FAST MEDICAL IMAGE RECONSTRUCTION TECHNIQUE USING GRAPHICS PROCESSING UNIT

BY
Mohammad Nazmul Haque
ID: 101-25-149

This Thesis Presented in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science and Engineering

Supervised By
Dr. Mohammad Shorif Uddin
Professor
Department of Computer Science & Engineering
Jahangirnagar University

DAFFODIL INTERNATIONAL UNIVERSITY
DHAKA, BANGLADESH
JULY 2011

APPROVAL

This Thesis titled **"Fast Medical Image Reconstruction Technique using Graphics Processing Unit"**, submitted by Mohammad Nazmul Haque to the Department of Computer Science and Engineering, Daffodil International University, has been accepted as satisfactory for the partial fulfillment of the requirements for the degree of M.Sc. in Computer Science and Engineering and approved as to its style and contents. The presentation has been held on 30th July 2011.

BOARD OF EXAMINERS

Dr. Syed Akhter Hossain — **Chairman**
Professor and Head
Department of Computer Science & Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Yousuf Mahbubul Islam — **Internal Examiner**
Professor
Department of Computer Science & Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Md Kabirul Islam — **Internal Examiner**
Associate Professor
Department of Computer Science & Engineering
Faculty of Science & Information Technology
Daffodil International University

Dr. Hafiz Md. Hasan Babu — **External Examiner**
Professor
Department of Computer Science & Engineering
University of Dhaka

DECLARATION

We hereby declare that, this thesis work has been done by me under the supervision of **Dr. Mohammad Shorif Uddin, Professor, Department of CSE** Jahangirnagar University. We also declare that neither this project nor any part of this work has been submitted elsewhere for award of any degree or diploma.

Supervised by:

_____

Dr. Mohammad Shorif Uddin
Professor
Department of Computer Science & Engineering
Jahangirnagar University

Submitted by:

_____

Mohammad Nazmul Haque
ID: 101-25-149
Department of Computer Science & Engineering
Daffodil International University

ACKNOWLEDGMENT

First I express my heartiest thanks and gratefulness to almighty Allah for His divine blessing makes me possible to complete this thesis work successfully.

I fell grateful to and wish our profound our indebtedness to **Dr. Mohammad Shorif Uddin, Professor, Department of CSE,** Jahangirnagar University, Savar, Dhaka, deep Knowledge & keen interest of my supervisor in the field of image processing and computer vision influenced me to carry out this thesis. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior draft and correcting them at all stage have made it possible to complete this thesis.

I would like to express my heartiest gratitude to **Dr. Syed Akhter Hossain**, Professor and Head**,** Department of Computer Science & engineering, for his kind help to finish my thesis and also to other faculty member and the staff of CSE department of Daffodil International University.

I would like to thank my entire course mate in Daffodil International University, who took part in this discuss while completing the course work.

Finally, I must acknowledge with due respect the constant support and patients of my parents, wife, daughter, relatives, friends and colleagues.

ABSTRACT

In this thesis work, we present a fast magnetic resonance images (MRI) reconstruction algorithm taking advantage of the prevailing general purpose graphics processing unit (GPGPU) programming paradigm. In a number of medical imaging modalities, the Fast Fourier Transform (FFT) is being used for the reconstruction of images from acquired raw data. 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 is in frequency domain, while the input image is in the spatial domain. In the frequency domain image, each point represents particular frequency information in the spatial domain image. The objective of the research is to develop an algorithm to run under CPU and also in GPU for the Fast Fourier Transform so that it will performs the Fast Fourier Transform (FFT) as well as Inverse Fourier Transformation (IFT) in much faster way. The algorithm is developed in both C language and MATLAB environments. The CUFFT library is used to run under device to study the improved performance of reconstructions. GPUMat is used to running CUDA code in MATLAB. This work describes the acceleration of MRI reconstruction algorithm on NVIDIA's GeForce G 103M based GPU and Intel® Core™2 Duo based CPU. Our FFT based reconstruction algorithm shows that GPU based MRI reconstruction achieved significant speedup compared to the CPUs for medical applications at a cheaper cost.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 Overview

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**].

1.2 Problem Statement

When Magnetic Resonance Imaging (MRI) machines take an image of the human body, the output is in the form of raw data. The Fourier Transform is used to reconstruct the image from this raw data. In medical imaging devices specialized hardware or CPUs are used to reconstruct images from acquired raw data. When the raw data size is relatively small, it takes moderate time to reconstruct an image. But, as the raw data size continues increasing, the time for processing the reconstruction increases as well.

Image reconstruction has reached a bottleneck where further speed improvement from the algorithmic perspective is difficult.

But some clinical practices such as real-time surgery monitoring demand faster reconstruction than what is currently available.

Should we stop our journey for questing faster image reconstruction technique due to algorithmic limitations?

That triggers the mission for a faster way to compute the Fourier Transform based image reconstruction technique.

1.3 Related Work

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**].

Medical imaging was one of the first GPU computing applications, with computed tomography (CT) reconstruction achieving 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 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. By contrast, MRI reconstruction on the GPU has not been studied extensively. Research in this area has focused on accelerating the fast Fourier transform (FFT).

1.4 Goals and Objectives of the Work

The goal of this thesis work basically revolves around the use of the Fourier Transform for reconstruction of an image in MRI machines. 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 reconstruction of images for 3D visualization in real-time.

1.5 Scope of Work

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 reconstruction.

1.6 Organization of the Report

This thesis report is organized in following chapters:

In "Chapter 2: Image Processing", a general introduction to image processing is provided.

"Chapter 3: The Fourier Transformations in Image Processing" discussed various FT techniques and concludes with the Fast Fourier Transformation (FFT) which is widely used for image processing for its some advantages

Next "Chapter 4: Magnetic Resonance Imaging" introduces the MR-Imaging and reconstruction techniques and explores the FFT based reconstruction.

"Chapter 5: Graphics Processing Unit" comes with the features, advantages and tools for GPU based programming technique which leads to GPU based image reconstruction.

The experimental results and their interpretations are discussed in next chapter titled "Chapter 6: Results and Discussions". This chapter explores the outcome of this intended thesis work.

Not but least, "Chapter 7: Conclusion" commend the thesis report by mentioning its limitations and some future scopes.

CHAPTER 2
IMAGE PROCESSING

Images are pictures: a way of recording and presenting information visually. Pictures are important to us because they can be an extraordinarily effective medium for the storage and communication of information. We use photography in everyday life to create a permanent record of our visual experiences. In showing photograph, we avoid the need for a lengthy, tedious and in all ambiguous verbal description of what were seen.

2.1 Image Acquisition

Image acquisition is a process of sensing our surroundings and then representing the measurements that are made in the form of image. Camera uses a lens to focus part of the visual environment onto a sensor. The most important characteristics of a lens are:

- magnifying power and
- Light gathering capacity.

Magnification factor as we know:

$$m = \frac{image\_size}{object\_size} \quad (1)$$

Which is in term of distance is:

$$\frac{v}{u} = \frac{image\_size}{object\_size} \quad (2)$$

2.2 Image Representation

The term image refers to a two-dimensional continuous light-intensity function, denoted by $\boldsymbol{i(x,y)}$, where x and y represent spatial coordinates and the value or amplitude of i at any point $\boldsymbol{(x,y)}$ gives the intensity (brightness) of the image at that point. To be suitable for computer processing, an image function $\boldsymbol{i(x,y)}$ must be discretized both spatially and in amplitude (brightness). Discretization of spatial coordinates (x,y) is called image sampling and amplitude discretization is called gray-level quantization [**9**].

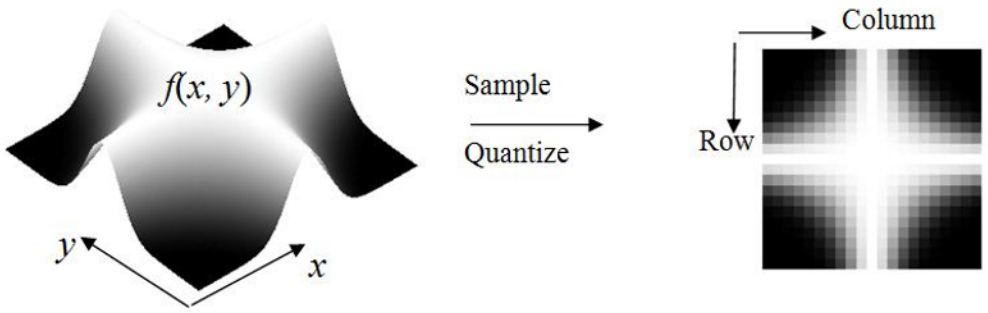


Figure 2-1: Coordinate System

2.3 Image Processing

Image processing is any form of signal processing for which the input is an image, such as a photograph or video frame; the output of image processing may be either an image or, a set of characteristics or parameters related to the image. Most image-processing techniques involve treating the image as a two-dimensional signal and applying standard signal-processing techniques to it. Image processing spans a sequence of three steps:

1. Import an image with an optical scanner or directly through digital photography.
2. Manipulate or analyze the image in some way. This stage can include image enhancement and data compression, or the image may be analyzed to find patterns that aren't visible by the human eye. For example, meteorologists use image processing to analyze satellite photographs.
3. Output the result. The result might be the image altered in some way or it might be a report based on analysis of the image.

In a nutshell, Image processing is the application of signal processing techniques to the domain of images – two-dimensional signals such as photographs or video **[10]**.

A digital image is an array of real or complex numbers represented by a finite number of elements. These elements are referred to as picture elements, image elements, pels, and pixels. Pixel is used to denote the elements of a digital image **[9]**. For example, a monochrome digital image $f(x, y)$ is a 2D array of luminance values,

$$f(x,y) = \begin{pmatrix} f(0,0) & f(0,1) & . & f(0,N-1) \\ f(1,0) & f(1,1) & . & f(1,N-1) \\ . & . & . & . \\ f(M-1,0) & f(M-1,1) & . & f(M-1,N-1) \end{pmatrix} \quad (3)$$

with $0 \leq f(x,y) \leq L$, and typically $L = 255$..

"Digital image processing is the use of computer algorithms to perform image processing on digital images" **[2]**. "Digital image processing refers to processing digital images by means of a digital computer**"[2]**. It encompasses processes whose inputs and outputs are images and extract attributes from images in addition, recognize individual objects. Low-level processes involve primitive operations such as image processing to reduce noise, contrast enhancement, and image sharpening. Mid-level processes on images involve tasks such as segmentation, description of those objects to reduce them to a suitable computer processing form, also classification or recognition of individual objects. Finally, higher-level processing involves "making sense" of collection of recognized objects, as in image analysis, and, performing the cognitive functions normally associated with human vision.

2.3.1 Fundamental Steps in Image Processing

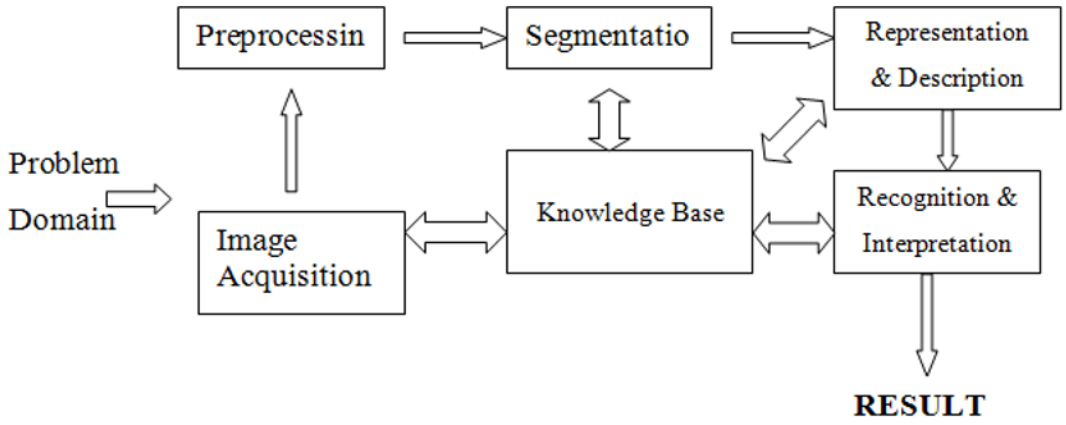


Figure 2-2: Fundamental steps in Image processing

- Image acquisition: to acquire a digital image
- Image preprocessing: to improve the image in ways that increases the chances for success of the other processes.

