Parallel Processing Methods Applied to Two and Three Dimensional Geo-electromagnetic Induction Modelling

Kenneth J MacDonald

A thesis submitted in fulfilment of the requirements
for the degree of Doctor of Philosophy
to the
University of Edinburgh
1996

Abstract

Two existing finite difference algorithms for solving the forward modelling problem of geoelectromagnetic induction have been recoded to take advantage of high performance massively parallel SIMD (single instruction multiple data) computer architectures. Poll's[48] solves the two scalar polarised fields in the two dimensional (2D) problem, and the other from Pu[51] solves for all three components of the magnetic field in three dimensional (3D) structures. Both models apply integral boundary conditions at the top and bottom of the grid to limit total mesh size. The 3D model introduces a thin sheet at the top of the model to describe near surface features. An efficient data parallel algorithm ensures the evaluation of the integrals maintains a high ratio of processor utilisation on the parallel hardware. Data parallel versions of the point Jacobian, Gauss-Seidel and successive overrelaxation iterative solvers have been developed. The latter two require two level black-white ordering, which to equalise the processor load balance, has been implemented it both a horizontally banded and chequer boarded remapping of grid nodes.

The 2D model was also developed to form a task farm, whereby the solution for each period is performed on one of a cluster of workstations. These solutions are independent of each other, so are executed simultaneously on however many workstations are available at the time.

Modern workstations, coupled with the original 2D Gauss-Jordan solver, are faster than the SIMD computers for all but the largest grid sizes. However, the 3D code certainly benefited from the parallel processing for any but the smallest models.

A new automatic meshing algorithm, which stretches a predefined number of grid points over the conductivity structure, has also been developed. In part, this was to control the mesh sizes and hence load balancing on the SIMD computers, but investigations into grid spacing for 2D models show that severely restricting the number of grid points results in a much faster estimated solution.

Acknowledgements

I would like to thank my supervisor Dr. Bruce Hobbs for his unfailing patience and encouragement over the duration of my studies, as an undergraduate and postgraduate. His enthusiasm for life as well as science has helped enormously in shaping my own approach to this study. Thanks must also go to my secondary supervisors Dr. Roger Banks and Mr Graham Dawes for their help and advice.

This work was carried out with the financial support of N.E.R.C. under studentship number GT4/91/GS/41, for which I am grateful.

A would also like to extend a special thank you to Dr Rosemary Hutton, lately of The University of Edinburgh, for introducing me to the subject of magnetotellurics and for her faith in and support of me as an undergraduate.

The geo-electromagnetic modelling group at the University of Victoria, Canada led by Dr. John Weaver are to be graciously thanked for inviting me to join their team for three months. Ashok Agarwal, Xinghua Pu and Helena Poll were not only excellent academic hosts, but they also made my stay away from home most enjoyable.

Credit should undoubtably go to my friends and family, who in combination, kicked me into completing this thesis and supported me through the difficult as well as good times.

Finally, I would like to thank my fiancée, Helena Taborda, whose faith in me and support has been, and always will be, a source of strength and encouragement.

Contents

List of Tables

List of Figures

Constants, Symbols and Nomenclature

μ_0	Magnetic permeability of free space ($4\pi \times 10^{-7}$ henry m^{-1}).
ε_0	Electrical permittivity of free space (8.85×10^{-12} Farads m^{-1}).
$\mathbf{B}$	Magnetic field.
$\mathbf{E}$	Electric field.
$\boldsymbol{F}$	Vector field, either $\mathbf{B}$ or $\mathbf{E}$.
X, Y, Z	$\mathbf{B}_x$, $\mathbf{B}_y$, $\mathbf{B}_z$
U, V, W	$\mathbf{E}_x$, $\mathbf{E}_y$, $\mathbf{E}_z$
F	Scalar field, either X or U.
$\tilde{F}$	Anomalous scalar field.
F^-, F^+	One dimensional scalar field solutions at the left and right edges of the model.
$\boldsymbol{u}$	Discrete representation of a scalar field.
$\boldsymbol{u}^{(p)}$	The p^{th} iteration level of $\boldsymbol{u}$.
$\Delta^{(p)}(\boldsymbol{u})$	Some measure of change between $\boldsymbol{u}^{(p)}$ and $\boldsymbol{u}^{(p-1)}$.
$u_{i,j}$	The scalar value of $\boldsymbol{u}$ at the grid node (y_i, z_j).
$\boldsymbol{\nu}$	$\boldsymbol{u}$ mapped on to a processor array.
$a \leftarrow b, c$	Value of a depends on the values of b and c.
η	Elastic coefficient for *stretched* grid calculations.
ζ	Net stress at a point in the *stretched* grid.

Chapter 1

Introduction

"Now here's the point, my friend. Electricity, radioactivity, atomic energy - the true initiate knows that these are metaphors, masks, conventional lies, or, at most, pathetic surrogates, for an ancestral, forgotten force, a force the initiate seeks and one day will know. We should speak perhaps" – he hesitated a moment – *"of telluric currents."*

"What?" one of us asked, I forget who.

— Signor Agliè in *Foucault's Pendulum* by Umberto Eco, 1988.

1.1 Historical Review

Although the Ancient Greeks recognised electricity, it is a Dane, Hans Christian Oersted (1777 – 1851) who is credited with the initial discovery of a link between electricity and magnetism in 1819. During a lecture, he placed a segment of conducting wire horizontally, and *at right angles* above a magnetic needle. Of course, nothing happened, and he continued the lecture without further thought on the result. However, while moving the apparatus at the end of the lecture, he noted, to his amazement, that if the wire was placed parallel to the needle, it would cause a 90° deflection in the needle.

It was only two years later that Michael Faraday (1791 – 1867), in his famous Christmas Day experiment of 1821, built the first prototype of the electric motor.

He continued his work, with the aim of finding a reciprocal link to Oersted's that show electric currents induced by magnetic fields. He built what is now known as an induction coil, and expected to find current induced in the secondary coil while current flowed in the primary. There was no such effect visible, but he noticed that whenever the current connection was made or broken, the galvanometer gave a slight kick. He quickly satisfied himself that this was the effect he was looking for, and became the father of electromagnetic induction.

Faraday's Law states that an electric current will be induced in a conducting material in the presence of a time varying magnetic field, and that the magnitude of the current will be proportional to the rate of change of the magnetic field. The converse is also true, in that a time varying electric current will result in an associated magnetic field. This phenomenon is most commonly observed in the multitude of electrical transformers and motors which surround us in modern life.

The same is also true on a global scale. The Earth has a finite, albeit inhomogeneous, electrical conductivity, and experiences a range of time varying magnetic fields. The electrical currents induced in the Earth are known as *telluric* currents. Schuster (1851 – 1934) was the first to separate the internal and external magnetic fields in 1889[57], and therefore the first to prove the existence of geo-electromagnetic induction. Table 1.1 details some of the main natural sources of geomagnetic field variations, along with their characteristic time scales.

Variation	**Time scale**
Main field reversals	10^6 Years
Reversal events	10^5 Years
Non-dipole field and secular variations	10 – 1000 Years
Regular magnetic storm activity	0.5 – 4 Days
Diurnal variations	24, 12, and 8 Hours
Pulsations	0.2 – 600 Seconds
Sferics	3 – 1 000 Hertz

Table 1.1. Time scales of natural geomagnetic variations.

The power spectrum of the main field reversals is not known, and since none have occured in historical times, they are not used for induction studies, but for tracing the tectonic history of ocean floor, where they leave the basalt magnetised parallel to the field at the time of cooling.

The non-dipole field and secular variations tell us more about the kinematics of the source of the main field, than the conductivity structure of the Earth. The long period of changes results in a very weak induced electric field, which combined with the difficulty of making accurate measurements of telluric currents over such long periods, means that they are not relevant for induction studies.

The more frequent disturbances are all used as natural power sources for geo-electromagnetic studies. As a rule of thumb, which will be shown mathematically later, the higher the frequency of the source, the finer the resolution and the shallower the penetration. Thus the diurnal variations allow global conductivity studies, and the high frequency spectra excited by thunder storms make excellent sources for regional magnetotelluric measurements.

1.2 Basic Electromagnetic Theory

James Clerk Maxwell (1831 – 1879) was a mathematician, unlike Faraday who confessed to an elementary understanding of mathematics. He preferred geometrical rather than analytic methods of solution, and when he heard of Faraday's lines of force, the geometry attracted him and he began to put Faraday's ideas into a mathematical form. His equations completely describing the interaction between electric and magnetic fields can be found in any elementary textbook on electromagnetism. In a linear, isotropic medium of uniform dielectric permittivity ε and magnetic permeability μ, they are, in S.I. units

$$\nabla \cdot \boldsymbol{E} = \frac{\rho}{\varepsilon} \tag{1.1}$$

$$\nabla \cdot \boldsymbol{B} = 0 \tag{1.2}$$

$$\nabla \times \boldsymbol{E} = -\frac{\partial \boldsymbol{B}}{\partial t} \tag{1.3}$$

$$\nabla \times \boldsymbol{B} = \mu \boldsymbol{j} + \mu\varepsilon \frac{\partial \boldsymbol{E}}{\partial t} \tag{1.4}$$

where ρ is the electric charge volume density, not to be confused with the electrical resistivity which is denoted by ρ in the rest of this work. $\boldsymbol{j}$ is the electric current volume density, which in a source free medium of electrical conductivity σ is

$$\boldsymbol{j} = \sigma \boldsymbol{E}. \tag{1.5}$$

The most striking consequence of his equations is that changes in electric and magnetic fields can be propagated as *waves*, with a velocity the same as that which had been measured of light. He wrote

> "We can scarcely avoid the inference that light consists in the *transverse undulations* of the *same medium* which is the cause of electric and magnetic phenomena."

thus being the first to recognise light as an electromagnetic wave.

Taking the curl of (1.3) and substituting from (1.4) and (1.5) leads to

$$\nabla \times (\nabla \times \boldsymbol{E}) + \mu\sigma \frac{\partial \boldsymbol{E}}{\partial t} + \mu\varepsilon \frac{\partial^2 \boldsymbol{E}}{\partial t^2} = 0. \tag{1.6}$$

Following the approach of Weaver[73], and introducing dimensionless coefficients $\nu_1 = \mu\sigma L^2/T$ and $\nu_2 = \mu\varepsilon L^2/T^2$ (L and T being characteristic length and time scales respectively), with $l' = l/L$, $t' = t/T$ and ∇', the grad operator with respect to the dimensionless space variables, (1.6) can be written

$$\nabla' \times (\nabla' \times \boldsymbol{E}) + \nu_1 \frac{\partial \boldsymbol{E}}{\partial t'} + \nu_2 \frac{\partial^2 \boldsymbol{E}}{\partial t'^2} = 0. \tag{1.7}$$

with the relative values of ν_1 and ν_2 determining the importance of each term.

The dielectric permittivity, ε, does not vary appreciably from that of free space for the vast majority of materials in the Earth, except that of water, which is 80 times greater. The magnetic permeability, μ, only varies from that of free space in concentrated highly magnetic minerals, such as magnetite (5 times greater). Even in hematite it is only 5% greater[61]. The constants can therefore be taken to be their free space values, namely

$$\varepsilon = \varepsilon_0 = 8.85 \times 10^{-12} \text{ F/m and } \mu = \mu_0 = 4\pi \times 10^{-7} \text{ H/m} \tag{1.8}$$

where $\mu\varepsilon = 1/c^2$ (c is the velocity of light). It therefore follows that $\nu_2 = (L/cT)^2$ and $\nu_1/\nu_2 = \mu\sigma c^2 T$.

In the case of global studies, where L is of the order of 10^7 m, then $\nu_2 \ll 1$ for time variations of less than 3 Hz ($\nu_2 < 0.01$). Taking $L \approx 10^4$ m (10 km) for regional studies, the same is true for frequencies of less than 3 kHz. The electrical conductivity of rocks varies between 10^{-6} to 10^{-4} S/m, and 4 S/m for sea water. Therefore, the ratio ν_1/ν_2 , will be much greater than one inside the Earth for frequencies less than 100 kHz. In the case of the air, where $\sigma = 0$, and hence $\nu_1 = 0$, the first term will still dominate due to the fact that ν_2 is very much less than unity.

The third term in (1.7) can therefore be dropped, which is equivalent to dropping the second term on the right hand side of (1.4), now written as

$$\nabla \times \boldsymbol{B} = \mu\sigma\boldsymbol{E} \tag{1.9}$$

where the definition of $\boldsymbol{j}$, (1.5), has been incorporated. Taking the divergence of (1.9) and noting that $\nabla \cdot (\nabla \times \boldsymbol{A}) = 0$, it is shown that

$$\nabla \cdot \boldsymbol{j} = 0. \tag{1.10}$$

In regions of homogeneous conductivity, therefore, (1.1) can be written

$$\nabla \cdot \boldsymbol{E} = 0 \tag{1.11}$$

which shows that finite electrical charge densities can only exist in those geo-electromagnetic conditions where there are spatial changes in conductivity.

It is assumed that the sources share a common harmonic time dependence with angular frequency ω so that the fields can be written

$$\begin{aligned} \boldsymbol{E} &= \boldsymbol{E}(x,y,z)e^{i\omega t} && (1.12)\\ \boldsymbol{B} &= \boldsymbol{B}(x,y,z)e^{i\omega t}. && (1.13) \end{aligned}$$

Using these harmonic properties dropping the third term, (1.6) can now be rewritten

$$\nabla \times (\nabla \times \boldsymbol{E}) + i\omega\mu\sigma\boldsymbol{E} = 0. \tag{1.14}$$

The vector calculus identity $\nabla \times (\nabla \times \boldsymbol{A}) = \nabla(\nabla \cdot \boldsymbol{A}) - \nabla^2\boldsymbol{A}$, allows us to write (1.14) as

$$\nabla^2\boldsymbol{E} + \nabla[(\nabla\sigma) \cdot \boldsymbol{E}/\sigma] = i\omega\mu\sigma\boldsymbol{E}. \tag{1.15}$$

In areas of constant conductivity, such as these inside our model cells, applying (1.11) leads to

$$\nabla^2\boldsymbol{E} = i\omega\mu\sigma\boldsymbol{E}, \tag{1.16}$$

the electrical diffusion equation.

A similar approach for $\boldsymbol{B}$ leads to

$$\nabla^2\boldsymbol{B} = i\omega\mu\sigma\boldsymbol{B}, \tag{1.17}$$

the magnetic diffusion equation.

In his now classic paper of 1950, Price[50] analysed the general problem of a uniform half space excited by a known source field above it.

1.2.1 Dimensionality

This thesis is concerned with the numerical modelling of two and three-dimensional resistivity structures. The one dimensional case is indirectly addressed, since it must be solved in order to set boundary conditions for two dimensional models.

The simplest geological structure to model is known as a *half-space*. This is defined by a horizontal plane at $z = 0$, which separates a non conducting vacuum for $z < 0$ and uniformly conducting material for $z > 0$. One dimensional structures only vary their properties with depth, such as a series of horizontal layers, as shown in Figure 1.1(a).

Many geological structures are inherently two-dimensional, in that they do not vary in character along one particular direction, known as the *strike* (along the x-axis in Figure 1.1(b)). This premise holds true for a large range of scales.

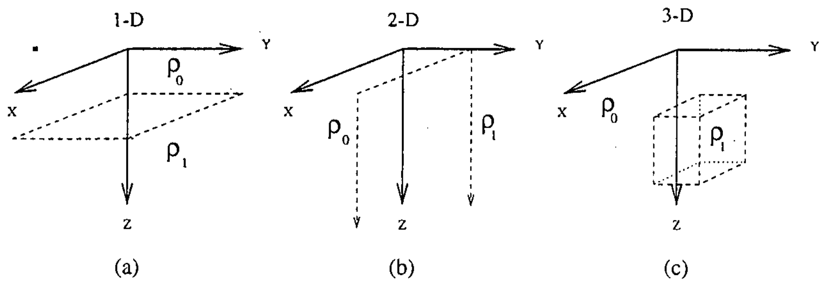

Figure 1.1. One, Two and Three Dimensional Structures

For example, faults and dykes tend to run in straight lines, and even coastlines can often be considered to be straight over hundreds of kilometers. Therefore we can restrict the variation in model structure and reap the benefits in the resulting reduction in the degrees of freedom of the problem.

The general case is of a fully three-dimensional conductivity structure. The three degrees of freedom allow a complete description of any geological formation and setting. Figure 1.1(c) illustrates a block embedded in a homogeneous host rock.

1.2.2 The Magnetotelluric Method

The origins of the magnetotelluric method can be found in the classic papers of Tikhonov[63] & Caignard[7]. Natural source fields over a broad frequency band, for example the last four entries in Table 1.1, are used to explore the subsurface conductivity. Time series of the horizontal components of both the electric and magnetic fields are measured at appropriate sites, and Fourier transformed into power spectra in the frequency domain. Caignard showed that if the region has a one-dimensional conductivity structure, then an *apparent resistivity* is defined as

$$\rho_a = \frac{\mu_0}{\omega} \frac{|\mathbf{E}|^2}{|\mathbf{B}|^2} \tag{1.18}$$

where μ_0 is the permeability and ω is the frequency. In the two-dimensional case the apparent resistivity depends on the direction of **B** and **E** so that the two possible values are

$$(\rho_a)_x = \frac{\mu_0}{\omega}\frac{|\mathbf{E}_x|^2}{|\mathbf{B}_y|^2}, \text{ and } (\rho_a)_y = \frac{\mu_0}{\omega}\frac{|\mathbf{E}_y|^2}{|\mathbf{B}_x|^2}. \tag{1.19}$$

In practical electromagnetic units, Caignard obtained

$$\rho_a = \frac{\tau}{5}\left|\frac{\mathbf{E}_x}{\mathbf{H}_y}\right|^2 \tag{1.20}$$

for the true resistivity of a half space, where **E** is measured in millivolts per kilometre, the electrodes being separated by typically one hundred metres, **H**, the magnetic field, in nanotesla and τ, the period, in seconds. Conveniently, the apparent resistivity then has units of ohm metres.

This method of geophysical prospecting has been applied to a variety of geological structures. For example, the EMSLAB[16][68][3] project studied the deep continental crust of North America on the regional scale. Magnetotellurics are frequently employed to evaluate potential geothermal regions, such as Jones *et al.*[28] in Southern Portugal, which discounted possible development, or the more successful studies of Lagios *et al.*[35] and Galanopoulos *et al.*[21]in the Greek Aegean islands.

1.3 Review of Geo-electromagnetic Modelling

Forward modelling is defined to be the process of deriving a set of electromagnetic field values, or response functions, from a given resistivity structure. The process of deducing a resistivity from a given response function, usually derived from real measurements, is known as *inversion.*

Inverse methods exist for one dimensional problems, such as those of Bailey[2] for a spherical Earth and Weidelt[77] for a flat Earth. These derive a conductance structure directly from the surface response functions. However, two and three dimensional inversion methods rely on calculating multiple forward models and somehow minimising the misfit between the modelled and measured responses. deGroot-Hedlin and Constable's[12] two dimensional *Occam* inversion searches for the smoothest varying conductivity structure. They discretise the model on a dense mesh and allow each cell to vary in conductivity, but not size. This mesh is then used as a rough grid for Wannamaker's[70] two dimensional finite element code. Agarwal and Weaver[1] prefer to progressively add structural blocks until the data are matched, thus arriving at a different, but equally valid, definition of *minimum complexity* model. Smith and Booker[59] have developed what they term a "rapid inversion" algorithm for two and three-dimensional structures. They perform a series of one dimensional inversions under each measurement site, which they interpolate to form a new input grid for the two or three dimensional forward model.

Such algorithms may require many forward models to be calculated, thus fast and accurate forward modelling is an essential tool for the exploration of the electrical conductivity of the Earth.

There are three distinct approaches to forward modelling geo-electromagnetic induction:

Analogue Construct physical scale models in the laboratory, apply a source field, and measure the field components at the site locations.

Analytic A series of mathematical formulæ can give the exact solution to a particular problem.

Numerical Approximate reality with a discrete, digital representation of the relevant physical properties, construct some system of equations which describe the electromagnetic field behaviour, and solve.

All of these techniques have undergone development since the earliest attempts to model the real world. Analogue modelling has the advantage of being able to represent arbitrarily complex physical structures, limited only by the skill and resources (financial and time) of the modeller. Dosso[14] presented a general review in 1973, Hu *et al.*[26] examine sea mount effects, and Chen *et al.*[8] investigate tectonic subduction zones. Chen[9] also constructed analogue models for the EMSLAB region. However, even minor alterations to the model require significant amounts of time, ruling this technique out for iterative inversions of in field recorded measurements. The scarcity of suitable materials with laboratory scale conductivities can also limit the construction of models.

The beauty and appeal of analytic models lie in their ability to give precise predictions of the electromagnetic field at the site locations. They are however, severely limited to a few special case conductivity structures and source configurations. d'Erceville and Kunetz[13] obtained the first analytic solution for lateral conductivity variations in 1962 by considering a vertical contact between two blocks, overlying a perfect conducting or insulating half space. Rankin[54] introduced another vertical boundary to construct a dyke in a homogeneous host rock, and Weaver[71] allowed the contacts to extend infinitely downwards. Perhaps the most useful application of such analytic models, is that they can be compared to numerical models of the same conductivity structure and source field. The first COMMEMI[85] model, as shown later in Figure 5.6(0) has an analytic solution, discovered by Weaver, LeQuang and Fischer[75][76].

Numerical models exhibit the greatest variety and effort of the three modelling families. As computers become able to handle larger sets of numbers more quickly, so the numerical models become more complex and finer scaled. Kaikkonen[31] reviews a variety methods. These generally involve superimposing a mesh of grid points over the conductivity structure to be modelled, and describing the local field behaviour in each block with a system of equations. This system, along with suitable boundary conditions is solved to obtain the modelled response.

Neves[43] developed the first numerical model using finite differences, which were later used by Jones and Price[29] in two dimensions and by Jones and Vozoff[30] in three dimensions. Wannamaker[70] developed a two dimensional finite element algorithm which has been integrated into the commercial Geotools magnetotelluric interpretation package[22]. Coggon[10] was perhaps first to apply this method to the geo-electromagnetic induction problem. Many others have followed, such as Reddy and Rankin[55], who took advantage of the method's ability to operate on non-rectangular meshes to model dipping contacts. Madden[38] and Ku *et al.*[33] modelled the conductivity structure as a network of resistors and inductors to which they applied Kirchoff's Laws of voltage and current.

Integral equation methods have proved popular in three dimensional modelling, where the large meshes required by finite elements and differences prove impossible to accommodate on typical computers. These methods calculate the anomalous fields inside inhomogeneous regions in the country rock, therefore requiring a much reduced mesh so long as the number of such regions is not too great. Raiche[52], Weidelt[78] and Hohmann[25] laid the groundwork for this particular technique.

When the main model complexity lies in the upper surface of the model then a thin sheet approach may be followed. Price[49] originally introduced this idea, whereby a physically thin region of arbitrarily varying conductivity is approximated by a sheet of zero thickness and conductance equivalent to the integrated original conductivity. Vasseur and Weidelt[65] allowed a general conductivity variation, but constrained by a surrounding uniform area, which was relaxed by McKirdy and Weaver[39], and McKirdy, Weaver and Dawson[40] who introduced two dimensional structures at the boundaries.

There is a great deal of effort concentrating on the three dimensional problem now, with the rapid development of high speed, large capacity computers. Pu[51] has developed a hybrid thin sheet and finite difference code which offers the potential to model virtually any geological structure, given a suitably powerful computing facility.

1.4 Review of work in this thesis

This thesis investigates possible approaches to take advantage of parallel processing methods in geo-electromagnetic modelling. There has been rapid development in the field of concurrent computing which is reviewed in Chapter 2.

Chapter 3 introduces two existing model algorithms which were adopted for this study. Poll's[48] two dimensional code, complete with automatic mesh generation, and Pu's[51] elaborate three dimensional code.

A massively parallel implementation of these models algorithm is developed and evaluated in Chapter 4. The same source code compiles either for a Thinking Machines CM 200 at Edinburgh, or a DEC mpp/12000SX in Victoria, British Columbia. This chapter also describes an alternative approach to parallelisation, utilising a cluster of workstations configured as a task farm, which is portable to a wide variety of platforms.

Some of the issues raised in Chapter 4 called for the development of a new automatic meshing algorithm. A description of past and present meshing techniques is presented in Chapter 5, along with a *stretched grid* algorithm which controls the number of nodes allocated to a model mesh. Comparisons of several different grids indicate where gridding can be relaxed.

Finally, Chapter 6 closes this thesis with a discussion of the results and suggestions for where future effort would be best applied.

Chapter 2

Parallel Processing

2.1 History of Parallel Processing

Far from being a recent technological development, the story of parallel processing is as old as data processing itself. As soon as people started to perform calculations on data, the concept of sharing the work, or "divide and conquer" surely occured to them.

With the faltering birth of mechanical computing in the nineteenth century, Menabrea (1842) wrote about Charles Babbage's Analytical Engine

> ... when a long series of identical computations is to be performed, such as those required for the formation of numerical tables, the machine can be brought into play so as to give several results at the same time, which will greatly abridge the whole amount of the processes.

Although the ideas of parallel processing surfaced from time to time, large scale parallelism remained unrealistic with the available technology. Wallace[67] describes how Lewis F. Richardson (1922) proposed that 64,000 human computers could calculate the weather for the whole globe, if they were 'coordinated by an official of higher rank'. His fantasy is illustrated in Figure 2.1.

Parallel processing, in its simplest form, also appeared during the race to construct an atomic bomb during the Second World War. When Richard Feynman was given the task of managing a large group of human particular algorithms.

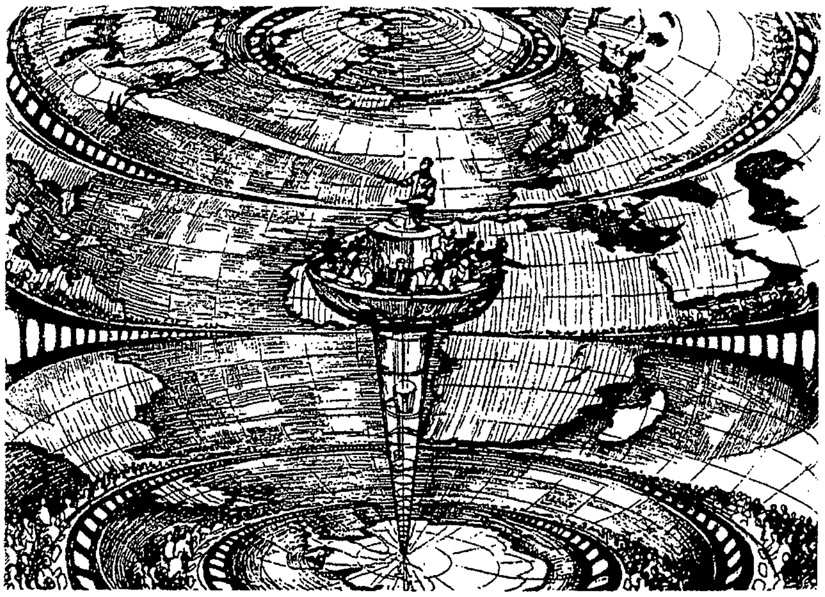

Figure 2.1. Richardson's scheme for numerical weather prediction by human parallel computers, as shown by Wallace (1988)

His superiors at Los Alamos, including Professor Pauli, were amazed at the time saved by Feynman's practices. Before his death, Richard Feynman completed the mathematical theory which enabled his friend Danny Hillis to build the Connection Machine, and found Thinking Machines Corporation, one of the major pioneers in parallel hardware.

The development of the earliest electronic computers in the post war years effectively represented a step away from these early ideas of parallel processing. EDSAC 1 (1949) and UNIVAC (1951) implemented a form of instruction pipelining in a classical von Neumann computer model, as shown in Figure 2.2.

In the forty-five years since EDSAC 1, there has been a steady twenty-five fold increase in floating point performance each decade. This has been achieved through advances in engineering and hardware technology, and until very recently, each generation of supercomputers has followed the Von Neumann model.

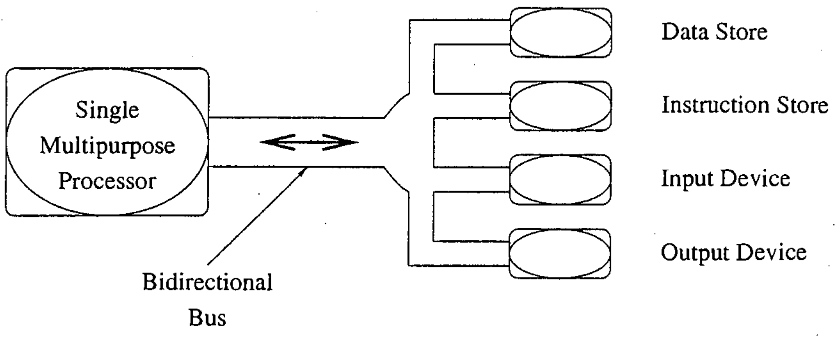

Figure 2.2. Classical Von Neumann Computer Architecture

In 1947 John Von Neumann proposed that an electronic general purpose programmable computer could be built using a single processing unit which communicates through an array of electrical connections known as a bus. Everything else required to operate the computer is attached to this bus.

The 1980s brought very large-scale integrated circuit multiprocessors, with directly connected memory, distributed between the processors, and/or acting as a global memory, accessible equally from all processors. Various network topologies have been designed for inter-processor communication pathways, from the simplistic lattice, through rings and toruses, to hypercubes.

2.2 Differing Computer Architectures

The taxonomy covering the variety of parallel architectures is as fluid as the hardware itself, due to the rapid development witnessed over the last couple of decades. Flynn's (1966) classification, described by Modi[41], and reproduced in Table 2.1 is commonly accepted. He identifies four types of computational architecture: SISD, SIMD, MISD and MIMD[1].

The MISD design highlights the age of this classification, and it is doubtful if any machines of this architecture were ever built. Today, only the multiple data

[1]Each is pronounced by making the word fragment of the first three letters and then adding a *"dee"* sound. For example SIMD $\rightarrow$ *"sim-dee"*.

	Single Data Stream	Multiple Data Stream
Single Instruction Stream	SISD	SIMD
Multiple Instruction Stream	MISD	MIMD

Table 2.1. Flynn's classification

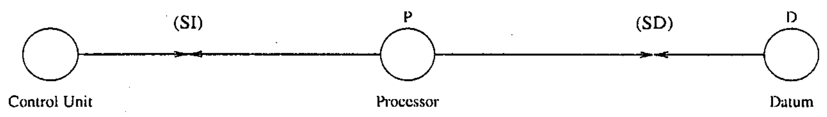

Figure 2.3. The SISD computer architecture

stream types are considered appropriate to describe parallel computers.

2.2.1 The SISD Computing Model

We will briefly concern ourselves with the SISD model, as it obviously represents the traditional serial computer. This will lay the basis for further understanding the novel features of parallel computers.

The SISD architecture, see Figure 2.3, is analogous to the von Neumann. In this design there exists one processing unit, one memory storage unit, and one bus linking the two. In operation, the processor fetches one instruction from the memory, followed by a datum (determined by the instruction) from the same memory. The necessary calculation is performed, and the result is placed back in memory.

At the time, primitive valves were fragile and unreliable, so this was a practical approach to building any computer, in that the von Neumann computer only had one of each item.

The following sections describe the two remaining, SIMD and MIMD, models, and introduce the particular machines on which the work presented in this thesis was carried out. The common features will be described in the SIMD section, and then the differences between the two will be highlighted in the MIMD section.

2.2.2 The SIMD Computing Model

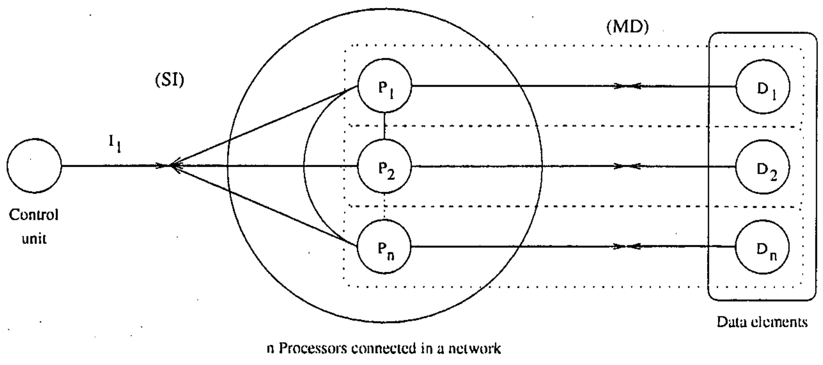

Figure 2.4. The SIMD computing model

SIMD is an acronym for Single Instruction, Multiple Data, as noted above in Flynn's classification (Table 2.1). This is often referred to *Data Parallel*computing, in that it is the data which is distributed, not the program. One distinguishing feature of many SIMD computers is that they are built of large numbers of processors, thousands, or even tens of thousands, working synchronously, i.e. locked to one clock.

The SIMD Control Unit

Because there can be only one instruction active at any time (Single Instruction), a powerful processor, generally a workstation, is promoted to be the controller. It is this, and only this, processor which can issue instructions to all the others.

Instructions may be conditional on data local to the processors. For example, a ratio may be calculated so long as the denominator is non-zero. However, any processor which fails the test will be idle for that computational cycle, reducing overall efficiency. This is one of the main drawbacks of the SIMD architecture.

