HØGSKOLEN
I NARVIK

SHO6264

Master of Science in Technology

Efficient customizable tools for big data processing: Incoherent scatter radar big data as a case study

Huy Trieu Thanh

August, 2016

Departement of technology
Narvik University College

HØGSKOLEN
I NARVIK

Title: Efficient customizable tools for big data processing: Incoherent scatter radar big data as a case study

Date: 15.08.2016

Classification: Open

Author: Huy Trieu Thanh

Pages: 80

Attachments: 1 CD

Departement:
Department of Technology
Studieretning:
M-IT
Supervisor:
Phuong Ha Hoai
Hien Bich Vo

Principal:
Norges Arktiske Universitet (UiT)

Principal contact:
Phuong Ha Hoai
Hien Bich Vo

Keywords:
GPGPU, Big Data, Plasma Line, Python, C++, CUDA, cuFFT.

Abstract (English):

This project is my master diploma project of the department of Computer Science at Norks Arktisk Universitet (UiT) in 2016. Every day, the satellites send to the ground stations huge amount of data for processing. Incoherent Scatter Radar (ISR) is a very interesting technology in the space science and it also sends a large amount of data in plasma line data back to the earth. Currently, the space scientists are using MATLAB and Python to process plasma line data which use the CPU to process and consume a large amount of power and time to process this data. In this master thesis, it will focus on the processing data to reduce the time and power consumption.

Departement of technology
Narvik University College

HØGSKOLEN
I NARVIK

Departement of technology
Narvik University College

Abstract

This project is my master diploma project of the department of Computer Science at Norks Arktisk Universitet (UiT) in 2016. Every day, the satellites send to the ground stations huge amount of data for processing. Incoherent Scatter Radar (ISR) is a very interesting technology in the space science and it also sends a large amount of data in plasma line data back to the earth. Currently, the space scientists are using MATLAB and Python to process plasma line data which use the CPU to process and consume a large amount of power and time to process this data. In this master thesis, it will focus on the processing data to reduce the time and power consumption.

Acknowledgements

I appreciate the useful advice from my main supervisor, Professor Phuong Ha Hoai, when I was developing this master project, and his help during the development of this thesis. In addition, I also appreciate my co-supervisor, Professor Hien Bich Vo, about the helping in physics in during the time I worked on this thesis. I also would like to thank you for the help from Professor Michael Sulzer about his help in Python code. Moreover, I would like to thank Prof. Hung Nguyen Thanh for his help.

I would like to thank my family for all the support they have given to me during the time I am studying in Norway.

Finally, I would like to thank my lab mates for cooperation and creating a very comfortable environment for study.

Norsk Artikes Universitet, August 15, 2016

Huy Trieu Thanh

Contents

List of Figures

List of Tables

Preface

Every day, Incoherent Scatter Radars (ISR) receives a huge amount of information from the universe. The data will collect by the Incoherent Scatter Radars (ISR) which located at Arecibo, USA, and EISCAT in Northern Scandinavia and some ground stations around the world. The main problem is to process the giant amount of data efficiently. Currently, they are using the Python and MATLAB to process data by several algorithms which can smooth and find the peaks of the plasma line data.

The main focus of my thesis is to investigate and develop an efficient tool for processing a huge plasma line data by using the GPGPU technology. In the physical science, the peaks represent for the scientist the signals of valuable data. The problem with the plasma lines is that there are a lot of noise data which show the fake peaks. However, the scientist wants to have a clear data to show the real peaks of data. There are several ways to clean the data and algorithms to solve the problem, but because of the big amount of data so it takes a lot of time to process and a huge amount of power to process those data. In my thesis, we choose the Match Filter algorithm to filter the data. In addition, we also implement the example algorithms by using Python code. We have received 1.8 TB data from Arecibo, USA station as example data to process.

For the development, I have chosen to use Qt's framework and Qt Creator to implement C++ coding. In addition, we also used Python code for implementing the example algorithms. Moreover, I have chosen the CUDA (Compute Unified Device Architecture) which developed by NVIDIA for implementing and processing data.

1. Introduction

1.1. *Some theories*

1.1.1. Incoherent Scatter Radars (ISR)

First, we should understand about what is RADAR (RAdio Detection And Ranging), this is the technique using to detect the presence of objects in the atmosphere. Radar was born shortly before World War II. From primary purpose, RADAR just used for detecting the presence of aircraft. Nowadays, Radar uses in a wide application in the life but, mainly using for detecting precipitation and other meteorological events.
[1]

Incoherent Scattering is a type of scattering phenomenon in physics. It is mostly to use when referring to scattering of an electromagnetic wave by random fluctuation in a gas of particles. Incoherent Scatter Radar (ISR) is the most well-known application of the Incoherent Scattering. ISR developed for studying the Earth ionosphere first proposed by Professor Bill Gordon in 1958. By this technology, the radar beam scatters off electrons in the ionospheric plasma, which created by Incoherent Scatter return. The Incoherent Scatter signal allows measurement of electron density, which relates to ion temperature, electron temperatures, ion composition and plasma velocity.
[2]

Years	Developer
1958-1959	Bill Gordon conceives of the idea to Incoherent Scatter and construction begin at Arecibo, Puerto Rico, with the money supported from the Defense Advanced Research Project Office (DARPA).
1961	The Jicamarca Observatory constructed at Lima, Peru, by the National Bureau of Standards.
1962	Construction of Arecibo is completed.
1963	The Millstone Hill zenith antenna constructed by MIT Lincoln Laboratories at a site near Boston, MA.
1971	The Chatanika Radar moved from Stanford University to a new place near Fairbanks, Alaska.
1982	The Chatanika Radar moved to a new location near Sondrestrom, Greenland.
1970 – 1980	The United States National Science Foundation takes over operation of four incoherent scatter radars located at: Sondrestrom, Millstone Hill, Jicamarca and Arecibo.
1981	The European began the construction of the EISCAT UHF system at Tromsø, Norway.
1985	The EISCAT VHF system begins operation at Tromsø.
1996	The EISCAT Svalbard Radar was built near Longyearbyen, Norway.

Table 1: The history of development of Incoherent Scatter Radars

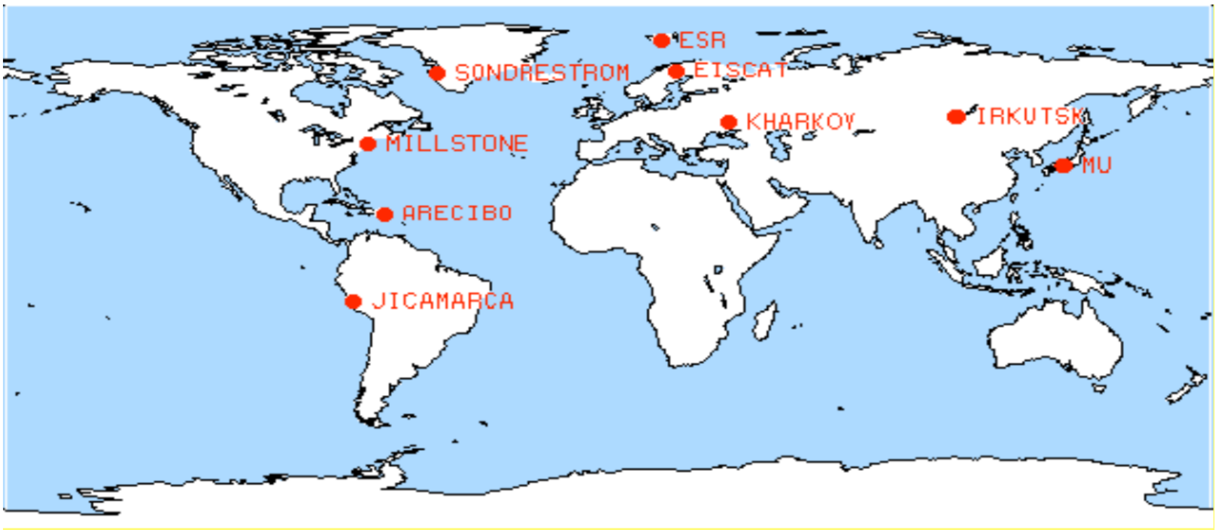


Figure 1: The global array of Incoherent Scatter Radars (2007).

[3]

1.1.2. The signal

"According to the Merrian – Webster dictionary, «a signal is a sources of information generally a physical quantity which varies with respect to time, space, temperature like any independent variable»".[4]

There are two kinds of signals. The wave signal which is present by the functions:
Y = S + Noise.
Where Noise is the random and S is the power.
Another kind of signal is the digital signal which is represented by the function:

$$y = \begin{cases} A + noise \\ -A + noise \end{cases}$$

Where A is the bit (0 or 1),
In this kind of noise, it must be following the Gaussian distribution
In the plasma line which we received by the Incoherent Scatter Radars, we have received the digital type. Mostly, we should be clear around 90% to 100% of noise by the algorithms.

1.1.3. Plasma

Plasma is one of the four fundamental of states of matter, the others being solid, liquid, and gas. In the properties, plasma have their properties different than other states. To define the plasma, we have three criteria to define:

- The plasma approximation is the criteria defined by charging particles must be close enough together that each particle influences many nearly charged particles, rather than just interacts with the closet particle. The plasma approximation is valid when the number of charged carries within the sphere of influence of a particular particle is higher than unity for provides the collective behavior of the charged particles.

- Bulk interactions: it is the criteria in which the interactions in the bulk of the plasma are more important than those at each eagle, where boundary effects may take place. The plasma is quasineutral when this criteria satisfied.
- Plasma frequency is satisfied, electrostatic interactions dominate over the process of ordinary gas kinetics. The plasma frequency is large compared to the electron – neutral collision frequency.

Plasma has also had three properties which define by the ranges of parameters, the degree of ionization and temperature.

In the real life, plasma is usually found in the most abundant form of ordinary matter in the universe, lighting and heating gas or subjecting it to a strong electromagnetic field applied with a laser or microwave generator.
[5]

1.1.4. Application of plasma line

There are a several potential uses of plasma line. The system constrains for the Incoherent Scatter Radar can be determined with good accuracy for using plasma line frequency measurement. In addition, the ion line analysis can use to determine the electron temperature independently. In combination of the ion line and plasma lines of the Incoherent Scatter Radar data, it is possible to resolve the temperature/ composition ambiguity in the ion line autocorrelation function and it also can make the high time resolution electron temperature estimates. The measurement of the plasma line strength in restricted frequency intervals can be used to estimate the super thermal electron flux in different energy ranges.
[6]

1.2. Tools which are using in the project

1.2.1. Qt

Qt is a cross-platform application framework which is widen used for development software application which can run in various software and hardware platform. Qt trademark and copyright belong to Digia Company. This cross platform is available both commercial and open source, GPL v3, LGPL v3 and LGPL v2 licenses. There are four available editions of Qt, they are a community, inside mobile, professional and enterprise. Qt supported for a wide range of platform such as on Android, Linux, iOS, Windows, BlackBerry and some other platforms.
[7]

1.2.2. Cmake

CMake is a cross-platform free and open-source software for managing the build progress of software using the complier-independent method. It has minimal dependencies, requiring only a C++ compiler on its own build system. In the original, ITK (Insight Segmentation and Registration Toolkit) funded by NLM as a part of the Visible Human Project where need a powerful cross – platform.
[8]
[9]

1.2.3. Git

Git is a free, open source distributed version control system designed to handle everything from small to large project. Git distributed under terms of the GNU General Public License v2. Git development began in April 2005 when many developers cannot access BitKeeper, a source control management that is previously used to maintain the project. In design, Git's design was inspired by BitKeeper and Monotone. Git was designed as a low-level version control system engine on top of which others could write front ends.
[10], [11]

1.2.4. CUDA

CUDA is a parallel computing platform and programing model which was invented by NVIDIA. By using CUDA, it enables dynamic increases in computing performance by using the power of the graphics processing unit (GPU). There are several uses of the GPU with CUDA, for example, identify hidden plaque in arteries, analyze the air traffic flow, visualize molecules, and more fields in the scientific which can apply CUDA in the real life.
[12]

1.2.5. HDF5

Hierarchical Data Format (HDF) is a set of file formats which designed to store and organize a huge amount of data. It is originally developed by the National Center for Supercomputing Applications, the HDF Group supported it, a non – profit corporation. The mission of HDF Group is to ensure that the Hierarchical Data Format continued development of HDF5 technology and continued accessibility of data stored in HDF.

To keep this goal, HDF Group keeps their libraries and associated tools are available under a liberal, BDS – like license for general use. This library also supported for a wide range of commercial and non-commercial software platforms, including Java, MATLAB, Scilab, Octave, IDL, Mathematica, Python, Jula, and R. The current version of this library is HDF5.
[13]

1.2.6. Python

The programming language, which accepted for the developer using in high-level, interpreted, and general – purpose and dynamic programming, is Python. Nowadays, it is widely used. This language design to allow programmers to express their code in a fewer line than in possible language such as C++, Java. This programming language also supported for both large scale and small scale application.

Python began implementation in December 1989 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI) in the Nertherlands. He is Python's principal author. On the history of Python, the version 2.0, Python 2.0 was released on 16 October 2000. On 3 December 2008, the version 3.0 was released after a long testing period.
[14]

1.2.7. cuFFT

The NVIDIA CUDA Fast Fourier Transform library (cuFFT) is the library developed by NVIDIA. It provides a simple interface for computing FFTs up to 10x faster. cuFFT use hundreds of processor cores inside NVIDIA GPUs, cuFFT can deliver the floating-point performance of GPU without having to develop your own custom GPU FFT implementation.

Nowadays, there is a widely used in applications which using the computational physics for processing image and general signal processing, the Fast Fourier Transform is an efficient algorithm for computing Discrete Fourier transforms of complex or real-valued datasets. [15]

1.3. Objective

The thesis is to create the tool which can process a huge amount of data quickly to save time and power consumption. Due to workload and time-consuming tasks, the project is therefore limited in a wide range of option to run, for example, not running in muitl-node clusters nor using the Apache Spark to solve the problem.

This idea is quite new when using GPGPU to solve this problem. Conventionally, space physicists solve the problem by running MATLAB and Python programs on the CPU. In this project, I just use the one node GPGPU to solve those problems based on Python code from Arecibo.

1.4. Chapter summary

This chapter will describe about the summary of the chapters in this thesis.

Chapter 1. Introduction
This chapter describes some theories which related to the thesis, the objective of the thesis and the introduction about the tools which used to solve the thesis problems.

Chapter 2. Ideas
Introduction about the current problems. For those problems, we have the ideas to solve it.

Chapter 3. Background
Information about the state of the art overview.

Chapter 4. Previous work
Introduction about some previous work.

Chapter 5. Architecture
Information about the architecture which is used in the system to solve the problem.

Chapter 6. Design
Information about the design details of the system to solve the problem.

Chapter 7. Results
Introduction about the results which got from applying the state of the art.

Chapter 8. Discussion
Information about the comparison of the results which got from the thesis and previous work. In addition, this chapter also contained the limitation and future work of this thesis.

Chapter 10. Conclusion
Conclusion for the thesis considering proposal.

2. Ideas

2.1. Overview

This section will show the recent problem of the space sciences. Based on this problem, we discuss about our ideas to solve. Moreover, this part also describes about the method which we plan to use to solve the problem.

2.2. Recently problem

Plasma line data is received every day and it is a huge amount of data when received it. As normally, the data which received from ISR (Incoherent Scatter Radars) always have noise depends on the weather condition and other conditions. It is a not large problem for the storage data space in computer, but the big problem is that how the space scientist can recognize the real data and noise data in real time. Currently, the space scientist just received and store it but they are processing those data later because due to the computer speech it cannot solve the huge data in real time. There are a lot of algorithms to solve this problem, however, the current code just writing in MATLAB and Python, which process low speech and huge amount of power consumption.

2.3. Method to solve this problem

Based on the above problem, running the system which can solve the real time analyze data to clear be a very convenient for the scientist to recognize quickly and make the data which received from Incoherent Scatter Radars (ISR) is valuable. Moreover, it also makes the scientist have a quickly report and have the decision faster than in the past.

To give the scientist this convenience, we have an idea is that solve this problem by using CUDA to apply those filter algorithms to clean data and given the quickly results in real time. As we know, the GPGPU is primary using for graphics processing but in recent year, it is applied in many fields of computer science and applications. It is a very new and interesting idea.

There are several algorithms to apply to find the peaks, which give for the scientist the clear data. However, we got the kindly help from the Arecibo station from the USA to give to us the data and also the Python Code. Finally, we decide to choose the plan is translate the code from Arecibo to GPGPU code.

3. Background

3.1. Overview

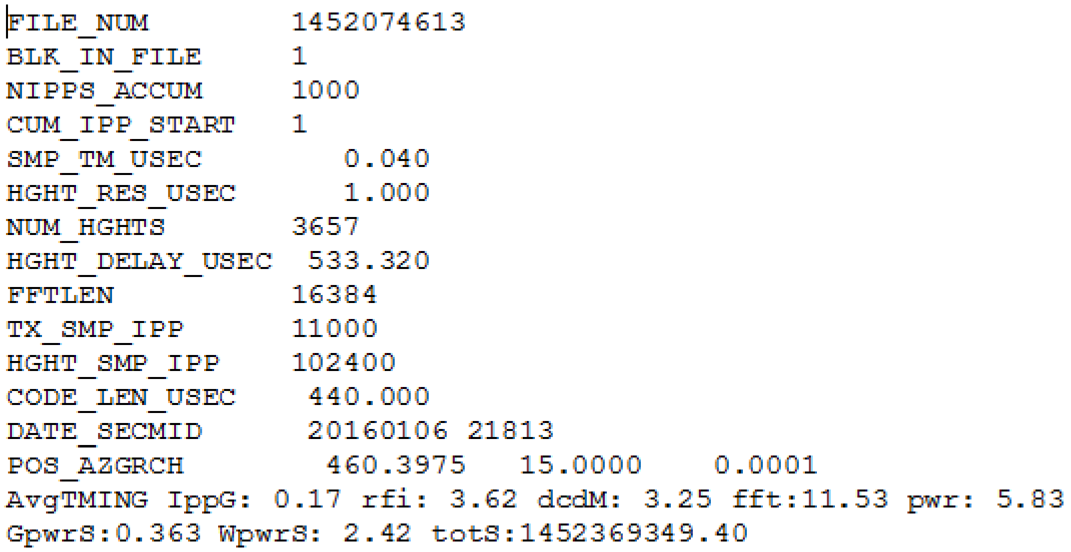
This part will represent the information about the algorithms and the background code which will using in the thesis to solve the problem.

3.2. Understanding about the data structure

The data, which received by Arecibo station is every 10 seconds and it occupied around 1.8 TB. The data will come by the package pairs which have the extension .dcd and .hdr.

The file which have extension .dcd include the real data, which received by ISR (Incoherent Scatter Radar) and have the cover for all height. By inspection about this file, this have only the number in the whole file. Moreover, this file includes the data in all height at the current time.

The file which has the extension .hdr included the metadata for the .dcd file. They have the same structure for all packages and the structure look like the figure below:



```
FILE_NUM         1452074613
BLK_IN_FILE      1
NIPPS_ACCUM      1000
CUM_IPP_START    1
SMP_TM_USEC        0.040
HGHT_RES_USEC      1.000
NUM_HGHTS        3657
HGHT_DELAY_USEC   533.320
FFTLEN           16384
TX_SMP_IPP       11000
HGHT_SMP_IPP     102400
CODE_LEN_USEC     440.000
DATE_SECMID       20160106 21813
POS_AZGRCH          460.3975   15.0000    0.0001
AvgTMING IppG: 0.17 rfi: 3.62 dcdM: 3.25 fft:11.53 pwr: 5.83
GpwrS:0.363 WpwrS: 2.42 totS:1452369349.40
```

Figure 2: The structure of header file for data package number 1452074613.

In this file, it contained some information's. However, we just focus to use in some information which we will use in the thesis. The FILE_NUM show the number of package. NUM_HGHTS show the height number of data, in the example picture above, it show the 3657 heights of data which contained in the package 1452074613. FFTLEN show the frequency of data which received from the ISR (Incoherent Scatter Radar). DATE_SECMID included the information about the date and second of the data package.

There are the function to give back to the date and the time from header file. For example, in the above data, the first column is 20160106 present the package received from the date 06 January 2016. The second column contained the information 21813 shows the time, which package received. To decode this number, we call the second column is secs then we apply the function:
Hours = secs//3600

Minutes = (secs – Hours*3600)//60
Seconds = (secs – Hours*3600 – Minutes*60)
To apply the second column as example, we can see that.
Hours = 21813//3600 = 6
Minutes = (21813 – 6*3600)//60 = 3
Seconds = (21813 – 6*3600 – 60*3) = 33
For this, it means the package received at 6:03:33 UTC + 0.

For every package, the time just change 10 seconds, for this, we can guess that the package number 1452074613 received at 06 January 2016 at 06:03:43 UTC + 0.

3.3. *Python code from Arecibo*

In this part, it shows about the Python code which received from Arecibo, USA. From this code, it shows about the information of the peaks in one height, the overview of all height and processing the whole data in one second (about 1.8TB).

3.3.1. Finding the peak in all height

This section will show about the figure of the information when the data is an analysis of the Python code. Below is the black and while data and the color data.

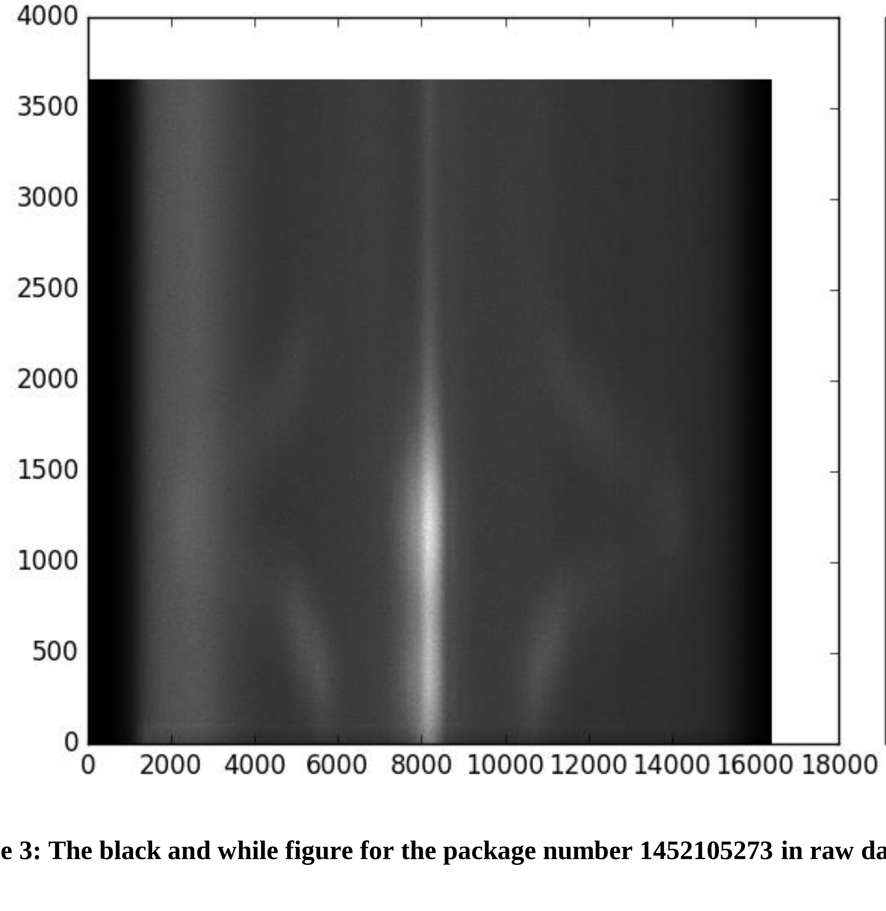


Figure 3: The black and while figure for the package number 1452105273 in raw data.