- Image segmentation: to partitions an input image into its constituent parts or objects.
- Image representation: to convert the input data to a form suitable for computer processing.
- Image description: to extract features that result in some quantitative information of interest or features that are basic for differentiating one class of objects from another.
- Image recognition: to assign a label to an object based on the information provided by its descriptors.
- Image interpretation: to assign meaning to an ensemble of recognized objects.
- Knowledge about a problem domain is coded into an image processing system in the form of a knowledge database.

2.3.1.1 Digital Image Processing Tree

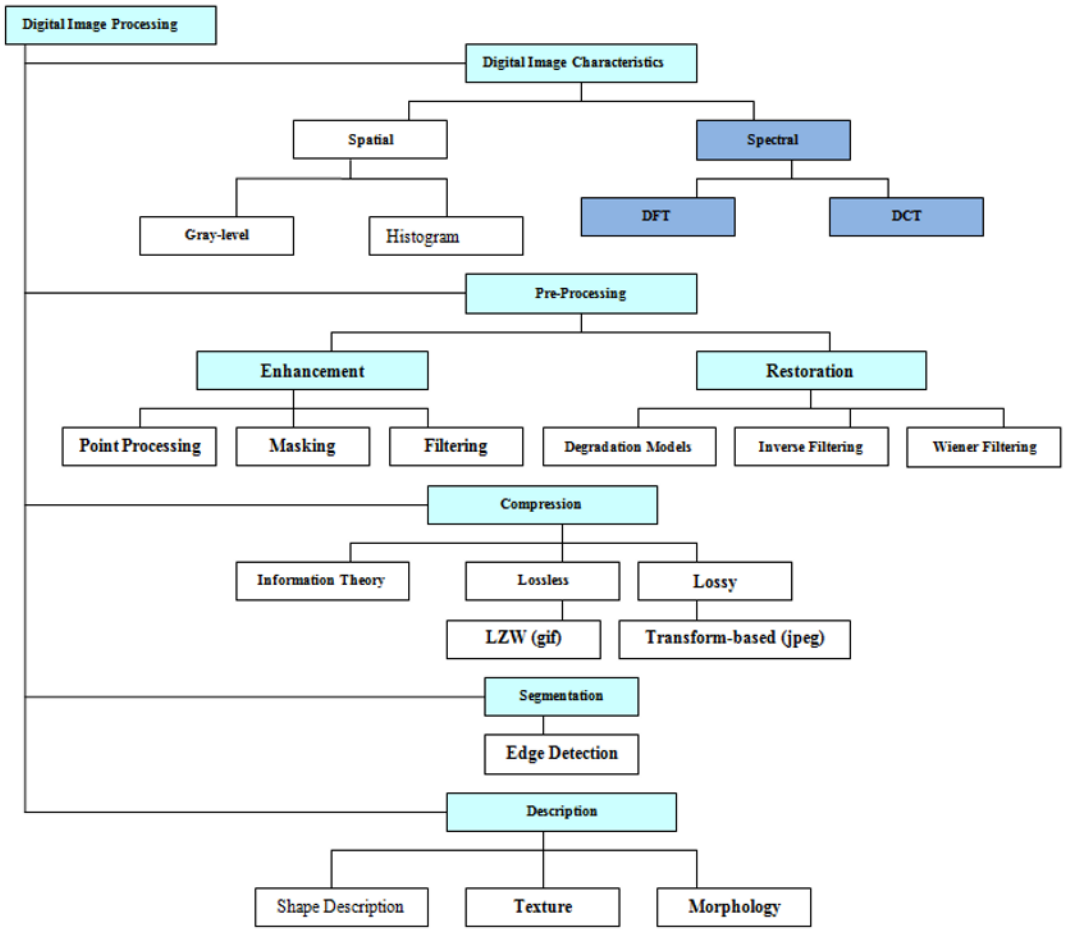


Figure 2-3: Digital Image Processing Tree

2.4 Image Representation Schemes

According to the number and nature of pixel values digital images can be classified as follows:

- binary
- grayscale
- color

2.4.1 Binary image

A binary image is a digital image that has only two possible values for each pixel. Typically the two colors used for a binary image are black and white though any two colors can be used [2]. The color used for the object(s) in the image is the foreground color while the rest of the image is the background color.

Binary images are also called *bi-level* or *two-level*. This means that each pixel is stored as a single bit (0 or 1). The names *black-and-white*, *B&W*, monochrome or monochromatic are often used for this concept, but may also designate any images that have only one sample per pixel, such as grayscale images. In Photoshop parlance, a binary image is the same as an image in "Bitmap" mode.

Binary images often arise in digital image processing as masks or as the result of certain operations such as segmentation, thresholding, and dithering. Some input/output devices, such as laser printers, fax machines, and bi-level computer displays, can only handle bi-level images.

2.4.1.1 Advantages

- Easy to acquire: simple digital cameras can be used together with very simple frame stores, or low-cost scanners, or thresholding may be applied to grey-level images.
- Low storage: no more than 1 bit/pixel, often this can be reduced as such images are very amenable to compression (e.g. run-length coding).
- Simple processing: the algorithms are in most cases much simpler than those applied to grey-level images.

2.4.1.2 Disadvantages

- Limited application: as the representation is only a silhouette, application is restricted to tasks where internal detail is not required as a distinguishing characteristic.
- Does not extend to 3D: the 3D nature of objects can rarely be represented by silhouettes. (The 3D equivalent of binary processing uses voxels, spatial occupancy of small cubes in 3D space).
- Specialized lighting is required for silhouettes: it is difficult to obtain reliable binary images without restricting the environment. The simplest example is an overhead projector or light box

2.4.2 Grayscale image

A grayscale digital image is an image in which the value of each pixel is a single sample, that is, it carries only intensity information. Images of this sort, also known as black-and-white, are composed exclusively of shades of gray, varying from black at the weakest intensity to white at the strongest.

Grayscale images are distinct from one-bit black-and-white images, which in the context of computer imaging are images with only the two colors, black, and white (also called *bi-level* or *binary images*). Grayscale images have many shades of gray in between. Grayscale images are also called monochromatic, denoting the absence of any chromatic variation (ie: no color).

Grayscale images are often the result of measuring the intensity of light at each pixel in a single band of the electromagnetic spectrum (e.g. infrared, visible light, ultraviolet, etc.), and in such cases they are monochromatic proper when only a given frequency is captured. But also they can be synthesized from a full color image. Usually, a grayscale image has 256 gray values (8 bit representation).

2.4.3 Color Image

A (digital) color image is a digital image that includes color information for each pixel. For visually acceptable results, it is necessary to provide three color channels for each pixel, which are interpreted as coordinates in some color space such as RGB, HSV, YCbCr etc. For RGB color images, for each color there are 256 gray values

[23]. A color image is usually stored in memory as a raster map, a two-dimensional array of small integer triplets; or (rarely) as three separate raster maps, one for each channel.

2.5 Applications of Image Processing

1. Office automation: optical character recognition; document processing; logo and icon recognition; identification of address on envelop; etc.
2. Industrial automation: automatic inspection of industrial parts; non-destructive testing; automatic assembling; process related to VLSI manufacturing; PCB checking; robotics; oil and natural gas exploration; seismography; process control application; etc.
3. Bio-medical: ECG, EEG analysis; cytological, histological and stereological applications; automated radiology and X-ray image analysis; mass-screening of medical images such as chromosome slides for detection of various diseases, mammograms, cancer smears; CAT, MRI, PET, and other tomographic images; routine screening of samples; 3D scene reconstruction and analysis, etc.
4. Remote sensing: Tracking and surveying of natural resources; geographical mapping, prediction and estimation of agricultural crops, hydrology, forestry, mineralogy; estimation of urban growth, urban planning; environment and pollution control; cartography, registration of satellite images with terrain maps; monitoring traffic along roads, docks and airfields; food and fire control; etc.
5. Scientific applications: high energy physics; bubble chamber and other forms of track analysis; etc.
6. Criminology: Human face registration, detection and recognition, finger print identification, Dermot glyphic analysis, forensic investigation; etc.
7. Astronomy and space application: restoration of images suffering from geometric and photometric distortions; computing close-up picture of planetary surfaces; etc.
8. Meteorology: short-term weather forecasting, long –term climatic change detection from satellite and other remote sensing data; cloud pattern analysis; etc.

9. Information technology: facsimile image transmission, video conferencing and videophones; etc.
10. Entertainment and consumer electronics: HDTV; multimedia and video-editing; etc.
11. Printing and graphic arts: color fidelity in desktop publishing; art conservation and dissemination; etc.
12. Military applications: missile guidance and detection; target identification; navigation of pilot less vehicles; reconnaissance; and range finding; etc. **[9]**

CHAPTER 3

THE FOURIER TRANSFORMATIONS IN IMAGE PROCESSING

3.1 Background

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 [**11**]. This important property of Fourier transform allows working in the "frequency domain" and then returning to the spatial domain without losing any information.

3.2 Fourier Transform and its Inverse

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

$$F(u) = \int_{-\infty}^{+\infty} f(x) e^{-j2\pi ux}\, dx \qquad (4)$$

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

$$f(x) = \int_{-\infty}^{+\infty} F(u) e^{j2\pi ux}\, du \qquad (5)$$

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*:

$$F(u,v) = \iint_{-\infty}^{+\infty} f(x,y) e^{-j2\pi(ux+vy)} dxdy \qquad (6)$$

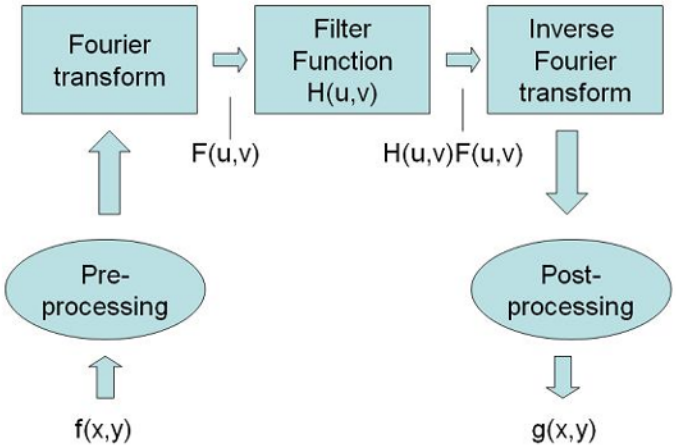


Figure 3-1: Basic steps for filtering in the frequency domain

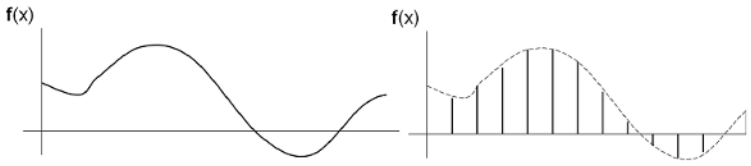


Figure 3-2: Left: a continuous function f(x). Right: the discrete function f(x)

Similarly, the inverse transform,

$$f(x,y) = \iint_{-\infty}^{+\infty} F(u,v)e^{j2\pi(ux+vy)}dudv \tag{7}$$

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 [**11**].

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.

Fourier Transformation is vital in the manipulation of Magnetic Resonance images as it allows analyzing the way in which the magnetic field around the subject is being changed.

