UNIVERSIDADE DA BEIRA INTERIOR
Faculdade Engenharia

Parallelization and Implementation of Methods for Image Reconstruction

Applications in GNSS Water Vapor Tomography

Fábio André de Oliveira Bento

Submitted to the University of Beira Interior in candidature for the
Degree of Master of Science in Computer Science and Engineering

Supervisor: Prof. Dr. Paul Andrew Crocker
Co-supervisor: Prof. Dr. Rui Manuel da Silva Fernandes

Covilhã, October 2013

Acknowledgements

I would like to thank my supervisor Prof. Paul Crocker for all his support, enthusiasm, patience, time and effort. I am grateful for all his guidance in all of the research and writing of this dissertation. It was very important to me to work with someone with experience and knowledge and that believed in my work.

I also would like to thank my co-supervisor Prof. Rui Fernandes for all his support in the research and writing of this dissertation. His experience and knowledge was really helpful especially in the GNSS area and the dissertation organization. I am really grateful for all his support and availability.

The third person that I would like to thank is André Sá. He was always available to help me with all my GNSS doubts, his support was really important for the progress of this dissertation. i am really thankful for his time, enthusiasm and patience.

I am also very grateful to Prof. Cédric Champollion for all his support and providing the ES-COMPTE data, David Adams for his support and providing the Chuva Project data and Michael Bender for his support and for providing German and European GNSS data.

I would also like to thank all the other SEGAL members that accompanied, supported and encouraged me in this journey. Thank you Miguel, João and Hugo.

Finally, I would like to thank my mother, father and my brother for all your support, patience and encouragement. I would also to thank my grandmother Joaquina, who unfortunately is no longer present, but always gave me strength and encouragement to achieve my dreams. Without you this work would not be possible. I am really grateful to all of you.

Abstract

Algebraic reconstruction algorithms are iterative algorithms that are used in many area including medicine, seismology or meteorology. These algorithms are known to be highly computational intensive. This may be especially troublesome for real-time applications or when processed by conventional low-cost personnel computers. One of these real time applications is the reconstruction of water vapor images from Global Navigation Satellite System (GNSS) observations. The parallelization of algebraic reconstruction algorithms has the potential to diminish significantly the required resources permitting to obtain valid solutions in time to be used for nowcasting and forecasting weather models.

The main objective of this dissertation was to present and analyse diverse shared memory libraries and techniques in CPU and GPU for algebraic reconstruction algorithms. It was concluded that the parallelization compensates over sequential implementations. Overall the GPU implementations were found to be only slightly faster than the CPU implementations, depending on the size of the problem being studied.

A secondary objective was to develop a software to perform the GNSS water vapor reconstruction using the implemented parallel algorithms. This software has been developed with success and diverse tests were made namely with synthetic and real data, the preliminary results shown to be satisfactory.

This dissertation was written in the Space & Earth Geodetic Analysis Laboratory (SEGAL) and was carried out in the framework of the Structure of Moist convection in high-resolution GNSS observations and models (SMOG) (PTDC/CTE-ATM/119922/2010) project funded by FCT.

Keywords

Algebraic Reconstruction Algorithms, CPU Parallelization, GPU Parallelization, Image Reconstruction, Water Vapor

Extended Abstract

Algoritmos de reconstrução algébrica são algoritmos iterativos que são usados em muitas áreas incluindo medicina, sismologia ou meteorologia. Estes algoritmos são conhecidos por serem bastante exigentes computacionalmente. Isto pode ser especialmente complicado para aplicações de tempo real ou quando processados por computadores pessoais de baixo custo. Uma destas aplicações de tempo real é a reconstrução de imagens de vapor de água a partir de observações de sistemas globais de navegação por satélite. A paralelização dos algoritmos de reconstrução algébrica permite que se reduza significativamente os requisitos computacionais permitindo obter soluções válidas para previsão meteorológica num curto espaço de tempo.

O principal objectivo desta dissertação é apresentar e analisar diversas bibliotecas e técnicas *multithreading* para a reconstrução algébrica em CPU e GPU. Foi concluído que a paralelização compensa sobre a implementações sequenciais. De um modo geral as implementações GPU obtiveram resultados relativamente melhores que implementações em CPU, isto dependendo do tamanho do problema a ser estudado. Um objectivo secundário era desenvolver uma aplicação que realizasse a reconstrução de imagem de vapor de água através de sistemas globais de navegação por satélite de uma forma paralela. Este software tem sido desenvolvido com sucesso e diversos testes foram realizados com dados sintéticos e dados reais, os resultados preliminares foram satisfatórios.

Esta dissertação foi escrita no Space & Earth Geodetic Analysis Laboratory (SEGAL) e foi realizada de acordo com o projecto Structure of Moist convection in high-resolution GNSS observations and models (SMOG) (PTDC/CTE-ATM/119922/2010) financiado pelo FCT.

Contents

List of Figures

List of Tables

Acronyms

ART	Algebraic Reconstruction Techniques
BLAS	Basic Linear Algebra Subprograms
CPU	Central Processing Unit
CUBLAS	CUDA Basic Linear Algebra Subroutines
CUDA	Compute Unified Device Architecture
DMA	Direct memory access
DOY	Day Of Year
GNSS	Global Navigation Satellite Systems
GPS	Global Positioning System
GPU	Graphics Processing Unit
IWV	Integrated Water Vapor
LAPACK	Linear Algebra PACKage
LOFTT_K	LOgiciel Français de Tomographie Troposphérique -version Kalman
OMP	OpenMP
PC	Personal Computer
PPE	Power Processor Element
SART	Simultaneous Algebraic Reconstruction Technique
SEGAL	Space Earth Geodetic Analysis Laboratory
SIRT	Simultaneous Iterative Reconstruction Techniques
SIWV	Slant Integrated Water Vapor
SMOG	Structure of Moist convection in high-resolution GNSS observations and models
SPE	Synergistic Processing Element
SWART	SEGAL GNSS Water Vapor Reconstruction Image Software
ScaLAPACK	Scalable Linear Algebra PACKage
TBB	Intel Threading Building Blocks

Chapter 1

Introduction

The algebraic reconstruction algorithms are iterative algorithms that were initially developed with success for medical imagery [1] although they are now used over other domains (*e.g.* medicine, seismology and meteorology). The data retrieved for the image reconstruction can be obtained from diverse sources such as X-ray, magnetic resonance imaging or seismic travel times. Algebraic reconstruction algorithms were first described in [2] by Stefan Kaczmarz and were later rediscovered in the field of image reconstruction from projections in [1] by Gordon *et. al*. The algebraic reconstruction algorithms have some advantages over other inversion algorithms, namely high numerical stability and computational efficiency [3]. In [3] Bender *et. al* successful applied the algebraic reconstruction technique for the water vapor reconstruction.

Water vapor plays an important role in the weather and climate phenomena such as rainstorms, thunderstorms and other strong convective weather events [4]. It is also important for precipitation forecast and nowcasting [5]. Therefore it is very important to measure the water vapor distribution and its variation in the atmosphere.

The signals of GNSS satellites must travel through the atmosphere in order to be received by the GNSS receivers. The atmosphere's atoms and molecules slow down the GNSS satellites signals [6] causing a delay. It is possible to separate the water vapor delay from other dry gases delays such as nitrogen, oxygen and carbon dioxide [6] and consequently estimate the water vapor in the atmosphere using GNSS.

The GNSS water vapor estimation has many advantages over other methods (*e.g.*, radiosonde, satellite platforms using infrared, microwave sounders), such as good temporal resolution, it operates in all weather conditions and it can run unattended [7]. GNSS water vapor tomography (image reconstruction) was firstly described in [8] by Bevis *et. al*. The water vapor image can be reconstructed using the diverse slant wet delays of a GNSS receiver in the direction of the visible satellites[1]. There are several techniques to reconstruct the water vapor image in the atmosphere using the slant wet delays, including the algebraic reconstruction algorithms.

Algebraic reconstruction algorithms are known to be highly computational intensive [9]. This fact may be troublesome if they are used for real time image reconstruction of GNSS water vapor fields, especially if processed in the conventional personal computers (PCs). Therefore, the improvement of the performance of the estimation is a critical issue on the implementation of these algorithms for water vapor tomography. Obviously the computational resources needed depend on the size of the reconstruction problem and this depends on the quantity of data available, satellites, receivers and slant wet delays, for instance in [3] large problems sizes (matrix with 26000 x 8280 elements) are generated.

[1] The slant wet delay consists in the delay caused by the water vapor in the satellite's signal mapped in the satellite's direction.

1.1 Objectives

The main objective of this dissertation was to study the the parallelization of algebraic reconstruction algorithms in order to evaluate its potential advantages. Although the parallelized algebraic reconstruction algorithms can also benefit other research areas, a complementary goal was to investigate its application on the estimation of GNSS water vapor fields. Nowadays there are several GNSS water vapor image reconstruction software packages, however none of them implements parallelized algebraic reconstruction algorithms.

As consequence of this study, an application to perform the GNSS water vapor reconstruction using the algorithms and techniques developed was implemented: SWART (SEGAL GNSS Water Vapor Reconstruction Image Software). An additional objective of the development of this application was to implement it in conventional low cost personal computers taking full advantage of modern multicore and GPU architectures.

1.2 Main Contributions

The main contributions of this dissertation are here presented.

The first contribution is the analysis of a range of parallel libraries and implementations for the linear algebra operations and consequently the parallelization of the algebraic reconstruction algorithms.

The second contribution consists in the development of a GNSS water vapor image reconstruction application that gathers all the necessary GNSS observations and performs the correspondent water vapor image reconstruction. The software parameters are also customizable to the user.

During the course of this dissertation the following documents and conference papers were written:

An article with the title “Analysis of algebraic reconstruction algorithms performance in CPU and GPU” was submitted and accepted for the ICEUBI conference (http://iceubi2013.ubi.pt).

A conference paper with the title “A Study of GNSS Water Vapor Parameters” was written and accepted as a presentation for the American Geophysical Union (AGU) 2013 Fall Meeting at San Francisco. This presentation will present an analysis of the various parameters to the GNSS water vapor reconstruction using the SWART program. Some of these parameters include covering diverse grid sizes and different number of receivers for the same water vapor image reconstruction. Also comparisons with LOFTT_K (LOgiciel Français de Tomographie Troposphérique -version Kalman) using synthetic data and results from Belem, Brazil which data was acquired in the framework of the project CHUVA will be presented. The AGU 2013 Fall Meeting website can be consulted in http://fallmeeting.agu.org/2013/.

A detailed technical report describing and comparing multithreading algebraic reconstruction algorithms was also written, “Multithreading ART: Comparison”[2].

[2]Available in publications at http://segal.ubi.pt/

1.3 Dissertation Structure

The structure of this dissertation is here described. The current chapter contains the problem definition, the principal objectives of this dissertation, the main contributions of this dissertation.

In Chapter 2 algebraic reconstruction definition is defined and techniques described.

Chapter 3 presents the state of the art of two main topics: GNSS water vapor tomography and algebraic reconstruction parallelization which consists in CPU, GPU and hybrid parallelization.

Chapter 4 introduces the parallelization of the algebraic reconstruction, namely the approach used, the libraries tested and the parallel implementations and the results of these tests.

In Chapter 5 the relation between the GNSS and the water vapor is described in more detail namely the definitions and the methods of estimating water vapor and reconstructing its image.

In Chapter 6 the GNSS water vapor image reconstruction software in development is presented. It is compared with another GNSS water vapor image reconstruction software named LOFTT_K and synthetic and case study results using the implemented software are presented and discussed.

And finally in Chapter 7 the conclusions of this dissertation are presented and future work is described.

Chapter 2

Algebraic Reconstruction

Algebraic Reconstruction is an iterative approach for imaging reconstruction using data obtained from a series of projections such as those obtained from electron microscopy, x-ray photography and in medical imaging like in computed axial tomography (CAT scans). It consists of obtaining data from cross sections of an object from measurements taken from different angular positions around the object and then solving for an array of unknowns that represent the interior of the object being analysed. In Figure 2.1 we can see 6 line projections from three angular positions through an unknown object. For medical applications algebraic reconstruction algorithms lack the accuracy and speed of implementation when compared to other methods [10]. However there are situations where is not possible to measure a sufficiently large enough number of projections or when the projections are not uniformly distributed over 180 or 360 degrees, which prevents the use of other techniques such as transform based techniques that can obtain higher accuracy. The algebraic reconstruction algorithms also have the advantages of having high numerical stability even with inaccurate initial data and are also computationally efficient and easily parallelized [3].

Algebraic techniques are also useful when energy propagation paths between the source and receiver positions are subject to ray bending on account of retraction or when energy propagation undergoes attenuation along ray paths [10].

In the algebraic techniques studied in this dissertation is essential to determine the ray paths that connect the corresponding transmitter and receiver positions. When refraction and diffraction effects are substantial it becomes impossible to predict ray paths which ends in obtaining meaningless results [10].

In this section the concept of algebraic reconstruction is introduced and the main algorithms are described.

2.1 Image and projection representation

Consider a two dimensional image $I = f(x, y)$. Figure 2.1 shows a square grid superimposed onto this image, we want to obtain this image from the projection data, shown as straight lines which traverse the (x, y) plane. We assume that for each cell $f(x, y)$ is a constant and let $f_j : j = 1..N$ be the constant values for each cell of the image. N is the total number of cells in the grid. A line integral will be defined as array-sum. This ray sum is referred as p_i, where i is the i-ray.

The relationship between f_j and p_i's is expressed as:

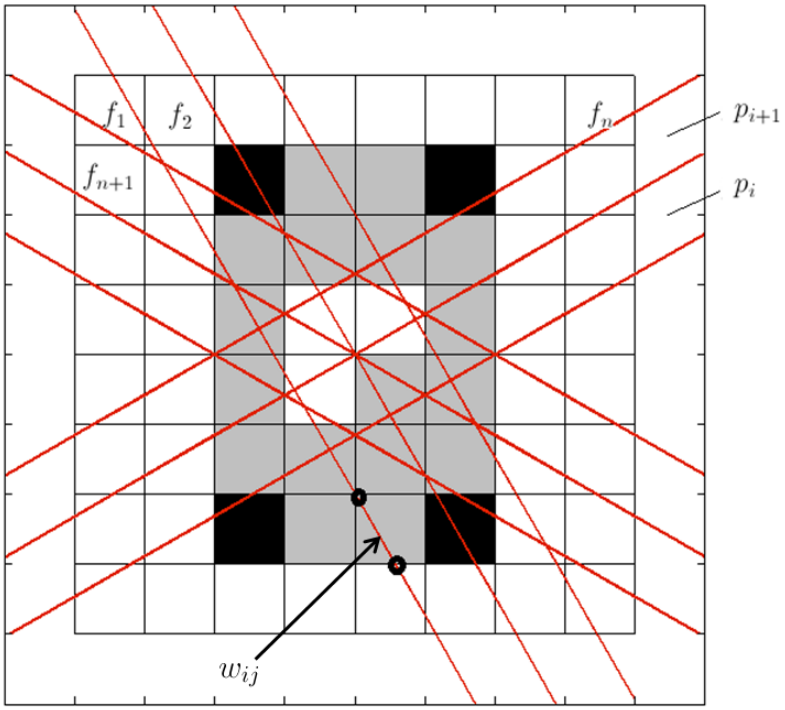

Figure 2.1: Unknown image on square grid. Each cell value is a unknown variable to be determined using the various projections.

$$\sum_{j=1}^{N} w_{ij} f_j = p_i, \qquad i = 1, 2, \cdots, M \quad (1)$$

M is the total number of rays (all projections), w_{ij} is the weight of the contribution of the jth cell to the ith ray integral (this is proportional to fraction of the jth cell intercepted by the ith ray as shown in Figure 2.1).

Depending on the context many of the w_{ij} may be zero because only a small number of the w_{ij} actually contribute to each ray-sum.

The equation in (1) can be rewritten in matrix form as follows:

$$Ax = b \quad (2)$$

where A is a matrix with all the w_{ij} contributions, b is a vector with every p_i ray sum and x is a vector that contains all f_j cells constant values (the original image). M and N correspond respectively to the rows and columns of A.

If M and N were small one could use traditional matrix theory methods to invert the equation system of (1). In practice N may be large, for example for a 256 x 256 image N would be 65.536. If M had more or less the same value then for these values the size of the matrix $[w_{ij}]$ in (1) would be 65.536 x 65.536 which basically rules out any chance of direct matrix inversion [10].

Also when noise is present in the measurement data and $M < N$, even for relatively small N it is not possible to use direct matrix inversion. In this case one can use use a least square method to obtain an approximate solution, however when M and N are large these methods are computationally impracticable [10].

There are however some very attractive methods for solving these equations. These methods are based on the "method of projections" which Kaczmarz first proposed [11]. First the equation (1) will be expanded to explain the computational procedure of these methods:

$$\begin{aligned} w_{11}f_1 + w_{12}f_2 + w_{13}f_3 + \cdots + w_{1N}f_N &= p_1 \\ w_{21}f_1 + w_{22}f_2 + w_{23}f_3 + \cdots + w_{2N}f_N &= p_2 \\ &\vdots \\ w_{M1}f_1 + w_{M2}f_2 + w_{M3}f_3 + \cdots + w_{MN}f_N &= p_M \end{aligned} \tag{3}$$

A grid with N cells gives an image with N degrees of freedom. The image represented by $(f_1, f_2, \cdots, f_N)$ can be seen as a single solution in an N-dimensional space.

Each of the equations in (3) represents a hyperplane. When there is only one solution to the equations it is represented as a single point, namely the intersection of all the hyperplanes. This is the main concept that's illustrated in Figure 2.2. In this figure we have only considered two variables f_1 and f_2 which satisfy the follow equations:

$$\begin{aligned} w_{11}f_1 + w_{12}f_2 &= p_1 \\ w_{21}f_1 + w_{22}f_2 &= p_2 \end{aligned} \tag{4}$$

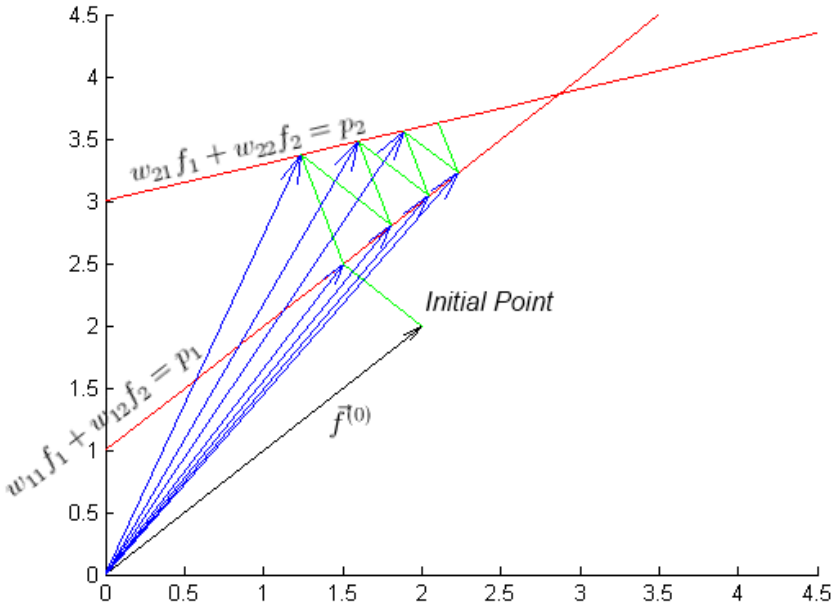

Figure 2.2: Kaczmarz method illustrated for two unknowns.

The computational procedure to calculate the solution is the following:

1. Start with a initial guess
2. Project the initial guess onto the first line
3. Reproject the resulting point from 2 to the second line (3)
4. Reproject the result point from 3 to the first line and so on
5. If there's a unique solution the algorithm will converge.

The initial guess is written as($f_1^{(0)}, f_2^{(0)}, \cdots, f_N^{(0)}$) or simply by the vector $\vec{f}^{(0)}$. Often the initial vector is simply a zero vector. This vector is then projected onto the hyperplane using the first equation on (3) resulting in the $\vec{f}^{(1)}$ vector. This can be seen in Figure 2.2 for a two dimensional space. After that $\vec{f}^{(1)}$ is projected by the second equation on (3) on the hyperplane resulting on $\vec{f}^{(2)}$ and so on. In 2.2 the projections are the green lines and the blue vectors represent the next estimates of the solution.

When $\vec{f}^{(i-1)}$ is projected on the hyperplane represented by the ith equation it can be mathematically be described as:

$$\vec{f}^{(i)} = \vec{f}^{(i-1)} - \frac{\vec{f}^{(i-1)} \cdot \vec{w}_i - p_i}{\vec{w}_i \cdot \vec{w}_i} \vec{w}_i \quad (5)$$

where $\vec{w}_i = (w_{i1}, w_{i2}, \cdots, w_{iN})$ and $\vec{w}_i \cdot \vec{w}_i$ is the dot product of the vector $\vec{w}_i$ by itself.

Regarding the algorithms convergence, it is easily seen that in the case of two perpendicular hyperplanes then for any initial guess in the (f_1, f_2) plane it is possible to find the solution in only two steps using (5). However if the two hyperplanes have a reduced angle between them there will be a greater number of iterations (depending also on the initial guess).

In fact if the M hyperplanes in (3) could all be made orthogonal (perpendicular) with respect to one another then the solution could be found in only one pass through all the equations in (3) (assuming that only one solution exists) [10].

This is theoretically possible using for example a method for orthonormalising a set of vectors such as the Gram-Schmidt procedure. However in practice it is not computationally viable as the orthnomalizing process itself takes too much time. Another problem with orthogonalization is that it amplifies the noise problem from the measurements into the final solution [10].

If we have $M > N$ in (3) no unique solution may exist, although a solution in a zone may still be determined. Figure 2.3 shows a two variable system with three noisy hyperplanes. In this case we see the result after projecting the initial point onto the first line (in green) and then iterating 100 times, the figure show the projections onto the hyperplanes. As can now be seen the procedure in (5) does not converge to a unique solution but instead it oscillates in the neighbourhood of the intersections of the hyperplanes.

In the case of $M < N$ a unique solution also doesn't exist, instead there are multiple solutions. For instance if we have only one equation of the two in (4) to calculate the two variables, then the solution can be on any point in the line that corresponds to this equation.

Another advantage of this method is the possibility of adding a priori information already known about the image being reconstructed in order to guide the solution. If we know for example

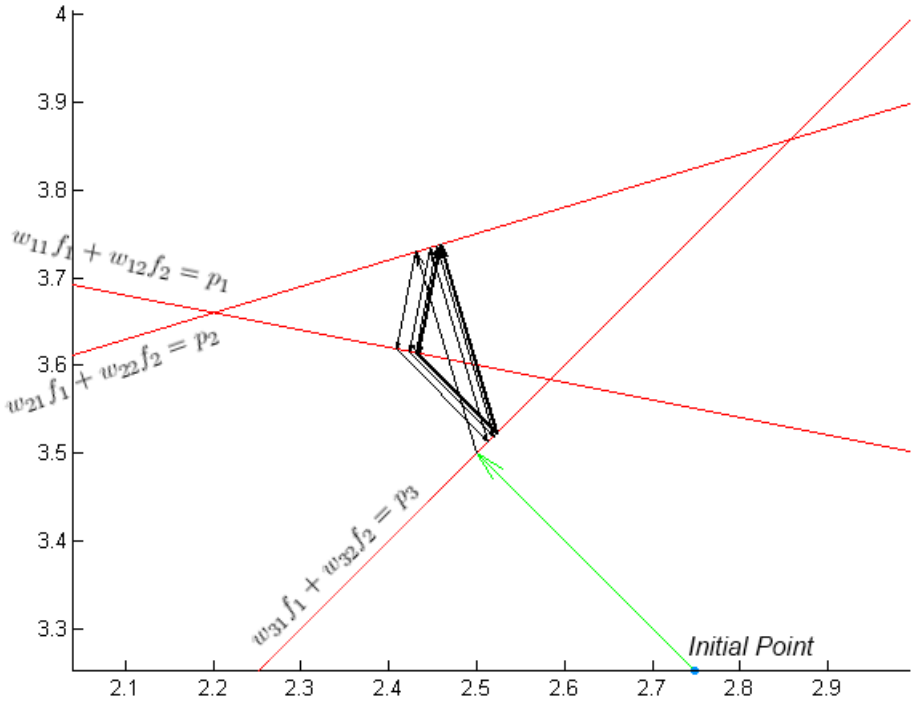

Figure 2.3: Case where the number of equations is greater than the number of unknowns and projections have been corrupted by noise.

that the image contains no negative values and if during the iterative process we obtain we some negative value one can simply reset those values to zero.

2.2 Techniques

There are two different techniques for algebraic reconstruction, namely: Algebraic Reconstruction Techniques (ART) and Simultaneous Iterative Reconstruction Techniques (SIRT) which will be described in the follow subsections.

Different methods of these techniques namely Kaczmarz, Symmetric Kaczmarz, Landweber, Cimmino, CAV, DROP and SART which were implemented for this dissertation will now be presented.

2.2.1 ART methods

The Algebraic Reconstruction Techniques (ART) are row-action methods that treat the equations one at time. This means that in each iteration each equation is solved individually. The updates in each ith iteration are made using the equation on (5) plus a relaxation parameter λ_k resulting in methods described by:

$$\vec{f}^{(i)} = \vec{f}^{(i-1)} + \lambda_k \frac{p_i - \vec{f}^{(i-1)} \cdot \vec{w}_i}{\vec{w}_i \cdot \vec{w}_i} \vec{w}_i \quad (6)$$

What distinguishes the various methods is the order in which row is processed [12]. The ART reconstructions usually suffer from so called "salt and pepper" noise, which is caused by the fact that the values of w_{ij} are only approximations as they are usually measured by experiments [10].

2.2.1.1 Kaczmarz

This is the most well know ART method in the literature [13] [14]. It uses a fixed $\lambda_k = \lambda \in (0, 2)$ in the original paper the value 1 was used. In the literature this method is also referred as ART which can give rise to some confusion since it is also used for Algebraic Reconstruction Techniques. Each kth iteration in Kaczmarz consists of one "sweep" in each row of A from the top to the bottom: $i = 1, 2, ..., m$.

2.2.1.2 Symmetric Kaczmarz

This method is a variant of the previous Kaczmarz method. It adds a new "sweep" using the rows in the reverse order. As result each kth iteration consists of 2 steps: $i = 1, 2, \cdots, m - 1, m, m - 1, \cdots, 3, 2$.

This method supports both a fixed ($\lambda \in (0, 2)$) and iteration-dependent λ_k.

2.2.2 SIRT methods

The Simultaneous Iterative Reconstruction Techniques (SIRT) are "simultaneous" because all equations are solved at the same time in one iteration (based on matrix multiplications) [12]. Usually these methods converge slower to the solution than the ART methods, however they result in better looking images [10].

The SIRT methods work by first solving all the equations and updating only the cell values at the end of each iteration. The change of each cell is the average of each of the changes for that cell [10].

The general form of these methods is as follows:

$$\vec{f}^{(k)} = \vec{f}^{(k-1)} + \lambda_k T A^T M (p - A\vec{f}^{(k-1)}), \qquad k = 0, 1, 2, \cdots \quad (7)$$

where A is the matrix with the various projections, λ_k is a relaxation parameter and the matrices M and T are symmetric positive definite. The various SIRT methods depend on these matrices. λ_k is defined as $2/\rho(TA^TMA) - \varepsilon$, where ρ is the spectral radius, ε the machine's epsilon and A^T is the matrix transpose.

Three of the SIRT methods presented here include positive weights $t_i > 0, i = 1, \cdots, m$. If the weights are not specified then all the weights are set to 1. The methods that include these weights are Cimmino, CAV and DROP and will be presented below.

2.2.2.1 Landweber

The Landweber method is described by the following form:

$$\vec{f}^{(k)} = \vec{f}^{(k-1)} + \lambda_k A^T (p - A\vec{f}^{(k-1)}), \qquad k = 0, 1, 2, \cdots$$

which corresponds to replacing $M = T = 1$ in (7).

2.2.2.2 Cimmino's method

This method was first introduced in [15] and it's often presented in a variant based on projections. The version presented in this dissertation is the same from [12]. This version includes a relaxation parameter λ_k and also a weights vector $\vec{t}$.

Using the matrix notation this method uses the equation (7) with $M = D$ and $T = I$, where D is defined as:

$$D = \frac{1}{m} diag\left(\frac{t_i}{\|\vec{w}_i\|_2^2}\right) \quad (8)$$

2.2.2.3 Component averaging (CAV)

Cimmino's original method uses equal weighting of the contributions from the projections, which looks fair when A is a dense matrix [12]. The CAV method was introduced in [16] as an extension to the Cimmino's method to take in account the sparsity information in a heuristic away [12]. Let s_j be the number of nonzero (NNZ) elements of column j:

$$s_j = \text{NNZ}(a_j), \quad j = 1, \cdots, n \quad (9)$$

Also define the diagonally matrix $S = \text{diag}(s_1, \cdots, s_n)$ and the norm $\|\vec{w}_i\|_S^2 = \vec{w}_i \cdot S\vec{w}_i = \sum_{j=1}^{n} w_{ij} s_j$ for $i = 1, \cdots, m$.

Using the matrix notation this method uses the equation (7) with $M = D_s$ and $T = I$.

D_s is defined as:

$$D_s = \text{diag}\left(\frac{t_i}{\|\vec{w}_i\|_S^2}\right) \quad (10)$$

2.2.2.4 Diagonally Relaxed Orthogonal Projections (DROP)

DROP is another Cimmino's extension. Using the matrix notation it uses the equation (7) with $T = S^{-1}$, $M = mD$. The matrix D is from (8) [12].