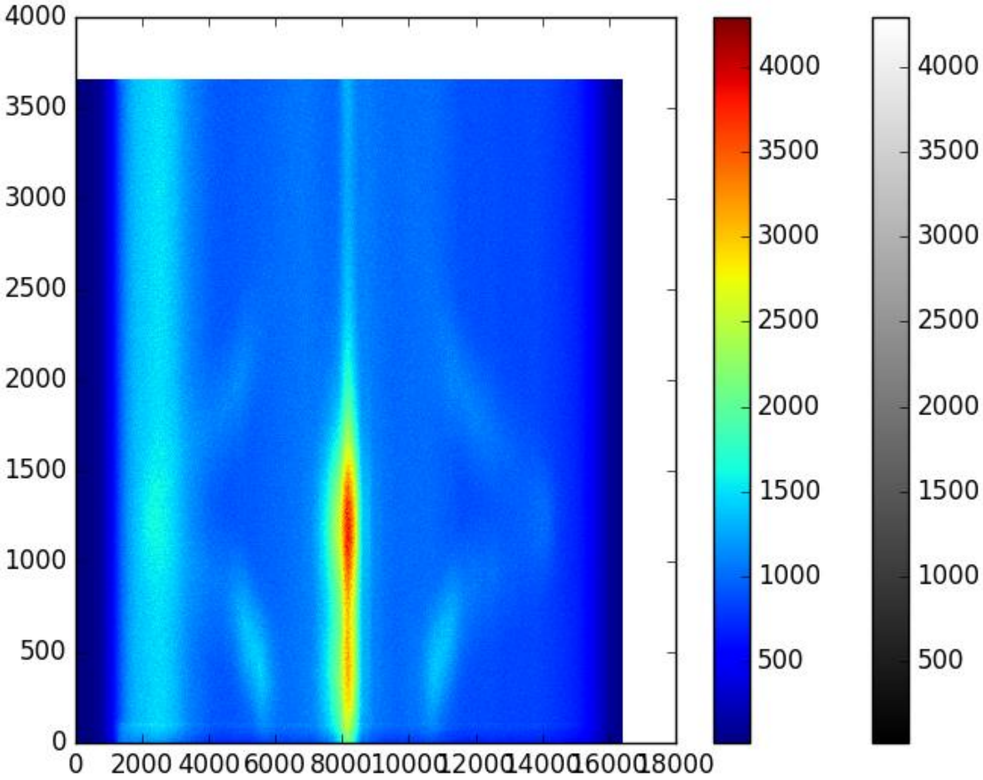


Figure 4: The color figure for the package 1452105273 in raw data.

In both two figures, we can see that on the X – Axis, it shows the frequency of data which we received from the Incoherent Scatter Radar (ISR), the Y-Axis show the height, there are 3657 heights and the color show the power of data. From black and while picture, we can see that there is the while color in the middle and it is also located in both two sides on the bottom and the top of the picture. We can see that the while color represents the peaks at the height, which we are coming to find it. We will see more clearly when we load the image in one height (2D).

From this picture, we can see that there are 3657 height, but how many kilometers of this. To solve that, they also have the rule to decode it to kilometer unit of one height. From the 0 height, it shows the 90 km, then we just take the real height by the number of height multiply by 0.3 and add with 90. To decode it, we have the function:
From the number of height we call h, then we have the function: height = h*0.3 + 90 (kilometer). For example, we have h=500, then we have height = 500*0.3 + 90 = 240 (Km).

3.3.2. Finding the peak in one height

In one height, we have the figure like the picture below:

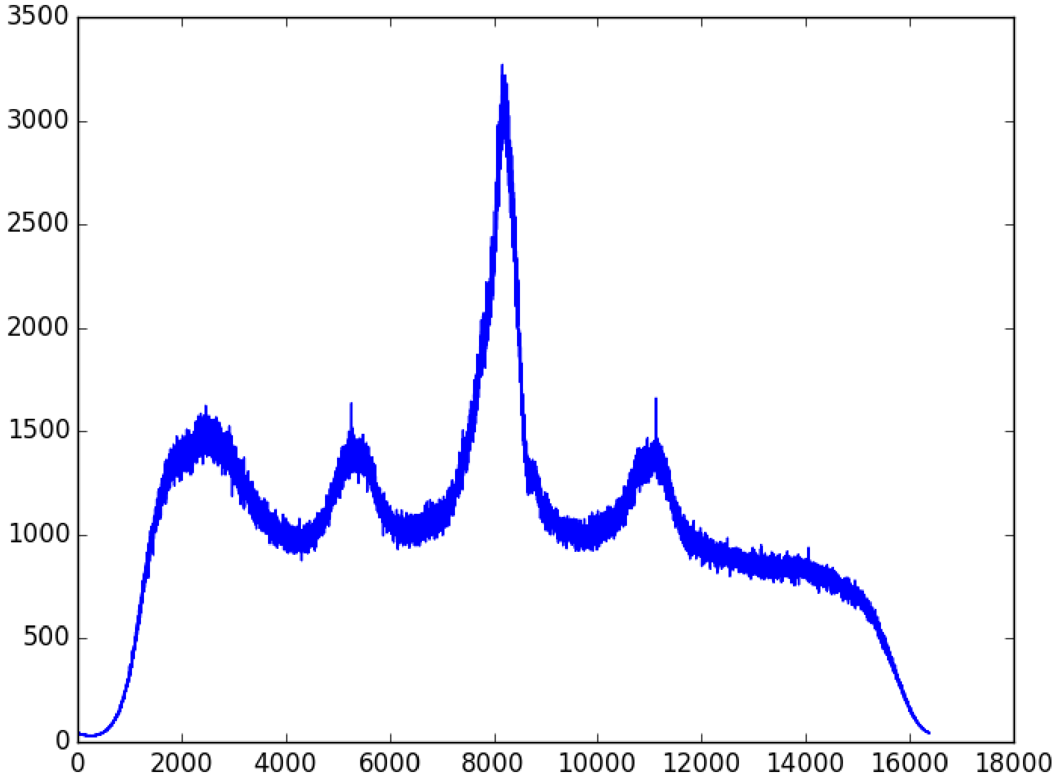


Figure 5: The figure show about the 2D images for the package number 1452105273 at the height 550 (255 km).

We can see that from the figure of the 3D above, we have the 3D picture with the height on the Y-Axis, then we just cut at the height number 550 we will get this figure. From this figure, the X-Axis shows the frequency and the Y- Axis show the power of data. Also from this picture, we can see that there are a lot of noises in the data and it makes difficult to read and know what the real peak of data is.

3.3.3. Processing the whole data

To make data easy to read and quick to process, they also use the algorithms to clean data first then it will give to the user the end data in the .hdf5 extension. From this file extension, we will plot the data which they are needed. There are some example of the plotting data from the final extension file in some figures below:

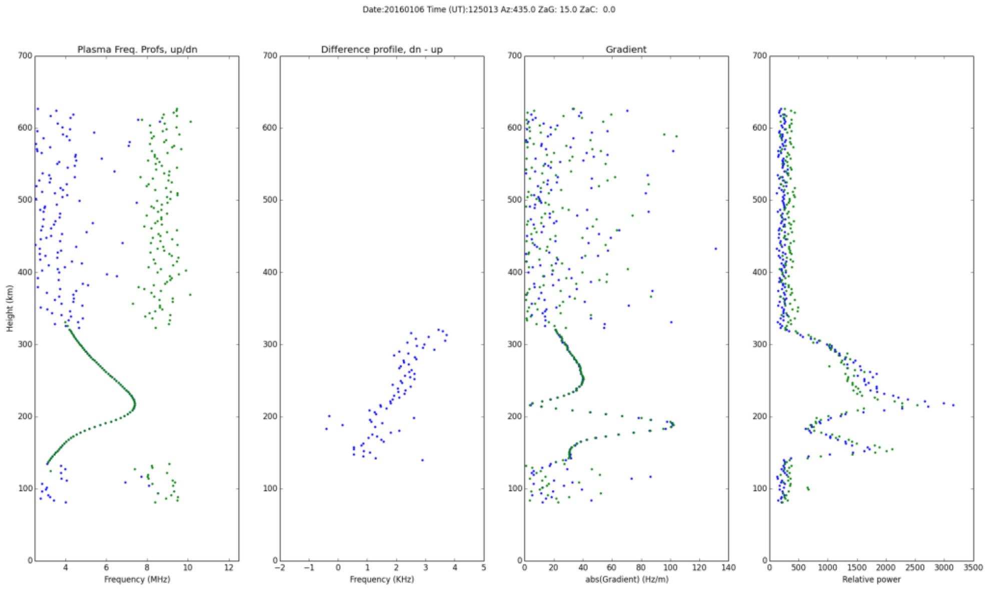


Figure 6: The figure show the plasma frequency of the clean data on the day 06 January 2016 at 15:00.

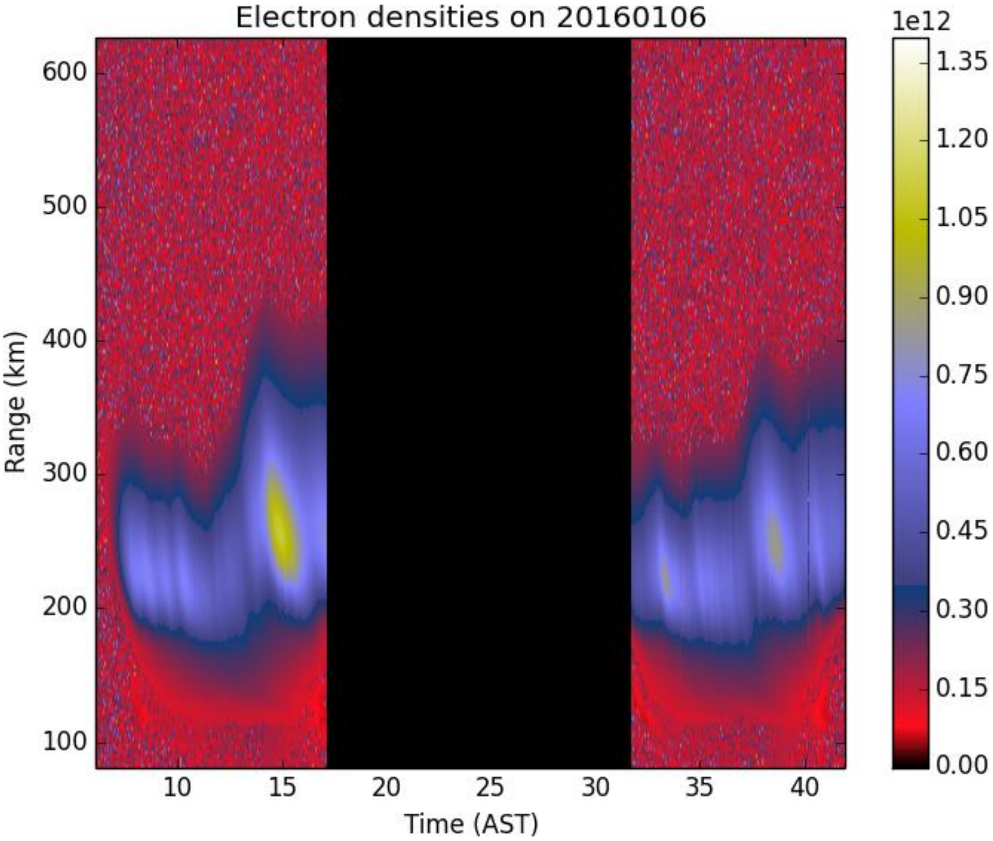


Figure 7: The figure show the plasma electron density of the clean data on 06 January 2016.

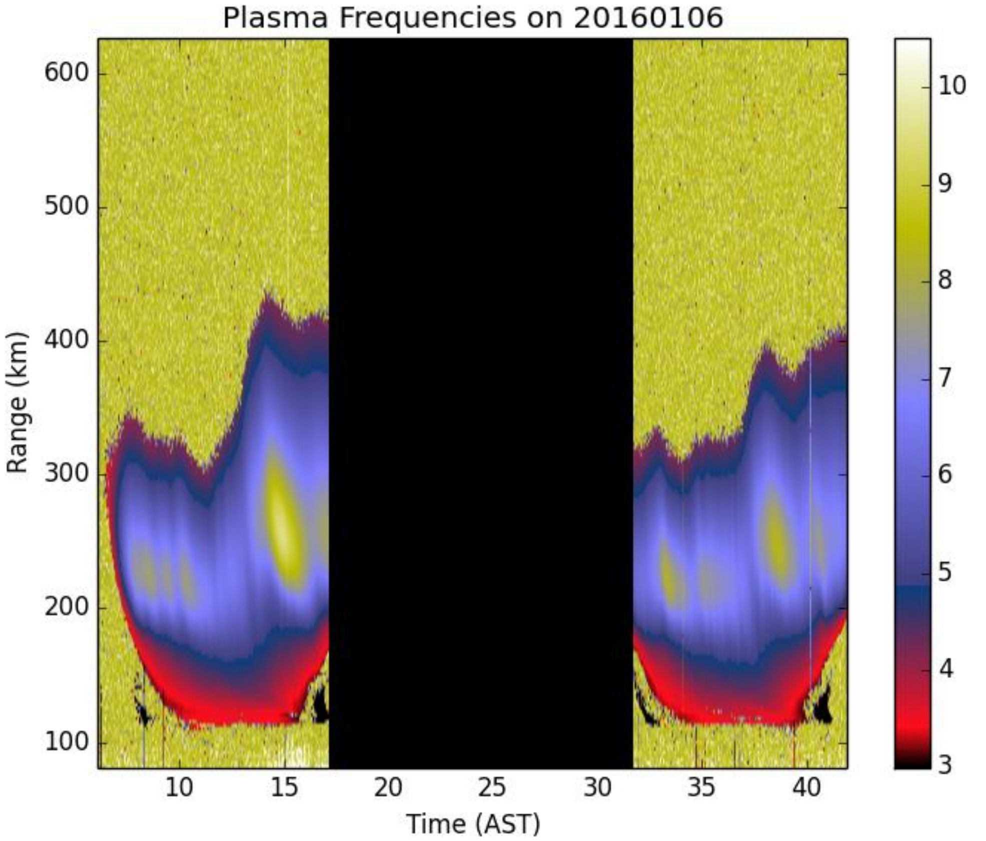


Figure 8: The figure show the plasma frequency of the clean data on 06 January 2016.

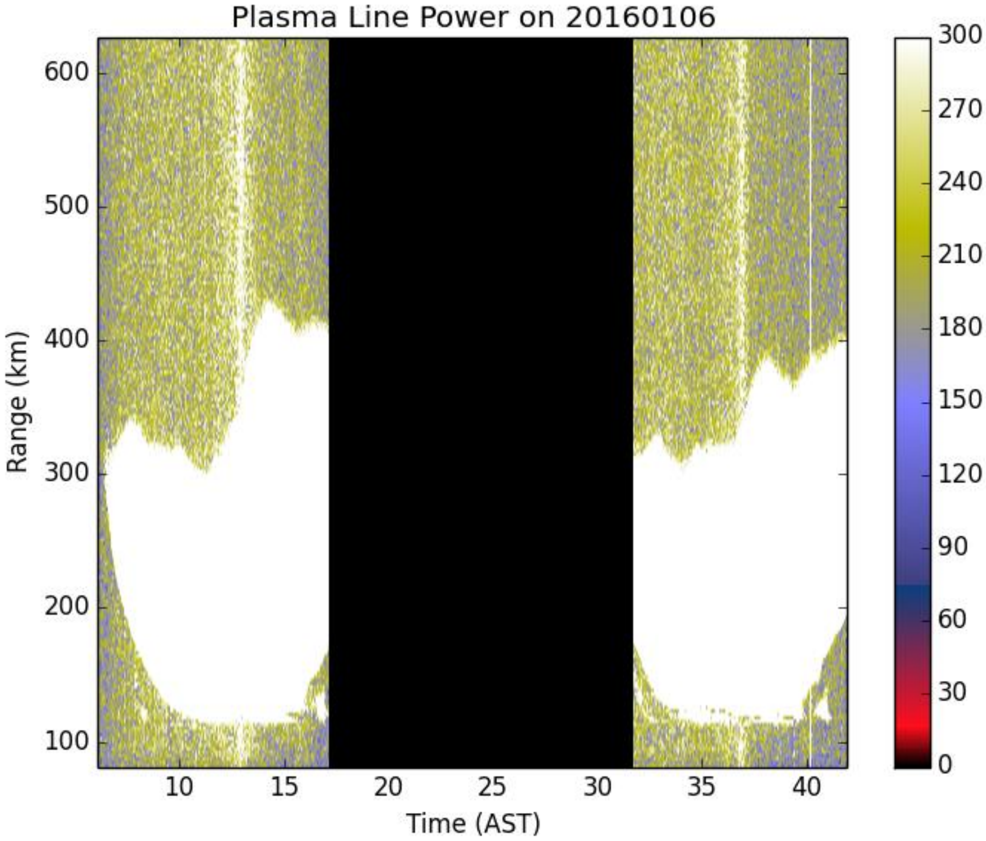
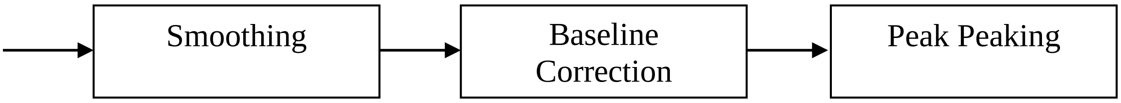


Figure 9: This figure show the plasma line power of the cleaning data on 06 January 2016.

3.4. Algorithms

In this section, it will show about some algorithms which are researching and using to solve the problem. First, we will discuss about the processing for peak detection process in the figure below:



There are a lot of algorithms for each step in the finding peaks. However, they can skip the baseline correction to go to peak peaking from the smoothing algorithms.

They also have some algorithms for the smoothing algorithms, they are:

- Moving Average filter
- Savitzky – Golay filter
- Gaussian filter
- Least Square Mothod filter
- Match filter
- Kaiser window
- Continuous Wavelet Transform
- Discrete Wavelet Transform
- Undicimated Discrete Wavelet Transform

For the baseline correction, we also have some algorithms to do this:

- Monotone Minimum
- Linear Interpolation
- Loess
- Continuous Wavelet Transform
- Moving Average of Minima

For the peak finding algorithms, we have some algorithms for this:

- SNR
- Detection/ Intensity Threshold
- Slopes of Peaks
- Local Maximum
- Shape Ratio
- Ridge Line
- Model-Based Criterion

[16]

In this section, we just discuss about the algorithms for the smoothing and finding peaks algorithms.

3.4.1. Smoothing algorithms

3.4.1.1. Moving Average

This part will show about the moving average filter algorithm. For this algorithm, the output of the moving average filter can call is y[n]. It can create by the function:
y[n] = x[n]*w[n] = $\frac{1}{2k+1}\sum_{i=-k}^{k} x[n-i]$
Where w[n] = $\frac{1}{2k+1}$, -k ≤ n ≤ k. The odd number 2k+1 represents filter width. The greater the filter width, the more intense the smooth effect.

In the function of the algorithm, k is the window size, x is the interval and y[n] is the output value.
[16]

3.4.1.2. Savitzky – Golay filter

The Savitzky – Golay fitting can be considered as a generalizer moving average filter. It performs a least square fit of the set of conclusive data points to a polynomial and takes the central point of the fitted polynomial curve as output.

The smoothed data point y[n] by applied Savitzky – Golay filtering is given by the following equation:
y[n] = x[n]*w[n] = $\frac{\sum_{i=-k}^{k} A_i x[n-i]}{\sum_{i=-k}^{k} A_i}$
where w[n] = $\frac{A_i}{\sum_{i=-k}^{k} A_i}$, -k ≤ n ≤ k.
Here A_i controls the polynomial orders.
[16]

3.4.1.3. Gaussian Filter

The output of the Gaussian Filter can show as the function below:

y(t) = x(t)*w(t) = $\int_{-\infty}^{+\infty} x(\tau) w(t-\tau) d\tau$
Where w(t) = $\frac{1}{\sqrt{2\pi\sigma}} e^{\frac{t^2}{2\sigma^2}}$
X(τ) is the signal.

The degree of smoothing is determined by the standard deviation σ. In fact, we can view Gaussian Filter as a weight moving average filter.

This filter sets large weight factors for points in the center and smaller weight factors for point away from the center.
[16]

3.4.1.4. Least Square Method Algorithm

This is the most basic filter to run on the data. For the least square method, the data will smooth by the function (least square function) to find the curve fitting. The following function is the linear combination of g_1... g_n.

$$\pi_{n-1} = \left\{\sum_{j=1}^{n} cjgj \,\middle|\, cj \in R\right\}$$

We have $g_j(x) = x^{j-1}$

To solve the problem, we choose function $g_1 \ldots g_n$ to represent polynomials and seek coefficients $c_1, \ldots, c_n$ so that

$P = \sum_{j=1}^{n} cjgj$

Minimizes R (c1,.., cn) = E(p). To compute $\frac{\partial R}{\partial ci}$ i=1… n. we denoted that:

$$\frac{\partial}{\partial ci}[yk - p(xk)]n - \frac{\partial}{\partial cj}\sum_{j=1}^{n} cjgj(xk) = -gi(xk)$$

Finally, the results is finding the coefficients matrix, then we replace it with x value (the value from raw data) to find the y value to fit it.
[17]

3.4.2. Peak peaking

This section will show the results, when apply the algorithm for finding the peak based on the above smooth algorithms. Those figures run on the package number 1452074613 of Arecibo example data.

The picture below show the peaks by using the least square method on the degree 7.

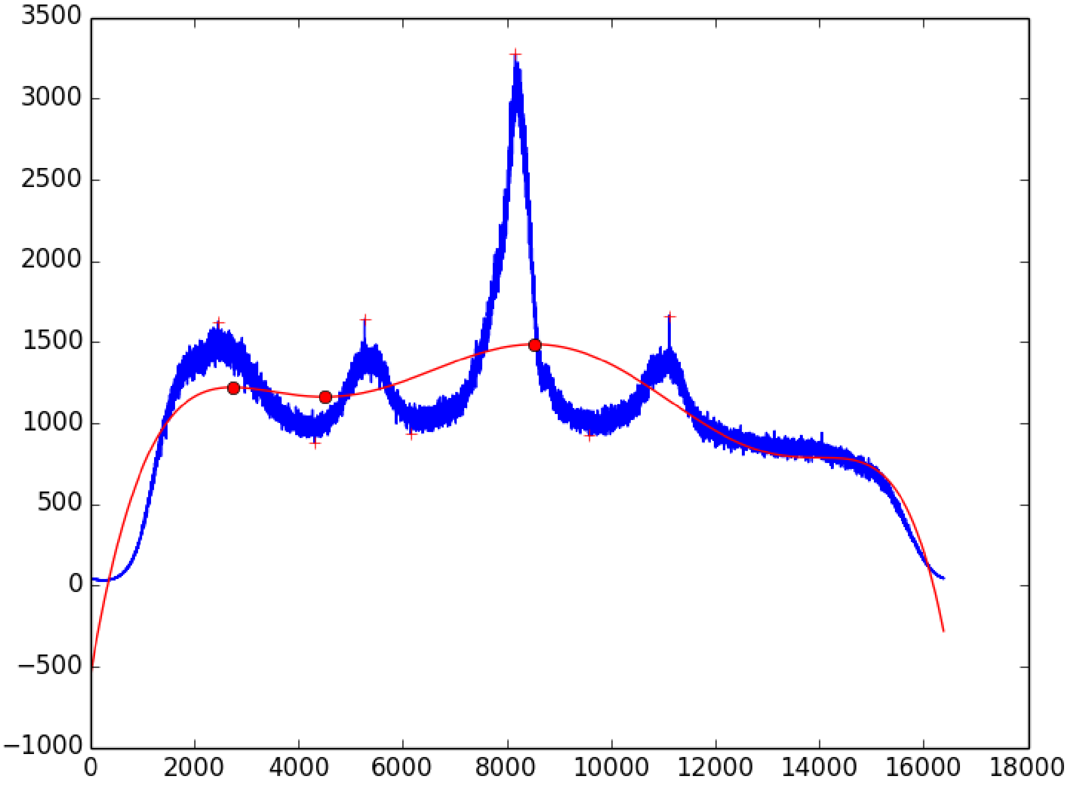


Figure 10: The figure shows the 2D smoothing data by Least Square Method and peaks on both smoothing data and raw data.

From this picture, we can see that the red plus show the finding maximum peaks and the minimum peaks of the original data. On the red circle show the maximum peaks and minimum peaks which finding based on the curve fitting data.

In addition, we will see the finding peak by the Savitzky-Golay algorithm method.