3.3 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

$$F(u) = \frac{1}{M}\sum_{X=0}^{M-1} f(x)e^{-\frac{j2\pi ux}{M}} \qquad \text{for } u = 0,1,2,\dots\dots, M-1 \qquad (8)$$

Note that f(x) in (3) 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:

$$f(x) = \sum_{u=0}^{M-1} F(u)e^{\frac{j2\pi ux}{M}} \qquad \text{for } x = 0,1,2,\dots\dots, M-1 \qquad (9)$$

The Discrete Fourier Transform (3) and its inverse (4) 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 a image function *f(x, y)* of size $M \times N$ is given by the equation:

$$F(u,v) = \frac{1}{M}\sum_{x=0}^{M-1}\left[\sum_{y=0}^{N-1} f(x,y)e^{-j2\pi(\frac{ux}{M}+\frac{vy}{N})}\right] \qquad (10)$$

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.

3.4 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 soresophisticated 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**].

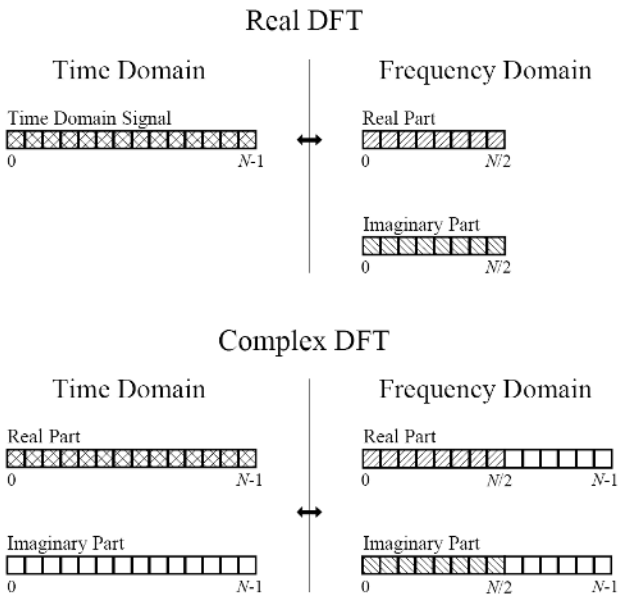


Figure 3-3: Compares how the real DFT and the complex DFT store data

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

$$F(u) = \frac{1}{M}\sum_{X=0}^{M-1} f(x) W_M^{ux} \tag{11}$$

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.(3) 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.

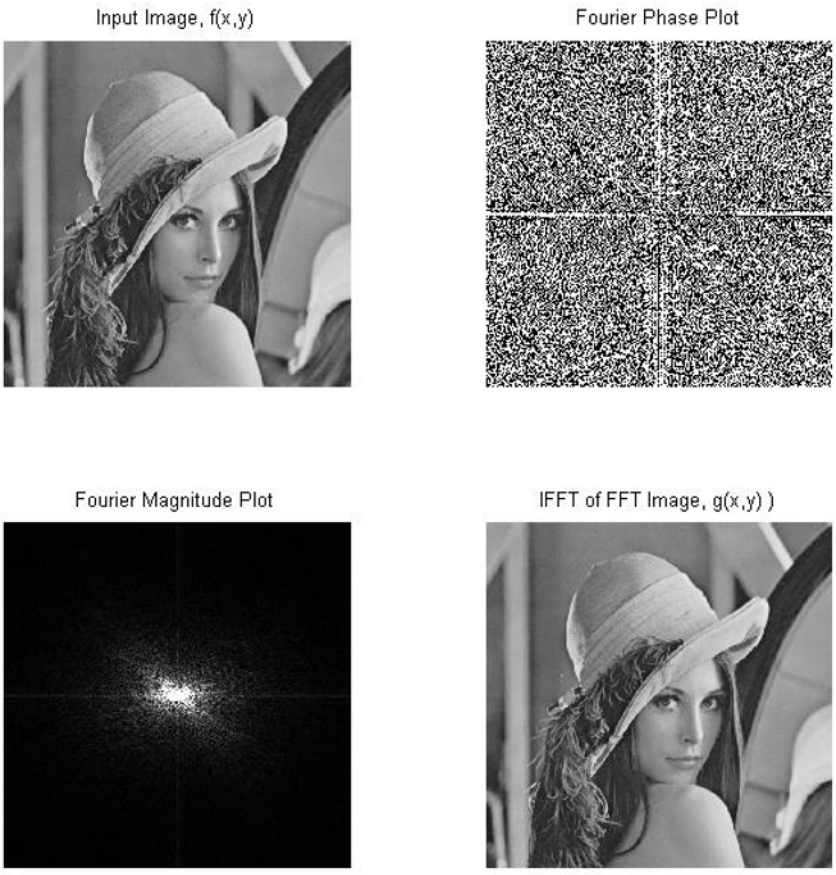


Figure 3-4: Fast Fourier Transform and its Inverse on Image

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. 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.

CHAPTER 4
MAGNETIC RESONANCE IMAGING

4.1 Introduction

In this work we dealt with Magnetic Resonance Imaging images to reconstruct the spectral domain image from raw machine data, the frequency domain data.

Magnetic resonance imaging (MRI) is an imaging technique used primarily in medical settings to produce high quality images of the inside of the human body. MRI is based on the principles of nuclear magnetic resonance (NMR), a spectroscopic technique used by scientists to obtain microscopic chemical and physical information about molecules. The technique was called magnetic resonance imaging rather than nuclear magnetic resonance imaging (NMRI) because of the negative connotations associated with the word nuclear in the late 1970's. MRI started out as a tomographic imaging technique, that is it produced an image of the NMR signal in a thin slice through the human body. MRI has advanced beyond a tomographic imaging technique to a volume imaging technique [**12**].

4.2 Overview of MRI

Magnetic Resonance Imaging (MRI) is based on the principles of tomographic imagine technique [**13**], [**14**], [**15**]. It is used primarily in medical fields to produce images of the internal structure of human body. MRI provides more diagnostic information than any of the existing imaging techniques, moreso, it does not involve the use of ionizing radiation hence, free from associated harmful effects known with other imaging techniques [**16**].

MRI imaging equation can be express as a two dimensional entity given as

$$S(k_x, k_y) = f[I(x, y)] \tag{12}$$

Where *f* represent spatial information encoding scheme [**13**]. If *f* is invertible, a data consistent *I* can be obtained from the inverse transformation such that

$$I(x, y) = f^{-1}S(k_x, k_y) \tag{13}$$

The desired image intensity function *I(x,y)* is a function of: Relaxation times, *T1* , *T2* and $T2^*$; spin density, ρ; Diffusion coefficients *D* and so on **[17]**. Using function notation, this can be written as

$$I = f[T1, T2, T2^*, \rho, D] \quad (14)$$

Generally, *T1* and *T2* are two independent processes and happen simultaneously. T1 is called spin lattice relaxation, because the energy from this process is released to the surrounding tissue (lattice) **[14]**, **[17]**, **[13]**. T1 happens along the z-component axis and its value is always greater than the spin-spin relaxation *T2*. The relationship between protons and their immediate surroundings (molecules) is described by the spin-spin relaxation *T2* and it happens along *x-y* plane.

4.3 MR-Imaging principles

There are three principles for MR-imaging. They are Spin physics, basic principles and Fourier transform principles. Among them, Fourier transform imaging methods are usually used in most imaging machines. Our work only concerns on FT bases imaging method.

4.3.1 Fourier Transform Principles

Fourier transform tomographic imaging is the most commonly used MRI method utilized today. It uses a type of magnetic field gradient, called a phase encoding gradient, in addition to the slice selection and frequency encoding gradients.

A simplified Fourier transform imaging sequence includes a 90^o slice selective pulse, a slice selection gradient pulse, a phase encoding gradient pulse, a frequency encoding gradient pulse, and a signal. The magnitude and duration of the magnetic field gradients are represented by the pulses.

A typical imaging sequence would start by turning on the slice selection gradient, while applying the slice selection RF pulse. Once the pulse is done, the slice selection gradient turns off, and the phase encoding gradient is turned on. After that is done, the frequency encoding gradient is turned on. At this point, a signal is recorded. This process is repeated 128 to 256 times, varying the magnitude of the phase encoding gradient each time, in order to obtain sufficient data, free induction decays or signals, for creating an image **[18]**.

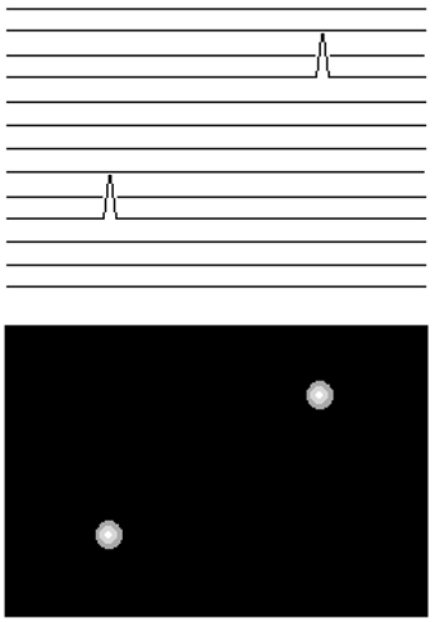


Figure 4-1: Tomographic image created from Fourier transformed data

Before actually creating the image, however, the signals must be Fourier transformed. This is done first in the direction in which the spins are located to extract the frequency domain information. Then it is done in the phase encoding direction to obtain information about the spin locations in that direction. The FT data finally becomes an image when the intensities of the data peaks are converted into intensities of pixels. This creates a tomographic image, shown in Figure 4-1.

4.4 MRI Data Acquisition and K-Space Data

4.4.1 MRI Data Acquisition

MRI data acquisition involves three major steps namely

a) **Gz, Slice selection by the use of Gz gradient:** This select axial slice in the object to be imaged.
b) **Gy, Phase encoding using the Gy gradient:** This creates rows with different phases.
c) **Gx, Frequency encoding using the Gx gradient:** This will create columns with different frequencies.

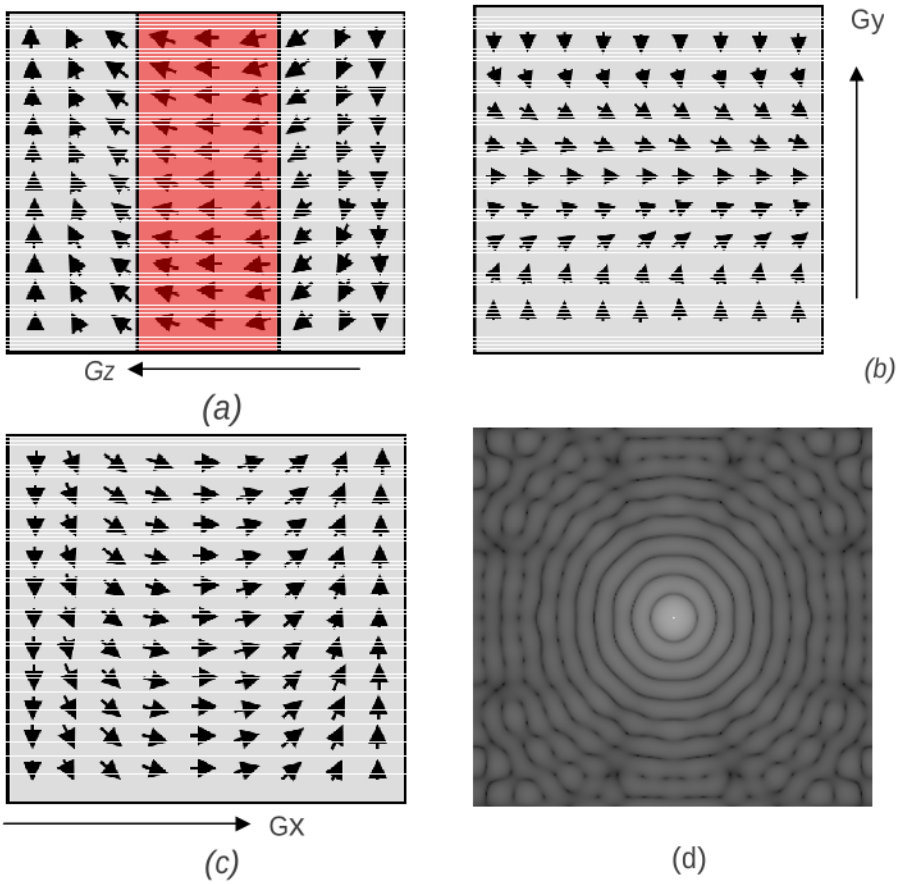


Figure 4-2: K-Space Data formation: a)G_z Gradient data Filling, b) G_y Gradient Data Filling, c) G_x Gradient Data Filling and d) K-Space Data

