Documentation of the Multitasked Tangent Linear and Adjoint Models of the Adiabatic Version of the NASA GEOS-2 GCM (Version 6.5)

Yan Yang†, I. Michael Navon* and Ricardo Todling‡

†Supercomputer Computations Research Institute
Florida State University

* Corresponding author, Department of Mathematics and
Supercomputer Computations Research Institute
Florida State University

‡NASA/Data Assimilation Office
General Sciences Corporation

December 11, 1998

Abstract

This document presents a description of the development of the tangent-linear and the adjoint models of the adiabatic version of the GEOS-2 GCM. The difference between this GCM and the GEOS-1 GCM, the development and validation of the tangent-linear and adjoint models as well as the multitasking of the above models are documented, described and accompanied with illustrative examples.

Contents

List of Figures

List of Tables

1 Introduction

The scope of the Goddard Earth Observing System (GEOS) Data Assimilation System (DAS) is to assimilate operational and Mission to Planet Earth (MTPE) data of the atmosphere, the land surface and the ocean surface. Its algorithmic theoretical basis is fully documented (DAO, 1996). Much of the GEOS DAS development was aimed at attaining the ability to accommodate new data types and the improved treatment of historical data sets. To achieve this goal, the development of new techniques of data assimilation is under way, such as 4-D variational data assimilation and the Kalman filter/smoother for retrospective assimilation (Todling et al., 1998). The prerequisite for the applications of these techniques is the availability of the tangent-linear model (TLM) and the adjoint model (ADJ) of the present GCM used in the GEOS DAS.

The GEOS-1 is the version number 1 of the GEOS data assimilation system. The TLM and adjoint (ADJ) of the adiabatic and moist processes package of the GCM used in it was developed by Yang et al. (1996, 1997). Presently the system is upgraded to GEOS-2 and GEOS-3. The GCM presently used in these systems is significantly different from that in GEOS-1. Therefore the TLM and ADJ for this new version have to be developed anew. In this document we describe the development and validation of the TLM and its adjoint for the adiabatic part of the GCM version 6.5.

Section 2 presents a description of the GEOS-1 and GEOS-2 GCM, with the emphasis on the latter and their differences. Sections 3 and 4 describe the development of the tangent-linear and the adjoint models, respectively as well as the verification of their correctness. Section 5 provides a detailed description of the parallelization of the TLM and ADJ codes along with some illustrating examples of the codes.

2 Description of the NASA GEOS-2 GCM

In this chapter we briefly describe the GCMs used in GEOS-1 and GEOS-2, with emphasis on the latter and the differences between them.

The GEOS GCM was developed by the Data Assimilation Office (DAO) at the Goddard Laboratory for Atmospheres (GLA), in collaboration with the Climate and Radiation Branch, for use in the system being developed to analyze EOS data. The GEOS-1 GCM was fully documented by Takacs et al. (1994). The GEOS Data Assimilation System (DAS) based on GEOS-1 GCM has been used to produce a multi-year global atmospheric data set for climate research(Schubert et al., 1993). It has also been used operationally to provide scientific flight guidance during NASA's participation in the Measurements for Assessing the Effects of Stratospheric Aircraft experiment. The GEOS-1 GCM has also been used to produce multiple 10-year climate simulations as part of the DAO's participation in the Atmospheric Model Intercomparison Project (AMIP) sponsored by the Program for Climate Model Diagnostics and Intercomparison (PCMDI).

The earliest predecessor of the GEOS-1 GCM was developed in 1989 based on the "plug-compatible" concepts outlined in Kalnay et al. (1989) and subsequently improved in 1991 (Fox-Rabinovitz, et al., 1991 and Helfand et al., 1991). The plug-compatibility of the physical parameterizations together with the plug-compatible "Dynamical Core" introduced by Suarez and Takacs (1995) facilitate the development and testing of new algorithms.

The GEOS-2 GCM is an incremental development over GEOS-1 GCM. The model has been prepared for broader future capabilities to meet the science requirements and scope of the GEOS DAS to assimilate operational and Mission to Planet Earth (MTPE) data. Based on extensive analysis and evaluation of the GEOS-1 system, the GEOS-2 GCM addresses some of the fundamental limitations of GEOS-1. The GEOS-2 GCM also provides the next benchmark and infrastructure base in the DAO's effort to develop the GEOS-3 system for the EOS observing period.

2.1 Atmospheric dynamic equations

As in the GEOS-1 GCM, the momentum equations used in the GEOS-2 GCM are written in the "vector invariant" form, as in Sadourney (1975) and Arakawa and Lamb (1981), to facilitate the derivation of the energy and potential enstrophy conserving scheme. The thermodynamic (potential temperature) and moisture (specific humidity) equations are written in flux form to facilitate potential temperature and moisture conservation.

The GEOS GCM uses a σ vertical coordinate defined by

$$\sigma = \frac{p - p_T}{\pi} \tag{1}$$

where p is the pressure, $\pi \equiv p_s - p_T$, p_s is the surface pressure and p_T is a constant prescribed pressure at the top of the model atmosphere. With $p_T = 0$ this coordinate reduces to the conventional σ coordinate proposed by Phillips (1957).

With this vertical coordinate, the continuity equation becomes

$$\frac{\partial \pi}{\partial t} = -\nabla_\sigma \cdot (\pi \mathbf{V}) - \frac{\partial(\pi\dot{\sigma})}{\partial \sigma} \tag{2}$$

where $\mathbf{V}$ is the horizontal velocity vector. Integrating (2) and assuming $\dot{\sigma} = 0$ at $p = p_s$ and $p = p_T$, we obtain the formula used in the model:

$$\frac{\partial \pi}{\partial t} = -\int_0^1 \nabla_\sigma \cdot (\pi \mathbf{V}) \mathbf{d}\sigma \tag{3}$$

and

$$(\pi\dot{\sigma}) = -\sigma\frac{\partial \pi}{\partial t} - \int_0^\sigma \nabla_\sigma \cdot (\pi \mathbf{V}) \mathbf{d}\sigma. \tag{4}$$

The equation of state for an ideal gas is $\alpha = RT/p$, where α is the specific density, T is the temperature, and R is the gas constant. The following alternative form is used:

$$\alpha = \frac{c_p\theta}{\sigma}\left(\frac{\partial P}{\partial \pi}\right)_\sigma \tag{5}$$

where $\theta \equiv T/P$ is the potential temperature, $P \equiv (p/p_0)^\kappa$, $\kappa = R/c_p$, c_p is the specific heat at constant pressure, and p_0 is a reference pressure which is taken as $p_0 = 1000\ hPa$.

The hydrostatic equation is

$$\frac{\partial \Phi}{\partial p} = -\alpha \tag{6}$$

where Φ is the geopotential. Another form of this equation which is used in the model is:

$$\frac{\partial \Phi}{\partial P} = -c_p\theta. \tag{7}$$

The momentum equation is written as:

$$\begin{aligned}
\frac{\partial \mathbf{V}}{\partial t} &= -(f+\zeta)\mathbf{k} \times \mathbf{V} - \dot{\sigma}\frac{\partial \mathbf{V}}{\partial \sigma} - \nabla_\sigma(\Phi + K) - c_p\theta\nabla_\sigma P - \frac{g}{\pi}\frac{\partial \mathcal{T}}{\partial \sigma} \\
&= -\eta\mathbf{k} \times (\pi\mathbf{V}) - \dot{\sigma}\frac{\partial \mathbf{V}}{\partial \sigma} - \nabla_\sigma(\Phi + K) - c_p\theta(\frac{dP}{d\pi})_\sigma\nabla\pi - \frac{g}{\pi}\frac{\partial \mathcal{T}}{\partial \sigma},
\end{aligned} \tag{8}$$

where

$$\eta = \frac{(f+\zeta)}{\pi}$$

is an "external" potential vorticity, f is the Coriolis parameter, $\mathbf{k}$ is the unit vector in the vertical, $\zeta \equiv \nabla_\sigma \times \mathbf{V}$ is the vertical component of the vorticity along σ surfaces, $K \equiv \frac{1}{2}(\mathbf{V}\cdot\mathbf{V})$ is the kinetic energy per unit mass, g is the acceleration of gravity, and $\mathcal{T}$ is the horizontal frictional stress.

The thermodynamic equation is written as:

$$\frac{\partial(\pi\theta)}{\partial t} = -\nabla_\sigma \cdot (\pi\mathbf{V}\theta) - \frac{\partial(\pi\dot{\sigma}\theta)}{\partial\sigma} + \frac{\pi\mathbf{Q}}{c_p P} \tag{9}$$

where $\mathbf{Q}$ is the diabatic heating per unit mass.

The Aries/GEOS Dynamical Core computes tendencies for an arbitrary number of atmospheric constituents, such as water vapor and ozone. These are also written in flux form:

$$\frac{\partial\left(\pi q^{(k)}\right)}{\partial t} = -\nabla_\sigma \cdot \left(\pi\mathbf{V}q^{(k)}\right) - \frac{\partial\left(\pi\dot{\sigma}q^{(k)}\right)}{\partial\sigma} + \pi S^{(k)} \tag{10}$$

where $q^{(k)}$ is the specific mass of the kth constituent, and $S^{(k)}$ is its source per unit mass of air.

2.2 Discretization of the NASA GEOS-2 GCM and its difference from GEOS-1 GCM

a. Horizontal and vertical differencing

A complete description of the finite-difference scheme used can be found in Suarez and Takacs (1995).

The GEOS-1 GCM uses version 1 of the Aries/GEOS Dynamical Core, which is the second-order potential enstrophy and energy conserving horizontal differencing scheme constructed in the horizontal using the staggered Arakawa C-grid. This scheme was developed by Sadourney(1975) and further described by Burridge and Haseler (1977). The GEOS-2 GCM employs Version 2 of this Dynamical Core. The important difference relative to the former version is that it uses fourth order horizontal differences instead of second order. This scheme conserves total energy and potential enstrophy for the non-divergent component of the flow in the shallow water equations. It is fourth-order in the sense that it reduces to the fourth-order Arakawa Jacobian for non-divergent flow. It thus provides fourth-order accuracy for the advection of a second-order vorticity by the non-divergent part of the flow. Horizontal advection of potential temperature and moisture is performed using the fourth-order scheme in use in the UCLA GCM. It also is fourth-order only in the advection by the non-divergent part of the flow. The use of fourth-order differencing substantially improves the phase propagation of synoptic scale waves.

In the vertical the Aries/GEOS Dynamical Core used a Lorentz or unstaggered vertical grid in generalized sigma coordinates. The vertical differencing scheme is that of Arakawa and Suarez (1983) which ensures that:

- The pressure gradient force generates no circulation of vertically integrated momentum along a contour of surface topography
- The finite-difference analogues of the energy-conversion term have the same form in the kinetic energy and thermodynamic equations

- The global mass integral of the potential temperature is conserved under adiabatic processes
- The hydrostatic equation for the lowest thickness has a local form
- The hydrostatic equation is exact for vertically isentropic atmospheres
- The pressure-gradient force is exact for three-dimensionally isentropic atmospheres.

The differences of GEOS-2 GCM from GEOS-1 GCM are that the model top is raised to 0.01 hPa and the vertical resolution is increased to 70 levels. The raising of the model lid is directed at improving stratospheric descent over the winter pole and addressing the stratospheric cold pole problem. Increasing the number of levels is directed at improving the representation of the planetary boundary layer, the troposphere at the altitude of the sub-tropical jet stream, and the stratosphere.

In GEOS-2 the capability is added to perform a coordinate rotation of the finite difference grid. This was implemented to address the polar noise problems, especially in the stratosphere. In addition, the coordinate rotation is at the basis for adaptive resolution capabilities to provide the infrastructure for possible regional applications.

b. Time integration scheme

The GEOS GCM has the ability to use the Matsuno time integration scheme or the Leapfrog time integration scheme together with an Asselin (1972) time filter. The GEOS GCM employs a unique method for incorporating adjustments due to diabatic processes (ie. moist convection, radiation and turbulence) and the analysis increments during an assimilation. It is designed to be completely time continuous with tendencies from the physics parameterizations and filters being incrementally added at every dynamics time step. Earlier versions had intermittent applications of these processes. The diabatic time tendencies are updated at a time step appropriate to the physical parameterizations using the current time index, and are held constant between physics calls. By gradually incorporating the diabatic adjustments during the model integration, shocks and dynamical imbalances are greatly reduced.

An eighth-order Shapiro filter with a reduced coefficient is applied to the wind, potential temperature and specific humidity to avoid non-linear computational instability. The reduced-coefficient filter is applied at every step in such a way that the amplitude of the two-grid interval wave is essentially removed in six hours. Applying the filter weakly at each step eliminates the shock that occurred in earlier assimilations using an intermittent applications of the filter. The Shapiro filtered time tendencies are updated every model time step.

The model also uses a polar Fourier filter to avoid linear instability due to violation of the linear stability condition for the Lamb wave and internal gravity waves. This polar filter is applied only to the tendencies of the winds, potential temperature, specific humidity and surface pressure.

Apart from the above-mentioned differences, there is another major difference on the coding of the model. The version 6.5 of the GEOS-2 GCM whose TLM and adjoint we developed is a multi-tasked version, with compiler directives added into the code to direct the compiler to produce an executable which can run in parallel on multi-CPUs. To enable multi-tasking, the code was modified and is quite different from the previous versions. This issue is addressed in more detail in section 5.

