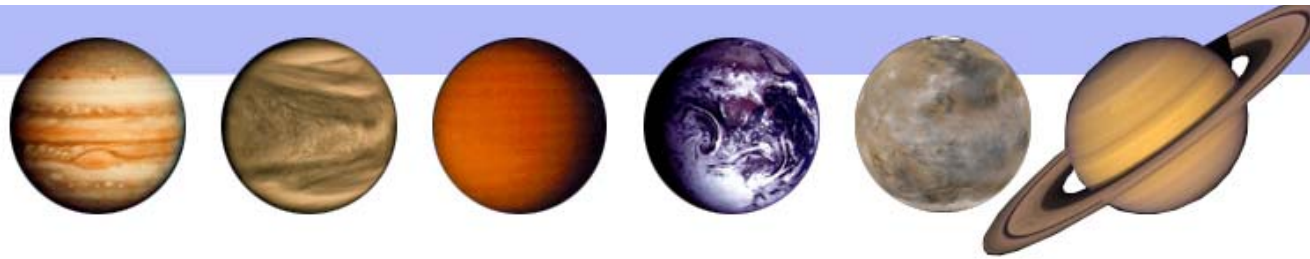
PlanetWRF

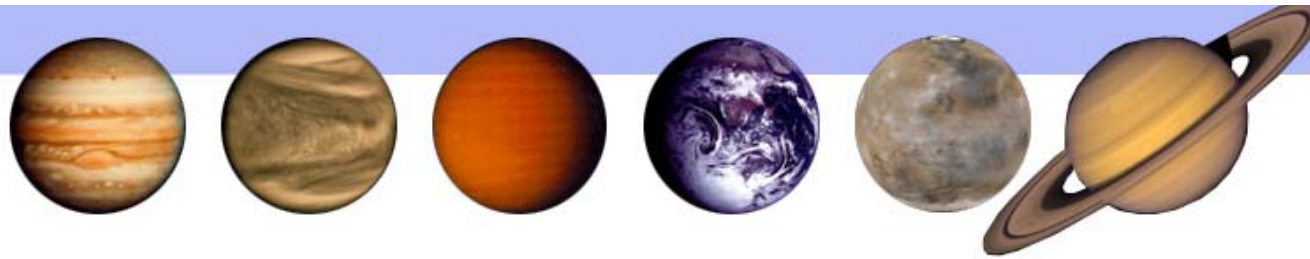


Non-Conformal Projection, Global, and Planetary Versions of WRF

MM5/WRF Workshop, Boulder, CO *June, 2005*

Mark I. Richardson, Anthony D. Toigo, Claire E. Newman **Caltech**

PlanetWRF



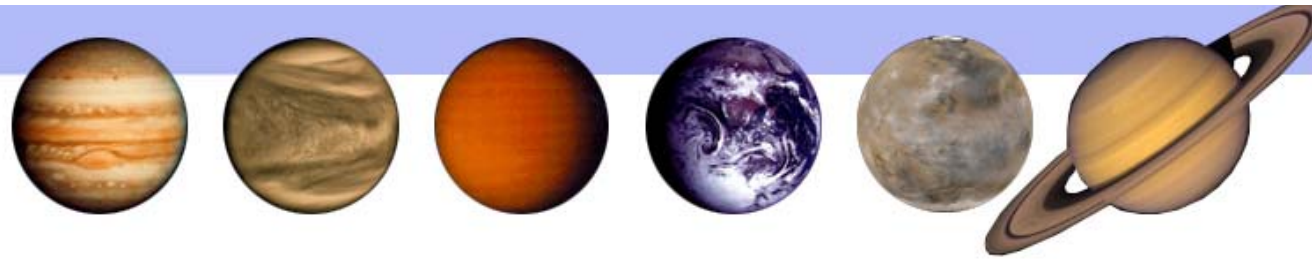
Goal: Develop WRF to be a "one stop" tool for planetary atmospheric dynamical modeling

Need:
- Global version of WRF
- Planetary version of WRF
- One-Dimensional (SCM) version of WRF

Constraint:
- Don't break anything! (backward compatible)
- Make more general, not more specific

Mark I. Richardson, Anthony D. Toigo, Claire E. Newman Caltech

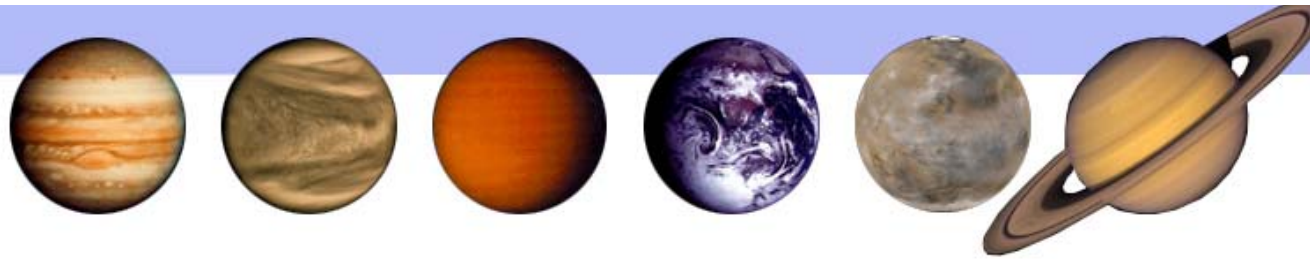
PlanetWRF



Globalization

- Traditional GCM or two-pole model?
  - We want both
  - Latter involves nesting and domain feedback - wait till this is done
  - Traditional GCM requires east-west periodic b.c.'s and north-south "polar" b.c.'s
  - To get standard GCM global coverage, need simple cylindrical projection <- i.e. non-conformal map projection…

PlanetWRF



Map Projection

$$\frac{\partial}{\partial X} = \frac{1}{h_x}\frac{\partial}{\partial x} = m_x \frac{\partial}{\partial x}$$

$$\frac{\partial}{\partial Y} = \frac{1}{h_y}\frac{\partial}{\partial y} = m_y \frac{\partial}{\partial y}$$

WRF is already projection non-specific - you can use polar, lambert, and mercator

To be flexible, WRF uses definable map scale factors (*msf* in the code)

Here, *X* is the physical distance and *x* is the computational domain distance - the *msf* relates one to the other

PlanetWRF

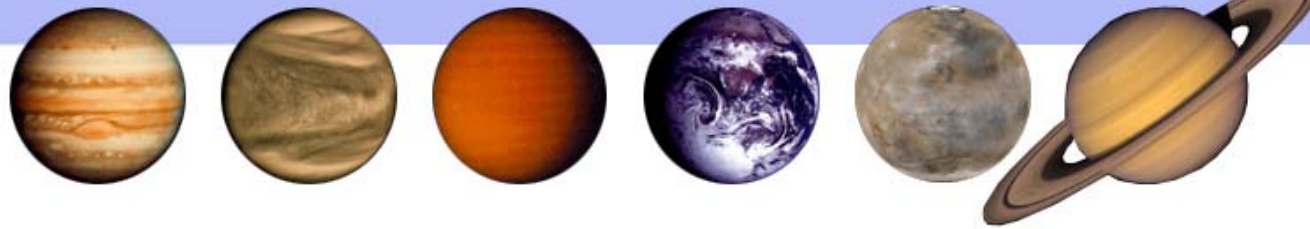
Map Projection

$$\frac{\partial}{\partial X} = \frac{1}{h_x}\frac{\partial}{\partial x} = m_x \frac{\partial}{\partial x}$$
$$\frac{\partial}{\partial Y} = \frac{1}{h_y}\frac{\partial}{\partial y} = m_y \frac{\partial}{\partial y}$$

