THE UNIVERSITY of ADELAIDE

School of Chemistry & Physics

3-D Magnetotelluric inversion using Hadoop MapReduce

by

Jake Brown

Supervisors:
A/Prof. Murray Hamilton, Dr. J. Craig Mudge

A thesis submitted towards the degree of
Bachelor of Science (Honours) - High Performance Computational Physics
at
The Faculty of Sciences
The University of Adelaide

November, 2012

This work contains no material which has been accepted for the award of any other degree or diploma in any University or other teritary institution and, to the best of my knowledge and belief, contains no material previouusly published or writen by another person, except where due reference has been made in the text.

I give my consent to this copy of my thesis, when deposited in the University Library, being available for loan and photocopying.

SIGNED: .....

DATE: .....

Abstract

As a geophysical imaging technique, magnetotellurics (MT) has shown great promise. Currently, the usefulness of the information we can extract from raw MT sensor data is limited in two ways. First, prohibitive running times of inversion limit the ability to quickly obtain useful data from on-site measurements. A parallel implementation somewhat overcomes this. Second, the availability of computing resources limits the ability of geophysicists to run these calculations on demand, without competing for time on existing high performance computing (HPC) infrastructure.

The emergence of cloud computing provides convenient resources, available on demand, and is ideally suited to problems which can be expressed in a trivially parallel manner. Exploiting different levels of parallelism inherent in the MT method allows such problems to be expressed in a trivially parallel structure named *map-reduce*, based on the fundamental *map* and *reduce* operations from functional programming.

We have explored using an existing implementation 3D MT inversion, expressing this in a map-reduce structure, and running it on an open source implementation of map-reduce called *Hadoop MapReduce*.

Acknowledgements

First and foremost I would like to thank my supervisors Craig Mudge and Murray Hamilton for their help and support throughout the year.

Additionally, I would like to acknowledge the assistance and expertise provided by members of the Geophysics department at the University of Adelaide; including Graham Heinson and in particular Stephan Thiel.

Finally, the assistance provided by Bradley Alexander has been of great help; and he has been an interesting source of information for innovative research pathways.

Contents

Chapter 1

Introduction

Magnetotellurics is a valuable passive geophysical exploration technique for determining the conductivity of materials below the surface. While data collection is relatively simple and cheap, a computationally intensive 'inversion' process is required in order to extract the most useful information from this recorded data. In practice, this can limit the practicality and adoption of this technique by geophysicists; both in the field and in research.

The primary goal of cloud computing is to provide remote access to computing resources on demand. This has the potential to provide easy, cheap and fast access to resources for running 3D MT inversion, but several hurdles must first be overcome. To illustrate this, we will look at the two main issues:

1. The long running time of 3D inversion is a prohibitive factor; cloud computing provides access to resources, but to be able to utilize these fully and in a cost effective manner we need to design our program to run in parallel across many machines. This presents its own unique challenges.

2. 3D MT inversion should be implemented in such a way that geophysicists are able to use it, without specialized computing skills or extensive training on the computers being used.

Both of these issues have been the focus of a collaboration between Geophysics and Computer Science at the University of Adelaide, and will be discussed in Chapter 4. They have been somewhat resolved, but in a very restrictive manner. The inversion algorithm has been successfully parallelized to address the first goal, and a web application was deployed to manage submission of jobs, allocation of cloud resources, and viewing of results from inversion.

Due to the huge variation in running time and memory requirements for different data-sets, we may wish to select our computing resources to suit the problem at hand. This might include utilizing multiple cores on a single high performance machine, a cluster of high performance machines, or a cluster of standard desktop machines, colloquially known as 'beige boxes'. Additionally, these resources may located locally, or we may wish to use the virtual machines offered by one of the public cloud computing providers.

This is not possible with the current implementation without extensive modifications for each use case, which requires a specialty knowledge.

The goal of our research is to improve this by implementing 3D inversion in such a way that it is more portable across different types of hardware, without specialized knowledge of parallel computing.

In Chapter 2 we shall discuss some motivation for the use of magnetotellurics when compared with other geophysical exploration techniques. We will then introduce the required mathematical background for the magnetotelluric method, and detail the *forward modelling* process required to simulate the Earth's electromagnetic response. We will then introduce will the important concept of *inversion*, and the '3-D data-space Occam inversion' algorithm developed by W. Siripunvaraporn, and a freely available implementation of this called 'WSINV3DMT' which will be the basis for our research.

In Chapter 3 we shall introduce cloud computing in some detail. As outlined above, we shall discuss the need for portability across different cloud providers and other infrastructure, and the need for a parallel computing framework; which shall also be defined. We will then introduce the *map-reduce* computational model, and *Hadoop MapReduce* as the framework and open source implementation of map-reduce that we shall be using.

In Chapter 4, we detail the results from the collaboration between Geophysics and Computer Science mentioned above. We then delve into the 3D inversion code package WSINV3DMT in Chapter 5, and analyse which parts of the code are the most computationally demanding. This will act as some motivation towards our approach, but is also valuable for suggesting directions for future research.

In Chapter 6 we detail the steps required to program our implementation of 3D inversion, using the parallel computing framework Hadoop. The ability to run computationally demanding Fortran code within the Hadoop framework, written in Java, is new, and much of the information here could significantly benefit future endeavors to run distributed trivially parallel computations in native/compiled languages within Hadoop. We then show the results of this implementation from a technical perspective, essentially confirming that the output is identical to the well tested serial implementation WSINV3DMT

In Chapter 7 we run the 3D inversion on Hadoop using a new data set. The data set chosen is from the Carrapateena region in South Australia, and has been chosen because it presents an opportunity to explore a data set that has not previously been inverted. We analyse this data set using a variety of 'models' which determine the resolution of the final output, and the associated running time. Previously, this kind of analysis using cloud computing services would be at a significant cost, and would not have been viable. Due to being able to utilise multiple cores on a single machine, we are now able to fully utilize available computational power in order to run this calculation in the shortest possible time, and within a reasonable budget.

In Chapter 8 we present our conclusion, and a discuss other ways Hadoop can be used to improve 3D inversion. In particular, paper [2] shows that Evolutionary Computing methods show great promise for 3D MT inversion. These methods are inherently parallelizable, and can benefit from using Hadoop MapReduce; perhaps even more than the work we have examined in this paper. We also discuss how our work can be used to run other computationally intensive scientific calculations on Hadoop.

Chapter 2

The magnetotelluric method

In Section 2.1, we shall briefly explore the main techniques used in geophysical exploration. Magnetotellurics (MT) will be introduced in Section 2.2, and we shall explore the advantages and disadvantages of MT when compared to several other popular geophysical imaging techniques. We will continue by examining the experimental set-up of a typical MT field campaign.

Then, in Section 2.3 we shall outline the necessary theory underlying the MT method, including initial data processing, and introduce the concept of MT inversion.

In 2.4 we will discuss 3-D MT inversion in further detail, introducing the 'data-space Occam inversion'. We will then introduce the important 3-D inversion code package WSINV3DMT, which will be the basis for our research.

Finally, in Section 2.5 we shall briefly discuss verification of the MT method; both in general, and for given field campaigns. We shall also discuss a process known as joint-inversion, which attempts to use other geophysical information *in* the MT inversion process, and not simply for verification.

2.1 Geophysical exploration

Rocks and minerals below the surface vary in density, electrical conductivity, polarisablity, acoustic wave propagation velocity and acoustic properties, among others. These variations permit a wide range of imaging techniques to be used for determining subsurface structure and details, and can be seen to belong to three main disciplines.

2.1.1 Seismology

Seismic waves are elastic, mechanical matter waves of propogating energy. The speed at which they travel is determined by the *acoustic impedance* of the medium in which they are traveling. At subsurface boundaries between mediums of different acoustic impedance, part of the energy is reflected from the boundary, while part refracts and continues to propogate through the new medium. Seismolological methods make use of both of these behaviours to determine subsurface structure.

Reflection seismology is generally accepted to provide the best spatial resolution, and

most information about the subsurface [6]. This an *active* seismological technique, meaning an energy source or transmitter is required. At its simplest, reflection seismology involves using controlled energy sources such as explosions and vibrations to generate seismic waves, and measuring times taken for the reflected part of the wave to be detected at a number of receivers at the surface. This information is used to reconstruct the subsurface structure in terms of its acoustic impedance.

Active seismic imaging as above is largely confined to the crust and mantle lithosphere[22], regions up to 120km deep. Passive seismology does not require a transmitter, and usually exploits seismic energy from earthquakes, and is capable of imaging structure at a variety of scales, from the shallow crust (5-75km) to the whole Earth[22].

2.1.2 Gravitational

Gravitational exploration is another passive technique. Equipment is used to measure the local gravitational fields, and this information can be related to the density of the materials below the surface. The spatial resolution here is usually poor in comparison to reflection seismology; although it has the advantage of being passive - no expensive or environment-disturbing energy sources are required.

2.1.3 Electromagnetic

Electromagnetic imaging encompasses any technique using electromagnetic radiation to image the subsurface. These methods are primarily used to measure electrical conductivity and polarization, and with some knowledge of different materials, other properties such as density can be deduced from this. *Magnetotellurics* comes under this classification, and is the focus of our research. *Ground penetrating radar* also comes under this classification, and is an active technique, whereby energy in the microwave band (UHF/VHF frequencies) is emitted into the earth from a transmitter, and detectors record the reflected signals at the surface.

2.2 Introduction to the magnetotelluric method

Magnetotellurics is in most cases[1] a passive imaging technique; meaning, as with the passive techniques mentioned above, that no power is used for generating source fields. Naturally occuring, time varying electromagnetic fields penetrate the earth, with penetration depth dependent on their frequency. These fields induce currents called 'telluric' and 'eddy' currents in conductive materials below the surface. These currents generate secondary fields, and we use the relationship between the **E** and **B** fields as measured at the surface, to determine the subsurface resisitvity.

We shall now look at the experimental setup in Section 2.2.1, and the pre-processing of data required after a field-survey.

[1] Controlled source electromagnetic is a notable exception

2.2.1 Experimental setup

Initial processing of field data

A typical experimental set-up is as shown in figure 2.1. The electric potential difference between the N-S and E-W facing electrodes is usually on the order of millivolts, and is measured by a simple voltmeter. The local magnetic field in those orthogonal directions is measured by similarly oriented magnetic sensors [2], and is usually within the range of a few nanoteslas, to a few hundred nanoteslas during periods of intense solar activity.

The raw data from the voltmeter and magnetic sensors is recorded with a sample rate of 1000Hz by a logging device. The logging device may be left to collect data over a period of several days or weeks, so the internal clock is regularly calibrated via a gps link.

We then transform the data into the frequency domain, where it is represented as a linear response function called the impedance tensor $\mathbf{Z}$. In the absence of noise, and with precise data, this may be written:

$$\mathbf{E} = \mathbf{Z}(\omega)\mathbf{B} \tag{2.1}$$

where $\mathbf{E}$ and $\mathbf{B}$ are two-dimensional vectors containing the horizontal electric and magnetic field components at a specific site and frequency. Because the response is linear, the power present at a particular frequency is not important, and the impedance tensor can be normalized and made time-independent.

Field measurements contain noise, so in practice equation (2.1) does not hold exactly, and it becomes necessary to use a statistical method to approximate the impedance tensor and its uncetainty. We use a method called Bounded Influence Remote Reference Processing (BIRRP), presented by Chave et al in [4]. BIRRP attempts to simultaneously limit the influence of both 'outliers' (unusual electric field data) and 'leverage points' (unusual magnetic field data) to produce a reliable magnetotelluric response function. Full details of this process can be found in [4]; all data-sets that we encounter in this paper have already been processed using this.

[2] In our case the magnetic sensors were induction coils, but *fluxgate sensors* are also used for longer period electromagnetic fluctuations due to physical limitations of induction coil based sensors.

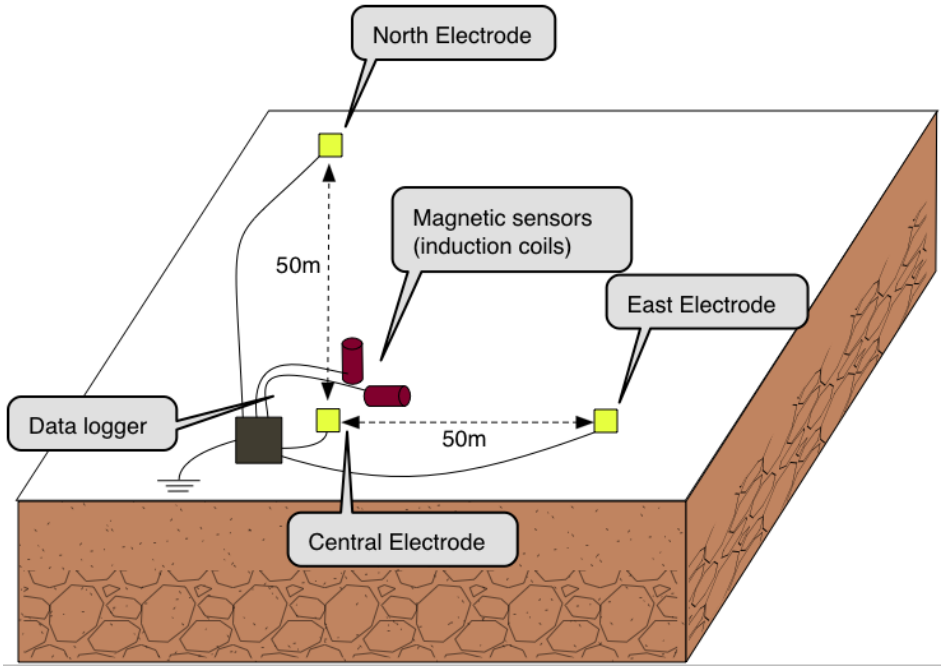


Figure 2.1: Experimental set-up of equipment for measuring raw data.

2.3 MT theory

We will now detail some necessary background on the Physics behind magnetotellurics. At the end of this section we shall briefly talk about inversion, but a detailed discussion of the algorithms used will be saved for the next section.

2.3.1 Skin depth

The penetration depth of Transverse Electric (TE) and Transverse Magnetic (TM) mode waves into a conductor is given by the simple skin depth equation from classical electrodynamics:

$$\kappa = \omega\sqrt{\frac{\epsilon\mu}{2}}\left(\sqrt{1+\left(\frac{\sigma}{\epsilon\omega}\right)^2}-1\right)^{1/2} \tag{2.2}$$

Where σ is the conductivity, ϵ is the magnetic permittivity, ω is the angular frequency of the electromagnetic radiation, and $\delta = \frac{1}{\kappa}$ is the skin depth.

This means that EM fields are attenuated to a value of e^{-1} of their surface amplitude at depth $\delta(T)$. We make an approximation, and consider the skin depth to be the penetration depth for electromagnetic radiation [24].

2.3.2 Sources

The magnetotelluric method makes the assumption of vertically incident TE and TM plane waves. The plane wave assumption is important, as we assume a common source at all detection stations.

Figure 2.2 outlines the frequencies commonly used in Magnetotellurics, with the corresponding depth range and geological region. The terestrial sources mentioned usually take the form of lightnight strikes. It is imporant that these lightning strikes are non local, in order to avoid saturation of the detectors, and to satisfy the plane wave assumption. The solar wind, consisting primarily of protons and electrons, interacts with the Earth's magnetic field, causing streams of charged partices to be deflected in opposite directions, establishing an electric field. Variations in the intensity of the solar wind, as well as complex interactions between the magnetosphere and ionosphere, generate fluctuations in this field, which eventually becomes our source field.

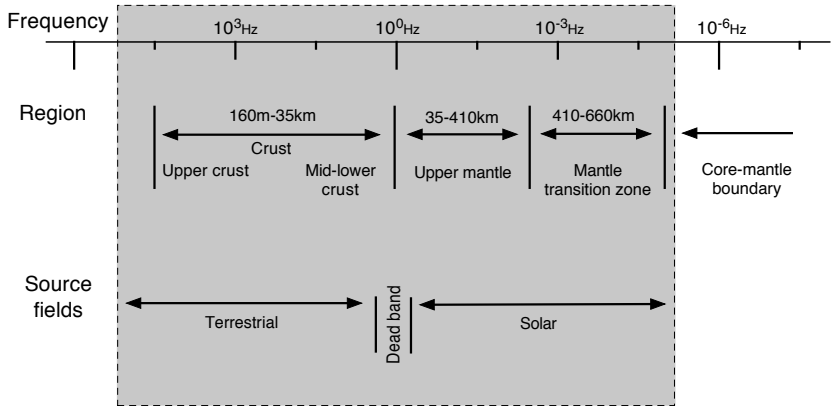


Figure 2.2: Frequency range used in Magnetotellurics (the shaded region), with the corresponding skin depth, geological region, and magnetotelluric source location. Terrestrial sources originate from the Earth. The lower frequency solar sources are primarily due to interactions of the solar wind with the magnetosphere and ionosphere. The dead band indicates the region of lower electromagnetic activity between the terrestrial and solar sources, making it difficult to image this area.

2.3.3 Apparent resistivity

We can learn a lot from simple mathematical relationships between the impedance tensor and its components. Apparent resistivity $\sigma_a(\omega)$ is a quantity defined as the average resistance within an equivalent uniform half-space. We can determine the apparent resistivity from the impedance tensor[33]:

$$\sigma_a(\omega) = \frac{\mu}{\omega}|\mathbf{Z}(\omega)|^{\mathbf{2}} \tag{2.3}$$

The radiation has a finite penetration depth, as given by the skin depth (2.2), so we can consider the half space to be the region between where the impedance tensor is measured, and the skin depth. By definition, the apparent resistivity, calculated from the components of the impedance tensor of a single frequency at the surface, does not give us any depth resolution; it is simply the average resistivity within the region defined by the surface and the skin depth.

