Advanced GNSS-R Signals
Processing with GPU

Cervelló i Nogués, Oriol

Year 2018-2019

Supervisor: Albert Guillén
Adriano Camps

Advisor: Daniel Pascual

AUDIOVISUAL SYSTEMS ENGINEERING

Bachelor Thesis
Advanced GNSS-R Signals Processing with GPU

Oriol Cervelló i Nogués

BACHELOR THESIS UPF / 2019

Supervisors
Albert Guillen, UPF
Adriano Camps, UPC

Advisor
Daniel Pascual, UPC

Audiovisual Systems Engineering,
Department of Information and Communication Technologies
To my parents, Montserrat and Claudi.
To Cecilia and Mirlo.
Acknowledgements

In this section I would like to express my gratitude to all who supported and helped me in the development of this project.

First, I would like to thank Professor Adriano Camps. More than a year ago you received me and offered the opportunity to work in this project. Moreover you have been helping me and guiding during all the project as my co-supervisor.

Second, I would like to express my gratitude to Professor Albert Guillen. You helped me steering my academic career onto my interests although they were different from your expertise field. Also you helped me co-supervising my degree thesis.

Besides, I would like to specially thank Daniel Pascual. You have been my advisor all along my degree thesis and introduced me to the project and the lab. You constantly helped me with the everyday issues and revised all my work during more than a year. Also I would like to thank the MIR team member and co-worker Raul Onrubia for all his help.

Finally, I would like to thank for the founding to work in this project the Spanish Ministry of Education and AGAUR for granting me with the Collaboration grant, and the Professor Vanessa Daza and Professor Enric Peig who helped me with the procedures to make it possible.
Abstract

Global Navigation Satellite Systems (GNSS) are mainly used for positioning. Nonetheless part of those signals emitted towards the Earth are reflected and can be used for GNSS-Reflectometry (GNSS-R), a novel Earth Observation technique. The Remote Sensing Lab has developed the Microwave Interferometric Radiometer (MIR) in order to capture direct and reflected signals from GNSS constellations. In order to extract information form the signals, these ones need to be processed. In many cases, the first approach of processing them sequentially with common tools, such as Matlab, is not feasible due to the time cost. More specific alternatives have to be found in order to process the large amounts of data. This study requires the analysis of GNSS-R techniques as well as the analysis of the different options of implementation. This Degree Thesis is the development of a time-efficient Data Processing Unit (DPU) to process the raw signals from the MIR instrument.

Resum

El principal us dels sistemes de posicionament per satèl·lit (Global Navigation Satellite Systems, GNSS) és la navegació i localització. Part de les ones que emeten aquests satèl·lits reboten a la superfície de la Terra. Si es capten aquestes senyals es poden utilitzar per observar la Terra, aquesta tècnica en concret s’anomena GNSS-Reflectometry. El Remote Sensing Lab ha creat el Microwave Interferometric Radiometer (MIR) per captar la senyal directa i reflectida de senyals de satèl·lit GNSS. Per poder-ne extraure la informació de la terra, aquestes senyals primer s’han de processar. Es un procés complex i les opcions més típiques per processar dades com Matlab són inefficients ja que requereix massa temps. Per trobar un bon candidat per processar grans quantitats de dades és necessari realitzar un estudi de les tècniques utilitzades en GNSS-R i estudiar diferents opcions d’implementació. En aquest treball de fi de grau s’exposa el desenvolupament de un Data Processing Unit (DPU) per processar les dades del MIR.
Contents

Index of figures xi
Index of tables xii

1 INTRODUCTION 1
   1.1 Framework .............................................. 1
   1.2 Background and Motivation .......................... 2
   1.3 Objectives .............................................. 3
   1.4 Outline .................................................. 4

2 THEORETICAL BACKGROUND 5
   2.1 GNSS .................................................. 5
      2.1.1 Frequency bands ................................. 6
      2.1.2 Overview ......................................... 7
      2.1.3 PRN codes and CDMA ............................. 8
      2.1.4 Acquisition of the signal ....................... 9
      2.1.5 Tracking of the signal ........................... 10
   2.2 GNSS-R ................................................ 12
      2.2.1 Theory (DDM) .................................... 12
      2.2.2 GNSS-R Techniques: cGNSS-R and iGNSS-R ...... 15
      2.2.3 Applications ...................................... 17
      2.2.4 Missions and instruments ....................... 19

3 IMPLEMENTATION OF A GNSS-R PROCESSING UNIT 23
   3.1 Considerations ........................................ 23
      3.1.1 Navigation vs Reflectometry .................... 23
      3.1.2 Circular correlation with FFT ................... 24
      3.1.3 Implementation Options ........................... 25
   3.2 Overview of CUDA ...................................... 27
   3.3 Considerations for FFT-based correlations .......... 29
   3.4 Final Data Processing Unit ............................ 33
3.5 Overview of the parallel processing modes .......................... 37
3.5.1 Processing of cGNSS-R signals ................................. 37
3.5.2 Processing of iGNSS-R signals ................................. 40
3.5.3 DDM processing ...................................................... 41

4 RESULTS .............................................................................. 45
4.1 Time results ......................................................................... 45
4.1.1 cGNSS-R processing time analysis ................................. 45
4.1.2 iGNSS-R processing time analysis ................................. 48
4.1.3 DDM processing time analysis ........................................ 50
4.1.4 Overall time results ....................................................... 52
4.2 Memory results ................................................................. 54
4.3 CUDA, Matlab and C++ comparison ................................. 56

5 MIR AIRBORNE CAMPAIGN ............................................. 63
5.1 Soil moisture estimation ....................................................... 63
5.2 Water / Land transitions ..................................................... 66
5.3 Altimetry ................................................................. 71

6 CONCLUSIONS ................................................................. 73
6.1 Future work ............................................................. 74

BIBLIOGRAPHY .............................................................. 75
# List of Figures

2.1 Different GNSS constellations. ................................. 5  
2.2 GPS, GLONASS and Galileo navigation frequency bands. .... 6  
2.3 Components of GPS L1 C/A signal. ............................ 7  
2.4 Spread spectrum technique concept. ........................... 8  
2.5 PRN correlations examples. .................................... 9  
2.6 ACF of GPS C/A code signal for several Dopplers. .......... 11  
2.7 GNSS-R scenario. .............................................. 12  
2.8 Iso-delay and iso-Doppler lines. ............................... 13  
2.9 Specular/Diffuse scattering reflection correlation. .......... 14  
2.10 Delay-Doppler Map generation. ............................... 15  
2.11 Correlations from different iso-delays and iso-Doppler. .... 15  
2.12 cGNSS-R and iGNSS-R techniques. ........................... 16  
2.13 Comparison from cGNSS-R and iGNSS-R. .................... 17  
2.14 Airborne measurements. ...................................... 18  
2.15 Ice measurements over Antarctica. ........................... 19  
2.16 Summary of some GNSS-R instruments. .................... 21  
2.17 MIR instrument. .............................................. 22  

3.1 Giga Floating-Point Operations per Second for the CPU and GPU. 26  
3.2 Comparison of CPU and GPU hardware. ........................ 27  
3.3 Automatic scalability through different GPU architectures. .. 28  
3.4 Scheme of serial / parallel coding. ........................... 29  
3.5 Power of 2 and non power of 2 size comparison in FFT. ...... 30  
3.6 Diagram of implementation options. .......................... 31  
3.7 Options tests comparison. .................................... 32  
3.8 Correlation with and without overlap. ........................ 33  
3.9 Screen-shot of the configuration file. ....................... 34  
3.10 Screen-shot of the GUI. ..................................... 34  
3.11 Work-flow schema. ........................................... 36  
3.12 Scheme of the processing of cGNSS-R signals. ............. 38  
3.13 Cross-correlation function. .................................. 39  
3.14 Incoherent addition. ........................................ 40
3.15 Scheme of the interferometric processing. .................................. 41
3.16 Scheme of the processing DDM. .............................................. 42
3.17 DDM from real data. ............................................................. 43
4.1 Total elapsed time in conventional mode. ................................. 46
4.2 Elapsed functions time in cGNSS-R processing mode. ............... 47
4.3 Elapsed functions time in iGNSS-R processing mode. ............... 49
4.4 Total elapsed time in conventional mode. ................................. 50
4.5 Elapsed times for each function in DDM computations. ............... 51
4.6 Comparison times between different DDM configurations. .......... 52
4.7 Total elapsed time in conventional mode. ................................. 53
4.8 DPU elapsed time vs data processed. ....................................... 54
4.9 Functions elapsed times for DDM computations. ......................... 57
4.10 Elapsed time vs data comparing between different programing languages. ................................................................. 59
4.11 Functions time comparison between different programming languages. ................................................................. 60
4.12 Time evolution when changing the chunk size. ......................... 61
5.1 Uncal. peak power ratio on Yanco flights. ............................... 64
5.2 Data from Yanko Site 10 (Y10). .............................................. 65
5.3 Data from Yanko Site B5d (YB5D). ........................................... 66
5.4 Flight over Port Phillip Bay. ................................................... 67
5.5 Zooms of the flight over Port Phillip Bay. ................................ 68
5.6 Lake/land transition. ............................................................ 69
5.7 Sea/land transition. ............................................................. 70
5.8 Altimetry. .................................................................................. 72
List of Tables

3.1 Parameters of the configuration file. ........................................ 35

4.1 Percentage of GPU dedicated memory used in processing for cGNSS-R. Computed with NVIDIA GeForce 750M with 4 GB of memory. 55
4.2 Percentage of GPU dedicated memory used in processing for iGNSS-R. Computed with NVIDIA GeForce 750M with 4 GB of memory. 55
4.3 Percentage of GPU dedicated memory used in DDM computations using cGNSS-R technique. Computed with NVIDIA GeForce 750M with 4 GB of memory. 56
4.4 Percentage of GPU dedicated memory used in DDM computations using iGNSS-R technique. Computed with NVIDIA GeForce 750M with 4 GB of memory. 56
Chapter 1

INTRODUCTION

This chapter presents the introduction of this degree thesis. First, the framework of the project is stated. After that, there is a section for the background and motivation of the thesis, followed by the objectives of the project. The chapter ends with a summary of the main contents of each chapter of the thesis.

1.1 Framework

This degree thesis has been carried out at the passive microwave remote sensing group of the Remote Sensing, Antennas, Microwaves and Superconductivity group (CommSensLab) at the Department of Signal Theory and Communications (TSC) of the Universitat Politècnica de Catalunya - BarcelonaTech (UPC-BarcelonaTech). This research team has been one of the pioneers, and world reference in the study and development of theoretical concepts, instruments and mission in the field of GNSS-Reflectometry and Microwave Radiometry techniques used for earth observation.

One important contributions of the research team was the development of the 2D Microwave Imaging Radiometer with Aperture Synthesis (MIRAS) [1]. MIRAS is the instrument on-board the Soil Moisture and Ocean Salinity (SMOS) mission [2]. The team was involved in the mission and the developing of the payload since its conception in 1993. This instrument receives the radiation emitted by the Earth’s surface and it is used to generate brightness temperature images, which can be related to moisture content of the soil in land and to salinity in the surface of the ocean water. Many experiments have been done in soil moisture and ocean salinity retrieval before and after SMOS launch in 2009.
The Microwave Interferometric Reflectometer (MIR) instrument is another important development of the research team. Conceived to compare advanced GNSS-R techniques and to assess the capabilities of new signals and bands of GPS and Galileo systems. The MIR is a dual-band reflectometer with two arrays. An up-looking Right Hand Circular Polarisation (RHCP) one, to capture the direct signal of the navigation satellites, and a down-looking one to capture the Left Hand Circular Polarisation (LHCP) signals reflected over the Earth’s surface [4]. The MIR validation campaign took place from April to June 2018 in Melbourne, Australia.

This degree thesis is the continuation of the research of this latest experiment with the MIR instrument, where the 2 TB of data from the airborne campaign has to be processed.

