Simulational modeling of time-domain magnetic induction using parallel computational schemes: (II) in the spherical coordinates

Wenchien Chou

Institute of Astronomy and Astrophysics, Academia Sinica, P.O. Box 1-87, Nankang, Taipei 11529, Taiwan. E-mail(WC): chou@asiaa.sinica.edu.tw

Ryoji Matsumoto,

Department of Physics, Faculty of Science, Chiba University, 1-33 Yayoi-Cho, Inage-ku, Chiba 263-8522, Japan

Fumiko Tajima,

Berkeley Seismological Laboratory, University of California at Berkeley, 215 McCone Hall, Berkeley, CA, 94720-4760, USA

Abstract

New parallel codes have been developed to carry out simulational experiments of magnetotelluric investigation in the spherical coordinates by solving the electromagnetic induction equation in the time domain. In our previous paper published in this journal[1] we tested the Cartesian code and show that the code can achieve good performances of parallel processing. The extension to the spherical coordinates is presented here in order to incorporate the global effect of magnetic induction and to avoid the restriction and errors caused by the boundary condition in the Cartesian code. We adopt the bi-conjugate gradient stabilized method and the conjugate gradient square method to solve the induction equation. We use the incomplete factorization preconditioner and the Neuman preconditioner for preconditioning. In this paper we compare the performance of these two methods associated with these two preconditioners in three different computers. We also show the effect of vectorization of the code in both of the vector and scalar computers. This spherical code has a promising prospect to carry out global magnetotelluric investigation through computer experiments.

Key words: Geomagnetic induction—Computational methods—Parallel computation
PACS: 91.25.Qi, 91.35. x, 91.60.Pn, 02.70.Bf, 02.60.Jn, 41.20.Bt

1 Introduction

The naturally occurring powerful, low-frequency electromagnetic (EM) fields have long been considered to be promising to study the Earth's interior using the magnetotelluric (MT) method.[2] Several limitations such as sparse observational sites and difficulties in data analyses, however, impeded efforts to build detailed 3-D images of electrical conductivity distribution within the Earth. To overcome the difficulties, we developed computer codes to do large-scale computational experiments, which we believe are effective to improve the image of the mantle structure. Simulational experiments can shed light on what information we may obtain from the limited observational data.

There are a number of papers that attempted to carry out computer simulations of MT investagion.[3–13] To our best knowledge, however, simulations presented in published papers so far are carried out primarily in the frequency domain. More importantly none of such efforts described above has successfully parallelized the computation, and therefore the power of the modern computer technology has not been exploited yet. Our recent paper (Chou et al. 2000, hereafter, paper I[1]) published in this journal reported a code development for time-domain MT computations in the Cartesian coordinates and some preliminary results of simulations using the code. Solving the induction equation in the Cartesian coordinates involves inverting a large sparse symmetric matrix.[14] We used incomplete factorization (or localized incomplete lower-upper, hereafter, ILU) preconditioner.[15,16] Here 'localized' means that when multiple CPUs are used, there is no message passing between CPUs in computing the preconditioner and thus an approximated incomplete factorization is obtained. Note that the ICCG method can be applied for a symmetric matrix. We demonstrated that this method (ICCG + localized ILU) achieved good parallel performance.

The code reported in Paper I is suitable to study the MT problem in a local or regional scale. A computational difficulty using the code is the restriction for the boundary conditions, and a proper treatment of the boundary condition is not trivial. With an assumption of a periodic boundary condition one can obtain a correct solution in the mathematical sense, which is, however, not realistic. A more realistic assumption is to adopt a free boundary condition, for which extrapolated values from the previous solution are used in the next time step. The disadvantage in this scheme is that errors from such extrapolations may be accumulated with increased time steps.

The above mentioned difficulty of boundary condition does not exist when in global-scale computation using the spherical coordinates. However, the finite difference code to compute magnetic induction in the spherical coordinates is not a trivial extension of the one in the Cartesian coordinate. In the spherical coordinates the matrix associated with the finite difference computation of the induction equation is not symmetric so the ICCG method is not applicable. We need to seek another algorithm to invert such an asymmetric matrix. Barrett et al.[18] provides templates of more than ten algorithms for the purpose of inverting a large sparse matrix. Here we adopt two of them, i.e., Bi-conjugate gradient stablized (BiCGstab, hereafter we simply refer it to BiCG method) and Conjugate Graduate Square (CGS) methods. The performance of these algorithms strongly depends on the preconditioner. A preconditioner is an approximated matrix of the sparse matrix that we want to invert. It should be close to the original matrix but be much easier to be inverted. The localized ILU method that we adopted in Paper I qualifies for the first criterion because it keeps the characters of the original sparse matrix effectively. The disadvantage of ILU preconditioning is that the computation is somewhat complicated, and accordingly the computational speed is slow, particularly in parallel computations. On the other hand the Neuman preconditioner is simple and the inversion is straightforward. The problem is whether this preconditioner is a reasonable approximation of the original matrix. We will compare the performance of these two preconditioners associated with the BiCG and CGS methods later.