In a MT survey, measurements at a discrete set of frequencies, with skin depths corresponding to the depths we wish to examine, are extracted from the recorded measurements as described above. Since different frequency components of $\mathbf{Z}$ can be used to examine details

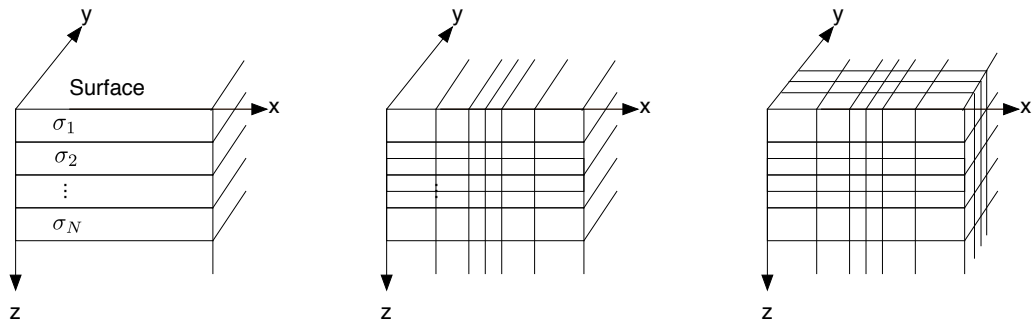
at different depths, we can use this information to generate a more useful 'model' of the subsurface resistivity; our exact approach depends on the dimensionality of the problem.

2.3.4 Forward modelling

Forward modelling involves simulating the Earth's electromagnetic response; but first we must define a model.

In general, a model is some data structure containing a set of resistivity values which describe the subsurface structure. The exact form of this model can further vary depending on whether we wish to work with a one, two, or three dimensional representation.

Examples of one two and three dimensional models are given in D resistivity models are given in Figure 2.3. For simlicity, these models are defined by a single set of values for block size in each of the x,y,z directions; which means that the block size in one axis with respect to the others is constant.



(a) 1-D model (b) 2-D model (c) 3-D model

Figure 2.3: Examples models of 1, 2 and 3 dimensions.

Forward modelling takes a model, and simulates the Earths electromagnetic response to arrive at a theoretical impedance tensor defined at the surface. This is done independently for each frequency ω.

For example, considering the one dimensional model in Figure 2.3a. Here we have a two dimensional representation where the resistivity is constant in the horizontal direction, and is made of a discrete set of layers of different resistivity in the vertical direction. We take that model, and starting at the bottom layer, considering below this to be a uniform half space, recursively calculate the earth's electromagnetic response for each horizontal layer until we arrive at the theoretical response function at the surface.

2.3.5 Inversion

The concept of forward modelling immediately suggests a simple algorithm for MT inversion, which we detail briefly now.

We wish to find a model $\mathbf{m}$ that when forward modelled, produces a theoretical response, matching our observed response to a precision defined by some *misfit*. We also wish the model to satisfy some condition of *smoothness*, in order for it to be considered a realistic representation of the subsurface geology. The goal of this smoothness is to ensure there are not large differences in resistivity between neighbouring blocks in our model.

A simple algorithmic outline is shown in Algorithm 1.

Algorithm 1 Iterative MT inversion algorithm outline. The number of iterations should be set to some reasonable number depending on estimated time for convergence. The 'improve model' step will be dependendent on the algorithm implementation and dimensionality.

```
while number of iterations not exceeded do
    Forward model to get the theoretical response ;
    if misfit of response and smoothness of model satisfy stopping conditions then
        end
    else
        improve model
    end if
end while
```

We briefly detail some advantages and disadvantages of inversions in one, two, and three dimensions:

1-D inversion In a 1-D inversion, we only consider variations in resistivity in the vertical direction. The model describing the subsurface structure will be made of layers of varying resistivity, in horizontal plane. This will only be considered when we wish to simplify some examples.

2-D inversion 2-D inversion also allows for variation in the horiontal direction. There are dangers in 2-D inversions being influenced by 3-D structures, as demonstrated with real data in [24] and [19]. According to [27], 'All studies indicate that if the data contains 3-D structures, 2-D inversion can mislead an interpretation'.

However, 2-D inversion is much less computationally demanding than 3-D inversion, and can be useful. According to [27], MT acquisitions are usually conducted along a profile, or several profiles running parallel to each other. Then, prior to running the 2D inversion, an analysis is carried out in order to determine which station data is consistent with a 2-D interpretation. 2D inversion is then performed to yield the cross sectional profile.

Three dimensional inversion Three dimensional inversion is preferred over 2 dimensional inversion, in order to adequately explain important features present in field data sets from geologically complex regions. It solves the issues inherent in 2D inversion mentioned above; however it is very computationally expensive, and not widely used for that reason. This is the area that we wish to improve, and will be the focus of all of our research. We will discuss this in more detail in the next section.

2.4 3D MT inversion

In this section we will examine 3-D inversion in more detail, and describe the inversion algorithm that is the focus of our research.

Magnetotelluric inverse problems usually have many more model parameters than they do data measurements. Data-space methods formulate the inverse problem in terms of the

data, rather than the number of model parameters. Representing the problem in this way can reduce both computational time and memory requirements, due to the smaller matrices that must be stored and operated upon. We present a detailed summary of relevant theoretical developments in 3-D inversion in Appendix D.1; interested readers should refer to this now.

Siripunvaraporn et al publish a 3-D Occam data-space approach in [30]. This has been released in a Fortran code package named WSINV3DMT, and was made freely available to the MT research community in 2006 [29]. Instructions for obtaining the code are available in [26]; and this code is our focus. There are other 3-D inversion code packages available, however we will save this discussion for Chapter 8, where we discuss directions for future research.

2.4.1 3-D data-space Occam inversion: WSINV3DMT

We use a 3-D Occam inversion, with the transformation to data-space to make calculations more efficient. This was published in [30], and we will explain the key parts here.

Occams inversion seeks the smoothest, or minimum norm, model subject to an appropriate fit to the data [5]. So the norm of the model is representative of the smoothness parameter discussed earlier. The root-mean-square of the pointwise difference between the response and the forward modelled model is representative of the misfit; as is common in the literature, we will simply refer to this as the RMS.

The inversion algorithm is detailed in Algorithm 2. The goal of each iteration is to find the minimum norm model subject to a given RMS target. Stage 1 computes the expensive sensitivity matrix and Cholesky decompositions required for Stage 2. Stage 2 happens in two 'phases'; Phase I finds the model with the lowest RMS, and Phase II starts with this, and perturbes this to find the model with the lowest norm. Phase I has the effect of achieving a better fit to the data, and Phase II attempts to remove extraneous data resulting from overfitting.

Norm and covariance

The model norm is defined as $\mathbf{m}^T {\mathbf{C_m}}^{-1} \mathbf{m}$. The model covariance $\mathbf{C_m}$ required to calculate this, is defined algorithmically inside the WSINV3DMT code and has not been modified in our work. As can be seen in the code [26], and described in [25], it allows for a general specification of prior information[25], taking into account the length scale (below), and other parameters such as known physical structures. This allows the norm to be computed while taking into account these details.

The data covariance matrix $\mathbf{C_d}$ is defined in a simlar manner, taking into account information relaing to the reliability of station data. This allows the user to specify which stations may produce less reliable information, due to interference, for example.

Smoothness and the τ parameter

We define a smoothing parameter τ, and a related *model length scale*. These parameters impose restrictions on the model covariance matrix $\mathbf{C_m}$ which is used in updating our model,

Algorithm 2 Algorithm for the data-space Occam inversion implemented in WSINV3DMT. $\mathbf{C_d}$ is the data covariance matrix, which acts as a weighting to lower the effect of data from less reliable stations.

Initialise with an initial model guess $\mathbf{m} = \mathbf{m}_0$, and input data $\mathbf{d}$.

1. Forward model and compute RMS (misfit) from model
2. Start Outer loop iteration k:

   Begin stage 1:

   3.1 For $i = 1$ to $N_s \times N_m \times N_p$
   Call forward modelling $\mathbf{F}[\mathbf{m_k}]$ to form sensitivity matrix $J_{k,i}$ for data i
   End

   3.2 Compute $\mathbf{d}_k = \mathbf{d} - \mathbf{F}[\mathbf{m_k}] + \mathbf{J_k}(\mathbf{m}_k - \mathbf{m}_0)$

   3.3 Compute $\Gamma = \mathbf{C_d}^{-1/2}\mathbf{J_k}\mathbf{C_m}\mathbf{J_k}^T\mathbf{C_d}^{-1/2}$

   Begin stage 2:

   3.4 For various values of λ

   3.4.1 Compute representer matrix

   $$\mathbf{R}_k^\lambda = [\lambda\mathbf{I} + \mathbf{\Gamma}_k^\lambda]$$

   3.4.2 Use Cholesky decomposition to update trial model:

   $$\mathbf{m}_{k+1}^\lambda = \mathbf{C_m}\mathbf{J}_k{}^\mathbf{T}\mathbf{C_d}^{-1/2}[\mathbf{R}_k^\lambda]^{-1}\mathbf{C_d}^{-1/2}\mathbf{d}_k + \mathbf{m}_0$$

   3.4.3 Forward model and compute RMS (misfit) from model

   3.4.4 **if Phase I** Once all λ have been tried, choose model with minimum RMS and begin Phase II.

   if Phase II Once all λ have been tried, choose model with minimum norm and break loop.

   End

   3.5 Exit when misfit is less than desired level with minimum norm

   End WSINV3D outer loop iteration

as can be seen in Alogirithm 2. By way of this, the τ parameter and length scale implicitly restrict the overall smoothness, and the relative smoothness of the model in each direction (x,y,z), respectively. We supply these parameters at the start of the inversion, and they remain fixed throughout the inversion.

We make a clear distinction between this τ parameter, and the model norm. The τ parameter, and the associated length scale, act as a constraint on allowable models, restricting large differences in resistivity between successive blocks (low smoothness) in each of the three directions. The model norm is essentially a second objective function, along with primary objective, RMS, and is an overall measure of the model smoothness.

Using a higher τ value can help avoid convergence to a local optimum; in many cases this may be a model with a low RMS, but a low smoothness, indicating that it is not a geologically useful model.

WSINV3DMT

An implementation of the algorithm described above is provided by W. Siripunvaraporn and is freely available. This implementation is approximately $20,000$ lines of Fortran 77, and while it produces reliable output, it does not run any calculations in parallel. This is what we wish to improve. Most of the Fortran code will be left untouched in our implementation; as will the input and output data formats, so it is worth explaining it here in some detail.

Firstly, note that each time forward modelling is applied, it is computed independently for each frequency. Since this forward modelling is a dominant part of the calculation, running these in parallel can speed up this serial algorithm significantly. This the approach taken by C3L and discussed in Chapter 4.

It is also important to understand the data format required by the Fortran code, so we will detail that now.

Response file (input) This file contains the information from the station data, after it has gone through the filtering process and converted to the impedance tensor in frequency space. This includes: the number of stations, number of periods, and number of responses[3]; along with the station locations, and the actual responses (impedance tensor components) for each station.

The data file also contains an 'error' section for each impedance tensor measurement. This allows us to reduce the weightings of stations that may have unreliable data, for example.

Model file (input and output) The model file is comprised of three sections:

1. The first line details the number of discrete blocks N_x, N_y, N_z, with directions explained below.

2. The next section contains $N_x+N_y+N_z$ floating point numbers, corresponding to the sizes of the blocks in their respective directions.

[3]The number of responses is either 4 or 8. Corresponding to the real and imaginary parts of the off diagonal elements of the impedance tensor (Z_{xy} and Z_{yx}, four components total); and the real and imaginary parts of the entire impedance tensor (8 components total).

3. The final section contains the resistivity values for each block. Each of the $N = N_x \times N_y \times N_z$ resistivity values may be defined independently, or there might just be one value here, indicating that all blocks have the same resistivity - this is is what we call a uniform half-space.

In the 3D MT inversion code WSINV3DMT, and our work to follow, the model directions are defined with x in the North-South direction, with North positive, y in the East-West direction with east positive and z positive downwards. We will use the cartesian coordinates (x, y, z) and the geographical coordinates interchangeably. In the horizontal plane, the centre point of the model is the origin of the models coordinate system, and the model is symmetric about its centre in the North-South and East-West directions.

Model files used for input (i.e. initial models) and those provided as output from the inversion process, are identical. The final section of the model which contains the resistivities for each block, may be in one of two formats:

The first is where there is a different resistivity value provided for each block. The second is the case when the model defines a uniform half space - each block has the same resistivity.

In cases where little is known about the subsurface geology, a uniform half-space is preferred as the initial model.

In the case where the model is generated by WSINV3DMT, as the output from an inversion process, the first line will also include the RMS value (misfit) of this models theoretical response, in comparison to the input data.

Since input and output models are identical, we can use the output from one inversion as input for another inversion. This will be explored in Chapter 7..

Response file (output) The program generates a Response file for each output model, which contains the simulated response data for that model. This is in the same format as the input data file, which is the actual response, and this makes it easy to calculate the RMS of the pairwise differences between the two.

Other files

In addition to the above, several other files are used:

- A 'startup' file contains information required to start the inversion. This includes max number of iterations, target RMS, desired smoothness etc.
- It is also possible to supply a file that specifies parts of the model which may be 'frozen'. This might represent known resistive bodies that we wish to remain fixed in the model.

2.5 Verification of the magnetotelluric method

Magnetotelluric inversion is an 'ill posed' problem; which means that (a) the solution is not unique, (b) the solution is dependent on the initial conditions. This is largely due to the fact that in a 3-D MT inverse problem, we usually have far more model paramers than data

parameters, which can easily result in overfitting of the data; hence the introduction of the smoothness parameter.

For any particular inversion, even after we arrive at a reasonable solution in terms of RMS and smoothness, the output must stx be subjected to a deal of interpretation based on a-priori knowledge of the geological structures below the surface. Discussions with Stephan Thiel made it clear that most papers published using MT results, these days, attempt verification via some other method. The ability to verify this will obviously be highly dependent on the physical depth that is being examined in an MT survey.

For high frequency MT studies, such as [36], with skin depths in the range of tens of metres to several hundred metres, verification may be possible via drilling boreholes. Other studies such as [14] use Geochemical data in conjunction with MT inversion results.

For greater penetration depths, different techniques must be used; including reflection seismology, or passive seismology for deep mantle studies.

One might ask for verification of the magnetotelluric inversion method in general. This is best thought of in the context of specific MT inversion algorithms, where we can determine whether they can correctly inerpret data in real and synthetic data sets, as discussed in 2.3.5.

2.5.1 Joint inversion

Although joint inversion is not technically a method for verifying MT data, it is worth mentioning it briefly in the context of our discussion so far. The goal of joint inversion is to reduce the set of acceptable models, i.e. reduce the ill-posedness of the problem, by combining several geophyiscal methods in a single inversion scheme, and requiring the model to explain all data simultaneously [16]. The paper [16] explores two different approaches to the coupling of MT, gravity and seismic refraction data, to form a 3-D joint inversion scheme. They report improvement of inversion results in both cases.

Chapter 3

Cloud computing

We will now discuss some important aspects of cloud computing. We will start by *defining* cloud computing, then continue by looking at more detailed concepts underlying the hardware and software abstraction which motivates our approach.

This chapter necessarily uses some common terminology from Computer Science and Java programming. While we aim for a gentle introduction, readers unfamiliar with some terminology should check the glossary in Appendix A.

3.1 Definition

The NIST supplies the following definition for Cloud Computing:

> 'Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction...'[15]

There are public cloud services, such as Amazon Web Services (Amazon), Windows Azure (Microsoft), and Google Cloud Platform (Google). In Australia, there is also the new 'National eResearch Collaboration Tools and Resources' (NeCTAR) project[12], a $47 million national research infrastructure funded by the Federal Government, which as one of its four sub-projects contains a Research Cloud. At the present time, the NeCTAR research cloud provides free[1]access to researchers at any academic institution in Australia.

3.2 Layers of abstraction

Cloud providers differ in the way in which they provide access to resources. This is often illustrated in a diagram similar to figure 3.1; where we detail the layers available on the Amazon Web Services cloud.

Starting from the top, an 'Application as a Service' is a program which is accessible remotely through a web browser, that performs a particular function. The example given

[1]Free access is currently limited to one virtual machine per user, however extended access can be requested.

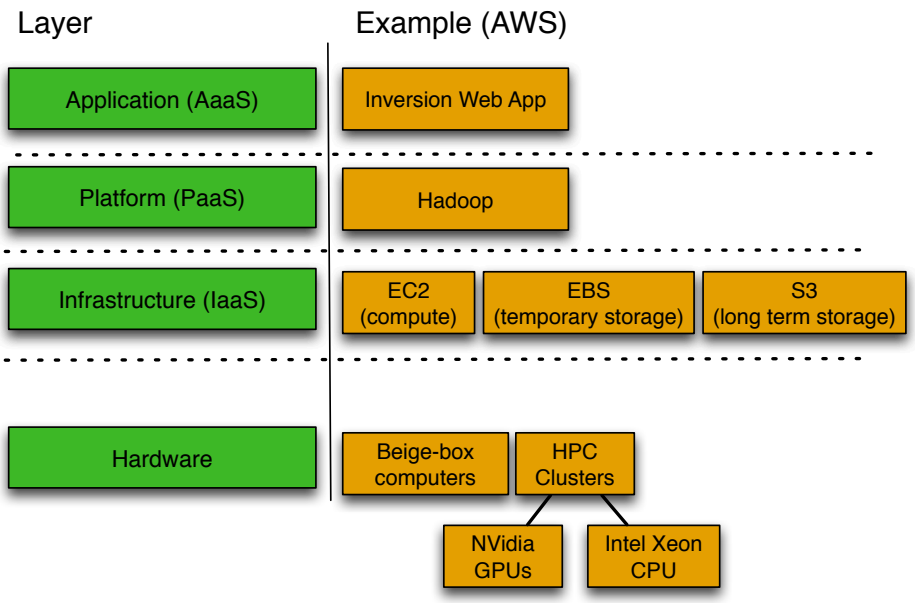


Figure 3.1: Layers of abstraction in cloud cloud computing. The Layers (in green) apply to any cloud service provider. The examples presented here (orange) are specifically relating to Amazon Web Services. We will discuss Hadoop in the next section, and the Inversion Web App later on.

in the figure refers to a web application designed to perform 3D MT inversion, that we will discuss in a later section.

