CRANFIELD UNIVERSITY

Andre Montenegro Ferreira

Parallel Prediction of Radio Propagation

SCHOOL OF ENGINEERING

MSc THESIS

CRANFIELD UNIVERSITY
School of Engineering

MSc THESIS
ACADEMIC YEAR 2010-2011

Andre Montenegro Ferreira

Parallel Prediction of Radio Propagation

Supervisor: Simon Wilcox

August 2011

This thesis is submitted in partial fulfilment of the requirements for the degree of Master of Science

Acknowledgements

I would like to thank Simon Wilcox and Dr. Stuart Barnes for their constant advice and prompt availability to have a conversation.

Special thanks go to my colleague Jacek Makura, for explaining me in minutes what I would have to “figure out” in hours, and to Michal Czapinski, whose expertise in *CUDA* helped me to unveil inconspicuous problems. I must also acknowledge Javier Salmeron for introducing me to a great LaTeX *IDE*.

I would also like to express my gratitude to Dr. Rosaldo Rossetti and Dr. A. Augusto Sousa, who helped me to achieve this opportunity in Cranfield University.

Abstract

The aim of this MSc thesis was to develop a parallel program to predict radio signal losses, in a given area.

The program used the Irregular Terrain with Obstructions Model (*ITWOM*), in order to obtain the loss prediction for each of the points around a fixed transmitter. The *ITWOM* is a correction, published in 2010, of the most widely used algorithm for radio signal prediction: the Longley-Rice Model.

The implementation encompassed the following stages: extraction of terrain elevation data, computation of the radio losses, and display of the graphical output.

A serial program was first created, and then adapted to be run in parallel, using a multi-core processor with *Hyper-Threading*. With *OpenMP*, it was possible to obtain performance improvements up to 500%.

Parallel possibilities were also exploited for *CUDA*, the technology provided by Nvidia to support general purpose computation on their graphic cards. Memory restrictions and other obstacles imposed by the *GPU*, which prevented a successful adaptation, are documented in this report.

Contents

List of Figures

List of Tables

Abbreviations

CUDA	Compute Unified Device Architecure
GPGPU	General Purpose computing on Graphics Processing Units
GNU	General Public License
GTD	Geometrical Theory of Diffraction
ITM	Irregular Terrain Model
ITS	Institute for Telecommunication Sciences
ITU	International Telecommunications Union
ITWOM	Irregular Terrain with Obstructions Model
NTIA	National Telecommunications and Information Administration
SIMD	Single Instruction Multiple Data
SRTM	Shuttle Radar Topography Mission
UHF	Ultra High Frequencies
VHF	Very High Frequencies

Chapter 1

Introduction

Radio waves are highly dependent on the physical media in which they travel. Many phenomena such as reflection, diffraction, scattering and absorption are originated by the characteristics of the atmosphere and of the geosphere. The understanding of these effects needs to be considered in the design and planning of radio communications systems. Such systems are found in military operations, television and radio broadcasting, amateur radio, wireless networks and many other fields that use communication links based on radio waves.

Prediction of radio propagation focuses on forecasting the signal strength of a receiver within a communication link. This is used in order to allow the designer to confirm in advance if his system will operate correctly. As the name indicates, there is no guarantee of a predicted value to match the value in reality. In fact, the exact result of the signal received can only be obtained by on-site experience on the side of the receiver. However, the option of performing on-site experiences is inconceivable for many cases as it would be too expensive in terms of time and money. Therefore, a prediction model becomes absolutely necessary when, given a certain area, the communications engineer wants to prognosticate the signal loss between many combinations of two locations (transmitter and receiver). Based on the results of these predictions, the engineer is able to infer how efficiently and under which constraints his communications would work on that same area. In the example of the military operations, the strategic locations to place the transmitters and receivers are decided in advance, in order to achieve a good communication system.

A propagation model returns the signal received as a function of the signal radiated, the distance, and other variables between transmitter and receiver. These models are often composed by deterministic functions based on electro-magnetic theory as well as empirical formulations and, in some cases, statistical methods. The empirical and statistical parts are very important in propagation modelling due to the innumerable variables that affect the radio waves. These variables can be the forestation and buildings in the area, water vapour in the troposphere, type of soil, irregularity of terrain, and many other factors in the physical world. Because of this high variability and complexity, in order to increase the reliability of results, functions based on experiments and statistics are used to complement the physics equations (Seybold [2005]).

There is a vast number of models used in different physical environments and wave frequency bands. This thesis is focused in ground-based (near-earth) communications where the main propagation occurs near to the ground. The frequency bands considered are *VHF* (very high frequencies) and *UHF* (ultra high frequencies). The total range of these band designations goes from 30MHz to 3GHz and it is widely used in the most common systems of radio communication. In this range, when line of sight exists in the link, the signal strength is essentially calculated through Free-space path loss. If obstacles like mountains or buildings are present between the devices (transmitter and receiver), other electromagnetic propagation modes such as diffraction, refraction and multipath reflections occur and influence the final signal loss on the side of the receiver.

Hereafter, on a high-level, the term radio *signal strength* shall be used to mean radio *path loss* and vice versa. Both terms represent the desired ouput of a prediction of radio propagation and they complement each other. The power radiated by the transmitter is reduced (attenuated) until it reaches the receiver. To this reduction we call it *loss* or *attenuation*, which, subtracted to the power radiated by the transmitter gives the *signal strength* on the side of the receiver. Therefore, in spite of having different values, both kinds of output are interpreted for the same purpose on the radio propagation model.

1.1 Thesis Goals

The requirements for this project consisted in developing a parallel program to calculate the radio loss from a fixed site into an area. The used radio propagation model should be suitable for military operations. This context implies near-earth communications with frequencies somewhere in the *VHF* or *UHF* bands. Moreover, the calculated areas should be of rural type and with dimensions between *10*10 km* and *120*120 km*. One additional requirement was that the program would have to be run in ordinary desktop or laptop computers, instead of large supercomputers and other distributed systems that would not fit for military operations.

The chosen model was the *ITWOM* (Irregular Terrain with Obstructions Model) Shumate [2010]. This model is a recent update of the most popular radio propagation model, the Longley-Rice Model Longley and Rice [1968], and is in entire compliance with the given requirements.

A serial version of the program had to be developed and then modified to be run in parallel in order to compare the performance of the execution time and the accuracy of results. The parallel possibilities of the program were exploited for multi-core processor architectures as well as general-purpose computing on *Graphical Processing Units* (*GPGPU*). One additional goal was to analyse the constraints of modifying the source code of the used model, so it could work in both of these technologies.

1.2 Literature Review

Summaries of the most commonly used models for radio propagation prediction were compiled by Seybold [2005]. In his book, the near-earth models are divided in *Foliage*, *Terrain* and *Built-up Areas* modelling. The former section refers to models in which vegetation has a major influence in the loss of radio signal such as in the case of forests. The *Built-up Areas* section is related with urbanized areas where buildings have to be taken into account for predicting radio propagation. The *Terrain* section contains three models that have the terrain heights as the main factor for the result of the predictions: the Egli Model by Egli [1957]; the *ITU* Model, which is a result from several studies on diffraction theory available from the Radiocommunication Sector *ITU* (International Telecommunications Union) publications; and finally, the Longley-Rice Model described in [Longley et al., 1967, Longley and Rice, 1968].

The Longley-Rice Model is by far the most widely used of the above-mentioned *Terrain* models. Officially called Irregular Terrain Model, it was described in the late 60's by the U.S. agency *NTIA* (National Telecommunications and Information Administration) from the *ITS* (Institute for Telecommunication Sciences). The model has undergone several changes over time, and the latest version of the algorithm (1.2.2) described by Hufford G. is freely available on the *NTIA* website (of Commerce NTIA/ITS). A useful guide of this version can be found in a report by Hufford et al. [1982]. Another helpful publication concerning the use of the algorithm was made by Weiner [1986] who discusses its modes of propagation and input parameters. Chamberlin and Luebbers [1982] provides useful description of the Longley-Rice model and some clarifications when comparing it with *GTD* (geometrical theory of diffraction) propagation models.

A considerable amount of literature in parallel radio propagation prediction has been published for urban areas and indoor spaces, using supercomputers to support the calculations. The motivation for such studies was driven by the high computational requirements of the cellular phone networks. Very recently, several publications were made regarding exploiting the *GPU* capabilities for radio propagation calculations. In the specific area of radio propagation prediction, Bai and Nicol [2010] used *GPGPU* to accelerate an indoor model for wireless networks.

In the University of Arizona, Song et al. [2009] published data and function partitioning possibilities for the *ITM* and achieved successful performance results using the *IBM* Cell Broadband Engine Processor. Very recently, Song and Akoglu [2011] published a comparison between the CELL *BE* Processor and the *GPGPU* for the parallel version of the Longley-Rice model. Parallelisation strategies were defined, and experiments were carried around the global and shared memory of the device. The performance tests lead to the conclusion that the *GPGPU* device is the best platform to support the computation of the algorithm. These studies were focused in the point-to-point mode of the *ITM* and their tests were made in order to calculate the reference attenuation, excluding the statistical part of the algorithm. This last part returns the probabilistic confidence on the output of the algorithm, adding more calculations and variables to the program. However, this leads to performance decreasing and it is not always required by the users. For this reason, Song Y. et al. only considered the reference attenuation, which represents the main output of the model.

Shumate [2010] concluded that the diffraction calculations in the *ITM* are inaccurate for ranges in the line-of-sight. In this work, incorrect functions as well as the incompleteness of the model are reported. Combining the best of the *ITM* with *ITU* recommendations and other radio propagation laws, Shumate released the source code of his new model *ITWOM*, considering it, in his article, the "*first truly point-to-point, terrain-specific international radio reception signal strength prediction model*".

