CESM Tutorial

CAM Physics:

Interfacing a parameterization

Cécile Hannay and Dani Coleman

National Center for Atmospheric Research (NCAR

Outline of this presentation

- What is an interface
  - in theory
  - in CAM
- Model structure related to CAM physics
- Example of physics parameterization interface: convect_deep
- Physics Utilities
  - user defined (derived) types or structues:
    physics_state, physics_ptend
  - chunks, cols
  - constants
  - phys_control
  - constituents
  - physics buffer
  - surface exchange types
  - simpler ways to run CAM

Physics Interface Design

- http://www.cesm.ucar.edu/models/atm-cam/docs/phys-interface/
- It is old* so some details have changed but the fundamental design still applies

*2002; going to be updated… someday

- Reference for coding standards: Kalnay, E., M. Kanamitsu, J. Pfaendtner, J. Sela, M. Suarez, J. stackpole, J. Tuccillo, L. Umscheid, and D. Williamson, 1989: Rules for Interchange of Physics Parameteriza- tions. Bull. Am. Met. Soc., 70(6), 620-622

Why an interface?

1) Make parameterization 'play nice' with the model (required)

2) Make parameterization portable between models (recommended)

Interfacing Requirements

‘play nice’ with the model

A physics parameterization

1. Must calculate a tendency (rate of change)
2. Must not change the model state
3. Must conserve vertical integrals of
   1. mass
   2. momentum
   3. total energy
   4. dry static energy

The physics package in CAM takes the tendencies from parameterizations and calculates the new state while checking energy and water balances.

Interfacing recommendation

portability between models

and different versions of the same model

The parameterization should consist of two parts:

1. An interface layer to communicate between CAM and the parameterization
2. The parameterization package, with as little of CAM structures in it as possible

This is more likely to make a parameterization which is portable between different models or model versions.

What is an interface in CAM?

A fortran module containing a set of methods (subroutines or functions) called by CAM's physics driver which calls methods in the parameterization module(s). Also applies to chemistry parameterizations.

The public methods of a CAM interface (PARAM = generic parameterization) . See online document for details (Utility Modules section); browse code for examples.

- PARAM_register

  This method is for registering fields that are managed by the physics buffer module, and for registering constituents in the constituent arrays.
- PARAM_implements_cnst

  A query method which returns true if the requested constituent name is implemented by the package.
- PARAM_init_cnst

  A package that manages constituents is responsible for initializing the constituent mixing
- PARAM_init

  This method is for package specific initialization including setting time-invariant
- constants, specifying fields to be included in the history files, and opening datasets.
- PARAM_timestep_init

  This method is for per-timestep initialization, (e.g. time interpolation from a boundary dataset.
- PARAM_timestep_tend

  This method calls the package run method which computes the one step tendencies.

Model structure: CAM's interface

The top-level of CAM interfaces to the coupled model:

models/atm/cam/src/control/cam_comp.F90:

```
subroutine cam_init:
        call initindx              dynamics
        call phys_init             physics
subroutine cam_run1(cam_in, cam_out)
        !  Runs first phase of dynamics and first phase of
        !  physics (before surface model updates).
        call stepon_run1           dynamics
        call phys_run1             physics
subroutine cam_run2( cam_out, cam_in )
        !  Run the second phase physics, run methods that
        !  require the surface model updates.  And run the
        !  second phase of dynamics that at least couples
        !  between physics to dynamics.
        call phys_run2             physics
        call stepon_run2           dynamics
```

Model structure: CAM physics

- CAM defines 'physics' as processes that are done in the vertical column without horizontal communication.
- The physics processes are divided between those that happen before and after coupling with the component models and

```
physpkg.F90
     registration:  <calls to subroutines>
     initialization:
              subroutine phys_inidat
              subroutine phys_init
      timestepping:
              subroutine phys_run1: call tphysbc
              subroutine phys_run2: call tphysac
```

before coupling

after coupling

Model structure: CAM physics

CAM defines 'physics' (vs 'dynamics') as processes that are done in the vertical column without horizontal communication. Module physpkg calls tphysbc and tphysac; in these modules you can see everything included in physics.

grep call /home/s07hsu00/cesm_collection/cesm1_0_3/models/atm/cam/src/physics/cam/tphys*c.F90

(edited)

tphysbc before coupling

```
call convect_deep_tend( prec_zmc, …
call convect_shallow_tend (ztodt …
call stratiform_tend(state, ptend, …
call macrop_driver_tend(state, ptend, …
call microp_driver_tend(state, ptend, …
call aerosol_wet_intr (state, ptend, …
call convect_deep_tend_2( state, ptend, …
call radiation_tend(state,ptend,pbuf, …
```

tphysac after coupling

```
call aerosol_emis_intr (state, ptend, …)
call tracers_timestep_tend(state, ptend, …
call aoa_tracers_timestep_tend(state, ptend, …
call chem_timestep_tend(state, ptend, cam_in,
call vertical_diffusion_tend (ztodt ,state ,…
call rayleigh_friction_tend( ztodt, state, ptend)
call aerosol_drydep_intr (state, ptend,, ztodt, &
call qbo_relax(state,ptend,state%uzm)
call iondrag_calc( lchnk, ncol, state, ptend, pbuf
call iondrag_calc( lchnk, ncol, state, ptend, pbuf
```