'Platform as a Service' is the level that we would like to be working with. The example of Hadoop, as we shall see, is a framework on which we can design our own programs to run, without regard for the underlying infrastructure or hardware.

'Infrastructure as a Service' provides explicit access to resources in the cloud; which may be useful for deploying Applications that cannot be run within an existing Platform. For example, the Elastic Compute Cloud (EC2) allows us to start a virtual machine to which we can connect (usually via SSH) and run our programs. EBS provides access to short term storage, such as a virtual hard drive which we can attach to an EC2 virtual machine. S3 provides access to cheaper long-term storage.

In cloud computing, the hardware layer is usually abstracted away. However, Amazon web services provide information regarding their High Performance Computing clusters.

3.3 Big data and map-reduce

Big data is concerned with data sets that are so large that conventional methods of data capture, storage, and analysis do not work. For example, keeping entries consistent in relational databases, holding tens of petabytes of data distributed across the globe, is not possible[17].

The 'map-reduce' computational model was introduced by Google in the paper [8], and this documented a simple way to define data-processing computations so that they could

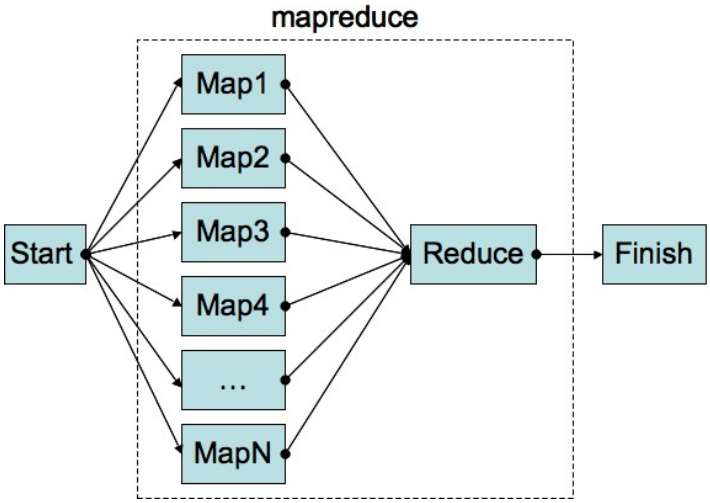


Figure 3.2: Simple map-reduce computation. We have N mappers, which means there are N different keys in the input data set. We have a single reducer, which means all Mapper outputs must have the same key. <key,value> pairs are not shown. [Need to modify diagram: Change MapN to MapperN]

be executed across many processors in parallel. Since then, much effort in industry has been dedicated towards developing this map-reduce model for big-data analysis. An open source project called *Hadoop* has emerged as the leading implementation of the map-reduce paradigm, and will be discussed in the next section.

3.3.1 Map-reduce computational model

The map-reduce computational model is based on the fundamental 'map' and 'reduce' operations from functional programming. In map-reduce, a 'map' operation is implemented by a function (called the 'mapper') that takes <key,value> pairs as input, and produces <key,value> pairs as output. A 'reduce' operation, similarly implemented by a function (called the 'reducer') takes <key,value> pairs output from the mappers, and produces <key,value> pairs as output. Usually the final output is considered to be the <key,value> pairs from the reducers.

The Chevron notation is common notation in Computer Science for representing a 2-tuple, or pair of objects, representing a key and a value. The data types of 'key' and 'value' are arbitrary. They might be String, numbers, or a binary data type. The key is usually used for organizing which mappers and reducers handle the computation; since each mapper and reducer handles a different key. The value is then used by the mappers to compute its output.

A simple map-reduce diagram is shown in Figure 3.2, illustrating a general map-reduce operation.

We can use this style of computation for handling many different types of trivially parallel algorithms. The keys might not provide meaningful information directly, but can be used to simply instruct the map-reduce framework on how many Mappers to instantiate - where we only pay attention to the number of unique keys. A simple example of using map-reduce for such a problem is shown in Figure 3.3, where we compute the scalar (inner) product of two

vectors.

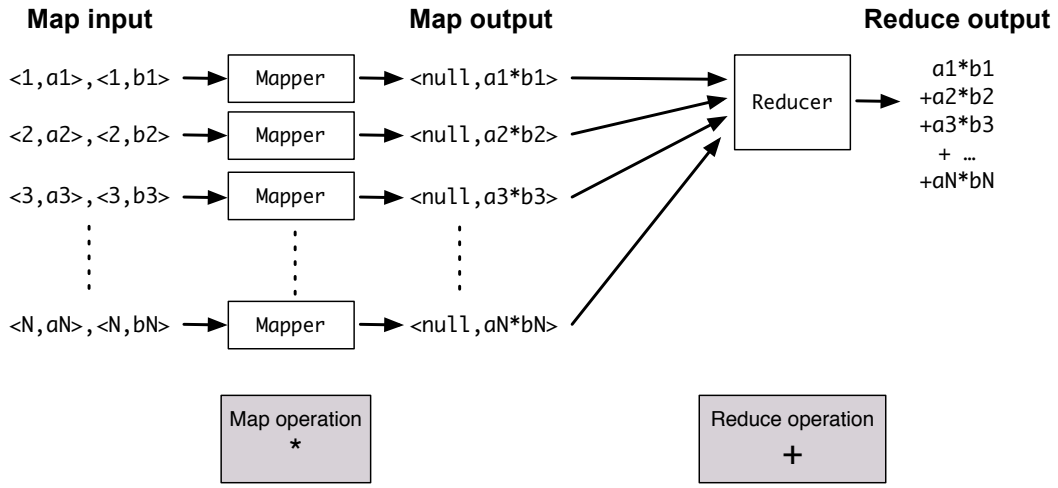


Figure 3.3: Dot product of two vectors: $(a_1, a_2, a_3, ..., a_N) \cdot (b_1, b_2, b_3, ..., b_N)$ illustrated using map-reduce. The map input is the <key,value> pair corresponding to <index,value>.

3.4 Hadoop

Hadoop is the name given to a collection of open source projects, designed for distributed high performance computing. It has been designed with 'big data' in mind, with the goal that it should run in a fault tolerant manner on ordinary desktop machines that have a relatively high failure rate.

At present, Hadoop is best known for its two main sub-projects [35]: the Hadoop Distributed File System (HDFS), a distributed file-system designed for storing large amounts of data across many machines in a fault tolerant manner; and an implementation of map-reduce, titled Hadoop MapReduce. Hadoop MapReduce is designed to work in conjunction with HDFS (although not necessarily), in such a way that the program that schedules Mapper and Reducer tasks, the 'TaskTracker', can schedule compute tasks to run on the same nodes[2]that store the data that those tasks require. This is known as 'bringing the computation to the data'. So if the data that is passed to a Mapper actually resides in HDFS, then Hadoop can schedule the computation to take place on the node where the data actually resides.

We shall primarily be focused on Hadoop MapReduce, but we will also use HDFS in order to transfer and store the data, both input and output, required for inversion. HDFS is a core component in Hadoop, so we can assume that it will be running wherever we have Hadoop MapReduce available.

[2]We will use the term 'node' to refer to the underlying hardware that runs the Mapper task. This may be a PC in an distributed Hadoop setup, or it may be a simulated node running on a single processor in a multiprocessor machine.

3.4.1 Hadoop in industry

Hadoop has seen great adoption in industry. Yahoo, for example, as of March 2009 were running 17 Hadoop clusters with a total of 24,000 nodes[35]. Facebook have for years used Hadoop MapReduce and HDFS for storing and processing huge data-sets [3].

Amazon Web Services (AWS) include a Hadoop MapReduce service called 'Elastic MapReduce'. This is designed for easily creating and configuring Hadoop clusters for running MapReduce jobs. Using Elastic MapReduce, it is possible to create a cluster containing the desired number and type of machines, specify the input data and MapReduce code, and then execute this MapReduce job, all in just a few clicks, and in a matter of minutes; with no knowledge required for network or cluster administration. Microsoft have recently released a 'Hadoop on Azure' service, providing similar functionality to AWS Elastic MapReduce.

It should be stated again that Hadoop and Hadoop MapReduce are thus far primarily targeted at 'big data', rather than strictly computationally intensive calculations, that may not require the ability to handle large amounts of data. However, it would be highly beneficial if the types of services listed above *could* be used for these types of calculations. In fact, there is nothing stopping this from happening. The main issue is one of running native programs (for example the Fortran program used for 3D MT inversion) within Hadoop. We shall resolve this in the main body of our research, but for now we shall assume we are just using one of the interpreted languages supported by Hadoop, for example Java.

The ability to utlise the vast industrial resources mentioned above is one clear benefit; we shall explore some more in the following sections.

3.4.2 Portability

Portability can be seen from the abstraction diagram in Figure 3.1. By accessing the infrastructure at the Platform level, i.e. writing our program to have its dependencies on Hadoop, we can take our program and run it on any of the Hadoop services listed above, with little modification. This enables us to take code written for Hadoop MapReduce and run it on any of the Hadoop services listed above, in addition to an in-house Hadoop cluster that we can create by installing Hadoop on local machines.

3.4.3 Fault tolerance

We define a 'task' as either a Map or a Reduce operation; these are the operations performed by Mappers and Reducers, respectively. Hadoop provides task level fault tolerance, which can be briefly explained as follows:

Each node has an individual 'tasktracker' which is in charge of running tasks on that node. All tasktrackers are contact with the 'Master Scheduler', which is a process running on the head node that assigns tasks to tasktrackers. The Master Scheduler is aware when a tasktracker becomes unresponsive, and can reassign tasks assigned to it to other tasktrackers. This is handled slightly differently for Map and Reduce tasks.

Failed Map tasks can simply be reassigned to another tasktracker and operation proceeds as normal. Map tasks run in isolation; each one executes independently and has no interaction

with any of the others, therefore it can be restarted without issue.

Reduce tasks write their output upon completion - so any already completed reduce tasks (and the Map tasks that they recieved output from) need not be run again. However, any upon failure of a Reduce task, its input is lost. Therefore, mappers whose output became input for this reduce task must be run again. The Reduce task is then run again with the newly generated input.

3.4.4 Load balancing (speculative execution)

Load balancing can be seen as an extension to the fault tolerance described above. When the Hadoop scheduler is running a series of tasks, it is possible that most of them will finish while some are still executing. These tasks may still be executing because they are more computationally demanding than the others, or the hardware of the node that they have been scheduled to run on is slower.

In these cases, a form of load balancing known as 'speculative execution' can be invoked by the Hadoop scheduler. It can duplicate the still-running tasks on nodes than have finished running their allocated tasks and are now free. The scheduler then takes the output from the one that finishes first, and terminates the othe duplicates. Again, it is the 'no outside effects', or isolation, explicitly enforced by Hadoop, that enables this to be possible.

3.4.5 Hadoop usage

We define a Hadoop MapReduce 'job' as a series of Map operations that execute in parallel, followed by a series of Reduce operation that also execute in parallel.

A Job can be programmed in a number of ways, and at different levels of abstraction. Since we have been focusing on the Amazon public cloud, we shall outline the options they have made available:

Plain Java This is the standard way of programming a Hadoop MapReduce job, supported by all Hadoop implementations.

The programmer codes a MapReduce job against the Hadoop API, which gives low-level access to all of Hadoop's mechanisms, such as access to a distributed filesystem (HDFS), and fault tolerant output channels. The Java classes are provided directly to Hadoop, or conveniently specified as an input parameter in Elastic Mapreduce.

The programmer is able to define Map and Reduce tasks by implementing the corresponding Hadoop interfaces in Java. A simple Java program, implementing the Hadoop API, can be written with a Main method that instantiates these tasks and submits them to Hadoop for computation. This is the path that we will follow, and it will be explored in more detail in Chapter 6.

Streaming Hadoop streaming allows you to define a section of code as a Mapper or a Reducer. This code is usually an interpreted language (Appendix A) such as Java or Python, but could be a compiled language, in which case a binary file would be supplied. This binary must be compiled for the specific hardware and operating system on which Hadoop is running.

The framework executes one of these Map and Reduce programs as a separate process for each Map and Reduce task, and supplies the input as stdin [3]. It then retrieves outut as stdout from each.

Streaming allows programmers to use a language they are familiar with, allowing you to rapidly develop these applications without needing to learn the Hadoop API. In addition, this could be an easy way of running legacy code in a massively parallel manner.

It does not offer us the level of control we require. From the Elastic MapReduce console, there is no easy way of chaining together MapReduce jobs.

High level Pig or Hive program Pig is a high level abstraction for programming Hadoop jobs. Its primary advantage is rapid development using a custom scripting language known as Pig Latin - a language designed for expressing massively (embarrassingly) parallel data analysis computations.

Hive is another high level abstraction, designed for data analysis on Hadoop. It allows programmers to express data-parallel calculations using an SQL-like language called HiveQL.

Neither of these options are suitable for our purposes.

[3]stdin and stdout is, in computer science, a standard protocol for supplying input and output to a process. It is usually in the form of human-readable text. For example, when running a console (command line) program, stout is the output displayed on the screen. stdin is the keyboard input.

Chapter 4

C3L parallel implementation

The Collaborative Cloud Computing lab (C3L) is an initiative established at the University of Adelaide in 2010, with the goal of exploiting cloud computing to conduct computational research in a lab without local computers[1]. Their first efforts have been focused on a collaborative effort between Computer Science and Geophysics, which we shall look at in detail now. These efforts are the basis for our work, which will be extending their parallel implementation of WSINV3DMT to run within Hadoop.

In this chapter we discuss the contribution made by C3L to 3-D MT inversion. Essentially, a parallelisable section was discovered in the WSINV3DMT code, and the program was redesigned with the ability for this section to be distributed across multiple computers. Virtual machines in the cloud were used to store the MT data and run the calculation, and a Web-Application[2]implemented in order to provide convenient access (a 'front end') to this 3D inversion program running in the Cloud.

This project was designed to run on AWS and Windows Azure, although the implementations on each were considerable different.

In the first section of this chapter we shall discuss how the WSINV3DMT code is paralelisable, which will be directly related to our project. We then detail their exact implementation, focusing on the AWS implementation as it is simpler and more closely related to the approach we wish to take.

Finally, we will discuss downfalls with the approach taken here, and the reason our research is warranted.

4.0.6 Paralellisation

As we saw earlier, foward modelling can be carried out independently for each frequency. Siri's inversion code carries out several forward modelling steps at each iteration, and it is these that are run in parallel for each frequency.

In order to do this, the logic controlling the main loop in WSINV3DMT, previously in Fortran, was implemented in a Python script. The computationally intensive parts were left

[1] Any computers being used were to be used only for connecting to various cloud service providers, including AWS and Windows Azure.

[2] 'A web application is a client/server appplication that uses a Web browser as its client program, and performs an interactive service by connecting with servers over the Internet'.[23]

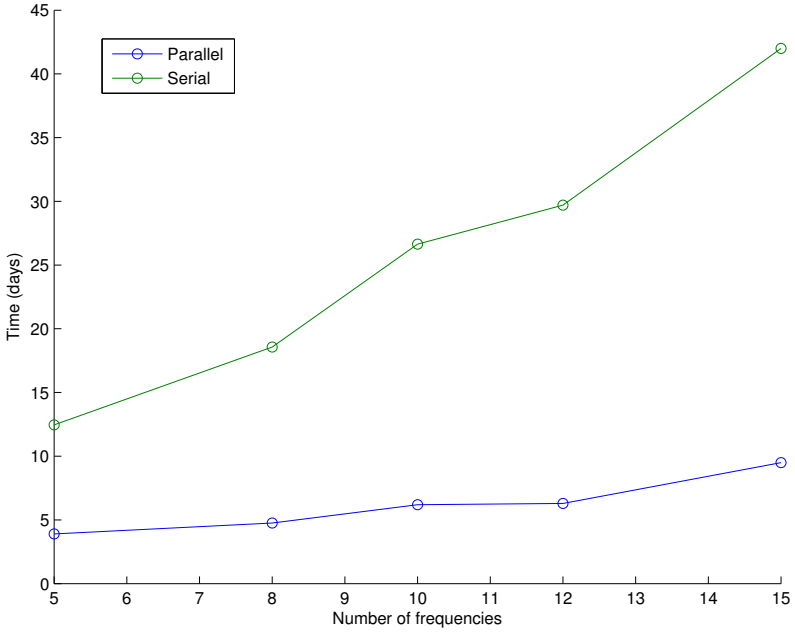


Figure 4.1: Comparison of execution time in serial (WSINV3DMT) vs. parallel (C3L) on a modest data-set, described in Appendix C. Here we carried out 5 iterations of the main loop, equivalently expressed in WSINV3DMT and the Python code used by the C3L implementation.

in Fortran, and split into 17 different source files which can be called from the Python script.

This type of parallel execution allows us to achieve speedup dependent on the number of frequencies present. A summary is shown in figure 4.1.

4.0.7 Implementation

Only two of the 17 parts of computationally intensive Fortran code can be run in parallel. These have been named Part2 and fwdcalc.

In order to run these in parallel, the Python script connects to the remote nodes via ssh, transfers the input data, then executes the Fortran executable binaries. It is assumed that these nodes are configured in advance, with the appropriate Fortran codes and libraries having been installed. These Fortran executables save their output to a file, and the Python script copies the output from each node back to the head node, which is running the Python script

This implementation is more low-level than, for example, using pure Fortran with MPI to implement the inversion in parallel. This approach was chosen primarily because it was simple and direct, without using any parallel frameworks (such as Hadoop MapReduce) which were still in their infancy at this stage. This also allowed researchers at C3L to gain experience in low level operations using infrastructure as a service (IAAS) on the cloud.

4.0.8 Web application

It should not be desirable, nor necessary, to instruct geophysicists on the details of setting up the virtual machines in the cloud and uploading files etc. in order to run a MT inversion. To fully realize a primary benefit of cloud computing - availability of resources on demand, it was necessary to provide an easy to use interface for the end-users of this software.

A web-application was created, providing an easy to use web-page where the geophysicist could upload input files, select parameters, and start the inversion. The web application would then create the required number of virtual machines in the cloud (one for each frequency) from a pre-defined virtual machine image, containing the Fortran binaries. The web application would then run the Python script, and display the output files for download once the inversion was complete. This was implemented in a web-application framework called Django, and provided a significant convenience for the end users.

