Volume 2 No.8, AUGUST 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences

http://www.cisjournal.org

Accelerating Fast Fourier Transformation for Image Processing using Graphics Processing Unit

[1]Mohammad Nazmul Haque, [2]Mohammad Shorif Uddin
[1]Dept. of Computer Science & Engineering, Daffodil International University, Dhaka 1207, Bangladesh,
[2]Dept. of Computer Science & Engineering, Jahangirnagar University, Savar, Dhaka 1342, Bangladesh,
[1]nazmul@daffodilvarsity.edu.bd, [2]shorifuddin@gmail.com

ABSTRACT

In a number of imaging modalities, the Fast Fourier Transform (FFT) is being used for the processing of images in its frequency domain rather than spatial domain. It is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the frequency domain, while the input image is the spatial domain equivalent. In the frequency domain image, each point represents a particular frequency contained in the spatial domain image. The objective of the paper is to develop FFT based image processing algorithm to run under Central Processing Unit (CPU) and also Graphics Processing Unit (GPU) for compairing performance. The algorithm is developed in the c language and MATLAB for host machine. The CUFFT library is used for run under device to study the improved performance of reconstruction. GPUMat is used for running CUDA based C code using MATLAB. This work describes the acceleration of FFT-IFT algorithm on NVIDIA's GeForce G 103M based GPU and Intel® Core™2 Duo based CPU. The experimental results show a significant speedup of the algorithm in GPU than that of CPU based implementation. It is expected that this will accelerate many compute intensive image processing application.

Keywords— *Fast Fourier Transformation, GPGPU, CUDA, Image Processing, Frequency Domain Image Processing*

1. INTRODUCTION

The Fourier Transform (FT) is a mathematical operation used widely in many fields. In medical imaging it is used for many applications such as image filtering, image reconstruction and image analysis. It is an important image processing tool which is used to decompose an image into its sine and cosine components. The output of the transformation represents the image in the frequency domain, while the input image is the spatial domain equivalent. In the frequency domain image, each point represents a particular frequency contained in the spatial domain image [1] [2]. FFT based Image processing has reached a bottleneck where further speed improvement from the algorithmic perspective is difficult. But some real-time application demand faster Fourier transformation than what is currently available. Should we stop our journey for questing Faster Fourier Transformation technique due to algorithmic limitations? That triggers the mission for a faster way to compute the Fourier Transform based image processing technique.

The FFT is used in transform-domain speech, audio, image, and video compression. It has its own significance in the different fields. For such dynamic compute intensive and large data volume based applications the Graphics Processing Unit (GPU) based FFT algorithm can be the cost effective solution. Because, the GPU can process large volume data in parallel when working in single instruction multiple data (SIMD) mode. The increasing programmability of GPU has become another hot research topic, which includes its application on image processing.

This entire work is aimed to develop a strategy to compute the Fast Fourier Transform more efficiently and to reduce the time it takes for calculation. This mathematical transform makes processing of images with larger data size practical.

2. LITERATURE REVIEW

General-purpose computing on graphics processing units (often termed GPGPU or GPU computing) supports a broad range of scientific and engineering applications, including physical simulation, signal and image processing, database management, and data mining [3]. There are several excellent reviews of image reconstruction and numerical methods by many other authors. These include: Calvetti, Reichel & Zhang (1999) on iterative methods; Hansen (1994) on regularization methods; Molina et al. (2001) and Starck, Pantin & Murtagh (2002) on image reconstruction in astronomy; Narayan & Nityananda (1986) on the maximum-entropy method; O'Sullivan, Blahut & Snyder (1998) on an information-theoretic view; Press et al. (2002) on the inverse problem and statistical and numerical methods in general; and van Kempen et al. (1997) on confocal microscopy [4]. All these works used fourier transformation as their fundamental algorithms.

Medical imaging is one of the main application areas of FFT. Fast GPU computing applications require with computed tomography (CT) reconstruction which achieves a speedup of two orders of magnitude on the SGI Reality Engine in 1994 [5]. A wide variety of CT reconstruction algorithms have since been accelerated on graphics processors [5], [6], [7], [8] and the Cell

Volume 2 No.8, AUGUST 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

Broadband Engine [3]. In [6] the GPU is used to accelerate Simultaneous Algebraic Reconstruction Technique (SART), an algorithm that increases the quality of image reconstruction relative to the conventional filtered back-projection algorithm under certain conditions. SART, which requires significantly more computation than back-projection, becomes a viable clinical option when executed on the GPU. Research in this area has focused on accelerating the fast Fourier transform (FFT).

