EPCC-SS99-03

Calculating the Earth's Magnetic Field using OpenMP

Josephine M Beech-Brandt

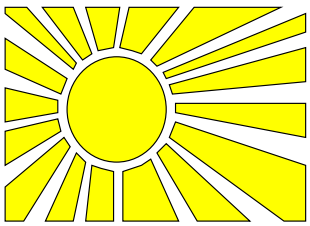
University of Edinburgh

Abstract

The magnetic field of the lithosphere has been computed, using data collected from the Magsat satellite mission and the technique of Downward Continuation.

The aim of this project was to take existing code which used the Message Passing Interface (MPI) and convert it into an OpenMP version and then to further develop this code by adding a preconditioner and investigating the effect on behaviour of a damping factor.

The work was completed as part of the ten week Summer Scholorship Programme at the Edinburgh Parallel Computing Centre (EPCC).



Contents

1 Introduction

The Magsat satellite mission was the first satellite mission that measured both the magnitude and the direction of the geomagnetic field. The measurements contain information about all of the magnetic fields generated within the earth such as the field generated within the Earth's liquid core by dynamo processes.

This project is only concerned with the magnetic field generated by the Earth's lithosphere (its crustal outer layer) and so the data is preprocessed removing the unwanted field elements and then fed into the programme to be downward continued (using conjuagte gradient to invert the large matrix derived from the satellite data) to the Earth's surface, thereby obtaining the actual magnetic field due to the lithosphere.

EPCC have just taken delivery of a shared memory machine and the project was to convert an existing parallel programme which uses the Message Passing Interface (MPI) into a parallel programme which uses OpenMP directives.

Once the conversion was completed the code was further developed by adding a preconditioner and a damping factor whose effect on convergence was then investigated. This was done because the Conjugate Gradient method does not converge particularly well for this problem and preconditioning has proved to be successful in improving convergence in the past.

The code was developed using a test dataset which is comprised of 360 data points evenly spread over the globe. The real dataset which is comprised of 11560 satellite points together with the three components of the lithospheric magnetic field was then used. The linear system which thus has to be solved consists of 34680 unknowns (matrix of size 34680×34680). Previously this dataset had not converged but with the addition of the damping factor will converge and with the addition of a preconditioner will converge faster.

2 Background Theory of Techniques Used

2.1 Conjugate Gradient

Conjugate Gradient is a well-established iterative method to solve systems of linear equations, $Ax = b$ where A is a postive definite symmetric matrix The method produces successively closer approximations to the solution, bu choosing a search direction mimimising the residual along this direction. Each search direction is orthogonal to every previous search direction. Although the length of the sequences can become large, one of the method's principle benefits is that it has short recurences i.e. only requires the immediately prior vectors. In terms of the project's dataset this is an important feature, due to the large size of the matrix and vectors involved.

As each search direction is orthogonal to each other, the limiting factor on the number of iterations is the number of orthogonal search directions which is N for an N x N matrix. Thus, it is guaranteed to converge within N iterations for an N x N matrix (in exact arithmetic).

The Conjugate Gradient algorithm is outlined below based upon that given by [1]. This is the unpreconditioned version.

The Conjugate Gradient Algorithm

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

for $i = 1, 2, \ldots$

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

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

2.2 Storage of Matrices

The storing of a sparse matrix, one where many elements are zero, is most efficient when its zero elements are not stored. Instead, all non-zero elements are stored continguously and there are various schemes which enable the indexing of these values.

Compressed Row Storage is one such scheme. It assumes no structure of the matrix and so is very general. This is needed for this project as the structure of the matrix is unknown *a priori*.

The matrix is stored using three arrays, one which stores the actual non-zero values, one which stores the column index of each non-zero value and finally one which stores the row index pointer. This is the number of previous non-zero elements plus one in the first column of each row. Below, is an example which makes it easier to see.