1.3 Existing Software

As part of the research stage of this project, some software packages having similar functionalities with the program required by this project, were tested. From these solutions, the *Probe 4* [V-Soft Communications, Version 4.2] was found to have the same feature of the program to be developed: it uses the Longley-Rice implementation provided by *NTIA* to predict radio loss over an area. The user enters the inputs required by the *ITM*, chooses a location to put the transmitter and finally calls the program to calculate the losses. The program then plots coloured zones (each with different intervals of loss values) on the map around the transmitter (figure 1.1 shows an example of this output). However, this program is not open-source and thus, it only contributed with a user perspective for the development stage of this thesis. Nevertheless, another program with similar features was found to be very helpful: *SPLAT!* [Magliacane, Version 1.4.0 (2011)]. This program was released under the *GPL* (General Public License) and its importance for this project is described in chapter 3.

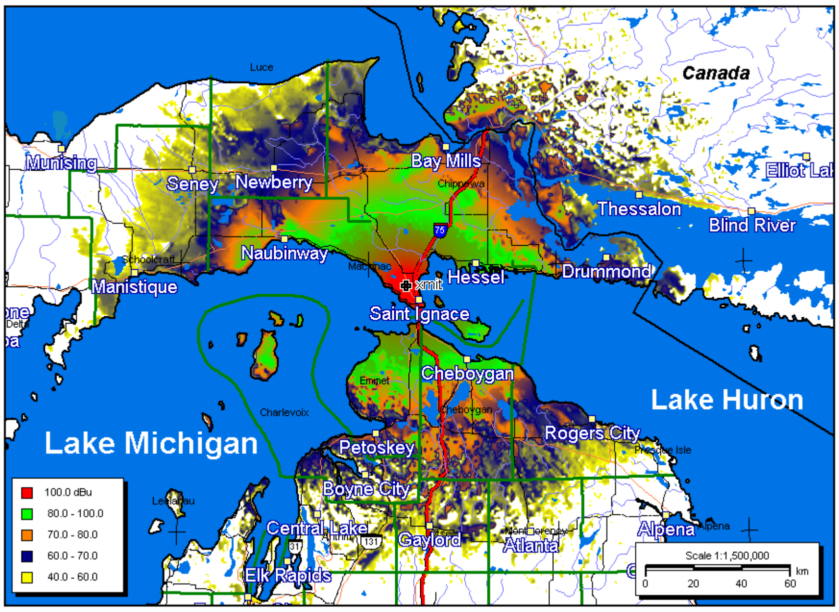

Figure 1.1: Example of a Longley-Rice coverage map on Probe 4, for an area in the state of Michigan, U.S. (retrieved from V-Soft's website)

1.4 Thesis Organisation

The high-level description of the Irregular Terrain Model can be found in chapter 2, as well as the modifications introduced by the Irregular Terrain with Obstructions

Model.

The design and discussion of the serial program, that applies the above-mentioned models in a given area, are documented in chapter 3.

Chapter 4 presents the design strategies and performance results of the Multi-Core parallel approach (4.1). The ideas behind the *GPGPU* adaptation to our program are also described in 4.2, as well as the limitations that prevented a successful implementation.

The conclusions drawn from both serial and parallel implementations are summarised in chapter 5. In this chapter, it can also be found a set of ideas for future work.

Finally, the appendix, with the averaged computational times of the implementations is presented at the end of this report, followed by the references used for this project.

Chapter 2

Irregular Terrain Model

This section describes the relevant aspects of the Longley-Rice Model (*ITM*), for better understanding of the program developed in this project. This model can be used in two modes: the area prediction mode and the point-to-point mode. The former, rather than using specific values of the elevations of the terrain, uses estimations in order to predict the signal strengths for the whole area. The latter, receives as an input the exact elevation points (terrain heights) between the transmitter and the receiver, returning a more accurate result. The area prediction mode takes much less computation and execution time that the point-to-point. However, the reliability of its results is not sufficient for many users, when comparing it with the latter. For this reason, this project only made use of the point-to-point mode.

The corrections and limitations that came with the recent Shumate's *ITWOM* Shumate [2010] are also summarized below.

2.1 Longley-Rice

The Longley-Rice model is a general purpose radio propagation model and it is widely used for ground-based predictions, based on the terrain characteristics of an area. It is based on electromagnetic theory as well as on statistics determined from experiments (on-site radio measurements) carried out by the American agency *NTIA/ITS* in the 60's. This same agency provides in their website the implementation of the model (algorithm) in the *C++* programming language.

2.1.1 Inputs

The model (in the point-to-point mode) receives inputs related with the antennas, the characteristics of the surface, the climate and the heights between the source and the destination. It also receives two values associated with the statistical part of the model: reliability and confidence. Although the explanation of each parameter being considered to be out of the scope of this document, the subsequent list can provide one idea of the dimension of the algorithm:

1. Elevation points between transmitter and receiver.
2. Distance between transmitter and receiver.
3. Antenna's heights and polarisation.
4. Frequency of the wave.
5. Surface Refractivity of the atmosphere
6. Relative permittivity of ground
7. Conductivity of the ground
8. Climate type (Climate codes categorised in the model, including Equatorial, Continental Temperate, Maritime Temperate, etc.)
9. Reliability of the variability (see 2.1.4)
10. Confidence of the variability (see 2.1.4)

From these inputs, the model calculates many other variables to be used on the computation of the final output. There is a total of 32 variables of type *double* initialised in the algorithm. Additionally we have the elevation points (array of *doubles*) that, for large areas, can represent a big issue in terms of memory management in the program.

2.1.2 Prediction Methods

With the input variables mentioned before, the algorithm creates the terrain profile, which consists in the elevations of the transmitter and receiver (including antennas heights) and also of the points in between.

Based on the profile of the terrain, the algorithm initially calculates two types of distances: the line-of-sight distance (d_L) and the scatter distance (d_x). Depending on the distance between the receiver and the transmitter, the Longley-Rice model uses one of three prediction methods. Each of these methods are represented in the source code by a sequential function that returns the final output of the program. Therefore, for each execution, only one of these functions is called by the program.

The value of the distance on the receiver (d_r) is in one of the following intervals:

- Line of Sight zone ($d_r < d_L$) - If the receiver is on the line-of-sight of the transmitter, i.e., the former is on the radio horizon of the latter and there are no obstacles in between, the model calculates the *reference attenuation* using two-ray multipath (a set of optics formulas).

- Diffraction zone ($d_L < d_r < d_x$) - After the line-of-sight distance and before the scatter distance, electromagnetic theories related with diffraction phenomena are used to calculate the attenuation.

- Forward Scatter zone ($d_r < d_x$) - If the receiver is located in this region, the *reference attenuation* is calculated according to empirical algorithms determined by Longley et al. [1967].

2.1.3 Main Output

The main output of the model is the *reference attenuation* on the receiver, which represents the median loss relative to free space. This free space loss is the reduction in signal strength of any electromagnetic wave, when the propagation medium between the source and the destination is the air. It it is calculated as a function of frequency and distance from the transmitter. On the other hand, the reference attenuation encompasses all the losses related with factors other than the free space loss, such as climate conditions and obstacles between transmitter and receiver.

If we add the *reference attenuation* to the free space loss, we obtain the total transmission loss on the side of the receiver, which is often the value required by the designer of the communication system. For this reason, the *ITM* algorithm provides the option to add the free space loss to the calculated *reference attenuation*.

2.1.4 Variability

Since the output described before represents a median value, the Longley-Rice Model provides the possibility of categorise the variations based on two variables introduced by the user: confidence and reliability (values between 1% and 99%). The explanation of this two terms in this context, can be found on Hufford et al. [1982, page 36]. These values are related with the variance of the calculated median loss that the radio system would notice during its use (over a period of time). If the given percentages of reliability and confidence are high, the system would likely observe loss values close to the one returned by the algorithm.

This part of the algorithm was discarded in the experiments made by Song et al. [2009], Song and Akoglu [2011], who only considered the main ouput of the model. The function that adds this variability factor to the main output contains 67 variables of type *double*, which requires a much higher demand of memory from the program using the algorithm. In addition, this function adds more execution time, leading to general performance decreasing.

2.2 ITWOM

The model used by this thesis in both serial and parallel programs was the *ITWOM*, which consists in a significant update to the Longley-Rice Model made by Shumate [2010]. This model was implemented in *C++* and replaces several subroutines of the *ITM*, without changing the input-ouput structure of the old model. This was made in order to permit the existing software solutions that use the *ITM*, to easily test with minimum refactoring, the new *ITWOM*.

According to Shumate, the *ITM* decreases its accuracy for terrain databases with less than a 30 arc-second scale. Moreover, its source code does not fully implement the features in Longley and Rice [1968] and has many deficiencies in the calculations for the diffraction zone. Finally, his work describes that many math errors and outdated approximations are found in the *ITM*. Apart from having these issues corrected, the *ITWOM* contains more accurate calculations for the line-of-sight zone, due to the including of several electromagnetic laws and theories.

Chapter 3

Serial Program

The program developed for this thesis returns the predicted radio losses in a given area. It receives as inputs the elevation files of that indended area, the necessary parameters for the propagation model, and the location of the transmitter. Having the transmitter always located in the same site, the program repeatedly uses the point-to-point algorithm (*ITWOM*) to return the radio loss of each point around the transmitter.