3. FFT IN IMAGE PROCESSING

Fourier Transform was a revolutionary concept to which it took mathematicians all over the world over a century to "adjust". Basically, the great contribution of Fourier Transformation states that any function can be expressed as the integral of sines and/or cosines multiplied by a weighting function. It works for any sort of complex functions, as long as it meets some mild mathematical conditions, it can be represented in such way. The function, expressed in a Fourier transform, can be reconstructed (recovered) completely via an inverse process [9]. This important property of Fourier transform allows working in the "frequency domain" and then returning to the spatial domain without losing any information[10].

a. Fourier Transform and its Inverse

The Fourier transform, *F(u)*, of a single variable, continuous function, *f(x)*, is defined by the equation

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-j2\pi ux}\, dx \qquad (1)$$

where $j = \sqrt{-1}$. Conversely, given *F(u)*, we can obtain *f(x)* by means of the inverse Fourier transform

$$f(x) = \int_{-\infty}^{+\infty} F(u) e^{j2\pi ux}\, du \qquad (2)$$

These two equations comprise the Fourier transform pair, which indicates the fact, mentioned before that the original function, can be recovered without loss of information. These equations can be easily extended to two variables, *u* and *v*:

$$F(u,v) = \iint_{-\infty}^{+\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy \qquad (3)$$

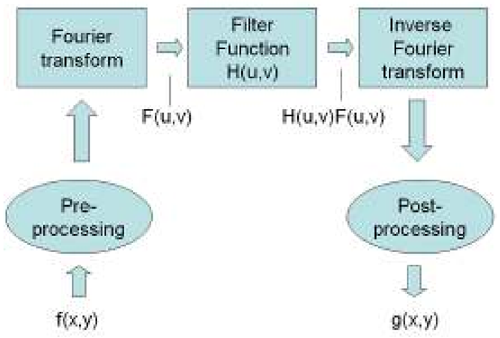

Figure 1: Basic steps for filtering in the frequency domain Similarly, the inverse transform,

$$f(x,y) = \iint_{-\infty}^{+\infty} F(u,v) e^{j2\pi(ux+vy)} dudv \qquad (4)$$

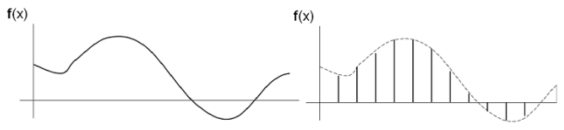

Figure 2: Left: a continuous function f(x). Right: the discrete function f(x)

The Fourier transform of an image, shows how signal intensity changes as a function of distance. It breaks down an image into its sine and cosine components with each point in the spatial domain image representing a particular frequency. This transformation has found its niche in image filtering, analysis, reconstruction and compression [9],[10].

Using fourier transformation, images in spatial domain can be converted to frequency domain. Once in spatial domain, images can be converted back to spatial domain with inverse fourier transformation.

b. Discrete Fourier Transform (DFT)

Since the digital images are model by discrete functions, we are more interested on the discrete Fourier transform. The one dimension of discrete fourier transform is given by the equation

$$F(u) = \frac{1}{M} \sum_{X=0}^{M-1} f(x) e^{-\frac{j2\pi ux}{M}} \qquad (5)$$

for $u = 0,1,2, \dots\dots, M-1$

Note that *f(x)* in (5) is a discrete function of one variable, while the *f(x)'s* in (1) (2) are continuous functions. See the Figure.2. Similarly, given *F(u)*, we can obtain the original discrete function f(x) by inverse DFT:

Volume 2 No.8, AUGUST 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

$$f(x) = \sum_{u=0}^{M-1} F(u) e^{\frac{j2\pi ux}{M}} \qquad (6)$$

for $x = 0,1,2, \dots \dots, M-1$

The Discrete Fourier Transform (5) and its inverse (6) is the foundation for the most frequency based image processing.

Extension of the One-dimensional DFT and its inverse to two dimensions is straightforward. The discrete Fourier transform of an image function *f(x, y)* of size $M \times N$ is given by the equation:

$$F(u,v) = \frac{1}{M} \sum_{x=0}^{M-1} \left[\sum_{y=0}^{N-1} f(x,y) e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})} \right] \qquad (7)$$

The Discrete Fourier Transform is a summation operation. The number of terms in the summing up is the same as the number of sampled points. The Discrete Fourier Transform is frequently evaluated for each data sample, and can be regarded as extracting particular frequency components from a signal.

c. Fast Fourier Transform

J.W. Cooley and J.W. Tukey are given credit for bringing the FFT to the world in their paper: "An algorithm for the machine calculation of complex Fourier Series," Mathematics Computation, Vol. 19, 1965, pp 297-301. The FFT is based on the complex DFT, a sore sophisticated version of the real DFT. These transforms are named for the way each represents data that is, using complex numbers or using real numbers.

The FFT is an algorithm for calculating the complex DFT. The real DFT transforms an N point time domain signal into two point frequency domain signals. The time domain N/2+1 signal is called just that: the time domain signal. The two signals in the frequency domain are called the real part and the imaginary part, holding the amplitudes of the cosine waves and sine waves, respectively [**11**].