An example of Compressed Row Storage

$$\begin{pmatrix} 1 & 3 & 0 & 5 \\ 0 & 4 & 2 & 0 \\ 2 & 0 & 1 & 3 \\ 4 & 2 & 3 & 1 \end{pmatrix}$$

In this example the three arrays are

non-zero elements [1,3,5,4,2,2,1,3,4,2,3,1]

column index [1,2,4,2,3,1,3,4,1,2,3,4]

the row index pointer [1,4,6,9,13].

This method of storing saves a lot of memory space especially for very sparse matrices. Instead of requiring n^2 storage locations as in traditional matrix storage, this method only requires $2nnz + n + 1$ storage locations for an $n \times n$ matrix with nnz being the number of non-zero elements. Obviously, for a large sparse matrix this is a tremendous saving on memory space. For example, the matrix used this project is 34680×34680 which requires 1202702400 storage locations using a traditional matrix storage method.

However, due to the sparse nature of the matrix, the number of non-zero elements [1] is 14526526 (with the threshold value set to 0.8). Using these values, it is easy to calculate the saving made in storage locations.

Full Matrix Storage : n^2 = 1202702400

Compressed Row Storage : $2nnz + n + 1$ = 29087739

Saving : = 1173614661

Clearly, the benefit can be seen from this example. However, it must be noted that due to this storage mechanism, vector and matrix multiplications become more complicated to implement.

2.3 Preconditioning

A preconditioner effects a transformation on another matrix in order to change its spectral properties into something more favourable for iterative methods. For example, the system $Ax = b$ has the same solution as the transformed system $M^{-1}Ax = M^{-1}b$ but the spectral properties of $M^{-1}A$ may be more favourable and thus will converge with less iterations.

However, there is a cost-trade off with all preconditioners. The time required to set the preconditioner up and then the extra cost per iteration needed need to be put against the resulting improvement in convergence.

2.3.1 Point Jacobi Preconditioner

The Point Jacobi Preconditioner consists of just the diagonal of the matrix. In this project, the preconditioner was comprised of the inverse of the square root of the diagonal elements. This has the effect of transforming all of the diagonal elements of the matrix to one after pre and post multiplication.

[1] In the programme there is a threshold below which all values are treated as zero, thus making the matrix sparse

The preconditioning matrix is shown below.

$$\begin{pmatrix} \frac{1}{\sqrt{\lambda_1}} & & & \\ & \frac{1}{\sqrt{\lambda_2}} & & \\ & & \ddots & \\ & & & \frac{1}{\sqrt{\lambda_N}} \end{pmatrix}$$

Pre and Post multiplying keeps the matrix symmetric which is necessary for Conjugate Gradient. There is also little overhead involved for this method.

It involves computing the diagonal elements which are then stored in diagarray and to scale the elements of the matrix.

Below, is the part of the code where the preconditioning takes place. The first loop involves multiplying the sparse matrix (gramsp) by the relevant diagonal element
i.e. $A_{i,j} = diagarray_j * diagarray_i * A_{i,j}$

!doing the preconditioning multiplication

```
kount=0
do i=1,n+1                                  !the row number}
   do j=ia(i), ia(i+1)-1                    !the values along the row}
      gramsp(kount)=diag_array(ja(kount)) * diag_array(i) * gramsp(kount)
      kount=kount+1
   end do
end do
```

A similar process is carried out on the input data to the conjugate gradient subroutine. This is more straight forward as there is no compressed row storage format involved.

!do the same preconditioning on data input to conjugate gradient

```
do i=1,n
   data(i)= diag_array(i) * data(i)
end do
```

Finally, once the solution has been found, it has the preconditioning factor removed from it, thus giving true the true solution.