1.2 Background and Motivation

This project borns from the need to process all the raw data, after the MIR airborne campaign is concluded. During the Australia campaign 2 TB of data were acquired in about 10 hours of flight.

Initially, the team did not had an efficient established way to process this huge amount of data. Some small data processing has been done with Matlab, as is a common programming language in academia. However, it is not feasible to process all the data in this way. In this context, the need to find a fast and efficient way to process the data appears.

The idea of the research team was to develop a processing unit to achieve the maximum possible efficiency. If possible, fast enough to be a real time processing unit for future campaigns. The main idea was to take advantage of parallel processing in Graphic Processing Units (GPU). Nowadays, they are becoming famous for being widely used to process large amounts of data. In the field of navigation using GNSS signals there are already some experiments that had tried to implement the GNSS receiver processing with GPUs as [5] or [6].
However in the field of remote sensing this implementations are still an exception. GPUs are already being used for image processing, as in [7]. For the computations of a GNSS-R receiver done with the raw signals there are some attempts to improve the processing, by trying to make CPU parallelisation processing as in [8] or [9]. However, CPU parallelisation is complex to implement. Only one experiment have tried to implement a GNSS-R receiver processing using GPUs as a cheap and fast hardware [10]. In this experiment only 2 parameters were extracted, the delay and the amplitude of the correlation peak, and was conducted for one constellation and one band (GPS L1), in order to compute altimetry.

The aim for our project is to follow a similar line of implementation as this last experiment reaching real-time speed. However, we are in need of further features. For the MIR processing campaign results, the whole waveform containing the peak is needed in order to extract all the possible observables of the reflection. This will allow not only perform altimetry, but to compute many other observations as soil moisture, sea surface roughness or sea surface salinity. In addition there is also the need to be able to compute for different constellations, at least GPS and Galileo, and for different bands, at least L1/E1 and L5/E5a, the ones acquired by the MIR instrument.

1.3 Objectives

The main objective of this degree thesis is to achieve an efficient system to process the data from the MIR instrument for Earth observation applications. This involves not only the implementation, but also the previous understanding of the GNSS signals and all the signal processing required behind to perform GNSS-Reflectometry, and the specific the requirements of the project. As well as the proper validation test and application to real data. The particular objectives of the degree thesis are listed below:

- Understand the complete theoretical background of GNSS signals and remote sensing, in particular the signal processing involved in GNSS - Reflectometry.

- Understand the requirements needed to process the data of the MIR instrument.

- Study the different processing methods and survey the different options of implementation in order to found the best suited for the project necessities, considering the own project limitations such as cost, time or resources.
• Design, implement, and test a real-time speed processing unit able to process signals from different constellations and for different bands satisfying the requirements of the MIR data.

• Design, implement, and test the processing unit in order to extract the maximum information of the correlation, the whole waveform containing the peak or DDMs, in order to be able to extract all the information desired from the signals.

• Begin the processing of the MIR airborne campaign data extracting the first results for different Earth observation applications.

1.4 Outline

The degree thesis has been divided in six chapters developing all the project, followed by the bibliography, list of publications. The outline of the degree thesis is the following:

• Chapter 1 introduces the thesis, explaining the framework in which it has been developed, the background in the field and the motivation and need for this project. It also states the objectives of this project. The chapter ends with the outline of the thesis.

• Chapter 2 presents all the theoretical background studied about GNSS signals, signal processing, remote sensing and GNSS-Reflectometry. This knowledge is required to understand the project, its need and to be able to continue the following steps of the project as designing, implementing or testing.

• Chapter 3 are the considerations taken into account for the implementation, the study of the different methods considered, and the testing of different options that lead us to the implementation of our final processing unit.

• Chapter 4 presents the different processing modes of our processing unit with the time and memory results of its testing, as well as a time comparison with other methods to check the advantage of our implementation.

• Chapter 5 are the first results of the MIR airborne campaign data processed.

• Chapter 6 states the conclusions of this degree thesis and future work of the project.

• Bibliography, Publications and Appendices.
Chapter 2

THEORETICAL BACKGROUND

2.1 GNSS

The Global Positioning System (GPS) deployed by the US was the first Global Navigation Satellite System (GNSS) to be operative. GNSS are constellations of satellites originally meant for positioning, navigation and timing. Among these constellations there are also other systems such as Russia’s GLONASS, EU’s Galileo, or China’s Beidou/COMPASS.

![Different GNSS constellations](image)

Figure 2.1: Different GNSS constellations (from [11, p. 4]).

The basic function of GNSS satellites is to broadcasts radio signals in two or more frequencies typically at L-band (1-2 GHz), the direct signals are used for navigation and positioning, as said, but they can also be used for other purposes as orbit determination, plate motion, crustal deformation, or geodynamics among others [11]. However, as we will see, the reflected signals can also be used in the following section GNSS-R.

In this section an overview of the theory behind the GNSS will be explained. All the GNSS constellations are based on the same principles.
2.1.1 Frequency bands

The GNSS constellations typically operate at the L-band (1-2 GHz). The allocation agreements were output from the World Radio Communication Conferences in 2000 and 2003 [12].

Each system normally operates in more than one sub-band. The aim of this is to compensate the ionospheric effects which can lead to errors, and be able to compute correctly the positions. With multiple bands crossing the ionosphere it is possible to correct this effect. The sub-band frequency correspond to the frequency of the carrier signal.

![Figure 2.2: GPS, GLONASS and Galileo navigational frequency bands (from [12]).](image)

There are two sub-bands reserved for the Aeronautical Radio Navigation Service (ARNS). These bands are reserved and no other users are allowed to transmit there or interfere with other signals. These ones correspond to the upper L-band (1559 - 1610 MHz), having the GPS L1, Galileo E1 and GLONASS G1, and to the bottom of the Lower L-Band (1151 - 1214 MHz) where GPS L5 and GLONASS G3 are located, with Galileo E5a and E5b coexisting in the same frequencies. Other sub-bands of these systems are allocated in Radio Navigation Satellite System (RNSS) frequencies, these are less restricted to use and are shared with ground radars, hence are more susceptible to interference.

Besides the ones mentioned, there are other countries and regional GNSS systems, and future projects of other GNSS currently in development.
2.1.2 Overview

The GNSS systems were conceived to have a low power spectral density in order to avoid interference with other microwave systems. To do so they were designed as spread spectrum systems. This implies that the spread signal occupies a much larger bandwidth than the one of the transmitted data [13, p. 23]. The transmitted navigation data is a bi-phase modulation with a symbol rate of 50 Hz. This navigation message contains information on the satellite ephemeris (satellite position and velocity), clock bias parameters and other complementary information. This one is mixed with a pseudo-random rectangular pulse train (PRN) at 1.023 MHz. [14]. In Figure 2.3, a sketch of both signals along with the carrier and the final result of combining all of them is shown.

Figure 2.3: Components of GPS L1 C/A signal (from [12]).

Figure 2.4 represents a simple diagram of the spread spectrum technique. The data $d(t)$ with a narrow bandwidth $B_d$ is spread by a PRN code (explained in the following section) $a(t)$ which has a much larger bandwidth $B_s$. This signal is sent through a channel with additive noise and interference. The receiver with a replica of the spreading code $a(t)$ is able to recover the original signal with some noise and interference error. In this figure, the carrier signal has been avoided for simplicity. All satellites use the same carrier, this technique is known as code division multiple access (CDMA). The key of this technique is that the PRN codes are unique for each satellite.
2.1.3 PRN codes and CDMA

The auto-correlation and cross-correlation characteristics of the PRN codes are similar to the Gaussian noise, though they are not random and can be regenerated, since they are deterministic. Each transmitter has its own PRN code and GNSS receivers have a local copy of them. Correlating them with the incoming signal allows to discriminate between them. This is the basis of the CDMA.

CDMA allows the receiver to multiplex several satellite signals onto the same frequency band. All the signals share the same band, but are modulated by a set of orthogonal (or near-orthogonal) codes, the PRN codes. In order to acquire an individual signal, the original code of that signal must be used to correlate with the received signal. Since all the signals use the same carrier frequency, the signals can interfere between them. This effect becomes more prominent if there is difference in the power level of the signals. All the signals should have approximately the same power levels at the receiver. [15, p. 71]

GNSS systems use mainly two types of PRN signals. The coarse acquisition (C/A), and the precision (P) codes. The C/A code is the one used for civilians applications. In GPS systems it operates at a chip\(^1\) rate of 1.023 MHz, with a main lobe bandwidth of 2.046 MHz. Each chip is about 977.5 ns (1/1023 MHz)

\(^1\)Each effective bit of the PRN code sequence is called a chip. They contain no information, as opposed to a data bit.
long. These codes have a period of 1 ms with a length of 1023 chips [15, p. 72]. P codes are not available to civilian users and they are encrypted. In contrast to C/A codes, P codes operate ten times faster. These ones have a chip rate of 10.23 MHz, and the main lobe in the spectrum is 20.46 MHz. The chip length is about 97.8 ns (1/10.23 MHz) [15, p. 71]. The code period is one week, so the acquisition of this code is more complicated and slower than the C/A is. However, since P codes have more bandwidth they offer much higher accuracy. There is also a third type of code, the modernised military (M) code, which is supposed to improve in performance and security of the P code. It is designed exclusively for military use and it is intended to replace the P code in the future [16, p. 150], meanwhile it will coexist with the other two.

![Figure 2.5: PRN correlations examples (from [17]).](image)

(a) Auto-correlation of C/A code 1  
(b) Cross-correlation of C/A codes 1 & 2

The receiver is able to distinguish the weak received signals due to the high auto-correlation peak in front of the cross-correlation peaks that PRN codes offer. In an idyllic case where the codes were orthogonal, the cross-correlation would be zero. However, as they are almost orthogonal, they have some very low levels of cross-correlation (possible values of -65/1023, -1/1.023 or 63/1.023) in comparison with the auto-correlation peak (1023/1023), see Figure 2.5. Only with the correct PRN code it is possible to find the auto-correlation peak.

### 2.1.4 Acquisition of the signal

The objective of a GNSS receiver is to find and track the peak of auto-correlation. When the signal arrives to the device a down-conversion is performed, still the signal may have a time delay offset, and a Doppler frequency shift. This makes the search of the peak a two-dimensional search process, in the time and frequency domains. For a stationary observer, the maximum frequency shift to expect is around ±5 kHz due to the motion of the satellite. For a high-speed vehicle it is
reasonable to assume a shift of ±10 kHz [15, p. 37].

In the past, a serial search technique was used. This consists of holding a frequency while shifting through all possible code offsets, then moving to the next frequency in order to find a peak. Nonetheless, this is a slow technique to detect a signal. Nowadays, techniques are often based on Fast Fourier Transform (FFT). Multiplication of signals in the frequency domain is equivalent to correlating signals in the time domain. Therefore, the circular correlation across all code offsets can be computed by taking the inverse Fourier transform [13, p. 33].

The steps for a FFT-based search with a known Doppler shift and a known PRN code are:

1. Multiply point by point the input data \( d(t) \) with the Doppler Frequency.
2. Perform the FFT on the base band data \( d(t) \) to convert them into frequency domain, \( D(k) \).
3. Perform the FFT of the local copy of the PRN code \( a(t) \), obtaining \( A(K) \).
4. Multiply point by point \( D(k) \) by the complex conjugate of the local copy \( A^*(K) \), to obtain \( R(K) \).
5. Transform the result into time domain by taking the inverse FFT of \( R(K) \) as \( r(t) \) and compute the absolute value of the \( |r(t)| \).
6. Find the maximum of \( |r(t)| \), giving the beginning point of C/A code.

If the PRN code is not known the process needs to be repeated changing the PRN code in step 3, one will return a high peak (high auto-correlation peak). In an ideal situation the results would be similar to Figure 2.5.