4.4.2 K-Space Data

MR imaging systems collect data over time. The data that are captured during MR imaging are called k-space data or, simply, raw data. Typically, the data are collected by using quadrature detection, which provides both real and imaginary k-space data. K-Space data include useful information but can be interpreted only after they are translated into images with the Fourier transform method (Figure 4-3). The K-space data contains all the necessary information required to reconstruct an image. Also, it gives a comprehensive way for classification and understanding of the imaging properties and method of reconstruction **[13]**, **[19]**, **[20].**

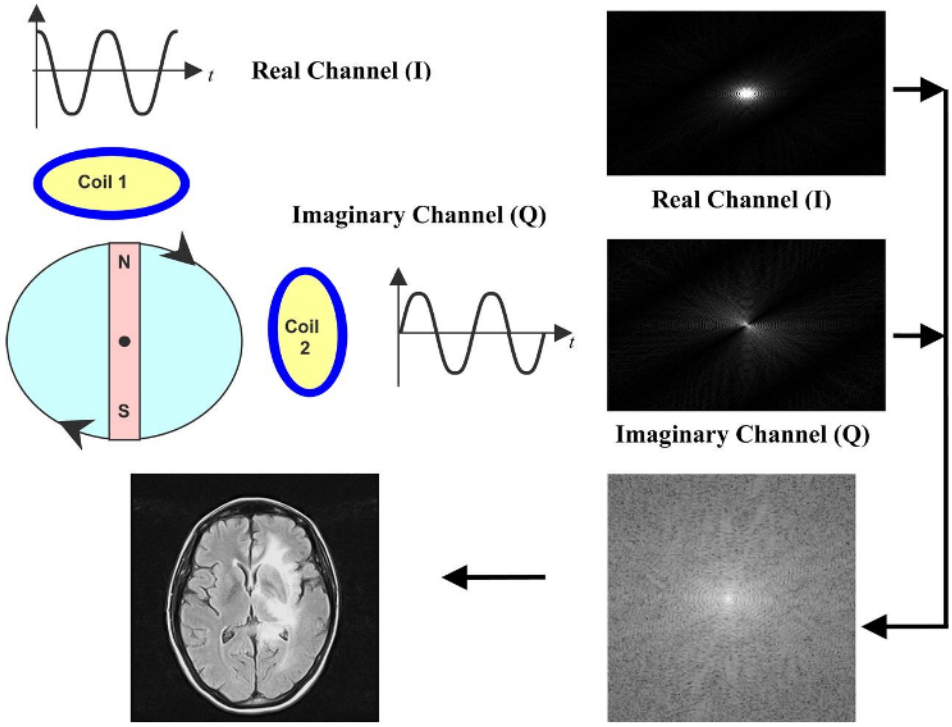


Figure 4-3: Schematic of the collection and combination of imaginary MR signal data to produce a complex map of k-space.

Sampling of the acquired signal in MRI takes place in the Fourier space. The k-space data is arranged with low frequencies signals are at the center of the acquired data and the high frequencies data are spaced around this center. The low frequencies signals contain information about contrast giving the greatest changes in grey level with highest amplitude value.

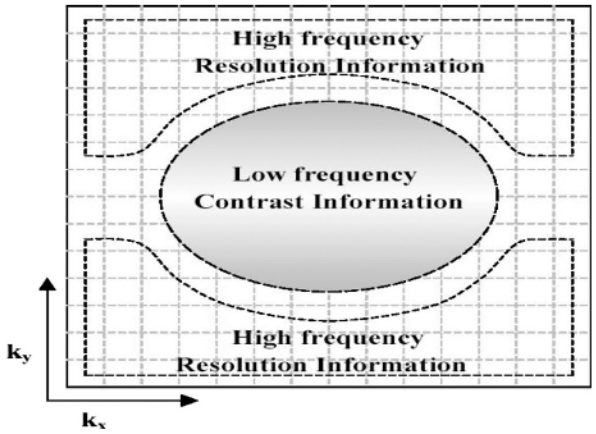


Figure 4-4: Location of Low-Frequency and High-Frequency in K-Space data

High frequencies signals contain information about the spatial resolution of the object or what is normally referred to as sharpness [**19**]. High frequency signals display rapid

changes of image signal as a function of position. Fig. 3 shows a typical uniformly sampled K-space data.

4.5 Image Reconstruction

There exist various methods of converting the acquired K-space data to real images in spatial domain. These include: the use of DFT, Radon Transform, Parametric technique, Artificial neural network based reconstruction technique and so on. DFT technique involves the application of Fourier series on the linearly or radially sampled k-space data. Radon transform involve the use of projection in obtaining the required images. Parametric approach (a non Fourier series) involves implicit or explicit data extrapolation to recover some of the unmeasured (presumably lost) high-spatial-frequency data. Some of the most widely used parametric technique includes, Autoregressive, Moving Average (MA), Autoregressive Moving average (ARMA) model **[21]**, **[22]**, **[23]**, **[24]**. This work deals with only DTF technique using FFT.

4.5.1 Application of Inverse FFT to K-Space Data

MRI 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: MRI_Reconstruction

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

Step 1: Read data header information: Load RAW data contain information about the MRI data file. It is a text file containing Information about Offset, DATA size, Kx Co-ordinate, Ky-Co-ordinate etc.

Step 2: Read in the K-space DATA.

Step 3: IFFT in K(x,y) Direction

Step 4: FFT shift

Step 5: Image Display

In practical MRI reconstruction, there are some other pre-processing activities that must be accomplished before the application of IFFT **[25]**, **[24]** **[26]**. The flow diagram for an MRI reconstruction is as shown in Figure 4-5 and the reconstruction algorithm is discussed herewith.

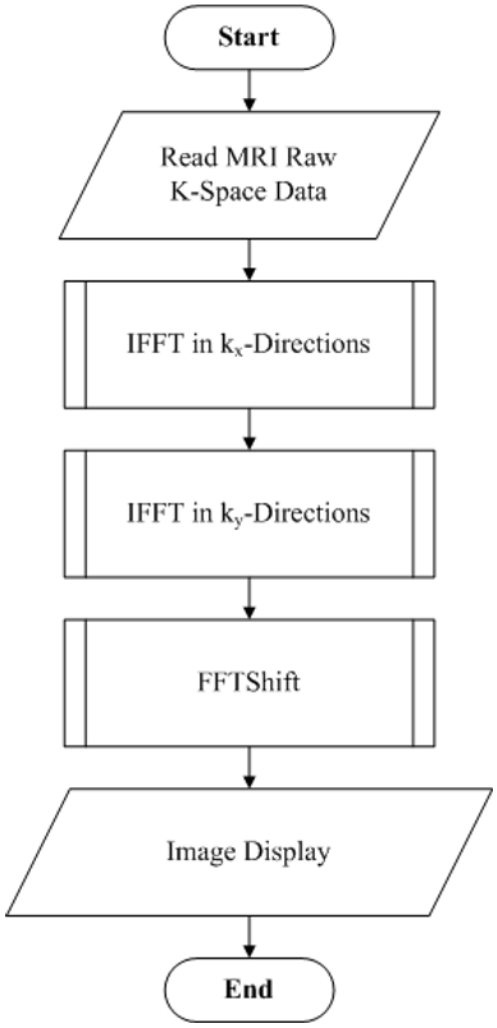


Figure 4-5: Flow diagram of MR Image Reconstruction

K-space is the rawest form of data obtained at MR imaging. An acquisition with a 256 x256 matrix contains 256 lines of data, and each of those lines contains 256 data points. The y-dimension in this 256 x256 array is called the phase encoding direction, and the x-dimension is called the frequency encoding direction. The distance between neighboring points in k-space determines the field of view of the object imaged, and the extent of k-space determines the resolution of the image **[27]**, **[28]**.

This work applies Fourier Transformation based reconstruction algorithm stated here for run under both CPU and GPU.

CHAPTER 5
GRAPHICS PROCESSING UNIT

5.1 Background

In recent years, much has been made of the computing industry's widespread shift to parallel computing. Nearly all consumer computers in the year 2010 will ship with multicore central processors. From the introduction of dual-core, low-end netbook machines to 8- and 16-core workstation computers, no longer will parallel computing be relegated to exotic supercomputers or mainframes. Moreover, electronic devices such as mobile phones and portable music players have begun to incorporate parallel computing capabilities in an effort to provide functionality well beyond those of their predecessors. More and more, software developers will need to cope with a variety of parallel computing platforms and technologies in order to provide novel and rich experiences for an increasingly sophisticated base of users. Command prompts are out; multithreaded graphical interfaces are in. Cellular phones that only make calls are out; phones that can simultaneously play music, browse the Web, and provide GPS services are in [**29**].

5.2 GPU Programming Premier

The release of GPUs that possessed programmable pipelines attracted many researchers to the possibility of using graphics hardware for more than simply OpenGL- or DirectX-based rendering. The general approach in the early days of GPU computing was extraordinarily convoluted. Because standard graphics APIs such as OpenGL and DirectX were still the only way to interact with a GPU, any attempt to perform arbitrary computations on a GPU would still be subject to the constraints of programming within a graphics API. Because of this, researchers explored general-purpose computation through graphics APIs by trying to make their problems appear to the GPU to be traditional rendering [**30**].

5.3 Parallel Computing Architectures and Models

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 [**31**].

5.3.1 NVidia CUDA

As the number of papers published shows, one of the most widely used parallel programming models is CUDA, developed by NVidia. 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 **[32]**, **[29]**, **[33]**.

From the hardware point of view, 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.

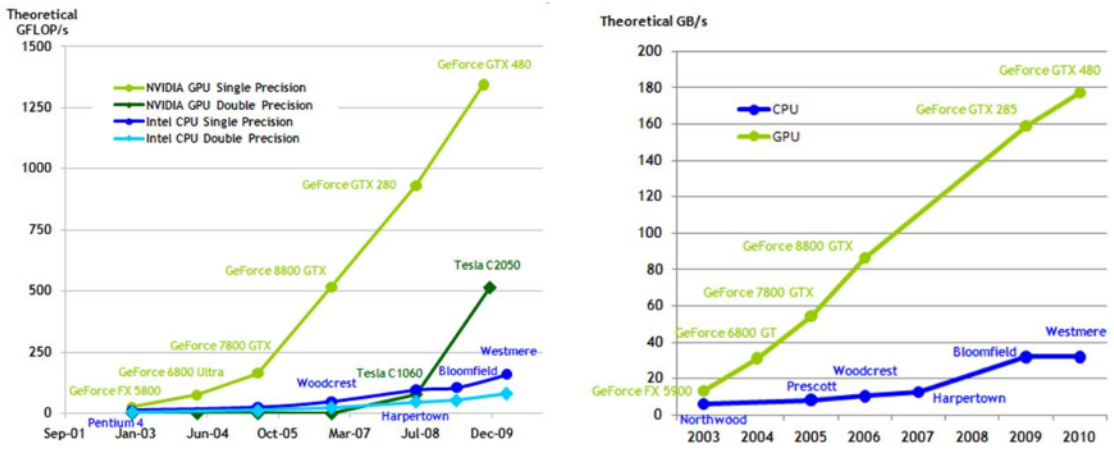


Figure 5-1: Floating-Point Operations per Second and Memory Bandwidth for the CPU and GPU

5.3.2 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 [**31**]. When a kernel function is invoked, the execution is moved to a device (GPU), 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 5-2 shows the execution of two Girds of threads [**34**].