All external data I/O, such as disc storage and networking, is routed through this control unit.

The SIMD Memory Model

Each processor has a relatively small bank of local memory, but when scaled by the number of processors, the total amount of memory available becomes significant. Only the local processor may access this memory, and in many respects it is the extent of this memory which dictates the maximum dimensions and *shape* of the problem which can be posed on the entire machine.

Communications

There are several forms of communications which may take place in a SIMD machine:

- Control unit communicates data or instructions with one processor. All other processors remain idle.
- Control unit broadcasts data or instructions to all processors, each of which stores a copy of the data in its own local memory.
- Arrays, or sections of arrays, pass between the control unit and the processors.
- Data reduction to control unit. Data from all, or a subset, of the processors is reduced to a scalar value which is received by the control unit.
- Regular inter-processor communication, in which data is moved homogeneously across the processors.
- General inter-processor communication, where each processor is communicating with any other processor, addressed by a pointer stored in local memory.

Topology

The topology of the architecture describes the configuration of the communication network connecting the processors together. Inter processor communications will obviously be more efficient if the data passes through the minimum number of

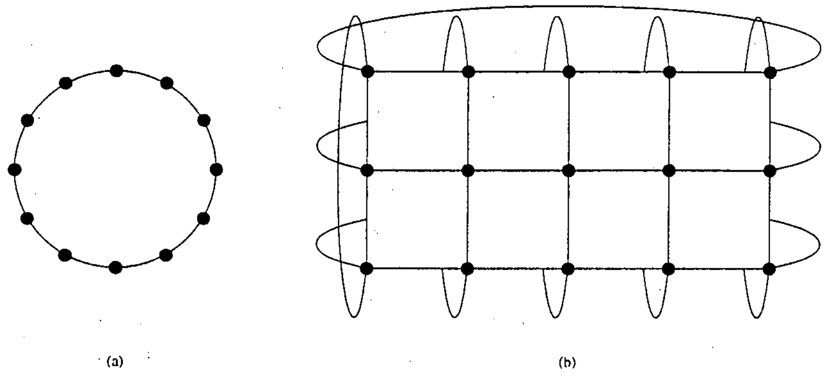

Figure 2.5. Lattice Topologies in (a) one dimension, and (b) two dimensions

processors on its route to its destination. The dimensionality of the network also affects the mapping of model parameters to processors.

An n-dimensional lattice is the simplest network topology. Figure 2.5 shows two lattice networks, in one and two dimensions. The last node has a direct connection back around to the first to allow for fast data wraparound. However, any two processors in this configuration may be separated by many intermediate processors.

A novel networking scheme has been developed to overcome this problem. An n-dimensional *hypercube* is constructed such that each of the 2^n nodes is connected to n neighbouring nodes. This has the advantage that the processor array is at most n processors long in any direction. Figure 2.6 illustrates how $(n + 1)$ dimensional hypercubes can be constructed from two n dimensional hypercubes connected together by an extra $2^{(n-1)}$ lines.

2.2.3 The MIMD Computing Model

In contrast to SIMD computers, MIMD machines have traditionally consisted of many fewer processors, tens or hundreds. This is undoubtedly mainly due to the expense of the more powerful processors utilised, but also the difficulty of building a network with a high enough 'bandwidth' to support large numbers of such processors. The bandwidth of any communication route is simply a measure

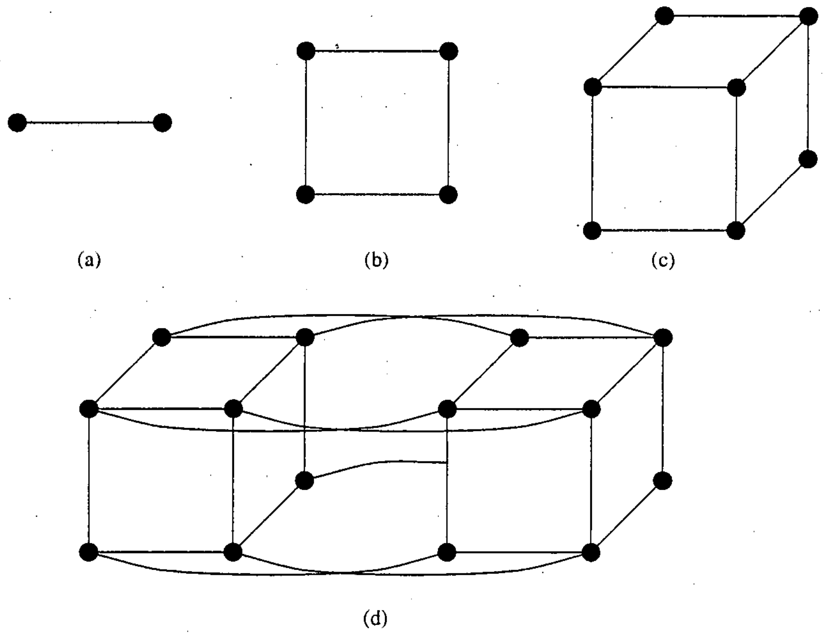

Figure 2.6. Hypercube Topologies in (a) one dimension, (b) two dimensions, (c) three dimensions, and (d) four dimensions

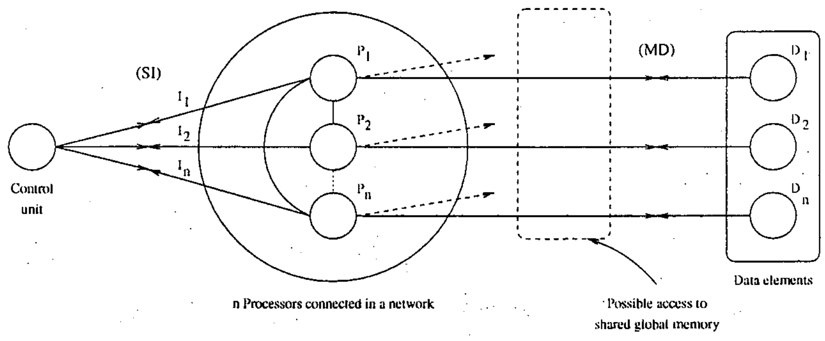
(SI)
I1
I2
In
Control unit
P1
P2
Pn
n Processors connected in a network
(MD)
Possible access to shared global memory
D1
D2
Dn
Data elements

Figure 2.7. The MIMD computing model

of how much information it can carry in a fixed time.

However, to overcome the lack of processor numbers, each processor is typically as powerful as a single processor workstation, and can be running code independently of all the others.

The MIMD Control Unit

There is still a need to delegate one, or more, processors to control the machine as a whole. Subsets of processors may be grouped together, along with their control processor, to form an independent MIMD machine within the global machine. At least one processor also acts as an interface to the external I/O channels. Whereas the SIMD control unit sends microcode instructions, the MIMD control unit sends program fragments to the set of processors under its control. Each processor is then free to complete its task in however long it may require.

The MIMD Memory Model

As in a SIMD machine, each of the processors will have its own local memory, scaled to match the greater processing power. Once again this is typically of the same order as a workstation : up to tens of megabytes. Some of this is reserved by the operating system, since a copy is running on each processor.

Figure 2.7 also shows an optional area of shared global memory, which is not associated with any particular processor. In fact, each processor is equally able to read from, or write to, this memory. Obviously, these operations are more expensive than local memory accesses.

Communications

Because each processor is, in effect, running a different program, communications are much more complicated in the MIMD model. The type and quantity of data is not fixed over the set of processors. Indeed, the data may in fact be a control message sent from one processor to another, unlike the SIMD machine.

The guaranteed synchronisation present in a SIMD machine cannot be relied on. Of course, the program can arrange the computations to be synchronous, but non-synchronous communications have to be catered for. Therefore, the range of communication modes can be summarised by the following selection.

- Synchronous, where all the processors are communicating simultaneously, which is the simplest form of communications. However, this places restrictions on the algorithm, similar to these found in the SIMD machine.
- Blocking non-synchronous, where processors are free to communicate whenever they require, but wait for the communication to complete before continuing their task. Unfortunately, this can result in inefficiencies in the program, as many processors may be idle at any one time. Even worse, there may develop a situation known as *deadlock* when the majority of processors are blocking, almost inevitably leading eventually to all processors being in the blocked state.
- Non-blocking non-synchronous, where processors continue their task once a communication is initiated. The processor will complete the communication at a later, undetermined time. This has the disadvantage that corrupt data are difficult to deal with, since once they have been sent, the transmitting process may have destroyed the originals.

2.3 Load Balancing and Granularity

The problem of keeping all the processors in a parallel machine busy is known as *load balancing*, and has already been alluded to in the earlier discussion of the SIMD and MIMD architectures.

Load balancing is intimately linked to the relationship between the *granularity* of the algorithm and the number of available processors. Simply put, the granularity is a measure of how fine, or small, the tasks given to each processor are. In a massively parallel SIMD machine, the granularity will almost always be very fine. A large MIMD machine is able to handle a range of granularities, and a cluster of workstations is best suited to a parallel algorithm with a coarse granularity.

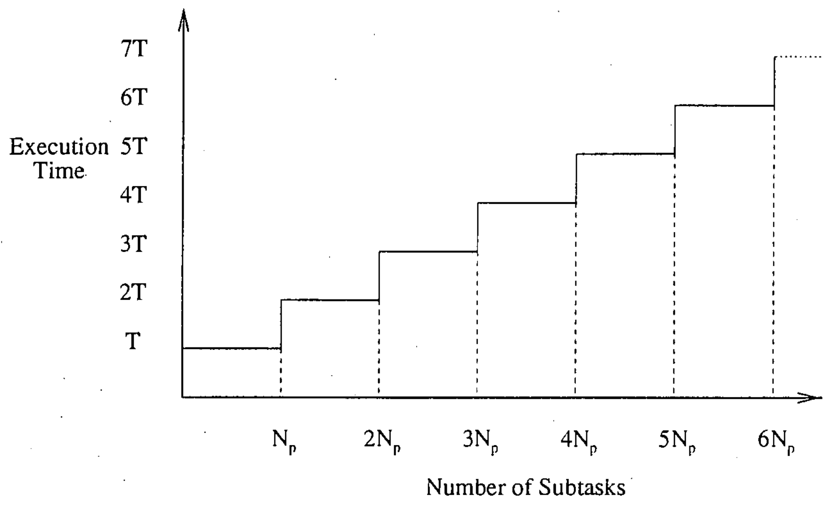

Figure 2.8. Graph showing how total execution time becomes quantised in the parallel regime. T is one quanta of execution time, and N_p is the number of processors.

In order for us to examine some aspects of load balancing, let us first consider a very coarse grained problem on two processors. If the amount of work required by each subtask is equivalent, and if there is an even number of subtasks, then both processors will be working until the whole problem is completed. However, as soon as an odd number of subtasks is required, one processor is going to be left idle while the other performs the last subtask. The worst case can be encapsulated in the following rule

$$N_t = N_p + 1 \tag{2.1}$$

where N_t is the number of subtasks, and $N_p > 1$ is the number of available processors. This rule can be also be applied to find the worst case in a massively parallel SIMD machine, where the number of processors could be several thousand. In this case, the problem will almost certainly be very fine grained, and $N_p - 1$ processors will be idle for half of the total execution time.

The total execution time, therefore, increases in a step function, as the number of subtasks increase. We can see in Figure 2.8, that so long as the number of subtasks lies in an interval between two integer multiples of the number of processors, the execution time will be the same. This is important, because it means the machine could be capable of carrying out significantly more work with no time penalty, resulting in a more efficient program.

When each subtask represents a different, possibly unknown, amount of work, the problem of load balancing becomes virtually intractable. These problems should only occur in a program written for a MIMD machine, as the programmer is always aware of how much work is being given to processors in a SIMD machine. The MIMD machine, with its ability to run different programs on different processors, can instruct the controlling processor to monitor the load balance of the allocated set of processors, and modify its approach to the problem appropriately.

2.4 Parallel Processing Algorithms

The process of solving an application problem on any computer can be broken down into roughly three stages: define the application, e.g. by mathematical formulæ; specify the algorithm (and write computer code); and finally, execution of the code on the computer.

Kung[34] identifies nine different models of computation on a research parallel computer.

1. local computation	4. multi-function pipeline	7. divide-and-conquer
2. domain partition	5. ring	8. query processing
3. pipeline	6. recursive computation	9. task queue

Each of his models corresponds to a different way in which data is passed between processors. I will describe only the first four; 1 and 2 being similar, but contrasting with the commonly implemented 3 and 4.

'Local computation' and 'domain partition' both involve decomposing the input data into a series of sub-domains, each of which is mapped to a single processor. Kung differentiates between the two, by noting that many algorithms depend only on a local datum, while others involve the communication of data between sub-domains. His choice of name for 2 is therefore not ideal, as it can describe 1 equally well. Figure 2.9 shows these two models.

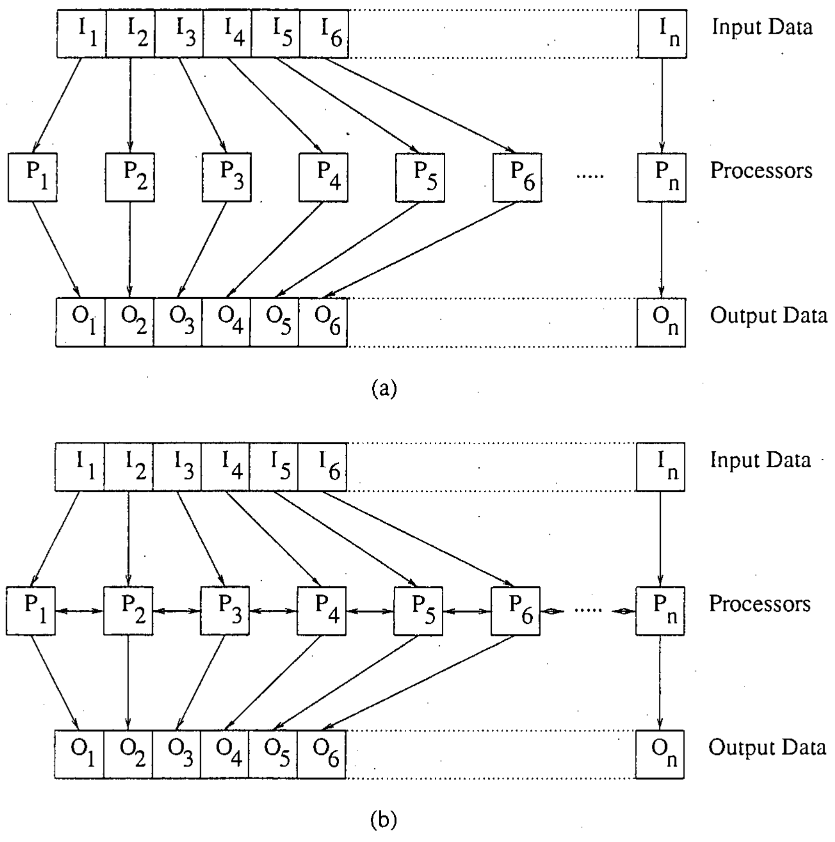

Figure 2.9. Domain partition processing with (a) local computation, (b) inter sub-domain communication

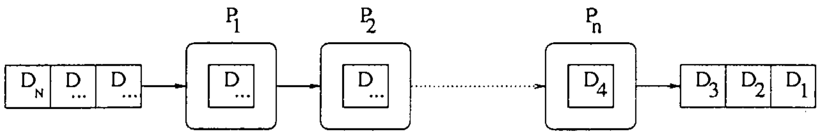

Figure 2.10. Pipeline processing: In multi-function pipelines, each process may be different operations

Later chapters will show how domain partition can be applied to both simultaneous and successive relaxations.

The 'pipeline' and, the more general, 'multi-function pipeline' models represent the process of passing all the data through a sequence of processes. As the first datum leaves the first processor, the second datum enters the beginning of the pipeline. This model is completely analogous to the UNIX shell pipe ('|').

2.5 Development Software

This section presents the two approaches referred to in the rest of the text, out of a myriad of possible parallel programming styles.

2.5.1 Data Parallel Languages

These languages, as the title suggests, treat the data as a parallel object. The most common data parallel language, and the one used in this work,.is derived from FORTRAN 90. The important development in this standard of FORTRAN over the FORTRAN 77 standard is the promotion of the multidimensional matrix to an intrinsic data type, along with `REAL`, `COMPLEX`, etc. Perhaps this difference can best be shown in the two code fragments in Table 2.2.

Each declares a one dimensional array of ten integers. The FORTRAN 77 code on the left must contain an explicit `DO` loop in order to set each element in the array. The FORTRAN 90 code, in contrast, can set every element in the array by simply referring to the array name without the subscript and parentheses. On a serial machine, the compiler will in effect generate an implicit `DO` loop to perform

FORTRAN 77	FORTRAN 90

`INTEGER N(10)`
`INTEGER I`
 	
`DO I = 1, 10`
`  N(I) = 1`
`END DO`
	`INTEGER N(10)`
	
	`N = 1`

Table 2.2. Initialising the elements of an array in FORTRAN 77 and FORTRAN 90

the same actions as the FORTRAN 77 code, but on a parallel machine, there is the possibility of each element being assigned to a different processor, and therefore, the complete array being set at once.

Until recently, each vendor of a parallel machine wrote their own parallel extensions to the FORTRAN 77 standard. It was in this environment of proprietary languages that the FORTRAN 90 standard slowly emerged. However, some parallel aspects are lacking in the standard, and yet another committee was given the task of defining a High Performance FORTRANstandard. I will simply call these modern languages FORTRAN 90 in the rest of this thesis.

There are also improvements to logical flow control statements, but I will ignore these, as they are largely irrelevant to this work. Instead, I will briefly describe some of the useful array constructs which map directly to a parallel, especially SIMD, architecture.

Element ↔ Processor Mapping

Since FORTRAN 90 is an inherently data parallel language, array elements must be mapped to processors in the machine. There are numerous ways of achieving this, but one thing they all have in common is that virtual processors are introduced to present a one to one mapping to the FORTRAN 90 programmer. The compiler arranges for each processor to timeshare between multiple array elements. This can have serious repercussions in performance on a SIMD machine. See Section 2.3 for a discussion of this load balancing problem.

Elemental Operations

If two arrays are conformant, that is with the same number and length of dimensions, then they may be combined by elemental operators. The following are examples of elemental operators.

Assignment `A = 1`

Simple Arithmetic `B = A+B, C = A*B`

Intrinsic Functions `B = SIN(A), C = LOG(A)`

Type Modifiers `R = REAL(C)`

As their name suggests, each of these operations act equally on all the elements in the arrays. It is important to remember that `A*B` is not matrix multiplication, but simply each element in `A` multiplied by the corresponding element in `B`.

Reduction Operators

These operators reduce the rank of an array, either by one, or all the way down to a scalar, and are new intrinsic functions, specific to arrays. If the function is called with only one argument (an array), then the result is a scalar calculated by the function. In addition, an integer constant representing a dimension may also be passed, which determines a direction over which the calculation is to take place, and the shape of the resulting array (with rank one less than the argument).

Examples of these reduction operators include

- `Total = SUM(A)` sets the scalar variable `Total` to be the sum of all the elements in `A`.
- `B = MAX(A, DIMENSION=1)` will result in `B` containing the maximum values along the first dimension of `A`. `B` has rank one less than `A`.

Regular Communications

Entire data arrays can be translated across the processor array very easily. Nearest neighbour communication, in particular, is highly efficient. High Performance FORTRANspecifies two new intrinsic functions to implement this kind if translation; SHIFT and CSHIFT. These functions require three arguments; the source array, a dimension index along which to translate, and a displacement. The CHSIFT form ensures that elements which "fall off" the array are wrapped back to the opposite edge. Figure 2.11 shows an array before and after a CSHIFT operation.

$$\begin{array}{ccccc}1&2&3&4&5\\6&7&8&9&10\\11&12&13&14&15\end{array} \quad \xrightarrow{\texttt{CSHIFT(A, SHIFT=1, DIM=1)}} \quad \begin{array}{ccccc}2&3&4&5&1\\7&8&9&10&6\\12&13&14&15&11\end{array}$$

Figure 2.11. The High Performance FORTRANCSHIFT intrinsic function.

Scope and Masks

The behaviour of these array intrinsics and arithmetic operators can be constrained by one of two methods. The array intrinsics allow an optional mask to be passed as an extra argument. A mask is simply a logical array, conformant with the source array, where a .TRUE. signifies that the operator is to be applied at that location, otherwise no action is performed. The second method is to modify the scope of a section of code inside a WHERE ...END WHERE construct. This is the direct analogue of the FORTRAN 77 IF ...END IF construct, except that it tests each element of an array to determine the logical flow. The classic example of this in use is to avoid division by zero. For example:

```
WHERE (A.NE.O)
  B = 1 / A
END WHERE
```

In a parallel machine, these masks and scope constraints can be thought of as specifying which processors should be turned on or off for the subsequent operations.

Array Sections

A sub-array can easily be extracted from a larger array by using the following syntax `B = A(n:m, ...)`. One `n:m` pair is required for each dimension in `A`. `m` indicates the start, and `n` the end of the section in that particular dimension. There are two special cases to be considered. First of all, if there is simply one constant, with no :, then that specifies the start and end of the section. If there is only a :, then the whole extent of the dimension is included in the section.

2.5.2 Message Passing Libraries

There are as many message passing libraries available as there are centres developing parallel code. However, a standard known as MPI[42] is emerging. These libraries are only applicable to MIMD machines, as SIMD machines cannot support processors running different code simultaneously.

In general, message passing is a much coarser grained form of parallelism than the FORTRAN 90 paradigm. However, there is a greater flexibility, allowing a broader range of algorithms to be implemented. The programmer has the choice of how to parallelise the code, whether to follow the data parallel path, or to implement a pipeline or any of the other computational models discussed in Section 2.4.

2.5.3 Task Farms

A task farm is built on top of a message passing library, and contains the following components, as shown in Figure 2.12:

Worker Processes which wait for sub-tasks to perform. This is where the bulk of the work is done in parallel, shared between them all.

Source Process which divides up the complete problem into manageable portions and passes them on to the workers.

Sink Process which listens to messages from the workers, and collects their results when they have completed their sub-tasks. It is the sink's job to recombine the results to form the complete solution of the problem.

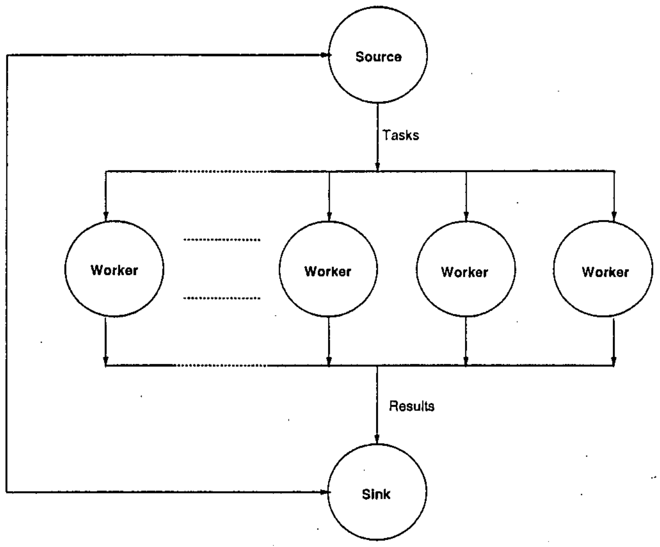

Figure 2.12. Classic Task Farm

The source process has overall control, and can influence the performance of the task farm by adapting its task partition policy depending on how the workers are coping.

2.6 Available Hardware

2.6.1 The DEC mpp/12000SX

Although now built by Digital Equipment Corporation, this type of computer was conceived and marketed by MasPar, a name which has been retained by DEC. I shall also refer to it as a MasPar in the rest of this work.

This machine was donated by DEC to the British Columbia Provincial Government Computer Centre in Victoria. In turn, they offered some very attractive computer-time packages to Canadian academic institutions, one of which was purchased by Dr Weaver to support this work.

The MasPar was one of the pioneering massively parallel SIMD machines, and is considered old technology by many in the numerical scientific community.

There was a very low demand for time on this machine, which resulted in a rapid development cycle, as I was usually the only user.

There are 8192 (2^{14}) simple processors arranged in a two dimensional lattice, similar to that shown in Figure 2.5(b), and each processor has 16kB of local memory. Therefore, the total memory available is 128MB, which is not significantly larger than a modern fast workstation class machine.

The fact that the processors are linked in a two dimensional, 64×128, network, facilitates mapping of two dimensional arrays, but can cause problems with higher dimensions. To overcome this, each processor partitions its own local memory, indexed by a third subscript on an array. Unfortunately, this has the effect of reducing the available storage space allocated to the original two dimensions.

The process of partitioning the local memory is also applied when virtual processors are required. If either of the first two dimensions exceed the number of physical processors in the lattice, the system partitions the local memories, and allocates each slice to a set of 8192 virtual processors. Computational time on the physical processors is now shared between the virtual processors.

This method of creating virtual processors has the unfortunate side effect of the likelihood of leaving a large percentage of processors idle for much of the time. For example, processing a 65×128 matrix would be using the machine at half efficiency. Section 2.3 introduced the idea of load balancing and efficiency.

The machine is hosted by a DEC microVAX workstation running the OSF/1 operating system. A MasPar FORTRAN compiler is used to build binaries from a very close variant of High Performance FORTRAN. When writing code, the author must be very careful to prevent what MasPar call "data sloshing". This occurs when an individual element of a parallel matrix is accessed. Instead of querying the responsible processor, the whole matrix is copied to the workstation memory, where the FORTRAN 77 type access is carried out. The next time the matrix is operated on as a whole, it is copied back to the processor lattice. This obviously poses a severe threat to overall performance, and must be avoided through careful algorithm design and implementation.

2.6.2 The Connection Machine 200

The Connection Machine is also a massively parallel SIMD machine, designed and constructed by Thinking Machines Corporation. The Edinburgh Parallel Computing Centre (EPCC) maintains a CM-200 to the United Kingdom academic community on behalf of the Engineering and Physical Sciences Research Council (EPSERC).

The basic unit of the CM-200 is a processing node constructed from two CM processor chips, two network chips, one optional 32 bit floating point processor and one megabyte of memory. Each processor chip contains sixteen one bit processors clocked at 8MHz. A sequencer interfaces the front end workstation to the rest of the machine, translating incoming code to nano-instructions for the simple processors.

There are several specialised buses linking the nodes. First of all, the instruction broadcast bus, which simply sends instructions from the sequencer to all the nodes. There is also a scalar memory bus which allows the sequencer to access any memory location for reading or writing, thus avoiding the problem of data sloshing from which the MasPar suffers. The global result bus returns a datum, combined from the single bit outputs of all the processors, to the sequencer. Finally there is the general interprocessor communication network. Each of the processor chips (16 processors) forms a vertex of a hypercube. The associated network chip supports three forms of communication across this topology.

Router Completely general processor to processor communications. Any processor can access any memory location throughout the network, with all processors making memory accesses simultaneously.

NEWS (North East West South) Nearest neighbour communication on an n-dimensional Cartesian grid.

Scans and Spreads Combines computation with communication on a NEWS grid. Especially efficient for finding sums, maximum values, *et cetera* in an array.

A fully configured CM-200 has 2048 processor nodes, or 64k processors, linked in a twelve dimensional hypercube, complete with two gigabytes of memory. The machine at Edinburgh is a quarter of that size, with 16k processors in a ten dimensional hypercube and 512 megabytes of memory.

The main programming language is Connection Machine FORTRAN, which is very close to High Performance FORTRAN. The front end host computer is a Sun 4 workstation, on which all software is developed before running on the back end parallel machine.

2.6.3 The Workstation Cluster

The Department of Geology & Geophysics at The University of Edinburgh possesses an impressive array of Unix workstations, from a variety of manufacturers. Subsets of these computers were available at different times to be configured as a task farm, as described in Section 2.5.3. They all share a common network filesystem (NFS) based file space running on top of TCP/IP, with generally unwritable local discs. The network is 10Mbit/s thin Ethernet which competes with a large Novell IPX LAN.

The particular task farm implementation was based upon the Parallel Utilities Library Task Farm (PUL-TF)[15][6] written at the EPCC. This piece of software is available for no charge to UK academics.

2.6.4 Other Serial Machines

I had access to a large Sun SparcCentre 2000 at the University of Edinburgh. This machine, with 256Mb of memory serves as a computer server for academics, which means that jobs are virtually always timesharing.

Digital Corporation, in an attempt to popularise their new line of Alpha processors, donated user accounts on a fully configured AXP class machine. In return for alerting them to bugs in the compilers and operating system, I was allowed network access and unlimited CPU time on this 190MHz Alpha21064 based machine, with 256Mb of core memory.

Chapter 3

Model Algorithms

3.1 Two Dimensional Model Formulation

The solution of a two dimensional problem splits into two distinct and independent modes, with either the electric or magnetic field polarised along the conductivity strike direction. Hobbs[24] suggests the use of the terms *B Polarisation* and *E Polarisation.* If we consider the strike to be along the x axis, then the conductivity structure does does not vary with x. The electromagnetic field can be expressed as $\mathbf{B} = (B_x, 0, 0)$ and $\mathbf{E} = (0, E_y, E_z)$ for B polarisation and similarly $\mathbf{B} = (0, B_y, B_z)$ and $\mathbf{E} = (E_x, 0, 0)$ for E polarisation. It is conventional to write the Cartesian field components as

$$E_x = U(y,z), E_y = V(y,z), E_z = W(y,z) \tag{3.1}$$

$$B_x = X(y,z), B_y = Y(y,z), B_z = Z(y,z) \tag{3.2}$$

Rewriting (1.3) and (1.9) in terms of these components gives six well known scalar equations

$$\frac{\partial W}{\partial y} - \frac{\partial V}{\partial Z} = -i\omega X \tag{3.3}$$

$$\frac{\partial X}{\partial z} = \mu_0 \sigma V \tag{3.4}$$

$$\frac{\partial X}{\partial y} = -\mu_0 \sigma W \tag{3.5}$$

and

$$\frac{\partial Z}{\partial y} - \frac{\partial Y}{\partial Z} = \mu_0 \sigma U \tag{3.6}$$

$$\frac{\partial U}{\partial z} = -i\omega Y \tag{3.7}$$

$$\frac{\partial U}{\partial y} = i\omega Z. \tag{3.8}$$

These equations have decoupled into two independent sets, with (3.3), (3.4) and (3.5) corresponding to the B polarisation and (3.6), (3.7) and (3.8) defining the E polarisation solution. Since Maxwell's equations allow the derivation of **E** from **B** and vice versa, the two dimensional solutions need only be for the scalar fields X and U respectively.

I will present only these equations necessary to the background for later sections here. A more detailed derivation can be found in Poll[48].

Figure 3.1 represents a general two dimensional conductivity model. The x axis is pointing into the page, and the structure is infinite along that axis. The structure is one dimensional at the extremities of y, in that it only varies with z, all the way to $\pm\infty$. z increases downwards into the Earth model, with $z = 0$ at the air boundary.

The model structure is defined as a series of regions of homogeneous and isotropic conductivity, σ_n, with discontinuities in conductivity at the boundaries. The atmosphere is treated as a perfect insulator, i.e. $\sigma = 0$ for $z < 0$. A half-space of constant conductivity, σ_0, underlies the whole model.

The vertical plane of the model is divided up into M by N smaller rectangles, forming a grid on which the model and solution are discretised. Each of these rectangular cells covers an area of constant conductivity, and will have a discontinuity at an edge, if and only if that edge coincides with the model structure. The vertices of the cells are labelled $X_{m,n}$, which are the same subscripts used to label the width, h_m, height, k_n, and conductivity, $\sigma_{m,n}$, of the cell.

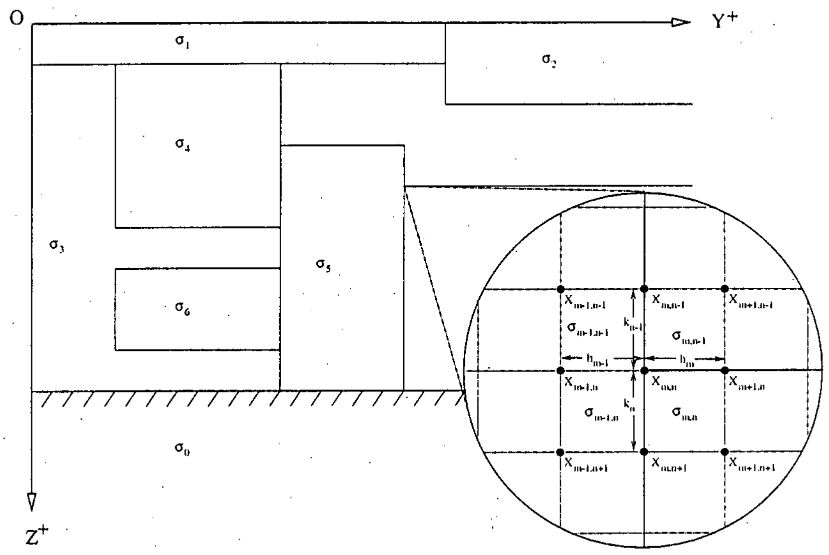

Figure 3.1. Two Dimensional Model Configuration. Insert shows the dimensions and values around a grid cell.

