose the diseases. To provide faithful treatment and maintain healthy communication with a client for that we may need to store and transmit it to the clients for further assistance, meanwhile we should provide huge storage capacity severs along with security and sometimes should provide a separate storage server for convenience of administration.

We can reduce the transmission time of images by adopting a compression technique. A large volume of data takes a long time for decompression, so we need to put-up some efforts to win the race. The CPU-GPU architectural design and their behavior have a discussion in section 2 [3], the GPGPU's computational environment and their structure has been discussed in section 3, the traditional compression techniques have discussed in the section 4, comparative study of medical images among the CPU and GPU in section 5 and finally hashed out on the results and conclusions in the section 6.

II. ARCHITECTURES OF CPU AND GPU

The CPU has often been called as the brain of the PC (Personal Computer). The PC has eventually been improved by the addition of the part, which is GPU, is called soul. The GPU is a special purpose processor for rendering the computer graphics, especially SIMD and MIMD operations. Especially the GPU is designed for rendering graphics calculations (by use of parallelism) and non-graphical calculations were performed by CPU details in [4].

CUDA processing flow:

1. Transfer the data from main memory to GPU memory
2. Pass the CPU instruct to GPU to process.
3. Each core in GPU executes parallelly.
4. Transfer the results from GPU memory to main memory

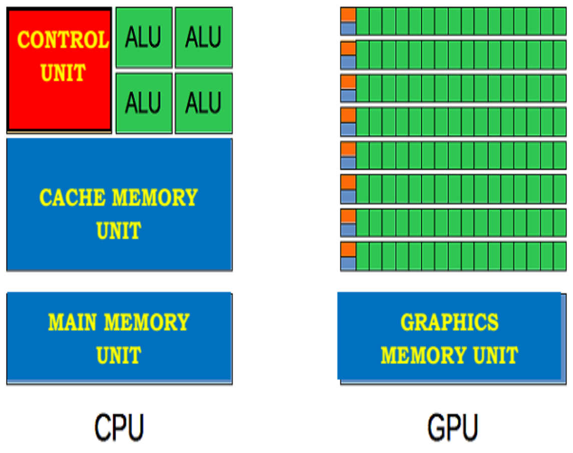


Figure 1(a) CPU (Single Core) and GPU (Multiple Cores) architectures

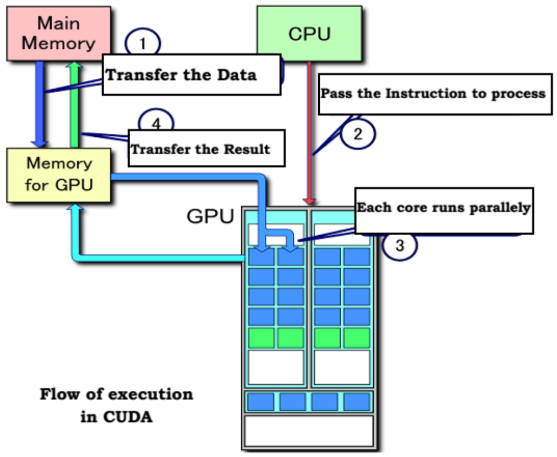


Figure 1 (b) Flow of execution in CUDA

Architecture, the CPU is designed with only a few cores, but which, having higher cache memory, which can be accessed any thread if it is needed, the structure of CPU and GPU architecture as shown in *figure-1* and the computational differences described in [7]. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. The GPU has the capability to process thousands of threads with more than 100 cores with the help of software 100x over the CPU. So the GPU can speed up the execution of a job with more power, cost effective and with a less time.

The latest operating systems like OpenCL (from Apple) and DirectCompute (from Microsoft) are supporting GPU environment. Especially, the GPU can process as many as multimedia applications such as Adobe Flash video, video translation formats to formats, image recognition, image compression, image segmentation, virus pattern matching, image analysis, video processing and signal processing which were have inherent parallel nature.

III. Introduction to GPGPU Computation Environment

The GPGPU becomes more popular after the year of 2001, which generally handles huge computations only for computer graphics. The GPU succeeds to achieve huge computations due to the parallelism and pipelining techniques details in [8]. Normally the traditional application's computations have been performed by the CPU; the computers allowing to use multiple graphics cards, or a large number of graphics chips; that is the reason to get ride the computations quickly. Not only GPU performs the computer graphics computations, but also can perform the traditional application's computations. Once the data migrates into graphical form and then the task is forward to GPU then the task gets results with a profound speedup. Eventually the GPU has been started to use to perform computations of normal applications. This has been done with the synchronization feature between the GPU-CPU. The overview of the CUDA internal structure described in the *figure 3* and compilation process in Fig.2.