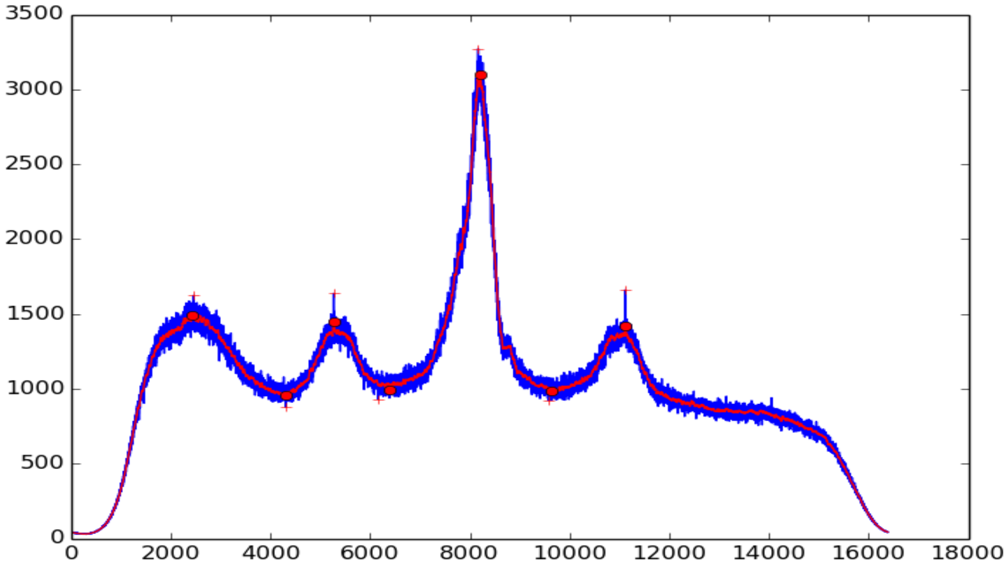


Figure 11: The figure show the 2D smoothing data by the Savitzky – Golay algorithm and the peaks of data on both smooth and raw data.

Moreover, we can see the finding peak by the moving average curve fitting algorithm in the figure below:

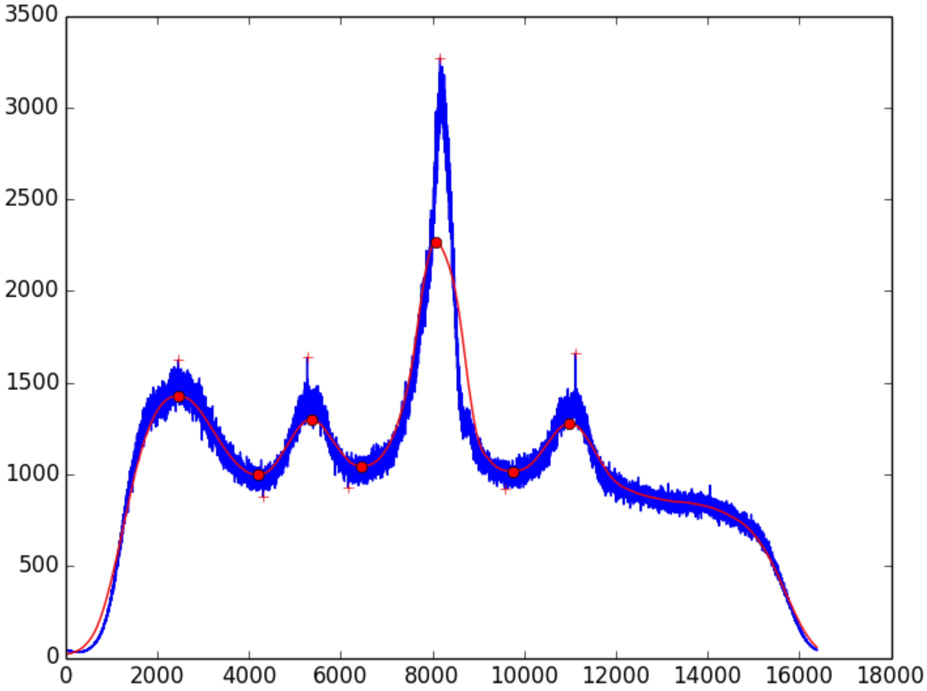


Figure 12: The figure show the 2D smoothing data by the Moving Average Algorithm and the peaks of data on both smooth and raw data.

In the figure below, we can see the figure of finding peaks by the Gaussian curve fitting algorithm.

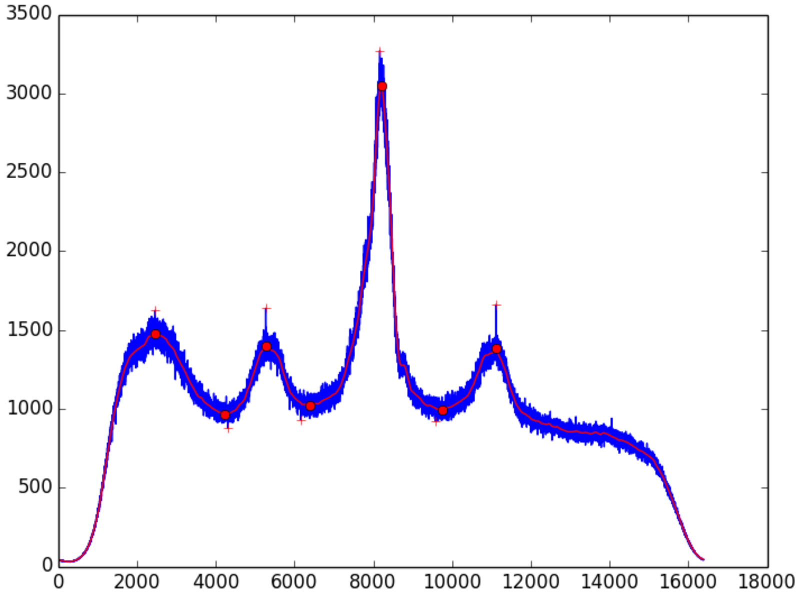


Figure 13: This 2D figure show the smoothing data by the Gaussian algorithm and the peaks of data on both smooth and raw data.

4. Previous work

4.1. Overview

One of the most important part of research or creating the application is to find and learning «the state of the art», from this learning, we can find the benefit and the mistake of each study and then we will understand the factor which can make this research or study become successful or not.

Today, there are some applications which use to analysis the data which received by Incoherent Scatter Radar (ISR). However, in the field of this thesis, it is quite new and a few people work on this. The reason is that GPGPU is the new technology and most of the development is coming from the physics departments in the station around the world. Some of the most know the program will be analyzed and discussed.

4.2. Python Code from Arecibo

One of the most helpful study is from Arecibo station Python code. As discussed above in the background part, they are already developing based on the Match Filter algorithm to clear data and finding peaks of data. Moreover, it is also available for loading the image which give to the scientist the overview of the information from Incoherent Scatter Radar (ISR).

It is very convenient code and very fully developed. However, it is taking much time to process the data. In addition, it also required the user need to install some special package of Python to run and the application running on the script which required the user need to have knowledge of Python to run this.

4.3. Miami previous work

This is the thesis of master student who submitted to the faculty of Miami University. In this thesis, they developed their application on the MATLAB script. They got some experimental results based on the algorithms which they got from their implementation.

In their work, they have developed their algorithms on the most three algorithms, global method, Lorentzian Method and Moment method. They did not provide more information about their study about the algorithms on their thesis. However, they provided for the reader their code in MATLAB and their example results which show that their thesis is successful to implement their code.

4.4. GPGPU

At the first time, we can see that this is the most nearly paper with this thesis. However, when we are coming on the details of the study, we can see that this is the study for the lowest level of data which we are not using it in the thesis. This is the paper of Nathaniel J. Hilliard submitted on August 2015 at the Department of Physics, University of Wisconsin – Madison. The title of the paper is GPGPU Acceleration of ISR Plasma Line Data and Application to Arecibo Plasma Line Striations.

In this paper, he used the cuFFT to transform the data and apply it to clean data while our thesis is developed for clean data and processing it in quickly as possible in the .hdf5 extension file which contained the information of many days.

4.5. Summary

There are several researches that based on the GPGPU idea. However, at the current time, there is no research idea as this thesis was found, with the concept of GPGPU idea. The ones found is that based on the CuFFT to process the data, but in the very low level of data while this thesis focuses on exporting the final data in .hdf5 extension. From this extension file, the end user can just use the simple script to load the data file in various formats and have the overview of the whole data in many days.

5. Architecture

5.1. Overview

This section gives the overview of the whole system when it is running in the real. In this section, we will overlook the physical facility in receiving signal and send the signal like radars, we just focus on the computer architecture which using to process the data.

For the whole system, first, the data will receive from Incoherent Scatter Radar (ISR) then store it on the database system, then, the backend system will work to process the data and then export the data into the .hdf5 file extension. From this format, user can available to use some simple script in various programming language to read those results and export it to the format which they want.

There is some requirement for the equipment which using for the thesis. Firstly, the computer which using to run the backend should have the NVIDIA graphics card, the reason is that the technology, which using in the thesis is CUDA, that require an NVIDIA graphics card to run. Moreover, the storage of the device which use for storing the data much larger space, one of the reasons is that it is a very large data to receive by Incoherent Scatter Radar (ISR) in the station, for example, the data which received by Incoherent Scatter Radar (ISR) on 06 January 2016 which we are using here is 1.8TB for just a 10 seconds**.**

5.2. Frontend

The frontend is the final results which provided to the user to use. In finally, the final results which show to the end user is just the simple file which contained the meaningful data. The final file will have a very small amount of data compared with the original data and it contained many information included date, hours, data, and so on.

To export the data into the pictures for overview of data, the end user should have a script to read the .hdf5 file. They are simple to read and simple to load it in the Python programming language. In the figure above on the background section, I already mention about those figures in final for the end user. Moreover, in the .hdf5 extension, it is easy for end users to write scripts to read, there are various programming languages available to read this file format.

5.3. Backend

Another part of this thesis architecture is the backend component. The backend is the most core processors to process the data.

As required, the backend computer should have the NVIDIA graphics card for processing data. The main functions of backend is that it will receive data from the storage space in the pair of package in .dcd and .hdr formats and processing pair by pair, to clean the data. In this process, the meaningful data will store in the .hdf5 file. Meanwhile, the meanless data will be eliminated.

5.4. Storage

The third component of this thesis is the storage facility. The storage facility can be one computer which has the largest space to store the data and included processing data in the backend component (we use this architecture). However, it may be a separate computer which just responsible to carry the data, then it will transfer data pair by pair to the processing computer to process data.

As we can see that the second architecture is more safety of the data than the first architecture because if it have any problems with the processing computer, it still have the backup data. And in the final processing procedure, the end user cannot get in touch on the original data, it give for the system have a stable status and the lower risk status because the end user cannot change any information on the original data.

6. Design

6.1. Overview

This section will show about the fundamental design choices will be explained and elaborated, why they were made and how they are designed like this. As mention above, this thesis will design mostly in the backend for improving the speed of calculating and improve the result.

In this thesis, the storage and the processing will run on the same machine. In addition, the results are also stored in the same machine. The script in Python code also the same. It makes for the developer easy to develop and looking back to the database and the results. It is also easy for the user to access the frontend data which show for them the real time result and they can easily to access the script which provided to them.

In this section, we also discuss about the code analysis, which we got from Arecibo station. Moreover, we also discuss about which part we decide to apply to solve.

However, they also have limited because when the processing machine make the system crash, the user also cannot access the computer and whole computer will down. Moreover, when system down, the system will run from the first time, it make the system rewrite it.

In this section, we will also see elaborated further in this chapter is how a system designed with the help from the third party provider and how the system design communicate in the backend component. It also describe the experiment code of the design.

6.2. The analysis from Arecibo Python code

This part will describe about the analysis of the Arecibo Python code. The contents will include measuring the time of the whole process, the time measuring of each part of the code and then we will decide to apply which parts into GPGPU to improve it. For every time measurement, I measured 10 testing times, then get the average time.

First, we will illustrate about the time measuring of the code (in seconds). Below is the table show the time measuring for the whole processing data for one pair of data in Arecibo code.

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Time
Main Process	82.596	74.604	67.046	69.721	68.764	68.302	68.057	68.07	68.106	68.07	70.334

Table 2. The table show the time processing for the whole data pairs in Python code.

From the table above, we can see that the average time for processing a pair of data is 70.334 seconds. This is quite large amount of data to process. For example, we have 1.8TB of data, it contained 7669 pairs of data, and it is simple to calculate that 7669*70.334 = 539391.446 seconds. It takes around 6.24 days to process this directory.

To inspect more details in the code, we can see that the code separate into two parts. One part is handled for reading the data, processing the data and exported it into the .hdf5 extension file while another part handled for plotting the result in the figures. We can see more clearly in the figure below:

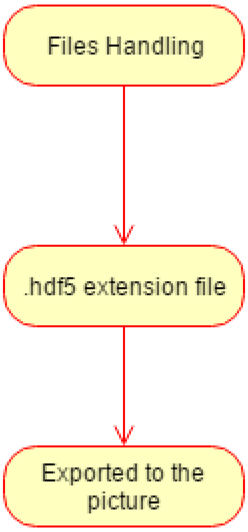


Figure 14. Figure show the data processing of the Python code from Arecibo.

Base on the figure, we can see that the most important part is the files handling part. We can see on the table below is the inspection of the code and also measuring time on each parts:

Functions	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Time
Reading file	1.26	1.38	1.19	1.2	1.19	1.28	1.29	1.2	1.2	1.19	1.238
Loading variable	0.02	0.08	0.09	0.08	0.06	0.02	0.02	0.09	0.06	0.06	0.058
Processing file	62.6	63.3	71.7	62.6	64.7	69.8	68.1	69.6	64.2	62.6	65.922
Writing data	0.03	0.08	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.03	0.036

Table 3. This table show the loading time for the part in the handling file in Python code.

From this table of data, we can see clearly in the error bar below:

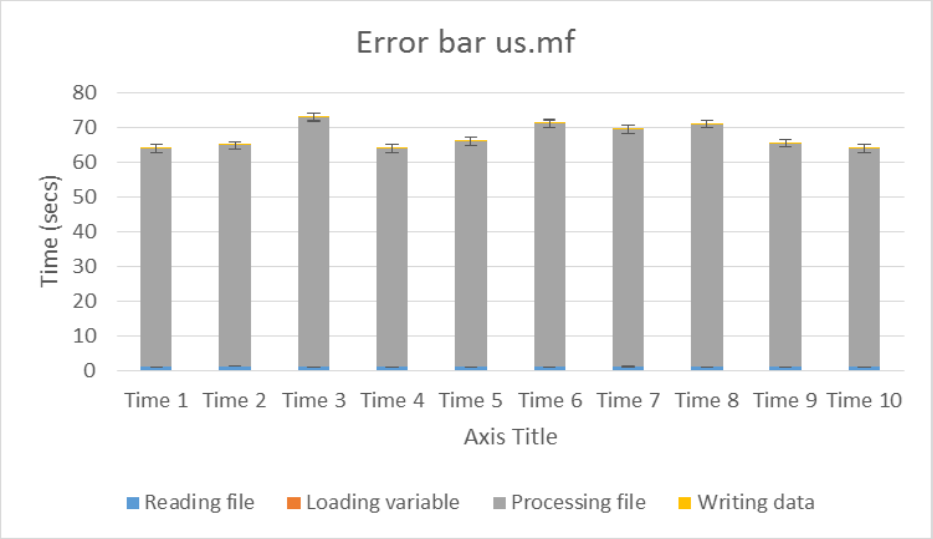


Figure 15. The figure show the error bar in the time measuring in Python code.

From this table, we can see that the most time taking part is the processing file part. To inspection this part, we will have the table in details of the function in the table below:

Function	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Time
Filter1Dauspr	62.9	62.9	62.3	62.4	62.7	72.4	64.5	65	63.5	64.6	64.326
Variable	0.01	0.03	0.05	0.03	0.01	0.01	0.05	0.01	0.01	0.01	0.02
Findpfpuspr	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
Findslopesupdn	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01

Table 4. This table show the data of time measuring on the function in Python code.

From this table, we have the error bar like the picture below:

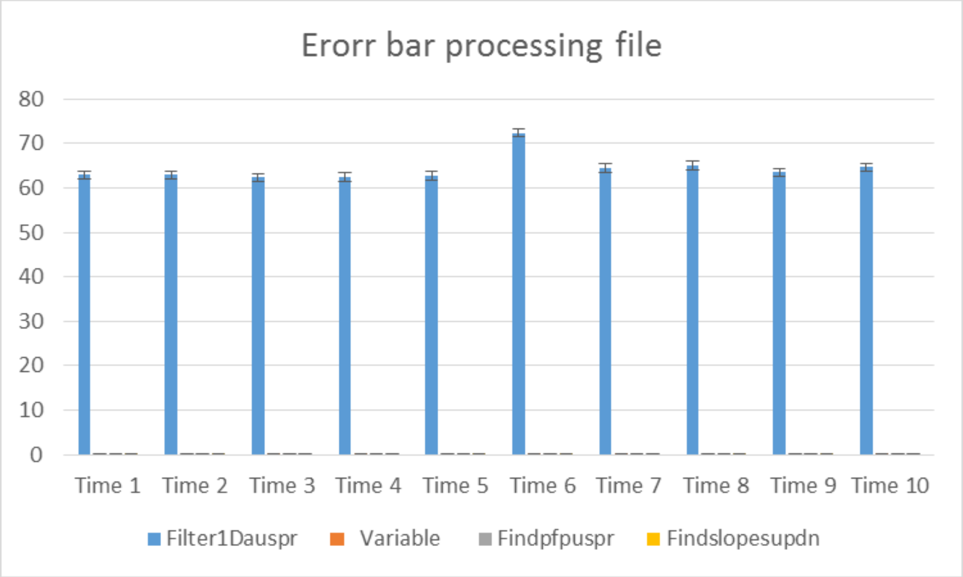


Figure 16. This figure show the error bar of the time measuring in functions of Python code.

From this inspection, we can see that the function name Filter1Dauspr is the function which took a lot of time. After discussion, we decide to apply the GPGPU into this part for implement the code in CUDA.

6.3. System layout

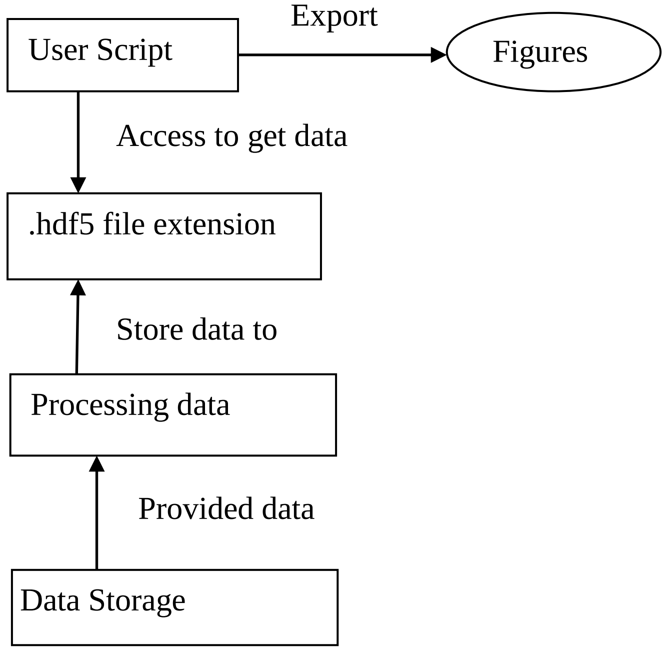


Figure 17: The figure show the design of complete system and the components connections.

From the figure above, we can see that they provided the visual overview of how the system design and the connection between components on frontend, backend and storage. On the frontend components, it provided to the user figures which depends on what user need by defined on the user script. As mention above on the background, we can see that user can define to load the data by electron density, power, frequency, and so on.

The backend component is in the processing data and store the data into the .hdf5 extension file. In the storage component, it just responsible to store the data and provided data for the backend to process. After process to clean the data, the backend will store the data into .hdf5 file (only one file) to prepare for the frontend component access.

6.4. Third-party provider

6.4.1. Transform the number to Discrete Fourier Transform and back

Discrete Fourier Transform (DFT) of a discrete set of real or complex number: x[n], for all integer n, is a Fourier series. NVIDIA provided a very interesting library which enable for the developer can change directly from the float number into Discrete Fourier Transform for calculating then transfer back from Discrete Fourier Transform to float number.

By using this library, the aspect of the algorithms to transfer the number into the Discrete Fourier Transform number and back was removed. It has also made for the developer make sure their results is more correctly and quickly in the process because the CuFFT library is using the NVIDIA graphics card to process which is very quick than using the CPU.

6.4.2. Storage the huge amount of data in small file

For processing the big data, the output data is a big problem because after processing that, the output data must be small. To solve this problem, HDF5 Group provided a very perfect data structure to store the data, but occupied the very small amount of data.

The Hierarchical Data Format product consisting of the data format specification and a supporting library implementation. By support of this data structure, we can classify the data from the original data, then store it into the group of data which can easy to process by the Python script. Moreover, the problem of the big amount of data also solved.

6.5. Backend

This part will show the backend design of the application. This is the core components of the application. In addition, in this part, we also discuss about the structure of the class inside the backend component, how they can interact together. It is implemented by the C++ language.

6.5.1. The overview of the backend

For the overview of the backend component, we can see the figure below:

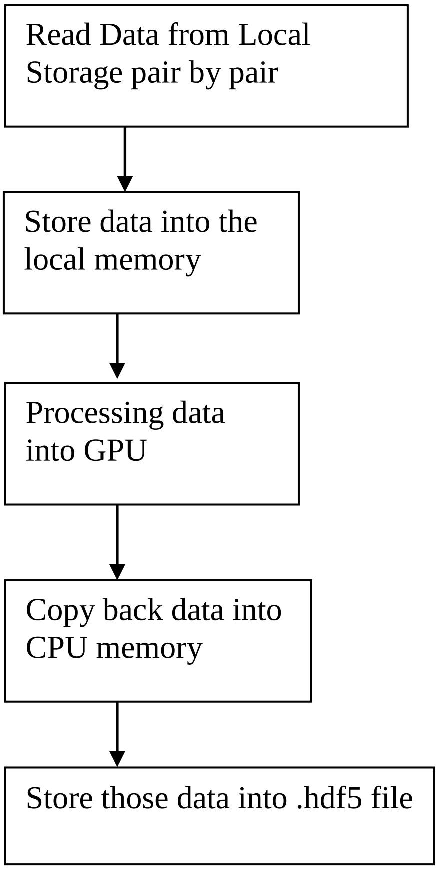

Read Data from Local Storage pair by pair
Store data into the local memory
Processing data into GPU
Copy back data into CPU memory
Store those data into .hdf5 file

Figure 18: The figure show the overview of the backend component.

From this figure, we can see that the application will access the database to read data pairs by pairs to load the whole data of one pair into the RAM memory. From RAM memory, it will transfer those data into the GPU for processing quickly by the NVIDIA technique. There are various libraries of NVIDIA for the developer to use to develop and make it more quickly than processing by using the CPU.

After processing the data by using the GPU, the data will transfer back and for the CPU to response to store those clean data into the .hdf5 file with the data groups and datasets for future use.

For the next pairs, the system process the same processing. However, in the final, the data will be written continuously with the .hdf5 file before, it make this file increase the occupied memory and information also.

6.5.2. The class design of backend component

This section will show about the class designed in the backend component and the class interaction between this.

The figure below will show the whole overview of the class design.