Any discrete, finite model must address the behaviour of the fields in the interior of the model and at the edges. The equations for the interior points are relatively straightforward to develop, but there are a great variety of approaches to the boundaries.

As was previously stated this finite difference formulation of the geoelectromagnetic induction problem was developed by Poll[48]. It draws upon many years experience in the different parts, bringing them together to form perhaps the most sophisticated finite difference model to date. The expressions are presented in terms of anomalous field differences, that is the difference from the one dimensional solution at $y = -\infty$.

I shall briefly summarise the historical development of this model's constituent parts:

interior points This is essentially that which was presented by Brewitt-Taylor and Weaver[4] for their finite difference model[5], but written in terms of the anomalous and host fields.

side boundaries The one dimensional solutions were originally derived by Wait[66] in 1953 for the propagation of radio waves, then developed by Schmucker[56] and later by Summers and Weaver[60]. The asymptotic relationship linking $U(\pm\infty)$ to the model boundary in E polarisation were developed by Weaver and Brewitt-Taylor[74].

surface boundaries The integral equations for the top surface E polarisation model are derived from the *thin sheet* approximation developed by Price[49], and generalised by, amongst others, Ranganayaki and Madden[53] and Weaver[73].

3.1.1 Interior Grid Points

The finite difference equations for the anomolous field[1] at the centre of the interior grid cells, each of height k_n, width h_m and resistivity $\rho_{m,n}$, can be written in the following general form

$$C_1\tilde{F}_{m,n-1} + C_2\tilde{F}_{m+1,n} + C_3\tilde{F}_{m,n+1} + C_4\tilde{F}_{m-1,n} = (C_5 + i)\tilde{F}_{m,n} + K \tag{3.9}$$

where $C_5 = C_1 + C_2 + C_3 + C_4$ and, for B polarisation

$$\begin{aligned}
\tilde{F} &= \tilde{X} && (3.10)\\
C_0 &= \frac{1}{2}\omega\mu_0 h_m^+ k_n^+ && (3.11)\\
C_1 &= \frac{h_{m-1}\rho_{m-1,n} + h_m\rho_{m,n-1}}{k_{n-1}C_0} && (3.12)\\
C_2 &= \frac{k_{n-1}\rho_{m,n-1} + k_n\rho_{m,n}}{h_m C_0} && (3.13)\\
C_3 &= \frac{h_{m-1}\rho_{m-1,n} + h_m\rho_{m,n}}{k_n C_0} && (3.14)\\
C_4 &= \frac{k_{n-1}\rho_{m-1,n-1} + k_n\rho_{m-1,n}}{h_{m-1}C_0} && (3.15)\\
K &= -C_1X_{n-1}^- - (C_2 + C_4 - C_5 - i)X_n^- - C_3X_{n+1}^- && (3.16)
\end{aligned}$$

[1]The anomolous field, $\tilde{F}$, is defined as the difference between the local field and the one-dimensional field at the left side of the grid, F_n^-.

and for E polarisation

$$\tilde{F} = \tilde{U} \tag{3.17}$$

$$C_0 = \omega\mu_0\overline{\sigma}_{m,n}h_m^+k_n^+ \tag{3.18}$$

$$C_1 = \frac{h_m^+}{k_{n-1}C_0} \tag{3.19}$$

$$C_2 = \frac{k_n^+}{h_m C_0} \tag{3.20}$$

$$C_3 = \frac{h_m^+}{k_n C_0} \tag{3.21}$$

$$C_4 = \frac{k_n^+}{h_{m-1}C_0} \tag{3.22}$$

$$K = i\left(1 - \frac{\overline{\sigma}_n^-}{\overline{\sigma}_{m,n}}\right)U_n^- \tag{3.23}$$

with $h_m^+ = h_{m-1} + h_m$ and $k_n^+ = k_{n-1} + k_n$ for simplicity.

The E polarisation is not expressed in terms of σ but the weighted average conductivities, as introduced by Brewitt-Taylor and Weaver[4].

$$\overline{\sigma}_{m,n} = \frac{h_m k_{n-1}\sigma_{m,n-1} + h_m k_n \sigma_{m,n}}{h_m^+ k_n^+} + \frac{h_{m-1}k_n\sigma_{m-1,n} + h_{m-1}k_{n-1}\sigma_{m-1.n-1}}{h_m^+k_n^+} \tag{3.24}$$

$$\overline{\sigma}_n^- = \frac{k_{n-1}}{k_n^+}\sigma_{n-1}^- + \frac{k_n}{k_n^+}\sigma_n^- \tag{3.25}$$

3.1.2 Boundary Conditions

The sets of equations developed for the interior points can not be assumed to be valid around the edges of the model grid. In particular, there are no nodes outside the declared grid, so that some of the terms in the equations will be undefined.

A *boundary condition* is a rule which governs, in some manner, the behaviour of the solution at that boundary of the model. The most common boundary conditions are classified as one of two families

Dirichlet The values of the solution at the boundary are prescribed, and remain fixed throughout the solution process. For example, $u_{i,1} = 0$.

Neumann The values of the solution at the boundary are constrained by prescribed normal derivatives. For example, $\frac{\partial \boldsymbol{F}}{\partial z}|_{z=0} = 0$.

Left and Right Boundary Conditions

As has been mentioned before, the conductivity structure degenerates to be one dimensional at the left and right edges of the model. The field at an infinite distance along the y axis will also be one dimensional, as the disturbance caused by variations in conductivity with y will have completely diffused to zero. This will be some considerable distance in some situations, especially ocean coasts[11][53].

The one dimensional solutions at $y = \pm\infty$ are denoted as F^+ and F^-, where $F = X$ for B polarisation, and $F = U$ for E polarisation. These solutions themselves can be used as Dirichlet boundary conditions for the B polarisation case, so long as the edges of the grid are suitably far away from the anomalous structures.

The E polarisation fields, however, do not recover to their one dimensional form so quickly, and the asymptotic boundary conditions derived by Weaver and Brewitt-Taylor[74] are employed. Poll writes the anomolous field boundary conditions at the top surface corners as

$$\Gamma_1 \tilde{U}_1 - \tilde{U}_2 = 0 \tag{3.26}$$

$$\Gamma_{N_y} \tilde{U}_{N_y} - \tilde{U}_{N_y-1} = (\Gamma_{N_y} - 1)(U^+ - U^-) \tag{3.27}$$

where $\Gamma_1 = \frac{1}{1-\epsilon_1} - 2\epsilon_1$, $\Gamma_{N_y} = 2\epsilon_{N_y} + \frac{1}{1+\epsilon_{N_y}}$ and $\epsilon_i = \frac{h_{i-1}}{y_i}$.

The side boundary conditions for $z > 0$, written in terms of the anomalous field, are

$$\tilde{X}_1(z) = 0 \tag{3.28}$$

$$\tilde{X}_{N_y}(z) = X^+(z) - X^-(z) \tag{3.29}$$

$$U^-(z_{n-1})\tilde{U}_{1,n} - U^-(z_n)\tilde{U}_{1,n-1} = 0 \tag{3.30}$$

$$\begin{aligned} U^+(z_{n-1})\tilde{U}_{N_y,n} - U^+(z_n)\tilde{U}_{N_y,n-1} &= \\ U^+(z_n)U^-(z_{n-1}) - U^+(z_{n-1})U^-(z_n) \end{aligned} \tag{3.31}$$

Top Boundary Conditions

For B polarisation it is obvious from (3.4) and (3.5) that above the surface of the Earth, which is non-conducting atmosphere, $\partial X/\partial y = 0$ and $\partial X/\partial z = 0$ so that

$$X(y, 0-) = X_0 \tag{3.32}$$

where X_0 is some constant. For a uniform horizontal source magnetic field of $\mathbf{B}_0$ the total field above a one dimensional Earth is $2\mathbf{B}_0$. Therefore we can write $X(y, 0-) = 2B_0$ since the model becomes one dimensional as $|y| \to \infty$.

In the E polarisation case, setting $\sigma = 0$ leads to

$$\frac{\partial Y}{\partial z} = \frac{\partial Z}{\partial y} \quad \text{and} \quad \frac{\partial Y}{\partial y} + \frac{\partial Z}{\partial z} = 0. \tag{3.33}$$

The solution of this differential equation, results in the same integral boundary condition as that obtained by Schmucker[56], which can be written in terms of the discrete anomalous electric field as

$$\begin{aligned} L_i\tilde{U}_{1,N_z} + M_i\tilde{U}_{N_y,1} + N_i\tilde{U}_{i-1,1} + P_i\tilde{U}_{i,1} + Q_i\tilde{U}_{i+1,1} + \left(\sum_{m=2}^{i-2} + \sum_{m=i+2}^{N_y-1}\right) R_i^m\tilde{U}_{m,1} \\ + \frac{\pi}{k_1}\tilde{U}_{i,2} = S_iU_1^- + T_iU_1^+. \end{aligned} \tag{3.34}$$

where L, M, N, P, Q, R, S and T are derived coefficients, as in Poll[48].

Bottom Boundary Conditions

It is possible to allow the model mesh to extend to great depths in the model as both X and U approach zero as $z \to \infty$. However, it is computationally expensive to dedicate these grid points for this purpose, so Poll has implemented an integral boundary condition at a depth $z = d$, below which lies a half space of constant conductivity, σ_0. These were originally developed by Green and Weaver[23] and can be written in terms of the discrete anomalous fields as

$$N_i\tilde{X}_{i-1,N_z} + P_i\tilde{X}_{i,N_z} + Q_i\tilde{X}_{i+1,N_z} + \left(\sum_{m=2}^{i-2} + \sum_{m=i+2}^{N_y-1}\right) R_i^m \tilde{X}_{m,N_z}$$
$$+ \frac{\pi\sigma_0 r_{i,N_z-1}}{k_{N_z-1}}\tilde{X}_{i,N_z-1} = S_i X_{N_z}^- + T_i X_{N_z}^+ + \sigma_0\pi(\sigma_0 r_{i,N_z-1}\sigma_{N_z-1}^- - \sigma_{N_z}^-)V_{N_z}^- \quad (3.35)$$

for B polarisation, and

$$L_i\tilde{U}_{i,N_z} + M_i\tilde{U}_{N_y,N_z} + N_i\tilde{U}_{i-1,N_z} + P_i\tilde{U}_{i,N_z} + Q_i\tilde{U}_{i+1,N_z}$$
$$+ \left(\sum_{m=2}^{i-2} + \sum_{m=i+2}^{N_y-1}\right) R_i^m \tilde{U}_{m,N_z} + \frac{\pi}{k_{N_z-1}}\tilde{U}_{i,N_z-1} = S_i U_{N_z}^- + T_i U_{N_z}^+ \quad (3.36)$$

for E polarisation.

3.2 Three Dimensional Model Formulation

The following description of the mathematical formulation of the three dimensional model is all based upon work developed by Xinghua Pu[51] at the University of Victoria, British Columbia, Canada. I will only present what is necessary to show the links between the mathematical and the parallel processing algorithms.

The model solves the magnetic field components for a general three dimensional structure, overlaid by an optional thin sheet of variable conductance, with general two dimensional structures at the vertical boundaries. Figure 3.2 shows such a model.

3.2.1 Vertical Edge Boundary Conditions

There are numerous ways to enforce boundary conditions around the vertical edges of the three dimensional model volume.

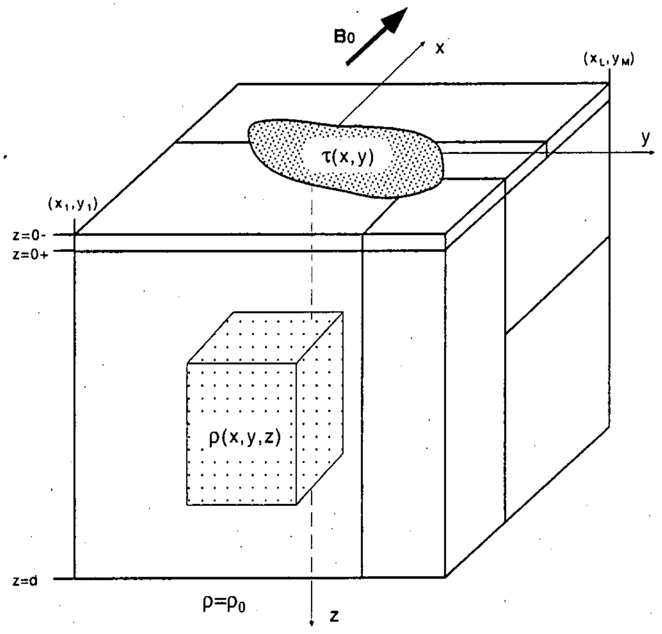

Figure 3.2. General 3D model configuration (After Pu, 1994)

If one assumes that the structure is periodic in the horizontal directions, then periodic boundary conditions can be applied, as in the case of the Fourier methods of Park[46] and Jiracek[27]. However, in order to eliminate the effects of repeating the structure, the boundaries are extended to great distances from the three dimensional structure, resulting in expensive grid point usage.

The other methods rely on previously calculated field values for the boundaries, which are then fixed. This class of boundary conditions is known as Dirichlet boundary conditions and there are, of course, a variety of methods of finding these values. Mackie et al.[37] divide their three dimensional model into a series of vertical slices, which are each embedded in a two dimensional B Polarisation grid, reflecting the regional two dimensional structure. Unfortunately, this approach has two obvious drawbacks. First of all, a large number of 2D problems need to be solved (one for each slice), and secondly, not all regional model configurations can be described by a single electrical strike.

To allow as general as possible regional structure, Pu imposes the condition that the model structure degenerates to two dimensions at the vertical model boundaries, each with a strike normal to the boundary. Thus, the regional structure can be constructed from two perpendicular strikes, as at some tectonic margins, for example.

As has been discussed in Section 3.1, two dimensional solutions split into two distinct and independent modes, "E Polarisation" and "B Polarisation". In order to describe the four vertical boundaries, two solutions of each polarisation are required. The solutions on opposing boundaries will be polarised in the same direction. Pu developed an E Polarisation solution for B_y and B_z in order to overcome the fact that the solution for E_x is dependent on a volume weighted average of electrical conductivity, whereas any solution for $\mathbf{B}$ depends on a weighted average of resistivity. Simply taking the reciprocal of either of these is not equivalent to the other, as noted by Brewitt-Taylor[4]. The reader is referred to Pu[51] for a full description of the two dimensional solutions.

In summary, the components of the magnetic field are fixed at their two dimensional approximations over the vertical boundary surfaces of the model.

3.2.2 Internal Grid Points

The three Cartesian components of the magnetic field vector need to be solved for all the internal grid points. These values are potentially dependent on all the surrounding points, as is shown in Figure 3.3. This figure also illustrates the point numbering scheme used in this section, as well for describing interprocessor communications in Chapter 4.

The internal governing differential equation is obtained by applying volume integration over the cuboid, surface S and volume V, surrounding a grid point, and utilising a vector relation

$$\int_V (\nabla \times \mathbf{A})\, dv = \oint_S d\mathbf{S} \times \mathbf{A} \tag{3.37}$$

where V is a volume bounded by a closed surface S, with $d\mathbf{S}$ positive outward from the enclosed volume. This relation is derived from the well-known Gauss divergence theorem

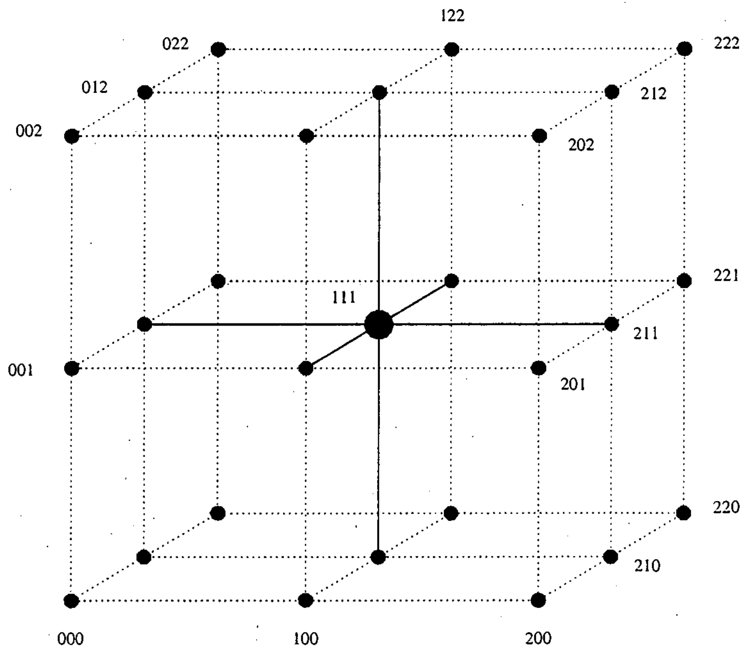

Figure 3.3. 3D internal points, showing numbering scheme

$$\int_V (\nabla \cdot \mathbf{A}) dv = \oint_S \mathbf{A} \cdot d\mathbf{S}. \tag{3.38}$$

A volume integration of the Maxwell equation (1.3) gives

$$\int_V \nabla \times \mathbf{E}\, dv = -i\omega \int_V \mathbf{B}\, dv \tag{3.39}$$

which can be further transformed by (3.37) and Maxwell equation (1.4) into

$$\int_V \nabla \times \mathbf{E}\, dv = \oint d\mathbf{S} \times \mathbf{E} = -\frac{1}{\mu_0} \oint \rho(\nabla \times \mathbf{B}) \times d\mathbf{S}. \tag{3.40}$$

Equation (3.39) then becomes

$$\oint_S \rho(\nabla \times \mathbf{B}) \times d\mathbf{S} = i\omega\mu \int_V \mathbf{B}\, dv. \tag{3.41}$$

Pu presents the finite difference equivalent of (3.41) to be

$$\begin{aligned}(i + c_{101} + c_{121} + c_{110} + c_{112})X_{111} &= c_{101}X_{101} + c_{121}X_{121} + c_{110}X_{110} + c_{112}X_{112}\\
&(c_{221} - c_{201} + c_{001} - c_{021})Y_{111} + c_{201}Y_{201} - c_{221}Y_{221} - c_{001}Y_{001} + c_{021}Y_{021}\\
&+(c_{021} - c_{001})Y_{011} - (c_{221} - c_{201})Y_{211} - (c_{201} - c_{001})Y_{101} + (c_{221} - c_{021})Y_{121}\\
&(c_{212} - c_{210} + c_{010} - c_{012})Z_{111} + c_{012}Z_{012} - c_{212}Z_{212} - c_{010}Z_{010} + c_{210}Z_{210}\\
&+(c_{012} - c_{010})Z_{011} - (c_{212} - c_{210})Z_{211} - (c_{210} - c_{010})Z_{110} + (c_{212} - c_{012})Z_{112}.\end{aligned} \tag{3.42}$$

$$\begin{aligned}(i + c_{011} + c_{211} + c_{110} + c_{112})Y_{111} &= c_{011}Y_{011} + c_{211}Y_{211} + c_{110}Y_{110} + c_{112}Y_{112}\\
&(c_{221} - c_{201} + c_{001} - c_{021})X_{111} + c_{201}X_{201} - c_{221}X_{221} - c_{001}X_{001} + c_{021}X_{021}\\
&+(c_{201} - c_{001})X_{101} - (c_{221} - c_{021})X_{121} - (c_{021} - c_{001})X_{011} + (c_{221} - c_{201})X_{211}\\
&(c_{122} - c_{102} + c_{100} - c_{120})Z_{111} + c_{120}Z_{120} - c_{122}Z_{122} - c_{100}Z_{100} + c_{102}Z_{102}\\
&+(c_{102} - c_{100})Z_{101} - (c_{122} - c_{120})Z_{121} - (c_{120} - c_{100})Z_{110} + (c_{122} - c_{102})Z_{112}\end{aligned} \tag{3.43}$$

$$\begin{aligned}(i + c_{011} + c_{211} + c_{101} + c_{121})Z_{111} &= c_{011}Z_{011} + c_{211}Z_{211} + c_{101}Z_{101} + c_{121}Z_{121}\\
&(c_{212} - c_{210} + c_{010} - c_{012})X_{111} + c_{012}X_{012} - c_{212}X_{212} - c_{010}X_{010} + c_{210}X_{210}\\
&+(c_{210} - c_{010})X_{110} - (c_{212} - c_{012})X_{112} - (c_{012} - c_{010})X_{011} + (c_{212} - c_{210})X_{211}\\
&(c_{122} - c_{120} + c_{100} - c_{102})Y_{111} + c_{102}Y_{102} - c_{122}Y_{122} - c_{100}Y_{100} + c_{120}Y_{120}\\
&+(c_{120} - c_{100})Y_{110} - (c_{122} - c_{102})Y_{112} - (c_{102} - c_{100})Y_{101} + (c_{122} - c_{120})Y_{121}\end{aligned} \tag{3.44}$$

3.2.3 Top and Bottom Boundary Conditions

The thin sheet (as introduced in Section 1.3) at the upper surface of the model allows shallow features to be incorporated without dedicating valuable grid nodes to their representation. The approximation holds so long as it is much thinner than the skin depth inside it, and that high conductivity layers are set deep in the model. The Z component remains unchanged across the thin sheet ($Z- = Z+$), and is used to calculate the horizontal components thus

$$X(\mathbf{r},0-) = B_0 - M_1 Z(\mathbf{r},0-), \quad Y(\mathbf{r},0-) = -M_2 Z(\mathbf{r},0-). \tag{3.45}$$

where B_0 is the source field and

$$M_2 Z = \frac{1}{2\pi}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} Z(u,v)\frac{y-v}{[(x-u)^2+(y-v)^2]}\,du\,dy. \tag{3.46}$$

The boundary condition above the thin sheet ($z = 0-$) can be written in terms of discrete finite differences as

$$\begin{aligned} X_{\lambda\mu 0} &= B_0 + \sum_{l=1}^{L}\sum_{m=1}^{M} A^{lm}_{\lambda\mu} Z_{\lambda\mu 0}, && (3.47)\\ Y_{\lambda\mu 0} &= \sum_{l=1}^{L}\sum_{m=1}^{M} B^{lm}_{\lambda\mu} Z_{\lambda\mu 0}, && (3.48)\\ Z_{\lambda\mu 0} &= Z_{\lambda\mu 1}, && (3.49)\\ &\lambda = 2\ldots L-1,\quad \mu = 2\ldots M-1,\quad \nu = 0 \end{aligned}$$

where $A^{lm}_{\lambda\mu}$ and $B^{lm}_{\lambda\mu}$ are given in Appendix A of Pu's thesis[51].

The boundary conditions under the thin sheet ($z = 0+$) do not involve a complete surface integral, yet are more complex to write in full than for the interior points.

The model grid is underlain by a half-space of constant conductivity ($\sigma = \sigma_o$ for $z > d$). The integral boundary condition as developed in Weaver's book[73] is applied over this contact surface. Once again, Pu[51] has a full derivation of the lengthy finite difference expressions for these boundary conditions.

Chapter 4

Parallel Implementations

This chapter describes the two and three dimensional modelling parallel algorithms. The iterative solvers are developed for the two dimensional case, but apply equally to the three dimensional problem.

4.1 Matrix Inversion

The forward modelling problem has been reduced to solving systems of equations

$$\boldsymbol{A}\boldsymbol{u} = \boldsymbol{b} \tag{4.1}$$

where $\boldsymbol{A}$ is the coefficient matrix[1], $\boldsymbol{u}$ is a column vector of field values, which are to be found, and $\boldsymbol{b}$ is a representation of the sources contained within the model.

4.1.1 Form of the coefficient matrix

The coefficient matrix is composed of $N_z \times N_z$ sub-matrices, each with $N_y \times N_y$ elements. The overall matrix is sparse, as three non-zero elements are located along the leading diagonal, and two further elements each N_y positions to either side of the diagonal, reflecting the essentially local nature of the equations. Figure 4.1.1 illustrates the form of a typical coefficient matrix.

[1]Often referred to as the *stiffness matrix*, from the numerical modelling of mechanical structures.

$$
\begin{array}{c}
\overbrace{\overbrace{\begin{array}{ccccc}&&&&\end{array}}^{N_y}\;\overbrace{\begin{array}{ccccc}&&&&\end{array}}^{N_y}}^{N_z}
\end{array}
$$

$$
N_z\left\{
\begin{array}{l}
N_y\left\{\begin{array}{ccccccccccc}
\bullet & \bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \cdots \\
\bullet & \bullet & \bullet & \bullet & \bullet & \circ & \bullet & \circ & \circ & \circ & \cdots \\
\bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \bullet & \circ & \circ & \cdots \\
\bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ & \bullet & \circ & \cdots \\
\bullet & \bullet & \bullet & \bullet & \bullet & \circ & \circ & \circ & \circ & \bullet & \cdots
\end{array}\right. \\
N_y\left\{\begin{array}{ccccccccccc}
\bullet & \circ & \circ & \circ & \circ & \bullet & \bullet & \circ & \circ & \circ & \cdots \\
\circ & \bullet & \circ & \circ & \circ & \bullet & \bullet & \bullet & \circ & \circ & \cdots \\
\circ & \circ & \bullet & \circ & \circ & \circ & \bullet & \bullet & \bullet & \circ & \cdots \\
\circ & \circ & \circ & \bullet & \circ & \circ & \circ & \bullet & \bullet & \bullet & \cdots \\
\circ & \circ & \circ & \circ & \bullet & \circ & \circ & \circ & \bullet & \bullet & \cdots
\end{array}\right.
\end{array}\right.
$$

$$
\begin{array}{ccccccccccc}
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \ddots
\end{array}
$$

Figure 4.1. Form of the 2D coefficient matrix. [○] marks zero elements, and [●] non-zero elements. This matrix is from an E Polarisation model, with $N_y \times N_z$ grid points ($N_y = 5$ in this case).

The integral boundary conditions at the Earth's surface, ($z = 0$), and at the top of the half space, ($z = d$), fully populate the upper and lower diagonal sub-matrices. Unfortunately, this destroys any possibility of utilising a straightforward sparse matrix compaction scheme, as large numbers of unrequired elements from the inner diagonal sub-matrices would also be stored.

4.1.2 Gauss-Jordan Direct Method

Poll's original algorithm solved the system of equations (4.1) by the Gauss-Jordan direct method, commonly found in any linear algebra textbook. The system is reduced to a triangular system and then back substitution leads to the final solution. This method has the advantage of providing a solution in a fixed number of steps ($\frac{1}{3}n^3 + O(n^2)$ multiplications).

However, like all direct methods, it can suffer from a lack of accuracy for ill conditioned systems, and just as importantly in this study, it requires the complete matrix to be constructed and stored. Poll's code writes the matrix to the filesystem, and reads from it as necessary when carrying out the back substitution stage.

4.1.3 Iterative Solvers

Iterative solvers start with an initial guess as to the solution, $\boldsymbol{u}^{(0)}$ and repeatedly *relax* it, until some criterion is satisfied. The superscript figure in parentheses denotes the *iteration level*, zero being the set of values before the relaxation commences. Each step can be represented by

$$\boldsymbol{u}^{(n+1)} = \boldsymbol{G}^{(n)}\boldsymbol{u}^{(n)} + \boldsymbol{k} \tag{4.2}$$

where $\boldsymbol{G}^{(n)}$ represents the relaxation process and $\boldsymbol{k}$ is the residual.

An iterative method is defined to be *stationary* if $\boldsymbol{G} = \boldsymbol{G}^{(n)}$ for all n, i.e. the relaxation step is the same at all iterative levels. Only stationary iterative methods are considered in this thesis.

The major advantage of iterative methods is that the coefficient matrix $\boldsymbol{A}$ does not need to be expressed explicitly. Instead, all the information contained in the finite difference equations and boundary conditions is expressed as a set of rules governing the relaxation towards the final solution.

Convergence

Ideally any iterative technique should converge to $\hat{\boldsymbol{u}}$, the solution of (4.1), for any starting vector, $\boldsymbol{u}^{(0)}$. The method is defined to be *weakly convergent* if the series $\boldsymbol{u}^{(0)}, \ldots \boldsymbol{u}^{(n)}$ converges for any $\boldsymbol{u}^{(0)}$, and is *strongly convergent* if it converges to the same limit, independent of $\boldsymbol{u}^{(0)}$. It is therefore necessary to show that the chosen method is strongly convergent as we wish the method to be stable, for all models.

Consistency

An iterative method, once converged to a solution, should not then deviate from that solution with further iterations. If it does remain at the solution, then the method it consistent.

Put formally, as by Young[81], the method (4.2) is consistent with the system (4.1) if and only if, for some n, $\boldsymbol{u}^{(n)}$ is a solution, say $\hat{\boldsymbol{u}}$ of (4.1), then $\boldsymbol{u}^{(n+1)} = \boldsymbol{u}^{(n+2)} = \cdots = \hat{\boldsymbol{u}}$.

Therefore, the *related system*

$$(\boldsymbol{I} - \boldsymbol{G})\hat{\boldsymbol{u}} = \boldsymbol{k} \tag{4.3}$$

can be derived from the iterative method (4.2) once convergence has been reached.

Conditions for Convergence

Smith[58] presents the general condition for convergence, the development of which follows.

Defining the error $\boldsymbol{e}^{(n)}$ in the n^{th} approximation to the exact solution as $\boldsymbol{e}^{(n)} = \hat{\boldsymbol{u}} - \boldsymbol{u}^{(n)}$ and substituting into (4.2) yields

$$\boldsymbol{e}^{(n+1)} = (\boldsymbol{I} - \boldsymbol{G})\hat{\boldsymbol{u}} + \boldsymbol{G}\boldsymbol{e}^{(n)} - \boldsymbol{k} \tag{4.4}$$

and using the related system (4.3)

$$\boldsymbol{e}^{(n+1)} = \boldsymbol{G}\boldsymbol{e}^{(n)} \tag{4.5}$$

and therefore

$$\boldsymbol{e}^{(n)} = \boldsymbol{G}^{n}\boldsymbol{e}^{(0)} \tag{4.6}$$

The series of iterates $\boldsymbol{u}^{(1)}, \boldsymbol{u}^{(2)}, \ldots, \boldsymbol{u}^{(n)}, \ldots$ will converge to to $\hat{\boldsymbol{u}}$ as n tends to infinity if

$$\lim_{n\to\infty} \boldsymbol{e}^{(n)} = \boldsymbol{0} \tag{4.7}$$

However, since the desired behaviour is for convergence for arbitrary $\boldsymbol{u}^{(0)}$ and $\boldsymbol{e}^{(0)}$ the condition for convergence is

$$\lim_{n\to\infty} \boldsymbol{G}^n = 0 \tag{4.8}$$

Assuming that the matrix $\boldsymbol{G}$ of order m has m linearly independent eigenvectors $\boldsymbol{g}_i, i = 1, m$, then these eigenvectors can be used as a basis of the m-dimensional vector space. The error vector can therefore be expressed as a unique linear combination of these eigenvalues, thus

$$\boldsymbol{e}^{(0)} = \sum_{i=1}^{m} a_i \boldsymbol{g}_i \tag{4.9}$$

and therefore

$$\boldsymbol{e}^{(1)} = \boldsymbol{G}\boldsymbol{e}^{(0)} = \sum_{i=1}^{m} a_i \boldsymbol{G}\boldsymbol{g}_i \tag{4.10}$$

The definition of eigenvectors state that $\boldsymbol{G}\boldsymbol{g}_i = \lambda_i \boldsymbol{g}_i$ if λ_i is the corresponding eigenvalue, so

$$\boldsymbol{e}^{(1)} = \sum_{i=1}^{m} a_i \lambda_i \boldsymbol{g}_i \tag{4.11}$$

and

$$\boldsymbol{e}^{(n)} = \sum_{i=1}^{m} a_i \lambda_i^n \boldsymbol{g}_i \tag{4.12}$$

Therefore $\boldsymbol{e}^{(n)}$ will tend to $\boldsymbol{0}$ as n tends to infinity, for any $\boldsymbol{e}^{(0)}$, if and only if $|\lambda_i| < 1$ for all i. This is equivalent to requiring the spectral radius $\rho(\boldsymbol{G})$ of $\boldsymbol{G}$ be less than one.

It is sufficient to require that $||\boldsymbol{G}|| < 1$ since $\rho(\boldsymbol{G}) \leq ||\boldsymbol{G}||$ (Young[81], Theorem 3.4, p.32).

Stopping Criteria

The number of iterations required to allow convergence to a solution is potentially infinite, so some kind of criterion as to when to stop must be checked against regularly.

The chosen criterion must be

1. an accurate assessment of the state of convergence of the problem, and

2. inexpensive to compute, or it will dominate the whole process.

These two requirements work against each other, in that the more accurate criteria are more expensive to compute. However, by only applying the criterion once every ten iterations or so, the second can be satisfied.

The essence of the criterion is that it should measure how much the solution is changing between iteration levels. As in all measurements of change, either the absolute or relative changes can be considered, and because the solution is discrete across the grid, these differences can be calculated on a local or global basis.

The local absolute maximum change can be calculated by

$$\Delta^{(p)}(\boldsymbol{u}) = \max_{i,j} \left| u_{i,j}^{(p)} - u_{i,j}^{(p-1)} \right| \tag{4.13}$$

which is also known as the *l-infinity* (l_∞) norm of the change vector.

