February 10, 2004

BOB: Model Description and User's Guide

R. K. Scott [1]
L. Rivier [2]
R. Loft [2]
L. M. Polvani [1]

NCAR Technical Note, June, 2003

[1] Department of Applied Physics and Applied Mathematics, Columbia University, New York, New York
[2] Computational Science Division, National Center for Atmospheric Research, Boulder, Colorado

Contents

List of Figures

1 Introduction

This document provides a users guide and detailed description of the Fortran implementation of BOB (Built On Beowolf). BOB solves the shallow water equations and dry, hydrostatic primitive equations in pressure coordinates on a spherical domain. It has been designed with the intention of providing an efficient spectral implementation of the shallow water and primitive equations that can be run on the inexpensive commodity clusters that are being used increasingly as an alternative to high performance computers.

A common problem encountered on such clusters is the limited cache and memory sizes. To overcome this limitation, BOB makes use of several optimization techniques. First, a recursion relation is used to calculate the associated Legendre polynomials (ALPs) at each timestep "on the fly", thus eliminating the need for the order N^3 storage encountered when loading the ALPs from memory. Second, the need to store the scalar derivatives of the ALPs is avoided by taking advantage of a simple relation between these and the ALPs themselves. The implementation of the Legendre transform is cache blocked in latitude to ensure that the working arrays fit in the cache of each processor, and is unrolled to keep memory traffic low. The cache blocking enables the high performance of the model to be maintained even at very high resolutions. Finally, a one-dimensional domain decomposition and transposition procedure is used in latitude and wavenumber space that ensures efficient load balancing among processors.

Two versions of the code are described in this report: the shallow water equations and the dry, adiabatic primitive equations, both on a spherical domain. The code has been designed to run in parallel using a message passing interface (MPI). For each system we provide the details of the numerical implementation of the model, as well as a description of the parallel implementation and the loop structure and file structure of the Fortran code itself. Finally, we have included a section describing how to obtain, compile and run the code, together with some test case results.

2 Numerical details

Since the horizontal equations and discretization of the primitive equations closely resemble those of the shallow water equations, in this section and throughout, the main discussion will be of the shallow water equations. The differences arising in the primitive equations are described as appropriate.

2a. Equations

(i) Shallow water equations

Following Hack and Jakob (1992), the shallow water equations on a spherical domain can be formulated in terms of absolute vorticity η, divergence δ, and geopotential $\Phi \equiv \bar{\Phi} + \Phi'$ (where

$\bar{\Phi}$ is the constant global average of Φ) as follows:

$$\begin{aligned}
\frac{\partial \eta}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial A}{\partial \lambda} - \frac{1}{a}\frac{\partial B}{\partial \mu} \\
\frac{\partial \delta}{\partial t} &= \frac{1}{a(1-\mu^2)}\frac{\partial B}{\partial \lambda} - \frac{1}{a}\frac{\partial A}{\partial \mu} - \nabla^2\Phi' - \nabla^2 E \\
\frac{\partial \Phi'}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial C}{\partial \lambda} - \frac{1}{a}\frac{\partial D}{\partial \mu} - \bar{\Phi}\delta.
\end{aligned} \tag{2.1}$$

Here, $\mu = \sin\phi$, ϕ is the latitude, λ is the longitude, and a is the planetary radius. The nonlinear terms on the right-hand sides are given by

$$\begin{aligned}
&A = U\eta, \qquad B = V\eta, \\
&C = U\Phi, \qquad D = V\Phi', \qquad E = \frac{U^2+V^2}{2(1-\mu^2)},
\end{aligned}$$

where $U \equiv u\cos\phi$, $V \equiv v\cos\phi$, and u and v are the zonal and meridional velocity components, respectively. The velocity $\mathbf{V} \equiv (u, v)$ is related to η and δ through

$$\mathbf{V} = \mathbf{k} \times \nabla\psi + \nabla\chi \tag{2.2}$$

for a streamfunction ψ and velocity potential χ satisfying $\eta = \nabla^2\psi + f$ and $\delta = \nabla^2\chi$, where $f = 2\Omega\sin\phi$ is the Coriolis parameter and Ω is the planetary rotation rate.

(ii) Primitive equations

The primitive equations with pressure p as the vertical coordinate can be written in a similar form in terms of prognostic variables η, δ, and the potential temperature θ:

$$\begin{aligned}
\frac{\partial \eta}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial A}{\partial \lambda} - \frac{1}{a}\frac{\partial B}{\partial \mu} \\
\frac{\partial \delta}{\partial t} &= \frac{1}{a(1-\mu^2)}\frac{\partial B}{\partial \lambda} - \frac{1}{a}\frac{\partial A}{\partial \mu} - \nabla^2\Phi - \nabla^2 E \\
\frac{\partial \theta}{\partial t} &= -\frac{1}{a(1-\mu^2)}\frac{\partial C}{\partial \lambda} - \frac{1}{a}\frac{\partial D}{\partial \mu} - \bar{\Phi}\delta - F,
\end{aligned} \tag{2.3}$$

where now the nonlinear terms are defined as

$$\begin{aligned}
&A = U\eta + V\delta + \frac{\partial V\omega}{\partial p}, \qquad B = V\eta + U\delta + \frac{\partial U\omega}{\partial p}, \\
&C = U\theta, \qquad D = V\theta, \qquad E = \frac{U^2+V^2}{2(1-\mu^2)}, \qquad F = \frac{\partial \omega\theta'}{\partial p},
\end{aligned}$$

and $\omega = dp/dt$ is the vertical pressure velocity. The system (2.3) is closed using the hydrostatic relation between Φ and θ,

$$-\frac{1}{C_p}\frac{\partial \Phi}{\partial \xi} = \theta, \tag{2.4}$$

and the continuity equation relating ω and δ,

$$\frac{\partial \omega}{\partial p} = -\delta, \tag{2.5}$$

where C_p is the specific heat at constant pressure, $\xi = (p/p_s)^\kappa$, is an auxiliary vertical coordinate, p_s is a reference surface presure, $\kappa = R/C_p$ and R is the gas constant.

2b. Horizontal discretization

The shallow water version of BOB is a parallel version of the shallow water model described in Hack and Jakob (1992) incorporating the optimization procedures mentioned in the introduction. These procedures are described in more detail in this section and in section 3 below, and some details can also be found in Rivier et al. (2002). We first outline the main ideas of the horizontal discretization, following Hack and Jakob (1992), to which the reader is referred for further details. These ideas are common to both the shallow water version and the primitive equation version of the model.

A physical field ξ is transformed from physical space into Fourier or latitude space, using the discrete Fourier transform:

$$\xi^m(\mu) = \frac{1}{I}\sum_{i=1}^{I} \xi(\lambda_i, \mu) e^{im\lambda_i}, \tag{2.6}$$

where there are I zonal grid points located at longitudes λ_i, and where m is the Fourier wavenumber. The field is then transformed from Fourier space to spectral space using the Legendre transform and associated Legendre polynomials, P_n^m:

$$\xi_n^m = \sum_{j=1}^{J} \xi^m(\mu_j) P_n^m(\mu_j) w_j, \tag{2.7}$$

where there are J Gaussian latitudes μ_j, with weights w_j, and where P_n is the Legendre polynomial of degree n.

Assuming $\xi^m(\mu)$ vanishes at the pole, the meridional derivative of ξ can be represented in spectral space as

$$\left(\frac{\partial \xi}{\partial \mu}\right)_n^m = -\sum_{j=1}^{J} \xi^m(\mu_j) \frac{H_n^m(\mu_j)}{a(1-\mu_j^2)} w_j, \tag{2.8}$$

where $H_n^m(\mu)$ is defined as

$$H_n^m(\mu) = (1-\mu^2)\frac{\partial P_n^m}{\partial \mu}(\mu). \tag{2.9}$$

Various properties of the associated Legendre polynomials $P_n^m(\mu)$ can be used to limit the memory load of the model, which, for the $P_n^m(\mu)$ and $H_n^m(\mu)$ is order N^3 and becomes prohibitively high at very high resolutions. First, the hemispherical symmetry of the P_n^m means that only half the J range is needed in the sum in (2.7), reducing the storage requirement by a factor of two. Second, the polynomials, $H_n^m(\mu)$ can be computed at each timestep from a relationship involving $P_{n+1}^m(\mu)$ and $P_{n+1}^m(\mu)$, and thus do not need to be stored at all (Rivier et al., 2002). Third, the polynomials $P_n^m(\mu)$ themselves can be computed at each timestep from a recurrence relationship expressing P_{n+1}^{m+1} in terms of P_{n-1}^{m-1}, P_{n+1}^{m-1}, and P_{n-1}^{m+1}. The storage requirements of the precomputed $P_n^m(\mu)$ and other coefficients needed to initiate this recurrence relation is only order N^2. The recurrence relation itself is numerically stable (Swarztrauber, 1993), and additionally, each computed value of P_n^m can be reused for each of the terms appearing on the right-hand sides of (2.1) and (2.3), resulting in an increase computation time of only 16% for the calculation of the spectral coefficients.

2c. Vertical discretization

The shallow water equations have no vertical dependence, so this section deals exclusively with the primitive equation version of the model. For information on how to run an uncoupled, multilevel stack of shallow water equations, see section 4 below.

The vertical discretization used for the primitive equation version is a standard, second order finite differencing, using pressure as the vertical coordinate. For full details, the reader is referred to Saravanan (1992). There are $K+1$ "half-levels" dividing the vertical domain $[0, p_s]$ into K arbitrary sub-intervals of width Δp_k for $k = 0, \ldots, K$. The model prognostic variables are defined at the "full-levels", which are the midpoints of these sub-intervals. The vertical pressure velocity ω, on the other hand, is defined at the half-levels. Figure 1 illustrates the relationship between the full-levels, half-levels, and the level thicknesses Δp_k.