Generally the CUDA is designed by the collection of a stream of multiprocessors, a global memory and for each of block having a shared memory with a dedicated local memory. As shown in *figure 3*, ultimately CPU is only hosting (kernels) for all grids which were in the GPU device. This device having with multiple grids, each grid consists of blocks and each block gets a number of threads, those were executed on a single stream of multiprocessor, which were designed with cores.

The wise programming languages are needed to synchronize the work on the complex platforms, especially the general purpose computing, that were implemented a framework with the support of OpenGL Shading Language (GLSL) and C for graphics (Cg). Newly CUDA and OpenCL frameworks were also introduced for general purpose tasks on Graphics Processing Unit. The OpenCL is an open standard for parallel programming is supported by AMD platform that will work on different devices, including GPUs, CPUs and field programming gate arrays (FPGAs) and also supports NIVIDIA and Intel, while CUDA GPU can be worked on NIVIDA environment described in [9].

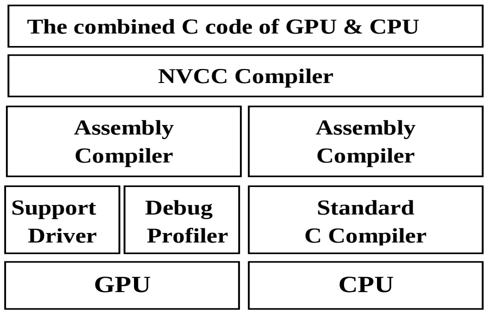


Figure 2 CUDA Compilation procedure

CPU-GPU Synchronization

The majority of parallel algorithms are needs be synchronized between the threads, whenever an atomic operation was needed to share among multiple threads explained in [10].

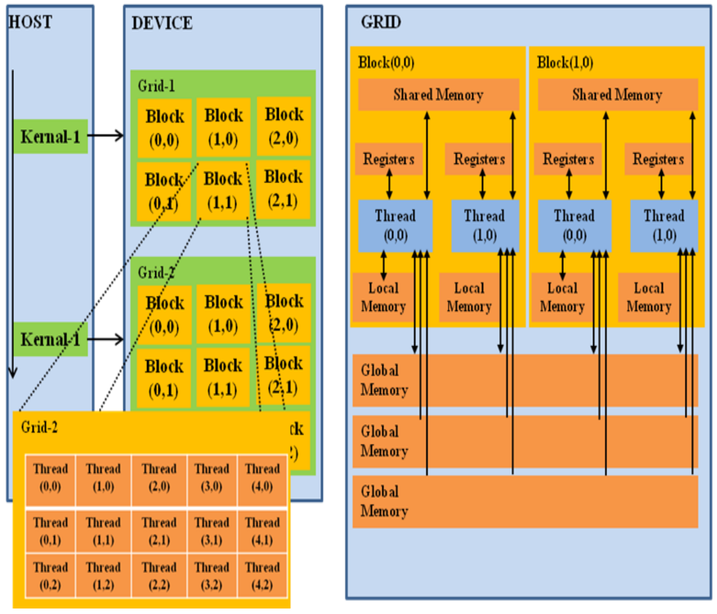


Figure 3 The parallel hierarchy of the CUDA architecture consists of grids (2), each grid with blocks (6), each block with threads (15) and with internal structure of the thread

To finish the job, other threads should wait until to finish the atomic operation; means that the atomic operations will be executed in serially but not in parallel. The synchronization is achievable by providing global memory to all threads. This will be done by executing multiple kernels, but which can be expensive because happens of overhead due to read and write, double buffering. Local and global synchronization provided by GPU architecture due to this, serialized operations were shuffled.
According to NVIDIA the global synchronization performance ratings as follows:
High: This performance is more than a hundred times, usually in iterative methods.
Medium: This performance is between 10 to 100 times.
Low: Only a few global or local synchronizations.
None: No synchronization.

The heterogeneous processing systems such as CPU-GPU can be shown the best performance by the technique of synchronization. CUDA programs, supports to execute a series of portions on the CPU, other portions which were supported parallel execution that will be on the GPU. We need to assign a right core for the right task. This has been vividly discussed by John Nickolas and William J. Dally in "*The GPU computing Era*" accepted by the group of NVIDIA and this article published by IEEE Computer Society in the year by 2010 [11].

These systems yield the best performance whenever the right portion (would be a serializable or parallelizable) is assigned to correct core (would be a CPU's or GPU's). Mostly a waste of time would be occurring when assigned parallelizable portions of CPU and serializable portions to the GPU.