In [17] it is proved that $\rho(S^{-1}A^TMA) \leqslant \max_i\{t_i\}$, this means that the convergence is guaranteed if $\lambda_k \leqslant (2-\varepsilon)/\max_i\{t_i\}$

2.2.2.5 Simultaneous Algebraic Reconstruction Technique (SART)

These methods were originally implemented as ART methods [18], but can also be implemented as SIRT method. They are described by the following :

$$\vec{f}^{(k)} = \vec{f}^{(k-1)} + \lambda_k D_r^{-1} A^T D_c^{-1}(p - A\vec{f}^{(k-1)}), \qquad k = 0, 1, 2, \cdots$$

D_r^{-1} and D_c^{-1} are diagonal matrices corresponding respectively to the row and column sums:

$$D_r = diag(||\vec{w}_i||_1) \qquad D_c = diag(||\vec{w}_j||_1).$$

where $\vec{w}_j = (w_{1j}, w_{2j}, \cdots, w_{Nj})$.

There's no need to include weights in this method as the convergence for this method was established in [19] [20] and it was shown that $\rho(D_r^{-1}A^TD_c^{-1}A) = 1$.

2.3 Summary

In this chapter an overview to algebraic reconstruction was realized. It was described the main advantages and applications of these techniques. The two different categories of algebraic reconstruction were presented (ART and SIRT) and some methods of each category were also described. The current chapter is important as basis to the reader understand the following chapter which will present the state of the art of GNSS Water Vapor Tomography and Parallel Algebraic Reconstruction Algorithms.

Chapter 3

State of the Art

In this chapter the state of the art of the water vapor tomography plus the parallelization of the algebraic reconstruction algorithms will be presented. The parallelization of the algebraic reconstruction algorithms will consist in a brief overview of the various implementations namely the cpu implementation, gpu implementation and the hybrid cpu/gpu implementation. This chapter is organized by topic and inside each topic it is organized chronologically.

3.1 GNSS Water Vapor Tomography

GNSS Water Vapour tomography is a technique that allows the distribution of the water vapor on the atmosphere to be calculated [21][22][5].

The calculation of the water vapor is and remains difficult to quantify due to the high variability in time and space plus the sparse set of available measurements [21].

The GPS was proved to measure the integrated water vapor at zenith with similar accuracy as other methods such as radiosondes [23] or water vapor radiometers [24][25]. Also compared to water vapor radiometers the GPS operates in all weather conditions without the need of difficult calibration procedures [21]. Studies have shown that it is possible to quantify the integrated water vapor on the line of sight of the GPS satellite [26] [27] [28]. This quantification is designated as slant integrated water vapor (SIWV) [21] and they are used to calculate the water vapor distribution. Water vapor is a key item in numerical weather prediction, since it plays an important role in atmospheric processes, especially precipitation forecast and nowcasting, hazard mitigation and water management [22] [5]. In the last years, the development of tomographic methods to calculate the 3D distribution of water vapor in the troposphere has been a main topic of research [29].

In [30] Michael Bevis *et al.* showed how to map zenith wet delays onto precipitable water (PW) with an root mean square (r.m.s) error less than 2 mm + 1% of the PW and long-term biases of less than 2 mm.

Niell has introduced in [31] two new mapping functions denominated as Niell Mapping Functions (NMF). These functions are used to map a zenith delay in elevations angles down to 3°. The first function is an hydrostatic mapping function which depends on the latitude, the height of the receiver and the day of year (DOY)

A. Flores *et al.* have shown in [32] how GPS data are processed to obtain the tropospheric slant wet delays. It was confirmed that the tropospheric tomography is a practical approach to the description of the spatio-temporal structure of the water vapor in the atmosphere. In [32] the software LOTTOS for obtaining the tomographic solution using data from Kilauea network, Hawaii in 1 February, 1997 is described.

In [33] John Braun and Christian Rocken used slant water observations as input into a tomography software based on the algorithm used at [32]. In [33] the linear system is solved using singular value decomposition on the original observation-state matrix without computing the normal equations. This method improved the sensitivity of the solution and allowed a more accurate determination of the water vapor.

Ulrich Foelsche and Gottfried Kirchengast developed in [34] a two dimensional, height-resolving tomographic imaging technique following the Bayesian approach for optimal combination of the information from different sources. This water vapor imaging technique combines ground based line integral measurements with an occultation profile which uses optional estimation. The authors mentioned that the occultation could be replaced by other profile data like radiosondes. The image algorithm was tested using simulated data. It was concluded that the technique is capable of reasonably reconstructing realistic atmosphere features. It was also concluded that areas with high absolute humidities and small-scale variations of water vapor density usually result in images of good quality.

In [35] LOTTOS software from [32] was used in a small scale GPS campaign (seven GPS receivers distributed within a 3 km radius). To calculate the inversion of the data collected it was used the singular value decomposition (SVD) technique described in [36]. The tomographic results were compared with radiosonde data and the agreement between the solutions was shown to be good. It was concluded that tropospheric tomography is reliable even with a reduced number of stations and that tomography is a potential tool to describe the spatio-temporal structure of the refractivity.

In 2005 a campaign denominated ESCOMPTE was used in [21] to run a GPS experiment. This GPS experiment had the following objectives: to estimate the integrated water vapor (IWV) in conjunction with the atmospheric dynamics, the study of the GPS-retrieved horizontal gradients and the development of tomography for three-dimensional reconstruction of the atmospheric humidity field. The campaign used 16 stations over the urban and northern border of Marseille within a area of approximately 20 x 20 km. Additional observations were also acquired using other type of systems: a water vapor radiometer, a solar spectrometer and a prototype water vapor Raman LiDAR. These instruments allowed independent IWV measurements for comparison and validation of the GPS results. The results of three inversions in [21] were shown to be consistent when compared with three radiosondes launches. The ESCOMPTE campaign provided the important data for the successful inversions, namely ZTD and horizontal gradient volumes of sight of the GPS satellites which were used for the calculations of the SIWV values used in the tomography.

An alternative approach to estimate the three-dimensional structure of the water vapor was presented in [37]. It uses the raw GPS phase observations. More specifically, instead of using one model for the slant delays in the GPS processing and another for the calculation of wet refractivity field, the voxel discretization of the wet refractivity in the GPS processing step is applied. The advantages of this method include that any error in the modelling of the slant wet delays in terms of zenith delays and gradients will disappear and that the number of steps required to obtain the wet refractivity field is reduced. The disadvantage is that there are many parameters that need to be estimated in the processing. The results have shown that having a spread in the station heights and using more than one GNSS will improve the retrievals of the refractivity fields. The results also indicate that the refractivity field can be obtained with an accuracy of approximately 20% or better to around 4 km with a height resolution of 1

km (provided that there are enough number of satellites and stations).

Troller *et al*. developed in [38] the software package AWATOS (atmospheric water vapor tomography software). This software is based on the assimilation of double differenced GPS observations. AWATOS applies a least-squares inversion to reveal the inhomogeneous spatial distribution of water vapor. Using this software an extensive investigation has been carried out in Switzerland using the national AGNES GPS network. For validation, 22 radiosondes profiles and the numerical weather model aLMo (alpine model in Switzerland, MeteoSwiss) were used to compare with the tomographic results. An overall goal agreement was achieved between the three methods with an root mean square (r.m.s) of better than $1.6g/m^3$ absolute humidity.

Miidla *et al*. have presented in [39] an overview of some mathematical methods for detection, monitoring and modelling of the tropospheric water vapor. It was concluded that the modelling environment works well for receiver-network geometry analysis. To finish Peep Miidla *et al*. conclude that future work will focus on data filtering and how to improve the poor voxel geometry in an optimal way, since they believe to be key issues in the construction of effective GPS-receiver networks for water vapor tomography.

Bender *et al*. developed a GNSS water vapour tomography system in [3] to reconstruct spatially resolved humidity fields in the troposphere. This system was designed to process the slant delays of about 270 German GNSS stations in near real-time with a temporal resolution of 30 minutes, horizontal resolution of 40 km and a vertical solution of 500 m or better. For the inversion Michael Bender *et al*. implemented different iterative algebraic reconstruction techniques (ART) comparing them with respect to their convergence and some numerical parameters. It was found that the multiplicative techniques (MART) provided the best results with the least time [3]. It was also found that noise added to data didn't disturbed the ART algorithms too much as long the noise did not exceed 50% of the SWV. This is a good feature since other inverse techniques often produce meaningless results over noise addition. The authors concluded that the iterative techniques were successful used in the GNSS tomography and could be parallelized to run on multicore processors or computer clusters.

In [22] Perler *et al*. introduce two new parametrizations of voxels in the Kalman filter-based GPS tomography software AWATOS 2 and their treatment in ellipsoidal coordinates. Also inter voxel constraint are presented. The ability of these parametrizations to reconstruct a four-dimensional water vapour field with high quality was demonstrated. These algorithms showed how to reduce the discretization effects without significantly increasing the number of parameters estimated. The tests performed indicated good performance of the parameterized algorithms. The accuracy increased about 10-20% for simulated data. Besides the simulations, data were also acquired from 40 GPS stations. The results using the real data also showed better performance compared to the non parameterized approach. The authors highlight that for future investigations the focus should be in the evaluation of longer periods and on comparisons with additional quantities like zenith wet delays.

3.2 CPU Algebraic Reconstruction Algorithms Parallelization

In this section we present some of the algebraic reconstruction algorithms that have been parallelized in recent years using multi-processors (CPU's).

Melvin *et al.* designed and developed a parallel ART algorithm (PART) in [40]. Its performance was studied for a network of workstations using the Message Passing Interface (MPI). In the algorithm implemented the projections data (*e.g.* delay between source and receptor) are partitioned and distributed over all processors. For each projection data of a processor, the reconstruction projections are calculated, it is also calculated an adjustment vector and the adjustments are applied to the image being reconstructed. In the last step the algorithm normalizes the images values. The major bottleneck in the algorithm was said to be caused by the communication time between processors.

In [41] a method to speed up a 3D simultaneous algebraic reconstruction technique using MPI was proposed using a parallel programming method on a computer cluster system. The amount of work is distributed to each computer node using a centralized dynamic load balancing and a work-pool scheduling scheme. This method was considered to be simple, fast and highly effective. The time cost of the reconstruction process decreases as the number of processing nodes increases. Four nodes accelerated the process 4.55 times when comparing with the sequential implementation. Also the proposed system decreased the reconstruction times up to 78% percent when comparing with a normal image reconstruction form projection in a single computer [41].

Gordon has shown in [42] that ART can be parallelized on a linear processor array. It was demonstrated that the reconstruction of an image of n pixels using $\Theta(n)$ equations can be done on a linear array of $p = O(\sqrt{n})$ processors with optimal efficiency (linear speed up) and $O(n/p)$ memory needed for each processor. This ART parallelization uses linear and rectangular arrays of processors which are also known as meshes. In this systems each processor is only connected directly to a small and fixed number of neighbouring processors. This PART implementation can be applied to various geometric models of image reconstruction and be extended to spherically symmetric volume elements (blobs) instead of voxels [42].

A tomographic reconstruction software, developed using Adaptive MPI (AMPI), was presented in [43]. AMPI is a user level thread framework that provides a solution to port legacy MPI code into a multithreaded environment where overlapping is possible (do not block objects involved in the communication).

In [43] the block iterative of component averaging methods, BICAV, has been parallelized using the Single Program Multiple Data (SPMD) approach. The volume being reconstructed is divided into slabs of slices. The slabs of slices will be distributed over the processes (MPI) or virtual processes (AMPI) for the parallelization. In MPI approach communication between the various computer nodes is involved and reducing latency becomes an issue. The AMPI library permits latency reduction using the computation-communication overlapping. In the same CPU while one thread is waiting, another one can progress with the application. Some tests were realized to compare the differences between the BICAV parallel implementation in both MPI and AMPI. The results have shown that for AMPI the walltime and cputime are almost the same while in MPI their differences may be significant, especially when increasing the number of processors. It was concluded that the threaded version of BICAV (AMPI) scales significantly better than the MPI version [43] and therefore concluded that user level threads are an attractive method for parallel programming of scientific applications.

Melvin *et al.* had examined the efficiency of the parallel ART on a shared memory machine on the Western Canada Research Grid Consortium [9]. Two algorithms using multithreading on the shared memory machine were implemented. OpenMP library were used for the parallel

implementation. Some tests were also executed to time the performance of the parallel algorithms. The results have shown a clear advantage with a greater number of threads. With the increasing of the number of processors the speed up also increased, with efficiency ranging from 59.52% to 96.75% [9]. A six processor IBM P-server reconstructs an image from 36 angles in approximately 5.038 seconds, with an efficiency of 93.35%. This means that a parallel ART algorithm can reconstruct the same image, in about the same time as a 180 angles sequential FBP (Filtered Back Projection) reconstruction, with about the same image quality and less radiation. With the same radiation the 5.038 parallel algorithm produces a higher quality image in approximately the same time [9].

In [44] a Modified Simultaneous Algebraic Reconstruction (MSART) was implemented. This algorithm implements a back projection technique (BPT) and an adaptive adjustment of corrections. The experimental results have shown that MSART can improve significantly the quality of reconstruction. A strategy to parallelize the MSART algorithm on DAWNING 4000H cluster system were also implemented. The parallelization of MSART has made for 3D volumes on cluster systems. Basically the 3D reconstruction problem is decomposed into a set of independent 2D reconstruction problems. The algorithm combines both MPI and OpenMP libraries. The MPI library is used to perform the coarse-grained parallelization of the reconstruction. OpenMP is used to perform the fine-grained parallelization [44]. This hybrid implementation has proven to be better than only the MPI one [45]. In the algorithm each slabs of slices is assigned to an individual node in the cluster and the reconstruction is made in parallel. The nodes communicate with each other only to complete the final result of the 3D reconstruction [44]. The results have shown that the speedup decreases with the increasing of the number of computer nodes. This is justified by the increase of the communication time between the nodes since the number of nodes also increase [44]. Overall the results on DAWNING 4000H cluster system show that the implemented 3D reconstruction parallel algorithm can achieve high and stable speed ups [44].

Xu and Thulasiraman have implemented a parallelized ordered subset [46] SART (OS-SART) on cell broadband engine [47]. The algorithm takes advantage of Cell BE architecture, using the SPEs (Synergistic Processing Elements) coprocessors to compute the fine grained independent tasks and PPE (Power Processor Element) performs the tasks of data distributing and gathering. The overlap of the computation and communication is realized with direct memory access available on the Cell BE, this to reduce the synchronizing and communications overheads [47]. The algorithm consists in four parts: forward projection, rotating the image, back projection and creating the reference matrix. The algorithm has tested in two different architectures: Cell BE and Sun Fire x4600. The more time consuming parts of the algorithm are forward and back projections the total complexity of the algoritm is $O(n^3)$ which turns the OS-SART in a computation intensive algorithm. This algorithm is also memory intensive consisting in a complexity of $O(n^4)$ [47]. Cell BE is a PowerXCell8i within a IBM QS22 Blade. This computer runs at 3.2 GHz and contains 16 GB of shared memory. The sunfire x4600 machine is a distributed shared memory system with an eight AMD dual-core opteron processor resulting in a total of 16 cores. Each core runs at 1 GHz with 1 MB cache per core and 4 GB memory per processor. In this system OpenMP was used for the implementation. A series of tests were realized in [47] comparing the execution time, number of subsets, number of cores, number of SPEs and the number of images rows per DMA (Direct memory access) transfer. The tests have shown that one drawback of the Cell BE is the limited memory storage on each of the SPEs. The approach the authors used to circumvent this problem were to use a rotation based algorithm which calculates the projection angles using less memory. However this rotation based algorithm

increases the number of transfers required to DMA from main memory to local memory on the SPE, which turned to be a bottleneck how the number of SPEs increased. Even with this bottleneck the Cell BE performed much better than the shared memory machine [47]. The tests also exposed that the number of subsets impacts the sequential processing time on one SPE. The Cell-based OS-SART on one SPE was five times faster than OS-SART on AMD opteron core for one subset and one iteration. With the increase of the number of subsets the speedup also increased. The authors referred that a future implementation the algorithm will use double buffering, in order to reduce the DMA transfers impact.

3.3 GPU Algebraic Reconstruction Algorithms Parallelization

Algebraic reconstruction algorithms are highly computational intensive. With this in mind various hardware platforms have been tested along the years including the Graphics Processing Units (GPUs) discussed in this section.

In 2000, Klaus Mueller and Roni Yagel made an implementation in OpenGL of the SART reconstruction technique using 2-D texture mapping hardware in [48]. This implementation was used for rapid 3-D cone-beam reconstruction. It was found that the graphics hardware allowed 3-D cone beam reconstructions with speed ups of over 50 comparing with the implementation on a epoch's CPU. The graphics hardware version of SART developed was named texture-mapping hardware accelerator SART (TMA-SART). Both the software implementation (SART) and the hardware implementation (TMA-SART) used as test image the 3D-extension of the Shepp-Logan phantom [49]. The projections were obtained by analytical integration of the phantom and were used to reconstruct a 128^3 volume in three iterations. Two graphic architectures groups were used in the experiments: an epoch's mid-range workstation, such the SGI Octane with a 12-bit framebuffer and low-end graphics PCs and graphics boards that only had a 8-bit framebuffer. The experiments indicated that the first group had speed ups between 35 and 68 when compared to an optimized CPU version of the SART algorithm (in the same system CPU) [48].

In 2005, Fang Xu and Klaus Mueller have shown in [50] how the PC graphics board technology had an enormous potential for the computed tomography. The authors decomposed three popular three-dimensional (3D) reconstruction algorithms (namely Feldkamp filtered backprojection, the simultaneous algebraic reconstruction technique and the expectation maximization) into modules which could be executed on the CPU and their output linked internally. As an added bonus the visualization of the reconstructions is easily performed since the reconstruction data are stored in the graphics hardware, which allowed to run a visualization module at any time. Several tests were performed on the three algorithms developed, the Feldkamp filtered backprojection, the simultaneous algebraic reconstruction technique and the expectation maximization. This test were performed on a 2.66 Ghz Pentium PC with 512 memory and a NVIDIA FX 5900 GPU. A 3D version of the Shepp-Logan phantom with size of 128^3 was used for the reconstruction tests. The results have shown that the epoch's inexpensive floating point GPUs could reconstruct a volume with SART in about 12 times faster than the same generation CPUs and five times faster than the older SGI hardware [48]. It was noticed that the projections were much faster than the back projections. The authors concluded that the GPU implementation of Feldkamp FBP produced excellent results. The GPU Feldkamp FBT reconstruction was seen to be very fast taking 5 seconds to reconstruct a 128^3 volume with good quality. The au-

thors stated that for the first time the quality of the GPU reconstruction could rival that of the CPU implementations. Also excellent GPU performance could be achieved in relatively cheap equipment ($500 for the epoch's).

Keck *et al*. have applied in 2009 the Common Unified Device Architecture (CUDA) to the SART algorithm in [51]. Two different implementations were made one using CUDA 1.1 and another using CUDA 2.0. Until this implementation the iterative reconstruction on graphics hardware used OpenGL and Shadding languages [50] [48]. With CUDA is possible to use the standard C programming language to program in a graphics hardware without any knowledge of graphics programming [51].

The back projection component of the SART algorithm performs a voxel-base back projection. The matrix-vector product is calculated for each voxel to determine the voxel corrective projection value.

The forward projection is parallelized using each thread to compute one corrective pixel of the projection. In both the back and forward projection the CUDA grid sizes is chosen based on experimental results [52].

After the implementation of the SART in the GPU (CUDA 1.1 and CUDA 2.0) and in the CPU some experiments were made to evaluate the performance of each method. The GPU used for the experiments were a NVIDIA Quadro FX 5600. For the CPU two different hardware were used: an Intel Core2Duo 2 GHz PC and a workstation with two Intel Xeon Quadcore 2.33 GHz. The image used for the reconstruction had a size of 512x512x350 [51].

Using CUDA 1.1 the implementation took approximately 1.15 seconds for a single texture update [51]. Using 3D textures the authors measured 0.11 seconds for the texture update on CUDA 2.0 which improves the performance by a factor of 10 [51].

The slowest GPU SART implementation revealed to be CUDA 1.1 even so it still was 7.5 times faster than the PC and 1.5 times faster than the workstation. Ordered subsets [46] optimizations in GPU had also been tested allowing even faster CUDA implementations. The CUDA 2.0 implementation with the ordered subset was 64 and 12 times faster compared to the PC and workstation respectively.

The authors concluded that GPU-OS CUDA 2.0 implementation was already applicable for specific usage in clinical environment, since this reconstruction was less than 9 minutes [51].

In [53] is presented some comparisons between GPU and CPU implementations of some iterative algorithms are presented: Kaczmarz's, Cimmino's, component averaging, conjugate gradient normal residual (CGNR), symmetric successive overrelaxation-preconditioned conjugate gradient and conjugate-gradient-accelerated component-averaged row projections (CARP-CG). Elble *et al*. have made a preliminary search to find the most appropriate algorithms to be implemented in the GPU [53]. It became clear that the Kackzmarz algorithm was not the most appropriate, the only available steps that can be parallelized are the dot product and the vector addition. Due to this it could only be superior in terms of performance when using very large and dense linear systems. The BLAS library was used in the GPU for this hardware parallelization. It parallelizes the algebra operations like matrix-matrix or matrix-vector multiplications. The authors have made a series of tests using the various algorithms implementations both on CPU and GPU. The GPU implementations were processed on a NVIDIA Tesla C870 system and the CPUs implementations on a PC with a pentium IV 2.8 GHz processor with 1 GB memory. Also a

linux cluster with 16 of this PCs connected by a 1 Gbit/s Ethernet switch were used [54]. The results have shown that CGNR was the most efficient algorithm for solving investigated partial differential equations on the GPU. The CAV algorithm highest GFLOPS but its slow convergence resulted in the slowest solution time [53]. The GPU had a clear advantage over the CPU, with computational results from five to twenty times faster than the CPU. On the 16 node cluster the GPU implementation was up to three times faster [53]. As conclusion the authors mentioned that the GPU offered a low-cost and high-performance computation system to solve large-scale partial differential equations [53].

In [55] some benchmarks tests were realized to determine the optional parameters in the iterative algorithm OS-SIRT. The tests were processed on NVIDIA 8800 GT GPU and were programmed GLSL. This tests showed to be decisive for obtaining optimal GPU performance also they show that iterative reconstruction is a clear option for noisy and few-view scenarios.

Johnston *et al*. presents an implementation of family of iterative reconstruction algorithms with total variation (TV) denoising on a GPU (NVIDIA Geforce GTX 280). A series of tests to optimize and compare the efficiency of each algorithm to reduce the artefacts were also executed using a generated 256^3 volume image. The results have shown that SART can achieve better performance than other implementations, namely steepest descent (SD) and the non linear conjugate gradient algorithm (CG).

In [56] implements the SART algorithm with motion compensation for a fast high quality computed tomography reconstruction using a CUDA GPU. Several tests were made to the implemented algorithm in order to test its performance, reconstruction quality and effect of the parameters. These tests were performed on a desktop PC composed of a Core2Quad Q6600 CPU (2.4 GHz), 4 GB memory and a NVIDIA 8800 GTX GPU with 768 MB ram and CUDA 2.0 [56]. The Shepp-Logan Phantom was chosen as the image to be reconstructed with a grid size of 128^3 voxels and from a set of 80 projections. The CPU implementation took nearly 18 minutes to do the reconstruction while the GPU implementation took 7.52 seconds for ten iterations being almost 150 times faster than the CPU implementation [56]. The quality of the reconstruction this was evaluated using peak signal-to-noise ratio (PSNR), line profiles, visually and with metrics like signal-to-noise ratio (SNR) and Mean Square Error (MSE). All this evaluations revealed satisfactory results [56]. The authors concluded that this SART GPU implementation allows instantaneous presentation of 3D CT volume to physicians once the projections are collected [56].

A distributed multi-GPU system (CUDA) was developed in [57]. The stated aim was to provide computing power for rapid constrained, iterative reconstruction of very large volumes. Shawn Q. Zheng *et al*. performed diverse tests with a system composed of 5 nodes where each node contained one GTX 295 card. Each GTX 295 card contains 2 GPUs resulting in a system composed by 10 GPUs. This system was connected using a high speed gigabit Ethernet network. For 10 iterations of SIRT reconstruction using a tomogram of 4096^2 x 512 from a input tilt series containing 122 projection images of 4096^2 pixels (using single precision) took a total of 1845 seconds of which 1032 seconds are pure computation while the remainder are from the system overhead. The same distributed multi-GPU system took only 39 seconds to reconstruct 1024^2 x 256 voxels from $122\ 1024^2$ pixel projections. The authors concluded in [57] that even with the additional system overhead the performance analysis indicated that adding extra GPUs would improve the overall performance.

Alexey Buzmakov *et al*. presents in [58] a fast version of RegART [59] which uses the NVIDIA

CUDA technology for the improved performance.

The implementation in CUDA uses a rotation technique involving 2D texture fetching [60] in order to reduce the memory usage. Two roughly independent stages are included in the bilinear rotation algorithm. One its the calculation of the exact coordinates on the source image and another it's the bilinear interpolation of the value over this point [58].

The authors have realized some tests for performance comparison and reconstruction quality comparison with the Filtered Back Projection (FBP) method. For this tests was used a PC with 2 AMD Opteron 275, 8 GB memory, NVIDIA GTX 285 with 240 cores executing Ubuntu Linux 9.10 64 bit as operating system. It was used double precision for the computations and Shepp-Logan phantom as image to reconstruct.

For a image of size 1500x1500 the CUDA implementation obtained a speed up of 6.6 relatively to the CPU implementation.

For the quality test a scanline of the reconstructed phantom was used, in this graph it is possible to see that the RegART is superior to FBP, resulting in fewer fluctuations of the abortion coefficient [58].

The authors concluded that the CUDA RegART algorithm allows its use in real tomography hardware while further optimization would be a plus to compete with FBP in computational efficiency.

In [61] it is demonstrated that making alternative design decisions in the GPU implementation could result in an additional speed up of an order of magnitude comparing with the Xu *et al.* implementation [62]. It is mentioned in [61] that as the bandwidth available for reading data from the GPU global memory it is necessary to make use of the shared memory and cached memory, even if it requires extra synchronization or memory writes. This especially important since tomography algorithms are very memory-read intensive. The projection and back projection are the steps more computational intensive so they were optimized using diverse techniques that exploit the memory locality. Same tests were carried to compare this implementation with Xu *et al.* implementation on [62]. In this tests two different generations of NVIDIA GPUs were used. A GTX 280 Geforce which contains a GT 200 GPU and was launched in 2008. This is the same GPU used by Xu *et al.* in [62]. The other generation used consisted in a Geforce GTX 480 and a Tesla C2070 both containing a Fermi GPU and launched in 2010. The results have shown that the optimizations that exploit the memory locality had an important impact on the running time of the projection and back projection steps of the GPU algorithm. Also a speed up of about 10 was obtained comparing with the results in [62] by Xu *et al.* using similar hardware [61].

3.4 Hybrid CPU and GPU Algebraic Reconstruction Algorithms Parallelization

Besides the CPU and GPU only implementations for the algebraic reconstruction other hybrid solutions have emerged using both of this hardware.

In [63] a hybrid approach is presented that takes full advantage of the whole computer power available in modern computers and that further reduces the processing time. This approach

makes the decomposition of a 3D reconstruction problem into a set of independent 2D reconstruction subproblems corresponding to the slices perpendicular to the tilt axis. In this implementation the slices are clustered in slabs of four slices to reduce the latencies and overheads. A pool of slabs to reconstruct is distributed over CPU (C-threads) and GPU (G-threads). As the threads become idle they are assigned to a new slab to reconstruct, the process is repeated until all slabs are processed. The authors explain in [63] that because GPU computation is much faster than in CPU more slabs are assigned to the GPU threads than to CPU threads. The authors also stated that for GPU was used a matrix approach (introduced in [64] that uses sparse structures to improve performance) of the iterative methods while in CPU it was used a vectorized and optimized implementation (taking advantage of SSE, Streaming SIMD Extensions).

The results in [63] have show that the hybrid approach improves the performance over 1.5 to 2 times when comparing with strategies based on pure CPU or GPU implementations.

Another hybrid approach was presented in [65], this approach besides the CPU and GPU hardware also includes a field-programmable array (FPGA). Here a new iterative reconstruction algorithm based on expectation maximization and total variation nominated as EM+TV is described.

In [65] the performance of the proposed hybrid architecture revealed to be the best when comparing with CPU and GPU only architectures. This architecture also delivers the minimum energy consumption, resulting in less than $1/3$ of the CPU only implementation and $1/5$ of the GPU only implementation.

3.5 Summary

In this chapter the state of the art of GNSS Water Vapor Tomography and Algebraic Reconstruction Algorithms Parallelization was presented. This chapter was important to understand what was been done in these areas and which approaches were followed. In the next chapter the work done in this dissertation will be presented for the parallelization of the algebraic reconstruction algorithms.

Chapter 4

Parallelizing Algebraic Reconstruction

One of the main objectives of this dissertation was to accelerate the algebraic reconstruction algorithms reducing their processing time and thereby permitting real-time applications. Algebraic algorithms consist in a series of linear algebra operations as presented in Chapter 2. If we parallelize these algebra operations we automatically parallelize the algebraic reconstructions algorithms. This was the approach used in dissertation for the parallelization of the algebraic reconstruction algorithms. The linear algebra operations parallelized were the vector-matrix operation and the matrix-matrix operation. The programs were written and tested in regular personal computers (pcs) which are accessible to every user.

4.1 Multi-threading Libraries

There are several multithreading libraries for C++. The most elemental approach is to use a low threading library such as Pthreads (Posix Threads) or Boost, common higher level libraries are OpenMP and Intel Threading Building Blocks (TBB) [66]. This work aims to use and then compare the popular OpenMP and TBB libraries, since they provides a high level of abstraction of the implementation and platform plus they are known to be stable [67].