Another important factor that affects the performance of computation is the vectorization of the code. In a vector-type computer a non-vectorized code consumes long CPU time and thus performs poorly. In our code the ILU preconditioner would be non-vectorized unless the index of do-loops is arranged in a special manner. In this paper we will discuss how to vectorize the code, and then compare the performances of the vectorized code with the non-vectorized one.

In the following sections we report how to program such parallel codes by Fortran MPI to solve the EM induction equation in the time domain and in the spherical coordinates. We then present some performance tests of the code.

2 Formalism

2.1 Magnetic Induction Equation

The Maxwell equations without displacement currents and free charges are

$$\frac{\partial \mathbf{B}}{\partial t} = -c\nabla \times \mathbf{E}, \tag{1}$$

$$\nabla \times \mathbf{B} = \frac{4\pi\sigma}{c}\mathbf{E}, \tag{2}$$

$$\nabla \cdot \mathbf{E} = 0, \tag{3}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{4}$$

where $\mathbf{B}$ is the magnetic field (Gauss), $\mathbf{E}$ is the electric field, σ the electrical conductivity (in cgs unit) and c the speed of light (3×10^{10}cm/s). These equations can be simplified by adopting a vector potential $\mathbf{A}$ and an electrical potential ϕ, with a Coulomb gauge

$$\nabla \cdot \mathbf{A} = 0, \tag{5}$$

and the electro-magnetic fields are expressed as

$$\mathbf{B} = \nabla \times \mathbf{A}, \tag{6}$$

$$\mathbf{E} = \frac{-1}{c}\frac{\partial \mathbf{A}}{\partial t} - \nabla\phi, \tag{7}$$

The Farady equation (1) and the divengent-free equation (4) are automatically satisfied by eqns. (6) and (7). If we further assume that there is no static electric field (e.g. see Everett and Schultz, 1996), then Ampere's law [Eq.(2)] becomes

$$\frac{4\pi\sigma}{c^2}\frac{\partial \mathbf{A}}{\partial t} - \nabla^2\mathbf{A} = 0. \tag{8}$$

This is the diffusion equation.

In the code we actually solve

$$\sigma\frac{\partial \mathbf{A}}{\partial t} - \nabla^2\mathbf{A} = 0. \tag{9}$$

The unit of σ can be expressed as

$$\begin{aligned}[\sigma_0] &= \frac{c^2[T]}{4\pi[L]^2}\mathrm{s}^{-1}(\mathrm{cgs}) \\ &= \frac{8.0 \times 10^5[T]}{[L]^2}\mathrm{S/m(SI)}.\end{aligned} \tag{10}$$

We adopt the time unit $[T]$ as 1 second, and length unit $[L]$ as 10 km. In this normalization, the normalization factor is $[\sigma_0] = 0.008$ S/m. The conductivity in the Earth interior, which ranges from $\sigma = 10^{-3}$ S/m on the surface to $\sigma = 1$ S/m in the upper mantle (about 700 km beneath the surface, see, e.g., Schultz et al, 1993[17]), is equal to a value that ranges from $\sigma = 0.125$ to $\sigma = 125$ in our code.

3 Finite Difference Equation

We use index i to denote the grids of the radial component $r(i)$, j to the poloidal component $\theta(j)$, and k to the azimuthal component $\phi(k)$. On the surface of the earth $90 - \theta$ corresponds to the latitude and ϕ corresponds to the longitude. The parallel computation is achieved by partitioning k into several CPUs. The vector potential $\mathbf{A}(r, \theta, \phi; t)$ is written as $\mathbf{A}^n(i, j, k)$ in finite difference form. Adopting Crank-Nicholson scheme, the finite difference form of the diffusive differential equation is

$$\sigma(i,j,k)\frac{A_r^{n+1}(i,j,k) - A_r^n(i,j,k)}{\Delta t} = \frac{1}{2}[(\nabla^2\mathbf{A}|_r)^{n+1} + (\nabla^2\mathbf{A}|_r)^n], \quad (11)$$

