MODELING OF SPLIT STEP PARABOLIC WAVE EQUATION USING THE GRAPHICS PROCESSING UNIT

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

SELİN SEKMEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

SEPTEMBER 2014

Approval of the thesis:

MODELING OF SPLIT STEP PARABOLIC WAVE EQUATION USING THE GRAPHICS PROCESSING UNIT

submitted by **SELİN SEKMEN** in partial fulfillment of the requirements for the degree of **Master of Science in Computer Engineering Department, Middle East Technical University** by,

Prof. Dr. Canan Özgen _____
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. Adnan Yazıcı _____
Head of Department, **Computer Engineering**

Assoc. Prof. Dr. Tolga Can _____
Supervisor, **Computer Engineering Department, METU**

Dr. Cevat Şener _____
Co-supervisor, **Computer Engineering Dept., METU**

Examining Committee Members:

Prof. Dr. Göktürk Üçoluk
Computer Engineering Department, METU _____

Assoc. Prof. Dr. Tolga Can
Computer Engineering Department, METU _____

Dr. Cevat Şener
Computer Engineering Department, METU _____

Dr. Onur Tolga Şehitoğlu
Computer Engineering Department, METU _____

Dr. Halit Ergezer
Team Lead, MİKES Inc. _____

Date: _____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: SELİN SEKMEN

Signature :

ABSTRACT

MODELING OF SPLIT STEP PARABOLIC WAVE EQUATION USING THE GRAPHICS PROCESSING UNIT

Sekmen, Selin
M.S., Department of Computer Engineering
Supervisor : Assoc. Prof. Dr. Tolga Can
Co-Supervisor : Dr. Cevat Şener

September 2014, 77 pages

Electromagnetic (EM) wave propagation is an important phenomenon in modeling radar or communication systems. To develop these kinds of reliable systems, the behavior of EM waves and cases that affect the propagation must be handled correctly. EM waves are strongly affected by atmospheric conditions such as temperature, pressure, and humidity, and by global circulation patterns. When they propagate in troposphere, their energy decreases due to change in medium. Other than natural effects, buildings, non-flat terrains also disturb wave propagation, since these kinds of terrain structures cause EM waves to reflect, refract or diffract while they propagate in their normal paths.

The main issue in electromagnetic wave propagation is to compute the propagation factor and path loss. Split Step Parabolic Equation (SSPE) is a commonly used parabolic equation that efficiently models the electromagnetic wave propagation in troposphere. It is a one way forward propagation approach, which models forward waves, and neglects backward ones. It is a highly accurate and a reliable method; however, algorithms that implement this model are computationally intensive.

GPUs are developed for visual graphical purposes, however general purpose use of GPUs become popular in the last decades. As methods for electromagnetic modeling are computationally intensive, GPGPUs (General Purpose Graphics Processing Unit)

take the attention of people who are interested in electromagnetic wave simulations, as their highly parallel architectures can offer better performance.

This thesis focuses on implementation of the Split Step Parabolic Wave Equation on the GPU architecture in 2D and 3D environments. We study implementation and performance analysis of SSPE on three different graphic cards.

Keywords: Electromagnetic wave propagation in troposphere, parabolic equation, Split Step Parabolic Equation (SSPE), High Performance Computing, Graphics Processing Unit (GPU), General Purpose Graphics Processing Unit (GPGPU), CUDA

ÖZ

GRAFİK İŞLEMCİ ÜNİTESİ İLE ADIM ADIM PARABOLİK DALGA DENKLEMİNİN PARALELLEŞTİRİLMESİ

Sekmen, Selin
Yüksek Lisans, Bilgisayar Mühendisliği Bölümü
Tez Yöneticisi : Doç. Dr. Tolga Can
Ortak Tez Yöneticisi : Dr. Cevat Şener

Eylül 2014 , 77 sayfa

Elektromanyetik dalga, kaynağından çıktıktan sonra alıcıya ulaşana kadar çeşitli çevresel etkilere maruz kalmaktadır. Bu etkiler insan kaynaklı ya da doğal etkenler olabilir. Elektromanyetik dalganın kaynağı ile alıcı arasında herhangi bir engel olmasa dahi, dalga atmosferik ortamdan etkilenir. Troposferde gerçekleşen olaylar, sıcaklık, yoğunluk ve basınç değişimleri elektromanyetik dalganın yayılımını etkileyen atmosferik faktörlerdir. Dalganın gücü atmosferde yol katettikçe azalmaktadır. Doğal etkenlerin yanı sıra çevremizde bulunan binalar, engebeler, dağlar, vb. leri de elektromanyetik dalganın yayılımını etkileyen diğer etmenlerdir. Çünkü bu engeller dalganın yayılımı sırasında kırılmasına, yansımasına ya da saçılmasına sebep olur. Dalganın yayılımı sırasında yansıma, kırılma ve saçılma sonucu oluşan dalgalar da yayılımı etkiler. Elektromanyetik dalganın modellenmesi tüm bu etkiler göz önünde tutularak gerçekleştirilmelidir. Modelleme yapılırken asıl amaç alıcıda toplanacak ortalama sinyal gücünü ve belli bir alandaki sinyal gücündeki varyasyonu hesaplamaktır.

Elektromanyetik dalganın yayılımını gerçeğe en yakın şekilde modelleyebilmek için çeşitli yöntemler geliştirilmiştir. Adım Adım Parabolik Dalga Denklemi bu amaçla geliştirilen tek yönlü yayılımı modelleyen bir metottur. Fakat bu yöntem oldukça yoğun hesaplama gerektiren bir algoritmaya sahiptir. Modern Grafik İşlemci Üniteleri (GPU), bilgisayar grafiklerini işleme ve göstermekte son derece verimlidirler ve bunun yanı sıra son 10 yılda mimarilerini genel amaçlı hesaplama yapabilecek şekilde

geliştirmişlerdir. Böylece yüksek paralel yapıları kompleks algoritmalar için oldukça etkili çözümler sunmaktadır.

Bu tezde, elektromanyetik dalga yayılımını modellemek için Adım Adım Parabolik Dalga Denklemi kullanılmıştır, bu yöntem GPU mimarisine uygun olarak, iki boyutlu ve üç boyutlu ortamları modelleyebilecek şekilde geliştirilmiş ve 3 farklı grafik kartı üzerinde performans testleri gerçekleştirilmiştir.

Anahtar Kelimeler: Electromanyetik dalga yayılımı, parabolik denklemler, Adım Adım Parabolik Dalga Denklemi, Grafik İşlemci Unitesi (GPU), Grafik İşlemcisinde Genel Amaçlı Hesaplama (GPGPU), CUDA

To my parents Cemil SEKMEN & Sevilay SEKMEN, and my two sweet sisters Selda SEKMEN & Esin SEKMEN

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Associate Professor Tolga Can and co-supervisor Dr. Cevat Şener for the continuous support of my MS study and research, for their motivation, guidance and helpful suggestions. Their guidance helped me in all the time of research and writing of this thesis.

Besides my advisor and co-supervisor, I would like to thank the rest of my thesis committee: Prof. Dr. Göktürk Üçoluk, Dr. Onur Tolga Şehitoğlu, and Dr. Halit Ergezer.

This work is also supported by SANTEZ Program from Ministry of Science, Industry and Technology for MS by 2 years (2012 - 2014). I would like to thank our SANTEZ team members Osman Günay, and Alaettin Zubaroğlu for their valuable contributions and supports to this thesis.

TABLE OF CONTENTS

ABSTRACT v

ÖZ vii

ACKNOWLEDGMENTS x

TABLE OF CONTENTS xi

LIST OF TABLES xv

LIST OF FIGURES xvi

LIST OF ABBREVIATIONS xviii

CHAPTERS

1 INTRODUCTION 1

1.1 Problem Definition and Motivation 1

1.2 Related Works 3

1.3 Thesis Contribution 4

1.4 Thesis Outline 4

2 BACKGROUND 7

2.1 Radar 7

2.2 Electromagnetic (EM) Waves 8

2.2.1 Reflection 8
2.2.2 Refraction 9
2.2.3 Diffraction 10
3 SPLIT-STEP PARABOLIC EQUATION (SSPE) 11
3.1 SSPE and Theoretical Definition 11
3.2 Fast Fourier Transform (FFT) 14
4 GRAPHICS PROCESSING UNIT (GPU) 17
4.1 CPU vs GPU 17
4.2 The GPU Architecture 19
4.2.1 Compute Unified Device Architecture (CUDA) . . 21
4.2.2 GPU Memory Types 24
4.3 Used Graphic Cards 25
4.3.1 Quadro 6000 25
4.3.2 Tesla C2075 25
4.3.3 Tesla K20 26
5 SSPE ON GPU 29
5.1 SSPE Model 30
5.1.1 Treatment of Irregular Terrain : Staircase Approach 31
5.1.2 Map Architecture 31
5.2 SSPE Work-flow 32
5.3 SSPE Algorithm 34

5.3.1 Refractivity in SSPE 34
5.3.2 Initial Field Calculation 36
5.3.3 A Single Field Calculation 37
5.4 Implementation 38
5.4.1 Implementation Environment 38
5.4.2 A Single Field Implementation in CPUsspe 39
5.4.3 A Single Field Implementation in 2D GPUsspe . . 39
5.4.4 A Single Field Calculation in 3D GPUsspe 43
5.5 Optimization in GPU 46
6 RESULTS 47
6.1 Accuracy of SSPE 47
6.1.1 Accuracy of SSPE with Flat Terrain 48
6.1.1.1 Dataset for Accuracy Measurements in Flat Terrain 48
6.1.1.2 Accuracy Test in CPUsspe 48
6.1.1.3 Accuracy Test in GPUsspe 50
6.1.1.3.1 Quadro 6000 51
6.1.1.3.2 Tesla C2075 53
6.1.1.3.3 Tesla K20 54
6.1.1.4 Horizontal and Vertical Polarization Comparison in SSPE 54
6.1.2 Accuracy of SSPE with Non-Flat Terrain 58

6.2 Performance of SSPE 61
6.2.1 Test Cases 61
6.2.1.1 Test 1 64
6.2.1.2 Test 2 64
6.2.1.3 Test 3 64
6.2.1.4 Test 4 64
6.2.1.5 Test 5 65
6.2.1.6 Test 6 65
6.2.1.7 Test 7 65
6.2.1.8 Test 8 65
6.2.2 Performance Results in 2D 66
6.2.3 Performance Results in 3D 70
7 CONCLUSION 73
7.1 Conclusion 73
7.2 Future Work 74
REFERENCES 75

LIST OF TABLES

TABLES

Table 6.1 CPUsspe vs GPUsspe Fixed Range in 2D 67

Table 6.2 CPUsspe vs GPUsspe Fixed FFT Size in 2D 68

Table 6.3 CPUsspe vs GPUsspe Test 7 and 8 in 2D 69

Table 6.4 GPUsspe (Tesla K20) Performance Results in 3D 71

LIST OF FIGURES

FIGURES

Figure 2.1 Reflection 9

Figure 2.2 Refraction 9

Figure 2.3 Diffraction 10

Figure 4.1 CPU vs GPU 18

Figure 4.2 Fermi Architecture 20

Figure 4.3 Host Device Execution 21

Figure 4.4 GPU Acceleration Works 22

Figure 4.5 Dynamic Parallelism 27

Figure 5.1 Split-Step Parabolic Equation Process 30

Figure 5.2 Staircase Approach for Modeling Non-flat Terrain 31

Figure 5.3 Split-Step Parabolic Equation Work Flow 35

Figure 6.1 PETOOL vs CPUsspe Propagation Comparison on Double Precision 49

Figure 6.2 PETOOL vs CPUsspe Propagation Comparison on Single Precision 49

Figure 6.3 PETOOL vs GPUsspe (Quadro 6000) Propagation Comparison on Double Precision 52

Figure 6.4 PETOOL vs GPUsspe (Quadro 6000) Propagation Comparison on Single Precision 52

Figure 6.5 PETOOL vs GPUsspe (Tesla C2075) Propagation Comparison on Double Precision 53

Figure 6.6 PETOOL vs GPUsspe (Tesla C2075) Propagation Comparison on Single Precision 54

Figure 6.7 PETOOL vs GPUsspe (Tesla K20) Propagation Comparison on Double Precision 55

Figure 6.8 PETOOL vs GPUsspe (Tesla K20) Propagation Comparison on Single Precision 55

Figure 6.9 Horizontal vs Vertical Polarization in SSPE 59

Figure 6.10 CPUsspe vs GPUsspe (Quadro 6000) with Terrain (50 km in rage - 300 m in height) 60

Figure 6.11 CPUsspe vs GPUsspe (Quadro 6000) with Terrain (100 km in rage - 1500 m in height) 61

Figure 6.12 CPUsspe vs GPUsspe (Quadro 6000) with Terrain (100 km in rage - 1500 m in height) after 5 km 62

Figure 6.13 CPUsspe vs GPUsspe (Quadro 6000) with Flat Terrain (100 km in rage - 1500 m in height) 62

Figure 6.14 CPUsspe vs GPUsspe (Quadro 6000) with Flat Terrain (100 km in rage - 1500 m in height) after 5 km 63

Figure 6.15 CPUsspe vs GPUsspe Test 7 and 8 in 2D 69

LIST OF ABBREVIATIONS

EM	Electromagnetic
PE	Parabolic Equation
SSPE	Split Step Parabolic Equation
FDM	Finite Difference Method
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
DST	Discrete Sine Transform
IDST	Inverse Discrete Sine Transform
DCT	Discrete Cosine Transform
IDCT	Inverse Discrete Cosine Transform
DMFT	Discrete Mixed Fourier Transform
GPU	Graphical Processing Unit
CPU	Central Processing Unit
AREPS	Advanced Refractive Effects Prediction System
TEMPER	Tropospheric Electromagnetic Parabolic Equation Routine
CUDA	Compute Unified Device Architecture

CHAPTER 1

INTRODUCTION

1.1 Problem Definition and Motivation

While EM waves travel in the troposphere, they are affected by various situations occurred in troposphere, such as reflection, refraction, diffraction, etc. These kind of occurrences cause abnormalities in propagation, that means unstable or changing atmospheric or tropospheric conditions act upon transmitted radio waves. Thus they prevent such waves from following their direct path, thereby cause difficulties and disruptions of communications. For instance, change in refractive index in the troposphere is one of main causes of these abnormalities[25].

In addition to tropospheric effects, irregular terrain surfaces, such as buildings, mountains, etc. have considerable influence on radio-wave propagation, since they reflect and diffract the electromagnetic waves in a complex way. When EM waves hit these surfaces, they loose remarkable amount of their power.

Moreover, ducting effect is another factor that changes the behavior of propagation. This effect is mostly observed over water when cold air is overrun by warm air. Ducting forms a thin atmospheric layer with a high refractive index, in which a signal reflects between the earth and the layer respectively along the propagation. In normal tropospheric conditions, radar signals propagate according to line of sight rule, however in ducting effect radar signals can propagate along the horizontal direction far greater than the normal radar range[28].

Hence, proper design of an effective radar or communication system can be achieved by using an accurate model that incorporates the tropospheric and environmental effects efficiently. There are many approaches that analyze these effects and design a suitable propagation model for the problem. Ray-Tracing, Normal-mode waveguide theory and Parabolic wave equations are main computational methods for modeling EM behavior [11, 13, 26].

Ray-Tracing is a highly used model for modeling the path of electromagnetic waves in propagation, however it is not so accurate at long distances, because it does not analyze refraction and diffraction effects properly. Ray tracing works on time domain, thus it computes the distance of a target from the radar by measuring time passed up to radar waves reaches the target. However, it cannot calculate the propagation factor at a given point, because it just knows the propagation path of EM waves.

Normal-mode waveguide theory accounts the diffraction, refraction effects, and basic geometric behavior of waves while analyzing EM wave propagation, it takes into account many factors while calculating propagation, and requires great precision in modeling the environment, however it is a computationally intensive method.

Parabolic wave equations provide accurate modeling of EM wave propagation. When compared with Ray-Tracing and Normal-mode waveguide theory approaches, parabolic wave equations fit between them in both accuracy and computation time. Additionally, the parabolic equations provide a complete solution at all points, and they account for atmospheric refractivity changes, ducting effects, and terrain factors, which has led to advances in propagation modeling and in the calculation of wave coverage range.

There are two main methods used to solve the parabolic wave equations. The first one is Split Step Parabolic Equation (SSPE) [12], and the other is Finite Difference Method (FDM) [27]. FDM is able to calculate the effects of the various boundary conditions, however it is much more time consuming and requires large amount of memory. However, The Split Step Parabolic Equation is more stable numerically, and enables flexible step sizes while computing propagation. Therefore, users can decide

the steps that the algorithm will run.

Graphics Processing Units (GPUs) are mainly designed for the gaming market and visual graphical purposes, however in the last decades they became more popular in scientific computing. As methods for electromagnetic modeling are computationally intensive, even small scenarios can have long running times. As a result, general purpose use of GPUs has increased, and take the attention of people who are interested in electromagnetic wave simulations. Also, the graphics cards are widely available, inexpensive, and rapidly improving, so GPUs become highly attractive when they are compared to their alternative special-purpose super computers which are very expensive.

1.2 Related Works

This section surveys previous works in modeling electromagnetic propagation in the troposphere using the parabolic equation approach. The parabolic equations are highly attractive formulations for radar and communication systems to analyze the electromagnetic wave propagation [12, 15, 34].

The studies on Split Step Parabolic Equation (SSPE), mostly aim to improve the algorithm's reliability by involving effects of real situations. There are many implementations of SSPE in CPU architecture. These are mostly developed for electromagnetic wave simulation purposes. AREPS (Advanced Refractive Effects Prediction System)[29], TEMPER (Tropospheric Electromagnetic Parabolic Equation Routine)[16], PETOOL [36] are mainly known examples.

The performance is significantly important for SSPE, and the main purpose in this thesis is to improve SSPE performance by using GPU resources. Therefore main related studies for us are mainly GPU based implementations of SSPE and their performance. In literature, there are a few GPU implemented SSPE works. The GPUwave [20], and the underwater acoustic propagation modeling study of Hursky [21] focus on the problem of underwater propagation. They simulate underwater sound waves using Split Step Parabolic Equation by taking advantages of modern graphics cards in 2D. Will explains the applicable electromagnetic wave modeling methods for GPU

architecture in his study "Electromagnetic Modeling with GPUs" [33], and implements the Finite Difference Time Domain (FDTD) method, which is an alternative method to SSPE, for modeling electromagnetic wave propagation.

1.3 Thesis Contribution

The main contribution of this research is to simulate EM wave propagation on non-flat 2D and 3D terrains by using GPUs' highly parallel architecture. Although, there are many studies on 2D implementations of SSPE in literature, to the best of authors knowledge, there is no study on 3D SSPE implementation on the GPU architecture. The other contribution of this thesis to show the effects of 3 different graphic cards and CPU on SSPE performance.

1.4 Thesis Outline

In this research, we have developed three implementation models for the SSPE algorithm in overall.

2D GPUsspe models one way propagation of 2D environments on the GPU architecture. This method accounts refractivity change in the troposphere, and additionally irregular terrain effects, and ducting effect are handled in an accurate way.

3D GPUsspe implementation is developed for 3D modeling of electromagnetic propagation. This approach reduces 3D environment to 2D environments, that is it divides the 3D environment into 2D slides and models each 2D slide independent from each other using GPU resources.

Lastly we have developed CPU implemented SSPE, called CPUsspe, to compare the CPU and GPU performance of the SSPE model.

The validation, verification, and calibration (VV&C) process is a critic point for electromagnetic wave simulations. To be sure about the reliability of a system, these validation, verification, and calibration tests must be applied correctly. We use PETOOL, which is a MATLAB based one way and two way Split Step Parabolic Equation tool

for radio wave propagation over variable terrain, for the VV&C process. It is stated in [36] that PETOOL uses AREPS (Advanced Refractive Effects Prediction System) to calibrate itself, and AREPS is a highly known and extensively used system for atmospheric propagation modeling. PETOOL is used for the accuracy tests, but for the performance tests, CPUsspe is used as a baseline, which is also developed as a parallel application.

Mainly, this thesis shows how the Split Step Parabolic Equation approach is adopted to the GPU architecture in 2D and 3D environments. The critical parts in the implementations, and the performance of SSPE in GPU and CPU architectures are also discussed.