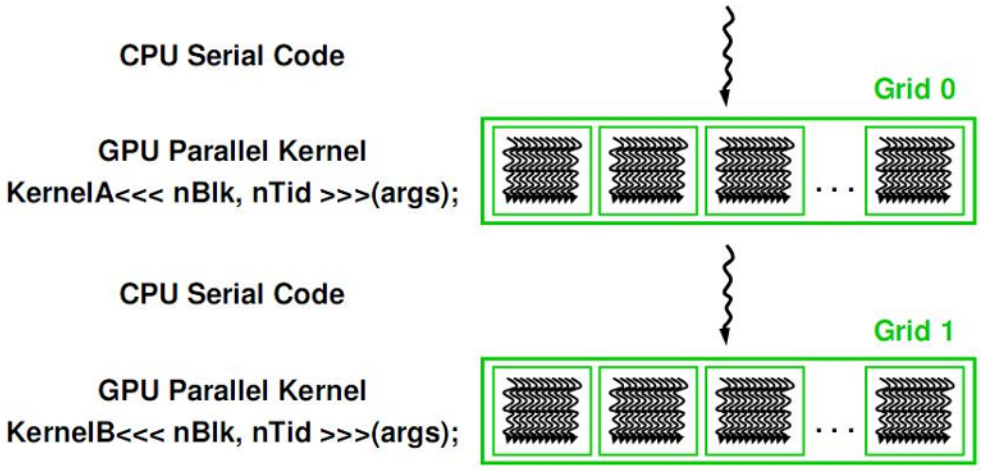


Figure 5-2: Execution of a CUDA program

5.3.3 Kernel Functions and Threading

From the programmer's point of view, every program consists of two parts, a host code ran on CPU (called a 'host'), and a kernel—a piece of code ran in parallel on GPU (called a 'device'). CUDA's extensions to the C programming language allow programmer to define how many threads performing a kernel will be executed on a device. The number of threads can greatly (up to 10,000x) exceeds the number of processing units which execute the program. This paradigm allows programmers to consider GPU as an abstract device providing a huge number of virtual resources is depict in Figure 5-3. When all threads of a kernel complete their execution, the corresponding grid terminates, the execution continues on the host until another kernel is invoked.

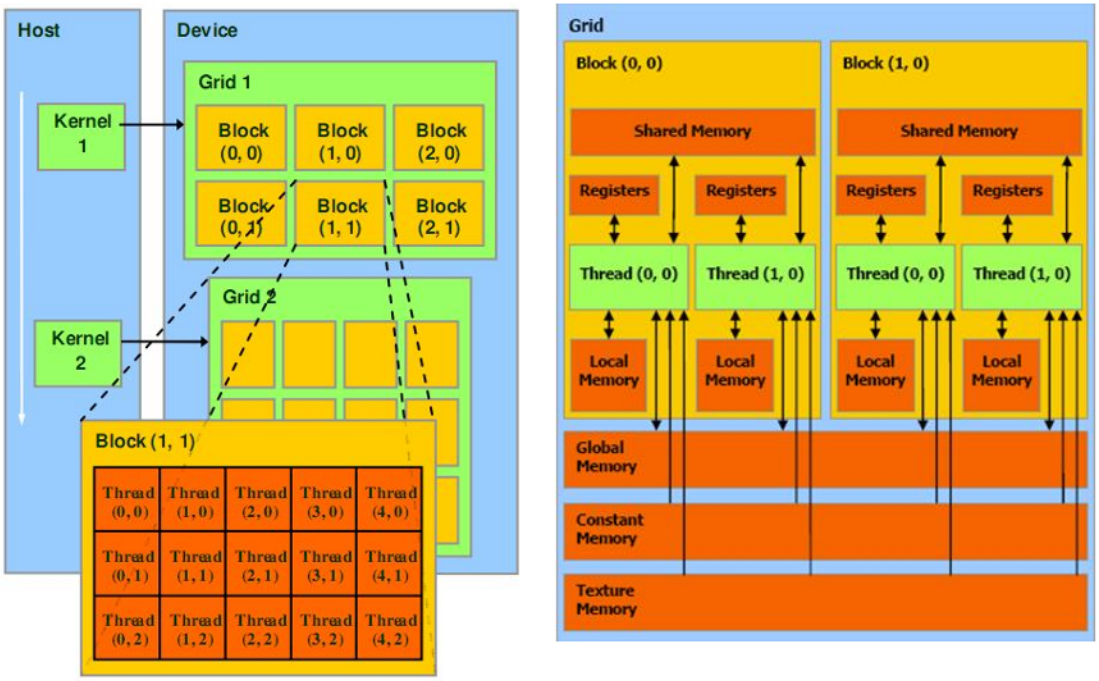


Figure 5-3: CUDA Thread Organization

Therefore, not only it provides a good scalability for future hardware, it also allows hiding global memory latency by very fast thread switching [CUD10c]. The threads executed on GPU form a 1D, 2-D, or 3-D structure, called a block (Figure **5-3**). The blocks are arranged in 1D or 2-D layout, called a grid. Thus, each thread can be exclusively defined by its coordinates within a block, and by coordinates of its block within a grid. Each block of threads is executed on a sigle SM; therefore, all threads of a single block can share data in a shared memory. Block's threads are executed in warps of 32 threads each **[35]**.

5.3.4 CUDA Memory model

Each CUDA device has several memories that can be used by programmers to achieve high Computation to Global Memory Access (CGMA) ratio and thus high execution speed in their kernels. Variables that reside in registers and shared memories can be accessed at very high speed in a highly parallel manner. Registers are allocated to individual threads; each thread can only access its own registers. A kernel function typically uses registers to hold frequently accessed variables that are private to each thread. Shared memories are allocated to thread blocks; all threads in a block can access variables in the shared memory locations allocated to the block. Shared memories are efficient means for threads to cooperate by sharing the results of

their work. At the middle of the table, we see global memory constant memory. These are the memories that the host code can write (W) and read (R) by calling API functions. The global memory can be accessed by all the threads at anytime of program execuction. The constant memory allows read-only access by the device and provides faster and more parallel data access paths for CUDA kernel execution than the global memory. Below is the table of types of CUDA memory:

Table 5-1: Different Types of CUDA Memory

Memory	Location	Cached	Access	Who
Local	Off-chip	No	Read/Write	One Tthread
Shared	On-chip	N/A	Read/write	All threads in a block
Global	Off-chip	No	Read/write	All threads + CPU
Constant	Off-chip	Yes	Read	All threads + CPU
Texture	Off-chip	Yes	Read	All threads + CPU

CUDA defines registers, shared memory, and constant memory that can be accessed at higher speed and in a more parallel manner than the global memory. Using these memories effectively will likely require re-design of the algorithm. It is important for CUDA programmers to be aware of the limited sizes of these special memories. Their capacities are implementation dependent. Once their capacities are exceeded, they become limiting factors for the number of threads that can be assigned to each SM [**31**], [**36**].

5.4 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 [**32**]:

- 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

5.4.1 Sample Code for 2D Complex-to-Real FFT Using CUFFT

```
#define NX 256
#define NY 128

cufftHandle plan;
cufftComplex *idata;
cufftReal *odata;

cudaMalloc((void**)&idata, sizeof(cufftComplex)*NX*NY);
cudaMalloc((void**)&odata, sizeof(cufftReal)*NX*NY);

/* Create a 2D FFT plan. */
cufftPlan2d(&plan, NX, NY, CUFFT C2R);

/* Use the CUFFT plan to transform the signal out of place. */
cufftExecC2R(plan, idata, odata);

/* Destroy the CUFFT plan. */
cufftDestroy(plan);

cudaFree(idata); cudaFree(odata);
```

5.5 FFTW – FFT for CPU

FFTW, the Fastest Fourier Transform in the West. FFTW is a comprehensive collection of fast C routines for computing the discrete Fourier transform (DFT) and various special cases thereof.

- FFTW computes the DFT of complex data, real data, even- or odd-symmetric real data (these symmetric transforms are usually known as the discrete cosine

or sine transform, respectively), and the discrete Hartley transform (DHT) of real data.

- The input data can have arbitrary length. FFTW employs O(*n log n*) algorithms for all lengths, including prime numbers.
- FFTW supports arbitrary multi-dimensional data.
- FFTW supports the SSE, SSE2, Altivec, and MIPS PS instruction sets.
- FFTW 3.2.2 includes parallel (multi-threaded) transforms for shared-memory systems.

5.6 CUFFT Performance vs. FFTW

Group at University of Waterloo did some benchmarks to compare CUFFT to FFTW. They found that, in general:

- CUFFT is good for larger, power-of-two sized FFT's
- CUFFT is not good for small sized FFT's
    - CPUs can fit all the data in their cache
    - GPUs data transfer from global memory takes

CUFFT starts to perform better than FFTW around data sizes of 8192 elements. nflops for CUFFT do decrease for non-power-of-two sized FFT's, but it still beats FFTW form most large size(>~10,000 elements) [**37**].

5.7 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 **[38]**.

5.7.1 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. The execution is transparent to the user.
- 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. The rapid code prototyping capability of the scripting language is combined with the fast code execution on the GPU.
- The most important MATLAB functions are currently implemented. GPUmat can be used as a Source development Kit to create missing functions and to extend the library functionality.
- Supports real/complex, single/double precision data types.

5.7.2 Perfomance

This Figure 5-4 shows the performance results of the element-by-element multiplication (A.*B) in MATLAB as a function of the number of elements. The hardware used for the tests is a PC Dual Core Intel 6600 at 2.4GHZ with a GPU NVIDIA 8800GTX (128 stream processors).

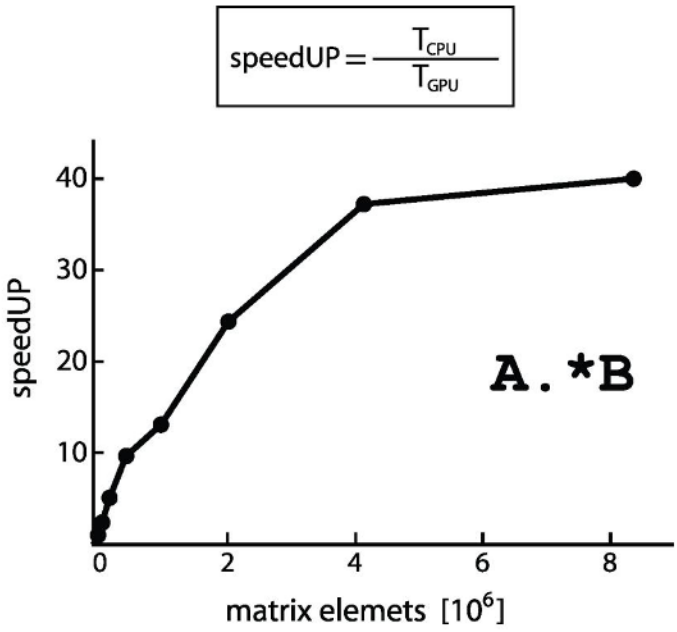


Figure 5-4: performance results of the (A.*B) using GPUmat

GPUmat uses a technology developed by NVIDIA called CUDA SDK which allows programming the GPU for general purpose applications. The GPUmat core is based on CUDA libraries, such as CUFFT and CUBLAS, and many other functions developed and optimized by the GP-you Group for the GPU architecture **[39], [4]**.