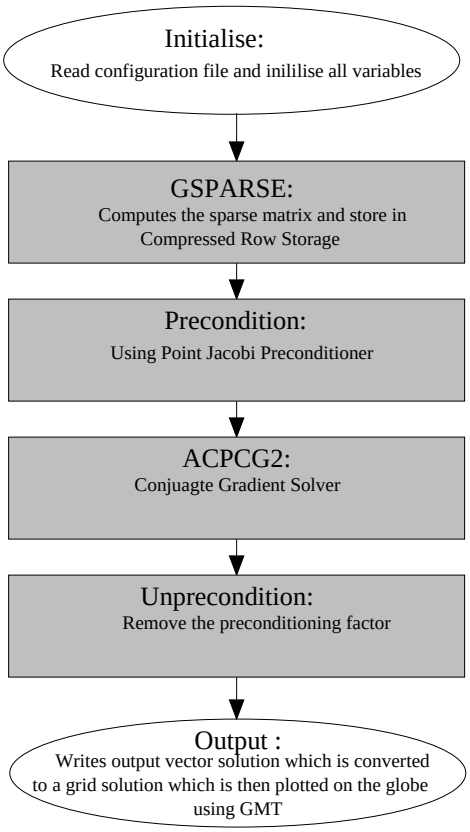
!removing the preconditioning factor

```
do i=1,n
  x(i)= diag_array(i) * x(i)
end do
```

3 Programme Structure and Parallelisation

The programme is comprised of several subroutines which each have their own specific task to perform.

The basic flow of the programme is represented below. The main areas of parallelisation are the subroutines gsparse and acpcg2 which will be discussed in detail later.