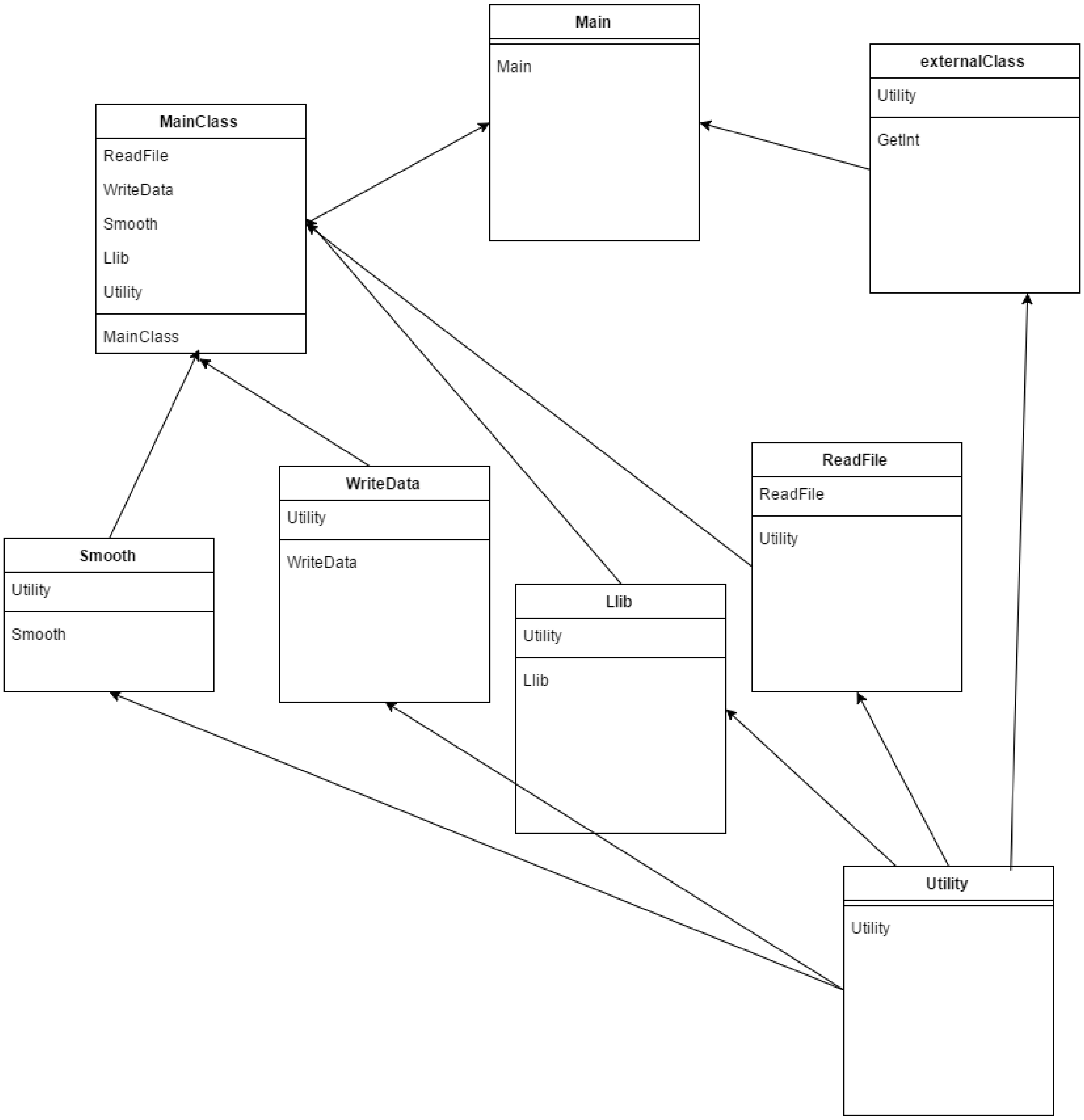


Figure 19: The figure show the classes' diagram in the CUDA and C++ Programing Language.

From this diagram, we can see that Main class is the most important class to call all functions in the thesis. From Main class, it calls the kernel which contained the CUDA code for transfer the data into GPU memory and it also calls the CPU code to process the data as normal. From this diagram, we can see clearly that there are two parts of the code, one is the kernel which contained the CUDA code and one is MainClass which contained the normal C++ code. Also from the diagram, we can see the Utility class is the lowest level class to contain the most basic function to processing data which using on both for kernel and the C++ normal code.

Moreover, we can see that every class contained the clear functions of this, the ReadFile contained the code which can read the file from the database. WriteData class contained the code handled to write information down to the .hdf5 file extension. Llib class contained the code to clean the data by the algorithm and Smooth class contained the class to smooth the data before to clean that.

6.5.3. The function of the backend component

First, let us discuss about the functions of the ReadFile class in the table below:

Function name	Meaning
ReadFile	This is the construction function
~ReadFile	This is the destruction function
readHdr	This is the function to read and return the string of header file
tommhhss	This is the function to read the second then return into the hours
readd	This function to read the .dcd data file
getsizeofFile	This function will get the total size of file
coverHdr	This function will return the string which contained the full path of the header package
coverDcd	This function will return the string which contained the full path of the data package
coverHdrMnf	This function will return the string which contained the full path of data package in the mnf mode
coverDcdMnf	This function will return the string which contained the full path of data package in mnf mode
returnNumHGHTS	This function will return the string which contained the NUM_HGHTS in the header file
returnFFTLEN	This function will return the string which contained the FFTLEN number in header file
rdusrpplHeader	This function to read the header file by the uspr mode
rdusrpplData	This function to read the data file by the uspr mode
readssHeader	This function to read the header file not by the uspr mode
readssData	This function to read the data file not by the uspr mode
returnSecs	This function return the string contained the second
readfnHeader	This function read the header file for the case not in any mode above
readfnData	This function read the header file for the case not in any mode above

Table 5: The table show the name and meaning of the functions in the ReadFile class.

Moreover, we can see that the function which have name WriteData will show the function name and the meaning in the table below:

Function name	Meaning of function
WriteData	The construction function

~WriteData	The destruction function
setVariableuspr	This function will set the initial value for the variable to prepare to write
writeUspr2HDF5	This is contained 2 functions which have the properties for the same name function, it response to write the data into the file in the uspr mode
writeComl2HDF5	This function which response to write the data into the .hdf5 file in coml mode
writeUpdn2HDF5	This function which response to write the data into the .hdf5 file in updn mode
save	This function which response to write data into .hdf5 file in none of above mode

Table 6: The table show the name and meaning of the functions in the WriteData class.

The most basic class is the Utility class. In this class, contained all the convenience function which can re-use for all class. The name and meaning of those function will describe in the table below:

Name of Function	Meaning
Utility	The construction class
~Utility	The destruction class
removeSpace	This function will remove the space in the line of string
fromStringToDouble	This function will return double number from input string
fromDoubleToString	This function will return string from double number
fromFloatToString	This function will return string from float number
fromIntToString	This function will return string from integer number
returnValue	This function will return the value of input string which contained text, number and space in the middle
fromCharToDouble	This function will return the double number from character
fromStringToLong	This function will return long number from string
fromStringToChar	This function will return character from string
array2DZeros	This function return the 2D zeros array in dynamic
array1DZeros	This function return the 1D zero array in dynamic
array3DZeros	This function return the 3D zero array in dynamic
linspace	This function translate the numpy linspace function to C
msqtf	This function return the so-called square

	triangle matrix
clip	This function translate the clip function in numpy library in Python
multiplies1DArray	This function return the multiply array
plus1DArray	This function return the plus array
roll	This function translate the roll function in numpy library of Python programming language
exp	This function return the exp function of double input
expf	This function return the exp function of float number input
expArray	This function return the array after enforcement exp function of input array
fileExists	This function return the path is exists or not
fromCharToString	This function return the string from character input
listFileandShorted	This function listed all the file in directory then short it by alphabet
returnFileNumberDirectory	This function return the number of file in the directory
zeroArray	This function set initial zero value for the array
reshape3D	This function return the 3D array which reshaped from 1D array
fromStringToInt	This function return the integer value from string input
oneArray	This function return the numpy function name ones of Python programming language
selectionShort	This function will sort the array in ascending
reshape	This function will reshape an array
swap	This function swap the value
getFFTLEN	This function get the FFTLEN from the string
array4DZeros	This function will return the zeros array in 4D
array2DZerosInt	This function will return the 2D array in integer number
print1DMatrix	This function will print 1D array
arrange	This function translate the arrange function in numpy library of Python programing language
array1DZerosComplex	This function create the 1D zeros array in complex type with initial value is 0
fftshift_3D	This function translate the fftshift function in numpy library of Python programing language
devide3DMatrix	This function divide the 3D matrix by the double number
returnFloatNumber	This function return float number from the complex number

print3DMatrix	This function print 3D matrix
print2DMatrix	This function print 2D matrix
returnMaxIndexValue3DThird	Return the max index value of matrix in the third dimensions
returnMaxIndexValue1D	Return the max index value of 1D matrix
maxpar	Return the maximum value and the x location in the matrix
setValueList1D	This function will set the value of the 1D list
printListVariable	This function print the list variable
array2DZerosComplex	This function will return the array in complex 2D
passMemory2D	This function will send the memory from 1D into 2D array
returnValueFromHd	This function return the string value from the input string and input substring then return the value
fromFloatToCufftReal	This function change the float number into cufftReal number in 1D and 2D

Table 7: The table show the name and meaning of the functions in the Utility class.

In addition, we also have the class to handle smooth the data. We can see the function and meaning of the function in the table below:

Name of the function	Meaning of the function
Smooth	The construction function
~Smooth	The destruction function
makeHc	Set the value for the matrix
makeHl	Set the value for the matrix
makeHq	Set the value for the matrix

Table 8: The table show the name and meaning of functions in the Smooth class.

The most important class is the class which apply the algorithms to clean the data. It is the Llib class. We will see the name and meaning of the function in this class in the table below:

Function name	Meaning of the function
Llib	This is the construction function
~Llib	This is the destruction function
calculateFilter1Dauspr	This function will calculate the first part of the filter in uspr mode
filter1Dauspr	This function will calculate the filter in uspr mode
filter1Daupdn	This function calculate the filter in updn mode
filter1Dacoml	This function calculate the filter in coml mode
getNcols	This function return the _ncols value
getNrows	This function return the _nrows value
getNslopes	This function return the _nslopes value

getNskip	This function return _nskip value
getSlopes	This function return _slopes value
getnhts	This function return _nhts value
getnfft	This function return _nfft value
getWidth	This function return the _width value
getSqtfs	This function return _sqtfs value
setfqtfs	This function will set the value of array fqtfs
mllib	This function will calculate the value for sqtfs, fsqtfs, widths, shifts array
Filter1Dauspr_snr	This function will read and pass the data into GPU

Table 9: The table show the name and meaning of the functions in the class Llib.

The important class in the CPU part coding is the MainClass. This is the class which contained all related class. The details about the name of the function and the meaning of those function will discuss in the table below:

Name of function	Meaning of the function
MainClass	This is the construction function
~MainClass	This is the destruction function
setValue	This is the function to set the initial value
getNumChan	This function return integer number which contained the NUM_CHAN value
getNumHGHTS	This function return the NUM_HGHTS value
getFFTLEN	This function return the FFTLEN value
getNsLopes	This function return the _nslopes value
getnfft	This function return the _nfft value
getWidth	This function return the _width variable
getsqtfs	This function return the _sqtfs array
getNcols	This function return the _ncols variable
findpfpcoml	Find the value from data in the coml mode
findpfpupdn	Find the value from data in the updn mode
findpfpursp	Find the value from data in the ursp mode
findslopescoml	Find the value slopes in coml mode
findslopesupdn	Find the value slopes in updn mode
setfqtfs	Set the value for fqtfs array
mf1	This is the loop function which will loop-over when read the data packages
processingData	This function will processing data to get the final results
getArr3D	Get the array in 3D of data
getar3Dx	Get the x dimension numbers of data 3D array
getar3Dy	Get the y dimension numbers of data 3D array
getar3Dz	Get the z dimension numbers of data 3D array
getnhts	Get the _nhts variable value

getNrows	Get the _nrows variable of data
getShifts	Get the _shifts variable of data

Table 10: This table show the name and meaning of the functions in the MainClass class.

6.6. Frontend

In my thesis, the frontend for the user is the final .hdf5 file. In this file, it contained the information which can show to the user easy by just using the Python script to see. My thesis is not writing this script, the script is written and copyright by Arecibo Station in USA. Below is some example results in the final plotting data for the end user:

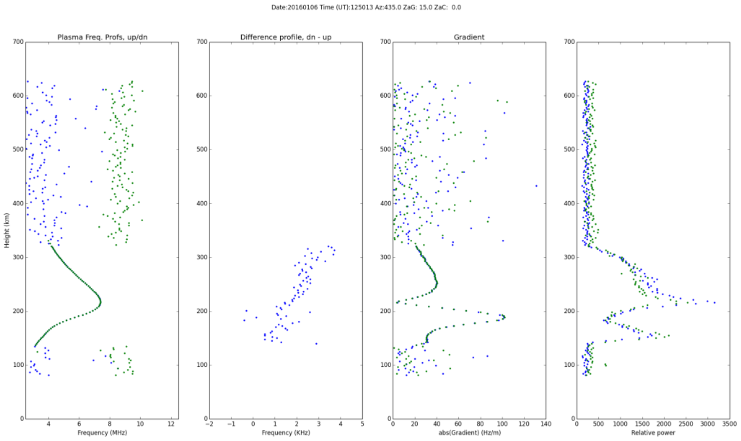


Figure 20: The figure show the final frontend in the picture of plasma frequency.

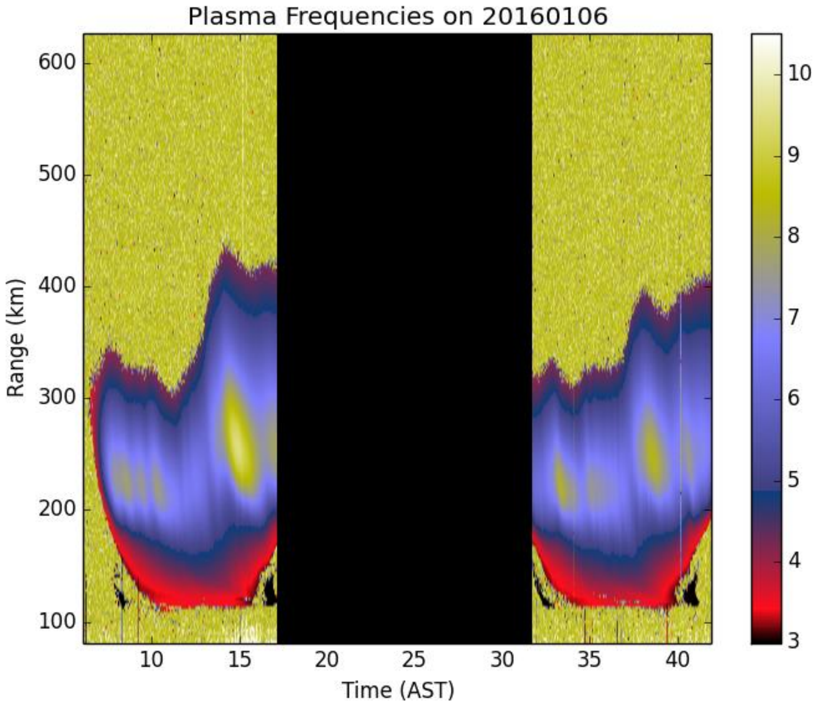


Figure 21: The example plasma frequency in plotting by the Python script from frontend .hdf5 extension file.

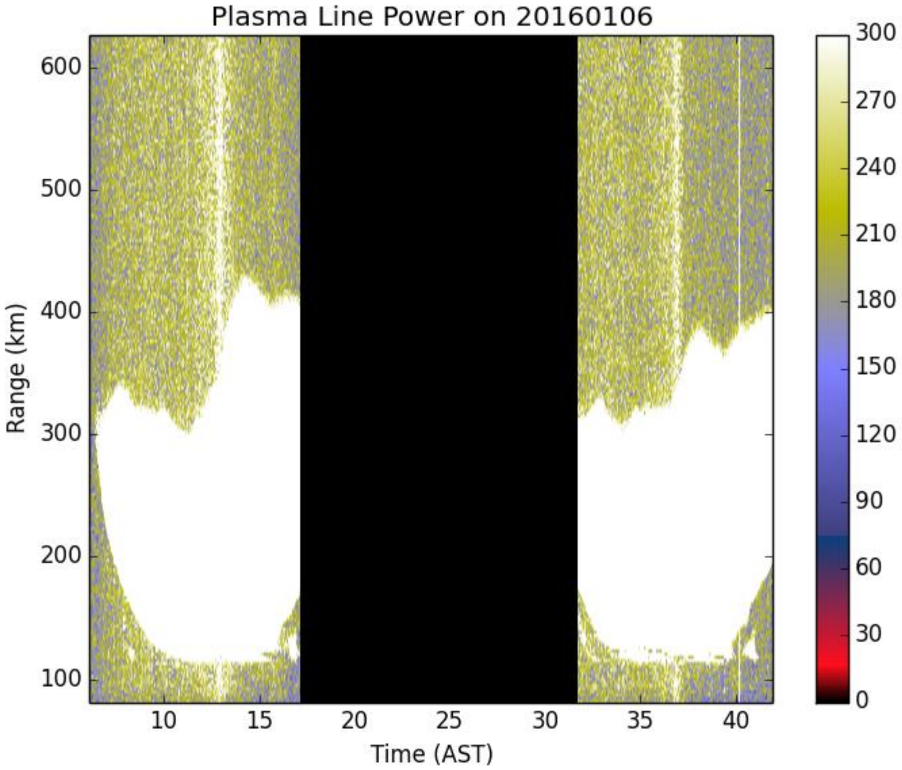


Figure 22: The figure show the plasma line power which plot by the Python script from frontend –hdf5 extension file.

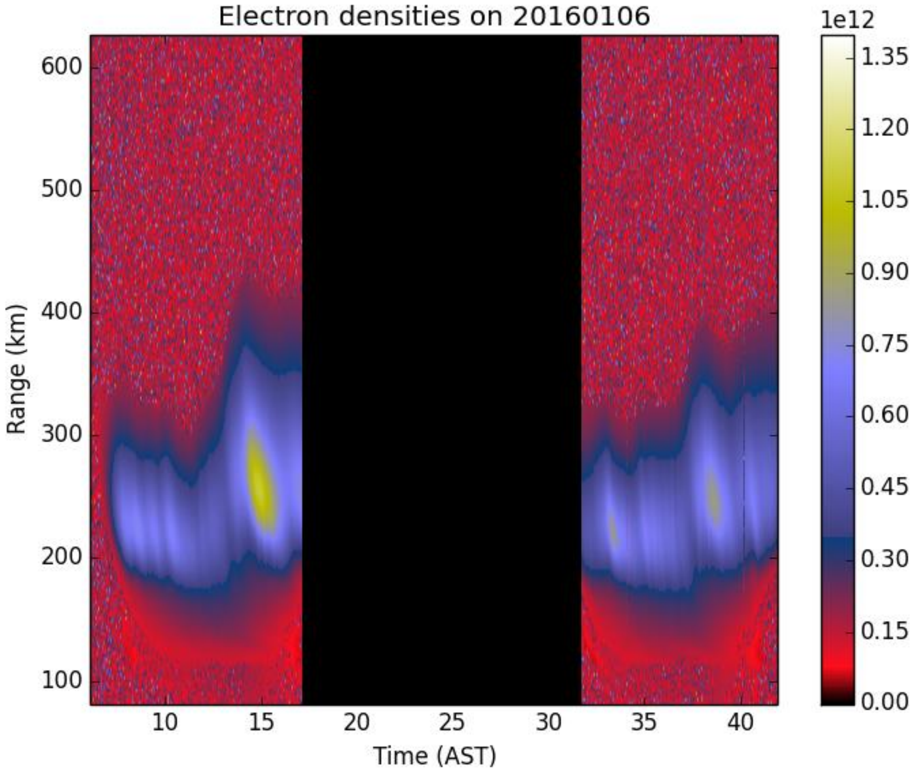
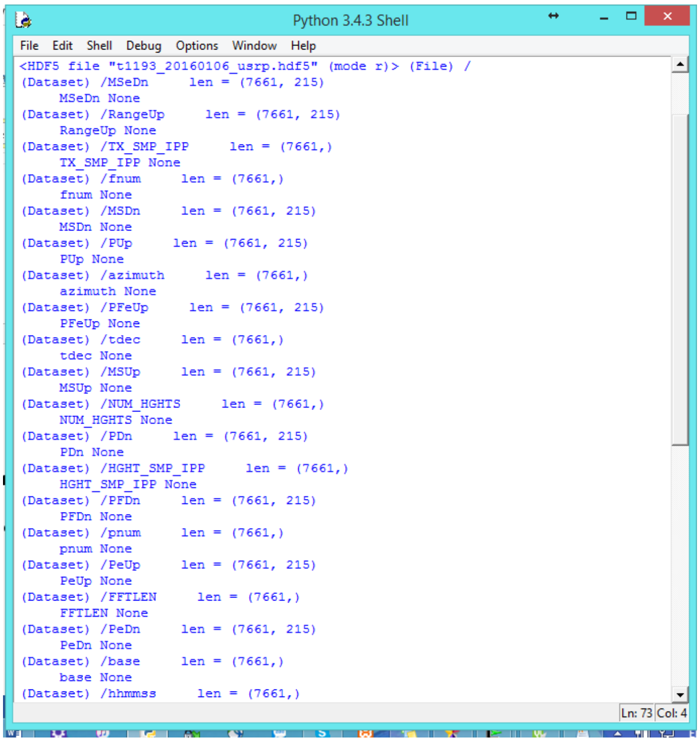


Figure 23: The figure show the plasma electron density which plot from the .hdf5 extension file by the Python script.

As we can see that, the user can plot by the script is quite easy. In the picture below, it will show the structure of the .hdf5 file.



```
<HDF5 file "t1193_20160106_usrp.hdf5" (mode r)> (File) /
(Dataset) /MSeDn     len = (7661, 215)
     MSeDn None
(Dataset) /RangeUp     len = (7661, 215)
     RangeUp None
(Dataset) /TX_SMP_IPP     len = (7661,)
     TX_SMP_IPP None
(Dataset) /fnum     len = (7661,)
     fnum None
(Dataset) /MSDn     len = (7661, 215)
     MSDn None
(Dataset) /PUp     len = (7661, 215)
     PUp None
(Dataset) /azimuth     len = (7661,)
     azimuth None
(Dataset) /PFeUp     len = (7661, 215)
     PFeUp None
(Dataset) /tdec     len = (7661,)
     tdec None
(Dataset) /MSUp     len = (7661, 215)
     MSUp None
(Dataset) /NUM_HGHTS     len = (7661,)
     NUM_HGHTS None
(Dataset) /PDn     len = (7661, 215)
     PDn None
(Dataset) /HGHT_SMP_IPP     len = (7661,)
     HGHT_SMP_IPP None
(Dataset) /PFDn     len = (7661, 215)
     PFDn None
(Dataset) /pnum     len = (7661,)
     pnum None
(Dataset) /PeUp     len = (7661, 215)
     PeUp None
(Dataset) /FFTLEN     len = (7661,)
     FFTLEN None
(Dataset) /PeDn     len = (7661, 215)
     PeDn None
(Dataset) /base     len = (7661,)
     base None
(Dataset) /hhmmss     len = (7661,)
```

Figure 24: The figure show the structure of the .hdf5 extension file.

6.7. Experiment code

In this section, I will discuss more detail about the several Python Script and the C++ class to use in the thesis. And it can be found in the apendix section.

- CMakeLists.txt is the script to use for the CMake to run. It make my project have clear to organize.

- main.cu is the main class which contained the main function to run this program. From this class, it contained both the CUDA code and normal C++ language code.

- externalClass.cu and externalClass.cuh is the class which contained the code for the CUDA code. The CUDA code just can run on the .cu file.

- llib.h and llib.cpp this is the class contained the code which handles to clean the data.

- mainclass.h and mainclass.cpp, this is the class contained for the whole process in the project. In this class, it calls all small class likely the class figure above. And the main class calls this class.

- Readfile.h and readfile.cpp, this is the class contained the code which handle for reading the file and all related code.

- Smooth.h and smooth.cpp, this is the class contained the code for smoothing data.

- Utility.h and utility.cpp, this is the class handle for all related functions which can re-use in all other classes.

- Gaussian_filter.py is the script contained the code for smoothing data in the Gaussian algorithm.

- Least_square.py is the script contained the code for smoothing data in the least square algorithm.

- Moving_average.py is the script contained the code for smoothing data in the moving average algorithm.

- Savitzky_golay.py is the script contained the code for smoothing data in the Savitzky Golay algorithm.

- Findpeak_gaussian.py is the script which show the figure by apply the smoothing data in Gaussian algorithm and the peaks of that.

- Findpeak_least_square.py is the script which show the figure by apply the smoothing data in least square method algorithm and the peaks of data also.

- Findpeak_moving_average.py is the script which show the figure by apply the smoothing data in moving average algorithm and peaks of data.