CHAPTER 6
RESULTS AND DISCUSSIONS

6.1 Experimental Setup

The experiment is divided into two sections. First is simulation of performance for FFT-IFT on different sized images. Then original MR Image reconstruction from raw spectral data is done. Required hardware, software and data are discussed in this section.

6.1.1 Hardware Requirements

All experiments are done using both CPU and GPU. The configurations for them are listed in table below:

Table 6-1: CUDA Device Configuration at Experimental Setup

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

Table 6-2: Host Machine Configuration at Experimental Setup

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

6.1.2 *Required Software and Tools*

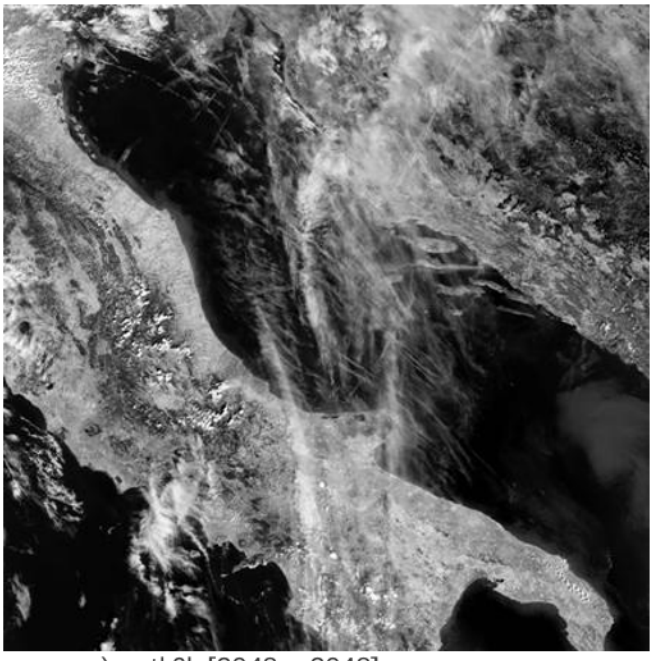
The experiment requires some software and tools for programming and documenting purpose. Following table lists up all used software and tools:

Table 6-3: 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	7.10.0.499	Simulation and Programming
CUFFT	2.3	CUDA capable FFT library
GPUmat	0.27	Wrapper for MATLAB to run CUDA Program

6.1.3 *Experimental Images and Data*

For smooth running the process of MRI Reconstruction using GPU we have used three images and MRI raw data. Images are chosen different resolution for figure out the performance of FFT and IFFT on CPU and GPU based implementation. MRI raw data has been collected from Duke Virtual Imaging Laboratory.



c) earth2k [2048 × 2048]

a) lena [512 × 512]

b) airplane [1024 × 1024]

Figure 6-1: Images used for Experiments

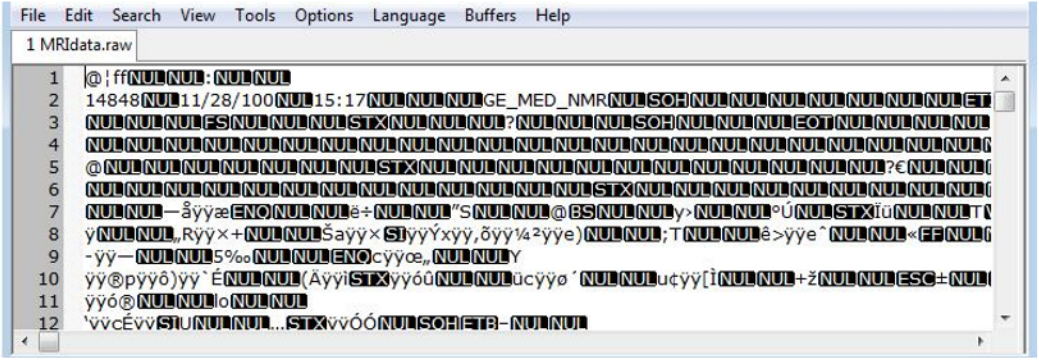


Figure 6-2: MRI Raw data text file

6.2 Simulation Result of DFT Based Reconstruction

Discrete Fourier Transformation (DFT) based reconstruction technique has been described in paper **[20]**. To visualize the performance of DFT based image reconstruction we have first implement it to run under CPU. The CPU based DFT implementation requires 4.7sec to reconstruct lena image 100 times which has 512x512 spatial resolution. The same image requires 3.48sec to reconstruct 100 times using FFT based algorithm runs under CPU. The performance variation illustrates in Figure 6-3 leads towards the FFT based reconstruction technique.

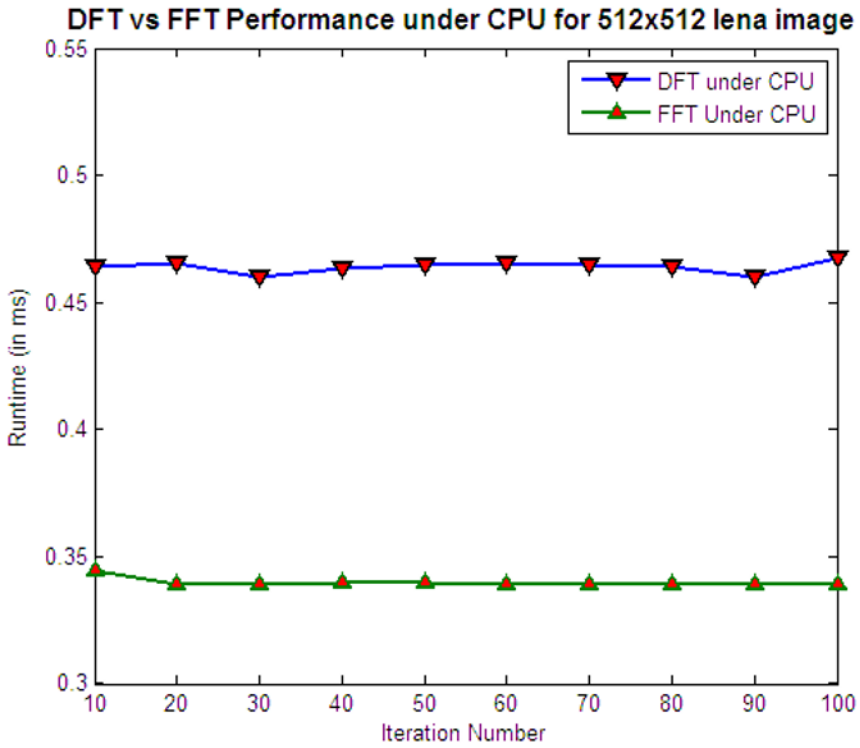


Figure 6-3: DFT vs FFT Performance under CPU for 512x512 lena image

6.3 Simulation Results of FFT Based Reconstruction

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.

6.3.1 Lena image Reconstruction

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

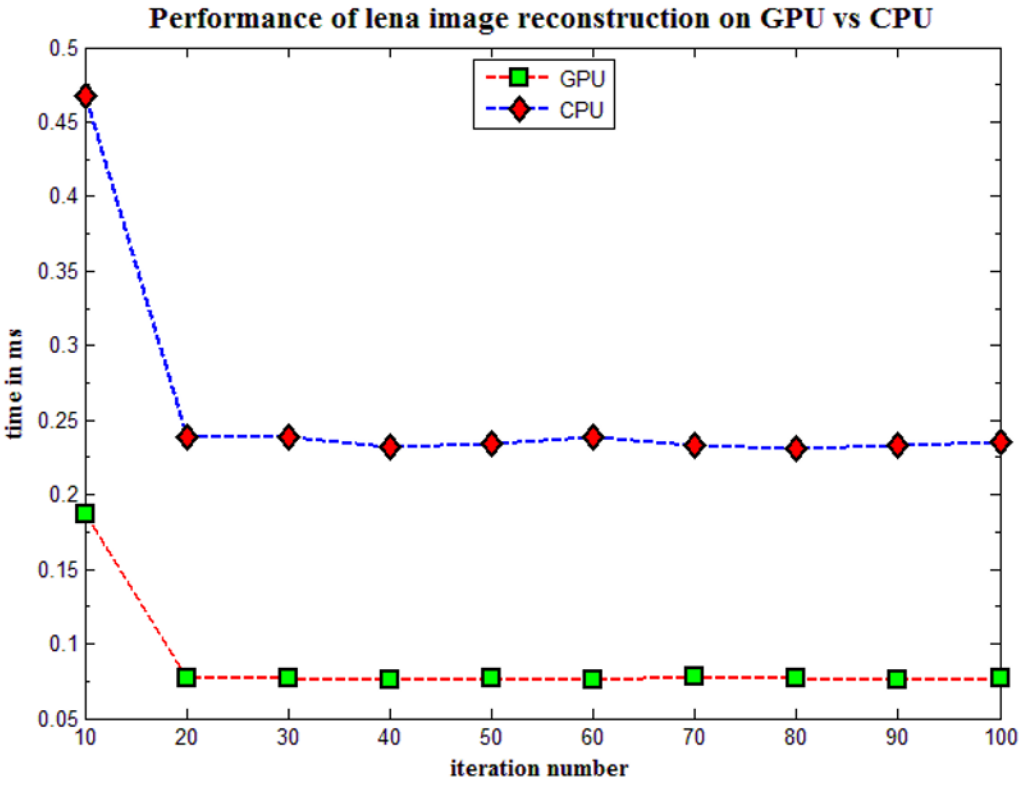


Figure 6-4: GPU vs. CPU Performance of Reconstruction for lena image

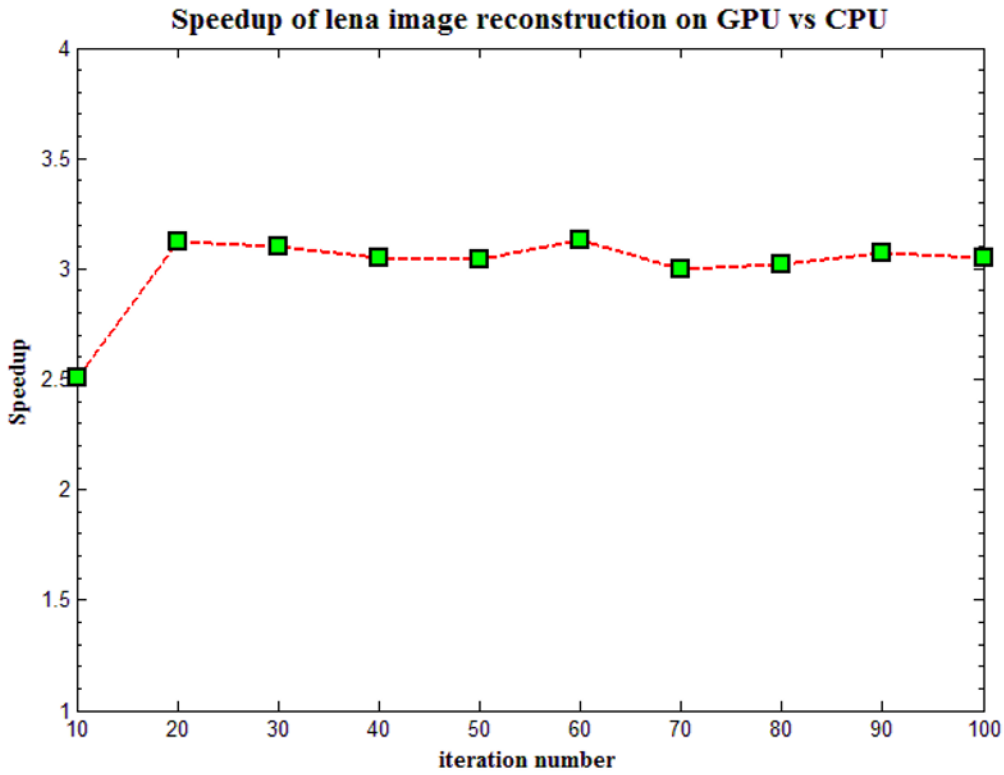


Figure 6-5 : Image Reconstruction Speedup by GPU vs CPU for lena image

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

6.3.2 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.