But, WRF assumes that the *x*- and *y*-directional factors at any point are identical (m_x=m_y) -> this makes the equations easier to write and is valid for a whole class of projections used in mesoscale modeling

- not so for standard GCM
- first task was to implement separated *msf*'s throughout the em dynamical core (NOTE: we have only done this for ARW)

PlanetWRF



Map Projection

$$\frac{\partial}{\partial t}\left(\frac{\rho u}{m_y}\right) = -m_x\frac{\partial}{\partial x}\left(\frac{\rho uu}{m_y}\right) - m_x\frac{\partial}{\partial y}\left(\frac{\rho uv}{m_x}\right) - \frac{1}{m_y}\frac{\partial}{\partial z}(\rho uw)$$
$$+\frac{\rho}{m_y}\left(\frac{uv\tan\phi}{a} - \frac{uw}{a} - 2w\Omega\cos\phi + 2v\Omega\sin\phi - m_x\frac{1}{\rho}\frac{\partial p}{\partial x} + F_x\right)$$

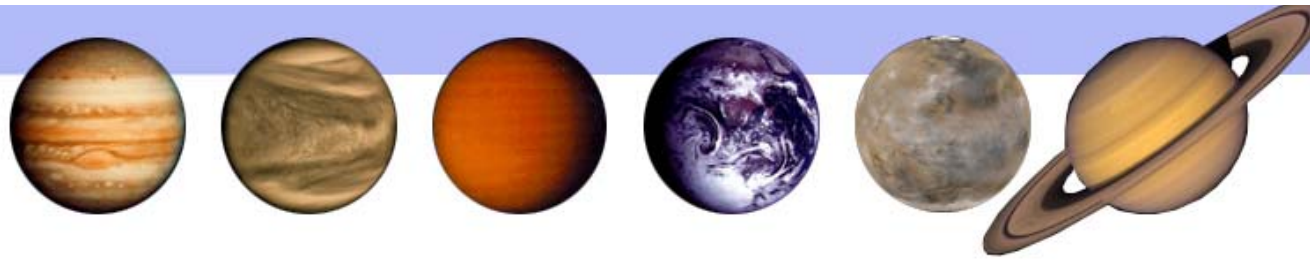
$$\frac{\partial}{\partial t}\left(\frac{\rho v}{m_x}\right) = -m_y\frac{\partial}{\partial x}\left(\frac{\rho uv}{m_y}\right) - m_y\frac{\partial}{\partial y}\left(\frac{\rho vv}{m_x}\right) - \frac{1}{m_x}\frac{\partial}{\partial z}(\rho vw)$$
$$+\frac{\rho}{m_x}\left(\frac{u^2\tan\phi}{a} - \frac{vw}{a} + 2u\Omega\sin\phi - m_y\frac{1}{\rho}\frac{\partial p}{\partial y} + F_y\right)$$

$$\frac{\partial}{\partial t}(\rho w) = -m_x m_y\frac{\partial}{\partial x}\left(\frac{\rho uw}{m_y}\right) - m_x m_y\frac{\partial}{\partial y}\left(\frac{\rho vw}{m_x}\right) - \frac{\partial}{\partial z}(\rho ww)$$
$$+\rho\left(\frac{u^2+v^2}{a} + 2u\Omega\cos\phi - \frac{1}{\rho}\frac{\partial p}{\partial z} + F_z\right)$$

$$\frac{\partial \rho}{\partial t} = -m_x m_y\left[\frac{\partial}{\partial x}\left(\frac{\rho u}{m_y}\right) + \frac{\partial}{\partial y}\left(\frac{\rho v}{m_x}\right)\right] - \frac{\partial}{\partial z}(\rho w)$$

Mark I. Richardson, Anthony D. Toigo, Claire E. Newman **Caltech**

PlanetWRF



Map Projection

$$x = a\lambda$$
$$y = a\phi$$
$$dx = a\,d\lambda$$
$$dy = a\,d\phi$$
$$dX = a\cos\phi\,d\lambda$$
$$dY = a\,d\phi$$

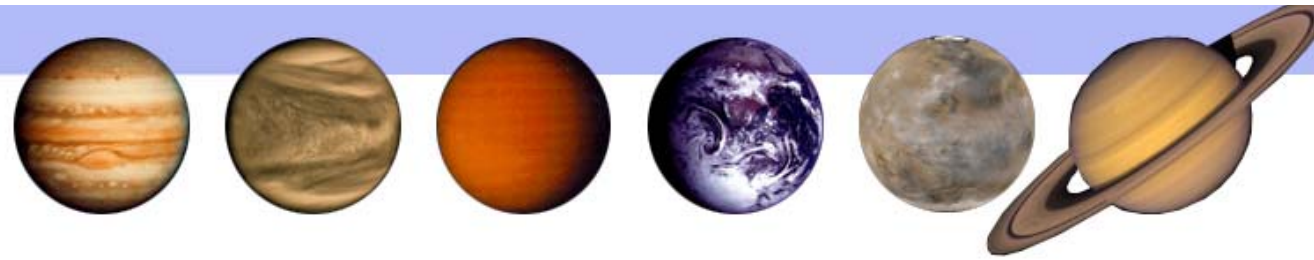
$$m_x \equiv \frac{dx}{dX}$$
$$m_y \equiv \frac{dy}{dY}$$
$$m_x = \sec\phi$$
$$m_y = 1$$

Implemented and tested in all of the EM (ARW) dynamical core EXCEPT option 2 diffusion and options 2 and 3 diffusion coefficient

The code is now designed to use the x- and y- directional factors. The conformal domains used in WRF to-date all still work as a special condition in which $m_x=m_y$

A version of *ideal.exe* was written to generate initial and boundary conditions for a WRF GCM run (only surface b.c.'s needed)

PlanetWRF



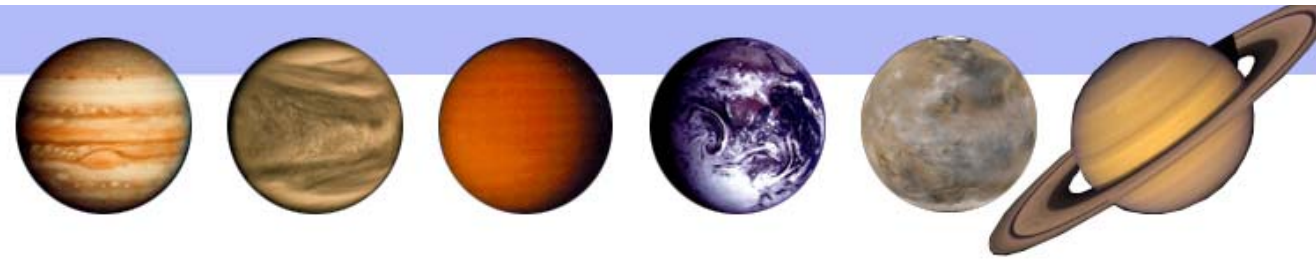
Example code from horizontal diffusion in module_big_step_utilities.F