2.3 Structure and flow chart of the adiabatic NASA GEOS-2 GCM

Fig. 1 shows the flow chart of the NASA GEOS-2 GCM and Fig. 2 shows the flow chart of the Aries/GEOS dynamical core, which is the core part of the GCM.

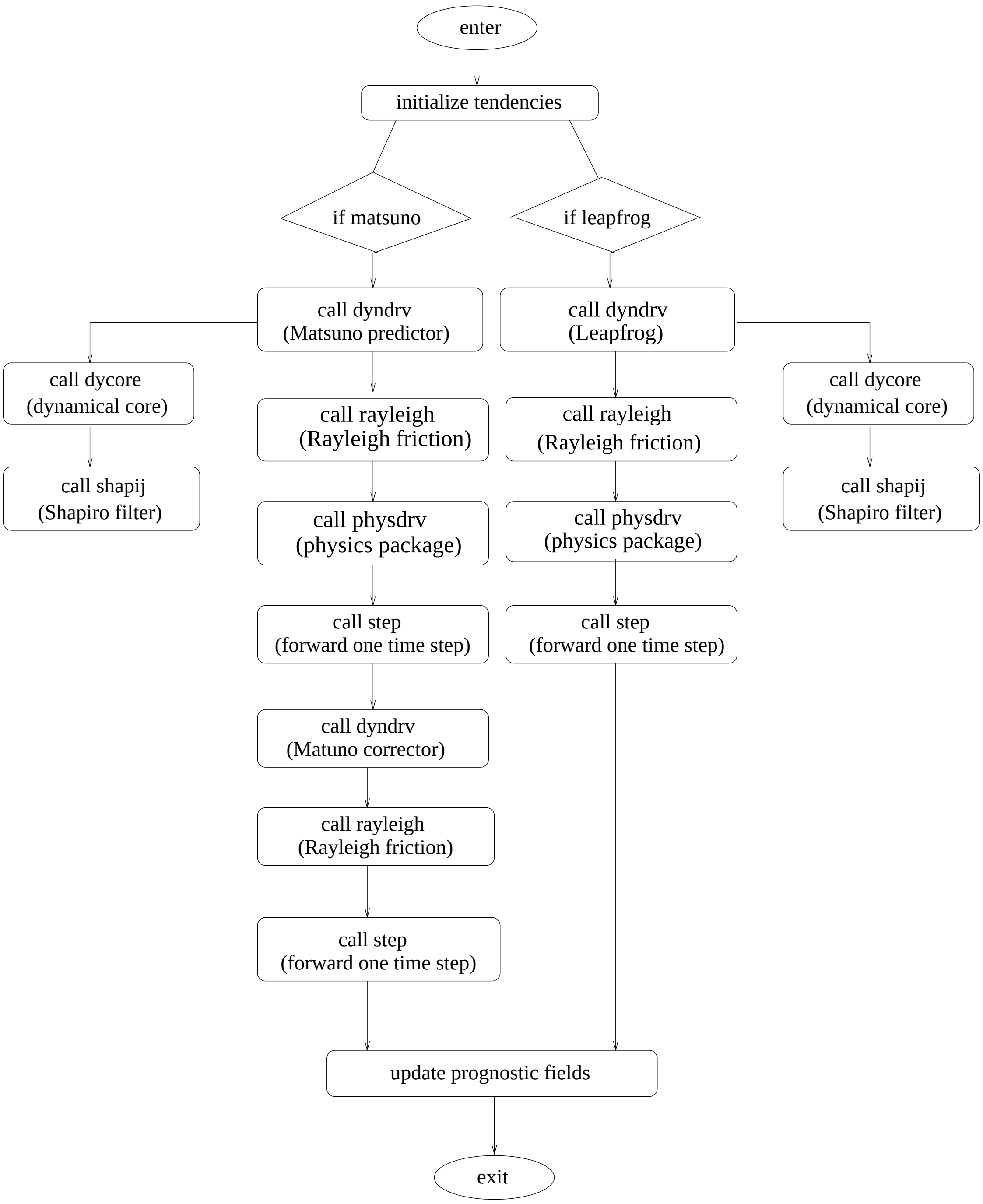


Figure 1: Flow chart of the NASA GEOS-2 GCM

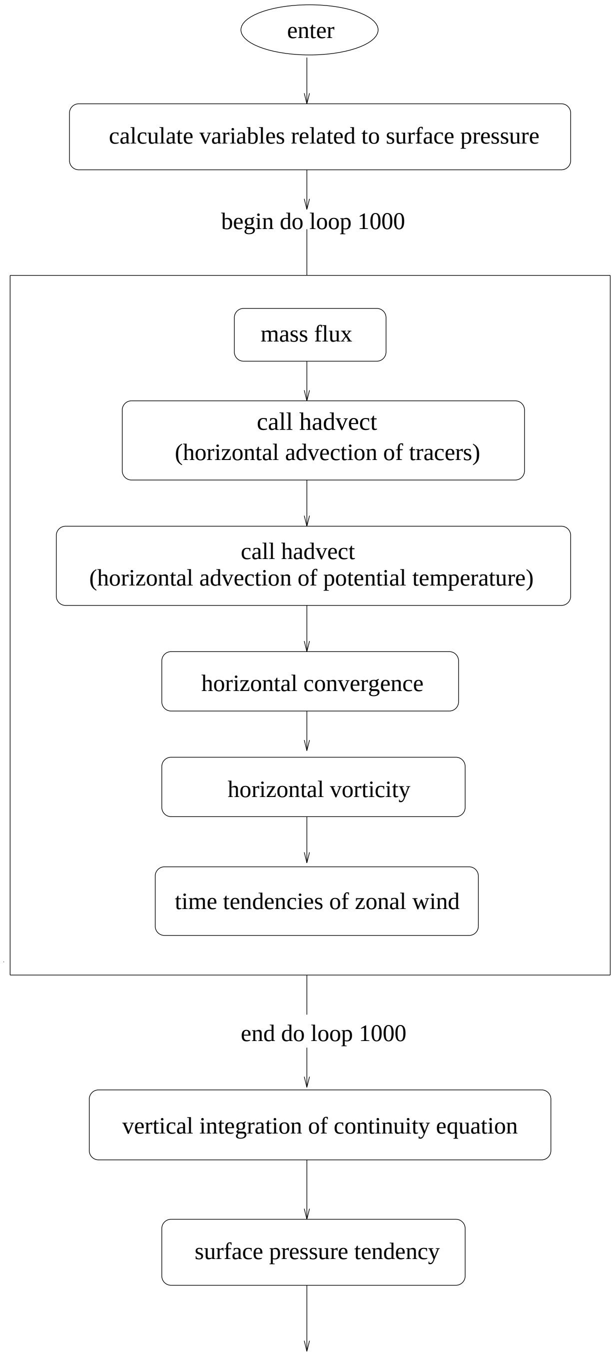


Fig. 2, to be continued

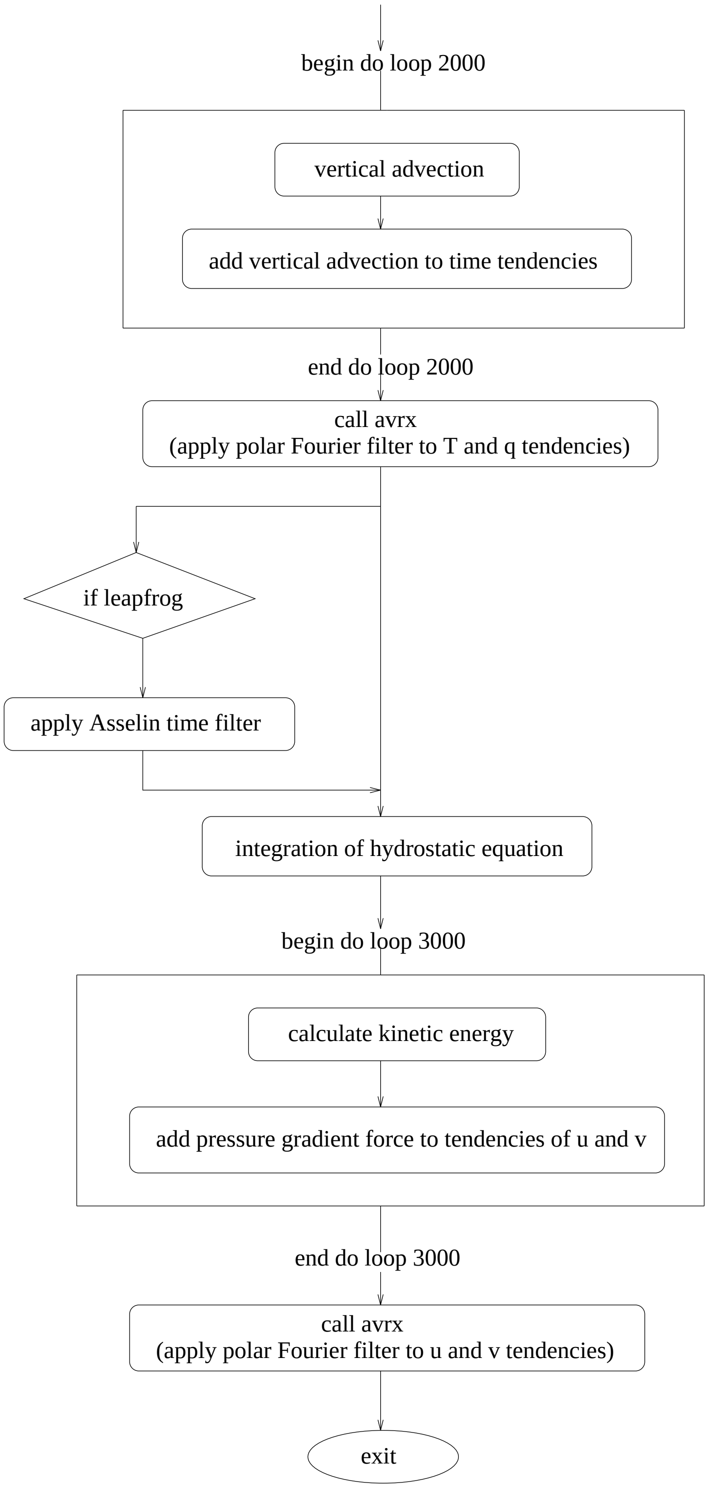


Figure 2: Flow chart of the Aries/GEOS dynamical core

3 Tangent linear model of the adiabatic version of NASA GEOS-2 GCM

The TLM is the basis upon which the adjoint model is to be developed. It is also a useful tool in the research on the dynamics of the evolution of atmospheric perturbations. Therefore we first developed the TLM of the NASA GEOS-2 GCM.

We linearized the GCM at the FORTRAN code level and developed the tangent-linear model. The method was essentially the same as that described in Yang et al. (1996).

The basic fields are calculated along with the perturbation fields exactly the same as in the original GCM. The compiler directives "`#ifdef _TLM_`" and "`#endif`" are added respectively before and after the statements related to perturbations so that if we do not specify `-D _TLM_` in the compile command, the model would be exactly the same as the original GCM, otherwise the TLM would produce the evolution of perturbations as well as the basic field which is the same as given by the original GCM. This has the advantage of making it easier to check the basic fields and ensuring that the basic field exactly corresponds to the trajectory given by the original nonlinear model.

If the basic field is not to be updated every time step, it can be read in at the beginning of each time step. This basic field is updated at certain time interval by storing the trajectory of the original nonlinear model run.

The flow charts of the TLM are essentially the same as that of the GCM give in Figs. 1 and 2, except that the subroutines were renamed according to the conventions outlined in the next subsection.

3.1 Notational convention for the tangent linear model

The notational convention we followed in developing the TLM code is quite different from that of Yang et al. (1996) in the TLM of GEOS-1 GCM.

The names of the variables related to basic fields are kept the same as in the original nonlinear model and the perturbation variables are represented by adding a prefix "`pt_`" before the corresponding basic variables.

For the subroutine names, we just add "`tl`" before the corresponding subroutine names in the nonlinear model. For example, "`tlHADVECT`" is the name of the subroutine calculating horizontal advection in the TLM, corresponding to the subroutine "`HADVECT`" in the original GCM.

Examples of the TLM code can be found in section 5.

3.2 Verification of the tangent linear model

The TLM is the linear approximation to the original nonlinear model. Its accuracy determines the accuracy of the adjoint model and the accuracy of the gradient of cost function calculated from the adjoint model in 4-D variational assimilation. Therefore we need to carefully evaluate its accuracy and its validity range.

To verify the correctness of the TLM, we first checked each subroutine by comparing the result of the TLM with the difference of the twice GCM call, with and without perturbations, respectively. After that, to check the whole GCM, we employed a more quantitative algorithm, as described in Yang et al. (1996).

The evolution of the vector of the atmospheric variables $\mathbf{X}$ is given by the integration of the model M between times t_0 and t_n as:

$$\mathbf{X}(t_n) = M\left(t_n, t_0\right)\left(\mathbf{X}\left(t_0\right)\right). \tag{11}$$

If at initial time $\mathbf{X}(t_0)$ has a perturbation $\delta\mathbf{X}(t_0)$, it would evolve to:

$$\mathbf{X}^{'}(t_n) = M\left(t_n, t_0\right)\left(\mathbf{X}_0\left(t_0\right) + \delta\mathbf{X}\left(t_0\right)\right) \tag{12}$$

The perturbation

$$\begin{aligned}
\Delta\mathbf{X} &= \mathbf{X}^{'}(t_n) - \mathbf{X}(t_n) \\
&= M\left(t_n, t_0\right)\left(\mathbf{X}_0\left(t_0\right) + \delta\mathbf{X}\left(t_0\right)\right) - M\left(t_n, t_0\right)\left(\mathbf{X}\left(t_0\right)\right) \\
&= M\left(t_n, t_0\right)\left(\mathbf{X}_0\left(t_0\right)\right) + R\left(t_n, t_0\right)\delta\mathbf{X}(t_0) + O(\delta\mathbf{X}(t_0)^2) - M\left(t_n, t_0\right)\left(\mathbf{X}_0\left(t_0\right)\right),
\end{aligned}$$

where R is the first derivative of M, or the tangent-linear model.