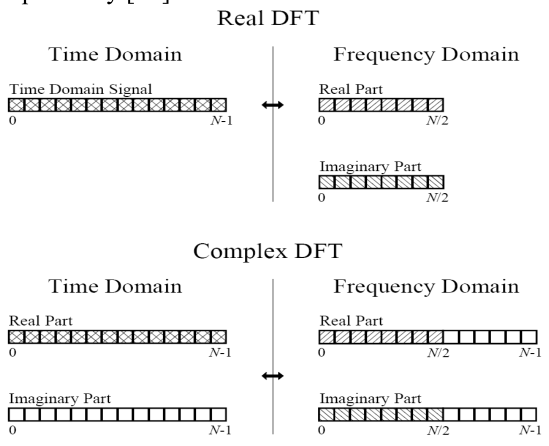

Figure 3: Compares how the real DFT and the complex DFT store data

The FFT algorithm developed in this section is based on the successive doubling method. Now we express Eq.(3) in the form

$$F(u) = \frac{1}{M} \sum_{X=0}^{M-1} f(x) W_M^{ux} \qquad (8)$$

Where $W_M = e^{-j2\pi/M}$ so $W_M^{ux} = e^{-j2\pi ux/M}$ and the number of points M is assumed to be the power of 2, like $M = 2^n$ with n being a positive integer [10].

In case of DFT, the computing the 1-D discrete Fourier transform of M points using Eq.(7) directly requires on the order of M^2 multiplication/addition operations. The FFT accomplishes the same task on the order of M $\log_2 M$ operations. When the problem grows bigger the greater computational advantage is achieved. The 2-D fast Fourier can be obtained by successive passes of a 1-D transform algorithm.

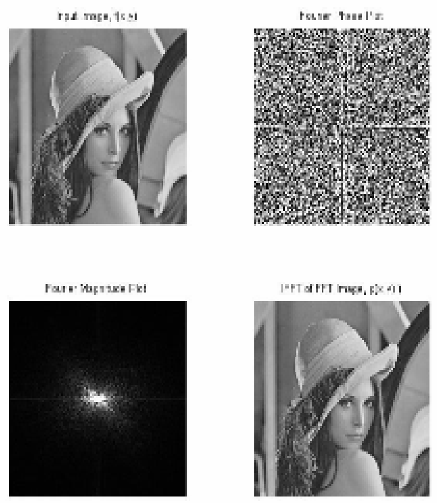

Figure 4: Fast Fourier Transform and its Inverse on Image

The input signal is broken in half by using an interlaced decomposition. The N/2 even points are placed into the real part of the time domain signal, while the N/2 odd points go into the imaginary part. An N/2 point FFT is then calculated, requiring about one-half the time as an N point FFT. The resulting frequency domain is then separated by the even/odd decomposition, resulting in the frequency spectra of the two interlaced time domain signals. These two frequency spectra are then combined into a single spectrum [10]. The FFT has another advantage besides raw speed. The FFT is calculated more precisely because the fewer number of calculations results in less round-off error. This can be demonstrated by taking the FFT of an arbitrary signal, and then running the frequency spectrum through an Inverse FFT. This reconstructs the original time domain signal, except for the addition of round-off noise from the calculations.

4. THE FFT IN IMAGE PROCESSING

Image reconstruction using FFT/IFFT is done in two steps; firstly the 2D- IFFT of the data is computed then data are shifted to center for display the image.

Algorithm: Image_Reconstruction

Input: *Spectral Domain Data*
Output: *Reconstructed Spatial Domain Image*

```
Step 1: Read in the Spectral Domain DATA.
Step 2: Apply IFFT in (x,y) Direction
Step 3: FFT shift
Step 4: Image Display
```

In practical image reconstruction, there are some other pre-processing activities that must be accomplished before the application of IFFT [12],[13],[11]. The reconstruction algorithm is expressed herewith.

5. SOFTWARE & TOOLS USED

a) Compute Unified Device Architecture (CUDA)

In November 2006, NVIDIA introduced CUDA™, a general purpose parallel computing architecture – with a new parallel programming model and instruction set architecture – that leverages the parallel compute engine in NVIDIA GPUs to solve many complex computational problems in a more efficient way than on a CPU[14].

It is very easy to use for programmers, since it introduces a small number of extensions to C language, in order to provide parallel execution. Another important features are flexibility of data structures, explicit access on the different physical memory levels of the GPU, and a good framework for programmers including a compiler, CUDA Software Development Kit (CUDA SDK), a debugger, a profiler, and CUFFT and CUBLAS scientific libraries[15],[16],[17].

b) Graphics Processing Unit (GPU)

NVidia graphics card architecture consists of a number of so-called streaming multiprocessors (SM). Each one includes 8 shader processor (SP) cores, a local memory shared by all SP, 16384 registers, and fast ALU units for hardware acceleration of trancendental functions. A global memory is shared by all SMs and provides capacity up to 4 GB and memory bandwidth up to 144 GB/s (to July 2010). FERMI architecture introduces new SMs equipped with 32 SPs and 32768 registers, improved ALU units for fast double precission floating point performance, and L1 cache[14],17].

c) CUDA Program Structure

The GPU is seen as a compute device to execute a portion of an application, a function for example, that:

- Has to be executed many times;
- Can be isolated as a function;
- Works independently on different data.