4.0.9 Area for improvement

The low-level implementation mentioned above has a number of issues; the main issue being one of portability inherent in the IAAS approach. The software works sufficiently well for distributing the computation among multiple virtual machines in the cloud, but would need to be modified in order to use it to utilise the multi-core nature of the virtual machines.

Also, fault tolerance and load balancing become important as more machines are used and the likelihood of a single machine crashing increases. These features would need to be implemented manually and have not been done here.

Chapter 5

Initial code analysis

In order to improve the work described in Chapt. 4, some analysis was carried out to see which parts of the Fortran code were the most computationally intensive. These results are presented here. Siri's original serial code was used here because it gives us a simpler environment to profile and debug, without the added complications present in the parallel implementation, such as transfer time, network latency etc.

5.1 Inversion call-chart

It is claimed in [18] that the inversion calculation is dominated by the forward modelling and sensitivity matrix calculations. We should like to verify that, or at least obtain a clearer idea as to the *hot-spots*[1] in the algorithm.

We can see that the sensitivity matrix calculations dominate (sens3d), followed by calculating the representer matrix (coprept) which is also closely related to the sensitivity matrix. Forward modelling is also dominant (fwd3d). This does not exactly solidify the claim that forward modelling is a computationally dominant part of the inversion algorithm; however, the analysis above was conducted with a relatively small model file.

5.2 Forward modelling times

Instead of conducting the above analysis on a large model, which would be very time consuming (several days on a M1.Large instance), we can analyze how the time spent in forward modelling increases as model size increases. In order to do this, an artificial data set was used, and a model was adapted from a study on the Paralana region.

This model was adjusted by adding and removing cells while keeping the total model size roughly the same; essentially varying the resolution of our model. The example is shown in figure 5.2. It must be stressed that this example is artificial and designed only to confirm the assumed increase in running time. It is consistent with approximate figures found by earlier research[1] where N=1600 takes 40-50 seconds and N=64000 takes 19-20 minutes.

[1] A hot-spot is a computer science term used to refer to small sections of code that a program may spend most of its time executing. See the glossary for more details.

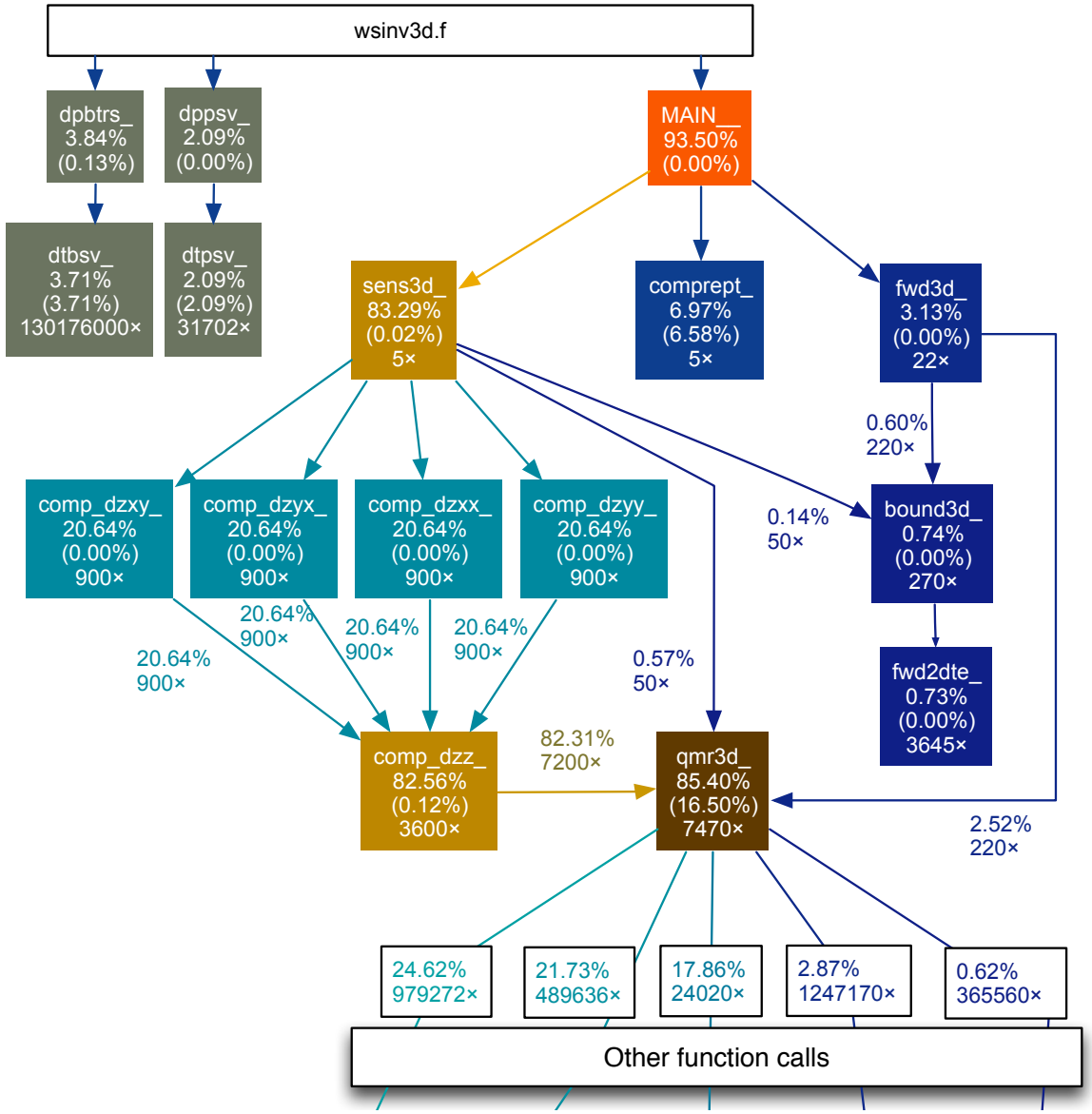


Figure 5.1: The 'call chart' for inversion on a small model and artificial data set, with 5 iterations. The labels represent the function name, the percentage represents the fraction of the total running time spent in this function and its children, and the percentage in parentheses represent time spent executing this function alone. So, for example, the sum of all of the values in parentheses should total to 100. When a function is called by more than one parent, then the arrows are labeled to indicate the percentage it is called from each parent. A statistical based profiler was used, so if the probability of finding the program inside a particular function is very small then the profiler may not detect this. The colours are intended for clarity and have no significance.

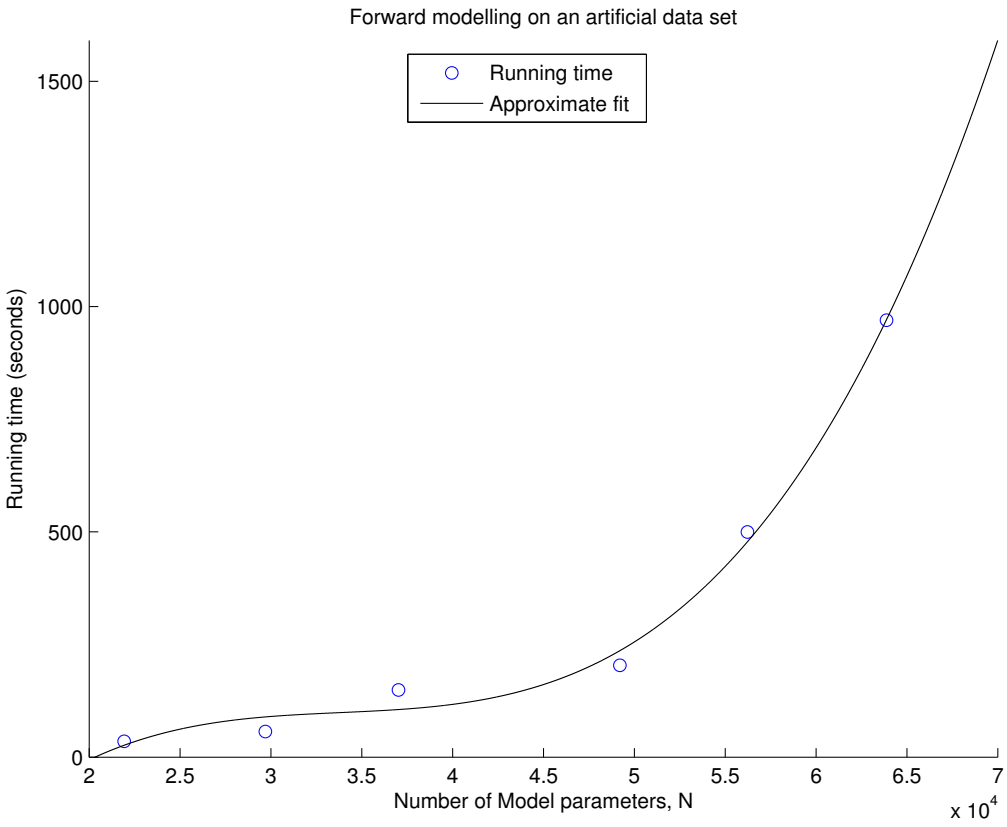


Figure 5.2: Forward modelling times on an artificial data set. The estimated fit is cubic and displays rapid increase in forward modelling time for increasing model sizes. The fit should not be expected to be smooth in practice, due to algorithmic optimisations depending on the model details.

A path of investigation was to determine whether forward modelling could be sped-up by offloading computation onto GPU units for parallel processing. As noted in the paper [18], the wsinv3d Fortran code package is complex and well tested, and it is beneficial to '...not rework any logic or algorithmic technique in the Fortran code which would require it to then go through a rigorous testing cycle'. This could be achieved by using a linear algebra library optimized for the GPU, such as CUBLAS [32] in place of library calls to the standard Fortran BLAS routines.

There is overhead involved in transferring data back and forth to the GPU unit, and to make this approach worthwhile, significant time (when compared to transfer time) must be used in these BLAS routines. An analysis of the forward modelling was carried out to determine the number of calls made to BLAS library, and the time spent in each call. This is outlined in table 5.1.

Function name	Number of calls	Total time (s)	Time per call (μs)
zdotc	122191	68.48	0.5604
zdotu	169920	64.21	0.378
zaxpy	23829	0.13	0.00546
zscal	916831	0.07	0.000076
zgeru	1818432	0.04	0.000022
ztbsv	20664	0.04	0.0019
zcopy	9796	0.02	0.0020
izamax	929880	0.01	0.000012
dscal	186837	0.01	0.000054
dcabs1	1842261	0	0
dsyr	186837	0	0

Table 5.1: Linear algebra library calls and the corresponding time spent in each. These calls would appear below 'other function calls' in figure 5.1. The total time for the overall forward modelling calculation was 1453 seconds. The cumulative time spent in the above calls is 133.01 seconds, or 9.15%.

From table 5.1, it is clear that even if we could completely eliminate the time required for these calculations, we would only achieve less than 10% speedup. It was decided that this route was not worth perusing.

Chapter 6

Hadoop implementation of 3D inversion

We have already introduced Hadoop as a parallel computing framework, and discussed the benefits in using it for computationally intensive scientific computing. We shall now explain how we have taken the parallel implementation of WSINV3DMT created by C3L, and implemented it using Hadoop MapReduce and Hadoop HDFS.

It is of benefit to start at a high level, showing the Class structure and execution path. We can then explain more about Java and Hadoop, and the libraries and APIs required to make this a possibility. Then we will show how we have designed our program to be modular, so that much of the 'boilerplate' code can be re-used in future projects.

Algorithm 2 has been left largely intact, so we will describe this in more detail, and explain the important functions of the seventeen source files into which it has been divided.

We then detail issues relating to Hadoop, that needed to be overcome in order to get this to work.

As in Chapter 3, we use some common terminology from Computer Science and Java programming. Users unfamiliar with this should check the glossary in Appendix A.

6.1 Class structure and Hadoop

Using the 'Plain Java' method (Section 3.4.5) of defining the parallel structure of our program using the Hadoop API, we design a Java Class file that handles the decision making logic for the WSINV3DMT algorithm. This is very similar to the C3L project, which transferred the decision making logic to a Python script. So, we were able to use this approx. 500 line Python script as a reference.

A major goal of this implementation, was to keep the code implementing the MT inversion process separated from the boilerplate code; which handles the interactions with the Hadoop framework and operating system. This enables much of our code to be reused, as we will discuss in Chapter 8.

Figure 6.1 provides an overview of the important class files pertinent to this discussion. We shall start by detailing the functions of the MT specifc class files, then present the more technical details of the boilerplate implementation, which is important for future work.

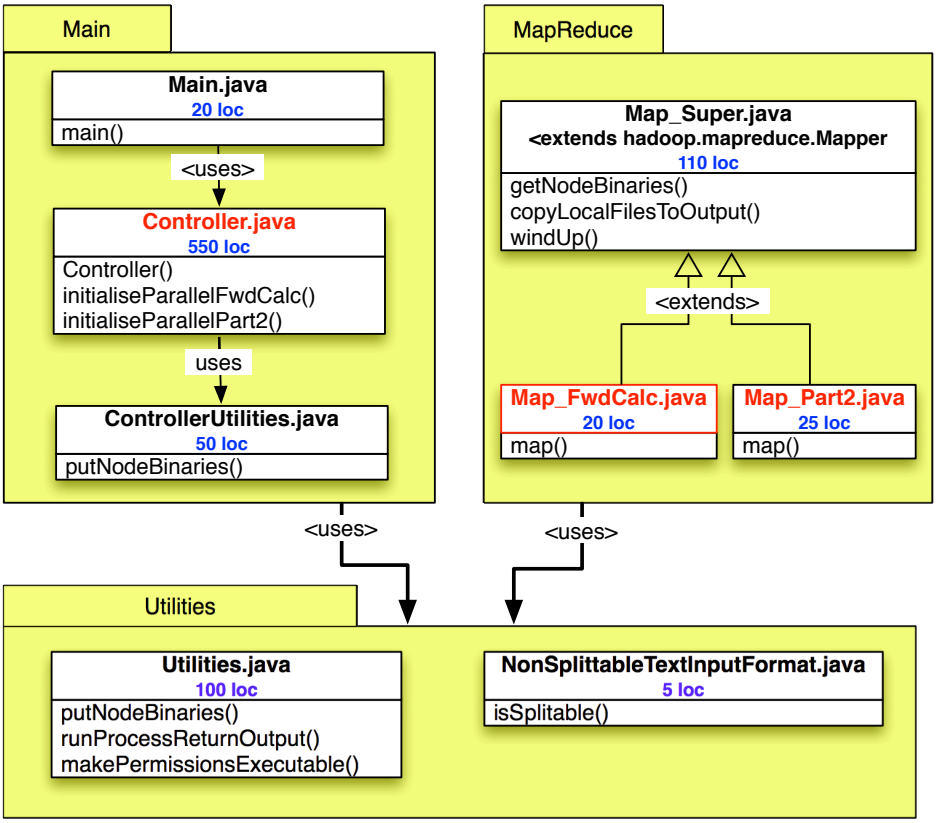


Figure 6.1: Class structure of the important source files in the implementation. The three packages; Main, MapReduce, and Utilities, represented by the three yellow boxes, contain code used for different purposes. The Main package contains the code for reading input and running all calculations on the head node. MapReduce contains the code designed for running on remote nodes. Utilities contains utility methods that may be used by any other class, or the Hadoop Framework. Classes labelled in red indicate that they contain MT specific code. In our explanations we shall mostly omit the extension .java. The *loc* label indicates the important lines of code contained within this class.

6.1.1 MT Inversion Logic

The logic handling the inversion process is implemented in the file Controller. This has the function of controlling execution of the 17 Fortran source files, which carry out the logic found in Algorithm 2. The controller also starts the mappers Map_Part2 and Map_FWDCalc, which receive the entire data set [1]and carry out the sensitivity matrix calculation and forward modelling, each instance handling a single frequency.

We can see a high level overview of how the classes interact during execution in Figure 6.2. This Main class reads the input data from the local file system, and instantiates the Controller object. The Controller object then performs Part1, before going in to the main loop in Algorithm 2 which is indicated in the diagram. In this loop the Controller runs the other 16 parts of the Fortran code, determining when to run MapReduce tasks for Part2 and FwdCalc on the remote nodes.

These 16 other Fortran files carry out different parts of Algorithm 2, for example, computing the Cholesky decomposition and updating the model file. They also handle reading input and writing output, and the matrix operations seen in Chapter 5. Further discussion on this is not relevant.

Note that the loc labels in 6.2 represent the important lines of code here in the class file, not including comments and import statements.

6.1.2 Mappers and Reducers

We extend the abstract Hadoop class Mapper, in order to define the code that should be run on the remote nodes. Map_Super contains general functionality required by both Forward Modelling and Sensitivity matrix calculation. It has two main purposes.

The first is to retrieve the files required by the Mapper, these files include both data and Fortran executables. The second is to provide a 'clean up' method, which supplies the output from the Fortran executables to the Hadoop framework for collection, and deletes any temporary data residing on the node on which it executes.

This Map_Super.java class is very general, and we further extend it in order to run the sensitivity matrix calculation and forward modelling calculation. This is done by the classes Mappers Map_Part2 and Map_FWDCalc, which are very simple, and need only 'run' their respective Fortran executable.

6.2 Technical details of implementation

We now delve into the technical details of how the classes above are able to perform the tasks described in 6.1

We will use the standard Hadoop input and output mechanism of <key,value> pairs only for instructing the node which frequency it is to process, which we discuss first. We then discuss issues relating to transferring data to the nodes, and execution of Fortran binaries on those nodes.

[1]Not an issue, since the data set in MT is simple the impedance tensor, a relatively small file usually less than 1mb.