- Findpeak_savitzky_golay.py is the script which show the figure by apply the smoothing data in Savitzky Golay algorithm and the peaks of data.

- 3D_gaussian.py is the script which will plot the figure in 3D of black and while and color in Gaussian Filter algorithm.

- 3D_least_square_method.py is the Python script which handle to plot the figure in 3D of black and while and color in least square method algorithm.

- 3D_Moving_Average.py is the script handling for plot 3D picture in black and while and color in moving average filter algorithm.

- 3D_savitzky_golay.py is the script which handle for plot the 3D figure in black and while and color in Savitzky Golay algorithm.

7. Results

7.1. Overview

This section will summarize the data collected from the experiment conducted in the previous chapter. Moreover, this chapter also show more clearly about the results in figure and the algorithms time measurement.

The experiment 1 part will show the experimental results which apply the algorithms in the background section in Python code. In addition, the experiment 2 will show the time measurement in applying the algorithm to clean data by using C++ and CUDA.

In the figures below will show the plot picture in 3D both color and black and while for the raw data. Moreover, it also illustrates the 2D figure at the height 550 (255 km) for the raw data which without applying any smooth algorithm.

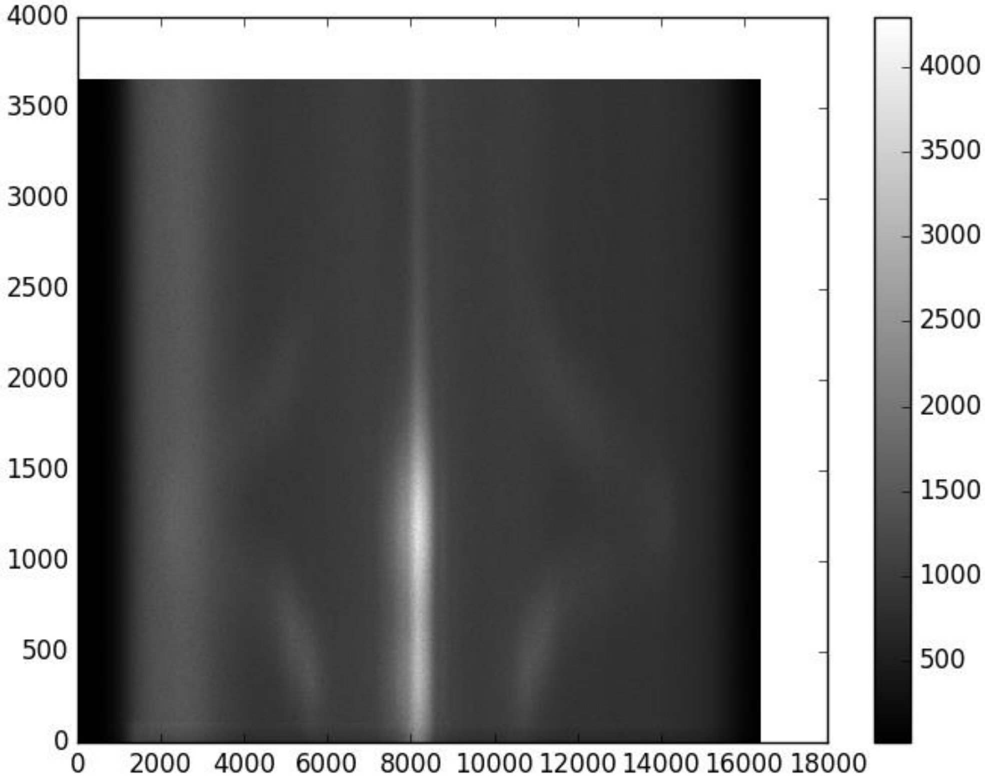


Figure 25: The 3D figure in black and while for all height in raw data for the package number 1452105273.

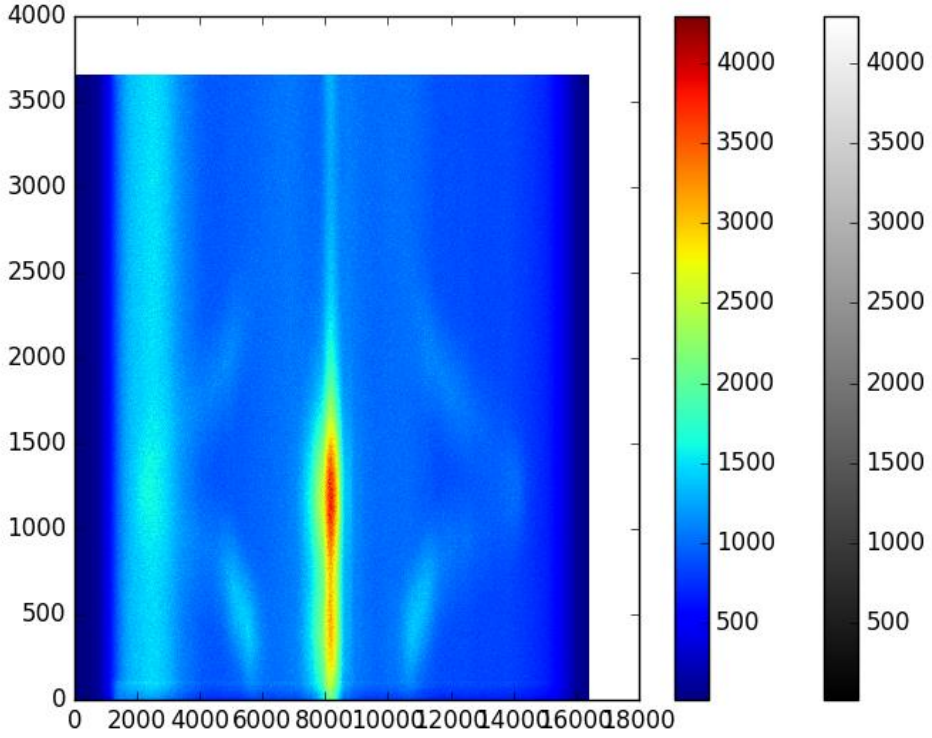


Figure 26: This figure show the 3D picture in color for all height in raw data for the package number 1452105273.

From the Figure 22 and Figure 23, we can see that both black and while and color figure is not clear and it contained a lot of noises. In these two figures, the X-Axis show the frequency of data while the Y-Axis shows the height number.

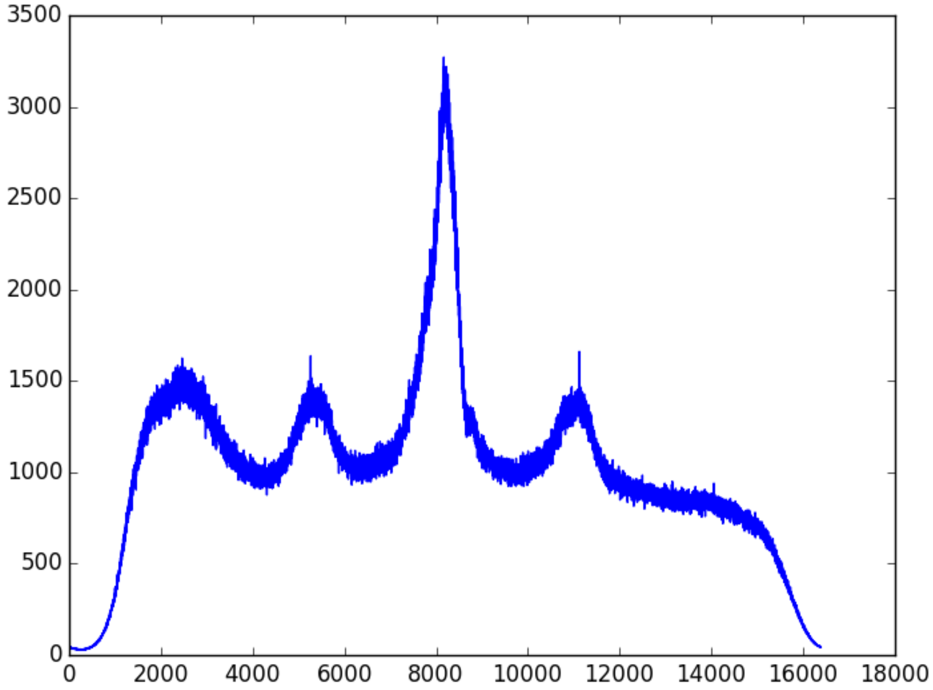


Figure 27: The figure illustrates the 2D image of the raw data at the height 550 (255 km) for the package number 1452105273.

As we can see in the figure 24, this is the figure of data when we cut in the height 550 (255 km). From this picture, the X-Axis show the frequency while the Y-Axis shows the power of data. It contained a wide range of noise which make us difficult to know where the real noise is and the fake noise.

7.2. Experiment 1

7.2.1. Smoothing data

This section will discuss about the results which I got from the smoothing data algorithms in the background section. It contained the figure in 2D. Moreover, this part also shows the figure in 3D in both color, and black and while figure.

7.2.1.1. Moving Average algorithm

By apply the algorithms in the section moving average algorithm in background part, firstly, I would like to show the result in the 2D figure in the picture below:

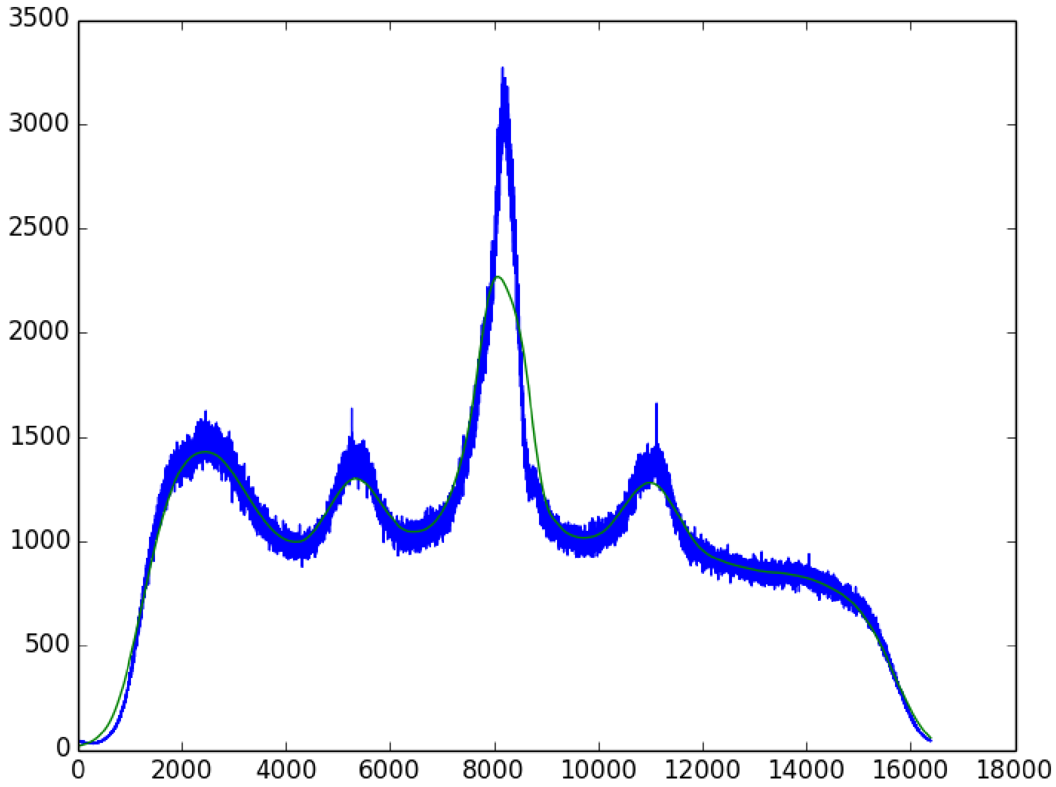


Figure 28: The 2D figure of the moving average algorithm in the height 550 (255 km) of the package number 1452105273.

From this figure, we can see that the X-Axis show the frequency of data and the Y-Axis show the power of data. The green line shows the original data which contained noise. After smoothing this by the moving average algorithm, we get the red line which show the smooth data by applying the moving average algorithm.

Moreover, I also implement the Python code which available for plot the data in the 3D picture. In the two figures below, it is the 3D figures in both color and black and while figures.

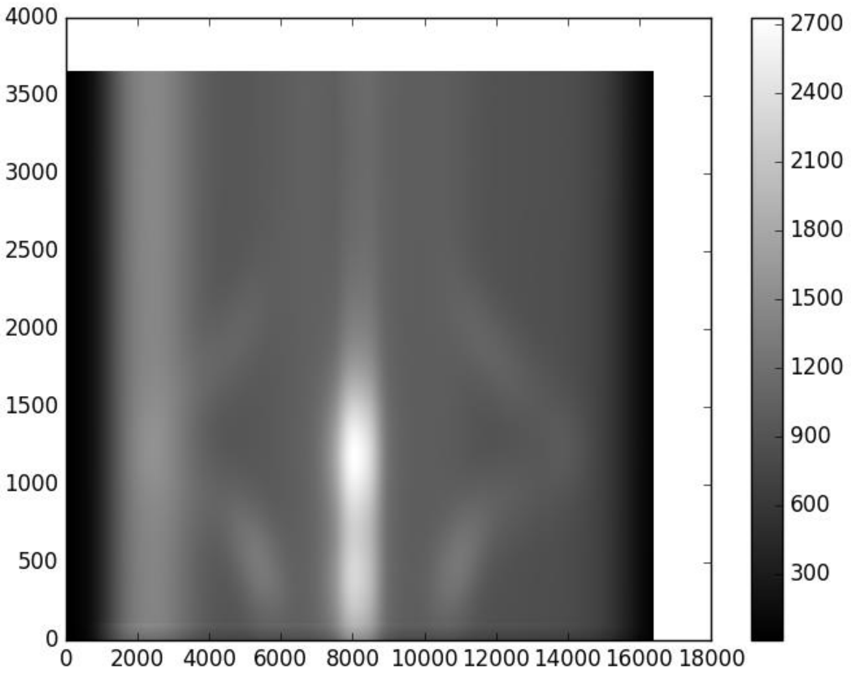


Figure 29: The figure show the Moving Average Algorithm in 3D at all height in black and while for the package number 1452150273.

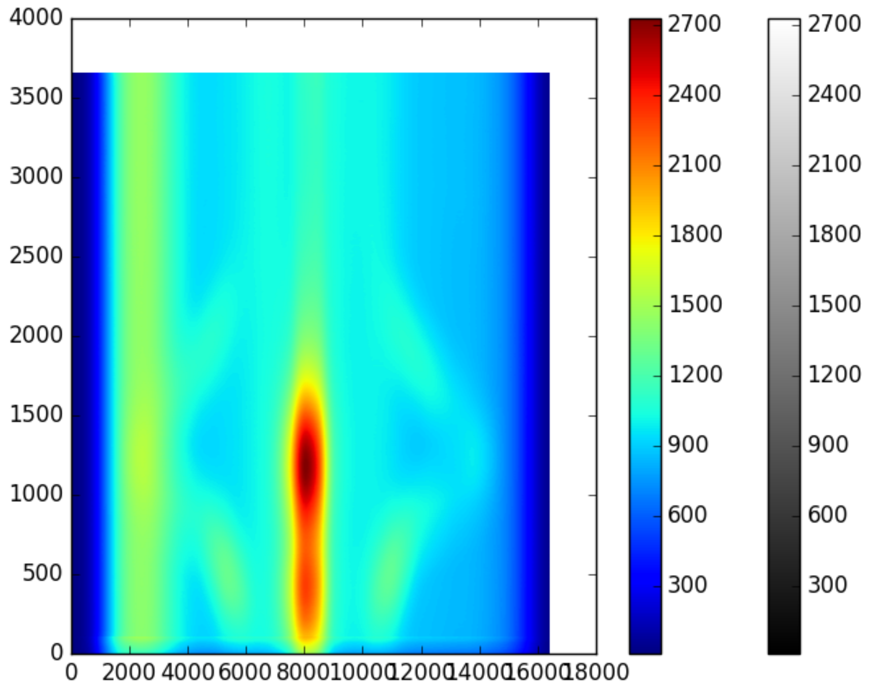


Figure 30: The figure show the 3D figure in color by applied the Moving Average Algorithm at all height for the package number 1452150273.

From these two figures, we can see that the X-Axis show the frequency of data while the Y-Axis show the height. The color shows the power and there are the power range to show number of power also. In the picture, there are a very high power in the middle while the power come slow in two sides. They just become high in some place it show where we have the peaks.

7.2.1.2. Savitzky – Golay algorithm

By apply the Savitzky – Golay algorithm in the background section, we can see that the data which we got will show likely the picture below:

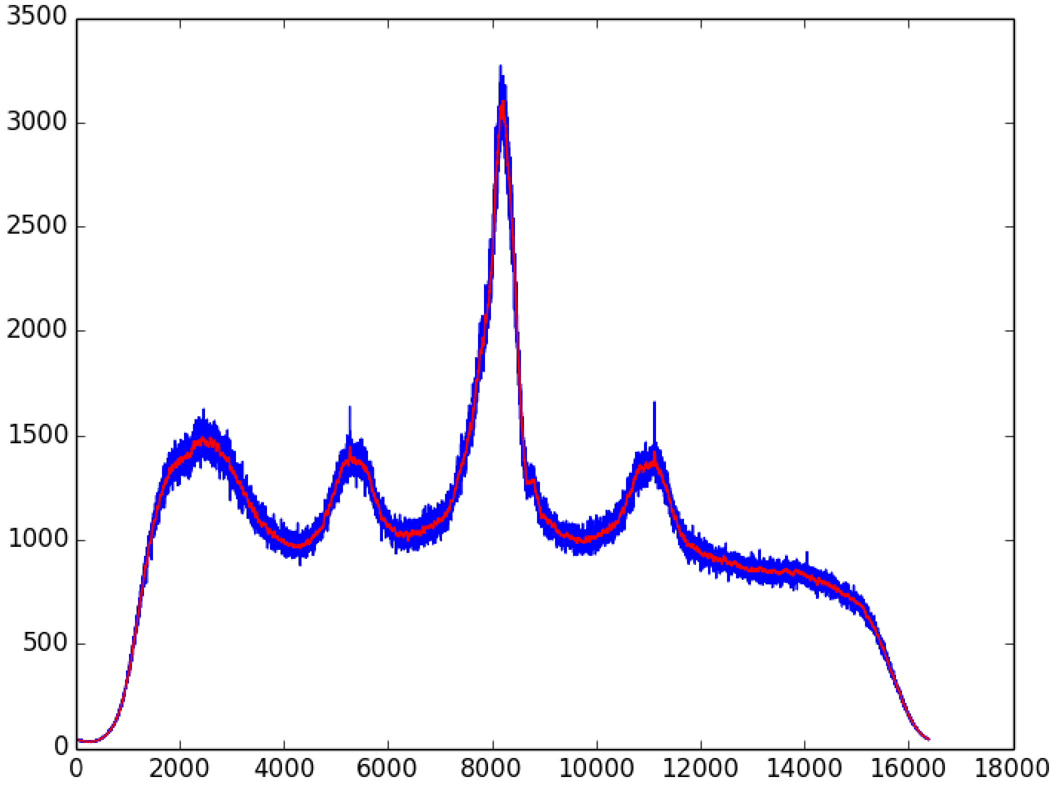


Figure 31: The figure show the 2D smoothing data by the Savitzky – Golay algorithm at the height 550 (255 km) of the package number 1452150273.

From this figure, we can see that the X-Axis show the frequency of data and the Y-Axis show the power of data. The green line shows the original data which contained noise. After smoothing this by the moving average algorithm, we get the red line which shows the smooth data by the Savitzky – Golay algorithm. Comparing with the moving average algorithm, we can see that the Savitzky – Golay algorithm is more correctly because we can see the noises just like shortly while the moving average algorithm show the smoothing line just look like the straight line.

Moreover, I also implement the Python code which available for plot the data in 3D picture. In the two figures below, it is the 3D figures in both color and black and while figures.

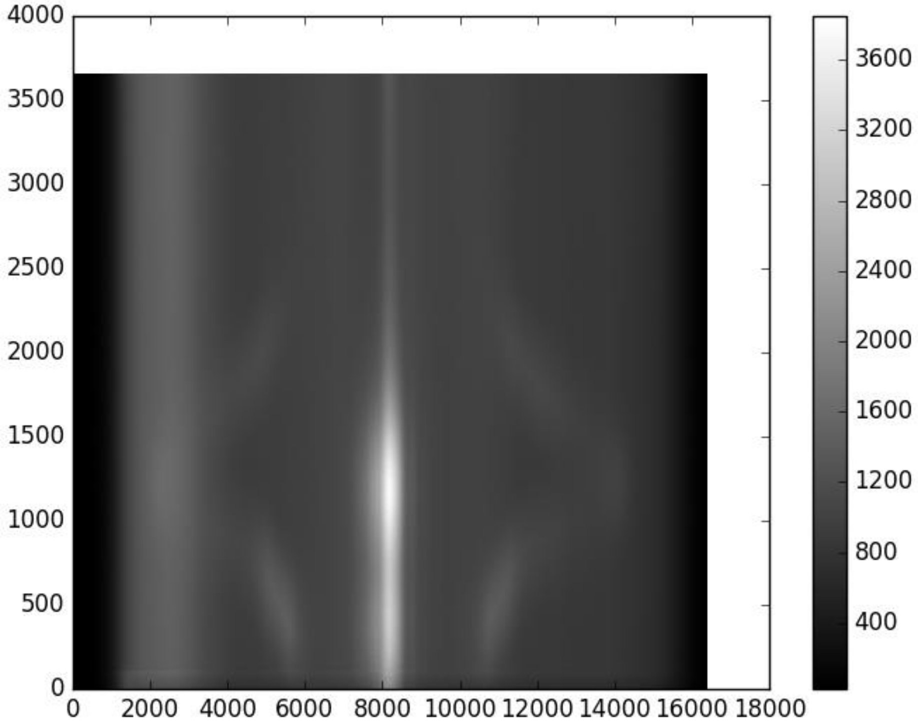


Figure 32: The figure show the 3D smoothing data in black and while by the Savitzky – Golay algorithm at all height of the package number 1452150273.

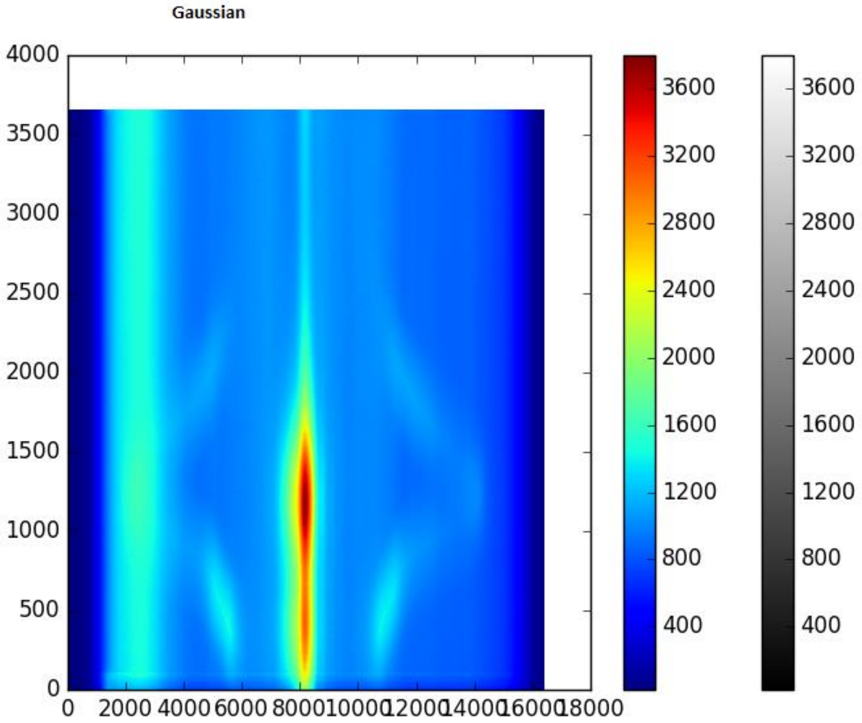


Figure 33: The figure show the 3D smoothing data in color by Savitzky – Golay algorithm at all height of the package number 1452105273.

From those figures, we can see that the 3D figure shows the frequency in the X-Axis and the Y-Axis show the height number which will cover from the function in the background section. By results comparing, we can see that this algorithm is a clearer picture than in the moving average algorithm and it also show clearly the peak of data.