4.1.1 OpenMP

OpenMP (OMP) is a collection of compiler directives, library routines and environment variables for shared-memory parallelism in C, C++ and Fortran programs. OMP is managed by the non-profit organization OpenMP Architecture Review Board whose members include: AMD, Fujitsu, HP, IBM, Intel, Microsoft, NEC, NVIDIA, Oracle Corporation, Texas Instruments and others [68]. The OMP API requires that the programmer explicitly write the actions to be taken to execute the program in parallel. While aiding the coding when compared to lower level API's by introducing a high level of abstraction, it is not an automatically parallel programming model. OMP doesn't check for data dependencies therefore conflicts such as race conditions, deadlocks and other multithreading problems may still occur [69]. It is up to the programmer to ensure correctness and deal with these specific problems. OMP supplies primitives for locks and other multithreading primitives. OMP is included in many compilers by default including (but not limited to) GCC, XL C/C++ / Fortran, Visual Studio 2008-2010 C++ and nagfor [70].

An example of the OMP library is the following extracted code from the multiplication algorithm:

Listing 4.1 Example of use of the OMP library

```
1  #pragma omp parallel for
2  for (int i=0; i<M1->rows-1;i+=2){
3          int k;
4          register double s00,s10;
5          s00=s10=0.0;
6          for (k=0;k<M2->rows;k++){
7                  s00 += M1->theMatrix[i] [k] v1[k];
8                  s10 += M1->theMatrix[i+1][k] v1[k];
9          }
10         result->theMatrix[i] [j] =s00;
11         result->theMatrix[i+1][j] =s10;
12 }
```

Notice the *#pragma omp parallel for* directive which automatically parallelize the for loop using the library.

4.1.2 Intel Threading Building Blocks

Intel Threading Building Blocks (TBB) is another library that helps the programmer write parallel programs that take advantage of multicore processors that is also portable and scalable. Using TBB the programmer may build general purpose parallel programs and doesn't require any special languages or compilers. It is based on the use of C++ templates aiming to be simultaneously flexible and efficient. TBB contains parallel algorithms and data structures it also provides scalable memory allocation and task scheduling [71].

The library uses tasks instead of threads and it internally maps the tasks onto threads [72].

TBB according to [72], was built with performance in mind, allowing computational intensive work parallelization. TBB is also compatible with other thread packages (for instance it is possible to write mixed OMP and TBB code).

An example of implemented code is given below which implements the same vector-matrix multiplication algorithm as was used for the OMP example (the details of dealing with the cases when the number of rows is odd are not shown):

Listing 4.2 Example of use of the TBB library

```
mMatrix M;
mVector V,result;

void operator()( const blocked_range<int>& range ) const {

        int i,j;
        register double s00,s01;

        for (i=range.begin(); i < range.end(); i+=2){
                s00=s01=0.0;
                for (j=0; j<M->columns;j++){
                        s00 += V->theVector[j] M->theMatrix[i][j];
                        s01 += V->theVector[j] M->theMatrix[i+1][j];
                }
                result->theVector[i] =s00;
                result->theVector[i+1] =s01;
        }

        //..case row number is not an even number
}

mVector Math::matrixVectorMultiplication(mMatrix M, mVector V){

        mVector result=new mVector(M->rows);
        int i,j;
        ITBvectorMatrixMultiplication myVMmultiplication;

        myVMmultiplication.result=result;
        myVMmultiplication.M=M;
        myVMmultiplication.V=V;

        parallel_for( blocked_range<int>(0, M->rows-1), myVMmultiplication );

        return result;
}
```

Notice the need of creating a new struct which is used for parallelizing the same loop as used in the OMP directive. This struct is then initialized in the main function and all variables it needs are then assigned. A call to the *parallel_for* function is made in the end so the for is executed using threads.

4.1.3 CUDA

CUDA (Compute Unified Device Architecture) is a general purpose parallel computing platform and programming model which uses the parallel compute engine in NVIDIA GPUs to solve complex computational problems. The CUDA was introduced in November 2006 by NVIDIA [73].

The CUDA Toolkit [74] comes with a software environment that allows the developers to use the C language to develop CUDA programs [73].CUDA allows the use of various languages, application programming interfaces and directives-based approaches such as FORTRAN, DirectCompute and OpenACC (Table 4.1) [73].

Table 4.1: Programming languages and applications programming interfaces supported by the CUDA platform.

CUDA GPU Computing Applications						
Libraries and Middleware						
cuFFT cuBLAS cuRAND cuSPARSE	LAPACK CULA MAGMA	Thrust NPP cuDPP	VSIPL SVM OpenCurrent	PhysX OptiX	Iray RealityServer	MATLAB Mathematica
Programming Languages						
C	C++	Fortran	Java Phyton Wrappers	DirectCompute	Directives (e.g. OpenACC)	
CUDA-Enabled GPU						
Kepler Architecture (compute capabilities 3.x) Fermi Architecture (compute capabilities 2.x) Tesla Architecture (compute capabilities 1.x)						

The CUDA applications of this dissertation used the C++ language and the CUBLAS library [75] (see subsection 4.2.3).

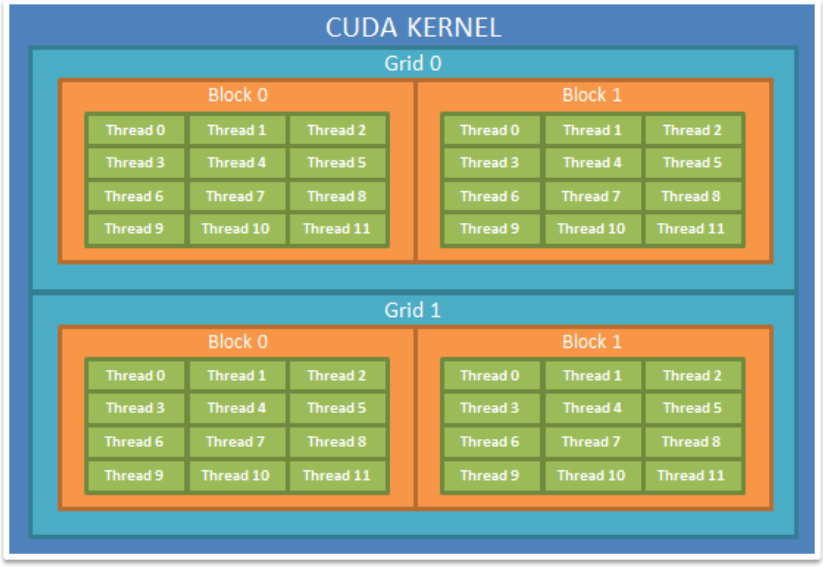

Figure 4.1: Example of CUDA kernel hierarchy.

The CUDA parallel model was designed to transparently scale its parallelism to a different number of GPU cores [73]. In this model the host (CPU) program launches a sequence of kernels which are GPU functions. Each kernel is organized as an hierarchy of threads. A group of threads is called block and a group of blocks is called grid. Each thread has a unique identifier (id) within its block and each block has a unique id within its grid. Figure 4.1 illustrates this CUDA hierarchy.

4.2 Underlying linear algebra libraries

4.2.1 Basic Algebra / Math library

For this work a basic computational linear algebra library was written with some essential functions like matrix-matrix multiplication, vector-matrix multiplication, scalar-vector division, matrix creation, vector creation. An ART and SIRT library was also written with the diverse algorithms used in algebraic reconstruction algorithms including: kaczmarz, symmetric kaczmarz, randomized kaczmarz, landweber, cimmino, component averaging, diagonally relaxed orthogonal projections and simultaneous algebraic reconstruction technique. Not all these algorithms have been used for the results of this dissertation but they were all implemented.

The language used to write these libraries was C++. C++ provides some advantages over other languages such as being an object-oriented programming which is well suited for complex systems [76]. It's also a compiled language which means it's usually faster than the interpreted and JIT-compiled languages. It's an open ISO-standardized language. Is portable, existing diverse compilers for the various operating systems. It's compatible with C code, which means that we can write C code with few or no modifications plus user external C libraries. C++ have also a great library support (including multithreading) with over 3000 results in the project management website SourceForge [77].

Because of all these advantages this was the language which was chosen to write the libraries and the tests discussed in this dissertation.

In order to illustrate what kind of libraries have been produced the headers of the Linear Algebra and Algebraic Reconstruction algorithms are shown below:

Listing 4.3 Header files of the implemented Linear Algebra / Math library

```
1  struct mVector{...}
2  struct mMatrix{...}
3  class Math {
4  public:
5          Math();
6          virtual ~Math();
7
8          static vector<int> retreiveNonZeroRows(mMatrix myMatrix);
9          static void retreiveSjVector(mMatrix A, mVector result);
10         static mVector scalarVectorSum(mVector V, double scalar);
11         static mVector scalarVectorMultiplication(mVector V, double scalar);
12         static mVector scalarVectorDivision(double scalar, mVector V);
13         static mMatrix scalarMatrixMultiplication(mMatrix M, double scalar);
14         static mMatrix scalarMatrixDivision(mMatrix M, double scalar);
15         static mMatrix matrixMatrixMultiplication(mMatrix M1, mMatrix M2);
16         static mVector matrixVectorMultiplication(mMatrix M, mVector V);
17         static void vectorSum(mVector V1, mVector V2, mVector result);
18         static mVector vectorSubtraction(mVector V1, mVector V2);
19         static void vectorMultiplication(mVector V1, mVector V2, mVector result);
20         static void cumulativeVector(mVector myVector, mVector result);
21         static void transposeMatrix(mMatrix M,mMatrix result);
22         static void normalizeVector(mVector V, mVector result);
23         static mVector powerMethod(mMatrix A);
24         static void diagMatrix(mMatrix M, mMatrix result);
25         static void diagVector(mVector V, mMatrix result);
26         static void showMatrix(mMatrix A);
27         static void showVector(mVector V);
28         static double level1Norm(mVector myVector);
29         static double euclidianNorm(mVector myVector);
```

```
30          static double euclidianNorm(mMatrix myMatrix);
31          static double dotProduct(mVector firstVector, mVector secondVector);
32          static double getMaxOfVector(mVector A);
33          static double getMaxAbsoluteOfVector(mVector V);
34          static double spectralRadius(mMatrix myMatrix);
35
36  };
```

A simple example of creating two matrices and multiplying them with this library is:

Listing 4.4 Example of multiplication of two matrices using the developed algebra/math library

```
1  mMatrix A = new mMatrix(3,3);
2  A->setMatrixToRand();
3  mMatrix B = new mMatrix(3,3);
4  B->setMatrixToRand();
5
6  mMatrix result;
7
8  result=Math::matrixMatrixMultiplication(A,B);
```

These libraries can be later extended to support additional math/algebra operations.

4.2.2 Eigen3 library

The Eigen3 library is described as a "C++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms" [78].

The algebraic reconstruction methods were implemented again using Eigen3. As it is a linear algebra library it was not necessary to used the developed basic math/algebra library.

This library implements both BLAS/LAPACK. BLAS has a complete implementation while LAPACK has a partial implementation [79].

This is versatile, supports all matrix sizes, standard numeric types, various matrix decompositions and geometry features, includes user modules that provides many specialized features like matrix functions and polynomial solver. It is also a fast library which includes explicit vectorization support performed for SSE 2/3/4, ARM NEON and AltiVec instructions. Fixed-size matrix are optimized. The Eigen3 algorithms are selected for reliability, it's tested with its own test suit, with the BLAS test suite and also with parts of the LAPACK test suite.

This library supports two different type of storages for matrices: row-major and column-major. Row-major means that the matrix is stored row by row, the first row is stored first, followed by second, the third and so on. Column-major in other side stores the matrix column by column, first stores the first column, then the second, the third and so on. By default the matrices store the data with column-major [80].

A illustrative example of the column-major and row-major is the follow.

Being A the matrix:

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

Storing A in the row-major order in memory it will be organized as follow:

1 2 3 4 5 6 7 8 9

However storing A with the column-major order in memory will organize it as:

1 4 7 2 5 8 3 6 9

Eigen3 also includes support for parallelization. This feature will be explained in the subsection 4.3.3.

4.2.3 CUBLAS library

CUBLAS (CUDA Basic Linear Algebra Subroutines) is a GPU accelerated version of the complete standard BLAS library implemented on the top of the NVIDIA CUDA runtime [75]. According to [81] it delivers 6 to 17 times faster performance than the latest MKL BLAS [82] which is a commercial highly optimized CPU BLAS implementation.

To use CUBLAS is necessary to first allocate the matrices and vectors on GPU memory, fill them with the data, call the desired CUBLAS functions and copy the results back to the computer main memory [75].

This library uses column-major store format (see Section 4.2.2) and 1-based indexing (meaning that the arrays begin at 1 position) for maintaining the maximum compatibility with the existing FORTRAN environments [75]. This needs to be taken in account when using C or C++ languages with CUBLAS (since they use row-major format and 0-based indexing).

An example of a simple matrix-matrix multiplication using CUDA is presented below. These matrices are first initialized in the CPU (C_A and C_B variables) and then copied to the GPU (G_A and G_B variables):

Listing 4.5 Simple CUDA CUBLAS multiplication

```
1
2       ... // Code to create and allocate CPU C_A and C_B arrays
3       ... // Code to create and allocate GPU G_A, G_B and G_C arrays
4       ... // Code to copy the CPU C_A and C_B arrays to GPU G_A and G_B arrays
5
6       int lda=m; //leading dimension of two-dimensional array used to store the matrix A.
7       int ldb=k; //leading dimension of two-dimensional array used to store matrix B.
8       int ldc=m; //leading dimension of a two-dimensional array used to store the matrix C.
9       const double alf = 1; //<type> scalar used for multiplication.
10      const double bet = 0; //<type> scalar used for multiplication. If beta==0, C does not have to be a valid input.
11      const double alpha = &alf;
12      const double beta = &bet;
13
14      cublasOperation_t transa, transb;
15
16      transa = transb = CUBLAS_OP_N;
17
18      //cuBlas call, result is stored in G_C
19      cublasDgemm(handle, transa, transb, m, n, k, alpha, G_A, lda, G_B, ldb, beta, G_C, ldc);
```

The result of this multiplication is stored in the G_C GPU variable.

4.2.4 Other linear algebra libraries

Before choosing Eigen3 and CUBLAS libraries (besides our own algebra/math library) other libraries were investigated. This libraries were respectively BLAS, LAPACK, ScaLAPACK and CVM and will be introduced in this section.

4.2.4.1 BLAS

BLAS (Basic Linear Algebra Subprograms) are routines written in the FORTRAN language that provide basic operations over vectors and matrices [83]. Computer programmers have frequently chosen to implement some operations such as dot product or matrix-vector product as separate subprograms. The BLAS approach encourages structured programming, improves the self-documenting quality of the software by using basic building blocks and identifying these operations with mnemonic names. Also these operations are optimized resulting in faster execution time programs [84].

Currently BLAS has 3 operations levels. The level 1 implements scalar and vector operations, level 2 implements matrix-vector operations and level 3 implements matrix-matrix operations [84].

There are machine-specific optimized BLAS libraries for a diverse number of computer architectures [83]. Alternatively the user can use ATLAS (Automatically Tuned Linear Algebra Software) to automatically generate a BLAS library optimized for his architecture [83].

A C interface is available that allows the use of the BLAS in C/C++ code [83].

4.2.4.2 LAPACK

LAPACK (Linear Algebra PACKage) is a set of routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems and singular value problems. There are also other operations included such as associated matrix factorizations and reordering Schur factorizations. It also handles dense and banded matrices but not general sparse matrices [85]. It supports real and complex matrices in both single and double precision [83]. LAPACK routines are written on the top of BLAS making much use as possible of the BLAS calls for the computations.

LAPACK was written in FORTRAN and a standard C language interface is available to use LAPACK routines in C/C++ code [85].

4.2.4.3 ScaLAPACK

ScaLAPACK (Scalable Linear Algebra PACKage) is a set of high-performance routines for solving linear algebra problems in parallel distributed memory machines [86]. It was initially investigated for this dissertation because of the initial possibility of use parallel distributed memory machines to use in the algebraic reconstruction algorithms. ScaLAPACK solves dense and banded linear systems, least squares problems, eigen values problems and singular value problems. ScaLAPACK uses three libraries as function basis: BLAS, LAPACK and BLACS (Basic Linear Algebra Communication Subprograms).

The goals of ScaLAPACK are to provide efficiency, scalability, reliability, portability, flexibility (build new routines from the existing ones) and ease-of-use (by making LAPACK and ScaLAPACK look as similar as possible) [87].

This library is currently written in FORTRAN language but has some exceptions routines written in C [86].

4.2.4.4 CVM

The CVM class library is a C++ class library that encapsulates the concept of vector and matrix (including sparse, band, symmetric and hermitian) supporting single and double precision numbers. It internally uses the BLAS and LAPACK routines to achieve the best performance possible [88]. Besides the basic vector and matrix operations like vector and matrix products it contains other algorithms like algorithms to perform norms, solve linear systems of the kind $Ax = b$ and $AX = B$, singular value decomposition, matrix rank and determinant computation [88]. This library simplifies the code that deals with matrices and vectors for example multiplying a vector with a matrix is as simple as the following code:

Listing 4.6 Vector-matrix multiplication code using CVM library

```
1  rvector a(10), b(20);
2  rmatrix A(20,10);
3  // Fill the vector / matrix with random numbers between -10 and 10
4  a.randomize(-10.,10.);
5  A.randomize(-10.,10.);
6  b = A a;
```

4.3 Linear Algebra Parallelization

In this section it is presented the linear algebra libraries used for the parallelization, their implementation and the results obtained from this parallelization. The parallelization of the linear algebra operations also allow the parallelization of the algebraic algorithms (since they consist in linear algebra operations).

The linear algebra libraries choose to implement besides our own algebra/math library were the Eigen3 library and the CUBLAS library. Note that all these three chosen libraries are free to use even on commercial projects [78] [89].

Eigen3 was chosen in relation with BLAS/LAPACK because it is much easy to implement complex programs due to the simpler syntax and internally it also uses the BLAS/LAPACK routines so comparable performance is achieved.

ScaLAPACK was not chosen because it was developed to be used in parallel distributed machines which is an approach not used in this dissertation (while it was thought).

Due to Eigen3 support parallelization in matrix-matrix multiplications it was chosen in relation with CVM. CVM also provides an easy interface for BLAS/LAPACK like Eigen3 but misses any automatically parallelization.

CUBLAS was also chosen to implement because it allows one to experiment a different approach than the other libraries. It parallelizes BLAS/LAPACK routines in GPU which is interesting to compare with other CPU results.

The algorithms used in our own written algebra/math library will be described next, explaining the implementations and optimizations.

The matrix-matrix algorithm pseudo-code used in OMP and TBB implementations is presented as follows :

Listing 4.7 Matrix-matrix algorithm pseudo-code used in OMP and TBB implementations

```
1  Result <— Zero
2  for (j = 0; j<=M2—>columnsNumber—1; j+=2)
3          for (i = 0; i<=M2—>rows; i++)
4                  Store M2—>column(i ) in Vector1
5                  Store M2—>column(i+1) in Vector2
6          for (i = 0; i<=M1—>rowsNumber—1; i+=2)
7                  for (k = 0; k<=M2—>rowsNumber; k++)
8                          Result(i,j) = Result(i,j) + M1(i,k) Vector1(k)
9                          Result(i,j+1) = Result(i,j+1) + M1(i,k) Vector2(k)
10                         Result(i+1,j) = Result(i+1,j) + M1(i+1,k) Vector1(k)
11                         Result(i+1,j+1) = Result(i+1,j+1) + M1(i+1,k) Vector2(k)
12
13         //..case row number is not an even number
14
15 //..case columns number is not an even number
16
17 return Result
```

Where M1 and M2 are the first and second matrices respectively and Result the result matrix of the multiplication. The details of dealing with the cases when the number of rows or columns is odd are not shown.

The procedure can be described by the following steps:

1. For each two columns in M2

   1.1. Store the two columns in two vectors

   1.2. For each row in M1

      1.2.1. Multiply each element of the row by each element of the vector 1 and vector 2

      1.2.2. Store the result of the multiplications in the result matrix

The matrix-vector algorithm pseudo-code used in OMP and TBB is presented as follow:

Listing 4.8 Matrix-vector algorithm pseudo-code used in OMP and TBB implementations

```
1  for (i = 0; i<=M->rows-1; i+=2)
2          for (j = 0; j<=M->columns; j++)
3                  Result(i) = Result(i) + V(j) M(i,j)
4                  Result(i+1) = Result(i+1) + V(j) M(i+1,j)
5
6  if (M->rows%2 is 1) then
7          set i to M->rows-1
8          for (j = 0; j<=M->columns; j++)
9                  Result(i) = Result(i) + V(j) M(i,j)
10
11 return Result
```

Where M is the matrix, V the vector and Result the matrix of the multiplication.

The procedure can be described by the following steps:

1. For each two rows in M
   1.1. Store the two rows in two vectors
   1.2. Multiply each element of the vector V by each element of the vector 1 and vector 2
   1.3. Store the result of the multiplications in the result vector

Some standard optimizations were made to these algorithms. For instance the explicit use of the registers via the register keyword for the sums of the multiplications, since this avoids accessing main memory every time the sum is updated. An example of this register optimization is the following:

Listing 4.9 Example of the use of the register keyword

```
1          register double s00;
2          s00=0.0;
3          for (k=0;k<M2->rows;k++){
4                  s00 += M1->theMatrix[i] [k] v1[k];
5          }
6          result->theMatrix[i][j] =s00;
```

Notice the use of the *register* keyword in the multiplication accumulator.

This technique increases the program speed by eliminating the instructions that control the loop, it also reduces the delay from reading data from the memory [90] [91].

The last optimization was to divide the matrices into vectors (known as block structure). Since vectors can be much smaller than a matrix they can fit or partially fit into cache memory which may decrease the time to retrieve their elements. This also makes the multiplications faster [92].

In the matrix multiplication one of the matrices were divided into columns vectors while in the vector multiplication the matrix were divided into rows vectors. An example of this optimization is below:

Listing 4.10 Example of division of matrix into blocks to take advantage of cache

```
1   for (i=0; i<M2->rows;i++){
2           v1[i]=M2->theMatrix[i][j];
3   }
4
5   for (i=0; i<M1->rows-1;i+=2){
6           register double s00,s10;
7           s00=s10=0.0;
8           for (k=0;k<M2->rows;k++){
9                   s00 += M1->theMatrix[i] [k] v1[k];
10                  s10 += M1->theMatrix[i+1][k] v1[k];
11          }
12          result->theMatrix[i] [j] =s00;
13          result->theMatrix[i+1][j] =s10;
14  }
```

In this case it caches one of the columns of the matrices.

These optimizations were implemented in all algorithms including parallel algorithms and the sequential one.

In next subsections the specific implementations of the various parallelization libraries namely OMP and TBB are explained. The Eigen3 library which includes OMP parallelization and CUBLAS which does GPU parallelization are also discussed.

4.3.1 Parallelization OMP

The OMP parallelization was by far the simplest to implement. For both cases (matrix-matrix and matrix-vector multiplication) the OMP directive *"#pragma omp parallel"* and *"#pragma omp for"* were used. The *"#pragma omp parallel"* directive initializes a parallel section which means all threads will execute the code in it. In this case it was used to create the vectors. The *"#pragma omp for"* directive distributes the number of threads available in a *for* loop. In the matrix-matrix multiplication algorithm was also used the *"#pragma omp parallel for"* directive which do the same of the other two directives combined. This means that it automatically parallelizes a for section and distributes the available threads on it [69]. By default the number of threads used in the Visual C++ implementation of OMP is the same as the number of processor cores detected by the Operating System (including virtual processors and hyperthreading CPUs) [93].

In the matrix-matrix multiplication algorithm the directive *"#pragma omp parallel"* was used for each thread allocate memory independent vectors. The *"#pragma omp for"* directive was used in the main matrix-matrix multiplication loop. This can been seen in the following extracted code:

Listing 4.11 Matrix-matrix use of *"#pragma omp parallel"* and *"#pragma omp for"* directives

```
#pragma omp parallel
{
        double v1=new double[M2->rows];
        double v2=new double[M2->rows];

#pragma omp for
        for (j=0; j < M2->columns-1; j+=2){ // Try to allocate 2 columns in cache each time
                int i,k;
                for (i=0; i<M2->rows;i++) {
                        v1[i]=M2->theMatrix[i][j]; // Fill the cache vector with the second matrix columns
                        v2[i]=M2->theMatrix[i][j+1];
                }

                for (i=0; i<M1->rows-1;i+=2){
                        register double s00,s01,s10,s11;
                        s00=s01=s10=s11=0.0;
                        for (k=0;k<M2->rows;k++){
                                s00 += M1->theMatrix[i] [k] v1[k]; // Multiplication operations
                                s01 += M1->theMatrix[i] [k] v2[k];
                                s10 += M1->theMatrix[i+1][k] v1[k];
                                s11 += M1->theMatrix[i+1][k] v2[k];
                        }
                        result->theMatrix[i] [j] =s00;
                        result->theMatrix[i] [j+1] =s01;
                        result->theMatrix[i+1][j] =s10;
                        result->theMatrix[i+1][j+1] =s11;
                }
        ...
        }
...
}
```

In the vector-matrix multiplication the same process with *"#pragma omp parallel"* and *"#pragma omp for"* directives was used in the main loop:

Listing 4.12 Vector-matrix use of *"#pragma omp parallel"* and *"#pragma omp for"* directives

```
#pragma omp parallel
{
        int i,j;
        register double s00,s01;

#pragma omp for
for (i=0; i < (M->rows-1); i+=2){
        s00=s01=0.0;
        for (j=0; j<M->columns; j++){
                s00 += V->theVector[j] M->theMatrix[i][j];
                s01 += V->theVector[j] M->theMatrix[i+1][j];
        }
        result->theVector[i] =s00;
        result->theVector[i+1] =s01;
}
```

Note that the original code was not modified, the only code added was the OMP pragma directives so OMP know what to parallelize and how.

4.3.2 Parallelization TBB

The TBB parallelization required much more modifications than the OMP case. The parallelization was applied exactly in the same loops of the OMP, however there was the need of creating a new struct for every new parallelized block of code.

For the matrix-matrix multiplication two structs were created. One for the multiplication of two columns simultaneously by two rows and another for the multiplication of only one column by two rows.

In the following code the implementation of the struct of only one column is shown:

Listing 4.13 TBB structure implementation for multiplication using one column

```
1   struct ITBmatrixMultiplication1C {
2           mMatrix M1, M2, result;
3           double v1;
4           int j;
5
6           void operator()( const blocked_range<int>& range ) const {
7
8                   for( int i=range.begin(); i<range.end(); i+=2 ){
9                           register double s00,s10;
10                          s00=s10=0.0;
11                          for (int k=0;k<M2->rows;k++){
12                                  s00 += M1->theMatrix[i] [k] v1[k];
13                                  s10 += M1->theMatrix[i+1][k] v1[k];
14                          }
15                          result->theMatrix[i] [j] =s00;
16                          result->theMatrix[i+1][j] =s10;
17                  }
18          }
19  };
```

The *operator()* is the function that gets called by the TBB library to be parallelized. The variables declared in the struct are variables that the function *operator()* needs in order to work properly. For example in this code we need the matrices we are going to multiply plus a result matrix to store the calculations. The majority of the variables are pointers because it's much more efficient to pass big objects via reference than via copy.

Besides these modifications it's also necessary to initialize the structs created. This is done with the following code:

Listing 4.14 Example of initialization of TBB structure

```
1   ITBmatrixMultiplication1C my1CMultiplication;
2
3   my1CMultiplication.result=result;
4   my1CMultiplication.j=j;
5   my1CMultiplication.M1=M1;
6   my1CMultiplication.M2=M2;
7   my1CMultiplication.v1=v1;
8
9   parallel_for( blocked_range<int>(0, M1->rows-1), my1CMultiplication );
```

The first line declares and initialize the struct. The variables necessary to the struct are then assigned. In the end the parallel code is executed using the function call *parallel_for*. This

function receives an object of the type *blocked_range* which receives the initializer for the loop condition, the for loop conditional expression and the struct that contains the code to be executed.

The call to the *parallel_for* must be made in the same place where the code of the struct is in the sequential program. As stated before the *range.begin()* and *range.end()* are calculated by the library for each thread using the *blocked_range* parameters especially defined for the loop.

Vector-matrix multiplication was parallelized using this method. It uses a struct for the main for loop which receives the matrix, vector and result vector. Then the structure is initialized and parallelized using a *parallel_for* call.

4.3.3 Parallelization Eigen3

The Eigen3 library include support for CPU parallelization. For this it uses the OMP library internally and automatically with the user only having the need to activate the OMP library in the project. Unfortunately this support is limited, Eigen3 only currently supports the parallelization of general matrix-matrix products [94].

Using Eigen3 the code for implementing the matrix-matrix multiplication is very simply. An example is shown below:

Listing 4.15 Matrix-matrix multiplication using Eigen3

```
1  MatrixXd a(2700,2500);
2  a.setRandom();
3
4  MatrixXd b(2500,2700);
5  b.setRandom();
6
7  MatrixXd result = ab;
```

In this example two matrices are created one with 2700 x 2500 dimensions and another with 2500 x 2700, then their are set values to random. To execute the multiplication of both matrices we only need to use the (overloaded) operator *. The result is then saved in the result matrix. All the matrices used in this example consist of the type *MatrixXd* which means that their element type is *double*. If OMP is enabled the operation will be parallelized.

4.3.4 Parallelization CUDA / CUBLAS

As stated in subsection 4.1.3 the CUDA parallel model was designed to transparently scale its parallelism to a different number of GPU cores. A GPU is built using an array of Streaming Multiprocessors (SMs). A multithread program is partitioned into blocks of threads that execute independently from each other. Consequently a GPU with more multiprocessors automatically execute the program in less time than one with less multiprocessors. This is illustrated in Figure 4.2.