The design, as well as some implementation parts, were essentially based on the open-source program *SPLAT!* [Magliacane, Version 1.4.0 (2011)] which is implemented in *C* language. Like *SPLAT!*, the computational part of this program was also implemented in *C*, having *C++* been used for reading the elevation files. It was decided to use object oriented programming to read the elevation files because it was considered as source code that could be "re-used" for other kinds of programs. Regarding the computational part, as it follows a procedural structure without the need for objects, it was implemented in *C*. In fact, to change it to *C++*, we would only have to replace the functions *memalloc* and *printf* from the *C* standard library, with *new* and *cout*. One additonal reason for using *C* functions in this part, was related with a prospect of adapting the code to be run on the *GPU* (using *CUDA*).

As this project aimed to study the possibilities of parallelising a program of this kind, other issues such as robustness and scalability were considered out of scope.

Although the tests presented in this document have been carried out in *Windows*, the serial program also works in *Linux*, due to the use of standard C/C++ equally compiled by *Visual Studio 2008* and *gcc*. The *CPU* used was the *Intel Core i7-860S Processor*.

The subsequent sections explain the functioning of the program and present a discussion around the results obtained.

3.1 Design

The design of this program was based on the "Plot Longley-Rice Map" feature present in *SPLAT!*.

On the first stage, the program reads the elevation files of the area and allocates them on memory. It then computes the losses for the whole area and writes the output into a file. Finally, the program plots the results displaying a coloured map. Figure 3.1 represents a high level sequencial structure:

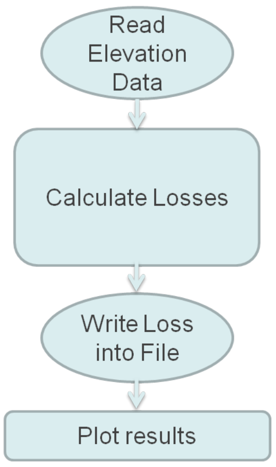

Figure 3.1: General structure of the serial program

Hereafter, both the terms *site* and *point* will be used to represent a location, which is a pair of cartesian coordinates (longitude and latitude).

After having the elevation values on memory, the program receives the location of the transmitter, which can be located anywhere inside the limits of the given area. For each point in the 4 edges (North, East, South and West), a structure *ray* is created to store all the relevant information between these points and the source (site where the transmitter is located). This structure contains the coordinates and the elevation values of all the points found between the source and the point in the edge. The way these points are obtained is explained in 3.1.2. Figure 3.2 is presented to help the understanding of the functioning of the program, on a high-level: This figure shows black lines connecting the source point to several points in

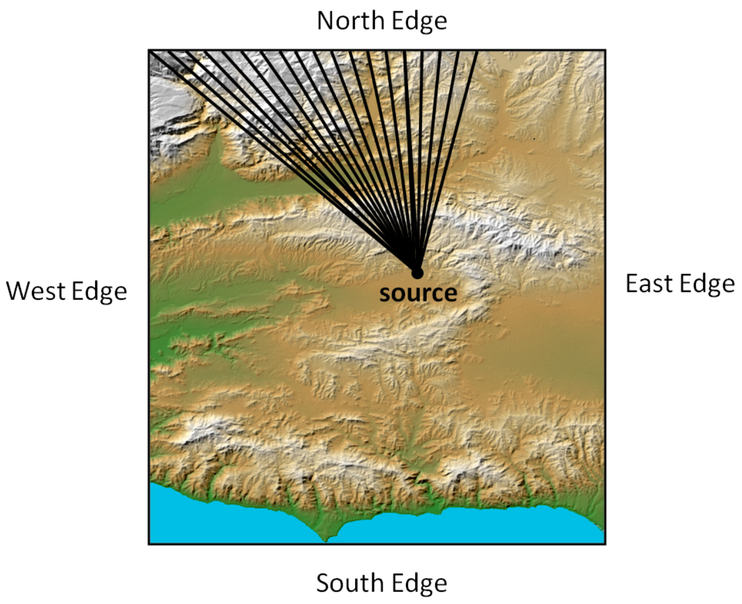

Figure 3.2: Illustration of the functioning of the program

the *North Edge*. These lines represent the *rays* of points that will be processed, in order to calculate the radio losses in each of them. In spite of figure 3.2 being only illustrating lines connecting part of the North points to the source, the program calculates the *rays* for all the points in the 4 edges (in order to cover all the area).

3.1.1 Read Terrain Data

The terrain data used was SRTM (Shuttle Radar Topography Mission) files with resolution of 3 arcseconds. The Shuttle Radar Topography Mission [National Aeronautics and Space Administration, and National Geospatial-Intelligence Agency, Version 2.1] was created from an international cooperation between the American, German and Italian Space Agencies and resulted in the most complete and accurate topographic database of planet Earth. Figure 3.3 shows the World Map with the regions that have available *SRTM* elevation data. It can be seen that the values of

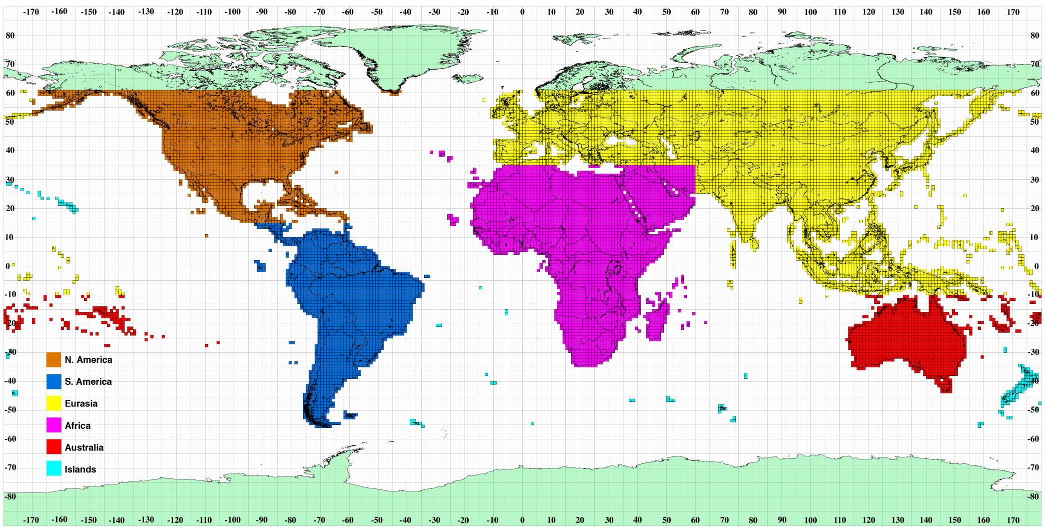

Figure 3.3: World Map with SRTM tiles - available on SRTM Website

the Latitude are negative in the South and positive in the North parts of the World. In the case of the Longitude, the values are positive for the East and negative to the West parts. This methodology is used in this program, as opposed to the one that identifies the coordinates by a positive value and the direction. For example, (40; -50) is used instead of (40N; 50W).

Figure 3.4 is a result of zooming in the previous map for the areas of Ireland and Great Britain.

Each *SRTM* file stores the elevation values for a tile of 1 by 1 degrees. these tiles are represented by the squares in both figures. The 3 arcseconds scale indicates that each file is composed by 1200 * 1200 points with approximately $3 * 1/(60 * 60)$

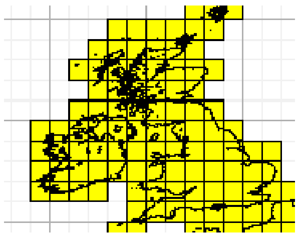

Figure 3.4: SRTM tiles in the areas of Ireland and Great Britain

degrees between each of them, which is equivalent to 90 meters in the *UK* (due to the Earth not being a perfect sphere, the distance in function of the angle is not constant).

The function that reads the elevations receives as input the coordinates from the southwest and northeast corners of the intended area. Subsequently, it searches for the required *SRTM* files, stores them temporarily on memory and returns a bi-dimensional array with the needed elevation values. This array of points will be further used in the computation section, which will then return the loss value for each of these points. Figure 3.5 helps the understanding of this part. The squares

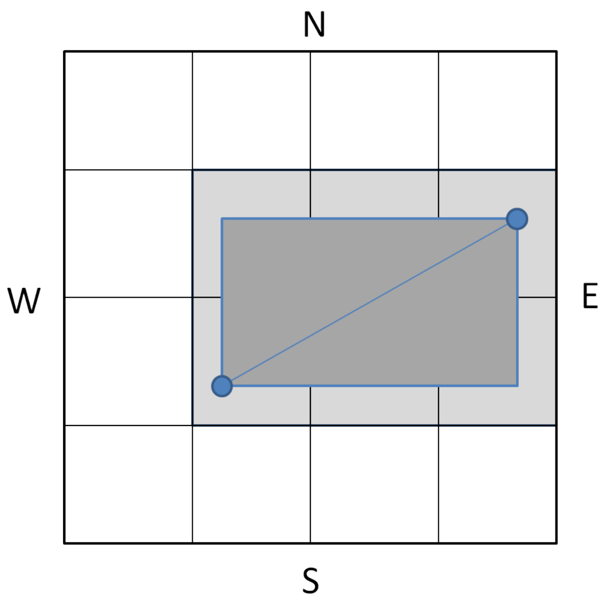

Figure 3.5: Illustration of Read Terrain Data section of the program

in half-tone gray represent the *SRTM* files that are read and temporarily stored in memory. From all these points, only the ones in the darker gray area are returned for the computation (thus being the only structure remaining on memory).

3.1.2 Computation

As stated before, one *ray* is a structure containing the points between the source and one point of the edge. Figure 3.6 illustrates one example for a representative area of 8 per 8 points, in which the source is located in the centre and the last point is in the northeast corner.