Similarly, a local relative maximum change can be calculated by

$$\Delta^{(p)}(\boldsymbol{u}) = \max_{i,j} \left| \frac{u_{i,j}^{(p)} - u_{i,j}^{(p-1)}}{u_{i,j}^{(p)}} \right| \tag{4.14}$$

The corresponding global maxima are found by summing the terms in (4.13) and (4.14). For example, the global relative change can be expressed as

$$\Delta^{(p)}(\boldsymbol{u}) = \frac{\sum_{i,j} |u_{i,j}^{(p)} - u_{i,j}^{(p-1)}|}{\sum_{i,j} |u_{i,j}^{(p)}|} \tag{4.15}$$

4.1.4 Simultaneous Relaxation

Traditional Point Jacobian Relaxation

Simultaneous relaxation indicates that all the elements u_i of the solution column matrix $\boldsymbol{u}$ are updated simultaneously ; i.e. the $(p+1)^{\text{th}}$ iteration level depends only upon the p^{th} level $(\boldsymbol{u}^{(p+1)} \leftarrow \boldsymbol{u}^{(p)})$.

For non-boundary nodes this can be written

$$u_{i,j}^{(p+1)} \leftarrow u_{i-1,j}^{(p)}, u_{i+1,j}^{(p)}, u_{i,j-1}^{(p)}, u_{i,j+1}^{(p)}, u_{i,j}^{(p)} \tag{4.16}$$

This simple dependency relation has two striking properties:

1. It is straightforward to parallelise on a SIMD computer, as every node can be updated independent of all the others.

2. It has been long acknowledged to be mathematically slow to converge.

Before continuing with the parallel implementation of this method, I will briefly discuss its behaviour for this particular modelling problem.

For convergence to be guaranteed, it is necessary that $||\boldsymbol{G}|| < 1$. In the point Jacobian, $\boldsymbol{G}$ takes the form

$$\boldsymbol{G} = \boldsymbol{D}^{-1}(\boldsymbol{L} + \boldsymbol{U}) \tag{4.17}$$

where $\boldsymbol{D}$, $\boldsymbol{L}$ and $\boldsymbol{U}$ are, respectively, the diagonal, strictly lower and strictly upper elements of $\boldsymbol{A}$.

Taking the i^{th} equation of $\boldsymbol{Au} = \boldsymbol{b}$ to be

$$a_{i1}x_1 + a_{i2}x_2 + \cdots + a_{ii}x_i + \cdots + a_{im}x_m = b_i$$

then the i^{th} row of $\boldsymbol{G}$ is

$$\frac{a_{i1}}{a_{ii}} \quad \frac{a_{i2}}{a_{ii}} \quad \cdots \quad \frac{a_{i,i-1}}{a_{ii}} \quad 0 \quad \frac{a_{ii+1}}{a_{ii}} \quad \cdots \quad \frac{a_{im}}{a_{ii}}$$

Choosing i, such that this row's l_1-norm[2] is the greatest for all i, and taking the infinity norm of the matrix, the Jacobian will converge if

$$|a_{i1}| + |a_{i2}| + \cdots + |a_{i,i-1}| + 0 + |a_{i,i+1}| + \cdots + |a_{im}| < |a_{ii}| \tag{4.18}$$

[2] $||x||_1 = |x_1| + |x_2| + \cdots |x_n|$, from Kreyszig[32], p.1024

This condition is better known as ensuring that the matrix $\boldsymbol{A}$ is strictly diagonally dominant, i.e. the modulus of the diagonal element must be greater than the sum of the moduli of the other elements in that row.

The question is now whether the matrix $\boldsymbol{A}$ generated from the equations in Section3.1 is diagonally dominant or not.

Examination of (3.9 shows that for diagonal dominance, $|(C_5 + i)|$ must be shown to be greater than $(|C_1| + |C_2| + |C_3| + |C_4|)$. Analysis of the derivation of these coefficients, (3.11) to (3.23) reveals that they are: (a) all positive; and (b) all purely real scalars. The former states that

$$|C_1| + |C_2| + |C_3| + |C_4| = C_1 + C_2 + C_3 + C_4 = C_5 \tag{4.19}$$

Now, rewriting (4.18) as

$$C_5 < |(C_5 + i)| \tag{4.20}$$

and remembering that C_5 is real, then (4.20) must be valid by the triangle inequality. Hence the interior points all lead to diagonally dominant rows in the matrix $\boldsymbol{A}$, for all models geometries.

In the B polarisation case, the side boundaries are governed by a Dirichlet boundary condition, in that $X^{\pm} = X^{\pm\infty}$, so are automatic, in that the diagonal element is unity, and all the others in that row are zero. The asymptotic relation between $U^{\pm}$ and $U^{\pm\infty}$ in the E polarisation case complicates matters. (3.31) and (3.31) show that for diagonal dominance

$$|U^{\pm}(z_{n-1})| > |U^{\pm}(z_n)| \tag{4.21}$$

or, that the magnitude of U in a one dimensional solution is always decreasing as z increases.

In a one dimensional Earth, the field is governed by the well known differential equation

$$\frac{\partial^2 U}{\partial z^2} = i\omega\mu_0\sigma U. \tag{4.22}$$

This equation holds true in layers of constant conductivity, and has solution

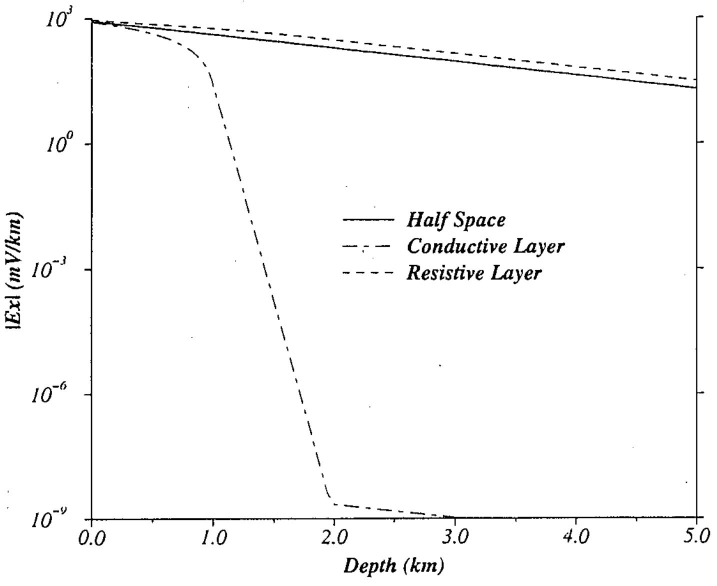

Figure 4.2. U versus z in a one dimensional stratified Earth

in the j^{th} layer, as expressed by Levy et al.[36]

$$U_j(z,\omega) = A_j e^{-\gamma_j(z_{j-1}-z)} + B_j e^{\gamma(z_{j-1}-z)} \tag{4.23}$$

where A and B (replacing their U and D to avoid confusion) are the amplitudes of up-going and down-going waves at the top of the j^{th} layer, and $\gamma_j = \sqrt{i\omega\mu_0\sigma_j}$. However, the recursive relations for the amplitudes, working from the bottom half space, where $A = 0$, up to $z = 0$, do not immediately lead to the desired result.

Figure 4.2 illustrates $|U(z)|$ calculated for three different models:

1. half space 1 Ωm,
2. 1km thick conductive layer (1000Ωm) embedded 1km in the half space, and
3. 1km thick resistive layer (0.001Ωm) embedded 1km in the half space.

The half space shows the expected behaviour, in that there is no upward travelling wave (A is zero), so that there is a simple exponential decay in U. The resistive layer slows the decay with depth, but does not reverse it. The conductive layer exhibits the greater dissipation of energy, which starts in the host material, well above the boundary. This is due to the phase change of the reflected wave decaying as it travels upwards.

Combining the evidence from this brief experiment, and a hand waving argument regarding energy dissipation, I conclude that U almost certainly does decrease monotonically with depth, no matter the model configuration.

The equations describing the coefficients for the top and bottom surface boundaries do not lend themselves to this analysis, but a check has been added to the subroutine which evaluates them. If a row is found to be non-diagonally dominant, then the operator is alerted and advised that the model may not converge. In my experience, this warning has never been issued, and the Jacobian method has converged for every model with which it has been tested.

Parallel Simultaneous Relaxation

I will now describe in some detail the implementation of the Point Jacobian iterative solution on a SIMD computer[3]. Each model grid point is mapped to a (virtual) processor element in the computer, and the local field value and coefficients are stored on that element's memory. Table 4.1 lists the High Performance FORTRANvariable names which are referred to by the code fragments.

The first iteration requires an initial estimate, $\boldsymbol{u}^{(0)}$, to be made. This is often set at zero, except where Dirichlet boundary conditions pertain, but the method is not dependent on $\boldsymbol{u}^{(0)}$, and will converge for any initial values, so long as $\boldsymbol{A}$ is diagonally dominant. However, if $\boldsymbol{u}^{(0)}$ is already close to the final solution, then fewer iterations will be required to reach satisfactory convergence. Therefore, N_y one dimensional solutions are calculated, one for each column of grid points in the model, and the iterative process is started from these values. In areas of the model distant from lateral changes in conductivity, the one dimensional values will approximate the final two dimensional solution, as discussed in Section 3.1.2.

[3]See Section 2.2.2 for a full description of the SIMD computing environment.

Variable	Algebraic name	Description
F	$u^{(p)}$	Current field values
New_F	$u^{(p+1)}$	Next iteration field values
Source	b	Right hand term
Self	a_0	Coefficient for $u^{(p)}_{i,j}$
Left	a_1	Coefficient for $u^{(p)}_{i-1,j}$
Right	a_2	Coefficient for $u^{(p)}_{i+1,j}$
Up	a_3	Coefficient for $u^{(p)}_{i,j-1}$
Down	a_4	Coefficient for $u^{(p)}_{i,j+1}$
Surface	$\boldsymbol{R}$	Surface integral coefficients

Table 4.1. 2D Arrays stored on processor elements

The High Performance FORTRANCSHIFT[4] intrinsic function operates on the field value array four times, as shown in the code fragment, Figure 4.3.

4.1.5 Successive Relaxation

The conceptual thrust behind the development of *successive relaxation* methods is to introduce newly calculated values into the ongoing calculations as early as possible. Hence these are no longer simultaneous, in that all the values cannot be updated in one discrete step.

The simplest example of successive relaxation, *Gauss-Seidel*, is defined by

$$u^{(p)}_{i,j} \leftarrow u^{(p+1)}_{i-1,j}, u^{(p+1)}_{i,j-1}, u^{(p)}_{i+1,j}, u^{(p)}_{i,j+1}. \tag{4.24}$$

Traditionally, in a serial computer, this has been implemented as a 'iteration front' propagating from $u_{1,1}$ down to $u_{N_xN_y,N_xN_y}$, with each line being updated at a different iteration level. Figure 4.4 shows this in action.

The dependencies in (4.24) do not, in themselves, demand numerous coexisting iteration levels in the array, but only two.

By labelling the elements as either black or white, a modified scheme can be followed. The values of all the black elements are updated simultaneously,

[4]See **Regular Communications** on page 29 for a description of CSHIFT.

```
C    Simultaneous relaxation of F(Ny, Nz)

     DO Iteration = 1, Max_Iteration
        :
C    Now update the central points

       New_F = Factor * (Left * CSHIFT (F, SHIFT=-1, DIM=1) +
         Right * CSHIFT (F, SHIFT=1, DIM=1) +
         Up * CSHIFT (F, SHIFT=-1, DIM=2) +
         Down * CSHIFT (F, SHIFT=1, DIM=2) - Self * F)
        :
C    Check for convergence

       IF (Residual .LT. Tolerance ) THEN
         RETURN
       END IF

     END DO
```

Figure 4.3. Five point High Performance FORTRANJacobian Iteration Step

using the initial values of the surrounding elements. The newly calculated black values are now available for the update of the white elements. These two updates constitute one iteration, as both are required to update the whole array.

This act of labelling the elements is known as *ordering* the iteration, and many different patterns exist, depending on the nature of the problem. The traditional, serial ordering presented in Figure 4.4 is known as the *natural* ordering. Ortega & Voigt [45] present an excellent review of many ordering schemes for a wide range of problems, including a three coloured ordering for nine point finite differences.

For this particular problem, two orderings are applied; horizontally banded and chequer board, each of which are described below.

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5\cdots \\ 2 & 3 & 4 & 5 & \\ 3 & 4 & 5 & & \\ 4 & 5 & & & \\ 5 & & & & \\ \vdots & & & & \end{bmatrix}$$

Figure 4.4. The Gauss-Seidel *iteration front.* The figures denote the iteration level at which each element in the array is first updated.

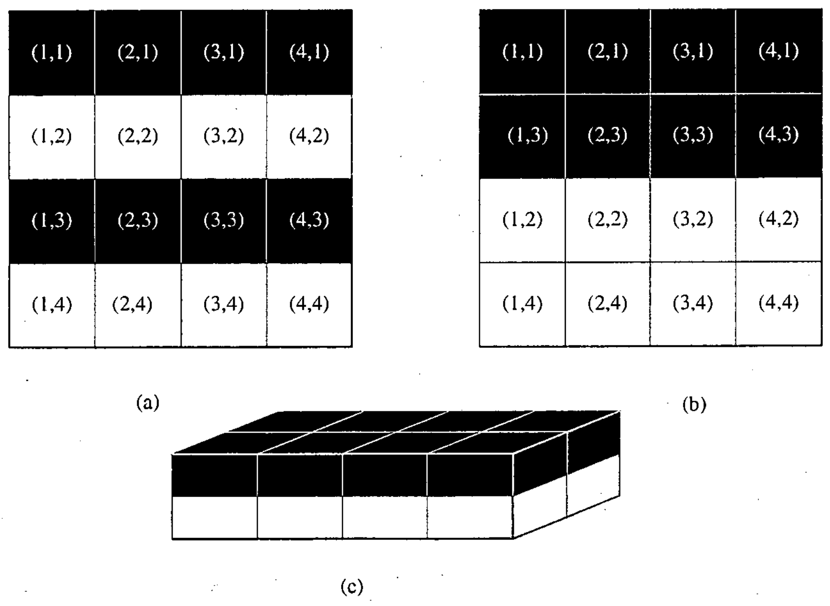

Figure 4.5. 2D Ordering Scheme: (a) Horizontal banding ; (b) Partitioned into black and white ; (c) Black stacked on top of white

Horizontally Banded Ordering

This ordering, as shown in Figure 4.5(a), was chosen as an intermediate step before attempting chequer board ordering, because it results in a simpler remapping of model nodes to processing elements. It also extends the partitioning already enforced by the horizontal surface boundary conditions.

The modified iteration dependencies can now be written as

$$\text{Black } u_{i,j}^{(p+1)} \leftarrow u_{i,j-1}^{(p)}, u_{i-1,j}^{(p)}, u_{i,j+1}^{(p)}, u_{i+1,j}^{(p)}, u_{i,j}^{(p)} \tag{4.25}$$

$$\text{White } u_{i,j}^{(p+1)} \leftarrow u_{i,j-1}^{(p+1)}, u_{i-1,j}^{(p)}, u_{i,j+1}^{(p+1)}, u_{i+1,j}^{(p)}, u_{i,j}^{(p)} \tag{4.26}$$

where the black elements are updated before the white.

A High Performance FORTRANmask[5], `Black`, can be created with the same pattern as Figure 4.5(a), and two successive point Jacobian iterations, one using the `WHERE (Black) ...END WHERE` and the other using the negated `WHERE (.NOT. Black) ...` scoping constructs.

However, splitting the iterative process into two sequential updates removes one degree of parallelism from the operation. The `WHERE` statements turn off half of the processing elements for each sub-iteration, resulting in a great drop of machine efficiency, as it is highly unlikely that the allocation of virtual processors to physical processors in the computer is such that all physical processors are busy for both sub-iterations.

This problem can be overcome by a non-trivial remapping of model grid nodes to virtual processors. Figure 4.5(b) and (c) illustrate this mapping. The two colours are segregated, and stacked white on top of black, giving each virtual processor one grid point of each colour. Under this allocation, all physical processors will be active, no matter which colour of grid nodes are being updated.

Instead of resorting to masks, it is simpler to operate on *array sections*[6]. For example, to select all the black nodes from the array use `G(:, :, Black)`, and for white use `G(:, :, White)`, where `G` is the remapped array of values, and

[5]See **Scope and Masks** on page 29.
[6]See **Array Sections** on page 30

`Black` and `White` are scalars, valued zero and one respectively.

From this point onwards, I will use the following notation to specify local data communication within a parallel data array. Individual elements are denoted $\nu_{\delta_x\delta_y\delta_z}$, the two dimensions, (YZ), of the model grid having been remapped to three dimensions in the parallel computer. The third dimension contains the colour information of each node. The central element in this local coordinate system is labelled ν_{111}, and since I am only considering nearest neighbour interactions, the range for these subscripts is $0 \leq \delta \leq 2$.

After transforming, the horizontally banded dependencies are

$$\text{Black } \nu_{111}^{(p+1)} \leftarrow \nu_{102}^{(p)}, \nu_{011}^{(p)}, \nu_{112}^{(p)}, \nu_{211}^{(p)}, \nu_{111}^{(p)} \tag{4.27}$$

$$\text{White } \nu_{111}^{(p+1)} \leftarrow \nu_{110}^{(p+1)}, \nu_{011}^{(p)}, \nu_{120}^{(p+1)}, \nu_{211}^{(p)}, \nu_{111}^{(p)} \tag{4.28}$$

These are much more complex than before the remapping, with some of the operations, (Black_{102} and White_{120}), requiring a shift along two dimensions, and therefore a longer time to complete the data communication. This apparent increase in communication overhead is overestimated, as all the $11x$ translations are simply moving data between two virtual processors, both of which are allocated to the same physical processor. In fact, all communication along the third (colour) dimension takes place in local memory, not between physical processors.

Chequer Board Ordering

Another regular remapping of the nodes to processors as shown in Figure 4.6 results in a higher proportion of new iterate to be used in the calculation of the current iteration level.

Special cases have to be made at the top and bottom of the model, where the integral boundary conditions (only the bottom for B Polarisation problems) demand a different ordering.

Black:

$$\nu_{111}^{(p+1)} \leftarrow \nu_{012}^{(p)}, \nu_{112}^{(p)}, \nu_{212}^{(p)}, \nu_{122}^{(p)}, \nu_{111}^{(p)} \tag{4.29}$$

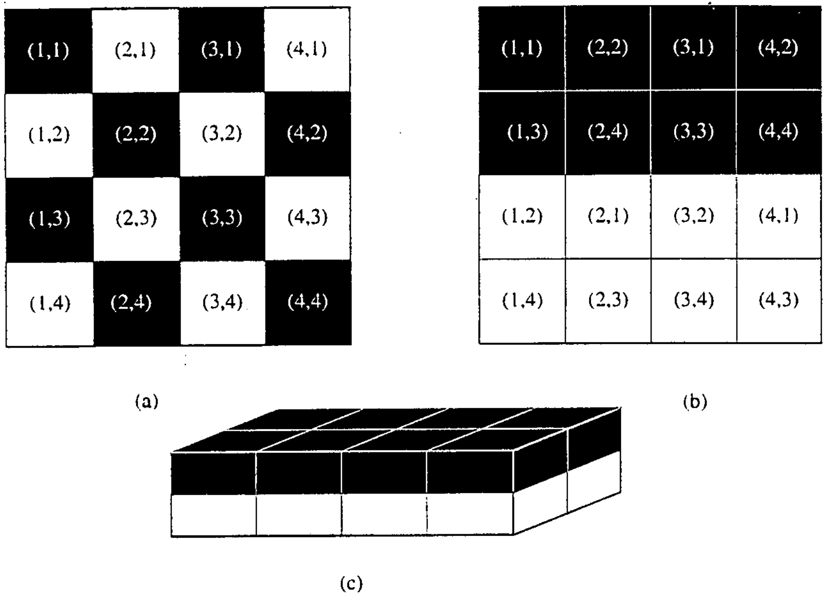

Figure 4.6. 2D Ordering Scheme: (a) Chequer banding ; (b) Partitioned into black and white ; (c) Black stacked on top of white

White:

$$\nu_{111}^{(p+1)} \leftarrow \nu_{010}^{(p)}, \nu_{100}^{(p)}, \nu_{210}^{(p)}, \nu_{110}^{(p)}, \nu_{111}^{(p)} \tag{4.30}$$

4.1.6 Successive Over-relaxation

Successive over-relaxation is a variant of the Gauss-Seidel which can improve the rate of convergence of the function u by adding an amount $\omega\Delta u$, where Δu is the change due to the standard Gauss-Seidel iteration. The quantity ω is termed the *acceleration parameter* or *relaxation factor*.

It can be shown that this converges for $\omega < 2$, for example in Varga[64], yet the exact choice of ω is crucial to the effectiveness of the method. Values less than unity lead to *underrelaxation*, more than unity to the desired overrelaxation, and when $\omega = 1$ the method degenerates to Gauss-Seidel. There is an optimum value, ω_{opt}, for most rapid convergence, but its computation is inordinately expensive for these forms of problem. Unfortunately, the convergence rate does not vary symmetrically around ω_{opt}. It improves gradually as ω approaches ω_{opt} from below, and rapidly deteriorates for $\omega_{\text{opt}} < \omega < 2$.

4.1.7 Iterative Starting Values

Unlike the serial Gauss-Jordan matrix inversion algorithm employed by Poll, these iterative solutions can be helped by setting the anomalous field values before iterating to the final solution. In fact, it is common practice to solve a system directly, and then apply an iterative technique to reduce rounding errors in the original solution[18].

Poll's one dimensional solutions for the $y = \pm\infty$ boundary conditions essentially follow those of Schmucker[56]. She only solves for $F^{\pm}$, and sets the anomalous fields to be $\tilde{F}^{-} = 0$ and $\tilde{F}^{+} = F^{+} - F^{-}$. As each one dimensional solution is strictly independent, I have implemented a parallel version which solves for each column in the grid simultaneously. The anomalous field is then set by subtracting F^{-} from each solution.

4.1.8 Parallel Integral Evaluation

The five point scheme breaks down at the edges of the model grid, for two reasons. One of the points will always be missing, but this can easily be overcome by setting the appropriate coefficient to be zero, so that when the whole field array is shifted the product for that imaginary element will be zero.

The integral boundary conditions employed at the bottom, and the top in E polarisation, require wider data communication between grid cells. The mechanics of calculating the surface integrals is the same for the top and bottom, so I will only describe how the top layer is evaluated in the E polarisation.

The data dependencies inherent in (3.34), (3.35) and (3.36) can be written as

$$u_{i,1}^{(p+1)} \leftarrow u_{i,2}^{(p)}, u_{1,1}^{(p)}, u_{2,1}^{(p)}, \cdots, u_{N_y-1,1}^{(p)}, u_{N_y,1}^{(p)} \tag{4.31}$$

except that $u_{i,2}^{(p)}$ is replaced by $u_{i,N_z-1}^{(p)}$ for (3.35) and (3.36). The formulæ can also be written in a general form

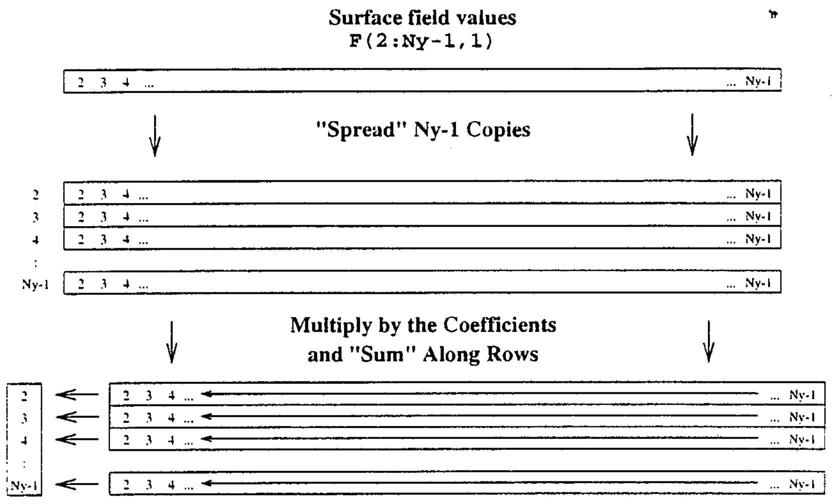

Figure 4.7. Evaluating the two dimensional surface integral in parallel.

$$u_{i,1}^{(p+1)} = \frac{1}{a_{i,1}} \left(\sum_{m \neq i}^{1,N_y} R_i^m u_{m,1}^{(p)} + b_{i,1} u_{i,2}^{(p)} + c_{i,1} \right) . \tag{4.32}$$

The SIMD computing model does not allow the central points and these boundaries to be calculated simultaneously, so the majority of processors (allocated to central points) are available to help when the integrals are being evaluated. To achieve this, the workload is distributed by performing the elemental matrix multiplication over the whole virtual processor space.

Every point along the surface is updated simultaneously by constructing a two dimensional matrix of the surface field values, which is conformant with the coefficient matrix $\boldsymbol{R}$ in (4.32). Each product $R_i^m u_{i,1}^{(p)}$ is then calculated locally on each virtual processor, and the summation is made along the rows of the resulting array, as illustrated in Figure 4.7.

The High Performance FORTRAN `SPREAD` intrinsic function allows the one dimensional array section `F(2:Ny-1,1)` to be replicated to form the two dimensional array. The `SUM` reduction operator[7] with the appropriate dimension specified then performs the summation, and reduces back to a one dimensional array,

[7]See **Reduction Operators** on page 28.

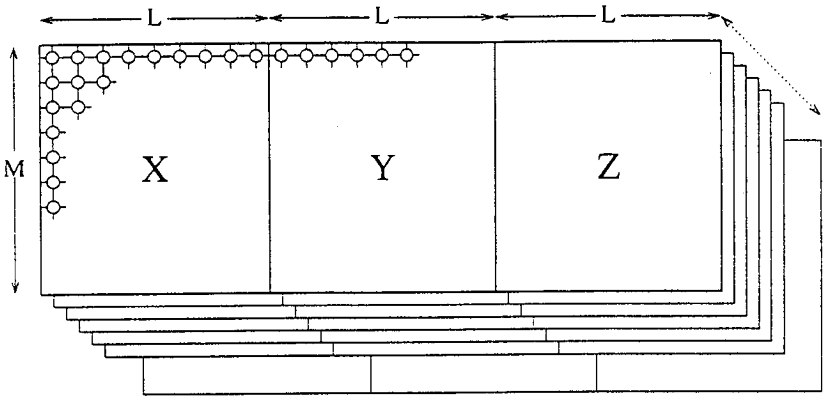

Figure 4.8. Three dimensional field component storage scheme

which is substituted into (4.32) and finally assigned to `New_F(2:Ny-1,1)`.

4.1.9 Three Dimensional Array Storage

The two dimensional algorithm only involves the solution of a scalar quantity, from which is deduced the other two field components. The three dimensional model, on the other hand, solves for all three components of the magnetic field, X, Y and Z. The finite difference equations for the interior points given in (3.42), (3.43) and (3.44) allow the independent updating of each component in a Jacobian solver.

In practice most models seem to have approximately the same number of grid points in the two horizontal axes, but the MasPar has a rectangular array of processors. Placing the three components on three sets of processors keeps more processors busy than arranging them as three elements in a local array on each processor. Figure 4.8 shows this partitioning for one horizontal plane in a $L \times M \times N$ model grid. The third dimension in the diagram does not represent z in the model, but the coefficients c_{ijk} in the equations.

When cross component terms are to be evaluated, a simple CSHIFT with $-L$ as an argument for the x axis will bring all the Y values to their corresponding X components, and so on.

4.2 Two Dimensional Task Farm

The major thrust of this chapter has been in describing the fine grained parallelisation of the two and three dimensional codes. However, since responses at multiple inducing frequencies are almost always calculated, and each solution is in practice independent of the others, it is possible to implement a classic task farm, as described in Section 2.5.3.

This approach required minimal changes to the original serial code, since the models are being executed on the same architecture. These changes were as detailed below.

- A *source* and a *sink* subroutine had to be written. The source reads in the model input file, counts how many tasks (models and periods) there are, and issues a task number to each *worker* as they became available. The sink was only necessary as a stub for the PUL-TF library.
- The main part of the program had to be able to pick an individual model and period out of the input file at random, and not solve each one sequentially in a loop.
- All file I/O had to be modified to read and write to uniquely named files, as several instances of the same code would be running simultaneously. The task number was used as an extra extension to the file name.

Appendix C contains code excerpts for most of these changes. The file I/O changes were critical since the workstation cluster operates on a shared filespace. That is, the same code running on two different machines reads and writes to the same directory.

4.3 Model Performance

4.3.1 Two Dimensional Model

Figure 4.9 shows how the parallel Point Jacobian solver converges. Both the axes are on a logarithmic scale to make the differences visible. The steady

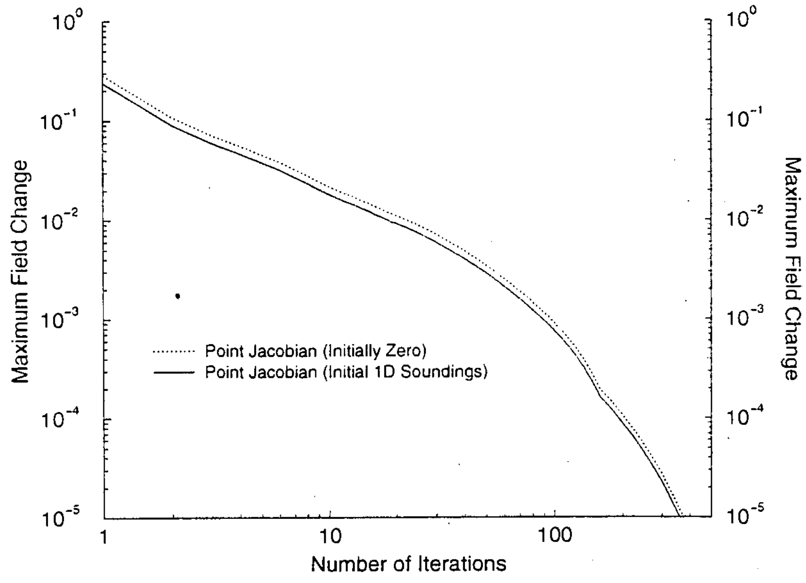

Figure 4.9. Convergence rates for the Point Jacobian parallel solver starting with zero anomalous field and with the one dimensional solutions.

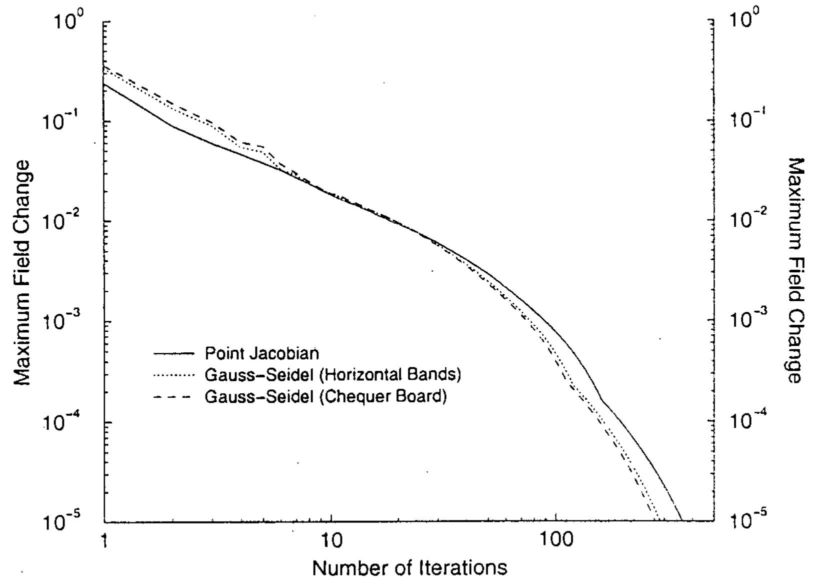

Figure 4.10. Convergence rates for the parallel Point Jacobian and Gauss-Seidel, both Horizontal Banding and Chequer Board, iterative schemes.

decrease is as expected, with the same fractional reduction in iteration error for each decade of iterations, as discussed for these methods by Forsythe[17].

Setting the initial guess for $u^{(0)}$ to be the one dimensional soundings at each mesh column has the effect of shifting the curve down a fraction. However, this is not an acceleration, since the gradient has not been changed. Of course, convergence is reached more quickly, since the solver is given a head start. All the iterative solutions in the rest of this section apply this initial estimate for the anomalous field.

Acceleration is evident in Figure 4.10. The solid curve represents the basic point Jacobian method, as seen in Figure 4.9. Both of the Gauss-Seidel methods, horizontal banded (dotted curve) and chequered (dashed curve) mappings, start off making bigger changes to the iterates. Half way to convergence, the Jacobian begins to change the iterates more, but the Gauss-Seidel methods reach the stopping criterion earlier.