7.2.1.3. Gaussian Filter algorithm

By apply the Gaussian filter algorithm in Python, we can see the results in 2D and 3D in the figures below:

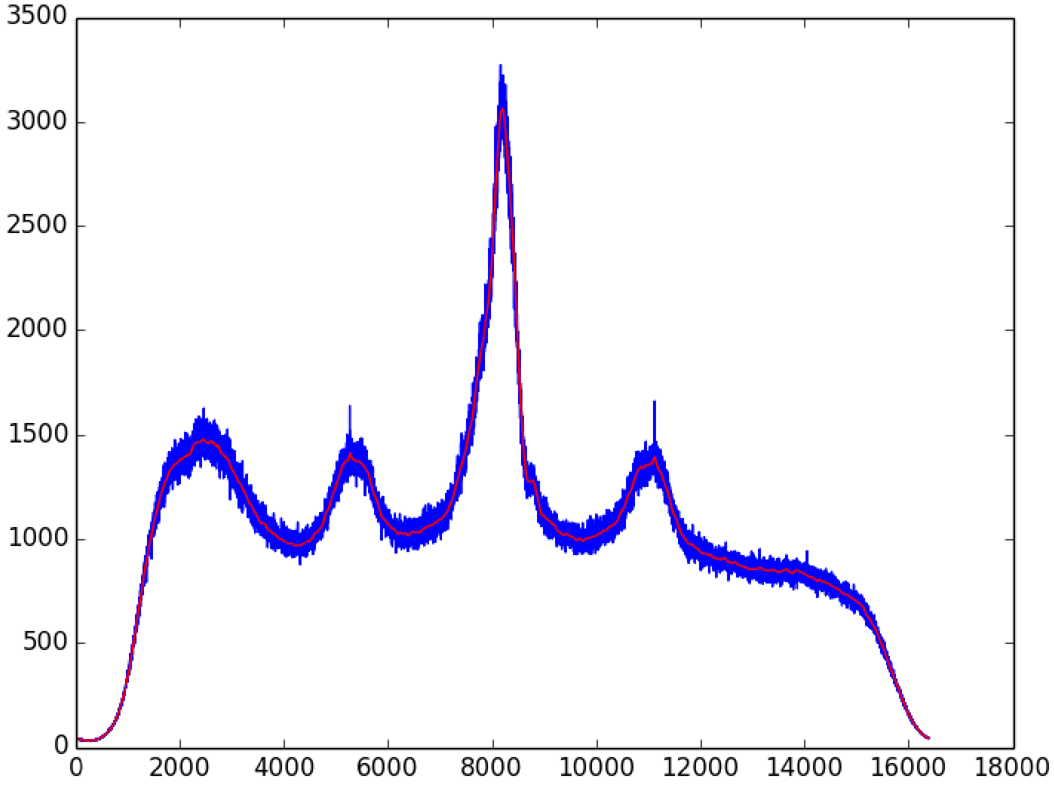


Figure 34: The figure show the 2D smoothing data by the Gaussian Filter algorithm at the height 550 (255 km) of the package number 1452150273.

From this figure, we can see that the X-Axis show the frequency of data and the Y-Axis show the power of data. From this picture, we can see that the green line shows the original data which contained noise. After smoothing this by the Gaussian algorithm, we get the red line which shows the smooth data.

Moreover, I also implement the Python code which available for plot the data in the 3D picture. In the two figures below, it is the 3D figures in both color and black and while figures.

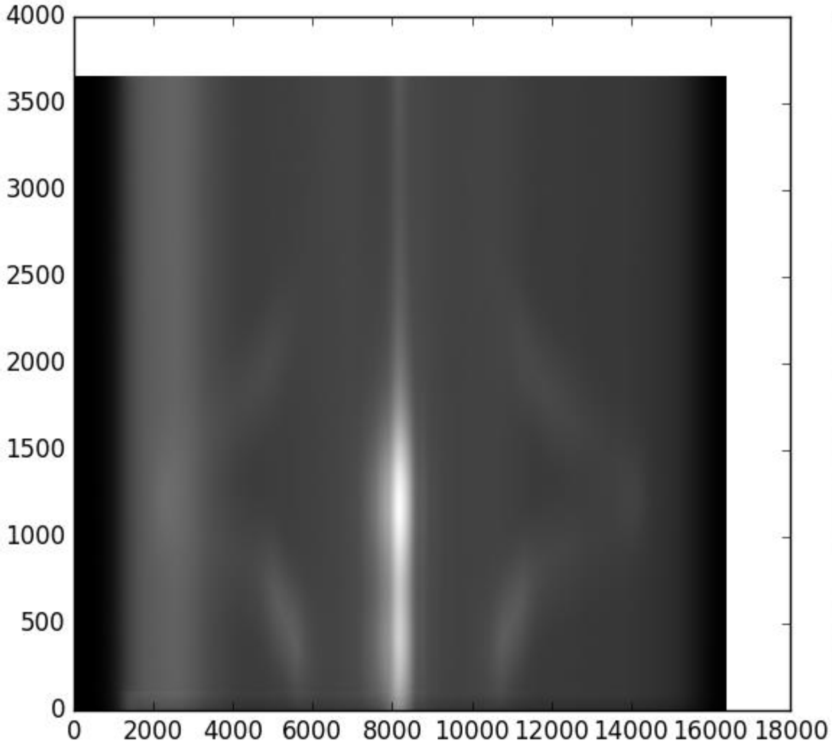


Figure 35: The figure show the 3D smoothing data in black and while by the Gaussian Filter algorithm at all height of the package number 1452150273.

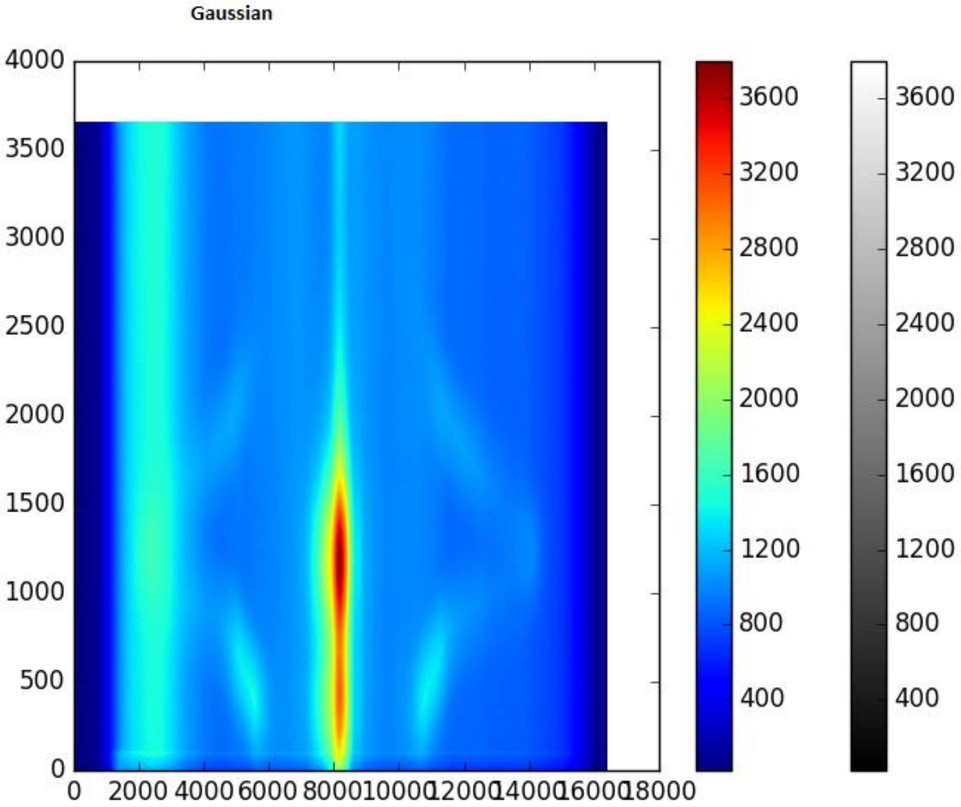


Figure 36: The figure show the 3D smoothing data in color by Savitzky – Golay algorithm at all height of the package number 1452105273.

From those figures, we can see that the 3D figure shows the frequency in the X-Axis and the Y-Axis show the height number, which will cover from the function in the background section. By results comparing, we can see that this algorithm is a clearer picture than in the moving average algorithm. However, we cannot know it is better than Gaussian Filter algorithm or not.

7.2.1.4. Least Square Method algorithm

By apply the least square method algorithm in Python code, we got the results in the figure in 2D and 3D in the figures below:

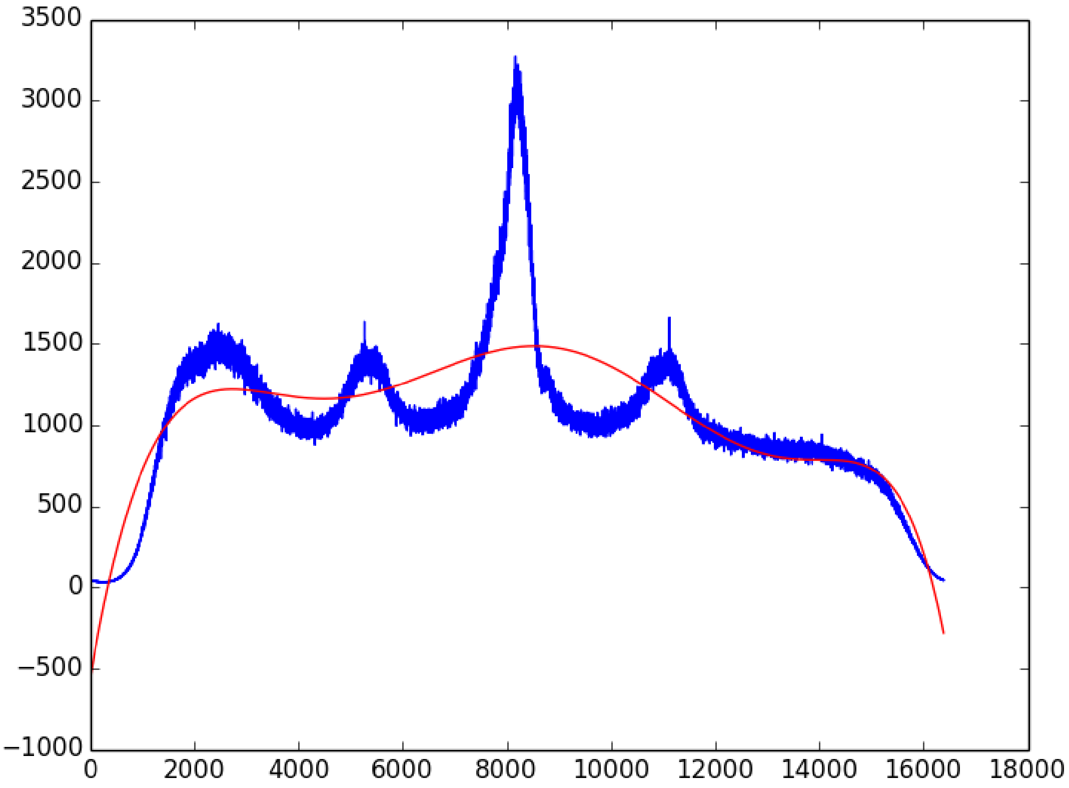


Figure 37: The figure show the 2D smoothing data by the Least Square Method algorithm at the height 550 (255 km) of the package number 1452150273.

From this figure, we can see that the X-Axis show the frequency of data and the Y-Axis show the power of data. From this picture, we can see that the green line shows the original data which contained noise. After smoothing this by the least square method algorithm, we get the red line which shows the smooth data. However, this is the most awful algorithm because it got the result less correctly than the last three algorithms.

Moreover, I also implement the Python code which available for plot the data in the 3D picture to east to compare with the last three algorithms. In the two figures below, it is the 3D figures in both color and black and while figures.

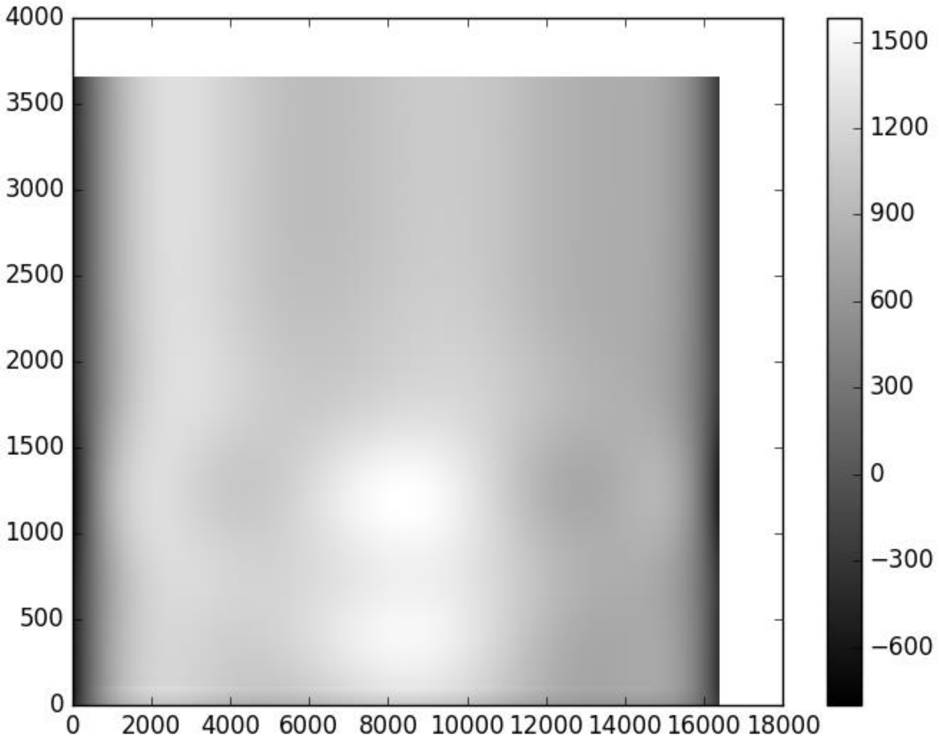


Figure 38: The figure show the 3D smoothing data in black and while by the Gaussian Filter algorithm at all height of the package number 1452150273.

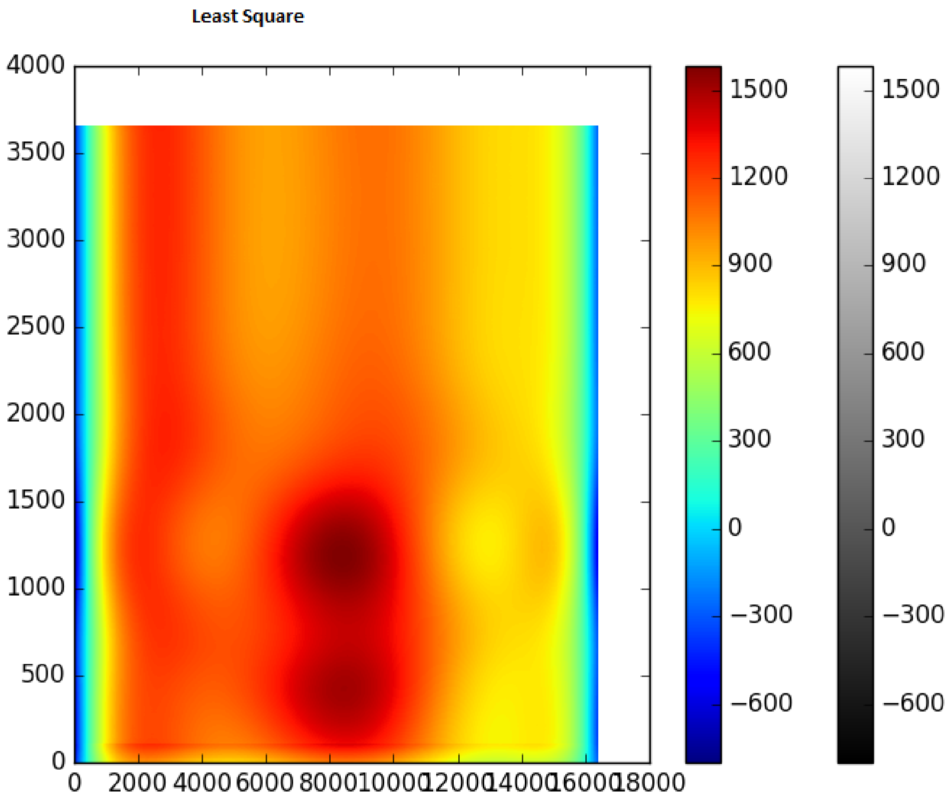


Figure 39: The figure show the 3D smoothing data in color by Savitzky – Golay algorithm at all height of the package number 1452105273.

From those figures, we can see that the 3D figure shows the frequency in the X-Axis and the Y-Axis show the height number, which will cover from the function in the background section. By results comparing, we can see that this is the very bad algorithm, it shows the unclear picture with the very noise data.

7.2.2. Find the peak

From those smoothing algorithms, I have applied the algorithm to find the peak. In the figures below, we can see the plus will show the finding local maximum peaks and minimum peaks of raw data. The red circle will show the local maximum peaks and minimum peaks of the smooth data.

The figure below is about the peaks finding based on the least square method smoothing algorithm.

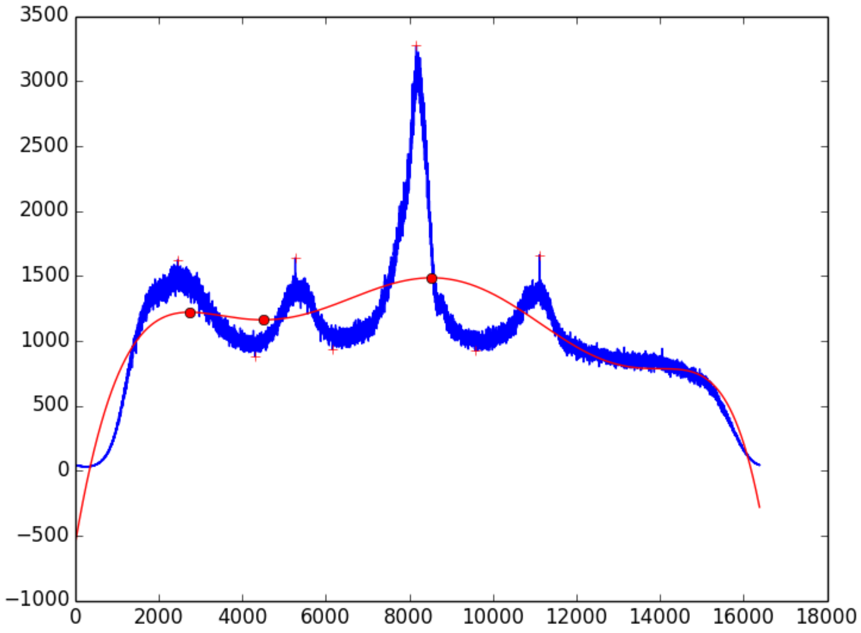


Figure 40: The figure show the peaks which found in both the smoothing data by Least Square Method algorithm and the raw data.

From this figure, we can see that this is not correctly algorithm. To apply to the raw data, it found four maximum peaks and three minimum peaks. However, when applied this finding peak algorithm, it just found total three peaks in the smoothing line and it looks like not a real peaks for maximum and minimum peaks.

To have more experiment data, I also applied to find the peaks in the raw data and smooth data by using the same finding peak algorithm but in the Gaussian filter algorithm. After applying it, I got the result. It will illustrate in the figure below:

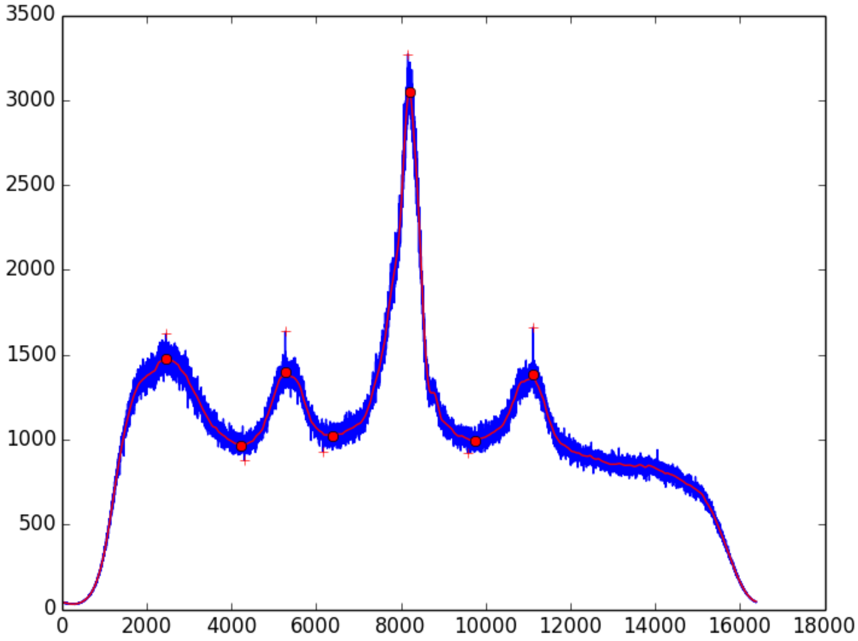


Figure 41: The figure show the peaks by applied the algorithm on the raw data and smoothing data by applied the Gaussian Filter algorithm.

From the figure above, we can see that is clearer in smoothing data. The number of maximum and minimum peaks is the same between the raw data and clean data. The changing in this figure has been just the power and frequency of the local maximum peaks and local minimum peaks. However, the gap between the peaks in raw data and the smooth data is not too large.

I also implement the Python code for plot the result in the moving average filter algorithm. After applying those algorithms, I got the figure below:

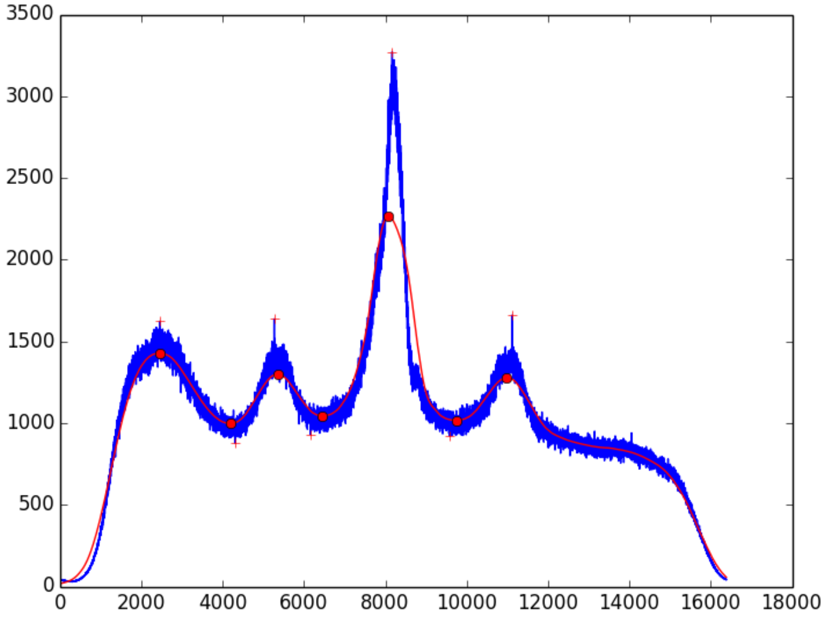


Figure 42: This figure show the peaks by applied the algorithm on both the raw data and smoothing data by applied the Moving Average algorithm.

From this figure, we also got the same number of local maximum peaks and local minimum peaks between the raw data and smoothing data. However, it looks like the gap between the peaks of raw data and smooth data is larger than the Gaussian Filter algorithm. It makes the frequency and power of peaks data between two data type is changed.