In the first order, the perturbation of $\mathbf{X}$ can be approximated by:

$$\delta\mathbf{X}\left(t_n\right) = R\left(t_n, t_0\right)\delta\mathbf{X}\left(t_0\right) \tag{13}$$

The difference is denoted as:

$$D(\delta\mathbf{X}) = \Delta\mathbf{X} - \delta\mathbf{X}. \tag{14}$$

To quantify the comparison, a norm is defined by:

$$\|\mathbf{X}\|^2 = \mathbf{X}^T W \mathbf{X} \tag{15}$$

where W is a diagonal matrix to give the proper weighting of each variables.

The relative difference between the TLM and the nonlinear model is defined as

$$r = \frac{\|D\|}{\|\delta\mathbf{X}\|}. \tag{16}$$

It gives a quantitative measure of the accuracy of the linear approximation.

The basic trajectory we used to carry out the verification is the model integration from the initial condition on December 21st, 1991 at 03GMT, which is derived from the NASA/DAO assimilated data set. The resolution is 5^o longitude by 4^o latitude. To produce the perturbations, we first integrated the model for 6 hours and take the difference between the result and the initial condition, then normalized it by $\|\mathbf{X}\|$. This normalized perturbation multiplied by different scaling factors α serves as the perturbations of a series of our experiments. For $\alpha = 1.0$, the global square root of the perturbation on the zonal and meridional wind, the potential temperature, the surface pressure and the specific humidity were 3.37 m/s, 3.19 m/s, 0.30 ^0K, 5.90 hPa and 3.87 $\times 10^{-4}$ kg/kg, respectively. This is not a large perturbation.

For each of the experiments, we calculated the correlation coefficients as well as the relative differences r between D and $\delta\mathbf{X}$ for each model variables. Table 1 gives the correlation coefficients between D field and $\delta\mathbf{X}$ field and Table 2 shows the relative errors. The results presented here were carried out with Matsuno scheme. The leapfrog scheme results were very similar. The integration period was 12 hours.

From the tables we can see that as α decreases from 10.0 to 10^{-5}, the correlation coefficients approach 1.0 with very high accuracy. The relative error decreases almost linearly between $\alpha = 1.0$ and $\alpha = 10^{-4}$.

To provide an idea of the validity range of the TLM, Figs. 3a and b show the variation of the correlation coefficients and the relative errors with time. We chose $\alpha = 10.0$, $\alpha = 1.0$ and $\alpha = 0.1$ to represent large, medium and small perturbations, respectively. It can be seen that for large initial perturbation, the error grew rapidly with time, whereas for the medium perturbation, which is about the same magnitude as the normal analysis error, the correlation coefficient was still higher than 0.99 for 72 hours and the relative error was smaller than 10%. For smaller perturbations, the error growth was even slower. These two figures provide the validity range of the TLM in terms of the magnitude of the perturbations and the time scale.

Table 1: Correlation Coefficients Between D Field and $\delta\mathbf{X}$ Field:

α	u	v	T	p_s	q
10.0	0.9524766514420	0.9488548638531	0.9581327588277	0.9772099692437	0.9058661066522
1.0	0.9994728793944	0.9994203127811	0.9995342074172	0.9997579574115	0.9989258521684
10^{-1}	0.9999946889274	0.9999941435555	0.9999953086670	0.9999975927674	0.9999892780251
10^{-2}	0.9999999468518	0.9999999413767	0.9999999530550	0.9999999759444	0.9999998928165
10^{-3}	0.9999999994684	0.9999999994137	0.9999999995304	0.9999999997594	0.9999999989281
10^{-4}	0.9999999999947	0.9999999999941	0.9999999999952	0.9999999999975	0.9999999999892
10^{-5}	0.9999999999994	0.9999999999994	0.9999999999998	0.9999999999999	0.9999999999995
10^{-6}	0.9999999999458	0.9999999999513	0.9999999999902	0.9999999999983	0.9999999999643
10^{-7}	0.9999999947244	0.9999999951117	0.9999999990432	0.9999999998310	0.9999999963746

Table 2: Relative Error Between D Field and $\delta\mathbf{X}$ Field:

α	u	v	T	p_s	q	*total*
10.0	30.68	31.85	28.79	21.00	43.53	29.50
1.0	3.246	3.401	3.052	2.130	4.634	3.121
10^{-1}	0.3259	0.3420	0.3064	0.2122	0.4630	0.3133
10^{-2}	3.261×10^{-2}	3.422×10^{-2}	3.065×10^{-2}	2.121×10^{-2}	4.629×10^{-2}	3.135×10^{-2}
10^{-3}	3.261×10^{-3}	3.422×10^{-3}	3.065×10^{-3}	2.121×10^{-3}	4.629×10^{-3}	3.135×10^{-3}
10^{-4}	3.263×10^{-4}	3.425×10^{-4}	3.066×10^{-4}	2.120×10^{-4}	4.631×10^{-4}	3.137×10^{-4}
10^{-5}	1.084×10^{-4}	1.053×10^{-4}	6.459×10^{-5}	2.726×10^{-5}	9.462×10^{-5}	9.656×10^{-5}
10^{-6}	1.040×10^{-3}	9.856×10^{-4}	4.433×10^{-4}	1.738×10^{-4}	8.445×10^{-4}	9.119×10^{-4}
10^{-7}	1.027×10^{-2}	9.880×10^{-3}	4.383×10^{-3}	1.775×10^{-3}	8.515×10^{-3}	9.066×10^{-3}

4 Adjoint model of the adiabatic version of NASA GEOS-2 GCM

4.1 Notational convention for the adjoint model

The adjoint model was constructed directly from the TLM code with the method as detailed in Yang et al. (1996). The major difference is in the notational convention.

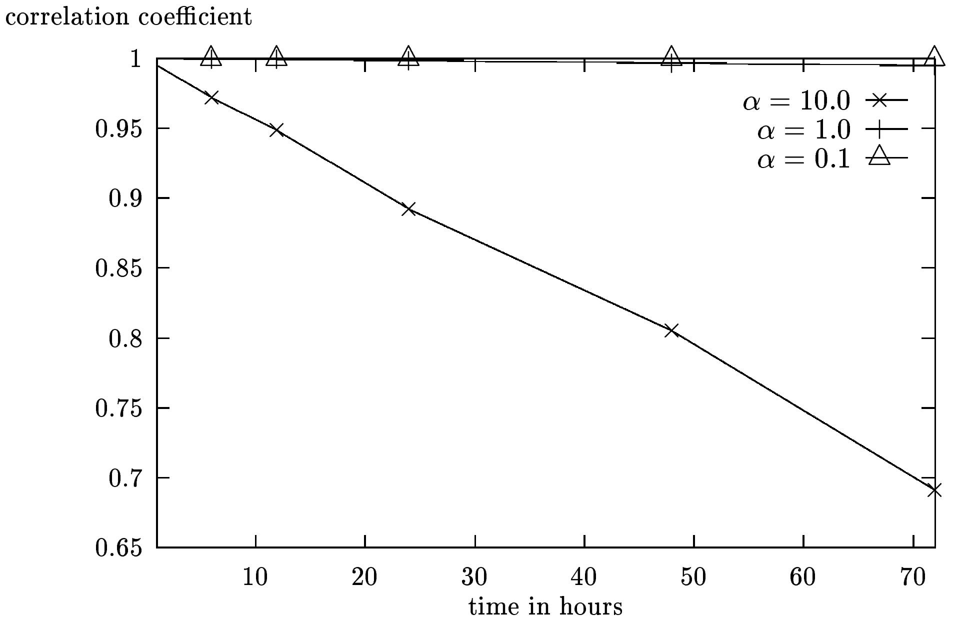
In accordance with the TLM, the variables for the basic fields keep the same names as that in the original GCM, while the adjoint variables have an prefix "`ad_` " before their corresponding names in the forward GCM.

For the subroutine names, we add "`ad`' before the corresponding subroutine names in the forward GCM. For example, "`adHADVECT`" is the adjoint of the subroutine "`tlHADVECT`", which is the tangent linear routine for the horizontal advection.

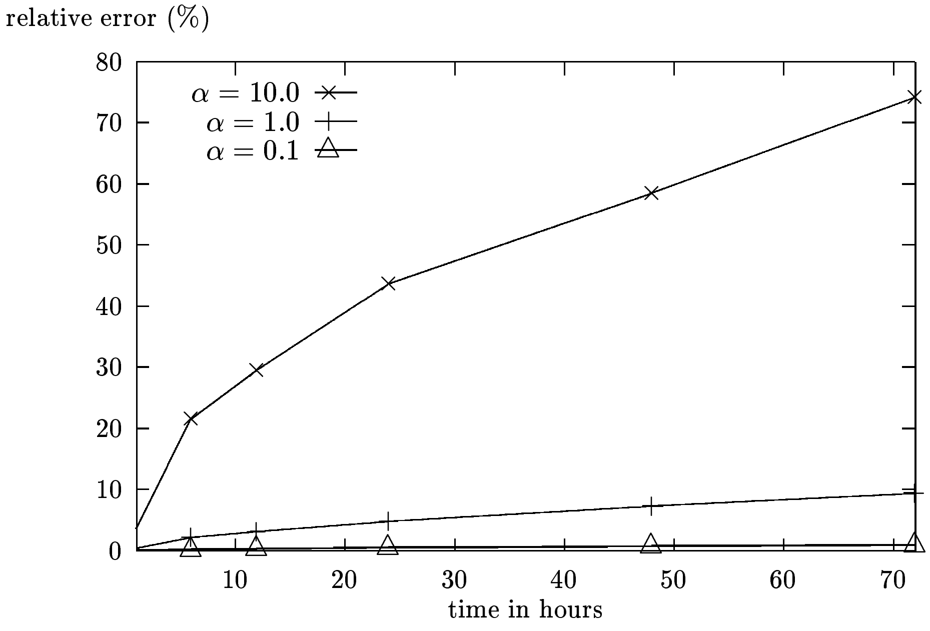
Examples of the adjoint code can be found in section 5.

4.2 Structure and flow chart of the adjoint model of the adiabatic version of NASA GEOS-2 GCM

The sequence of execution in the adjoint model is generally opposite to that of the original forward GCM. Figs. 4 and 5 show the flow chart of the adjoint model and the adjoint dynamic core part. In the flow charts, we use the same terms in explaining the functions of each segment as in Figs. 1 and 2, but actually they represent the adjoint of the corresponding segments in the latter.



(a)



(b)

Figure 3: Verification of the TLM model. (a): Variation of correlation coefficient with time; (b): Variation of relative error $\|D\|/\delta\mathbf{X}$ with time.

4.3 Verification of the adjoint model

We tested the correctness of the adjoint model by applying the following identity (see Navon et al., 1992):

$$(AQ)^T(AQ) = Q^T(A^T(AQ)) \tag{17}$$

where Q represents the input of the original code, A represents the original code or a segment of it, say a subroutine, a do loop or even a single statement. A^T is the adjoint of A. If (17) holds within the machine accuracy, it can be said that the adjoint is correct versus the TLM code.