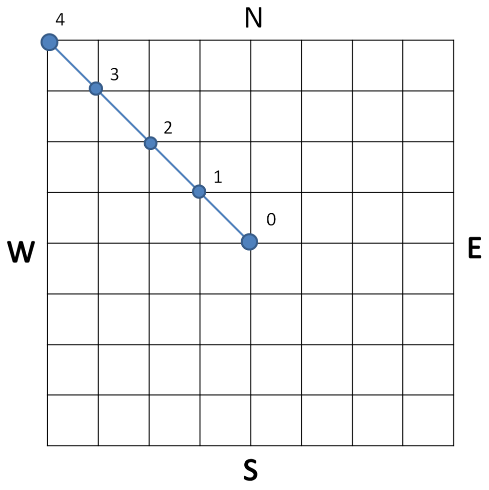

Figure 3.6: Illustration of a *ray* structure

From each pair of source and destination (point in the edge), it is necessary to find the best fit for each of the intermediary points. For this purpose, the function *ReadPath* was taken from *SPLAT!*. It represents the only portion of source code of this open-source software that was used in our program. This function uses geographical calculations based on the longitude and latitude values of the points. Due to its complexity, the explanation of its functioning was considered outside the scope of this thesis. As shown in figure 3.7, if we draw a straight line between source and destination, it will not match with many points. This is the inevitable constraint of simulating 360° propagation around a point in a grid. Due to not being possible to model the reality in a perfect way (one needs approximations), this "best-fitting" techniques are often used by all kinds of coverage problems.

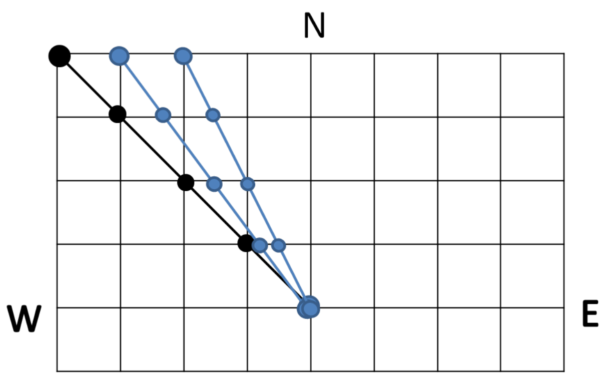

Figure 3.7: Illustration of the overlaping with multiple *rays*

One of the consequences of this geometric approach is the overlaping of points in the *rays*, that is, each *ray* will store points that will also be stored by its "neighbours". These overlaps are naturally more accentuated in the points near to the source. As it is further explained in this section, the program uses the propagation algorithm to calculate the radio losses for the points of each *ray*, often leading to different results for each location (due to the previously mentioned overlaps). Therefore, the order in which the *rays* are processed will influence the final result. However, this does not influence the final purpose of our program, because there is no such thing as a 100% correct result in prediction problems. Moreover, if this order is changed, the variation in results is never significant for the final user.

Another consequence is that a lot of points near the edges (specially in the corners) will not be processed and, therefore, a radio loss can not be obtained for those locations. This does not become a problem if the user requests the program to process a slightly larger area than the one intended in reality. The understanding of these two consequences can be helped by the graphical output of our program, shown in Section 3.1.3.

In the case of figure 3.6, our program will iterate over the 4 points (does not consider the source) calling the *ITWOM* to obtain the radio loss for each of them. Each time the propagation algorithm is called, it receives the source (always constant), the destination (point in which the loss is returned) and the elevation points between. In the example of figure 3.8, considering the 3^{rd} iteration, the source is the point 0 and the destination is the point 3. Therefore, based on the terrain information provided by the elevation values in the points 1 and 2, the algorithm will return the radio loss for the point 3. Using big-O notation, we can conclude that

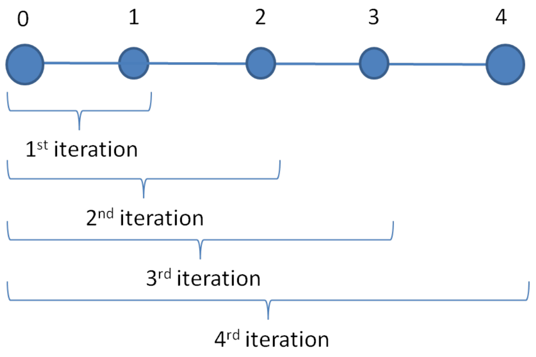

Figure 3.8: Illustration of iterations in a *ray*

the program will call the radio propagation algorithm *O(n-1)* times for each *ray*.

If the area corresponds to an entire *SRTM* tile (1200 * 1200 points), having the transmitter on the centre, we trivially notice that the *ray* with greater length corresponds to an approximation of $\sqrt[2]{600^2 * 600^2}$, which gives 849 points, and therefore, 848 calls to the algorithm. The number of *rays* is given by the sum of all the points in the 4 edges (the corners are not repeated): $1200 + 1199 + 1199 + 1198 = 4796$. It can be seen that, for an area of this dimension, the computations are quite demanding, both in terms of execution time and memory, since a complex algorithm needs to be called more than 3 million times. More details regarding this subject can be found below in the Results and Discussion section (3.2).

3.1.3 Plot Results

After computing each *ray*, a global bi-dimensional array (*global_losses*) is updated with the correspondent results (radio losses). When there are repeated points in the “neighbouring” *rays*, we have different results overwriting the same positions in *global_losses* (as stated before, this is not a concern for our program). Subsequently, when the program finishes the processing of all the *rays*, the *global_losses* is entirely written into a file (*.txt*). The program then calls Gnuplot [Kelley and Williams, Version 4.4.3] which produces the graphical output. Figures 3.9 and 3.10 are examples of the output of our program for an arbitrarily chosen area with dimensions of 1200 per 1200 points, located near Vyatka, Russia (transmitter is located in the centre point). In the left and bottom we have the dimension (number of points) of the area, and on the right, the radio loss values (in decibel, as returned by the propagation algorithm) which determines the colour value.

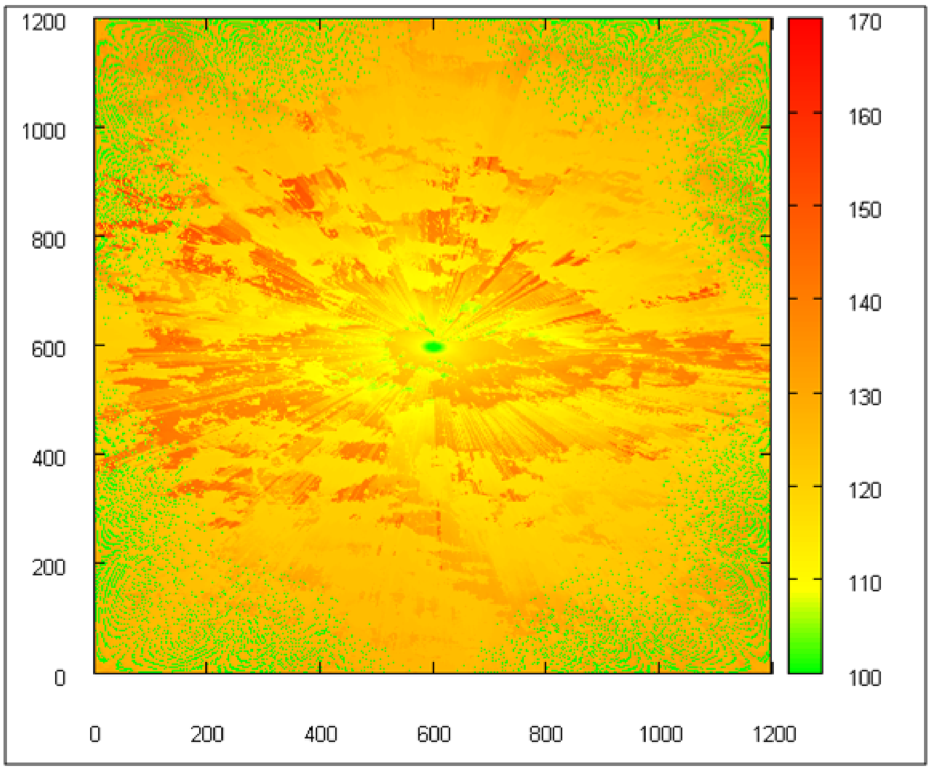

Figure 3.9: Example of graphical output with multiple colours. Transmitter located in (58N, 48E). Area (1200*1200 points) located near Vyatka, Russia.

The previous figure is composed by multiple *RGB* (Red Green and Blue model) values that gradually vary around the main colours: Green, Yellow, and Red. However, this figure is here presented only to better show the characteristics of the loss results. What the users often require for an output is a graphical model composed of 3 colours as shown in figure 3.10.