If the Doppler shift is not known, the process needs to be repeated changing the Doppler frequency in step 1. In the correct frequency shift, the peak will appear. In Figure 2.6 it is the square result of a two-dimensional search along different code offsets (time delay) and different frequency shifts. In the correct parameters (delay and Doppler shift) the auto-correlation peak appears.

### 2.1.5 Tracking of the signal

When the data is just processed coherently in time, the process is called coherent integration. Strong signals usually can be found from 1 ms of data with coherent
Figure 2.6: Auto-correlation function of GPS C/A code signal for several Dopplers (adapted from [18, p. 33]).

integration, but weak signals require longer data lengths and non-coherent integration. Sometimes a weak signal cannot be detected in 1 ms of data, and longer data records are needed to acquire it. The drawback of increasing the data length is that the process requires more operations, thus more time and resources. The way to process more data is through non-coherent integration. When few ms of data are being used, they are split in 1 ms blocks and each one is processed separately as explained previously. Finally, all the blocks are added together. Most of the noise which is random with a zero mean will cancel itself, while the auto-correlation peak which is present in all the blocks will be reinforced and a clearer peak will be obtained. Extending the integration period allows the detection of weaker signals, but it also increases the number of operations needed.

GPS receivers assign a dedicated channel to each signal being tracked. In order to ensure tracking of the signals in each processing channel, receivers are continuously estimating and correcting the offset code delay and Doppler frequency. With these parameters, receivers are able to synchronise the signal and demodulate the navigation message (modulated at 50 bps). Using the information of the code delay, the Doppler shift and the demodulated message, the receiver is able to determine pseudo-ranges to each satellite, and to compute a navigation solution [12].
2.2 GNSS-R

As mentioned, the GNSS reflected signals on Earth can be very useful. GNSS-Reflectometry (GNSS-R) is a passive remote sensing technique used to obtain information about the Earth surface with reflected GNSS signals. GNSS-R receiver concept is to capture (working as a bistatic radar [11, p. 175]) the direct and the reflected GNSS signals form a satellite. When processed, they can be used to extract a lot of information such as the ocean level and tides, wind speed and direction over the ocean, soil moisture and biomass, ice and snow thickness [11]. This provides an image the Earth’s surface environments playing a key role in atmospheric sounding, ocean remote sensing and land/hydrology mapping.

![GNSS-R scenario](image)

Figure 2.7: Example of a GNSS-R scenario. A GNSS-R receiver acquiring direct and reflected signals over the ocean from two satellites (from [8]).

2.2.1 Theory (DDM)

The main result from which the information will be extracted is the Delay-Doppler Map (DDM). This can be computed by the GNSS-R receiver in real-time or afterwards from the raw signals. The process to obtain a DDM is very much the same to the one used for GNSS explained at Section 2.1.4. It consists of cross-correlating the recorded signal from the down-looking antenna with the replica of the PRN code of the satellite for different time delays and frequency shifts [14]. However this case is more complex, as the signal is reflected on the Earth surface and thus more noisy.

When the GNSS signal reflects on the Earth’s surface, it is not just a single clean reflection, but involves the surface roughness and motion of transmitter and re-
ceeiver. It becomes a scattered signal with many reflections at different delays and Dopplers shifts. In the reflection surface (Earth’s Surface) the lines where the distance from the transmitter to the surface and to the receiver are the same are called the iso-delay or iso-range lines and have the characteristic that the reflected signal has the same travelling path length. Moreover the transmitter and receiver are in motion which originates some Doppler shifts. The iso-Doppler lines indicate where the reflected signal has a constant Doppler shift. If the Earth’s surface is locally approximated by a plane, they become hyperbolas [17, p. 479]. Figure 2.8 shows the iso-delay and ido-Doppler lines together and are centred at the specular direction point (0,0), where the antenna is pointing.

![Figure 2.8: Iso-delay and iso-Doppler lines overlaid (from [17]).](image)

The specular point is the first reflection to arrive at the receiver, as it has the shortest path from the transmitter. Other iso-delays have a longer path to travel. These contributions add power to the trailing edge of the peak. All the delays accumulate power and this distorts the obtained peak of auto-correlation. In top of Figure 2.9, there is a representation of a single specular reflection which results on a clear peak where the time delay can be obtained. In the bottom there is a sketch representation of a diffuse scattered reflection in the different iso-delays. The specular reflection is the first one to arrive, but all other reflections distort the resulting peak. The distortion with respect to the ideal auto-correlation peak gives information about the roughness of the surface and its origin (for example, wind speed and direction over the ocean) [11, p.184].
Figure 2.9: a) Resulting auto-correlation peak of a specular reflection from a C/A-like code. b) Conceptual sketch of of the same process with a diffuse scattering reflection (from [11]).

Figure 2.9 shows the correlation at the Doppler frequency of the specular reflection point. However, there are iso-Doppler lines with different frequency shifts. If the same correlation is done for different frequency shifts, different peaks for different time delays are obtained, generally with lower power levels and further delayed, as the frequency differs from the specular one, nonetheless they contain information and if the specular frequency is unknown may help to localise the peak. If all the correlations are put together the whole cross-correlation function is obtained, not only for time delays, but also for frequency shifts. The complete set of delay-Doppler correlations is called delay-Doppler map (DDM). On Figure 2.10 an example of a reflecting surface is shown with the iso-Doppler lines drawn above. On the right side of the Figure, different correlations of different iso-Doppler lines (frequency shifts) are shown (see also Figure 2.11), and the whole DDM as a two dimensional function. These are the same principles as the ones explained at Section 2.1.4 for the GPS (GNSS) in which the objective is to
track the auto-correlation peak for positioning. Here, the objective is not only to obtain the time delay and Doppler shift to track the peak, but to obtain the whole DDM which is the one that will provide the information on the Earth’s surface properties.

Figure 2.10: Sketch of how different iso-Doppler lines generate a two dimensional graphic of time delay vs frequency shift. This function is called delay-Doppler map (from [11]).

Figure 2.11: Correlations from different iso-delays and iso-Doppler (from [8]).

2.2.2 GNSS-R Techniques: cGNSS-R and iGNSS-R

The technique explained until now, correlating the reflected signal with a replica of the PRN code, is called the conventional GNSS-R (cGNSS-R). Besides, the interferometric GNSS-R (iGNSS-R) consists of capturing the reflected signal as in cGNSS-R, but the direct signal from the satellite as well, and performing the
correlation between these two signals, without any clean replica.

The comparison between both methods is not straightforward. In cGNSS-R, as explained, a locally-generated replica of the code is used to correlate with the down-looking antenna, therefore it has no noise. Moreover this allows us to separate between different satellites by their codes. Also for this method smaller antennas (less directive) can be used to track the reflected signals [17, p. 503]. The main drawback for this method is that only the public codes can be used, which implies a limited bandwidth and, therefore a limited resolution. Also, for this method of operation is needed to adjust the delay and Doppler frequency more often for a proper performance [19, 20]. For iGNSS-R, there is not need to know the code as the direct signal is used instead. This offers a much larger bandwidth, as the correlation is between the two signals with all the codes in it, not just the public ones. Encrypted codes which are in the signal with wider bandwidth are also being correlated without the need of knowing them, but offering a better resolution. In addition, this technique has smaller delay and Doppler shifts between signals and it becomes easier to track [19, 20]. On the other side iGNSS-R, requires a larger antenna size (more directivity) for the up-looking antenna, typically it has to be a multibeam antenna with beam-steering capabilities, and it is more susceptibility to radio frequency (RF) interference. iGNSS-R can also be performed with other types of signal such as satellite radio signals or satellite television signals or any other sources of opportunity [21]. Figure 2.12 shows the basic diagrams of a conventional and an interferometric GNSS-R instrument.

Figure 2.13 shows an example comparing the results of measuring wind speeds over the ocean with both methods. The peak amplitude (normalised) is maintained in both of them, however the slope of the leading edge of the iGNSS-R method is much steeper that of the cGNSS-R method, which in principle implies a higher resolution.
2.2.3 Applications

The most complete GNSS-R observable is the DDM one, as explained in Section 2.2.1. Also a single waveform peak, which is a cut of the DDM in the delay dimension already contains information for altimetry for example. The waveform peak amplitude, maximum value of the waveform also contains enough information to derive wind speed, soil moisture, or vegetation biomass. It is also used the ratio of peaks at two polarisation (LHCP and RHCP).

It is true that a fully dedicated instrument such as the ones used in active remote sensing (i.e. a radar) can achieve higher resolution and accuracy. However, the benefits of using GNSS-R are that it already disposes of world wide coverage thanks to GNSS constellations. Using this technique there is no need for developing or launching any transmitter, just a GNSS-R receiver is enough to acquire the data.

Ocean applications were the firsts applications of GNSS-R. Ocean wind retrieval was an experiment carried out with NASA’s DMR instrument [22]. Actually, also wind direction is feasible [23]. Extracting the altimeter information is one of the other applications [24], as in ESA’s PARIS mission concept.

Soil moisture are another type of possible measurements, what is being studied, thanks to the change in the dielectric constant, or in the surface’s reflectivity. There are also other techniques to measure vegetation parameters from reflections in the vegetation or the vegetation-ground. Figure 2.14 shows an example of
soil moisture detection in an airborne campaign over Australia. Higher reflectivity correspond to wetter soils (or water bodies), and lower reflectivity correspond to drier soils.

![Airborne measurements from GELOz campaigns, Yanco Region, NSW, Australia (September and October 2013) (from [17]).](image)

Cryospheric applications are a group of measurements on ice and snow. Different applications are possible such as ice thickness, ice permittivity and surface roughness, snow depth or dry snow substructure (measuring through different layers of snow even water below the snow, sub-glacial lakes). Figure 2.15 shows an example of GNSS-R measurements over ice and snow. A tower of 45 m was installed at Concordia station, in Dome C, Antarctica, with two antennas, a zenith looking one and a horizon looking to measure direct and reflected GNSS signals.

In addition to the ongoing studies in all these fields, future applications will involve the use of new digital technology and software-defined radio, new mitigation techniques and development of new lower-power GNSS-R receivers that can be boarded in constellations of nano-satellites, with the first ones already launched.
2.2.4 Missions and instruments

At present, GNSS-R is a field of study in which many agencies are getting involved. GNSS-R remote sensing looks promising, as it is a unique way to obtain measurements for Earth Observation. There are multiple modern space-borne missions operating. The following list shows some of them and a brief description of their current objectives (adapted from [14]):

- NASA’S Cyclone GNSS (CYGNSS) was launched on December 2016, and it consists of a constellation of eight satellites in an equatorial orbit, each carrying a delay-Doppler mapping receiver. CYGNSS’s primary mission goal is to sense ocean wind and wave conditions in the inner core of tropical cyclones to improve intensity forecasting.

- UK TechDemoSat-1 satellite was launched in 2014, and it tracks L1 C/A code reflections and generates delay Doppler maps of onboard, it also detects and maps reflections from the GPS L2 signals from space for the first time.

- ESA PARIS In-orbit Demonstrator (PARIS IoD) mission have tested the Passive Reflectometry and Interferometry System (PARIS) concept [25]. The satellite included a double-sided 1.1m high-gain antenna to intercept reflected GNSS signals and extract data on changing contours of Earth’s land, sea and ice surfaces.

- The ESA GEROS experiment on board the International Space Station stands for GNSS REflectometry, Radio Occultation and Scatterometry. It was
meant to be a scientific experiment dedicated to use signals from the currently available (GNSS) for GNSS-R and GNSS-RO of the Earth system to study the climate change.

3Cat-2 was a 3 x 2 CubeSat-based GNSS-R research/demonstration mission carried out by the Remote Sensing Lab and the NanoSat Lab at the Universitat Politècnica de Catalunya-Barcelona Tech. It was launched on August 15th, 2016.

3Cat-4 is 1U CubeSat design meant to demonstrate the suitability of this type of platforms to perform Earth Observation. The 3Cat-4 will fly a GNSS-R instrument capable of measuring several weather phenomenon, geographical characteristics and ocean parameters. The mission is being carried out by the Remote Sensing Lab and the NanoSat Lab at the Universitat Politècnica de Catalunya-Barcelona Tech and its launching programed for 2019.