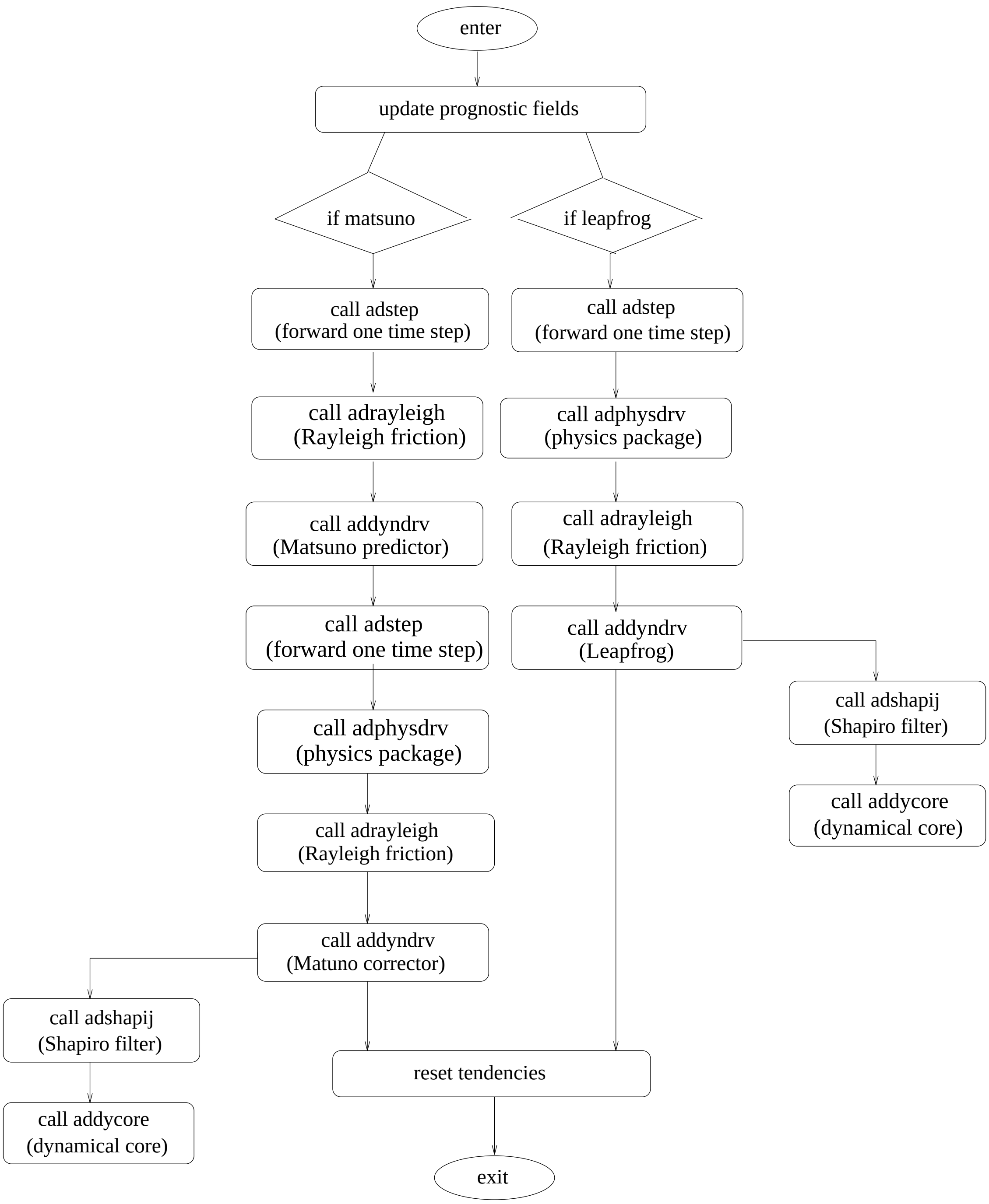


Figure 4: Flow chart of the adjoint of the NASA GEOS-2 GCM.

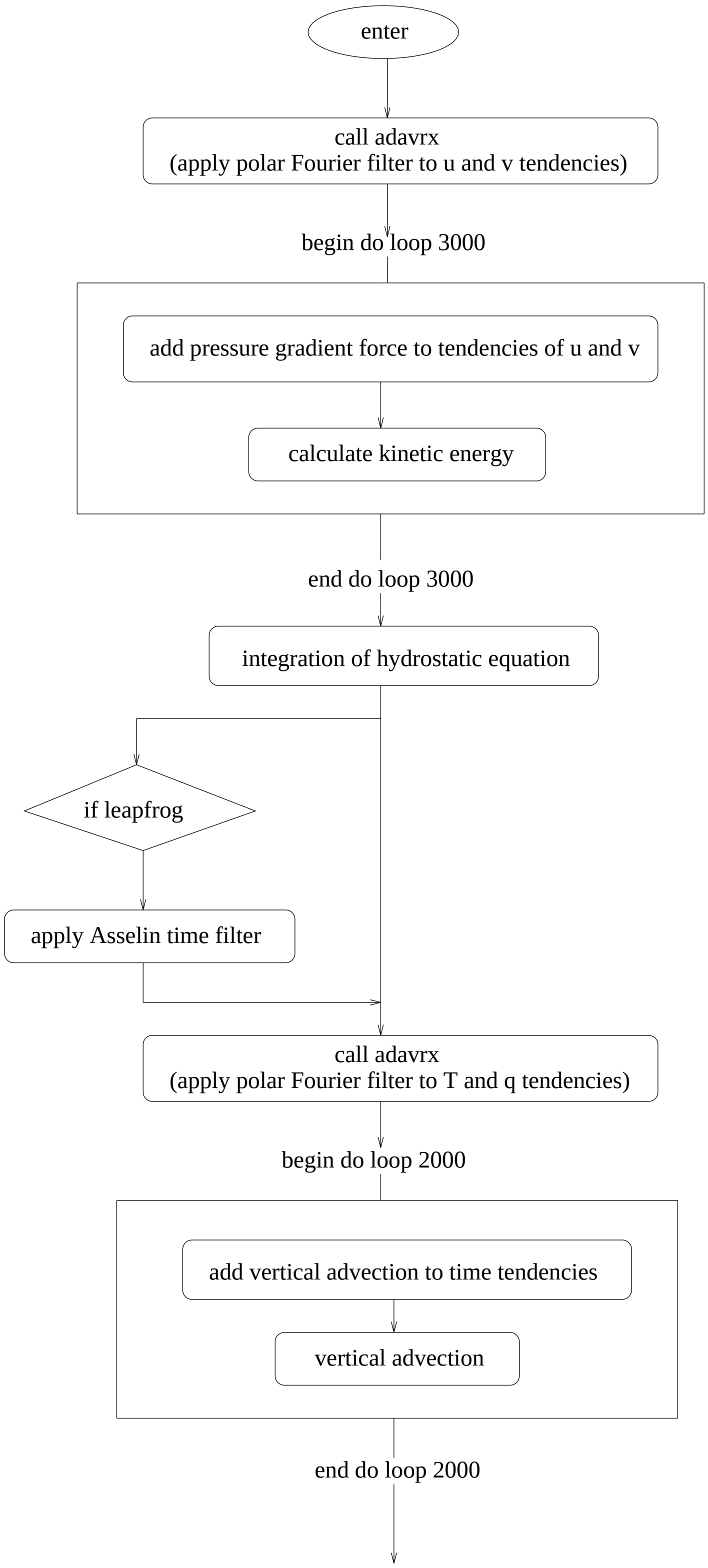


Fig. 5, to be continued

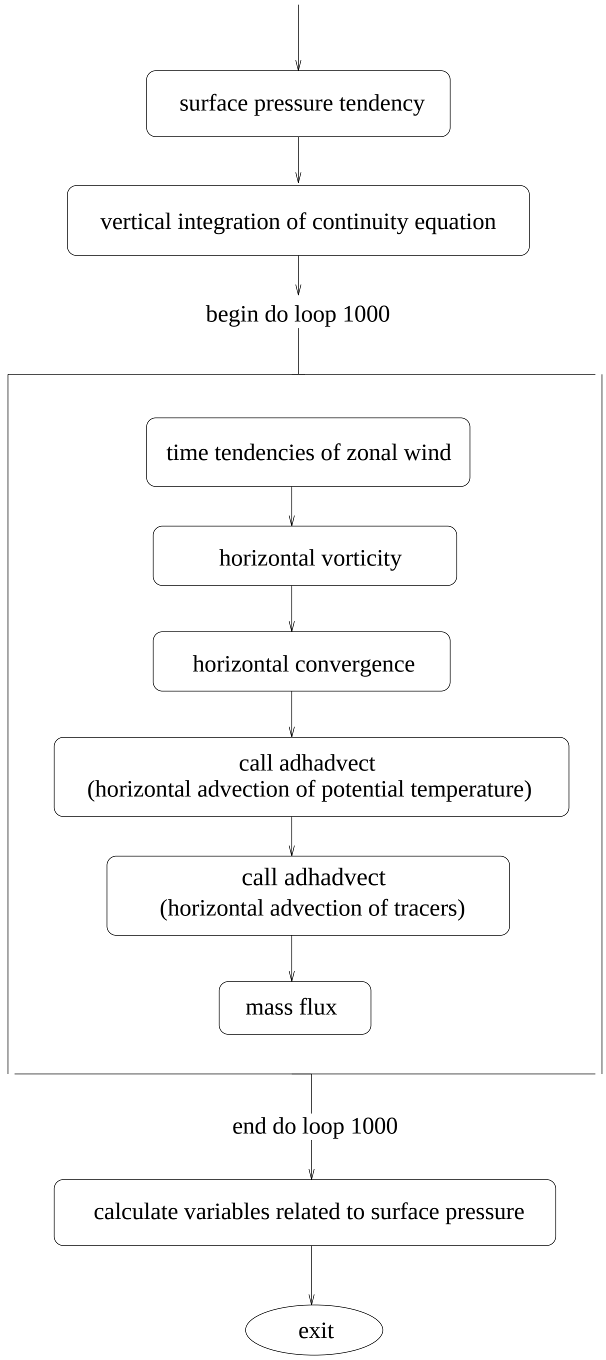


Figure 5: Flow chart of the adjoint of the Aries/GEOS dynamical core.

One has to be cautious when using (17) to check the adjoint code. Sometimes different variables are involved in Q and the identity represents an integral of all the variables on all the grid points. If some variables are very small in magnitude as compared to the other ones, their error may not show up in the integral. In this case we need to change its magnitude in the ADJ code (also change the TLM accordingly) in order to make sure that every and each variable is checked.

We checked the adjoint model segment by segment, do loop by do loop and subroutine by subroutine. With double precision, the identity (17) was always accurate within 14 digits or better. This verified the correctness of the adjoint model against TLM.

4.4 Gradient test of the tangent linear and the adjoint model

In the 4-D variational data assimilation, the adjoint model is used to calculate the gradient of a cost function J to the initial disturbance $\delta\mathbf{X}(t_0)$. J can be defined as:

$$J(\mathbf{X}(t_0)) = \frac{1}{2}\sum_{r=0}^{R}\left(\mathbf{X}(t_r) - \mathbf{X}^{obs}(t_r)\right)^T \mathbf{W}(t_r)\left(\mathbf{X}(t_r) - \mathbf{X}^{obs}(t_r)\right) \tag{18}$$

where $\mathbf{X}(t_r)$ is the model state at time t_r, $\mathbf{X}^{obs}(t_r)$ is the observation; R is the number of time levels for the analyzed fields in the assimilation window; $\mathbf{W}(t_r)$ is an $N \times N$ diagonal weighting matrix, where $\mathbf{W}_u$, $\mathbf{W}_v$, $\mathbf{W}_T$, $\mathbf{W}_{Ps}$ and $\mathbf{W}_q$ are diagonal submatrices consisting of weighting factors for each variable, respectively. Their respective values were $\mathbf{W}_u = 10^{-3}\mathbf{I}\;\; s^2m^{-2}$, $\mathbf{W}_v = 10^{-3}\mathbf{I}\;\; s^2m^{-2}$, $\mathbf{W}_T = 10^{-1}\mathbf{I}\;\; K^{-2}$, $\mathbf{W}_{Ps}= 10^{-2}\mathbf{I}\;\; hPa^{-2}$ and $\mathbf{W}_q= 10^{+4}\mathbf{I}\;\; (kg/kg)^{-2}$.

Now suppose the initial $\mathbf{X}(t_0)$ has a perturbation $\alpha\mathbf{h}$, where α is a small scalar and $\mathbf{h}$ is a vector of unit length (such as the normalized vector of the adjoint model output). According to Taylor expansion we get:

$$J(\mathbf{X}(t_0) + \alpha\mathbf{h}) = J(\mathbf{X}(t_0)) + \alpha\mathbf{h}^T\nabla J(\mathbf{X}(t_0)) + O(\alpha^2), \tag{19}$$

We can define a function of α as:

$$\Phi(\alpha) = \frac{J(\mathbf{X}(t_0) + \alpha\mathbf{h}) - J(\mathbf{X}(t_0))}{\alpha\mathbf{h}^T\nabla J(\mathbf{X}(t_0))} = 1 + O(\alpha). \tag{20}$$

For values of α which are small but not too close to the machine zero, one should expect to obtain values of $\Phi(\alpha)$ which are close to unity. Here the gradient $\nabla J(\mathbf{X}(t_0))$ is calculated by the adjoint model.

We first generated the $\mathbf{X}^{obs}(t_r)$ by integrating the original GCM for 6 hours starting from the analysis data at 03GMT on December 21st, 1991. The difference between this result and the initial field, multiplied by 0.1, was then added to the initial field to be used as the initial condition to generate the trajectory $\mathbf{X}(t_r)$. Fig. 6a and b show the variation of $\Phi(\alpha)$ and $log|1 - \Phi(\alpha)|$ with α, respectively. The abscissas of both of the figures are in log_{10} scale. The integration time was 6 hours. From these figures one can see that as α decreases from 10^{-3} to 10^{-6}, $\Phi(\alpha)$ approaches unity almost linearly and then stays close to unity with a high degree of accuracy until $\alpha \sim 10^{-11}$. This means that for perturbations within this range, the gradient calculated with the adjoint model is reliable.

Please note that the accuracy of the adjoint gradient not only depends on the accuracy of the tangent linear and the adjoint models, but also on the approximation involved in linearizing (18), which in turn depends, to some extent, on the relative magnitude of the distance $\mathbf{X}(t_r) - \mathbf{X}^{obs}(t_r)$ versus the perturbation $\alpha\mathbf{h}$. If we did not multiply the difference by 0.1 when constructing the initial condition for $\mathbf{X}(t_r)$, the result should be even better.

