|epcc|

THE UNIVERSITY OF EDINBURGH

EPCC-SS-98-03

Application of Conjugate Gradient Methods to Large Satellite Magnetic Datasets

Adam Carter

Abstract

The aim was to write programs in Fortran and MPI to perform the technique of Downward Continuation, which is used to determine the magnetic field at the Earth's surface from data describing the field at the Satellite's altitude. The process involves solving a large set of linear equations. The equations are written in matrix form; the matrix being stored using compressed row storage and a parallel conjugate gradient algorithm is then used to solve these equations. The code was developed using test data and runs successfully on this dataset.

Contents

1 Introduction **3**

2 Background **4**
2.1 The Conjugate Gradient Algorithm 4
2.1.1 The General Idea 4
2.1.2 The Algorithm Itself 4
2.1.3 Relevance to the Problem 4
2.1.4 Conjugate Gradient Versus Bi-Conjugate Gradient Methods 5
2.1.5 Sparse Matrices and CRS Form 5
2.2 Parallelising the Problem 6
2.2.1 How the Problem can be Broken Up 6
2.2.2 Difficult Areas 6
2.2.3 Speedup 6

3 The Effect of Preconditioning **7**
3.1 Jacobi Preconditioning 7

4 A Discussion of the Code **9**
4.1 The Process 9
4.2 Notes on Development 10
4.2.1 The Matrix Pre-processor 10
4.2.2 The Parallel Conjugate Gradient Solver 10

5 Results- A Downward-Continued Solution **11**

6 Further Work **12**
6.1 The Code 12
6.2 The Results 12

7 Conclusion **12**

1 Introduction

This project is concerned with the determination of the magnetic field at the Earth's surface from data provided by satellites. In particular, we are interested in the magnetic anomaly field – the part of the Earth's field that remains once the effects of the geodynamo processes, which occur deep inside the earth, have been removed. The anomaly field is due to the magnetisation of the Earth's near-surface rocks. There are both advantages and disadvantages to using satellite data to determine this field. On the one hand, the satellites are able to provide global results which cannot be obtained from aeromagnetic data, but on the other, the images they provide are relatively blurred and imprecise. This can be treated in two ways:

- Downward continuation of the field to the Earth's surface, thereby obtaining a more detailed view, or
- Modelling of the distribution of the magnetisation which causes the anomaly field.

There are in fact many solutions that fit the data, but it is standard practise to adopt the smoothest of these. This assumption means that the two problems have a very similar mathematical formulation, requiring the solution of a large system of linear equations, or equivalently, a matrix-vector equation. This is where the computational effort is required in order to solve the problem. With such a large system it is impractical to obtain an exact solution, but the matrix is sparse, symmetric and positive definite and these properties enable the approximate numerical solution to be found relatively efficiently.

The primary goal of this project was to write parallel code to perform the conjugate gradient algorithm in parallel and use it to solve these equations. The problem is simplified by the fact that the matrices involved in the matrix-vector equations (equivalent to the set of linear equations) are sparse. Although there are only a few elements which are exactly zero, it is possible to neglect elements smaller than a certain threshold with little effect on the solution, especially for larger datasets. This approach allows the matrices to be efficiently stored in Compressed Row Storage (CRS) format, hence reducing storage and calculation requirements but complicating matrix multiplication slightly. Furthermore, the matrices are both symmetric and positive definite which means that the Conjugate Gradient method is guaranteed to converge on the solution, at least in the absence of computational rounding errors.

There have been several different approaches used to tackle the downward continuation problem in the past which have worked satisfactorily on continental-sized areas, but they have not been able to provide good global solutions. One main aim of this project was to use a parallel computer to obtain these solutions.

2 Background

2.1 The Conjugate Gradient Algorithm

2.1.1 The General Idea

The Conjugate Gradient Algorithm is an iterative process used to solve large sets of linear equations. It works its way through the space of possible solutions in such a way that every search direction is orthogonal to every previous search direction. It is this property that gives the upper bound on convergence time. Furthermore, each step of the search gives a result which is closer to the true solution.