The vertical derivatives appearing on the right-hand sides of (2.3) are in flux form. For a flux $F_{k+\frac{1}{2}} = \omega_{k+\frac{1}{2}} \bar{q}_{k+\frac{1}{2}}$, where q represents one of U, V, or θ, and $\bar{q}_{k+\frac{1}{2}} = (q_{k+1} + q_k)/2$, the vertical derivative of F is defined at the full-level k as

$$\left(\frac{\partial F}{\partial p}\right)_k = \frac{F_{k+\frac{1}{2}} - F_{k-\frac{1}{2}}}{\Delta p_k} \tag{2.10}$$

At the upper and lower pressure half-levels, the boundaries of the vertical domain, the pressure-velocity is set to zero, $\omega_{\frac{1}{2}} = \omega_{K+\frac{1}{2}} = 0$. This implies that the flux of any quantity throught these surfaces, i.e. into or out of the model domain, is also zero.

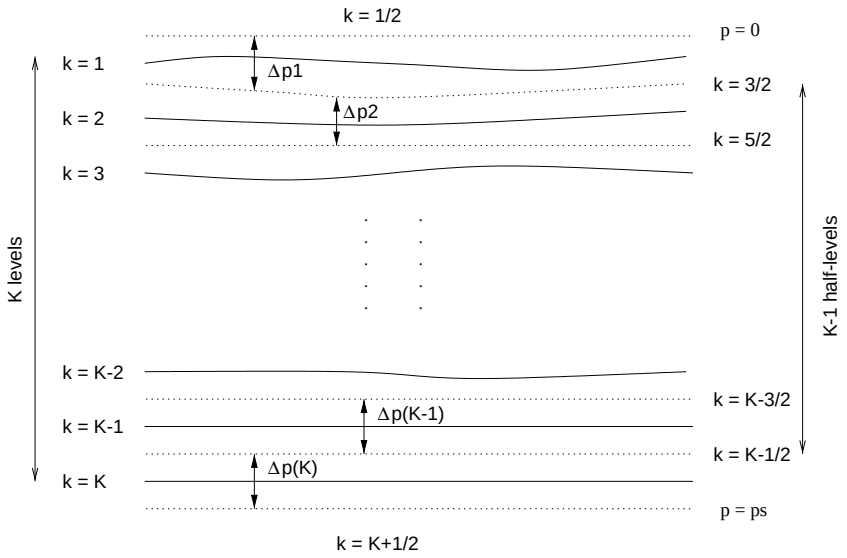

Figure 1: Sketch of the vertical discretization of the pressure level version of the model. Levels are in solid lines, half-levels in dotted lines.

The choice of boundary condition means that the vertically integrated divergence is zero, from (2.5), and so the external gravity wave mode is excluded. This implies that the K divergences on model full-levels, are not linearly independent. It is therefore convenient to work with the $K-1$ linearly independent quantities $\hat{\delta}_{k+\frac{1}{2}} = (\delta_{k+1} - \delta_k)/2$ defined on interior model half-levels.

The hydrostatic relation (2.4) is discretized as follows:

$$\frac{\Phi_{k+1} - \Phi_k}{\zeta_{k+1} - \zeta_k} = -C_p \bar{\theta}_{k+\frac{1}{2}} \tag{2.11}$$

or, equivalently

$$\hat{\Phi}_{k+\frac{1}{2}} = -C_p \hat{\zeta}_{k+\frac{1}{2}} \bar{\theta}_{k+\frac{1}{2}} \tag{2.12}$$

where $\zeta = (p/p_s)^\kappa$ is an auxiliary vertical coordinate and where $\hat{\Phi}_{k+\frac{1}{2}}$ and $\hat{\zeta}_{k+\frac{1}{2}}$ are defined analagously to $\hat{\delta}_{k+\frac{1}{2}}$. This allows a matrix representation of Φ:

$$\vec{\hat{\Phi}} = M_{\theta \to \hat{\Phi}} \vec{\theta} \tag{2.13}$$

where $\vec{\hat{q}} = (\hat{q}_{1+\frac{1}{2}}, \ldots, \hat{q}_{K-\frac{1}{2}})$ denotes a column vector of length $K-1$ and where M is a $(K-1) \times K$ matrix defined by

$$[M_{\theta \to \hat{\Phi}}]_{k,k'} = \begin{cases} -\frac{1}{2} C_p \hat{\zeta}_{k+\frac{1}{2}} & \text{if } k' = k+1 \text{ or } k' = k, \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

Similarly, the discretization of the continuity equation can be written in matrix form,

$$\vec{\omega} = M_{\delta\to\omega} M_{\hat{\delta}\to\delta} \vec{\hat{\delta}} \tag{2.15}$$

where, $M_{\delta\to\omega}$ and $M_{\hat{\delta}\to\delta}$ are $K \times K$ and $(K-1) \times K$ matrices, respectively, given by

$$[M_{\delta\to\omega}]_{k,k'} = \begin{cases} -\Delta p_{k'} & \text{if } k' \le k, \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

and

$$[M_{\hat{\delta}\to\delta}]_{k,k'} = \begin{cases} 2p_{k'+\frac{1}{2}}/p_{K+\frac{1}{2}} & \text{if } k' < k, \\ 2(p_{k'+\frac{1}{2}}/p_{K+\frac{1}{2}} - 1) & \text{if } k' \ge k. \end{cases} \tag{2.17}$$

These relatively small "vertical" matrices are computed once at the beginning of each calculation and then stored in a common block. The same is done with matrices arising from the decomposition of the potential temperature field into reference and perturbation components, needed for the semi-implicit time-stepping, and from hyperdiffusion terms added to the right-hand sides of (2.3) (see sections 2d and 2e below). For further details of these matrices see Appendix A, and Saravanan (1992).

2d. Hyperdiffusion

For dynamically interesting initial conditions, equations (2.1) and (2.3) will typically exhibit extremely rapid formation of large gradients in the vorticity and potential temperature, leading to small scale features that are beyond the capabilities of the model resolution. Horizontal diffusion, or higher order hyperdiffusion terms are therefore added to the right-hand sides of the equations to stop the cascade of enstrophy to small scales and to render the solution computationally well-posed. These terms appear in the equations in the form

$$\frac{\partial \xi}{\partial t} = \ldots + \gamma \nabla^{2n} \xi \tag{2.18}$$

where ξ represents each of the prognostic variables, ζ, δ, θ, or Φ, and the dots represent the other terms on the right-hand sides of (2.1) and (2.3). The coefficient γ is chosen according to the dynamical problem and the horizontal grid resolution. The order n of the diffusion operator governs the extent to which the diffusion is scale-selective, acting preferentially on the small scales in the model. The larger n the greater the scale-selectivity, and the less impact the diffusion has on the large scale features. Typical values used for meteorological applications are $n = 1, 2, 3, 4$; $n = 1$ corresponds to ordinary diffusion.

2e. Time stepping

The time stepping is a leapfrog scheme in which the terms associated with gravity waves and with the horizontal diffusion are treated implicitly. Full details of the semi-implicit scheme can be found in Hack and Jacob (1993) for the case of the shallow water equations and in Saravanan (1992) for the case of the primitive equations. For the purposes of the semi-implicit scheme, the potential temperature field in the primitive equations is decomposed into a reference profile, θ^R, and deviations from this, $\theta' = \theta - \theta^R$. The linearization of the right-hand side of the θ equation in (2.3) is then treated implicitly, while the remaining nonlinear contribution is treated explicitly. The reference potential temperature used in the primitive equation version of BOB corresponds to that of an isothermal atmosphere.

Since the leap frog scheme produces a computational mode, a Robert-Asselin filter is used to damp it out. Both the semi-implicit leapfrog scheme and Robert-Asselin filter are standard and widely used in the this type of model.

3 Implementation

In this section we describe the implementation of the numerical equations described in the previous section. This includes a description of the decomposition of the physical, Fourier, and spectral domains used for the parallelization, details of the Fortran loop structures used to cycle over the decomposed fields in each domain, details of the transposition between the Fourier space and spectral space decompostion, and, finally, a description of the filestructure and dependencies of the Fortran code itself.

3a. Domain decomposition for parallelization

The domain is divided into subdomains in the horizontal direction only, that is, there is no vertical decomposition. What follows in this subsection therefore applies equally to both the shallow water version and the primitive equation version of BOB.

In the horizontal direction we use a one-dimensional decomposition. This is in contrast to earlier work on two-dimensional decompostions by Foster and Worley (1994). Although the 2D decomposition allows for a finer grain decomposition this also results in considerably more network traffic, requiring expensive low-latency–high-bandwidth networks. The 1D decomposition used here avoids this high network traffic and also makes use of load balancing among processors. Further, the decomposition is designed such that the memory footprint of each processor is small and can fit in the relatively small L2 cache of less expensive Pentium-based clusters.

(i) Physical space and Fourier space decomposition

In both physical space and Fourier space the domain is decomposed such that the latitudes are distributed uniformly over the processing elements (PEs); there is no decomposition in the zonal direction. This means that the fast Fourier transforms (FFTs) are carried out locally on each processor. Since these the FFTs are one-dimensional, they fit easily into cache, even at fairly high resolutions (beyond T341).

The physical domain is also tiled to ensure that the working set needed to compute the nonlinear terms on the right-hand sides of (2.1) or (2.3) fits easily into the memory cache of each PE. An example of this tiling, and its relation to the latitudinal decomposition, is shown in Figure 2, for the case of a spectral resolution of T42 (that is, with 128 longitudes and 64 latitudes) using 4 PEs. The latitudes are distibuted uniformly over the 4 PEs, and, for a given latitude, the longitudes are all on the same PE.