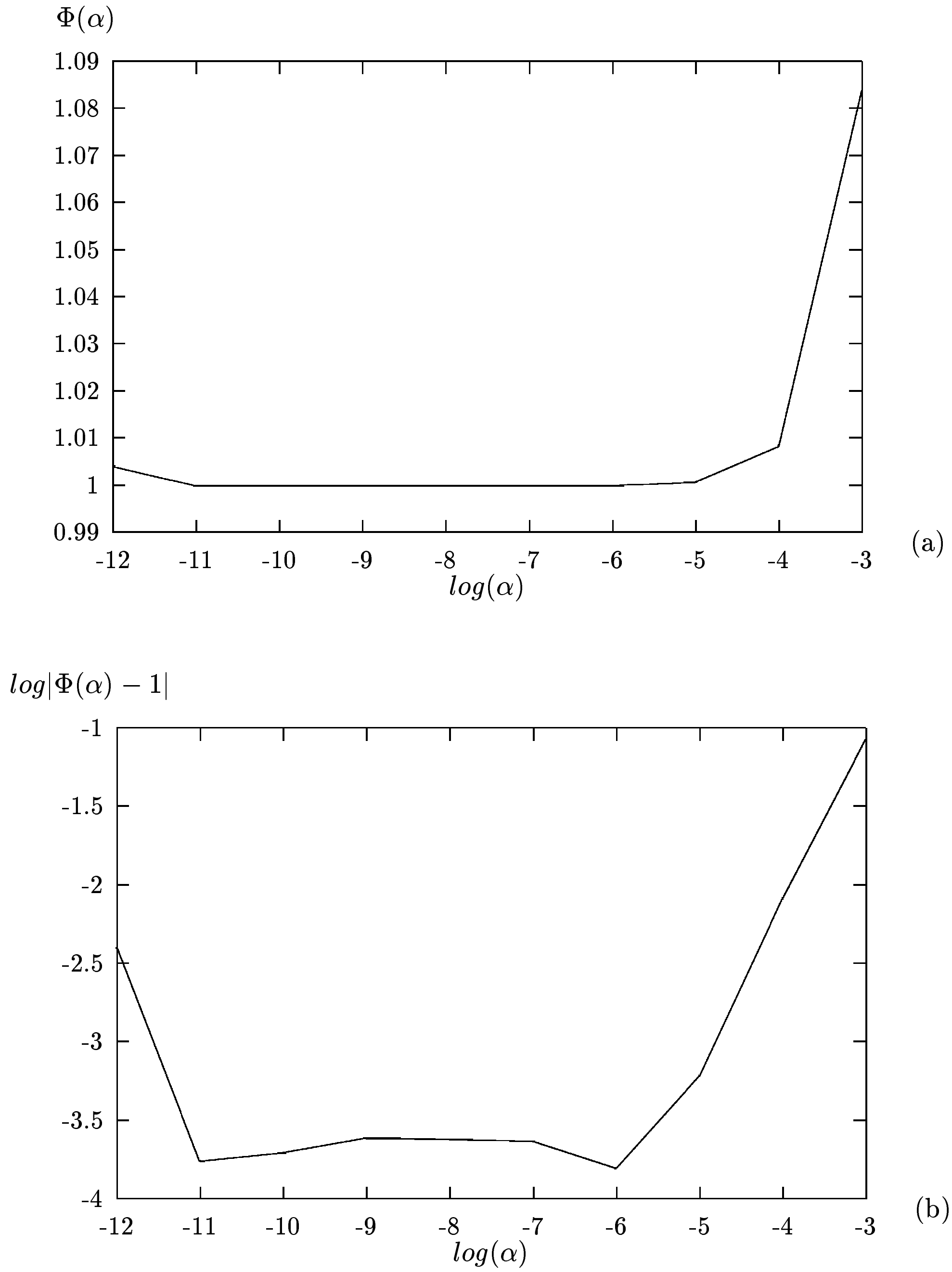


Figure 6: Results of the gradient check of the TLM and ADJ model. (a): Variation of $\Phi(\alpha)$ with respect to $log(\alpha)$; (b): Variation of $|1 - \Phi(\alpha)|$ with respect to $log(\alpha)$.

5 Multitasking of the TLM and Adjoint Model

With the development of massively parallel processing computer technology, today's mainframe supercomputers and workstations tend to have multiprocessors which allow a single job to be split into pieces which execute on different processors concurrently. Such a technique is called parallel processing or multi-tasking. Its main aim is to exploit to the fullest the computer resources and reduce run time of the job. In the present day data assimilation and forecast operations, which is exhausting the capacity of the most powerful computers available, multi-tasking of the models becomes an inevitable trend.

There are different types of algorithms for parallelization, depending on the architecture of the hardware platforms and software systems provided. The version 6.5 of the GEOS-2 GCM, whose TLM and ADJ we developed, is multi-tasked by inserting compiling directives into the code to make the compiler produce the executable file which can run on multi-processors in parallel. This kind of multi-tasking involves less programmer effort to rewrite the code than some other parallel algorithms such as MPI (Message-Passing Interface). In accordance with the original model, the TLM and ADJ were also multi-tasked with the same algorithm. In this chapter we discuss the issue of multi-tasking the codes with emphasis on the adjoint model since there are some parts in it which need special treatment.

5.1 General concepts and algorithmic aspects of multi-tasking

a. General concepts of multi-tasking

The processes that participate in the parallel execution of a task are arranged in a master/slave organization. The original process is the master. It creates several (or zero) slave processes at the beginning of execution of the job according to the user's specification and the resources available. The master and each of the slave processes are called a thread of execution or simply a thread.

The technique of multi-tasking we employed works at the FORTRAN do loop level. Whenever a parallel do loop is encountered, the master asks the slaves for help. The iterations of the do loop is automatically divided among them and each slave executes different indexes of the do loop. For each multitasked do loop, the processors work independently and concurrently and the programmer can not control the sequence of the execution of the iterations. Therefore, the basic principle is that for multi-processing to work properly, the iterations of the loop must not depend on each other; each iteration must stand alone and produce the same answer regardless of whether any other iteration of the loop is executed. This property is called data independence. Otherwise, the loop is called to have data dependency and can not be correctly executed in parallel without modification.

For a loop to be data-independent, no iterations of the loop can write a value into a memory location that is read or written by any other iteration of that loop. It is all right if the same iteration reads and/or writes a memory location repeatedly as long as no others do; it is also all right if many iterations read the same location, as long as none of them write to it. In a FORTRAN program, memory locations are represented by variable names. So we need to examine the way variables are used in the loop, paying particular attention to variables that appear on the left-hand side of assignment statements. If a variable (including elements of an array) is both read and written within a loop, there is possibility of data dependency associated with it.

Our main task in parallelizing the model is to analyze the data dependencies of each do loop and if a do loop is data dependent, to locate the statement(s) in it which can not be made parallel and try to find another way to express it so that it doesn't depend on any other iteration of the loop. If this fails, we have to try to separate this statement from the remainder of the original loop.

b. Multitasking directives on SGI workstation

The Fortran Compiler on Silicon Graphics multiprocessor workstation supports several simple directives to generate multitasked executable for a job. Those compiler directives are added right before a do loop to have it multitasked. The essential one is `C$DOACROSS`. It directs the compiler to parallelize the do loop immediately after it. Note that the compiler recognizes the directive only when the option `-mp` is turned on in the compiling command, otherwise it treats the directive line as a comment line.

The `C$DOACROSS` has clauses, of which the three most often used are: `SHARE`(*variable list*), `LOCAL`(*variable list*) and `LASTLOCAL`(*variable list*).

`SHARE`(*variable list*) is a list of variables or arrays whose value are used by all the participating processors. They carry their respective values into the do loop and if one processor modifies a shared variable or a part of a shared array, its value is updated in all the other processors. Therefore a variable or array can be declared as `SHARE` under the following conditions:

1. it is only read (not written) within the loop, or
2. it is an array where each iteration if the loop use (read and/or write) a different element of the array (i.e., the overwritten element is not used by any other iteration in the same loop)

`LOCAL`(*variable list*) is a list of variables or arrays for which each processor has its own copy. It should be noted that local variables are undefined when they enters the do loop no mater whether or not the variables with the same name are defined prior to this loop. A variable or array can be declared as local under the following conditions:

1. its value does not depend on any other iteration, or
2. its value is used only within a single iteration and is redefined or not used after this do loop.

A local variable is just temporary and its value is unpredictable when it exits the do loop, therefore its value can not be used after the loop. If only the very last value of a variable computed on the vary last iteration is used outside the loop(but would otherwise qualify as a local variable), it can be declared `LASTLOCAL`. The iteration index should be declared as `LOCAL` or `LASTLOCAL`.

The SGI FORTRAN compiler take the variables as `SHARE` by default if they are not listed.

If the list is too long for one line, the `C$&` directive is used to continue the directive into multiple lines.

c. Multitasking directives on CRAY

The concept of multitasking on CRAY supercomputers is generally the same as on the SGI workstations, though the syntaxes are different. The declarer of a parallel do loop is `CMIC$ do all` [*clauses*]. The *clauses* are `SHARED`(*variable list*) and `PRIVATE`(*variable list*). The former is the correspondence to `SHARE`(*variable list*) and the latter is the correspondence to `LOCAL`(*variable list*) on SGI.

On CRAY machines, a parallel region which contains multiple do loops can be declared by adding

`CMIC$PARALLEL`[clauses]

and

`CMIC$ENDPARALLEL`

respectively at the beginning and end of the parallel region.

5.2 Examples of multitasking the tangent linear and adjoint model codes

Since the structure and sequence of the TLM is the same as the original GCM, it is quite straightforward to multitask the TLM, in which the corresponding perturbation variables are added to the directives of the original GCM to declare them as `SHARE` or `LOCAL`.

Most of the do loops in the ADJ model can not be multitasked directly by following their counterparts in the GCM and TLM, mainly due to the fact that in many cases the read-only variables at the right-hand side of a assignment statement in the original code move to the left-hand side and are overwritten. Also many adjoint variables has to be accumulated, thus increasing the possibilities of data dependency. Therefore some manipulations are necessary before the adjoint loops can be parallelized. In the following we discuss some examples of the multitasking of TLM and ADJ codes.

a. example 1

Here is some parts of the code in the original GCM:

```
      REAL PIV(IM,JM)
      REAL PBI(IM,JM)
      REAL PBJ(IM,JM)
      REAL ...

      DO I =1,IM*(JM-1)
       PBI(I,1) = (PIB(I,1) + PIB(IE(I),1)) * HALF
       PIV(I,1) = ONE / PIB(I,1)
      ENDDO

      DO I =1,IM*(JM-2)
       PBJ(I,2) = (PIB(I,2) + PIB(I,1)) * HALF
      ENDDO


      DO 1000 L=1,LM

C    MASS FLUXES

      DO I =1,IM*(JM-1)
       USB(I,1) = DYUIJ(I,1) * PBI(I,1) * UOB(I,1,L)
      ENDDO

      DO I =1,IM*(JM-2)
       VSB(I,2) = DXVIJ(I,2) * PBJ(I,2) * VOB(I,2,L)
      ENDDO

C   COMPUTE CONVERGENCE

      DO I =1,IM*(JM-1)
       PSD(I,1,L) = ( (USB(IW(I),1)-USB(I,1))
     *               + (VSB(I    ,1)-VSB(I,2)) ) * (D2PIN(I,1)*DSG(L))
      ENDDO

1000  CONTINUE


C   VERTICAL INTEGRAL OF CONTINUITY EQUATION

      DO L =2,LM
       DO I =1,IM*(JM-1)
        PSD(I,1,L) = PSD(I,1,L-1) + PSD(I,1,L)
       ENDDO
      ENDDO


      DO 2000 LL=1,2
```

```
      DO 2000 L=LL,LM-1,2

      DO I =1,IM*(JM-1)
          PSD(I,1,L) = PSD(I,1,L) - PSD(I,1,LM)*SIG(L+1)
      psigdot(i,1,L) = psd(i,1,L)   ! PSIGDOT Diagnostic
      ENDDO

       DO I=1,IM*(JM-1)
        ST1 = SIG(L+1)*PSD(I,1,LM) + PSD(I,1,L)
        ST2 = HALF*(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *       * SIG(L+1)*D2PIN(I,1)*PIV(I,1)
        OMG(I,1,L+1) = OMG(I,1,L+1) + BET(I,1) * (ST1 + ST2)
       ENDDO

2000  CONTINUE
```

Please note that loop 2000 and 1000 are large do loops. Here we only pick out a small piece from each of them. This applies to the following examples.

In the above code, `SIG` is a constant vector which stores the sigma value of each vertical level, `VT3`, `VT4` and `BET` are local arrays which are defined elsewhere inside loop 2000.

Several points should be considered when multitasking the above code:

1. In do loop 1000, `USB` and `VSB` are overwritten by each iteration and used only in the same iteration, so they should be declared as `LOCAL`.

2. In the do loop on L right after loop 1000, there is data dependency for L iterations since `PSD(I,1,L)` depends on `PSD(I,1,L-1)`. However, there is no data dependency related to I iteration. Therefore we can parallelize inner I loop instead of outer L loop.

3. In do loop 2000, since the iteration does not extend to L= LM, there is no data dependency between `PSD(I,1,L)` and `PSD(I,1,LM)`, no iteration uses the element which is modified by another iteration, so the inner L loop can be parallelized.