where the superscript indicates the time step and the subscript denotes the component of the vector. The component in the θ and ϕ directions can be written in a similar way. The ∇^2 operator on a vector in the spherical coordinates can be expressed explicitly as

$$\begin{aligned}\nabla^2\mathbf{A}|_r &= \nabla^2 A_r - \frac{2}{r^2(i)}A_r(i,j,k) - \frac{2\cos\theta(j)}{r^2(i)\sin\theta(j)}A_\theta(i,j,k) \\ &- \frac{1}{r^2(i)\Delta\theta}[A_\theta(i,j+1,k) - A_\theta(i,j-1,k)] \\ &- \frac{1}{r^2(i)\sin\theta(j)\Delta\phi}[A_\phi(i,j,k+1) - A_\phi(i,j,k-1)] \end{aligned} \quad (12)$$

$$\begin{aligned}\nabla^2\mathbf{A}|_\theta &= \nabla^2 A_\theta - \frac{1}{r^2(i)\sin^2\theta(j)}A_\theta(i,j,k) \\ &+ \frac{1}{r^2(i)\Delta\theta}[A_r(i,j+1,k) - A_r(i,j-1,k)] \\ &- \frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi}[A_\phi(i,j,k+1) - A_\phi(i,j,k-1)] \end{aligned} \quad (13)$$

$$\nabla^2\mathbf{A}|_\phi = \nabla^2 A_\phi - \frac{1}{r^2(i)\sin^2\theta(j)}A_\phi(i,j,k)$$

$$
\begin{aligned}
&+\frac{1}{r^2(i)\sin\theta(j)\Delta\phi}[A_r(i,j,k+1)-A_r(i,j,k-1)] \\
&-\frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi}[A_\theta(i,j,k+1)-A_\theta(i,j,k-1)]
\end{aligned} \tag{14}
$$

where

$$
\begin{aligned}
\nabla^2 f = &\frac{1}{\Delta r^2}[f(i+1,j,k)-2f(i,j,k)+f(i-1,j,k)] \\
&+\frac{1}{r(i)\Delta r}[f(i+1,j,k)-f(i-1,j,k)] \\
&+\frac{1}{r^2(i)\Delta\theta^2}[f(i,j+1,k)-2f(i,j,k)+f(i,j-1,k)] \\
&+\frac{\cos\theta(j)}{2r^2(i)\sin\theta(j)\Delta\theta}[f(i,j+1,k)-f(i,j-1,k)] \\
&+\frac{1}{r^2(i)\sin^2\theta(j)\Delta\phi^2}[f(i,j,k+1)-2f(i,j,k)+f(i,j,k-1)].
\end{aligned} \tag{15}
$$

is the Laplace operator operating on a scalar function.

This matrix equation can be written in a matrix form

$$
S\cdot A^{n+1} = T\cdot A^n + A_b^{n+1}, \tag{16}
$$

where A_b is the boundary term, and S and T are large sparse matrix. Figure 1 illustrates how the matrix S looks like. The large sparse matrix is divided into nine blocks. Block 1 collects the A_r terms in the $\hat{r}$ component, block 2 the A_θ terms in the $\hat{r}$ component, etc. In each block the sub-matrix is indexed by $l\times l$, where l is defined by $l=(i-1)Nj\cdot Nk+(j-1)Nk+k$. There are only seven non-zero terms in a row of the block. In our notation we use $SD(i,j,k,IB)$ to denote the diagonal terms in nine sub-matrixes where IB is the block number, and use $S(i,j,k,n,IB)$ to denote the other six non-zero terms, where n is an integer between 1 through 6 and its correspondence is indicated in the figure. The matrix component can be written as:

$$
\begin{aligned}
SD(i,j,k,1) = &\frac{\sigma(i,j,k)}{\Delta t}+\frac{1}{r^2(i)} \\
&+\frac{1}{\Delta r^2}+\frac{1}{r^2(i)\Delta\theta^2}+\frac{1}{r^2(i)\sin^2\theta(j)\Delta\phi^2}
\end{aligned} \tag{17}
$$

$$
SD(i,j,k,2) = \frac{\cos\theta(j)}{r^2(i)\sin\theta(j)} \tag{18}
$$

$$SD(i,j,k,5) = \frac{\sigma(i,j,k)}{\Delta t} + \frac{1}{2r^2(i)\sin^2\theta(j)}$$
$$+\frac{1}{\Delta r^2} + \frac{1}{r^2(i)\Delta\theta^2} + \frac{1}{r^2(i)\sin^2\theta(j)\Delta\phi^2} \tag{19}$$