The execution of a typical CUDA program is illustrated in Figure 2.2. The execution starts with host (CPU) execution. When a kernel function is invoked, the execution is moved to a device (GPU)[15], where a large number of threads are generated to take advantage of abundant data parallelism. All the threads that are generated by a kernel during an invocation are collectively called a grid. Figure 5 shows the execution of two Girds of threads [18].

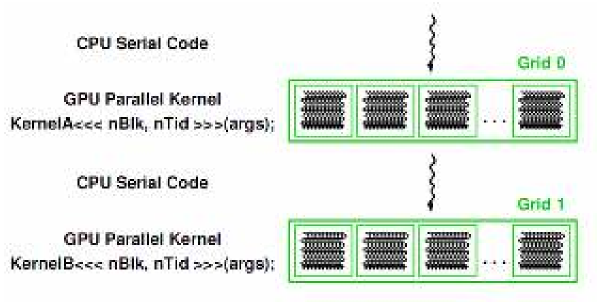

Figure 5: Execution of a CUDA program

d) CUFFT - FFT for CUDA

The CUFFT library provides a simple interface for computing parallel FFTs on an NVIDIA GPU, which allows users to leverage the floating- point power and parallelism of the GPU without having to develop a custom, GPU-based FFT implementation.

FFT Libraries typically vary in terms of supported transform sizes and data types. For example, some libraries only implement Radix-2 FFTs, restricting the transform size to a power of two, while other implementations support arbitrary transform sizes. The current version of the CUFFT library supports the following features:

- 1D, 2D, and 3D transforms of complex and real-valued data
- Batch execution for doing multiple transforms of any dimension in parallel
- Transform sizes up to 64 million elements in single precision and upto 128 million elements in double precision in any dimension, limited by the available GPU memory
- In-place and out of place transforms for real and complex data

- Double-precision transformas on campatible hardware (GT200 and later GPUs)

Support for streamed execution, enabling simultaneous computation together with data movement [16].

e) GPUmat - GPU toolbox for MATLAB

GPUmat allows standard MATLAB code to run on GPUs. The execution is transparent to the user as shown in the following example:

A=rand(100,GPUsingle); % A is on GPU memory **B=rand(100,GPUsingle); % B is on GPU memory** **C = A+B; % executed on GPU.** **D = fft(C); % executed on GPU**
Executed on GPU

A = single(rand(100)); **% A is on CPU memory**
B = double(rand(100)); **% B is on CPU memory**
C = A+B; % executed on CPU.
D = fft(C); % executed on CPU

Executed on CPU

Every MATLAB variable has been converted to the GPUsingle class ("*A = rand(100)*" becomes "*A = rand(100, GPUsingle)*"). From here the code remains as the original one, i.e. after a specific declaration any instruction follows the classic MATLAB syntax but any operation on GPUsingle, like A + B in the example, is executed on the GPU [19].

Benefits and key features:

- GPU computational power can be easily accessed from MATLAB without any GPU knowledge.
- MATLAB code is directly executed on the GPU.
- GPUmat speeds up MATLAB functions by using the GPU multi-processor architecture.
- Existing MATLAB code can be ported and executed on GPUs with few modifications.
  - GPU resources are accessed using MATLAB scripting language.
  - Supports real/complex, single/double precision data types.[20]

6. Experimental Setup

The experiment is divided into two sections. First is simulation of performance for FFT-IFT on different sized images.

a. Hardware Requirements

All experiments are done using both CPU and GPU. The configurations for them are listed in Table 1 and Table 2.

Table 1: CUDA Device Configuration at Experiment

Features	Specification
Name:	GeForce G 103M
CUDA Driver Version:	3.2
Total Global memory:	521601024 bytes
#Multiprocessors	1 (MP)
#Cores	8 (Cores)
Total Constant memory:	65536 bytes
Total Shared memory/block:	16384 bytes
Total registers/block:	8192
Warp size:	32
Max number of threads per block:	512
Max sizes of each block dimension:	512 x 512 x 64
Max sizes of each grid dimension:	65535 x 65535 x 1
Maximum memory pitch:	2147483647 bytes
Texture alignment:	256 bytes
Clock rate:	1.60 GHz

Table 2: Host Machine Configuration at Experiment

Feature	Specification
System Model:	Compaq Presario CQ40 Notebook PC
System Manufacturer:	Hewlett-Packard
Operating System:	Windows 7 Ultimate 32-bit
Processor:	Intel(R) Core(TM)2 Duo CPU
#CPU	2
Clock Speed:	2.00GHz
Memory:	2048MB RAM

b. Required Software and Tools

The experiment requires some software and tools for programming and documenting purpose. Following table lists up all used software and tools:

Table 3: Required Software and Tools

Software	Version	Purpose
NVIDIA GPU Computing SDK	3.2	Software Development Kit required for NVIDIA's GPU
CUDA Toolkit	3.2	Toolkit for CUDA programming
MATLAB R2010a	7.10	Simulation and Programming
CUFFT	2.3	CUDA capable FFT library
GPUmat	0.27	Wrapper for MATLAB to run CUDA Program

Volume 2 No.8, AUGUST 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

C. Experimental Image Data

For smooth running the process of FFT based image Reconstruction using GPU we have used three images. Images are chosen different resolution for figure out the performance of FFT and IFFT on CPU and GPU based implementation.

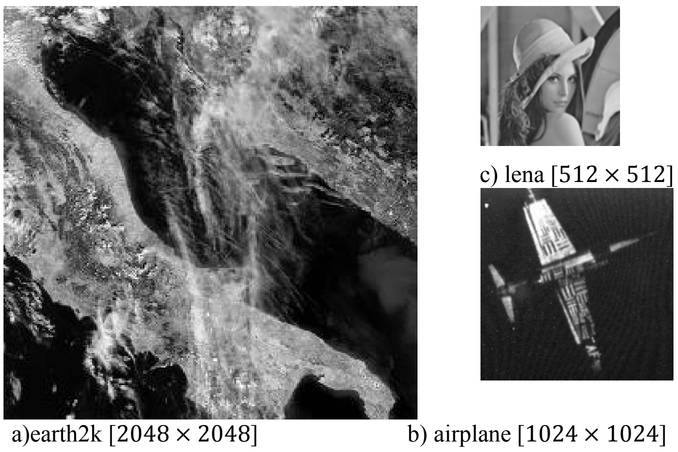

Figure 6: Images used for Experiments

7. Experimental Results

GPU and CPU performance results are obtained by computing FFT of experimental images in MATLAB and measuring the execution time using commands tic and toc. All CPU iterations are measured and averaged over 100 iterations.

a) Lena image Reconstruction

Spatial resolution of lena image is 512x512. Both CPU based and GPU based reconstruction is done 100 times for measuring runtime.

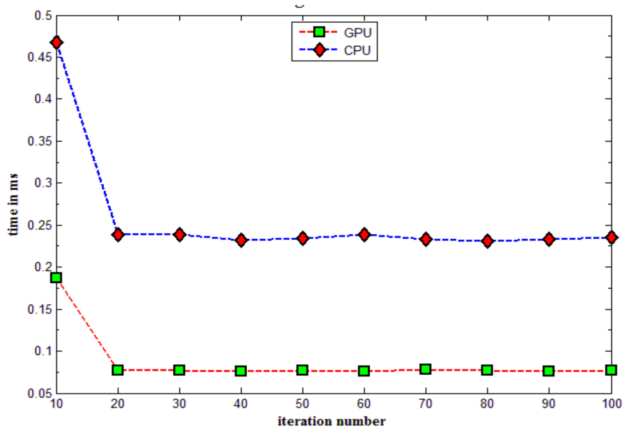

Figure 7: GPU vs. CPU Performance of FFT based Image Processing for lena image

The simulation shows an average speed up of GPU vs. CPU by a factor of 3.1x which is 310% speedy than CPU in Figure 8. Minimum Speedup achieved by GPU vs CPU is 2.5 times, whereas maximum was 3.2x.

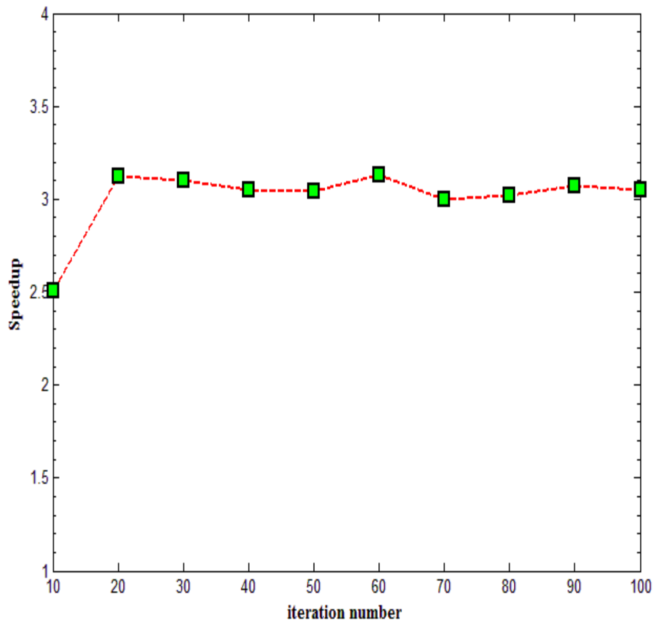

Figure 8: FFT based Image Processing Speedup by GPU vs CPU for lena image

b) Airplane image Reconstruction

Spatial resolution of airplane image is 1024x1024. Both CPU based and GPU based reconstruction is done 100 times for measuring runtime.