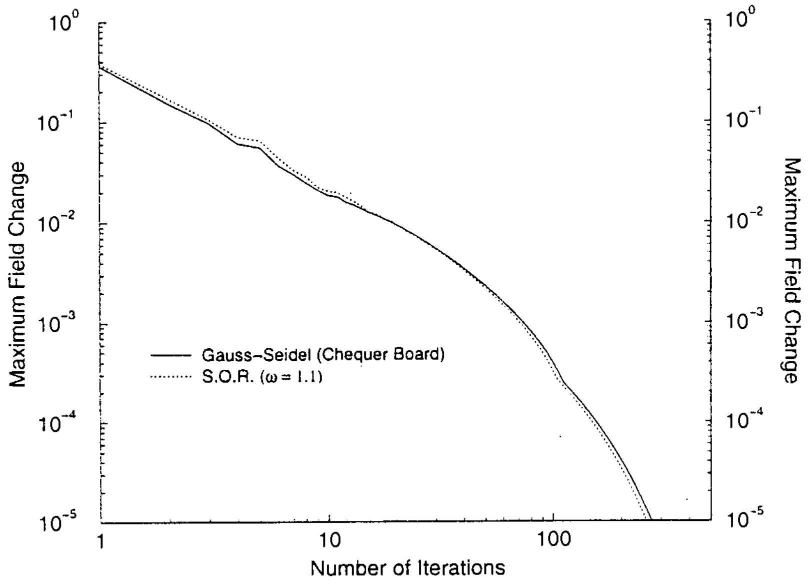

Figure 4.11. Convergence rates for the parallel Chequer Board Gauss-Seidel and Successive Overrelaxation ($\omega = 1.1$) iterative schemes.

The difference between the Jacobian and the horizontally banded Gauss-Seidel is greater than the extra advantage of the chequer board mapping. Analysis of the ratio of iterate levels in the update stages suggests the horizontally banded could require approximately two thirds the iterations of the Jacobian. Similarly the chequer board mapping should reduce this fraction to one half. This particular model attained convergence in 80% and 75% of the Jacobian iterations respectively. These ratios are dependent on the model structure, especially the proportion of horizontal boundary points to body points. For example, the model in Table 4.3 attained convergence within 64% and 55% of the Jacobian iteration count.

At first, the notion of applying successive over relaxation to the chequer board mapping offered the prospect of significantly more rapid convergence. However, it was soon found that any choice of ω much greater than 1.1 was susceptible to divergence for some models. Figure 4.11 shows that this method (dotted

Computer	Solver	Iterations	Time (s)	Iteration Time (ms)
Sparc2	Gauss-Jordan	–	2	–
Sparc2	Chequered G-S	494	3.5	7
Sparc2	Banded G-S	528	4	7
Sparc2	Chequered S.O.R.	509	4	8
Sparc2	Jacobian	627	9	6
CM200	Chequered G-S	494	8	16
MasPar	Chequered G-S	494	8	15
CM200	Banded G-S	528	8	15
MasPar	Banded G-S	528	8	15
CM200	Chequered S.O.R.	509	9	17
MasPar	Chequered S.O.R.	509	9	17
CM200	Jacobian	627	9	14
MasPar	Jacobian	627	9	14

Table 4.2. Performance for two dimensional (23×9) model

curve) does indeed accelerate the Gauss-Seidel (solid curve), but with such a low relaxation parameter, the number of iterations is not significantly reduced.

Although Figures 4.10 and 4.11 show that the S.O.R. algorithm requires the fewest iterations to converge, it is not the quickest in real calculations. This can be attributed to the greater amount of work required in each iteration. Had larger acceleration parameters resulted in convergence, then this extra multiplication and addition may have been worth while. Tables 4.2 and 4.3 show that in terms of wall clock time, the chequer board, and sometimes the horizontally banded, Gauss-Seidel is faster. The last column in these tables indicates that there was indeed a time penalty per iteration for all the machines.

Tables 4.2 and 4.3 summarise the performance of the two dimensional code on a variety of different hardware for two models. Comparing the execution times on the Sun Sparc2 serial workstation, it is obvious that the direct Gauss-Jordan solution performs well on this type of machine, being an order of magnitude faster for the large model. For smaller models, the typical workstation running this code at least as fast as or faster than the parallel supercomputers. However, as Table 4.3 shows, it starts to fall behind the iterative solutions for larger and

Computer	Solver	Iterations	Time (s)	Iteration Time (ms)
CM200	Chequered G-S	4306	69	16
CM200	Chequered S.O.R.	4251	74	17
CM200	Banded G-S	5012	80	16
CM200	Jacobian	7781	105	14
Sparc2	Gauss-Jordan	–	129	–
MasPar	Chequered G-S	4306	129	30
MasPar	Chequered S.O.R.	4251	140	33
MasPar	Banded G-S	5012	145	29
MasPar	Jacobian	7781	200	26
Sparc2	Chequered G-S	4306	1257	292
Sparc2	Chequered S.O.R.	4251	1339	315
Sparc2	Banded G-S	5012	1464	292
Sparc2	Jacobian	7781	1891	243

Table 4.3. Performance for two dimensional (123×66) model

larger models.

It is also interesting to note how the time per iteration changes for each of the SIMD machines between the two models. The CM200's times remain constant, whereas the MasPar takes approximately twice the time for each iteration of the larger model. Referring back to Section 2.6, it becomes apparent that this discrepancy is due to the capacities of the two machines. The MasPar has had to simulate two virtual processors on each physical processor to accommodate the larger grid, but the CM200, with double the processor count does not need to. If the number of vertical grid points had been reduced to less than 64 (the processor array is 128×64) then the MasPar's times would have been equivalent to the CM200. This is a perfect illustration of the need to design suitable grids for these types of computer.

4.3.2 Three Dimensional Model

The three dimensional model's performance exhibited much the same behaviour as the two dimensional one discussed above. Table 4.4 summarises some

Computer	Model	Solver	Iterations	Time (s)	Iteration Time (ms)
AXP	$7 \times 7 \times 7$	Serial G-S	81	2	25
Sparc2	$7 \times 7 \times 7$	Serial G-S	81	11	136
CM200	$7 \times 7 \times 7$	Jacobian	134	23	172
MasPar	$7 \times 7 \times 7$	Jacobian	134	53	400
AXP	$15 \times 15 \times 10$	Serial G-S	349	116	330
CM200	$15 \times 15 \times 10$	Jacobian	596	228	380
MasPar	$15 \times 15 \times 10$	Jacobian	596	367	620
Sun 2000	$15 \times 15 \times 10$	Serial G-S	349	401	1150
Sparc2	$15 \times 15 \times 10$	Serial G-S	349	685	1960
CM200	$29 \times 29 \times 9$	Jacobian	671	487	730
AXP	$29 \times 29 \times 9$	Serial G-S	404	1587	3930
Sun 2000	$29 \times 29 \times 9$	Serial G-S	404	3132	7750

Table 4.4. Performance for three dimensional models

timings on a variety of hardware. Once again, it's only for the larger models that the SIMD machines are faster than the serial machines.

The MasPar could only solve the smaller models, performing less than twice as fast as a typical desktop workstation for the medium sized grid. The MasPar's relatively poor performance, as measured against the CM200, is due to its two dimensional processor array. The CM200, with its dynamic interprocessor communication network is more efficient.

The largest model has the maximum number of grid points which can be accommodated on the CM200. When this machine is running at full capacity, i.e. optimum load balance, it is significantly quicker than even a highly specified serial machine. However, it is limited by these constraints of memory limits.

A closer examination of the computational profile of the three dimensional iteration steps reveals that the same proportion of time is being spent in each task for both the serial and parallel codes. Assuming the parallel machines are processing the interior points efficiently, then this equivalence indicates the boundary condition surface integrals evaluations are also efficient.

Subroutine	Serial	Parallel
Thin Sheet & Top	32%	35%
Interior	42%	41%
Bottom	26%	22%
Residuals	–	2%

Table 4.5. Profile of serial and parallel three dimensional iteration step for $15 \times 15 \times 15$ model.

Workstations	Execution Time	Speed Up
1	143	1.0
3	58	2.5
5	40	3.6

Table 4.6. Performance of the two dimensional task farm.

4.3.3 Task Farm

It proved difficult to gain consistent performance figures for the task farm runs, as each workstation was a resource shared between many students. One or more workstations could be busy with another CPU intensive task without the source process being aware, thus making load balancing problematic and haphazard.

Table 4.6 shows the execution times and speed up factors for a model run of ten periods. The five workstation task farm achieved a lower than expected speedup (3.6 as opposed to closer to 5) because one workstation solved two larger grids. The others were then left idle, waiting for the last task to complete.

The ease of running the task farm version of the two dimensional code, instead of submitting jobs to a batch queue on a SIMD machine, prompted me to make the task farm my first choice for running routine models.

Chapter 5

Two Dimensional Gridding Strategies

5.1 What Makes a Good Grid?

The validity of any particular grid is a complex concept. Several criteria, over and above the conductivity structure, have to be weighed against each other

- Different programmes, using different models, will have different error responses to the same grid.
- The accuracy of the generated responses. Generally, the higher the grid point density, the more accurate the calculated response will be.
- The computation time allocated to the particular problem. Grids with numerous points will take longer to solve than sparse grids.

The dominant consideration is the conductivity structure itself. Combined with the temporal frequency of the field, the structure creates a distribution of *skin depths* over the model space. Parkinson[47], along with many others. eloquently introduces the concept of the electromagnetic skin depth. It is defined as

$$s = \sqrt{\frac{2}{\mu\omega\sigma}} \tag{5.1}$$

where ω is the angular frequency of the field and σ is the conductivity. Considering $\boldsymbol{B}$ as a damped wave, propagating in a conductor, with magnitude (derived from the diffusion equation)

$$B = B_0 e^{-z/s} e^{-i(\omega t - z/s)} \tag{5.2}$$

it becomes apparent that for every increase in depth by s, the amplitude decreases by a factor e and the phase changes by one radian. The skin depth, combining the conductivity of the host medium and the frequency of the field, is therefore the scale length against which model grids are measured.

Weaver[73] gives some guidance on grid generation, which is particularly relevant to finite difference solutions. In paraphrase, he presents these three guidelines

1. Grid spacing should be no more than one quarter a skin depth, except more than two skin depths away from a conductivity boundary. Finer spacing is desirable very close to the boundary.

2. The grid should extend, horizontally, at least three skin depths away from a vertical conductivity boundary.

3. The grid should be locally symmetric around conductivity boundaries, and should vary as smoothly as possible, with no more than a doubling or halving anywhere.

Wannamaker *et al.*[69] present the following, more detailed. guidelines in the user documentation for their two dimensional finite element model.

1. Adjacent element dimensions should not change by more then a factor of 3 to 5.

2. Element dimensions should be approximately $\delta/4$ in the vicinity of changes in resistivity.

3. No single resistivity block should be less than 4 elements wide or 3 elements thick to fit galvanic components of the field.

4. 2 or 3 δ away from variations in conductivity element dimensions may be increased to the order of δ of the medium.
5. Vertical element dimensions may be increased approximately exponentially downwards, but the maximum should still be held to 1 or 2 δ.
6. The mesh should extend horizontally to 8 or 10 δ away from the nearest 2-D structure, or 10 to 12 times the height of the inhomogeneous structure.
7. The bottom mesh boundary should be extended to 8 to 10 δ of the background conductivity from the air interface, or 10 to 12 times the width of the inhomogeneous structure.

Even following these guidelines, some additional tuning may be required after inspecting the results and any estimates of errors which the model provides. Wannamaker et al.[70] experimented with different grid geometries to overcome problems with differing machine precision.

5.2 Automatic Grid Generation

5.2.1 Review

The construction of grids has long been attempted by the modelling programs themselves. This is seen as increasingly important, as the size and complexity of models has grown over time. The advent of three dimensional modelling on generally available computing hardware will soon be adding to the demand for automatic grid generation. At the moment, the number of grid points is still small for these three dimensional models, and a degree of intelligence is required to ensure that the nodes are placed where they are most required, as the traditional criteria break down in a node famine.

As discussed in Chapter 1, geoelectromagnetic numerical models fall into several distinct categories. As far as gridding is concerned, another division is apparent: whether the grid is Cartesian or not. The bulk of research in automatic gridding in other disciplines, such as computational fluid dynamics, is directed

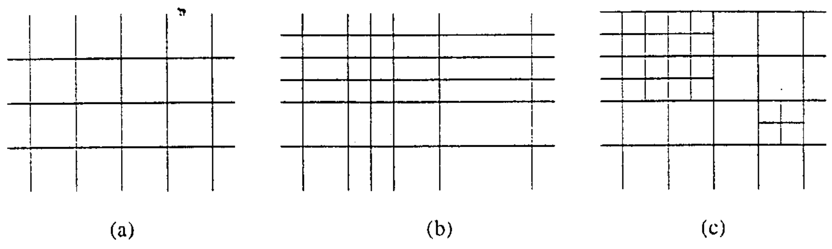

Figure 5.1. Cartesian grids in two dimensions: (a) Regular; (b) Irregular; (c) Disconnected

towards non-Cartesian grids in two dimensions, and disconnected Cartesian grids in three dimensions.

Madden's EMCAL [38] program of 1971 allowed the user to define a sparse grid, consisting mainly of the block boundaries and site locations. EMCAL would then examine the grid spacing, and insert more lines of nodes if required. The original grid was not altered, merely lines of grid points added half way between existing lines. This simplistic approach had the unfortunate result of rapidly varying grid spacings, often closest to the conductivity boundaries, thus clearly violating Weaver's condition (3).

Poll[48] devised a method which took the opposite approach, in that it generated a grid automatically, and then allowed the operator to make adjustments if he so wished. The algorithm tries to follow the guidelines set out above. It treats the problem as that of generating two independent one dimensional grids, one for the horizontal spacings and one for the vertical spacings.

The conductivity structure is *compressed* along one axis, as shown in Figure 5.2, to form a series of segments from one conductivity junction to the next. Independent local grids are grown outwards from the segment boundaries, with the spacing starting at one quarter of a skin depth and increasing exponentially. These local grids are then reconciled to form one grid covering the whole axis of the model. She describes the process in more detail in her Ph.D. Thesis[48].

The Geotools Corporation gives the following prominent warning in their user manual[22]:

Although Geotools attempts to provide the user with some guidance

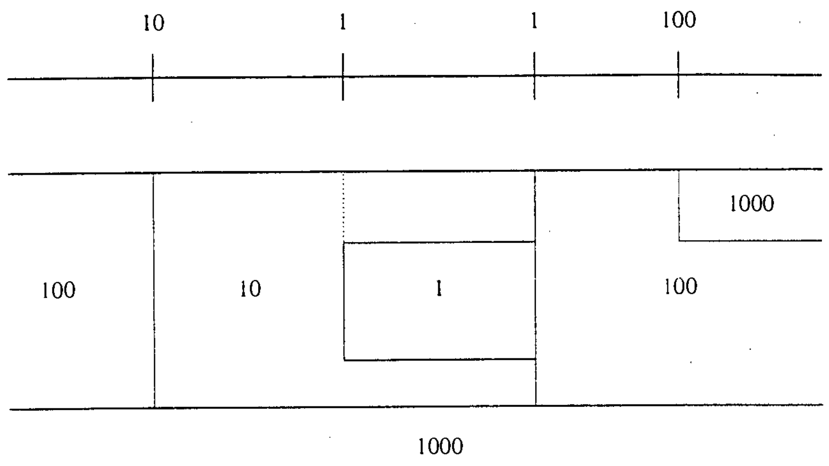

Figure 5.2. Reduction of model structure to one dimension.

regarding the construction of a computationally valid mesh, the user must exercise care to ensure that the earth model is properly discretized.

They then go on to repeat Wannamaker's guidelines[69], and introduce their own algorithm based upon them. It starts with the coarsest grid which accommodates the conductivity structure and then repeatedly examines the rows and columns to determine which require to be split. The splitting criteria are weighted averages calculated along only one edge of the row or column, and they admit that this can cause noticeable problems in the top row. Their algorithm is deliberately generous with grid point allocation, with the aim of depending upon the operator to then manually remove lines where they are not required, if the mesh size is required to be constrained to a minimum.

5.3 Elastic Membrane Grid

5.3.1 Why another algorithm?

The massively parallel versions of the modelling programmes developed in Chapters 4 can exhibit wild fluctuations in efficiency over a narrow range of numbers of

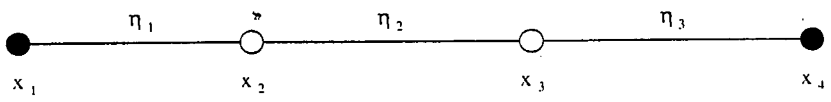

Figure 5.3. Points on an Elastic Grid. η_i are the elastic coefficients. x_i are the grid point locations. • are fixed, unmovable points. ∘ are floating points.

allocated grid nodes. It can often be vital to control the numbers of nodes, so as to avoid a dramatic drop in efficiency as the problem's load balance[1] deteriorates, as is illustrated in Section 4.3.1 on page 72. The automatic and semi-automatic gridding algorithms discussed do not allow fine control of the number of grid points, merely having maxima dictated by size of arrays allocated at compilation.

An algorithm which can most usefully distribute a fixed number of grid nodes over the model space was therefore developed to overcome this problem.

5.3.2 Grid Generation

Since the modelling programmes require a connected Cartesian grid, e.g. Figure 5.1(b), each dimension can be treated independently. The vertical dimension is a special case, but can be handled elegantly by the same algorithm.

As the title suggests, a series of linearly connected grid points is *stretched* over the conductivity structure. Variations in spacing are achieved through varying the elastic coefficients over the model. I will first develop the relaxing algorithm, by which the points are distributed over the model, and then detail how the variations in the elastic coefficients are calculated.

Elastic Relaxation

A prescribed number, N, of grid points, hereafter referred to simply as *'points'*, are distributed evenly along the length of the axis in question. The two end points are fixed at their initial positions, and all the others are *'floating'*; i.e. given the freedom to vary their position.

[1]See Section 2.3 for a discussion of this problem.

Only if the model space is a half-space, and the elastic coefficient invariant along the line, will the equal spacing of the points represent a system with no elastic strain. The net stress at a floating point can be calculated by taking the difference of the opposing elastic stresses on each side of the point, thus

$$\zeta_i = \eta_{i-1}(x_i - x_{i-1}) - \eta_i(x_{i+1} - x_i) \tag{5.3}$$

The point x_i is then displaced to a new location x_i', reducing the elastic strain according to Hooke's Law

$$x_i' = x_i - a\zeta_i, \; 2 \leq i < N \tag{5.4}$$

where a is a scaling factor, generally set to 0.5. It can also be thought of as having a damping influence if it is less than unity. If this value is set too high, then instabilities and oscillations will result; too low and the convergence to equilibrium will be too slow.

Each floating point is therefore displaced in turn, thus reducing the total stress in the system. A check on the displacement must be made. however, to ensure that sequential points do not 'hop' over each other. If this is the case, the current point is moved to within 90% of the distance to the point it wishes to cross.

The serial FORTRAN 77 implementation updates the position of each point in turn, so that the following point, for example, will be updated using the new position of the previous. After all the floating points have been displaced, a convergence test is applied to decide whether the membrane has reached equilibrium. The most straightforward test is to compare the largest displacement to a required tolerance, and succeed if it is smaller.

Although not explicitly included in Weaver's grid criteria, all conductivity boundaries must be sampled by a grid point, which complicates this algorithm. In theory, fixed points could be initially allocated, and being fixed would be guaranteed to remain at the conductivity boundary right through to convergence. However, this would prohibit floating points migrating across boundaries, resulting in local minima of stress being found, instead of near global minima.

Two strategies for allowing floating points to cross boundaries were considered. Firstly, potential wells, which can be thought of as *sticky patches* could be located at the conductivity boundaries. These would act as traps for points, and could be implemented by requiring a large net stress to build up at that point before allowing it to be displaced, as shown in (5.5).

$$x_i' = \begin{cases} x_i - a\zeta_i, & |\zeta_i| \geq \zeta_0 \\ x_i, & |\zeta_i| < \zeta_0 \end{cases} \tag{5.5}$$

This would undoubtably help keep points at the required locations, but could not guarantee their presence, as there is no way to unconditionally replace a point once it has left the trap.

The second approach, which is the one implemented, is to simply allow all points to float, and to rely on the fact that the relaxed state will exhibit a high density of points around the boundary[2]. The closest point is then moved and fixed to the exact boundary location, and the strain relaxed again. However, when two adjacent points are chosen to be fixed to boundaries, the second point is instead moved to between them, set free to move, and its neighbouring point is fixed to the second boundary. This special case guarantees at least one internal grid point in any small conductivity blocks, which otherwise may have been overlooked.

This approach has the advantage of simplicity and does not interfere with the search for the global stress minimum. It is also computationally inexpensive, as only a handful of iterations are generally required for the secondary relaxation stage.

It is equally apparent that field measurement locations must also be sampled by grid points, and these can be inserted into the grid in the same manner as those at conductivity boundaries.

Estimation of the Elastic Function

Each dimension is treated as a projection of conductivity boundaries, resulting in a line segmented into different skin depths. The chosen skin depths are the minima encountered on either side of the boundary. This is analogous to the scheme developed by Poll[48] and described earlier. Figure 5.2 illustrates the

[2]Weaver's first criterion

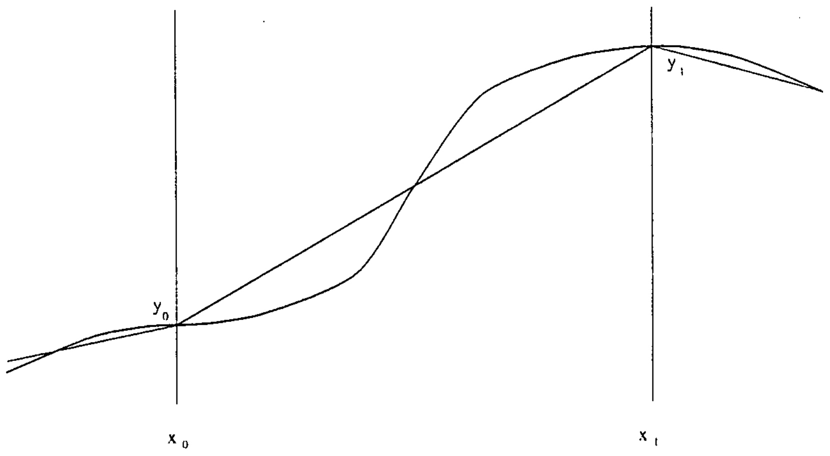

Figure 5.4. Linear and cubic approximations to the elastic function.

procedure. I will first of all develop the method for the horizontal dimension, and then make the necessary adjustments to deal with the special case of the vertical.

Each segment therefore has a skin depth specified at each end, and another, local skin depth in the interior. This algorithm departs from that of Poll, in that it uses these skin depths to generate a grid density function along the line, instead of individually placing points inside each segment.

The fact that the selected skin depths do not vary across a segment boundary, ensures that the elastic function is continuous. This will help result in a smoothly varying grid density across the boundary.

The function should have local maxima at the conductivity boundaries, and local minima between them. The obvious way forward would be to construct a series of linear functions, connecting the proscribed maxima and minima with straight line approximations. However, this would have the unfortunate property of making the elastic function piecewise smooth, destroying some of the symmetry around the boundaries.

In order to preserve global smoothness, third degree polynomials, with stationary points at the maxima and minima were calculated. Figure 5.4 shows the linear and cubic approximations for one half segment, from the internal local minimum at x_0 to the right boundary at x_1.

The cubic approximation can be found, since x_0, x_1, y_0 and y_1 are all known. The constraints are

$$
\begin{aligned}
f(x_0) = ax_0^3 + bx_0^2 + cx_0 + d &= y_0 && (5.6)\\
f(x_1) = ax_1^3 + bx_1^2 + cx_1 + d &= y_1 && (5.7)\\
f'(x) = 3ax^2 + 2bx + c &= 0 \text{ at } x = x_0 \text{ and } x = x_1 && (5.8)
\end{aligned}
$$

Substituting $x_0 = 0$ and some simple algebra gives $d = y_0, c = 0, b = -\frac{3}{2}ax_1$ and

$$a = \frac{y_1 - y_0}{x_1^3 - \frac{3}{2}x_1^2} \tag{5.9}$$

The nonzero coefficients a and b, along with y_0, y_1 and x_1, are stored for each half segment, so that the elastic coefficient can be calculated anywhere along the grid.

The actual values of y_0 and y_1 need to be determined from the skin depths, in a manner that preserves the relationship between skin depth and grid spacing. Since the grid spacing is directly proportional to the skin depth, the elastic coefficient must be inversely proportional to the skin depth.

Figure 5.5 shows a typical horizontal grid generated by this algorithm.

Vertical Grid Generation

As the electromagnetic fields diffuse into the Earth, their amplitudes diminish in a fashion linked to the skin depth. Therefore, the grid for the vertical axis should become more sparse as depth increases. The finite difference approximation still requires, however, that the spacing be regular and more dense around conductivity boundaries.

The elastic membrane algorithm can generate such grids, simply by altering the elastic coefficient function. The cubic approximation developed for the horizontal grid ensures that the spacing around the conductivity boundaries is valid, and the elastic function can be made to decrease with depth by applying an envelope. This envelope is simply a linearly decreasing function, ranging from 1 at

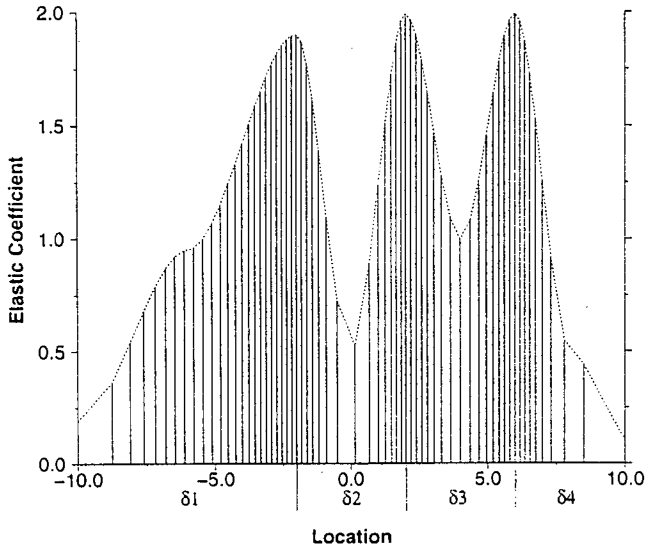

Figure 5.5. Example elastic function calculated at the grid points, denoted by the impulses. The δ's are the skin depths in each partitioned region.

the surface to 0.1 at the base of the model.

5.3.3 Grid Verification

As it stands the elastic membrane algorithm does not render operator intervention unnecessary. Some method of verifying the grid is required, and a measure of its validity presented to the operator, who can then take further action to insert or adjust the position of individual points.

The verification takes several steps, with each contributing a weighted score.

- The grid spacing is checked to vary by no more than a factor of two.
- The grid points on either side of a conductivity boundary are checked to be at least 90% symmetric.
- The grid spacing, around a conductivity boundary, is no more than one quarter of a skin depth.

- No grid spacing is more than two skin depths.

5.4 The COMMEMI 2-D Project

The COMMEMI, or COmparison of Modelling Methods in Electro-Magnetic Induction problems, was first proposed by Zhdanov at the 6th IAGA Workshop on Electromagnetic Induction in the Earth and Moon in 1982. It aimed to:

1. estimate the accuracy, effectiveness and universality of existing modelling programs;
2. select the most suitable programs for the International Laboratory of Numerical Electromagnetic Modelling at the University of Oulu in Finland;
3. generalise methods of model design and data presentation:
4. determine directions for further development of modelling methods.

The relevance of COMMEMI to this thesis is that it supplied a series of seven standard models, which were made widely available in [83], [85] and [84]. The models vary from a simple conductor buried in a resistive host in Figure 5.6(1) to a very complex regional structure in Figure 5.6(5). The model in Figure 5.6(0) was added to the original six, after Weaver, LeQuang and Fischer[75][76] found its analytic solutions.

5.5 Comparing Grids

Poll's[48] code was altered to treat every surface grid point as a field measurement site, in order to force the calculation of apparent resistivity and phase all along the model. Choosing a limited number of static sites common to all the models would have left the possibility of missing variations away from these sites, especially at higher frequencies, where the shallow skin depth allows localised perturbations to the background field. Subroutines to read in grids from a file were also added. to enable testing of manually generated meshes.

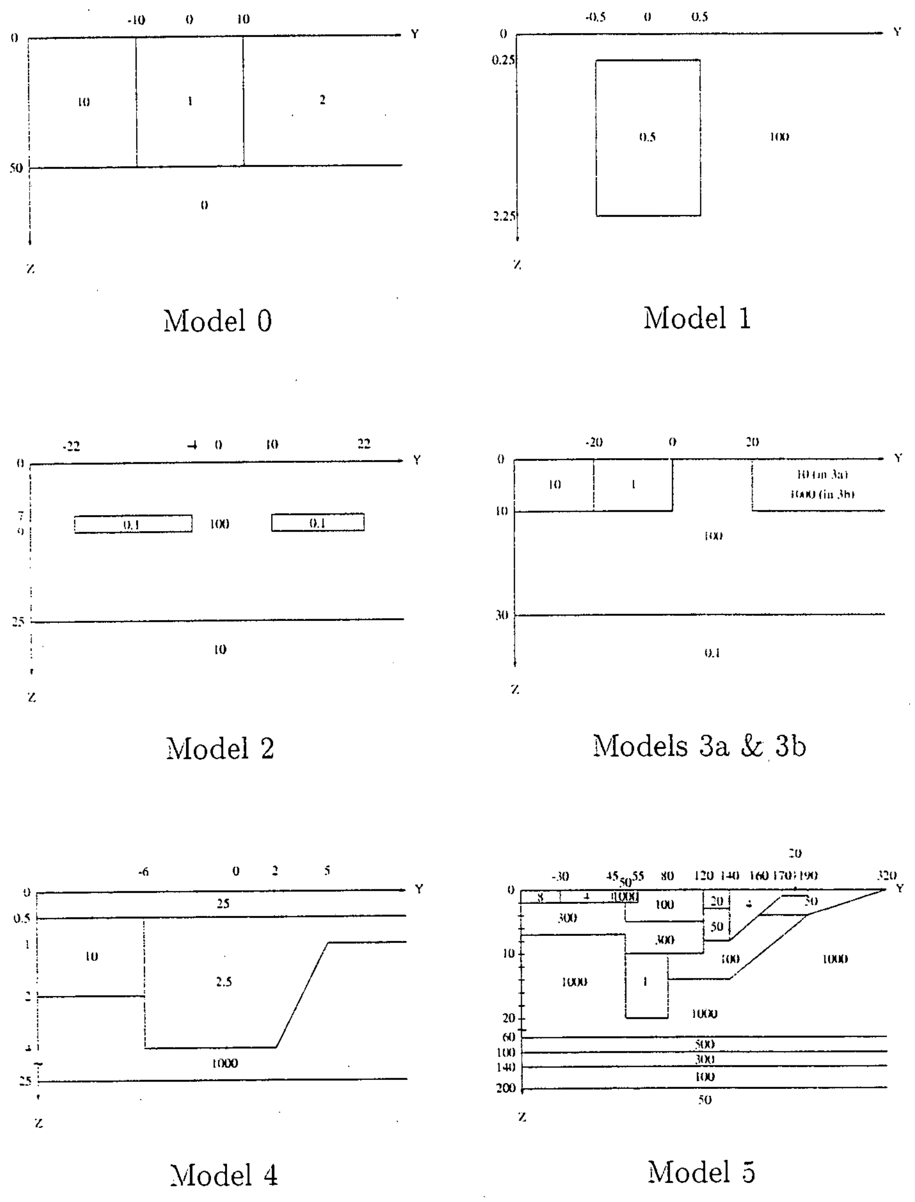

Figure 5.6. The seven standard COMMEMI models

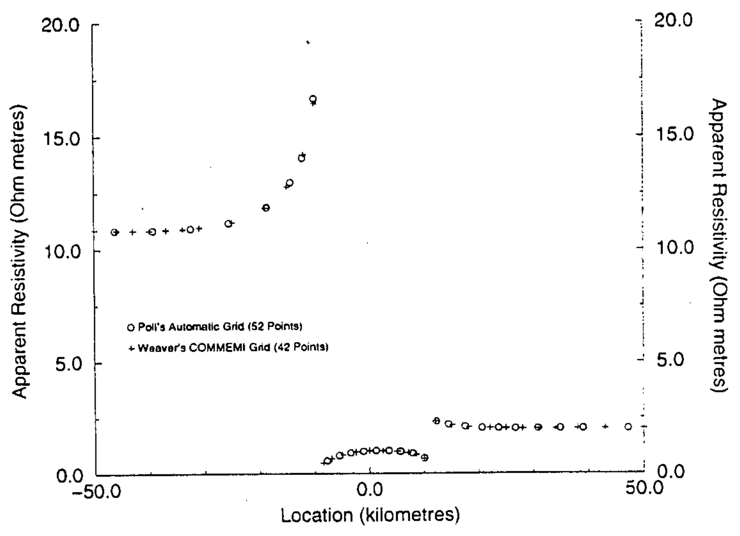

Figure 5.7. COMMEMI 2D-0 Poll and Weaver grids: Apparent Resistivity

Poll versus Weaver's COMMEMI Grid

Figures 5.7 and 5.8 show the B polarisation solutions for the first COMMEMI model (Figure 5.6(0)). Weaver's grid is taken from his published results[72]. The two series of points obviously fall on the same locus, for both the apparent resistivity and phase.