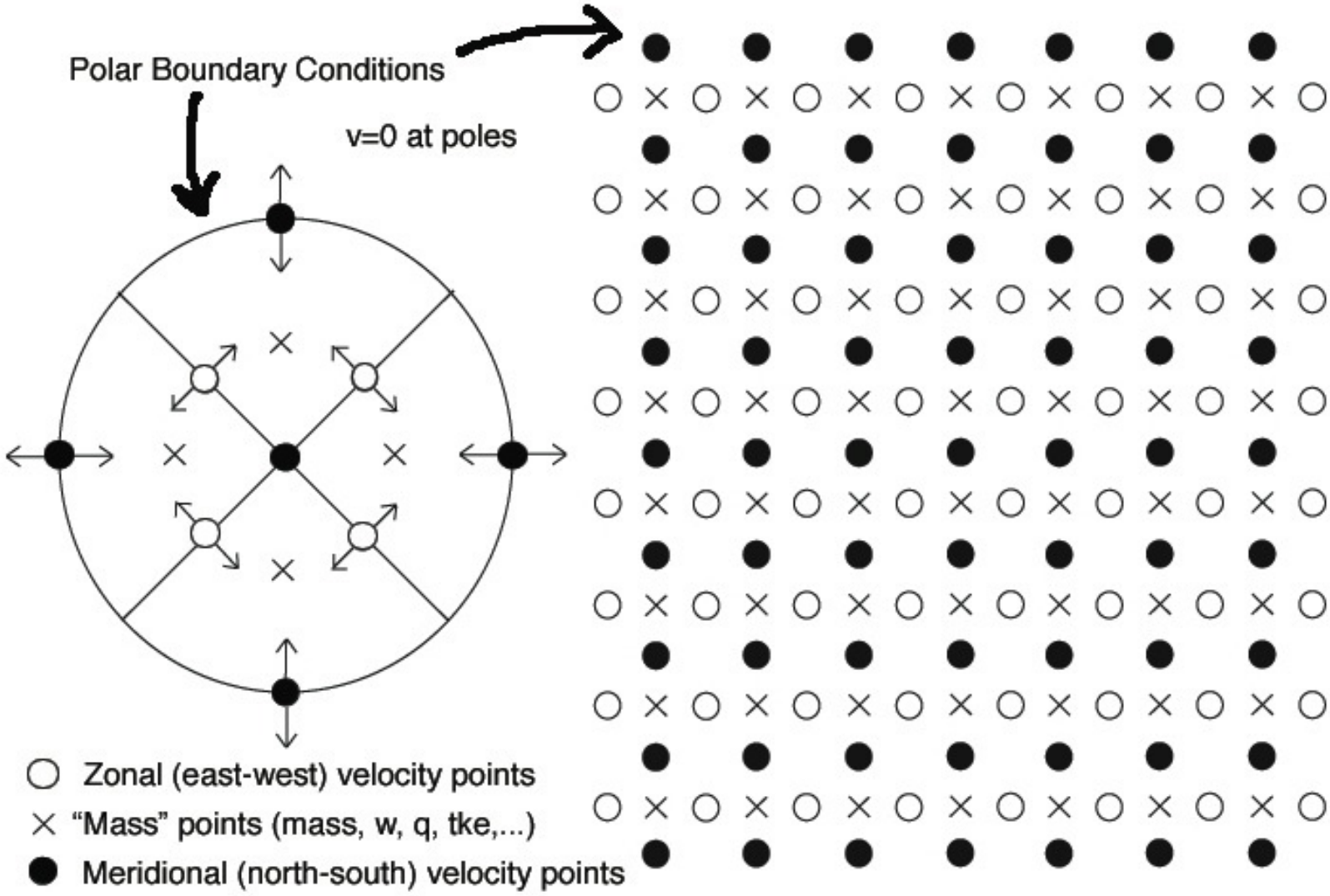
```
DO j = j_start, j_end
DO k=kts,ktf
DO i = i_start, i_end

   mkrdxm=msft(i-1,j)*xkmhd(i-1,k,j)*rdx
   mkrdxp=msft(i,j)*xkmhd(i,k,j)*rdx
   mrdx=msfu(i,j)*rdx
   mkrdym=0.5*(msfu(i,j)+msfu(i,j-1))*    &
          0.25*(xkmhd(i,k,j)+xkmhd(i,k,j-1)+xkmhd(i-1,k,j-1)+xkmhd(i-1,k,j))*rdy
   mkrdyp=0.5*(msfu(i,j)+msfu(i,j+1))*    &
          0.25*(xkmhd(i,k,j)+xkmhd(i,k,j+1)+xkmhd(i-1,k,j+1)+xkmhd(i-1,k,j))*rdy
   mrdy=msfu(i,j)*rdy

      rcoup=0.5*(mu(i,j)+mu(i-1,j))
      tendency(i,k,j)=tendency(i,k,j)+rcoup*( &
                      mrdx*(mkrdxp*(field(i+1,k,j)-field(i  ,k,j))  &
                           -mkrdxm*(field(i  ,k,j)-field(i-1,k,j))) &
                     +mrdy*(mkrdyp*(field(i,k,j+1)-field(i,k,j  ))  &
                           -mkrdym*(field(i,k,j  )-field(i,k,j-1))))

ENDDO
ENDDO
ENDDO
```

```
DO j = j_start, j_end
DO k=kts,ktf
DO i = i_start, i_end

   ! The interior is grad: (m_x*d/dx), the exterior is div: (m_x*m_y*d/dx(/m_y))
   ! setting up different averagings of m^2 partial d/dX and m^2 partial d/dY

   mkrdxm=(msftx(i-1,j)/msfty(i-1,j))*xkmhd(i-1,k,j)*rdx
   mkrdxp=(msftx(i,j)/(msfty(i,j))*xkmhd(i,k,j))*rdx
   mrdx=msfux(i,j)*msfuy(i,j)*rdx
   ! we have no pre-cal'd msf:
   mkrdym=0.5*( (msfuy(i,j)+msfuy(i,j-1))/ (msfux(i,j)+msfux(i,j-1))  )*    &
          0.25*(xkmhd(i,k,j)+xkmhd(i,k,j-1)+xkmhd(i-1,k,j-1)+xkmhd(i-1,k,j))*rdy
   IF( j == jds ) mkrdym = 0.
   mkrdyp=0.5*( (msfuy(i,j)+msfuy(i,j+1))/ (msfux(i,j)+msfux(i,j+1))  )*    &
          0.25*(xkmhd(i,k,j)+xkmhd(i,k,j+1)+xkmhd(i-1,k,j+1)+xkmhd(i-1,k,j))*rdy
   IF( j == jde-1 ) mkrdyp = 0.
   ! need to do four-corners (t) for diffusion coefficient as there are
   ! no values at u,v points
   ! msfuy - has to be y as part of d/dY
   !         has to be u as we're at a u point
   mrdy=msfux(i,j)*msfuy(i,j)*rdy

   ! correctly averaged version of rho~ * m^2 *
   !    [partial d/dX(partial du^/dX) + partial d/dY(partial du^/dY)]
      rcoup=0.5*(mu(i,j)+mu(i-1,j))
      tendency(i,k,j)=tendency(i,k,j)+rcoup*( &
                      mrdx*(mkrdxp*(field(i+1,k,j)-field(i  ,k,j))  &
                           -mkrdxm*(field(i  ,k,j)-field(i-1,k,j))) &
                     +mrdy*(mkrdyp*(field(i,k,j+1)-field(i,k,j  ))  &
                           -mkrdym*(field(i,k,j  )-field(i,k,j-1))))

ENDDO
ENDDO
ENDDO
```