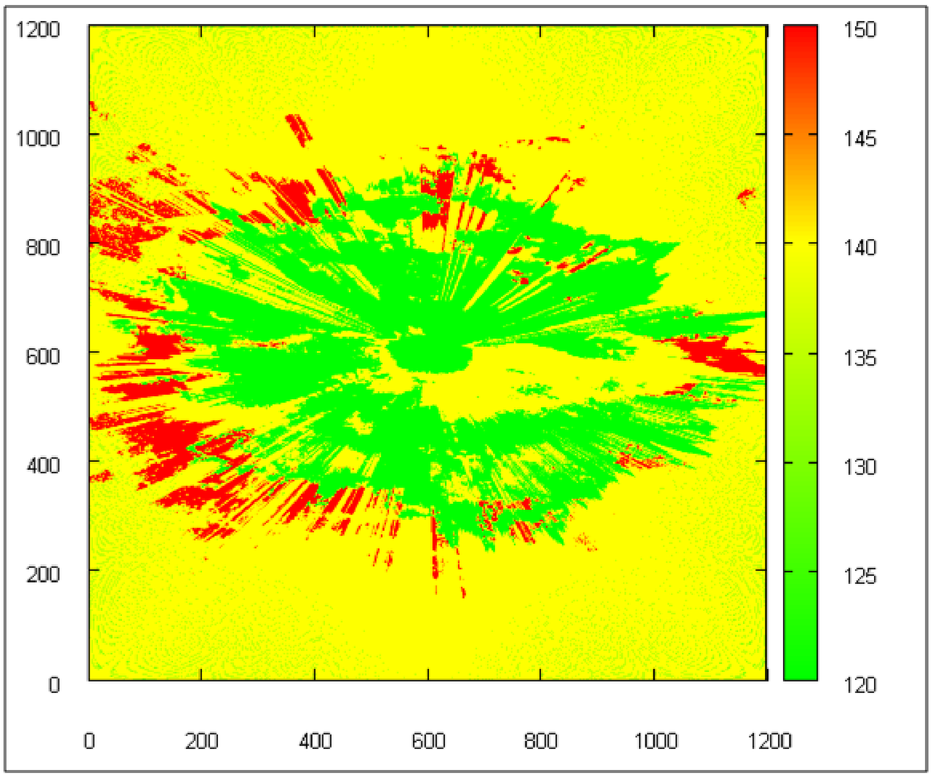

Figure 3.10: Example of graphical output with 3 colours. Transmitter located in (58N, 48E). Area (1200*1200 points) located near Vyatka, Russia.

The green colour indicates the zone where the receiver would work with a maximum level of confidence. The zone in yellow, as an intermediary between green and red, usually means that the radio system would work but with less confidence. If the receiver is located in the red zone, most likely the communication link would not work. All of this is decided by the user, who determines the intervals of radio losses that should correspond to each colour. In the case of figure 3.10 the intervals were the following:

$loss < 120$	Green
$140 < loss < 150$	Yellow
$loss > 150$	Red

3.2 Results and Discussion

After the completion of the serial program, performance tests were carried out for the example shown by figures 3.9 and 3.10. The antenna's heights and frequency were arbitrarily chosen, while the other parameters were taken from an example in the documentation of the Terrain Analysis Package software [SoftWright, Version 6.0]. It was prefered to rely on the documentation of a commercial software in order to get a normal behaviour from the algorithm, since it is very sensible to the entered parameters. This sensibility exists due to the fact that great part of the model was based on experiments, which means that any combination of unrealistic input values will result in unexpected behaviour and possibly errors. The parameters used for the *ITWOM* were the following:

Variable	Meaning	Value
eps	Relative permittivity of ground	15.0
sgm	Ground conductivity	0.005
en0	Surface refractivity	301.0
pol	Polarisation	0 (horizontal)
klim	Climate Code	5 (Continental Temperate)
rel	Reliability	0.99
conf	Confidence	0.99
hg1	Height of transmitter antenna (m)	10.0
hg2	Height of receiver antenna (m)	10.0
frq	Frequency (MHz)	50.0

The validation of the results is not straightforward since it is not possible to obtain exactly correct values for the predicted losses and, the domain of output values is very large. Moreover, the existing software packages do not provide the output values, except by graphical plots of the coverage zones. Using the same parameters, area and location of transmitter, the output verified with *SPLAT!* is shown by figure 3.11.

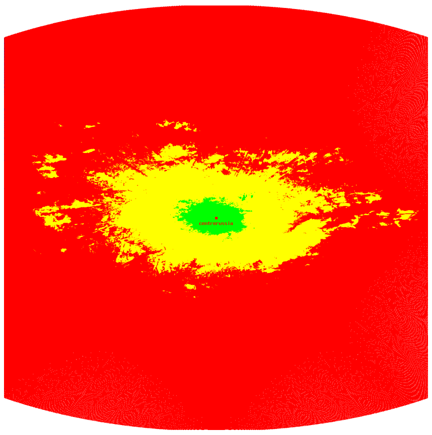

Figure 3.11: Example of graphical output in *SPLAT!* with 3 colours. Transmitter located in (58N, 48E). Area located near Vyatka, Russia.

However, it was only possible to run a previous version which does not contain the *ITWOM* (as all the current commercial software packages), and therefore, the figure corresponds to results returned by the old *ITM*. Moreover, because of the several differences on the way our program plots the data and the one found in *SPLAT!*, it was not possible to compare the validity of the returned radio losses with this or any other existing software solutions. Nevertheless, the arrangement of the green, yellow and red zones (figures 3.9 and 3.10) gives a good indication for the successful functioning of the *ITWOM* algorithm.

It can be noticed that the graphical output differs greatly between the two models due to the *ITWOM* arguing that the diffraction losses (found in areas behind terrain obstacles) were not being correctly calculated by the *ITM* (they should have greater values). Consequently, in the latter we find a monotone decrease of loss from the centre to the edges, whereas in the former we have red before the yellow zones, which corresponds to the areas near the obstructions (hills or mountains), in the opposite side of the transmitter.

For the area and parameters described before, the following computation times were obtained in *Windows* with the above-mentioned *Intel Core i7-860S*:

The first column indicates the number of total elevation points processed, whereas the second corresponds to an aproximation of the area dimensions for that part of

Number of points	Area Dimension(km)	Computation time
1200*1200=1440000	108*108	54.76525 s
1000*1000=1000000	90*90	35.93775 s
800*800=640000	72*72	20.285 s
600*600=360000	54*54	10.1565 s
400*400=160000	36*36	4.207 s
200*200=40000	18*18	0.98075 s

Table 3.1: Serial Implementation - Average Computation Times.

the Earth (distance between each point is roughly 90 meters). These computation times are averaged values obtained from 4 executions each. It can be seen that performance is not a concern for 200*200 km areas. On the other hand, from 4 seconds on (400*400 km), the overall execution time starts to be a negative issue for the user, specially in the larger areas.

Figure 3.12 presents a chart with the time divided by total processed points. This chart shows that the computation time per point increases (almost linearly) with the number of data processed, that is, for larger areas the performance degrades. The memory management, i.e, storing and overwriting very large arrays, is the most likely reason for this.

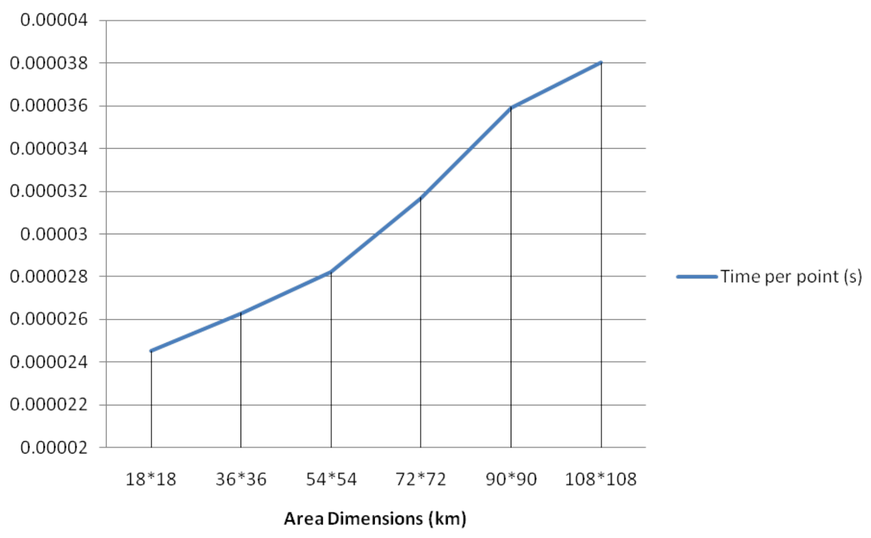

Figure 3.12: Chart with Times per Point of serial implementation

Figure 3.13 shows that the distribution of execution time is not the same for small and larger areas.

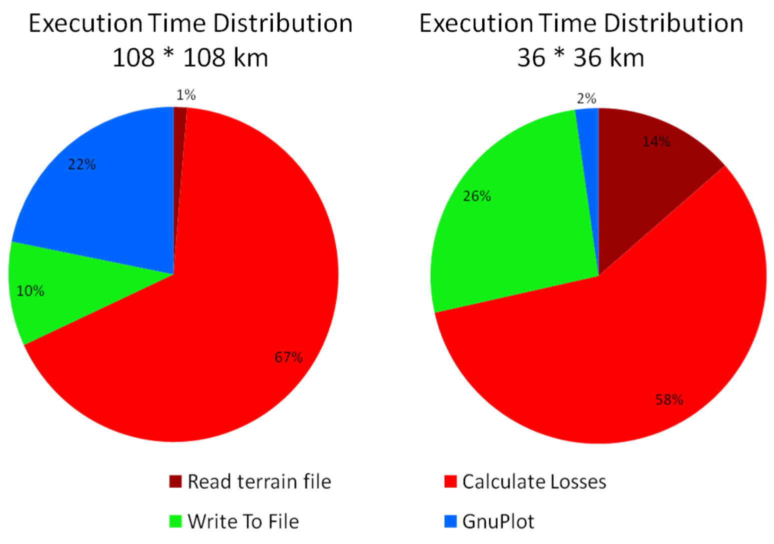

Figure 3.13: Execution time distribution for 108*108 km and 36*36 km areas.

Reading the terrain files is not an issue for larger areas as it is the writing and plotting the output. In fact, writing values into a file and then use a software (Gnuplot) to plot it, is not a good approach for presenting graphical output of large data. *SPLAT!*, for example, writes directly the loss results into a portable *pixmap* image file (*.ppm*), which is faster than as in our program (although the user as to manually open the *.ppm* files). Apart from this, there are many other ways to improve this part of the program. However, this thesis focuses only in the computation which, as it can be seen on figure 3.13, represents the largest share of the total execution time.