$$SD(i,j,k,9) = SD(i,j,k,5) \tag{20}$$

$$SD(i,j,k,3) = SD(i,j,k,4) = SD(i,j,k,6)$$
$$= SD(i,j,k,7) = SD(i,j,k,8) = 0, \tag{21}$$

and

$$S(i,j,1,1) = \frac{-1}{2}\left(\frac{1}{\Delta r^2} - \frac{1}{r(i)\Delta r}\right) \tag{22}$$

$$S(i,j,2,1) = \frac{-1}{2}\left(\frac{1}{r^2(i)\Delta\theta^2} - \frac{\cos\theta(j)}{2r^2(i)\sin\theta(j)\Delta\theta}\right) \tag{23}$$

$$S(i,j,3,1) = \frac{-1}{2}\frac{1}{r^2(i)\sin^2\theta(j)\Delta\phi^2} \tag{24}$$

$$S(i,j,4,1) = S(i,j,3,1) \tag{25}$$

$$S(i,j,5,1) = \frac{-1}{2}\left(\frac{1}{r^2(i)\Delta\theta^2} + \frac{\cos\theta(j)}{2r^2(i)\sin\theta(j)\Delta\theta}\right) \tag{26}$$

$$S(i,j,6,1) = \frac{-1}{2}\left(\frac{1}{\Delta r^2} + \frac{1}{r(i)\Delta r}\right) \tag{27}$$

$$S(i,j,1,2) = S(i,j,3,2) = S(i,j,4,2) = S(i,j,6.2) = 0 \tag{28}$$

$$S(i,j,2,2) = \frac{-1}{2}\frac{1}{r^2(i)\Delta\theta} \tag{29}$$

$$S(i,j,5,2) = \frac{1}{2}\frac{1}{r^2(i)\Delta\theta} \tag{30}$$

$$S(i,j,1,3) = S(i,j,2,3) = S(i,j,5,3) = S(i,j,6.3) = 0 \tag{31}$$

$$S(i,j,3,3) = \frac{-1}{2}\frac{1}{r^2(i)\sin\theta(j)\Delta\phi} \tag{32}$$

$$S(i,j,4,3) = \frac{1}{2}\frac{1}{r^2(i)\sin\theta(j)\Delta\phi} \tag{33}$$

$$S(i,j,1,4) = S(i,j,3,4) = S(i,j,4,4) = S(i,j,6.4) = 0 \tag{34}$$

$$S(i,j,2,4) = \frac{1}{2}\frac{1}{r^2(i)\Delta\theta} \tag{35}$$

$$S(i,j,5,4) = \frac{-1}{2}\frac{1}{r^2(i)\Delta\theta} \tag{36}$$

$$S(i,j,1,6) = S(i,j,2,6) = S(i,j,5,6) = S(i,j,6.6) = 0 \tag{37}$$

$$S(i,j,3,6) = \frac{1}{2}\frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi} \tag{38}$$

$$S(i,j,4,6) = \frac{-1}{2}\frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi} \tag{39}$$

$$S(i,j,1,7) = S(i,j,2,7) = S(i,j,5,7) = S(i,j,6.7) = 0 \tag{40}$$

$$S(i,j,3,7) = \frac{-1}{2}\frac{1}{r^2(i)\sin\theta(j)\Delta\phi} \tag{41}$$

$$S(i,j,4,7) = \frac{1}{2}\frac{1}{r^2(i)\sin\theta(j)\Delta\phi} \tag{42}$$

$$S(i,j,1,8) = S(i,j,2,8) = S(i,j,5,8) = S(i,j,6.8) = 0 \tag{43}$$

$$S(i,j,3,8) = \frac{-1}{2}\frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi} \tag{44}$$

$$S(i,j,4,8) = \frac{1}{2}\frac{\cos\theta(j)}{r^2(i)\sin^2\theta(j)\Delta\phi} \tag{45}$$

$$S(i,j,*,5) = S(i,j,*,9) = S(i,j,*,1) \tag{46}$$

4 Performance Tests of the Spherical Code

4.1 Computational Scheme

We adopt BiCGstab and CGS algorithms to invert a large a sparse matrix. Appendix A includes brief descriptions of these two algorithms. The most CPU time consuming steps in these two methods are the preconditioning (e.g., solve $M\hat{s} = s$) and the matrix multiplications. We observe that both of the above algorithms involve two preconditioning and two matrix multiplications. On the other hand, the BiCGstab algorithm requires two more calculations of inner production, which is of minor importance, however.