Poll versus Equivalent Elastic Grid

The stretched grid algorithm was given the same number of points as Poll's automatic gridder requested, and was instructed to distribute them over the same model dimensions as Weaver's COMMEMI grid.

Figures 5.9 and 5.10 show the B polarisation solutions for the third COMMEMI model (Figure 5.6(2)).

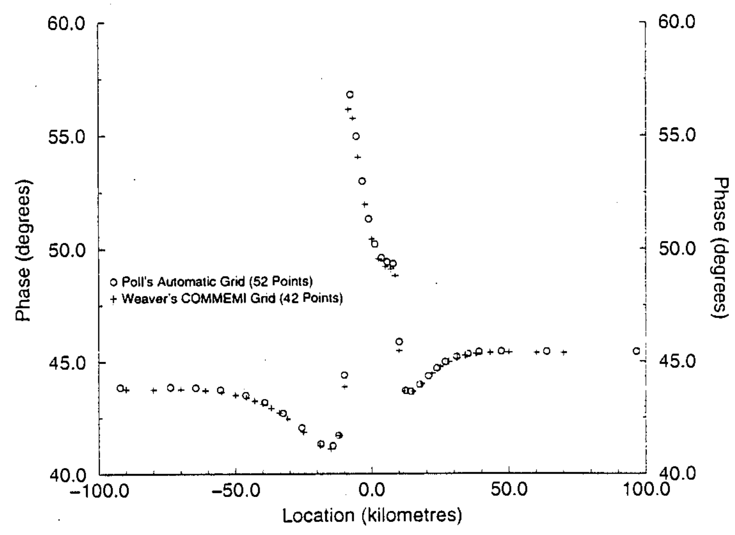

Figure 5.8. COMMEMI 2D-0 Poll and Weaver grids: Phase

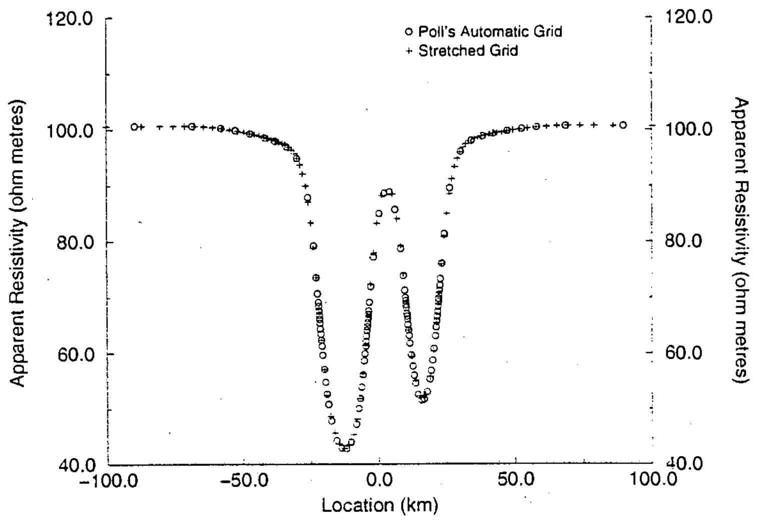

Figure 5.9. COMMEMI 2D-2 Poll and stretched grids: Apparent Resistivity

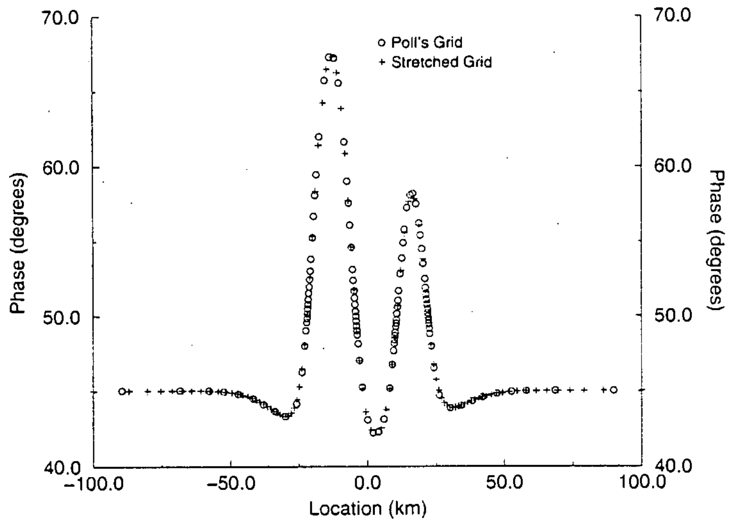

Figure 5.10. COMMEMI 2D-2 Poll and stretched grids: Phase

Half the Grid Points

Reducing the number of grid points by one half from 98 to 49 for the COMMEMI 2D-2 model, does not seem to have had an adverse effect on the response accuracy, as is shown in Figures 5.11 and 5.12. There were only two grid nodes where there was too rapid a change of spacing, as the output from the grid validation subroutine indicates...

```
Checking for variations in spacing...
  (Good between 50% and 200%)
*Too rapid change of spacing at  -82.4431122 (  44.6539268%)
*Too rapid change of spacing at   79.6015286 (  231.893173%)

Checking for symmetry around boundary points...
  (Good between 85% and 115%)
Grid Validation Summary:-
 Spacing:   95.9183655%
```

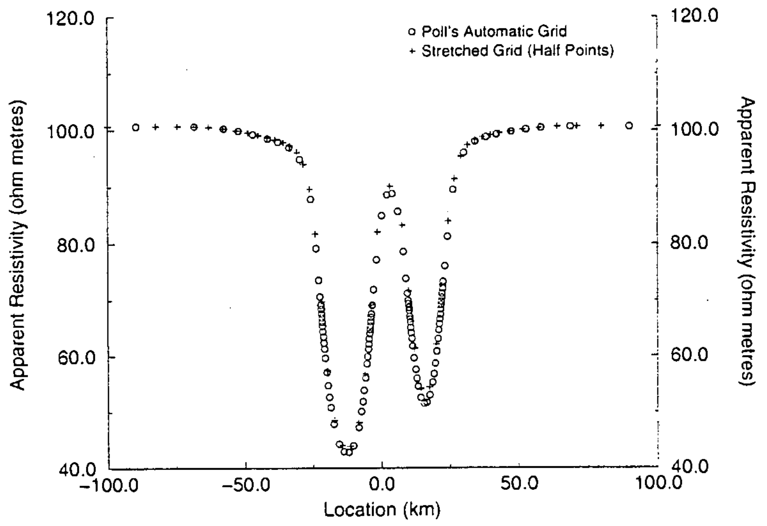

Figure 5.11. COMMEMI 2D-2 Poll and sparse stretched grids: Apparent Resistivity

```
Symmetry:    100.%
```

One Third of Grid Points

When the number of grid points was reduced to only one third suggested by Poll's automatic gridder, there were four grid locations found with unacceptably rapid changes in grid spacing...

```
Checking for variations in spacing...
  (Good between 50% and 200%)
*Too rapid change of spacing at  -78.4005369 (  42.6403961%)
*Too rapid change of spacing at  -1.06711347 (  277.345428%)
*Too rapid change of spacing at   7.06711341 (  36.0561218%)
*Too rapid change of spacing at   74.4641702 (  232.689255%)

Checking for symmetry around boundary points...
```

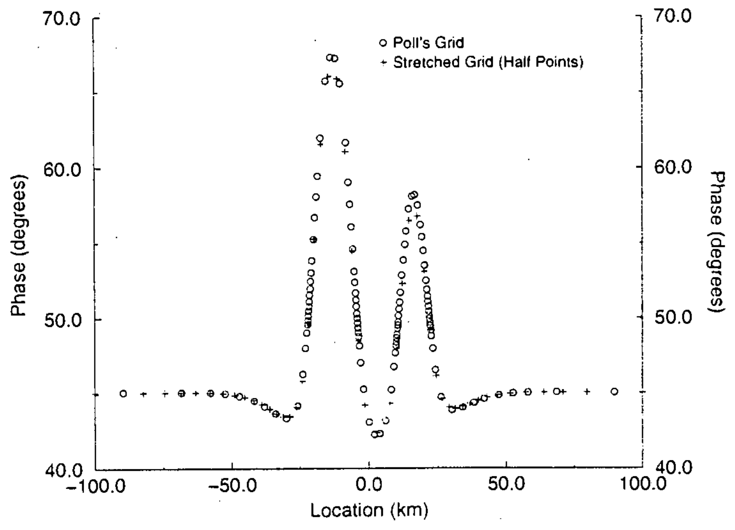

Figure 5.12. COMMEMI 2D-2 Poll and sparse stretched grids: Phase

```
  (Good between 85% and 115%)
Grid Validation Summary:-
 Spacing:    87.8787842%
 Symmetry:    100.%
```

Figures 5.13 and 5.14 show the apparent resistivity and phase for the stretched grid diverging from Poll's finer grid. However, the divergence is not very great, and the much smaller, and therefore quicker, model grid still gives an excellent estimate of the model response.

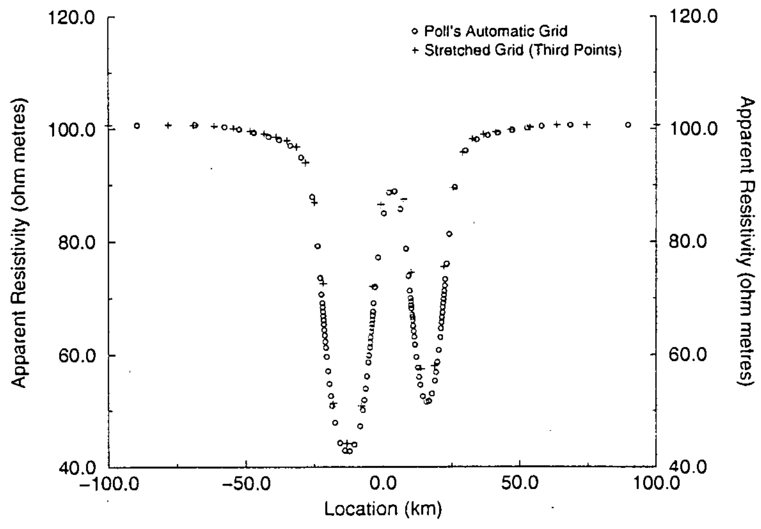

Figure 5.13. COMMEMI 2D-2 Poll and very sparse stretched grids: Apparent Resistivity

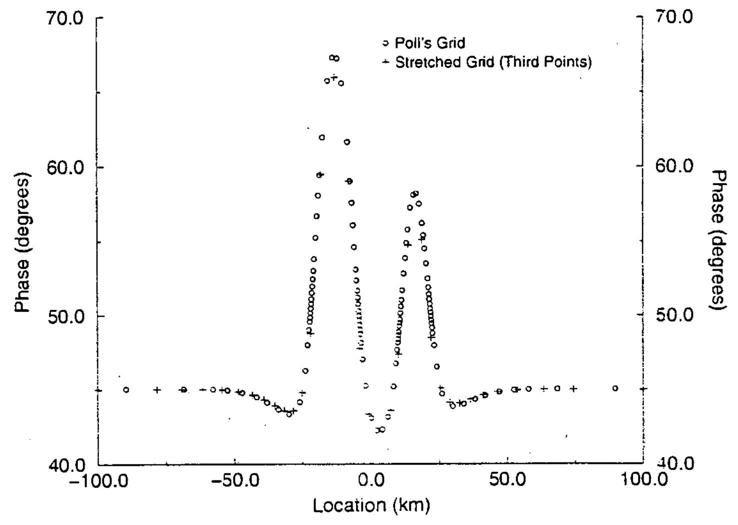

Figure 5.14. COMMEMI 2D-2 Poll and very sparse stretched grids: Phase

Chapter 6

Conclusions and Further Work

Both the two and three dimensional model algorithms were successfully adapted for the massively parallel SIMD computers, having rewritten their solver cores with fine grained parallel iterative methods. However, their relatively poor performance, combined with the fact that these machines operate batch queue systems. make them impractical for routine use.

Investigations into two dimensional model solution behaviour, utilising a new automatic mesh generation algorithm, have shown that model accuracy can be maintained with coarser grids in many cases. It has also been confirmed that there is no gain in precision in constructing finer grids than are commonly in use today. The elastic mesh grid generator could prove useful for prototyping models rapidly, such as may be required in the initial stages of some two dimensional inversion packages. The inversion routine could choose the number of grid points, allowing more, and hence slower iterations, as the model converges to its final state.

It seems that the task farm approach is ideal for computing large numbers of two dimensional models, as modern workstations can solve a single frequency in a short time, at most a few minutes. It is also much more likely that an academic institution will have access to a suitable workstation cluster, rather than a state of the art parallel supercomputer. Indeed, a laboratory of tens of modern PC class Intel Pentium based running appropriate software would make a valuable computing resource when dedicated to out of hours task farms.

If anything, the rapid pace in the increase of computing power has accelerated recently with the development of machines such as the Cray T3D. This architecture is based on large numbers of DEC Alpha processors sharing enormous amounts of very fast core memory. Such a machine with 512 processors and 32 gigabytes of core memory is now installed at the Edinburgh Parallel Computing Centre.

This scale of machine is at last able to model three dimensional structures at the same kind of detail as is routine today in two dimensional analysis and interpretation. It is sobering to note that each processing element in a T3D is approximately equivalent to the AXP described in Chapter 2 which performed so well for the three dimensional code (see Table 4.4 on page 73). However, a High Performance FORTRANcompiler only became available on the T3D in the last few months, unfortunately too late to port the code in this thesis.

However, Wilson has implemented a task farm version of Newman and Hohmann's[44] three dimensional integral equation model, and Yu and Edwards'[82] finite difference simulation for axisymmetric models. He has recently reported some impressive performance results on the T3D[80]. Wilson*et al.*[80] has raised the possibility of changing Yu and Edwards' code to implement a parallel solver, as well as operate in a task farm. I am certainly of the opinion that both fine and coarse grained parallelisation are valid and should be pursued for three dimensional models. Schultz and Smith at the University of Cambridge have developed a staggered grid three dimensional model with an accelerated iterative solver. This model would, along with Pu's, make an excellent candidate for further development on the T3D.

Three dimensional model performance can be greatly improved with the application of faster solver techniques. Mackie *et al.*[37] propagate an impedance matrix through their three dimensional model to solve the problem. This results in multiple small matrix inversions rather than one huge inversion. Freund[19][20] has developed some promising looking Krylov Subspace methods for complex non-Hermitian linear system which may be applicable to this problem.

Bibliography

[1] A K Agarwal and J T Weaver. Inversion of the COPROD2 data by a method of modeling. *Journal of Geomagnetism and Geoelectricity*, 45(9):969–983, 1993.

[2] R C Bailey. Inversion of the geomagnetic induction problem. *Proceedings of the Royal Society of London*, 315(A):185–194, 1970.

[3] J R Booker and A D Chave. Introduction to the special section on the EMSLAB Juan de Fuca experiment. *Journal of Geophysical Research*, 94:14093–14098, 1989.

[4] C R Brewitt-Taylor and J T Weaver. On the finite difference solution of two-dimensional induction problems. *Geophysical Journal of the Royal Astronomical Society*, 47:375–396, 1976.

[5] C R Brewitt-Taylor and J T Weaver. *A Computer Program for the Solution of Electromagnetic Induction Problems.* Dept. of Physics, University of Victoria, revised edition, February 1976-86.

[6] R A A Bruce, S Chapple, N B MacDonald, A S Trew, and S Trewin. CHIMP and PUL: Support for portable parallel computing. *Journal of Future Generation Computer Systems*, 1995.

[7] L Cagniard. Basic theory of the magnetotelluric method. *Geophysics*, 18:605–635, 1953.

[8] J Chen, H W Dosso, and M Ingham. Electromagnetic induction in New Zealand: Analogue model and field results. *Physics of the Earth and Planetary Interiors*, 62:257–270, 1990.

[9] J Chen, H W Dosso, and W Nienaber. Laboratory electromagnetic model results for the EMSLAB region. *Journal of Geophysical Research*, 84:14167–14172, 1989.

[10] J H Coggon. Electromagnetic and electrical modelling by the finite element method. *Geophysics*, 36(1):132–155, February 1971.

[11] T W Dawson, J T Weaver, and U Raval. B-polarization induction in two generalized thin sheets at the surface of a conducting half space. *Geophysical Journal of the Royal Astronomical Society*, 69:209–234, 1982.

[12] C deGroot Hedlin and S Constable. Occam's inversion to generate smooth, two-dimensional models from magnetotelluric data. *Geophysics*, 55(12):1613–1624, December 1990.

[13] I d'Erceville and G Kunetz. The effect of a fault on the Earth's natural electromagnetic field. *Geophysics*, 27:651–665, 1962.

[14] H W Dosso. A review of analogue model studies of the coast effect. *Physics of the Earth and Planetary Interiors*, 7:294–302, 1973.

[15] Edinburgh Parallel Computing Centre. *PUL-TF User Guide.*

[16] EMSLAB Group. The EMSLAB electromagnetic sounding experiment. *EOS Transactions of the American Geophysical Union*, 69:98–99, 1988.

[17] G E Forsythe. *Numerical Analysis and Partial Differential Equations.* Surveys in Applied Maths. John Wiley & Sons, 1958.

[18] G E Forsythe and W R Wasow. *Finite-Difference Methods for Partial Differential Equations.* Applied Maths Series. John Wiley & Sons, 1960.

[19] R W Freund. Krylov subspace methods for complex non-Hermitian linear systems. Technical Report 91.11, Research Institute for Advanced Computer Science (RIACS), Mail Stop Ellis Street, NASA Ames Research Center, Moffett Field, CA 94035 USA, May 1991.

[20] R W Freund. Transpose-free quasi-minimal residual methods for non-Hermitian linear systems. Numerical Analysis Manuscript 92-07, AT&T Bell Laboratories, Room 2C-420, 600 Mountain Avenue, Murray Hill, NJ 07974-0636, July 1992.

[21] D Galanopoulos, V R S Hutton, and G J K Dawes. The Milos geothermal-field - modeling and interpretation of electromagnetic induction studies. *Physics of the Earth and Planetary Interiors*, 66(1-2):76–91, 1991.

[22] Geotools Corporation, 5808 Balcones Drive Suite 202, Austin TX 7873I, USA. *Geotools: A complete system for magnetotelluric interpretation*, 1993.

[23] V R Green and J T Weaver. Two-dimensional induction in a thin sheet of variable integrated conductivity at the surface of a uniformly conducting Earth. *Geophysical Journal of the Royal Astronomical Society*, 55:721–736, 1978.

[24] B A Hobbs. Terminology and symbols for use in studies of electromagnetic induction in the Earth. *Surveys in Geopyhsics*, 13:489–515, 1992.

[25] G W Hohmann. Three-dimensional induced polarization and electromagnetic modeling. *Geophysics*, 40:309–324, 1975.

[26] W B Hu, , W Nienaber, and H W Dosso. Vertical magnetic field response of a seamount. *Physics of the Earth and Planetary Interiors*, 54:135–139, 1989.

[27] G R Jiracek, R P Reddig, and R K Kojima. Application of the Rayleigh-FFT technique to magnetotelluric modeling and correction. *Physics of the Earth and Planetary Interiors*, 53:365–375, 1989.

[28] F W Jones, A Correia, G K Dawes, V R S Hutton, P Jones, and K J MacDonald. Preliminary results of a magneto-telluric survey over a geothermal anomaly in Portugal. *Physics of the Earth and Planetary Interiors*, 73:274–281, 1992.

[29] F W Jones and A T Price. The perturbations of alternating geomagnetic fields by conductivity anomalies. *Geophysical Journal of the Royal Astronomical Society*, 20:317–334, 1970.

[30] F W Jones and K Vozoff. The calculation of magnetotelluric quantities for three-dimensional conductivity inhomogeneities. *Geophysics*, 43:1167–1175, 1978.

[31] P Kaikkonen. Numerical electromagnetic modeling including studies of characteristic dimensions: A review. *Surveys in Geophysics*, 8:301–337, 1986.

[32] E Kreyszig. *Advanced Engineering Mathematics.* Wiley, sixth edition, 1988.

[33] C C Ku, M S Hsieh, and S H Lim. The topographic effect in electromagnetic fields. *Canadian Journal of Earth Science*, 10:645–656, 1973.

[34] H T Kung. Computational models for parallel computers. *Philosophical Transactions of the Royal Society of London*, A 326:357–371, 1988.

[35] E Lagios, A Tzanis, N Delibasis, J Drakopoulos, and G K J Dawes. Geothermal-exploration of Kos-island, Greece - magnetotelluric and microseismicity studies. *Geothermics*, 23(3):267–281, 1994.

[36] S Levy, D Oldenburg, and J Wang. Subsurface imaging using magnetotelluric data. *Geophysics*, 53(1):1040117, 1988.

[37] R L Mackie, T R Madden, and P E Wannamaker. Three-dimensional magnetotelluric modeling using difference equations - Theory and comparisons to integral equation solutions. *Geophysics*, 58(2):215–226, February 1993.

[38] T R Madden. *EMCAL*, 1971.

[39] D McA McKirdy and J T Weaver. Induction in a thin sheet of variable conductance at the surface of a stratified earth - I. Two-dimensional theory. *Geophysical Journal of the Royal Astronomical Society*, 78(1):93–103, 1984.

[40] D McA McKirdy, J T Weaver, and T W Dawson. Induction in a thin sheet of variable conductance at the surface of a stratified earth-II. Three-dimensional theory. *Geophysical Journal of the Royal Astronomical Society*, 80(1):177–194, 1985.

[41] J J Modi. *Parallel Algorithms and Matrix Computation.* Oxford Applied Mathematics and Computing Science Series. Clarendon Press, 1988.

[42] MPI Consortium. Message Passing Interface specification/description.

[43] A S Neves. *The Magneto-Telluric Method in Two-Dimensional Structures.* PhD thesis, MIT, Department of Geology & Geophysics, 1957.

[44] G A Newman, G W Hohmann, and W L Anderson. Transient electromagnetic response of a three-dimensional body in a layered earth. *Geophysics*, 51:1608–1627, 1986.

[45] J M Ortega and R G Voigt. Solution of partial differential equations on vector and parallel computers. *SIAM Review*, 1985.

[46] S K Park, A S Orange, and T R Madden. Effects of three-dimensional structure on magnetotelluric sounding curves. *Geophysics*, 48:1402–1405, 1983.

[47] W D Parkinson. *Introduction to Geomagnetism.* Scottish Academic Press, 1983.

[48] Helena E Poll. *Automatic Forward Modelling of Two-Dimensional Problems in Electromagnetic Induction.* PhD thesis, University of Victoria, B.C., 1994.

[49] A T Price. The induction of electric currents in non-uniform thin sheets and shells. *Quarterly Journal of Mechanics and Applied Maths*, 2:283–310, 1949.

[50] A T Price. Electromagnetic induction in a semi-infinte conductor with a plane boundary. *Quarterly Journal of Mechanics and Applied Maths*, 3:385–410, 1950.

[51] Xinghua Pu. *Three Dimensional Geomagnetic Forward Modelling.* PhD thesis, University of Victoria, 1994.

[52] A P Raiche. An integral equation approach to three-dimensional modelling. *Geophysical Journal of the Royal Astronomical Society*, 36:363–376, 1974.

[53] R P Ranganayaki and T R Madden. Generalized thin sheet analysis in magnetotellurics: an extension of price's analysis. *Geophysical Journal of the Royal Astronomical Society*, 60:445–457, 1980.

[54] D Rankin. The magnetotelluric effect of a dike. *Geophysics*, 27:666–676, 1962.

[55] I K Reddy and D Rankin. Magnetotelluric response of a two-dimensional sloping contact by the finite element method. *Pure and Applied Geophysics*, 105:847–857, 1973.

[56] U Schmucker. Interpretation of induction anomolies above non-uniform surface layers. *Geophysics*, 36:156–165, 1971.

[57] A Schuster. The diurnal variation of terrestrial magnetism. *Philosophical Transactions of the Royal Society of London*, 1889.

[58] G D Smith. *Numerical Solution of Partial Differential Equations: Finite Difference Methods.* Oxford applied mathematics and computing science series. Clarendon Press, third edition, 1985.

[59] J T Smith and J R Booker. Rapid inversion of 2-dimensional and 3-dimensional magnetotelluric data. *Journal of Geophysical Research*, 96:3905–3922, 1991.

[60] D M Summers and J T Weaver. Electromagnetic induction in a stratified conducting half space by an arbritary periodic source. *Canadian Journal of Physics*, 51:1064–1074, 1973.

[61] W M Telford, L P Geldart, and R E Sheriff. *Applied Geophysics.* Cambridge University Press, second edition, 1990.

[62] Thinking Machines Corporation, Cambridge, Massachusetts, USA. *Connection Machine CM-200 Series Technical Summary*, first edition, June 1991.

[63] A N Tikhonov. Dtermination of the electrical characteristics of the deep strata of the Earth's crust. *Dokl. Akad. Nauk SSSR*, 73:295–297, 1950.

[64] R S Varga. *Matrix iterative analysis.* Pretence Hall, 1962.

[65] G Vasseur and P Weidelt. Bimodal electromagnetic induction in non-uniform thin sheets with an application to the northern Pyrenean induction anomaly. *Geophysical Journal of the Royal Astronomical Society*, 51:669–690, 1977.

[66] J R Wait. Propagation of radio waves over a stratified ground. *Geophysics*, 18:416–422, 1953.

[67] D J Wallace. Scientific compuation on SIMD and MIMD machines. *Philosophical Transactions of the Royal Society of London*, A 326:481–498, 1988.

[68] P E Wannamaker, J R Booker, J H Filloux, A G Jones, G R Jiracek, A D Chave, P Tarits, H S Waff, G D Egbert, C T Young, J A Stodt, M Martinez, L K Law, T Yukutake, J S Segawa, A White, and A W Green. Magnetotelluric observations across the Defuca, Juan subduction system in the EMSLAB project. *Journal of Geophysical Research*, 94:14111–14125, 1989.

[69] Philip E Wannamaker, John A Stodt, and Luis Rijo. *PW2D Finite element prgram for solution of magnetotelluric responses of two-dimensional earth resistivity structure.* Earth Science Laboratory, University of Utah Research Institute, 391 Chipeta Way, Suite C, Salt Lake City, Utah 84108, USA, revised edition, June 1987.

[70] Philip E Wannamaker, John A Stodt, and Luis Rijo. A stable finite element solution for two-dimensional magnetotelluric modelling. *Geophysical Journal of the Royal Astronomical Society*, 88:277–296, 1987.

[71] J T Weaver. The electromagnetic field within a discontinuous conductor with reference to geomagnetic micropulsations near a coastline. *Canadian Journal of Physics*, 41:484–495, 1963.

[72] J T Weaver. Finite difference calculations for COMMEMI. Technical report, Department of Physics, University of Victoria, British Columbia, Canada, May 1984.

[73] J T Weaver. *Mathematical methods for geo-electromagnetic induction.* Applied and engineering mathematical series. Research Studies Press Ltd, 1994.

[74] J T Weaver and C R Brewitt-Taylor. Improved boundary conditions for the numerical solution of E-polarization problems in geomagnetic induction. *Geophysical Journal of the Royal Astronomical Society*, 54:309–317, 1978.

[75] J T Weaver, B V LeQuang, and G Fischer. A comparison of analytical and numerical results for a 2-D control model in electromagnetic induction - I. B-polarisation calculations. *Geophysical Journal of the Royal Astronomical Society*, 82:263–277, 1985.

[76] J T Weaver, B V LeQuang, and G Fischer. A comparison of analytical and numerical results for a 2-D control model in electromagnetic induction - II. E-polarisation calculations. *Geophysical Journal of the Royal Astronomical Society*, 87:917–948, 1986.

[77] P Weidelt. The inverse problem of geomagnetic induction. *Zeitschrift für Geophysik*, 38:257–289, 1972.

[78] P Weidelt. Electromagnetic induction in three-dimensional structures. *Journal of Geophysics*, 41:85–109, 1975.

[79] K M Westaway. *The endless quest - Three thousand years of science.* Blackie and Son Ltd, 1934.

[80] Andrew J S Wilson, Kenneth J MacDonald, Liming Yu, Bill Day, and Hamish Mills. *Electromagnetic Modelling on Parallel Computers.* Schlumberger-Doll Research, 1996. In Print.

[81] David M Young. *Iterative solution of large linear systems.* Computer Science and Applied Mathematics. Academic Press, 1971.

[82] L Yu. Computation of the electrical responses of mid-ocean ridge structures. Technical Report 5, University of Toronto, 1994.

[83] M S Zhdanov and Iv M Varentsov. Progress report on the COMMEMI project. page 438. 7th IAGA Workshop on Electromagnetic Induction in the Earth and Moon, 1984. August 15-22, Ile-Ife, Nigeria.

[84] M S Zhdanov and Iv M Varentsov. Progress report II on the COMMEMI project. In *IAGA/IAMAP Abstracts*, volume 2, page 438. 5th General Assembly of IAGA, 1985. August 5-17, Prague, Czechoslovakia.

[85] M S Zhdanov and Iv M Varentsov. Revised program of COMMEMI project. Technical report, Institue of Terrestial Magnetism, Ionosphere and Radio Wave Propogation, USSR Academy of Sciences, 1985.