The thesis is introduced in seven chapters as follows:

Chapter 2 gives background information for the electromagnetic wave propagation, and the important concepts in EM wave propagation are explained.

Chapter 3 presents the Split Step Parabolic Equation (SSPE) method and its theoretic definition. Fast Fourier Transform (FFT) is a critical part of the SSPE model, so FFT is also explained in detail in this chapter.

Chapter 4 gives brief information about graphic cards and GPU architectures. Computing environment of GPU provided by NVIDIA, namely Compute Unified Device Architecture (or CUDA for short), and the graphic cards that are used in the implementation and tests are explained.

Chapter 5 describes the GPUsspe and CPUsspe algorithms and their implementations. It emphasizes critical parts of models and gives some kernel code samples for important code segments.

Chapter 6 compares CPUsspe and GPUsspe implementations, and presents the performance and accuracy results of both algorithms.

Finally, Chapter 7 draws some conclusions and presents future work.

CHAPTER 2

BACKGROUND

2.1 Radar

Radar is an object-detection system which is utilized to find the location, direction, and speed of moving or stationary objects such as aircrafts, ships, and spacecrafts by using the electromagnetic waves.

Radars generate electromagnetic waves and send them to the atmosphere. Radars send electromagnetic especially radio waves, and when these waves hit any target in the path, most of the energy are scattered at collision (The objects in the propagation path of the radars are called "targets" or "echoes"). The remaining energy is reflected back toward the radar. The backward waves are received by the antenna which is mostly located at the same place with the transmitting radar.

The main goal of the radar is to find the distance between the object and itself, which is an easy process. Radar transmits radio waves to the direction of the target, and target reflects some of waves back to the radar. The time passed between the emission and receipt of the wave can be measured. As it is known that electromagnetic waves move at the speed of light, the distance can be calculated basically according to principles of physics as travel time multiplied by the speed of light.

The power of returned signal is related to the number of targets and their sizes. When electromagnetic waves hit a target, they are scattered in all directions. Also, the amount of scattering is directly proportional to the target size: when target size increases, the amount of scattering increases according to it. Additionally, as the num-

ber of targets increases, the returned signal gets stronger also. The returned signal to the antenna is called "reflectivity" which is relative to the number of encountered targets and their size[30, 31].

2.2 Electromagnetic (EM) Waves

Electromagnetic waves are generated from the disturbance formed by electric and magnetic field and they propagate uniformly in all directions starting from their origin. Some of EM waves are visible but some of them are not. Light, microwaves, x-rays, TV and radio transmissions are different kinds of electromagnetic waves.

Electromagnetic waves do not require any medium such as air or water to propagate. They move in any environment and repeat themselves over a distance that is called the wavelength. They travel at the speed of light which is 186,282 miles (299,727 kilometers) per second at space. This speed changes when the medium is changed.

While EM waves propagate, they are affected some outside effects as explained following subsections.

2.2.1 Reflection

The change in the propagation environment can cause change in the path of EM waves. If EM waves encounter a significant change in the density of environment while they are propagating, according to change level, some or all of them can pass to the new medium or they are reflected from it. The waves that pass to the new medium form transmitted portion and the other are called as reflected portion.

The part which is reflected has a very simple rule governing its behavior.

$$The Angle of Reflection = The Angle of Incidence$$

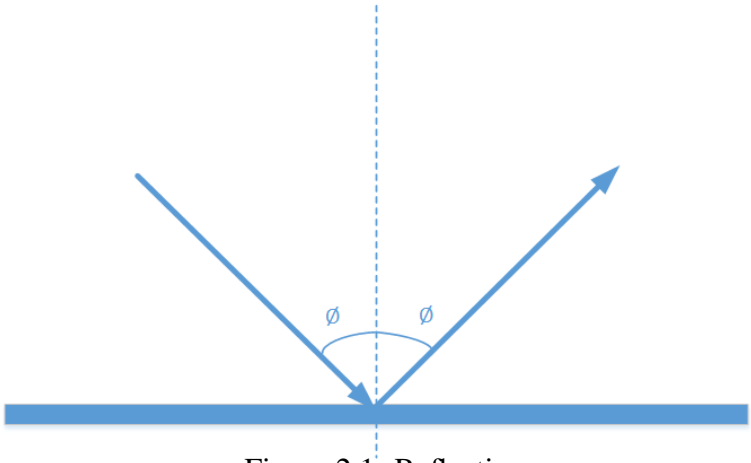

Figure 2.1: Reflection

2.2.2 Refraction

As stated in Reflection section, in the change of medium, some of EM waves pass to the new medium, and some of them are reflected. The direction of the passed waves change according to new medium properties. If the new medium has higher refraction index than previous one, the speed of waves decreases, and in order to stabilize the frequency, the wavelength of EM waves become shorter. Additionally, the angle between the perpendicular and the transmitted wave diminishes.

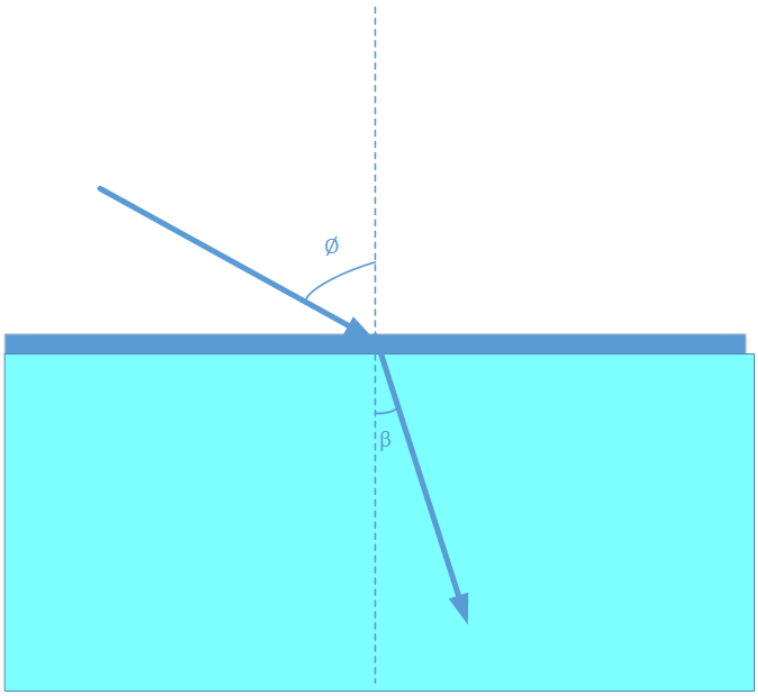

Figure 2.2: Refraction

In refraction, it is most significant to know in which direction the wave will refract than how much it will refract.

The relationship between the angles and indices of refraction is given by Snell's Law:

$$n_\beta \sin(\theta) = n_\theta \sin(\beta)$$

2.2.3 Diffraction

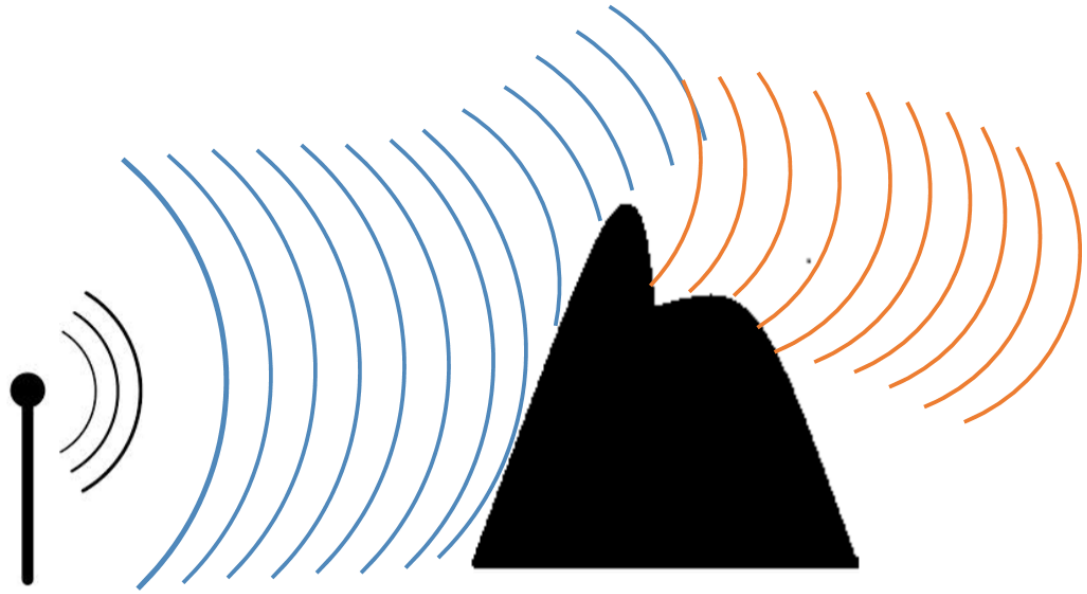

Figure 2.3: Diffraction

When radio waves hit an obstacle, they bend around the obstacle as shown in Figure 2.3. The bending, called diffraction, changes the direction of some radio waves from the normal line-of-sight path. Diffracted radio wave energy usually is weak, however it can be detected by a suitable receiver. Diffraction enables radio waves to propagate beyond the visible horizon.

CHAPTER 3

SPLIT-STEP PARABOLIC EQUATION (SSPE)

3.1 SSPE and Theoretical Definition

The electromagnetic wave propagation is concerned mainly with the Earth's atmosphere between the transmitting antenna and the receiving one. The Earth's curvature, atmospheric refractivity and terrain factors affect the propagation process seriously. The parabolic equation, which models one-way propagation, is introduced firstly by Leontovich and Fock [24], and this propagation model has been in use for many decades to analyze behaviors of waves during propagation.

Split Step Parabolic Equation (SSPE) method is a widely used parabolic equation for modeling the electromagnetic wave propagation in a homogeneous or an inhomogeneous atmosphere. It is a two dimensional (2D) ground-wave propagation model, which handles the Earth's curvature, the atmospheric refractivity variations, non-flat terrain effects, and the boundary losses.

The following explanations clarify the theoretic definition and its evolution in a detailed way[35].

Mainly, the parabolic wave equation is an approximation to the Helmholtz wave equation as follows:

$$\nabla^2\phi + k^2 n^2 \phi = 0 \tag{3.1}$$

or alternatively,

$$\frac{\partial^2 \phi}{\partial^2 x^2} + \frac{\partial^2 \phi}{\partial^2 z^2} + k^2 n^2 \phi = 0 \tag{3.2}$$

where x denotes range and z denotes altitude coordinates, and $n = n(x, z)$ is refractive index in range and altitude.

Additionally, $k = \frac{2\pi}{\lambda}$ is wave number, and ϕ is for electric or magnetic field in horizontal or vertical polarization.

The boundaries for the problem is sea/ground at the bottom and infinite extent from the top.

It is assumed, electromagnetic waves travel in forward and horizontal direction in general, and by using this approach the parabolic wave equation is approximated as following:

$$\frac{\partial^2 u}{\partial^2 z^2} + 2jk\frac{\partial^2 u}{\partial^2 x} + k^2(n^2 - 1 + \frac{2z}{a_e})u = 0 \tag{3.3}$$

where $u = e^{-jkx}\phi$ is the reduced function of electric or magnetic field when electromagnetic waves propagate in range, and $\frac{2z}{a_e}$ is Earth's curvature. Ignoring Earth's curvature means that the Earth is flat which is not a concern in many applications.

The approximation of the parabolic wave equation is a initial-value problem, that is the algorithm starts with an initial complex vector. The initial vector is calculated according to radar parameters. After computing initial field, the algorithm begins to process near the antenna and continues to process step by step in range. Each step uses the previous step as an input for its computation. There is a huge data dependency between steps.

After computing initial field, we can apply following expression to find the electric or magnetic field at range $x + \Delta x$

$$u(x + \Delta x) = \exp(jkm\frac{\Delta x}{2}) \times F^{-1}\left\{\exp\left[-jp^2\frac{\Delta x}{2k}\right] F\left\{u\left(x, z\right)\right\}\right\} \tag{3.4}$$

where F is Fourier Transform, p is transform variable defined as $p = k\sin(\theta)$, and $m = n^2 - 1 + \frac{2z}{a_e}$ is the modified refractive index.

In standard case, m is a function of height and/or range, however in SSPE as the problem is solved step by step in range, and the range between each step is small enough, the refractivity is assumed as constant between consecutive ranges.

When the initial field is known, equation (3.4) can be used to calculate $u(x, z)$ along x with steps of Δx.

In radio wave propagation, main aim is to calculate the propagation factor and propagation loss (path loss). After analyzing the fields inside the interested area via (3.4), the propagation factor is computed by the formula

$$PF = 20\log|u| + 10\log x + 10\log\lambda \tag{3.5}$$

And the path loss, the ratio of power radiated by the transmitter antenna and the power at a point in space, can be determined by following equation

$$PL = -20\log|u| + 20\log(4\pi) + 10\log x - 30\log\lambda \tag{3.6}$$

From theoretic formulation, Split Step Parabolic Equation is a computationally intensive method, it is highly dependent on Fourier Transform, and most of heavy computation is done during Fourier Transform. Therefore, the performance of Fourier Transform is crucial for SSPE performance.

The DFT is the most important discrete transform, which is preferred to perform Fourier analysis [32]. This approach is used in many fields, however computing it directly from the definition is often too slow to be practical. FFT is a way to compute the same result more quickly. FFT stands for Fast Fourier transform which is a method to compute Discrete Fourier transform (DFT) and its inverse in a more practical way[17].

For implementing Fourier Transform in Split Step Parabolic Equation(SSPE) algorithm, we use Fast Fourier Transform (FFT) method that reduces program running time.

3.2 Fast Fourier Transform (FFT)

Fast Fourier Transform is a widely used method in digital signal processing and image processing fields. These fields receive signals in the time domain; however, they require the frequency spectrum of the signal. Fourier analysis converts from time domain to frequency domain or from frequency domain to time domain.

The input samples are complex numbers, and the output coefficients are complex as well.

Let $x_0, x_1, ..., x_{N-1}$ be complex numbers. The DFT is defined by the formula

$$X_k = \sum_{n=0}^{N-1} x_n \exp^{-\frac{2\pi i}{N} kn} \qquad k = 0, 1, 2, ..., N-1 \tag{3.7}$$

This equation does calculation for N outputs X_k, and for each output N summations are calculated, so DFT is an $O(n^2)$ algorithm. DFT computationally takes so much time as the size of k increases.

FFT is a mathematical approach that computes the same results as DFT, however it is faster than DFT. It utilizes the symmetry of the DFT algorithm to make it run in a faster way. FFT is an $O(n \log n)$ algorithm.

Following FFT approach explains step by step how DFT is converted to FFT[14]

In DFT formulation, each input vector element is multiplied by $\exp^{-\frac{2\pi i}{N} kn}$ where N is the size of input vector, n is the corresponding index of element in the input vector, and k is the index of the element of the output vector.

$$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{N-1} \end{pmatrix} \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = X(k) \tag{3.8}$$

The input vector is divided into two parts and thus two separate summation formulas are created, in which one going from

$n = 0, 1, 2, .., N-1,$

and the other going from

$n = \frac{N}{2}, \frac{N}{2}+1, ..., N-1.$

$$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{\frac{N}{2}-1} \end{pmatrix} \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} + \begin{pmatrix} x_{\frac{N}{2}} \\ x_{\frac{N}{2}+1} \\ \dots \\ x_{N-1} \end{pmatrix} \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = F(k) \tag{3.9}$$

The summation index on the second sum is changed to match that of the first sum.

$$\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{\frac{N}{2}-1} \end{pmatrix} \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} + \begin{pmatrix} x_{0+\frac{N}{2}} \\ x_{1+\frac{N}{2}} \\ \dots \\ x_{\frac{N}{2}+\frac{N}{2}-1} \end{pmatrix} (-1)^k \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = F(k) \tag{3.10}$$

Let us take $n \to n + \frac{N}{2}$ in the second sum. This introduces an exponential term of the form

$$e^{-\frac{2\pi i}{N}\frac{N}{2}k} = e^{-\pi ik} = (-1)^k \tag{3.11}$$

By factoring appropriately, multiplication per input element is eliminated. (This translates to a net savings of N/2 MADD operations.)

$$\left[\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{\frac{N}{2}-1} \end{pmatrix} + \begin{pmatrix} x_{0+\frac{N}{2}} \\ x_{1+\frac{N}{2}} \\ \dots \\ x_{\frac{N}{2}+\frac{N}{2}-1} \end{pmatrix} (-1)^k\right] \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = F(k) \tag{3.12}$$

The output vector is divided into two, one calculates the results for odd ks, the other for even ks.

$$\left[\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{\frac{N}{2}-1} \end{pmatrix} + \begin{pmatrix} x_{0+\frac{N}{2}} \\ x_{1+\frac{N}{2}} \\ \dots \\ x_{\frac{N}{2}+\frac{N}{2}-1} \end{pmatrix}\right] \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = F_{even}(k) \tag{3.13}$$

$$\left[\begin{pmatrix} x_0 \\ x_1 \\ \dots \\ x_{\frac{N}{2}-1} \end{pmatrix} - \begin{pmatrix} x_{0+\frac{N}{2}} \\ x_{1+\frac{N}{2}} \\ \dots \\ x_{\frac{N}{2}+\frac{N}{2}-1} \end{pmatrix}\right] \begin{pmatrix} e^{-\frac{2\pi i}{N}kn} \end{pmatrix} = F_{odd}(k) \tag{3.14}$$

Each output element requires $\log_2 N$ operations, and since there are N output elements, in total there are $O(Nlog_2N)$ operations.

There are many algorithms that compute Fast Fourier Transform, and many libraries are developed to analyze FFT accurately. The 2D implementation of SSPE in CPU, which is called CPUsspe, uses Intel Math Kernel Library's (IMKL) FFT library [22]. FFT is a suitable algorithm to be parallelized in GPU architecture, CUDA also has an FFT library. Therefore, 2D GPUsspe uses NVIDIA's cuFFT library to handle FFT operations. However, in 3D GPUsspe, we use Dynamic Parallelism and we need to call FFT functions inside kernels, but cuFFT does not allow to call its FFT functions inside a kernel. Therefore, we implemented the previously explained FFT method which is highly proper for GPU's grid architecture. This FFT implementation is used in 3D GPUsspe, and it only works on 2^n sized input sets.

CHAPTER 4

GRAPHICS PROCESSING UNIT (GPU)

The Graphics Processing Unit (GPU) is a processor that was specialized for processing graphics. GPUs are introduced for PC industry in August 31, 1999. The technical definition of a GPU is "a single chip processor with integrated transform, lighting, triangle setup/clipping, and rendering engines that is capable of processing a minimum of 10 million polygons per second."[5]

The main concern of GPUs are graphical processes, they are optimized for 2D/3D graphics, video, visual computing, and display. They can process images more quickly than central processing units via the ability they get from their highly parallel architecture, which efficiently processes large data blocks in parallel. GPUs provide a highly parallel and multi-threaded multiprocessor for visual computing. Thus, they can perform in real-time on graphics images and videos.

The high performance of GPUs make them very popular for general purpose algorithms that are suitable for GPU's grid architecture. Although GPUs are mostly used in graphical purposes in past years, they have evolved towards a more flexible architecture and now they can implement any algorithm, not only graphics. GPUs are now very popular in many fields from academic to industry such as finance, medical, biophysics, audio, video, imaging.

4.1 CPU vs GPU

It is not correct to state that GPU is better than CPU or vice versa. For different types of problems, different processing units may be more suitable for the problem at hand.

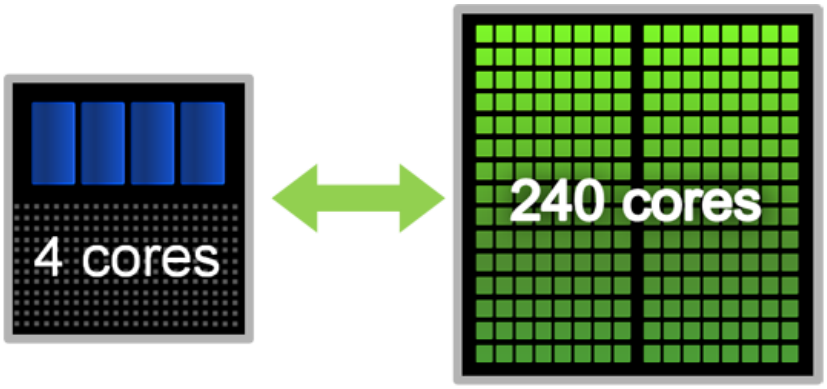

Figure 4.1: CPU vs GPU

Comparing the processing procedure of CPU and GPU can give us a good understand of their abilities and differences.