FSSCat is an innovative mission concept consisting of two federated 6U Cubesats in support of the Copernicus Land and Marine Environment services. They carry a dual microwave payload (a GNSS-Reflectometer and a L-band radiometer with interference detection/mitigation), and a multispectral optical payload to measure soil moisture, ice extent, and ice thickness, and to detect melting ponds over ice. The mission is being carried out by the Remote Sensing Lab and the NanoSat Lab at the Universitat Politècnica de Catalunya-Barcelona Tech.

There are also many missions carried out airborne. Figure 2.16 lists of some of the existing instruments either space-borne, air-borne or ground based employed in GNSS-R and their main features.

**MIR instrument**

The Microwave Interferometric Reflectometer (MIR) instrument, Figure 2.17 is a new airborne instrument developed at Universitat Politècnica de Catalunya. It was built to assess the performance of a PAssive Reflectometry and Interferometry System-In Orbit Demonstrator (PARIS-IoD) like instrument and to compare the performance of different GNSS-R techniques and signals. The instrument is a multi-beam dual-band GNSS-Reflectometer with beam-steering capabilities.

The MIR instrument has two steerable arrays. Each array has 19 dual-band patch antennas hexagonally distributed. The up-looking one points directly to the GNSS
satellites, while the down-looking one points where the signals from these satellites are reflected on the Earth’s surface. Each of these arrays is connected to an analog beamformer that creates two beams at L1/E1 (1575.42 MHz) and two beams at L5/E5a (1176.45 MHz) [26], therefore the instrument can track up to 4 different satellites, and their corresponding specular reflection points. The signals to be sampled are down-converted and sampled at 32.736 MSps and 1 bit for both the I and Q components of the signal, and then stored in a computer for post-processing [26].
Summary

The important keypoints of this Chapter of theoretical background are:

- The basis of GNSS are the PRN codes and CDMA.
- GNSS reflected signals on Earth are the basis for GNSS-Reflectometry.
- There are two main techniques of GNSS-R: conventional (cGNSS-R) or interferometric (iGNSS-R).
- Applications such as altimetry, ocean wind retrieval, Soil moisture detection or cryospheric applications.
Chapter 3

IMPLEMENTATION OF A GNSS-R PROCESSING UNIT

3.1 Considerations

This section is focused in some relevant issues needed to be taken into account for developing a GNSS-R processing unit. These considerations are important to understand some of the aspects of the project.

3.1.1 Navigation vs Reflectometry

Despite the final purpose of GNSS-R is different from the GNSS, another noticeable difference that needs to be remarked is the amount of data that uses each of them. This subsection illustrates the huge increase of data in GNSS-R.

The GPS L1A operates with a bandwidth of 2.046 MHz. The navigation data within that bandwidth has in a speed rate of 50 bits/s, as said in the previous chapter, which is quite a slow rate. From it, three parameters are extracted for positioning: latitude, longitude and altitude. Assuming that they are floating number of 4 B each, the output speed will not be much higher than 12 B/s. Some high performance GNSS receivers can also send small messages in the order of kB/s.

On the other side, a GNSS-R system, specifically for the MIR instrument, has a bandwidth of 16.368 MHz (8 times larger than GNSS). Not focusing in all the computations, which will need to be done with all the spectrum, but focusing in the output rate. If 10 chips around the peak are saved for each repetition of the PRN code (repeats every millisecond), at a sampling frequency of 32.736 MHz, 320 samples are obtained every millisecond (the whole PRN would be 32736 sam-
amples). This becomes a 320 kSamples/s flux. If each sample is a complex number with a real and an imaginary part compound by a floating number of 4 bytes, results: 

$$320 \cdot 2 \cdot 4 = 2560 \text{kBytes/s}$$

or 2.5 MB/s. This is only counting all the processing in one frequency shift, but as explained, for the full DDM is necessary to look at different frequency shifts. If 10 Doppler frequency shifts for each PRN code are computed that makes a 25 MB/s. That is 26214400 B/s of output rate against the 12 B/s of the simple GNSS receiver, \(2.18 \cdot 10^6\) times more with only 10 Doppler cuts, but even more could be needed.

These values of course could change between instruments and experiments. In this example the values given, are the ones used in the MIR instrument which is the target data to process. The intention is to provide an illustrative example of the amount of data that are being dealt with, and was taken in consideration as an important issue before starting the implementation of the processing unit.

### 3.1.2 Circular correlation with FFT

The method explained in the theory section indicates that the processing is performed by a circular cross-correlation. The correlation is performed between the incoming signal and the PRN code replica. A correlation is very similar to the convolution of two functions. These operations are very complex and time consuming computations.

The Fast Fourier Transform (FFT) is a collection of efficient algorithms to implement the discrete Fourier transform (DFT). These algorithms manage to considerably reduce the complexity of the operations of the DFT. Taking advantage of the periodicity and symmetry of the DFT, the number of multiplications and additions can be drastically reduced. Meanwhile a DFT algorithm has an operations complexity of \(O(N^2)\), while a FFT algorithm gets the same results with an overall improvement of \(O(N \cdot \log(N))\) [27, p. 632]. The difference in speed can be enormous, especially for long data sets where \(N\) may be in the millions, as it is the case of the data-set from the MIR instrument.

When implementing the computation of a convolution/correlation, the efficiency of the FFT is so high that in many cases the most efficient procedure is to compute the Fourier transforms of the two sets of data to be convoluted, multiply them, and finally compute the inverse transform of the product of transforms [27, p. 630] instead of implementing the linear cross-correlation for each delay. Section 2.1.4 explained the acquisition of the correlation peak, and the method explained was a circular cross-correlation based on FFTs. After this consideration it is obvious why our processing unit is going to be implemented using FFTs methods.
3.1.3 Implementation Options

After considering the amount of data needed to be processed, the following consideration is how to process them. The most reasonable and plausible ideas for processing data includes a FPGA, a CPU program, multithread CPU program, GPU multithread program or a distributed system. Probably all of them in the correct way would be a good solution to process the MIR data, but some of them suit better than the others for this implementation, as it will be explained next.

Between the ones considered, the FPGA is the first one to be discarded due to its complexity of implementation as compared to the others. Also, because a FPGA is less configurable than a PC program. In addition, it is an expensive option, as it requires a performant FPGA. Similar to the FPGA, a distributed system implies high complexity in the development. It requires less specific hardware, but more amount, ending in another expensive method. Due to its complexity and cost these two methods have been discarded.

That reduces our options to use a single computer, which is much more appropriate. This idea allows to use hardware that the Remote Sensing Lab (UPC) already owns. Using a computer is a very configurable solution and permits to implement a CPU program, multithread CPU program or GPU multithread program. Another advantage is that a computer program is easy portable and easy to replicate between different computers, which makes it less hardware-dependent. In case there is the option, better hardware can be acquired in order to improve the processing time.

A generic computer program is a collection of instructions telling the machine what to do, generally in a logical order and interpreted by a CPU, one thread, one at a time. Nonetheless, sometimes calculation or processes can be computed simultaneously, with multiple threads, this is called parallel computing. This technique allows to divide large problems or large amounts of data in parallel threads executing them independently at the same time which allow to process much faster. An average home PC is able to parallelise tasks either in the CPU or the GPU. For our necessities, a multithread program is the best suited for the task.

The speed improvement depends on the number of cores. A CPU nowadays always have more than one core, but GPUs that are designed for video signals processing usually have hundreds of them. They include special vector instructions that support implementation of massive parallelism [5], and are able to process
both graphic and general data. A generic GPU of a home PC can already offer a really great improvement in computation and bandwidth. In Figure 3.1 there is a comparison between the performance of an Intel CPU and a NVIDIA card.

![Figure 3.1: GFLOPS (Giga Floating-Point Operations per Second) for the CPU (blue) and GPU (green) (adapted from [28]).](image)

Generally speaking, a CPU has less number of arithmetic logic units (ALUs) than a GPU, but more powerful and a larger cache with control logic. Whereas a GPU has a larger number of a more simple ALUs with little cache and simple control logic. However, the spare ALUs in the GPU makes the difference for parallel signal processing for operations such as multiplication, accumulation, discrete correlation and convolution, FFT, and more [5]. In Figure 3.2 there is a comparison in CPU/GPU hardware.

GPU parallel processing have become a trend in all computing fields due to its great improvement with simple programming and low cost. The most general open source language is Open Computing Language (Open CL), is supported by most of the GPU vendors. Compute Unified Device Architecture (CUDA) language is another freeware for parallel computing developed for NVIDIA graphic cards. One key point of the CUDA language for us is that features optimised signal processing libraries including efficient implementation of the FFT, called cuFFT which are widely used for its performance. Comparing both of them, CUDA offers a less complex programming language while presenting better performance, the constraint is that it only can be run in NVIDIA graphic cards. OpenCl is better if the application has to be a cross-platform program that has to be distributed in
Figure 3.2: Comparison of CPU and GPU hardware. GPU includes more ALUs for data processing (from [28]).

many different devices with different graphic cards [29].

Our availability of NVIDIA graphic cards in our personal and laboratory computers offers the opportunity to have a great parallel computing machine with a long-term tested and well working language as CUDA which will allow us to implement a fast correlator, with no initial cost. In case of investing for better performance the same program will be fully compatible with the new NVIDIA graphic card. This leads us to the decision of implementing the processing unit in CUDA.

### 3.2 Overview of CUDA

The CUDA language was introduced in November 2006 as a general purpose parallel computing platform and programming model to solve many complex computational problems. It is a GPU oriented computing API for NVIDIA cards that allows to distribute computations through the GPU. It is under a free-software licence, and it is designed to work with the programming languages C, C++, and Fortran.

In order to work with CUDA, it is necessary to understand the hierarchy of thread groups. To maximise the performance of CUDA, the problem has to be partitioned in sub-problems that can be solved independently in parallel threads. CUDA division is by blocks of threads. Each block contains a set amount of threads that will be solved simultaneously in a multiprocessor (A GPU is built around an array of Streaming Multiprocessors from the GPU (SMs), each SM includes different processors. An average GPU has between 10-20 SM and around 100 processors each.). The blocks are used to enable automatic scalability, each block of threads
can be scheduled on any of the available multiprocessors within a GPU, in any
order, concurrently or sequentially, so that a compiled CUDA program can ex-
cute on any number of multiprocessors, for any GPU. The threads in a block will
always be computed simultaneously, the blocks processed simultaneously will de-
pend on the GPU. In Figure 3.3 there is a CUDA problem divided in 8 blocks and
can be reordered to compute in a GPU of 2 multiprocessor, 2 blocks computed si-
multaneously, or in a GPU of 4 multiprocessors and compute 4 blocks each time.
This process is made automatically by CUDA. The user only has to divide the
problem in threads and decide the amount of threads contained by each block.
Normally each block can include up to 1024 threads.

![Figure 3.3: Automatic scalability through different GPU architectures (from [28]).](image)

The CUDA programming model assumes that the CUDA threads are executed on
a physically separated device that operates as a co-processor to the host running
the C program. The kernels, which are the functions that are executed in parallel,
execute on the device (GPU) and the rest of the C program executes on the
host (CPU) in a sequential single thread. Figure 3.4 shows a scheme of how the
program is run.
3.3 Considerations for FFT-based correlations

When we decided to start implementing the GNSS-R processing unit we found a dilemma. FFTs that are powers of two perform more efficiently that the ones that are not. The MIR data is sampled at a frequency of 32.736MHz, so a PRN code of a GPS signal which lasts 1 ms will have 32736 samples, which is not a power of two. The nearest one is 32768. The chunks of data that our program will use have to be the size of a PRN. So, we implemented a test and either FFT and IFFT of 32768 size are considerably faster than 32736, (Figure 3.5).