Appendix A

Two Dimensional Code Excerpts

A.1 Matrix-Iter.FCM

```
#include "Arch.h"

C Original subroutine written by Helena Poll 1988.
c (This subroutine, called by FR2D, directs the matrix
c elimination procedure and prints out the Fields.POLL'88)

C Converted to Iterative version by Kenneth MacDonald October 1993
C Tidied up by Kenneth MacDonald February 1994
C Added support for CM Fortran, and cpp, Kenneth MacDonald March 1994

        SUBROUTINE MATRIX(YGrid, ZGrid, ResGrid, Field, NPol, MD)

C This subroutine calculates the coefficients for each grid point and
C stores them in arrays, ready for the iterative solver. The solution is
C then written to the output file.

C Set up the names of the subroutines to call, depending on Architecture

#ifdef CM200_Arch
#define FIELD_to_PARALLEL FEField2CM
#define CELL_to_PARALLEL FECell2CM
#define SURF_to_PARALLEL FESurf2CM
#define PARALLEL_to_FIELD CMField2FE
#endif
#ifdef MasPar_Arch
#define FIELD_to_PARALLEL FEField2DPU
#define CELL_to_PARALLEL FECell2DPU
#define SURF_to_PARALLEL FESurf2DPU
#endif

C No implicit variable names allowed

        IMPLICIT NONE

C Include files

        INCLUDE 'limits.inc'

C Common Blocks

        INTEGER         NY, NZ
        COMMON          /LIMITS/ NY, NZ
        REAL*8          Pi, Omega, RMu, PRCond, PRInsul
        COMPLEX*16      RSlope
        COMMON          /CONST/ Pi, Omega, rmu, PRCond, PRInsul, RSlope
        INTEGER         NSM, Kstart, KFlag, KEnd, KFirst
        COMMON          /BAND/ NSM, KStart, KFlag, KEnd, KFirst
        INTEGER         NQ
        COMMON          /SV1/NQ
        INTEGER         MaxIter
        COMMON          /Iterative/ MaxIter

C Arguments

        INTEGER         MD
        REAL*8          YGrid(MD)
        REAL*8          ZGrid(MD)
        REAL*8          ResGrid(MD, MD)
        COMPLEX*16      Field(MD, MD)
        INTEGER         NPol

C Local FE Variables

        INTEGER         I, J
        COMPLEX*16      LeftElement
        COMPLEX*16      CentreBlock(MaxGrid)
        COMPLEX*16      RightBlock(MaxGrid)
        COMPLEX*16      RHS
        COMPLEX*16      CellMatrix(MaxGrid, MaxGrid, 6)
        COMPLEX*16      SurfMatrix(MaxGrid, MaxGrid, 2)
        REAL*8          Error

C Local parallel variables

#ifdef Parallel_Arch
        COMPLEX*16      PCellMatrix(NY, NZ, 6)
        COMPLEX*16      PSurfMatrix(NY+2, NY, 2)
        COMPLEX*16      PField(NY, NZ)  ! Parallel copy of the field
#endif

C Compiler Directives

#ifdef CM200_Arch
CMF$    COMMON FEONLY /SV1/
CMF$    COMMON FEONLY /Limits/
CMF$    COMMON FEONLY /Const/
CMF$    COMMON FEONLY /Band/
CMF$    LAYOUT YGrid(:SERIAL)
CMF$    LAYOUT ZGrid(:SERIAL)
CMF$    LAYOUT ResGrid(:SERIAL, :SERIAL)
CMF$    LAYOUT Field(:SERIAL, :SERIAL)
CMF$    LAYOUT CentreBlock(:SERIAL)
CMF$    LAYOUT RightBlock(:SERIAL)
CMF$    LAYOUT CellMatrix(:SERIAL, :SERIAL, :SERIAL)
CMF$    LAYOUT SurfMatrix(:SERIAL, :SERIAL, :SERIAL)
CMF$    LAYOUT PCellMatrix(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT PSurfMatrix(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT PField(:NEWS, :NEWS)
#endif

C Initialise Variables

        MaxIter = 1000
        Error = 1.0D-5
        RSlope = (Field(NY, NZ) - Field(1, NZ))/(YGrid(NY) - YGrid(1))
        KFirst = 1
        KEnd = 0
        KFlag = 0
        KStart = 1
        NSM = NY - 2
        NQ = 0

C Top BC First, but only for E Polarisation

        IF (NPol.EQ.1) THEN
          J = 1
          DO I = 2, NY - 1
            CALL CPNTA(YGrid, ZGrid, ResGrid, CentreBlock, RHS,
     &        RightBlock, LeftElement, Field, NPol, J, I, MD, MaxGrid)
            CALL SURF_STORE (SurfMatrix, MaxGrid, LeftElement,
     &        CentreBlock, RightBlock(I-1), RHS, I, J)
          END DO
        END IF

C General Block

        DO J = 2, NZ - 1
          DO I = 2, NY - 1
            CALL CPNTA(YGrid, ZGrid, ResGrid, CentreBlock, RHS,
     &        RightBlock, LeftElement, Field, NPol, J, I, MD, MaxGrid)
            CALL CENTRE_STORE (CellMatrix, MaxGrid, LeftElement,
     &        CentreBlock, RightBlock(I-1), RHS, I, J)
          END DO
        END DO

C Bottom Block

        KEND = 1
        J = NZ
        DO I = 2, NY - 1
          CALL CPNTA(YGrid, ZGrid, ResGrid, CentreBlock, RHS,
     &      RightBlock, LeftElement, Field, NPol, J, I, MD, MaxGrid)
          CALL SURF_STORE (SurfMatrix, MaxGrid, LeftElement,
     &      CentreBlock, RightBlock(I-1), RHS, I, J)
        END DO

C Start the timer

#ifdef MasPar_Arch
        CALL mpTimerStart ()     ! Start clock
#endif /* MasPar_Arch */

C Now Solve Iteratively
C First the code to solve on a serial machine. Simply call ITER_SOLVE

#ifdef Serial_Arch
        CALL ITER_SOLVE (Field, CellMatrix, SurfMatrix, MD, MaxGrid,
     &   MaxGrid, MaxIter, Error, NPol)
#endif

C Now the code to solve on a parallel machine.  First of all, we have to
C copy all the arrays across to the DPU, then call ITER_SOLVE, and finally
C copy the field back again.

#ifdef Parallel_Arch

C Copy the arrays to a parallel array

        CALL FIELD_to_PARALLEL (Field, PField, MD, NY, NZ)        ! Field
        CALL CELL_to_PARALLEL (CellMatrix, PCellMatrix, MaxGrid,
     &   NY, NZ)                                                  ! Cells
        CALL SURF_to_PARALLEL (SurfMatrix, PSurfMatrix, MaxGrid, NY)! Surfaces

C Now call the solver with the parallel arrays
```

```
        CALL ITER_SOLVE (PField, PCellMatrix, PSurfMatrix, NY, NZ,
     &           MaxIter, Error, NPol)

C Copy the parallel field back to the serial version

        CALL PARALLEL_to_FIELD (Field, PField, MD, NY, NZ)

#endif

C Stop timer and print time taken

#ifdef MasPar_Arch
        Time = mpTimerElapsed ()
        PRINT *, "Time Elapsed (ms): ", Time
#endif /* MasPar_Arch */

C Print Field in Formatted output

        CALL WRITE(YGrid, ZGrid, Field, MD, NPol)
        CALL DUMP_FIELD (YGrid, ZGrid, Field, MD, NPol, 'raw')

C Finished

        RETURN
        END
```

A.2 Store-Iter.FCM

```
C **************************************************************

C This subroutine stores the five coefficients plus the right hand side
C constant from the finite difference equations in an array. This is only
C possible for the central points, as the top and bottom need many more
C coefficients (due to the integral boundary conditions).

      SUBROUTINE CENTRE_STORE (Matrix, Size, LeftElement, CentreBlock,
     &                        RightElement, RHS, IY, IZ)

      IMPLICIT NONE

C Common blocks

      COMMON /LIMITS/NY,NZ
      INTEGER NY, NZ

C Arguments

      INTEGER         Size                   ! The size of the matrix
      COMPLEX*16      Matrix(Size, Size, 6)  ! The matrix to be filled
      COMPLEX*16      LeftElement            ! Single diagonal element in the left block
      COMPLEX*16      CentreBlock(NY)        ! Vector of elements in the centre block
      COMPLEX*16      RightElement           ! Element in the right block
      COMPLEX*16      RHS                    ! The Right hand side value
      INTEGER         IY, IZ                 ! The current grid coordinates of the point

C Local Variables

      INTEGER         Counter                ! Simple general purpose counter

C The SELF coefficient

      Matrix(IY, IZ, 1) = CentreBlock(IY - 1)

C The LEFT coefficient

      IF (IY.NE.1) THEN
              Matrix(IY, IZ, 2) = CentreBlock(IY - 2)
      END IF

C The RIGHT coefficient

      IF (IY.NE.NY) THEN
              Matrix(IY, IZ, 3) = CentreBlock(IY)
      END IF

C The UP coefficient

      IF (IZ.NE.1) THEN
              Matrix(IY, IZ, 4) = LeftElement
      END IF

C The DOWN coefficient

      IF (IZ.NE.NZ) THEN
              Matrix(IY, IZ, 5) = RightElement
      END IF

C The RHS Constant

      Matrix(IY, IZ, 6) = RHS

      RETURN
      END
```

```
C ****************************************************************

C This subroutine stores the integral equation finite difference coefficients
C in a martrix for use in the iterative solver. Called for each point on the
C bottom of the central grid, and for the top in E-Polarisation.
C Must be called for only top or bottom points. Gives Error message if it
C is called from a central point.

      SUBROUTINE SURF_STORE (SurfCoef, Size, LeftElement, CentreBlock,
     &                       RightElement, RHS, IY, IZ)

      IMPLICIT NONE

C Common Blocks

      INTEGER NY, NZ
      COMMON /LIMITS/NY, NZ

C Arguments

      INTEGER        Size                    ! The max size of the arrays
      COMPLEX*16     SurfCoef(Size, Size, 2) ! The coeff matrix to fill
      COMPLEX*16     LeftElement             ! The left off diag element
      COMPLEX*16     CentreBlock(Size)       ! The centre elements
      COMPLEX*16     RightElement            ! The right off diag element
      COMPLEX*16     RHS                     ! The RHS constant
      INTEGER        IY, IZ                  ! The grid coordinates

C Local Variables

      INTEGER        WhichSurface            ! Top or bottom?
      INTEGER        Counter                 ! General counter

C Compiler Directives

CMF$  LAYOUT SurfCoef(:NEWS, :NEWS, :SERIAL)

C Check matrix is big enough. Must be 2*NY*(NY+2)

      IF (Size.LT.(NY + 2)) THEN
        PRINT *, "Error: SURF_STORE called with too small a matrix."
        RETURN                 ! Don't continue, just return.
      END IF

C Are we at the Top or Bottom Surface ? Set the Up or Down coeffs and
C set the index 'WhichSurface' to 1 or 2.

      IF (IZ.EQ.1) THEN
        SurfCoef(NY + 1, IY, 1) = RightElement      ! Down coeff
        WhichSurface = 1                            ! Top
      ELSE IF (IZ.EQ.NZ) THEN
        SurfCoef(NY + 1, IY, 2) = LeftElement       ! Up coeff
        WhichSurface = 2                            ! Bottom
      ELSE
        PRINT *, 'Error: SURF_STORE called from central gridpoint.'
        RETURN                          ! Error, shouldn't be in here.
      END IF

C Now fill in the line in the array

      DO Counter = 2, NY - 1
        SurfCoef(Counter, IY, WhichSurface) = CentreBlock(Counter - 1)
      END DO

C Now store the RHS constant value in the end of the line
```

```
      SurfCoef(NY + 2, IY, WhichSurface) = RHS

C Done

      RETURN
      END
```

A.3 Jacobian.FCM

```
C Jacobian (v.07) MasPar High Performance Fortran
C Kenny MacDonald 18th November 1993
C This version of the Jacobian solver calculates the maximum
C absolute residuals and stops iterating according to that.

      SUBROUTINE ITER_SOLVE (Field, Matrices, Surface,
    &          NY, NZ, MaxIter, Error, NPol)

      IMPLICIT NONE                          ! Important this

C Arguments

      INTEGER         NY, NZ  ! The model grid size
      COMPLEX*16      Field(NY, NZ)
      COMPLEX*16      Matrices(NY, NZ, 6)
      COMPLEX*16      Surface(NY+2, NY, 2)
      INTEGER         MaxIter                ! Max number of iterations
      REAL*8          Error                  ! Required Error
      INTEGER         NPol                   ! Polarisation of the problem

C Local Variables

      INTEGER         I, J                   ! Coordinates of a grid point
      INTEGER         Direction              ! Current direction
      INTEGER         Iteration              ! Iteration counter
      COMPLEX*16      NewField(NY, NZ)       ! Updated field vaules
      COMPLEX*16      Value(NY)              ! Temp values for surface calcs
      COMPLEX*16      Products(NY, NY)       ! Surface temp values
      COMPLEX*16      Change(NY, NZ)         ! Errors at each iteration
      REAL*8          Residual(NY, NZ)       ! Absolute residual each point
      REAL*8          MaxResidual            ! The maximum absolute residual
      INTEGER         Top                    ! The top layer of the problem
      LOGICAL         ActiveNodes(NY, NZ)    ! Active nodes are .TRUE.
      LOGICAL         OffDiag(NY, NY)        ! Off diagonal surface elements
      LOGICAL         Central(NY, NZ)        ! The central grid points

C Compiler Directives

#ifdef MasPar_Arch
cmpf ONDPU      Field, NewField, Surface, Matrices, Value, Products
cmpf ONDPU      Residual, ActiveNodes, OffDiag, Central, Change
#endif

#ifdef CM200_Arch
CMF$    LAYOUT Field(:NEWS, :NEWS)
CMF$    LAYOUT Matrices(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT Surface(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT NewField(:NEWS, :NEWS)
CMF$    LAYOUT Value(:NEWS)
CMF$    LAYOUT Products(:NEWS, :NEWS)
CMF$    LAYOUT Change(:NEWS, :NEWS)
CMF$    LAYOUT ActiveNodes(:NEWS, :NEWS)
CMF$    LAYOUT OffDiag(:NEWS, :NEWS)
CMF$    LAYOUT Central(:NEWS, :NEWS)
#endif

C Display required tolerances

      PRINT *, 'Required maximum residual ', Error
#ifdef MasPar_Arch
      PRINT *, 'MasPar High Performance Fortran Version'
#endif

#ifdef CM200_Arch
      PRINT *, 'Connection Machine Fortran Version'
```

```
#endif
      PRINT *, 'Using Jacobian (v8) solver, absolute residuals'

C Set the upper most layer of interest

      IF (NPol.EQ.0) THEN
        Top = 2                 ! BPol, ignore surface
      ELSE
        Top = 1                 ! EPol, need surface
      END IF

C Set up Active Nodes according to Polarisation

      ActiveNodes = .FALSE.
      ActiveNodes(2:NY-1, Top:NZ) = .TRUE.

C Loop over the calculations

      DO Iteration = 1, MaxIter

C First Calculate the Top Surface iteration if EPol

        IF (NPol.EQ.1) THEN
          OffDiag = .FALSE.                         ! Initialise to FALSE
          Value(2:NY-1) = Surface(NY + 2, 2:NY-1, 1)  ! Set to RHS values
          FORALL (I = 1:NY, J = 1:NY)
     &      OffDiag(I, J) = I.NE.J                  ! False on Diagonal
          WHERE (OffDiag)
            Products = -Surface(1:NY, 1:NY, 1) *
     &        SPREAD (Field(1:NY, 1), DIM=2, NCOPIES=NY)! Calculate products
          END WHERE
          Value(1:NY) = Value(1:NY) +         ! Sum up products at each point
     &      SUM (Products(1:NY, 1:NY), MASK=OffDiag, DIM=1)
          Value(1:NY) = Value(1:NY) -
     &      Surface(NY + 1, 1:NY, 1) * Field(1:NY, 2) ! The point below
          FORALL (I = 2:NY-1)
     &      NewField(I, 1) = Value(I) / Surface(I, I, 1)! Div by self coef
        END IF

C Now do the central points in parallel using 'CSHIFT's for communication

        Central = .FALSE.                    ! Initialise to FALSE
        Central(2:NY-1, 2:NZ-1) = .TRUE.     ! Set central points true
        WHERE (Central)
          NewField =  Matrices(:, :, 6) -                       ! RHS
     &      Matrices(:, :, 2) * EOSHIFT (Field, SHIFT=-1, DIM=1) -   ! Left
     &      Matrices(:, :, 3) * EOSHIFT (Field, SHIFT=1, DIM=1) -    ! Right
     &      Matrices(:, :, 4) * EOSHIFT (Field, SHIFT=-1, DIM=2) -   ! Up
     &      Matrices(:, :, 5) * EOSHIFT (Field, SHIFT=1, DIM=2)      ! Down
          NewField = NewField / Matrices(:, :, 1)                  ! Self
        END WHERE

C Finally Calculate the Bottom Surface iteration.

        OffDiag = .FALSE.                         ! Initialise to FALSE
        Value(1:NY) = Surface(NY + 2, 1:NY, 2)    ! Set to RHS values
        FORALL (I = 1:NY, J = 1:NY)
     &    OffDiag(I, J) = I.NE.J                  ! False on Diagonal
        WHERE (OffDiag)
          Products(1:NY, 1:NY) = -Surface(1:NY, 1:NY, 2) *
     &        SPREAD (Field(1:NY, NZ), DIM=2, NCOPIES=NY)! Calculate products
        END WHERE
        Value(1:NY) = Value(1:NY)     +       ! Sum up products at each point
     &     SUM (Products(1:NY, 1:NY), MASK=OffDiag, DIM=1)
        Value(1:NY) = Value(1:NY) -
     &     Surface(NY + 1, 1:NY, 2) * Field(1:NY, NZ-1)! The point above
        FORALL (I = 2:NY-1)
     &          NewField(I, NZ) = Value(I) / Surface(I, I, 2)  ! Div by self coef

C Calculate the residual at each point.

          WHERE (ActiveNodes)
            Change = NewField - Field
            Residual = SQRT (REAL (Change)**2 +
     &      AIMAG (Change)**2)
          END WHERE
          MaxResidual = MAXVAL (Residual, mask=ActiveNodes)

C Copy NewField to Field

          WHERE (ActiveNodes)
            Field = NewField
          END WHERE

C Print iteration number and maximum residual

          PRINT *, 'Iteration ', Iteration, ' MaxResidual ',
     &            MaxResidual

C Is the maximum residual less than the prescribed error?
C If so, then just return now.

          IF (MaxResidual.LT.Error) THEN
            PRINT *, 'Needed ', Iteration, ' Iterations'
            RETURN
          END IF

        END DO

        PRINT *, 'Reached Maximum Iteration Count of ', MaxIter
        PRINT *, 'Solution not converged to required residual.'

        RETURN
        END
```

Appendix B

Three Dimensional Code Excerpts

B.1 Iter3D.FCM

```
C------------------ ITER  --------------------------------------------
C Original by Xinghau Pu
C APR. 15,1992
C---------------------------------------------------------------------
C Rewritten in MasPar High Performance FORTRAN to use Data Parallel
C features by Kenneth MacDonald and Ashok Agarwal, Autumn 1993.
C---------------------------------------------------------------------
C added support for Connection Machine FORTRAN and C preprocessor by
C Kenneth MacDonald, March 1994.
C---------------------------------------------------------------------

C---------------------------------------------------------------------
C This is the main controlling subroutine for the iterative solver.  All
C the parallel arrays are declared and allocated in this subroutine, and
C are then passed as arguments to the subsequent subroutines.
C---------------------------------------------------------------------

/* Define Parallel or Serial version */

#include "Arch.h"

/* Do we have a timer available? */

#undef Timer_Available

#ifdef CM200_Arch
#define Timer_Available
#endif /* CM200_Arch */

#ifdef MasPar_Arch
#define Timer_Available
#endif /* MasPar_Arch */

#ifdef SunOS_Arch
#define Timer_Available
#endif /* SunOS_Arch */
```

```
      SUBROUTINE ITER (NX, NY, NZ)

      IMPLICIT NONE

      INCLUDE 'EM3D.F'

#ifdef Timer_Available
#ifdef CM200_Arch
      INCLUDE '/usr/include/cm/timer-fort.h'
#endif /* CM200_Arch */
#endif /* Timer_Available */

C Arguments

      INTEGER NX, NY, NZ             ! The size of the model grid

C Common Blocks

      INTEGER IData(NI)              ! Misc. INTEGER variables
      COMMON  /IData/IData
      REAL*8  RData(NR)              ! Misc REAL variables
      COMMON  /RData/RData
      COMPLEX*16 CData(NC)           ! Misc COMPLEX variables
      COMMON  /CData/CData

      COMPLEX*16 XX(LD,MD,ND)        ! Bx
      COMPLEX*16 YY(LD,MD,ND)        ! By
      COMPLEX*16 ZZ(LD,MD,ND)        ! Bz
      COMPLEX*16 XO(LD,MD)           ! Bxo (Thin Sheet)
      COMPLEX*16 YO(LD,MD)           ! Byo (Thin Sheet)
      COMMON  /BXBYBZ/ XX,YY,ZZ,XO,YO

      LOGICAL ThinSheet              ! Thin Sheet Flag
      COMMON  /THIN/ ThinSheet

C Local Variables

#ifdef Parallel_Arch
C   Field Values

      COMPLEX*16 Field(3*NX,NY,NZ)     ! B Field
      COMPLEX*16 ThinField(3*NX,NY)    ! Thin Sheet B Field
      COMPLEX*16 DeltaField(3*NX,NY,0:NZ) ! Change in the Field

C   Top surface coefficients

      REAL*8 TopSx(NX*NY, NX*NY)      ! Surface integral X=Sx*Z
      REAL*8 TopSy(NX*NY, NX*NY)      ! Surface integral Y=Sy*Z
      REAL*8 TopA(3*NX,NY,7)          ! Self components X=TopA*X,..
      REAL*8 TopB(3*NX,NY,6 )         ! Cross component+1 X=TopB*Y,..
      REAL*8 TopC(3*NX,NY,6 )         ! Cross component+2 X=TopC*Z,..
      COMPLEX*16 TopAo(3*NX, NY )     ! Singular point X=X/TopAo,..

C   Internal cell coefficients

      REAL*8 CelA(3*NX, NY, NZ, 7)    ! Self components X=CelA*X,..
      REAL*8 CelB(3*NX, NY, NZ, 19)   ! Cross component+1 X=CelB*Y,..
      REAL*8 CelC(3*NX, NY, NZ, 19)   ! Cross component+2 X=CelC*Z,..

C   Bottom surface coefficients

      REAL*8 BotA(3*NX, NY, 6)        ! Self components X=BotA*X,..
      REAL*8 BotB(3*NX, NY, 6)        ! Cross component+1 X=BotB*Y,..
      REAL*8 BotC(3*NX, NY, 6)        ! Cross component+2 X=BotC*Z,..
      COMPLEX*16 ParSI4to9(NX,NY)     ! Extra X Terms X=X+ParSI4to9
      COMPLEX*16 BotS(NX*NY,NX*NY )   ! Surface integral X=S*X,..
```

```
        COMPLEX*16 BotAo(3*NX, NY)       ! Singular point X=X/BotAo,..

C   Loop counters

        INTEGER I, J, K

#endif /* Parallel_Arch */

C   Misc scalar variables

        INTEGER NSTART, ITEMAX, NNITER, IOUTPT, MCHECK, NCHEC3,
     &   II, MSG, IP, JP, KP, NNIT
        REAL*8  Error, EPS
        COMPLEX*16 Bo

#ifdef Timer_Available
C   Local timing variables

        INTEGER Time, TotalTime
#ifdef SunOS_Arch
        INTEGER TimeStart, TimeEnd
#endif /* SunOS_Arch */
#endif /* Timer_Available */


#ifdef Parallel_Arch

C Residual Function Declaration

        REAL ParResidual
#endif


#ifdef Timer_Available
C Timer Function Declarations

#ifdef MasPar_Arch
        INTEGER mpTimerElapsed
#endif /* MasPar_Arch */
#endif /*Timer_Available */

#ifdef CM200_Arch

C Compiler Directives

CMF$    COMMON FEONLY /IData/
CMF$    COMMON FEONLY /RData/
CMF$    COMMON FEONLY /CData/
CMF$    COMMON FEONLY /THIN/
CMF$    COMMON FEONLY /BXBYBZ/
CMF$    LAYOUT Field(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT ThinField(:NEWS, :NEWS)
CMF$    LAYOUT DeltaField(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT TopSx(:NEWS, :NEWS)
CMF$    LAYOUT TopSy(:NEWS, :NEWS)
CMF$    LAYOUT TopA(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT TopB(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT TopC(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT TopAo(:NEWS, :NEWS)
CMF$    LAYOUT CelA(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT CelB(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT CelC(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT BotS(:NEWS, :NEWS)
CMF$    LAYOUT BotA(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT BotB(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT BotC(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT BotAo(:NEWS, :NEWS)
```

```
CMF$    LAYOUT ParSI4to9(:NEWS, :NEWS)
#endif /* CM200_Arch */

C Equivalence Statements

        EQUIVALENCE (ITEMAX,IData(7)),(NCHEC3,IData(8)),
     &    (IOUTPT,IData(9)),(NNITER,IData(10)),(MSG,IData(15)),
     &    (IP,IData(16)),(JP,IData(17)),(KP,IData(18))
        EQUIVALENCE (EPS,RData(3)),(Error,RData(4))

C Print out some diagnostics.

#ifdef DEBUG
        PRINT *, 'Got into ITER, with NX, NY, NZ = ', NX, NY, NZ
#endif /* DEBUG */

#ifdef Parallel_Arch
        PRINT *, 'Parallel Iterative Solver Version 1.0'
#ifdef CM200_Arch
        PRINT *, 'Connection Machine FORTRAN'
#endif /* CM200_Arch */
#ifdef MasPar_Arch
        PRINT *, 'MasPar High Performance FORTRAN'
#endif /* MasPar_Arch */
#endif /* Parallel_Arch */

#ifdef Serial_Arch
        PRINT *, 'Serial Iterative Solver Version 1.0'
#ifdef SunOS_Arch
        PRINT *, 'SunOS FORTRAN 77'
#endif /* SunOS_Arch */
#ifdef OSF_Arch
        PRINT *, 'OSF/1 FORTRAN 77'
#endif /* OSF_Arch */
#endif /* Serial_Arch */

        PRINT *, 'Jacobian Method'
        IF (ThinSheet) THEN
          PRINT *, 'Including Thin Sheet Approximation'
        END IF

C Set Bo, ie. the source field.

        Bo = CData(1)

#ifdef Parallel_Arch
C Initialise the parallel arrays, if on a parallel machine. On the
C Connection Machine, this should help force the compiler to allocate the
C arrays on the CM (DPU).

        Field = (0.0, 0.0)
        ThinField = (0.0, 0.0)
        DeltaField = (0.0, 0.0)
        TopSx = (0.0, 0.0)
        TopSy = (0.0, 0.0)
        TopA = 0.0
        TopB = 0.0
        TopC = 0.0
        TopAo = (0.0, 0.0)
        CelA = 0.0
        CelB = 0.0
        CelC = 0.0
        BotA = 0.0
        BotB = 0.0
        BotC = 0.0
        ParSI4to9 = (0.0, 0.0)
        BotS = (0.0, 0.0)
        BotAo = (0.0, 0.0)

C If we're on a parallel machine, then all the field values and
C coefficients need to be copied over to the back end, in a symmetric
C shape.

C Copy the field values on to the field array

         DO K = 1, NZ
          DO J = 1, NY
            DO I = 1, NX
              Field(I,J,K) = XX(I,J,K)
              Field(I+NX,J,K) = YY(I,J,K)
              Field(I+2*NX,J,K) = ZZ(I,J,K)
            END DO
          END DO
        END DO

C Initialise the Thinsheet Field array from the field at the surface
C above the thin sheet (in case it is present)

        DO J = 1, NY
          DO I = 1, NX
            ThinField(I,J) = XX(I,J,1 )
            ThinField(I+NX,J) = YY(I,J,1 )
            ThinField(I+2*NX,J) = ZZ(I,J,1 )
          END DO
        END DO

C Copy the coefficients to the parallel representation

        CALL SBCCOE (TopA, TopB, TopC, TopSx, TopSy, TopAo,
     &          ThinSheet, NX, NY)
        CALL CELCOE (CelA, CelB, CelC, NX, NY, NZ)
        CALL BBCCOE (BotA, BotB, BotC, ParSI4to9, BotS, BotAo,
     &          NX, NY, NZ)

#endif /* Parallel_Arch */

#ifdef Timer_Available
C Do anything we need for timers, before the iterations start

#ifdef SunOS_Arch
        CALL clock (TotalTime)          ! Start SunOS clock
#endif /* SunOS_Arch */
        TotalTime = 0
#endif /* Timer_Available */

C Start the iteration loop

        NSTART=NNITER+1
        DO II = NSTART, ITEMAX

#ifdef Timer_Available
#ifdef MasPar_Arch
          CALL mpTimerStart ()          ! Start MasPar clock
#endif /* MasPar_Arch */
#ifdef CM200_Arch
          CALL CM_timer_clear (0)       ! Clear CM clock
          CALL CM_timer_start (0)       ! Start CM clock
#endif /* CM200_Arch */
#endif /* Timer_Available */

          NNIT=II
          NNITER=II
          Error=0
          MCheck = MOD (NNIter, NChec3)
```

```
C Update the top, middle, and bottom of the grid...

#ifdef Parallel_Arch

#ifdef Timer_Available
#ifdef CM200_Arch
          CALL CM_timer_clear (1)        ! Clear CM clock
          CALL CM_timer_start (1)        ! Start CM clock
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */

          CALL ParITESBC (Field, ThinField, TopSx, TopSy,
     &          TopA, TopB, TopC, TopAo, Bo, DeltaField,
     &          NX, NY, NZ, ThinSheet )

#ifdef Timer_Available
#ifdef CM200_Arch
          CALL CM_timer_stop (1)
          Time = INT (CM_timer_read_cm_busy (1) * 1000.0)
          CALL CM_timer_clear (1)
          CALL CM_timer_start (1)
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /* SunOS_Arch */

          PRINT *, 'Top Surface: ', Time, ' (ms)'

#ifdef CM200_Arch
          CALL CM_timer_clear (1)        ! Clear CM clock
          CALL CM_timer_start (1)        ! Start CM clock
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */

          CALL ParITECEL( Field, CelA, CelB, CelC, DeltaField,
     &          NX, NY, NZ)

#ifdef Timer_Available
#ifdef CM200_Arch
          CALL CM_timer_stop (1)
          Time = INT (CM_timer_read_cm_busy (1) * 1000.0)
          CALL CM_timer_clear (1)
          CALL CM_timer_start (1)
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /* SunOS_Arch */

          PRINT *, 'Central Points: ', Time, ' (ms)'

#ifdef CM200_Arch
          CALL CM_timer_clear (1)        ! Clear CM clock
          CALL CM_timer_start (1)        ! Start CM clock
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */
```

```
          CALL ParITEBBC(Field, BotS, BotAo, BotA, BotB, BotC,
     &           ParSI4to9, DeltaField, NX, NY, NZ)

#ifdef Timer_Available
#ifdef CM200_Arch
          CALL CM_timer_stop (1)
          Time = INT (CM_timer_read_cm_busy (1) * 1000.0)
          CALL CM_timer_clear (1)
          CALL CM_timer_start (1)
#endif /* CM200_Arch */
#ifdef SunOS_Arch
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /*SunOS_Arch */

          PRINT *, 'Bottom Surface: ', Time, ' (ms)'

#endif /* Timer_Available */

          Error = ParRESIDUAL (DeltaField, NX, NY, NZ, IP, JP, KP,
     &      ThinSheet )

#endif /* Parallel_Arch */

#ifdef Serial_Arch

#ifdef Timer_Available
#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */

          CALL ITESBC (MCheck)

#ifdef Timer_Available
#ifdef SunOS_Arch
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /* SunOS_Arch */

          PRINT *, 'Top Surface: ', Time, ' (ms)'

#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */

          CALL ITECEL (MCheck)

#ifdef Timer_Available
#ifdef SunOS_Arch
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /* SunOS_Arch */

          PRINT *, 'Central Points: ', Time, ' (ms)'

#ifdef SunOS_Arch
          CALL clock (TimeStart)
#endif /* SunOS_Arch */
#endif /* Timer_Available */

          CALL ITEBBC (MCheck)

#ifdef Timer_Available
#ifdef SunOS_Arch
```

```
          CALL clock (TimeEnd)
          Time = (TimeEnd - TimeStart) / 1000
#endif /*SunOS_Arch */

          PRINT *, 'Bottom Surface: ', Time, ' (ms)'

#endif /* Timer_Available */

#endif /* Serial_Arch */

#ifdef Timer_Available

C Get the time for this iteration, and the running total time so far,
C both in milliseconds

#ifdef MasPar_Arch
          Time = mpTimerElapsed ()
          TotalTime = TotalTime + TIME
#endif /* MasPar_Arch */
#ifdef CM200_Arch
          CALL CM_timer_stop (0)
          Time = INT (CM_timer_read_cm_busy (0) * 1000.0)
          TotalTime = TotalTime + Time
#endif /* CM200_Arch */
#ifdef SunOS_Arch
            CALL clock (Time)
            Time = Time / 1000
            Time = Time - TotalTime
            TotalTime = TotalTime + Time
#endif /* SunOS_Arch */

C Report progress if we need to...

          IF(MCHECK.EQ.0) THEN
            WRITE(*,789) NNITER, Error, IP, JP, KP,
     &        TotalTime, Time
          END IF
#else /* Timer_Not_Available */
          IF(MCHECK.EQ.0) WRITE(*,789) NNITER, Error, IP, JP, KP
#endif /* Timer_Available */

C Has it converged yet?

          IF(MCHECK.EQ.0.AND.Error.LE.EPS) THEN
            MSG=111

#ifdef Timer_Available
            PRINT *, 'Successfully converged after ', TotalTime,
     &        ' milliseconds.'
#else /* Timer_Not_Available */
            PRINT *, 'Successfully converged.'
#endif /* Timer_Available */
            PRINT *, 'Took ', NNITER, ' iterations to reach ',
     &        Error, ' error.'

#ifdef Parallel_Arch
C After successful convergence, copy the field values

            IF (ThinSheet) THEN
              DO J= 2, NY-1
                DO I = 2, NX-1
                  XO(I,J) = ThinField(I,J)
                  YO(I,J) = ThinField(I+NX,J)
                END DO
              END DO
            END IF
            DO K = 1, NZ
              DO J = 2, NY-1
                DO I = 2, NX-1
                  XX(I,J,K) = Field(I,J,K)
                  YY(I,J,K) = Field(I+NX,J,K)
                  ZZ(I,J,K) = Field(I+2*NX,J,K)
                END DO
              END DO
            END DO
#endif /* Parallel_Arch */

            RETURN
          ENDIF          ! Converged?

C Should we write the current state to a file?

          IF(NNITER.EQ.IOUTPT.AND.NNITER.NE.ITEMAX) THEN
            CALL RDSA3D(2)
          ENDIF

C End of iteration loop

        END DO

C Didn't converge

#ifdef Timer_Available
        PRINT *, 'No convergence after ', TotalTime,
     &    ' milliseconds.'
#else /* Timer_Not_Available */
        PRINT *, 'No convergence...'
#endif /* Timer_Available */
        PRINT *, 'Took ', NNITER, ' iterations to reach ',
     &    Error, ' error.'

#ifdef Parallel_Arch
C After maximum iteration exhausted, copy the field values

        IF (ThinSheet) THEN
          DO J = 2, NY-1
            DO I = 2, NX-1
              XO(I,J) = ThinField(I,J)
              YO(I,J) = ThinField(I+NX,J)
            END DO
          END DO
        END IF

        DO K = 1, NZ
          DO J = 2, NY-1
            DO I = 2, NX-1
              XX(I,J,K) = Field(I,J,K)
              YY(I,J,K) = Field(I+NX,J,K)
              ZZ(I,J,K) = Field(I+2*NX,J,K)
            END DO
          END DO
        END DO
#endif /* Parallel_Arch */

        MSG= -111

#ifdef Timer_Available
 789    FORMAT(1X, 'Iteration = ', I6, ' Error = ', E10.3,
     &    ' Location = ', 3I3, ' Time (ms) = ', I10, I10 )
#else /* Timer_Not_Available */
 789    FORMAT(1X, 'Iteration = ', I6, ' Error = ', E10.3,
     &    ' Location = ', 3I3)
#endif /* Timer_Available */
```

```
C Finished, so return.

      RETURN
      END
```