The central processing unit, or CPU for short, serves all essential processing needs. They are generally optimized for serial tasks, and have a few powerful cores. They can handle intensive computational works with just these few cores, however these types of computationally intensive processes take so much time.

General purpose GPUs (GPGPU) are mainly developed to be a solution to the performance problem of CPUs, since some problems could not be handled in CPU efficiently. GPUs have thousands of smaller cores which are designed for processing parallel small tasks [9]. They can do small, data independent tasks efficiently in parallel. Studies show suitable problems can perform in a much more better way in GPUs than CPUs[4].

One of the main differences between these two processors is that creating a CPU thread is much more costly than creating a GPU thread. GPU threads are extremely lightweight, they have very little creation overhead. Therefore, a GPU thread provides better performance in small tasks, which is run in parallel, however CPU threads are designed for computationally intensive and serial processes.

Another point is that, GPUs have significantly faster and more advanced memory interfaces. GPUs execute on independent data sets and each thread is responsible from the data set for itself, and the data is needed to be shifted around a lot more compared to CPUs in execution. GPUs are much more deterministic than CPUs in their operation. Determinism is a crucial issue for many applications, mostly for

real-time applications. Therefore, for problems that need high determinism GPUs are more suitable.

GPUs are highly parallel architectures, however the important point is to decide which architecture is suitable for the solution of a problem, because in some cases GPUs can give terrible results if you do not know whether your problem fits to the GPU architecture or not.

4.2 The GPU Architecture

The CPU and GPU can work in parallel with each other inside the PC. There is a communication bridge between them, called the host interface. GPU receives commands from the CPU via this host interface. The communication is supported by a command buffer in this communication bridge. There are two threads, one runs on CPU, the other is on GPU. CPU sends commands to this communication buffer, and GPU receives them from there. If the command buffer is empty, GPU waits until the new input comes. Otherwise, it is full, CPU waits for GPU to finish its operations.

Except from the communication bridge, there is also a PCI express connection between CPU and GPU to enable data flow. The data is sent to device memory from the host memory or carried from the device memory to host memory by this PCI expresses bus.

GPUs do not follow the traditional sequential execution model, therefore synchronization becomes a critical issue most of the times. Although GPU and CPU are two different cards, they are not so independent from each other from the programming/programmer view. GPU computing process is started by CPU inside host code, therefore in some parts of the code, CPU needs to wait for the output of GPU process. Critically, they need to wait each other if they are trying to reference the same data to ensure synchronization. For these kinds of synchronization problems, semaphore style operations are preferred.

Figure [19] is an example of Fermi architecture. GPUs are combinations of Stream Multiprocessors (SM). The structure of SM can differ from GPU to GPU according

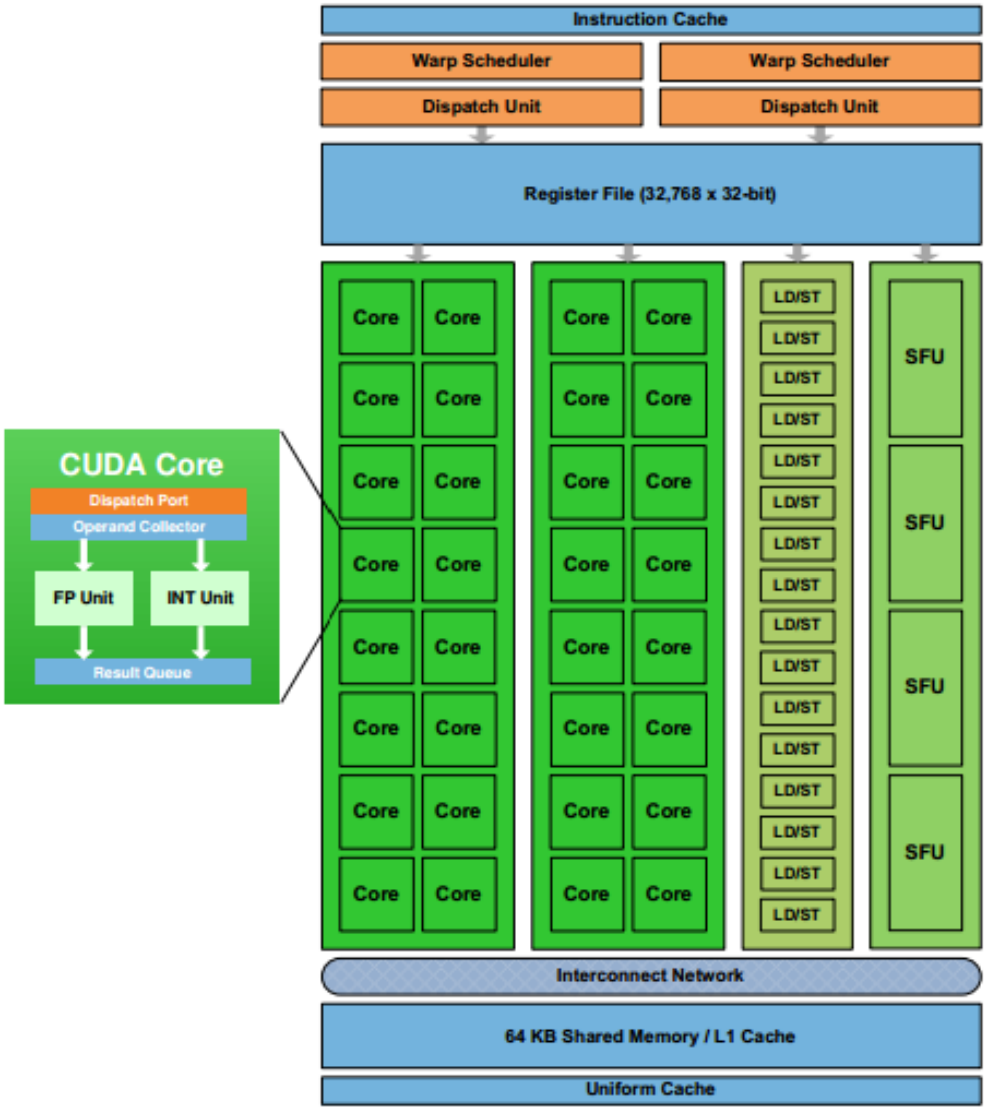

Figure 4.2: Fermi Architecture

to architecture type such as Tesla, Fermi, Kepler, etc. "Each Fermi SM includes 32 cores, 16 load/store units, four special-function units, a 32K-word register file, 64K of configurable RAM, and thread control logic. Each core has both floating-point and integer execution units" [19].

One of the most important specification of GPU architecture is that they serve as both a programmable graphics processor and a scalable parallel computing platform. CUDA, OpenCL, and Direct Compute provides GPU programming platforms for developers. Briefly, CUDA, which stands for Compute Unified Device Architecture, is developed by NVIDIA, and is a C based language with some extensions[1]. OpenCL is initially developed by Apple Inc., but now in collaboration with technical teams at AMD, IBM, Intel, and NVIDIA[7]. Direct Compute is developed by Microsoft, and works independent of GPU hardware[2].

We have chosen CUDA for GPU programming in the thesis. It is only supported for NVIDIA graphic cards. CUDA provides abilities and advantages of using high-level languages such as C to develop applications. The abilities of CUDA platform are always developing, and each new version introduces new functionalities.

4.2.1 Compute Unified Device Architecture (CUDA)

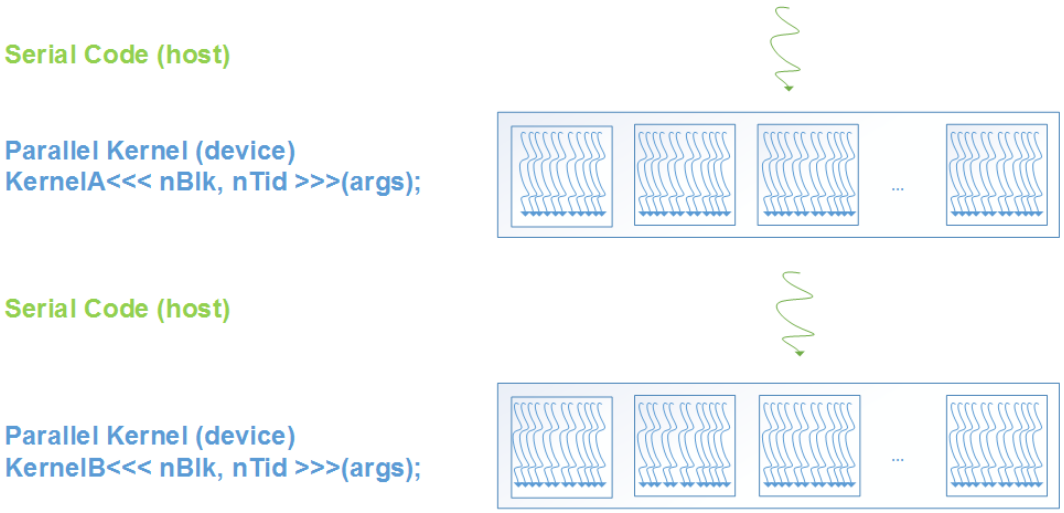

Figure 4.3: Host Device Execution

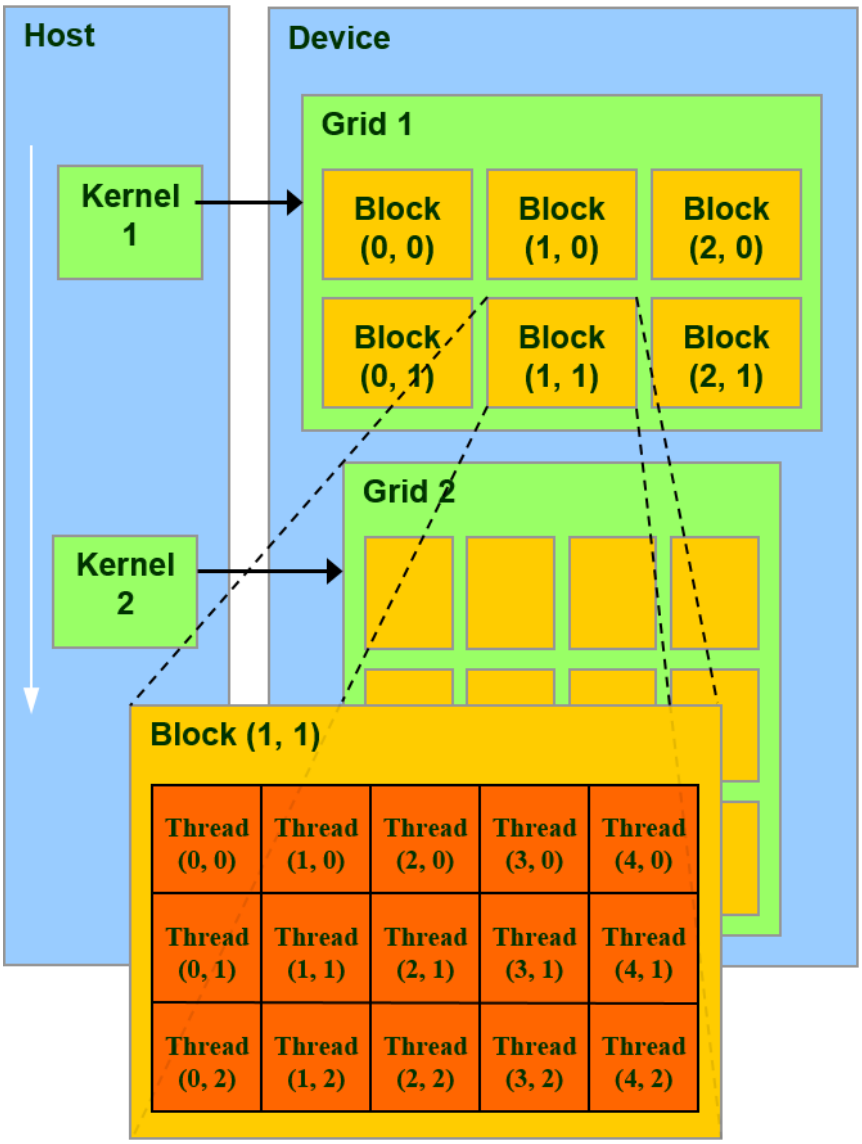

Figure 4.4: GPU Acceleration Works

CUDA is a parallel computing platform and programming model invented by NVIDIA[1]. It provides a development environment for NVIDIA generated graphic cards. The parallel parts of applications are executed on GPUs by invoking device kernels programmed in CUDA.

CUDA provides instruction sets to do parallel operations and to access the device memory. Also, it has GPU accelerated libraries to do many arithmetic and vector operations in parallel, such as cuFFT, cuBLAS, NPP, cuSPARSE, Thrust, etc.

In CUDA, CPU is called as "host", GPU as "device". Kernel is a function that runs on the device. It is executed by a grid of thread blocks. A thread block is a collection of threads that have a shared memory and run synchronized with each other. Threads of different blocks are not synchronized at their execution. Except the Kepler architecture, a single kernel is executed at a time, that is same kernel code is executed by group of threads on different parts of data. Each thread in a block has a unique id to access the memory and make control operation.

In Figure 4.3, serial code parts are run on the host (CPU), the code portion "kernel«<nBlock, nTid»(args)" run in device (GPU) which is invoked by host. A kernel launch run in blocks of a single "Grid" as demonstrated in Figure 4.4.

Number of blocks in a grid, and thread count in each block is specified in kernel launch, they are not fixed, they are user defined parameters. Also, the arguments for kernel function is sent at the time of the launch. The block and thread count numbers are critical for kernel performance. They should be optimized to get full performance. Maximum number of blocks for a grid, and maximum number of threads for a block are dependent on the GPU architecture, these restrictions should be taken into account during GPU programming.

In "Parallel Kernel (device)", a single kernel code is run for each thread at a time. The execution is done according to Single Instruction Multiple Data (SIMD) rule, that is each instruction is executed sequentially and at the same time for all threads. To be more clear, for instance in if clause statements, the threads that will run the if case will do their tasks, the others that are waiting for else case will just wait, and not do any other work. When if case finishes, the process go into else statement, and the

threads which are run in if statement previously will wait, and the other threads will execute inside else case. Thus, no two different instructions will be executed at the same time.

CUDA platform provides a powerful debugging and profiler tool, NVIDIA Nsight[6]. This tool enables user to debug the kernel codes, otherwise it is so difficult to find errors in parallel executing tasks. Also, Nsight profiler shows the bottleneck parts of the code, SM utilization, and kernel execution times, that is useful for optimizing the GPU performance.

4.2.2 GPU Memory Types

GPUs have different memory types, called as register, shared, constant, texture, local, and global memory. Register and shared memory actually resides on the GPU chip. However local, constant, global, and texture memory reside off the chip. Local, Constant, and Texture are all cached. In terms of speed, when we sort them from fastest one to slowest, Register File is the fastest one, then Shared Memory is the second fastest memory type. Constant Memory, and Texture Memory are on the third and fourth orders, respectively. Lastly, Local Memory and Global Memory are the slowest ones among others, and have the same speed.

Each memory is designed for different purposes and has specific features:

Register memory, belongs to a thread. The data stored in the register file is only accessible from the thread that wrote it. The life time of that data lasts until the lifetime of that thread ends.

Local memory, has the same scope rules as register memory, but performs slower. The variables declared inside kernel code are stored inside local memory.

Shared memory, belongs to all the threads off same block. Threads from different blocks cannot access each other's shared memory. It exists until the block ends, and enables the communication between the threads within a block, thus they can share data among them via this memory.

The data inside the global memory is visible to all threads created in the application.

The CPU (host) has also access to global memory. The data in global memory last until the deallocation from the host.

Constant memory, is used for data allocation, whose value will not change during kernel execution. Constant memory is read only. This memory type increases performance when a warp of threads read the same constant memory.

Texture memory, is also read only memory. Big data sets that do not change during kernel execution can be stored inside texture memory to access it more quickly than global memory.

4.3 Used Graphic Cards

NVIDIA CUDA parallel computing program works on 3 types of graphics cards, namely GeForce, Quadro and Tesla. GeForce and Quadro are developed for visualization, Tesla cards are designed for parallel computing and programming that requires intensive computation capabilities.

In this study, we work with 3 graphics cards, one is Quadro 6000 from Quadro family, the other two are from Tesla group namely Tesla C2075 and Tesla K20.

4.3.1 Quadro 6000

Quadro 6000 is specialized for graphical operations such as animation and video applications, it is a Fermi based graphic card. Fermi is the first ECC memory supported architecture, and it has also improved atomic memory operation performance. ECC memory (Error-correcting code memory) is important in long computational processes, since it provides more accurate results. All graphic cards that we used in this study have ECC memory, which is important for our problem accuracy.

4.3.2 Tesla C2075

Both Quadro 6000 and Tesla C2075 are developed on the Fermi architecture and have 448 cuda cores. The memory bandwidth of them is same that means they have equal

limitation in graphical data transfer. Although these two graphics cards have the same architecture, Tesla C2075 is better from Quadro 60000 in computational performance. The reason for that is Tesla GPUs are designed with "exclusive features" to maximize their performance[10]. To explain in detail, Tesla cards have full double precision floating point performance, faster PCIe communication (two DMA engines for bi-directional PCIe communication), higher performance CUDA drivers for Windows OS that is TCC (Tesla Computer Cluster) driver that reduces CUDA kernel overhead and enables Windows Remote Desktop and Windows Services.

4.3.3 Tesla K20

The third graphic card is Tesla K20, it is built on the Kepler architecture which is developed by Nvidia as the successor to the Fermi micro architecture. Kepler architecture brings a new ability "Dynamic Parallelism", which enables kernels to call child kernels inside themselves. In Fermi, only CPU can launch a kernel, now GPU kernels can call new kernels; thus, there is no need to go back to CPU. It supports recursive kernel calls and we use this ability in 3D GPUsspe.

In the Fermi architecture, each stream multiprocessor has 32 CUDA processing cores. In Kepler, each SM is called "Next-generation Streaming Multiprocessor", which NVIDIA abbreviates as "SMX"; each SMX has 192 CUDA cores. Therefore, Tesla K20 has 2496 CUDA cores in total.

In Dynamic Parallelism 4.5 , a thread that launches new grids belongs to the parent grid, the new grid created by the invocation is the child grid. The parent grid does not end until the child grid finishes its task. Thus, the parent grid can access the output of child grid without CPU involvement.

Dynamic Parallelism is supported on graphic cards which have Compute Capability (CC) 3.0 or more. Compute Capabilities describe the features supported by a CUDA hardware. Quadro 6000 and Tesla C2075 have compute capability 2.0, Tesla K20 has CC 3.5.