The ILU method has been described in our previous paper. Here we summarize it here for completeness. So-called incomplete-lower-upper (ILU) factorization is to adopt a matrix **M** as the preconditioner such that it can be factorized as

$$M = (D + L)D^{-1}(D + U) = (D + L)(I + D^{-1}U), \tag{47}$$

where L and U are strictly triangular lower and upper matrixes, respectively, and D is a diagonal matrix. We use the triangular lower and upper parts of the original matrix A to be the matrix U and L. Inverting $Mx = b$ is proceeded by the following two steps:

$$(D + L)z = b \Longrightarrow z = D^{-1}(b - Lz), \tag{48}$$

$$(I + D^{-1}U)x = z \Longrightarrow x = z - D^{-1}(Ux). \tag{49}$$

The Neuman preconditioning method is that we re-write the original matrix equation $Ax = y$ as

$$(D + a)x = D(I + D^{-1}a)x = y, \tag{50}$$

where D is a diagonal matrix and a is the remaining part of the matrix A. If we assume $D^{-1}a$ is much smaller than the unity matrix, this equation can be solved as:

$$x = (I - D^{-1}a)D^{-1}y = [D^{-1} - (D^{-1}a)D^{-1}]y. \tag{51}$$

We test the performance of the above two algorithms associated with two preconditioners. In all of the following tests we adopt $n_r \times n_\theta \times n_\phi = 102 \times 52 \times 202$ grids and run the code for 40 time steps. We solve equation (9) with the boundary condition at $R_0 = 650$[L] as

$$A_\phi = \frac{A_0 \sin\theta}{(1 + a\sin\theta)^{3/2}} \sin(\phi - \Omega t), \tag{52}$$

where the parameter A_0, a, Ω are adopted as 10, 0.1, and $2\pi/(10^{-4}s)$, respectively. The other two components A_r and A_θ are zero at the boundary. In the inner boundary which is at $r = 550$[L] we assume $A = 0$. When there is no $\sin(\phi - \Omega t)$ term in eq. (52), it corresponds to a dipole magnetic field. We include this term here in order to demonstrate the code can solve the time-dependent equation and to check if the solution has a correct dependence on three coordinates. This condition is, however, not physical because it does not satisfy the divergence free condition (i.e., $\nabla \mathbf{A} \neq 0$). We regard this condition as tests of computer algorithms without physical correspondences. We assume the conductivity σ as a function of position:

$$\sigma(r,\theta,\phi) = \begin{cases} 0.125 & : \quad (r > 645) \\ 1.25 & : \quad (r < 645, -45 < 90 - \theta < 45, 45 < \phi < 135) \\ 6.25 & : \quad (\text{else}) \end{cases} \tag{53}$$

Figure 2 shows the result at $t = 500$s. Panel (a) is the iso-contours of B_r between latitude $\pm 60°$, and panel (b) is the polar view.

Figure 3 through figure 5 shows the results of these performance tests in the Fujutsu VPP 300 (figure 3), Hitachi SR2201 (figure 4), and IBM SP2 (figure 5). The solid line is for the CGS method with the Neuman preconditioner, dashed line for BiCG method with the Neuman preconditioner, and dotted line for the BiCG method with the ILU preconditioner. When the same preconditioner is adopted, we find that the CGS method and the BiCG method have almost the same speed in VPP and SR machine, while CGS is faster than BiCG in the SP machine. In these tests the Neuman preconditioner is faster than the ILU preconditioner. However, please see section 3.3 for a further discussion.

Figure 6 compare the performance of these three computers when the same numerical algorithm (CGS method with the Neuman preconditioner) is used. As we have seen in the previous section, VPP300 has the fastest speed, followed by SR2201, and then SP2.