Same loop: original WRFV2 and global/non-conformal WRFV2

PlanetWRF





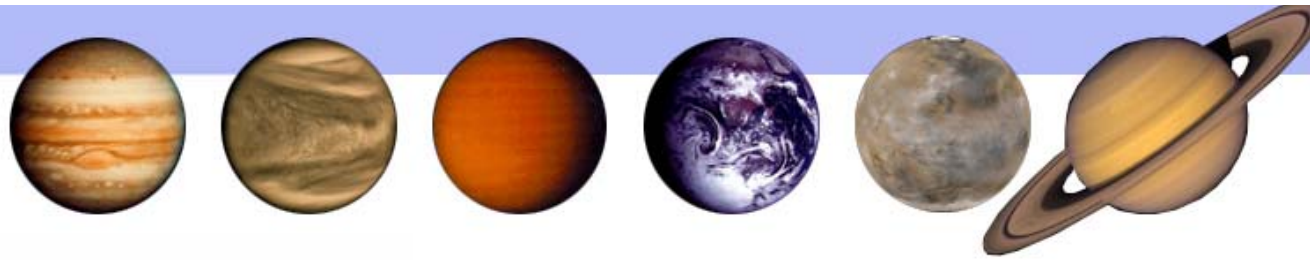
Standard polar b.c.'s for a c-grid GCM have the v-point at the pole with a no-flow constraint

implemented for the WRF GCM

Mark I. Richardson, Anthony D. Toigo, Claire E. Newman

Caltech

PlanetWRF



```
IF(j > j_start) THEN

  DO k=kts,ktf
  DO i = i_start, i_end
    mrdy=msfu(i,j-1)*rdy
    tendency(i,k,j-1) = tendency(i,k,j-1) - mrdy*(fqy(i,k,jp1)-fqy(i,k,jp0))
  ENDDO
  ENDDO

ENDIF
```

Original WRFV2

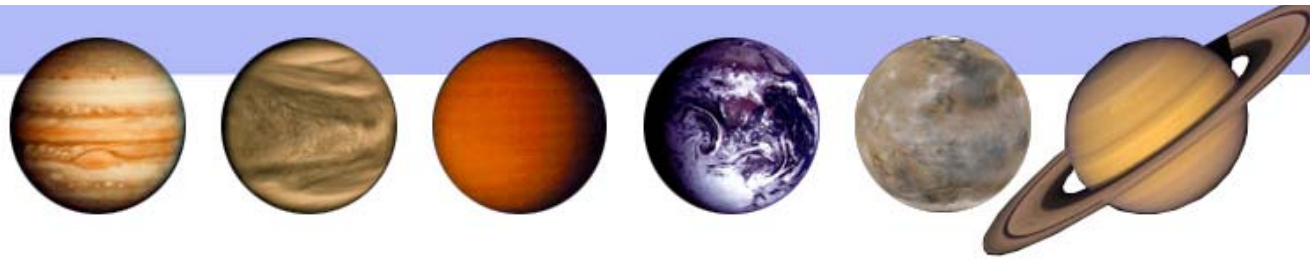
Global / polar WRFV2

```
IF( (j == jds+1) .and. (config_flags%polar) ) THEN
   DO k=kts,ktf
   DO i = i_start, i_end
      mrdy=msfux(i,j-1)*rdy                ! ADT eqn 30, 2nd term on RHS
      tendency(i,k,j-1) = tendency(i,k,j-1) - mrdy*fqy(i,k,jp1)
   END DO
   END DO
ELSE IF( (j == jde-1) .and. (config_flags%polar) ) THEN
   DO k=kts,ktf
   DO i = i_start, i_end
      mrdy=msfux(i,j)*rdy                  ! ADT eqn 30, 2nd term on RHS
      tendency(i,k,j) = tendency(i,k,j) + mrdy*fqy(i,k,jp1)
      mrdy=msfux(i,j-1)*rdy                ! ADT eqn 30, 2nd term on RHS
      tendency(i,k,j-1) = tendency(i,k,j-1) - mrdy*(fqy(i,k,jp1)-fqy(i,k,jp0))
   END DO
   END DO
ELSE  ! normal code

IF(j > j_start) THEN

  DO k=kts,ktf
  DO i = i_start, i_end
    mrdy=msfux(i,j-1)*rdy                  ! ADT eqn 30, 2nd term on RHS
    tendency(i,k,j-1) = tendency(i,k,j-1) - mrdy*(fqy(i,k,jp1)-fqy(i,k,jp0))
  ENDDO
  ENDDO

ENDIF
END IF
```

Same code from `module_advect` - conditionals protect one-sided advection at the pole

Mark I. Richardson, Anthony D. Toigo

PlanetWRF



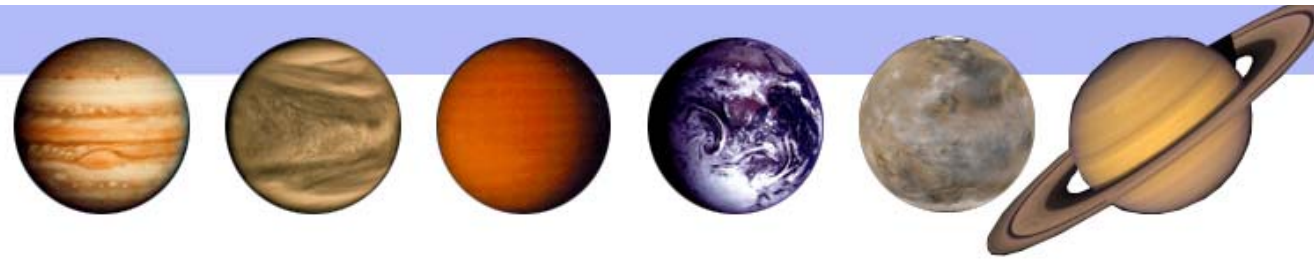
Namelist file entries:

```
&bdy_control
periodic_x                          = .true., .false.,.false.,
symmetric_xs                        = .false.,.false.,.false.,
symmetric_xe                        = .false.,.false.,.false.,
open_xs                             = .false.,.false.,.false.,
open_xe                             = .false.,.false.,.false.,
periodic_y                          = .false.,.false.,.false.,
symmetric_ys                        = .false.,.false.,.false.,
symmetric_ye                        = .false.,.false.,.false.,
open_ys                             = .false.,.false.,.false.,
open_ye                             = .false.,.false.,.false.,
nested                              = .false., .true., .true.,
polar                               = .true.,  .true., .true.,
/
```

"Plain-old" periodic_x b.c.'s work just fine

New projection option "polar", when set to "true" activates the polar boundary condition at both poles and allows polar FFT and zonal-mean damping

PlanetWRF



Namelist file entries:

```
&dynamics
dyn_opt                             = 2,
rk_ord                              = 3,
diff_opt                            = 1,
km_opt                              = 1,
damp_opt                            = 101,
zdamp                               = 4000.,  4000.,  4000.,
dampcoef                            = 0.2,    0.2,    0.2
khdif                               = 100000, 0,      0,
kvdif                               = 0,      0,      0,
smdiv                               = 0.1,    0.1,    0.1,
emdiv                               = 0.0,    0.01,   0.01,
epssm                               = 0.1,    0.1,    0.1
non_hydrostatic                     = .false.,.true., .true.,
time_step_sound                     = 8,      4,      4
h_mom_adv_order                     = 5,      5,      5,
v_mom_adv_order                     = 3,      3,      3,
h_sca_adv_order                     = 5,      5,      5,
v_sca_adv_order                     = 3,      3,      3,
polar_filter                        = 2,
one_d_model                         = .false.,
pert_coriolis                       = .false.,
fft_filter_type                     = 1,
/
```