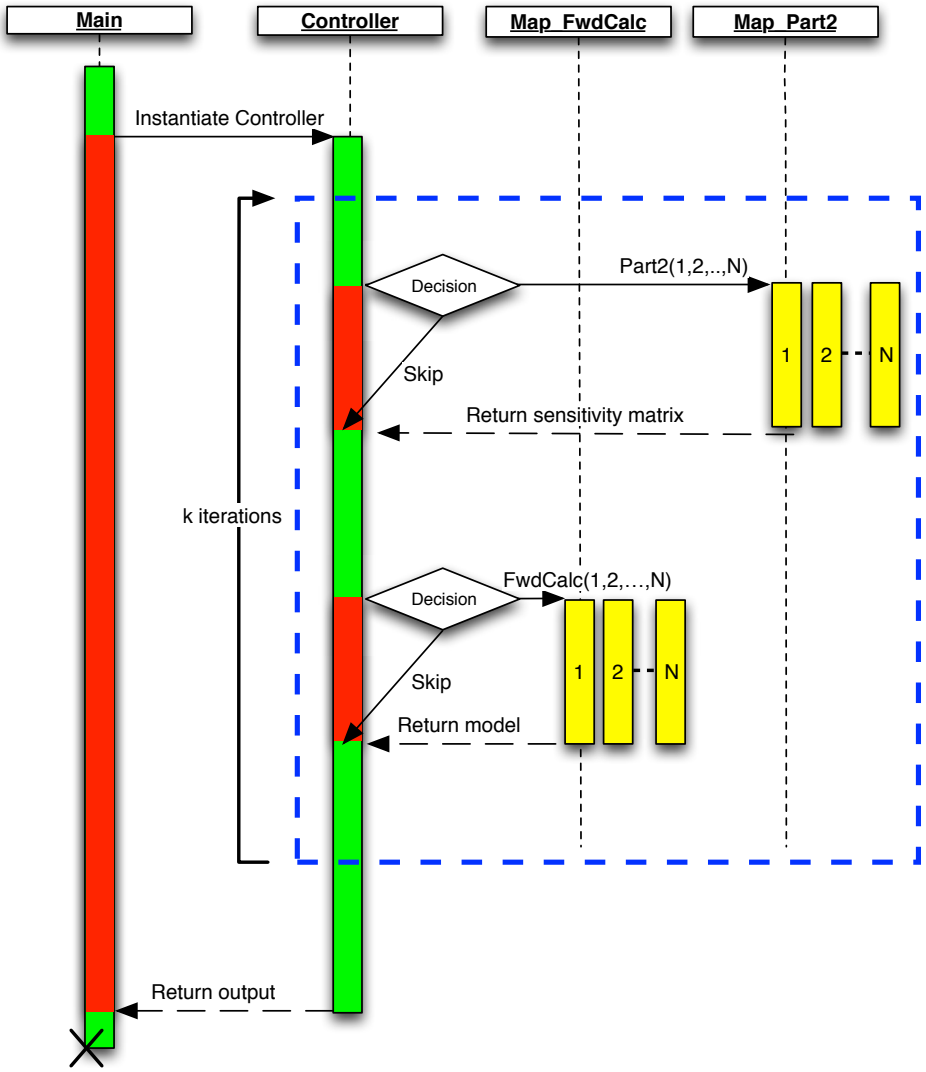


Figure 6.2: The white labels at the top represent the class names, corresponding to the ones in the class diagram Figure 6.1. Execution time proceeds vertically downwards. The vertical bars indicate that a function is running; green when it is currently carrying out a calculation on the head node, red when it is waiting for a sub-process (child) to complete, yellow when the process executing on a remote node/nodes. Solid lines with a label ending with '()' represent a function call, dashed lines representing control (and information) being returned to the parent. The dashed blue rectangle indicates that the section contained within it is iterated over k times. This diagram is not intended to illustrate any complex logic.

6.2.1 Mapper and reducer input and output

As mentioned in Chapter 3.3, Hadoop MapReduce uses <key,value> pairs as input and output for the Mappers and Reducers. The design of Hadoop MapReduce is heavily directed towards data processing, where Hadoop automatically splits input files into <key,value> pairs and sends these pairs to Mappers and Reducers.

We use this standard Hadoop MapReduce <key,value> system as a simple *messsage passing system*, which controls the number of Mappers, and the frequency that each Mapper processes.

We do not use the key, and it can take any value. This may be useful for specifying which tasks are allocated to each node, but that is not necessary in our work. In the Map stage, each unique value defines a unique Map task. Thus, we specify the Map class (either Map_Part2 or Map_FWDCalc), and then create a list of input values in the range 1,2,...,n, where n is the number of frequencies we wish to process. We specify this list of values to Hadoop as the input to the Map-Reduce job, and Hadoop handles creation of the appropriate number of Mappers, and passing each of them one of these values, which corresponds to the index of the frequency they are to process.

6.2.2 Issue 1: Data flow

We still need to handle transferring of the data, including Fortran binaries, to the nodes. We wish to use inbuilt Hadoop mechanisms to achieve this, because using resources explicily reduces portability.

We introduced HDFS in Section 3.4, and this is what we will use to transport data to our Nodes for computation. HDFS is complicated, but we do not need to get into the fine details of HDFS, indeed that is one of its advantages.

We will use HDFS slightly differently for input (to the Mapper) and output (from the Mapper). We will use the 'Distributed Cache' for input, and the standard Hadoop output mechanism 'mapred.output.dir' for output.

Distributed Cache

The Distributed Cache is a feature within Hadoop, primarily designed for transferring data to the remote nodes. Once a Hadoop Job has begun execution, data in the Distributed Cache cannot be modified - it can only be read. This enables HDFS to efficiently[2] replicate the information in such a way that each node will read the same information. This is contrasted to the input passed to Mappers and Reducers in the form of <key,value> input pairs, in that the data can be, and usually is, different [3].

We have two types of data that need to be sent to the nodes - the Fortran executables and the MT data. While sending the same Fortran files to the remote nodes every time a job is initiated may seem unnecessary, we do so for a number of reasons:

[2]With regard to network latency and storage redundancy, as HDFS is designed to scale from one node to several thousand nodes, possibly located in different physical locations

1. The data transfer time (on the order of a few seconds) is small when compared to the execution time (several minutes to several hours)

2. It is simpler, in that we do not need to concern ourselves with storing persistent data on the nodes

3. There is no need to pre-configure different nodes; which means we can add new compute nodes to the Hadoop cluster at will, without needing to install software or pre-configure them.

The Distributed Cache has a simple API. The code for used by the controlling script to put files in the DistributedCache is in ControllerUtilities.java; and the code for retrieving those files at the remote nodes is contained within the Mapper superclass Map_Super.java.

mapred.output.dir

The mechanism we use to transfer output from the Mappers to the head node is the standard Hadoop MapReduce output directory in HDFS. There are some subtleties to how this works. When a Map task has completed, we transfer data to a temporary output directory, referred to by Hadoop MapReduce as 'mapred.output.dir'. Hadoop then handles making this available on the head node. The reason for this indirect approach is that it allows the fault tolerance and load balancing mechanisms in Hadoop to function correctly. If duplicate mapper jobs are running, then the framework ensures that only one of their outputs are transferred through to the final output directory made available on the head node, allowing it to work transparently to the user. The code for using this output mechanism is contained within Map_Super.java and Controller.java.

6.2.3 Issue 2: Executing Fortran code within Java

Compilation

While the Java code we have written is portable across any Hadoop system [4], the Fortran code must be compiled to suit the operating system on which Hadoop is running. This is a necessary pitfall of using a compiled language such as Fortran, rather than an interpreted language such as Java. However, much like the way consumer software is distributed, this could be done in advance so it would not be required of the user of our software package.

ProcessBuilder

In order to run our Fortran executables, we need to somehow 'wrap them up' so that they can be executed from within Java. This is done using a standard feature of the Java library, java.lang.ProcessBuilder. There are some caveats when using this, best illustrated through a code sample.

[3] A notable exception to this is the canonical Hadoop example - Word Count, where a long document is split into several blocks of text. These blocks are passed to the Mappers, which output <word, 1> where a different pair is output for every 'word' in the text. Each reducer processes a different word, adding together the values, to give a total count for that word.

[4] With the exception of changing Permissions, also discussed here

Listing 6.1: Code sample for using ProcessBuilder

```
1 ProcessBuilder pb = new ProcessBuilder(commands);
2 pb.redirectErrorStream(true);
3 Process process = pb.start();
4
5 InputStream inputStream = process.getInputStream();
6 StreamProcessingThread outputProcessor =
7   new StreamProcessingThread(inputStream, "Process output:");
8 outputProcessor.start();
9 int waitCode = process.waitFor();
```

We want to retrieve the console output printed by the Fortran executables. Additionally, if this output is not consumed, the executable will be halted and will not complete. The first line in Listing 6.1 instantiates the ProcessBuilder object, where 'commands' are the commands we wish to execute, as if we were running the executable from the terminal. The second line redirects stderr to stdout, so we can collect them with the one stream reader. The third line starts the process. The following lines attach a stream reader to consume output from the executed process, and send it through the Hadoop logging system, so we are able to read it from our head node. The final line tells the Java program to halt execution until the process has finished.

Permissions

When we use Hadoop to transfer the executables files to the remote nodes, we must change the permissions of those files to make them 'executable' before we can run them. In a Linux based operating system such as Ubuntu, permission changes are made by running the inbuilt 'chmod' program, with the path of the executable as a parameter.

We need our program to perform this permission change, so we will use the ProcessBuilder to run chmod before running the Fortran executable. The code in Listing 6.1 is used for this purpose, and we use the appropriate 'commands' for each executable file, eg. 'chmod a+rwx Part2'.

We have only implemented this permission change for a Linux based system, on which all of our testing was performed. On a Windows based operating system these permission changes would be different, and we would need to specify the appropriate command within our program.

6.3 Hadoop configuration

We stated earlier that Hadoop makes it easy to choose the computational resources to match our inversion. We would like to be able to finely tune Hadoop to make full use of those resources. The Hadoop configuration files can be used to specify the Map task capacity per node, thus making use of multi-core processing capabilities of the hardware, if present. See Appendix E for more details and an example.

6.4 Testing Hadoop implementation

With a complete implementation of WSINV3DMT using Hadoop MapReduce, we would like to be able to test it. We have made no modifications that we would expect to result in different output, thus we would like to confirm that output is identical to the existing serial and parallel implementations. We will do this on the well tested data-set detailed in Appendix C, and we are able to directly compare our running time to that of the 5 frequency data set in Figure 4.1.

Several other data-sets were also tested to ensure output at each iteration was also identical. We found the output to be identical for all three implementations, in all cases.

The testing was conducted on AWS, using the cc2.8xlarge instance type described in Appendix B.1. We can examine the execution time, detailed in Table 6.1. The results for WSINV3DMT and C3L Parallel were collected during the earlier research by C3L, and the instance type m1.large, described in Appendix B.1, was used. Our testing (See Appendix B.1) showed us that the single threaded performance of cc2.8xlarge was approximately two times that of the m1.large instance. In particular, serial inversino using WSINV3DMT took just over twice as long to complete on the m1.large instance.

We have included the cost in Table 6.1 as this is a valuable measure to determine the usefulness of our implementation. For identical hardware, we do not expect the running time of our implementation to be any different than the C3L parallel implementation. However, we have much greater flexibility when choosing our instance type, as we can utlitse both multi-core machines, and different machines, completely transparently through Hadoop. The C3L implementation could theoretically achieve the 50 hour time that we recorded, but it would cost $50 * 8 * \$2.40 = \$960$, as their implementation cannot utilize the multi-core nature of the cc2.8xlarge instance type.

We see that of the two parallel implementations, C3L and Hadoop MapReduce, Hadoop MapReduce is both cheaper and faster. When comparing WSINV3DMT to the Hadoop MapReduce implementation, a cost increase of 25% results in a reduction of execution time by a factor of 6. Alternatively, comparing WSINV3DMT to the C3L implementation, a cost increase of 248% results in a reduction of execution time by only 3.2. In many cases, these ratios may mean that our Hadoop MapReduce implementation is a viable alternative to WSINV3DMT, where the C3L implementation was not.

Implementation	Hardware	Execution time (hours)	Cost
WSINV3DMT	m1.large	300	\$96
C3L	m1.large * 8	93.7	\$239
Hadoop MapReduce	cc2.8xlarge	50	\$120

Table 6.1: Time and expense comparison for the Paralana 5 frequency data set. Both the C3L implementation and the Hadoop MapReduce implementation run the forward modelling and sensitivity matrix calculations in parallel. The costs are calculated by the instance cost per hour, multiplied by the number of hours. For C3L and Hadoop MapReduce, this is multiplied by the number of parallel machines.

Chapter 7

New data set: Carrapateena region

We shall now investigate using our software package on a new data set. We will need to tune the input parameters of the inversion in order to complete the inversion in a reasonable time. Reasonable, is in most cases, determined by time and budgetary constraints. We will be using the cc2.8xlarge instance type on AWS for our inversion and we will aim to design our inversion to finish in under 48 hours, bringing the total cost for the inversion to around $116.

Our new data set is from the Carrapateena region in South Australia. We will start this Chapter by examining the station data and locations, and remove data in order to speed up our calculations. We will then run some experiments with different starting models, all using the same half-space format, but varying in both resolution and physical extent. This will provide a good opportunity to examine the trade-off between the physical information we can obtain; and the practical issue of running time.

We will see that with these restrictions, we are only able to complete a preliminary analysis on this new data-set. However, we should be able to determine some useful information, and provide suggestions for performing a more extensive inversion. We are performing our inversion on the Amazon cloud, but our Hadoop implementation can easily be installed locally on machines dedicated to performing these inversion tasks, providing a more cost effective solution in some cases.

7.1 Data set

Our data set contains 48 stations, each with 8 response function directions, and each of these directional components are recorded for different 7 frequencies. These components correspond to the components of the impedence tensor in 2.1, and we make the conversion from frequency (Hz) to period (s) due to the convention in [30].

This data has already been pre-processed using BIRRP; and we will examine the resulting data in detail now.

7.1.1 Station data

We note that a computationally intensive part of the inversion algorithm is the sensitivity matrix calculation, which is dependent on both the number of model parameters M multiplied by the number of data parameters N.

The measurements from each of our 48 stations contains the 8 components of the impedance tensor; giving a total number of data parameters of $N = 384$.

We would like to remove some of these, in order to provide a smaller data-set on which to perform our inversion. We do this by selecting stations and completely removing all of their data from the data file. However, with 48 stations, we need to decide which station data to remove.

Figure 7.1 shows a plot of all of the the station locations. The ones we wish to keep have been marked. We have chosen to remove stations from tightly packed regions, maintaining an even coverage over the entire survey area. Another approach might have been to focus only on a region with tightly packed stations; however we wanted to examine a larger region. We have reduced the number of stations to 36, and this will be our starting point to examine different models. We will not examine calculation times with different numbers of stations, as we wish to concentrate our efforts on the effects of model size and resolution.

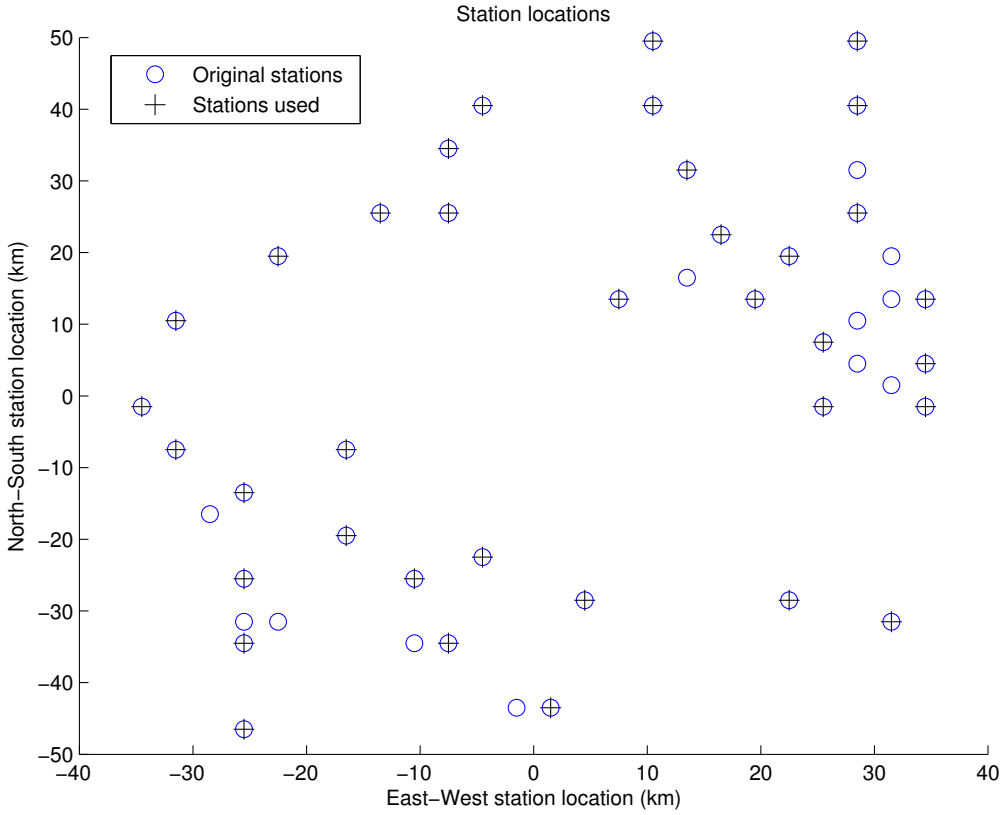


Figure 7.1: MT Station locations. The 'Original stations' indicate all station locations in the original data set. 'Stations used' indicate the ones we have chosen for this inversion. In total, we have removed 12 of the original stations from our data set.

7.1.2 Skin depth

The skin depth equation (2.2) is often simplified using some reasonable approximations. Since the magnetic permeability μ does not vary substantially in the Earth [33], we can approximate the skin depth δ (in m) by:

$$\delta(T) \approx 500\sqrt{T\rho_a} \tag{7.1}$$

where $\rho_a = 1/\sigma_a$ is the apparent resistivity (in Ωm) and T is the period (in seconds). (7.1) is a widely used practical equation in MT [20], due to its usefulness in quickly approximating the depth of exploration.

For the apparent resistivity, 10 Ωm is a reasonable value to use in sedimentary environments, but is not applicable when the skin depth exceeds the thickness of the sediments. In many cases, a value of 100 Ωm becomes a good approximation[34], and is what we shall use here.