IV. The Traditional Compression Techniques

Some of conventional approaches are discussed in the environment of CPU as well as in the environment of GPGPU under some hardware and software constraints. Most prominent techniques have been discussed in the following chapters, which are Huffman coding, Run length coding, Arithmetic coding and Wavelet transformation coding are commonly used in the traditional applications. 4.1 HUFFMAN CODING

Serial Coding Algorithm

Initially, the process of compression has been made a sequential approach or parallel approach [6] *prefix coding* technique which was a lossless data compression widely used in the area of compression where actually needed to concise the data. This was designed by David A. Huffman in the year of 1952 under the name of "*A Method for the Construction of Minimum-Redundancy Codes*" [5]. This algorithm deduces a table based on the frequency of occurrence of a source symbol (*i.e.* weight of the symbol). This table consists of an entropy encoding symbol, where the most commonly used symbols represent with fewer bits than less commonly used symbols. This can be producing a binary tree while in running and tree may have a leaf node or an internal node.

Parallel Coding Algorithm

This algorithm implemented based on MLP (Multi Level Progressive) approach [15]. The Parallel Random Access Machine (PRAM) is a straightforward method [16], which is embedded in the algorithm [8]. This allows reading and writing (limited) processors concurrently on the parallel memory; by entering a bit 1/0 in the completion register and it's initialized to '0' in each time step. All the pixels are coded as bits by disjoint and independently.

1. Assign a pixel to a processor for computing the encoding process along the code length of its pixel independently.
2. Instead of producing all bits for the first pixel, then for second pixel and so on at a time, every processor generating an output of its pixel's bit as one by one for every time step.
3. Once finished a processor's task, then it has to register the completion as write 1, then only a new pixel is allocated to that processor.
    - Since phase2 every processor fall into 3 categories:
        I. Completed
        II. In-Progress
        III. Untouched
    - Untouched pixels are redistributed to all available processors.

Example of reallocation process: There are 5 processors, P1, P2, P3, P4 and P5 with 15 pixels, 1, 2, 3,...., 15. This is shown in Figure 4.
Step1: **Initialize**, each processor obtains 3 pixels.
Step2: **Ends up a phase,** phase 1 ends when the processors P3, P5 completed with pixels 9, 15. By this time each processor produces 4 bits as output.
Step3: **Balance the load,** pixels 3, 10 and 15 are partially encoded by the processors P1, P2 and P5 and remaining untouched pixels are reallocated to P3 and P1 after finishing the P1 and P4 in phase 2.
Step4: **Active Processors,** in phase 3 the processor P2 comes into active so the processor P3's task is shifted to P2.

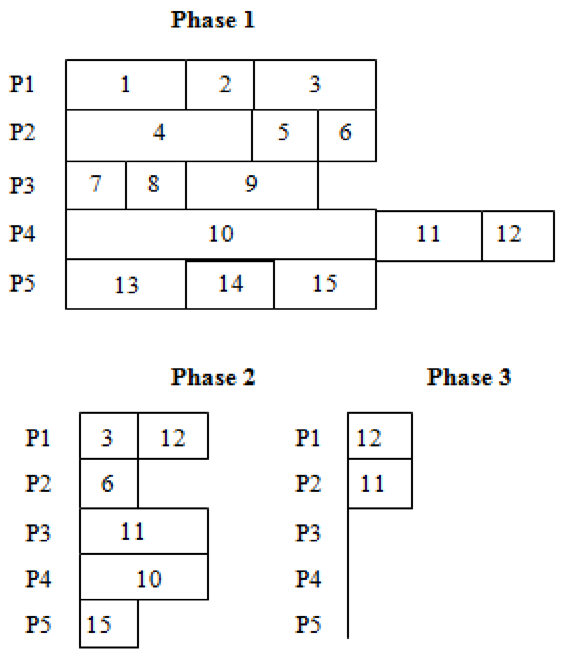


Figure 4. Reallocation coding

The worst case is *L* $log_2(2n/p)$, when the number of reallocations required to be coded in a level with *p* processors for *n* pixels and *L* denoted as the number of bits in the longest single code word. If number of *n* pixels are reduced to *p* processor's number in that case complexity is $long_2(n/p)$ so the number of phases required only $L.log_2(n/p)$.