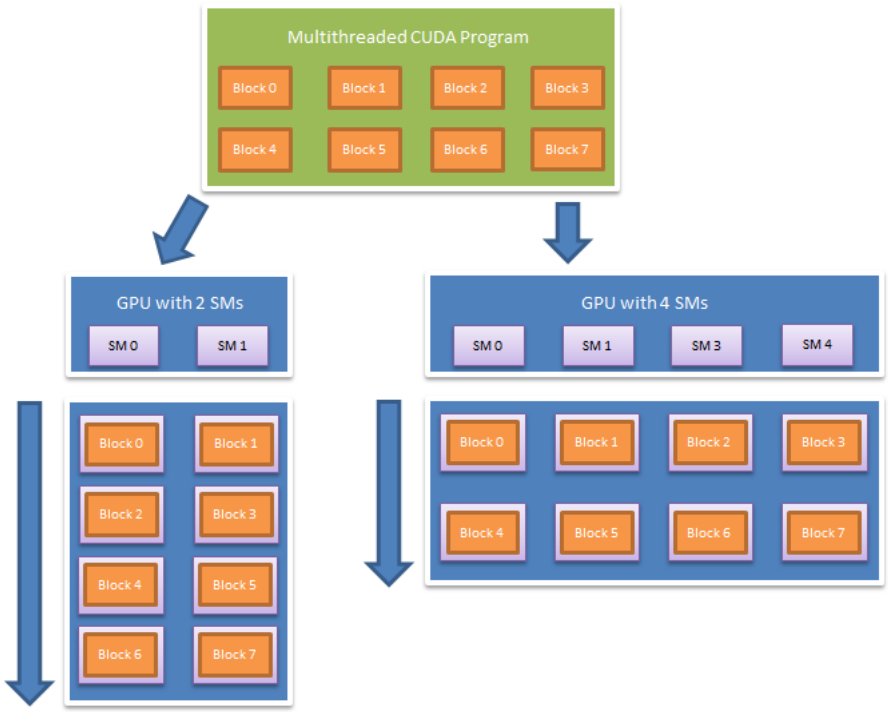

Figure 4.2: Example of automatically scalability for 2 GPUs with different numbers of SMs.

CUDA C extends C and allows the program to create kernels which are C functions and that differ of C regular functions by being executed in parallel by multiple CUDA threads. The kernels are defined using the *__global__* keyword and the number of threads, blocks and grids that executes the kernel are specified using the $<<< \cdots >>>$ syntax. As stated in the subsection 4.1.3 each thread has a unique id and this id is accessible within the kernel using the built in *threadIdx* variable [73].

An example of a simple CUDA C program which calculates the sum of the vector A and B and saves the result in the C vector is seen below.

Listing 4.16 Example of CUDA vector-vector sum

```
1  // Kernel definition
2  __global__ void VecAdd(float A, float B, float C)
3  {
4          int i = threadIdx.x;
5          C[i] = A[i] + B[i];
6  }
7  int main()
8  {
9          ...
10         // Kernel invocation with N threads
11         VecAdd<<<1, N>>>(A, B, C);
12         ...
13 }
```

In this code each of the N threads perform one element addition. The kernel here presented only has one thread block containing the N threads.

For the CUDA parallelization in this dissertation it was used CUBLAS routines for the linear algebra operations. In this dissertation CUDA implementations were only parallelized for a

single GPU, multiple GPU support will be left for future work.

4.3.5 Results

In this section the results obtained from the paralellization of the linear algebra operations matrix-matrix multiplication and vector-matrix multiplication are presented. Five parallel implementations namely a sequential implementation, an OpenMP implementation, an Intel Threading Building Blocks implementation, an Eigen3 implementation and a CUBLAS implementation. The CUBLAS implementation is presented in two different ways, one which takes in account the CPU-GPU memory transfers (CUBLAS WT) and one without (CUBLAS NT). The implementation is exactly the same, except that in CUBLAS NT the transfer times between the CPU-GPU are ignored in the benchmark. These different forms of measuring the CUBLAS implementation are to demonstrate the impact that CPU-GPU memory transfers have in the CUDA programs.

All these tests were made on three different computers whose specifications are given in Table 4.2.

Table 4.2: Computers Specifications.

	Computer 1	Computer 2	Computer 3
CPU Name	Intel Core i7-3610QM	Six-Core AMD Opteron 2435	Intel Core i7 CPU 920
CPU Speed (GHz)	2,30 (up to 3,3)	2,60	2,66 (up to 2,93)
CPU Number Cores	4	6	4
CPU Number Logical Processors	8	6	8
Number CPUs	1	2	1
Main Memory	6 GB	16 GB	8 GB
Video card	GeForce GT 650M	None	GeForce GTX 295
Video card Memory	2 GB	None	896 MB

The specifications for these tests were the follow:

Matrix-matrix multiplication:

- Matrix A with the size of 2700 x 2500, Matrix B with the size of 2500 x 2700.
- The results were obtained after calculating an average of 50 multiplications.

Matrix-vector multiplication:

- Matrix with the size of 7700 x 7000.
- Vector with the size of 7000.
- The results were obtained after calculating an average of 50 operations.

All implementations had the SSE2 (Streaming SIMD Extensions 2) instructions activated in the compiler. For all the CPU tests the number of threads used was the same as the number of the physical processors detected. This decision was made because it was detected in various tests that the Eigen3 performance was significantly altered if the number of logical processors were used instead of the number of physical processors. An example of this effect can be seen in Table 4.3 where it is possible to see the matrix-matrix multiplication using the logical processors and physical processors for the parallel implementations. Note that the number of logical processors in computer 2 is the same as the number of physical processors while in the computer 1 and computer 2 the number of logical processors is the double of the physical

processors (hyperthreading technology). The CUBLAS tests were not performed in the computer 2 because it does not have a video card.

Table 4.3: Matrix-matrix multiplication in the same computer with logical and physical processors.

Matrix-Matrix multiplication			
		Computer 1 Logical Processors	Computer 1 Physical Processors
Time	OpenMP	2,89287	3,87227
	TBB	3,58860	4,17747
	Eigen3	3,35090	1,50113

In Table 4.3 it is possible to see that the Eigen3 results using logical and physical processors are very distinct: the implementation with physical processors is more than two times faster (the same was also detected in other tests not included here). Using only physical processors there is a difference of more than 2 seconds to the other parallel implementations. This fact was taken in consideration for the final GNSS water vapor reconstruction algorithms.

In the next subsections the tests matrix-matrix multiplication and matrix-vector multiplication are presented.

4.3.5.1 Matrix-matrix multiplication

The results of the matrix-matrix multiplication are displayed below in Table 4.4 and in Figure 4.3. The Eigen3 library automatically parallelizes the multiplication of the matrices being only necessary to activate OpenMP option in the compiler.

Table 4.4: Matrix-matrix multiplication.

Matrix-Matrix multiplication				
		Computer 1	Computer 2	Computer 3
Time	Sequential	8,38920	44,16280	14,84200
	OpenMP	3,87227	9,63773	3,69492
	TBB	4,17747	14,07250	4,00668
	Eigen3	1,50113	2,22307	2,05756
	CUBLAS WT	1,30964	N/A	0,84300
	CUBLAS NT	0,00002	N/A	0,00002

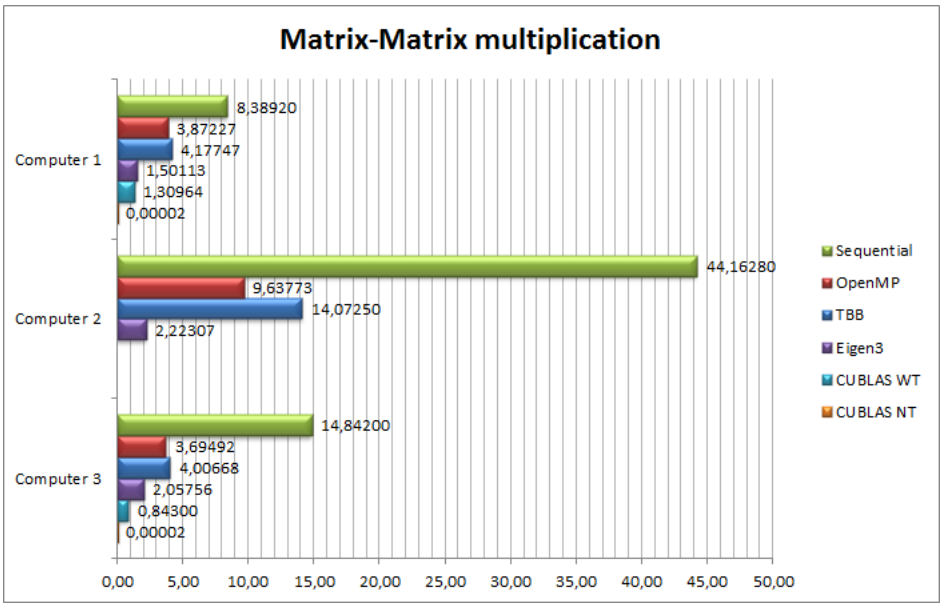

Figure 4.3: Matrix-matrix multiplication.

It is visible a large difference between the sequential implementation and the parallel versions. This difference allow us to conclude that the parallelization of matrices is essential for a fast execution of the algebraic reconstruction algorithms, more specifically the SIRT algorithms. The OMP version is faster than the TBB implementation. The faster CPU implementation is the Eigen3, reaching up to twelve seconds of difference to the TBB implementation in computer 2. The overall faster version is the CUBLAS version with computer 3 being faster than computer 1. Note that the CUBLAS method without counting the CPU-GPU transfers (CUBLAS NT) is about 65 000 times faster than the CUBLAS method with transfers (CUBLAS WT) in the computer 1. This demonstrates that the main bottleneck in the CUBLAS implementation is the CPU-GPU transfer time as it is not possible to create these CUDA applications without these transfers.

4.3.5.2 Matrix-vector multiplication

The results of the matrix-vector multiplication are displayed below in Table 4.5 and in Figure 4.4. Note that the Eigen3 library don't make any parallelization of this multiplication although being a sequential implementation it contains some optimizations.

Table 4.5: Matrix-vector multiplication.

Matrix-Vector multiplication				
		Computer 1	Computer 2	Computer 3
Time	Sequential	0,03406	0,21326	0,05710
	OpenMP	0,02633	0,06573	0,03190
	TBB	0,02580	0,06360	0,03160
	Eigen3	0,03313	0,11387	0,05884
	CUBLAS WT	0,02578	N/A	0,02062
	CUBLAS NT	0,00002	N/A	0,00002

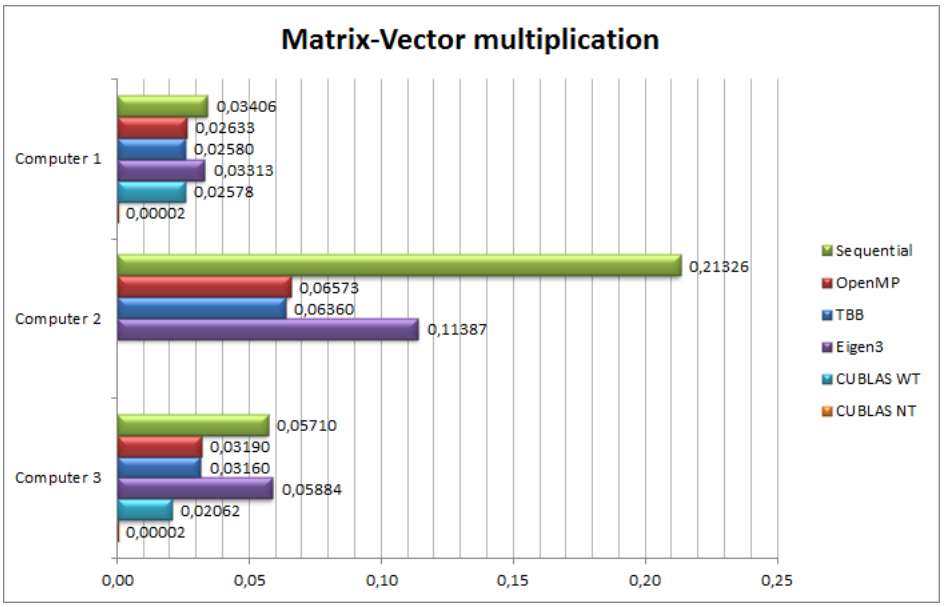

Figure 4.4: Matrix-vector multiplication.

Analysing the results is possible to verify that the parallel versions are more efficient than the sequential and Eigen3 versions. The difference between the parallel versions (OMP and TBB) are minimal. The Eigen3 implementation is slower in this test (than the parallel implementations) since is not parallelized, which indicates that perhaps it would be appropriate to parallelize this library matrix-vector multiplication. Also it is important to note that even though the difference between the sequential version and the parallel versions may be significant (case of computer 2) this multiplications does not have so much importance on the overall performance as the matrix-matrix multiplication. For example in the matrix-vector multiplication there are not any implementation which reaches one second (even with a matrix and vector much bigger than in matrix-matrix multiplication test), on the other hand in the matrix-matrix multiplication even with smaller matrices sizes the execution time reaches up to 44 seconds in the sequential version of computer 2 and the faster parallel version which is the CUBLAS implementation of computer 3 is still about four times slower than the slowest implementation of matrix-vector multiplication version. In the CUBLAS implementation it is possible to see that the CPU-GPU transfer are still the main bottleneck of the operation like it already was in the matrix-matrix multiplication, this is checked by the difference between CUBLAS WT and CUBLAS NT methods of measuring.

4.4 Algebraic Reconstruction Algorithms Parallelization

In this section the results obtained from the parallelization of the algebraic algorithms with the linear algebra operations (matrix-matrix and matrix-vector multiplications) are presented. Note that what is being tested is only the performance of each implementation, the ART/SIRT algorithms convergence are not tested. There will be tested five implementations namely a sequential implementation, an OpenMP implementation, an Intel Threading Building Blocks implementation, an Eigen3 implementation and a CUBLAS implementation. For the ART methods only four implementations: sequential, Eigen3 Column-major, Eigen3 Row-major and CULAS will

be tested. When not specified in the Eigen3 tests, is is assumed that the column-major order was used. All these tests were made in the same computers were the linear algebra operations were also tested. The specification can be checked in Table 4.2.

The specifications for the SIRT Landweber, SIRT SART and ART Kaczmarz tests were as follows:

- Matrix with the size of 2700 x 2500.
- Vector with the size of 2700.
- The number of iterations for each algorithm was 50.
- The results were obtained after calculating the average of 50 algorithm calls.

Note that all SIRT implementations (including Eigen3 and CUBLAS) calculate the spectral radius using the power method since it is very efficient in terms of performance. Also like the linear algebra operations testes every implementation had the SSE2 instructions activated in the compiler and for all the CPU implementations the number of threads used was the same as the number of the physical processors detected.

4.4.1 Validation

The algorithms implemented were validated using the Shepp-Logan phantom [49] image which is a standard image for image reconstruction tests. Two tests were performed one presenting only the resultant reconstructions images and another validating mathematically the reconstructions.

The first test used a original image with a resolution of 80 x 80. The projection data was generated using 75 parallel rays over 36 different angles.

All the implemented algebraic reconstruction algorithms namely Kaczmarz, Symmetric kaczmarz, Randomized kaczmarz, Landweber, Cimmino, CAV, DROP and SART were validated using this test. However here will only be presented the algorithms that were tested in this chapter: Landweber, SART and Kaczmarz. In Figure 4.5 the original image can be seen and in the figures 4.6, 4.7, 4.8 the Landweber, SART and Kaczmarz reconstructions using 50 iterations of the CPU parallel implementations described earlier in this chapter are presented. As can be seen all algorithms reconstructed the original image reasonably well.

Figure 4.5: Shepp-Logan phantom original Image

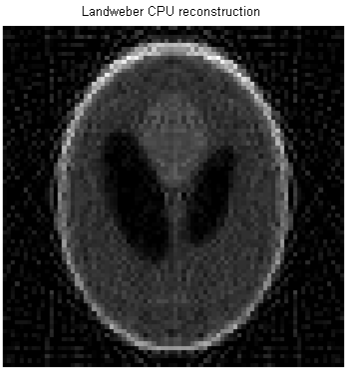

Figure 4.6: Shepp-Logan phantom Landweber reconstruction.

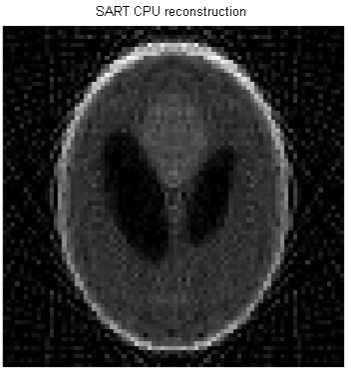

Figure 4.7: Shepp-Logan phantom ART reconstruction.

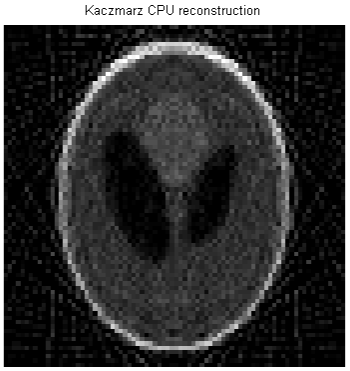

Figure 4.8: Shepp-Logan phantom Kaczmarz reconstruction.

In the second test, Shepp-Logan phantom was again used as image to reconstruct. It was used two different resolutions one being 80 x 80 and another being 50 x 50. The projection data was the same of previous test: 75 parallel rays over 36 different angles. Using Kaczmarz, Landweber and SART algorithms the two different resolutions images were reconstructed using 50, 1000 and 10 000 iterations. The standard deviation (std) of the residuals[1] was calculated for each reconstruction. The max (z max) and min (z min) of the residuals were also calculated. The results are presented in Tables 4.6, 4.7, 4.8, 4.9, 4.10 and 4.11.

Table 4.6: Shepp-Logan 80 x 80 - 50 iterations

	std	z min	z max
Kaczmarz	0.116510	-0.4746	0.639204
Landweber	0.122607	-0.47022	0.692239
SART	0.119124	-0.46668	0.651125

Table 4.7: Shepp-Logan 50 x 50 - 50 iterations

	std	z min	z max
Kaczmarz	0.074213	-0.298224	0.341835
Landweber	0.098106	-0.395006	0.519073
SART	0.095274	-0.432999	0.508764

From the results is possible to draw some conclusions. Firstly all algorithms continued to converge to original solution with the increasing of the number of iterations. It is important to mention that while there are no noise in the reconstruction, the reconstruction is not perfect due to the round-off errors. In all reconstructions the Kaczmarz algorithm obtained a standard deviation smaller than Landweber or SART. However all three algorithms obtained similar re-

[1]The residuals are the difference between the original image and the reconstructed image.

Table 4.8: Shepp-Logan 80 x 80 - 1000 iterations

	std	z min	z max
Kaczmarz	0.116133	-0.477247	0.636647
Landweber	0.116482	-0.475663	0.640159
SART	0.114891	-0.495548	0.591570

Table 4.9: Shepp-Logan 50 x 50 - 1000 iterations

	std	z min	z max
Kaczmarz	0.057617	-0.214230	0.256787
Landweber	0.073086	-0.296873	0.335897
SART	0.071769	-0.279004	0.335304

Table 4.10: Shepp-Logan 80 x 80 - 10 000 iterations

	std	z min	z max
Kaczmarz	0.116052	-0.947615	0.634917
Landweber	0.116259	-0.476399	0.636267
SART	0.114778	-0.498273	0.587139

Table 4.11: Shepp-Logan 50 x 50 - 10 000 iterations

	std	z min	z max
Kaczmarz	0.047456	-0.201650	0.2053409
Landweber	0.060048	-0.222268	0.2663149
SART	0.058609	-0.220531	0.2605460

sults without be possible to favor one unequivocally. Additionally is possible to see that the 80 x 80 reconstructions have more difficulty to converge than the 50 x 50 reconstructions. This is explained because there are less projections than cells.

4.4.2 Results

In this subsection several tests including SIRT Landweber, SIRT SART and ART Kaczmarz are presented. The main objectives of this section is to compare the parallel techniques (see which one performs better) and also to compare the speedup of the parallel versions over the sequential versions.

4.4.2.1 SIRT Landweber

The results of the Landweber method are displayed below in Table 4.12 and in Figure 4.9.

Table 4.12: Landweber method.

SIRT Landweber				
		Computer 1	Computer 2	Computer 3
Time	Sequential	8,40140	45,52840	13,89250
	OpenMP	3,93527	10,74650	4,14008
	TBB	4,38747	16,28730	4,42188
	Eigen3	2,14413	6,41347	3,24294
	CUBLAS	1,82480	N/A	0,95724

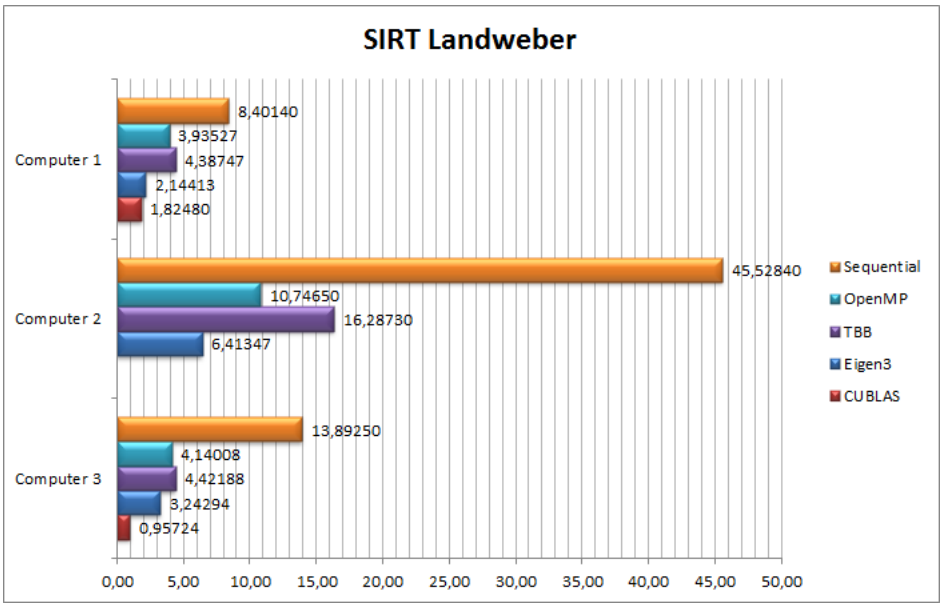

Figure 4.9: Landweber method.

At the first glance it is possible to notice a big difference between the sequential version and the others. This can be explained because the matrix-matrix multiplication contained in this algorithm are extremely slow compared with the other operations, the matrix-vector multiplications are also included but have obviously less impact. The fastest implementation is the GPU CUBLAS implementation which does not reach one second in the computer 3. It is interesting to find out that relatively to the CPU implementations the computer 1 is faster than the computer 3 while looking at the GPU implementations the inverse is verified. This reveals that while computer 1 has a faster CPU than computer 3, computer 3 has a faster GPU than computer 1. Also it can be seen that the difference between the parallel CPU versions and the Eigen3 version which occurred in the matrix multiplications tests (especially in computer 2) are not so obvious here. This fact is due to SIRT methods that contain diverse multiplications of matrix-vector. In this specific case the algorithm is executed with 50 iterations where each iteration contains two matrix-vector multiplications summing a total of 100 multiplications. The matrix-matrix multiplication case only occurs one time for the calculation of the spectral radius.

4.4.2.2 SIRT SART

The results of the SART method are displayed below in Table 4.13 and in Figure 4.10.

Table 4.13: SART method.

SIRT SART				
		Computer 1	Computer 2	Computer 3
Time	Sequential	16,14280	70,26480	26,50780
	OpenMP	8,05060	26,48040	7,92380
	TBB	8,42993	33,45620	8,52993
	Eigen3	3,75800	8,95913	5,70680
	CUBLAS	3,09307	N/A	1,55940

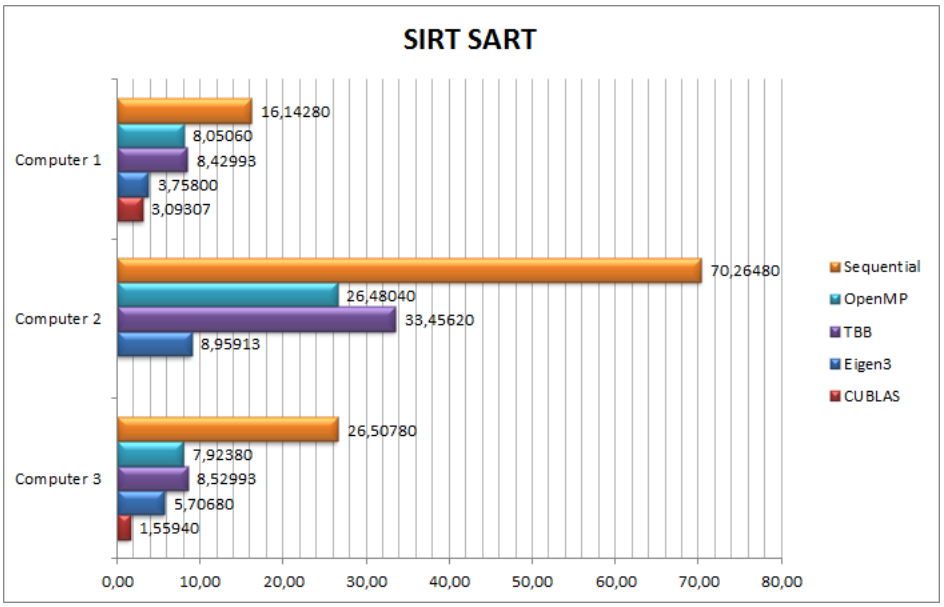

Figure 4.10: SART method.

The SART algorithm presents similar results when compared to the Landweber algorithm. SART takes however some more time to execute than the Landweber. The difference between the sequential version and the others are still high (approximately the double in computer 1/computer 2 and triple in computer 3 when comparing to the slower parallel implementation). This difference can be explained because there are one additional multiplication of matrices in this algorithm. The CUBLAS version is still the fastest version and in computer 3 is approximately 17 times faster than the sequential implementation. Like Landweber, it also contains two multiplications of matrix-vector per iteration.

4.4.2.3 ART Kaczmarz

Regarding the ART algorithms only the sequential, Eigen3 and CUBLAS implementations were tested. This is because these algorithms don't include matrix-matrix and matrix-vector multiplications which were what was parallelized in OMP and TBB versions. There were tested two different Eigen3 methods: column-major and row-major which were discussed in Section 4.2.2 of this dissertation. This two different versions were tested because each one have very different performance results in this ART methods. The results of the Kaczmarz method are displayed below in Table 4.14 and in Figure 4.11.

Table 4.14: Kaczmarz method.

ART Kaczmarz				
		Computer 1	Computer 2	Computer 3
Time	Sequential	1,35727	3,22750	2,18028
	Eigen3 - ColMajor	3,84413	8,45700	4,41112
	Eigen3 - RowMajor	0,91287	2,69847	1,50042
	CUBLAS	14,11020	N/A	15,18900

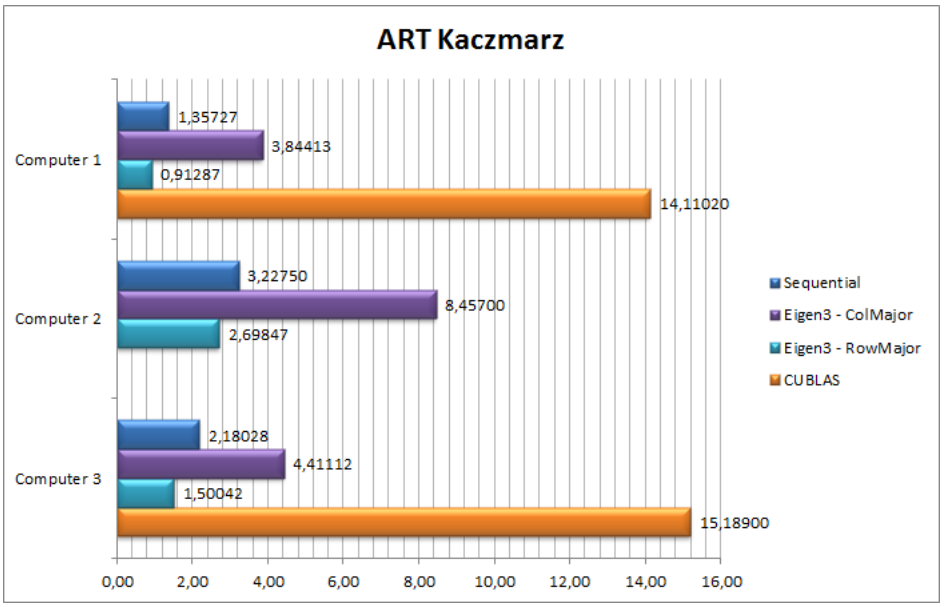

Figure 4.11: Kaczmarz method.

The results presented confirm the big difference between each of the Eigen3 implementations. All the ART methods make use of the diverse rows of the input matrix in the various iterations. How can be checked the row-major is the more efficient because the matrix is organized sequentially in the memory by rows, being the access to the rows sequential. The sequential implementation performance is similar to the row-major this because in its implementation the matrix is also organized in the memory by rows. The version column-major is much more slower than the row-major. Since the matrix in this implementation is organized in memory by columns the access to each row is not sequential. Besides the Eigen3 implementations differences we can observe that there is a large difference between the CPU implementation and GPU implementation (in computer 1 the GPU version is about 15 times slower and in the computer 3 is about 10 times slower).

The symmetric version of Kaczmarz method was also tested, however it took approximately the double of the time in all implementations because of the “double sweep” which this symmetric Kaczmarz implements.

4.5 Summary

In this chapter it was presented diverse multi-threading libraries and linear algebra libraries including the basic algebra / math library developed. The parallelization approach in these linear algebra libraries and the respective results were also presented and discussed. The results of the parallization and the validation of the algebraic reconstruction algorithms were also shown. The reconstruction algorithms were validated using two different tests, one presenting only the resultant reconstructions images and another validating mathematically the reconstructions. The results shown that the three methods produce similar results without be possible to favor one unequivocally.

Some conclusion may be taken from the algorithms tests. Looking first at the CPU implementations the Eigen3 library proved to has the best overall performance. It was always faster than

OMP and TBB solutions. Over the sequential versions and looking only at the computer 1 Eigen3 had a speedup of about 4.5 in SIRT Landweber, 5 in SIRT SART and 1.5 in ART Kaczmarz.