The shaded rectangle in the figure represents a single tile, of size (*nlon* $\times$ *nlat*). The tiling, or blocking size parameters *nlon* and *nlat* are specified in the header file **dims.h** (see section 3d below), and are chosen such that the tile fits easily into the memory cache of each PE. For a 512kB L2 cache the optimal blocking size is *nlat* $= 4$ and *nlon* $= 8$, but these can be adjusted for easy perfomance tuning and portability.

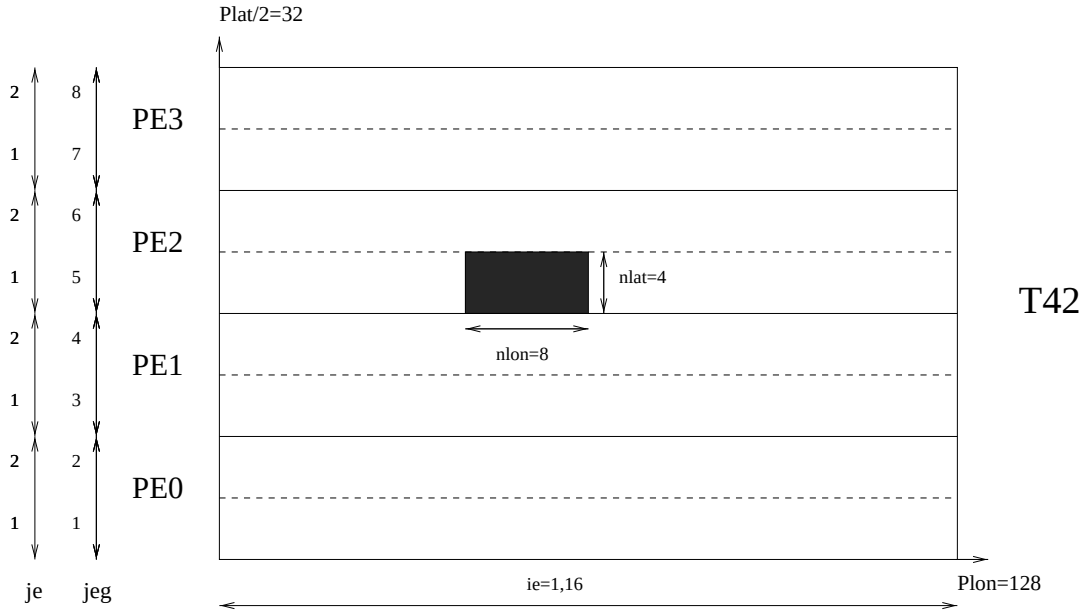

Figure 2: Decomposition of the physical space.

The indices *je* and *jeg* are shown along the latitude axis in Figure 2. *jeg* is a global index that represents the position in latitude of a specific tile. *je* is an index local to each PE that represents the position in latitudes of a tile on an individual PE. For the spectral resolution of T42 on 4 PEs used in this example, each PE contains 2 latitudinal tiles. The figure shows latitudes up to *plat/2*, this being the size of the latitudinal array dimension. In physical space a separate

loop is used to cycle the two hemispheres; in spectral space, only half the latitudinal domain of the associated Legendre polynomials is needed because of the hemispherical symmetry. Again, it is the tiling in latitude space that allows the working set of the Fourier coefficients and the associated Legendre polynomials, needed for the Legendre transform (2.7), to fit into the cache on each PE (see also section 3c below).

The index *ie* is shown along the longitude axis. It is a global index that represents the position in longitude of a specific tile. For details of how the tiling shown in Figure 2 is implemented in the Fortran code to loop over the whole of physical space, see the physical space loop description in section 3b below.

(ii) Spectral space decomposition

In spectral space the domain is decomposed such that the zonal wavenumbers are distributed over the PEs; there is no decomposition of the latitudinal wavenumber. Unlike the latitudinal decomposition above, a uniform decomposition cannot be used efficiently, since the triangular spectral truncation with a uniform decomposition gives rises to an unbalanced load distributed over the PEs. To ensure a balanced load (i.e. to provide roughly the same amount of work to each PE) it is necessary to pair zonal wavenumbers from opposite ends of the spectrum. That is, if M is the highest wavenumber retained in the truncation, the decomposition is applied on the pairing $(M, 0), (M-1, 1), \ldots, (M/2, M/2+1)$ if M is odd, and on the paring $(M, 0), (M-1, 1), \ldots, (M/2)$ if M is even.

The distribution of the spectral coefficients over the PEs is illustrated in Figure 3 for a resolution of T42 on 4 PEs (only the distribution for the first two PEs is shown). To distribute roughly the same number of spectral coefficients to each PE, the *m* coefficients are grouped into chunks of "high" and "low" *m* coefficients. To save memory space, the spectral coefficients are then packed in a single 1D structure that contains all the (*m*,*n*) coefficients for every *m*. The vector *scptr*, of length $M+1$ points to the corresponding positions of these coefficients within this 1D structure; that is *scptr*(*m*) gives the starting position where all the (*m*,*n*) coefficients associated with that *m* reside. The values of *scptr*(*m*) are shown on the figure for selected values of *m*.

Tables 1a and 1b below show the distribution of the low ($mch = 0$) and high ($mch = 1$) zonal wavenumbers, respectively, together with the values of *scptr*(*m*) for all 4 PEs of this example, and the total number of spectral wavenumbers distributed to each processor in each of the low and high *m* chunks. The total number of spectral wavenumbers residing on each PE is then the sum of the two totals, giving 263, 263, 239, and 197 wavenumbers on PE0, PE1, PE2, and PE3, respectively.

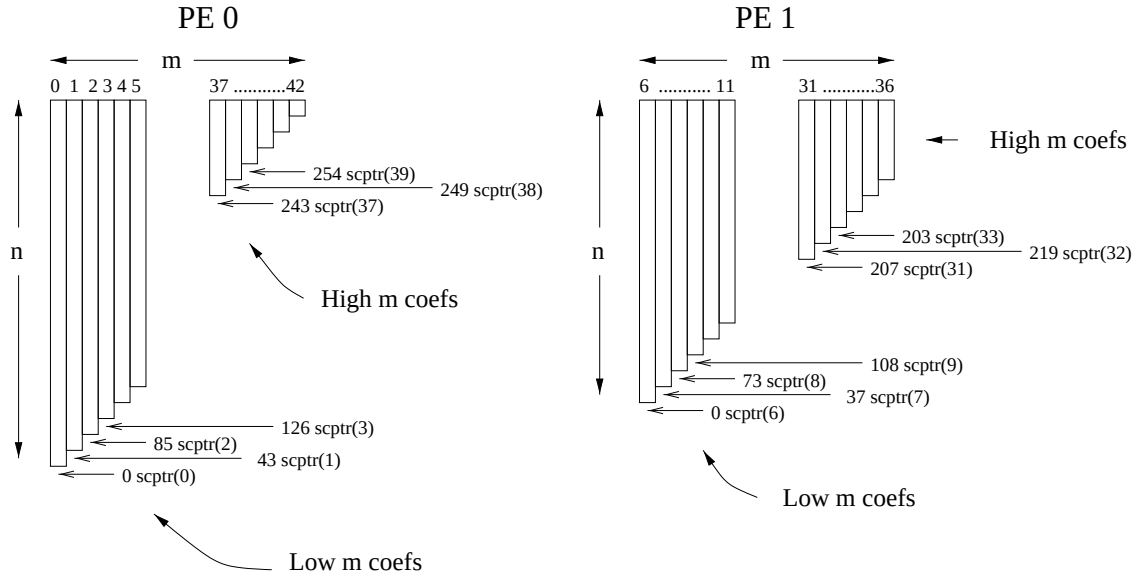

Figure 3: Decomposition of the spectral space and grouping of the zonal space into chunks of high and low zonal wavenumbers.

a	$mch = 0$, Low m			
PE	0	1	2	3
$m =$	0-5	6-11	12-16	17-21
$scptr(m)$	0	0	0	0
	43	37	31	26
	85	73	61	51
	126	108	90	75
	166	142	118	98
	205	175		
total	242	206	144	119

b	$mch = 1$, High m			
PE	0	1	2	3
$m =$	37-42	31-36	26-30	22-25
$scptr(m)$	243	207	145	120
	249	219	162	141
	254	230	178	161
	258	240	193	180
	261	249	207	
	263	257		
total	21	57	75	78

Table 1: Values of $scptr(m)$ and the number of wavenumbers stored on each PE in the example of T42 resolution on 4 PEs. (a) low m, (b) high m.

3b. Basic loop description

As an illustration of the domain decompositions described above, it is useful to consider the structure of the basic Fortran DO loops that cover the whole physical, Fourier, or spectral domains. Examples are again given for a resolution of T42 on 4 PEs.

(i) Physical space loop structure

First consider the physical domain. We want to cycle over all latitudes, longitudes, and all vertical levels (recall that the shallow water version can be run with *plev* uncoupled levels and that standard shallow water has *plev* = 1). Because of the tiling in latitude and longitude, the loops first cycle over the tiles and then over the latitude and longitude within each tile. This is illustrated in the following section of code that cycles over all the grid points in the physical domain:

```
do jeg=jebeg(rank),jeend(rank) ! rank specifies a PE (between 0 and NPE-1)
   je=jeg-jebeg(rank)+1
   dx=(2.0d0*Pi)/plonmax(jeg)  ! tile longitude spacing
   iptr=1
   do ie=1,ielem(jeg)
      do k=1,plev
        do ins=0,1
          do j=1,nlat2
            do i=1,nlon
              array(i,j,ins,k,ie,je)=0.0d0
              lon(i,ie,je)=(iptr-1)*dx
              iptr=iptr+1
            end do
          end do
        end do
      end do
   end do
end do
```

Here, the local variable *array* is set to zero on each PE. The union of all copies of *array* on the different PEs covers the physical domain. Similarly, the local variable *lon* is set to the longitude on each PE.