4.2 Vectorization

All the do loops in our code are vectorized, except the ILU part containing loops like

```
 do k=2,mk-1
 do j=1,mj
 do i=1,mi
    x1(i,j,k)=diagmat1(i,j,k)*( yin1(i,j,k)
&     -x1(i-1,j,k)*smain(i,j,k1,1,1)
&     -x1(i,j-1,k)*smain(i,j,k1,2,1)
&     -x1(i,j,k-1)*smain(i,j,k1,3,1) )
 enddo
 enddo
 enddo
```

It is not vectorized because the calculation require the results from the previous iteration. This can be vectorized by the following method. We define a new variable $l = i+j$. When the sum of indexes i and j are fixed (i.e. constant l) the calculation can be performed independently and hence it is vectorized. The above loop becomes:

```
      do k=2,mk-1
      do l=2,mi+mj
        jl=max(1,l-mi)
        jh=min(mj,l-1)
      do j=jl,jh
        i=l-j
         x1(i,j,k)=diagmat1(i,j,k)*( yin1(i,j,k)
     &    -x1(i-1,j,k)*smain(i,j,k1,1,1)
     &    -x1(i,j-1,k)*smain(i,j,k1,2,1)
     &    -x1(i,j,k-1)*smain(i,j,k1,3,1) )
      enddo
      enddo
      enddo
```

The range of j needs to be calculated for every l because it changes as l increases. Another direction of vectorization is to define $l = i + j + k$ and $m = j + k$, and then perform the calculation by iterating k, m, l in order. We call this direction a 3D vectorization because all of the three indexes are involved in the substitution of indexes. The code of 3D vectorization is:

```
      lh=mi+mj+mk-1
      do l=4,lh
       ml=max(3,l-mi)
       mh=min(l-1,mj+mk-1)
      do m=ml,mh
       kl=max(2,m-mj)
       kh=min(m-1,mk-1)
      do k=kl,kh
        i=l-m
        j=m-k
         x1(i,j,k)=diagmat1(i,j,k)*( yin1(i,j,k)
     &    -x1(i-1,j,k)*smain(i,j,k1,1,1)
     &    -x1(i,j-1,k)*smain(i,j,k1,2,1)
     &    -x1(i,j,k-1)*smain(i,j,k1,3,1) )
      enddo
      enddo
      enddo
```

Figure 7 shows the performance of these three codes in a vector computer (Fujitsu VPP300). When the code is not vectorized, the computational speed becomes extremely slow. In a single CPU case the 3D vectorization is slightly faster than usual vectorization because the vector length is slightly longer. The 3D vectorization, however, has a much slower parallel performance as we can observed in figure 7. It is because there are several more arithmetic manipulations in the 3D vectorization and the number of these manipulations does not reduced proportionally to the increase of CPU numbers.

Figure 8 shows the performance of a vector code in a scalar computer. Vectorized loop has no advantage in a scalar computer such as SR2201 or SP2. Since a vectorized code makes several index substitutions in addition to the main calculation, it runs slower than a non-vectorized code. In figure 8 we also note that the non-vectorized code in a VPP machine runs even slower than the other two machines although the peak performance of a VPP machine is much faster.

4.3 Stability of Convergence

In section 6.1 it appears that the Neuman preconditioner is superior to the ILU preconditioner. Here we further investigate the iteration history when these two preconditioners are adopted. We find that the Neuman preconditioner is not stable.

Figure 9 shows the iteration history when these two preconditioners are used in a CGS method or a BiCG method. The residual ϵ is defined by

$$\epsilon = |\mathbf{X}^n - \mathbf{X}^{n-1}|/|\mathbf{X}^{n-1}|, \tag{54}$$

where $\mathbf{X}$ is the unknown vector to be solved, n is the iteration step, and $|\mathbf{X}|$ is the norm of a vector. We find that the number of iterations in the Neuman preconditioner is about twice of that in the ILU preconditioner. But since the Neuman precondition is much simpler than the ILU precondition, the overall speed of the Neuman preconditioner is still faster.

Here we try a different boundary condition at $r = R_0$ as

$$A_y = A_0[1 - a\sin\theta\cos(\phi - \Omega t)], \tag{55}$$

where y is the direction toward the rotational axis. Its components in the spherical coordinates are $A_r = A_y\cos\theta, A_\theta = A_y\sin\theta, A_\phi = 0$. Figure 10 shows the iteration history when the induction equation associated with this boundary is solved by the BiCG method with the ILU preconditioner (solid line) and the Neuman preconditioner (dashed line). In this case we find that the Neuman preconditioner does not converge to the desired residual after 10 times of iteration steps at which ILU has converged. Even the Neuman preconditioner requests less time per each iteration, it still takes much longer overall computational time because the number of iteration is increased tens of times. In another words, in some physical cases the Neuman preconditioner does have an advantage in computational speed, but in some physical cases it fails to converge. We thus conclude that the ILU preconditioner is more stable and reliable although it is slower.