The CUBLAS solution was even faster than Eigen3 with the larger difference occurring in computer 3. In SIRT methods CUBLAS proved to be the faster solution with the exception of ART methods where it proved to be much slower. If we look at the Kaczmarz equation (5) we can easily see that the only operations are two dot products, a scalar division, a scalar multiplication, a scalar-vector multiplication and one vector-vector sum. None of these operations seem to be highly computational intensive, so parallelizing them should not bring much improvement. Also taking into account that all GPU kernal calls contain a fixed time cost (latency) [95] and that there are 5 kernel launches per ith iteration and in this case there are 2700 ith iterations this will lead to about 13500 kernel launches per kth iteration which is a very large number of transactions. With this in mind we rapidly can conclude that Kaczmarz, and more generally the ART approach type of algorithms, are not the best type of algorithms to execute in the GPU. The same conclusion was also reached in [53]. However it was still interesting to demonstrate this in practice and compare CPU with GPU results.

Based on these tests, it was concluded that the Eigen3 is the better solution for CPU implementations and that CUBLAS is the ideal solution in GPU implementations with the exception of the ART algorithms where Eigen3 should also be used.

In the next chapter an overview to the GNSS and Water Vapor will be presented. It will be also demonstrated how the GNSS Water Vapor Estimation works and how the reconstruction of water vapor image is done.

Chapter 5

GNSS and Water Vapor

5.1 GNSS Overview

Global positioning system (GPS) is a technology that has revolutionized the surveying and mapping industry. This technology involves satellites, ground reference station infrastructure and user equipment. It can estimate the position and velocity continually and accurately around the world [96]. Besides GPS there are other similar systems: Russia operates Russia Global Navigation Satellite System (GLONASS), Europe is developing Galileo and China is developing BuiDou Navigation Satellite System (BDS). To refer to all these systems the term Global Navigation Satellite Systems (GNSS) is currently employed.

GPS is currently the most used and well known space-based satellite navigation system consequently and it will be primary focus in this dissertation. GPS is composed of 24 satellites arranged in 6 orbital planes where each plane contains 4 satellites. Besides the position / velocity information already mentioned, GPS also provides a form of Coordinated Universal Time (UTC). The services of the GPS can be used by an unlimited number of users because receivers communicate passively (they only receive the data) [97].

The GPS satellites broadcast ranging codes and navigation data on two frequencies: L1 (1575.42 MHz) and L2 (1227.6 MHZ). All the satellites transmit these frequencies however each one transmits a different ranging code [97]. The ranging code allows each receiver to determine the propagation time of the signal and consequently determine the satellite-to-user range while the navigation data allows each receiver to determine the location of each satellite at the transmission signal [97]. This technique is known as one-way time of arrival (TOA).

Together with satellite's location and the delay of the signal (between the satellite and the receiver) it is possible to calculate the receivers three-dimensional location. However to determine the receiver location requires at least four satellites for the TOA ranging measurements [97].

5.2 Water Vapor Overview

Water vapour has a crucial role in atmospheric processes that act over a wide range of temporal and spatial scales from global climate to micrometeorology [8]. Some of the atmospheric processes that are significantly influenced by the water vapor distribution are, radiation transfer, energy balance, cloud formation/composition and the convective initiation and development of precipitation systems [98]. It is the constituent that most contributes to the greenhouse effect [8]. Water vapor is also the most variable constituent of the atmosphere and is an essential component of climate and numerical weather prediction because of its relation with the atmosphere processes [99].

Due to water vapor's importance scientists have developed a variety of methods to measure the distribution of water vapor in the atmosphere. Some of these methods are radiosondes, microwave radiometers, spectrometers, lidars and GPS which is the method described in this dissertation.

5.3 GNSS Water Vapor Estimation

GNSS water vapor estimation has large advantages over other existing methods. The advantages include good temporal resolution (down to 15 minutes), the fact that it can operate in all weather conditions and can be run unattended [7]. Also the number of GNSS stations continues to increase in many regions of the world.

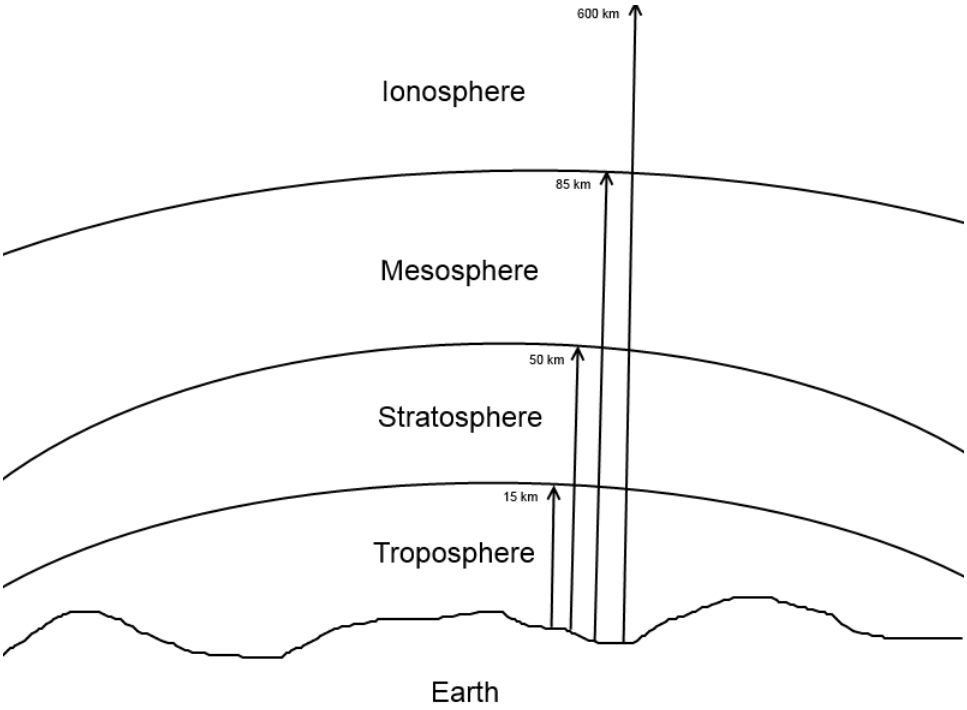

Figure 5.1: Vertical layers of the Earth's atmosphere

When the GPS signal passes through the atmosphere it suffers a number of propagation effects. The magnitude of those effects depends on the satellite-receiver elevation angle and the atmospheric environment. The atmosphere causes the following effects [99]:

1. Ionospheric delay and scintillation
2. Delay caused by the wet and dry troposphere
3. Atmospheric attenuation in the troposphere and stratosphere

The various atmosphere, the vertical layers, can be seen in Figure 5.1. The prime source of error in GPS is the tropospheric group delay which is caused by the tropospheric refractivity [99]. The Zenith Total Delay (ZTD) is the atmospheric delay of the GPS signal which arrives from the zenith direction. In Figure 5.2 an illustration of the GPS signal, zenith direction and elevation angle (e) can be seen. ZTD can be divided in two major delay effects in the troposphere. The first is Zenith Hydrostatic Delay (ZHD) which has the larger effect and it is caused by dry air containing

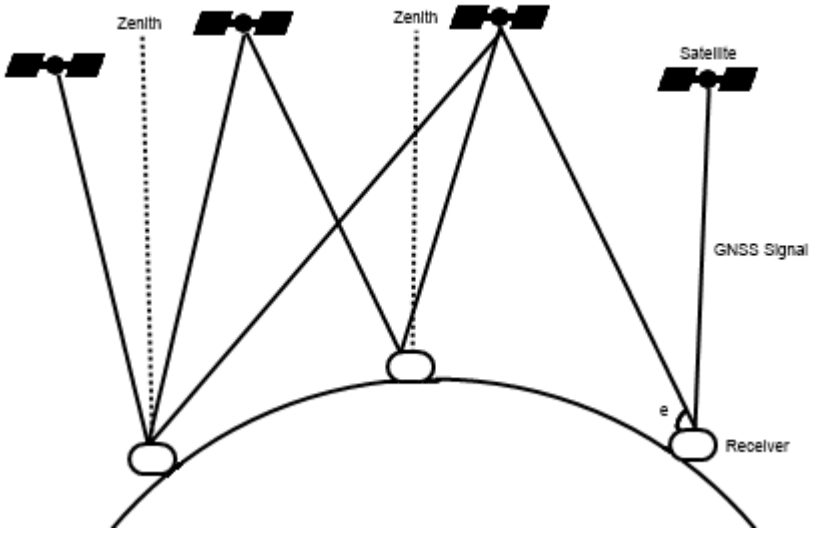

Figure 5.2: GPS signal between the satellites and receiver.

mainly N_2 (nitrogen gas) and O_2 (oxygen). This delay corresponds to about 2.1 meters at sea level and it varies mainly with the temperature and atmospheric pressure. This makes it predictable [99]. The second effect is called Zenith Wet Delay (ZWD) and is normally smaller than ZHD. The normal values of ZWD oscillate between 1 and 80 centimetres. The ZWD usually have higher variation and is more difficult to predict than ZHD [99].

The ZHD can be calculated easily and with precision, using measurements of surface atmospheric pressure using the following formula [100] [101]:

$$ZHD = 2.2768 \pm 0.0015 \frac{P_0}{f(\phi, H)} \quad (11)$$

$$f(\phi, H) = 1 - 0.00265cos(2\phi) - 0.000285H \quad (12)$$

where P_0 is the surface pressure in millibars, ϕ is the latitude and H is geodetic height in kilometres. ZHD is given in millimetres.

Calculating ZHD and knowing ZTD we can calculate the ZHD using the formula:

$$ZTD = ZHD + ZWD \quad (13)$$

After the ZWD has been calculated it can be multiplied by the variable Π [101] to obtain the quantity of water vapour integrated in the zenith direction (IWV):

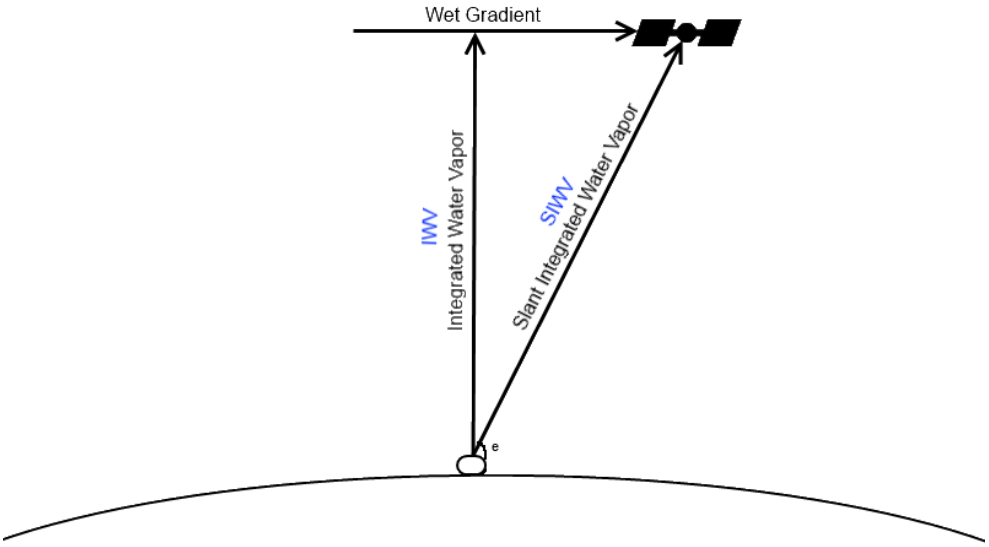

Figure 5.3: IWV, SIWV and Wet Gradient representation

$$IWV = \Pi \cdot ZWD \quad (14)$$

where Π is given by:

$$\Pi = \frac{10^6 m_w}{(k_2 - k1\frac{m_w}{m_d} + \frac{k3}{T_m})R^*} \quad (15)$$

where T_m is the weighted mean temperature of the atmosphere, m_w is the molar mass of water vapour, m_d is the molar mass of dry air, R^* the ideal gas constant, k_1, k_2 and k_3 are physical constants that are based in part on theory and in part on experimental observations [102]. According to [30] $k_1 = 77.60 \pm 0.05 K/hPa, k_2 = 70.4 \pm 2.2 K/hPa, k_3 = (3.739 \pm 0.012) \cdot 10^5 K^2/hPa$.

Bevis *et. al* [8] have derived the following formula:

$$Tm = 0.72t + 70.2(^\circ K) \quad (16)$$

A slant water vapour delay (SWVD) can be divided in two components, the first component is a isotropic part, which is essentially the IWV. The second components of $SWVD$ is a nonisotropic part. This last component describes the deviation of the water vapour field from the isotropic field.

The Slant Water Vapour Delay (SWVD) for each individual satellite can be calculated with base in the following formula [103]:

$$SWVD = M(e)IWV + S \quad (17)$$

where $M(e)$ is the mapping function which maps the zenith IWV to the elevation ray e. This mapping function can be the Niell mapping function or any other. The Niell mapping function is widely used because it is highly accurate and does not need any meteorological parameters [31]. S represents the nonisotropic component of water vapour. From the equation is possible to observe that $SWVD$ can be calculated once we calculate S.

GPS signals contain two frequencies L1 and L2, which can be affected by ionosphere clock error and troposphere. The phase delay can be linearly combined to remove the effects of ionosphere. The clock error can be nullified using double differences. The total delay of troposphere (STD) [104] can be presented as:

$$STD = M_h(e)ZHD + M_w(e)ZWD + M_\Delta(e)[(G_N \cos\theta + G_E \sin\theta)] + R_e \quad (18)$$

G_N and G_E are the north and east components of atmospheric delay gradients. R_e corresponds to the residual. e and θ are the elevation and azimuth angle respectively. M_h is the hydrostatic mapping function, M_w is the wet mapping function and M_Δ is the gradients mapping function. There are several M_Δ which can be used [105]. Chen and Herring have proposed in [106] the use of the following function:

$$M_\Delta(e) = \frac{1}{\sin e \tan e + c} \quad (19)$$

In [106] the value of 0.0032 was used for c.

The residual represents the difference between the observation and the model. The residual is used to cover the deficiency for modelling the inhomogeneous atmosphere by using only the gradients. If residual is not used the delay will depend mainly of the mapping function and would be larger at lower elevations [105].

To obtain the $SWVD$ it is necessary to remove the hydrostatic gradients from the total gradients. Two methods exist for removing the hydrostatic gradients. The first one is to estimate the gradients parameters using a numerical model. The second method consists in the assumption that the hydrostatic gradient is stable and that it is constant over a period ($e.g.$ 12 hours) so it can be removed or reduced to a minimum by averaging the gradient solution on that period [107]. With the hydrostatic gradient calculated the wet gradient can be determined. It's still necessary to remove the hydrostatic gradient or it will affect $SWVD$ calculation.

Removing the hydrostatic parameters and adding residue the slant water vapour can be calculated using [103]:

$$SWVD = M(e)IWV + S = M(e)IWV + \Pi[M_{\Delta}(e)(G_N \cos\theta + G_E \sin\theta) + R_e] \quad (20)$$

where G_N is the north component of wet gradient and G_E is the east component.

The representation of the IWV, $SIWV$ and wet gradients can be seen in Figure 5.3. As seen the main difference between the IWV and SIWV is the direction where the integrated water is calculated: IWV is in zenith direction while SIWV is in the satellite direction.

5.4 GNSS Water Vapor Image Reconstruction

The tropospheric tomography consists of the retrieval of a 3D scalar field of the water vapour in the troposphere [21].

The first tropospheric tomography studies of water vapour were performed by Flores *et al.* [32] and Flores [108] on the Kilauea volcano and by Gradinarsky [109] in Göteborg region.

The GNSS water vapor reconstruction defines an inversion that is ill-posed, the solutions generally are not unique and stable [3].

A spatial grid is superimposed over the atmosphere and in the area that we want to know the water vapor distribution. The vertical axis of the grid usually does not exceed above 15 km which is the approximate size of the troposphere.

The slant wet delays are used to calculate the water vapor distribution over all the grid. Usually the ray bending which occur in the atmosphere is neglected and it is assumed that slant paths are completely straight-lines. To calculate the slant wet delays we apply the formulas of Section 5.3.

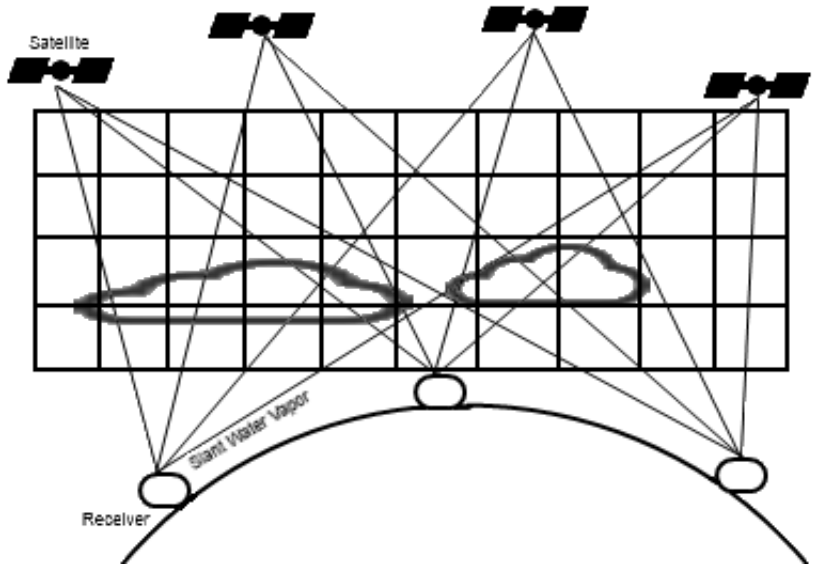

Figure 5.4: GNSS water vapor tomography.

The cell grids structure can within reasonable limits be freely chosen [3]. In Figure 5.4 an illustration of the mentioned GNSS water vapor tomography principles is presented.

The GNSS water vapor tomography problem is described mathematically with the equation (2).

Where b are all the observations, *i.e.* the slant wet delays, A is the contribution of each voxel for each slant wet delay *i.e.* the length crossed by the slant in the voxel and x is the current atmosphere water vapor distribution.

The equation (2) can be inverted using various techniques as SVD. In this dissertation the algebraic reconstruction methods will be used to perform the tomography inversion because the advantages discussed in Chapter 2.

A good spatial coverage of the atmosphere by the slant wet delays is required or there will be many voxels without any slant wet delay resulting in a highly indeterminable reconstruction. To minimize this problem slant wet delay data is usually collected over a time interval *e.g.* 30 minutes [3]. Additionally other GNSS systems can be used, *e.g.* combining GPS with GLONASS the number of the slant wet delay doubles [110].

Another technique to minimize the spatial coverage problem is to make use of extra information and constraints. For example data obtained form using synoptic observations, radiosonde profiles, radio occultation profiles or other independent observations [110].

5.5 Summary

In this chapter an overview to the GNSS and Water Vapor was done. It was also presented how the GNSS Water Vapor estimation is calculated and how it is performed the GNSS Water Vapor Image reconstruction. The following chapter will present the SEGAL GNSS Water Vapor Reconstruction Image Software which has been developed to perform GNSS Water Vapor Reconstruction. The various components will be described and some preliminary test presented.

Chapter 6

SEGAL GNSS Water Vapor Reconstruction Image Software

A GNSS water vapor image reconstruction software has been developed, this software was nominated **S**EGAL GNSS **WA**ter Vapor **R**econs**T**ruction Image Software (SWART). This software can compute the IWV distribution over a user's specified area using GPS observations. It can also perform the water vapor reconstruction using algebraic reconstruction algorithms and plot the result in latitude, longitude and height slices.

6.1 SWART Components

SWART is composed by five executables that will be called as components from now. Each of these components contribute in some way to the GNSS tomography. Specifically the SWART components are WaterVaporReconstruction, SlantDelayJoinProcessing, GridRayIntersection, AlgebraicAlgorithms and PlotWaterVapor. This division in components was taken because it allow the users to only execute some steps of the GNSS water vapor image reconstruction easily and giving the user more flexibility. For example it allows the user to only plot the water vapor image of previous made tomography inversions. All these components are console executables with the only exception being the PlotWaterVapor executable which was generated with MATLAB (MATrix LABoratory) compiler [111]. The console applications were developed using C++. For the WaterVaporReconstruction, GridRayIntersection and AlgebraicAlgorithms components boost C++ libraries [112] were used for the command line parameters parsing. Please note while this software has been developed in Windows platform it has been developed thinking that it can ported to other operating systems like Linux or MacOS. Because this portable code was written and portable libraries were used. A UML component diagram of SWART can be seen in Figure 6.1.

6.1.1 WaterVaporReconstruction Component

This component has the purpose to link and call the other components of SWART in order to provide all the functionalities needed for the GNSS water vapor.

The command line options available for this component are the following:

WaterVaporReconstruction Command Line Options

```
-h [ --help ]                        produce help message
-c [ --config ] arg                  SWART configuration file to use.
-s [ --SlantDelayJoinProcessing ]    Reads and synchs rinex files, meteo files
                                     and sp3 files to produce a file with slant
                                     delays.
-g [ --GridRayIntersection ]         Calculates the distance crossed by each
                                     cell in the grid by each slant delay
-a [ --AlgebraicAlgorithms ]         Uses one algebraic algorithm to
                                     reconstruct the image with base in the
                                     grid ray intersections and slant delays
-p [ --PlotWaterVapour ]             Plots the reconstructed image, contour is
                                     used.
```

The configuration file specified with -c or –config option is used for the user customize the options for all other components, namely: SlantDelayJoinProcessing, GridRayIntersection, AlgebraicAlgorithms and PlotWaterVapor. The options are specified using the option name and value, separated by a 'tab' character. This file allow comments. The components start with '#' character. All the other configurations files of other components of SWART have this syntax and work in the same way. A sample SWART configuration file an be visualized in A.0.1.

6.1.2 SlantDelayJoinProcessing Component

The SlantDelayJoinProcessing component synchronizes all the files needed for the GNSS Water Vapor Tomography and produces as result a file with slant wet delays between the GNSS receivers and GNSS satellites. The slant wet delays are produced with the Neill Wet Mapping function. Besides the slant wet delays it also outputs the following information associated with them: time of the slant delay, satellite number, satellite Cartesian coordinates, receiver name, receiver Cartesian coordinates, elevation angle in degrees/radians and azimuth angle in degrees/radians. An example of this file can be found in Appendix A.0.2.

This component only accepts one parameter which is a configuration file. This configuration file contains everything that this component needs to work. An example of this component configuration file can be found in Appendix A.0.3.

The available options are described in Table 6.1, the options grayed out can be specified multiple times (one per receiver).

Table 6.1: SlantDelayJoinProcessing Configuration File Options

Option	Description
SP3FILE	SP3 orbits filename
STARTSECONDS	Seconds to begin the files synchronization
ENDSECONDS	Seconds to end the files synchronization
SP3INTERPOLATION	True or False. Specifies if the user wants to perform satellites position interpolation (produces about 3 times more slant wet delays)
RECEIVER	Name of the receiver
RINEXFILE	RINEX observation file of the current receiver
VARSFILE	File with meteorological data (with pressure and temperature)
ZTDFILE	File containing ZTD value and gradients

The SP3FILE refers to a National Geodetic Survey Standard GPS Format SP3 file [113]. SP3 files contain the orbits of the GPS satellites for one determined day specified by its filename. The filename is of the type GPS Week + Day of Week. For example a SP3 file with the filename 'igs16383.sp3' corresponds to GPS orbits of the 1638 week and the 3th day of week which converting to day/month/year format corresponds to 01/06/2011.

The SP3 file format contains also other information besides the time and position of each satellite like satellite clock correction [113]. The complete description of the SP3 format can be found in [113]. An example file can also be found in Appendix A.0.4.

The SlantDelayJoinProcessing component only retrieves the satellites Cartesian coordinates and the its time from this files.

The STARTSECONDS and ENDSECONDS option allow the user to specify the interval to retrieve the tomography image, its only necessary to specify the seconds for a day, this means that the seconds consist of the instance hours + minutes + seconds converted to seconds. For example with STARTSECONDS=41400 and ENDSECONDS=43200 the interval that will generate the reconstructed image will consist in the day specified by the SP3 file starting at 11 hours and 30 minutes and ending at 12 hours of that day.

The SP3INTERPOLATION allow the SlantDelayJoinProcessing component to interpolate the satellite positions within the SP3 file. The SP3 files only give the position for each satellite for one day within 15 minutes of interval. The other files used by this component namely rinex files, meteorological files and ZTD files all contain data available in intervals of 5 minutes. This means if the SP3 interpolation is made for every 5 minutes the number of the slant wet delays generated can be increased in about 3 times (because there are 3 times more information for its calculation).

The interpolation strategy used in this component is the same that in [114]:

$$C = A_0 + A_1T + A_2T^2 + A_3T^3 + \cdots + A_NT^N \quad (21)$$

where C represents the X, Y, Z, interpolated coordinates, T is the time and A_0 to A_N are the coefficients of the polynomial which are adjusted to fit the satellite orbits.

In [114] the author explain that using the typical 32-bit architecture of a PC the formula (21) rapidly begins to fail due to a dynamic range problem. However the Neville's algorithm avoids

this limitation providing a simple recursive algorithm for computing the function's value [114]. Because of this Neville's algorithm [115] was selected to perform the polynomial interpolation.

To improve the numerical precision all the satellite position datasets, represented as c, were normalized using the same manner as [116]:

$$c'_i = \frac{(c_i - \bar{c})}{\sigma_c} \quad (22)$$

where $\bar{c}$ is the mean value of satellite dataset, σ_c is the dataset standard deviation, c_i is the position to be normalized and c'_i is the normalized position.

Table 6.2: Example of interpolation of satellites positions at the 300 seconds

X	Y	Z	t (s)
-20927	14289	-8245	0
?	?	?	300
-22223	14305	-2748	900

Note that because each satellite position has three coordinates (X, Y, Z) it is necessary to calculate the interpolation for each of the three coordinates separately at each time. For example to interpolate the satellite's position in Table 6.2 at the 300 seconds, firstly it is interpolated the X coordinate with the -20927 and -22223 values, then the Y coordinate it is interpolated with 14289 and 14305 values respectively and finally the Z coordinate it is interpolated with -8245 and -2748. After all the interpolations we obtain the satellite position at the 300 seconds at the interpolated X, Y, Z coordinates. In [114] the polynomial interpolation with 11 terms showed the best results, with this in mind the same 11 terms parameter was chosen for neville's polynomial interpolation in SlantDelayJoinProcessing component.

Using the options RECEIVER, RINEXFILE, VARSFILE and ZTDFILE in the SlantDelayJoinProcessing configuration file, the various receivers information is passed to the SlantDelayJoinProcessing component. The RECEIVER option just specifies the name of the receiver. The RINEX file corresponds to the RINEX (Receiver Independent EXchange Format) observation file. This file contains diverse information and a full description can be found in [117]. For the SlantDelayJoinProcessing component we use this file to retrieve which satellites were available at a determined time to the receiver. An example of this file can be found at in Appendix A.0.5. VARSFILE option specifies a meteorological file with diverse meteorological data. The data that this component takes from the file is surface pressure (mbar) and surface temperature (Celsius). An example of this file can be found in Appendix A.0.6. The ZTDFILE is the last option and specifies a SINEX_TRO (Solution INdependent EXchange Format for combination of TROpospheric estimates) file [118]. It includes the ZTD value (mm) and the gradients values. A full description of this format can be found in [118]. An example of this file can be found in Appendix A.0.7.

Overall the SlantDelayJoinProcessing component works as follow:

1. For the receivers that need to be processed, both the SP3 file and Rinex file are read and the program retreives every satellite instance which is at the two files at the same time (Rinex file allows us to know which sattelites the receiver had visible at a time. This operation results in a file named "rinexsp3.sync" containing all the synchronization instances.

1.1. If specified the SP3 interpolation is executed here. The SP3 interpolation is executed over the existing 15 minutes data interval to provide 5 minutes data interval. The interpolated data is then used for the synchronization in 1. .

1.2. In this step the program also calculates the elevation and the azimuth angle accordingly with the following formulas [119]:

$$e = arcsin(\vec{V} \cdot \vec{U}) \quad (23)$$

$$\vec{V} = \begin{pmatrix} (x' - x)/d \\ (y' - y)/d \\ (z' - z)/d \end{pmatrix} \quad (24)$$

$$d = \sqrt{(x' - x)^2 + (y' - y)^2 + (z' - z)^2} \quad (25)$$

$$\vec{U} = \begin{pmatrix} x/r \\ y/r \\ z/r \end{pmatrix} \quad (26)$$

$$r = \sqrt{x^2 + y^2 + z^2} \quad (27)$$

where e is the elevation angle, $\{x, y, z\}$ are ECEF (Earth-Centered, Earth-Fixed) receiver's coordinates, $\{x', y', z'\}$ are the ECEF satellite's coordinates, $\vec{V}$ is the unit direction vector from the receiver to the satellite, d is the distance from the receiver to the satellite, $\vec{U}$ is the "up" unit vector, r is the norm of the receiver ECEF vector.

$$\theta = atan(\frac{V_E}{V_N}) \quad (28)$$

$$V_E = \vec{V} \cdot \vec{E} \quad (29)$$

$$V_N = \vec{V} \cdot \vec{N} \quad (30)$$

$$\vec{E} = \begin{pmatrix} -y/p \\ x/p \\ 0 \end{pmatrix} \quad (31)$$

$$\vec{N} = \begin{pmatrix} -x \cdot z/(p/r) \\ -y \cdot z/(p/r) \\ p \cdot r \end{pmatrix} \quad (32)$$

where θ is the azimuth angle, $\vec{N}$ is the north unit vector, $\vec{E}$ is the east unit vector and p is the norm of the cross product of the unit vector $\vec{z} = \{0, 0, 1\}$ by the receiver position $\{x, y, z\}$ vector.