B.2 Bottom.FCM

```
C ParITEBBC.f - Subroutine to perform one iteration over all the grid
C points on the bottom plane of the model.  Calculates the new field at
C all the bottom plane points, and only updates the old field and array of
C changes at the points of interest, ie.  not on the boundaries.

C MasPar High Performance Fortran

C Kenneth MacDonald
C 30th November 1993
C Updated 6th December 1993
C Support for Connection Machine FORTRAN and cpp added,
C Kenneth MacDonald March 1994

C ==========================================================================

        SUBROUTINE ParITEBBC (Field, S, Ao, A, B, C, SI4to9,
     &    DeltaField, NX, NY, NZ)

C Do one iteration on the bottom boundary of the model.  This differs
C from the central points in that there is a surface integral to be
C calculated for each component, and an extra term for the X component too
C (SI4to9).

        IMPLICIT NONE

C Arguments

        INTEGER NX, NY, NZ                ! The size of the model
        COMPLEX*16 Field(3*NX, NY, NZ)    ! The 3 components of the field
        COMPLEX*16 S(NX*NY, NX*NY)        ! The surface integral coefficients
        COMPLEX*16 Ao(3*NX, NY)           ! Singular point coefficient
        REAL*8  A(3*NX, NY, 6)            ! Same component coefficients
        REAL*8  B(3*NX, NY, 6)            ! Component+1 coefficients
        REAL*8  C(3*NX, NY, 6)            ! Component+2 coefficients
        COMPLEX*16 SI4to9(NX, NY)         ! Extra to add to X comp
        COMPLEX*16 DeltaField(3*NX, NY, 0:NZ)      ! The field changes

C Local Variables

        COMPLEX*16 FieldShift(3*NX, NY)          ! Shifted field
        COMPLEX*16 NewField(3*NX, NY)            ! The new field
        COMPLEX*16 Integrands(NX*NY, NX*NY)      ! Surface integral (B * S)
        COMPLEX*16 StretchField(NX*NY)           ! Bottom plane in a vector
        LOGICAL OffDiagonal(NX*NY, NX*NY)        ! True for off the diagonal
        LOGICAL All2D(NX, NY)                    ! True everywhere
        INTEGER I, J                             ! Grid point indices
        REAL*8  Pi                               ! Just Pi

C Compiler directives

#ifdef MasPar_Arch
CMPF    ONDPU FieldShift
CMPF    ONDPU NewField
CMPF    ONDPU Integrands
CMPF    ONDPU StretchField
CMPF    ONDPU OffDiagonal
CMPF    ONDPU All2D
#endif
#ifdef CM200_Arch
CMF$    LAYOUT Field(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT S(:NEWS, :NEWS)
CMF$    LAYOUT Ao(:NEWS, :NEWS)
CMF$    LAYOUT A(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT B(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT C(:NEWS, :NEWS, :SERIAL)
```

```
CMF$    LAYOUT SI4to9(:NEWS, :NEWS)
CMF$    LAYOUT DeltaField(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT FieldShift(:NEWS, :NEWS)
CMF$    LAYOUT NewField(:NEWS, :NEWS)
CMF$    LAYOUT Integrands(:NEWS, :NEWS)
CMF$    LAYOUT StretchField(:NEWS)
CMF$    LAYOUT OffDiagonal(:NEWS, :NEWS)
CMF$    LAYOUT All2D(:NEWS, :NEWS)
#endif


C First of all we calculate each integral for each component in
C sequence.  The 2D bottom surface is reshaped to a 1D vector, which is
C spread over the 2D array of coefficients (S).  These two are multiplied
C elementally, and then summed over one dimension to calculate the
C integral at each point.  These values are then reshaped back to the 2D
C surface grid.  The code for each component is given explicitly, no loop
C over the three components.

C Initialise Pi

        Pi = 4.0 * ATAN (1.0)

C Set the pack / unpack masks to true everywhere

        All2D = .TRUE.

C Set the mask to true on the off diagonal elements

        OffDiagonal = .TRUE.                   ! Default is true
        FORALL (I=1:NX*NY, J=1:NX*NY)
     &    OffDiagonal(I, J) = (I.NE.J)         ! False on diagonal

C First reshape the X component at the bottom plane of the model

        StretchField = PACK (ARRAY=Field(1:NX, :, NZ), MASK=.TRUE.,
     &    VECTOR=StretchField)

C Spread this and multiple by the coefficients to get the integrands,
C under control of the off diagonal mask

        WHERE (OffDiagonal)
          Integrands = S * SPREAD (StretchField, DIM=2, NCOPIES=NX*NY)
        END WHERE

C Now sum up the integrands at each grid point

        StretchField = SUM (Integrands, MASK=OffDiagonal, DIM=1)

C Reshape the stretched field back to the 2D grid shape

        NewField(1:NX, :) = UNPACK (VECTOR=StretchField,
     &    MASK=All2D, FIELD=NewField(1:NX, :))

C This is the X component, so add on the extra terms

        NewField(1:NX, :) = NewField(1:NX, :) + SI4to9

C Now divide by (2Pi)

        NewField(1:NX, :) = NewField(1:NX, :) / (2D0 * Pi)

C The same needs to be done for Y component now

        StretchField = PACK (ARRAY=Field(NX+1:2*NX, :, NZ),
     &    MASK=.TRUE., VECTOR=StretchField)
```

```
        WHERE (OffDiagonal)
          Integrands = S * SPREAD (StretchField, DIM=2, NCOPIES=NX*NY)
        END WHERE

C Sum integrands

        StretchField = SUM (Integrands, MASK=OffDiagonal, DIM=1)

        NewField(NX+1:2*NX, :) = UNPACK (VECTOR=StretchField,
     &    MASK=All2D, FIELD=NewField(NX+1:2*NX, :))

        NewField(NX+1:2*NX, :) = NewField(NX+1:2*NX, :) / (2D0 * Pi)

C And finally the Z component

        StretchField = PACK (ARRAY=Field(2*NX+1:3*NX, :, NZ),
     &    MASK=.TRUE., VECTOR=StretchField)

        WHERE (OffDiagonal)
          Integrands = S * SPREAD (StretchField, DIM=2, NCOPIES=NX*NY)
        END WHERE

C Sum integrands

        StretchField = SUM (Integrands, MASK=OffDiagonal, DIM=1)

        NewField(2*NX+1:3*NX, :) = UNPACK (VECTOR=StretchField,
     &    MASK=All2D, FIELD=NewField(2*NX+1:3*NX, :))

        NewField(2*NX+1:3*NX, :) = NewField(2*NX+1:3*NX, :) / (2D0 * Pi)

C Now do the rest of the terms in parallel

C The extra self compnents. The sixth coefficient is for the NZ-1 field
C value, so the (NZ-1) plane is used.

        NewField = NewField +
     &    A(:, :, 2) * CSHIFT (Field(:, :, NZ), DIM=1, SHIFT=-1) +      ! 011
     &    A(:, :, 3) * CSHIFT (Field(:, :, NZ), DIM=1, SHIFT=1) +       ! 211
     &    A(:, :, 4) * CSHIFT (Field(:, :, NZ), DIM=2, SHIFT=-1) +      ! 101
     &    A(:, :, 5) * CSHIFT (Field(:, :, NZ), DIM=2, SHIFT=1) +       ! 121
     &    A(:, :, 6) * Field(:, :, NZ-1)                                ! 110

C Copy the bottom plane to the shifted array for the component+1 terms

        FieldShift = CSHIFT (Field(:, :, NZ), DIM=1, SHIFT=NX)

C Add on the component+1 terms. The sixth coefficient is multiplied by
C the (NZ-1) plane shifted by (+NX).

        NewField = NewField +
     &    B(:, :, 1) * FieldShift +                                     ! 111
     &    B(:, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +           ! 011
     &    B(:, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +            ! 211
     &    B(:, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +           ! 101
     &    B(:, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +            ! 121
     &    B(:, :, 6) * CSHIFT (Field(:, :, NZ-1), DIM=1, SHIFT=NX)      ! 110

C Shift the bottom plane of the field by another component (NX)

        FieldShift = CSHIFT (FieldShift, DIM=1, SHIFT=NX)

C Add on the component+2 terms. This time the sixth coefficient is
C multiplied by the (NZ-1) plane shifted by (-NX), which is the same as
C (+2NX), but faster.

        NewField = NewField +
     &    C(:, :, 1) * FieldShift +                                ! 111
     &    C(:, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +      ! 011
     &    C(:, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +       ! 211
     &    C(:, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +      ! 101
     &    C(:, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +       ! 121
     &    C(:, :, 6) * CSHIFT (Field(:, :, NZ-1), DIM=1, SHIFT=-NX)  ! 110

C Divide through by the singular point coefficient, but being careful of
C Ao=0

      WHERE (Ao.NE.0)
        NewField = NewField / Ao
      END WHERE

C Update the array of changes in the field, at the bottom plane.  Update
C all the points on the plane, even the boundaries.  These should then be
C ignored when the residuals are being calculated.

      DeltaField(:, :, NZ) = NewField - Field(:, :, NZ)

C Copy the new values to the old field now, component at a time

      Field(2:NX-1, 2:NY-1, NZ) = NewField(2:NX-1, 2:NY-1)    ! X component
      Field(NX+2:2*NX-1, 2:NY-1, NZ) =
     &  NewField(NX+2:2*NX-1, 2:NY-1)                         ! Y component
      Field(2*NX+2:3*NX-1, 2:NY-1, NZ) =
     &  NewField(2*NX+2:3*NX-1, 2:NY-1)                       ! Z component

C Done

      RETURN
      END
```

B.3 Top.FCM

```
C ParITESBC.f - Subroutine to perform one iteration over the top surface
C of the model grid.  A Thin Sheet layer is catered for, if present.  The
C new field values are calculated for all the gridpoints in the thin sheet
C and top layer of the model, and only the relevent values are updated in
C the array of changes, and the old field values.

C MasPar High Performance FORTRAN 90

C Kenneth MacDonald
C 30th November 1993
C Updated 6th December 1993
C Added support for Connection Machine FORTRAN and cpp March 1994

C ============================================================================

        SUBROUTINE ParITESBC (Field, Sheet, SX, SY,
     &    A, B, C, Ao, Bo, DeltaField, NX, NY, NZ, ThinSheet)

C The subroutine calculates one iteration at the top surface of the
C model.  The possibility of a Thin Sheet layer is catered for, but isn't
C necessary.  The X-, and Y- components (above the sheet) are calculated
C separately, and then the X+, Y+ and Z components are calculated in
C parallel.  If there is no thin sheet the X- and Y- components are copied
C to X+ and Y+ respectively.

        IMPLICIT NONE

C Arguments

        INTEGER          NX, NY, NZ       ! The model size
        COMPLEX*16       Field(3*NX, NY, NZ)! Field components (Bx|By|Bz)
        COMPLEX*16       Sheet(3*NX, NY) ! 3 components for Thin Sheet, Z=0-
        REAL*8           SX(NX*NY, NX*NY)! X component integral coefficients
        REAL*8           SY(NX*NY, NX*NY)! Y component integral coefficients
        REAL*8           A(3*NX, NY, 7)  ! Same component coefficients,X=A*X,..
        REAL*8           B(3*NX, NY, 6)  ! Component+1 coefficients, X=B*Y,..
        REAL*8           C(3*NX, NY, 6)  ! Component+2 coefficients, X=C*Z,..
        COMPLEX*16       Ao(3*NX, NY)    ! Singular point values
        COMPLEX*16       Bo              ! Source field scalar
        COMPLEX*16       DeltaField(3*NX, NY, 0:NZ)! Change in the field
        LOGICAL          ThinSheet       ! Indicates presence of Thin Sheet

C Local Variables

        COMPLEX*16       NewField(3*NX, NY)        ! The new calculated values
        COMPLEX*16       NewSheet(3*NX, NY)        ! The new thin sheet values
        COMPLEX*16       FieldShift(3*NX, NY)      ! Plane of shifted field
        COMPLEX*16       Integrands(NX*NY, NX*NY)! Integral (coeffs * field)
        COMPLEX*16       StretchField(NX*NY)       ! Field stretched to a vector
        LOGICAL          All2D(NX, NY)             ! All set True
        LOGICAL          ActiveNodes(3*NX, NY)     ! Active grid points
        INTEGER          I, J                      ! Counters
        REAL*8           Pi                        ! Pi

C Compiler Directives

#ifdef MasPar_Arch
CMPF    ONDPU NewField
CMPF    ONDPU NewSheet
CMPF    ONDPU FieldShift
CMPF    ONDPU Integrands
CMPF    ONDPU StretchField
CMPF    ONDPU All2D
CMPF    ONDPU ActiveNodes
#endif
```

```
#ifdef CM200_Arch
CMF$    LAYOUT Field(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT Sheet(:NEWS, :NEWS)
CMF$    LAYOUT SX(:NEWS, :NEWS)
CMF$    LAYOUT SY(:NEWS, :NEWS)
CMF$    LAYOUT A(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT B(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT C(:NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT Ao(:NEWS, :NEWS)
CMF$    LAYOUT DeltaField(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT NewField(:NEWS, :NEWS)
CMF$    LAYOUT NewSheet(:NEWS, :NEWS)
CMF$    LAYOUT FieldShift(:NEWS, :NEWS)
CMF$    LAYOUT Integrands(:NEWS, :NEWS)
CMF$    LAYOUT StretchField(:NEWS)
CMF$    LAYOUT All2D(:NEWS, :NEWS)
CMF$    LAYOUT ActiveNodes(:NEWS, :NEWS)
#endif

C Begin Code

C Set the value of Pi

        Pi = 4.0 * ATAN (1.0)

C Set the logicals All1D and All2D, to .TRUE., which are used in the
C packing and unpacking of the 2D <-> 1D representations of the field.

        All2D = .TRUE.

C Calculate the X- component, by the surface integral.  This is needed
C for both cases.

C Reshape the field into a 1D vector array.  If there is no thin sheet,
C then use Z=1, else use Sheet

        StretchField = PACK (ARRAY=Field(2*NX+1:3*NX, :, 1),
     &    MASK=.TRUE., VECTOR=StretchField)

C Calculate the integrands by spreading the field, and multiplying by the
C coefficients

        Integrands = SX * SPREAD (StretchField, DIM=2, NCOPIES=NX*NY)

C Sum up the integrands at each point

        StretchField = SUM (Integrands, DIM=1)

C Reshape the 1D vector field back to it's 2D shape

        NewSheet(1:NX, :) = - UNPACK (VECTOR=StretchField,
     &    MASK=All2D, FIELD=NewSheet(1:NX,:))

        NewSheet(1:NX, :) = NewSheet(1:NX, :) / (2 * Pi)

C Add on the Bo value (a complex scalar)

        NewSheet(1:NX, :) = NewSheet(1:NX, :) + Bo

C Calculate the Y- component, by the surface integral.  This too is
C needed for both cases.

        StretchField = PACK (ARRAY=Field(2*NX+1:3*NX, :, 1), MASK=.TRUE.,
     &    VECTOR=StretchField)
        Integrands = SY * SPREAD (StretchField, DIM=2, NCOPIES=NX*NY)
        StretchField = SUM (Integrands, DIM=1)
        NewSheet(NX+1:2*NX, :) = - UNPACK (VECTOR=StretchField,
     &    MASK=All2D, FIELD=NewSheet(NX+1:2*NX,:))
        NewSheet(NX+1:2*NX, :) = NewSheet(NX+1:2*NX, :) / (2 * Pi)

C Now calculate the three components for the Z=1 (X+, Y+, Z).  X+ and Y+
C will be zero if no thin sheet.

C Same component terms. Use Sheet (0-) values for Z-1 direction.

        NewField = A(:, :, 2) * CSHIFT (Field(:, :, 1), DIM=1, SHIFT=-1) +! 011
     &    A(:, :, 3) * CSHIFT (Field(:, :, 1), DIM=1, SHIFT=1) +         ! 211
     &    A(:, :, 4) * CSHIFT (Field(:, :, 1), DIM=2, SHIFT=-1) +        ! 101
     &    A(:, :, 5) * CSHIFT (Field(:, :, 1), DIM=2, SHIFT=1) +         ! 121
     &    A(:, :, 6) * Sheet +                                           ! 110
     &    A(:, :, 7) * Field(:, :, 2)                                    ! 112

C Component + 1 terms. Copy the field to a temporary array.

        FieldShift = CSHIFT (Field(:, :, 1), DIM=1, SHIFT=NX)

C Add up the terms

        NewField = NewField +
     &    B(:, :, 1) * FieldShift +                                      ! 111
     &    B(:, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +            ! 011
     &    B(:, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +             ! 211
     &    B(:, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +            ! 101
     &    B(:, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +             ! 121
     &    B(:, :, 6) * CSHIFT (Field(:, :, 2), DIM=1, SHIFT=NX)          ! 112

C Component + 2 terms.

        FieldShift = CSHIFT (FieldShift, DIM=1, SHIFT=NX)

C Add up the terms

        NewField = NewField +
     &    C(:, :, 1) * FieldShift +                                      ! 111
     &    C(:, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +            ! 011
     &    C(:, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +             ! 211
     &    C(:, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +            ! 101
     &    C(:, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +             ! 121
     &    C(:, :, 6) * CSHIFT (Field(:, :, 2), DIM=1, SHIFT=-NX)         ! 112

C Divide through by the singular point value, checking if it is 0

        WHERE (Ao.NE.0)
          NewField = NewField / Ao
        END WHERE

C Copy the field components according to the presence of a thin sheet.
C If there is, then Z- = Z+.
C If there isn't, then X+ = X-, and Y+ = y-.

        IF (ThinSheet) THEN
          NewSheet(2*NX+1:3*NX, :) = NewField(2*NX+1:3*NX, :)! Copy Z to sheet
        ELSE
          NewField(1:2*NX, :) = NewSheet(1:2*NX, :)      ! Copy X, Y to Field(1)
        END IF

C As a safety measure, copy Z to the sheet. Z=A*Z needs Sheet value.

        NewSheet(2*NX+1:3*NX, :) = NewField(2*NX+1:3*NX, :)

C Update the array of changes in fields.

        IF (ThinSheet) THEN
          DeltaField(:, :, 0) = NewSheet - Sheet
```

```
      END IF
      DeltaField(:, :, 1) = NewField - Field(:, :, 1)

C Set the active node points. Leave the boundaries alone.

      ActiveNodes = .FALSE.
      ActiveNodes(2:NX-1, 2:NY-1) = .TRUE.          ! Set X components true
      ActiveNodes(NX+2:2*NX-1, 2:NY-1) = .TRUE.     ! Set Y components true
      ActiveNodes(2*NX+2:3*NX-1, 2:NY-1) = .TRUE.   ! Set Z components true

C Copy the new values to the old arrays

      WHERE (ActiveNodes)
        Field(:, :, 1) = NewField
        Sheet = NewSheet
      END WHERE

C Done

      RETURN
      END
```

B.4 Interior.FCM

```
C ParITECEL.f - Subroutine to perform one iteration over all the central
C grid points.  It also updates the array of changes.

C MasPar High Performance Fortran

C Kenny MacDonald 29 Nov 93
C Updated 6 Dec 1993
C Added support for Connection Machine FORTRAN and cpp,
C Kenneth MacDonald March 1994.

C ========================================================================

      SUBROUTINE ParITECEL (Field, A, B, C, DeltaField, NX, NY, NZ)

C Do one iteration over the central points.  The coefficients are stored
C in three arrays (A, B, and C).  Each consists of several 'planes'
C conformable with the field array.

C A links corresponding components
C B links the component to component + 1
C C links the component to component + 2

C Essentially, we have the following three equations for the three
C components.

C X = A(1:NX) * X + B(1:NX) * Y + C(1:NX) * Z
C Y = A(NX+1:2NX) * Y + B(NX+1:2NX) * Z + C(NX+1:2NX) * X
C Z = A(2NX+1:3NX) * Z + B(2NX+1:3NX) * X + C(2NX+1:3NX) * Y

      IMPLICIT NONE

C Arguments

      INTEGER         NX, NY, NZ                ! The size of the model
      COMPLEX*16      Field(3*NX, NY, NZ)       ! The B field (Bx|By|Bz)
      REAL*8          A(3*NX, NY, NZ, 7)        ! Same component coefficients
      REAL*8          B(3*NX, NY, NZ, 19)       ! Coefficients for component+1
      REAL*8          C(3*NX, NY, NZ, 19)       ! Coefficients for component+2
      COMPLEX*16      DeltaField(3*NX, NY, 0:NZ)! The field changes

C Local Variables

      COMPLEX*16      NewField(3*NX, NY, NZ)    ! The updated field
      COMPLEX*16      FieldShift(3*NX, NY, NZ)! The shifted field
      LOGICAL         Central(3*NX, NY, NZ)     ! Central points
      COMPLEX*16      i                         ! The square root of (-1)

C Compiler directives

#ifdef MasPar_Arch
CMPF    ONDPU NewField
CMPF    ONDPU FieldShift
CMPF    ONDPU Central
#endif
#ifdef CM200_Arch
CMF$    LAYOUT Field(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT A(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT B(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT C(:NEWS, :NEWS, :NEWS, :SERIAL)
CMF$    LAYOUT DeltaField(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT NewField(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT FieldShift(:NEWS, :NEWS, :NEWS)
CMF$    LAYOUT Central(:NEWS, :NEWS, :NEWS)
#endif
```

```
C Begin Code

C First Set 'i'

        i = (0.0, 1.0)

C Define the central points

        Central = .FALSE.                              ! All off by default
        Central(2:NX-1, 2:NY-1, 2:NZ-1) = .TRUE.       ! X components on
        Central(NX+2:2*NX-1, 2:NY-1, 2:NZ-1) = .TRUE.  ! Y components on
        Central(2*NX+2:3*NX-1, 2:NY-1, 2:NZ-1) = .TRUE. ! Z components on

C Calculate the new field values for all the points, and only worry
C about which ones we are interested in at the end, when we copy back to
C old field.

C Multiply the same components

        NewField = A(:, :, :, 2) * CSHIFT (Field, DIM=1, SHIFT=-1) +    ! 011
     &    A(:, :, :, 3) * CSHIFT (Field, DIM=1, SHIFT=1) +              ! 211
     &    A(:, :, :, 4) * CSHIFT (Field, DIM=2, SHIFT=-1) +             ! 101
     &    A(:, :, :, 5) * CSHIFT (Field, DIM=2, SHIFT=1) +              ! 121
     &    A(:, :, :, 6) * CSHIFT (Field, DIM=3, SHIFT=-1) +             ! 110
     &    A(:, :, :, 7) * CSHIFT (Field, DIM=3, SHIFT=1)                ! 112

C Now the component to the right. Eg. X -> Y, Y -> Z, Z -> X
C Copy the field to the shifted field

        FieldShift = CSHIFT (Field, DIM=1, SHIFT=NX)

        NewField = NewField +                  ! Remember the same components
     &    B(:, :, :, 1) * FieldShift +                                  ! 111
     &    B(:, :, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +        ! 011
     &    B(:, :, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +         ! 211
     &    B(:, :, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +        ! 101
     &    B(:, :, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +         ! 121
     &    B(:, :, :, 6) * CSHIFT (FieldShift, DIM=3, SHIFT=-1) +        ! 110
     &    B(:, :, :, 7) * CSHIFT (FieldShift, DIM=3, SHIFT=1) +         ! 112
     &    B(:, :, :, 8) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=2,SHIFT=-1)+   ! 001
     &    B(:, :, :, 9) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=2,SHIFT=1)+    ! 021
     &    B(:, :, :, 10) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=3,SHIFT=-1)+   ! 010
     &    B(:, :, :, 11) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=3,SHIFT=1)+    ! 012
     &    B(:, :, :, 12) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=-1),DIM=3,SHIFT=-1)    ! 100
        NewField = NewField +
     &    B(:, :, :, 13) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=-1),DIM=3,SHIFT=1)+    ! 102
     &    B(:, :, :, 14) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=1),DIM=3,SHIFT=-1)+    ! 120
     &    B(:, :, :, 15) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=1),DIM=3,SHIFT=1)+     ! 122
     &    B(:, :, :, 16) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=2,SHIFT=-1)+    ! 201
     &    B(:, :, :, 17) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=2,SHIFT=1)+     ! 221
     &    B(:, :, :, 18) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=3,SHIFT=-1)+    ! 210
     &    B(:, :, :, 19) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=3,SHIFT=1)      ! 212

C Now for the next components, shift by another NX in the X direction
```

```
        FieldShift = CSHIFT (FieldShift, DIM=1, SHIFT=NX)

C Now sum up the products of each direction

        NewField = NewField +                    ! Remember the same components
     &    C(:, :, :, 1) * FieldShift +                           ! 111
     &    C(:, :, :, 2) * CSHIFT (FieldShift, DIM=1, SHIFT=-1) +       ! 011
     &    C(:, :, :, 3) * CSHIFT (FieldShift, DIM=1, SHIFT=1) +        ! 211
     &    C(:, :, :, 4) * CSHIFT (FieldShift, DIM=2, SHIFT=-1) +       ! 101
     &    C(:, :, :, 5) * CSHIFT (FieldShift, DIM=2, SHIFT=1) +        ! 121
     &    C(:, :, :, 6) * CSHIFT (FieldShift, DIM=3, SHIFT=-1) +       ! 110
     &    C(:, :, :, 7) * CSHIFT (FieldShift, DIM=3, SHIFT=1) +        ! 112
     &    C(:, :, :, 8) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=2,SHIFT=-1)+   ! 001
     &    C(:, :, :, 9) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=2,SHIFT=1)+    ! 021
     &    C(:, :, :, 10) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=3,SHIFT=-1)+   ! 010
     &    C(:, :, :, 11) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=-1),DIM=3,SHIFT=1)+    ! 012
     &    C(:, :, :, 12) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=-1),DIM=3,SHIFT=-1)    ! 100
        NewField = NewField +
     &    C(:, :, :, 13) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=-1),DIM=3,SHIFT=1)+    ! 102
     &    C(:, :, :, 14) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=1),DIM=3,SHIFT=-1)+    ! 120
     &    C(:, :, :, 15) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=2,SHIFT=1),DIM=3,SHIFT=1)+     ! 122
     &    C(:, :, :, 16) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=2,SHIFT=-1)+    ! 201
     &    C(:, :, :, 17) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=2,SHIFT=1)+     ! 221
     &    C(:, :, :, 18) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=3,SHIFT=-1)+    ! 210
     &    C(:, :, :, 19) *
     &      CSHIFT(CSHIFT(FieldShift,DIM=1,SHIFT=1),DIM=3,SHIFT=1)      ! 212

C Now divide by the singular point coefficient


        NewField = NewField / (A(:, :, :, 1) + i)              ! Add 'i' first

C Only update DeltaField, and Field for the central grid points.

        WHERE (Central)

C Update the central elements in the difference array

          DeltaField(:, :, 1:NZ) = NewField - Field

C Copy New Field to the Old Field

          Field = NewField

        END WHERE

C Done

        RETURN
        END
```

Appendix C

Task Farm Code Excerpts

C.1 Taskfarm.F

```
C taskfarm.F
C Initialise the task farm and issue tasks to the main program
C Kenneth MacDonald 1994

#define SIZEOF_PARAM_STRUCT 4

        PROGRAM AUTO2DFARM

        IMPLICIT NONE

C Parameters

        INTEGER MAX_JOBS
        PARAMETER (MAX_JOBS = 256)

C Include files

        INCLUDE 'chimp.inc'
        INCLUDE 'pul-tf.inc'
        INCLUDE 'pul.inc'


C Common Blocks

        INTEGER NumJobs
        INTEGER Job
        COMMON /JobsInfo/ NumJobs, Job

C PUL-TF Variables

        INTEGER thisproc
        INTEGER retval
        INTEGER farm
        INTEGER mode
        INTEGER status

        EXTERNAL TFFFN(maketask)
        EXTERNAL TFFFN(dotask)
        EXTERNAL TFFFN(processresult)

C External functions
```

```
        INTEGER GetNumJobs

C Code

        thisproc = chpinit ()
        retval = PULsetDebug (MODULEALL, DEBUGALL)
        retval = PULinit ('auto2dfarm', CHPWILD)
        farm = TFinit ('auto2dfarm')

        IF (chpexport () .NE. CHPOK) THEN
          PRINT *, 'Error: chpexport failed'
          retval = chpexit (0)
        END IF

#ifdef MASTER
        status = TFopen (farm, TFSRCSNK, SIZEOF_PARAM_STRUCT,
     &    SIZEOF_PARAM_STRUCT)
#else
#ifdef WORKER
        status = TFopen (farm, TFWRK, SIZEOF_PARAM_STRUCT,
     &    SIZEOF_PARAM_STRUCT)
#else
        status = TFopen (farm, TFSRCWRKSNK, SIZEOF_PARAM_STRUCT,
     &    SIZEOF_PARAM_STRUCT)
#endif
#endif
        IF (status .NE. TFOK) THEN
          PRINT *, 'Error: TFopen failed with error ', status
        ELSE
          PRINT *, 'TFopen succeeded'
        END IF

        mode = TFquery (farm)
        IF (mode .LT. 0) THEN
          PRINT *, 'Error: TFquery failed with error ', mode
        ELSE
          PRINT *, 'TFquery succeeded. Returned ', mode
        END IF

        IF (mode.EQ.TFSRC.OR.mode.EQ.TFSRCSNK.OR.mode.EQ.TFSRCWRK) THEN
          NumJobs = GetNumJobs ()
          PRINT *, 'Number of Jobs found: ', NumJobs
          Job = 1
        END IF

        status = TFoperate (farm, TFFFN(maketask), TFFFN(dotask),
     &    TFFFN(processresult))
        IF (status.NE.TFOK) THEN
          PRINT *, 'Error: TFoperate failed with error ', status
        ELSE
          PRINT *, 'TFOperate succeeded'
        END IF

        status = TFclose (farm)
        IF (status.NE.TFOK) THEN
          PRINT *, 'Error: TFclose failed with error ', status
        ELSE
          PRINT *, 'TFclose succeeded'
        END IF

        retval = PULexit ()
        retval = chpexit (0)

        STOP
        END
```

```
C ********************************************************************

      SUBROUTINE maketask (taskout, outlength, retval)

C Arguments

      INTEGER taskout
      INTEGER outlength
      INTEGER retval

C Common Blocks

      INTEGER NumJobs
      INTEGER Job
      COMMON /JobsInfo/ NumJobs, Job

C Code

      IF (Job.GT.NumJobs) THEN
        retval = 0
      ELSE
        taskout = Job
        Job = Job + 1
        retval = SIZEOF_PARAM_STRUCT
      END IF

      RETURN
      END

C ********************************************************************

      SUBROUTINE dotask (taskin, inlength, taskout, outlength, retval)

C Arguments

      INTEGER taskin, inlength
      INTEGER taskout, outlength
      INTEGER retval

C Code

      taskout = taskin

      CALL AUTO2D_Main (taskin)
      PRINT *, 'Calling Job Number ', taskin

      retval = SIZEOF_PARAMETER_STRUCT

      RETURN
      END

C ********************************************************************

      SUBROUTINE processresult (taskin, taskout, retval)

C Arguments

      INTEGER taskin
      INTEGER taskout
      INTEGER retval

C Code

      PRINT *, 'auto2dfarm: Job ', taskin, ' completed.'

      retval = 1
```

```
RETURN
END
```

C.2 auto2d-main.FCM

```
C       ==================================================
C       Auto2D
C       Original code by Helena Poll
C       Task farm and additional options by Kenneth MacDonald
C       ==================================================

#ifndef TASK_FARM /* Only have a main program if not for the task farm */
        PROGRAM AUTO2D

        CALL AUTO2D_MAIN (1)

        STOP
        END
#endif  /* TASK_FARM */

        SUBROUTINE AUTO2D_MAIN (ToDoJobNumber)

        IMPLICIT NONE

C       Include any inc files here

        INCLUDE 'limits.inc'

C       Common blocks

        .
        .
        .

C       Arguments

        INTEGER ToDoJobNumber   ! Worker process's job in taskfarm

C       Variable Declarations

        .
        .
        .

C       Start the program!

        .
        .
        .

C       Loop over the number of models, solving each one

        DO Model = 1, NumModels

           OneDModel = .FALSE.  ! Always assume 2D initially

           .
           .
           .

C       Loop over number of periods for this model

           DO PerLoop = 1,NumPeriods

              .
              .
              .

#ifdef TASK_FARM
              IF (ThisJob.EQ.JobNumber) THEN
```

```
#endif
                  Period = Periods(PerLoop)

C       Generate a grid

                  IF (OwnGrid) THEN
                     CALL ReadGrid ('grid.mod', Period, YGrid, ZGrid, ResGrid,
        1                 CY, DY, CZ, DZ, Resistivities, NumBlocks, Sites,
        2                 SiteNums, NumSites)
                  ELSE IF (Stretch) THEN
                     NY = Sizes(PerLoop)
                     CALL StretchGrid (CY, CZ, DY, DZ, YGrid, ZGrid,
        1                 ResGrid, LeftEnd(PerLoop), RightEnd(PerLoop),
        2                 Resistivities, NumBlocks, Sites, SiteNums,
        3                 NumSites, Period, OneDModel, Interactive,
        4                 Granularities(PerLoop))
                  ELSE
                     CALL AutoGrid (CY, CZ, DY, DZ, YGrid, ZGrid,
        1                 ResGrid, LeftEnd(PerLoop), RightEnd(PerLoop),
        2                 Resistivities, NumBlocks, Sites, SiteNums,
        3                 NumSites, Period, OneDModel, Interactive,
        4                 Granularitiss(PerLoop))
                  ENDIF

                  .
                  .
                  .

C       Solve for this model

                  IF (.NOT.OneDModel) THEN
                     CALL MATRIX (YGrid, ZGrid, ResGrid, Field,
        1                 Polarisation, MaxGrid)

                     .
                     .
                     .

C       Write results

                     .
                     .
                     .

                  END IF

#ifdef TASK_FARM
               END IF            ! (ThisJob .EQ. JobNumber)
#endif

            END DO               ! Period Loop

         END DO                  ! Model Loop

C        Finished

         RETURN

         END
```