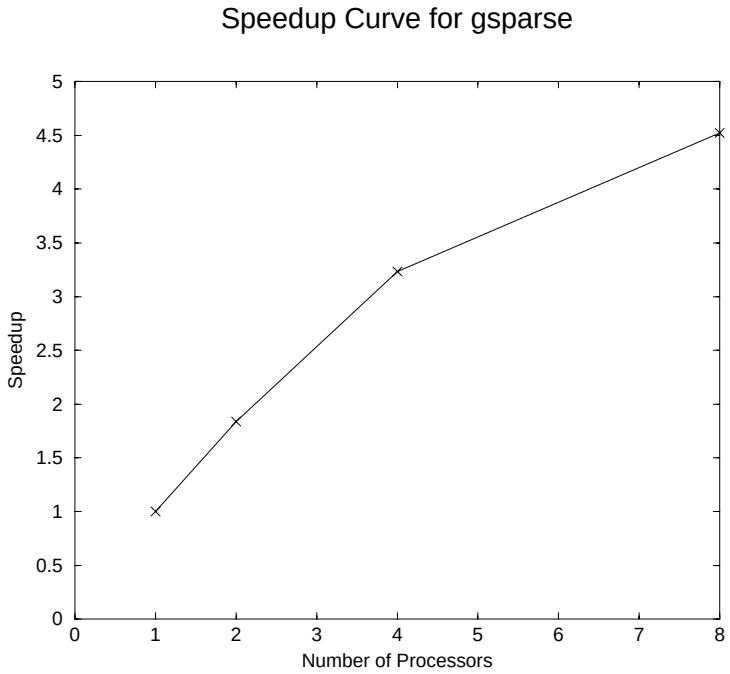


4 Results

4.1 GSPARSE

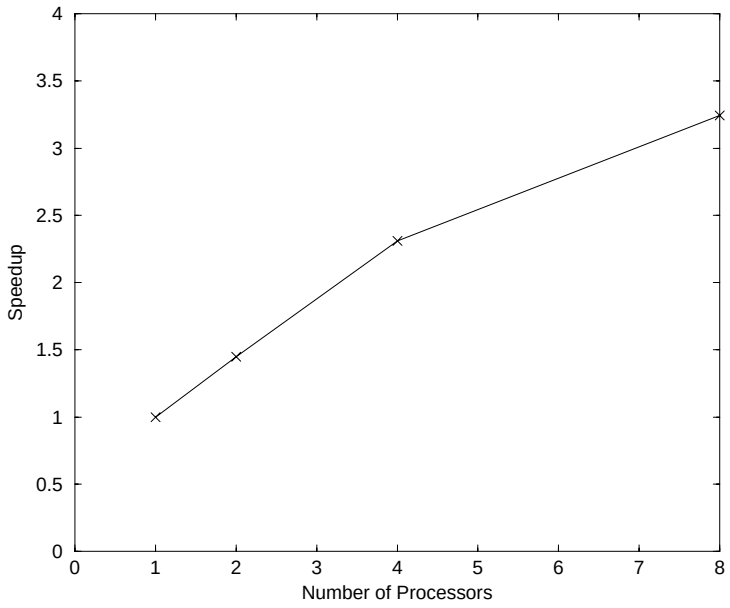
This subroutine produces the matrix and stores it in compressed row storage. It is by far, the most time consuming part of the programme and therefore is the most important aspect for parallelisation.

However, due to the matrix storage system it proved difficult to parallelise. It was necessary to have an OMP SINGLE block in the subroutine which indicates that the block of code is only to be executed by a single thread. This was necessary as the storing of elements in compressed row storage must be executed in sequence to ensure that the matrix retains its original structure. The first thread to reach the SINGLE directive executes the block while all the other threads wait until the block has been executed. Obviously, this causes several threads to be inactive at one time and therefore will not give good speedup results.



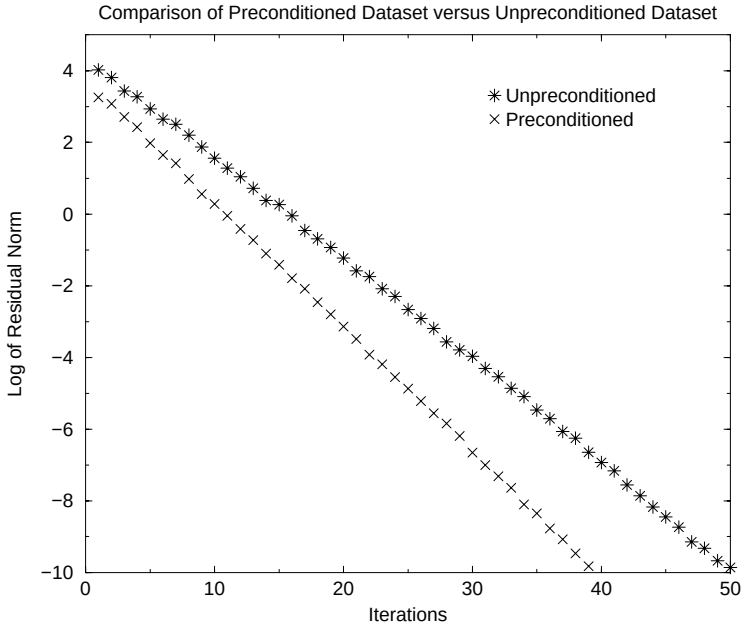
4.2 ACPCG

This subroutine contains the parallel conjugate gradient solver. It takes a low percentage total execution time (about 1.5% for the full dataset with the damping factor set to 200 (see section 4.4)). This subroutine does not give very good speedup results. Various attempts were made to optimise parallel performance, however, they all gave basically the same speedup results.



4.3 The Effect of the Preconditioner

It was found that with the preconditioner included the convergence did in fact require less iterations than without it. Although, this was expected the extent of the increase was not known. On the test dataset this turned out to be 23% of convergence rate.