The simulation shows an average speed up of GPU vs. CPU by a factor of 4.1x which is 410.00% speedy than CPU. Minimum Speedup achieved by GPU vs CPU is 4.05 times, whereas maximum was 4.128x.

GPU runtime was almost linear for airplane image.

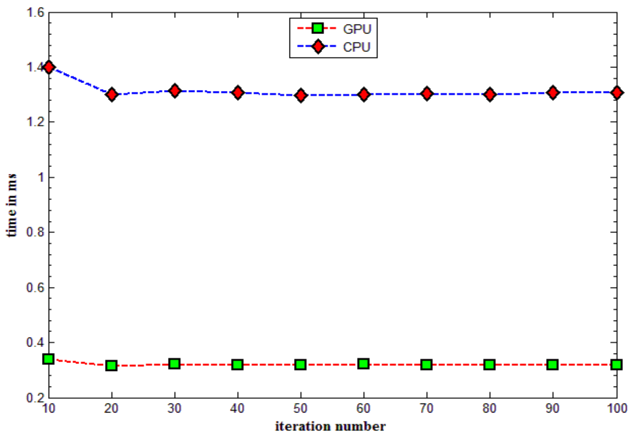

Figure 9: GPU vs. CPU Performance of FFT based Image Processing for airplane image

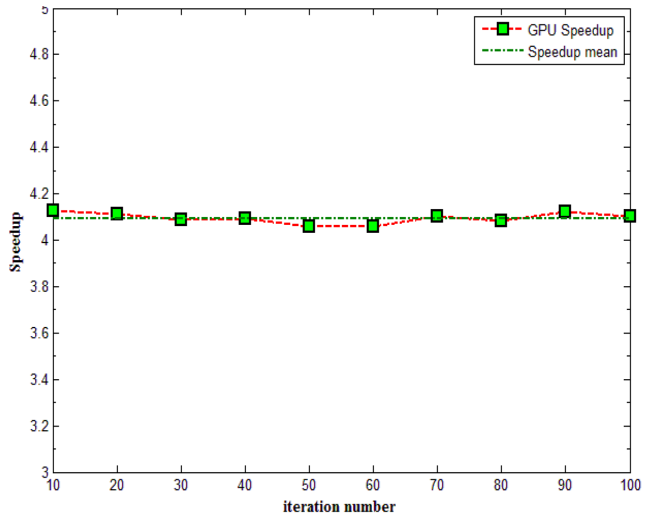

Figure 10 : FFT based Image Processing Speedup by GPU vs CPU for airplane image

c) Earth2k image Reconstruction

Spatial resolution of *earth2k* image is *2048x2048*. Both CPU based and GPU based reconstruction is done 100 times for measuring runtime.

The simulation shows an average speed up of GPU vs. CPU by a factor of 4.349x which is 434.90% speedy than CPU. Minimum Speedup achieved by GPU vs CPU is 4.29 times, whereas maximum was 4.585x.

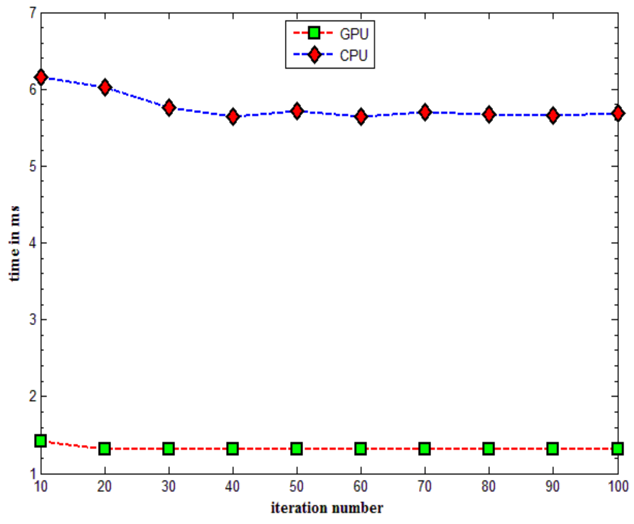

Figure 11: GPU vs. CPU Performance of FFT based Image Processing for earth2k image

Table 4: Image FFT-IFT Summary for 100 iterations

Image Name	Spatial Resolution	FFT Time *(ms)*		IFT Time *(ms)*		Total Time (in ms)		Speedup factor of GPU
		CPU	*GPU*	*CPU*	*GPU*	*CPU*	*GPU*	
lena	512x512	0.0392	0.0132	0.0410	0.0127	0.0802	0.0259	3.10x
airplane	1024x1024	0.0973	0.0305	0.1103	0.0297	0.2075	0.0602	3.45x
earth2k	2048x2048	0.4078	0.0924	0.4504	0.0792	0.8582	0.1715	5.00x