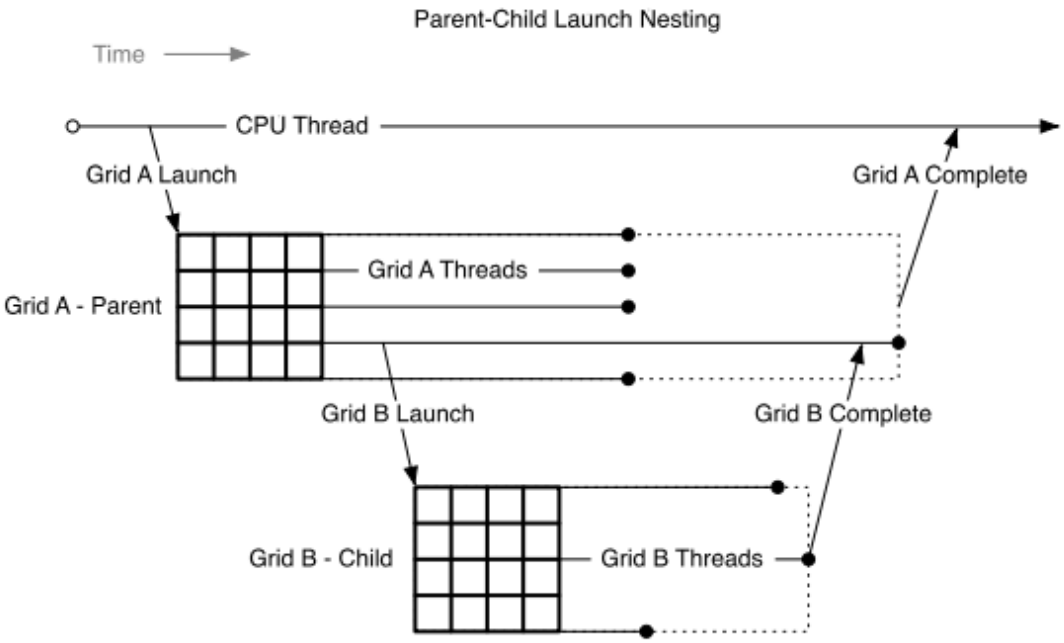

Figure 4.5: Dynamic Parallelism

CHAPTER 5

SSPE ON GPU

In this thesis, for modeling the tropospheric propagation in 2D, we have used the SSPE method that is developed by Ozgun[35] which is explained in Chapter 3. We just implemented the one-way forward propagation method, because two-way propagation approach does not suit well to GPU architectures. Two-way SSPE uses a buffering mechanism for backward waves, that is, the process collects the reflected waves and buffers them, and after forward waves finish, it starts to compute backward waves. The backward waves affect previously computed areas, so the same calculations are done repeatedly. Executing in a such manner is not suitable for the independent small tasks nature of the GPU architecture.

For 3D propagation modeling, the 3D environment is accepted as combination of 2D slices, thus 3D is adapted to GPU grid architecture, and each 2D slice is modeled independent of each other. The out-of-plane scatterings and interactions between the 2D slices are ignored and this ignorance does not cause big effects on propagation outputs. TEMPER is a commonly used 3D propagation modeling tool, and it neglects out-of-plane scatterings. The study "Modeling Radar Propagation in Three-Dimensional Environment"[18] by Awadallah analyzes out-of-plane scattering effects on 3D propagation model (TEMPER does not account for these effects), the results show the difference is about 10 - 20 dB in urban area, however in digital terrain map the error rate is much less than this. Therefore, we ignore these effects in our model also, otherwise 3D wave propagation becomes computationally very intensive, and hard to implement in GPU.

5.1 SSPE Model

The SSPE model is visualized to make the process flow more accurate, as seen in Figure 5.1. The algorithm is an initial value problem, so it takes initial field, which is calculated according to radar parameters, as an input to initiate the propagation. Then it starts to iterate near the antenna and this process continues step by step along the x direction (in range) until the maximum range is achieved.

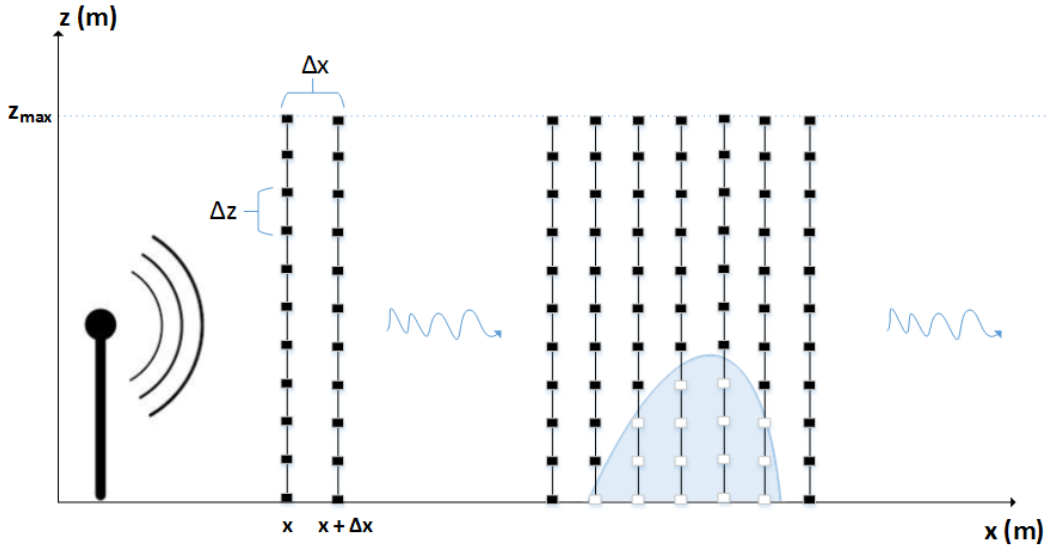

Figure 5.1: Split-Step Parabolic Equation Process

Each line drawn symbolizes the field at corresponding range. Each field is a complex array, and an input to the next iteration process. Therefore this algorithm is highly a sequential process, which is a disadvantage for the GPU architecture. However, when it is optimized for GPU it performs better than CPU.

The SSPE domain operates between z and p domains, Fourier Transform pairs, in a continuous manner. In the discretization process, it executes at z_{max} and p_{max}, that means the process travels Δx at each step, and the field vectors are defined according to Δz. When maximum height is decided (z_{max}), p_{max} is calculated according to Nyquist criterion $z_{max} \times p_{max} = \pi N$ where N is the FFT size. $p_{max} = k \sin(\theta_{max})$ where θ_{max} is the maximum allowable propagation angle. As $\Delta z = z_{max}/N$, the altitude increment should satisfy $\Delta z \leq \lambda/(2 \sin \theta_{max})$. The Δz selection is critical, but Δx can be chosen by user without any restrictions.

SSPE is a forward propagation algorithm, it processes forward waves and neglects backward ones. Therefore, during propagation, when the field hits the terrain, the parts which intersect with the terrain are set to zero and the other part of field continues to move in forward direction, and the reflected waves are ignored. Ignoring backward waves causes some propagation loss, however as most of the wave energy travels in forward direction, the loss does not affect total propagation so much.

5.1.1 Treatment of Irregular Terrain : Staircase Approach

SSPE models wave propagation over a flat or a non-flat terrain. For flat terrains there is no elevation in the terrain, therefore no need to model the terrain; however in non-flat ones, they need to be modeled to be processed by the algorithm easily.

SSPE is a discretization process, therefore non-flat terrains should be adopted to this process also. For forward propagation, staircase approach is used. In this method, terrain is divided into segments in which each segment has constant height value. Thus the slope of corners are ignored and the terrain becomes more convenient for the algorithm. Although it seems that this is a simple way of modeling non-flat terrain, it gives accurate results in the approximate sense.

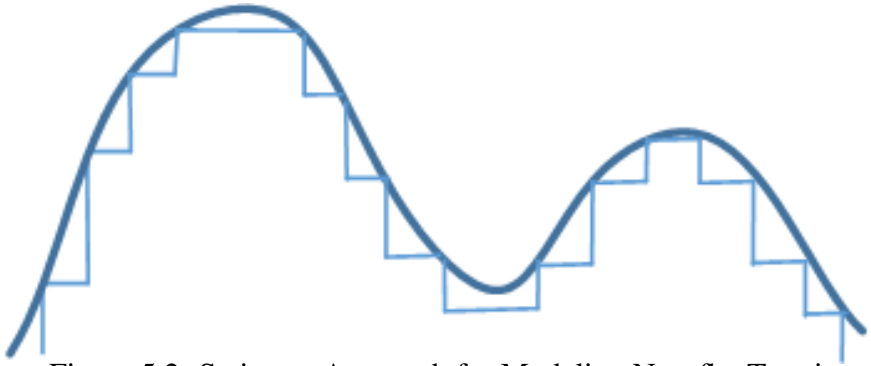

Figure 5.2: Staircase Approach for Modeling Non-flat Terrain

5.1.2 Map Architecture

The map structure is created for two and three dimensional environments. 2D GPUsspe uses 2D map structure, and 3D GPUsspe uses 3D map; however, as 3D GPUsspe is implemented as a combination of 2D slices, in practice it also uses 2D map for each

of its 2D slide.

2D map structure holds six different data, namely maximum range, maximum altitude, map resolution, map elevation values, and map surface types in same file structure. There are $maximum\_range/map\_resolution$ elevation values in the map structure. Map elevation values are stored in single precision domain, because single precision is enough to hold altitude values.

Maps are created by user defined parameters. The user enters maximum range and maximum altitude values with map resolution, and then s/he enters some elevation values for some indexes of the map. Map index values are a set of sequentially listed numbers, started from 1 to N where $N = maximum\_range/map\_resolution$. Then the elevation values are used to create terrain by applying linear interpolation on the entered elevation values.

Surface map gives information about the surface type of terrain. The surface type is a range dependent value. There are 5 different surface types used: sea, fresh water, wet ground, medium dry ground, and very dry ground. These are also user defined values. The user decides the surface type of each range interval.

The surface type is used for conductivity and relative permittivity of the Earth's surface in impedance boundary conditions while calculating Discrete Mixed Fourier Transform.

5.2 SSPE Work-flow

The work flow of SSPE algorithm lists each iteration of the algorithm step by step:

1. Load input and antenna parameters.

   The propagation coverage in range and height, terrain features, troposphere effects, and ground surface features are defined as an input parameter in the initial step of the algorithm.

   These parameters are mainly domain parameters (max-range, max-altitude, etc.), antenna parameters (polarization, 3 dB beam-width, elevation angle, antenna

height, frequency, etc.), surface parameters (perfectly conducting or impedance surface, surface type (sea, fresh water, wet ground, etc)) and atmosphere parameters(range dependent or independent refractivity)

According to these input parameters, the initialization process is performed, and the computations that will not change in each iteration are also done in the initialization step.

2. Apply window function.

A window function is a frame that draws the borders of the interested interval of propagation. It is zero-valued outside of that frame. There are two types of window functions that are used in the implementation, namely Hamming and Hanning windows.

3. Calculate atmospheric refractivity.

There are two types of atmosphere, range dependent and range independent in refractivity. That means if range independent refractivity is chosen, the refractivity is same for all ranges. It just changes according to height value.

Otherwise, in range dependent refractivity, the refrectivity values change in range and height. Refractivity calculations are done according to atmosphere types; standard atmosphere, surface duct, surface-based duct, elevated duct and evaporation duct. The user can define the range and height intervals, then assign atmosphere types to these intervals.

4. Calculate "Initial Field" using antenna parameters at range x_0

"Initial Field" is a complex array, which holds the amplitude and phase of each height point separated by Δz.

5. Using "Initial Field" array as an input to the Equation (3) calculate next field at $x_0 + \Delta x$.

Next field is a complex array and has same structure with "Initial Field". It uses "Initial Field" as an input parameter for its computation.

6. Use new calculated field as an input to the next iteration and calculate the field at $x_0 + 2\Delta x$.

7. SSPE algorithm iterates sequentially and this process keeps on to the last step in range.

After all steps in range are computed, we get a 2D complex matrix of all interested area.

8. Calculate propagation factor on this 2D complex matrix and get propagation factor in each point.

9. Calculate path loss, which is the decrease in the power of electromagnetic wave while it propagates in the troposphere.

5.3 SSPE Algorithm

The work-flow of CPUsspe and 2D GPUsspe are same, therefore there are some parts in the implementation where the CPU and GPU version SSPE use commonly. First 4 steps of SSPE work flow are common, that is window function, refractivity, and initial field calculations are done by a common library.

The separation starts at the fifth step, each architecture computes each single field according to its architecture and resources, so the parallelization process become important in each single field computation which is a heavy process.

5.3.1 Refractivity in SSPE

Electromagnetic wave propagates in free space at maximum speed it can have, however when it enters a material, it slows down. The refractive index or index of refraction (n) is the ratio of its velocity in vacuum to that in the material. As all the parts of troposphere does not have same medium features, the velocity and propagation direction of EM waves change while they propagate.

A new refractivity model, which also involves Earth's curvature, called "modified

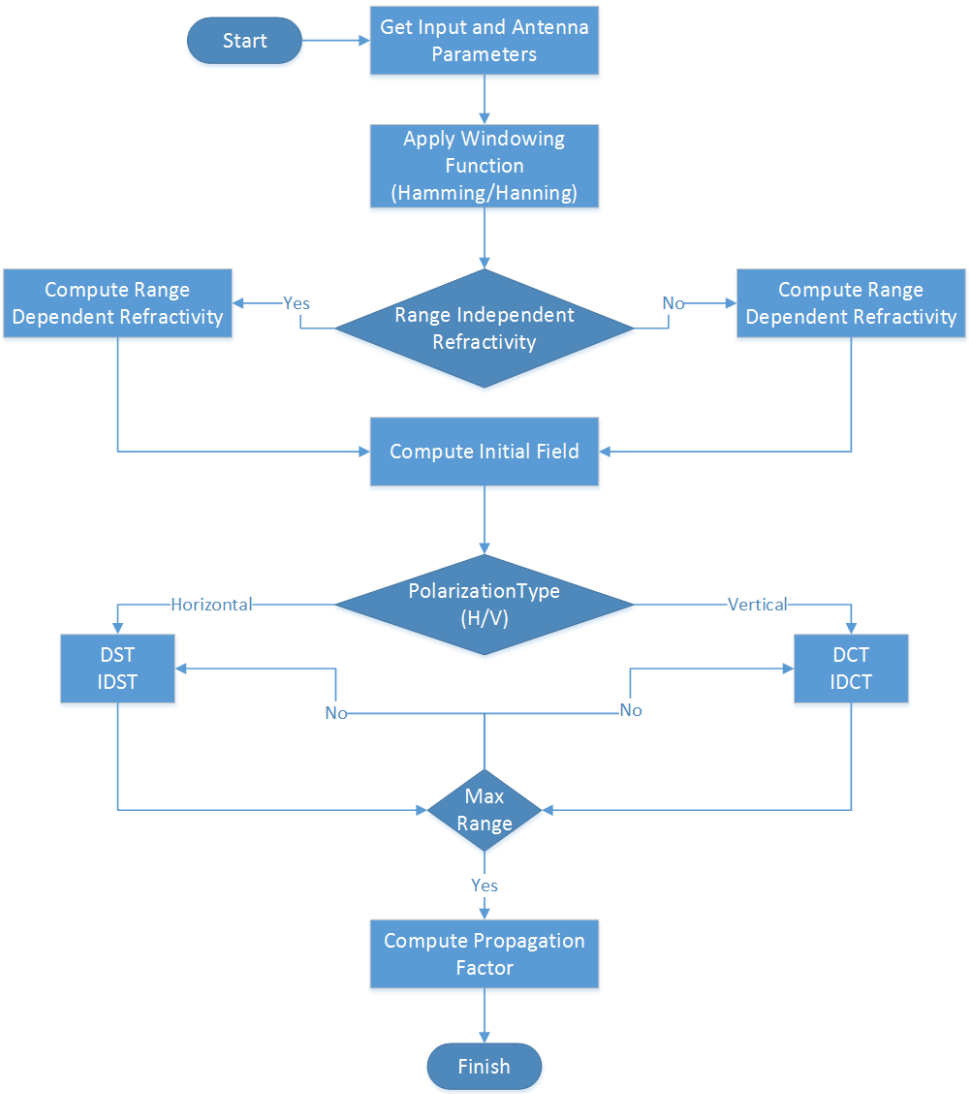

Figure 5.3: Split-Step Parabolic Equation Work Flow

refractivity" is defined as

$$M = (n^2 - 1 + 2z/a_e) \times 10^6 \quad (M - units) \tag{5.1}$$

where a_e is the Earth's radius, z is the height above the surface, and $2z/a_e$ corresponds the Earth's curvature.

The pressure, temperature, and water vapor levels change in both space and time in troposphere, which are the main causes of refractivity. The refractivity is categorized into 4 different types according to the vertical gradient dM/dz of modified refractivity.

* Sub-refraction: $dM/dz > 118M - units/km$

* Standard: $dM/dz = 118M - units/km$

* Super-refraction: $dM/dz < 118M - units/km$

* Ducting: $dM/dz < 0/km$

Except from standard case, the electromagnetic waves behave abnormal due to refraction. In sub-refraction, waves go in upward direction, otherwise in super-refraction and ducting they bend towards the downward way. Ducting is critical in refraction, since in ducting case radar waves propagate more than their normal horizon, thus radars can sense the targets that are far from their normal line of sight view. There are 4 different ducting effects that are implemented in the SSPE, namely surface duct, surface-based duct, elevated duct, and evaporation duct.

5.3.2 Initial Field Calculation

To compute the initial field, three parameters are needed: the height, the beam width, and the tilt angle of the antenna that will be modeled. The initial field is usually start at range, x = 0.

The first step is to specify the initial field in the p domain by

$$U(0,p) = f(p)\exp(-ipz_a) - f^*(-p)\exp(ipz_a) \tag{5.2}$$

The initial field in spatial z-domain is found by taking the inverse Fourier Transform (IFFT) of previous equation $U(0,p)$.

The antenna pattern is specified by three parameters: height z_a, the 3dB beamwidth θ_{BW}, and the elevation angle θ_{tilt}.

Gaussian antenna pattern is preferred for most applications to model the antenna pattern, so horizontally polarized Gaussian antenna pattern can be defined by the following equation

$$f(p) = \exp\left[-p^2w^2/4\right] \tag{5.3}$$

The tilt angle is introduced where $w = \frac{\sqrt{2\ln 2}}{k\sin(\theta_{BW}/2)}$ by shifting the antenna pattern, i.e., $f(p) \rightarrow f(p - k\sin(\theta_{tilt}))$

5.3.3 A Single Field Calculation

The standard SSPE can not model the boundary conditions automatically. The boundary conditions are handled according to surface type.

The boundary conditions that must be satisfied over the Earth's surface is expressed as

$$\left[\alpha_1\frac{\partial}{\partial z} + \alpha_2\right] u(x,z) = 0 \tag{5.4}$$

where α_1 and α_2 are constants.

In perfectly conducting surfaces, it is assumed as all electromagnetic waves are reflected without any loss. According to polarization type, Dirichlet(horizontal polarization) and/or Neuman(vertical polarization) types can be used. For perfectly conducting surfaces in horizontal polarization,where $\alpha_1 = 0$, and in vertical polarization, where $\alpha_2 = 0$, boundary conditions are handled by the Sine and/or Cosine Fourier

Transform. That is SSPE implementation changes according to antenna polarization. If antenna is polarized horizontally Discrete Sine Transform and its inverse IDST, otherwise in vertically polarized position, Discrete Cosine Transform (DCT) and IDCT are applied. Each single line shown in Figure 5.1 is mainly calculated as a combination of DST and IDST or DCT and IDCT processes.

For the impedance boundary conditions, the idea of Mixed Fourier Transform (MFT) and Discrete Mixed Fourier Transform (DMFT) are introduced by Kutter and Dockery[15, 23]. For lossy ground surfaces, DMFT is used to investigated the radio wave propagation in the troposphere. For horizontal polarization $\alpha_1 = 1$, and $\alpha_2 = ik(\epsilon_r + i60\sigma\lambda)^{1/2}$, for vertical polarization $\alpha_1 = 1$, and $\alpha_2 = ik(\epsilon_r + i60\sigma\lambda)^{-1/2}$, where σ and ϵ_r are the conductivity, and relative permittivity of the Earth's surface respectively.

Additionally, the propagation problem is an open region in vertical direction, therefore the condition $u(x,z)|_{z\to\infty} = 0$ must be satisfied. While the field is traveling in z direction, the field is cut at specified height, this causes non-physical reflections. To diminish these unwanted reflections, the maximum height is increased. Thus, the field at the real maximum height is computed more realistically. The enlargement of upper boundaries is done by using windowing functions (Hamming, Hanning, etc.). The maximum altitude of interested field is increased and then after process finishes, the propagation region is returned to the boundaries of the actual scenario.

5.4 Implementation

5.4.1 Implementation Environment

Common parts and CPUsspe are implemented using IBM Rational Rhapsody 7.6.1, which is a design, development and test environment [8]. Codes are generated using C++ programming language.

GPUsspe in 2D and 3D have been developed by us in Visual Studio 2010 using CUDA version 5.5. For debugging and profiling purposes NVIDIA Nsight 3.5 is integrated to the CUDA platform.

The system is developed and tested on 2.9 GHz-CPU powerful workstation, which has 32 GBs of RAM. It has 2 Intel Xeon E5-2690 processor, each with 8 cores, 16 cores in total (32 logical cores).