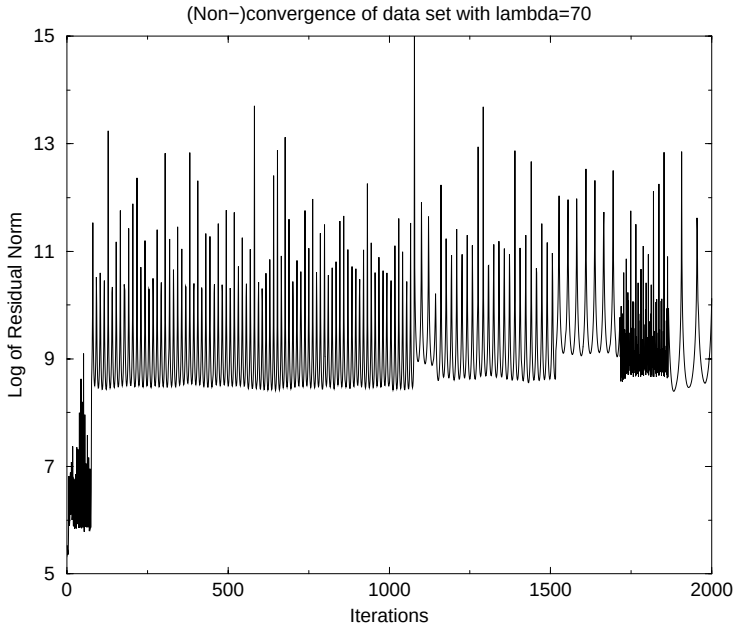
4.4 The Effect of the Damping Factor

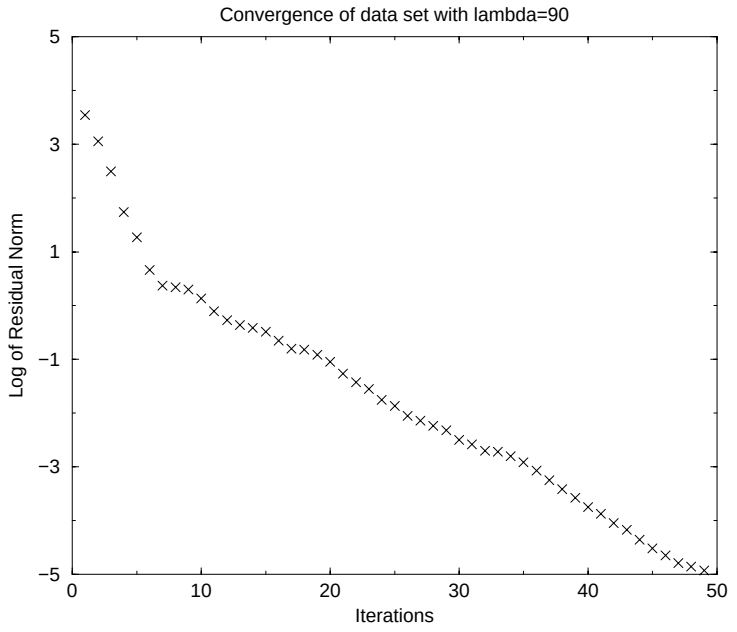
It was found that the full dataset would not converge as it was, so a damping factor, λ was added to each diagonal element. The ideal value for the damping factor is not known and so an experimental trial and error exercise was started.

The equation to be solved is now
$(A + \lambda I)x = b$
which damps out low-lying eigenvalues of the matrix, which can affect convergence.

The converegnce rate is obviously highly dependent on λ and the higher the value of the damping factor the quicker the convergence i.e. with the damping factor set to 5000 it converged within 5 iterations.

However, it was found that the dataset would converge with the damping factor set to 90 but not with it set to 70. When set to 70, the behaviour was very erratic (the run was limited to 2000 iterations).