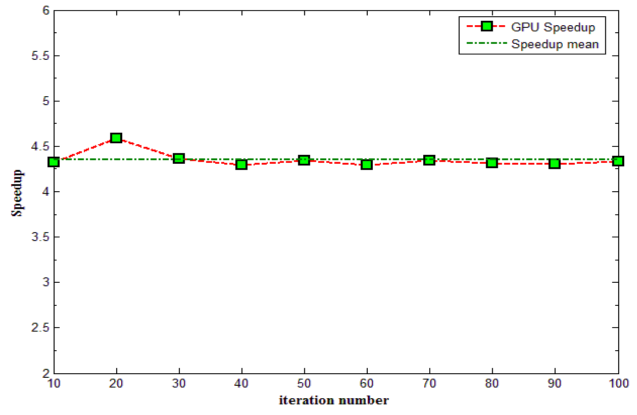

Figure 12 : FFT based Image Processing Speedup by GPU vs CPU for airplane image

d) Simulation Summary

The simulation is done for 100 iterations. FFT and IFT time for 100 iterations is measured and tabulated. Total reconstruction time is equals to the sum of FFT time and IFT time. Then speedup factor is calculated and put into the summary Table 4 for all test images.

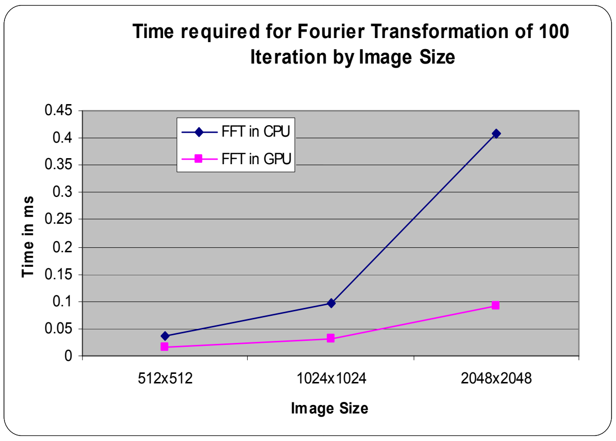

Figure 13: FFT Performance by Image Size

Volume 2 No.8, AUGUST 2011 ISSN 2079-8407
Journal of Emerging Trends in Computing and Information Sciences
©2010-11 CIS Journal. All rights reserved.
http://www.cisjournal.org

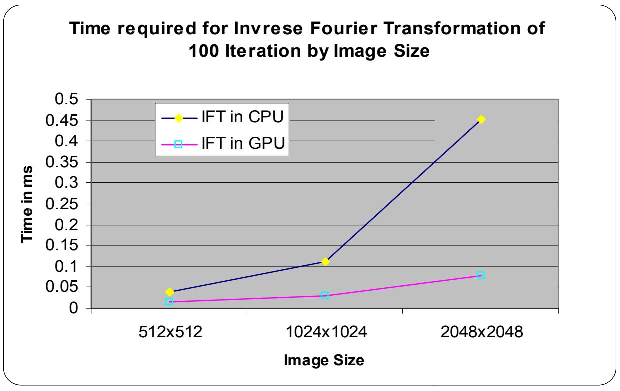

Figure 14: IFFT Performance by Image Size

From the Figure 14 it has been observed that the boost in performance for FFT is gradually enhancing. This performance for both CPU and GPU is nearly same for image size 512x512. While the image size increasing the performance of CPU lags behind that of GPU. The scenario is similar for IFFT performance in GPU vs CPU in Figure 15.

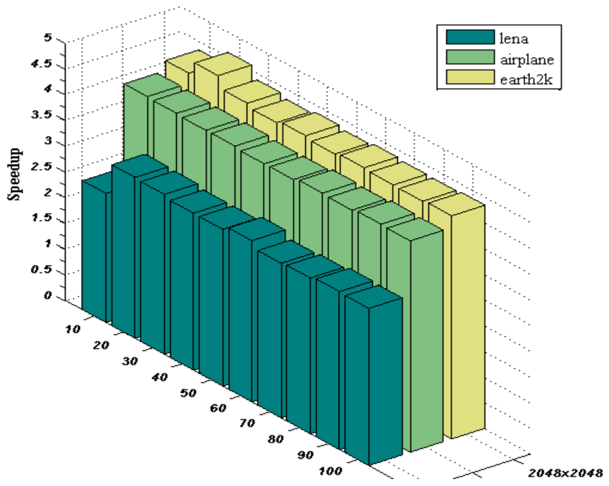

Figure 15: GPU Speedup of FFT based Image Processing by Image Size

As a result the FFT based image processing performance of GPU over CPU was augmented gradually while the image size increased.

8. LIMITATIONS AND FUTURE WORK

The Performance improvement of the work is impressive. However, the main purpose of this work was to determine the absolute speed difference computational efficiency between CPU and GPU implementations using CUDA.