Additionally, the figures of propagation factor are drawn using MATLAB 2012b.

5.4.2 A Single Field Implementation in CPUsspe

In CPUsspe, all vector operations and FFT calculations are done in parallel by the help of Intel Math Kernel Library (IMKL)[22], that is CPUsspe is parallelized as much as IMKL supports.

Other than IMKL, to improve the performance of CPUsspe, some optimizations are done at the operation level. We avoid to use costly operations as much as possible such as mod, floor, etc.

However, the parts of CPUsspe code that cannot be parallelized are implemented as serial such as propagation factor calculation 3.5.

5.4.3 A Single Field Implementation in 2D GPUsspe

In 2D GPUsspe, we used cuFFT, and cuBLAS libraries of the CUDA platform. cuFFT is the FFT library of CUDA, and cuBLAS is a commonly used vector library for vector based operations. These are main difference of GPUsspe from CPUsspe in the two dimensional environment. FFT is a highly parallel algorithm, and GPUs are very good at FFT performance, so CUDA math libraries mostly perform more efficiently than Intel MKL in performance.

The other implementation difference of GPUsspe from CPUsspe is complex vector-vector multiplication. In SSPE, element wise complex vector - vector multiplication is a highly used operation; however, cuBLASS library does not have any interface for this operation. Vector - vector multiplication is highly adaptable for GPU grid architecture, that is each element in each vector set is multiplied with each other in a separate thread, thus all N complex multiplication is done in O(1) complexity.

Additionally, we used "Fast Complex Multiplication" method to improve complex - complex multiplication performance compared to standard implementation. In standard, complex multiplication has four multiplications and two additions:

$$(a + bi) * (c + di) = (ac - bd) + (bc + ad)i \tag{5.5}$$

The Fast Complex Multiplication approach, which is found by Gauss, reduces four multiplications to three [3], the product $(a + bi) * (c + di)$ is computed as following:

$$k1 = c * (a + b) \tag{5.6}$$

$$k2 = a * (d - c) \tag{5.7}$$

$$k3 = b * (c + d) \tag{5.8}$$

$$Realpart = k1 - k3 \tag{5.9}$$

$$Imaginarypart = k1 + k2 \tag{5.10}$$

The kernel code for vector-vector element wise multiplication by using Fast Complex Multiplication method is implemented as follow:

```
1
2  /*
3   *Multiply complex vector - vector by element wise
4   * vec3 = vec1 * vec2
5   */
6  __global__ void kfnMulVectors
7  (
8    int length,
9    cuDoubleComplex *vec1,
10   cuDoubleComplex *vec2,
11   cuDoubleComplex *vec3,
12 )
13 {
14   //Unique thread ID
15   int index = blockIdx.x * blockDim.x + threadIdx.x;
16
```

```
17    //Only length count threads used, others idle
18    if(index < length)
19    {
20
21      //Fast Complex Multiplication
22      double k1 = vec2[index ].x * (vec1[index ].x + vec1[index ].y);
23      double k2 = vec1[index ].x * (vec2[index ].y + vec2[index ].x);
24      double k3 = vec1[index ].y * (vec2[index ].x + vec2[index ].y);
25
26      vec3[index ].x =  k1 - k3;
27      vec3[index ].y =  k1 + k2;
28    }
29  }
```

GPUs are efficient at implementing vector based operations, because these are data independent operations. As seen from previous code segment each complex vector element is multiplied by a complex vector element from other set. Each vector-vector multiplication is done in parallel.

The other implementation difference of GPUsspe from the CPU implementation is calculation of propagation factor. The propagation calculation formula 3.5 is a serial process in CPUsspe, however this can be parallelized in the CUDA environment by using GPU facilities.

The propagation factor computation is O(n^2), for each step in range, N_x, N operations are done.

Pseudo code for propagation factor calculation is as follows:

```
1
2   /*
3    * propFactConstMatrix, initially computed vector, dependent to
4    * range index
5    * uMatrix, is two-dimentional matrix in where single field
6    * calculations are kept
7    * iNx, indexes of ranges
8    * iNz, indexes of altitutes
```

```
 */

for each range Δx
  for each height Δz
    uMatrix[iNx][iNz] = 20  star log10(uMatrix[iNx][iNz]) +
        propFactMatrix[iNx];
  endfor
endfor
```

The GPU based implementation of propagation factor has O(N) complexity, that is the calculation is done for each step in O(1) complexity as follows.

```
/*
 * Calculate PROPAGATION FACTOR
 * source, current single field vector
 * dest, two dimentional total matrix
 * propFactConstMatrix, initially computed vector, dependent to
 * range index
 * N, number of altitute steps
 * Nx, number of steps in range
 */

__global__ void kfnCalcPropagation
(
  int N,
  int Nx ,
  cuDoubleComplex *source,
  double *dest,
  int startRangeIndex,
  double *propFactConstMatrix,
  int deptIndex
)
{
  double tmp;
  int index = blockIdx.x * blockDim.x + threadIdx.x;

  if(index < N)
```

```
26    {
27      tmp = sqrt( source[deptIndex * N + index].x * source[deptIndex *
            N + index].x + source[deptIndex * N + index].y * source[
          deptIndex * N + index].y);
28      tmp = 20.0 * log10( tmp);
29      tmp =  tmp + propFactConstMatrix[startRangeIndex];
30      dest[deptIndex * N * Nx + startRangeIndex * N + index] = tmp;
31    }
32  }
```

Kernel functions are called with number of blocks and thread count for each block. These parameters are highly important for performance, and need to be used carefully. We have test the applications with different block, and thread counts to get the best combination. The parallelism is mostly done over vectors, so the important point in GPUsspe is to allocate more thread than the vector sizes to get full performance. Then, we have decided to run performance tests on 32 block, and 512 thread for each block.

The 2D GPUsspe can run on all of the graphics cards, Quadro 6000, Tesla C2075, and Tesla K20, there is no restriction for it.

SSPE is a data dependent algorithm, that is each single field is calculated according to the previous field. Therefore each single field calculation is done sequentially in both CPU and GPU architectures. This sequential process affects the SSPE performance inefficiently in the GPU architecture. If each field calculation does not depend on the previous field, these fields can be calculated at the same time in parallel in graphics processing units. Thus, we can obtain highly remarkable performance speed ups.

5.4.4 A Single Field Calculation in 3D GPUsspe

3D modeling of SSPE is computationally very intensive. Therefore, the 3D environment is divided into 2D slices, and each 2D slice is processed independent of each other in 3D GPUsspe, and the out-of-plane scattering and diffraction effects associated with lateral variations in the realistic terrain are ignored [18].

From this perspective, 3D GPUsspe is applicable for the GPU architecture, each 2D slice can run in parallel simultaneously. However, 3D GPUsspe and 2D GPUsspe applications are implemented separately, which means there is no code reuse except common library between these applications. The main reason for this inefficiency is cuFFT library of CUDA. cuFFT only allows to call FFT methods from the host code, however we need to call FFT inside kernel functions for 3D GPUsspe, therefore 2D GPUsspe code is not directly applicable.

We have implemented Fast Fourier Transform for GPU according to the FFT algorithm that is explained in detail in Chapter 3, and used it instead of cuFFT thanks to the flexibility that Dynamic Parallelism brings.

The kernel code for FFT is implemented as follows:

```
1  /*
2   * FAST FOURIER TRANSFORM (FFT) implementation
3   * inputVec is complex vector input
4   * outputVec holds output of FFT which is also complex vector
5   * length is input vector s size, which should be 2^n
6   * dir is for direction, if it is -1, it applies inverse FFT
7   * depth is for index of 2D slice in 3D environment
8   */
9
10 __global__ void kfnFFT
11 (
12   int length,
13   cuDoubleComplex *inputVec,
14   cuDoubleComplex *outputVec,
15   int colLen,
16   int depth,
17   int dir
18 )
19 {
20    int k = blockIdx.x * blockDim.x + threadIdx.x;
21
22    cuDoubleComplex result;
23    result.x = 0;
```

```
24      result.y = 0;
25
26      cuDoubleComplex tmpExp;
27      tmpExp.x = 0;
28      tmpExp.y = 0;
29
30      cuDoubleComplex tmpMul;
31      tmpMul.x = 0;
32      tmpMul.y = 0;
33
34      cuDoubleComplex tmpAdd;
35      tmpAdd.x = 0;
36      tmpAdd.y = 0;
37
38      FLOATING tmp = 0;
39
40      int halfLength = length/2;
41
42      if(k < len)
43      {
44        tmp = (-2.0 * CUDARTPI * k)/ length;
45        for(int n = 0; n < halfLength; n++)
46        {
47            tmpExp.x = 0;
48            tmpExp.y = tmp * n;
49            tmpExp = complexExp(tmpExp);
50
51            if( k % 2 == 0)
52            {
53              addComplex(&inputVec[2 * colLen * depth + n], &inputVec[2
                    * colLen * depth + n + halfLength], &tmpAdd);
54            }
55            else
56            {
57              subComplex(&inputVec[2 * colLen * depth + n], &inputVec[2
                    * colLen * depth + n + halfLength], &tmpAdd);
58            }
```

```
59
60            mulComplex(&tmpAdd, &tmpExp, &tmpMul);
61            addComplex(&result, &tmpMul, &result);
62          }
63
64          if(dir == 1)
65          {
66            outputVec[2 * colLen * depth + k] = result;
67          }
68          else // inverse FFT
69          {
70            outputVec[2 * colLen * depth + k].x = result.x / length;
71            outputVec[2 * colLen * depth + k].y = result.y / length;
72          }
73      }
74  }
```

The 3D GPUsspe is only runnable for compute capability (CC) 3.0 or more, therefore Quadro 6000 and Tesla C2075, which are CC 2.0, do not support 3D GPUsspe implementation. The only suitable GPU card is Tesla K20 for this implementation.

5.5 Optimization in GPU

In optimization level, we have mostly used Nsight Profiler, and identify the bottlenecks of GPUsspe implementation. NVIDIA profiler tool shows time percentages of each function in the application. Therefore, we worked on the ones that take extensive amount of time. FFT functions are the most time consuming ones among others, but cuFFT performs the best so the studies to improve the FFT performance do not give a noticeable increase in FFT performance. The vector-vector multiplication was another time consuming kernel, so we applied Fast Complex Multiplication approach to increase its performance, in total it contributes to GPUsspe performance remarkably. Additionally, to improve the performance of GPUsspe, some optimizations are done in operation level and we avoid the use of costly operations as much as possible such as mod, floor, if and else case, etc.

CHAPTER 6

RESULTS

6.1 Accuracy of SSPE

The accuracy of a measurement in an output shows how much the result is close to the actual (true) value in real situation. The error rate tolerance in the accuracy can change from domain to domain. Some problems can tolerate high error rate in accuracy, however some of them cannot. The accuracy tolerance for an algorithm is important for the scientific measurements, since most of scientific problems work on very small numbers. So, very small differences on these values can cause huge differences in the output.

The accuracy of the SSPE algorithm is a critical issue, since the algorithm models EM wave propagation for the radar and communication systems. Beside the accuracy, the performance is another important phenomenon, therefore the accuracy of SSPE model is tested on both single and double precision domain.

It is a known fact that the single precision performs better than double precision. Therefore, the SSPE algorithm is implemented in 2D for both "Single Precision" and "Double Precision" domains on both CPU and GPU architectures to test whether single precision implementation meets the required accuracy in results. For this reason we have compared double and single precision implemented propagation factor outputs.

PETOOL is the MATLAB based developed SSPE algorithm [36]. It is implemented in double precision. We take it as a reference for our implementations' accuracy.

Each figure in this section consists of 3 parts, first one shows PETOOL's propagation factor in range and height, second one is the the propagation factor computed by our SSPE implementation in CPU (CPUsspe) or GPU (GPUsspe) respectively, finally the last one draws the difference between the first and second propagation factor in dB.

6.1.1 Accuracy of SSPE with Flat Terrain

6.1.1.1 Dataset for Accuracy Measurements in Flat Terrain

The algorithm is tested in flat terrain, that is there is no terrain effect. Flat terrain is selected since this is just to see the effects of single or double precision in results.

The algorithm is tested in an area by 50 km in range and 300 m in height. The SSPE range step is the user defined value. The number of iterations can change according to user's range resolution parameter. The range resolution parameter is set as 200 meter in this test case, thus the algorithm runs over 250 range steps (That is figures of this section has 250 points in range, each consecutive point meets 200 meters in reality). The height resolution is initialized as 0.3 meter, which is also the FFT step size. There are almost 1000 points in height for each figure. The height resolution is selected so small, because as FFT step size gets larger, the accuracy of SSPE algorithm decreases.

6.1.1.2 Accuracy Test in CPUsspe

Figure 6.1 compares PETOOL and CPUsspe model in double precision domain. The result shows the difference is about 0.1 dB, which is not so significant. It is an expected result, since both CPUsspe and PETOOL use same CPU architecture. The little difference is result of programming languages of MATLAB and C++.

As seen near the initial ranges, some points show small differences, which are due to initial field calculation. Initial field starts with very small values, something like that e^{-321}, so very little changes in these values causes some differences. However, this is just 0.1 - 0.2 dB which does not cause a problem in propagation results. We cannot draw the same conclusions for Figure 6.2, which displays the difference in accuracy between PETOOL double precision implementation and CPUsspe single precision

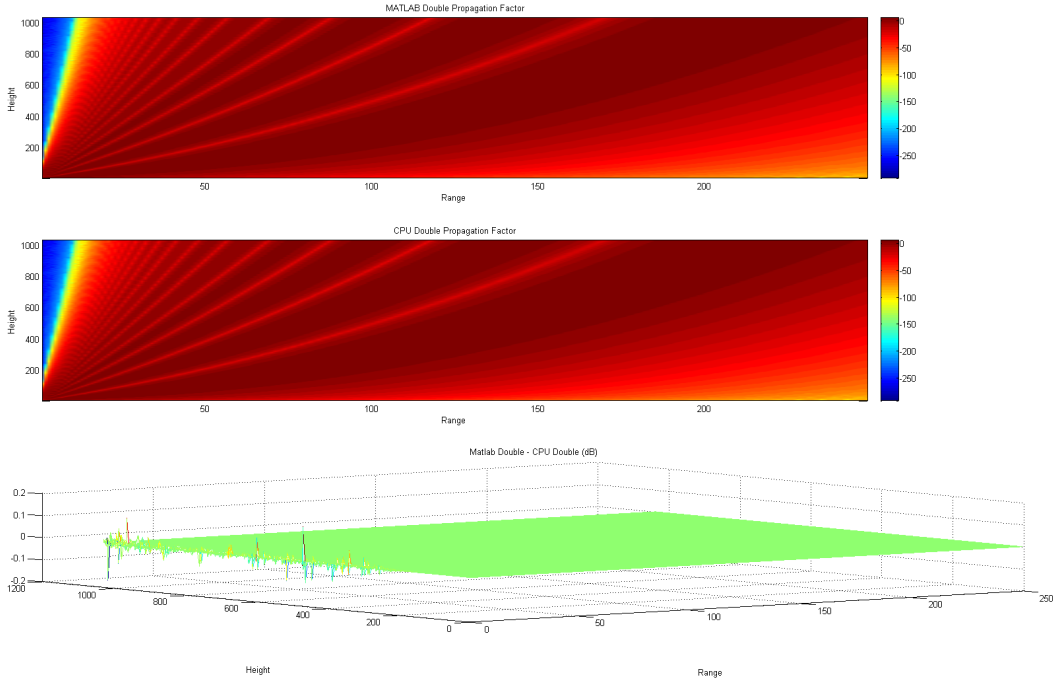

Figure 6.1: PETOOL vs CPUsspe Propagation Comparison on Double Precision

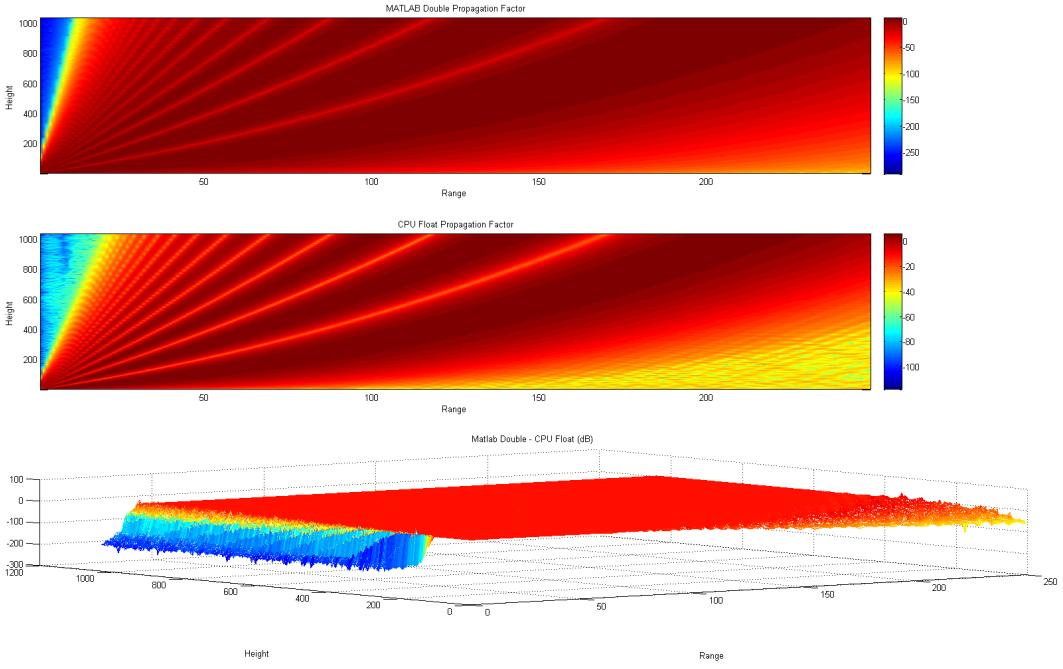

Figure 6.2: PETOOL vs CPUsspe Propagation Comparison on Single Precision

implementation. There are differences up to 200 dB that means error rate percentage is up to $10^{20}\%$ which cannot be ignored. From this error percentage, it is clear that single precision in CPU architecture does not work for the SSPE model. Most of the error occurs at the beginning ranges as a result of initial field calculation. In single precision, half of the values in initial field vector are computed as zero(0), because the precision of float is not enough for these values which are calculated in double precision accurately.

6.1.1.3 Accuracy Test in GPUsspe

Previous GPUs before the Fermi architecture do not support double precision in their hardware. For instance NVIDIA GeForce 8 Series GPUs do not have double precision floating unit in their stream processor. Therefore, the double precision is handled by software support which cause high performance loss. However, modern GPUs are now very good at double precision computation, because they start to support double precision in hardware. In the Fermi architecture, there are 2 double precision FPUs (Floating Point Unit) per SM (Streaming Multiprocessor) in where each SM has 16 cores which means 1/8 performance, i.e., double precision computation has 1/8 performance of single precision in the Fermi architecture.

The scientific Tesla cards of Fermi architecture have 8 double precision FPUs per SM instead of 2 FPUs, so they are better than other Fermi architecture graphic cards, and they provide 1/2 of single precision floating point performance for double precision. Tesla C2075 is a scientific Tesla card based on Fermi architecture, so its double precision performance is just half of single precision performance.

Quadro 6000 is from Fermi architecture, but it is not a Tesla based scientific card. However, it is also as good as Tesla C2075 on double precision accuracy, because it has fast double precision capability to ensure the accuracy and fidelity of results. While the single precision of Quadro 6000 is 1030.4 Gigaflops, the double precision is 515.2 Gigaflops, that is 1/2 of single precision performance also.

Tesla K20 has 64 double precision FPUs for each SMX, which has 192 FPUs in total, therefore it has 1/3 (64/192) of floating point performance for double precision.

As a result, all graphic cards that we used in the thesis have double precision support by hardware. There are some differences in their double precision performance compared to single precision one, however it is not a comparison criteria for performance while comparing these three graphic cards with each other, it just shows single precision and double precision performance rate inside themselves.

Additionally, all three graphics cards have ECC memory, this memory finds and corrects internal data corruptions. This is critical for circumstances where data corruption can not be ignored such as in scientific and financial problems. However, it is not so important for graphics cards, which are designed for visual processes and gaming purposes, because error in a few pixel is not so critical.