This originated the problem of which one of the two chunk sizes was better for the computing. However, as the program has other functions in addition to the FFT and IFFT, three different versions for each possible way to make the process were implemented. In Figure 3.6 there is a diagram of the three options.
Figure 3.5: Power of 2 (32768) and not power of 2 (32736) size comparison in FFT and IFFT. Test done in a NVIDIA GT 750M with cuFFT library.

- Option 1: works with chunk sizes of 32736 for the data, and with just one replica also of 32736.

- Option 2: works with chunks sizes of 32768 without overlapping the data, and extending the replica to 32768 by repeating it circularly. In each chunk of data the replica has to be recomputed by shifting it circularly to keep synchronisation with the data.

- Option 3: makes the chunks of data of 32768 catching the few samples of the beginning of the next PRN code by overlapping them 32 samples (1 chip), so the next chunk would begin correctly in the beginning of the PRN code and catch a few samples from the following one again. The replica is extended to 32768 with 32 samples from its beginning repeated without any need to recompute it for every chunk. The results will also be of 32768, but its time stamp are the corresponding to the original PRNs.

As an option it is also possible to make a re-sampling of the signal so each PRN has 32768 samples. However this option was not implemented as it would be much more time consuming.

After the tests, it was seen that Option 3 was the fastest one, Option 2 also had good execution times, but was more complex in the implementation, and having to compute a replica for each chunk of data, made it more time consuming. Option 1, as it is not power of 2, was slower in the computations. The whole time results for many different attempts can be seen in Figure 3.7. The only drawback of the
Figure 3.6: Diagram of implementation options. Chunks of data are placed horizontally as they are computed in parallel.
Figure 3.7: Tests comparison of implementation of the 3 Options. Tests done in a NVIDIA GT 750M.

Option 3 was that adding those 32 samples generate some artifacts, it is not noise as it is deterministic and is always the same but slightly changes the result (Figure 3.8). The correlation peak stays in the same location, but the amplitude decreases a bit deterministic with the delay. This effect can be corrected, or leave it as it is, if amplitude is not needed.

We continued developing from Option 3. However, soon enough, the amount of parameters needed for the computations became too big and a configuration file was implemented to write them down so it was easier to use the program and more configurable to allow the user set the parameters needed. One of the parameters was the FFT size because the MIR instrument also captures signals from the Galileo constellation and its PRN are 4 times longer than the GPS ones, 130944 samples. So the overlapping samples was set as another parameter because the nearest power of two is 13107, which corresponds to adding 128 samples (4 chips). At the end, the program ended being very configurable, and with the parameters that the user can set not only Option 3 is available, but also Option 1, setting no overlap and obtaining a cleaner results a bit slower.

The configuration file includes other information as the path of the data, the path of the results, if the computation is in conventional or interferometric mode, if different DDM has to be computed and in which resolution, among other parameters. The complete configuration file and a description of the whole processing unit is going to be explained in the following section.
3.4 Final Data Processing Unit

As seen in the previous sections the final Data Processing Unit (DPU) is a very configurable program. As summary our DPU has 4 way to process data. The first one is to process cGNSS-R signals obtaining waveforms as output. The second one is the same to process iGNSS-R signals. The third one is to compute DDMs as output for cGNSS-R signals and the fourth for iGNSS-R signals. To be able to set all the setting one of the most important things is the configuration file. This is a text file where the user introduces all the data on how to process the data and which data to process, Figure 3.9. The first thing the processing unit does when it is executed is to read this file to know what it has to do. This configuration file can be written raw if desired, but we also have developed a GUI to make it easier for the user set the settings, Figure 3.10.

The basic settings to be set for the program are the same in the configuration file than in the GUI. The GUI has more descriptions and some conditions that will warn the user in case of some expected mistakes. The parameters to be set are the ones listed on Table 3.1.

The DPU has been developed in CUDA featuring the cuFFT and NPP libraries. cuFFT are implementations of the FFT to be processed in the GPU and NPP is a library with signal processing functions (i.e. compute STD, find maximum...), also to be computed in the GPU. Also some functions have been written to be performed in the GPU. In the following Figure 3.11, a work-flow diagram of the program is shown, with a representation of its main functions.
Figure 3.9: Screen-shot of the configuration file for our processing unit.

Figure 3.10: Screen-shot of the GUI for our processing unit.
The main two stages of the DPU are a preparation stage which is sequential and only executed once at the beginning and a loop of processing data. In the preparation stage it starts by reading the configuration file. Once the DPU has information of the computations that is going to perform it reserves the memory space for those computations and allocates the variables. The final stage of preparation is to prepare the clean replica of the PRN code that is going to be used if this is operating to process cGNSS-R signals. These functions are executed in the host (CPU).

Each loop is going to process the data form a data-line from the configuration file. The information in each data-line is the one that is going to be parallelised in the GPU, i.e. if 1000 num of FFTs was set in the configuration file, for each loop 1000 chunks of data will be prepared from the path of the data-line. The data is distributed in threads, and grouped in blocks of threads for the GPU functions.
Figure 3.11: Work-flow schema of the Data Processing Unit.

The blocks are distributed along the GPU for parallel computing as explained in Section 3.2.

After reading the data to compute in that loop, the DPU prepare them to be processed. The data from the MIR instrument comes in bits and have to be masked and shifted to be transformed to floats. In case that a DDM are required also has to replicate the data for the different Doppler bins. If the DPU is operating to process for iGNSS-R mode, it prepares in the same way the data from the direct signal as it will not be using the clan replica of the PRN, otherwise these functions will be avoided. These functions are half green half purple because the reading part is executed in the host (CPU), but the preparation is already done in the device (GPU).

The following functions are the ones for the processing. All of them are computed in the GPU taking advantage of the parallel computing performance. Firstly the Doppler is applied. Then, the FFT is performed using the cuFFT library. Afterwards a function that performs the complex conjugate and complex multiply point by point the transform of both signals. Then the IFFT is performed to obtain the cross-correlation functions, again with the cuFFT library. The following step is
to perform the incoherent additions. Finally the peaks are found and prepared to be saved. In this final step the STD and Mean of the non-correlation region are computed. To find the maximum, STD and mean, the NPP library is used.

The final step of the loop is to write to disk the output parts to be saved, this part is done again in the host (CPU). Next the DPU is going to start again the loop with the data in the following data-line, and repeat all the loop. The DPU will terminate after processing all the data lines ordered. In the following Section 3.5 a detailed explanation of the parallel computing process for the different ways of processing are presented. In the Chapter 4 results of time and memory statistics are shown. In Chapter 5 results from real data of the MIR airborne campaign are presented.

### 3.5 Overview of the parallel processing modes

In the previous section, a general description of the DPU is presented. This Sections explains the different ways of processing of the DPU. The fist is to process to obtain waveforms from cGNSS-R signals. The second one is the same but for iGNSS-R. The third is how to compute DDMs, and can be computed either for cGNSS-R or iGNSS-R signals. All the processes are based on FFT methods and are computed in parallel in the GPU.

#### 3.5.1 Processing of cGNSS-R signals

Figure 3.12 helps to better understand the whole process for processing cGNSS-R signals. In this case the input data are the reflected signals from the Earth. There are different blocks marked with a hashtag and a number. Each of them represents a chunk of the continuous signal reflected on earth the size of a PRN. There are chunks from #1 to #n, and the computing of them is conducted at the same time for n blocks of the signal in each loop of processing. This value of n is the one set in the configuration file as number of FFT to process each time, and the data will be extracted from the path in the data-line. The maximum value of n will change in different GPUs as they will have different memory available, but to have an idea it could be 5000 with GPS data which corresponds to approximately 5 seconds of real data computed simultaneously for a 4 GB GPU. More information on the memory usage is presented in Section 4.2. When preparing the cGNSS-R data for processing the clean replica of the corresponding PRN being processed (the PRN in the top right of the Figure) is also going to be needed. In case it was
Figure 3.12: Scheme of the processing of cGNSS-R signals implemented in our DPU.

not known it would be necessary to try for different PRN codes first. For the MIR instrument data it is known as it is provided by an auxiliary GNSS that provides the list of satellites in view.

The first step to perform with the input data is to complex multiply it by a complex sine with the correct Doppler frequency shift in order to correct it. It could change from different chunks of data, but the phase is maintained along all the signal, even in the following data-lines. Then comes the correlating part based on the FFT, first the FFT is performed simultaneously for all the data and the replica. Then, each chunk from the data is complex multiplied point by point with the complex conjugate of the local replica, and finally the inverse FFT is performed. With this, the circular cross-correlation functions are obtained.

The cross-correlation functions, like the ones in Figure 3.13 processed from the MIR instrument can be very noisy. As explained in Section 2.1.5, incoherent averaging helps to reduce the noise and reinforce the peak. The addition is done with
Figure 3.13: Cross-correlation function of 50 consecutive chunks the size of a PRN, showing 50 peaks, as processed by our CUDA implementation from real data obtained with the MIR instrument.

the squared absolute value of the cross-correlation function. In Figure 3.12, the incoherent averages are shown between two chunks as example, but in the program any number can be configured, or even to do not perform any and search the peak directly in the cross-correlation functions. In Figure 3.14a, the same peaks as in Figure 3.13 are shown, but cropped to the 5 chips around the peak. In this case the peak is obvious, but they are a bit noisy, when the incoherent sum of the 50 peaks is done (Figure 3.14b), it is obvious how the peak is reinforced and the noise smoothed. There are cases in which in one coherent integration time it is not possible to distinguish the peak from the noise, and incoherent averaging becomes completely necessary.

Once the incoherent averaging is performed, the position of the maximum (peak) has to be found. With the location of the peak found in the incoherent averaging, the program goes back to the cross-correlation functions and save the waveform in that same location, in order to save the original peak. Each incoherent averaging affects on the location of those which have been added. The output is going to be the peak and surroundings of the coherent time. As said in Section 3.4, in the configuration file there is a field to specify how many samples around the peak are saved. These samples around the peak are needed to study the SNR and the eventual spread of the main triangle. This is useful to reduce the amount of output data,
Figure 3.14: Incoherent addition from the addition of 50 coherent peaks. Processed by our implementation in CUDA from real data of the MIR instrument.

if all the signal were saved would be very time and memory consuming, while in this way only the part that interest is saved. The rest of the cross-correlation function that do not contain the peak is used to compute a STD and mean of that part of the signal and then discarded.

### 3.5.2 Processing of iGNSS-R signals

As explained in Section 2.2.2 the interferometric technique is very similar to the conventional one. However, instead of correlating with a locally-generated replica of the PRN code the correlation is done with the direct signal captured by an up-looking antenna. Figure 3.15 shows the parallel processing for interferometric signals of our DPU. For the input reflected signals it is exactly the same as when processing for cGNSS-R mode. The difference is that there is no local replica, instead there are more input data which are the direct signals, painted in the background. The Doppler frequency in this case is only necessary to be applied to one of the signals not both, and is not Doppler frequency which is applied (because both signals will have it), but the relative Doppler between both of the signals, which from airborne or ground scenarios is almost zero. Afterwards the FFT is computed on both signals, and one is complex conjugated and complex multiplied point by point with the other. Then the cross-spectra of the cross-correlation functions is obtained and the following part is finding the peak, the same process as in the conventional method explained in the previous subsection.
3.5.3 DDM processing

Computing DDMs is a bit more tricky than just the waveforms. In this case Figure 3.16 is followed. The input data is in top of the figure. The first difference is that this time the input data is replicated multiple times, in case of the Figure three times. The data is replicated to apply different Doppler shifts on them. The first column applies the Doppler frequency minus a step shift to the first replica of data. The central column increases the step in one so is only applying the Doppler frequency in this case, $f - 1s + 1s = f + 0s = f$. The third column is again increasing the step of the shift one more time. It is made in the way that the Doppler frequency always stays central. In this case there are 3 steps, but it is a configurable parameter where any number of steps can be set. Replicating the data allows as to do all the process in parallel so taking the maximum advantage of the GPU. i.e. if in conventional mode only 5000 PRN could be processed at a time, the user can compute any desired combination up to 5000 to, for example 100 PRNs with 50 Doppler steps, when replicating the 100 input PRNs that is going to make 5000.
Once the Doppler frequency with the appropriate steps are applied the process follows similar to the one explained for the waveform cases. The FFT is computed to the whole set of data and then is multiplied by the complex conjugate of the FFT of the replica PRN if it is the cGNSS-R case, Section 3.5.1. Otherwise for iGNSS-R case, Section 3.5.2, instead of the replica it only needs the direct signal. The FFT of it is computed, complex conjugated, and then multiplied to the reflected signal. Each chunk of the direct signal is going to be multiplied to all the replicas of the reflected signal corresponding to the same chunk. The FFT corresponding to the #1 of the direct signal is going to be multiplied to all the replicas of the reflected signal corresponding to the #1, in the case of the figure 3 times. After the multiplications, the IFFT is performed and the cross-correlation functions are obtained.