Our work currently works only on data that resides in GPU memory. External memory algorithms based on the hierarchical algorithm can be designed to handle larger data. Computation can also be performed on multiple GPUs. However, for both of these scenarios, data must be transferred between GPU and system memory, which can dramatically lower the performance.

It can possible to do other processing, such as de-noising and de-blurring on spectral domain data before output the spatial domain image. The same process can be applied to reconstruction of 3D holographic image, MR-Image reconstruction, 3D visualization of objects and Super Resolution Imaging. The work can be ported to real-time visualization of MR image that is very important for envision situation of sensitive internal organs while operating a patient

9. CONCLUSION

Besides the performance advantage of using a GPU over a CPU for FFT based image processing, there are other advantages as well. In some imaging device, the CPU can be preoccupied with time-critical tasks such as controlling the data acquisition hardware. In this case, it is beneficial to use the GPU for image processing, leaving the CPU to do data acquisition. Moreover, because of the GPU is free of interrupts from the operating system, it results better performance than interrupt driven CPU. The rate of increase in performance of GPUs is expected to outshine that of CPUs in the next few years, increasing the demand of the GPU as the processor of choice for image processing.

REFERENCES

[1] V. Jagtap, "Fast Fourier Transform Using Parallel Processing for Medical Applications," MSc Thesis, Biomedical Engineering, University of Akron, Ohio, 2010.

[2] (2011, Apr.) Fourier transform From Wikipedia, the free encyclopedia. [Online]. http://en.wikipedia.org/wiki/Fourier_transform

[3] O. Bockenbach, M. Knaup, and M. Kachelrie, "Implementation of a cone-beam backprojection algorithm on the Cell Broadband Engine processor." in SPIE Medical Imaging 2007: Physics of Medical Imaging, 2007.

[4] D. C. no-Díez, D. Moser, A. Schoenegger, S. Pruggnaller, and A. S. Frangakis., "Performance evaluation of image processing algorithms on the GPU," Journal of Structural Biology, vol. 164, no. 1, pp. 153-160, 2008.

[5] K. Mueller, F. Xu, and N. Neophytou, "Why do commodity graphics hardware boards (GPUs) work so well for acceleration of computed tomography?" SPIE Electronic Imaging 2007 , 2007.

[6] K. Mueller and R. Yagel., "Rapid 3-D cone-beam reconstruction with the simultaneous algebraic

reconstruction technique (SART) using 2-D texture mapping hardware," in IEEE Transactions on Medical Imaging, vol. 19, 2000, pp. 1227-1237.

[7] K. Chidlow and T. M. oller, "Rapid emission tomography reconstruction," in Int'l Workshop on Volume Graphics, 2003.

[8] X. Xue, A. Cheryauka, and D. Tubbs, "Acceleration of uro-CT reconstruction for a mobile C-Arm on GPU and FPGA hardware: A simulation study," in SPIE Medical Imaging, 2006.

[9] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 3rd, Ed. Prentice Hall, 2008.

[10] S. W. Smith, The Scientist and Engineer's Guide to Digital Signal Processing. California Technical Publishing.

[11] D. Moratal, A. Vallés-Luch, L. Martí-Bonmat, and M. Brummer, "k-Space tutorial: an MRI educational tool for a better understanding of k-space," Biomedical Imaging and Intervention Journal, 2008.

[12] L. Chaˆari, J. .-C. Pesquet, A. Benazza-Benyahia, and P. Ciuciu, "Autocalibrated Parallel MRI Reconstruction in the Wavelet Domain," in IEEE International Symposium on Biomedical Imaging, Paris, France, 14-17 May, 2008, pp. 756-759.

[13] G. Schultz, et al., "K-Space Based Image Reconstruction of MRI Data Encoded with Ambiguous Gradient Fields," in Proc. of International Society for Magnetic Resonance in Medicine, 2011.

[14] General-Purpose computation on Graphics Processing Units. [Online]. http://www.gpgpu.org/

[15] CUAD Tutorial: The Golden Energy Computing Organization. [Online]. http://geco.mines.edu/tesla/cuda_tutorial_mio/index.html

[16] (2010) CUDA(TM) CUFFT Library 3.1. [Online]. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf

[17] NVIDIA CUDA Compute Unified Device Architecture Programing Guide. [Online]. http://developer.download.nvidia.com/compute/cuda/1_0/NVIDIA_CUDA_Programming_Guide_1.0.pdf

[18] D. B. Kirk and W.-m. W. Hwu, Programming Massively Parallel Processors:A Hands-on Approach. Burlington, MA 01803, USA: Elsevier Inc, 2010.

[19] (2010) AccelerEyes - MATLAB GPU Computing. . [Online]. http://www.accelereyes.com

[20] (2010, Dec.) GPUmat: GPU Power in MATLAB. [Online]. http://www.gp-you.org/index.php?option=com_content&view=article&id=46&Itemid=54