6.1.1.3.1 Quadro 6000

Following Figure 6.3 compares PETOOL and GPUsspe model that is implemented in double precision. The implementation runs on Quadro 6000 graphic card which has fast double precision support. The figure represents propagation difference is up to 0.5 dB. That means there are some points in which the PETOOL propagation factor ratio to GPUsspe in Quadro 6000 is about 1.12.

The accuracy difference between PETOOL and GPUsspe (Quadro 6000) is higher than PETOOL and CPUsspe, because while PETOOL and CPUsspe run on same CPU architecture, GPUsspe run on a separate graphic card. However, the difference is also not a problem up to 1 dB, that is Quadro 6000 is a sufficient graphic card for the SSPE algorithm on the accuracy.

Figure 6.4 presents the accuracy rate between PETOOL and single precision implemented GPUsspe. The graphic illustrates the results are not so different from the single precision result of CPUsspe. The error rate is also about 200 dB in initial ranges. Therefore we can conclude from the figure, Quadro 6000 single precision is not adequate also.

In initial ranges, while error is about 200 dB, the error rate decreases in later steps. The reason of this situation although single precision is not sufficient for initial field vector's values which has high precision values, the later field's values can be shown

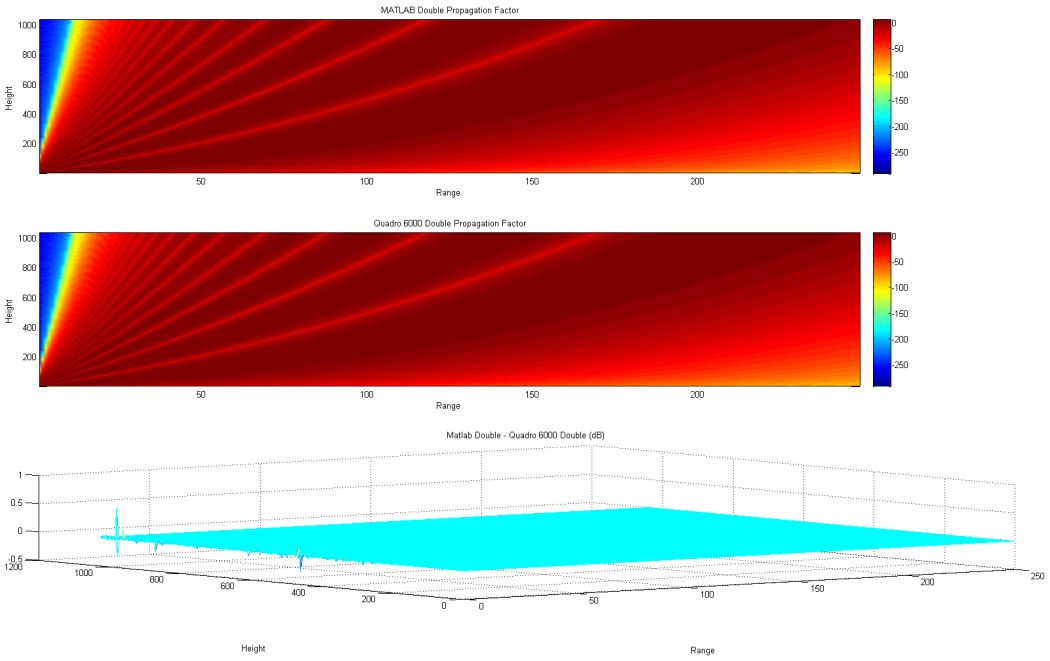

Figure 6.3: PETOOL vs GPUsspe (Quadro 6000) Propagation Comparison on Double Precision

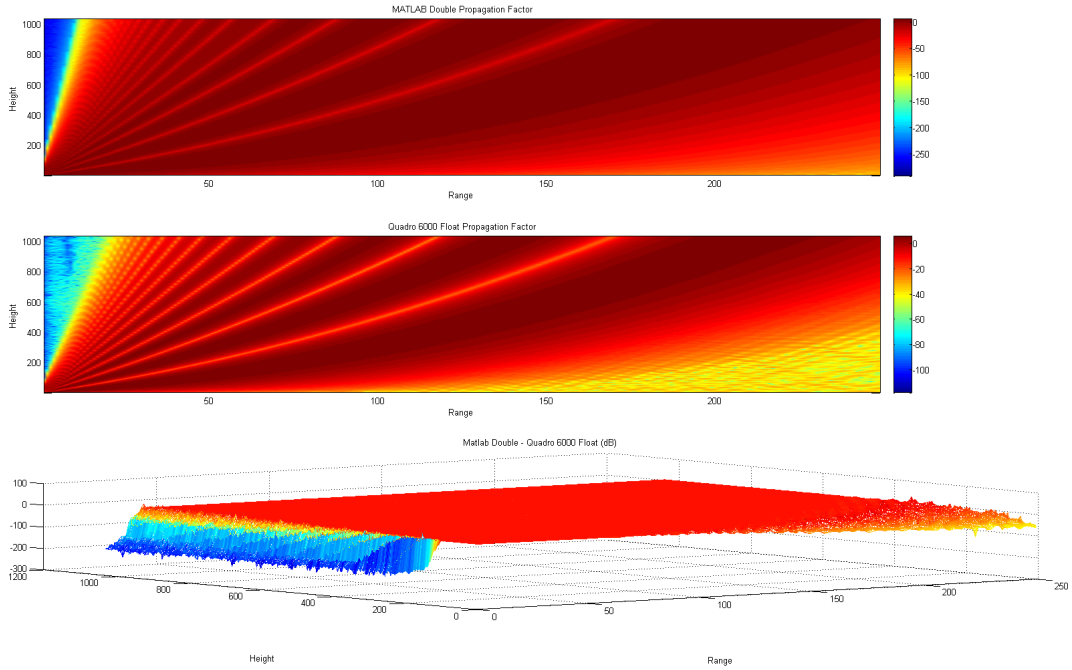

Figure 6.4: PETOOL vs GPUsspe (Quadro 6000) Propagation Comparison on Single Precision

more accurately in single precision. The error rate decreases in following ranges, however it does not disappear, and it ends with an error of 40 dB.

6.1.1.3.2 Tesla C2075

The same test case is run over Tesla C2075 which is a scientific graphic card. The results shown in Figure 6.5 and 6.6 are not so different from Quadro 6000 results.

In double precision there is error near to 0.5 dB, however in single precision this error rate increases up to 200 dB. The Quadro 6000 and Tesla C2075 are products of the same GPU architecture, but they are designed for different purposes. However, they are almost same at accuracy as seen from the results.

The accuracy of single precision represents Tesla C2075 is not applicable for the SSPE model in single precision. However, it is not true about Tesla C2075 or other graphic cards (Quadro 6000 and Tesla K20), the reason for that is the single precision accuracy does not meet the SSPE model requirements in any CPU or GPU.

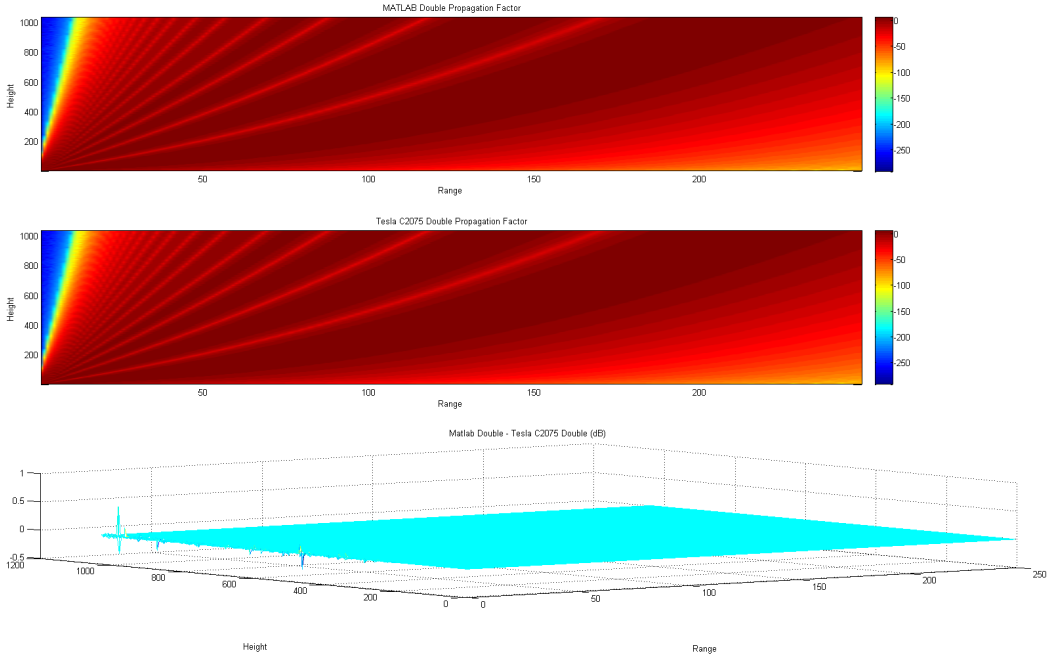

Figure 6.5: PETOOL vs GPUsspe (Tesla C2075) Propagation Comparison on Double Precision

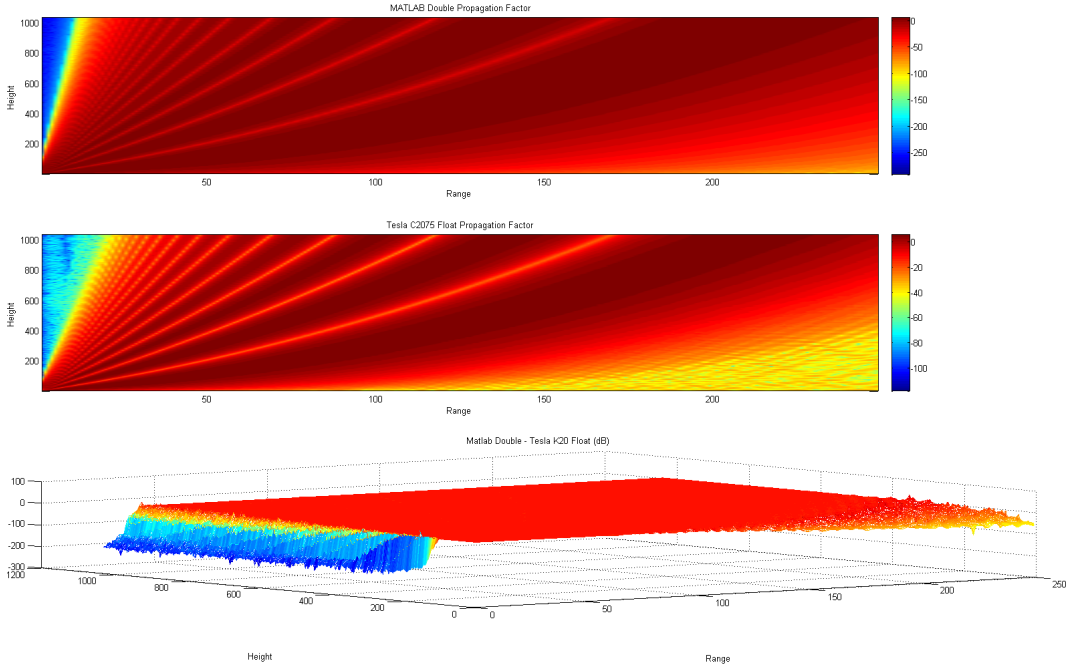

Figure 6.6: PETOOL vs GPUsspe (Tesla C2075) Propagation Comparison on Single Precision

6.1.1.3.3 Tesla K20

Figure 6.7 and Figure 6.8 display the comparison results in Kepler architecture.

After seeing the results of Tesla K20 in double precision and single precision, it can be easily concluded, these three graphic cards have same accuracy in SSPE problem.

Additionally, when we compare graphic cards with CPU, there are small differences between the double precision accuracy of CPU and GPU cards, however in the single precision they are almost same. Modern graphic cards, which have double precision support, solved accuracy problem. They are more reliable, which is a major step for their evolution.

6.1.1.4 Horizontal and Vertical Polarization Comparison in SSPE

Split Step Parabolic Wave Equation algorithm works on two different linear polarization, namely horizontal and vertical. The computation of a single field shows

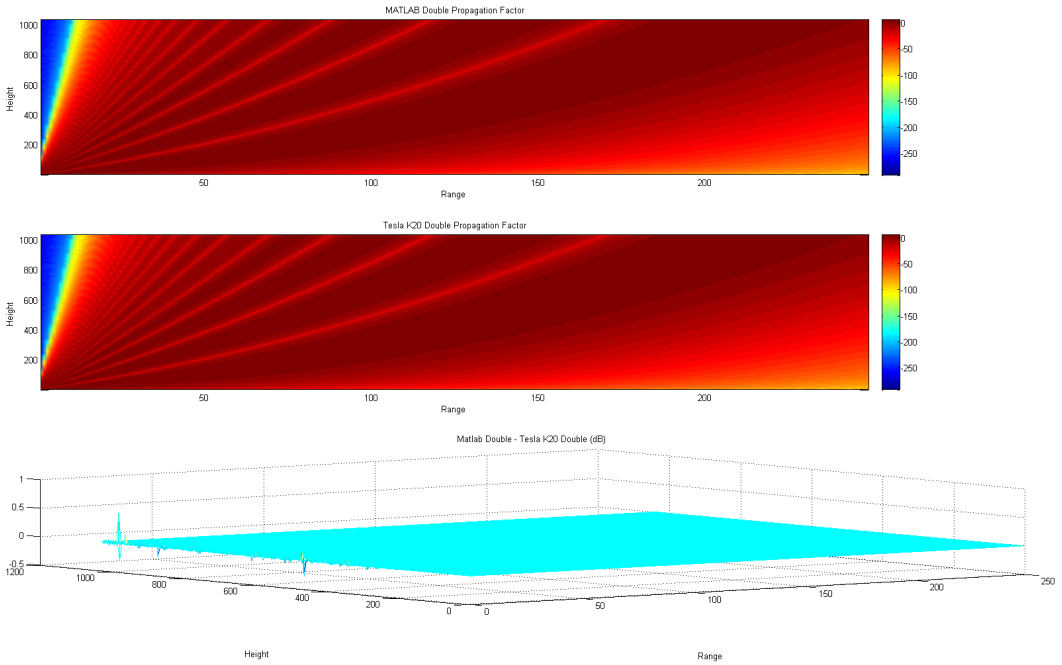

Figure 6.7: PETOOL vs GPUsspe (Tesla K20) Propagation Comparison on Double Precision

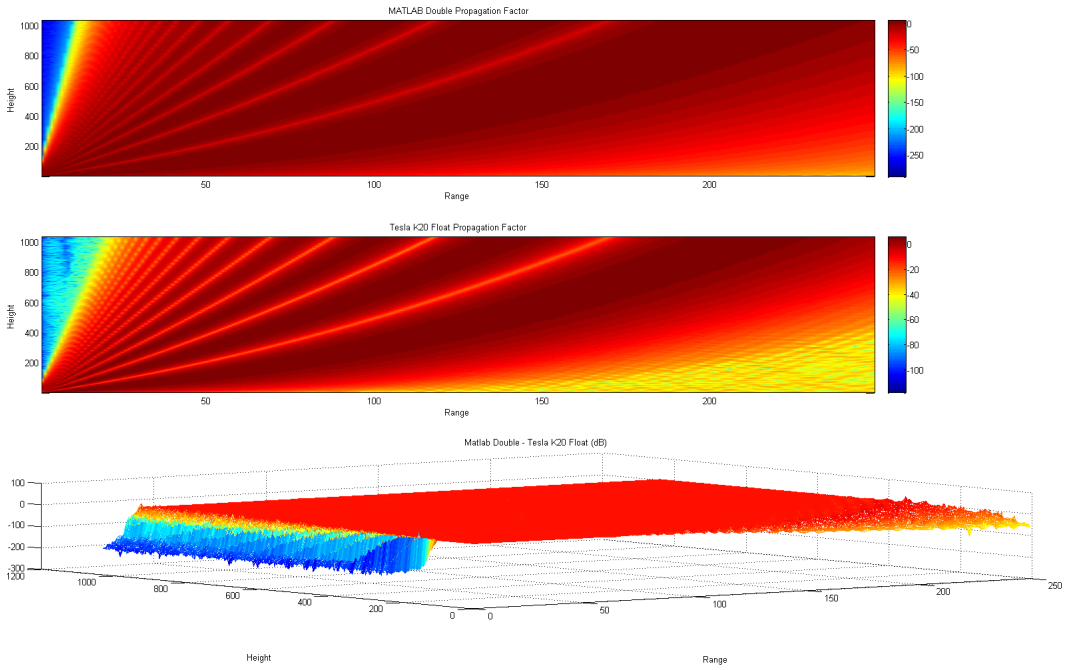

Figure 6.8: PETOOL vs GPUsspe (Tesla K20) Propagation Comparison on Single Precision

differences according to polarization type. If antenna polarization is set as horizontal polarization, Discrete Sine Transform (DST) and its inverse IDST are executed sequentially for each single field computation, otherwise in vertical polarization Discrete Cosine Transform (DCT) and its inverse IDCT are run.

Discrete cosine transform and its inverse IDCT are computationally more costly than DST and its inverse IDST. As seen in following pseudo code segments, to compute DCT, extra multiplications are done.

In this section, we compare polarization effects on propagation factor. If the difference between them is negligible, we use horizontal polarization, since DST and IDST take shorter in timing compared to DCT and IDCT.

Pseudo code for DST and IDST algorithms:

```
1
2  /*
3   *Discrete Sine Transform
4   *(DST)
5   */
6
7  dst (inputVector, N)
8  1. Create a 2N size array tmpVector
9  2. Put inputVector into first N portion of tmpVector
10 3. Reverse inputVector element wise
11 4. Multiply the reversed vector by -1
12 5. Put the result vector in the second N portion of tmpVector
13 6. Apply FFT on tmpVector
14 7. Take first N part of tmpVector, and divide each element into 2i
      (complex value)
15 8. Return result vector
16
17
18 /*
19  *Inverse Discrete Sine Transform
20  *(IDST)
21  */
22
```

```
idst (inputVector, N)
1. dst(inputVector, N)
2. Multiply each element of dst result with 2/(N+1)
3. Return result vector
```

Pseudo code for DCT and IDCT algorithms:

```

//isReal method checks whether all imaginary parts in inputVector
    are 0

/*
 *Discrete Cosine Transform
 *(DCT)
 */

dct (inputVector, N)
1. Create a 2N size array tmpVector
2. if( N2 == 1 or !isReal (inputVector))
  2.1. Put inputVector into first N portion of tmpVector
  2.2. Reverse inputVector element wise
  2.3. Put it in the second N portion of tmpVector
  2.4. Apply FFT on 2N sized tmpVector
  2.5. Take first N part and multiply it with weightVector by
      element wise
  2.6. Retun the result vector
3. Else
  3.1. Put inputVector by passing one element in each step into
      first N/2 portion of tmpVector
  3.2. Reverse inputVector element wise
  3.3. Put it into second N/2 portion of tmpVector by passing one
      element in each step
  3.4. Apply FFT on N sized portion of tmpVector.
  3.5. Multiply the FFT result by 2.
  3.6. Multiply it with weightVector by element wise
  3.7. Retun the result vector

```

```
/*
 *Inverse Discrete Cosine Transform
 *(IDCT)
 */

idct (inputVector, N)
1. Multiply inputVector with weightVector by element wise.
2. Put the result first N porition of 2N sized tmpVector.
3. Reverse inputVector element wise
4. Multiply it with (weightVector * -i)
5. Put the result vector to the second N portion of tmpVector
6. Apply FFT on 2N sized tmpVector
7. Return first N part of tmpVector
```

Weight vector is computed for each index in input vector as follows:

$$ww[ii] = \frac{\exp(\frac{-j \times ii \times \pi}{2N})}{\sqrt{2N}} \tag{6.1}$$

The antenna polarization has a significant role in propagation result. Horizontally polarized antenna receives horizontally polarized EM waves, and neglects vertical ones, otherwise vertically polarized antenna takes vertically polarized signals, and neglects others. Therefore, the propagation results of horizontally and vertically polarized antenna should be different from each other, but the critical part is the amount of difference.

In Figure 6.9, we can see that there is about from 50 to 150 dB difference between them, which is a significant number. As a result, both vertical and horizontal polarization results are measured in performance tests.