The following step is finding the peaks. In the Figure 3.16 is not shown for simplicity, but incoherent additions are possible, and that maximum would affect to all the group which was added. Once having the maximum it is necessary to find the maximum across all the shifts, in this case across the 3 steps, are represented in the bottom of the Figure 3.16. For example for the #1 it is going to look which
of the 3 shifts corresponding to #1 has bigger amplitude (as example in the Figure for #1 is the central) with this location it will go back to the cross-correlation function and catch the waveform in that same location in the 3 different shifts (the same sample delay). In this way, when the different peaks of the same PRN are joined to construct the delay-Doppler map it will be continuous. Having in one axis of the DDM the time domain and in the other the frequencies.

Figure 3.17: DDM from real data. Computed with 55 steps with a step size of 25Hz.

Figure 3.17 shows a delay-Doppler map. There are 55 shifts (waveforms), each one with a different Doppler but corresponding to the same chunk of data, with a step size of 25 Hz between each one, the central frequency is 517 Hz. Each waveform are 311 samples. This DDM was processed with our DPU using real data from the MIR instrument.

Summary

The important keypoints of this Chapter about the implementation are:

- GNSS-R uses more amount of data than GNSS.
- FFT-methods are the most efficient way to process GNSS-R signals.
- Parallel processing in the GPU with CUDA is the best candidate to implement a Data Processing Unit (DPU).
• Using sizes of power of 2 improves the performance of FFTs, but adds small artifacts.

• It has been implemented a very configurable DPU which takes advantage of the GPU.
  – Can use different GNSS constellations signals, and bands (L1/E1 + L5/E5a).
  – Can process cGNSS-R and iGNSS-R signals computing waveforms or DDMs.
Chapter 4

RESULTS

In this chapter results of testing our GNSS-R Data Processing Unit (DPU) are presented. First a full time analysis is presented. After that, a review of the memory used of our full DPU, and a comparison between CUDA, C++ and Matlab are presented.

4.1 Time results

For this Section, the tests have been done with two different NVIDIA graphic cards. A NVIDIA GeForce GT 750M (384 cores at 967 MHz with 4 GB of dedicated memory) which corresponds to an average home computer graphics card, in an average computer with an Intel i7 at 2.20 GHz, and a top home graphics card, a NVIDIA RTX 2080 (2944 cores at 1472 MHz with 8 GB of dedicated memory) in a more powerful computer with an i7 at 3.20 GHz.

4.1.1 cGNSS-R processing time analysis

Figure 4.1 shows the elapsed time of conventional processing between the two GPUs. To compute 5 s of real data from the MIR, in which the PRN to correlate and the Doppler to apply are known, with the GT 750M card the DPU requires 1.94 seconds when the size for processing is not power of 2 (32736), a 39% of the real time represented by the data (the 5 s), and less than 1.5 seconds when it is power of 2 (32768), a 27% of the 5 s. With the RTX 2080 graphic card only around 250 milliseconds are needed in both cases when computing 5 s of data, that is only a 5% of the time.
Figure 4.1: Total elapsed time when processing 5 s of data for cGNSS-R signals of two different sizes, one power of 2 (32768) and the other not power of 2 (32736), in two different NVIDIA graphic cards.

All the process is divided in different functions in the program. The following list explains the main functions, and if it is computed in the GPU from the program processing loop in chronological order of processing and using the same abbreviations as the figures elapsed time per function in percentages. To better understand the loop process the reader is referred to Figure 3.11 which contains a work-flow diagram.

1. Read: reading input data in bits.
2. Mask: transforming input data from bits to floats to operate with them (in GPU).
3. Extend: replicate the input data for applying different Doppler in case of DDM computing (in GPU).
4. Doppler: apply Doppler frequency to the signal (in GPU).
5. FFT: compute the fast Fourier transform of the data (in GPU).
6. Mul: compute the complex conjugate of the FFT of one signal and complex multiply it with the other (in GPU).
7. IFFT: Compute the inverse FFT of the product, obtains the cross correlation function (in GPU).
8. Incoh: compute the incoherent additions (in GPU).
9. Max: find the maximum (the peak of correlation) (in GPU).
10. STD: computes the standard deviation (STD) and mean of the cross correlation function (in GPU).

11. SaveP.: save peak, it saves the part of the signal that contains the peak (in GPU).

12. Write: write output on hard disk.

Figure 4.2: Elapsed functions time in cGNSS-R processing mode operating with power of 2 size FFTs. With 2 different graphic cards: (a) with GT 750M, (b) with RTX 2080.

FFT and IFFT are the more time consuming functions as it can be seen in Figure 4.2. With both graphic cards, they take 26% of the processing time each, that is 52% of the total time. These tests have been done with a power of 2 size, with a non-power of 2 size they take even a bit more each, around 30%. This is how the overall time is distributed across the functions, as seen in the overall
times, the RTX 2080 is much faster than the GT 750M, but the distribution patterns are similar. These functions have been implemented with the cuFFT library, a well-optimised library for parallel computing Fourier transforms developed by NVIDIA and included in CUDA. The next two more time-consuming functions for the GT 750M are applying the Doppler frequency and the Mul, which complex conjugates and complex point by point multiply the Fourier transforms of the signals. These two functions have been implemented for this project, also in a way that they are parallelised and computed in the GPU. In the RTX 2080 case, the read and write functions have more percentage as the GPU functions as Doppler or Mul are executed in the GPU and this one is remarkably fast, reducing the percentage of its functions. One of the exceptions that does not follow this pattern of distributions is when processing small amounts of data (50 PRN) with the RTX 2080. In this case the operations done in the GPU are done so fast due to the small amount of data that the operations done in the CPU are the slow ones taking most of the percentage. When this happens the read and write functions cover around 30% each as they are the most expensive CPU functions. The rest of the functions deal with less data and less complicated calculations and thus they are less time consuming.

4.1.2 iGNSS-R processing time analysis

Figure 4.4 shows the elapsed time for processing interferometric signals between the two GPUs. For computing 5 s of real data with the GT 750M card the correlator needs 2.60 s when the size for processing is not power of 2 (32736), a 52% of the time computed, and less than 1.73 s when it is power of 2 (32768), a 34% of the 5 s. With the RTX 2080 graphic card only around 300 milliseconds are needed for both cases when computing 5 s of data, that is only a 6% of the time.

Despite the RTX 2080 is much faster than GT 750M the distribution patterns are again repeated as when processing for cGNSS-R signals. When processing for interferometric signals the FFT is the function that consumes more time overall (Figure 4.3). In this case, it is slower than the IFFT because it has to compute the FFT of the whole reflected signal and the same amount in direct signal from the up-looking antenna to correlate, as compared to the processing for cGNSS-R that only computes the FFT of one of the signals and one FFT of the local replica PRN to correlate with all the signals. This reason makes the interferometric mode a bit more slower than the conventional one when computed with the same hardware. For the other functions the elapsed times are similar to the conventional. The second more time consuming GPU function is the IFFT followed by the Doppler, and the multiplication. When processing very small amounts of data of iGNSS-R
signals with the RTX 2080 card, as it have been seen for the processing of cGNSS-R, the GPU operations are much faster than the CPU ones and the distributions change increasing the read and write percentages.

Figure 4.3: Elapsed functions time in iGNSS-R processing mode operating with power of 2 size FFTs. With 2 different graphic cards: (a) with GT 750M, (b) with RTX 2080.
Figure 4.4: Total elapsed time when processing 5 s of data for interferometric signals of two different sizes, one power of 2 (32768) and the other non-power of 2 (32736), in two different NVIDIA graphic cards.

4.1.3 DDM processing time analysis

For DDMs the elapsed times the distributions are very similar to their respective modes. In Figure 4.5a shows the elapsed times of the functions for DDM computation for cGNSS-R technique. There are different distributions of possible DDM combinations that could have been set in the configuration file, but all of them have to compute around 5000 chunks simultaneously. In $100 \times 51$ only 100 chunks of data are used but replicated 51 times for different Doppler shifts, in $500 \times 11$ 500 chunks of data are used but only replicated 11 times for Doppler shifts, in $1000 \times 5$ 1000 chunks are used and just replicated 5 times for different Doppler shifts, the $5000 \times 1$ is the normal processing for cGNSS-R, where 5000 chunks of data are processed and only one Doppler is applied. One of the main changes is the function Extend that appears whenever DDMs are computed to replicate the signal for the different DDMs. On the other hand when DDMs are computed as less input data is used each time, less time is elapsed in Reading and Masking. For FFT and IFFT the worst performing is in $5000 \times 1$, in this case 5000 overlapped chunks are together while in the others they are grouped in smaller groups of only 1000, 500 or 100, but the distribution percentages are very similar positioning FFT and IFFT in the slower functions, as in Figure 4.3.

Figure 4.5b shows the same graph when processing DDMs using interferometric mode. As explained in the processing for iGNSS-R part the FFT takes longer than in the processing of cGNSS-R due to it has to process also the direct signal. However the percentages are maintained with the ones presented before in Figure 4.3 situating the FFT as the more time-consuming function.
Figure 4.5: Elapsed times for each function using different data configurations (PRN-Doppler bins) of DDM, (a) for cGNSS-R technique (b) for iGNSS-R technique. Computed with NVIDIA GT 750M.

Figure 4.6 is the addition of all the functions to compare between data combinations and conventional/interferometric modes. Interferometric mode is always slower, this is caused for the extra FFT from the direct signal. Also the program performs better when, although having the same amount (or more) of data, this one is divided in different Doppler bins. The slower case is the 5000·1 in which no DDM computation is done, and the 5000 FFTs are overlapped in one group. The other cases where the amount of data is the same, and they are separated in smaller groups because they are different Doppler bins, are faster. Nonetheless, the overall times to compute 5 s of data (5000 chunks = 5000 ms) takes 1 - 1.6 s which are really fast allowing for real-time applications. The following section is going to focus in overall time performance.
4.1.4 Overall time results

Figure 4.7 summarizes the results of one loop of processing or data-line (from reading data to writing the outputs) to process 5 s of GPS data, which corresponds to 5000 chunks (5000 FFTs) of size 32768 if using a power of 2 size or 32736 if not. Our DPU always take less than the 5 s to process the 5 s of data. With the GT 750M the fastest case is when processing processing for conventional mode and the FFTs are a power of 2 size, 1.36 seconds (27% of the real time). The slowest is the processing for interferometric mode case when the size is not a power of 2, 2.60 seconds (52% of the time). With the RTX 2080, the fastest and slowest cases are the same ones, and the overall time are much smaller. The fastest case takes 0.232 s and the slowest 0.336 s (around the 5% of the 5 seconds). These results indicate that the processing is being done always in half or less than the real time that represents that data. Real time processing can then be performed with this DPU as it process faster than the input data flow.