2.1.2 The Algorithm Itself

The following is a description of the Conjugate Gradient Algorithm, based on that given by Barrett $et\ al$ [1]. The approach to preconditioning is slightly different to that given in [1] and so the following algorithm is an unpreconditioned version. Preconditioning is discussed later.

The Conjugate Gradient Method

Compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

for $i = 1, 2, ...$

$\rho_{i-1} = (r^{(i-1)})^T r^{(i-1)}$

if $i = 1$

$p^{(1)} = r^{(0)}$

else

$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$

$p^{(i)} = r^{(i-1)} + \beta_{i-1} p^{(i-1)}$

end if

$q^{(i)} = Ap^{(i)}$

$\alpha_i = \rho_{i-1}/(p^{(i)})^T q^{(i)}$

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

check for convergence; continue if necessary

end

2.1.3 Relevance to the Problem

As has already been mentioned, the problem of finding the magnetic anomaly field at the Earth's surface reduces to that of solving a large set of linear equations, or equivalently, the matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{1}$$

where **A** is a very large, sparse, symmetric, positive definite matrix, whose elements are functions of the position of the satellite alone, and **b** is a vector representing the magnetic field at orbit-height. The fact that the matrix is symmetric and positive definite means that the problem is well suited to a conjugate gradient approach.

2.1.4 Conjugate Gradient Versus Bi-Conjugate Gradient Methods

When I originally set about this project, it had been proposed that the best way of obtaining a solution was to use the biconjugate gradient method. This method is useful because it will always converge, even when the matrix is not symmetric. However, the convergence is slow, and can be difficult to predict. When it was clear that the matrices were exactly symmetric and positive definite I decided to use conjugate gradient methods because (a) they require fewer calculations and (b) they are guaranteed to converge in a number of iterations which is less than n where the dimensions of the matrix are $n \times n$. In fact, it turns out that the performance is better than this.

2.1.5 Sparse Matrices and CRS Form

Because the matrices in question are so large, they are too big to store, even on most high performance computers such as the Cray T3D and the Cray Y-MP. It was necessary to store and process the matrices in some compressed form. This is facilitated by the sparse nature of the matrix involved. I chose to use $Compressed\ Row\ Storage$, or CRS, form, which stores the matrix as three arrays; an array of non-zero elements, an array of column indices and an array of row indices. The element array has the same number of elements as there are non-zero elements in the array and they are listed in order from right to left and then top to bottom. The array of column indices is of the same length and stores the column of the matrix in which the corresponding element in the first array could be found. Finally the third array has one more element than there are rows in the matrix. Its elements consist of the positions in the first array of the first element in each row of the matrix. The final element is one more than the number of non-zero elements in the matrix ($i.e.$ the value which would come next had the matrix had another row). This is probably best illustrated by an example: Consider the matrix

$$\begin{bmatrix} 4 & 0 & 2 & 3 \\ 1 & 0 & 0 & 5 \\ 6 & 3 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 7 & 0 & 0 \end{bmatrix} \tag{2}$$

The first array consists of the non-zero elements and so would be $[4, 2, 3, 1, 5, 6, 3, 1, 7]$. The second array corresponds to the columns which these entries belong in: $[1, 3, 4, 1, 4, 1, 2, 3, 2]$ and finally the row index array would read $[1, 4, 6, 9, 9, 10]$. This example also illustrates what happens if a row of zeros occurs: If the next element in the matrix *would* have been the 9^{th}, as in the above example, this means that the value in both the all-zero row, and the subsequent row would both be '9'.

2.2 Parallelising the Problem

There are several reasons why it is advantageous to parallelise this problem. Firstly, it requires a large amount of memory, even once the matrices are compressed, and secondly, it is a slow process. The calculations need to be performed on large matrices and vectors and furthermore, the process is iterative so that all the calculations have to be performed at each step. Parallelising the problem allows each processor to work on part of the matrix so reducing the memory requirements of any one processor, and speeding up the calculations by reducing the amount of computation which needs to be performed by any one processor at each iteration.