From the above consideration, the code is multitasked as follows:

```
      REAL PIV(IM,JM)
      REAL PBI(IM,JM)
      REAL PBJ(IM,JM)
      REAL ...

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (IM, JM, ONE, HALF, IE)
cmic$*       shared (PIB, PIV, PBI)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
```

```
      DO I =1,IM*(JM-1)
       PBI(I,1) = (PIB(I,1) + PIB(IE(I),1)) * HALF
       PIV(I,1) = ONE / PIB(I,1)
      ENDDO

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (HALF, IM, JM, ONE, PBJ, PIB)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
      DO I =1,IM*(JM-2)
       PBJ(I,2) = (PIB(I,2) + PIB(I,1)) * HALF
      ENDDO

#if (multitask && CRAY)
cmic$ do all shared (DSG, DXVIJ, DYUIJ, D2PIN, IM, IW, JM, LM)
cmic$*       shared (PSD, PBJ, PBI)
cmic$*       shared (UOB, VOB)
cmic$*       private (I, L)
cmic$*       private (USB, VSB)
#endif
#if (multitask && SGI)
c$doacross   local   (I, L)
c$&          local   (USB, VSB)
#endif

      DO 1000 L=1,LM

C   MASS FLUXES

      DO I =1,IM*(JM-1)
       USB(I,1) = DYUIJ(I,1) * PBI(I,1) * UOB(I,1,L)
      ENDDO

      DO I =1,IM*(JM-2)
       VSB(I,2) = DXVIJ(I,2) * PBJ(I,2) * VOB(I,2,L)
      ENDDO

C  COMPUTE CONVERGENCE

      DO I =1,IM*(JM-1)
       PSD(I,1,L) = ( (USB(IW(I),1)-USB(I,1))
     *              + (VSB(I    ,1)-VSB(I,2)) ) * (D2PIN(I,1)*DSG(L))
      ENDDO

1000  CONTINUE


C  VERTICAL INTEGRAL OF CONTINUITY EQUATION

      DO L =2,LM
#if (multitask && CRAY)
```

```
cmic$ do all vector shared (IM, JM, L, PSD) private (I)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
       DO I =1,IM*(JM-1)
        PSD(I,1,L) = PSD(I,1,L-1) + PSD(I,1,L)
       ENDDO
      ENDDO


      DO 2000 LL=1,2

#if (multitask && CRAY)
cmic$ do all shared (D2PIN, HALF, IE, IW, IM, JM, LL, LM)
cmic$*       shared (PSD, PSIGDOT, SIG)
cmic$*       shared (PIV,OMG)
cmic$*       private (BET, I, L, ST1, ST2, VT3, VT4)
#endif
#if (multitask && SGI)
c$doacross   local (BET, I, L, ST1, ST2, VT3, VT4)
#$endif

      DO 2000 L=LL,LM-1,2

      DO I =1,IM*(JM-1)
          PSD(I,1,L) = PSD(I,1,L) - PSD(I,1,LM)*SIG(L+1)
      psigdot(i,1,L) = psd(i,1,L)   ! PSIGDOT Diagnostic
      ENDDO

       DO I=1,IM*(JM-1)
        ST1 = SIG(L+1)*PSD(I,1,LM) + PSD(I,1,L)
        ST2 = HALF*(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *      * SIG(L+1)*D2PIN(I,1)*PIV(I,1)
        OMG(I,1,L+1) = OMG(I,1,L+1) + BET(I,1) * (ST1 + ST2)
       ENDDO

2000  CONTINUE
```

The `multitask` is defined at compiling stage to turn the multitasking on or off. `CRAY` and `SGI` are specified to make sure the right directives are used on different platforms. Please note that the index of the multitasked loop should be declared as `private` or `local`, while the upper and lower boundary should be declared as `shared`.

The linearization of the above code and its multi-tasking were quite straightforward:

```
      REAL PIV(IM,JM)
      REAL PBI(IM,JM)
      REAL PBJ(IM,JM)
      REAL pt_PIV(IM,JM)
      REAL pt_PBI(IM,JM)
```

```
      REAL pt_PBJ(IM,JM)
      REAL ...

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*      shared (IM, JM, ONE, HALF,IE)
cmic$*      shared (PIB, PIV, PBI)
cmic$*      shared (pt_PIB,pt_PIV, pt_PBI)
cmic$*     private (I)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
      DO I =1,IM*(JM-1)
       PBI(I,1) = (PIB(I,1) + PIB(IE(I),1)) * HALF
       PIV(I,1) = ONE / PIB(I,1)
#ifdef _TLM_
       pt_PBI(I,1) = (pt_PIB(I,1) + pt_PIB(IE(I),1)) * HALF
       pt_PIV(I,1) = -PIV(I,1)*PIV(I,1)*pt_PIB(I,1)
#endif
      END DO

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*      shared (HALF, IM, JM, ONE, PBJ, PIB)
cmic$*      shared (pt_PBJ, pt_PIB)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
      DO I =1,IM*(JM-2)
       PBJ(I,2) = (PIB(I,2) + PIB(I,1)) * HALF
#ifdef _TLM_
       pt_PBJ(I,2) = (pt_PIB(I,2) + pt_PIB(I,1)) * HALF
#endif
      ENDDO

#if (multitask && CRAY)
cmic$ do all shared  (DSG, DXVIJ, DYUIJ, D2PIN, IE, IM, IW, JM, LM)
cmic$*      shared  (PSD, PBJ, PBI)
cmic$*      shared  (UOB, VOB)
cmic$*      private (I, L)
cmic$*      private (USB, VSB)
cmic$*      shared  (pt_PSD, pt_PBJ, pt_PBI)
cmic$*      shared  (pt_UOB, pt_VOB)
cmic$*      private (TMPC)
cmic$*      private (pt_USB, pt_VSB)
#endif
#if (multitask && SGI)
c$doacross   local   (I, L)
c$&          local   (USB, VSB)
c$&          local   (TMPC)
c$&          local   (pt_USB, pt_VSB)
#endif
```

```
      DO 1000 L=1,LM

C   MASS FLUXES

      DO I =1,IM*(JM-1)
       USB(I,1) = DYUIJ(I,1) * PBI(I,1) * UOB(I,1,L)
      ENDDO

      DO I =1,IM*(JM-2)
       VSB(I,2) = DXVIJ(I,2) * PBJ(I,2) * VOB(I,2,L)
      ENDDO

#ifdef _TLM_
      DO I =1,IM*(JM-1)
       pt_USB(I,1) = DYUIJ(I,1) * (pt_PBI(I,1) * UOB(I,1,L)
     &          +  PBI(I,1) * pt_UOB(I,1,L))
      ENDDO

      DO I =1,IM*(JM-2)
       pt_VSB(I,2) = DXVIJ(I,2) *( PBJ(I,2) * pt_VOB(I,2,L)
     &             + pt_PBJ(I,2) * VOB(I,2,L) )
      ENDDO
#endif

C  COMPUTE CONVERGENCE

      DO I =1,IM*(JM-1)
       PSD(I,1,L) = ( (USB(IW(I),1)-USB(I,1))
     *              + (VSB(I    ,1)-VSB(I,2)) ) * (D2PIN(I,1)*DSG(L))

#ifdef _TLM_
       pt_PSD(I,1,L) = ( (pt_USB(IW(I),1)-pt_USB(I,1))
     *              + (pt_VSB(I,1)-pt_VSB(I,2)) ) * (D2PIN(I,1)*DSG(L))
#endif
      ENDDO

1000  CONTINUE


C  VERTICAL INTEGRAL OF CONTINUITY EQUATION

      DO L =2,LM
#if (multitask && CRAY)
cmic$ do all vector shared (IM, JM, L, PSD) private (I)
cmic$*              shared (pt_PSD)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
       DO I =1,IM*(JM-1)
        PSD(I,1,L) = PSD(I,1,L-1) + PSD(I,1,L)
#ifdef _TLM_
        pt_PSD(I,1,L) = pt_PSD(I,1,L-1) + pt_PSD(I,1,L)
```

```
#endif
       ENDDO
      ENDDO


      DO 2000 LL=1,2

#if (multitask && CRAY)
cmic$ do all shared (D2PIN, HALF, IE, IW, IM, JM, LL, LM)
cmic$*       shared (PSD, PSIGDOT, SIG, TMPC)
cmic$*       shared (PIV,OMG)
cmic$*       shared (pt_PSD, pt_PSIGDOT, pt_PIV, pt_OMG)
cmic$*       private (BET, I, L, ST1, ST2, VT3, VT4)
cmic$*       private (pt_BET, pt_ST1, pt_ST2, pt_VT3, pt_VT4)
#endif
#if (multitask && SGI)
c$doacross   local (BET, I, L, ST1, ST2, VT3, VT4)
c$&          local (pt_BET, pt_ST1, pt_ST2, pt_VT3, pt_VT4)
#endif

      DO 2000 L=LL,LM-1,2

      DO I =1,IM*(JM-1)
          PSD(I,1,L) = PSD(I,1,L) - PSD(I,1,LM)*SIG(L+1)
      psigdot(i,1,L) = psd(i,1,L)   ! PSIGDOT Diagnostic
      ENDDO

#ifdef _TLM_
      DO I =1,IM*(JM-1)
          pt_PSD(I,1,L) = pt_PSD(I,1,L) - pt_PSD(I,1,LM)*SIG(L+1)
      pt_psigdot(i,1,L) = pt_psd(i,1,L)   ! PSIGDOT Diagnostic
      ENDDO
#endif

       DO I=1,IM*(JM-1)
        ST1 = SIG(L+1)*PSD(I,1,LM) + PSD(I,1,L)

#ifdef _TLM_
        pt_ST1 = SIG(L+1)*pt_PSD(I,1,LM) + pt_PSD(I,1,L)
        TMPC=HALF*SIG(L+1)*D2PIN(I,1)
#endif

        ST2 = HALF*(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *      * SIG(L+1)*D2PIN(I,1)*PIV(I,1)
        OMG(I,1,L+1) = OMG(I,1,L+1) + BET(I,1) * (ST1 + ST2)

#ifdef _TLM_
        pt_ST2 = TMPC* ( (pt_VT3(IW(I),1)+pt_VT3(I,1)+pt_VT4(I,2)
     *         +pt_VT4(I,1))*PIV(I,1)
     &         +(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *         *pt_PIV(I,1) )
        pt_OMG(I,1,L+1) = pt_OMG(I,1,L+1) + pt_BET(I,1) * (ST1 + ST2)
     &                  +  BET(I,1) * (pt_ST1 + pt_ST2)
#endif
```

```
      ENDDO

 2000  CONTINUE
```

Note that in TLM we calculated the basic field exactly as in the original GCM. Normally we only need to add the perturbation variables into the list of `SHARED`, `PRIVATE` or `LOCAL` the same way as their original variables are declared in the GCM.

Now we develop the adjoint for this loop. According to the aforementioned convention, we would write the serial adjoint code as the following:

```
      REAL PIV(IM,JM)
      REAL PBI(IM,JM)
      REAL PBJ(IM,JM)
      REAL ad_PIV(IM,JM)
      REAL ad_PBI(IM,JM)
      REAL ad_PBJ(IM,JM)
      REAL ...


      DO 2000 LL=2,1,-1
      DO 2000 L=LM-LL,LL,-2

       DO I=IM*(JM-1),1,-1
        ST1 = SIG(L+1)*PSD(I,1,LM) + PSD(I,1,L)
        ST2 = HALF*(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *      * SIG(L+1)*D2PIN(I,1)*PIV(I,1)
        TMPC=HALF* SIG(L+1)*D2PIN(I,1)

         ad_ST1 =  + BET(I,1)*  ad_OMG(I,1,L+1)
         ad_ST2 =  + BET(I,1)*  ad_OMG(I,1,L+1)
         ad_BET(I,1) = ad_BET(I,1) + (ST1 + ST2)*  ad_OMG(I,1,L+1)
         ad_OMG(I,1,L+1) = ad_OMG(I,1,L+1)

         ad_PIV(I,1) = ad_PIV(I,1)+TMPC*(VT3(IW(I),1)+VT3(I,1)+
     &                  VT4(I,2)+VT4(I,1)) *ad_ST2
         ad_VT3(IW(I),1) = ad_VT3(IW(I),1) + TMPC*PIV(I,1)*ad_ST2
         ad_VT3(I,1) = ad_VT3(I,1)+ TMPC*PIV(I,1)*ad_ST2
         ad_VT4(I,2) = ad_VT4(I,2)+ TMPC*PIV(I,1)*ad_ST2
         ad_VT4(I,1) = ad_VT4(I,1)+ TMPC*PIV(I,1)*ad_ST2
         ad_ST2 = 0.0

         ad_PSD(I,1,LM) = ad_PSD(I,1,LM) +SIG(L+1)*ad_ST1
         ad_PSD(I,1,L) = ad_PSD(I,1,L) +ad_ST1
         ad_ST1 = 0.0

       ENDDO

      DO I =1,IM*(JM-1)
         ad_psd(i,1,L) =  ad_psd(i,1,L) +  ad_psigdot(i,1,L)
         ad_psigdot(i,1,L) = 0.0
```

```
         ad_PSD(I,1,LM) = ad_PSD(I,1,LM) - SIG(L+1)* ad_PSD(I,1,L)
C        ad_PSD(I,1,L) = ad_PSD(I,1,L)
      ENDDO

2000  CONTINUE

C  VERTICAL INTEGRAL OF CONTINUITY EQUATION

      DO L =LM, 2,-1

       DO I =1,IM*(JM-1)
         ad_PSD(I,1,L-1) = ad_PSD(I,1,L-1) +ad_PSD(I,1,L)
         ad_PSD(I,1,L) = ad_PSD(I,1,L)
       ENDDO
      ENDDO


      DO 1000 L=LM,1,-1

C  COMPUTE CONVERGENCE

      DO I = IM*(JM-1), 1,-1
       ad_tmp = (D2PIN(I,1)*DSG(L))* ad_PSD(I,1,L)
       ad_VSB(I,1) = ad_VSB(I,1) + ad_tmp
       ad_VSB(I,2) = ad_VSB(I,2) - ad_tmp
       ad_USB(IW(I),1) = ad_USB(IW(I),1) + ad_tmp
       ad_USB(I,1) = ad_USB(I,1) - ad_tmp
       ad_PSD(I,1,L) = 0.0
      ENDDO

C   MASS FLUXES

      DO I =1,IM*(JM-2)

       ad_tmp = DXVIJ(I,2) *ad_VSB(I,2)

       ad_VOB(I,2,L)= ad_VOB(I,2,L) + PBJ(I,2) *ad_tmp
       ad_PBJ(I,2) = ad_PBJ(I,2)  +VOB(I,2,L) *ad_tmp
       ad_VSB(I,2) =  0.0

      ENDDO

      DO I =1,IM*(JM-1)

        ad_tmp = DYUIJ(I,1) *ad_USB(I,1)

        ad_UOB(I,1,L)= ad_UOB(I,1,L) +  PBI(I,1) *ad_tmp
        ad_PBI(I,1)= ad_PBI(I,1) +UOB(I,1,L) *ad_tmp
        ad_USB(I,1) =  0.0
      ENDDO

 1000  CONTINUE

       DO I =1,IM*(JM-2)
```

```
  ad_PIB(I,2) = ad_PIB(I,2) + HALF* ad_PBJ(I,2)
  ad_PIB(I,1) = ad_PIB(I,1) + HALF* ad_PBJ(I,2)
  ad_PBJ(I,2) = 0.0
 ENDDO

DO I =IM*(JM-1),1,-1

 PIV(I,1) = ONE / PIB(I,1)

  ad_PIB(I,1)= ad_PIB(I,1) -PIV(I,1)*PIV(I,1)* ad_PIV(I,1)
  ad_PIV(I,1) = 0.0
  ad_PIB(I,1)= ad_PIB(I,1) + HALF * ad_PBI(I,1)
  ad_PIB(IE(I),1) = ad_PIB(IE(I),1) + HALF * ad_PBI(I,1)
  ad_PBI(I,1) = 0.0
END DO
```