2. Both the VarsFile and ZTDFile are read and the program synchronizes the data over the time interval. Finally it outputs the result to the file "ztdvars.sync". This file contains the synchronized temperature, pressure, ZTD and gradients.
3. After the previous two steps the application synchronizes the their output: the "rinexsp3.sync" file containing the satellites position and elevation, azimuth angles and the "ztdvars.sync" file containing temperature, pressure, ZTD and the gradients. Their synchronization results in the file "elevationSatInfo.sync".
4. In this step the application reads the file generated in the previous step ("elevationSatInfo.sync"), calculates the IWV and the Slant Wet Delays for each instance in the file using the procedure of Chapter 5.3. Finally this output is written to the "SlantResults.sync" file.
5. If there is more receivers, one is selected and is processed going back to the step 1. . This is repeated until there are no any more receivers to process. The data of all receivers is saved in the "SlantsJoined.sync" file.

6.1.3 GridRayIntersection Component

This component takes care of the process of superimposing a grid over the atmosphere, obtain the intersections with the Slant Wet Delays and measuring the line segments for each Slant Wet Delay which allows us to determine the contribution of each ith voxel to the jth Slant Wet Delay (w_{ij}). This process was explained in Chapter 2 and in Chapter 5.

The options available for this component are the following:

WaterVaporReconstruction Command Line Options

```
--help                produce help message
--gridfile arg        grid file name (to be generated or used)
--slantsfile arg      slant delays file name
--cutoff arg          cutoff angle, only use observations above this
--gengrid arg         generates the grid using a file with grid
                      specifications
```

The -gengrid option allows the generation of a regular grid automatically, specifying only a few parameters: latitude max, latitude min, longitude min, longitude max, height max, height min and latitude/longitude/height intervals. An example file that can be used is the following:

Example configuration file to generate a grid

```
# Grid parameters
# Comments start with '#'

# Latitude
LatitudeMin -1.5
LatitudeMax -1.1
LatitudeInterval 0.1

# Longitude
LongitudeMin -48.5
LongitudeMax -48.1
LongitudeInterval 0.1

# Height
HeightMin 0
HeightMax 16000
HeightInterval 1000
```

The -gridfile when used with -gengrid allows to specify the filename of the file to be generated while when used alone specifies the grid file to be used. The slant delays file is specified with the -slantsfile option. A cut off angle may be specified using the -cutoff option, the program will only use slant delays with elevation angle above the specified value.

The intersections of the slant delays with the grid are performed here in the following away: Each slant delay line is projected from the receiver to the satellite using the elevation and azimuth angles. Then this slant delay is decomposed by a constant. The constant used is 10 meters. This means that each ray segment of the slant delay will have a length of 10 meters. Each ray segment is then checked to see if it is inside each voxel. If the ray segment is inside the voxel, the distance which relates the slant delay and that voxel is updated. At the end this component generates a matrix and a vector file. The matrix file generated is called "outputFileM.txt" and saves the matrix A of the equation (2) in the ASCII Market format (a standard sparse matrix file format) [120]. This format is a usual text format to save sparse matrices which is the case. The advantage of using this format is that it saves space and time (to writing and reading the file). This matrix contains every contribution of slant delays to each of the voxels of the grid. The vector file generated in this component is called "outputFileV.txt" and it saves every slant delay value in binary format, this improves the time for reading and saving the file. This vector corresponds to the vector b in the equation (2).

6.1.4 AlgebraicAlgorithms Component

This component loads the matrix A and the vector b created by the GridRayIntersection component and executes the inversion of the equation (2) to compute the vector x with the solution

of the tomography inversion.

The options available for this component are the following:

AlgebraicAlgorithms Command Line Options

```
--help                produce help message
--matrixfile arg      matrix file name (market sparse format)
--vectorfile arg      vector file name (binary)
--algorithm arg       Algebraic algorithm to use. Possible options:
                      kaczmarz, kaczmarzSymetric, kaczmarzRandomized,
                      landweber, cimmino, cav, drop, sart
 -iterations arg      number of iterations desired
```

The -matrixfile argument specifies the ASCII Market format matrix file, -vectorfile specifies the vector binary file, -algorithm specifies the algebraic reconstruction algorithm to use in the inversion and the -iterations argument specifies how many iterations should the selected algebraic reconstruction algorithm execute. The algorithms avaiable for the inversion are the of the sections 2.2.1 and 2.2.2. This component saves the result vector of the 3D inversion in the file “reconsImage.dat” in the binary format.

6.1.5 PlotWaterVapor Component

This component simply plots the result image of the tomography inversion problem. The 3D image of the water vapor is presented in 2D slices. The slices can be in latitude, longitude or height. It reads the binary vector file created in AlgebraicAlgorithms component. It also reads the grid file to know which are the grid specifications.

The options available for this component are the following:

PlotWaterVapor Command Line Options

```
Example: PlotWaterVapour.exe reconsImage.dat GridFile_BELEM.dat 48 lat-height
param1          vector file which contains the reconstructed image
param2          grid file name which was used to create the image
param3          slice to show the image (for example 48)
param4          slice type, possible options: lat-lon, lat-height, lon-height
param5(opt)     type of countour used with contourf function [default: 15]
param6(opt)     limitTo5000, limits the plot to 5000 altitude [default: false]
```

Parameter 1 specifies the filename of the vector file created with the AlgebraicAlgorithms component, parameter 2 specifies the grid file, parameter 3 specifies the slice which the component will present, parameter 4 specifies the slice type (latitude/longitude/height), parameter 5 specifies the type of contour and parameter 6 specifies if the height to be displayed will be limited to 5 km. Parameters 5 and 6 are optional. The contour command displays isolines calculated from the image matrix and fills the areas between the isolines using constant colors corresponding to the figure’s colormap. If it was not used the images would look pixelated.

An image of the water vapor in the atmosphere created with the PlotWaterVapor can be visualized in Figure 6.2. The water vapor density can be checked looking at the color bar. The density

values are in g/m^3.

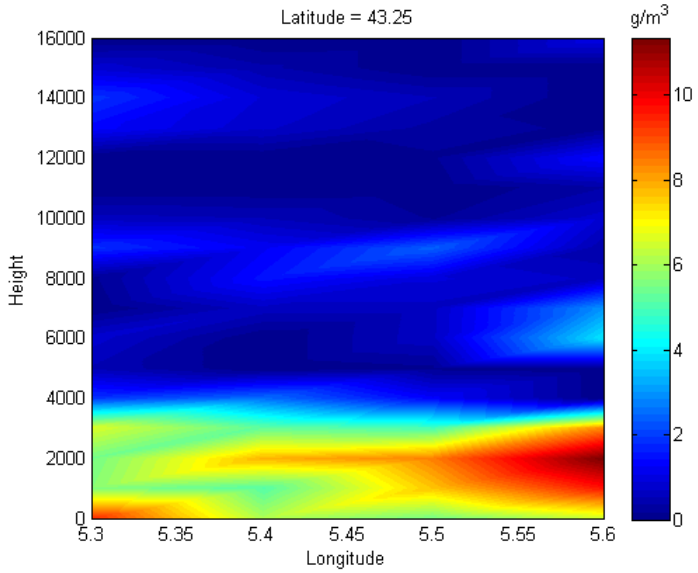

Figure 6.2: Image created with PlotWaterVapor component in 43.25 latitude slice.

6.2 Comparison with LOFTT_K

In this section a brief introduction to LOgiciel Français de Tomographie Troposphérique -version Kalman (LOFTT_K) is made and a comparison with SWART is presented.

LOFTT_K is a GNSS water vapor tomography software developed in the laboratoire dynamique de la lithosphere and the service d'aéronomie by C. Champollion *et al.* [21]. It also reconstructs the 3D water vapor distribution in the atmosphere like SWART. We now give a comparison of the main features of LOFTT_K and SWART.

- LOFTT_K was written in FORTRAN language while SWART was mainly written in C++ with a component written in MATLAB.
- LOFTT_K uses a digital terrain model which results in a grid that takes in account the actual terrain form like if it has mountains or rivers. SWART does not use a digital terrain model yet.
- LOFTT_K LOFTT_K supports irregular grids (voxels with different sizes), SWART also supports irregular grids, however the grid generation procedure is more complex.
- The method to calculate the length of the voxel intersections for each slant wet delay (see Chapter 2).) is the same in both softwares.
- LOFTT_K implements a Kalman filter which allow the program to take in account the rapid or slow variation of the water vapor in the atmosphere [121]. SWART doesn't implement a Kalman filter yet.
- The principal difference between SWART and LOFTT_K consists in the matrix inversion method. While LOFTT_K uses the Single Value Decomposition (SVD) method SWART uses

parallel algebraic reconstruction algorithms. Algebraic reconstruction algorithms have several advantages over other methods for the water vapor tomography (see Chapter 2).

Both the programs enable the resulting images to be presented in several slices of the 3D image, namely slices in latitude/longitude and height.

6.3 Synthetic data results

SWART was firstly tested with synthetic data. In these tests synthetic slant (artificial) wet delay data was used. Some real data from the COPS (Convective and Orographically-induced Precipitation Study) campaign (https://www.uni-hohenheim.de/spp-iop/index.htm), was also used including the grid data, satellite positions and receiver positions. The results were also compared with LOFTT_K in order to evaluate our SWART program. For the SWART software the SART algorithm was chosen with 500 iterations; this choice was determined by convergence results and advantages over other algorithms which will presented in the next section (Section 6.4). The area covered by these tests was the eastern of France. The matrix used in both programs had a size of 1087x350 (1087 slant wet delays and 350 voxels).

Two images enabling a comparison of the results of both programs are shown in the next two figures. Each pair of images uses a different 2D slice (latitude and longitude). The values measured correspond to the density of water vapor in g/m^3.

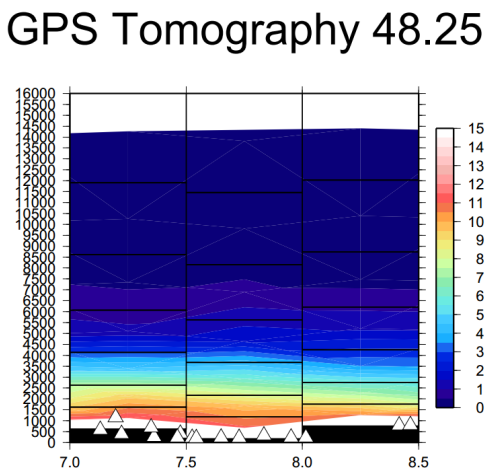

Figure 6.3: LOFTT_K 48.25 latitude slice.

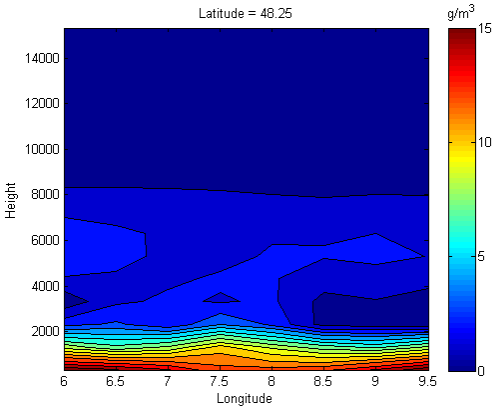

Figure 6.4: SWART 48.25 latitude slice.

Figure 6.3 shows the reconstruction using LOFT_K in the 48.25 latitude slice. Figure 6.4 is the same slice but using SWART. One large observable difference is the use of the digital terrain model in LOFTT_K. With the digital terrain model the tomography does not begin exactly at the zero height. While in our implementation it always begins at zero height. The SWART image also does not seem to be as smooth as the LOFTT_K image *e.g.* especially between 2 and 8 km. This may be a consequence of not using the Kalman filter that LOFTT_K implements. The values of the density of the water are approximately the same for both figures.

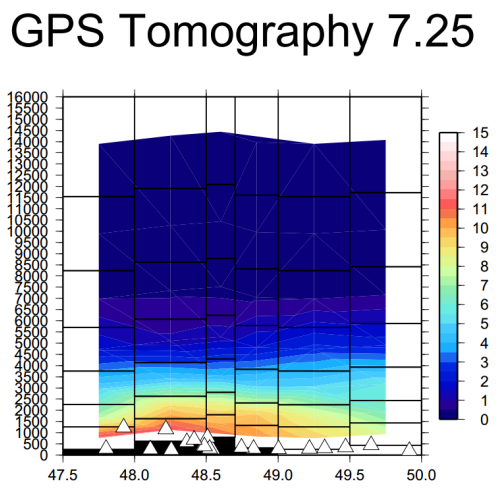

Figure 6.5: LOFTT_K 7.25 longitude slice.

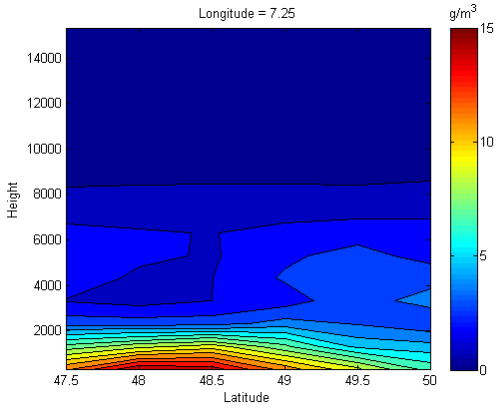

Figure 6.6: SWART 7.25 longitude slice.

Figure 6.5 and Figure 6.6 shows an other comparison of the program results but now for the 7.25 longitude slice. The values of the density of water remain very close and the images are also very alike. SWART still does not have the digital terrain model that LOFTT_K implements. Also it is observable the SWART image is not as smooth as LOFTT_K between 2 and 8 km.

6.4 Results of the Case Studies

In this dissertation two case studies were analysed. The two networks of GPS receivers that were used to perform the water vapour image reconstruction were located in Brazil and the other in France. The parallel SART algorithm was chosen for the water vapor image reconstruction. This decision was taken because SART has some advantages over other algebraic reconstruction algorithms including faster convergence and computational efficiency [18]. The convergence of the SART algorithm can be seen in Figure 6.7, this figure was produced reconstructing a real GNSS water vapor image and calculating the root mean square (rms) of each reconstructed imaged in each iteration. According with [3] the number of necessary iterations depends on the quality of the initial field, the data quality and other parameters. Looking at Figure 6.7 we can conclude that less than 50 iterations is not sufficient in order to obtain a good image reconstruction. For the tests and figures of this section 500 iterations was chosen as the variation of the convergence in Figure 6.7 seem to be very small at this number of iterations. In the next sections these two case studies will be described and the results of the GNSS water vapor tests using the SART algorithm with 500 iterations are presented.

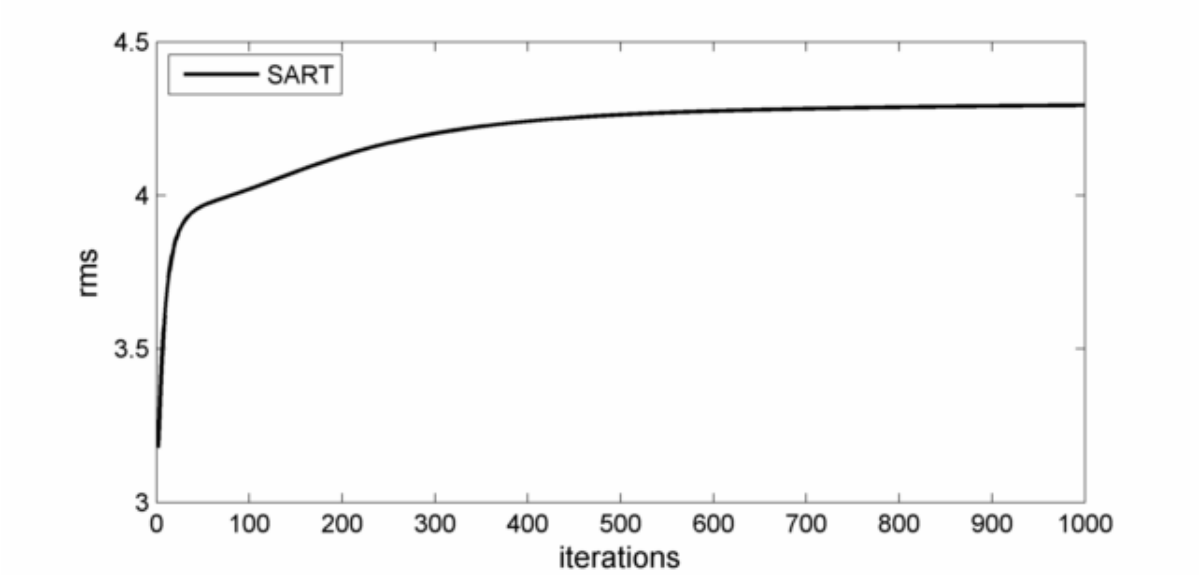

Figure 6.7: Convergence of SART algorithm for 1000 iterations.

6.4.1 Marseilles Network

The first network used is a dense network used in the ESCOMPTE [21] campaign with 17 dual frequency GPS receivers within a 20 x 20 km area around Marseilles, France. This campaign was run in 2001 in June and July months [21]. The altitude of this GNSS network area varies from the sea level to the top of Etoile chain (up to 700 meters). The Marseilles network area and receivers position can be seen in Figure 6.8 [21].

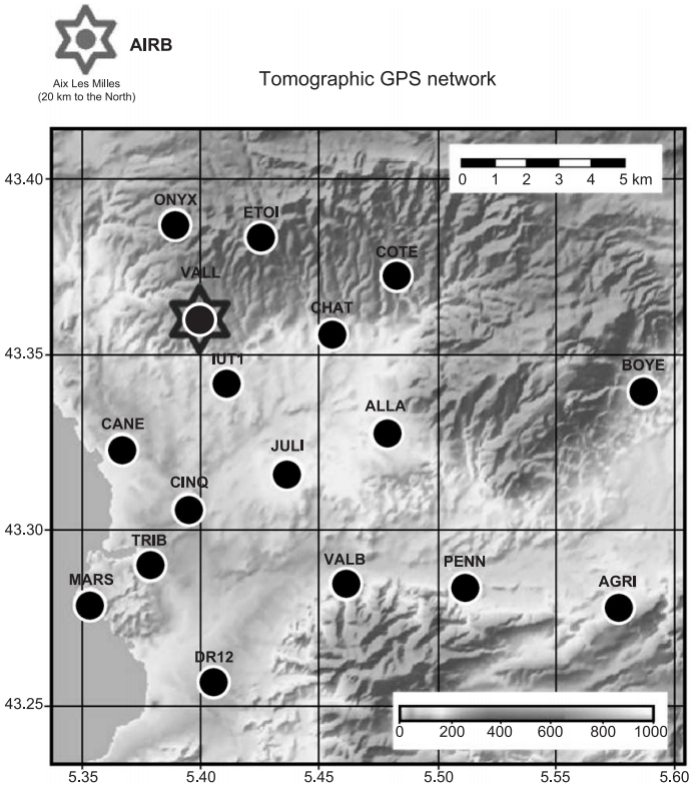

Figure 6.8: Marseilles network area and receivers positions.

The Marseilles network tests were realized with a horizontal step size of 0.1°x 0.1°and a vertical step size of 1000 meters for. The tests were realized using data from June 24,2001 (DOY 175) for the 14 hours.

The Marseilles network tests are presented in two different slices of latitude, longitude and height and can be seen in the figures 6.9, 6.10, 6.11, 6.12, 6.13 and 6.14.

Looking at the reconstructed images we can see that the distribution of the water vapor is concentrated up to the 2 km mark. In [122] it is mentioned that half of the water vapor in the atmosphere is distributed below an altitude of about 1.5 km, less than 5% is above 5 km and 1% above 12 km. The Marseilles network images are in line with this general description of the water vapor distribution. The results obtained also show that the maximum density of the water does not surpass the 15-20 g/m^3.

6.4.2 Belem Network

The second case study is a GPS network project that was installed in Belem, Brazil for the Chuva Project [123]. This campaign was run in 2011 between May and July. The Belem network was composed of 15 receivers within an area of approximately 110 x 110 km. The Belem area and receivers position can be seen in Figure 6.15.

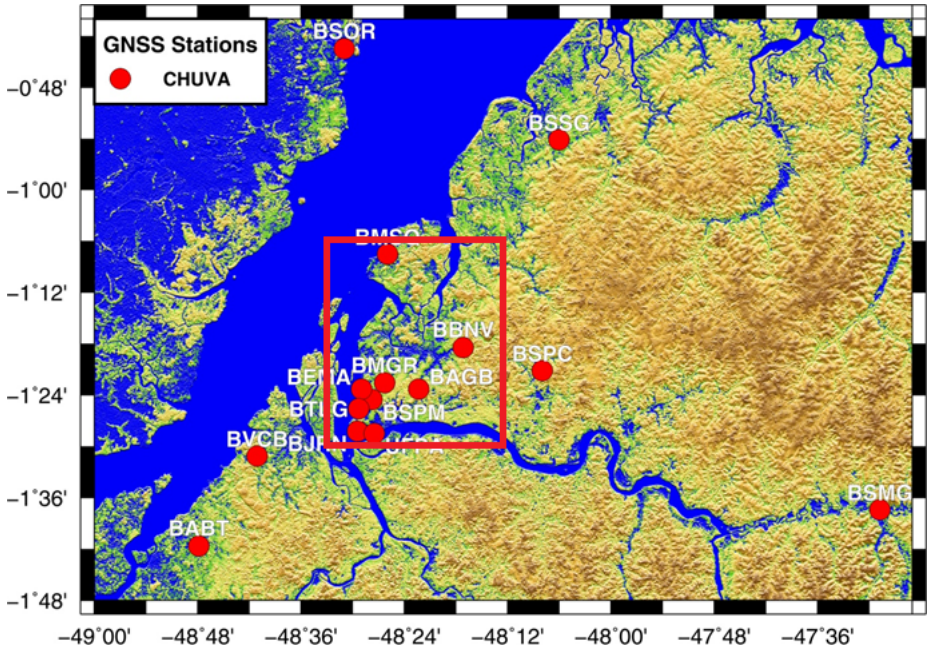

Figure 6.15: Belem network.

From previous tests it was concluded that the best results were obtained if only the area within the red rectangle area of Figure 6.15 was used as the receivers are denser in this sub area when compared to the total number of stations available in the much larger area. This sub-area is about 45 x 35 km and was used in the following network tests.

The tests were made using a horizontal step size of 0.1°x 0.1°and a vertical step size of 1000 meters for the 6th June (DOY 157) at 11:30 hours. As in the Marseilles network tests the results

are presented in two different slices of latitude, longitude and height and are shown in figures 6.16, 6.17, 6.18, 6.19, 6.20 and 6.21.

The Belem network reconstructions also seem to generally fit with [122] water vapor distribution descriptions for this area of Brazil. The maximum water vapor density is distributed below the 2 km and reaches up to 30 g/m^3, clearly more than the Marseilles experiment. Please note that the area of this reconstruction is larger than the Marseilles (35 x 45 km against 20 x 20 km) and less receivers were available (15 against 17).

Notice that the distribution of water vapor in figures 6.17 and 6.18 shows regions with significant water vapor density between 4 and 7 km. However as the main focus of this dissertation has not been to compare or validate our software with real data produced by for instance radiosondes, a discussion of such effects is best left to future and continued work using our software.

6.5 Summary

In this chapter the SWART software that has been developed for Water Vapor Image reconstruction was presented. The various components were described and explained. It was also compared with LOFTT_K an another GNSS Water Vapor Image Reconstruction software. Some preliminary tests were realized namely with synthetic data and the results were compared with LOFTT_K. Also some preliminary tests with case of studies were realized. This tests were realized with data from campaigns in Brazil and in France. The campaigns water vapor image reconstruction results showed to be coherent with the normal distribution of the water vapor in the atmosphere which was described in this chapter. For all the tests SART algorithm was used due to its advantages over other algorithms [18] and was concluded that the SART algorithm reconstruction needed at least 50 iterations to begin to reconstruct the image and up to 500 iterations to attain the final result.

In the next and final chapter the overall conclusions and future work will be presented.

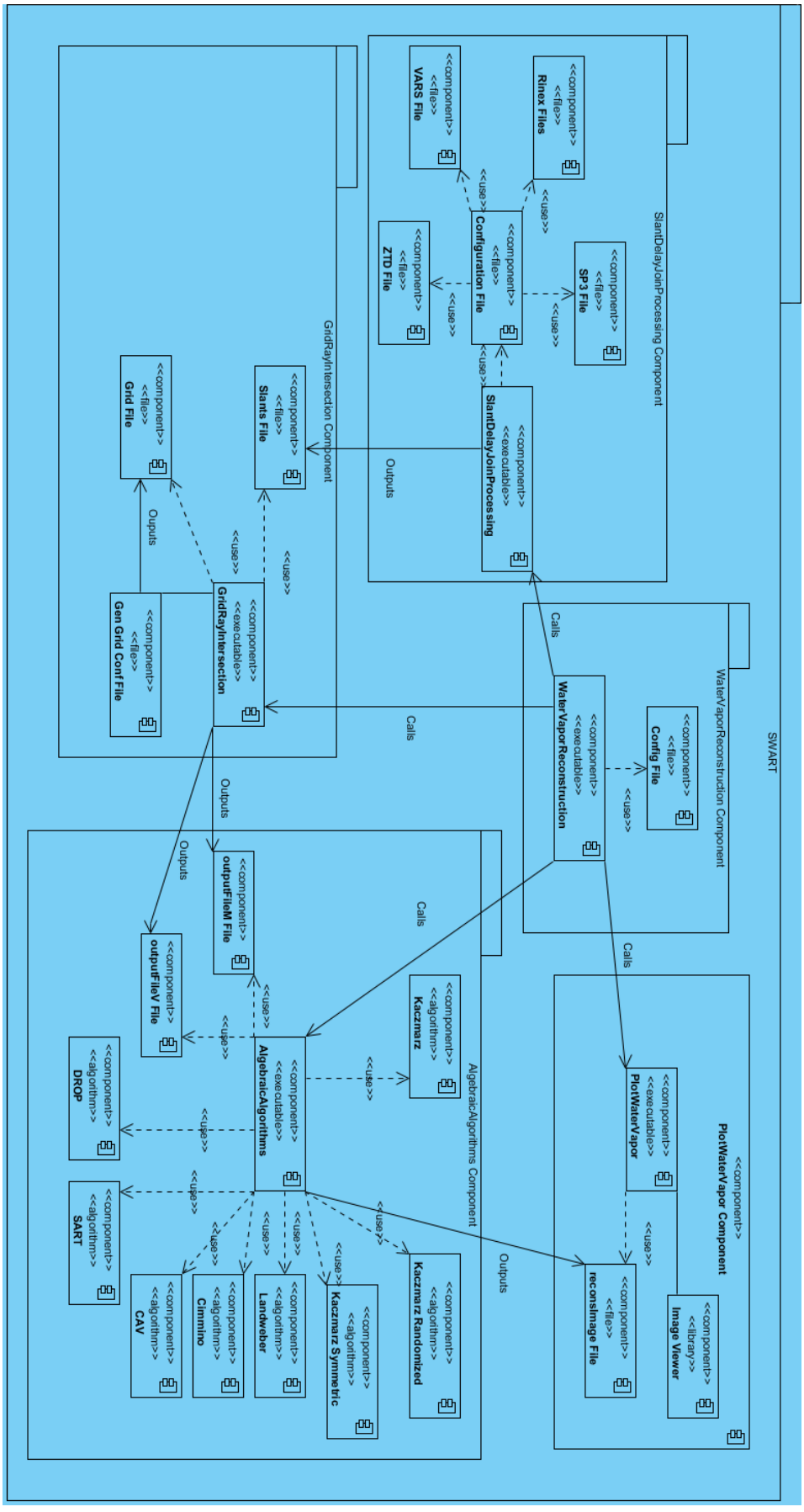

Figure 6.1: SWART UML component diagram.

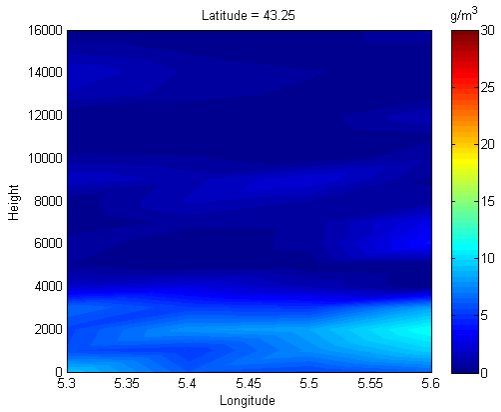

Figure 6.9: SWART slice in latitude 43.25 for Marseilles network.

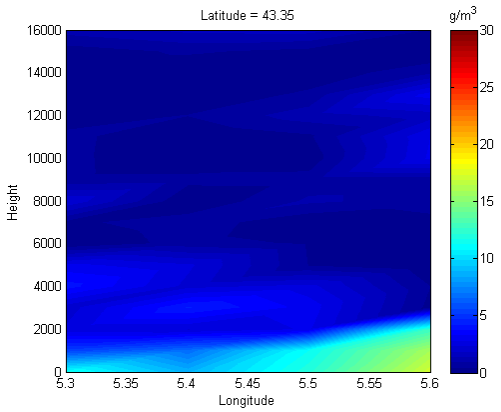

Figure 6.10: SWART slice in latitude 43.35 for Marseilles network.

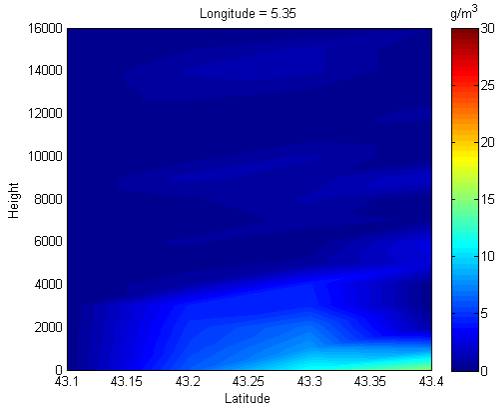

Figure 6.11: SWART slice in longitude 5.35 for Marseilles network.