2.2.1 How the Problem can be Broken Up

The program that I wrote basically parallelises the problem by storing and evaluating different rows of the matrix on different processors. It was decided to keep all information about any one row together because otherwise CRS would have complicated matters considerably. I chose to store an equal number of rows on every processor with the possibility that the last processor may have fewer rows to process. However, the redundant processor time is unavoidable in this case and is minimised using this distribution. I chose to use a block distribution of rows to processors, $i.e.$ assigning a contiguous block of rows to each processor, which is the simplest to implement. A very slight performance gain could probably be gained by using a cyclic distribution, as rows in the middle of the matrix tend to have more non-zero entries, but I elected to implement the more straight-forward technique first.

2.2.2 Difficult Areas

The parallelisation is complicated by only a few things. For much of the time, each processor can work independently of all of the others, working on its own section of the matrix and the associated vectors. The need for communication arises only three times in the main part of the algorithm. Firstly, before the matrix-vector multiplication can be performed, it is necessary to gather on to all processors the vector in question. This is done using the MPI call MPI_ALLGATHER. The other occasion is at the 2 dot products. This is done by performing local dot products and then using MPI_ALLREDUCE with MPI_SUM to obtain the global dot product.

2.2.3 Speedup

Figure 1 shows a speedup curve for the code which performs the parallel conjugate gradient algorithm. In order to clearly illustrate the advantage of this method over serial solutions, I have omitted the time taken for I/O in these calculations as this more time consuming area is generally common to any method of solution

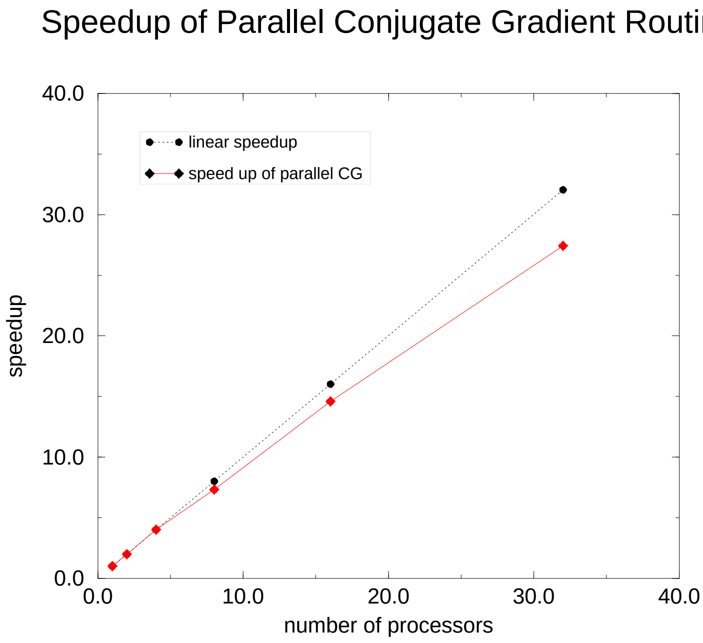

Figure 1: A Speedup Curve - The Effect of Parallelism

3 The Effect of Preconditioning

Preconditioning is a way of improving convergence. Earlier attempts to use CG methods for downward continuation adopted a preconditioning technique which destroyed the symmetry of the matrix **A**. For this reason, it was necessary to use the Biconjugate Gradient (BiCG) method. We originally planned to implement the technique below, but it was abandoned because it requires complex arithmetic and the need to store all numbers with a real and imaginary part results in the program requiring double the memory. Based on observations of small dataset runs, it was also decided that the program would converge suitably quickly even on the large dataset, although this assumption is yet to be tested. I have included the following description of the method which is already partially implemented in the code and it is thought that the change to complex variables and the implementation of a complex dot-product routine would be sufficient to get the program working with this preconditioning. In the mean time, the parameter *topreco* should be set to `.FALSE.`.