Finally, I applied the finding peak algorithm to find the local maximum peaks and local minimum peaks by using the Savitzky – Golay algorithm to smooth data. After applied, I got the figure below:

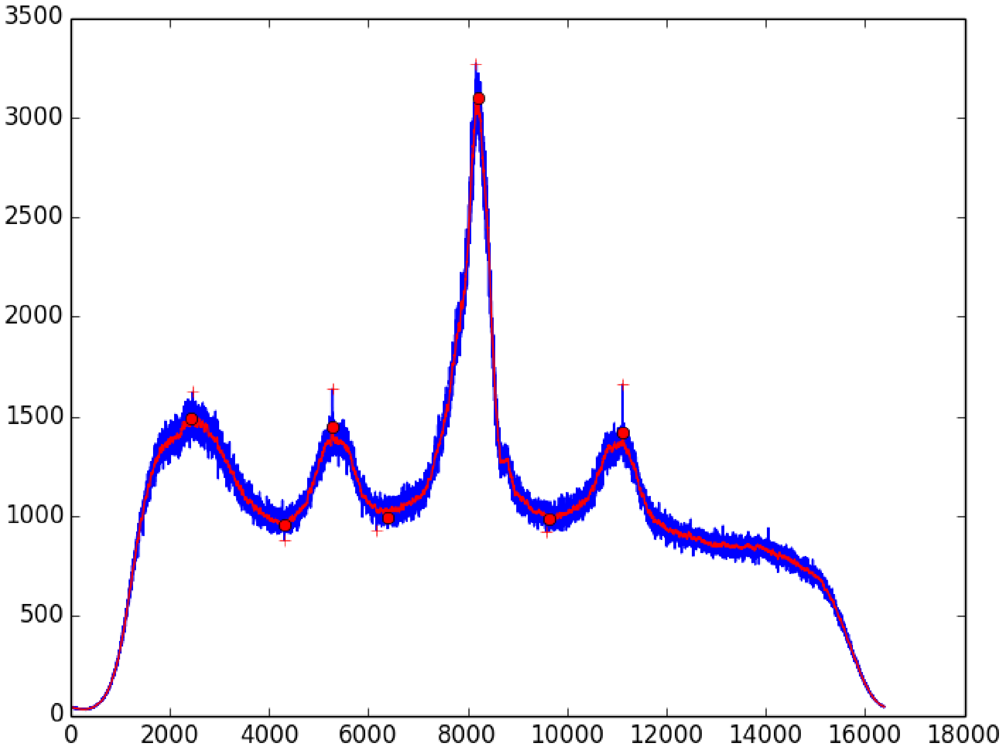


Figure 43: The figure show the peaks by applied the algorithm on both the raw data and the smooth data by the Savitzky – Golay algorithm.

From this figure, we can see that this is the clearest figure because the number of peaks between the raw and smooth data is the same. Moreover, the gap between the smooth data peaks and the raw data peaks is not too much. As we can see that, because this exactly, the frequency just has a little change while the power also has a big change.

7.3. Experiment 2

In this section, we will see the measurement of the time consuming in the coding by C++ language and apply the CUDA also. For measuring the time of the process, I measured 10 times, then I calculated the average to get the final time of the process.

From this, we can see that the algorithms for us to apply to the C++ language is matched filter algorithm. Below is the table for the results in time consuming by applying the C++ language and the CUDA technology for processing for one pair of data (in seconds).

Below are the results in time measurement when running on the Arctic Green Computing Group server system. We can see the result like that:

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Tine
Main process	92.87	93.06	93.41	92.89	93.07	93.02	92.85	93.48	92.94	93.50	93.109

Table 11. The time processing in seconds when apply the algorithm to run on the AGC computer server by CUDA and C++ Programming Language.

On the figure below, we can see the errors bar of this table:

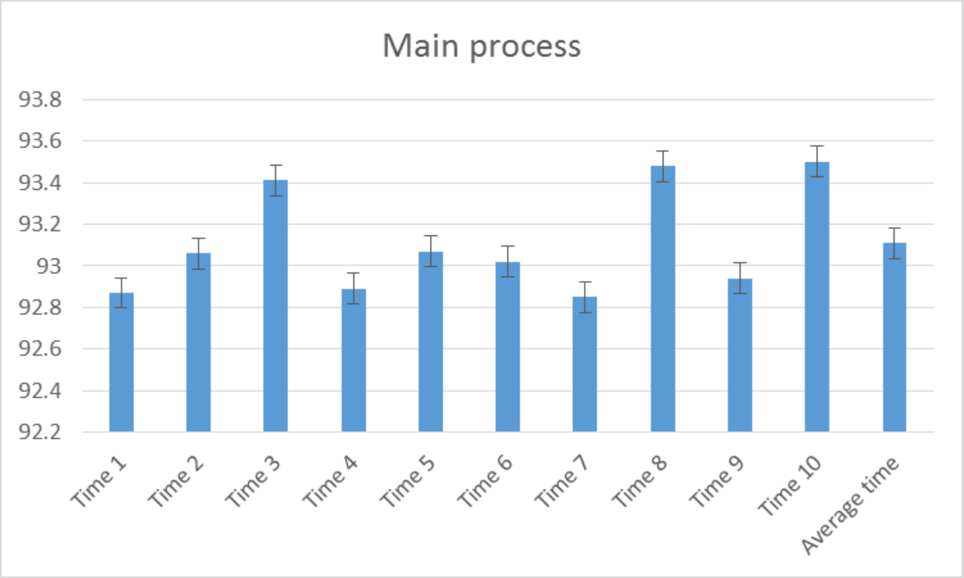


Figure 44. The error bar when applied the algorithm in C++ programing language time processing measurement on ACG Computer Server.

This part will describe a little about the results which got from Arecibo code. From Arecibo station, we have the function to load the raw data in 2D. However, in 3D figure, I had developed it by myself. The table below show the time measuring in both 2D and 3D in second's unit.

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Time
2D	0.64	0.549	0.564	0.601	0.618	0.646	0.676	0.765	0.764	0.744	0.657
3D	108.808	240.939	108.978	242.19	167.643	136.557	1014.099	238.804	395.215	742.447	339.586

Table 12. The table show the time measurement of the 2D and 3D data plot in raw data.

For more details, we can see the error bar figure below:

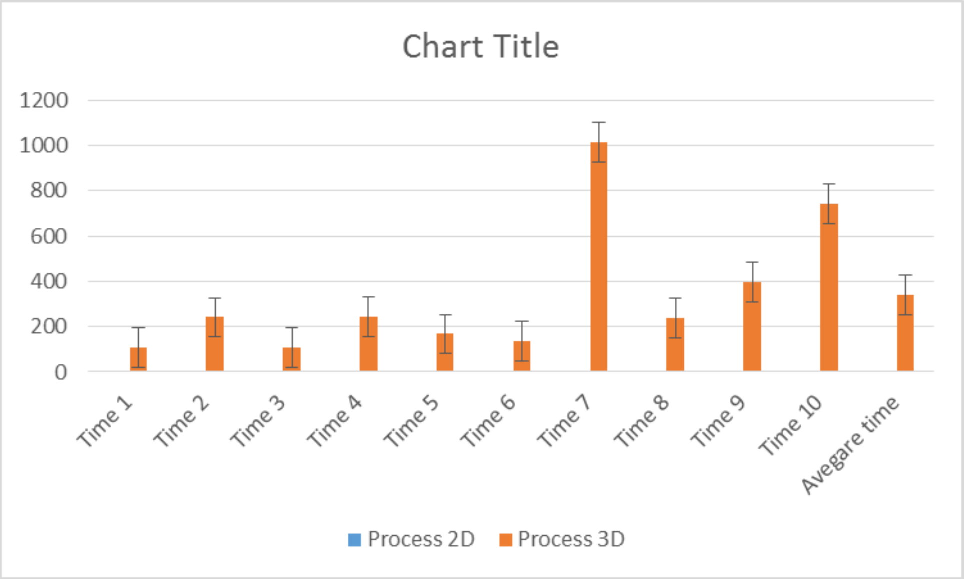


Figure 45. This figure shows the time measuring error bar by plot the data in 3D and 2D in raw data.

Moreover, in the tables below will describe about the time plotting for Python code in plot the data in 2D and 3D.

The table below will show the time plotting when applied the Least Square Method algorithm:

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Tine
2D	0.749	0.628	0.607	0.626	0.65	0.659	0.685	0.696	0.721	0.731	0.6752
3D	627.916	290.456	436.654	290.496	343.133	387.295	810.921	313.199	291.729	350.099	414.19

Table 13. The table of time in plotting the Least Square Method algorithm by Python Code.

The figure below will show the error bar of Least Square Method algorithm.

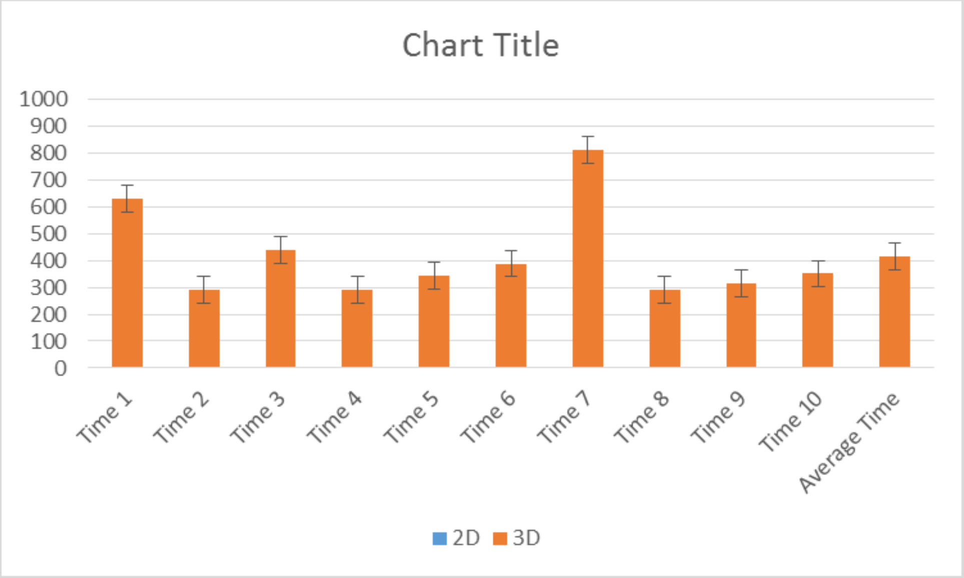


Figure 46. The figure show the error bar of the Least Square Method algorithm in time measurement.

For plotting data in 2D and 3D by the Moving Average algorithm. This table below will show the data

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Tine
2D	0.608	0.501	0.501	0.497	0.506	0.504	0.51	0.504	0.515	0.51	0.5156
3D	211.625	343.033	200.065	337.265	197.916	368.46	261.803	299.662	232.27	295.65	274.775

Table 14. The table show the time measurement by apply the Moving Average algorithm in Python code.

The figure below will show the error bar of the time measurement of Moving Average algorithm.

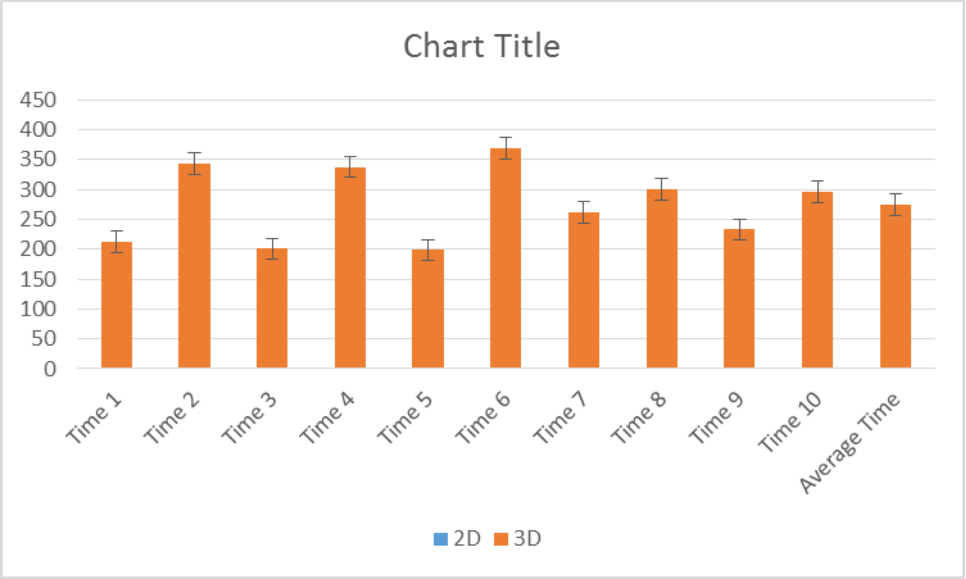


Figure 47. The figure show the error bar of the time measurement of Moving Average algorithm.

In addition, I also make the measurement of time 2D and 3D plot in the Gaussian Filter algorithm. The table below show the time measurement in second's unit.

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Tine
2D	0.677	0.555	0.548	0.56	0.578	0.597	0.614	0.642	0.658	0.692	0.612
3D	117.919	245.036	199.739	276.14	194.398	273.392	162.89	300.222	117.188	248.276	213.52

Table 15. The table show the time measurement of Gaussian Filter algorithm.

We will see more clearly in the error bar figure below:

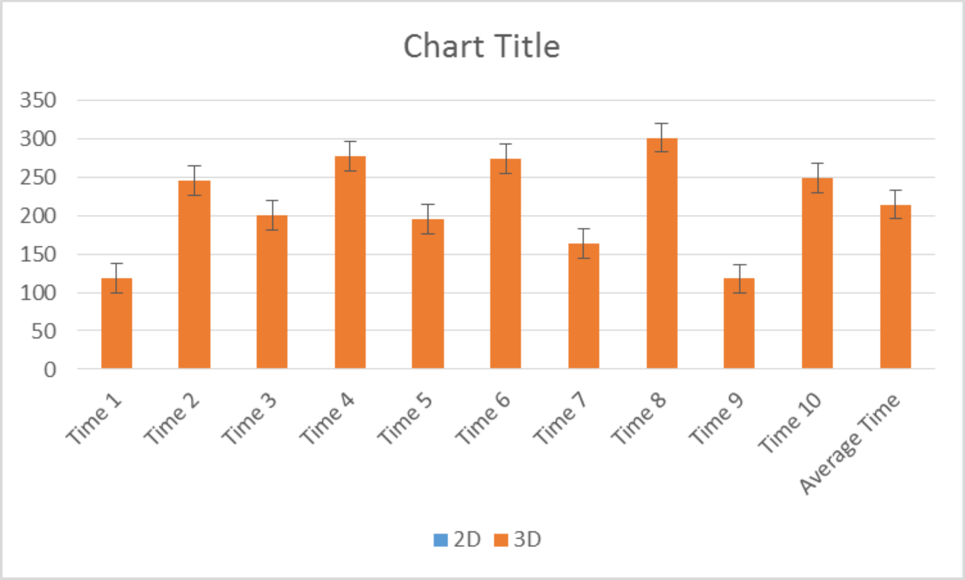


Figure 48. The figure show the Gaussian Filter error bar in term of time measurement.

Finally, I also measure the time in second's unit of the Savitzky – Golay algorithm. The table below will show the measurement data:

Process	Time 1	Time 2	Time 3	Time 4	Time 5	Time 6	Time 7	Time 8	Time 9	Time 10	Average Tine
2D	0.837	0.783	0.765	0.804	0.816	0.851	0.87	0.901	0.922	0.962	0.851
3D	191.122	271.801	202.969	243.942	697.051	111.635	242.672	155.961	245.167	956.834	327.915

Table 16. This table show the time measurement in 2D and 3D of the Savitzky – Golay algorithm.

For more details, we can see the figure in the error bar below:

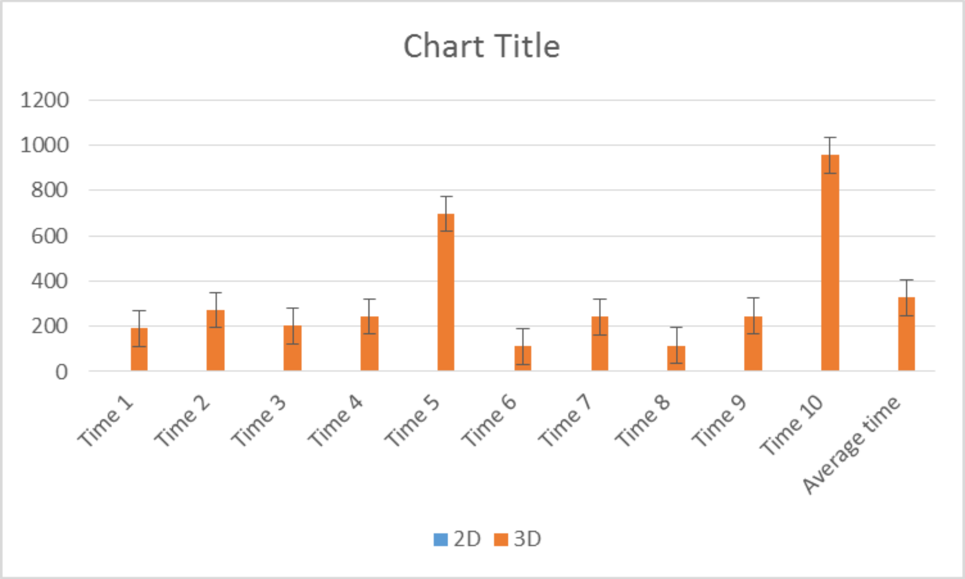


Figure 49. The figure show the error bar of time measurement in the Savitzky – Golay algorithm.

8. Discussion

8.1. Overview

This chapter will discuss about the design, implementation choice make, the comparison of results which got from the implementation in time measurement and the example in previous implementation, as well as discussing the key lessons learn through the process of the thesis. Moreover, the limitation and future work also discussed in this chapter.

Firstly, the thesis is based on the most physics topics, it did for me a lot of difficult to understand about the theories. In addition, the thesis topic is new, it has an advantage when I have investigated the way to solve problem but it also have disadvantage is that it is very few documents and previous work before in reference.

The programming language chooses is not a big problem with me. At the first time, we had discussed about that and earlier agree that we will choose the C++ programing language as the implement language. However, when I had received the example code from Arecibo Station in Python, we also have a branch to develop in Python for the algorithms code and example simulation code.

The design of the code in the backend components in the code is based on the code which received from Arecibo. In Python Code, it provided for the user various results in for comparing with the example results.

8.2. The results from Python Programing Language

This part will discuss about the result which we got from the Python code both from my code and the implementation code of the Arecibo.

By comparing the results which we got from the raw data in table 12 and the time measurement of the plotting data in the raw data and the Least Square Method implementation which show data in the Table 13. In overall, in the term floating data 2D and 3D, we can see that the plotting results time in the raw data is quicker than the Least Square Method. The average time to plot the 2D figure in the raw data is 0.657 seconds while the 2D figure in the Least Square Method took 0.6752, based on this number, we can see that the Least Square Method run slower than raw data plot around 2.732 %. In the term of the 3D figure, the raw data plot is occupied 339.586 seconds while the Least Square Method took 414.19 seconds. Based on this, the raw data are plot about 19.92 % quickly then Least Square Method. The figure below will show more clearly about the time difference between these two:

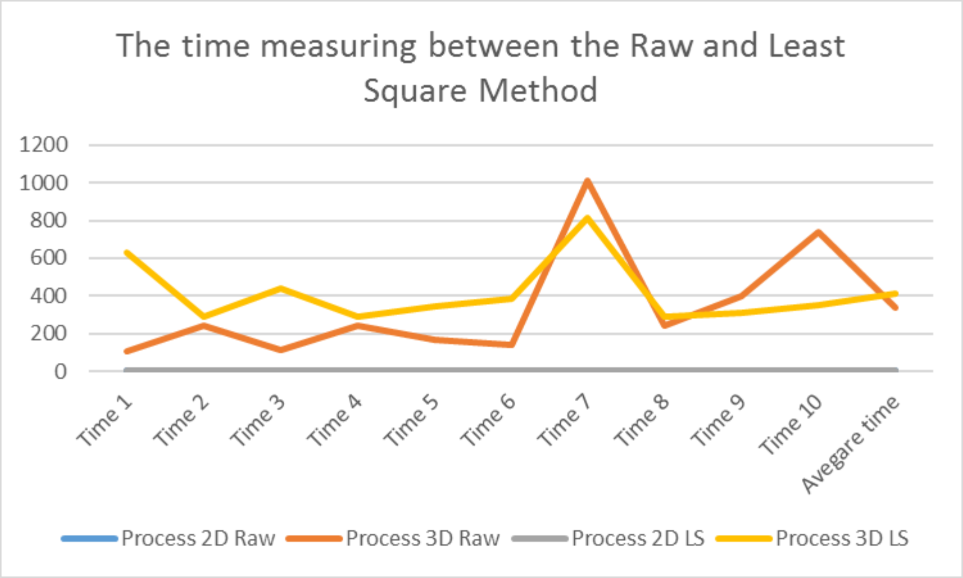


Figure 50. The figure show the time measuring between Least Square Method algorithm and the raw data plotting.

Next, we will compare the results which we got from the raw data in table 12 and the time measurement of the plotting data in the raw data and the Moving Average Algorithm implementation which show data in the Table 14. In overall, in the term floating data 2D and 3D, we can see that the plotting results time in the raw data is more slowly than the Moving Average Algorithm. The average time to plot the 2D figure in the raw data is 0.657 seconds while the 2D figure in the Moving Average Algorithm took 0.5156, based on this number, we can see that the raw plotting run slower than raw data plot around 24.12 %. In the term of the 3D figure, the raw data plot is occupied 339.586 seconds while the Moving Average Algorithm took 274.775 seconds. Based on this, the raw data are plot about 21.1 % slowly than Moving Average Algorithm. The figure below will show more clearly about the time difference between this two plotting time:

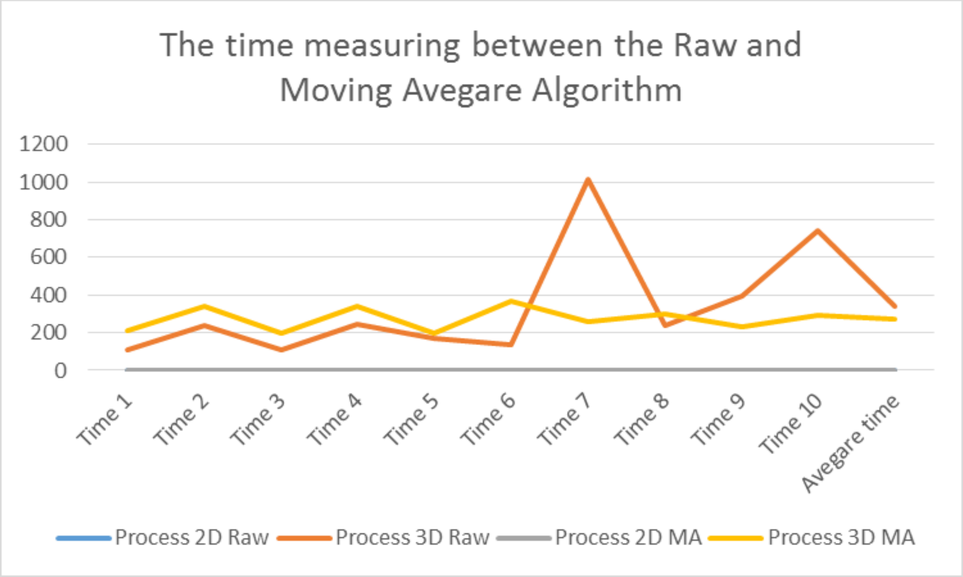


Figure 51. This figure show about the time comparing between the Moving Average Algorithm and raw data plotting time.