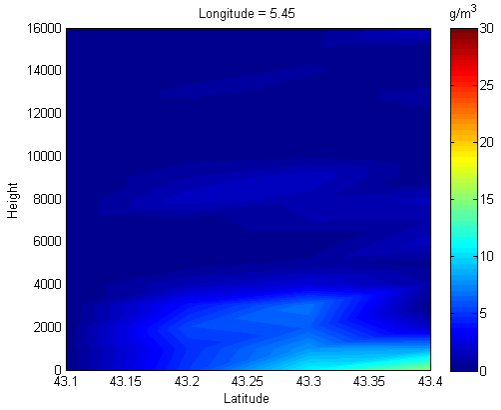

Figure 6.12: SWART slice in longitude 5.45 for Marseilles network.

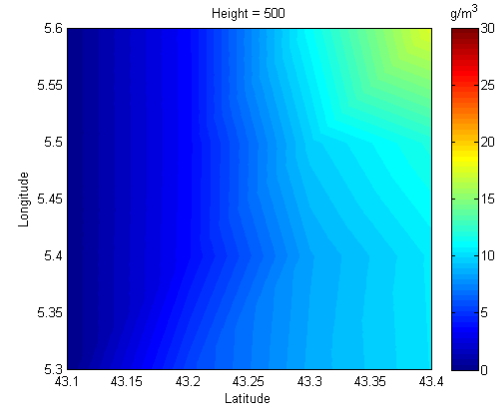

Figure 6.13: SWART slice in height 500 for Marseilles network.

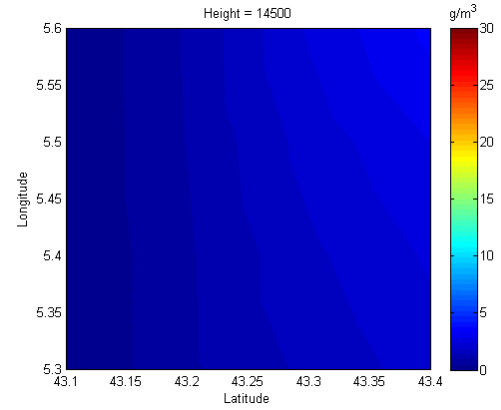

Figure 6.14: SWART slice in height 14500 for Marseilles network.

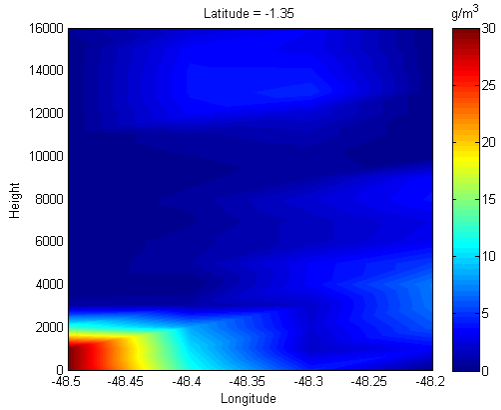

Figure 6.16: SWART slice in latitude -1.35 for Belem network.

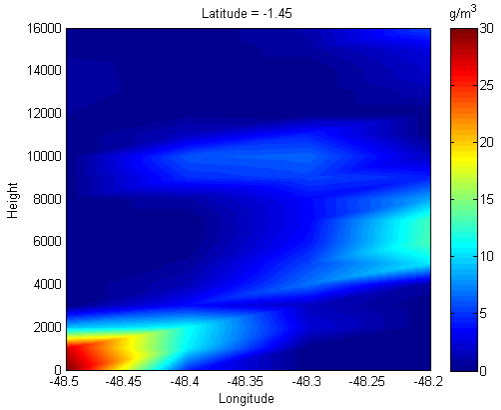

Figure 6.17: SWART slice in latitude -1.45 for Belem network.

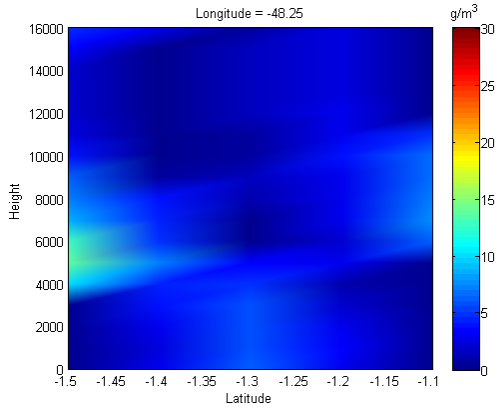

Figure 6.18: SWART slice in longitude -48.25 for Belem network.

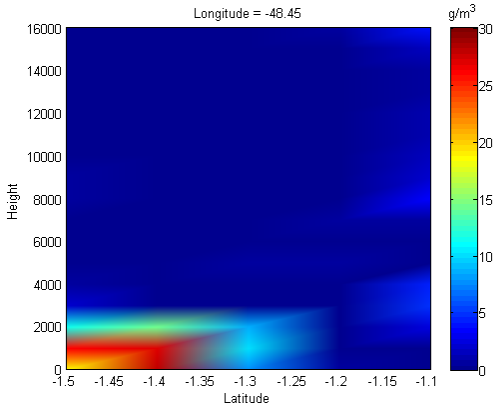

Figure 6.19: SWART slice in longitude -48.45 for Belem network.

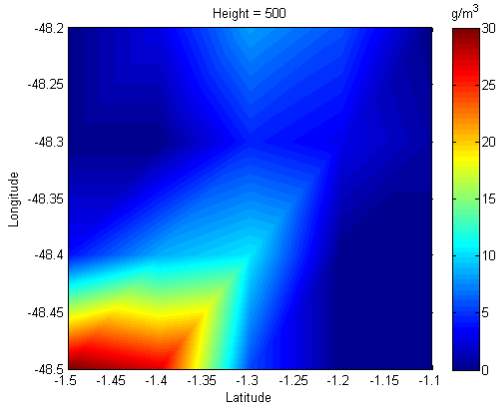

Figure 6.20: SWART slice in height 500 for Belem network.

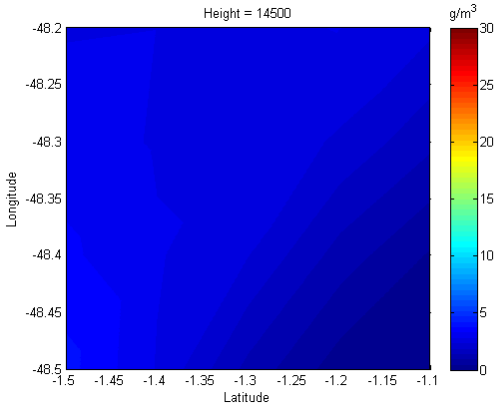

Figure 6.21: SWART slice in height 14500 for Belem network.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

The main objective of this dissertation was to perform a detailed study on the advantages of implementing parallel algebraic reconstruction algorithms and this was completed with success.

The main algebraic reconstruction algorithms were implemented (Kaczmarz, Symmetric Kaczmarz, Randomized Kaczmarz, Landweber, Cimmino, CAV, DROP and SART) resulting in a software suite of C++ that will be used and made available by SEGAL. The implementations were tested by reconstructing a Shepp-Logan phantom image. In this dissertation the results of the SART, Landweber and Kaczmarz Shepp-Logan reconstructions were presented and it was observed that the original image that the main features of the original image could be reproduced by all the algorithms. The results of the tests performed show that the three methods produce similar results without be possible to favor one unequivocally, even if the Kaczmarz was slightly better. More tests using different original images and number of iterations could help to better clarify if the results that we obtained here could be more generalized. One particular issue that the tests performed using the Shepp-Logan phantom image could not clarify is the identified noise that several images reconstructed with the Kaczmarz method can suffer. Such issue can be a major problem for GNSS water vapor tomography. Therefore, in the following implementations, the SART method was adopted.

Regarding the parallelization of the algebraic reconstruction algorithms, it was concluded that the parallelization compensates over sequential implementations. In the CPU tests the Eigen3 library was shown to have the best overall performance, although it was found that this library is less efficient when the architecture of the machine has more virtual than physical cores. The overall speedup for the CPU implementation of the algorithms was found to be much more significant for the SART and Landweber algorithms than for the ART Kaczmarz. Overall the GPU (CUBLAS) implementations were found to be slightly faster than the CPU implementations and the speedups obtained were better, except for the Kaczmarz case. The choice of algorithm and architecture for any particular problem depends on many factors, if a GPU is available, the size of the problem, etc.. Nevertheless, the results obtained in the test carried out are sufficient to act as a guide for real problems, such as the water vapor tomography software also developed in this dissertation.

The acquired experience permitted to start the developing of a dedicated software package called SWART for GNSS water vapor tomography. Although not finalized, the preliminary results were shown to be satisfactory. Such software package is particular important for the scientific community attending the lack of reliable (and free available) software for GNSS water vapor tomography. The synthetic data results were shown to be almost identical to results obtained from the software described in [121] for exactly the same data. Regarding the results from the two case studies, it was observed that the SART algorithm reconstruction needed at least 50 iterations to begin to properly reconstruct the image. After 500 iterations, the image did

not suffer significant improvements. Furthermore, the results obtained using real data (from a network in Marseille) were coherent with the normal distribution of the water vapor in the atmosphere as shown in Section 6.4. The density of the water did not surpass the 15-20 g/m^3 in the analysed images. The results for Belem also generically fit the results generally also fit with normal distribution of the water vapor in the atmosphere. The maximum water vapor density in Belem reached approximately 30 g/m^3, which is considerable more than the maximum density in Marseille. This is expected attending to the humidity of both regions.

In respect to the GNSS water vapor reconstruction it must be referred that obtaining real campaign data was not an easy task, *e.g.* the Marseilles' data was only recently received and also the data needed to be converted and synchronized to different formats because it was different than the expected formats and in fact completely different again from the formats used for Belem reconstruction. A lack of standardization in data formats is in fact a significant problem and one that hopefully will be addressed by projects that drive data and meta-data standardization such as EPOS (The European Plate Observing System, http://www.epos-eu.org).

There were also many other limitations to a complete and thorough analysis of the software being developed *e.g.* the lack of other data to compare with SWART results such as radiosondes data, microwave radiometers or meteorological reports for the date and times studied.

7.2 Future Work

There are many possibilities for future work. Relatively to the algebraic reconstruction algorithms it will be important:

- to analyse the converge and the quality of the images of the different parallelized algorithms for different parameters;
- to extend the GPU parallelization for multiple GPUs;
- create hybrid parallelization approaches that make use of both CPU and GPU power;
- evaluate a cloud based approach to parallelize these algorithms such as Windows Azure.

Relatively to the GNSS water vapor image reconstruction software there are several topics still should be considered:

- validate the image reconstructions with data from other instruments such as radiosondes, microwave radiometers and Lidars;
- implement other more recent mapping functions besides the Niell's, for example the Vienna Mapping Functions (VMF) [124] or the Global Mapping Functions (GMF) [125];
- integrate digital terrain models in order that the program take into account the actual terrain form;
- implement a Kalman Filter approach to take into account the temporal variation vapor in the atmosphere.

References

[1] R. Gordon, R. Bender, and G. T. Herman, "Algebraic reconstruction techniques (art) for three-dimensional electron microscopy and x-ray photography." *Journal of Theoretical Biology*, vol. 29, no. 3, pp. 471–481, 1970. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/5492997 1

[2] S. Kaczmarz, "Angenaherte auflosung von systemen linearer gleichungen," *Bull. Acad. Polon*, vol. Sci Lett. A, pp. 355–357, 1937. 1

[3] M. Bender, G. Dick, M. Ge, Z. Deng, J. Wickert, H.-G. Kahle, A. Raabe, and G. Tetzlaff, "Development of a gnss water vapour tomography system using algebraic reconstruction techniques," *Advances in Space Research*, vol. 47, no. 10, pp. 1704–1720, 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0273117710003790 1, 5, 15, 56, 57, 69

[4] W. Hao, L. Guoping, and W. Dan, "A method of inserting and mending for the gps precipitable water vapor," in *Multimedia Technology (ICMT), 2011 International Conference on*, 2011, pp. 3350–3353. 1

[5] M. Bender and A. Raabe, "Preconditions to ground based gps water vapour tomography," *Annales geophysicae*, vol. 25, no. 8, pp. 1727–1734, 2007. [Online]. Available: http://hal.archives-ouvertes.fr/hal-00318359/ 1, 13

[6] S. Lutz, M. Troller, D. Perler, A. Geiger, and H.-G. Kahle. (2013) Innovation: Better weather prediction using gps. GPS World. Accessed: 15/12/2012. [Online]. Available: http://gpsworld.com/innovation-better-weather-prediction-using-gps/ 1

[7] J. Van Baelen, J.-P. Aubagnac, and A. Dabas, "Comparison of near-real time estimates of integrated water vapor derived with gps, radiosondes, and microwave radiometer," *Journal of Atmospheric and Oceanic Technology*, vol. 22, no. 2, pp. 201–210, 2005. [Online]. Available: http://journals.ametsoc.org/doi/abs/10.1175/JTECH-1697.1 1, 52

[8] M. Bevis, S. Businger, T. A. Herring, C. Rocken, R. A. Anthes, and R. H. Ware, "Gps meteorology - remote sensing of atmospheric water vapor using the global positioning system," *Journal of Geophysical Research*, vol. 97, no. D14, pp. 15 787–15 801, 1992. [Online]. Available: http://dx.doi.org/10.1029/92JD01517 1, 51, 54

[9] C. Melvin, M. Xu, and P. Thulasiraman, "HPC for iterative image reconstruction in CT," in *Canadian Conference on Computer Science & Software Engineering*, 2008, pp. 61–68. 1, 16, 17

[10] A. C. Kak and M. Slaney, Eds., *Principles of Computerized Tomographic Imaging*. New York: IEEE Press, 1988. 5, 6, 7, 8, 10

[11] S. Kaczmarz, "Angenäherte Auflösung von Systemen linearer Gleichungen," *Bulletin Internat. Acad. Polon. Sciences et Lettres*, pp. 355–357, 1937. 7

[12] P. C. Hansen and M. Saxild-Hansen, "AIR tools - A MATLAB package of algebraic iterative reconstruction methods," *J. Computational Applied Mathematics*, vol. 236, no. 8, pp. 2167–2178, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.cam.2011.09.039 10, 11, 12

[13] G. T. Herman, *Fundamentals of Computerized Tomography: Image Reconstruction from Projections*. Springer-Verlag, 2009. [Online]. Available: http://www.springer.com/computer/computer+imaging/book/978-1-85233-617-2 10

[14] C. D. C. D. Meyer, *Matrix analysis and applied linear algebra*. pub--SIAM:adr: Society for Industrial and Applied Mathematics, 2000. [Online]. Available: http://www.loc.gov/catdir/enhancements/fy0668/00029725-d.html;http://www.loc.gov/catdir/enhancements/fy0668/00029725-t.html 10

[15] G. Cimmino, "Calcolo approssimato per le soluzioni dei sistemi di equazioni lineari," *Ric. Sci. Progr. Tecn. Econom. Naz.*, vol. 9, pp. 326–333, 1939. 11

[16] Y. Censor, D. Gordon, and R. Gordon, "Component averaging: An efficient iterative parallel algorithm for large and sparse unstructured problems," *Parallel Computing*, vol. 27, no. 6, pp. 777–808, 2001. [Online]. Available: http://dx.doi.org/10.1016/S0167-8191(00)00100-9 11

[17] Y. Censor, T. Elfving, G. T. Herman, and T. Nikazad, "On diagonally relaxed orthogonal projection methods," *SIAM Journal on Scientific Computing*, vol. 30, no. 1, pp. 473–504, 2008. 12

[18] A. H. Andersen and A. C. Kak, "Simultaneous algebraic reconstruction technique (SART): A superior implementation of the art algorithm," *Ultrasonic Imaging*, vol. 6, no. 1, pp. 81–94, Jan. 1984. [Online]. Available: http://cobweb.ecn.purdue.edu/RVL/Publications/SART_84.pdf 12, 69, 72

[19] Y. Censor and T. Elfving, "Block-iterative algorithms with diagonally scaled oblique projections for the linear feasibility problem," *SIAM Journal on Matrix Analysis and Applications*, vol. 24, pp. 40–58, 2002. [Online]. Available: http://www.optimization-online.org/DB_HTML/2001/12/418.html 12

[20] M. Jiang and G. Wang, "Convergence of the simultaneous algebraic reconstruction technique (SART)," *IEEE Trans. Image Processing*, vol. 12, no. 8, pp. 957–961, Aug. 2003. [Online]. Available: http://dx.doi.org/10.1109/TIP.2003.815295 12

[21] C. Champollion, F. Masson, M.-n. Bouin, A. Walpersdorf, E. Doerflinger, O. Bock, and J. Van Baelen, "Gps water vapour tomography: preliminary results from the escompte field experiment," *Atmospheric Research*, vol. 74, no. 1-4, pp. 253–274, 2005. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0169809504001474 13, 14, 56, 67, 70

[22] D. Perler, A. Geiger, and F. Hurter, "4D GPS water vapor tomography: new parameterized approaches," *Journal of Geodesy*, vol. 85, pp. 1–12, 2011. 13, 15

[23] P. Tregoning and M. Hendy, "Accuracy of absolute precipitable water vapor estimates from gps observations," *Journal of Geophysical Research*, vol. 103, no. D22, pp. 28 701–28 710, 1998. [Online]. Available: http://www.agu.org/pubs/crossref/1998/98JD02516.shtml 13

[24] C. Rocken, T. V. Hove, J. Johnson, F. Solheim, R. Ware, M. Bevis, S. Chiswell, and S. Businger, "GPS/STORM—GPS Sensing of Atmospheric Water Vapor for Meteorology," *Journal of Atmospheric and Oceanic Technology*, vol. 12, 1995. 13

[25] E. Doerflinger, R. Bayer, J. Chery, and B. Burki, "The global positioning system in mountainous areas: effect of the troposphere on the vertical gps accuracy," *Comptes Rendus*

De L Academie Des Sciences Serie Ii Fascicule a Sciences De La Terre Et Des Planetes, vol. 326, no. 5, pp. 319–325, 1998. 13

[26] R. Ware, C. Alber, C. Rocken, and F. Solheim, "Sensing integrated water vapor along gps ray paths," *Geophysical Research Letters*, vol. 24, no. 4, pp. 417–420, 1997. [Online]. Available: http://www.agu.org/pubs/crossref/1997/97GL00080.shtml 13

[27] C. Alber, R. Ware, C. Rocken, and J. Braun, "Obtaining single path phase delays from gps double differences," *Geophysical Research Letters*, vol. 27, no. 17, p. 2661, 2000. [Online]. Available: http://www.agu.org/pubs/crossref/2000/2000GL011525.shtml 13

[28] J. Braun, C. Rocken, and J. Liljegren, "Comparisons of Line-of-Sight Water Vapor Observations Using the Global Positioning System and a Pointing Microwave Radiometer," *Journal of Atmospheric and Oceanic Technology*, vol. 20, 2003. 13

[29] P. MIIDLA, K. RANNAT, and P. UBA, "Simulated studies of water vapour tomography," in *WSEAS TRANSACTIONS on ENVIRONMENT and DEVELOPMENT*, 2008, pp. 181 – 190. 13

[30] M. Bevis, S. Businger, S. Chiswell, T. A. Herring, R. A. Anthes, C. Rocken, and R. H. Ware, "GPS Meteorology: Mapping Zenith Wet Delays onto Precipitable Water," *Journal of Applied Meteorology*, vol. 33, pp. 379–386, 1994. 13, 54

[31] A. E. Niell, "Global mapping functions for the atmosphere delay at radio wavelengths," *Journal of Geophysical Research*, vol. 101, no. B2, pp. 3227–3246, 1996. [Online]. Available: http://www.agu.org/pubs/crossref/1996/95JB03048.shtml 13, 55

[32] A. Flores, G. Runi, and A. Rius, "4D tropospheric tomography using GPS slant wet delays," Oct. 13 1999. [Online]. Available: http://citeseer.ist.psu.edu/290077.html;http://www.ieec.fcr.es/gps/Annales_Preprint.ps 13, 14, 56

[33] J. J. BRAUN and C. ROCKEN, "Water vapor tomography within the planetary boundary layer using gps," in *International Workshop on GPS Meteorology*, Tsukuba, Japan, January 2003, http://dbx.cr.chiba-u.jp/Gps_Met/gpsmet. 14

[34] U. Foelsche and G. Kirchengast, "Tropospheric water vapor imaging by combination of ground-based and spaceborne GNSS sounding data," *Journal of Geophysical Research*, vol. 106, pp. 27 221–27 232, 2001. 14

[35] A. Flores, L. P. Gradinarsky, P. Elósegui, G. Elgered, J. L. Davis, and A. Rius, "Sensing atmospheric structure: Tropospheric tomographic results of the small-scale GPS campaign at the Onsala Space Observatory," *Earth and Planetary Science Letters*, vol. 52, pp. 941–945, 2000. 14

[36] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. Cambridge University Press, 2007, vol. 1. [Online]. Available: http://www.amazon.com/Numerical-Recipes-3rd-Edition-Scientific/dp/0521880688?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=0521880688 14

[37] T. Nilsson and L. Gradinarsky, "Water vapor tomography using GPS phase observations: simulation results," *IEEE T. Geoscience and Remote Sensing*, vol. 44, no. 10-2, pp. 2927–2941, 2006. [Online]. Available: http://dx.doi.org/10.1109/TGRS.2006.877755 14

[38] M. Troller, A. Geiger, E. Brockmann, J.-M. Bettems, B. Bürki, and H.-G. Kahle, "Tomographic determination of the spatial distribution of water vapor using GPS observations," *Advances in Space Research*, vol. 37, pp. 2211–2217, 2006. 15

[39] P. MIIDLA, K. RANNAT, and P. UBA, "Simulated studies of water vapour tomography," 2008. 15

[40] C. Melvin, P. Thulasiraman, and R. Gordon, "Parallel algebraic reconstruction technique for computed tomography," in *Parallel and Distributed Processing Techniques and Applications*, 2003, pp. 532–536. 16

[41] K. Kalarat, W. Narkbuakaew, C. Pintavirooj, and M. Sangworasil, "Rapid simultaneous algebraic reconstruction technique (sart) for cone-beam geometry on clustering system," in *TENCON 2005 2005 IEEE Region 10*, 2005, pp. 1–4. 16

[42] D. Gordon, "Parallel ART for image reconstruction in CT using processor arrays," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 21, pp. 365–380, 2006. 16

[43] J. A. Alvarez, J. Roca, and J. J. Fernandez, *Multithreaded tomographic reconstruction*, 2007, vol. 4757, pp. 81–88. 16

[44] X. Wan, F. Zhang, and Z. Liu, "Modified simultaneous algebraic reconstruction technique and its parallelization in cryo-electron tomography," in *Parallel and Distributed Systems (ICPADS), 2009 15th International Conference on*, 2009, pp. 384–390. 17

[45] B. Wilkinson and M. Allen, *Parallel Programming: Techniques and Applications Using Networked Workstations and Parallel Computers*. Upper Saddle River, New Jersey: Prentice-Hall, 2004. 17

[46] H. M. Hudson and R. S. Larkin, "Accelerated image reconstruction using ordered subsets of projection data," Aug. 24 1994. [Online]. Available: http://citeseer.ist.psu.edu/459212.html;http://www.osem.server-web.com/osem_IEEE_TMI.ps 17, 19

[47] M. Xu and P. Thulasiraman, "Mapping iterative medical imaging algorithm on cell accelerator," *International Journal of Biomedical Imaging*, vol. 2011, p. 843924. 17, 18

[48] K. Mueller and R. Yagel, "Rapid 3-d cone-beam reconstruction with the simultaneous algebraic reconstruction technique (sart) using 2-d texture mapping hardware," 2000. 18, 19

[49] L. Shepp and B. Logan, "The fourier reconstruction of a head section," *Nuclear Science, IEEE Transactions on*, vol. 21, no. 3, pp. 21–43, 1974. 18, 43

[50] F. Xu and K. Mueller, "Accelerating popular tomographic reconstruction algorithms on commodity pc graphics hardware," *Nuclear Science, IEEE Transactions on*, vol. 52, no. 3, pp. 654–663, 2005. 18, 19

[51] B. Keck, H. Hofmann, H. Scherl, M. Kowarschik, and J. Hornegger, "Gpu-accelerated sart reconstruction using the cuda programming environment," *Proceedings of SPIE*, vol. 7258, pp. 72 582B–72 582B–9, 2009. [Online]. Available: http://link.aip.org/link/PSISDG/v7258/i1/p72582B/s1&Agg=doi 19

[52] A. Weinlich, B. Keck, H. Scherl, M. Kowarschik, and J. Hornegger, *Comparison of High-speed ray casting on GPU using CUDA and OpenGL*. KIT Scientific Publishing,

2008, vol. 1, p. 25. [Online]. Available: http://books.google.com/books?hl=en&lr=&id=1dihoTBSy3QC&oi=fnd&pg=PA25&dq=Comparison+of+High-Speed+Ray+Casting+on+GPU+using+CUDA+and+OpenGL&ots=CaCfOwEX2H&sig=9vLUyMgTPJWjRzOR3dTKIJFgOxg 19

[53] J. M. Elble, N. V. Sahinidis, and P. Vouzis, "Gpu computing with kaczmarz's and other iterative algorithms for linear systems." *Parallel Computing*, vol. 36, no. 5-6, pp. 215–231, 2010. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2879082&tool=pmcentrez&rendertype=abstract 19, 20, 49

[54] R. Bramley and A. Sameh, "Row projection methods for large nonsymmetric linear systems," *SIAM J. Scientific and Statistical Computing*, vol. 13, pp. 168–193, 1992. 20

[55] W. Xu and K. Mueller, "Accelerating regularized iterative ct reconstruction on commodity graphics hardware (gpu)," in *Biomedical Imaging: From Nano to Macro, 2009. ISBI '09. IEEE International Symposium on*, 2009, pp. 1287–1290. 20

[56] W.-M. Pang, J. Qin, Y. Lu, Y. Xie, C.-K. Chui, and P.-A. Heng, "Accelerating simultaneous algebraic reconstruction technique with motion compensation using cuda-enabled gpu," *International Journal of Computer Assisted Radiology and Surgery*, vol. 6, no. 2, pp. 187–199, 2010. [Online]. Available: http://dx.doi.org/10.1007/s11548-010-0499-3 20

[57] S. Q. Zheng, E. Branlund, B. Kesthelyi, M. B. Braunfeld, Y. Cheng, J. W. Sedat, and D. A. Agard, "A distributed multi-gpu system for high speed electron microscopic tomographic reconstruction." *Ultramicroscopy*, vol. 111, no. 8, pp. 1137–1143, 2011. [Online]. Available: http://linkinghub.elsevier.com/retrieve/pii/S0304399111001227 20

[58] A. Buzmakov, D. Nikolaev, M. Chukalina, and G. Schaefer, "Efficient and effective regularised art for computed tomography," in *Engineering in Medicine and Biology Society,EMBC, 2011 Annual International Conference of the IEEE*, 2011, pp. 6200–6203. 20, 21

[59] M. Chukalina, D. Nikolaev, and A. Simionovici, "Art in x-ray tomography: Image noise reduction," in *21th European Conference on Modelling and Simulation*, 2007, pp. 309–312. 20

[60] Nvidia, "Nvidia cuda programming guide version 2.3.1," p. 31, 2009. 21

[61] W. J. Palenstijn, K. J. Batenburg, and J. Sijbers, "Performance improvements for iterative electron tomography reconstruction using graphics processing units (gpus)." *Journal of Structural Biology*, vol. 176, no. 2, pp. 250–3, 2011. [Online]. Available: http://www.ncbi.nlm.nih.gov/pubmed/21840398 21

[62] W. Xu, F. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller, "High-performance iterative electron tomography reconstruction with long-object compensation using graphics processing units (gpus)." *Journal of Structural Biology*, vol. 171, no. 2, pp. 142–153, 2010. [Online]. Available: http://www.pubmedcentral.nih.gov/articlerender.fcgi?artid=2885506&tool=pmcentrez&rendertype=abstract 21

[63] J. I. Agulleiro, F. Vázquez, E. M. Garzón, and J. J. Fernández, "Hybrid computing: Cpu+gpu co-processing and its application to tomographic reconstruction," *Ultramicroscopy*, vol. 115, pp. 109–114, 2012. [Online]. Available: http://www.scopus.com/inward/record.url?eid=2-s2.0-84859853592&partnerID=40&md5=37d460156c136496b9d19732140f1005 21, 22

[64] F. Vázquez, E. Garzón, and J. Fernández, "A matrix approach to tomographic reconstruction and its implementation on {GPUs}," *Journal of Structural Biology*, vol. 170, no. 1, pp. 146 – 151, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S104784771000033X 22

[65] J. Cong, L. A. Vese, J. Villasenor, M. Yan, and Y. Zou, "A hybrid architecture for compressive sensing 3-d ct reconstruction," *Emerging and Selected Topics in Circuits and Systems IEEE Journal on*, vol. 2, no. 3, pp. 616–625, 2012. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6338301 22

[66] J. Reinders, *Intel threading building blocks - outfitting C++ for multi-core processor parallelism*. O'Reilly, 2007. [Online]. Available: http://www.oreilly.com/catalog/9780596514808/index.html 23

[67] P. Wang. (2008) Compare windows* threads, openmp*, intel® threading building blocks for parallel programming. Accessed: 15/12/2012. [Online]. Available: http://software.intel.com/en-us/blogs/2008/12/16/compare-windows-threads-openmp-intel-threading-building-blocks-for-parallel-programming/ 23

[68] R. Friedman. (2012, Oct.) About the openmp arb and openmp.org. Accessed: 15/12/2012. [Online]. Available: http://openmp.org/wp/about-openmp/ 23

[69] OpenMP Architecture Review Board, "Openmp application program interface," Specification, 2011. [Online]. Available: http://www.openmp.org/mp-documents/OpenMP3.1.pdf 23, 34

[70] O. ARB. (2012) Openmp compilers. Accessed: 18/12/2012. [Online]. Available: http://openmp.org/wp/openmp-compilers/ 23