3.1 Jacobi Preconditioning

Take the symmetric matrix **A**. If it is postmultiplied by some matrix **M** and premultiplied by the transpose of this matrix $\mathbf{M}^T$, it will retain its symmetry since

$$(\mathbf{MAM^T})^T = (\mathbf{AM^T})^T\mathbf{M^T} = \mathbf{MA^TM^T} = \mathbf{MAM^T} \tag{3}$$

Jacobi preconditioning involves setting **M** to:

$$\begin{bmatrix} \lambda_1^{-1/2} & 0 & 0 & 0 \\ 0 & \lambda_2^{-1/2} & 0 & 0 \\ 0 & 0 & \ddots & 0 \\ 0 & 0 & 0 & \lambda_n^{-1/2} \end{bmatrix} \tag{4}$$

which has the effect of setting all diagonal entries in the matrix to one, hence lowering its condition number. The square roots lead to the introduction of complex numbers.

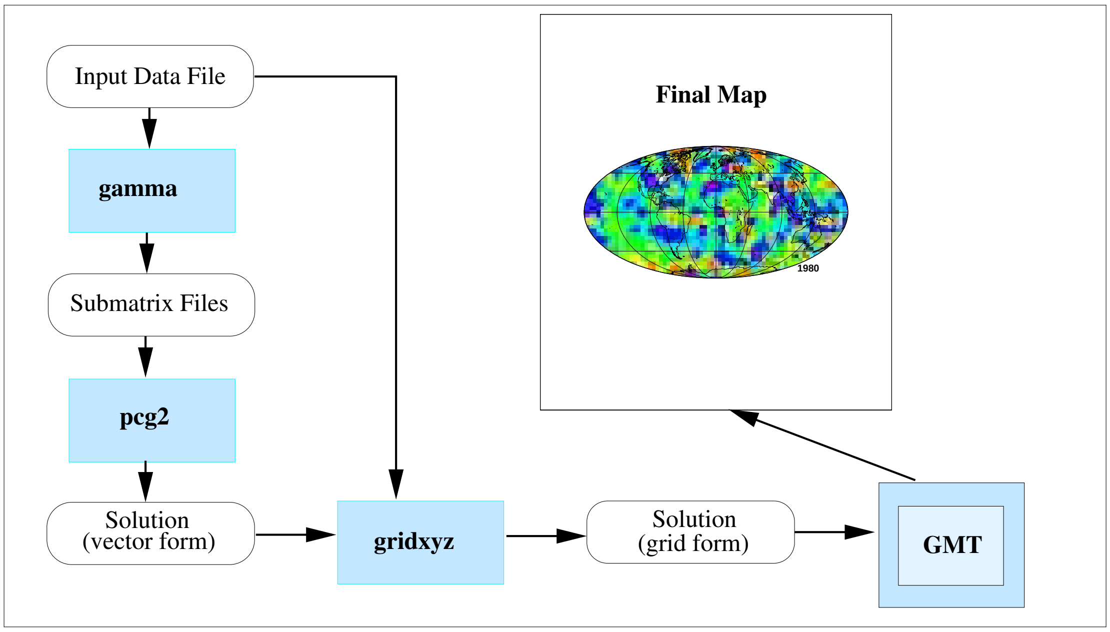

Figure 2: The Process involved in turning satellite data in to a map of the anomaly field at the Earth's surface

4 A Discussion of the Code

4.1 The Process

Figure 2 is a representation of the process of obtaining a map of the anomaly field from the original satellite data.

- The original input file contains the processed satellite data. It is an input file consisting of five columns, the first two containing the longitude and latitude of the satellite and the other three containing the anomaly field in each of three orthogonal directions.
- The input files are read by the matrix pre-processor, $gamma$. This is a serial programme which still requires high performance computers due to its memory requirements. It pre-processes the datafile, producing the matrix, in a compressed format, and then splits it in to smaller submatrix files.
- These smaller files each hold information about blocks of rows of the matrix. There is one file for each processor used in the following program.
- These files are the input to the parallel conjugate gradient solver, $pcg2$. This is the 'clever bit'; it's the program which performs the CG algorithm, in parallel, on the matrix, while it remains stored in a compressed format,
- The output from pcg2 is the solution, in vector form.
- This is combined with the original datafile, to match the solution to the grid points.
- The resulting grid is processed by a set of Generic Mapping Tools, GMT, which produces the final results in an easily readable format.