Rayleigh damping option

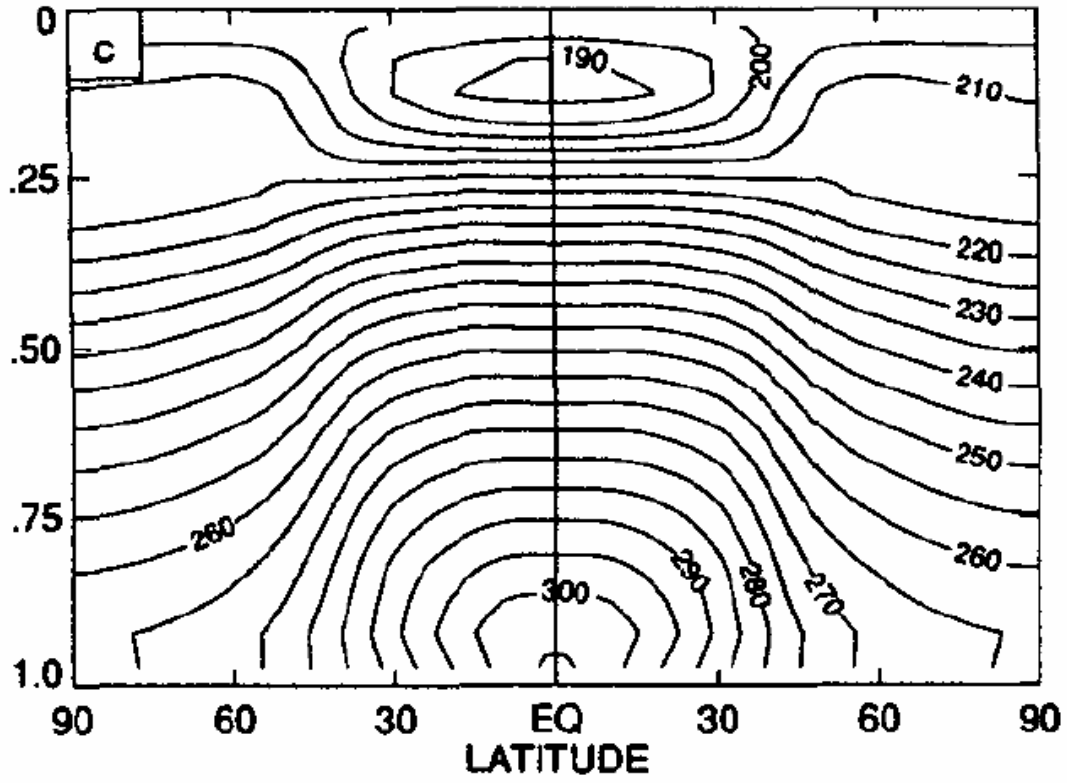
If non-zero, the polar_filter option activates polar FFT, some choice of where and what variables

We'll talk about this later on

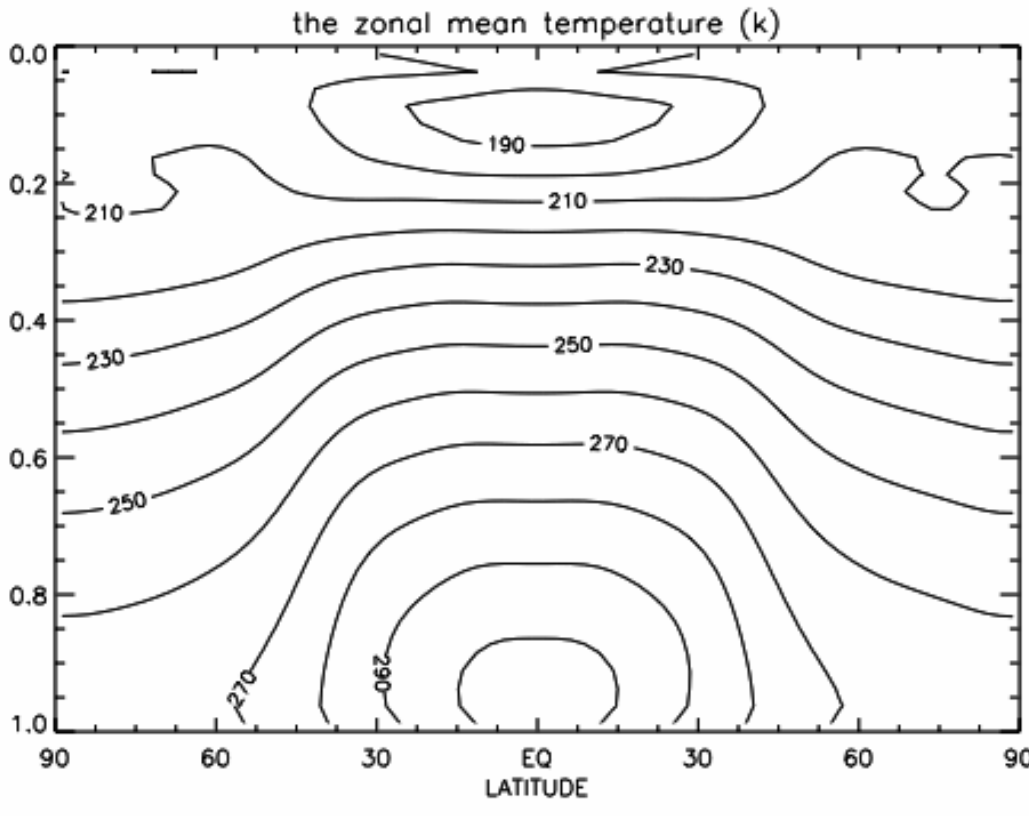
Options for cut-off and gouge filter functions