Figure 4.8 shows the data vs elapsed time graphics for both graphic cards. It is very clear that the GT 750M needs more time than the RTX 2080 when more data is processed at the time. In general interferometric processing is slower as more data is needed to be processed, and the cases where the size is not a power of 2 are also slower than their equals with a power of two size. All the cases are always below the slope equal to 1, this means that it takes less time to process the data, than the time in which has been acquired in the different cases of processing and data amount and size.

Looking at the equations for more detail is important to see the slope of each line. For each extra chunk (1 ms) of data processed simultaneously, between 0.52 and 0.27 ms more are needed to process when is computed with the GT 750M. These
Figure 4.7: Total elapsed time when processing 5 s of data in two different sizes, one power of 2 (32768) and the other non-power of 2 (32736). Computed in two different NVIDIA graphic cards.

cases involve a speed up of 1.9x-3.7x times faster than the real time depending on the case. With the slopes lower than 1, becomes very notorious when huge amounts of data are being processed. Results like this indicate that the data is being processed fast enough to enable real time processing. If more powerful hardware is used the data is processed even faster. With more powerful hardware, as the RTX 2080, when an extra millisecond of data is added to process, it would only take between 0.047 to 0.067 ms more to process it depending on the case. These cases are a speed up of 14.9x-21.5x times faster than the real time depending on the case and a speed up of between 5.8x-7.7x times faster respect the GT 750M.

This results states that is not clear that processing with power of 2 size is always the obvious choice. With an NVIDIA RTX 2080 the increment in time is of 1% the real time of the data, which is completely negligible and this avoids to have the artifacts explained on Sections 3.3. With less powerful hardware as NVIDIA GT 750 M the increment is higher, from 27% to 38%, but still much faster than real-time.
Figure 4.8: DPU elapsed time vs data processed in different processing techniques and sizes. (a) with NVIDIA GT 750M, and (b) with NVIDIA RTX 2080.

4.2 Memory results

In this section the memory involved in the processes are going to be presented. The memory of the GPU (dedicated RAM) is going to be analysed and some details of the CPU RAM will be given. In the GPU used memory, the memory occupied needs to be divided in two groups: the declared variables and the cuFFT buffer. The cuFFT buffer is a memory space that needs to be reserved for the FFT and IFFT functions in order to perform the computations, if it was not reserved the program would terminate in error.

The tests have been done with the NVIDIA GT 750M GPU that has a dedicated memory of 4 GB. Tables 4.1 and 4.2 shows the memory used by the whole program when X chunks of data are processed in each data-line no matter the amount
Table 4.1: Percentage of GPU dedicated memory used in processing for cGNSS-R. Computed with NVIDIA GeForce 750M with 4 GB of memory.

<table>
<thead>
<tr>
<th>Chunks of data</th>
<th>50</th>
<th>500</th>
<th>3000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuFFT Buffer</td>
<td>0,31%</td>
<td>3,05%</td>
<td>18,31%</td>
<td>30,52%</td>
</tr>
<tr>
<td>Variables declared</td>
<td>0,35%</td>
<td>3,49%</td>
<td>20,89%</td>
<td>34,82%</td>
</tr>
<tr>
<td>Total memory used</td>
<td>0,66%</td>
<td>6,54%</td>
<td>39,20%</td>
<td>65,34%</td>
</tr>
</tbody>
</table>

Table 4.2: Percentage of GPU dedicated memory used in processing for iGNSS-R. Computed with NVIDIA GeForce 750M with 4 GB of memory.

<table>
<thead>
<tr>
<th>Chunks of data</th>
<th>50</th>
<th>500</th>
<th>3000</th>
<th>5000</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuFFT Buffer</td>
<td>0,31%</td>
<td>3,05%</td>
<td>18,31%</td>
<td>30,52%</td>
</tr>
<tr>
<td>Variables declared</td>
<td>0,65%</td>
<td>6,53%</td>
<td>39,20%</td>
<td>65,33%</td>
</tr>
<tr>
<td>Total memory used</td>
<td>0,96%</td>
<td>9,58%</td>
<td>57,51%</td>
<td>95,85%</td>
</tr>
</tbody>
</table>

of data-lines. As the program reuses the variables the program can process any number of data-lines without any change in the memory. Changing between GPUs does not change the memory used, as the variables are the same, the only that would change is the upper limit of simultaneous chunks that can be computed per data-line. Having a memory of 4 GB enables to compute up to 7000 chunks per data-line. Changing between power of 2 or non power of 2 sizes for the computations has almost no effect on the memory, it only increments a few samples the arrays which is negligible in comparison to the general amount of data. For 5000 chunks in processing for conventional mode only the 65% of the memory is used. When computing 50 PRN, not even 1% of the memory is used. The cuFFT buffer is a large amount of the memory needed in conventional is always a 46.7% of the total memory used, almost the half.

For processing interferometric mode more memory is needed for the allocating, as two signals are opened at the time, the direct and the reflected ones, instead of one. For 5000 chunks computed in interferometric mode 95.85% of the memory is used. The cuFFT buffer in this case remains the same and is the allocated memory the one that increases. In this case the cuFFT buffer represents 31.8% of the memory used in all the cases.

The program uses always less CPU memory (RAM) than the GPU. For process-
ing for cGNSS-R the RAM used is always 48.2% of the total GPU memory used. Processing in iGNSS-R 64.4% of the memory is used. In the CPU no buffer is requested as the computations are done in the GPU and less variables are allocated.

Table 4.3: Percentage of GPU dedicated memory used in DDM computations using cGNSS-R technique. Computed with NVIDIA GeForce 750M with 4 GB of memory.

<table>
<thead>
<tr>
<th>Data configurations</th>
<th>100*51</th>
<th>500*11</th>
<th>1000*5</th>
<th>5000*1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuFFT Buffer</td>
<td>0,61%</td>
<td>3,05%</td>
<td>6,10%</td>
<td>30,52%</td>
</tr>
<tr>
<td>Variables declared</td>
<td>34,27%</td>
<td>37,06%</td>
<td>33,82%</td>
<td>34,82%</td>
</tr>
<tr>
<td>Total memory used</td>
<td>34,88%</td>
<td>40,11%</td>
<td>39,93%</td>
<td>65,34%</td>
</tr>
</tbody>
</table>

Table 4.4: Percentage of GPU dedicated memory used in DDM computations using iGNSS-R technique. Computed with NVIDIA GeForce 750M with 4 GB of memory.

<table>
<thead>
<tr>
<th>Data configurations</th>
<th>100*51</th>
<th>500*11</th>
<th>1000*5</th>
<th>5000*1</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuFFT Buffer</td>
<td>0,61%</td>
<td>3,05%</td>
<td>6,10%</td>
<td>30,52%</td>
</tr>
<tr>
<td>Variables declared</td>
<td>65,39%</td>
<td>70,62%</td>
<td>64,34%</td>
<td>65,33%</td>
</tr>
<tr>
<td>Total memory used</td>
<td>66,00%</td>
<td>73,67%</td>
<td>70,44%</td>
<td>95,85%</td>
</tr>
</tbody>
</table>

Tables 4.3 and 4.4 show the percentages of memory used when computing DDMs. The allocated memory for variables declared stay constant across different DDM configurations of data because in the end similar number of chunks are computed. However, in any of both processing modes, the cuFFT memory increases a lot. For example in the cases of 100·51 only a buffer of 0.61% is needed as the FFTs will be grouped in packs of 100, while in the 5000·1 cases is a 30.52%. In these cases the RAM used is also always less than the total GPU memory used, however it oscillates more in value as they are different combinations, between the 48 - 90%.

### 4.3 CUDA, Matlab and C++ comparison

This test checks the times of the basic functions of the GNSS-R conventional processing with FFT-based methods. It has been implemented in Matlab, C++
and CUDA the main 5 functions for processing GNSS-R signals. Matlab has been chosen as interpreted language and because it is widely used in research and academia. C++ is used as a reference compiled language and a fast procedural language. And CUDA is the one we are using for the implementation and want to compare with. The analyzed functions are: FFT, IFFT, conjugate and point to point complex multiplication, point to point complex summation, and maximum finding. The function used for the Fourier transforms in Matlab R2018b is the one already implemented, in C++ (using Microsoft Visual C++ 2017), the free software FFTW library version 3.3.5 has been used, and in the GPU our processing unit using the freeware library cuFFT included in CUDA Toolkit 10.

![Elapsed time results of 5 functions](image1)

Figure 4.9: Total elapsed time when processing 1 s of data in four different sizes, the first two: 32736 and 32768 in (a), in this case the process is of 1000 chunks. And the other two 130944 and 131072 in (b) in this case as the chunk is longer there are less in a second, 250. 32736 and 130944 are not power of two sizes, and 32768 and 131072 are. These tests are computed in two different graphic cards: GT 750M and RTX 2080.

Four FFT sizes of has been selected for the tests. The first two are: 32736 and
32768. The first one corresponds exactly to 1 ms of data acquired by the MIR instrument, and it is a multiple of the chipping rate used in GNSS modulations, but not a power of 2. The latter is the nearest power of 2, which allows theoretically faster FFT operations. The other two sizes are 130944 and 131072. With these sizes it can be checked if by increasing the size of the FFTs there are significant changes in the elapsed times. 1310944 corresponds to 4 times the first size (32736) and it corresponds to 4 ms of data acquired by the MIR instrument and is also a multiple of the chipping rate used in GNSS modulations, and not power of 2. The Galileo constellation instead of working with a C/A code of 1 ms as GPS uses C/A codes of 4 ms. 131072 again is the nearest power of 2.

The tests have been done with the two different NVIDIA graphic cards. A GeForce GT 750M (384 cores at 967 MHz with 4 GB of dedicated memory) which corresponds to an average home computer graphic card and an actual top home graphic card, a RTX 2080 (2944 cores at 1472 MHz with 8 GB of dedicated memory). As it can be seen in Figure 4.9 for both graphic cards results are very similar. In the RTX 2080 test the results are shifted down in the Y axis as they are faster, but share the same pattern.

The results presented in the following Figures are for the first 2 sizes (32736 and 32768). For the 2 longer sizes (130944 and 131072) results are very similar and for an easier understanding results for just the first two sizes are presented. Only at the end of the section results for the 4 sizes will be presented.

Looking at Figure 4.10 there is a more detailed graphics of the results for the 2 first FFT sizes where different amounts of data are processed for these sizes. Matlab and C++ are very similar in time, C++ is faster in power of 2 sizes, and slower in not power of 2 sizes. Matlab achieve less variance. The main drawback is that both of them have to go operation by operation. However CUDA is far ahead of both of them, as GPU allows to make all the operations simultaneously. In the RTX 2080 test using this powerful computer for Matlab and C++ with a power of 2 size the slope is also less than 1 (taking in to account that is only 5 functions not the whole program for processing). Even though with a proper powerful computer is possible to achieve very good performance time, or even it is possible to make multi-thread computing in the CPU also. However besides of being more difficult to implement multi-thread on CPU, or more expensive hardware the CPU will always be limited by the few processors that it has as compared to the GPU. For the average GPU (GT 750M) CUDA processing (0.18-0.30 slope) is already far ahead of any Matlab or C++ configuration with the more powerful hardware of the computer with the RTX 2080 (0.73 the best of them). Already more than a speed improvement of 2x times.
Figure 4.10: Elapsed time of the five functions tested vs data amount comparison when implemented in CUDA, C++ and Matlab. For each language 2 sizes has been tested, a size power of 2 (327368) and a non power of 2 size (32736). Tested in two different graphic cards: (a) with NVIDIA GT 750M, and (b) with NVIDIA RTX 2080.

Focusing at the time cost of each function when processing 1000 chunks of data per loop, FFT, Conj & Mul (complex conjugate of a signal and point to point complex multiplications), and IFFT are the ones that require more time, see Figure 4.11. In general better times are achieved with a better computer, but CUDA with a less powerful computer achieves considerable better times than C++ or Matlab with any configuration or computer. Only CUDA with a powerful graphics card is significantly more faster.