4.2 Notes on Development

4.2.1 The Matrix Pre-processor

Although some of the code for this part of the process was provided, it turned out to be as difficult as the conjugate gradient solver itself to work with. It is this program which is used to make the matrix **A**. The number of elements from the original matrix which can be retained is ultimately limited by the memory of the machine which this program is being run on. Even on the Supercomputers we have been using, it is only possible to retain the top one or two percent of the elements of the matrix. However this should be enough to obtain a result with the desired accuracy. Memory limitations meant that this program was often difficult to develop. The test data which was provided could be run quickly, however moving up to a larger dataset introduced more problems than were immediately obvious. Firstly, the program as it originally stood took longer than 6 hours to run, when it was producing a large amount of datafiles. It was therefore only possible to run in the weekend queue. A slight adaptation allowed this program to run overnight. Also, the number of elements in the matrix which are retained, is controlled by a threshold under which entries are approximated as zero and so it is not possible to chose exactly the number of non-zero elements which are going to appear in any subfile. I later found out that a program already exists which could determine the total number of non-zero elements obtained at any threshold. I should really have asked for that sooner.

4.2.2 The Parallel Conjugate Gradient Solver

It was able to implement this part of the code without many difficulties. There were some areas which initially caused me problems, notably the transition between testing on 8 and 16 processors. Although much of the code was written on 8 processors I had a bug which only became apparent when I ran on a number of processors which did not divide exactly in to the number of rows in the matrix. The solution to this problem and the resulting generalisation to any number of processors complicated slightly communication between processors, because the MPI function MPI_ALLGATHER which is used at several points in the code expects the same amount of data from each process when in fact, the last processor often deals with a smaller number of rows. In the end a relatively simple solution was found which kept the code concise, with the trade-off that slightly more information is communicated than is actually ever used in the program.

When developing this program I also had to decide how I was going to get the matrix pre-processor to communicate its results to the conjugate gradient solver. I decided to use a set of output files which could, in theory, be read in in parallel. In practise, I believe there is a limit to how many can be read in simultaneously, and this may explain why the conjugate gradient program has a bottleneck at its I/O stage.