Figure 7.2 shows the approximate skin depths for the periods recorded in our data. We can use this information to make adjustments to our model file.

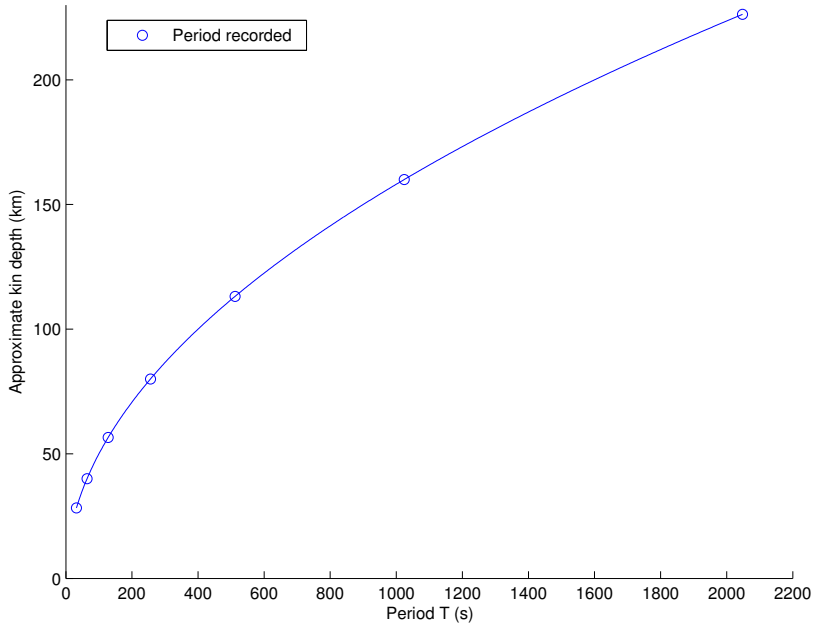


Figure 7.2: Approximate skin depths for the Carrapateena region data set with a 100 Ωm apparent resistivity.

7.2 Model file choice and output

Recall that a model file defines a 3-dimensional grid, with M_x,M_y,M_z blocks in the x,y,z directions respectively, each block having a resistivity value. While blocks can be different sizes, there are some restrictions on the model file that can be used with WSINV3DMT. The main one of concern is: (1) The blocks in the model must be distributed so that the station

locations, as defined in the data file, appear in the centre of the blocks at the surface.

Our stations are distributed over a 100 km×80 km region, so the blocks within this region of the centre of the model file must be correctly aligned.

There are some rules of thumb on model file dimensions, see Appendix D.2. The goal is to extend the model far beyond the region we wish to examine, which will allow edge effects to be minimized within the region that we do wish to examine. We usually have a higher resolution, i.e. smaller blocks, in the central region, and the external region consists of larger blocks, which are primarily there to minimise edge effects. We call these larger blocks the *buffer zone*.

There are no such restrictions as (1) when it comes customizing our model in the vertical direction. However, we do wish it to be appropriate for the skin depths in Figure 7.2. We have chosen our model grid locations to cover a depth up to 668 km, as plotted in Figure 7.3.

We will use a half-space resistivity format for all initial models, with a value of 10 Ωm. This means that every block in the model starts with the value 10 Ωm. As mentioned in 2.4.1, the half-space format is preferred when little is known about the subsurface geology.

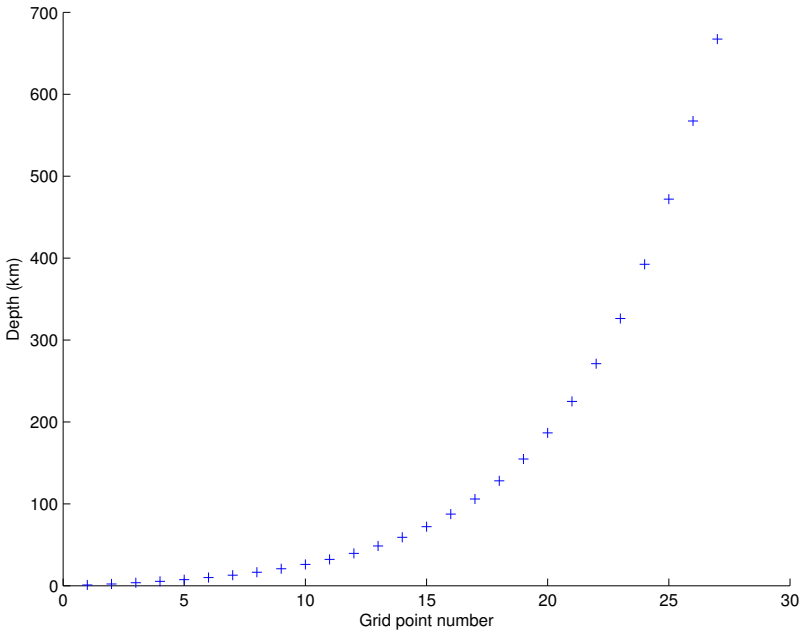


Figure 7.3: Model file grid point locations in the z direction, indicating the depth of the model.

We start with a model file generated by a commercial software package called WinGLink[1]. This software generates a model file where the grid spacing in the x-y plane satisfies requirement 1, and the vertical grid spacing is as above.

We then make some modifications to the external region of the model file in the x-y plane, without modification to the central region. We will examine three different modifications, each with the same grid spacing in the vertical direction. All quoted times, unless stated

[1]WinGLink[11] is a commercial software package for processing and interpreting geophysical data, primarily focused on Magnetotellurics

otherwise, are for the initial model file, with the highest frequency.

7.2.1 Model 1: benchmark

This model will be our benchmark. The model file has number of parameters 52 ×40 ×27, with coverage in the x-y plane of 512 km×476 km, as displayed in Figure 7.4a. The sensitivity matrix calculation time was 23 minutes, and the forward modelling time was 69 minutes.

7.2.2 Model 2: decreasing surface coverage by block removal

For this, we remove the second largest padding blocks (shaded in Figure 7.4a) from Model 1, and shit the outer padding blocks inward, contracting the model. This must be done symmetrically about the centre of the model, in order to maintain requirement 1. This gives us a model file of 50 ×38 ×27, with coverage in the x-y plane of 416 km×380 km.

The sensitivity matrix calculation time was 15 minutes, and the forward modelling time was 30 minutes.

7.2.3 Model 3: increasing surface coverage by adding a block

For this, we add a block of length 96 kmin both the x and y axis to Model 1, in order to extend the range. This block is added inside the largest block (100 km). This must be done symmetrically about the centre of the model, as per the previous model. This gives us a model file of 54 ×42 ×27, with coverage in the x-y plane of 704 km×668 km. This has not been plotted.

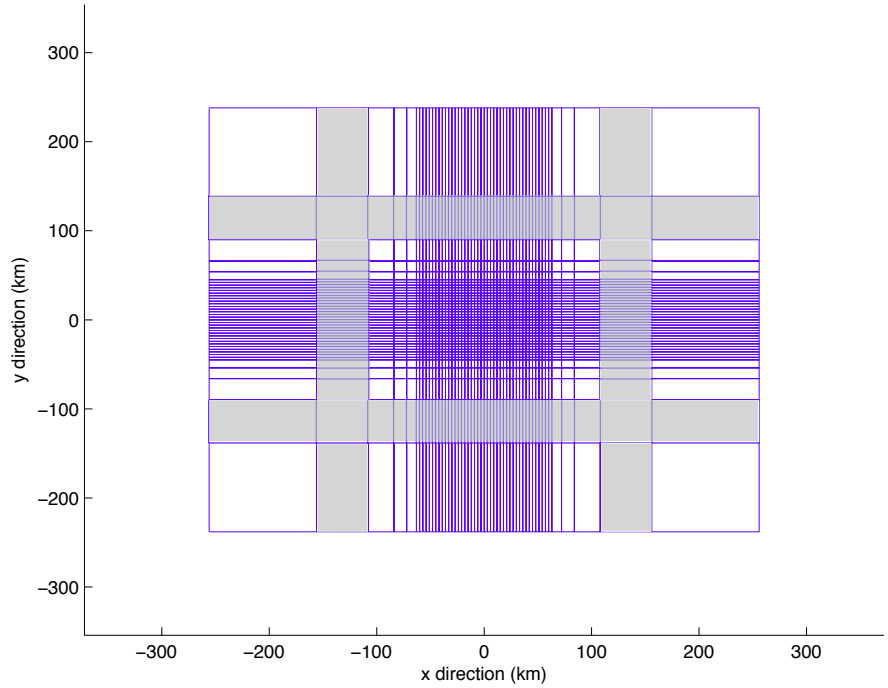
Forward modelling time was 87.75 minutes; sensitivity matrix calculations, however, ran in excess of 15 hours for all frequencies, and we did not wait longer for them to complete. Currently, with the Fortran code running these calculations, there is no way to check the progression, so it could have taken much longer.

This has showed that extending the model range with an extra block has dramatically increased the sensitivity matrix calculation times. We wish to know whether that is due to the extra parameters introduced, or whether it is due to the greater physical extent. We will conduct an experiment next, by simply scaling up all of the buffer blocks to give the same physical extent as this mode, without increasing the number of parameters.

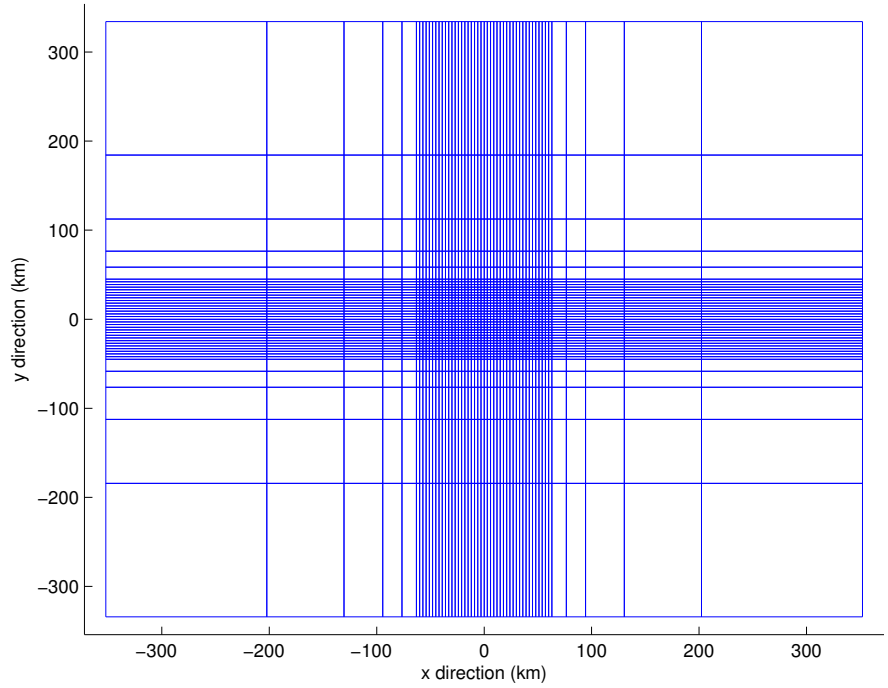
7.2.4 Model 4: increasing surface coverage by scaling blocks

We will increase surface coverage by multiplying the size of all padding blocks by 1.4980. This gives us a model file of 52 ×40 ×27, with coverage the same as Model 3 (704 km×668 km).

Sensitivity matrix calculation time was 29 minutes, and forward modelling time was 67 minutes. This is close to Model 1; clearly, then, it is the addition of a model parameter that caused the calculation to take so long for Model 3.



(a) Model 1. The shaded region represents those buffer blocks that will be removed in Model 2



(b) Model 4. The buffer block sizes in the buffer zone have been scaled by 1.4980.

Figure 7.4: Models 1 and 4 grid spacing the x-y plane. The scale is identical for both models, showing their relative sizes. For both models, the spacing in the densely packed central region is 1 km in both directions.

7.3 Inversion analysis

7.3.1 Model progression

Recall the discussion in Section 2.4.1, where we discuss that the τ value imposes a restriction on the model smoothness.

This suggests a two-stage inversion process. The first stage would be complete inversion using a high τ parameter, forcing the program to converge on a smooth model, but one that may not necessarily include smaller conductive bodies, giving us a relatively high RMS. The second stage would be to lower this τ parameter, and run inversion again. This second inversion allows the model to develop some roughness in the form of these smaller conductive bodies, which provide a better match to our observed response, and thus a lower RMS. This helps avoid the case where the algorithm quickly converges on unphysical models in the first few iterations; and enables us to approach a more reasonable model than if we left complete control up to the program. This also highlights the ill-posedness of the inverse problem, and some of the steps required to produce a useful model.

We have decided to use a high τ value for this inversion in order to produce some preliminary results. The second inversion stage will not be completed here, due to time and budgetary constraints, but will be a focus for further research using the code we have developed.

7.3.2 Inversion

With our 7 frequency data set, a single inversion runs 8 calculations in parallel; one for the head node running the controlling process, and the other 7 for forward modelling and sensitivity matrix calculations of each frequency. Since our cc2.9xlarge instance can handle 16 parallel threads with no noticable performance penalty, we will run inversion on all of our models above.

Table 7.1 shows the total time required for inversion, for each of these models. This presents no surprises, and again we see that the time difference between Model 1 and Model 2 is to small to be considered significant.

Model number	Inversion time
1	19 hours, 3 minutes.
2	13 hours, 3 minutes.
4	18 hours, 53 minutes.

Table 7.1: Time comparison for inversion (10 iterations) on each of our models. Model 3 has been omitted.

Figure 7.5a shows the RMS and norm of the models generated by the inversion process, at each iteration. There is no simple way to calculate the norm of the starting model without code modification. The other values are calculated during the inversion process, by our Fortran code adapted from the WSINV3DMT package.

We see in Figure 7.5a that the first few iterations rapidly reduce the model RMS, with a corresponding increase in the model norm seen in Figure 7.5b. Conductive bodies closer

to the surface contribute more strongly to the sensitivity matrix; thus they are discovered[2]in the first few iterations and 'fixed' in the model.

The behaviour after iteration 3, a gradual increase in RMS, has been observed to be common behaviour with certain models [34]. This behaviour could be related to our high τ parameter, as making length scales too large can result in difficulties in finding any models that fit the data [25].

At around iteration 5-6 we have a relatively low model norm, and a stable RMS, for models 1 and 4. We see greater instability in both the norm and RMS of model 2, which was significantly smaller in extent, and had fewer model parameters. This could be due to edge effects, or even conductive bodies within those buffer regions. In any case, models 1 and 4 provide much more stable results.

A RMS value of 1 corresponds to a fit to within the data errors specified. RMS values of between 2-3 are common in the literature [9] [13], and the RMS value for the final iteration of the Paralana region (Section 6.4, Appendix C) was 3.0.

In this case, our results in Figure 7.5a show RMS values significantly higher than this, but that is not a concern; the goal of a second inversion, conducted with our output model as the initial model, would be to lower the RMS to better fit the measured response.

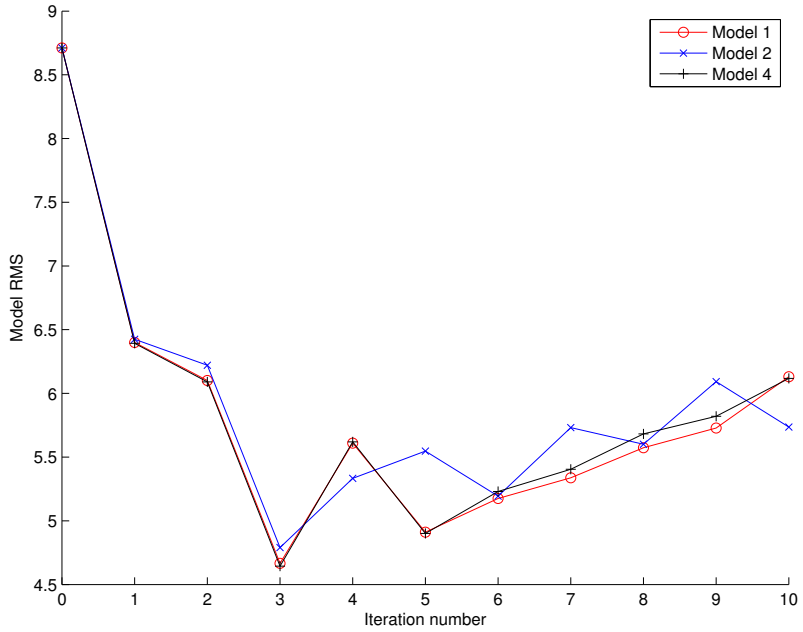
7.3.3 Analysis

Although we have not produced what could be considered a conclusive output model, we can still examine it to look for significant features. This examination will be brief, due to the preliminary nature of our investigation.

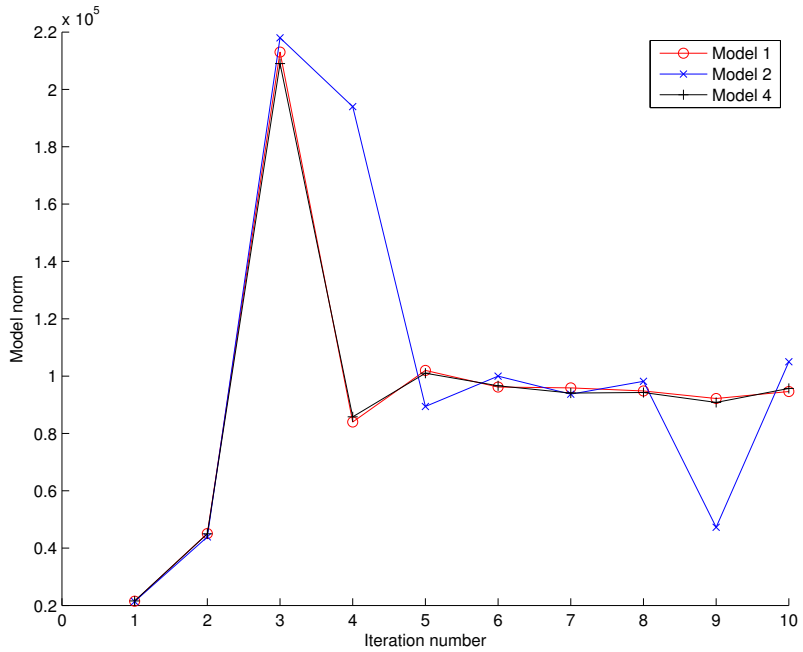
We will examine model 4, since that model is the largest in extent, which should help minimise the edge effects mentioned earlier. We saw in the previous section that the model reaches a minimum RMS at around iteration 3, and by iteration 5 the RMS has stabilised, while still maintaining a relatively low norm. We present the output at iteration 10; the model at iteration 5 might provide an equally good starting point for the second inversion, however we found little visible difference between either model.