6.1.2 Accuracy of SSPE with Non-Flat Terrain

From previous test results, it is obvious that single precision is not enough for SSPE accuracy, therefore the remaining tests are done on double precision. Also, as CPUsspe has almost same accuracy with PETOOL, the comparisons will be performed by tak-

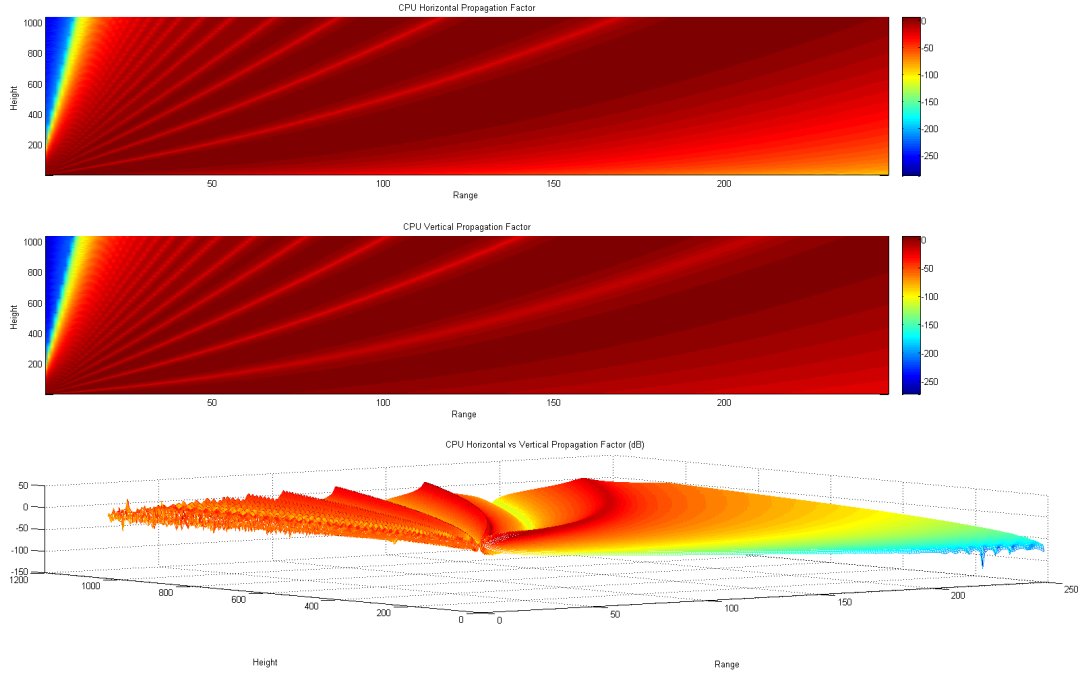

Figure 6.9: Horizontal vs Vertical Polarization in SSPE

ing CPUsspe as reference, not PETOOL anymore, because PETOOL has some limitation in range and height.

For the following tests, the test area has same range and height properties with the previous accuracy test cases, there is a terrain (mountain) additionally. Thus, the effects of terrain on propagation process can be experienced from the following test cases.

Figure 6.10 compares CPUsspe and GPUsspe accuracy in existence of terrain with Quadro 6000. The propagation factor decreases behind the terrain, because non-flat terrain prevents EM waves to pass through.

From the comparison of propagation between CPUsspe and GPUsspe, they are almost same in most of places, there are some difference about 1 dB, however it is very rare, so it does not cause a problem.

The coverage area of propagation test is increased, and algorithm is tested in an area by 100 km in range and 1500 m in height. The height resolution is kept as same with previous test as 0.3 meter. The range resolution is decreased to 100 meter, which was

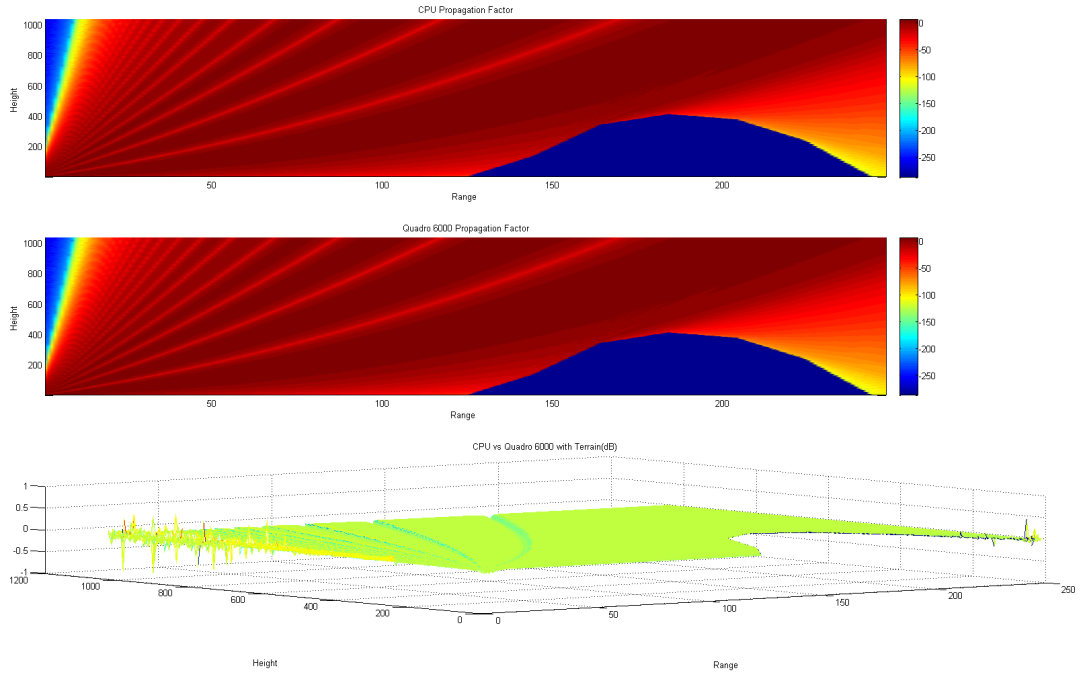

Figure 6.10: CPUsspe vs GPUsspe (Quadro 6000) with Terrain (50 km in rage - 300 m in height)

200 meter previously. Decreasing range step gives more accurate test results, since number of iterations for same range increase, and SSPE field calculations are done more frequently.

Figure 6.11 compares CPUsspe and GPUsspe accuracy in existence of terrain with Quadro 6000 in a wider area. The error is about 10 dB which is a critic, however it seems the error mostly in early ranges. To be sure, we compare the difference between the CPUsspe and GPUsspe after 5 km. By removing first 5km from the test case, the result is demonstrated in Figure 6.12. The propagation factor difference decreases sharply to 5×10^{-4}, which is negligible.

The reason for the high difference at beginning ranges is not because of non-flat terrain, since same case is applied with flat terrain as in Figure 6.13. The error ratio is also about 10 dB, however when removing first 5 km from propagation results, the error rate is also very small as shown in Figure 6.14.

The reason for that error at initial 5 km is the difference at precision of sqrt() function

in CPU and GPU architectures. GPU is not good at taking square root of very small floating point values. Additionally, in this test case, FFT size is increased, and as initial field values are so small, the accuracy deviates at initial ranges much more than other ranges.

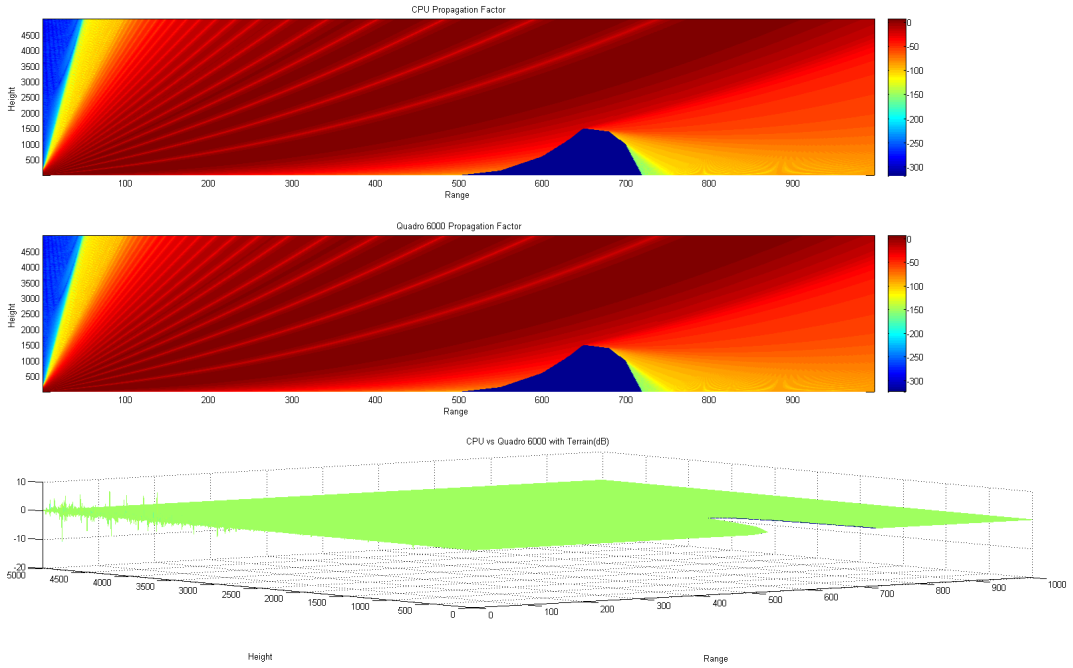

Figure 6.11: CPUsspe vs GPUsspe (Quadro 6000) with Terrain (100 km in rage - 1500 m in height)

The same tests are run on other graphic cards Tesla K20 and Tesla C2075, the results are same with Quadro 6000 in all test cases. It means all three graphic cards behave in a same manner in accuracy in flat and non-flat terrains, so terrain does not change the propagation results in accuracy.

6.2 Performance of SSPE

6.2.1 Test Cases

There are two main factors that affect the SSPE performance, these are range step count and FFT step count.

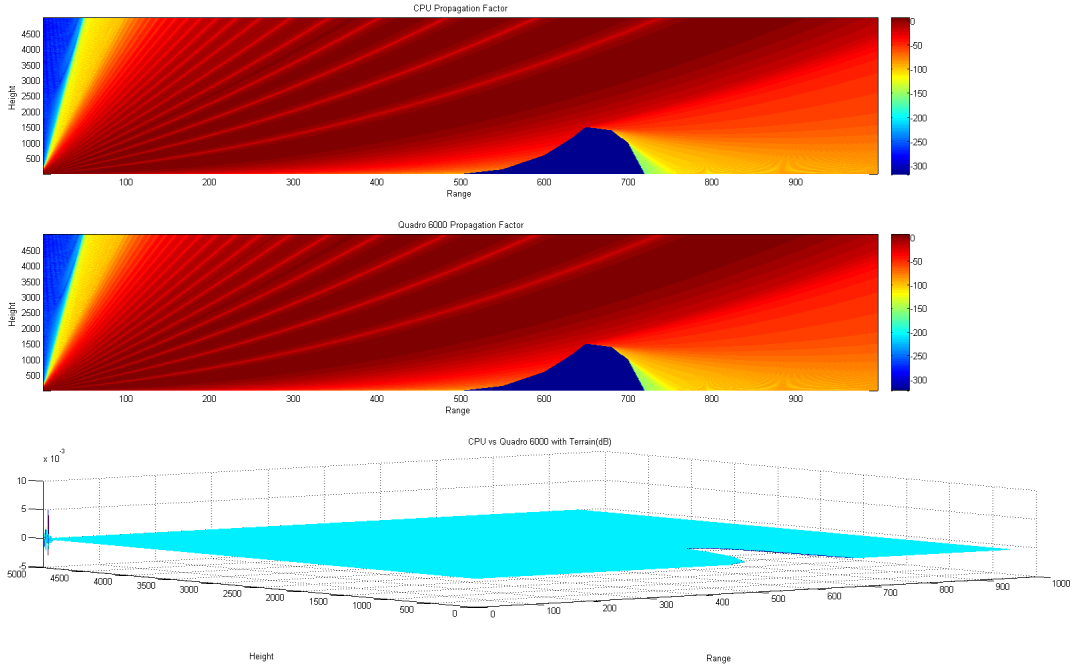

Figure 6.12: CPUsspe vs GPUsspe (Quadro 6000) with Terrain (100 km in rage - 1500 m in height) after 5 km

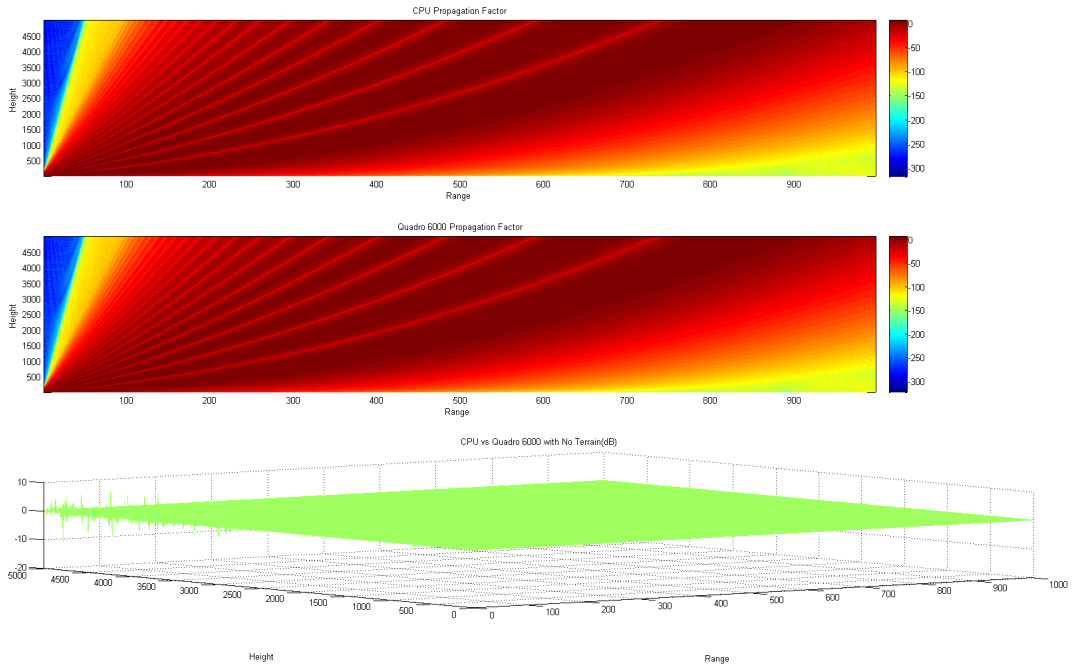

Figure 6.13: CPUsspe vs GPUsspe (Quadro 6000) with Flat Terrain (100 km in rage - 1500 m in height)

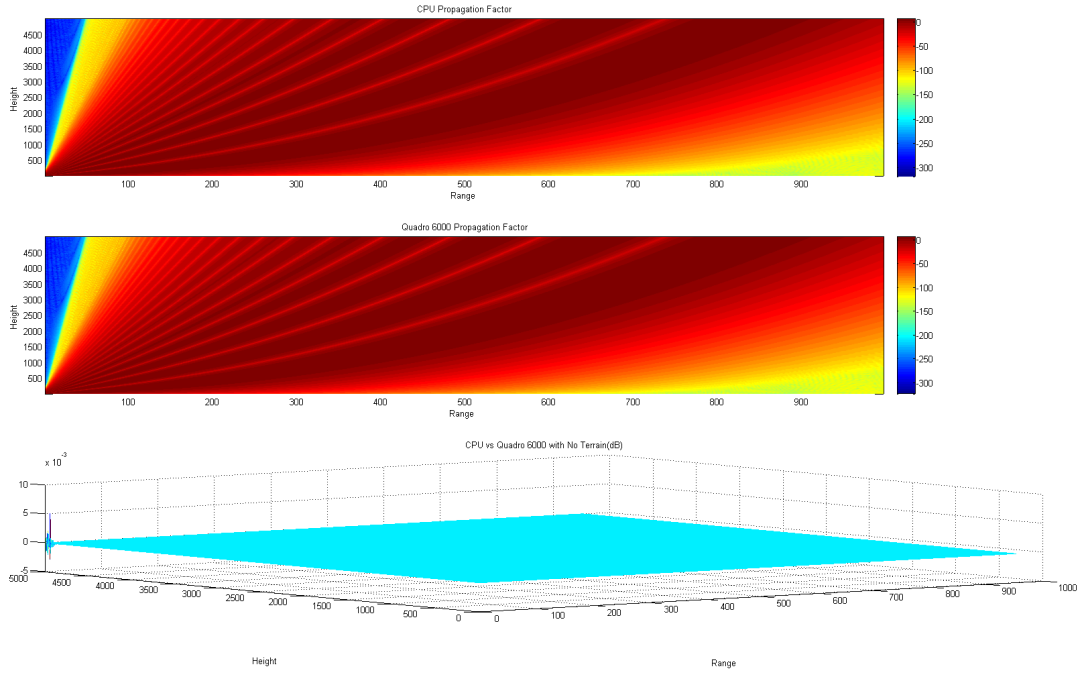

Figure 6.14: CPUsspe vs GPUsspe (Quadro 6000) with Flat Terrain (100 km in rage - 1500 m in height) after 5 km

We have created 8 test cases. Test cases 1, 2, and 3 are created to test FFT step count effect on the performance of CPUsspe and GPUsspe. Therefore, the range size and range step size are fixed in these test cases, and FFT size increases in each test case. Test cases 4, 5, and 6 aim to see the effect of range step count in the performance of CPUsspe and GPUsspe (where FFT size is fixed). The FFT size selected as 2^n. Test 7, and 8 display the performance differences in wider coverage area (both in range and FFT step count).

All test cases run 100 times and average of these 100 samples are taken both in CPU and GPU architectures. The standard deviation is also calculated over these 100 samples, CPU has higher standard deviation rate than GPUs, which is a well known problem for CPUs. GPUs are highly deterministic architectures, the GPU architecture supports the quick interchange of data in the grid.

6.2.1.1 Test 1

- Maximum range is 50 km, where range step size 200 m. There are 250 steps to iterate in the SSPE computation.
- Maximum height is 300 m, where altitude step size is 0.3 m. There are 1552 steps in height, however discrete sine and cosine transforms take FFT of about 3100 complex points (two times height step count).

6.2.1.2 Test 2

- Maximum range is 50 km, where range step size 200 m. There are 250 steps to iterate in the SSPE computation.
- Maximum height is 2048 m, where altitude step size is 1.5 m. There are 2048 steps in height, and 4096 is the FFT size.

6.2.1.3 Test 3

- Maximum range is 50 km, where range step size 200 m. There are 250 steps to iterate in the SSPE computation.
- Maximum height is 1000 m, where altitude step size is 0.2 m. There are 7500 steps in height, and 15000 is the FFT size.

6.2.1.4 Test 4

- Maximum range is 50 km, where range step size 200 m. There are 250 steps to iterate in SSPE computation.
- Maximum height is 2048 m, where altitude step size is 1.5 m. There are 2048 steps in height, and 4096 is the FFT size.

6.2.1.5 Test 5

- Maximum range is 100 km, where range step size 100 m. There are 1000 steps to iterate in the SSPE computation.
- Maximum height is 2048 m, where altitude step size is 1.5 m. There are 2048 steps in height, and 4096 is the FFT size.

6.2.1.6 Test 6

- Maximum range is 300 km, where range step size 100 m. There are 3000 steps to iterate in SSPE computation.
- Maximum height is 2048 m, where altitude step size is 1.5 m. There are 2048 steps in height, and 4096 is the FFT size.

6.2.1.7 Test 7

- Maximum range is 300 km, where range step size 100 m. There are 3000 steps to iterate in SSPE computation.
- Maximum height is 4096 m, where altitude step size is 1.5 m. There are 4096 steps in height, and 8194 is the FFT size.

6.2.1.8 Test 8

- Maximum range is 300 km, where range step size 100 m. There are 3000 steps to iterate in the SSPE computation.
- Max height is 1000 m, where altitude step size is 0.2 m. There are 7500 steps in height, and 15000 is the FFT size.