Next, we will compare the results which we got from the raw data in table 12 and the time measurement of the plotting data in the raw data and the Gaussian Filter Algorithm implementation which show data in the Table 15. In overall, in the term floating data 2D and 3D, we can see that the plotting results time in the raw data is more slowly than the Gaussian Filter Algorithm. The average time to plot the 2D figure in the raw data is 0.657 seconds while the 2D figure in the Gaussian Filter Algorithm took 0.612, based on this number, we can see that the raw plotting run slower than raw data plot around 7.09%. In the term of the 3D figure, the raw data plot is occupied 339.586 seconds while the Gaussian Filter Algorithm took 213.52 seconds. Based on this, the raw data is plotted about 45.58 % slowly than Gaussian Filter Algorithm. The figure below will show more clearly about the time difference between this two plotting time:

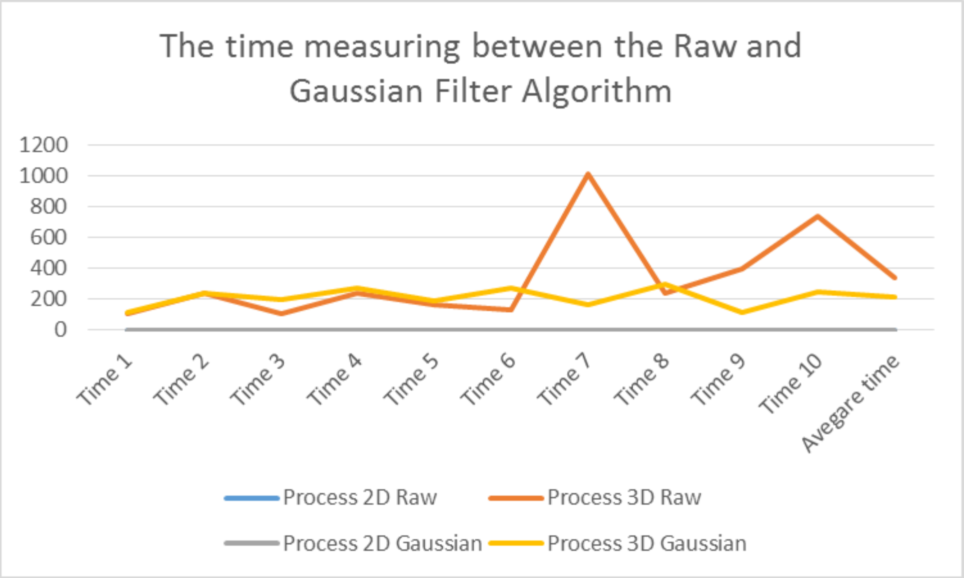


Figure 52. This figure show about the time comparing between the Gaussian Filter Algorithm and raw data plotting time.

Finally, we will compare the results which we got from the raw data in table 12 and the time measurement of the plotting data in the raw data and the Savitzky – Golay Algorithm implementation which show data in the Table 16. In overall, in the term floating data 2D and 3D, we can see that the plotting results in 3D time in the raw data is more slowly than the Savitzky - Golay Algorithm. However, the 2D time in the Savitzky – Golay algorithm is slowly than the raw plotting time. The average time to plot the 2D figure in the raw data is 0.657 seconds while the 2D figure in the Savitzky - Golay Algorithm took 0.851, based on this number, we can see that the raw plotting run faster than Savitzky – Golay Algorithm data plot around 25.73%. In the term of the 3D figure, the raw data plot is occupied 339.586 seconds while the Savitzky - Golay Algorithm took 327.915 seconds. Based on this, the raw data is plotted about 3.5 % slowly than Savitzky – Golay Algorithm. The figure below will show more clearly about the time difference between this two plotting time:

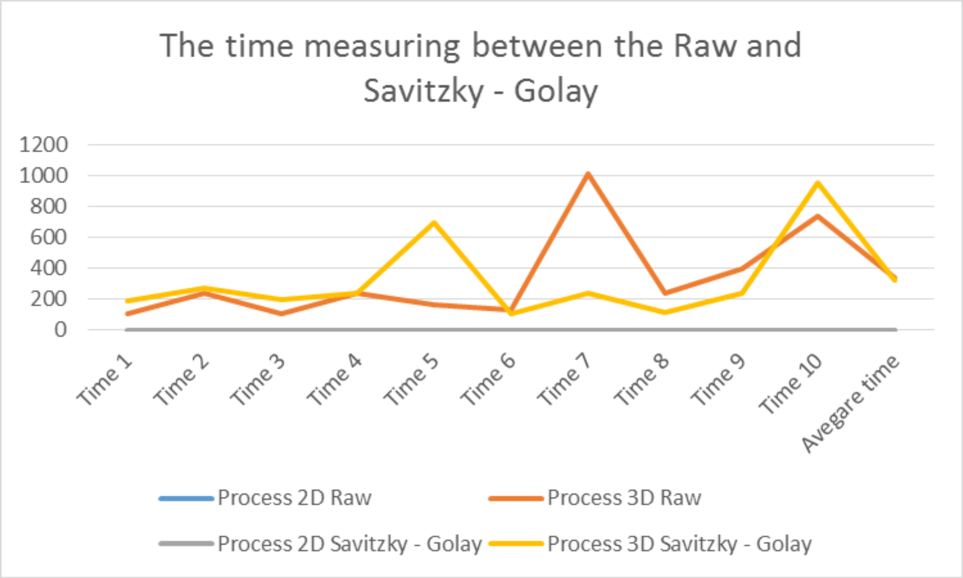


Figure 53. This figure show about the time comparing between the Gaussian Filter Algorithm and raw data plotting time.

By comparison between the four algorithms, we can see that the Moving Average algorithm has the time to plotting the 2D faster than other algorithms. However, Gaussian Filter algorithm has the plotting time in 3D faster than other algorithms. To comparing together, we can see the figure below to more clearly.

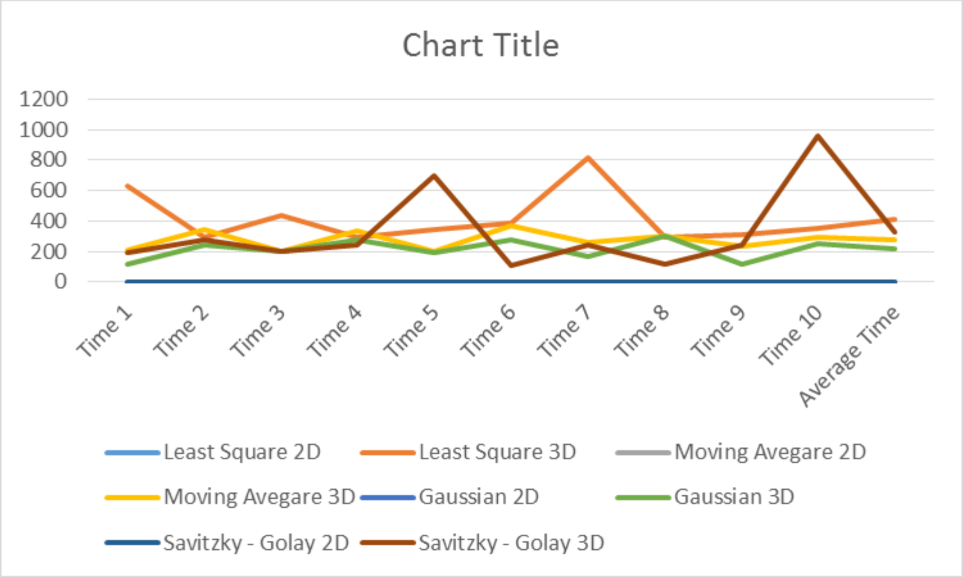


Figure 54. This figure show the time measurement in fours algorithms.

8.3. The results which got from C++ Programing Language

This part will discuss about the results of time consuming of CUDA and C++ programming language code. We will compare the results with the total time in running on the Python, which described in the section 6.2. Based on the results of time measurement in the table 11 and the table 2, we can see that the implement code of CUDA and C++ programming language is slower than the Python code. By comparison, we can see that the code in C++ consumes about 93.109 seconds to processing one pair of data while the Python just occupied 70.334 seconds to processing one pair of data. It is 27.869 % faster the C++ code. The figure below will show more details about the time consuming in every testing time and total in average time for both Python code and C++ code.

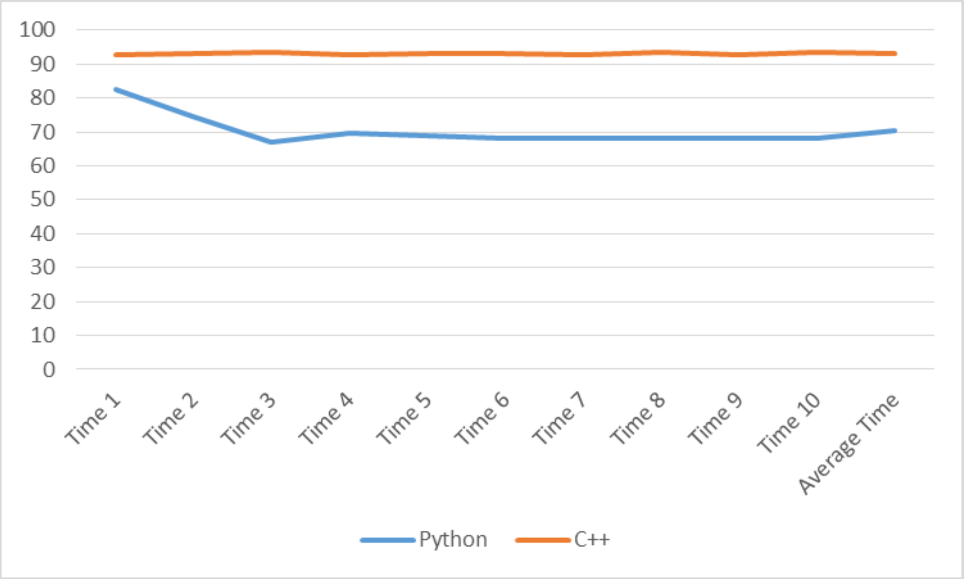


Figure 55. The figure show the time measurement of the Python and C++ code for processing one pair of data.

For the reason is that the Python also build the library which is convince to use. However, this library is not available in C++. I need to build it by myself in C++ and it makes the time consuming increase. In addition, when I applied the algorithm to CUDA, it is not supported in CUDA, one available for use is cuFFT to transfer and calculate in Discrete Fast Fourier Transform. It takes the most time in reading, writing and transfer the data into the GPU and transfer back. One of the reasons for that is the data to transfer is too big, even I applied the stream transfer.

8.4. Limitation

This part will discuss about the limitation of the thesis. There are a various problems in the first to start this thesis because of my background is in information technology. I spend too much time to learn about the algorithms and the theories in space physics. In addition, the topic of this thesis is new. It has less documents and various work for reference.

In the final, I decide to develop it in the C++ Programing Language. However, we received the code from Arecibo, it is writing in Python. I do not learn Python before, that why I need to spend my time to learn Python to understand their work. And it has two programing language which I used in this thesis, Python and C++.

For Python code, it is provided for the reader some figures which are in the picture for the reader to understand about the implementation of the algorithm. However, it is not for writing the big system.

In C++ Programing Language code, I had implemented C++, CUDA, and cuFFT. But in the final results it just provided for the end user the files in .hdf5 extension. From this results, the frontend user can use the Python script to plot the data in the final.

8.5. Key lession learned

Creating such a system has given to me a lot of experience for later project and in the future work. Some of the key lessons that are given to me is:

- The thesis is given to me a new knowledge in space physics
- In addition, it was given to me the knowledge of how to build a large system.
- It also given to me the experimentation in the term of implementation and testing.
- It has shown to me the experience in developing the plan for a long-term project.
- It also given to me the experience in the critical and how to evaluate this criticize to concordant me and the project.
- It has shown to me the experience of working in the group and how to communicate with the group member together for the best results.

8.6. Future work

Firstly, there are various limitations on this thesis, in the future work, this work should fix those limitations. In addition, the system did not provide to the frontend user any user interface, it will be very interested to the user if the system provided to the end user the interface for them to be easy to use.

Moreover, this thesis just work on the one node code. In future, it can be used in multi-node code by some various programing languages such as MPI (Message Passing Interface), UPC (Unified Parallel C) to connect computers together and sharing the tasks together to solve it. It engages to make the system work more quickly and more efficiently.

In addition, the system can future work by apply the Apache Spark or some big data solution to solve this problem.

9. Conclusion

This thesis presents the idea, architecture, design, implementation, and the results which got from the project, a system for processing the big data of plasma line by applying the GPGPU technique.

The system was designed to run on the large computer system which have a large data storage space to store the data, and have a high performance system to process the algorithms. One of the limitation of this thesis is that the system does not provide the user interface. Moreover, the speed of the system has not improve as well.

There are applied two programming languages to implement the code and the algorithms. Those programming language is Python and C++. There are the weakness and the bottlenecks in the experimentation, include in Python and C++. Some of these weaknesses were solved by other frameworks and libraries.

The most significant different technologies used in this thesis are to apply the GPGPU to solve the space physics big data problem. This thesis documents the idea and the development of the system, in addition to the experiment results, as well as present the related work which somebody done before.

The documented approach provides the valuable insight into building the big data system which carried a huge amount of data. The next step is applying other technologies to improve the performance and also implement the user interface for the frontend user easily to use.

References

[1] "What is radar?" [Online]. Available: http://www.njaudubon.org/SectionOases/Whatisradar.aspx. [Accessed: 06-Aug-2016].

[2] "Incoherent scatter," *Wikipedia, the free encyclopedia*. 09-Jan-2015.

[3] "THF09a ROBINSON LFW AMISR talk.pdf." .

[4] "Signals and Systems/Definition of Signals and Systems - Wikibooks, open books for an open world." [Online]. Available: https://en.wikibooks.org/wiki/Signals_and_Systems/Definition_of_Signals_and_Systems. [Accessed: 06-Aug-2016].

[5] "Plasma (physics)," *Wikipedia, the free encyclopedia*. 14-Mar-2016.

[6] ">Plasma line research." [Online]. Available: http://www.irf.se/upatm/old_020302/pll.html. [Accessed: 16-Mar-2016].

[7] "Qt (software)," *Wikipedia, the free encyclopedia*. 30-Jun-2016.

[8] "Overview | CMake." .

[9] "CMake," *Wikipedia, the free encyclopedia*. 21-Jul-2016.

[10] "Git," *Wikipedia, the free encyclopedia*. 05-Aug-2016.

[11] "Git." [Online]. Available: https://git-scm.com/. [Accessed: 06-Aug-2016].

[12] "Parallel Programming and Computing Platform | CUDA | NVIDIA | NVIDIA." [Online]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Accessed: 06-Aug-2016].

[13] "Hierarchical Data Format," *Wikipedia, the free encyclopedia*. 02-Jun-2016.

[14] "Python (programming language)," *Wikipedia, the free encyclopedia*. 09-Aug-2016.

[15] "cuFFT," *NVIDIA Developer*, 26-Jan-2012. [Online]. Available: https://developer.nvidia.com/cufft. [Accessed: 10-Aug-2016].

[16] C. Yang, Z. He, and W. Yu, "Comparison of public peak detection algorithms for MALDI mass spectrometry data analysis," *BMC Bioinformatics*, vol. 10, p. 4, Jan. 2009.

[17] *Curve fitting*. .

[18] P. Du, W. A. Kibbe, and S. M. Lin, "Improved peak detection in mass spectrum by incorporating continuous wavelet transform-based pattern matching," *Bioinformatics*, vol. 22, no. 17, pp. 2059–2065, Sep. 2006.

[19] "Find local maxima - MATLAB findpeaks - MathWorks Nordic." [Online]. Available: http://se.mathworks.com/help/signal/ref/findpeaks.html. [Accessed: 25-Apr-2016].

[20] "04_55_Calibration_rpt.pdf." .

[21] "thesis.pdf." .

[22] "Hilliard_Poster.pdf." .

[23] "miami1344024880.pdf." .

[24] "angeo-14-1462-1996.pdf." .

[25] "1-s2.0-0029554X80907909-main.pdf." .

[26] "paper.pdf." .

[27] "Digital topology_Introduction and survey_J.ComputerVision_1989.pdf." .

[28] "Filtering and Smoothing Data - MATLAB & Simulink - MathWorks Nordic." [Online]. Available: http://se.mathworks.com/help/curvefit/smoothing-data.html?s_tid=gn_loc_drop. [Accessed: 16-Mar-2016].

[29] "GPU Accelerated Computing with C and C++," *NVIDIA Developer*, 25-Nov-2013. [Online]. Available: https://developer.nvidia.com/how-to-cuda-c-cpp. [Accessed: 16-Mar-2016].

[30] "CUDA Toolkit Documentation." [Online]. Available: http://docs.nvidia.com/cuda/index.html#axzz3vbq83QlC. [Accessed: 16-Mar-2016].

[31] "CUDA C Programming Guide." [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#axzz3vbq83QlC. [Accessed: 16-Mar-2016].

[32] "Intro. to Signal Processing:Smoothing." [Online]. Available: https://terpconnect.umd.edu/~toh/spectrum/Smoothing.html#algorithms. [Accessed: 16-Mar-2016].

[33] "Digital signal processing," *Wikipedia, the free encyclopedia*. 18-Dec-2015.

[34] "Discrete Fourier transform," *Wikipedia, the free encyclopedia*. 15-Mar-2016.

[35] "Filter design," *Wikipedia, the free encyclopedia*. 09-Mar-2016.

[36] "Minimum phase," *Wikipedia, the free encyclopedia*. 06-May-2015.

[37] "Z-transform," *Wikipedia, the free encyclopedia*. 16-Mar-2016.

[38] "Goertzel algorithm," *Wikipedia, the free encyclopedia*. 22-Feb-2016.

[39] "Signal-Smoothing Algorithms." [Online]. Available: http://www.chem.uoa.gr/applets/appletsmooth/appl_smooth2.html. [Accessed: 16-Mar-2016].

[40] "numpy - how to smooth a curve in python - Stack Overflow." [Online]. Available: http://stackoverflow.com/questions/22988882/how-to-smooth-a-curve-in-python. [Accessed: 16-Mar-2016].

[41] "Signal Smoothing - MATLAB & Simulink Example - MathWorks Nordic." [Online]. Available: http://se.mathworks.com/help/signal/examples/signal-smoothing.html. [Accessed: 16-Mar-2016].

[42] "Peak detection in Python," *Gist*. [Online]. Available: https://gist.github.com/sixtenbe/1178136. [Accessed: 22-Mar-2016].

[43] "moble/MatchedFiltering," *GitHub*. [Online]. Available: https://github.com/moble/MatchedFiltering. [Accessed: 04-Apr-2016].

[44] "python code lorentzian - Tìm với Google." [Online]. Available: https://www.google.com.vn/#q=python+code+lorentzian. [Accessed: 04-Apr-2016].

[45] "How to access HDF5 data from Python - LCLS Data Analysis - SLAC Confluence." [Online]. Available: https://confluence.slac.stanford.edu/display/PSDM/How+to+access+HDF5+data+from+Python#HowtoaccessHDF5datafromPython-CheckiftheHDF5itemis"File","Group",or"Dataset." [Accessed: 08-Apr-2016].

[46] N. Nguyen, H. Huang, S. Oraintara, and A. Vo, "Mass spectrometry data processing using zero-crossing lines in multi-scale of Gaussian derivative wavelet," *Bioinformatics*, vol. 26, no. 18, pp. i659–i665, Sep. 2010.

[47] "Wavelet-Based Peak Detection - Online Technical Discussion Groups—Wolfram Community." [Online]. Available: http://community.wolfram.com/groups/-/m/t/91868. [Accessed: 08-Apr-2016].

[48] T. A. Kristian Hovde Liland, "Optimal Choice of Baseline Correction for Multivariate Calibration of Spectra," *Appl. Spectrosc.*, vol. 64, no. 9, pp. 1007–16, 2010.

[49] V. Agostini and M. Knaflitz, "An Algorithm for the Estimation of the Signal-To-Noise Ratio in Surface Myoelectric Signals Generated During Cyclic Movements," *IEEE Trans. Biomed. Eng.*, vol. 59, no. 1, pp. 219–225, Jan. 2012.

[50] N. Nguyen, H. Huang, S. Oraintara, and A. Vo, "Mass spectrometry data processing using zero-crossing lines in multi-scale of Gaussian derivative wavelet," *Bioinformatics*, vol. 26, no. 18, pp. i659–i665, Sep. 2010.

[51] "Fast Fourier Transform (FFT) - MATLAB & Simulink - MathWorks Nordic." [Online]. Available: http://se.mathworks.com/help/matlab/math/fast-fourier-transform-fft.html. [Accessed: 26-Apr-2016].

[52] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," *IEEE Trans. Acoust. Speech Signal Process.*, vol. 27, no. 1, pp. 13–18, Feb. 1979.

[53] Y. Zhu and C. Huang, “An Improved Median Filtering Algorithm for Image Noise Reduction,” *Phys. Procedia*, vol. 25, pp. 609–616, 2012.

[54] M. Al-Rawi and H. Karajeh, “Genetic algorithm matched filter optimization for automated detection of blood vessels from digital retinal images,” *Comput. Methods Programs Biomed.*, vol. 87, no. 3, pp. 248–253, Sep. 2007.

[55] “Lecture83.pdf.” .

[56] “cuFFT.” [Online]. Available: http://docs.nvidia.com/cuda/cufft/index.html#axzz49iokfgdF. [Accessed: 26-May-2016].

[57] “AndiH/CMake,” *GitHub*. [Online]. Available: https://github.com/AndiH/CMake. [Accessed: 27-May-2016].

[58] “Fast Fourier Transform (FFT).” [Online]. Available: http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html. [Accessed: 30-May-2016].

[59] “CUFFT_Library.pdf.” .

[60] “cudaMemcpy3D and cudaMalloc cudaDeviceToDevice copies - NVIDIA Developer Forums.” [Online]. Available: https://devtalk.nvidia.com/default/topic/465886/cudamemcpy3d-and-cudamalloc-cudadevicetodevice-copies/. [Accessed: 03-Jun-2016].

[61] “HDF5 File Creation.” [Online]. Available: http://beige.ucs.indiana.edu/I590/node122.html. [Accessed: 02-Jun-2016].

[62] “cuda-2d-profiling.pdf.” .

[63] “HDF5 Datatypes.” [Online]. Available: https://www.hdfgroup.org/HDF5/release/dttable.html. [Accessed: 05-Jun-2016].

[64] “CUDA C Programming Guide.” [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz4BSw2G7kO. [Accessed: 13-Jun-2016].

[65] “CUDA C Best Practices Guide.” [Online]. Available: http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#axzz4BSw2G7kO. [Accessed: 13-Jun-2016].

[66] “CUDA Education & Training,” *NVIDIA Developer*, 21-Mar-2011. [Online]. Available: https://developer.nvidia.com/cuda-education-training. [Accessed: 13-Jun-2016].

[67] “GPU Computing Webinars,” *NVIDIA Developer*, 20-May-2011. [Online]. Available: https://developer.nvidia.com/gpu-computing-webinars. [Accessed: 13-Jun-2016].

[68] “CUDA Toolkit Documentation.” [Online]. Available: http://docs.nvidia.com/cuda/#axzz4BSw2G7kO. [Accessed: 13-Jun-2016].

[69] “GPU Accelerated Computing with C and C++,” *NVIDIA Developer*, 25-Nov-2013. [Online]. Available: https://developer.nvidia.com/how-to-cuda-c-cpp. [Accessed: 13-Jun-2016].

[70] “Training Material and Code Samples,” *NVIDIA Developer*, 05-Jun-2013. [Online]. Available: https://developer.nvidia.com/cuda-education. [Accessed: 13-Jun-2016].

[71] “GPU Computing Webinars,” *NVIDIA Developer*, 20-May-2011. [Online]. Available: https://developer.nvidia.com/gpu-computing-webinars. [Accessed: 13-Jun-2016].

[72] “cuFFT and streams | Answers Well.” [Online]. Available: http://answerswell.com/question/4878663/p5i6c3/cuFFT-and-streams. [Accessed: 23-Jun-2016].

[73] “Tips for Writing Technical Papers.” [Online]. Available: http://cs.stanford.edu/people/widom/paper-writing.html. [Accessed: 28-Jun-2016].

[74] “CUDA Support/Measuring kernel runtime - CS Support Wiki.” [Online]. Available: https://www.cs.virginia.edu/~csadmin/wiki/index.php/CUDA_Support/Measuring_kernel_runtime. [Accessed: 28-Jun-2016].

[75] “How to measure time in NVIDA CUDA?,” *Ivan’s blog*, 09-May-2011. .

[76] “A Step-by-Step on How to Do a Background Study for a Thesis.” [Online]. Available: http://education.seattlepi.com/stepbystep-background-study-thesis-1626.html. [Accessed: 06-Jul-2016].

Appendices

Appendix A

Abbreviation

ISR: Incoherent Scatter Radars
EISCAT: the European Incoherent Scatter Radars Scientific Association
GPGPU: General – Purpose Computing on Graphics Processing Units
CUDA: Compute Unified Device Architecture
RADARS: Radio Detection And Ranging
CPU: Central Processing Unit
GPU: Graphics Processing Unit
2D: Two dimensions
3D. Three dimensions

Appendix B

Source Code

The source code is contained in the attachment CD.