As shown in Figure 2, for the case of T42 resolution on 4 PEs, the global integer valued variables *jebeg* and *jeend* take the following values:

jebeg(0)=1 *jeend*(0)=2
jebeg(1)=3 *jeend*(1)=4
jebeg(2)=5 *jeend*(2)=6
jebeg(3)=7 *jeend*(3)=8;

The loop first runs over *jeg*, the latitudinal index of each tile, then over *ie*, the longitudinal index of each tile. Thus these two loops cycle over all the tiles in the domain. Thus, *jeg* is a global integer valued variable that provides the latitudinal index of each tile, and is also used to access global arrays such as *plonmax*(*jeg*). In contrast, *je* is a local integer valued variable that runs over the number of tiles in latitude on an individual PE. Similarly, *ie* is a local integer

valued variable that provides the longitudinal index of each tile on an individual PE. Note that the array variable *ielem*(*jeg*) has the same dimension as the number of latitudinal tiles. This is for the provision, in future model revisions, of a computational grid that has varying numbers of longitude points for varying latitudes (typically fewer near the poles). In the present version of the code, only a uniform rectaungular grid is supported and so the value of *ielem*(*jeg*) is currently the same for all *jeg* (defined in **dims.h**).

The remaining loops run over vertical level (*k*), the two hemispheres (*ins*), the latitude within each tile (*j*) and the longitude within each tile (*i*).

(ii) Fourier space loop structure

To illustrate the loop structure in Fourier space we describe in more detail how the associated Legendre polynomials are calculated "on the fly". Below is the section of the code from the subroutine **spectral.F** (see section 3d below), which is performed at each timestep before carrying out the Legendre transform. For each Fourier coefficient *m* we need to calculate the $N - m$ polynomials $P_n^m(\mu)$ as a functions of latitude. At this point the latitude-Fourier mode transposition has already been made so that Fourier modes, rather than latitudes, are distributed over the PEs. As described for the spectral space decomposition in section 2a above, the Fourier modes are grouped in low mode and high mode chunks to ensure load balancing across the PEs of the subsequent spectral arrays.

```
    do mch=0,mchmax(rank) ! loop over the chunks of m
       do ioe=0,1         ! odd even index
          ix=0            ! P memory pointer for compute on fly
          do jeg=1,jelemg
             call loadalp(Pseed(1,0,jeg,ioe,mch),
   :                      P(1,0,ix,jeg),
   :                      mbeg(mch,rank)+ioe)
          end do

          do m=mbeg(mch,rank)+ioe,mend(mch,rank),2

             do ipe=0,P_NODE-1
                do jeg=jebeg(ipe),jeend(ipe)
                   je=jeg-jebeg(ipe)+1

                   if (m.ge.mbeg(mch,rank)+2) then
                       call genalp(P(1,0,0,jeg),
   :                               genp(1,1+vscptr(m)),
   :                               ix,
   :                               m)

c        (1) call to analysis
```

```
                  end if
               end do
            end do

c        (2) call to advance
c        (3) call to synthesis

            ix=1-ix

          end do    ! End loop on m
       end do       ! End loop on ioe
    end do          ! End loop on mch
```

Here, the *mch* loop runs over the two chunks (high and low) of Fourier modes contained on each PE, that is, *mchmax* $= 2$ on all PEs. The index *ioe* allows the subsequent loop on Fourier modes to be split into odd and even componenets, necessary for efficient implentation of the ALP recurrence relation, which involves $P_{n\pm1}^{m-1}$ and $P_{n\pm1}^{m+1}$ coefficients on each pass (see Rivier et al., 2002, Equation (15)).

Before the recurrence relation can be used, a subset of the ALPs, the "seed values", need to be loaded into the ALP array *P(*,*,*,*)*. These seed values are computed once only during the initialization phase of the model (see section 3d below). The subroutine **loadalp**, called separately for each latitudinal tile indexed by *jeg*, loads the seed values of the ALPs (*Pseed*) into the array *P(*,*,*,*)*.

Once the seed values are loaded into *P*, the recurrence relation can be used to calculate the remaining ALPs. The *m* loop runs over the values of *m* located on each PE, and is performed separately for the high and low mode chunks, and for odd and even modes. Within the *m* loop, the loops over *ipe* and *jeg* together cover all of tiles in latitude space: *P_NODE* is equal to the number of PEs, and *jeg* cycles over the tiles associated with each PE. Recall that, at this stage, the decomposition over PEs is in Fourier space, so that a horizontal array local to a given PE contains all the latitude points and a subset of the Fourier modes. For a given Fourier mode *m* and latitude tile *jeg*, **genalp** then computes the all the $N - m$ Legendre polynomials associated with that *m*, as a function of latitude within that tile, and saves the resulting (*n*,latitude) dependent array in *P*. This array is then used to perform the Legendre transform, taking the terms of the equations from Fourier space to spectral space.

Note the location of the calls to **analysis**, **advance** and **synthesis** within the same *m* loop. The subroutine **analysis** performs the Legendre transform, the subroutine **advance** performs the timestepping of the spectral coefficients, and the subroutine **synthesis** performs the reverse Legendre transform back to Fourier space. Thus the entire spectral part of the timestepping is unrolled with respect to Fourier wavenumber. Further, because the domain is tiled in latitude, and separate calls to **analysis** and **synthesis** for the forward and reverse Legendre transform are made for each tile, each of these routines uses only a subset of the ALP array *P* and this subset

is likely to remain cache resident throughout each call.

(iii) Spectral space loop structure

Finally we consider the loop over spectral space, using as an example the call to **advance** (labelled (2) in the Fourier space loop illustrated above) where the timestepping is performed. The first part of the loop over spectral space has the same structure as that over Fourier space: three loops (over *mch*, *ioe*, *m*) fix the value of *m*. As noted above, this allows the entire spectral part of the model, comprising the forward Legendre transform, the timestepping, and the reverse Legendre transform, to be placed within the same single loop over Fourier modes as that used for the calculation of the ALPs.

Within the Fourier space loops, the pointers *nlt_p* and *sc_p* are used to point to the correct position within the 1D arrays that contain, respectively, the spectral coefficients of the nonlinear terms (*nltsc*) and the prognostic variables at the current and previous timestep (*sc* and *scm1*). The subset of these arrays defined by the pointers are then passed to the subroutine **advance**:

```
      do mch=0,mchmax(rank)
         do ioe=0,1
            do m=mbeg(mch,rank)+ioe,mend(mch,rank),2

c     (0) call to genalp
c     (1) call to analysis

               nlt_p=1+2*plev*nnlt*vscptr(m)
               sc_p =1+2*plev*nsp*scptr(m)
               for_p =1+2*plev*scptr(m)

               call advance(nltsc(nlt_p),
     :                     sc(sc_p),
     :                     scm1(sc_p),
     :                     eps(1+vscptr(m)),
     :                     m)

c     (3) call to synthesis

            end do ! End loop on m
         end do ! End loop on ioe
      end do ! End loop on mch
```

In the subroutine **advance**, these 1D arrays are declared as multi-dimensional arrays in which the separate dimensions represent the vertical levels ($k = 1, \ldots, K$), the latitudinal wavenumber ($n = m, \ldots, N$), the type of nonlinear term or prognostic variable (*nnlt* or *nsp*), and the real or complex part of the coefficient ($ic = 0, 1$). See the section of code listed below. Note that, although each spectral coefficient is a complex number, it is treated as two real

numbers for more efficient arithmetic. Thus, $ic = 0$ corresponds to the real part of the spectral coefficient and $ic = 1$ corresponds to the imaginary part. The pointer *nnlt* refers to the different nonlinear terms $A, \ldots, F$, appearing on the right-hand sides of (2.1) and (2.3), while the pointer *nsp* refers to the prognostic variables, $\zeta, \delta,$ and θ or Φ.

```
      subroutine advance(nltsc,
     :                   sc,
     :                   scm1,
     :                   eps,
     :                   m)

      INT  m
      REAL nltsc(0:1,nnlt,m:nn+1,plev)
      REAL sc(0:1,nsp,m:nn,plev)
      REAL scm1(0:1,nsp,m:nn,plev)
      REAL eps(m:nn+1)

      do k=1,plev
         do n=m,nn
            do ic=0,1
               vorterm =  w*nltsc(ic,a_p,n,k) + ...
            enddo
         enddo
      enddo
```

It is also possible to pass forcing functions to the timestepping subroutine in the same manner as the nonlinear terms and prognostic variables. For example, this is done in the primitive equation version of the code to include Newtonian cooling and Rayleigh friction terms in the model equations as simple parametrizations of atmospheric forcings. This allows the simulation of realistic atmospheric flows such as that used for the Held and Suarez (1994) test case, described in section 4b below. For example, to allow for Rayleigh friction, an additional pointer *for_p* is used that points to the correct position in the arrays *forvorsc* and *fordivsc*, which are passed to **advance** as before:

```
               for_p =1+2*plev*scptr(m)

               call advance(nltsc(nlt_p),
     :                      sc(sc_p),
     :                      scm1(sc_p),
     :                      forvorsc(for_p),
     :                      fordivsc(for_p),
     :                      eps(1+vscptr(m)),
     :                      m)
```

and in **advance** these arrays are redefined to have the following structure:

```
      REAL forvorsc(0:1,m:nn,plev)
      REAL fordivsc(0:1,m:nn,plev)
```

3c. Transposition Algorithm

As described in section 3a, the parallelization uses a one-dimensional decomposition in which the latitudes of the physical/Fourier space variables and the longitudinal wavenumbers of the spectral space variables are distributed over the PEs. For the Fourier transform stage between physical and Fourier space, the latitudes are distributed over the PEs so that the transform requires only local operations, summing over longitudes as in (2.6). On the other hand the Legendre transform between Fourier and spectral space, (2.7), requires a summation over the Gaussian latitudes μ_j, and therefore is most efficient when the latitudes are all stored locally on each PE and the longitudinal wavenumbers *m* are distributed over the PEs. Between the Fourier and the Legendre transforms it is therefore necessary to transpose between a decomposition over latitude space and one over longitudinal wavenumber. The two decompositions and the direction of the transposition between the forward Fourier and Legendre transforms are illustrated in Figure 4.