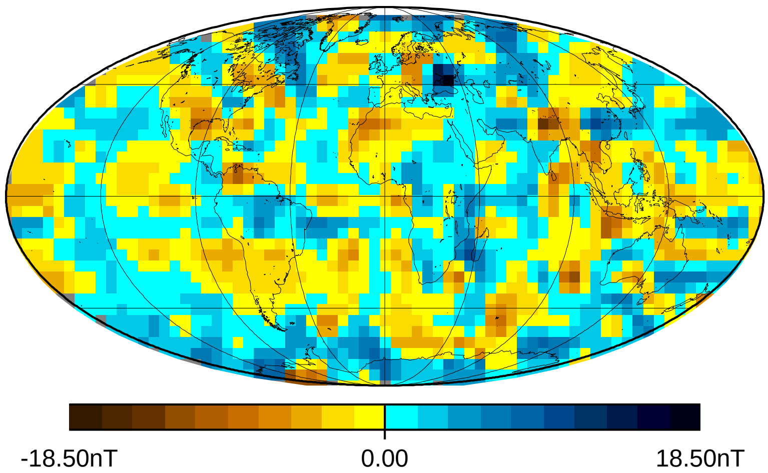

Figure 3: Field data at satellite altitude

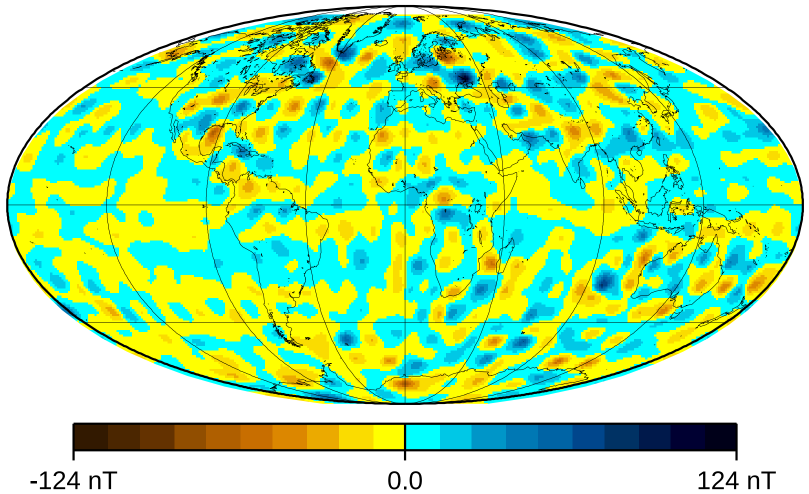

Figure 4: Field at Earth's Surface determined by Downward Continuation

5 Results- A Downward-Continued Solution

This section illustrates the final results obtained using the programs `gamma`, `pcg2`, `gridxyz`, and GMT on the test data with which these programs were developed. Figure 3 shows the vertical (z-) component of the anomaly field at the satellite altitude (400km). This can be determined straightforwardly from the input data files. Figure 4 shows the z-component of the anomaly field at the Earth's surface, which was obtained using the process described in section 4. The program can also be used to display the other components of the field; x-, y-, and z- components can be obtained as well as the component which lies along the direction of the Earth's main magnetic field.

6 Further Work

6.1 The Code

There are several areas of this project which suggest possible further work. At the moment, the program's I/O still forms a slight bottleneck. A possible solution would be to use MPI I/O or to combine the programs `gamma` and `pcg2`. Also, this project concentrated mainly on the parallelisation of the main linear equation solver and it is possible that there might be stages in the pre-processing program which can be parallelised also. The programs as they stand can be easily modified to produce field data as well as magnetisation. It is necessary to deal with vector quantities instead of scalars in some cases but otherwise the problem is almost equivalent.

6.2 The Results

The results included in this report are results obtained using a test dataset which is smaller than that which will eventually be used. The program has been tested on a full-size real-life dataset. This would be the obvious next step in the project and when a full data set becomes available, it would be interesting to see the results the program can give. Also, with the results obtained from a full-sized dataset it would be interesting to examine correlation lengths.

7 Conclusion

In this project, code has been produced which provides a relatively efficient parallel solution to a real-life problem in geomagnetism. The programs have been successfully run on a set of test data and most of the problems with dealing with a full sized dataset have been overcome. The results obtained from the test data agree closely with the results obtained from the technique of Cholenski decomposition and back substitution, a technique which is less adaptable for use with large datasets.

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. Van der Vorst. *Templates for the Solution of Linear Systems: Building blocks for Iterative Methods*. SIAM, Philadelphia, PA, 1994.

[2] R. D. da Cunha and T. Hopkins. *PIM 2.0 The Parallel Iterative Methods package for Systems of Linear Equations*.

[3] M. E. Purucker, T. J. Sabaka, and R. A. Langel. Conjugate gradient analysis: A new tool for studying satellite magnetic datasets. *Geophysical Research Letters, Vol.23, No.5, pp. 507-510*, March 1996.

[4] K. A. Whaler. Downward continuation of magsat lithospheric anomalies to the earth's surface. *Geophys. J. Int., Vol.116, pp.267-278*, 1994.

About the author: I am a student of the University of Edinburgh where I have been studying for the an MPhys Computational Physics degree for the last four years. I will complete the course in Summer 1999.

Acknowledgements: I would like to thank all of my supervisors who helped me over the course of this project:

- Professor Kathy Whaler, Department of Geology and Geophysics, The University of Edinburgh
- Dr Doug Smith, Edinburgh Parallel Computing Centre
- Dr Rob Baxter, Edinburgh Parallel Computing Centre

My thanks to Mike Puruker at Goddard Space Flight Center, NASA for his help in providing datasets used in the project. Thanks also to the rest of the EPCC staff who I pestered for ideas and assistance at several points during the programme.