5 Conclusion

We reported the new development of a parallel code that solves the induction equation [eq. (9)] in the spherical coordinates. The matrix equation becomes much more complicated than the Cartesian version. We compare the performances of the CGS method and the BiCGstab method. These two methods have almost the same performance in the VPP 300 and SR2201 computers, and the CGS is slightly faster than the BiCG in the SP2 computer. Using these methods we can achieve a good parallel performance in these three computers. The Neuman preconditioner is faster than the ILU preconditioner when the number of iterations of the former is about twice of the latter. In some physical conditions, however, the difference in the number of iterations becomes more than 10 times. In such a case the ILU preconditioner is preferred because of its stability.

We describe how to vectorize the ILU code. In a VPP computer such a vectorization is necessary, otherwise the performance is extremely poor. We also tested the 3D vectorization in order to enlength the vector length but found that it is not advantageous in parallel computing. Vectorization of the code slows down the performance in a scalar computer such as SP2 or SR2201 because the vectorization needs extra time to do index substitutions so that it is not benefical to improve the efficency.

6 Acknowledgement

This project has been carried out as part of the Earth Simulator funded by Science Technology Agency of Japan. Most of the work presented in this paper was carried out while the first author was employed by the Dept. of Research for Computational Earth Science, Research Organization for Information Science & Technology (RIST). We thank the members of RIST who provided us with support, advice and encouragement, and W. Horton in the University of

Texas at Austin for discussion at an early stage of this project. We also benefitted from the members at the Ocean Hemisphere Center of the Earthquake Research Institute at the University of Tokyo, who provided useful information on MT research. FT acknowlegdes partial support by her NSF grant No. EAR-9805006 and JSPS Fellowship

The computational work is carried out in the supercomputer clusters in the Japan Atomic Energy Research Institute (JAERI), and in the National Astronomical Observatory of Japan.

References

[1] Chou, W., Matsumoto, R., and Tajima, F., Simulational modeling of time-domain magnetic induction using parallel computational schemes: (I) in the Cartesian coordinates to be appear in Comp. Phys. Comm (2000), Paper I..

[2] Cagniard, L., Basic theory of the magneto-telluric method of geophysical prospecting, Geophysics 18 (1953) 605-635.

[3] Parker, R.L., The inverse problem of electromagnetic induction: existence and construction of solutions based on incomplete data, J. Geophys. Res. 85, No.B8 (1980) 4421-4428.

[4] Parker, R., Whaler, K.A., Numerical methods for establishing solutions to the inverse problem of electromagnetic induction, J. Geophys. Res. 86, No.B10 (1981) 9574-9584.

[5] Wannamaker, P.E., Advances in three-dimensional magnetotelluric modeling using integral equations, Geophysics, 56, No.11 (1991) 1716-1728.

[6] Mackie, R.L., and Madden T.R., Conjugate direction relaxation solutions for 3-D magnetotelluric modeling, Geophysics 58, No.7 (1993) 1052-1057.

[7] Mackie, R.L., Madden, T.R., and Wannamaker, P.E.,Three-dimensional magnetotelluric modeling using difference equations—theory and comparisons to integral equation solutions, Geophysics 58, No.2 (1993) 215-226.

[8] Zhang T.S. and Schultz, A., A 3-D perturbation solution for the EM induction problem in a spherical Earth—the forward problem, Geophys. J. Int. 111 (1992) 319-334.

[9] Everett, M.E. and Schultz, A., Geomagnetic induction in a heterogenous sphere: azimuthally symmetric test computations and the response of an undulating 660-km discontinuity, J. Geophys. Res. 101 (1996) 2765-2783.

[10] Martinec, Z., Spectral-finite-element approach to two-dimensional electromagnetic induction in a spherical Earth, Geophys. J. Int. 130 (1997) 583-594.

[11] Smith, J.T., Conservative modeling of 3-D electromagnetic fields, part I: properties and error analysis, Geophysics 61, No.5 (1996) 1308-1318.

[12] Smith, J.T., Conservative modeling of 3-D electromagnetic fields, part II: biconjugate gradient solution and an accelerator, Geophysics 61, No.5 (1996) 1319-1324.

[13] Wang, T. and Hohmann, G.W., A finite-difference, time-domain solution for three-dimensional electromagnetic modeling, Geophysics 58, No.6 (1993) 797-809.

[14] Press, W.H.,Teukolsky, S.A., Vetterling, W.T., and Flannery, B.P., Numerical Recipes in Fortran (Cambridge University Press, New York, 1992) ch.2.