To avoid a high volume of network traffic and the possibility of consequent conflicts, the transposition is carried out in stages. Figure 5 illustrates how the data is moved around during the transposition when there are four PEs. The four columns in each matrix represent the data present locally on the four PEs. At the beginning of the forward transposition the latitudes are distributed along each row (distributed over the PEs) and the Fourier modes are distributed along each column.

The zeroth stage shows the data on the diagonal staying in place for the transposition. The PEs are then paired together as indicated by the arrows and the data is swapped between paired PEs using a call to the MPI routine **mpi_sendrecv**. This pairing and swapping is performed $NPE - 1$ times where *NPE* equals the number of PEs; thus each PE exchanges data with all others. After $NPE - 1$ stages the transposition is complete and data that was local to one PE is distributed over the other PEs. At the end of the forward transposition the Fourier modes are distributed over each row (distributed over the PEs) and the latitudes are distributed over each column. A similar procedure is used for the reverse transform.

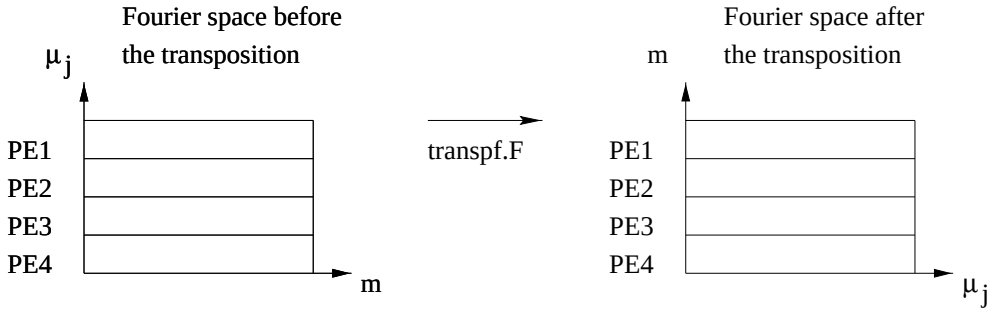

Figure 4: Distribution of the 2D Fourier space over the PEs. The arrow indicates the forward transposition.

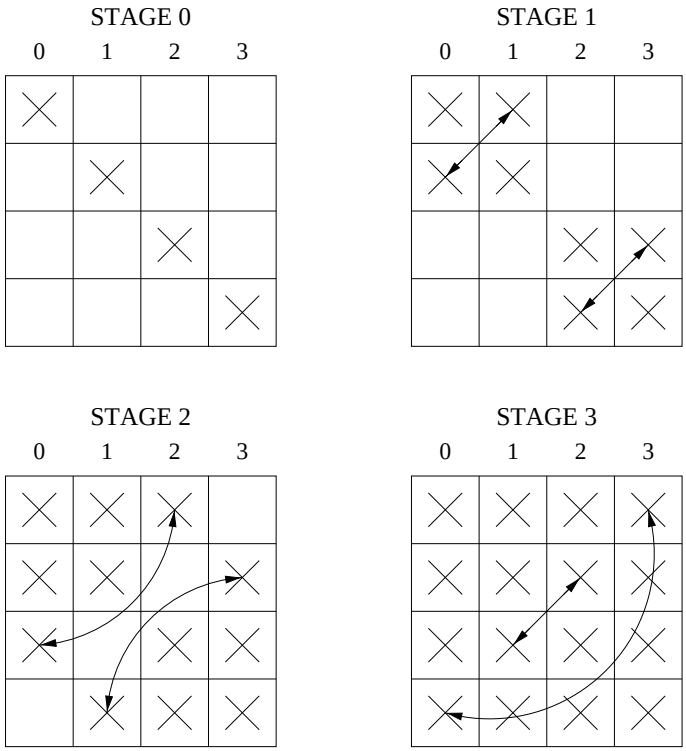

Figure 5: Transposition.

The method of pairwise transposition over *NPE* -1 stages limits the number of messages present in the network at any one time and thus limits conflicts which would otherwise slow down the transposition process.

3d. Fortran implementation and filestructure

To summarize the foregoing, the structure of the the main time stepping loop comprises the following main stages:

- calculation of the nonlinear terms in physical space at time t
- forward Fourier transform
- transposition
- forward Legendre transform
- calculation of the prognostic variables at time $t + \Delta t$
- inverse Legendre transform
- transposition
- inverse Fourier transform

In addition, the main BOB program has an initialization stage that sets up the arrays, reads in the Gaussian grid and the initial conditions, and constructs the vertical matrices (in the case of the primitive equations). The Gaussian grid and initial conditions themselves are constructed

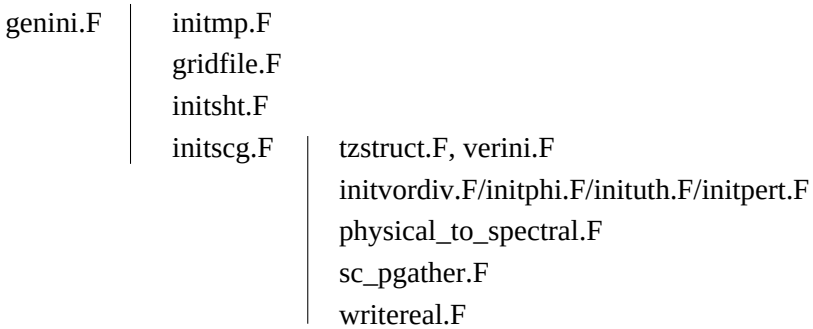

Figure 6: The example calling for the code generating the executable **genini.F**. The exact form of calls from **initscg.F** depends on the particular initial condition being generated. The calls to **tzstruct.F** and **verini.F** are only present in the primitive equation version.

by running two other programs, which are compiled and run separately. There are thus three separate programs that are involved in running the model: **gengrid.F**, which generates the grid, **genini.F**, which generates the initial conditions, and the main BOB program itself, which integrates the initial condition forward in time. These three programs, and their subroutine dependencies, are described in this section.

The first program, **gengrid.F**, calculates the Gaussian latitudes and weights and writes these to a file that is read subsequently by the other programs. The main calculation takes place in the subroutine **pgaqd**. The whole program is run serially on a single processor. It need only be run once for a given horizontal resolution.

The second program, **genini.F** generates the initial condition. It calculates the spectral coefficients of the three prognostic variables ζ, δ, Φ (or θ in the case of the primtive equations) and writes these to a single initial condition file. This program is run in parallel, making full use of the domain decomposition and other features of the main model (most of the subroutines involved in calculating the spectral coefficients of the prognostic variables are simply borrowed from the main BOB program).

Figure 6 shows the calling tree of **genini.F**. The first call, to the subroutine **initmp.F**, performs the MPI initialization required for communication between the different PEs. The subroutine **gridfile.F** then reads in the Gaussian latitudes and weights and broadcasts these to all the PEs as global variables. These global latitudinal variables are then decomposed over the PEs according to the tiling procedure described in section 3a by the subroutine **initsht.F**. This subroutine also initializes the weights required for the Fourier transform and the seeds required for the recurrence relation that computes the Legendre polynomials.

The subroutine **initscg.F** has a variety of dependencies, according to the version of the model. In the primitive equation version calls are made to the subroutines **tzstruct.F** and **verini.F**, which initialize the vertical structure. These are absent in the shallow water version. Depending on the type of initial condition required, a combination of **initvordiv.F**, **initphi.F**,

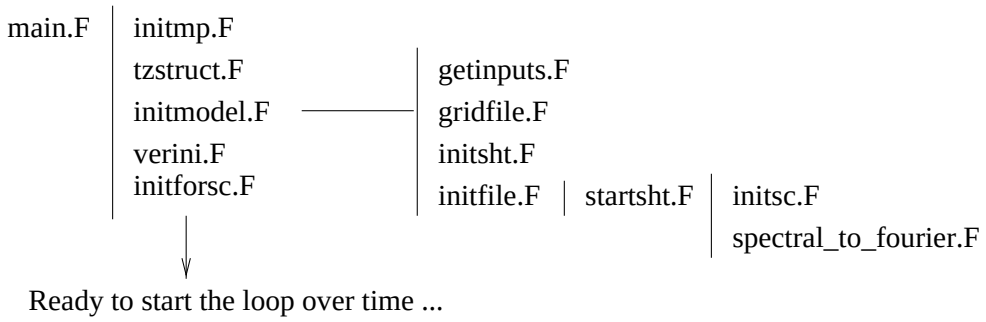

Figure 7: Calling tree for the main BOB during the initialization stage up to the loop over time. The calls to **tzstruct.F** and **verini.F** are only present in the primitive equation version.

inituth.F and **initpert.F** initializes the three prognostic variables in physical space. Finally the subroutine **physical_to_spectral.F** transforms these fields into spectral space and **sc_pgather.F** gathers the fields on one PE for output to file using **writereal.F**.