Please note that in the second do statement, the iteration on L begins with LM-L instead of LM-1, otherwise it will do the same iterations when LL=1 as when LL=2.

After testing this code with one CPU and verifying that it is the right adjoint, we went on to multitask it. As in the TLM, we declare the adjoint variables the same as their original GCM variables. But in the adjoint code new data dependency may appear, therefore we have to re-analyze carefully the data dependency for each do loop. For the above code, several points should be aware of:

1. The ad_PSD(I,1,LM) in loop 2000 is overwritten by each iteration, so we have to think of ways to break the data dependency associated with it. The original GCM and TLM would not have this problem since PSD(I,1,LM) and pt_PSD(I,1,LM) are only read but not written in them.

   Since ad_PSD(I,1,LM) is related to other local variables inside the large do loop 2000, it is not easy to separate it from this do loop. Therefore we need to add another 3-dimensional SHARED variable ad_PSDTMP to temporarily store the results for each iteration and then accumulate them among L to ad_PSD(I,1,LM) after this do loop.

2. ad_PIV also has a problem. In the GCM and TLM code, it is a 2-dimensional variable which is calculated before do loop 2000 and is only read inside this loop. However in the ADJ code, not only is it overwritten, but also the results of each iteration need to be accumulated. Therefore to break the data dependency, another dimension has to be added to ad_PIV to store the results of each iteration in order to prevent it from being overwritten. After this loop, the results are to be accumulated to the layer (L = LM) for later use.

3. ad_PBI and ad_PBJ both have the same problem as ad_PIV does. They also have to be augmented to 3-dimensional in order to store the result of each iteration on L.

Now the multitasked adjoint code is as follows:

```
REAL PIV(IM,JM)
REAL PBI(IM,JM)
```

```
      REAL PBJ(IM,JM)
      REAL ad_PBI(IM,JM,LM)
      REAL ad_PBJ(IM,JM,LM)
      REAL ad_PIV(IM,JM,LM)
      REAL ad_PSDTMP(IM,JM,LM)
      REAL ...

        call setzero(ndim3,ad_PSDTMP)

      DO 2000 LL=2,1,-1

#if (multitask && CRAY)
cmic$ do all shared (D2PIN, HALF, IE, IW, IM, JM, LL, LM)
cmic$*       shared (PSD, PSIGDOT, SIG, TMPC)
cmic$*       shared (PIV,OMG)
cmic$*       shared (ad_PSD, ad_PSDTMP, ad_PSIGDOT, ad_PIV, ad_OMG)
cmic$*       private (BET, I, L, ST1, ST2, VT3, VT4)
cmic$*       private (ad_BET, ad_ST1, ad_ST2, ad_VT3, ad_VT4)
#endif
#if (multitask && SGI)
c$doacross   local (BET, I, L, ST1, ST2, VT3, VT4)
c$&          local (ad_BET, ad_ST1, ad_ST2, ad_VT3, ad_VT4)
#$endif

      DO 2000 L=LM-LL,LL,-2

       DO I=IM*(JM-1),1,-1
        ST1 = SIG(L+1)*PSD(I,1,LM) + PSD(I,1,L)
        ST2 = HALF*(VT3(IW(I),1)+VT3(I,1)+VT4(I,2)+VT4(I,1))
     *       * SIG(L+1)*D2PIN(I,1)*PIV(I,1)
        TMPC=HALF* SIG(L+1)*D2PIN(I,1)


         ad_ST1 =  + BET(I,1)*  ad_OMG(I,1,L+1)
         ad_ST2 =  + BET(I,1)*  ad_OMG(I,1,L+1)
         ad_BET(I,1) = ad_BET(I,1) + (ST1 + ST2)*  ad_OMG(I,1,L+1)
         ad_OMG(I,1,L+1) = ad_OMG(I,1,L+1)

         ad_PIV(I,1,L) = ad_PIV(I,1,L)+TMPC*(VT3(IW(I),1)+VT3(I,1)+
     &                   VT4(I,2)+VT4(I,1)) *ad_ST2
         ad_VT3(IW(I),1) = ad_VT3(IW(I),1) + TMPC*PIV(I,1)*ad_ST2
         ad_VT3(I,1) = ad_VT3(I,1)+ TMPC*PIV(I,1)*ad_ST2
         ad_VT4(I,2) = ad_VT4(I,2)+ TMPC*PIV(I,1)*ad_ST2
         ad_VT4(I,1) = ad_VT4(I,1)+ TMPC*PIV(I,1)*ad_ST2
         ad_ST2 = 0.0

         ad_PSDTMP(I,1,L) = ad_PSDTMP(I,1,L) +SIG(L+1)*ad_ST1
         ad_PSD(I,1,L) = ad_PSD(I,1,L) +ad_ST1
         ad_ST1 = 0.0

       ENDDO

      DO I =1,IM*(JM-1)
         ad_psd(i,1,L) =  ad_psd(i,1,L) +  ad_psigdot(i,1,L)
```

```
         ad_psigdot(i,1,L) = 0.0
         ad_PSDTMP(I,1,L) = ad_PSDTMP(I,1,L) - SIG(L+1)* ad_PSD(I,1,L)
c        ad_PSD(I,1,L) = ad_PSD(I,1,L)
      ENDDO

2000  CONTINUE

#if (multitask && CRAY)
cmic$ do all shared (ad_PSD, ad_PSDTMP, SIG, LM, IM, JM)
cmic$*       private (I,L)
#endif
#if (multitask && SGI)
c$doacross   local   (I,L)
#endif
      DO 2030 I =1,IM*(JM-1)

        do L=1, LM-1
          ad_PSD(I,1,LM) = ad_PSD(I,1,LM) + ad_PSDTMP(I,1,L)
        ENDDO

2030  continue

      call reduce (ad_piv, im*jm, LM)

C  VERTICAL INTEGRAL OF CONTINUITY EQUATION

      DO L =LM, 2,-1
#if (multitask && CRAY)
cmic$ do all vector shared (IM, JM, L, PSD) private (I)
cmic$*              shared (ad_PSD)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif

       DO I =IM*(JM-1),1,-1

         ad_PSD(I,1,L-1) = ad_PSD(I,1,L-1) +ad_PSD(I,1,L)
         ad_PSD(I,1,L) = ad_PSD(I,1,L)
       ENDDO
      ENDDO


#if (multitask && CRAY)
cmic$ do all shared (DSG, DXVIJ, DYUIJ, D2PIN, IE, IM, IW, JM, LM)
cmic$*       shared (PSD, PBJ, PBI)
cmic$*       shared (UOB, VOB)
cmic$*       private (I, L)
cmic$*       private (USB, VSB)
cmic$*       shared (ad_PSD, ad_PBJ, ad_PBI)
cmic$*       shared (ad_UOB, ad-VOB)
cmic$*       private(TMPC)
cmic$*       private (ad_USB, ad_VSB, ad_tmp)
#endif
```

```
#if (multitask && SGI)
c$doacross   local   (I, L)
c$&          local   (USB, VSB)
c$&          local   (TMPC)
c$&          local   (ad_USB, ad_VSB, ad_tmp)
#endif


      DO 1000 L=1,LM

C  COMPUTE CONVERGENCE

      DO I = IM*(JM-1), 1,-1
       ad_tmp = (D2PIN(I,1)*DSG(L))* ad_PSD(I,1,L)
       ad_VSB(I,1) = ad_VSB(I,1) + ad_tmp
       ad_VSB(I,2) = ad_VSB(I,2) - ad_tmp
       ad_USB(IW(I),1) = ad_USB(IW(I),1) + ad_tmp
       ad_USB(I,1) = ad_USB(I,1) - ad_tmp
       ad_PSD(I,1,L) = 0.0
      ENDDO

C   MASS FLUXES

      DO I =1,IM*(JM-2)

       ad_tmp = DXVIJ(I,2) *ad_VSB(I,2)

       ad_VOB(I,2,L) = ad_VOB(I,2,L) + PBJ(I,2) *ad_tmp
       ad_PBJ(I,2,L) = ad_PBJ(I,2,L)  +VOB(I,2,L) *ad_tmp
       ad_VSB(I,2) =  0.0

      ENDDO

      DO I =1,IM*(JM-1)

        ad_tmp = DYUIJ(I,1) *ad_USB(I,1)

        ad_UOB(I,1,L)= ad_UOB(I,1,L) + PBI(I,1)   *ad_tmp
        ad_PBI(I,1,L)= ad_PBI(I,1,L) + UOB(I,1,L) *ad_tmp
        ad_USB(I,1) =  0.0
      ENDDO

 1000  CONTINUE

       call reduce (ad_pbi, im*jm, LM)
       call reduce (ad_pbj, im*jm, LM)

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (IM, JM, HALF)
cmic$*       shared (PIB, PBJ)
cmic$*       shared (ad_PIB, ad_PBJ)
cmic$*      private (I)
#endif
```

```
#if (multitask && SGI)
c$doacross   local (i)
#endif
       DO I =1,IM*(JM-2)
        ad_PIB(I,2) = ad_PIB(I,2) + HALF* ad_PBJ(I,2,LM)
       ENDDO

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (IM, JM, HALF)
cmic$*       shared (PIB, PBJ)
cmic$*       shared (ad_PIB, ad_PBJ)
cmic$*      private (I)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
       DO I =1,IM*(JM-2)
        ad_PIB(I,1) = ad_PIB(I,1) + HALF* ad_PBJ(I,2,LM)
        ad_PBJ(I,2,LM) = 0.0
       ENDDO

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (IM, JM, ONE, HALF)
cmic$*       shared (PIB, PIV, PBI)
cmic$*       shared (ad_PIB,ad_PIV, ad_PBI)
cmic$*      private (I)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
      DO I =IM*(JM-1),1,-1
        PIV(I,1) = ONE / PIB(I,1)
        ad_PIB(I,1)= ad_PIB(I,1) -PIV(I,1)*PIV(I,1)* ad_PIV(I,1,LM)
        ad_PIV(I,1,LM) = 0.0
        ad_PIB(I,1)= ad_PIB(I,1) + HALF * ad_PBI(I,1,LM)
      END DO

#if (multitask && CRAY)
cmic$ do all vector private (I)
cmic$*       shared (IM, JM, ONE, HALF,IE)
cmic$*       shared (PIB, PBI)
cmic$*       shared (ad_PIB, ad_PBI)
cmic$*      private (I)
#endif
#if (multitask && SGI)
c$doacross   local (i)
#endif
      DO I =IM*(JM-1),1,-1
        ad_PIB(IE(I),1) = ad_PIB(IE(I),1) + HALF * ad_PBI(I,1,LM)
        ad_PBI(I,1,LM) = 0.0
      END DO
```

Some remarks on the above code:

1. The new subroutine `reduce (A, im*jm, LM)` is used to accumulate the values of all the vertical levels of the array `A(im,jm,LM)` to the last level `LM`.
2. The subroutine `setzero(N, A)` serves to set an array `A(N)` to zero.
3. For do loop 2030, the multitasking is carried out on I iteration instead of L, since L iterations has data dependency.
4. Always bear in mind that all the local variables should be initialized in the loop, especially for the adjoint variables which in many cases need to be added to themselves.
5. After the value of `ad_PIV` is reduced to the last level `LM`, the later reference to it should use `ad_PIV(I,1,LM)`. If later do loops also need to use the three-dimensional `ad_PIV` the same way as above, one needs to make sure that the value of `ad_PIV(I,1,LM)` is stored and accumulated to the result of the later loops properly. The same applies to `ad_PBI` and `ad_PBJ`.
6. The two do loops right after loop 1000 were separated since there is data dependency between `ad_PIB(I,1)` and `ad_PIB(I,2)`, by noting that the iteration on I goes to `IM*(JM-2)`. The last two do loops were also separated due to the data dependency related to `ad_PIB(IE(I),1)` and `ad_PIB(I,1)`. which will be explained in more detail in the following example.
7. After a do loop is multitasked, the sequence of its iterations is no longer significant.

b. example 2
Following is another piece of code in GCM:

```
      REAL DPX(IM,JM)

#if (multitask && CRAY)
cmic$ do all vector shared (DPX, IE, IM, JM, VT1) private (I)
#endif
#if (multitask && SGI)
c$doacross   local   (I)
#endif
      DO I=1,IM*(JM-1)
       DPX(I,1) = (VT1(IE(I),1) - VT1(I,1))
      ENDDO


#if (multitask && CRAY)
cmic$ do all shared (DSG, DXUIN, HALF, IE, IM, JM, LM)
cmic$*       shared (DPX, PHI, UOI)
cmic$*       private (GAM, I, L, ST1)
#endif
#if (multitask && SGI)
```

```
c$doacross   local   (GAM, I, L, ST1)
#endif

      DO 3000 L=1,LM
      DO I =1,IM*(JM-1)
       ST1        = (PHI(IE(I),1,L) - PHI(I,1,L))
     *            + DPX(I,1) * HALF*(GAM(IE(I),1)+GAM(I,1))
       UOI(I,1,L) = UOI(I,1,L) - ST1*DXUIN(I,1)
      ENDDO

3000  CONTINUE
```

The IE(I) is defined before as:

```
      do i=1,im*jm

       IF(MOD(I,im).NE.0) THEN
        IE(i) = i + 1
       ELSE
        IE(i) = i + 1 - im
       ENDIF

     enddo
```

That is, IE(I) denotes the eastern point of I.
In the TLM, the above two loops are linearized and multi-tasked as follows:

```
      REAL DPX(IM,JM)
      REAL pt_DPX(IM,JM)

#if (multitask && CRAY)
cmic$ do all vector shared (DPX, IE, IM, JM, VT1) private (I)
#ifdef _TLM_
cmic$*       shared (pt_DPX, pt_VT1)
#endif
#endif
#if (multitask && SGI)
c$doacross   local   (I)
#endif
      DO I=1,IM*(JM-1)
       DPX(I,1) = (VT1(IE(I),1) - VT1(I,1))
#ifdef _TLM_
       pt_DPX(I,1) = (pt_VT1(IE(I),1) - pt_VT1(I,1))
#endif
      ENDDO
```

```
#if (multitask && CRAY)
cmic$ do all shared (DSG, DXUIN, HALF, IE, IM, JM, LM)
cmic$*       shared (DPX, PHI, UOI)
cmic$*       shared (pt_DPX, pt_PHI, pt_UOI)
cmic$*       private (GAM, I, L, ST1)
cmic$*       private (pt_GAM, pt_ST1)
#endif
#if (multitask && SGI)
c$doacross   local (GAM, I, L, ST1)
c$&          local (pt_GAM, pt_ST1)
#endif

      DO 3000 L=1,LM
      DO I =1,IM*(JM-1)
       ST1        = (PHI(IE(I),1,L) - PHI(I,1,L))
     *            + DPX(I,1) * HALF*(GAM(IE(I),1)+GAM(I,1))
       UOI(I,1,L) = UOI(I,1,L) - ST1*DXUIN(I,1)

       pt_ST1        = (pt_PHI(IE(I),1,L) - pt_PHI(I,1,L))
     *            + pt_DPX(I,1) * HALF*(GAM(IE(I),1)+GAM(I,1))
     *            + DPX(I,1) * HALF*(pt_GAM(IE(I),1)+pt_GAM(I,1))

       pt_UOI(I,1,L) = pt_UOI(I,1,L) - pt_ST1*DXUIN(I,1)

      ENDDO

3000  CONTINUE
```

Note that GAM and pt_GAM are defined at other parts inside loop 3000. The serial adjoint of the above code would be like this:

```
      REAL DPX(IM,JM)
      REAL ad_DPX(IM,JM)

      DO 3000 L=LM,1,-1

      DO I = IM*(JM-1),1,-1
        ad_ST1 = - DXUIN(I,1) * ad_UOI(I,1,L)
        ad_UOI(I,1,L) = ad_UOI(I,1,L)

        ad_GAM(IE(I),1) = ad_GAM(IE(I),1) + HALF*DPX(I,1) * ad_ST1
        ad_GAM(I,1) = ad_GAM(I,1) + HALF*DPX(I,1) * ad_ST1
        ad_DPX(I,1) = ad_DPX(I,1) + HALF*(GAM(IE(I),1)
     &               +GAM(I,1)) * ad_ST1
        ad_PHI(IE(I),1,L) = ad_PHI(IE(I),1,L) + ad_ST1
        ad_PHI(I,1,L) = ad_PHI(I,1,L) - ad_ST1

        ad_ST1 = 0.0
```

```
      ENDDO

 3000 CONTINUE

      DO I=IM*(JM-1),1,-1
        ad_VT1(IE(I),1)= ad_VT1(IE(I),1) +ad_DPX(I,1)
        ad_VT1(I,1) = ad_VT1(I,1) - ad_DPX(I,1)
        ad_DPX(I,1) = 0.0
      ENDDO
```

We found that the above adjoint code could not be directly multitasked as in the GCM and TLM for the following two reasons:

1. `ad_DPX` in do loop 3000 should be declared as `SHARED`, since its value is accumulated among the iterations and used after this loop. However since it is a two-dimensional array, so each iteration overwrites it, causing data dependency. Therefore another dimension is required to store the result of each iteration. `DPX` in the GCM and `pt_DPX` in the TLM do not have such a problem since they are only read.
2. The do loop on I also has data dependency since the iteration I modifies the `ad_VT1(IE(I),1)` as well as `ad_VT1(I,1)`. To break the data dependency, this do loop has to be separated into two. In the original code they are both read, therefore do not cause such a problem.

Now the multitasked code read like this:

```
      REAL DPX(IM,JM)
      REAL ad_DPX(IM,JM,LM)

#if (multitask && CRAY)
cmic$ do all shared (DSG, DXUIN, HALF, IE, IM, JM, LM)
cmic$*       shared (DPX, PHI, UOI)
cmic$*       shared (ad_DPX, ad_PHI, ad_UOI)
cmic$*       private (GAM, I, L, ST1)
cmic$*       private (ad_GAM, ad_ST1)
#endif
#if (multitask && SGI)
c$doacross   local (GAM, I, L, ST1)
c$&          local (ad_GAM, ad_ST1)
#endif

      DO 3000 L=LM,1,-1

      DO I = IM*(JM-1),1,-1
        ad_ST1 = - DXUIN(I,1) * ad_UOI(I,1,L)
        ad_UOI(I,1,L) = ad_UOI(I,1,L)

        ad_GAM(IE(I),1) = ad_GAM(IE(I),1) + HALF*DPX(I,1) * ad_ST1
        ad_GAM(I,1) = ad_GAM(I,1) + HALF*DPX(I,1) * ad_ST1
```

```
        ad_DPX(I,1, L) =  + HALF*(GAM(IE(I),1)
     &                 +GAM(I,1)) * ad_ST1
        ad_PHI(IE(I),1,L) = ad_PHI(IE(I),1,L) + ad_ST1
        ad_PHI(I,1,L) = ad_PHI(I,1,L) - ad_ST1

        ad_ST1 = 0.0

      ENDDO

3000  CONTINUE

       call reduce (ad_dpx, im*jm, LM)

#if (multitask && CRAY)
cmic$ do all vector shared (IE, IM, JM, VT1) private (I)
cmic$*       shared (ad_DPX, ad_VT1)
#endif
#if (multitask && SGI)
c$doacross   local   (I)
#endif

      DO I=IM*(JM-1),1,-1

        ad_VT1(IE(I),1)= ad_VT1(IE(I),1) +ad_DPX(I,1, LM)
      ENDDO

#if (multitask && CRAY)
cmic$ do all vector shared (IM, JM, VT1) private (I)
cmic$*       shared (ad_DPX, ad_VT1)
#endif
#if (multitask && SGI)
c$doacross   local   (I)
#endif

      DO I=IM*(JM-1),1,-1

        ad_VT1(I,1) = ad_VT1(I,1) - ad_DPX(I,1, LM)
        ad_DPX(I,1, LM) = 0.0
      ENDDO
```

5.3 Evaluation of the speed-up by multitasking

In order to evaluate the gain from multitasking the codes, we ran the models with the multitasking turned either on or off and stored the wall-time consumed by each of the three models. The speed-up ratio was calculated by the following formula:

$$speed-up\ ratio = \frac{wall-time\ without\ multitasking}{wall-time\ with\ multitasking} \tag{21}$$

With 8 cpus, the ratios we obtained for TLM and ADJ were both around 2, while that for GCM was about 3.6. In this experiment the length of our integration period was only 6 hours. Longer integration times would yield better speed-up ratios due to relatively less overhead. Also note that we were not running this test on a dedicated system. However we can see that there are still much room for further optimization.

Acknowledgments
We acknowledge the support by NASA grant NAG 5-1660 managed by Dr. Ken Bergman, section head of Climate Modeling, NASA headquarters. Thanks are also due to Dr. Samson H. Cheung from MRJ Technology Solutions for his kind help in our multitasking of the models.

6 References

Arakawa, A. and V. R. Lamb, 1981: A potential enstrophy and energy conserving scheme for the shallow water equations, *Mon. Wea. Rev.*, **109**, 18-36.

Arakawa, A. and M. J. Suarez, 1983: Vertical differencing of the primitive equations in sigma coordinates, *Mon. Wea. Rev.*, **111**, 34-45.

Asselin, R., 1972: Frequency filter for time integrations. *Mon. Wea. Rev.*, **100**, 487-490.

Burridge, D. M. and J. Haseler, 1977: A model for medium range weather forecasting - adiabatic formulation, *Tech. Report, No. 4, European Center for Medium Range Weather Forecasts*, Bracknell, Berkshire, UK.

DAO, 1996: Algorithm Theoretical Basis Document Version 1.01, Data Assimilation Office, Goddard Space Flight Center, NASA.

Fox-Rabinovitz, M., H. M. Helfand, A. Hou, L. L. Takacs, and A. Molod, 1991: Numerical experiments on forecasting, climate simulation and data assimilation with the new 17 layer GLA GCM, *Proceedings of the AMS Ninth Conference on Numerical Weather Prediction.* 21-25 October 1991, Denver, CO, 506-509.

Helfand, H. M., M. Fox-Rabinovitz, L. L. Takacs, and A. Molod, 1991: Simulation of the planetary boundary layer and turbulence in the GLA GCM, *Proceedings of the AMS Ninth Conference on Numerical Weather Prediction.* 21-25 October 1991, Denver, CO, 514-517.

Kalnay, E., M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. Stackpole, J. Tuccillo, L. Umscheid and D. Williamson, 1989: Roles for the interchange of physical parameterizations, *Bull. Amer. Meteor. Sci.*, **70**, 620-622.

Navon, I. M., X. Zou, J. Derber, and J. Sela, 1992: Variational data assimilation with an adiabatic version of the NMC spectral model, *Mon. Wea. Rev.*, **120**, 1433-1446.

Phillips, N. A., 1957: A coordinate system having some special advantages for numerical forecasting, *J. Meteor.*, **14**, 184-185.

Sadourney, R., 1975: The dynamics of finite difference models of the shallow water equations, *J. Atmos. Sci.*, **32**, 680-689.

Schubert, S. D., J. Pfaendtner and R. Rood, 1993: An assimilated data set for Earth Science applications, *Bull. Amer. Meteor. Sci.*, **74**, 2331-2342.

Suarez, M. J., and L. L. Takacs, 1995: Documentation of the Aries/GEOS Dynamical core: version 2, *NASA Technical Memorandum 104606*, Vol.**5**, Goddard Space Flight Center, NASA.

Takacs, L. L., A. Molod and T. Wang, 1994: Documentation of the Goddard Earth Observing System (GEOS) General Circulation Model-Version 1. *NASA Technical Memorandum 104606*, Volume **1**, Goddard Space Flight Center, NASA.

Todling, R., S. E. Cohn and N.S. Sivakumaran, 1998: Suboptimal schemes for retrospective data assimilation based on the fixed-lag Kalman smoother, *Mon. Wea. Rev.*, **126**, 2274-2286.

Yang, W. and I. M. Navon, 1996: Documentation of the tangent linear model and its adjoint of the adiabatic version of the NASA GEOS-1 C-grid GCM - version 5.2, *NASA Technical Memorandum 104606*, Vol. **8**, Goddard Space Flight Center, NASA.

Yang, W., I. M. Navon and R. Todling, 1997: Documentation of the tangent linear and adjoint models of the relaxed Arakawa-Schubert moisture parameterization package of the NASA GEOS-1 GCM (version 5.2), *NASA Technical Memorandum 104606*, Vol. **11**, Goddard Space Flight Center, NASA.