PlanetWRF

Held-Suarez Tests - temperature

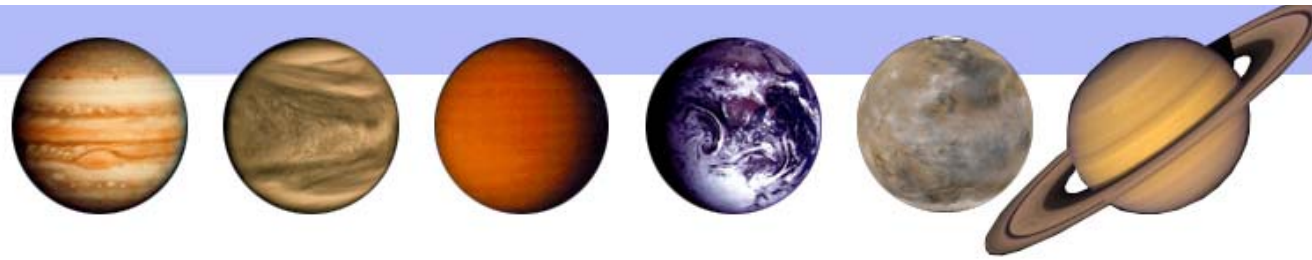


Held and Suarez, 1994

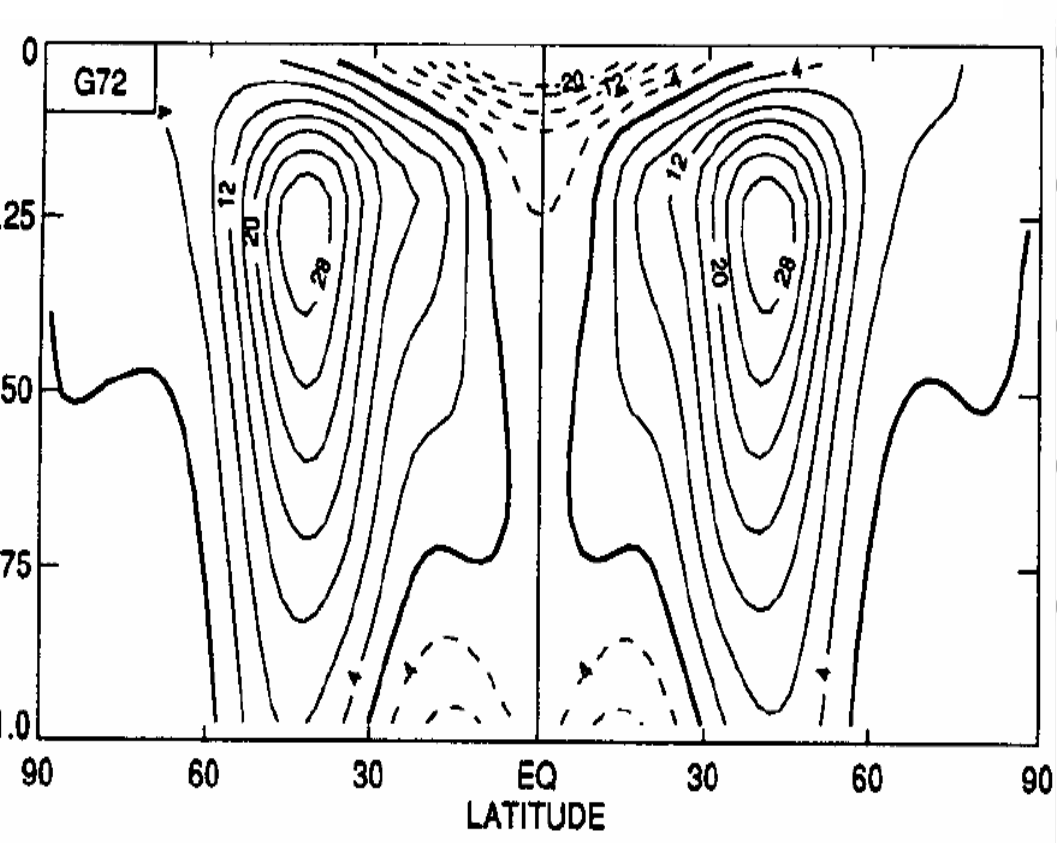


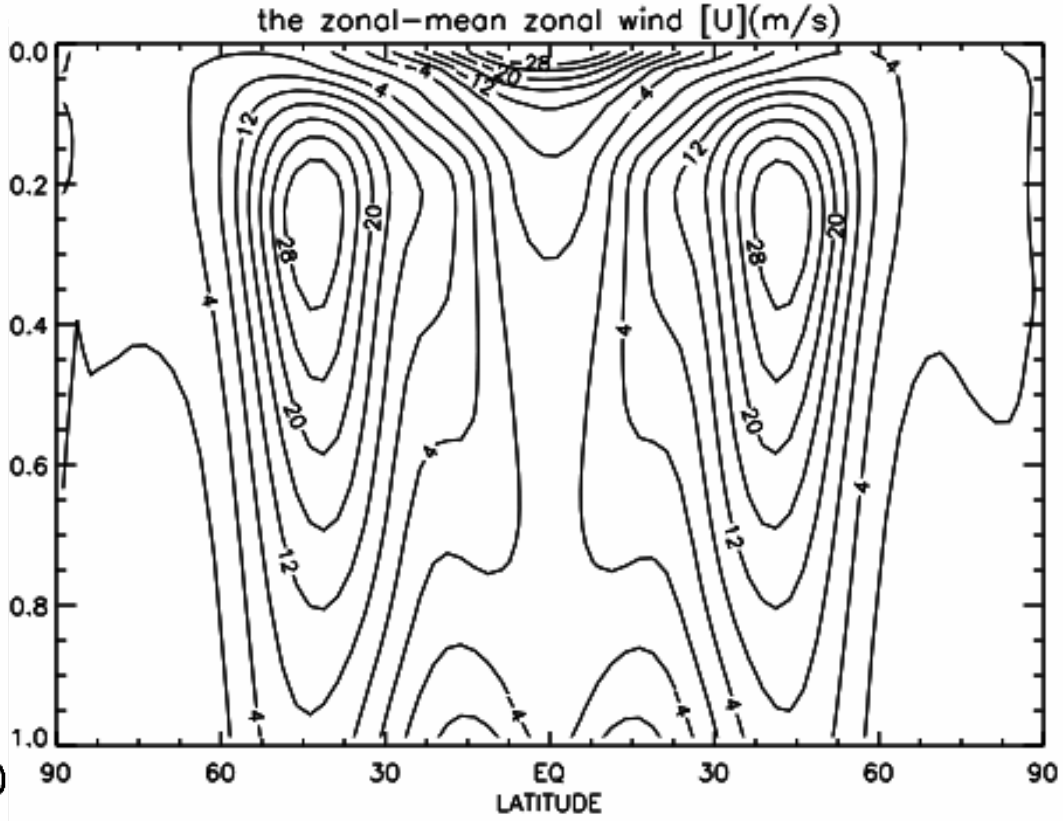
WRF GCM

PlanetWRF



Held-Suarez Tests - zonal wind

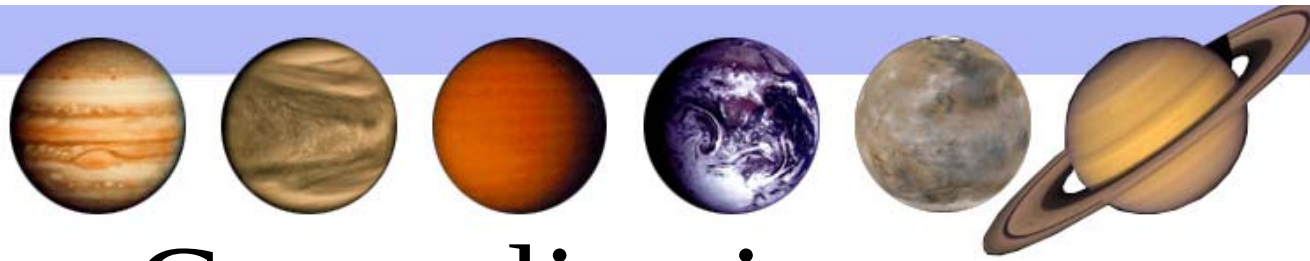




Held and Suarez, 1994

WRF GCM

PlanetWRF



Planetary Generalization

```
ideal.exe*
namelist.input
wrfbdy_d01
wrf.exe*
wrfinput_d01
wrfout_d01_0001-00001_00:00:00
wrfout_d01_0001-00238_00:00:00
wrfout_d01_0001-00556_00:00:00
wrfrst_d01_0001-00556_00:00:00
```

No month
5 digit days

Added the concept of planetocentric solar longitude (L_s) for dates

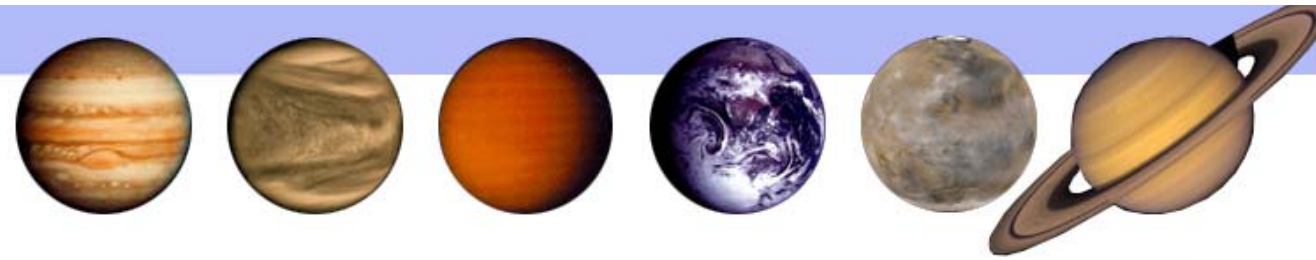
In "planetary" mode:
- month is meaningless
- use planetary and SI time - have *p2si* variable that gives conversion