The third program is the main BOB model, which does the time-stepping and outputs various diagnostic fields. The calling tree of this program can be conveniently divided into two main sections, describing the initialization and timestepping. Figure 7 illustrates the model up to the point where the loop over time starts. Figure 8 describes the main time-loop where the equations are "advanced" in time.

As in the case of **genini.F**, the MPI initialization is done first with a call to **initmp.F**. This is followed in the primitive equation version with a call to **tzstruct.F** which initializes the vertical temperature profile.

The main part of the initialization then begins with **initmodel.F**. Within this subroutine, **getinputs.F** reads the input namelist in *bobx.in*, which contains the filename of the initial condition, the total number of timesteps to be computed, the timestep length, the frequency of the diagnostics output, etc., and broadcasts this information to all the PEs. The Gaussian latitudes and weights are then read in and broadcast by the **gridfile.F**, after which **initsht.F** decomposes the latitudes over the PEs and initializes the various weights and seeds required for the Fourier and Legendre transforms.

The subroutine **initfile** and its dependencies deal with the initial conditions. In **startsht.F**, **initsc.F** reads the initial condition file (the output from **genini.F**) which contains the spectral coefficients of the prognostic variables. The subroutine **spectral_to_fourier.F** then transforms these initial spectral coefficients into Fourier space to be ready to start the loop over time.

Finally, in the case of the primitive equation version, the subroutine **verini.F** computes all the "vertical matrices" which result from the vertical discretization (following Saravanan, 1992; see also Appendix B) and **initforsc.F** computes the spectral coefficients of the forcing terms (for example, thermal relaxation and surface friction).

Figure 8 shows the calling tree of BOB during the time loop. The main part of this loop, involving the subroutine **sht.F**, is divided into four stages. At the beginning of the loop, the prog-

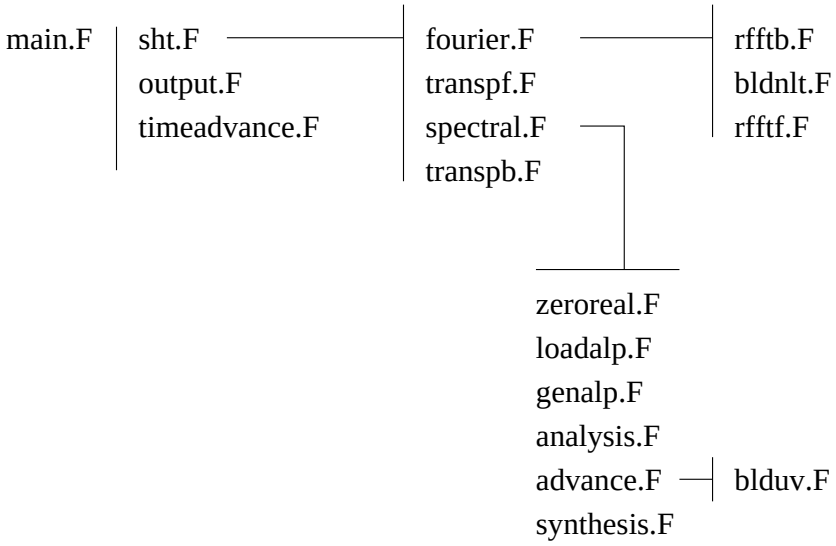

Figure 8: Calling tree for the main BOB program code during the loop over time.

nostic variables and velocity fields $\zeta, \delta, \Phi, U, V$ are in Fourier space, with latitude distributed over the PEs. The first stage of the loop, **fourier.F**, transforms these fields from Fourier space to physical space, calculates the nonlinear terms, $A, \ldots, F$ in physical space, and transforms the nonlinear terms from physical space to Fourier space, using, respectively, **rfftb.F**, **bldnlt.F**, and **rfftf.F**. Throughout this stage, the latitudes are distributed over the PEs so that the forward and inverse real FFTs can be done on a single PE for a whole latitudinal circle.

Since the Legnedre transform from Fourier space to spectral space requires a summation over the Gaussian latitudes, before this stage can be performed the decomposition of the arrays over the PEs must be rearranged so that all latitudes sit locally on each PE. Thus, the second stage, **transpf.F**, performs a transposition from a latitude space decomposition to a longitudinal mode decomposition, as described in section 3c. At the end of this stage the fields are still in Fourier space, but with the Fourier modes distributed over the PEs.

The third stage of the time loop involves all the operations in spectral space, including the time stepping of the prognostic equations, and is contained in the subroutine **spectral.F**. First the Legendre transforms are performed: **loadalp.F** loads the seed values of the associated Legendre polynomials that are then used by **genalp.F** to generate the Legendre coefficients "on the fly" as described in section 3b. By not storing the entire set of ALPs in memory, this procedure allows BOB to be run at high resolution with a low memory footprint. The Legendre transform itself is calculated in **analysis.F**: this transform is essentially a matrix-vector multiplication for each Fourier mode, summing over all the gaussian latitudes. Having the Fourier modes distibuted over the PEs therefore allows efficient parallelization of this stage. After **analysis.F**, all the fields $A, \ldots, F$ are in spectral space. The subroutine **advance.F** is where the actual leapfrog time-stepping takes place. The "new" spectral coefficients at time $t + \Delta t$ are computed using the spectral coefficients $A_n^m, \ldots, F_n^m$ as well as the spectral coefficients

of the prognostic variables (ζ, δ, Φ) at time t and $t - \Delta t$. At the end of **advance.F**, a call to **blduv.F** computes the wind field from the vorticity and divergence fields (see Hack and Jakob (1992)). The subroutine **synthesis.F** performs the inverse Legendre transforms on the new prognostic variables and wind field at time $t + \Delta t$.

We are now back in Fourier space with the Fourier modes *m* distributed over the PEs. The fourth stage, **transpb.F**, transposes the fields so that the latitudes are again distributed over the PEs. Thus, when the program returns to **fourier.F** the arrays are distributed such that the inverse FFTs can be done over all longitudes in the transformation back to physcial space. The full cycle in the time-stepping process is now complete.

At each timestep a call to **output.F** computes and outputs several diagnostic quantities. Note that when outputting variables in physical space, the prognostic fields used are those computed before the last call to **advance.F**, that is, before the last time-step. For this reason an extra timestep than the number requested is always performed at the of the integration. All output is serial so the fields are first gathered on the root PE before being output. Finally, **Timeadvance.F** is the subroutine which updates time as the timesteps are taken.

4 Obtaining and running the software

The source code for the shallow water and primitive equation versions of the model are available separately from http://www.scd.ucar.edu/; files **swbob.tar.gz** and **pebob.tar.gz** respectively. After unzipping and unpacking these archives have the following directory structure:

```
bob ---> bin
         grids
         postproc
         ics
         jobs
         run
         src
         test
```

Here and in what follows, we use **bob** to denote the main directory; that is **bob** is written for **swbob** or **pebob**, according to which version of the model is used. The source code itself is in the directory **src**. The directory **run** contains the run scripts used to compile the code and launch a model integration. The model is run in, and the model output is written to, a job-specific subdirectory of **jobs**. Files containing the initial conditions and the Gaussian grids are stored in **icns** and **grids**, respectively, and are created automatically by the grid initialization scripts. The **postproc** directory contains a support program to convert 2-D BOB binary output files to 2-D netcdf files.

In addition to the description of the run scripts in the following section, the README files that unpack with the model into the main **bob** directory provide further details of how to run the

model with various test cases. These test cases are described in section 4b below.

The code is written using the FORTRAN 77 subset of Fortran 90, with include files and namelist input parameters. The message passing uses MPI for interprocessor communications. Fortran 90 features such as dynamic memory allocation and array syntax notation have been avoided as these have introduced performance problems in the past. The Fourier transforms are carried out using FFTPACK 4.0 (Schwarztrauber, 1982).

4a. Description of the run scripts

In the **run** directory there are three run scripts to generate the Gaussian grid, the initial conditions, and to launch a model integration. In each of these run scripts, various parameters need to be defined to determine the resolution, integration length, job name, etc. To run the model for the first time, the *rungrid*, *runic*, and *runbob* scripts should be configured and executed in order.

(i) rungrid

Before the code can be run, the Gaussian grid for the appropriate resolution must first be generated (one needs to do this only once for a given resolution). Set the spectral resolution using the variable *res* in the run script *rungrid* (e.g. T170, T341, etc.). Now execute the script *rungrid*: this compiles and builds the executable **gridx** using the Makefile in **bob/src/grid**. The executable calculates the Gaussian latitudes and weights and outputs these in the grid file GRID.T*res*, where *res* is the spectral resolution. This grid file is then moved to the directory **bob/grids**.

(ii) runic

Once the grid has been generated, the initial condition can be computed using the run script *runic*. Again, set the spectral resolution in the run script, as well as the number of vertical model levels, *plev*[1], and the type of initial condition, *icn* (dependent on the particular test-case or integration to be done). Now execute the script *runic*: this compiles and builds the executable **inix**, using the Makefile in the appropriate *icn*-dependent subdirectory of **bob/src/genini**. The executable is launched using LoadLeveler if the script is running on AIX, or run directly on MacOS X.

The executable calculates the spectral coefficients of the initial condition on the prognostic variables and outputs these in the file *icn*.T*res*L*plev*.ini, where *icn*, *res*, and *plev* are as defined in **runic**. This file is then placed in the directory **bob/ics**.

[1]The parameter *plev* must be set even for the shallow water equations: by setting $plev > 1$, it is possible to run the shallow water equations in an uncoupled, stacked, multilevel mode, where *plev* defines the number of uncoupled layers to be run. Setting $plev = 1$ gives the usual single layer shallow water system.