Chapter 4

Parallel Program

The correct approach to improve the performance in the computation is to exploit data partitioning. It is proved by the serial profiling found in chapter 3 that the execution time increases exponentially with the increasing area dimensions and, consequently, with the number of *rays* to be processed.

This chapter presents two solutions to parallelise the implemented serial program: one using *OpenMP* (version 2.0) technology for multi-core processing and the other using *CUDA* for *GPGPU*. This section discusses the results obtained in the first solution and explains the nonachievement of the *GPGPU* implementation. A brief introdution to *Hyper-Threading* can also be found in this chapter, as the tests were carried on a machine with this technology.

4.1 Multi-Core Approach

4.1.1 OpenMP

OpenMP was the technology chosen for the multi-core solution due to its simplicity and consistency in supporting multi-threading, as well as portability and recognition by most of the "small-scale" shared memory systems.

The *OpenMP API* is composed by compiler directives, environmental variables and library routines that can easily be added to working software, only with minor changes on the original source code [OpenMP Architecture Review Board, Version 3.0 (2008)]. With these preprocessed directives, new threads can be created in runtime from the main thread (master) of the program, in order to process sections of code independently. Using the *fork/join* design pattern, *OpenMP* also allows to share an iterated data structure in order to divide the computation of its data. However, it is the responsability of the developer to ensure the correctness of the loop and used clauses, in order to avoid concurrency inconsistencies and other errors originated by inconvenient shared variables.

In this implementation, *Microsoft Visual Studio 2008* was used due to its built-in support for version 2.0 *OpenMP*, which permits the processing of its clauses and directives by *Visual C++* compiler (thus avoiding time consuming setups).

4.1.2 Hyper-Threading

Hyper-Threading is a technology developed by *Intel* for their recent processor families. It was created to enhance the performance of multi-threaded applications by presenting a single physical core as a pair of virtual processors to the operating system (which must be specifically optimised for *Hyper-Threading*). This feature is transparent to the operating system as well as to the programs running on it, that is, the logical/virtual processors are treated as if they were physical cores.

The reason that aroused the creation of this technology is simple: in a middle of a task, the processors very often become stalled due to memory access or data dependency (waiting for used resources), wasting their execution resources in these periods of waiting. With *Hyper-Threading*, the execution resources of one physical processor are duplicated, creating the possibility of having 2 streams of instructions for each core. These streams are still executed one at a time, but their switching is made in a way that takes advantage of the latency periods that might occur in each task [Pessach, 2005]. Therefore, the performance improvement provided by this technology highly depends on the characteristics of the software. In fact, the best results are achieved for multi-threaded programs (using *OpenMP* for example) where threads share very few resources [Tian et al., 2002].

4.1.3 Parallelisation Strategies

The execution time increases with the dimensions of the area, making it necessary to distribute the data processing across the available threads.

It is not possible to partition each *ray* because, from the source to the edge, the calculation of the loss in each point depends on the elevation of the previous points. However, the calculations are independent from *ray* to *ray*. Therefore, the strategy is to distribute all the *rays* of the area and write the results obtained into a single global structure located in memory. As there are several results being overwritten in the same location of this global structure, the order in which the threads return will influence the final results. However, as it is explained in 3.1.2, this does not represent a problem in our case.

In the serial program, 4 *for* loops are used to divide the computations by the 4 edges. This is because the iteration over the edges is different for each of them: for the North and the South, the longitude is incremented and the latitude is constant, while for the East and the West it happens the reverse. For the same purpose, a single *for* loop could also be used, having 4 nested *if* statements, but it could not exploit the *OpenMP* directives as well as the first form.

Two approaches using *OpenMP*, hereafter called of "parallel for" and "sections", take advantage of two different directives provided by this technology.

Figure 4.1 illustrates the first approach, that made use of the *parallel for* directive. The data processed in each of the 4 loops is partitioned across the available threads. The number of threads is previously defined by the *omp_set_num_threads* routine.

The second approach (figure 4.2 uses the *sections* directive to assign a thread to each of the loops. Therefore, this approach needs 4 threads to compute the *rays* that connect the source to each edge. As it is discussed in 4.1.4, this happened to be the best solution due to the characteristics of the hardware used.

```
omp parallel for
  for(i=0; i<end_of_Edge; i++)
    //Compute rays from transmitter to points of North edge
omp parallel for
  for(i=0; i<end_of_Edge; i++)
    //Compute rays from transmitter to points of East edge
omp parallel for
  for(i=0; i<end_of_Edge; i++)
    //Compute rays from transmitter to points of South edge
omp parallel for
  for(i=0; i<end_of_Edge; i++)
    //Compute rays from transmitter to points of West edge
```

Figure 4.1: Illustration of *OpenMP* “parallel for” strategy

```
omp parallel sections
    section
        for(i=0; i<end_of_Edge; i++)
          //Compute rays from transmitter to points of North edge
    section
        for(i=0; i<end_of_Edge; i++)
          //Compute rays from transmitter to points of East edge
    section
        for(i=0; i<end_of_Edge; i++)
          //Compute rays from transmitter to points of South edge
    section
        for(i=0; i<end_of_Edge; i++)
          //Compute rays from transmitter to points of West edge
```

Figure 4.2: Illustration of *OpenMP* “sections” strategy

4.1.4 Results and Discussion

This multi-core approaches were executed in the same machine as the serial program, using the *Intel Core i7-860S Processor*. This *CPU* is composed by 4 physical cores, each with *Hyper-Threading* technology, therefore providing a total of 8 logical processors.

Figure 4.3 shows the speedup for the first parallel strategy. For large areas, the program takes advantage of the logical processors up to 7 threads, despite of the speed-up being a logarithmically increasing funcion. *OpenMP* is not able to distinguish the virtual from the physical cores. Therefore, it is not possible to assign the first 4 threads to the 4 physical separated cores and then use virtual processors with the remainder (which could lead to increased performance if the user chooses to use only 4 threads). Nevertheless, when computing large areas, *Hyper-Threading* becomes a very positive influence for this program. The speed-up decline when adding the 8th thread, is due to the operating system tasks and *Visual Studio 2008* being using the resources of one logical processor, during the execution of our program. For small dimension areas, however, this approach only improves the performance until 5 threads. Consequently, we infer that the overhead of spawning more than this number of threads becomes an undesired effect when we have less data to be computed.

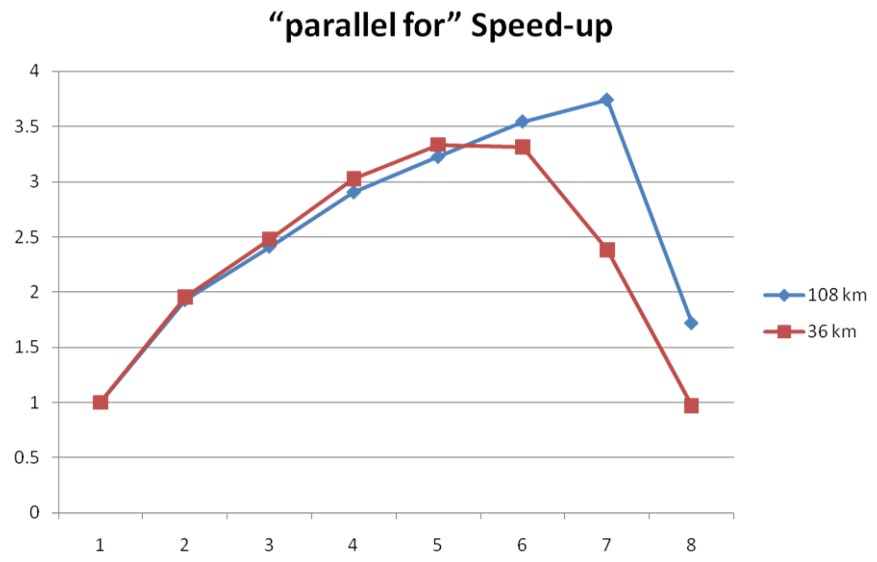

Figure 4.3: Chart with Speed-up in "parallel for" strategy

Figure 4.4 shows the speed-up obtained with the "sections" approach for different area dimensions. It can be seen that the initialisation overhead of the 4 threads only affects the speed-up in the smallest areas (18 * 18 km). Nevertheless, as it can easily be observed in figure 4.5, this solution is better than the previous for all dimensions, having a constant speed-up of approximately 5.5.

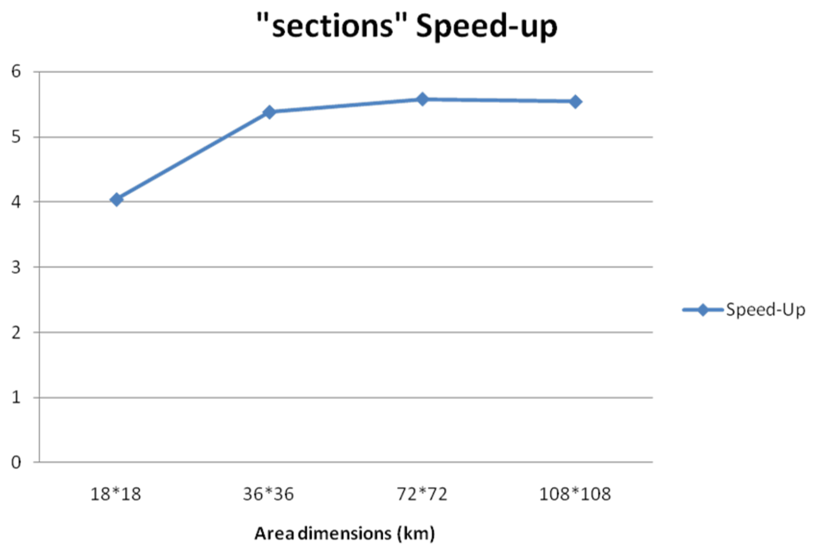

Figure 4.4: Chart with Speed-up in "sections" strategy

The inconvenience found in the "parallel for" approach is the fact that this directive uses fork/join for each loop. This overhead in spawning threads and joining them in the end, only happens once in the "sections" strategy. Consequently, the former has worse results than the latter for the hardware used. On the other hand, it can easily be infered that, with a *CPU* equipped with more than a certain number of physical cores, the "parallel for" would overcome the "sections" approach for large areas.