If the elapsed times of the functions are normalised with the times with the slower case for each, CUDA reduces the elapsed time always to less than 15% than the
Figure 4.11: Main functions from a GNSS-R processing unit execution time comparison when implemented in CUDA, C++ and Matlab. For each language 2 sizes has been tested, a size power of 2 (327368) an a non power of 2 size (32736). Tested in two different graphic cards: (a) with NVIDIA GT 750M, and (b) with NVIDIA RTX 2080.

slower case when computing with GT 750M, and to less than 6% when computing with the RTX 2080. This demonstrates a significant improvement in time as compared to the other languages. This time efficiency is also reflected on the equations, shown in Figure 4.10, looking at the slope values, they are significantly lower than 1, 0.18-0.30 with the GT 750M and 0.032-0.037 with the RTX 2080. This gives the opportunity to increase the amount of data computed without extremely increasing the computing time and always maintaining it below the slope 1, which could enable real-time processing.

With the two longer sizes (130944 and 131072), these tests where made in order to check if changing to a longer size of the FFT would affect to the computing time. Figure 4.12 shows the results of the sum of the 5 functions when computing 1 s of data. The same amount of samples has been processed at the time, but with
different data chunk sizes. The left side points of the graphics are the times for 1000 chunks of 32736/32768 sizes. On the right side as the size is 4 times more, there are 4 times less chunks, 250 of 130944/131072. For both graphic cards changing the size for CUDA has almost no effect, the elapsed time is almost the same for both sizes. For C++ the elapsed time increases a bit, while for Matlab it reduces a bit, more in the non power of 2 case than in the other one. The objective of this comparison was to check that the behaviour of the languages was not only when computing with a 32736 or 32768 size, the ones used for GPS constellation. Also in case that the size were changed to a larger one, for example 130944 or 131072, the ones used for Galileo constellation. CUDA has demonstrated to maintain its better performance and efficiency using different sizes of data chunks.

![Time evolution for chunk size increasing](image)

**Figure 4.12:** Time evolution when changing the PRN/FFT size, analysed in different languages and sizes being power of 2 and not. The measure always have the same amount of samples, when increasing the chunk size the number of chunks decrease, in this way 1 s of data is always processed. These tests are computed in 2 different NVIDIA graphic cards GT 705M(a) and RTX 2080(b).
Summary

The conclusions of this Chapter presenting the time results are:

- Tested in different hardware.
- Using sizes of power of 2 improves the performance.
  - The difference in time is barely noticeable and the artifacts can be avoided.
- Enables Real-time processing, speed up of 14.9x to 21.5x times the real time of the data.
- CUDA outperforms Matlab and C++.
Chapter 5

MIR AIRBORNE CAMPAIGN

In this chapter, preliminary results of an airborne campaign performed in Australia in April - June 2018 are presented. These results are useful to validate the proper functioning of the DPU, as well as to validate the data from the MIR instrument. In addition, they will show the potential of some of the applications that have been explained in previous chapters.

5.1 Soil moisture estimation

Based on the reflectivity changes it is possible to estimate soil moisture with GNSS-R. Two flights were conducted over land in the zone of Yanco, NSW. The 'Dry' flight took place on 1 May 2018 after 15 days with no rain, and with just 9.2 mm accumulated rainfall in previous 89 days. The other, the 'Wet' flight on 18 June 2018 took place right after 8.2 mm rainfall, and with 24.6 mm accumulated rainfall in the previous 9 days [30]. Figure 5.1 shows the ratio of power of the reflected signal over the direct one in the same area for the 4 beams of the MIR. The reflected peak is lower than the direct one, as it has scattered in the reflection. However, knowing that the signal will be weaker, in MIR the down-looking beams have a powerful amplifiers. For these examples, the power gain has not been compensated in neither of the two flights which just shift the power up in both cases. It can be clearly seen that in the 'Wet' flight the power ratio of the peak is 5 to 10 dB higher than in the 'Dry' flight. This effect is due to the humidity in the ground, a more humid ground increases the reflectivity.

The flights are conducted over the OzNet soil moisture sensor network [31] in Yanco, NSW to validate the data. The station Yanko Site 10 (Y10), equipped with soil moisture probes, Figure 5.2, shows that the 1 May the surface humidity was
Figure 5.1: Uncalibrated peak power ratio on Yanco flights. (a) Dry flight: 15 days with no rain, 9.2 mm accumulated rainfall in previous 89 days accumulated rainfall in the previous 9 days. (b) Wet flight: right after 8.2 mm rainfall, 24.6 mm. Y10 Figure 5.2, YB5D Figure 5.3.
5% and the 18 June it was 23%. It can also be confirmed by other ground sites in the area as YB5d (Figure 5.3), which is a dryer zone in comparison to Y10. And other ground stations in the area from the OzNet as YB5a, YB5b, YB7a.

With this experiment it can also be stated that GNSS-R mainly detects surface humidity on land. The humidity in deeper soils is not affected by rains as shown in Figure 5.2. So, the variations in the peak power of our MIR data are caused by the surface humidity, as it behaves following the SM 0-5cm.

Figure 5.2: Data from Yanko Site 10. Sampling SM every 20 min and rainfall every 24h. (-35.00535, 146.30988). Data from [31]
Figure 5.3: Data from Yanko Site B5d. Sampling SM every 20 min. (-34.98483, 146.29299). Data from [31]

5.2 Water / Land transitions

A flight took place over Port Phillip Bay. Figure 5.4 shows the ratio of power of the reflected signal over the direct of the whole flight, for the 4 beams uncalibrated. It is possible to detect the land/water transitions. In the zooms shown in the Figure 5.5 is clearly visible the transitions between water and land. Water has a much higher reflectivity than land. It not only detects the ocean but also small water reserves in-land as lakes or rivers, (B).

In these previous Figures the reflected points are incoherent averages of 1000 ms, for each 1000 choerent ms only results one point as output. If the incoherent averaging is done in with less time i.e. 20 ms, there is an output for every 20 ms, which results in a higher sampled output and more spatial resolution. Figure 5.6a is one of the beams of the same lake as Figure 5.5b, but instead of being incoherently averaged for every 1000 ms, it is averaged for 20 ms. Figure 5.6b shows the peak power ratio with different incoherent averages values over time. It corresponds to the transitions of the lake, Figure 5.6a corresponds to 20 ms incoh line, and the same beam of Figure 5.5b corresponds to 1000ms incoh line, plus other values have been computed. The high peak power ratio areas corresponds to the water which has a higher reflectivity and the low areas to land and these ones can be clearly classified. With lower values of incoherent time there is more sampling but also more noise, nonetheless some effects as the transient state just before the changing of surface can be detected as shown in the 20 ms incoh. With a higher value of incoherent time there is less spatial resolution. Figure 5.7 shows a tran-
transition from land to sea, and with 1000 ms the transition is wider. Whereas with lower values of incoherent time as 100 ms or 20 ms the transition is significantly sharper, which means more time/spatial resolution where the change from water to land has happened.
Figure 5.5: Zooms of the flight over Port Phillip Bay showing the uncalibrated peak power ratio. It can clearly be seen the water/land transitions. Whole flight on Figure 5.4. (The map has a small geolocation error of meters as it is a picture.)
Figure 5.6: Transition between water/land in a in-land lake. (a) the spatial map and the incoherent averages of 20 ms. (b) evolution in time of the peak power ratio of the same lake transition for different incoherent times. Note that for shorter integration times there is more noise, but the ripples due to the different Fresnel zones is clearly visible.
Figure 5.7: Transition between water/land in the sea. (a) the spatial map and the incoherent averages of 1000 ms. (b) evolution in time of the peak power ratio of the same lake transition for different incoherent times. Note that for shorter integration times there is more noise, but the ripples due to the different Fresnel zones is clearly visible.
5.3 Altimetry

Ocean altimetry is another of the applications of GNSS-R. The receiver height can be computed from the delay between the direct signal and the reflected signal. The reflected signal covers a longer path, as it has to first arrive to the Earth and be reflected. So, the direct signal will be the first to arrive, and after the reflected will arrive. A flight over the sea took place to test height capabilities.

The position of the cross-correlation peak indicates the time of arrival, $T$, of the signal. For airborne heights, it can easily be transformed into distance with the speed of electromagnetic waves, $c$, and the angle of elevation, $\sigma_e$ of the incoming signal:

$$h_{airborne} = \frac{T_{reflected} - T_{direct}}{2 \sin(\sigma_e)} \cdot \frac{1}{c}$$

(5.1)

If the sea height is computed over all the sea flight Figure 5.8b is obtained. It is not calibrated, but from one vertex near the land to the other in the middle of the sea there are 10 m difference. Comparing with Figure 5.8a which shows the real geoid height value of the same tracks of the flight, the difference is of 10 m. The geoid indicates the height over the theoretical ellipsoid which represents the Earth due to the gravitational field. The flight was conducted in this spot for the big gradient (10 m) in few km of the geoid. Fully calibrating the height retrieved by the MIR it is possible to subtract the geoid height and obtain the sea surface height over the geoid and wave heights.
Figure 5.8: (a) Height of the Earth’s geoid. (b) Uncal. retrieved height from the MIR data.
Chapter 6

CONCLUSIONS

The objective of the project has been to develop a time-efficient DPU to process the data of the MIR instrument. It was necessary because until the date there was no feasible way for the Remote Sensing Lab to process the 10 hours raw data from the MIR as they took so much time. After a year working on it, this goal has been successfully achieved and the DPU satisfies all the requirements to process time efficiently the MIR data. To do so, it has been taken a software approach that can be implemented in any computer, and that will take advantage of the GPU to parallelise the work load. Moreover this implementation is a very configurable design satisfying the requirements to be able to process signals from different GNSS constellations.

The first tests were conducted to the 3 options of design that first were implemented. This showed which implementation was faster and helped us to decide in which way the design should go. It was possible to take advantage of the implementations of the FFTs, the ones which size is power of two are computed faster. Also this tests helped us to validate that the processing was correct.

Time results show that time efficiency has been fully achieved. It was conducted a test with Matlab and C++, CUDA (the one used in our implementation) to compare the elapsed time in the GNSS-R processing. It was tested in different hardware and CUDA always outperforms MATLAB and C++ considerably by far.

It is remarkable to state that our DPU always process the data in less time than the real time of the data, a reduction of time of 95%. Also this is confirmed for less powerful hardware as NVIDIA GT 750M which also process in less than the real time. These results enables real-time processing, and for future campaigns this DPU will be able to be set to perform the processing in real-time.
After having the time results it is clear that processing with power of 2 size is not always the obvious choice. With an NVIDIA RTX 2080 the increase in time goes from a 4% to a 5% of the real time of the data, which is completely negligible and this avoids to have some artifacts that increase the SNR. With less powerful hardware as NVIDIA GT 750M the increment is higher, from 27% to 38%, but still much faster than real-time. The size to use is a configurable parameter, so if maximum time efficiency is the objective it can be set a power of two size for the FFTs, elsewhere with the original size from the MIR the results will have less noise/artifacts.

As final validations the first data from the MIR airborne campaign have been processed, they validate the DPU and the instrument and have shown great potential for GNSS-R applications. They confirmed the detection of surface soil moisture detection, water/land transitions detection and altimetry applications.

The results obtained in this degree thesis have been shown in IEEE GNSS+R 2019 Specialist Meeting on Reflectometry using GNSS and other Signals of Opportunity.

### 6.1 Future work

This degree thesis and the developed DPU are located inside the MIR instrument project and such a pioneer project is far from an end. Future work starts by fully processing all the data from the past airborne campaign where many results can be extracted. The next steps involve to analyse more specific techniques, such as fine Doppler frequency estimation [32] or the acquisition of secondary codes [33] and try to apply them into our DPU to study the improvement and extract finer results. Long term work could involve upgrading and fixing bugs from the DPU or the MIR instrument, adapt and install the DPU with the MIR to process the data real-time during the flight or the planning and realisation of future MIR campaigns.
BIBLIOGRAPHY