Note that the only horizontal resolution parameter that needs to be set is the spectral resolution. The number of longitudes, *plon* is determined from the spectral resolution in the run scripts and placed in the include file, *params.h*. From this number, other dimensions, such as the number of latitudes *plat*, are computed in *dims.h*. Many dimensions only involve *plat*/2, taking advantage of the hemispherical symmetry of the associated Legendre polynomials (ALPs) as discussed in Rivier et al. (2002). In setting the number of longitudes, one needs to keep in mind that the number of latitudes divided by 2: *plat*/2 must be divisible by *nlat2* so that the tiling in physical space can be done correctly (see details of the tiling in section 3a above).

In addition to the initial condition file created in **bob/ics**, the script *runic* also creates a job directory *jobname* in **bob/jobs**, where *jobname* = *label*.*icn*.T*res*L*plev*.ic, for some identifier *label* defined in *runic*. The executable **inix** is run in this job directory and the following diagnostic files are copied/generated there:

runic:	a copy of the runscript,
inix.out:	an output file containing various WRITE statements from the code (standard out),
mpi.err, mpi.out:	error and output files generated by MPI,
*proclog.*PE#:	output files used for debugging purposes, containing information from each processing element (PE).

(iii) runbob

Once the initial condition is generated, one can run the main code. Set the various paramters in the run script *runbob* to define the resolution, length of integration, timestep, type of initial condition, number of nodes, etc. These parameters are used to create a namelist in an input file *bobx.in* that is read by the main program. Executing the script *runbob* creates this input file, compiles and builds the executable **bobx** using the Makefile in **bob/src/model/**, and submits the executable to the batch queue using *llsubmit* on AIX or runs it directly on MacOS X. The executable runs in the job directory **bob/jobs/***label*.*XX*.T*res*L*plev*. Similar diagnostic files as those for *runic* listed above are output in this directory. In addition, various model fields and physical quantities computed during the course of the integration are also output in this directory. See the README files provided with the model archive for details of the fields output in the various test cases.

(iv) Interactive job submission

The two run scripts, *runic* and *runbob* described above, run the model using the LoadLeveler batch submission protocol that may be machine specific. For testing and development purposes we have included two interactive scripts *runic_interactive* and *runbob_interactive* that are identical to *runic* and *runbob* except that they exit after the compilation of the code and the creation

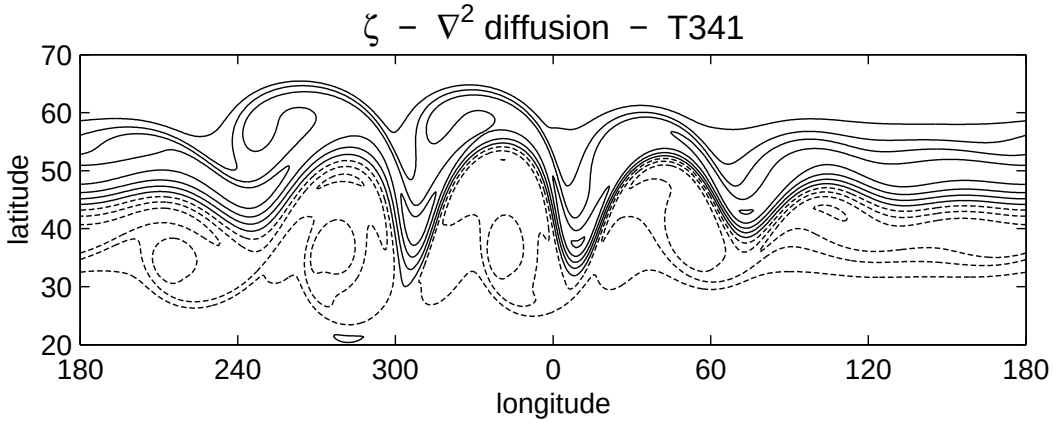

Figure 9: Relative vorticity after 160 hours for the shallow water barotropic instability test case, with a resolution of T341. Contour interval is $2 \times 10^{-5} \times (\ldots, -3, -1, 1, 3, \ldots)$ and negative contours are dashed. (From the output file bti.T341.z.0040.)

of the job directory. For example, an interactive PBS session can then be launched using a command such as

```
qsub -I -l nodes=4:ppn=2
```

following the instructions issued by the interactive script.

4b. Test case solutions

(i) Shallow water equations

To provide a means of verifying the correct installation of the code, we provide test cases that can be computed and checked against known solutions. The first of these, for the shallow water equation version, describes the evolution of a zonal jet from a perturbed, baroclinically unstable initial condition. Full details can be found in Galewsky et al. (2003). The model is run for 10 days for a fixed value of the diffusion coefficient, and the resolution is increased systematically until the solution has converged numerically. The vorticity field after 160 hours of this converged solution is shown in Figure 9.

To run the barotropic instability test case, the variable *icn* is set to *icn* = *bti* in the scripts *runic* and *runbob*. The horizontal resolution and the length of the timestep (as number of timesteps per day) should be set in conjunction, the timestep decreasing for increasing resolution to satisfy the CFL condition. Typical values for the pair (*res*, *tstd*) are (21,36), (42,72), (85,144), (170,288), (341,576). The total number of model days to be run can be set to *ndays* $= 10$. Finally, the order of the diffusion can be set to *delorder* $= 2$, for ordinary diffusion, or *delorder* $= 4$ for scale-selective hyperdiffusion. Having set the order of the diffusion, an appropriate diffusion coefficient is set automatically in the script. The values (*res*, *tstd*) = (341,576),

ndays $= 10$, and *delorder* $= 2$ were used to produce 9, the data for which is output in the file *jobname*.z.*dddd*. Also output from the model are the geopotential height and divergence fields, in the files *jobname*.t.*dddd* and *jobname*.d.*dddd*, respectively.

(ii) Primitive equations

Finally, we provide brief details of how to run two test cases with the primitive equation version of BOB. The first of these is a baroclinically unstable initial value problem presented in Polvani et al. (2004). The second is the Held and Suarez (1994) climatology.

A particular test case is selected by setting the variable *icn* in the scripts *runic* and *runbob*: *icn* = *ps* corresponds to Polvani et al. and *icn* = *hs* corresponds to Held-Suarez. In addition it is necessary to define *tstd*, the number of timesteps per day, *plev* the number of vertical levels, *ndays* the number of days to be integrated, and *delorder*, the order of the hyperdiffusion operator used to prevent enstrophy build up at the small scales.

For the Polvani et al. test case *tstd* should be set according to the spectral resolution *res*. Specifically, the pair (*res*, *tstd*) can take the values (21,36), (42,72), (85,144), (170,288), (341,576). That is the time-step should be halved for each doubling of the horizontal resolution. The number of vertical levels can be chosen to be any reasonable number, with 20 being the default. The order of the hyperdiffusion can be chosen to be 2 or 4. According to the definition *delorder* the diffusion coefficient is set to the appropriate value to reproduce the results of Polvani et al. (2004).

For a list of all the diagnostic files produced during the integration of this test case, see the file README_ps in the model archive. Among the diagnostics are files of the form *jobname*.z.*dddd*, containing the relative vorticity field in the whole domain on each of the days *dddd*. Figure 10 shows the relative vorticity at the lowest model level of the Polvani et al. test case with *delorder* $= 4$, at a spectral resolution of T341, with 20 vertical levels after 10 days of integration.

For the Held-Suarez test case the values of the variables that need to be set are: (*res*, *tstd*) $= (42, 72)$; *delorder* $= 8$; *plev* $= 20$, and *ndays* $= 1200$. Among the diagnostic fields output from this integration (again, see the README_hs for a full list) are the zonal averages of the zonal velocity and potential temperature fields in the files *jobname*.u.zavg and *jobname*.t.zavg respectively. These quantities are defined at the Gaussian latitudes, the model levels, and on each day. To reproduce the corresponding figure from Held and Suarez (1994) the time average of these fields is taken over the last 1000 days of the integration, discarding the initial 200 day spin-up period. These fields are shown in Figure 11.

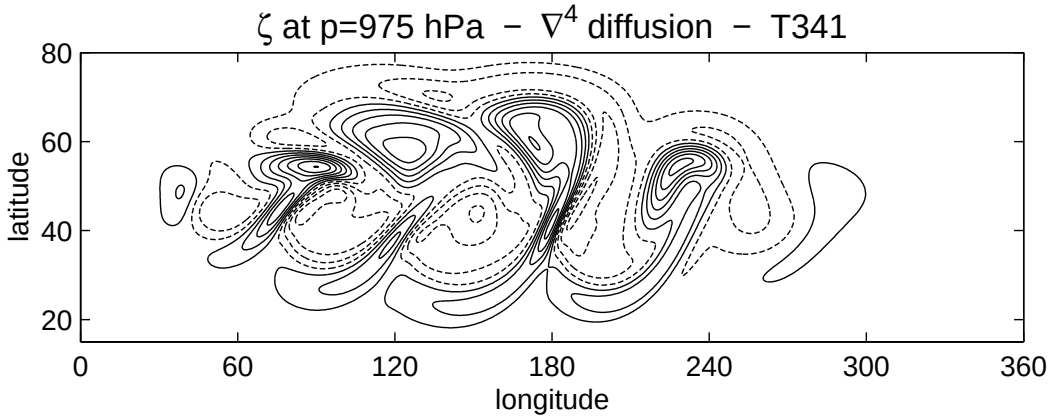

Figure 10: Surface (975 hPa) relative vorticity at day 10 for the primitive equation baroclinic instability test case, with a resolution of T341 with 20 vertical levels. Contour interval is $2 \times 10^{-5} \times (\ldots, -3, -1, 1, 3, \ldots)$ and negative contours are dashed. (From the output file ps.T341L20.z.0010.)