[15] Venkatakrishnan, V., Parallel implicit methods for aerodynamic applications on unstructured grids, in Domain-based parallelism and problem decomposition methods in computational science and engineering, ed. Keyes, D.E., Saad, Y., and Truhlar, D.G. (SIAM, 1995) ch. 4.

[16] Nakajima, K., Nakamura H., and Tanahashi, T., Parallel iterative solvers with localized ILU preconditioning, in High-performance computing and networking, ed. Hertzberger, B. and Sloot, P., (Springer, Berlin, 1997) 342-350.

[17] Schultz, A., Kurtz,R.D., Chave, A.D., and Jones, A.G., Conductivity discontinuities in the upper mantle beneath a stable craton, Geophys. Res. Lett. 20 No.24 (1993) 2941-2944.

[18] Barrett, R., Berry, M., Chan, T., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., and Van der Vorst, H., Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods (SIAM, Philadephia, PA, 1994).

A BiCGstab and CGS algorithms

Here we adapt the templates of BiCGstab and CGS algorithms from Barrett et al.(1994).

The procedure to solve the matrix equation $A \cdot x = b$ by a BiCGstab algorithm is:

Compute $r^{(0)} = b - Ax^{(0)}$ for an initial guess $x^{(0)}$
Choose $\tilde{r}$ (we use $\tilde{r} = r^{(0)}$)
for $i = 1, 2, \ldots$
  $\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
  if $i = 1$
    $p^{(1)} = r^{(0)}$
  else
    $\beta_{i-1} = (\rho_{i-1}/\rho_{i-2})(\alpha_{i-1}/\omega_{i-1})$

$p^{(i)} = r^{(i-1)} + \beta_{i-1}(p^{(i-1)} - \omega_{i-1}v^{(i-1)})$
endif
solve $M\hat{p} = p^{(i)}$
$v^{(i)} = A\hat{p}$
$\alpha_i = \rho_{i-1}/(\tilde{r}^T v^{(i)})$
$s = r^{(i-1)} - \alpha_i v^{(i)}$
solve $M\hat{s} = s$
$t = A\hat{s}$
$\omega_i = (t^T s)/(t^T t)$
$x^{(i)} = x^{(i-1)} + \alpha_i \hat{p} + \omega_i \hat{s}$
$r^{(i)} = s - \omega t$
check convergence ($|x^{(i)} - x^{(i-1)}|/|x^{(i-1)}| < \epsilon$); continue if necessary
end

The CGS algorithm is:

Compute $r^{(0)} = b - Ax^{(0)}$ for an initial guess $x^{(0)}$
Choose $\tilde{r}$ (we use $\tilde{r} = r^{(0)}$)
for $i = 1, 2, ...$
$\rho_{i-1} = \tilde{r}^T r^{(i-1)}$
if $i = 1$
$u^{(1)} = r^{(0)}$
$p^{(1)} = u^{(1)}$
else
$\beta_{i-1} = \rho_{i-1}/\rho_{i-2}$
$u^{(i)} = r^{(i-1)} + \beta_{i-1}q^{(i-1)}$
$p^{(i)} = u^{(i)} + \beta_{i-1}(q^{(i-1)} + \beta_{i-1}p^{(i-1)})$
endif
solve $M\hat{p} = p^{(i)}$
$\hat{v} = A\hat{p}$
$\alpha_i = \rho_{i-1}/(\tilde{r}^T \hat{v})$
$q^{(i)} = u^{(i)} - \alpha_i \hat{v}$
solve $M\hat{u} = u^{(i)} + q^{(i)}$
$x^{(i)} = x^{(i-1)} + \alpha_i \hat{u}$
$\hat{q} = A\hat{u}$
$r^{(i)} = r^{(i-1)} - \alpha_i \hat{q}$
check convergence ($|x^{(i)} - x^{(i-1)}|/|x^{(i-1)}| < \epsilon$); continue if necessary
end

Table A.1
Specifications of the JAERI computers in which the parallel code is tested

Machine	# of PEs	peak CPU speed/PE	memory size/PE	data transferring speed
Cray T94	4	1.8 GFLOPS	1GB/4	128GB/s
Fujitsu VPP 300	16	2.2 GFLOPS	512 MB	570 MB/s
Hitachi SR 2201	64	0.3GFLOPS	256 MB	0.3GB/s
IBM SP2	48	266 MFLOPS	128MB	40MB/s
NEC SX-4	6	2GFLOPS	512MB	40MB/s