It is important to note that this program used dynamic memory allocation for the *rays*, unlike *SPLAT!*, which declares static arrays with a fixed size determined by the user, during its installation. In fact, this software asks the user to choose the memory he is willing to dedicate to the computations: $25MB$ for a maximum dimension of 2400 * 2400 points, $52MB$ for 3600 * 3600 points, and so on.

Apart from the inconveniences found in using static allocation for large arrays, if the radio losses are being calculated for a small area, *SPLAT!* wastes a lot of unused memory. Moreover, during our implementation, it was possible to conclude that using either one or the other way of allocating memory does not affect performance.

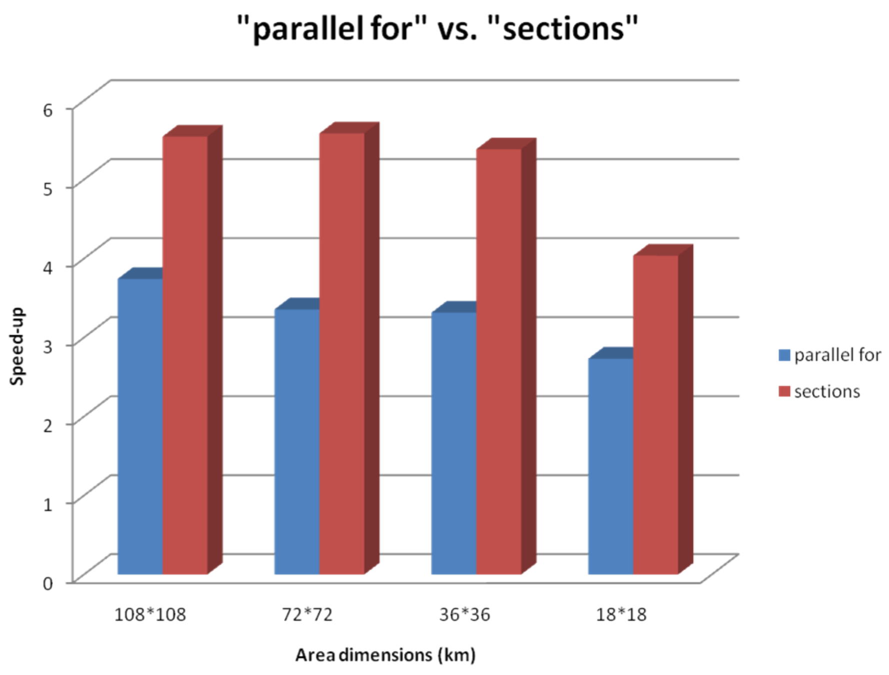

Figure 4.5: Comparison between speed-up in both "parallel for" and "sections" strategies

Finally, it is worth mentioning one problem that occurred during the implementation of this parallelization. The propagation algorithm was returning infinite results for some locations, resulting in the graphical output presented on the left side of figure 4.6. This unexpected behaviour, after some time consuming debugging, was found to be due to some variables in the *ITWOM* being declared as *static*. The address space of these variables were therefore being shared by the threads, and the problem was solved by simply deleting their *static* declarations.

4.2 GPGPU Approach

The repeated use of the radio propagation algorithm for thousands and some times millions of points, represented a likely opportunity to take advantage of general-purpose *GPU*. The highly parallel structure of these devices is now being exploited for intentions other than computer graphics, making it possible to run many different scientific applications with outstanding performance improvements. These

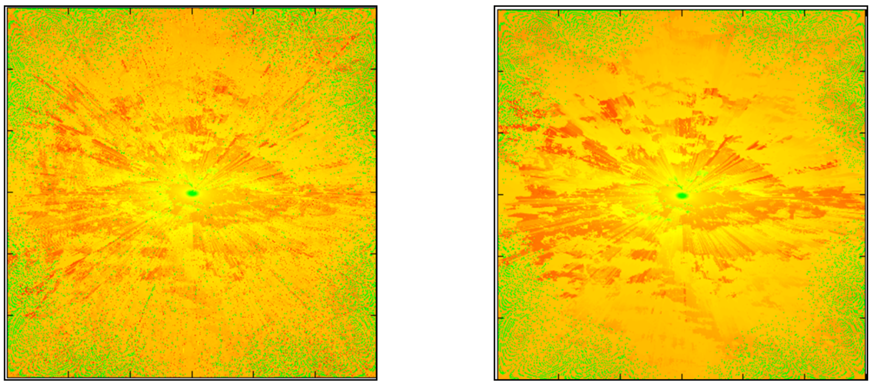

Figure 4.6: Graphical output of parallel program before (left) and after (right) removing *static* variables in the *ITWOM*

applications with *SIMD* (*Single Instruction, Multiple Data*) architectures, have the very recent possibility to benefict from the hundreds of processors provided by the *GPU*.

Nvidia, the greatest vendor and pioneer in *GPGPU*, created *CUDA* (*Compute Unified Device Architecture*), a parallel computing architecture that simplified general-purpose programming using their graphic cards. Its recent and widely strong recognition, and the relatively short learning curve for a *C/C++* developer (it is basically a *C/C++* extension), made *CUDA* the elected technology for this *GPGPU* approach. In order to adapt any application to *CUDA*, it is essential to understand the memory hierarchy of the *GPU*. There are several types of memory, each with different latency and access patterns, which have to be taken into account when designing the programs:

- Registers - fast on-chip memory (small capacity) used to store automatic variables;
- Global memory - memory with higher storage capacity but also with the highest access latency (ideally to be used only in transfers between host and device);
- Per-thread local memory - have the same latency as the previous (should also be avoided in internal computations), used for variables declared in each thread that can not be stored in the registers;

- Per-block shared memory - fast on-chip memory (small capacity), available only to the threads of the same block.

- Constant memory - cached (read-only), form of the Global memory, that is, its memory space resides in the device memory but its data can also be stored on the on-chip cache of the *GPU*.

- Texture memory - as the previous, it is read-only and cached, but it can store larger data structures and also optimise access patterns with 2D spatial locality.

The desctiption of these types of memory can be found in any version of the *CUDA C* Programming Guide.

The hardware used was a *Quadro®FX 1800*, the graphic card of the same machine mentioned in the multi-core approach.

4.2.1 Parallelisation Strategy

With the possibility of processing thousands of threads simultaneously, the obvious strategy is to have each *ray* being calculated by one thread. For example, with the largest area, the number of threads would be 4800 (1200 * 4 edges). As figure 4.7 illustrates, the computations shall be carried out by the graphic card (device) while the remainder sections are processed by the *CPU* (host). As an extension to *C/C++*, *CUDA* uses its compiler and assembler (*nvcc* and *PTX ISA*) to compile the source code that is to be run on the device, while the remainder parts are compiled by the native compiler (*Visual C++*, *gcc*, etc.).

The function that the host calls to be run by each thread on the *GPU* is the *kernel*. In this approach, the *kernel* will execute the computations corresponding to one *ray*:

1. Determine the point of the edge based on the *ID* (identification number) of the thread.

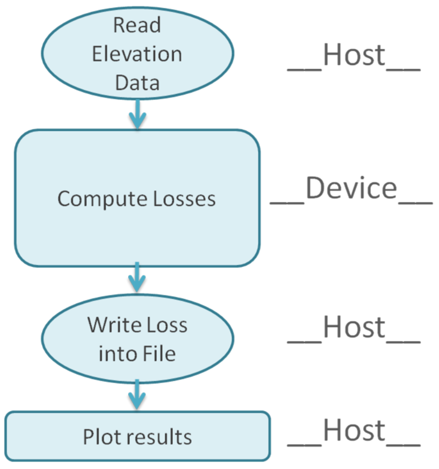

Figure 4.7: Illustration of the *GPGPU* approach

2. Create the *ray* with the points between the source and the point of the edge.

3. Call the *ITWOM* algorithm for each point of the *ray* and write the loss into a global structure of results (located on the device Global memory).

As the device needs to access the elevation data for the *ITWOM*, before calling the *kernel*, it is necessary to allocate the elevations of the whole area on the device memory. While attempting to implement the *GPGPU* approach, the Texture Memory was used to store these elevations, as they are only read by the computations. However, the spatial locality optimisation provided by this memory would probably not be exploited by the accesses to the elevation data. Therefore, the cache in Texture Memory would possibly not represent an advantage over the Global Memory. The evaluation of the most suitable way of storing the elevations, needs still to be performed in future works.

After all threads have completed their executions, the global structure with the loss results should then be sent back to the host. This and the initial transfer of the elevation data, were actually tested for the largest areas, both taking nearly 1 second of time. Taking into account that the best multi-core approach ("sections") took approximatelly 10 seconds to finish the calculations, we conclude that a successful *CUDA* implementation would most likely far surpass this result. However, there were many obstacles and constraints encountered in our program and in the *ITWOM* algorithm, which prevented to have a successful *GPGPU* implementation. The problems found are described in the section below.