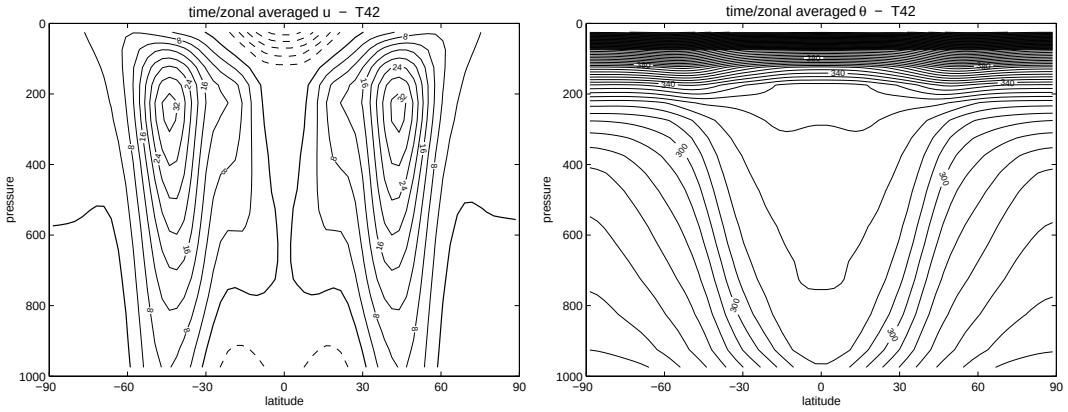

Figure 11: Time averaged and zonally averaged zonal velocity and potential temperature, run at T42 with 20 equally spaced pressure levels. Time average is over the last 1000 days of a 1200 day run. Contour interval for $\bar{u}$ is 4 m/s, negative contours dashed; contour interval for $\bar{\theta}$ is 5 K.

Acknowledgements

The BOB model was written at the National Center for Atmospheric Research and is the result of a collaboration between the Computational Science Section of the Scientific Computing Division of NCAR and the Department of Applied Physics and Applied Mathematics at Columbia University. The authors would like to thank the support received from the many individuals who have assisted in this work, particularly John Dennis and Dr. Steve Thomas at NCAR, and Dr. Joe Galewski at Columbia University. The authors gratefully acknowledge financial support received by NCAR through the Department of Energy Climate Change Prediction Program to perform this work.

References

Foster, I. T. and Worley, P. H., 1994: Parallel algorithms for the spectral transform method. ORNL/TM-12507, 40 pp.

Galewski, J., Scott, R. K., and Polvani, L. M. 2003: An initial-value problem for testing numerical models of the global shallow water equations. *Tellus B*, Submitted.

Hack, J. J. and Jakob, R., 1992: Description of a Shallow Water Model Based on the Spectral Transform Method. NCAR Technical Note NCAR/TN-343+STR, 39 pp.

Held, I. H. and Suarez, M. J., 1994: A proposal for the intercomparison of the dynamical cores of atmospheric general circulation models. *Bull. Amer. Meteor. Soc.*, **75,** 1825–1830.

Polvani, L. M., Scott, R. K., and Thomas, S., 2004: Numerically converged solutions of the global primitive equations for testing the dynamical core of atmospheric GCMs models, general circulation models. *Mon. Weather Rev.*, Submitted.

Rivier L., Loft R. and Polvani, L. M., 2002: An Efficient Spectral Dynamical Core for Distributed Memory Computers. *Mon. Weather Rev.*, **130,** No. 5, pp 1384–1390.

Saravanan R., 1992: A Mechanistic Spectral Primitive Equation Model using Pressure Coordinates. http://www.cgd.ucar.edu/gds/svn/svn.html.

Swarztrauber, P. N., 1982: Vectorizing the FFTs. *Parallel Computations*, G. Rodrigue, Ed., Academic Press, 51–83.

Swarztrauber, P. N., 1993: The vector harmonic transform method for solving partial differential equations in spherical geometry. *Mon. Weather Rev.*, **121,** 3415–3437.

Williamson, D. L., Drake, J. B, Hack, J. J., Jakob, R., and Swarztrauber, P. N., 1992: A standard test set for numerical approximations to the shallow water equations in spherical geometry. *J. Comput. Phys.*, **102,** 211–224.

Appendix A – The Vertical Matrices

In addition to the matrices, $M_{\theta\to\hat{\Phi}}$, $M_{\delta\to\omega}$, and $M_{\hat{\delta}\to\delta}$ defined by (2.14), (2.16), and (2.17), used in the discretization of the hydrostatic and continuity equations, various other "vertical" matrices are used in the formulation of the primitive equation version of the model. For completeness, these are defined here. Full details of these matrices and their construction can be found in Saravanan (1992).

The $(K-1)\times K$ matrix $M_{\hat{\delta}\to\delta}$ defined by (2.17) has a generalized left inverse $M_{\delta\to\hat{\delta}}$ with the property $M_{\delta\to\hat{\delta}}M_{\hat{\delta}\to\delta}=I$, where I is the $(K-1)\times(K-1)$ identity matrix. The $K\times(K-1)$ matrix $M_{\delta\to\hat{\delta}}$ can be written as:

$$[M_{\delta\to\hat{\delta}}]_{k,k'}=\begin{cases}-\frac{1}{2} & \text{if } k'=k,\\ +\frac{1}{2} & \text{if } k'=k+1,\\ 0 & \text{otherwise.}\end{cases} \tag{4.1}$$

This matrix is used to calculate the half-level values of a prognostic quantity such as the divergence defined on full levels.

Two other matrices arise from the separation of the potential temperature field into a reference profile $\theta^R(z)$ and departure $\theta'=\theta-\theta^R$. This separation is needed for the purposes of the semi-implicit time-stepping scheme, which treats terms associated with gravity waves implicitly and the nonlinear terms explicitly. Thus, we define a $K\times(K-1)$ matrix $M_{\omega\to H}$, indicating the operation from ω to the linearization about $\theta^R(z)$ of the nonlinear terms appearing on the right-hand side of the last equation in (2.3). This matrix has the form

$$[M_{\omega\to H}]_{k,k'}=\begin{cases}\frac{\hat{\theta}^R_{k+\frac{1}{2}}}{\Delta p_k} & \text{if } k'=k \text{ or } k'=k-1,\\ 0 & \text{otherwise.}\end{cases} \tag{4.2}$$

To allow for the calculation of these linear terms directly from the density, the $K\times K$ matrix $M_{\delta\to H}$ is then defined as the matirx product $M_{\delta\to H}=M_{\omega\to H}M_{\delta\to\omega}$. Similarly we define the $K\times K$ matrix $M_{\theta\to\Phi}$ as the matrix product $M_{\theta\to\Phi}=M_{\hat{\delta}\to\delta}M_{\theta\to\hat{\Phi}}$. This allows the direct computation on model full-levels of the $\nabla^2\Phi$ term in the right-hand side of the middle equation in (2.3).

A final set of vertical matrices arise from the implicit treatment of the hyperdiffusion terms added to the right-hand sides of (2.3). First, we define the following (implicit) diffusion factors:

$$\gamma_u^{corr}=\left[1+\alpha 2\Delta t\gamma_u\left(\nabla^2+\frac{a^2}{2}\right)^n\right]^{-1} \tag{4.3}$$

$$\gamma_\theta^{corr}=\left[1+\alpha 2\Delta t\gamma_\theta\nabla^{2n}\right]^{-1} \tag{4.4}$$

where $0\le\alpha\le 1$ is the implicitness of the scheme ($=0$ for explicit), Δt is the time step, n is the order of the hyperdiffusion, and γ_u and γ_θ are the hyperdiffusion coefficients. The term $a/2$

in (4.3) represents a correction appropriate to diffusion on a sphere of radius a, and ensures that a state of solid-body rotation is not affected by the diffusion. We then define two further $K \times K$ matrices by

$$M_{impcor1} = M_{\delta \to \hat{\delta}} \left[I - (\alpha 2 \Delta t)^2 \gamma_u^{corr} \gamma_\theta^{corr} \nabla^2 M_{\theta \to \hat{\Phi}} M_{\hat{\delta} \to H} \right]^{-1} M_{\hat{\delta} \to \delta} \gamma_u^{corr} \tag{4.5}$$

$$M_{impcor2} = M_{impcor1} \gamma_\theta^{corr} \nabla^2 M_{\theta \to \Phi} (\alpha 2 \Delta t) \tag{4.6}$$

where $M_{\hat{\delta} \to H} = M_{\delta \to H} M_{\hat{\delta} \to \delta}$. Because these matrices are operators in spectral space (because of the Laplacians) they also depend on spectral wavenumber.

All the above matrices are defined in the subroutine **verini.F**. Of these, the matrices $M_{\delta \to H}$, $M_{\Theta \to \delta}$, $M_{impcor1}$, and $M_{impcor2}$ are used in the time-stepping subtroutine **advance.F** and are stored in the header file **vertstruct.h**. The following list indicates the correspondence between the Fortran variable names in the program and the matrices defined above:

d2dcap	$\to$	$M_{\delta \to \hat{\delta}}$
dcap2d	$\to$	$M_{\hat{\delta} \to \delta}$
d2w	$\to$	$M_{\delta \to \omega}$
w2tt	$\to$	$-M_{\omega \to H}$
t2gpcp	$\to$	$M_{\Theta \to \hat{\Phi}}$
d2tt	$\to$	$-M_{\delta \to H}$
nnt2dt	$\to$	$M_{\hat{\delta} \to \delta} M_{\Theta \to \hat{\Phi}} = M_{\Theta \to \delta}$
impcor	$\to$	$M_{impcor1}$
impco2	$\to$	$M_{impcor2}$