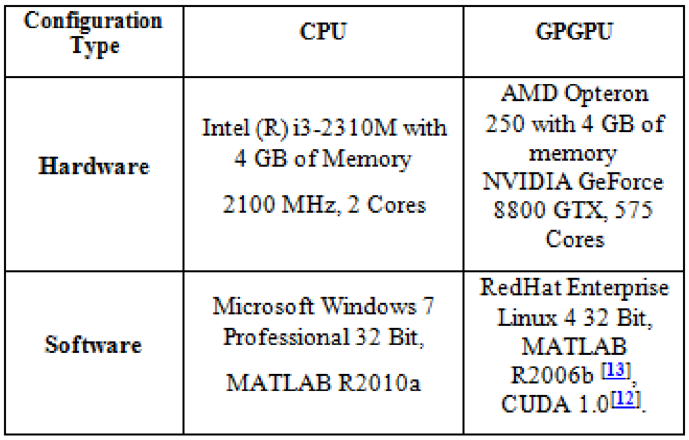
I. COMPARATIVE STUDIES ON GPU AND CPU ENVIRONMENT

Architectural Setup:

The following configuration has been used to implement the basic compression methods in the CPU as well as in the GPGPU environment, showing comparative configurations in table-1. In the floating point operations GPU's speed is several times faster than CPU's. The NVIDIA GeForce80xx has capable to perform up to 529GFlops and Intel 64-bit dual core CPU can make up to 32GFlops only. The NVIDIA has been using more than 0.7 billions of transistors in Quadro Fx5600, that is the reason why the GPU computationally more powerful [14].

Input Images:

This study has been explained the time consumption in two different environments of CPU and GPGPU. According to the table-1 the medical images have been compressed within less time over than CPU does. The CT-Brain image has 570x400 dimensions with size of 83KB and MRI-Head image has 650x367 with 72KB have been used as an input in the experiment as in shown Fig.5.

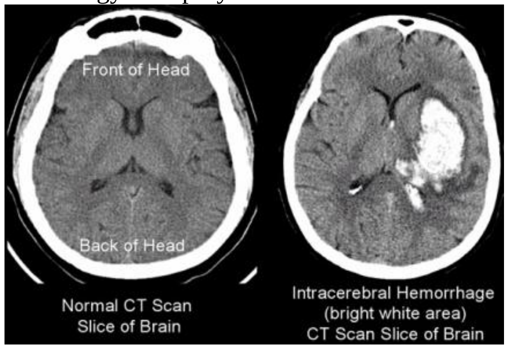


Configuration Type	CPU	GPGPU
Hardware	Intel (R) i3-2310M with 4 GB of Memory 2100 MHz, 2 Cores	AMD Opteron 250 with 4 GB of memory NVIDIA GeForce 8800 GTX, 575 Cores
Software	Microsoft Windows 7 Professional 32 Bit, MATLAB R2010a	RedHat Enterprise Linux 4 32 Bit, MATLAB R2006b [13], CUDA 1.0[12].

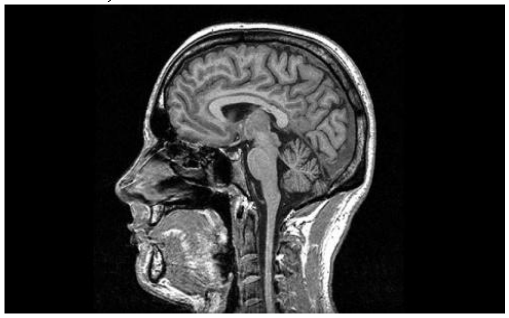
Time Comparison of Traditional Compression Techniques:

The Huffman coding technique implemented in two methods, one in serially and the other one in parallel. The serial coding developed in CPU environment and the parallel coding developed with the GPU environment. The parallel Huffman coding technique has been completed the image compression task in a short time than the traditional CPU time, the experimental results showed in the following tables.

The Huffman coding technique used to complete compression, 54.2597715 Sec for CT image and 44.3768928 Sec for MRI image of the serial coding on the CPU platform for the same input on the GPU platform with the parallel coding has taken 14.2534012 Sec for CT image and 11.1370235 Sec for MRI image, the speedup is 3.9 times faster than the CPU. The comparative bar chart has shown in Fig.6. Therefore the GPU technology could be complete the tasks on an average of 3.9 times faster than the CPU, so GPU is the most appropriated technology to deploy.



CT-Brain, 570x400 with 83KB



MRI-Head, 650x367 with 72KB

Figure 5 CT-Brain and MRI Input Images

Table-1 Compression techniques examined on CPU & GPGPU environment.