Full three dimensional plots for model 4 at iteration 10 are shown in Figure 7.6. We know that conductive layers are more easily sensed than resistive layers[33], so we remove the resistive structures, generating a transparent model with only the more conductive structures visible. This shows a conductive body forming at a depth of 50-200 km. We have little physical information to compare this with, so it will be interesting to see how these preliminary results compare with the output from a second inversion, which is left for future work.

[2]We say discovered, meaning that these this reproduces the measured response, but due to the ill-posedness of the problem this simply means that we have found *one* possible way of recreating the measured response.

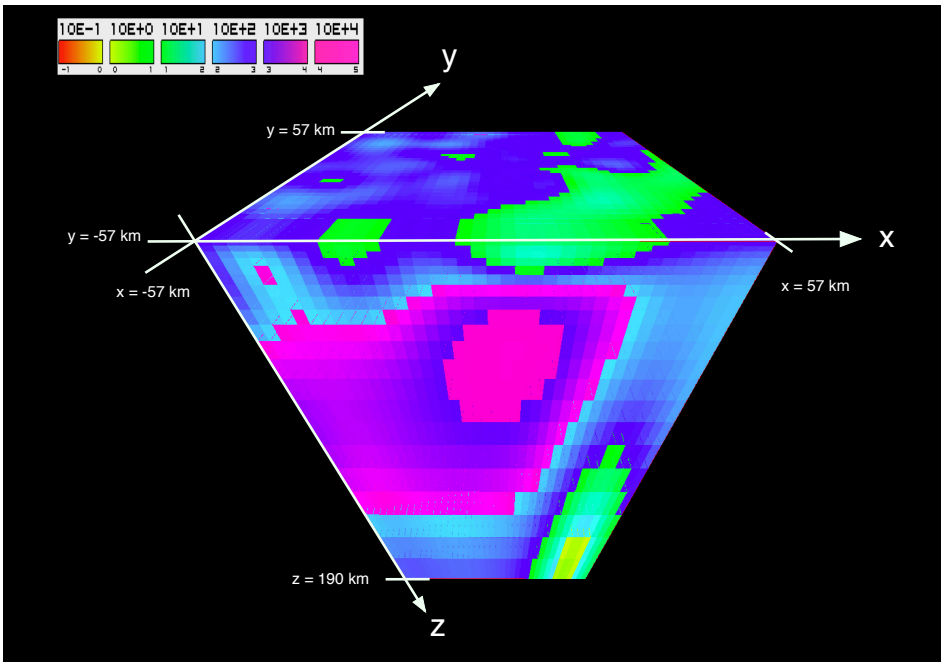


(a) RMS (misfit) at the end of each iteration. The value at iteration 0 is the RMS of the initial model.

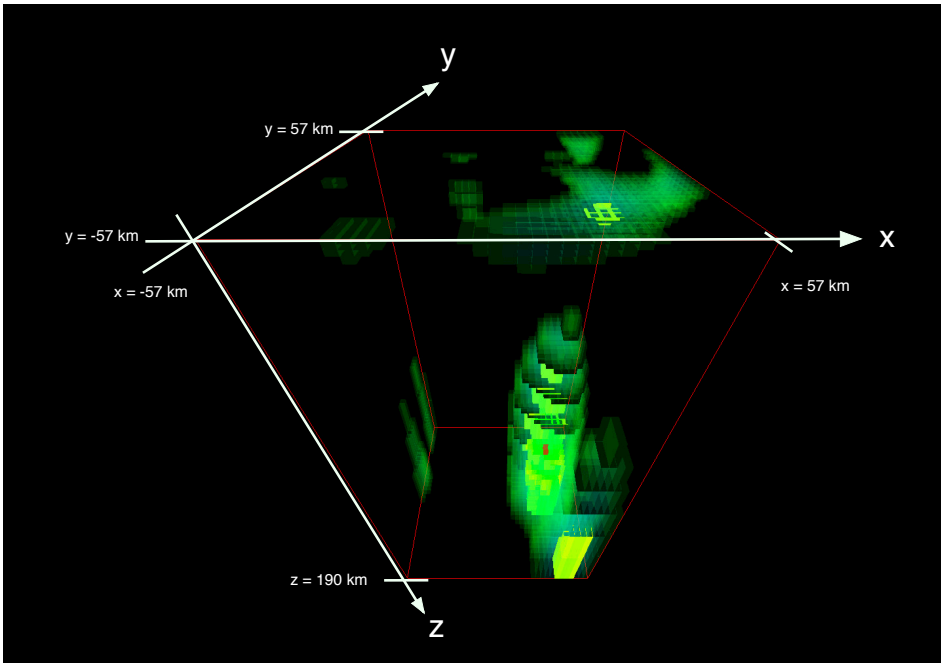


(b) Model norm at the end of each iteration.

Figure 7.5: Model RMS (a) and norm (b) at each iteration.



(a) Opaque model with all resistivities shown. Only the outer region is visible.



(b) The transparency of the highly resistive regions has been increased, allowing the conductive region in the interior of the model to become visible. The scale is the same as that of (a).

Figure 7.6: Output at iteration 10 using model 4. We have used the coordinate convention from our model and station data, where the origin is at the centre of the model file in the x-y plane, with $z = 0$ at the surface and increasing downwards. The scale indicates the magnitude of the resistivity, higher values indicate a higher resistivity. We have cropped the model to display a restricted region with surface area extending just beyond our station locations (Figure 7.1). This is intended to be a qualatative scale only.

Chapter 8

Conclusion

In many cases, 3-D MT inversion can provide a more detailed and reliable analysis of subsurface structures than 2-D inversion. However, the primary issue in running 3-D MT inversion is one of excessive computational time. We have examined how a parallel implementation of 3-D inversion helps overcome this, and that cloud computing provides convenient access to resources on demand for running this parallel calculation.

By taking advantage of Hadoop, a parallel computing framework, we were able to create a parallel implementation of a popular 3-D inversion code package, WSINV3D. Our implementation can be easily tuned to fully utilize the parallel processing power of the hardware that is available, from workstations with multiple cores, to clusters of computers, both local and in the cloud.

We were able to show that Hadoop, primarily designed for high performance data processing, is also a viable option for running computationally intensive scientific problems. We demonstrated that Hadoop MapReduce, commonly used with interpreted languages such as Java, is also capable of running compiled executables.

We also provided a test case for 3-D MT inversion, which exploited both the parallel capabilities of Hadoop, and high performance computing in the Amazon cloud, in order to generate some preliminary results on a new data set.

In writing our implementation of 3-D MT inversion on Hadoop, a primary goal was to keep the code relating specifically to MT, completely separate from any 'boilerplate' code. This provides a valuable resource for anyone wishing to use the Hadoop framework with existing code packages that can be run in a trivially parallel manner.

This provides an excellent starting point for high performance scientific computing using Hadoop, and magnetotelluric inversion in particular. There are some immediate pathways for future work, which we will discuss briefly now.

8.1 Future work

Other 3-D inversion code packages

A new 3-D MT inversion code package[10] is being trialled in the Geophysics department at the University of Adelaide. We will not discuss the technical or algorithmic details of this

code, however preliminary inversions have found good results[34]. This code is designed to be run in parallel, however initial work has found the technical challenge of using MPI to run it in parallel to be tedious. Implementing this using Hadoop MapReduce would provide a more stable, fault tolerant implementation than can easily be achieved using MPI, and much of our code would be reusable, making it a significantly less challenging endeavour than what we have achieved here. This will be investigated in the near future.

Evolutionary methods for 3D Inversion

Covariance Matrix Adaptation Evolution Strategies, or CMAES, is a type of stochastic optimization method belonging to the wider class of Evolutionary Algorithms. Like many population based stochastic methods, this lends itself well to easy paralellisation.

There has been some work on using CMAES as the search technique in 3-D MT inversion [2], which could greatly benefit from our research. The application of CMAES to 3-D MT inversion can briefly be outlined as follows:

We start with a population of possible solutions (models) and mutate (perturb) each one slightly slightly. This mutation is drawn from an M[1] dimensional Gaussian distribution, with zero mean, covariance matrix $\mathbf{C}$, and step size (FWHM) σ. Forward modelling is then run on each model in the population, and if the perturbation has increased our fitness (decreased the RMS value) then this is recorded and stored in a 'search path', which becomes a record of successful mutations. The covariance matrix $\mathbf{C}$[2] is adapted based on this search path as the algorithm progresses, as is the step size as we near an optimal solution. The entire process is then iterated, starting again with mutation with this new covariance matrix; for a number k of 'generations'.

Each member of the population can be forward modelled in parallel, furthermore, each frequency can also be calculated in parallel, as in our work. Population sizes of many thousands are possible using this method[1], giving a huge number of calculations that can be run in parallel. At this scale the fault tolerance and load balancing provided by Hadoop become highly beneficial, more so than in our work. This represents a viable research pathway for extensive parallelization of 3D MT inversion.

[1] M= the number of model parameters

[2] this definition of covariance matrix adheres to the standard mathematical definition for the covariance of an M dimensional Gaussian distribution, found in many texts such as [21]

Appendix A

Glossary

A.1 Computer Science terms

Source file	A file containing human-readable source code. This must be compiled before execution. In Java, these files have the extension .java.
Compiler	A program designed to convert source code to a set of instructions that is executable by a computer.
Compiled language	A language designed so that the compiler generates a set of instructions (commonly called a *program*, or *executable*) that can be executed directly by the processor. This program must be compiled specifically for a given operating system and processor instruction set.
Interpreted language	A language designed so that the compiler generates a set of instructions that can be executed by an 'interpreter'. The interpreter is a program that reads an interpreted language, and converts it to instructions that run directly on a machines instruction set.
Object orientated programming	A programming paradigm where all functions/methods belong to an 'object'. Objects are created (instantiated) and their functions/methods can then be called. Different instances of a class contain information relating to their state, and can be considered independent objects within a program. Further discussion is beyond the scope of this paper, and can be found in any modern introductory Computer Science text.
Java	An interpreted, object orientated programming language.

JVM	The Java Virtual Machine, the interpreter for the Java language. This is implemented on most Operating Systems. Compiled Java files can be run on any JVM without recompilation [1]
Class	(Java) The name of a type of 'object' in Java. Classes can be *instantiated*, creating an instance of that class, which is an object.
Import statement	(Java) A statment that tells the Java compiler to import the specified Class into the current source file, making its funciotnality available directly within the class in which the import is made.
Hot-spot	This refers to a small section of code within a larger program, that the larger program spends much of its time executing. This may be because the code is more computationally demanding than other sections, or it is called many times.
Hadoop	An open source collection of projects, designed for large-scale high performance computing. Includes Hadoop MapReduce and the Hadoop Distributed File System (HDFS).
Mapper	(Hadoop) A mapper is a a task that runs on a remote node, and recieves input <key,value> pairs. Typically, we have a different mapper for each unique key, and each of these mappers only handles input for a single key.
Map operation	(Hadoop) A map operation, or map task, is an operation carried out by a mapper upon recieving a <key,value> pair.
Thread	A set of instructions, or piece of code executing in serial. In parallel programming, multiple threads may execute in parallel.
MPI	Message Passing Interface. A low level[2] standard for parallel computing, which is based on explicit passing of instructions and data between threads.

A.2 Symbols and terms

Model **m**	A representation of the 3-D resistivity structure below the surface. In the form of a discrete set of blocks with finite extent in each cartesian direction.

[1] Providing we have a full JVM, not one with limited functionality, designed for mobile devices, for example.
[2] in comparison to Hadoop MapReduce

Forward modelling	The simulation of the Earth's electromagnetic response through a computer program. Used to determine the theoretical response of a model.
RMS	(2.4.1) The RMS of a model is defined as the root-mean-square of the pairwise distance between a models response $F[\mathbf{m}]$ and the recorded response $\mathbf{d}$.
norm	(2.4.1) A measure of the smoothness of a model. Defined as $\mathbf{m}^T {\mathbf{C_m}}^{-1} \mathbf{m}$, where $\mathbf{C_m}$ is the model covariance.
$\mathbf{C_m}$	(2.4.1) Model covariance. Defined algorithmically inside the Fortran program.
$\mathbf{C_d}$	(2.4.1) Data covariance. Defined algorithmically inside the Fortran program, utilizing 'data error' information in input files.

Appendix B

Amazon web services

Amazon Web Services (AWS) have been the cloud provider of choice for our research. This is for a number of reasons. Firstly, provide Infrastructure As A Service (IAAS), meaning we can easily configure the virtual machines with our operating system of choice, Ubuntu Linux Server 12.04 LTS. Our familariaty with Ubuntu makes debugging and troubleshooting much easier. The completed Hadoop implementation can then theoretically run on any Hadoop cluster, which might be running on a Windows based operating system (i.e. Windows Azure Elastic MapReduce), or other operating system supported by Hadoop.

B.1 EC2 instance types

The main AWS service we will be using is EC2, which provides access to virtual machines running our operating system of choice, and the abilty to choose the capabilities of the underlying hardware; termed the 'instance type'. These instance types vary in compute capability and memory (RAM), and are divided into two main classes. The first class contains instances designed for hosting web applications, databases, and carrying out other tasks where the underlying hardware is not imporant.

The second is the Cluster Compute instances, designed for high performance scientific computing. We shall exclusively focus on the highest performing cluster compute instance type, the 'cc2.8xlarge', but we will provide details on another instance type 'm1.large' for reference.

cc2.8xlarge

Price: $2.40/h
Amazon provide the folloiwng specifications:

```
60.5 GB of memory
88 EC2 Compute Units
(2 x Intel Xeon E5-2670,
eight-core "Sandy Bridge" architecture)
3370 GB of instance storage
64-bit platform
```

```
I/O Performance: Very High (10 Gigabit Ethernet)
EBS-Optimized Available: No*
API name: cc2.8xlarge
```

m1.large

Price: $0.26/h
Amazon provide the folloiwng specifications:

```
7.5 GB memory
4 EC2 Compute Units
(2 virtual cores
with 2 EC2 Compute Units each)
850 GB instance storage
64-bit platform
I/O Performance: High
EBS-Optimized Available: 500 Mbps
API name: m1.large
```

Where the AWS documentation states that

> One EC2 Compute Unit provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor. This is also the equivalent to an early-2006 1.7 GHz Xeon processor referenced in our original documentation.

B.1.1 Performance comparison

The C3L did much of its work using the m1.large instance type, whereas we have used the cc2.8xlarge instance type. We conducted a simple performance test by running the forward modelling code from WSINV3DMT on each. Both instances were running Ubuntu Server 12.04 LTS, and the Fortran code was compiled using the Intel Fortran compiler with the -O3 optimisation flag.

For later reference, the model file is called 'highresparalana_threeblobprime.0', with the number of model parameters $41 \times 41 \times 38$.

Time taken:

- cc2.8xlarge: 965 seconds
- m1.large 1991 seconds

So the execution time was over twice as fast ont he cc2.8xlarge. But note that this is single threaded performance, and the cc2.8xlarge can handle 16 concurrent threads, whereas m1.large can only handle two. These factors must be taken into account when choosing the instance type to run a calculation.

Appendix C

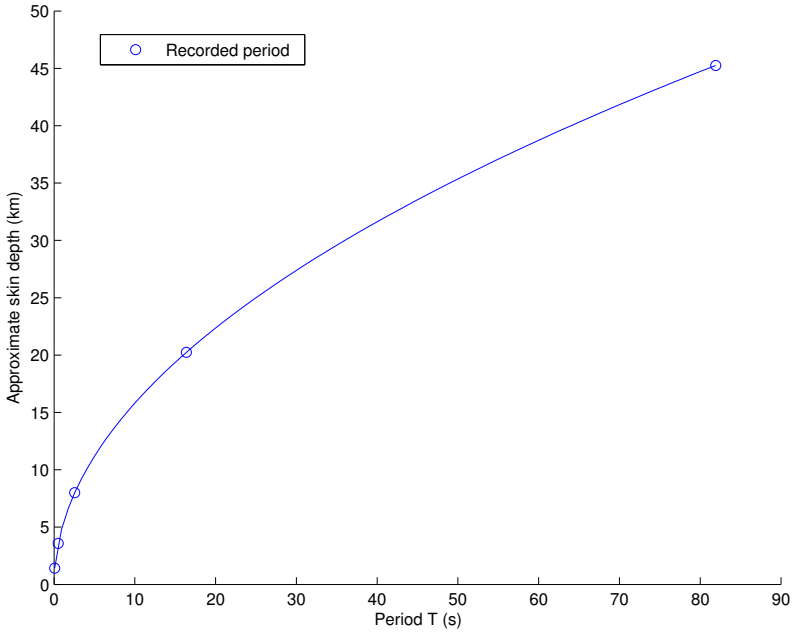
Field data: Paralana region (Run 7)

This is a well tested data set, and has served as the benchmark for testing WSNIV3DMT against the parallel C3L implementation.