[71] Intel. Why use intel® tbb? Accessed: 15/12/2012. [Online]. Available: http://threadingbuildingblocks.org/ 24

[72] ——. Intel® threading building blocks benefits. Accessed: 15/12/2012. [Online]. Available: http://software.intel.com/sites/products/documentation/doclib/tbb_sa/help/index.htm 24

[73] Nvidia, "Nvidia cuda c programming guide," *Changes*, no. 350, p. 175, 2012. 25, 26, 38

[74] ——. Cuda toolkit. Accessed: 16/09/2013. [Online]. Available: https://developer.nvidia.com/cuda-toolkit 25

[75] ——, "Cublas library user guide v5.0," p. 104, 2012. 26, 29

[76] B. Overland, *C++ Without Fear: A Beginner's Guide That Makes You Feel Smart*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2011. 27

[77] S. Amaya. Information on the c++ language: A brief description. Accessed: 15/12/2012. [Online]. Available: http://www.cplusplus.com/info/description/ 27

[78] G. Guennebaud, B. Jacob *et al.* (2011) Eigen v3. Accessed: 15/12/2012. [Online]. Available: http://eigen.tuxfamily.org 28, 31

[79] ——. (2011) Some important changes between eigen 2 and eigen 3. Accessed: 15/12/2012. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=3.0 28

[80] ——. (2011) Column-major and row-major storage. Accessed: 15/12/2012. [Online]. Available: http://eigen.tuxfamily.org/dox/TopicStorageOrders.html 28

[81] Nvidia. Cublas. Accessed: 16/09/2013. [Online]. Available: https://developer.nvidia.com/cublas 29

[82] Intel. Intel math kernel library. Accessed: 17/09/2013. [Online]. Available: http://software.intel.com/en-us/intel-mkl 29

[83] B. Team. (2005) Blas frequently asked questions (faq). Accessed: 17/09/2013. [Online]. Available: http://netlib.org/blas/faq.html 30

[84] L. S. Blackford, J. Demmel, J. Dongarra, I. Duff, S. Hammarling, G. Henry, M. Heroux, L. Kaufman, A. Lumsdaine, A. Petitet, R. Pozo, K. Remington, and R. C. Whaley, "An updated set of Basic Linear Algebra Subprograms (BLAS)," *ACM Transactions on Mathematical Software*, vol. 28, no. 2, pp. 135–151, Jun. 2002. 30

[85] L. Team. Lapack linear algebra package. Accessed: 17/09/2013. [Online]. Available: http://netlib.org/lapack/ 30

[86] S. Team. (2012) Scalapack scalable linear algebra package. Accessed: 17/09/2013. [Online]. Available: http://www.netlib.org/scalapack/ 30, 31

[87] J. Choi, J. Demmel, I. Dhillon, J. Dongarra, S. Ostrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, "ScaLAPACK: A portable linear algebra library for distributed memory computers — design issues and performance," *Lecture Notes in Computer Science*, vol. 1041, p. 95, 1996. 31

[88] S. Nikolaev. (2013) Cvm class library. Accessed: 18/09/2013. [Online]. Available: http://cvmlib.com 31

[89] Intel. (2012) Intel(r) threading building blocks - release notes. Accessed: 15/12/2012. [Online]. Available: http://threadingbuildingblocks.org/sites/default/files/resources/tbb-release-notes-4-1.txt 31

[90] A. V. Aho and J. D. Ullman, *Principles of Compiler Design*. Reading, MA: Addison-Wesley, 1979. 33

[91] W. Petersen, *Introduction to parallel computing : [a practical guide with examples in C*. Oxford New York: Oxford University Press, 2004. 33

[92] V. C. V. Rao, "Tutorial notes on optimizing performance of parallel programs," Presentation on 5th International Conference and Exhibition on High-Performance Computing in the Asia-Pacific Region, June 2002. 33

[93] MSDN. Visual studio omp library reference. Accessed: 15/12/2012. [Online]. Available: http://msdn.microsoft.com/en-us/library/yw6c0z19(v=vs.80).aspx 34

[94] G. Guennebaud, B. Jacob *et al.* (2011) How does eigen compare to blas/lapack? Accessed: 15/12/2012. [Online]. Available: http://eigen.tuxfamily.org/index.php?title=FAQ 37

[95] V. Pistulkar and C. Uttarwar, "Analyzing the cuda applications with its latency and bandwidth tolerance," 2012. 49

[96] C. Rizos, M. B. Higgins, and S. Hewitson, "NEW GNSS DEVELOPMENTS AND THEIR IMPACT ON SURVEY SERVICE PROVIDERS AND SURVEYORS," in *Sequences, Subsequences, and Consequences*. 51

[97] E. D. Kaplan, *Understanding GPS-Principles and Applications*. Boston: Artech House Publisher, 1996. 51

[98] J. van Baelen, J.-P. Aubagnac, and A. Dabas, "Comparison of Near Real Time Estimates of Integrated Water Vapor Derived with GPS, Radiosondes, and Microwave Radiometer," *Journal of Atmospheric and Oceanic Technology*, vol. 22, 2005. 51

[99] G. Guerova, "Application of gps derived water vapour for numerical weather prediction in switzerland," Ph.D. dissertation, University of Bern, 2003. 51, 52, 53

[100] J. Saastamoinen, "Atmospheric correction for the troposphere and stratosphere in radio ranging of satellites, in the use of artificial Satellites for geodesy," *Geophysical Monograph 15*, vol. 16, pp. 247–251, 1972. 53

[101] J. Askne and H. Nordius, "Estimation of tropospheric delay for microwaves from surface weather data," *Radio Science*, vol. 22, pp. 379–386, 1987. 53

[102] *GPS Satellite Surveying*. Wiley, 2004. 54

[103] Y. M. Bi, J. T. Mao, X. Y. Liu, Y. Fu, and C. C. Li, "Remote sensing of the amount of water vapor along the slant path using the ground-base gps," *Chinese Journal of GeophysicsChinese Edition*, vol. 49, no. 2, pp. 335–342, 2006. 55

[104] Y. E. Bar-Sever, P. M. Kroger, and J. A. Borjesson, "Estimating horizontal gradients of tropospheric path delay with a single gps receiver," *Journal of Geophysical Research*, vol. 103, no. B3, pp. 5019–5035, 1998. [Online]. Available: http://www.agu.org/pubs/crossref/1998/97JB03534.shtml 55

[105] D. D. McCarthy and G. Petit, "IERS Conventions (2003)," 2004. 55

[106] G. Chen and T. A. Herring, "Effects of atmospheric azimuthal asymmetry on the analysis of space geodetic data," *Journal of Geophysical Research*, vol. 102, no. B9, pp. 20 489–20 502, 1997. [Online]. Available: http://www.agu.org/pubs/crossref/1997/97JB01739.shtml 55

[107] L. X.-Y. F. Y. L. C.-C. BI Yan-Meng, MAO Jie-Tai, "Remote sensing of atmospheric integrated water vapor along slant paths using ground based gps," *Chinese Journal of Geophysics*, 2006. 55

[108] A. F. Jiménez, "Atmospheric tomography using satellite radio signals," Mar. 10 2003. [Online]. Available: http://www.tdx.cesca.es/TDX-0306103-162444/ 56

[109] L. Gradinarsky, "Sensing atmospheric water vapor using radio waves," 2002. 56

[110] U. Foelsche and G. Kirchengast, "Tropospheric water vapor imaging by combination of ground-based and spaceborne gnss sounding data," *Journal of Geophysical Research*, vol. 106, no. D21, pp. 27 221–27 231, 2001. [Online]. Available: http://www.uni-graz.at/igam7www_ufandgk-jgr-v106p27221y2001.pdf 57

[111] MathWorks. (2013) Matlab compiler. Accessed: 10/09/2013. [Online]. Available: http://www.mathworks.com/products/compiler/ 59

[112] B. Community. (2013) Boost c++ libraries. Accessed: 11/09/2013. [Online]. Available: http://www.boost.org 59

[113] P. Spofford and B. Remondi. The national geodetic survey standard gps format sp3. Accessed: 12/09/2013. [Online]. Available: http://igscb.jpl.nasa.gov/igscb/data/format/sp3_docu.txt 61

[114] M. Schenewerk, "A brief review of basic gps orbit interpolation strategies," *GPS Solutions*, vol. 6, no. 4, pp. 265–267, 2003. [Online]. Available: http://dx.doi.org/10.1007/s10291-002-0036-0 61, 62

[115] E. Ziegel, W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, *Numerical Recipes: The Art of Scientific Computing*, W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, Eds. Cambridge University Press, 1987, vol. 29, no. 4. [Online]. Available: http://www.jstor.org/stable/1269484?origin=crossref 62

[116] M. Horemu? and J. Andersson, "Polynomial interpolation of gps satellite coordinates," *GPS Solutions*, vol. 10, no. 1, pp. 67–72, 2006. [Online]. Available: http://dx.doi.org/10.1007/s10291-005-0018-0 62

[117] W. Gurtner. (2007) Rinex: The receiver independent exchange format version 2.10. Accessed: 14/09/2013. [Online]. Available: ftp://igscb.jpl.nasa.gov/pub/data/format/rinex210.txt 62

[118] I. E. Rotation and Reference. (1997)) Solution (software/technique) independent exchange format for combination of tropospheric estimates. Accessed: 15/09/2013. [Online]. Available: http://igscb.jpl.nasa.gov/igscb/data/format/sinex_tropo.txt 62

[119] J. Feltens, "Vector methods to compute azimuth, elevation, ellipsoidal normal, and the cartesian (x, y, z) to geodetic (?, ?, h) transformation," *Journal of Geodesy*, vol. 82, no. 8, pp. 493–504, 2008. [Online]. Available: http://dx.doi.org/10.1007/s00190-007-0198-1 63

[120] R. F. Boisvert, R. Pozo, and K. Remington, "The Matrix Market exchange formats: Initial design," National Institute of Standards and Technology, Gaithersburg, MD, USA, Tech. Rep. NISTIR 5935, Dec. 1996. 65

[121] C. Champollion, "Quantification de la vapeur d'eau troposphérique par gps (modèles 2d et tomographies 3d) - application aux précipitations intenses," Ph.D. dissertation, Université Montpellier II, 2005. 67, 77

[122] D. J. Seidel, "Water Vapor: Distribution and Trends," 2002. 71, 72

[123] D. K. Adams, R. M. S. Fernandes, E. R. Kursinski, J. M. Maia, L. F. Sapucci, L. A. T. Machado, I. Vitorello, J. F. G. Monico, K. L. Holub, S. I. Gutman, N. Filizola, and R. A. Bennett, "A dense gnss meteorological network for observing deep convection in the amazon," *Atmospheric Science Letters*, vol. 12, no. 2, pp. 207–212, 2011. [Online]. Available: http://dx.doi.org/10.1002/asl.312 71

[124] J. Boehm, B. Werl, and H. Schuh, "Troposphere mapping functions for gps and very long baseline interferometry from european centre for medium-range weather forecasts operational analysis data," *Journal of Geophysical Research: Solid Earth*, vol. 111, no. B2, pp. n/a–n/a, 2006. [Online]. Available: http://dx.doi.org/10.1029/2005JB003629 78

[125] J. Boehm, A. Niell, P. Tregoning, and H. Schuh, "Global mapping function (gmf): A new empirical mapping function based on numerical weather model data," *Geophysical Research Letters*, vol. 33, no. 7, pp. n/a–n/a, 2006. [Online]. Available: http://dx.doi.org/10.1029/2005GL025546 78

Appendix A

SWART Files

A.0.1: Example SWART Configuration File

```
# Segal GNSS Water Vapour Reconstruction Image Software (SWART)
# Configuration File
#
# Comments start with '#'

# If you have any doubt in parameters you can try run that program in console to see the description
# of each parameter. When the file isn't in the same directory than the executables, please provide
# the full path (absolute path) to the file
# When providing the full path to a file, please put it inside quotes ex.: "C:\myMatrixFile.dat"
# Next follows the parameters for each of the SWART package programs.


########################################
# SlantDelayJoinProcessing.exe Parameters
########################################
configurationFile inputBelemModPrecipitation.txt
########################################



########################################
# GridRayIntersection.exe Parameters
########################################
gridfile gridBelem.dat
slantsfile SlantsJoined.sync
cutoff 10
gengrid gridBelemTinyDenseParams.txt
########################################



########################################
# AlgebraicAlgorithms.exe Parameters
########################################
matrixfile outputFileM.txt
vectorfile outputFileV.txt
# Possible options: kaczmarz, kaczmarzSymetric, landweber, cimmino, cav, drop, sart
algorithm sart
iterations 50000
########################################



########################################
# PlotWaterVapour.exe Parameters
########################################
# Since this' a matlab program, please always provide the absolute patch for the files, NOT the relative
#
vectorFile "C:\WaterVapourReconstructionProject\Release\reconsImage.dat"
gridFile "C:\WaterVapourReconstructionProject\Release\gridBelem.dat"
slice 5
# Possible options: lat-lon, lat-height, lon-height
sliceType lat-height
#contour 15
#limitTo5000 true
########################################
```

A.0.2: Example Output file of SlantDelayJoinProcessing Component (the actual files have more precision than this example here the precision was reduced to fit in the document)

Year	Day	Seconds	Slant WV	Satellite	SAT_X	SAT_Y	SAT_Z	Receiver	REC_X	REC_Y	REC_Z	elAngle(rad)	elAngle(deg)	Azimuth(rad)	Azimuth(deg)
2011	157	41400	187254	G25	556	-15136.303	-21838.704	BABT	4199.5	-4796.73	-187.3128	0.25037355	14.345347	0.41836624	23.97062
2011	157	41400	86834.865	G5	26282	-1814.3107	3779.3100	BABT	4199.5	-4796.73	-187.3128	0.56826608	32.55924	1.342027	76.892519
2011	157	41400	126184.90	G21	-601.7	-21207.457	16015.390	BABT	4199.5	-4796.73	-187.3128	0.37850179	21.686555	5.5638297	318.7839
2011	157	41400	82095.502	G12	14602	-11880.887	-18644.717	BABT	4199.5	-4796.73	-187.3128	0.60572751	34.70562	6.1102205	350.0898
2011	157	41400	105398.99	G29	-2356.	-23691.649	-11746.187	BABT	4199.5	-4796.73	-187.3128	0.45890343	26.293229	0.99577691	57.0538

A.0.3: Example SlantDelayJoinProcessing Configuration File

```
# Comments start with '#'
SP3FILE igs16391.sp3
STARTSECONDS 41400
ENDSECONDS 43200
SP3INTERPOLATION false

# BABT - 1
RECEIVER BABT
RINEXFILE BELEM_DAY_157_ModPrec\BABT\BABTnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BABT\BABT2011157.TRO

# BAGB - 2
RECEIVER BAGB
RINEXFILE BELEM_DAY_157_ModPrec\BAGB\BAGBnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BAGB\BAGB2011157.TRO

# BBNV - 3
RECEIVER BBNV
RINEXFILE BELEM_DAY_157_ModPrec\BBNV\BBNVnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BBNV\BBNV2011157.TRO

# BEMA - 4
RECEIVER BEMA
RINEXFILE BELEM_DAY_157_ModPrec\BEMA\BEMAnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BEMA\BEMA2011157.TRO

# BJRN - 5
#RECEIVER BJRN
#RINEXFILE BELEM_DAY_157_ModPrec\BJRN\BJRNnavRinexFile.txt
#VARSFILE meteoBelem157.txt
#ZTDFILE BELEM_DAY_157_ModPrec\BJRN\BJRN2011157.TRO

# BMGR - 6
RECEIVER BMGR
RINEXFILE BELEM_DAY_157_ModPrec\BMGR\BMGRnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BMGR\BMGR2011157.TRO

# BMSQ - 7
RECEIVER BMSQ
RINEXFILE BELEM_DAY_157_ModPrec\BMSQ\BMSQnavRinexFile.txt
VARSFILE meteoBelem157.txt
ZTDFILE BELEM_DAY_157_ModPrec\BMSQ\BMSQ2011157.TRO

# BOGT - 8
#RECEIVER BOGT
#RINEXFILE BELEM_DAY_157_ModPrec\BOGT\BOGTnavRinexFile.txt
#VARSFILE meteoBelem157.txt
#ZTDFILE BELEM_DAY_157_ModPrec\BOGT\BOGT2011157.TRO
```

A.0.4: Example of SP3 File (igs16391.sp3)

```
#cP2011  6  6  0  0  0.00000000      96 ORBIT IGS08 HLM  IGS
## 1639  86400.00000000   900.00000000 55718 0.0000000000000
+   32   G01G02G03G04G05G06G07G08G09G10G11G12G13G14G15G16G17
+        G18G19G20G21G22G23G24G25G26G27G28G29G30G31G32  0  0
+          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
+          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
+          0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
++         0  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2  2
++         2  2  2  2  2  2  2  2  2  2  2  2  3  2  2  0  0
++         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
++         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
++         0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
%c G  cc GPS ccc cccc cccc cccc cccc ccccc ccccc ccccc ccccc
%c cc cc ccc ccc cccc cccc cccc cccc ccccc ccccc ccccc ccccc
%f  1.2500000  1.025000000  0.00000000000  0.000000000000000
%f  0.0000000  0.000000000  0.00000000000  0.000000000000000
%i    0    0    0    0      0      0      0      0         0
%i    0    0    0    0      0      0      0      0         0
/* FINAL ORBIT COMBINATION FROM WEIGHTED AVERAGE OF:
/* cod emr esa gfz grg jpl mit ngs sio
/* REFERENCED TO IGS TIME (IGST) AND TO WEIGHTED MEAN POLE:
/* PCV:IGS08_1639 OL/AL:FES2004  NONE     Y  ORB:CMB CLK:CMB
*  2011  6  6  0  0  0.00000000
PG01 -17378.692000  13943.822741 -14786.495966 999999.999999
PG02 -17009.399032   1925.701331 -20137.006261    340.225624  7  9  9 140
PG03  20571.803406   5057.114510  15788.568374    724.722841  5  7  8 108
PG04 -11316.611313 -12064.363483 -20931.144153    135.981375  7  8  7 109
PG05 -26544.308834   1319.882076  -1491.306992   -170.525153  9  8 11 109
PG06  20649.665267   9600.188004  14015.950932    269.222723  5  8  8 104
PG07   4106.741275 -24188.049267   9894.412086     13.378564  9  9  7 141
PG08  -2513.410361 -19094.095018  18049.597020      8.053171 11 10  8 137
PG09 -14652.829518  20602.704472   7583.446210     84.547790  8  5 10 104
PG10 -19530.942168  -9855.824294 -15529.682922     -2.796441  9  6 10 127
PG11  13110.713280 -19654.254485  11473.961937   -157.045654  8  8  7 127
PG12 -18248.752769  11639.515063 -15253.671884      0.477921  9  8  5 113
PG13  -1928.210839 -22200.667954 -14545.953825    266.783561 10  6  8 126
PG14  14960.575737  21958.312017  -1753.942447    154.319895  8  9  9 153
PG15 -12529.960849   9047.103039  21605.157990   -123.399628  8  7  8 127
PG16  26659.329193    -98.263492  -1929.248749   -177.068618  7  7  7 151
PG17 -15611.553733 -21559.596108  -1730.239185    184.051463 10  9  9 136
PG18   3727.718469  15669.705629  21462.417674    140.727776  8  7  7 126
PG19  14880.288393  -4527.106625  21556.614555   -148.277036  7 10  6 131
PG20  14951.784010 -14359.095200 -16715.244048     49.714365  9  7  8 126
PG21  -1292.866790  23524.289909  12058.077336   -149.223352  9  7  8 129
PG22  16310.743164  11927.211193  17466.333379    147.848567  8  7  8 137
PG23   7372.080763 -14730.239211 -20758.904752    301.288620  9  5  6 135
PG24  15090.890795 -21174.356556   4654.752281    396.120036  7  5  7 134
PG25  -5141.158235  15253.885301 -21142.673649      2.527819 10  6  6 152
PG26 -17472.491135  -4985.173077  18924.491198    -51.797804  7  9  7 142
PG27 -16619.636112  15486.572889  14131.392647    272.192683  6  6  8 137
PG28 -12551.639880 -13036.032208  20046.942554     42.764766  8  6  7 141
PG29    574.533966  21278.277352 -15838.844623    213.049350  8  5  6 137
PG30  23704.202470   4918.769438 -11369.416345      7.742884  9 11 16 153
PG31  11774.418094   9203.348618 -21740.062425     91.152922  9 10  7 135
PG32  22184.613692  -8103.673206 -11444.168104   -267.785478 10  9 10 143
*  2011  6  6  0 15  0.00000000
PG01 -18900.672345  14076.899278 -12569.382878 999999.999999
PG02 -16212.352864   -315.637228 -20911.977020    340.227526  7  9  8 132
PG03  21626.021266   6483.103627  13815.011598    724.726652  4  6  8  66
PG04 -10305.982457 -14155.249852 -20087.999904    135.991070  7  7  7 107
PG05 -26249.669661   1018.669496  -4302.804493   -170.530178  8  8 11 131
PG06  21397.191147  10867.177062  11786.369219    269.145511  4  7  8 103
```

A.0.5: Example of Rinex Navigation File

```
     2.10           OBSERVATION DATA    G (GPS)             RINEX VERSION / TYPE
teqc  2006Dec12                         20130422 13:56:53UTCPGM / RUN BY / DATE
MSXP|IAx86-PII|bcc32 5.0|MSWin95->XP|486/DX+                COMMENT
   0.000                                                    COMMENT
  -1.69424406                                               COMMENT
 -48.79759576                                               COMMENT
-011.500                                                    COMMENT
BIT 2 OF LLI FLAGS DATA COLLECTED UNDER A/S CONDITION       COMMENT
BABT                                                        MARKER NAME
BABT                                                        MARKER NUMBER
-Unknown-           -Unknown-                               OBSERVER / AGENCY
4906K34406          TRIMBLE             NP 4.17 / SP 0.00   REC # / TYPE / VERS
38353437                                                    ANT # / TYPE
  4199581.4390 -4796736.4534  -187312.4433                  APPROX POSITION XYZ
        0.1100        0.0000        0.0000                  ANTENNA: DELTA H/E/N
     1     1                                                WAVELENGTH FACT L1/2
     6    L1    L2    C1    P2    P1    D1                  # / TYPES OF OBSERV
 SNR is mapped to RINEX snr flag value [1-9]                COMMENT
  L1: 3 -> 1; 8 -> 5; 40 -> 9                               COMMENT
  L2: 1 -> 1; 5 -> 5; 60 -> 9                               COMMENT
  2011     6     6     0     0    0.0000000     GPS         TIME OF FIRST OBS
                                                            END OF HEADER
 11  6  6  0  0  0.0000000  0 10G 8G23G11G30G24G16G 7G19G13G20
 126770653.23149  98782392.25546  24123659.1884   24123667.7504

 120725477.49049  94071890.11747  22973274.3594   22973280.3954

 109993288.64249  85709184.36947  20931010.7584   20931017.1094

 128694231.57649 100281341.83847  24489689.3054   24489698.0354

 105820336.35649  82457513.57648  20136921.9924   20136929.4344

 121207499.32249  94447536.30147  23065010.4064   23065016.1804

 114832021.032 9  89479428.186 8  21851765.313

 127547422.49049  99387712.06146  24271451.6024   24271457.1804

 122441818.13849  95409330.56747  23299890.2504   23299898.2464

 115080458.78749  89673173.96448  21899070.7974   21899077.3714

 11  6  6  0  0  5.0000000  0 10G 8G23G11G30G24G16G 7G19G13G20
 126753323.77049  98768888.83146  24120361.7504   24120369.8444
      3465.8924
 120730163.81049  94075541.79647  22974166.9454   22974172.4144
      -937.2644
 110002116.06249  85716062.88347  20932691.0234   20932697.0864
     -1765.4844
 128702986.01049 100288163.47047  24491354.9924   24491364.2664
     -1750.8874
 105825603.90949  82461618.16748  20137924.4844   20137931.2424
     -1053.5114
 121208991.67549  94448699.17947  23065294.2194   23065300.0984
      -298.4714
```

A.0.6: Example of Meteorological File

```
Year J.Day   Mon Day  h   m Press   TEMP RHumidity Precip WindSpeed  WindDir  PW    CTT
2011 1.0000   1   1   0   0 995.90  25.80  81.10   0.00   1.10      101.00 -999.99  259.44
2011 1.0007   1   1   0   1 995.90  25.80  80.70   0.00   1.80      105.00 -999.99 -999.99
2011 1.0014   1   1   0   2 995.90  25.90  81.00   0.00   0.30      136.00 -999.99 -999.99
2011 1.0021   1   1   0   3 996.00  25.80  80.90   0.00   0.90      120.00 -999.99 -999.99
2011 1.0028   1   1   0   4 996.00  25.80  80.80   0.00   1.00       96.00 -999.99 -999.99
2011 1.0035   1   1   0   5 996.00  25.80  81.00   0.00   0.90       94.00 -999.99 -999.99
2011 1.0042   1   1   0   6 996.00  25.80  81.30   0.00   1.30      111.00 -999.99 -999.99
2011 1.0049   1   1   0   7 996.10  25.80  82.00   0.00   0.10      156.00 -999.99 -999.99
2011 1.0056   1   1   0   8 996.10  25.80  81.40   0.00   1.50      110.00 -999.99 -999.99
2011 1.0063   1   1   0   9 996.20  25.80  81.30   0.00   0.80      111.00 -999.99 -999.99
2011 1.0069   1   1   0  10 996.20  25.70  81.20   0.00   2.20      111.00 -999.99 -999.99
2011 1.0076   1   1   0  11 996.20  25.70  81.50   0.00   0.60      140.00 -999.99 -999.99
2011 1.0083   1   1   0  12 996.20  25.70  81.20   0.00   1.20      118.00 -999.99 -999.99
2011 1.0090   1   1   0  13 996.20  25.70  82.10   0.00   0.20      336.00 -999.99 -999.99
2011 1.0097   1   1   0  14 996.20  25.70  81.70   0.00   1.80      103.00 -999.99 -999.99
```

A.0.7: Example of SINEX File

```
%=TRO 0.00 DUT 13:104:02862 DUT 11:157:00000 11:157:86399 P MIX
*
+TROP/DESCRIPTION
*_________KEYWORD_____________ __VALUE(S)_____________________________________
 SAMPLING INTERVAL                              300
 SAMPLING TROP                                  300
 TROP MAPPING FUNCTION          GMF
 ELEVATION CUTOFF ANGLE                          08
 SOLUTION_FIELDS_1              TROTOT STDDEV
-TROP/DESCRIPTION
*
+TROP/STA_COORDINATES
*SITE PT SOLN T __STA_X_____ __STA_Y_____ __STA_Z_____ SYSTEM REMRK
 BABT  A    1 P  4199580.417 -4796735.504  -187312.815  IGS00 DEOS
-TROP/STA_COORDINATES
*
+TROP/SOLUTION
*SITE ___EPOCH____ __TROTOT  STD ______SINGRAD _____STD ______COSGRAD _____STD
 BABT 11:156:75600  2557.69  1.8 -8.051147E-07 3.01E-07  5.434167E-07 2.19E-07
 BABT 11:156:75900  2557.69  1.8 -8.030134E-07 2.95E-07  5.513513E-07 2.13E-07
 BABT 11:156:76200  2557.84  1.6 -7.983172E-07 2.83E-07  5.794047E-07 2.01E-07
 BABT 11:156:76500  2558.10  1.6 -8.132069E-07 2.74E-07  5.798400E-07 1.95E-07
 BABT 11:156:76800  2558.22  1.6 -8.172714E-07 2.67E-07  5.672646E-07 1.90E-07
 BABT 11:156:77100  2558.16  1.5 -8.081007E-07 2.62E-07  5.504190E-07 1.87E-07
 BABT 11:156:77400  2558.20  1.5 -7.968851E-07 2.56E-07  5.382312E-07 1.84E-07
 BABT 11:156:77700  2558.46  1.5 -8.023873E-07 2.52E-07  5.277810E-07 1.81E-07
 BABT 11:156:78000  2558.93  1.4 -8.123833E-07 2.47E-07  5.083568E-07 1.77E-07
 BABT 11:156:78300  2559.16  1.4 -8.073813E-07 2.42E-07  4.893996E-07 1.73E-07
 BABT 11:156:78600  2559.05  1.4 -7.821076E-07 2.39E-07  4.727531E-07 1.71E-07
 BABT 11:156:78900  2559.27  1.3 -7.606168E-07 2.36E-07  4.232840E-07 1.69E-07
 BABT 11:156:79200  2559.35  1.3 -7.170313E-07 2.34E-07  3.708509E-07 1.68E-07
 BABT 11:156:79500  2559.23  1.3 -6.388179E-07 2.32E-07  3.545411E-07 1.67E-07
 BABT 11:156:79800  2559.44  1.3 -5.653471E-07 2.29E-07  3.213962E-07 1.65E-07
 BABT 11:156:80100  2559.64  1.3 -4.931071E-07 2.27E-07  2.951329E-07 1.63E-07
 BABT 11:156:80400  2560.03  1.3 -4.292869E-07 2.24E-07  2.500323E-07 1.62E-07
 BABT 11:156:80700  2560.68  1.3 -3.659333E-07 2.20E-07  2.084826E-07 1.60E-07
 BABT 11:156:81000  2561.83  1.3 -2.970185E-07 2.17E-07  1.960727E-07 1.58E-07
 BABT 11:156:81300  2562.78  1.3 -2.411207E-07 2.13E-07  1.547433E-07 1.57E-07
 BABT 11:156:81600  2562.74  1.3 -2.337512E-07 2.10E-07  3.413743E-08 1.57E-07
 BABT 11:156:81900  2562.80  1.3 -2.253639E-07 2.07E-07 -8.871019E-08 1.56E-07
 BABT 11:156:82200  2563.38  1.3 -2.652816E-07 2.05E-07 -1.549358E-07 1.55E-07
 BABT 11:156:82500  2563.61  1.3 -2.897200E-07 2.04E-07 -2.344732E-07 1.56E-07
```