Huffman Coding	Serial Coding CPU (ms)	Parallel Coding GPU (ms)	Speed up
Brain CT Scan	54.2597715	14.2534012	3.80
Head MRI Scan	44.3768928	11.1370235	3.98

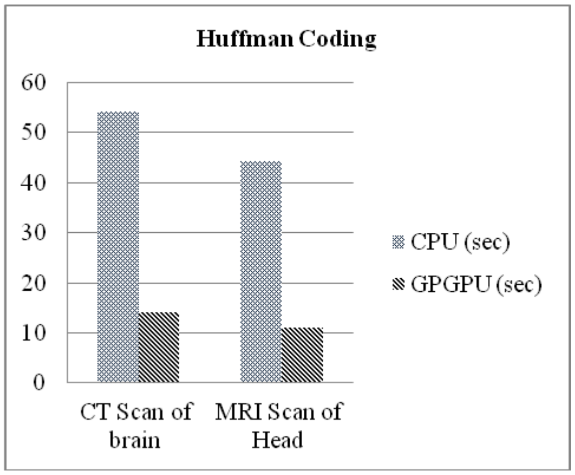


Figure 6 Image Compression Method's computational time in CPU & GPGPU environment

CONCLUSION

The image compression is extremely needed, especially in the field of medicine, the volumes of images are necessary to diagnose the disease. This makes huge computations in traditional CPU and it may the cause to degrade the system performance drastically. So where greatly required huge computations there we have an opportunity to divert the computational jobs to the GPU, which will have an ability to perform computations in parallel because it has an outstanding features like parallel processing and pipelining, the data can be bidirectional between GPU and CPU. This paper is presented that the traditional CPU computing power is enhanced by accumulating the GPU as a coprocessor. Some of the novel image compression techniques have been developed in the CPU environment with and without GPU support.

However, we observed according to the showed tables and bar charts, the profound differences in CPU and GPU environments especially speeding up the jobs. The experimental results proved that the GPU has accomplished compression with the speed of 3.9 times on an average than the CPU's speed and especially this depends on the system's configuration setup and on the paralyzed coding.

References

i. *Anders Eklund, Paul Dufort, Daniel Forsberg and Stephen LaConte, "Medical Image Processing on the GP: Past, Present and Future", 2013, Medical Image Analysis. Volume 17, Issue 8, Pages 1073–1094.*

ii. *M. Rabbani, P. W. Jones, "Digital Image Compression Techniques"., 1991, SPIE.*

iii. *M. Arora, "The architecture and evolution of CPU-GPU survey, University of California, San Diego, 2012.*

iv. *J. Nickolls and I. Buck. "NVIDIA CUDA software and GPU parallel computing architecture", In Microprocessor Forum, May 2007.*

v. *.D. A. Huffman, "A Method for the Construction of Minimum Redundancy Codes," Proceedings of the Institute of Radio Engineers 40 (1952), 1098-1101.*

vi. *P. Howard and J. Vitter. "Parallel Lossless Image Compression using Huffman and Arithmetic Coding", Information Processing Letters, Vol. 59, No. 2, pp. 65-73. 1996.*

vii. *Shuichi Asano, Tsutomu Maruyama and Yoshiki Yamaguchi, "Performance Comparison of FPGA, GPU and CPU in Image Processing", IEEE Xplore Conference Paper, August 2009, ISBN: 978-1-4244-3892-1.*

viii. *John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips, "GPU Computing", IEEE Proc.., Vol. 96, No. 5, May 2008, DOI: 10.1109/JPROC.2008.917757.*

ix. *S. Che, J. Li, J. Lach, and K. Skadron. "Accelerating compute intensive applications with GPUs and FPGAs". In Proceedings of the 6th IEEE Symposium on Application Specific Processors, June 2008.*

x. *Erik Smistad, Thomas L. Falch , Mohammadmehdi Bozorgi, Anne C. Elster, Frank Lindseth, "Medical image segmentation on GPUs – A comprehensive review", Elsevier Medical Image Analysis, 1361-8415, 2014.*

xi. *John Nickolls and William J. Dally, "The GPU computing Era", IEEE Computer Society, 0272-1732, 2010.*

xii. *www.nvidia.com/docs/io/116711/sc11-cuda-c-asics.pdf.*

xiii. *https://in.mathworks.com/help/distcomp/gpu-cuda-and-mex-programming.html.*

xiv. *WU En Hua, "State of the Art and Future Challenge on General Purpose Computation by Graphics Processing Unit", Journal of Software, vol. 15, no. 10, 2004, pp.1493~1504.*

xv. *P. G. Howard & J. S. Vitter, "New Methods for Lossless Image Compression Using Arithmetic Coding," Information Processing and Management 28 (1992), 765-779.*

xvi. *http://pages.cs.wisc.edu/~tvrdik/2/html/Section2.html*