Calendar system changes and planetary physical constants activated by compile-time option selected via "./configure"

```
! JM NOTE -- can we name this grav instead?
#ifdef MARS
!  Mars
   REAL    , PARAMETER :: g = 3.711  ! acceleration due to gravity (m {s}^-2)
#else
#ifdef TITAN
!  Titan
   REAL    , PARAMETER :: g = 1.358  ! acceleration due to gravity (m {s}^-2)
#else
   REAL    , PARAMETER :: g = 9.81  ! acceleration due to gravity (m {s}^-2)
#endif
#endif
```

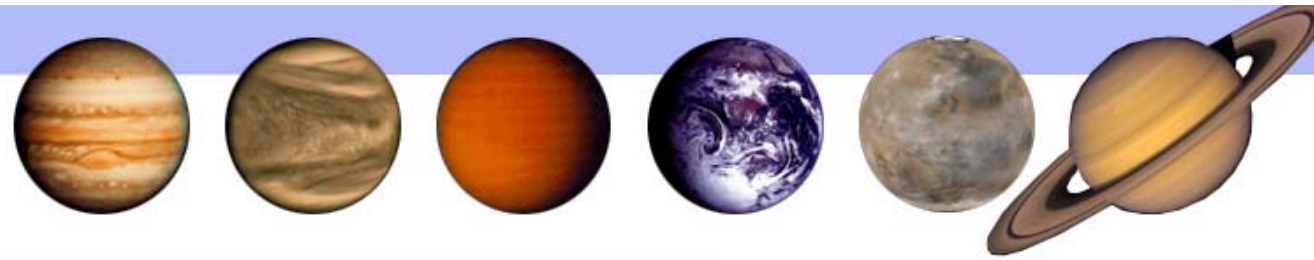
← Please!? ;-)

PlanetWRF



```
balmy:/home/galileo3/mir/ADTa/WRFV2> ./configure
checking for perl5... no
checking for perl... found /usr/bin/perl (perl)
Will use NETCDF in dir: /opt/netcdf
PHDF5 not set in environment. Will configure WRF for use without.
------------------------------------------------------------------------
Please select from among the following supported platforms.

   1.  PC Linux i486 i586 i686, PGI compiler  (Single-threaded, no nesting)
   2.  PC Linux i486 i586 i686, PGI compiler  (single threaded, supports nesting using RSL without MPI)
   3.  PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, no nesting)
   4.  PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, supports nesting using RSL without MPI)
   5.  PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL, MPICH, support nesting)
   6.  PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL_LITE, MPICH, No nesting)
   7.  Intel xeon i686 ia32 Xeon Linux, ifort compiler (single-threaded, no nesting)
   8.  Intel xeon i686 ia32 Xeon Linux, ifort compiler  (single threaded, supports nesting using RSL without MPI)
   9.  Intel xeon i686 ia32 Xeon Linux, ifort compiler (OpenMP)
  10.  Intel xeon i686 ia32 Xeon Linux, ifort compiler SM-Parallel (OpenMP, supports nesting using RSL without MPI)
  11.  Intel xeon i686 ia32 Xeon Linux, ifort compiler  DM-Parallel (RSL, MPICH, supports nesting)
  12.  MARS: PC Linux i486 i586 i686, PGI compiler  (Single-threaded, no nesting)
  13.  MARS: PC Linux i486 i586 i686, PGI compiler  (single threaded, supports nesting using RSL without MPI)
  14.  MARS: PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, no nesting)
  15.  MARS: PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, supports nesting using RSL without MPI)
  16.  MARS: PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL, MPICH, support nesting)
  17.  MARS: PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL_LITE, MPICH, No nesting)
  18.  MARS: Intel p4 i686 ia32 Linux, ifort compiler (Single-threaded, no nesting)
  19.  TITAN: PC Linux i486 i586 i686, PGI compiler  (Single-threaded, no nesting)
  20.  TITAN: PC Linux i486 i586 i686, PGI compiler  (single threaded, supports nesting using RSL without MPI)
  21.  TITAN: PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, no nesting)
  22.  TITAN: PC Linux i486 i586 i686, PGI compiler  SM-Parallel (OpenMP, supports nesting using RSL without MPI)
  23.  TITAN: PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL, MPICH, support nesting)
  24.  TITAN: PC Linux i486 i586 i686, PGI compiler  DM-Parallel (RSL_LITE, MPICH, No nesting)
  25.  TITAN: Intel p4 i686 ia32 Linux, ifort compiler (Single-threaded, no nesting)

Enter selection [1-25] : 
```