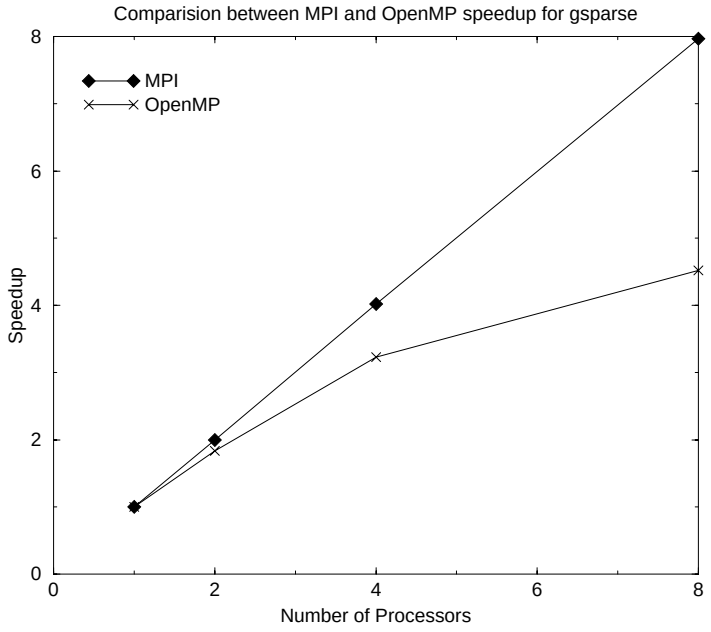
4.5 Comparison with MPI version

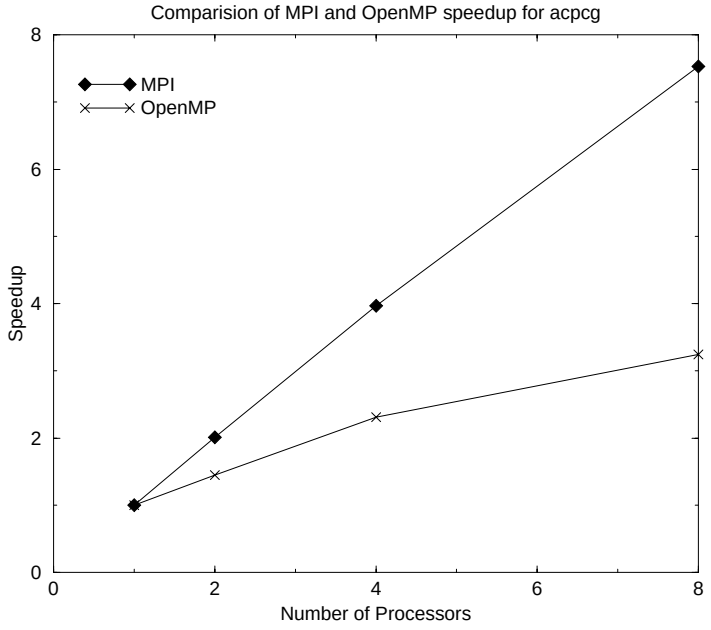
The EPCC have taken delivery of a Sun E3500 which supports both MPI and OpenMP and thus one of the interesting conclusions from the project is the comparison of the fairly new standard of OpenMP and its more well established friend MPI.

The techniques are quite different in their approach (although both essentially aim for the same thing, good speedup results). Programmes which use MPI have to have their data decomposed and given to each processor to work with by the programmer. MPI library routines are then utlilised to gather this information together at the final stage.

The MPI version of gsparse and acpcg2 proceeds in this manner. The programme takes the large matrix and subdivides it into submatrices which are then given individual processors. Each processor works with its own submatrix independently of the other processors. Once all the processors have completed their work the large solution vector is built from the smaller sub solution vectors. Doing it this way has proved to give good speedup results [3].

Below are graphs with both the MPI and OpenMP speedup results for the two subroutines.