4.2.2 Discussion

During the process of adapting the serial program to the *GPGPU* implementation, the first obstacle to came up was the *ITWOM* itself. In this approach the *ITWOM* had to be executed on the device, and therefore needed some modifications in order to comply with the *CUDA* requirements:

- All the instructions related with the *C++* library for complex numbers had to be exchanged for *CUDA* functions ("cuComplex.h");
- *C++* assertion functions and "static" declarations had to be removed.
- All the double precision floating-point variables and functions were changed to singe precision.

It is important to notice that, during this process, relevant bad programming practices were found in the *ITWOM* and also in the *ITM*: declared unused variables and variables initialised in conditional statements.

Despite the above-mentioned corrections that should be made to both of these radio propagation algorithms, there is one major issue that is worth to be concerned about: the available memory. In the *Quadro®FX 1800*, it is not possible to have more than 16KB of local memory per thread. This restrictive limit imposed by the hardware, represented the major obstacle to this approach. In fact, together with the calculations used to find the points of the *rays*, the several calls to these complex propagation algorithms demanded too much memory from the *GPU*. It is then important to distinguish this project from the work made by Song and Akoglu [2011], in the sense that this publication describes the use of the *ITM* on the *GPU*, assuming that the points of the *rays* were already determined. In our case, the design of the *ray* computations would have to be reconsidered and many tests would have to be carried out, in order to solve the memory constraints imposed by the *GPU*. Consequently, the efforts required to adapt our program were too challenging to succeed in this approach.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

After the design and implementation of our serial program, consistent radio loss predictions were returned for the tested areas between 18 by 18 km and 108 by 108km. Being influenced by the *SPLAT!* software, this program brings improvements regarding memory management, mainly by introducing dynamic allocation as opposed to the static used by *SPLAT!*. However, it is worth to mention that these differences did not result in performance changes.

In spite of the *ITWOM* being the mainly used model, the *ITM* was also tested to check if there were any variance in the performance, which turned out to be similar for both algorithms.

From the tests carried out in the serial implementation, it was inferred that the overall performance decreases exponentially with increasing dimensions. The long execution times for the largest areas (approximately one minute) urge the need for using data partitioning in this, as well as in other software with similar goals.

With the hardware used and for all area dimensions, the best multi-core (*OpenMP*) approach was "sections", with which there was a speed-up of approximately 5. However, the tests were carried out on a *CPU* with 8 logical processors. Because of this, we concluded that, if there were more processors available, the “parallel for” would possibly overcome the “sections” approach.

The *Hyper-Threading* technology provided by the *Intel Core i7*, turned out to be a positive influence, because it did not bring significant performance degradation, as it was expected. Nevertheless, for future implementations, it would be better to have a mechanism that differentiates the logical from the physically separated processors, before proceeding with the parallel computations.

Due to memory limitations imposed by *CUDA*, it was not possible to successfully adapt our program and take advantage of the high-parallelism capacities of *GPGPU*. The *ray* calculations, together with the radio propagation model, demanded more than the maximum memory provided by the *GPU*. This high memory usage is due to the large number of variables in the *ITWOM* algorithm (similar to the *ITM*), added to the big arrays required by the *ray* calculations (worse for largest areas). Therefore, in order to overcome the constraints imposed by the hardware and the *GPGPU* programming model, the design of our program needs to be reconsidered. Additionally, due to the complexity of the memory hierarchy in *CUDA*, many experiments would have to be tried out to achieve a successful implementation.

5.2 Future work

Several improvements and features would have to be added to the current program before being available to the end user:

- The section responsible to reproduce the graphical output should be replaced by more recent and powerful geographical visualisation technologies (e.g. *KML* data used by *Google Earth* or *Matplotlib* (library for *python*)). Additionally, it could be combined with databases of cities and landmarks for better convenience in the usability.
- Error checking conditions for the parameters entered by the user.

- Low-level routines to automatically query the hardware, in order to obtain the number of physical and logical processors available, and also to know if *Hyper-Threading* is enabled or not. Based on these informations, the program would decide in runtime, the best parallel approach for the host hardware.

There is a very high level of data partitioning when calculating radio losses for a given area. Therefore, as stated in the previous section, further research shoud be conducted on developing a *CUDA* implementation for this purpose.

Appendix

Table 5.1: Computation Times and Times per Point of serial program

Area Dimension (km)	Computation Times (ms)	Time per Point (ms)
18*18	980.75	0.025
36*36	4207.00	0.026
54*54	10156.50	0.028
72*72	20285.00	0.032
90*90	35937.75	0.036
108*108	54765.25	0.038

Table 5.2: Computation Times of *OpenMP* "sections" approach

Number of Points	Area Dimension (km)	Computation Times (ms)	Speed-Up
1440000	108*108	9879.00	5.543603
640000	72*72	3632.75	5.583924
160000	36*36	781.50	5.383237
40000	18*18	243.00	4.036008

Table 5.3: Computation Times of *OpenMP* “parallel for” approach

Threads	Area Dimension (km)	Computation Times (ms)	Speed-Up
2	108*108	28410.50	1.93
2	72*72	10523.25	1.93
2	36*36	2152.75	1.95
2	18*18	523.75	1.87
3	108*108	22707.00	2.41
3	36*36	1695.50	2.48
4	108*108	18859.50	2.90
4	72*72	6765.50	3.00
4	36*36	1390.50	3.03
4	18*18	359.00	2.73
5	108*108	16980.50	3.23
5	36*36	1261.50	3.33
6	108*108	15449.25	3.54
6	72*72	5758.00	3.52
6	36*36	1269.25	3.31
6	18*18	387.00	2.53
7	108*108	14636.50	3.74
7	72*72	6043.00	3.36
7	36*36	1765.50	2.38
7	18*18	539.25	1.82
8	108*108	16539.00	1.72
8	72*72	8605.00	1.22
8	36*36	2215.00	0.97
8	18*18	597.75	0.88

Bibliography

S. Bai and D.M. Nicol. Acceleration of wireless channel simulation using gpus. In *Wireless Conference (EW), 2010 European*, pages 841 – 848, April 2010.

K. Chamberlin and R. Luebbers. An evaluation of Longley-Rice and GTD propagation models. *Antennas and Propagation, IEEE Transactions on*, Nov. 1982.

J.J. Egli. Radio propagation above 40 mc over irregular terrain. *Proceedings of the IRE*, 45(10):1383 – 1391, Oct. 1957.

L. Foore and N. Ida. Path loss prediction over the lunar surface utilizing a modified Longley-Rice Irregular Terrain Model. Technical report, National Aeronautics and Space Administration, July 2007.

G.A Hufford, A.G. Longley, and Kissick W.A. A guide to the use of ITS irregular terrain model in the area prediction mode, April 1982.

C. Kelley and T. Williams. Gnuplot, Version 4.4.3. Retrieved from: `http://www.gnuplot.info/`.

A.G. Longley and P.L. Rice. Prediction of tropospheric radio transmission loss over irregular terrain: A computer method, June 1968.

A.G. Longley, P.L. Rice, K. A. Norton, and A.P. Barsis. Transmission loss prediction for tropospheric communication circuits, Jan. 1967.

J.A. Magliacane. Splat!, Version 1.4.0 (2011). Retrieved from: `http://www.qsl.net/kd2bd/splat.html`.

National Aeronautics and Space Administration, and National Geospatial-Intelligence Agency. Shuttle radar topography mission (SRTM), Version 2.1. Retrieved from: `http://www2.jpl.nasa.gov/srtm/`.

NVidia. NVidia CUDA C Programming guide, Version 3.1. (2009). Retrieved from: `http://developer.nvidia.com/nvidia-gpu-computing-documentation`.

U.S. Department of Commerce NTIA/ITS. ITM - Irregular Terrain Model. Retrieved from: `http://flattop.its.bldrdoc.gov/itm.html`.

OpenMP Architecture Review Board. OpenMP application programming interface, Version 3.0 (2008). Retrieved from: `http://www.openmp.org/`.

Y. Pessach. *Juice Up Your App with the Power of Hyper-Threading.* Microsoft Developer Network Magazine, June 2005. Retrieved from: `http://msdn.microsoft.com/en-us/magazine/cc300701.aspx`.

J.S. Seybold. *Introduction to RF Propagation.* John Wiley & Sons, Inc., 2005.

S.E. Shumate. Longley-Rice and ITU-P.1546 combined: A new international terrain-specific propagation model. In *Vehicular Technology Conference Fall (VTC 2010-Fall), 2010 IEEE 72nd*, Sept. 2010.

SoftWright. Terrain analysis package, Version 6.0. Retrieved from: `http://www.softwright.com/faq/support/prop_longley_rice_settings.html`.

Y. Song and A. Akoglu. Parallel implementation of the Irregular Terrain Model (ITM) for radio transmission loss prediction using GPU and Cell BE processors. *Parallel and Distributed Systems, IEEE Transactions on*, 22:1276–1283, Aug. 2011.

Y. Song, J.A. Rudin, and A. Akoglu. Parallel implementation of Irregular Terrain Model on IBM Cell Broadband Engine. In *Parallel Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, pages 1–7, May 2009.

X. Tian, A. Bik, M. Girkar, P. Grey, H. Saito, and E. Su. Intel OpenMP C++/Fortran compiler for Hyper-Threading technology: Implementation and performance. *Intel Technology Journal Q1.*, 6, 2002.

V-Soft Communications. Probe 4, Version 4.2. Retrieved from: `http://www.v-soft.com/web/demo.htm`.

M. Weiner. Use of the Longley-Rice and Johnson-Gierhart tropospheric radio propagation programs: 0.02-20 GHz. *Selected Areas in Communications, IEEE Journal on*, 4(2):297 – 307, Mar. 1986.