PlanetWRF

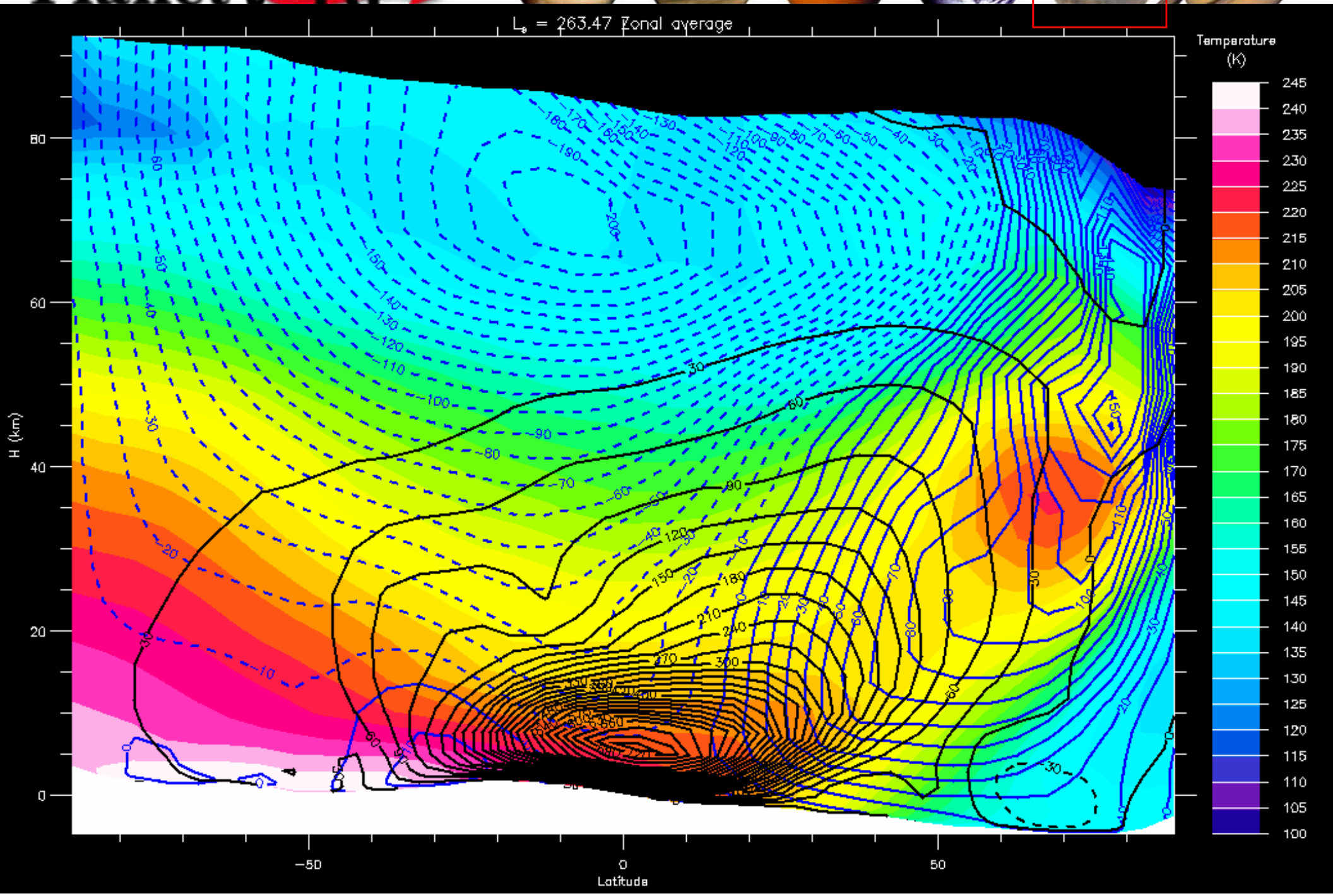


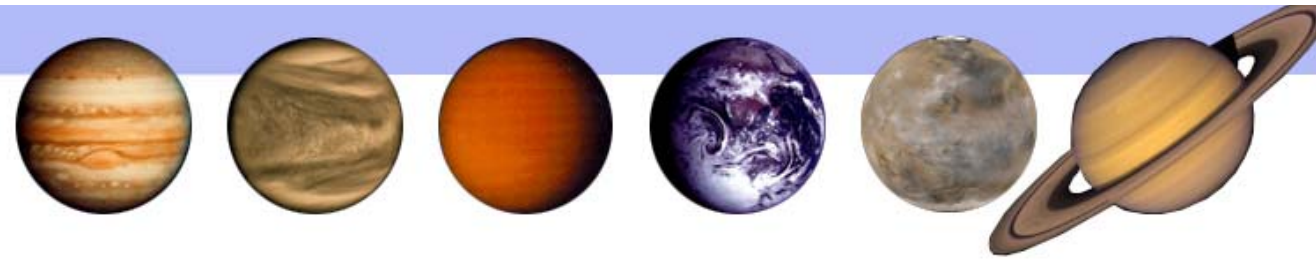
```
Enter selection [1-25] : 12
------------------------------------------------------------------------

You have chosen: MARS: PC Linux i486 i586 i686, PGI compiler  (Single-threaded, no nesting)
These are the default options for this platform:
------------------------------------------------------------------------
#
FC              =       pgf90
LD              =       pgf90
CC              =       gcc
SFC             =       $(FC)
CFLAGS          =
FCOPTIM         =       -fast
FCDEBUG         =       #-g
#FCBASEOPTS      =       -w -byteswapio -Ktrap=fp -Mfree -tp p6 $(FCDEBUG)
FCBASEOPTS      =       -w -byteswapio -Mfree -tp p6 $(FCDEBUG)
FCFLAGS         =       $(FCOPTIM) $(FCBASEOPTS)
ARCHFLAGS       =       -DDEREF_KLUDGE -DIO_DEREF_KLUDGE -DIWORDSIZE=4 -DDWORDSIZE=8 -DRWORDSIZE=4 -DLWORDSIZE=4 \
                         -DNETCDF \
                        -DTRIEDNTRUE   \
                        -DLIMIT_ARGS
INCLUDE_MODULES =       -module ../main -I../external/io_netcdf -I../external/io_int -I../external/esmf_time_f90 \
                        -I../external/psmf_time_f90 \
                        -I../frame -I../share -I../phys -I../inc
EXTRAMODULES    =
PERL            =       perl
REGISTRY        =       Registry
LIB             =       -L../external/io_netcdf -lwrfio_nf -L/opt/netcdf/lib -lnetcdf \
                        -L../external/io_grib1 -lio_grib1 \
                        ../frame/module_internal_header_util.o ../frame/pack_utils.o -L../external/esmf_time_f90 -lesmf
_time \
                        -L../external/psmf_time_f90 -lpsmf_time
LDFLAGS         =       $(FCFLAGS) -byteswapio
CPP             =       /lib/cpp -C -P -traditional
POUND_DEF       =       $(COREDEFS) -DNONSTANDARD_SYSTEM -DF90_STANDALONE -DCONFIG_BUF_LEN=$(CONFIG_BUF_LEN) -DMAX_DOMA
INS_F=$(MAX_DOMAINS) -DPLANET -DMARS
CPPFLAGS        =       -I$(LIBINCLUDE) -C -P $(ARCHFLAGS) $(POUND_DEF)
AR              =       ar ru
M4              =       m4
RANLIB          =       ranlib
NETCDFPATH      =       /opt/netcdf
CC_TOOLS        =       $(CC)

------------------------------------------------------------------------
These will be written to the file configure.wrf here in the top-level
directory.  If you wish to change settings, please edit that file.
If you wish to change the default options, edit the file:

     arch/configure.defaults

Configuration successful. To build the model type compile .
------------------------------------------------------------------------
```

"PSMF" calendar system based on ESMF routines

Planet and Mars options

PlanetWRF


Ls = 263.47 Zonal average
Temperature (K)
245
240
235
230
225
220
215
210
205
200
195
190
185
180
175
170
165
160
155
150
145
140
135
130
125
120
115
110
105
100
H (km)
80
60
40
20
0
−50
0
50
Latitude



One-dimensional model

Use x- and y- direction domain sizes of 2 and 2 (for u and v defn)

Use x- and y- periodic boundary conditions

Assume that u and v are ageostrophic components, with the geostrophic components defined by *u_frame* and *v_frame*

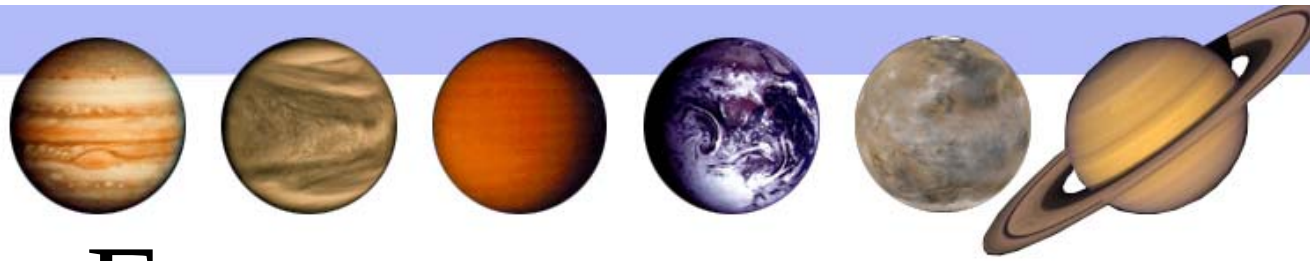
Modify a `module_initialize_xxx` routine to define initial values

Force map scale factors all to be unity

Force *e* and *f* to be the same at “all four” points

(Can easily also make an axisymmetric WRF using similar tricks…)

PlanetWRF



Future

- Want to infuse global (and 1D/2D) option back into main release WRF
- Haven’t spent anytime on real.exe or on ideal.exe initialize routine for Earth WRF GCM
- Problem with current polar FFT: can’t break zones - very inefficient after nproc>8… need to fix
- Test nesting in a global WRF
- Want to build a seamed, two-pole global WRF similar to the implementation for global MM5