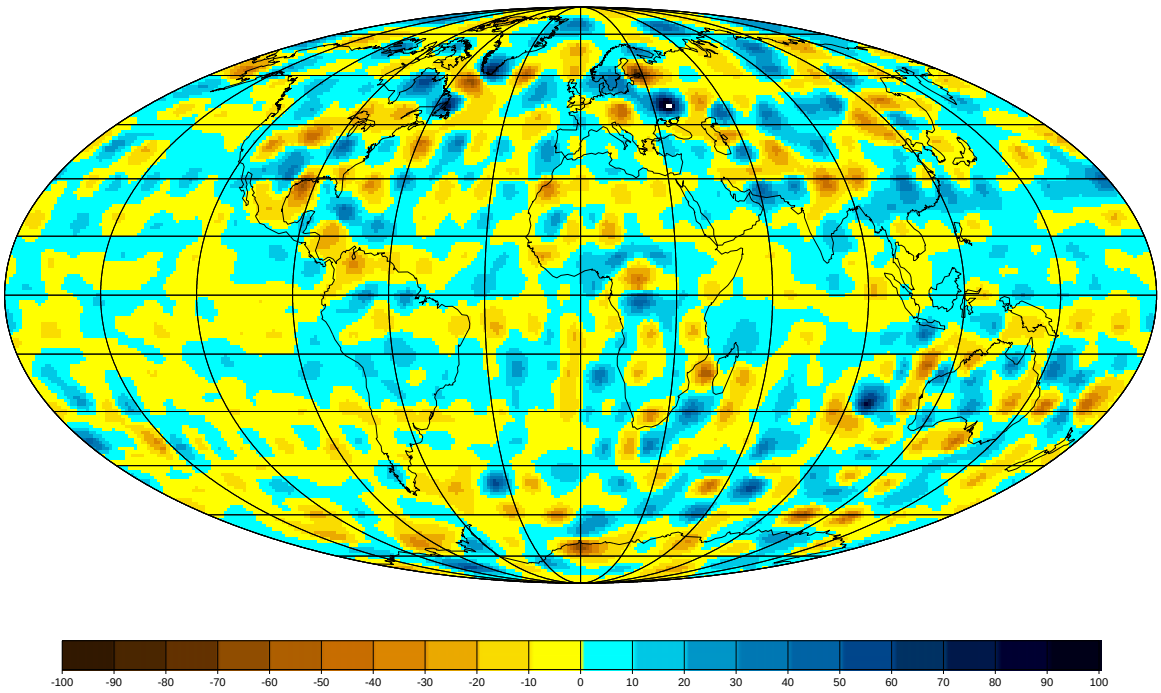
Although, the MPI version clearly scales better than the OpenMP version, it should be noted that the OpenMP directives are easier to implement than the data decomposition required by MPI.