Except from test cases which are numbered from 1 to 8, tests are run under different polarization, and terrain effects which are grouped under 4 different categories, for each category all test cases (1..8) are repeated. "Horizontal Flat Terrain" means test

is run under horizontally polarized antenna and there is no elevation in the terrain (all elevation values are zero(0)). In horizontal polarization, DST and IDST are used in propagation calculations. "Vertical Flat Terrain" is for vertically polarized antenna, in where DCT and IDCT are used for propagation computations. There is also no elevation in test terrains. "Horizontal Non-flat Terrain" specifies test case of horizontally polarized antenna, and terrain with elevation values. "Vertical Non-flat Terrain" category uses vertically polarized antenna and a map with elevation values.

6.2.2 Performance Results in 2D

Table 6.1 displays the effect of FFT size on the SSPE algorithm in CPU and GPU architectures. As FFT size gets bigger, application times both in CPU and GPU increase in most of the cases, however there can be some exceptions when FFT size is power of 2, because as FFT is O(nlogn) algorithm, it can perform more efficiently in 2^n cases. Additionally, increase ratio in CPU is greater than graphic cards, because cuFFT performs better than Intel MKL in FFT performance.

In Fixed Range Tests all graphics cards performances are compared with CPU performance, and their speed ups respect to CPU are given in speed up columns. Quadro 6000 does not show a good quality of performance. For both categories of horizontal and vertical test, it performs maximum 2.8 speed up compared with CPUsspe. Also, in some situations as displayed in red color, Quadro 6000's performance is worse than CPUsspe. GPUsspe performance in Tesla K20 has a speed up between 4.2 and 4.8 for test case 3. In test 1 and 2, Tesla K20 does not show remarkable speed ups also. From these findings, we can say Tesla K20 performs better in big FFT sizes. Tesla C2075 executes up to 5 times faster than CPUsspe, and its performance not so different from Tesla K20.

Performance results of fixed FFT size are shown in Table 6.2. The table presents the effect of FFT size on SSPE performance in CPU and GPU architectures. Range sizes are 50 km (250 steps), 100 km (1000 steps), 300 km (3000 steps) for test 4, 5, and 6 respectively. The FFT size is set to 4096 for these tests. The step size count affects the performance of SSPE seriously. Quadro 6000 runs almost at the same performance with the CPUsspe in these test cases, and does not show any valuable result. On the

Table 6.1: CPUsspe vs GPUsspe Fixed Range in 2D

	Test Case	CPU (ms)	Quadro 6000 (ms)	Speed Up	Tesla K20 (ms)	Speed Up	Tesla C2075 (ms)	Speed Up
Horizontal Flat Terrain	Test 1	67.6	90.4721	0.747191676	65.6127	1.030288344	37.7313	1.79161598
	Test 2	65.22	69.9859	0.931901997	45.5427	1.432062658	32.9704	1.97813797
	Test 3	507	272.539	1.860284216	121.2735	4.180633032	110.733	4.57858091
Vertical Flat Terrain	Test 1	197.14	96.1204	2.050969409	70.6447	2.790584432	47.0523	4.18980581
	Test 2	186.22	192.7791	0.965976084	74.7983	2.489628775	76.8396	2.42348997
	Test 3	927.43	470.237	1.972260796	208.0385	4.457972923	241.6561	3.83780918
Horizontal Non-flat Terrain	Test 1	66.4	91.4858	0.725795697	66.4772	0.9988387	42.3774	1.56687291
	Test 2	63.19	72.382	0.873007101	44.7786	1.411165155	33.1175	1.90805465
	Test 3	516.8	273.9475	1.88649285	121.9743	4.236958113	113.3206	4.56051239
Vertical Non-flat Terrain	Test 1	233.91	97.827	2.391057683	70.8191	3.302922517	46.9127	4.98606987
	Test 2	194.57	193.8542	1.003692466	75.9502	2.561810239	75.5154	2.57656054
	Test 3	998.38	352.7625	2.83017611	206.5414	4.833800875	242.3456	4.11965392

Table 6.2: CPUsspe vs GPUsspe Fixed FFT Size in 2D

	Test Case	CPU (ms)	Quadro 6000 (ms)	Speed Up	Tesla K20 (ms)	Speed Up	Tesla C2075 (ms)	Speed Up
Horizontal Flat Terrain	Test 4	65.22	69.9859	0.931901997	45.5427	1.432062658	32.9709	1.978107968
	Test 5	278.89	426.7782	0.653477614	179.5986	1.552851748	130.1427	2.142955387
	Test 6	1880.36	816.097	2.304088852	520.3324	3.613766892	363.3644	5.174860278
Vertical Flat Terrain	Test 4	186.22	192.7791	0.965976084	74.7983	2.489628775	76.8396	2.423489971
	Test 5	746.07	762.749	0.978133042	300.5542	2.482314338	295.5321	2.524497339
	Test 6	3645.46	1548.1929	2.354654901	866.1549	4.208785288	870.1546	4.18943944
Horizontal Non-flat Terrain	Test 4	63.19	72.382	0.873007101	44.7786	1.411165155	33.1175	1.908054654
	Test 5	276.19	283.6458	0.9737144	182.1884	1.515958206	121.4629	2.273863048
	Test 6	1941.62	819.1732	2.370219143	518.2415	3.746554454	352.259	5.511910271
Vertical Non-flat Terrain	Test 4	194.57	193.8542	1.003692466	75.9502	2.561810239	75.5154	2.576560543
	Test 5	745.26	761.7813	0.978312279	301.6453	2.470650131	303.8719	2.452546616
	Test 6	4589.64	2210.1114	2.076655503	869.6726	5.27743429	868.2732	5.285939955

other hand, Tesla K20 and Tesla C2075 become more attractive, they performs up to 5.3 - 5.5 times faster than CPU. As the iteration count increases, GPUs show better results thanks to their more deterministic and quick architectures in iteration cycles rather than CPU nondeterministic architecture.

Table 6.3: CPUsspe vs GPUsspe Test 7 and 8 in 2D

	Test Case	CPU	Tesla K20 (ms)	Speed Up	Tesla C2075 (ms)	Speed Up
Horizontal Flat Terrain	Test 7	5847	566	10.33038869	396	14.76515152
	Test 8	9687	703	13.77951636	596	16.2533557
Vertical Flat Terrain	Test 7	7615	991	7.684157417	1058	7.197542533
	Test 8	12835	1197	10.72263993	1389	9.240460763
Horizontal Non-flat Terrain	Test 7	5672	561	10.11051693	433	13.09930716
	Test 8	9780	704	13.89204545	586	16.6894198
Vertical Non-flat Terrain	Test 7	9328	995	9.374874372	1047	8.909264565
	Test 8	13124	1198	10.95492487	1441	9.107564192

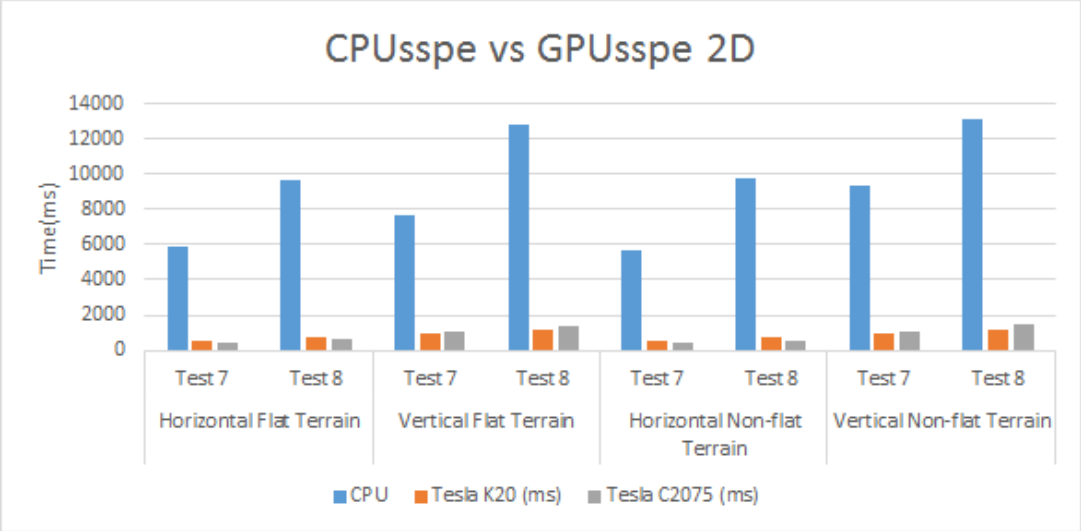

Figure 6.15: CPUsspe vs GPUsspe Test 7 and 8 in 2D

Test 7 and 8 in Figure 6.15, compare CPUsspe and GPUsspe on wider coverage area in range and altitude, the results are impressive, the Tesla K20 performs up to 13-14 times, and Tesla C2075 has 14-15 times speed ups compared to CPUsspe. These test results show that when FFT size increases GPUsspe performs significantly better than CPUsspe.

In overall, the performance of CPUsspe and GPUsspe are tested on eight distinct test cases among four different categories in 2D. There are not so much difference

when we compare horizontal flat and non-flat categories, or vertical flat and non-flat categories. The results are so close to each other, because non-flat terrain does not bring so much work to the algorithm's execution. However, the big difference appears when we compare horizontal and vertical categories, horizontal test results are most efficient in timing than vertical ones, because DST is a faster analysis method than DCT. Other than analysis methods, FFT vector size affects total performance remarkable, when FFT vector size gets larger, GPU performs better. As GPUs are more deterministic architectures, and they support quick interchange of data in the grid, SSPE running time takes less in GPU than CPU.

From performance test results, Quadro 6000 does not perform efficiently compared to Tesla based scientific cards. Tesla C2075, and Tesla K20 have high performance CUDA drivers, namely Tesla Computer Cluster (TCC) driver, different from Quadro 6000. These drivers reduces kernel launch overheads. Otherwise, if kernel overhead takes more than the actual task processing time, this process performs better in CPU than GPU. Therefore, for computationally intensive problems Tesla graphic cards are more suitable.

6.2.3 Performance Results in 3D

Number of slices is an important point in the 3D GPUsspe performance. Test 1, 2, and 3 are the performance results of 10, 50, 100 2D slices respectively as shown in Table 6.4.

Other test environment issues are as follow:
* Maximum range is 50 km, where range step size 100 m. Therefore, there are 500 steps to iterate in the SSPE computation.
* Maximum height is 2048 m, where altitude step size is 1.5 m. There are 2048 steps in height, and 4096 is the FFT size.

The average time for a single 2D slice in 2D GPUsspe under same test conditions is about 50-70 ms. Therefore the time for 10 2D slices is about 5-7 seconds in a serial process of 2D slices, however in 3D GPUsspe it is about 99 seconds, which is not an efficient process. In CPUsspe, a single 2D slice is executed in 70-190 ms, in where

	Test Case	Tesla K20 (s)
Horizontal	Test 1 (10 2D Slice)	99
	Test 2 (50 2D Slice)	459
	Test 3 (100 2D Slice)	891
Vertical	Test 1 (10 2D Slice)	98
	Test 2 (50 2D Slice)	457
	Test 3 (100 2D Slice)	894

Table 6.4: GPUsspe (Tesla K20) Performance Results in 3D

10 2D slices take 7-19 seconds, which is also not comparable with 3D GPUsspe. Therefore, 3D environment can be modeled by using 2D GPUsspe in a serial process much more efficiently than 3D GPUsspe.

As an initial impression, 3D GPUsspe implementation seems preferable for the GPU architecture, since 3D environment is divided into 2D slices, and each 2D slice can be processed independent from each other in parallel grids. However, GPU cores are specialized for small amount of tasks, and processing a single 2D slice requires high amount of computation. That is, each 2D slice SSPE computation requires a sequential process of each single fields, and each single field takes FFT on large vectors. FFT is a highly parallel method and appropriate for GPU grid architecture when enough amount of GPU resources are allocated for its computation. Otherwise, it takes remarkable running time. In 3D GPUsspe we divide the GPU resources for each 2D slice, and FFT resources are decreased compared to 2D GPUsspe. FFT running time is an important criteria for SSPE performance; however, FFT performance in Dynamic Parallelism takes significantly much more time than cuFFT, since Dynamic parallelism shares GPU resources to kernels, then these resources become insufficient to take FFT efficiently. Therefore, 3D GPUsspe does not give expected performance improvement in test results, so it does not worth to execute in the GPU architecture.

CHAPTER 7

CONCLUSION

7.1 Conclusion

Split Step Parabolic Equation (SSPE) method is capable of simulating one way, forward scattering propagation efficiently, however it is a computationally intensive algorithm, so takes remarkable time while analyzing larger areas. In this thesis, we have developed an implementation for the SSPE model in the GPU architecture for both two dimensional and three dimensional environments. In terms of efficiency, scientific based GPUs are highly suited for the task of computing electromagnetic wave propagation. We have tested CUDA based SSPE implementations with three different graphic cards in 2D environments. Our test results indicate that, GPUsspe performs 10-12 times faster than the reference CPU implementation on scientific Tesla based cards, namely Tesla K20, and Tesla C2075. However, Quadro 6000 has even worse performance than scientific cards. 3D environment modeling as a combination of 2D slices in the GPU architecture does not achieve a performance improvement. The main reason behind this is that when GPU resources are divided between kernels, FFT takes much more time compared with CUDA FFT library, cuFFT.

To sum up, our thesis introduces an efficient 2D GPUsspe application to model EM wave propagation, however the results of our studies on 3D GPUsspe indicates that using 2D GPUsspe in a serial process for modeling 3D environment can be more efficient than 3D GPUsspe.

7.2 Future Work

3D GPUsspe is not worthy enough in performance to apply it in the GPU architecture, and also it ignores out-of-plane scattering effects. Therefore, another propagation algorithms, which are applicable to the GPU grid architecture, can be developed to model 3D environments both in efficient and more accurate way.

Additionally, GPU technology continues to evolve and CUDA platform provides new features in each new versions. Therefore, if CUDA environment provides cuFFT functions to be called inside a kernel, the 3D GPUsspe implementation with appropriate changes can be tested again to obtain the new performance.

REFERENCES

[1] CUDA Zone, NVIDIA. https://developer.nvidia.com/cuda-zone, last visited on August 2014.

[2] Direct Compute. https://developer.nvidia.com/directcompute, last visited on August 2014.

[3] Fast Complex Multiplication. http://en.wikipedia.org/wiki/Multiplication, last visited on June 2014.

[4] GPU Applications. http://www.nvidia.com/object/gpu-applications.html, last visited on June 2014.

[5] Graphics Processing Unit (GPU), NVIDIA. http://www.nvidia.com/object/gpu.html, last visited on August 2014.

[6] NVIDIA Nsight. http://www.nvidia.com/object/nsight.html, last visited on June 2014.

[7] OPENCL Zone. http://developer.amd.com/tools-and-sdks/opencl-zone/, last visited on August 2014.

[8] Rational Rhapsody Family. http://www-03.ibm.com/software/products/en/ratirhapfami, last visited on August 2014.

[9] What is GPU Accelerated Computing? http://www.nvidia.com/object/what-is-gpu-computing.html, last visited on June 2014.

[10] Why Choose a TESLA GPU? http://www.nvidia.co.uk/object/why-choose-tesla-uk.html, last visited on August 2014.

[11] T. E. Athanaileas, G. E. Athanasiadou, G. V. Tsoulos, and D. I. Kaklamani. Parallel radio-wave propagation modeling with image-based ray tracing techniques. *Parallel Computing*, 36(12):679–695, Dec 2010.

[12] A. E. Barrios. A terrain parabolic equation model for propagation in the troposphere. *Antennas and Propagation, IEEE Transactions on*, 42(1):90–98, Jan 1994.

[13] W. K. Burns. Normal mode analysis of waveguide devices. i. theory. *Lightwave Technology, Journal of*, 6(6):1051–1057, Jun 1988.

[14] K. Despain. Fast Fourier Transforms and Graphics Processing Unit. http://www.umiacs.umd.edu, last visited on August 2014.

[15] G. D. Dockery and J. R. Kuttler. An improved impedance-boundary algorithm for fourier split-step solutions of the parabolic wave equation. *Antennas and Propagation, IEEE Transactions on*, 44(12):1592–1599, Dec 1996.

[16] D. J. Donohue and J. R. Kuttler. Modeling radar propagation over terrain. *Johns Hopkins Apl Technical Digest*, 18(2), 1997.

[17] P. Duhamel and M. Vetterli. Fast fourier transforms: A tutorial review and a state of the art. *Signal Process.*, 19(4):259–299, Apr 1990.

[18] J. Z. Gehman, J. R. Kuttler, and M. H. Newkirk. Modeling Radar Propagation in Three-Dimensional Environments. *Johns Hopkins Apl Technical Digest*, 25(2), 2004.

[19] P. N. Glaskowsky. NVIDIA's Fermi: The First Complete GPU Computing Architecture . http://sbel.wisc.edu/Courses/ME964/Literature/whitePaperFermiGlaskowsky.pdf, last visited on July 2014.

[20] S. H. Gunderson. GPUwave: An implementation of the split-step Fourier method for the GPU . http://gpuwave.sesse.net/, last visited on August 2014, 2007.

[21] P. Hursky and M. B. Porter. Accelerating underwater acoustic propagation modeling using general purpose graphic processing units. In *OCEANS 2011*, pages 1–6, Sept 2011.

[22] Intel. Math Kernel Library. http://developer.intel.com/software/products/mkl/, last visited on August 2014.

[23] J. R. Kuttler and G. D. Dockery. Theoretical description of the parabolic approximation/fourier split-step method of representing electromagnetic propagation in the troposphere. *Radio Sci.*, 26(2):381–393, 1991.

[24] M. A. Leontovich and V. A. Fock. Solution of the problem of electromagnetic wave propagation along the earth's surface by the method of parabolic equation. *J. Phys. USSR*, 10:13–23, 1946.

[25] C. Levis, J. T. Johnson, and F. L. Teixeira. *Radiowave Propagation: Physics and Applications*. John Wiley & Sons, 2010.

[26] B. Ma, X. Zhang, and Z. Zhang. The modeling of radar electromagnetic propagation by parabolic equation. In Song Lin and Xiong Huang, editors, *Advances in Computer Science, Environment, Ecoinformatics, and Education*, volume 215 of *Communications in Computer and Information Science*, pages 137–149. Springer Berlin Heidelberg, 2011.

[27] H. M. Masoudi, M. A. AlSunaidi, and J. M. Arnold. Time-domain finite-difference beam propagation method. *Photonics Technology Letters, IEEE*, 11(10):1274–1276, Oct 1999.

[28] Y. S. Meng and Y. H. Lee. Measurements and characterizations of air-to-ground channel over sea surface at c-band with low airborne altitudes. *Vehicular Technology, IEEE Transactions on*, 60(4):1943–1948, May 2011.

[29] W. L. Patterson. Advanced refractive effects prediction system (areps). In *Radar Conference, 2007 IEEE*, pages 891–895, Apr 2007.

[30] M. A. Richards. *Fundamentals Of Radar Signal Processing*. McGraw-Hill Education (India) Pvt Limited, 2005.

[31] M. I. Skolnik. *Introduction to Radar Systems /2nd Edition/*. McGraw Hill Book Co., New York, 1980.

[32] S. W. Smith. *The Scientist and Engineer's Guide to Digital Signal Processing*. California Technical Publishing, San Diego, CA, USA, 1997.

[33] I. Will. Electromagnetic Modeling with GPUs. `http://iancwill.com/blog/wp-content/uploads/2010/11/gpu-em-paper.pdf`, last visited on August 2014, 2010.

[34] L. Yang, L. Chengguo, Z. Miao, W. Yuan, and Y. Ming. Radio wave propagation path loss in the irregular terrain environments. In *Microwave, Antenna, Propagation and EMC Technologies for Wireless Communications, 2009 3rd IEEE International Symposium on*, pages 627–630, Oct 2009.

[35] Ö. Özgün. Petool: Matlab-based one-way and two-way split-step parabolic equation tool for radiowave propagation over variable terrain. *Antennas and Propagation, IEEE Transactions on*, 57(9):2706–2714, Sept 2009.

[36] Ö. Özgün, G. Apaydın, M. Kuzuoğlu, and L. Sevgi. Petool: Matlab-based one-way and two-way split-step parabolic equation tool for radiowave propagation over variable terrain. *Computer Physics Communications*, 182(12):2638–2654, Sept 2011.