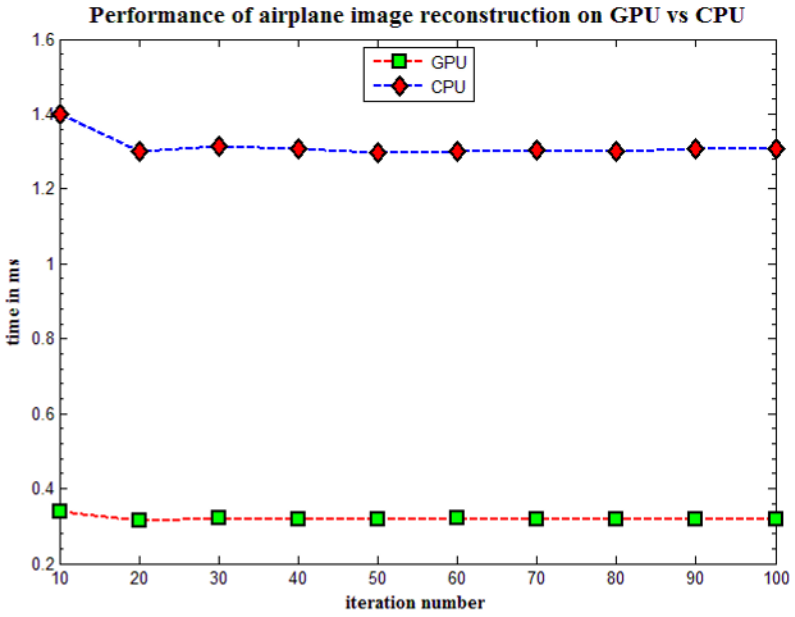


Figure 6-6: GPU vs. CPU Performance of Reconstruction for airplane image

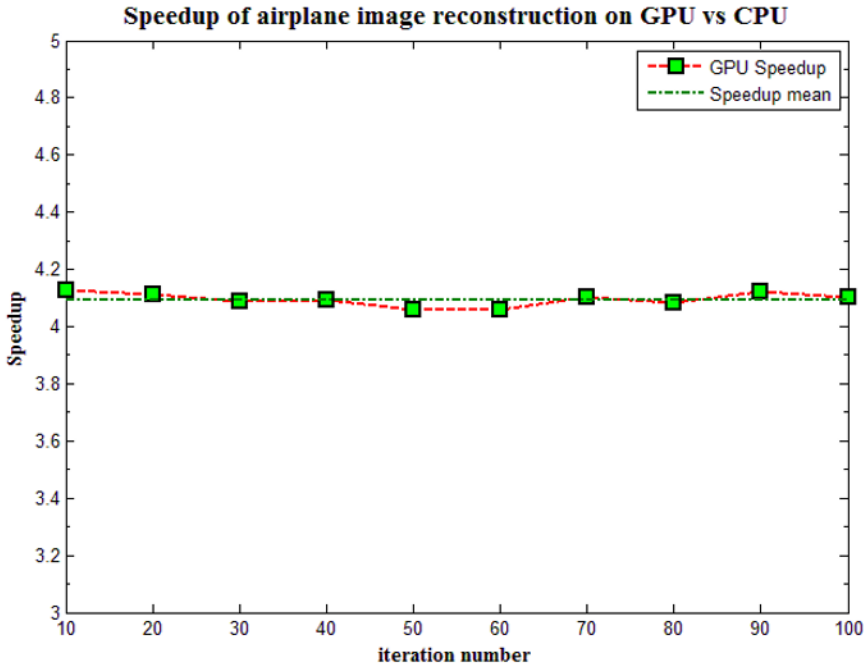


Figure 6-7 : Image Reconstruction Speedup by GPU vs CPU for airplane image

6.3.3 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.

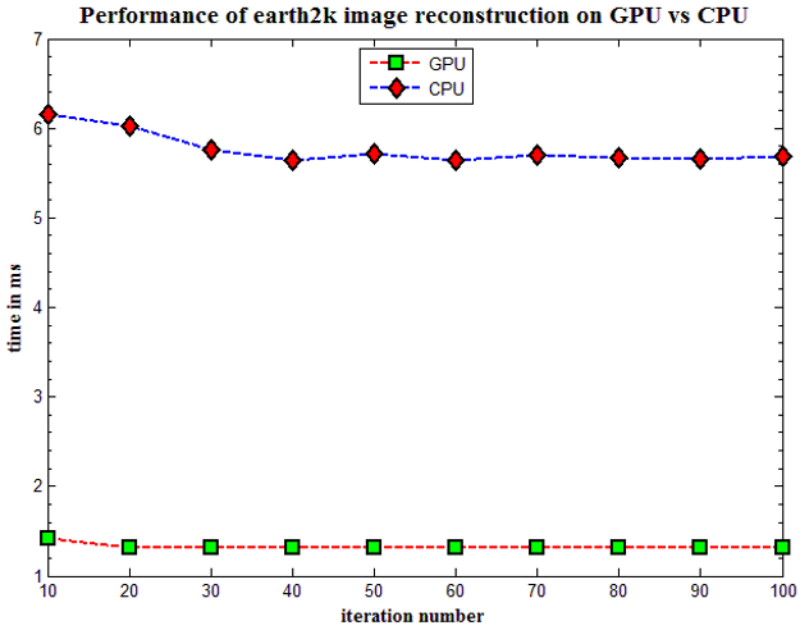


Figure 6-8: GPU vs. CPU Performance of Reconstruction for earth2k image

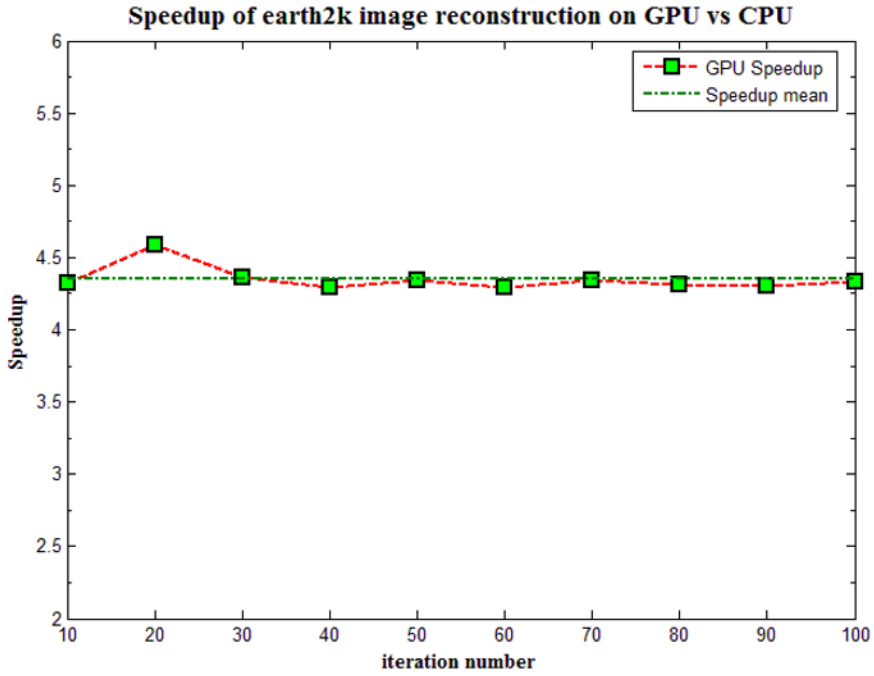


Figure 6-9 : Image Reconstruction Speedup by GPU vs CPU for airplane image

6.3.4 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 for all test images.

Table 6-4: Reconstruction Summary for 100 iterations

Image Name	Spatial Resolution of Image	FFT Time *(ms)*		IFT Time *(ms)*		Reconstruction 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

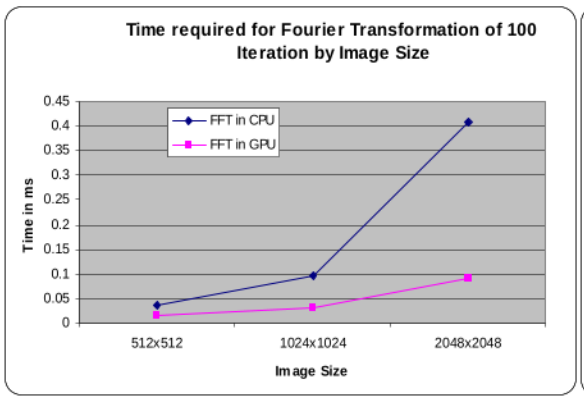


Figure 6-11: Fourier Transformation Performance by Image Size

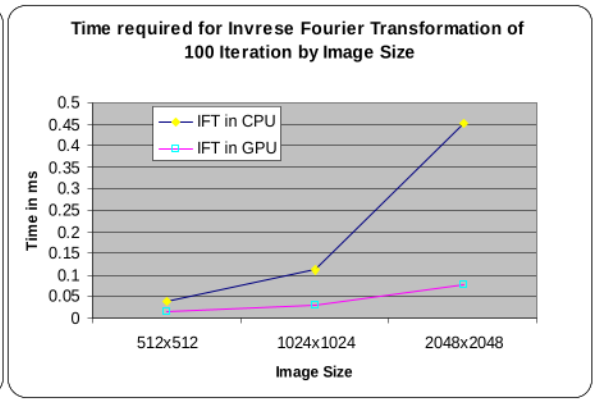


Figure 6-10: Inverse Fourier Transformation Performance by Image Size

From the Figure 6-11 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 6-10.

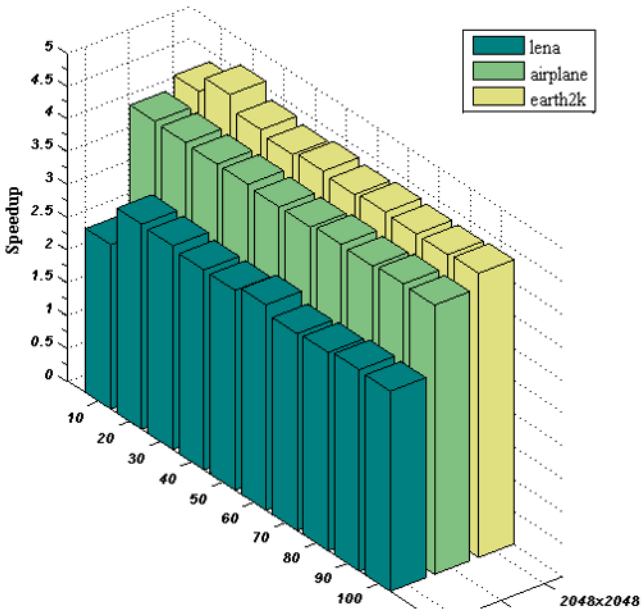
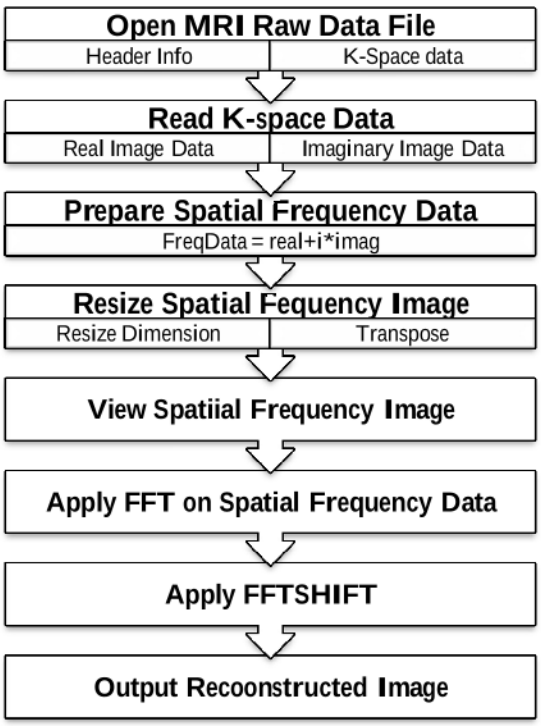