5 GMT Output

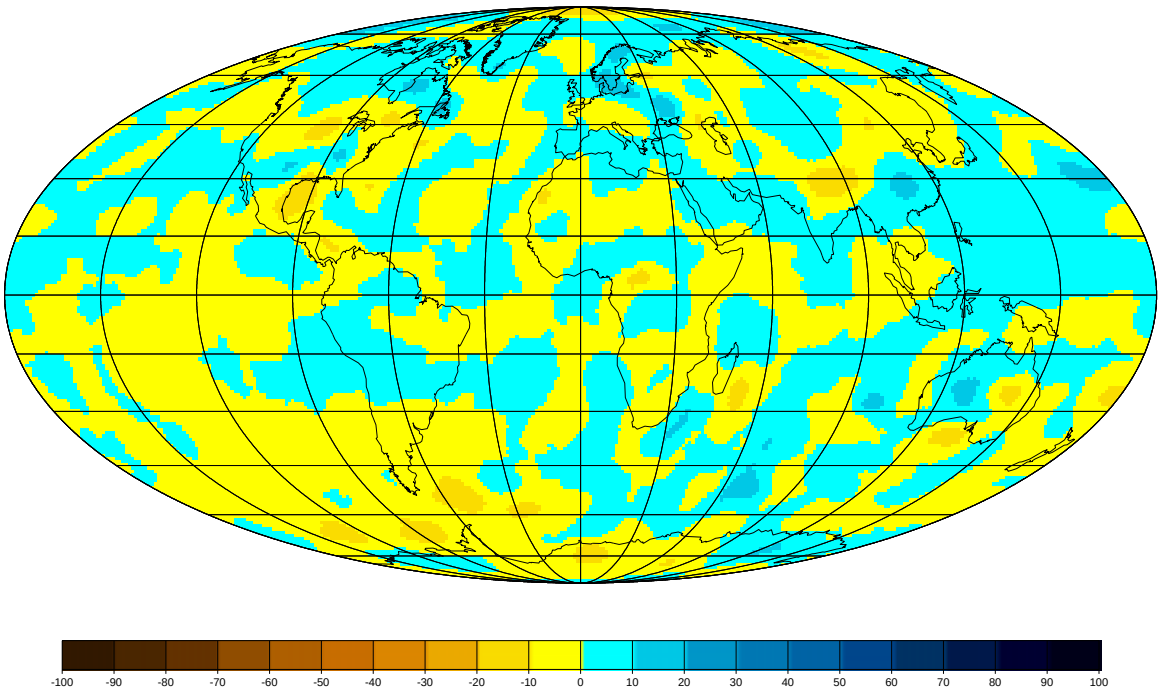
Once the vector solution has been found, this can then be processed through a utility called gridxyz which converts the vector solution into a grid solution. This grid solution can then be processed through the Generic Mapping Tool (GMT) which produces a global image of the dataset. Below, are the data output from both the test dataset and the full dataset with a damping factor set to 90.

Looking at the test dataset, certain known features such as the Bangui anomaly in Central Africa, the highs of the Central Plains in the United States and the anomalies in Europe can be clearly seen. [5] [4].

However, looking at the output for the large dataset these features are not so clear. This is due to the damping factor which has been introduced which obviously affects the data output. It should however be possible to choose a better damping factor and thus produce better, more detailed images. It should be noted that although some of the features are not very prominent, they can still be seen which is encouraging.



Output from GMT for the Test Dataset



Output from GMT for the full Dataset

6 Conclusion

The code was successfully ported to OpenMP and its performace was compared with MPI.

The OpenMp and MPI speedup comparision was disappointing for the OpenMP version but could perhapes be improved upon using a different storage mechanism.

The addition of the preconditioner proved very successful and gave a large improvement in the convergence rate of the two datasets. It was fairly easy to implement once the logic to find only diagonal elements was completed.

The damping factor is still under investigation though it was encouraging to see that the full dataset will converge with the addition of a damping factor.

Due to time constraints, it was not possible to investigate the eigenvalue structure of the matrix although prelimary reading was carried out on a itertive method to find the eigenvalues [2] which may provide insight into the best damping factor.

The project proved to be both an enjoyable and educational experience. It was entered into without any knowledge of parallel computing or geology and finished with some knowledge of both.

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, 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] Burkhardt Bunk, Karl Jansen, Martin Luscher, and Hubert Simma. Conjugate gradient algorithm to compute the low-lying eigenvalues of the dirac operator in lattice qcd. September 1994.

[3] Adam Carter. Application of conjugate gradient methods to large satellite magnetic datasets. EPCC-SS-98-03.

[4] Magnus Hagdorn. Analysis of large data sets. June 1999.

[5] K. A. Whaler. Downward continuation of Magsat lithospheric anomalies to the Earth's surface. *Geophys. J. Int.*, 116:267–278, 1994.

I have just completed my third year of a Computational Physics degree at the University of Edinburgh and am due to graduate in June 2000.

I would like to thanks Dr Douglas Smith (EPCC) for always being available to help and assist.
Also, thanks to Professor Kathy Whaler (University of Edinburgh, Gelogy and Geophysics Department) for her encouragement and advice across the Atlantic.
Finally, a note of thanks to all other EPCC staff members who have worked to make the SSP both a worthwhile and enjoyable experience.