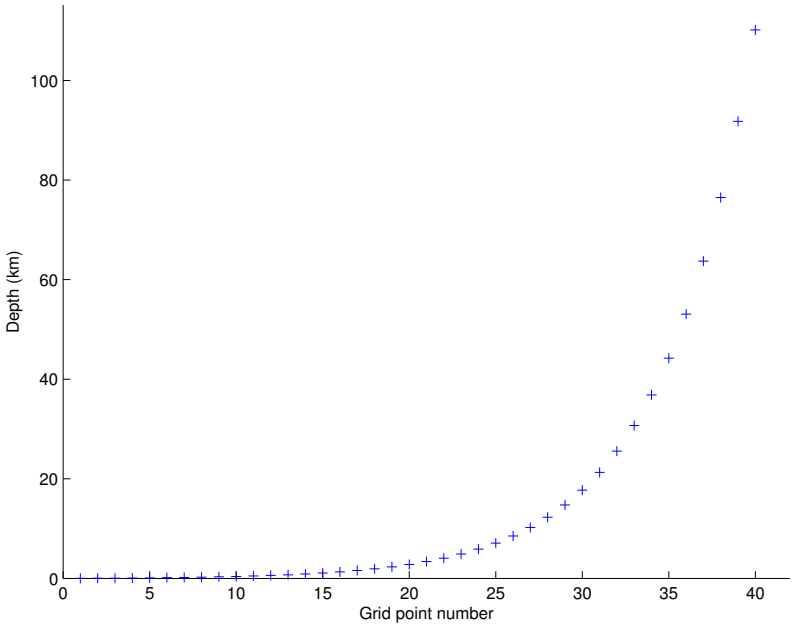
The data-set contains 53 stations, and 8 impedance tensor components for each of the 5 periods. These components correspond to the components of the impedence tensor in 2.1, with the conversion from frequency to period used due to the convention in [30].

The station locations are plotted in Figure C.2a. The periods recorded are: $T = 0.080000s$, $0.511999s$, $2.560000s$, $16.383988s$, $81.920213s$. We use an approximate half-space resistivity to calculate the skin depth for these periods, and this is plotted in Figure C.1a.

The model file has dimension $43\times47\times40$, plotted in Figure C.2b. In all cases, an inversion is started with a half-space model format with resistivity $10\Omega m$. The model grid points in the z direction is plotted in C.1b.

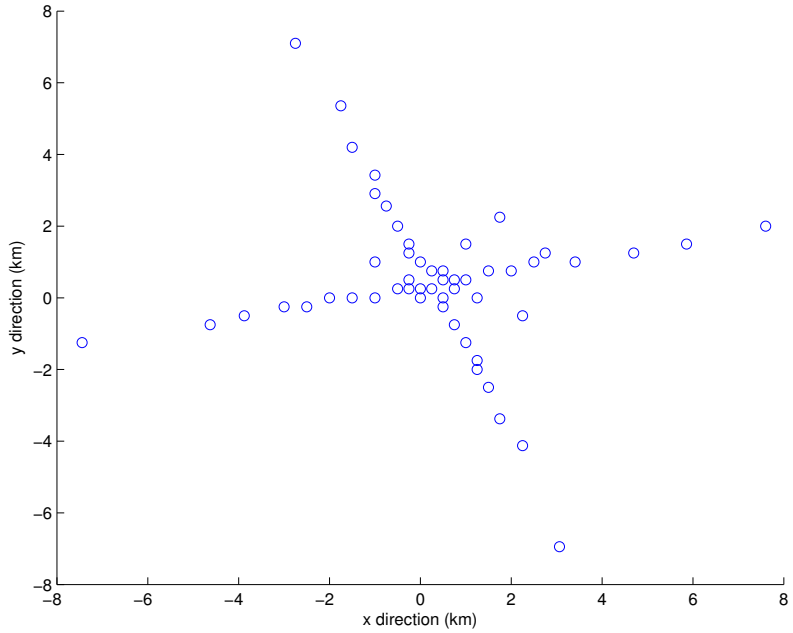


(a) Skin depth for the Paralana region. This has been calculated using an approximate value for the half-space of 100 Ωm.

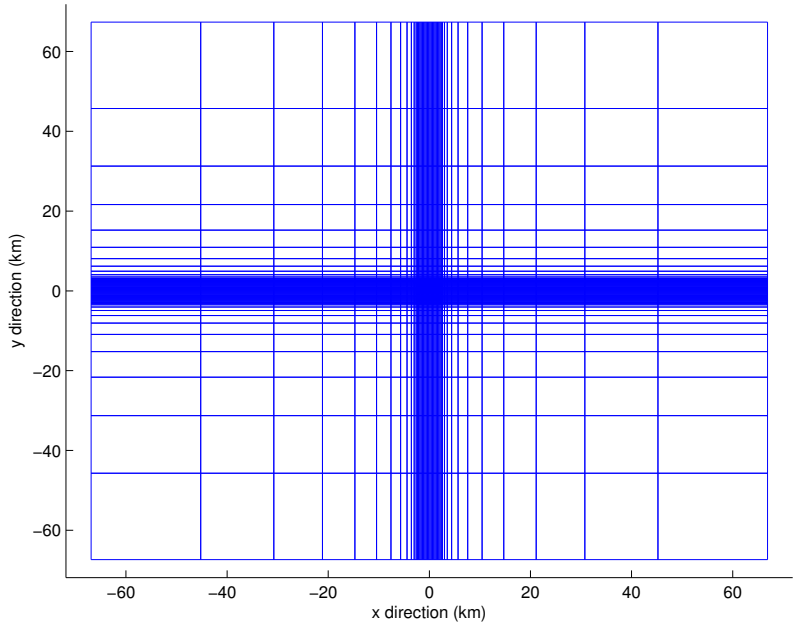


(b) Grid points in the z directon.

Figure C.1: Vertical direction data and model details. Plots of skin depth and model grid points.



(a) Station locations for the Paralana region.



(b) Model grid in the horizontal plane. Grid size in the central region is 750 m in the x and y directions.

Figure C.2: Data and model details for the Paralana region data set.

Appendix D

Additional MT information

D.1 3-D MT inversion, literature review

An excellent overview of the 3D MT inversion algorithms can be found in [27]. We will present a summary here.

The 2D Occam inversion method is introduced in [7], and Siripunvaraporn et al publish an efficient 2D Occam approach that works in the data-space, rather than the model space, in paper [25]. In the data space, many calculations and representation matrices depend on the number of data parameters, rather than the number of model parameters. Magnetotelluric inverse problems, in particular 3-D inversion, usually have far more model parameters than we do data values which the model parameters must satisfy; so data-space methods can be of huge benefit in reducing computational time, and the size of the matrices that must be stored in memory during calculation.

Siripunvaraporn et al extend the 2D data-space Occam variant to the 3 dimensional case in [30]. This has been released in a Fortran code package named WSINV3DMT, and was made freely available to the MT research community in 2006 [29]. This is our focus. A reference guide for this code package also provided in [26].

As mentioned, the Occam data-space approach used allows us to reduce the size of the system of equations that must be solved from $m \times m$ to $n \times n$, where m is the number of model parameters, and n is the number of data parameters. However, we still need to store the $n \times m$ sensitivity matrix in memory, which can be on the order of tens of gigabytes for large models (in the sense of number of parameters) and data.

Several improvements have since been published; and although they will not be our focus, it is worth mentioning them here for completeness. A conjugate gradient approach is introduced for 2D inversion in [28], that allows the explicit formulation of the sensitivity matrix to be avoided. This is extended to the 3D case in paper [31]. Modifications to WSINV3DMT to handle the vertical magnetic field transfer function, and also an algorithm for a basic parallel implementation, are introduced in [29]. However, as mentioned, the code base for the original WSINV3DMT is freely available, and the amount of code is significant ($\approx 20,000$ lines of Fortran 77). We wish to stick with this code base, so these approaches will not be mentioned.

D.2 Rules of thumb for model selection

There are a few rules of thumb to aid in choosing a starting model [34].

1. Usually block size should increase with depth d as $S \propto d^{1.5}$. This is in account with the poorer resolution of MT at depth.

2. Usually the first layer is roughly 1/10th of the skin depth of the shortest period (assuming roughly 10 Ωm near surface).

3. A good starting point to avoid edge effects is usually 1000 km$\times$ 1000 km$\times$ 1000 km model. Edge sizes could be reduced to 700 km.

4. The blocks should extend to several times the expected skin depth away from the edge, and away from the surface. This may or may not be in accordange with 3.

5. When we have no idea of subsurface structure we usually go for a half-space model.

6. Setting it to a half-space resistivity of 10 Ohm/Metre (roughly that of the silt) allows the inversion code to do what it does best, fix structure on the surface first because it has a higher sensitivity. Then as iterations progress, surface structure is fixed and deeper structure develops variations.

Appendix E

Hadoop configuration

While using Hadoop is simple, installing and configuring it can be complicated; which is why Hadoop 'services' such as AWS Elastic MapReduce, and Hadoop on Azure are so convenient. There are plenty of resources for setting up a Hadoop cluster manually, for example [35]. At the time of writing, Yahoo contribues to Hadoop extensively, and also provides excellent documentation which can be found through a simple web search.

However, one configuration setting that we would like to explicitly note here, is the configuration file for Hadoop MapReduce that determines how the Map and Reduce tasks behave. This determines important behavour, such as the number of tasks per node, and the timeout we wish to specify before a task (Map or Reduce) is considered failed. This is done in a configuration file mapred-site.xml, and an example is given in Listing E.1.

Listing E.1: mapred-site.xml example

```
1 <?xml version="1.0"?>
2 <?xml-stylesheet type="text/xsl" href="configuration.xsl"?>
3
4 <!-- Put site-specific property overrides in this file. -->
5 <configuration>
6   <property>
7     <name>mapred.job.tracker</name>
8     <value>localhost:9001</value>
9   </property>
10
11 <property>
12   <name>mapred.tasktracker.map.tasks.maximum</name>
13   <value>15</value>
14   <description>The number of milliseconds before a task will be
15                terminated if it neither reads an input, writes
16                an output, nor updates its status string.
17   </description>
18 </property>
19
20 <property>
21     <name>mapred.task.timeout</name>
22     <value>36000000</value>
23     <description>The number of milliseconds before a task will be
24                terminated if it neither reads an input, writes
25                an output, nor updates its status string.
26     </description>
27 </property>
28
29 </configuration>
```

Appendix F

Fortran 77

F.1 Compilation

The instructions here are applicable to the intel Fortran compiler, with hadoop running in Pseudo-distributed mode. The complete compilation command is:

```
ifort -heap-arrays 1024 -O2 -shared-intel -mcmodel=medium
```

The 'heap-arrays' is not always necessary, particarly for small model sizes, and is explained below; as are the other flags.

F.2 Memory issues

The Intel Fortran compiler allocates automatic arrays on the stack, while the GNU Fortran compiler allocated them on the heap. This means that in some cases, for large models or data, programs compiled with the Intel compiler may run out of stack space and crash with an error resembling:

```
forrtl: severe (174): SIGSEGV, segmentation fault occurred
```

Increasing the stack size with the command 'ulimit' failed to correct the problem on Ubuntu 12.0 LTS.

A workaroud when using the Intel compiler is to use the *-heap-arrays* option, however as of the *Version 13.0* release, there is a bug that causes a memory leak with this switch. Intel case number: DPD200235943. This means that the memory footprint of the program will gradually increase, and even a modest size model will run out of memory in under 5 iterations.

It is likely that Intel will release a fix for this soon. Explicitly allocating the arrays, by declaring them *allocatable*, should cause them to be allocated on the heap and thus we would not need the -heap-arrays command. However, since this issue was discovered towards the end of the project (and explains some of the issues that occured along the way), this is left for future work.

We will attempt to use the Intel compiler except where the stack size becomes an issue.

F.3 Configuring the environment for ifort

After installing ifort on a Ubuntu system, the command

```
source /opt/intel/bin/ifortvars.sh intel64
```

must be run in order to put certain Intel Fortran libraries on the path. It needs to be run before any binaries compiled by ifort without static linking (-shared-intel -mcmodel=medium) will run. It must be run whenever a new bash shell is started. This presents problems when running the Fortran files within Hadoop. When running in Pseudo-Distributed mode, it can be overcome by adding the line above to the file:

```
hadoop/conf/hadoop-env.sh
```

When running in fully-distributed mode, it would be necessary to install the ifort libraries, and similarly declare the variables in each of the config files; however this was not tested.

Bibliography

[1] Bradley Alexander. Private communication, June 2012.

[2] Bradley Alexander, Stephan Thiel, and Jared Peacock. Application of evolutionary methods to 3d geoscience modelling. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*, GECCO '12, pages 1039–1046, New York, NY, USA, 2012. ACM.

[3] Dhruba Borthakur, Jonathan Gray, Joydeep Sen Sarma, Kannan Muthukkaruppan, Nicolas Spiegelberg, Hairong Kuang, Karthik Ranganathan, Dmytro Molkov, Aravind Menon, Samuel Rash, Rodrigo Schmidt, and Amitanand Aiyer. Apache hadoop goes realtime at facebook. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, SIGMOD '11, pages 1071–1080, New York, NY, USA, 2011. ACM.

[4] Alan D. Chave and David J. Thomson. Bounded influence magnetotelluric response function estimation. *Geophysical Journal International*, 157(3):988–1006, 2004.

[5] Steven C. Constable, Robert L. Parker, and Catherine G. Constable. Occam's inversion: A practical algorithm for generating smooth models from electromagnetic sounding data. *Geophysics*, 52(3):289–300, 1987.

[6] J.K. Costain and C. Çoruh. *Basic Theory Of Exploration Seismology.* Handbook of Geophysical Exploration: Seismic Exploration. Elsevier, 2004.

[7] CD de Groot-Hedlin and SC Constable. Occam's inversion to generate smooth, two-dimensional models from magnetotelluric data. *Geophysics*, 55(12):1613–1624, 1990.

[8] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. In *Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6*, OSDI'04, pages 10–10, Berkeley, CA, USA, 2004. USENIX Association.

[9] H. DONG, G. YE, W. WEI, and S. JIN. 3d inversion testing of sparse field data using wsinv3dmt code.

[10] Gary D. Egbert and Anna Kelbert. Computational recipes for electromagnetic inverse problems. *Geophysical Journal International*, 189(1):251–267, 2012.

[11] SRL GEOSYSTEM. Winglink® user's guide. *GEOSYSTEM SRL, Milan, Italy*, 2008.

[12] Philip Mackey (Program Manager) Glenn Moloney (Project Director). National eresearch collaobration tools and resources, draft final project plan, May 2011.

[13] W. Heise, T. G. Caldwell, H. M. Bibby, and S. C. Bannister. Three-dimensional modelling of magnetotelluric data from the rotokawa geothermal field, taupo volcanic zone, new zealand. *Geophysical Journal International*, 173(2):740–750, 2008.

[14] N. Hoffmann, H. Jödicke, P. Gerling, et al. The distribution of pre-westphalian source rocks in the north german basin: evidence from magnetotelluric and geochemical data. *Netherlands Journal of Geosciences (Geologie en Mijnbouw)*, 80(1), 2001.

[15] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. Technical report, July 2009.

[16] Max Moorkamp, Björn Heincke, Marion Jegen, Alan W. Roberts, and Richard W. Hobbs. A framework for 3-d joint inversion of mt, gravity and seismic refraction data. *Geophysical Journal International*, 184(1):477–493, 2011.

[17] J. C. Mudge. Private communication, November 2012.

[18] J.C. Mudge, P. Chandrasekhar, G.S. Heinson, and S. Thiel. Evolving inversion methods in geophysics with cloud computing - a case study of an escience collaboration. In *E-Science (e-Science), 2011 IEEE 7th International Conference on*, pages 119 –125, Dec. 2011.

[19] Gregory A. Newman, Stephan Recher, Bülent Tezkan, and Fritz M. Neubauer. 3d inversion of a scalar radio magnetotelluric field data set. *Geophysics*, 68(3):791–802, 2003.

[20] K.H. Olsen. *Continental Rifts: Evolution, Structure, Tectonics: Evolution, Structure, Tectonics*. Developments in Geotectonics. Elsevier Science, 1995.

[21] 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.

[22] N. Rawlinson and BR Goleby. Seismic imaging of continents and their margins: New research at the confluence of active and passive seismology. *Tectonophysics*, 2012.

[23] L. Shklar and R. Rosen. *Web Application Architecture: Principles, Protocols and Practices*. John Wiley & Sons, 2003.

[24] Fiona Simpson and Karsten Bahr. *Practical magnetotellurics*. Cambridge University Press, 2005.

[25] W. Siripunvaraporn and G. Egbert. An efficient data-subspace inversion method for 2-D magnetotelluric data. *Geophysics*, 65:791, 2000.

[26] Weerachai Siripunvaraporn. Wsinv3dmt version 1.0.0 for single processor machine: User manual. User Manual, 2006.

[27] Weerachai Siripunvaraporn. Three-dimensional magnetotelluric inversion: An introductory guide for developers and users. *Surveys in Geophysics*, pages 1–23, 2011.

[28] Weerachai Siripunvaraporn and Gary Egbert. Data space conjugate gradient inversion for 2-d magnetotelluric data. *Geophysical Journal International*, 170(3):986–994, 2007.

[29] Weerachai Siripunvaraporn and Gary Egbert. Wsinv3dmt: Vertical magnetic field transfer function inversion and parallel implementation. *Physics of the Earth and Planetary Interiors*, 173:317 – 329, 2009.

[30] Weerachai Siripunvaraporn, Gary Egbert, Yongwimon Lenbury, and Makoto Uyeshima. Three-dimensional magnetotelluric inversion: data-space method. *Physics of the Earth and Planetary Interiors*, 150:3 – 14, 2005.

[31] Weerachai Siripunvaraporn and Weerachai Sarakorn. An efficient data space conjugate gradient occam's method for three-dimensional magnetotelluric inversion. *Geophysical Journal International*, 186(2):567–579, 2011.

[32] Edward Stewart. Leveraging the nvidia cuda blas in the imsl fortran library. Technical report, Rogue Wave Software, 2010.

[33] S. Thiel. *Modelling and Inversion of Magnetotelluric Data for 2-D and 3-D Lithospheric Structure, with Application to Obducted and Subducted Terranes*. University of Adelaide, School of Earth and Environmental Sciences, Discipline of Geology and Geophysics, 2008.

[34] S. Thiel. Private communication, November 2012.

[35] T. White. *Hadoop: The Definitive Guide*. Oreilly and Associate Series. Oreilly & Associates Incorporated, 2012.

[36] Qibin Xiao, Xinping Cai, Xingwang Xu, Guanghe Liang, and Baolin Zhang. Application of the 3d magnetotelluric inversion code in a geologically complex area. *Geophysical Prospecting*, 58(6):1177–1192, 2010.