Figure 6-12 : GPU Speedup of Image Reconstruction by Image Size

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

6.4 MR-Image Reconstruction

MR-Image reconstruction steps are depicted in below:



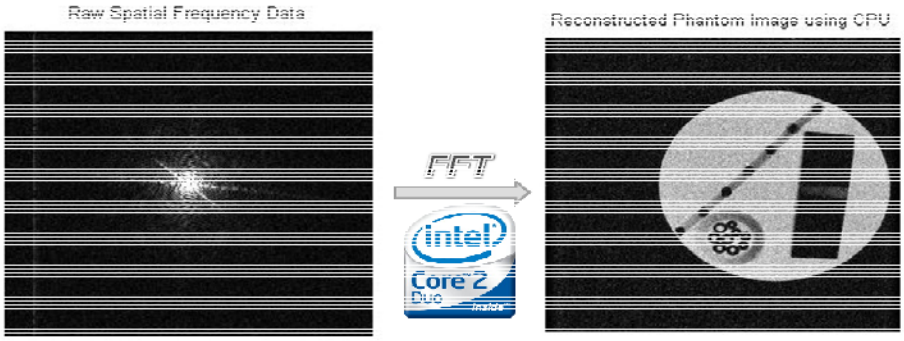


Figure 6-13: MRI Reconstruction in CPU

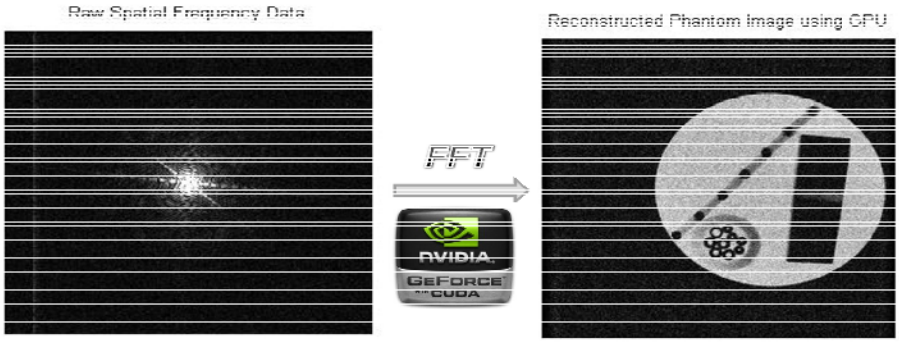


Figure 6-14: MRI Reconstruction in GPU

6.4.1 MRI Reconstruction Speedup by GPU than CPU

GPU based MR image reconstruction algorithm is implemented along with CPU based implementation using MATLAB.

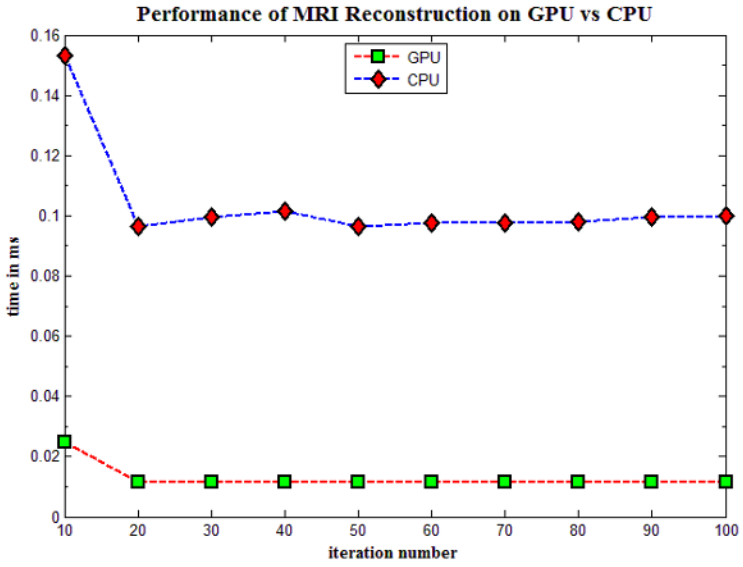


Figure 6-15: MRI Reconstruction Performance by GPPU vs CPU

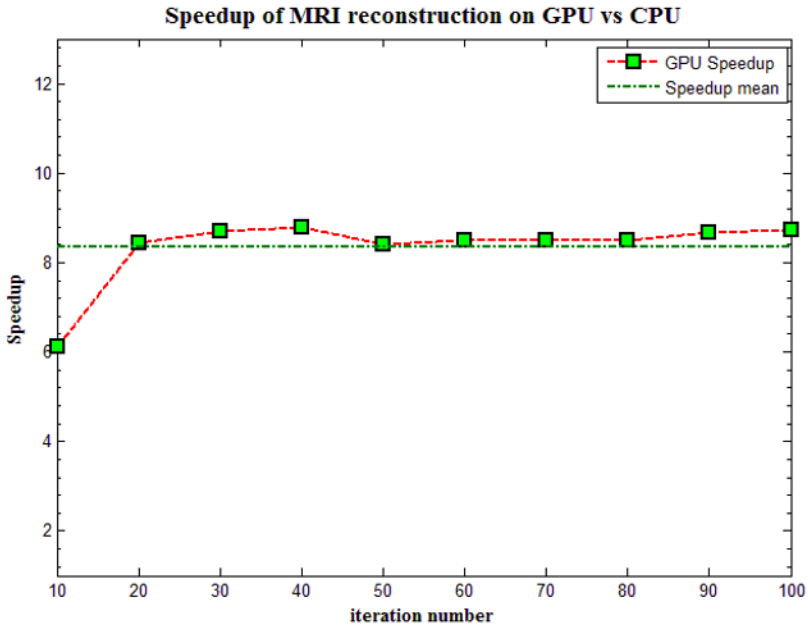


Figure 6-16: Speedup of MRI Reconstruction by GPU vs CPU

GPU Based FFT improves the performance of the reconstruction algorithm by a factor of 8.2. The resulting image quality is virtually the same for both CPU-based and GPU-based implementations. In future, the real-time GPU-based MR image reconstruction will be possible.

CHAPTER 7
CONCLUSION

7.1 Summary

The GPU programming was something new for us. We had to learn by ourselves how to get start with the CUDA programming and also how to make an adapted program for the GPU programming. We had to understand the procedure of MRI reconstruction.

The image reconstruction algorithm, as we have seen, is made up of the FFT and IFT. We implemented this algorithm using MATLAB. With the CUFFT library, the parallel algorithm runs on GPU.

Nvidia's CUDA coupled with the NVIDIA GeForce G 103M produced impressive speed increases on the FFT-IFFT algorithm used in MRI reconstruction than that of CPU based implementation. The performance speedup is not only impressive but also explores the possibility of using consumer general purpose Graphics Card instead of Application Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) based costly system for compute intensive image processing system.

7.2 Limitations and Future Work

The Performance improvement of the work is impressive. However, the main purpose of this thesis was not to reconstruct MRI, but rather 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.

The amount of medical data collected through Magnetic Resonance Imaging, Computerized tomography, ultrasound scanners and computer-aided high resolution microscopes increases day by day. The recent advances in computer technology

have enabled analysis and visualization of such data on relatively cheaper computers. Since most diagnosis systems are able to output 3D data, 3D rendering methods are indispensable in order take full advantage of such data.

It can also possible to do other processing, such as de-noising and de-blurring on MRI raw data before output the reconstructed image. The same process can be applied to reconstruction of 3D holographic image. 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.

7.3 Conclusion

Besides the performance advantage of using a GPU over a CPU for MR-image reconstruction, there are other advantages as well. In MRI 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 reconstruction, 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 medical applications.

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] R. Gonzalez, C. Woods, and S. L. Eddins, *Digital Image Processing Using MATLAB*. Prentice Hall, 2009.

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

[12] B. MA and S. RC, *MRI basic principles and applications*. Hoboken, New Jersey: Wiley, 2003.

[13] Z. P. Liang and P. C. Lauterbur, "Principles of Magnetic Resonance Imaging, A signal processing perspective," *IEEE Press*, 2000.

[14] E. M. Haacle and Z. P. Liang, "Challenges of Imaging Stucture and Function with MRI," *IEEE Transactions on Medicine and Biology*, vol. 19, pp. 55-62, 2000.

[15] D. G. Nishimura, "Principles of Magnetic Resonance Imaging," Apr. 1996.

[16] MRI Basics: MRI Basics. [Online]. http://www.cis.rit.edu/htbooks/mri/inside.htm

[17] E. M. Haacle and Z. P. Liang, "Challenges of Imaging Stucture and Function with MRI," in *IEEE Transactions on Medicine and Biology*, vol. 19, 2000, pp. 55-62.

[18] I.S.Uzun and A. A. A.Bouridane, "FPGA Implementations of Fast Fourier Transforms for Real-Time Signal and Image Processing," in *In Proceedings. 2003 IEEE International Conference on Field-Programmable Technology (FPT)* , 2003.

[19] J. Zhuo and R. P. Gullapalli, "MR Artifacts, Safety, and Quality," *Physics Tutorial for Residents*, pp. 275-279, 2006.

[20] A. M. Aibinu, M. J. E. Salami, A. A. Shafie, and A. R. Najeeb, "MRI Reconstruction Using Discrete Fourier Transform: A tutorial," *World Academy of Science, Engineering and Technology*, 2008.

[21] S. DK, M. CA, O. MA, Y. EN, and Price, "Recent advances in image reconstruction, coil sensitivity calibration, and coil array design for SMASH and generalized parallel MRI," *MAGMA*, vol. 13, pp. 158-163, 2002.

[22] M. P., "Multi-planar image formation using NMR spin echoes. ," *J Phys* , vol. 10, pp. 55-58, 1977.

[23] E. Carmi, S. L. T, N. Alona, A. Fiat, and D. F. T, "Resolution enhancement in MRI," *Magnetic Resonance Imaging*, p. 133–154, 2006.

[24] 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.

[25] 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.

[26] 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.

[27] S. S. Stone, et al., "Accelerating Advanced MRI Reconstructions on GPUs," in *Proceedings of the 5th International Conference on Computing Frontiers*, 2008.

[28] D. Kleut, M. Jovanović, and P. D. B. Reljin, "3D Visualisation of MRI images using MATLAB," *JOURNAL OF AUTOMATIC CONTROL*, vol. 16, no. 1-3, 2006.

[29] "NVIDIA® CUDA™ Architecture Introduction & Overview," Technical Whitepaper, 2009.

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

[31] CUAD Tutorial: The Golden Energy Computing Organization. [Online].

http://geco.mines.edu/tesla/cuda_tutorial_mio/index.html

[32] (2010) CUDA(TM) CUFFT Library 3.1. [Online]. http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/CUFFT_Library_3.1.pdf

[33] 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

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

[35] J. Sanders and E. Kandrot, *CUDA by Example : An IntroductIon to General Purpose GPU Programming*. Boston, MA 02116 , USA: Pearson Education, Inc, 2010.

[36] S. A. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformatics*, 2008.

[37] M. Frigo and S. G. Johnson. FFTW 3.2.2. [Online]. http://www.fftw.org/fftw3.pdf

[38] (2010) AccelerEyes - MATLAB GPU Computing. . [Online]. http://www.accelereyes.com

[39] (2010, Dec.) GPUmat: GPU Power in MATLAB. [Online]. http://www.gp-you.org/index.php?option=com_content&view=article&id=46&Itemid=54

INDEX