These calls are to methods in parameterization interfaces

Example of an interface

Deep convection interface (registration and initialization)

Interface = models/atm/cam/src/physics/cam/convect_deep.F90 methods call:

Parameterization/ package: zm_conv.F90 (Zhang-Macfarlane)

1. Registration (allocates memory)

```
initindx.F90:      call convect_deep_register
convect_deep.F90: subroutine convect_deep_register
                          call zm_conv_register
```

2. Initialization (done once at the beginning of model run or restart)

```
physpkg.F90:       call convect_deep_init
convect_deep.F90: subroutine convect_deep_init
                          call zm_conv_init
```

physpkg doesn't know anything about the deep convection package, so a new parameterization could be swapped in. Also, higher level changes in the model shouldn't touch zm_conv.F90; just convect_deep.F90

Ex. Deep convection interface (time-stepping)

3. Time-stepping (as the model is running)

```
tphysbc.F90:       call convect_deep_tend(state, ptend,…)
                   call physics_update(state, ptend,…)
                   call check_energy_chng
```

The physics package gets the tendencies from the convection interface, calculates the new state and checks that energy and water were conserved.

```
convect_deep.F90: subroutine convect_deep_tend(state, ptend,…)
        intent(in) state
        intent(out) ptend
        ptend%name = "convect_deep”
        call zm_conv_tend(… state    ,ptend …
```

The interface can only modify tend, not state. It calls the parameterization method

physics_types.F90

~/cesm_collection/cesm1_0_3/models/atm/cam/src/physics/cam/physics_types.F90

```
module physics_types
! Public types:

  public physics_state
  public physics_tend
public physics_ptend
```

User-derived types (structures): state & ptend are shown on next page

```
! Public interfaces

  public physics_update
  public physics_ptend_reset
  public physics_ptend_init
  public physics_state_set_grid
  public physics_dme_adjust  ! adjust dry mass and energy for change in water
                             ! cannot be applied to eul or sld dycores
  public physics_state_copy  ! copy a state type
  public physics_ptend_sum   ! add 2 ptend types
  public physics_tend_init   ! initialize a tend type
```

Subroutines that are used by physpkg and physics parameterizations

Data structures: physics_state

```
defined in models/atm/cam/src/physics/cam/physics_types.F90
type physics_state
integer ::&
        lchnk, &! chunk index
        ncol ! Number of columns
real(r8) ::&
        calday ! calendar day at end of current timestep
        real(r8), dimension(pcols) ::&
        lat, &! latitude (radians)
        lon, &! longitude (radians)
        ps, &! surface pressure (Pa)
        phis ! surface geopotential
real(r8), dimension(pcols,pver) ::&
        t, &! temperature (K)
        u, &! zonal velocity (m/s)
        v, &! meridional velocity (m/s)
        dse, &! dry static energy (J/kg)
        omega, &! vertical velocity (Pa/s)
        pmid, &! pressure at midpoints (Pa)
        pdel, &! pdel(k) = pint(k+1) - pint(k)
        rpdel, &! 1./pdel(k)
        lnpmid, &! ln(pmid)
        zm, &! geopotential height above surface, at midpoint (m)
        exner ! inverse exner func w.r.t. surface pressure (ps/p)^(R/cp
real(r8), dimension(pcols,pver+1) ::&
        pint, &! pressure at interface (Pa)
```

e.g.

```
subroutine convect_deep_tend(..., state,...

use physics_types, only: physics_state
type(physics_state), intent(in ) :: state

lchnk = state%lchnk
ncol = state%ncol

temp = state%t(:ncol,:)
```

Declare dummy variable named ‘state’, of type physics_state.
Intent(in) means it cannot be modified by the subroutine

Data structures: physics_ptend

(parameterization tendency)

```
 defined in models/atm/cam/src/physics/cam/physics_types.F90
type physics_ptend
   character*24 :: name    ! name of parameterization which produced tendencies.

   logical ::          &
      ls,           &! true if dsdt is returned
      lu,           &! true if dudt is returned
      lv,           &! true if dvdt is returned
      lq(pcnst)       ! true if dqdt() is returned

   integer ::          &
      top_level,      &! top level index for which nonzero tendencie
      bot_level        ! bottom level index for which nonzero tenden

   real(r8), dimension(pcols,pver)          :: &
      s,            &! heating rate (J/kg/s)
      u,            &! u momentum tendency (m/s/s)
      v              ! v momentum tendency (m/s/s)
   real(r8), dimension(pcols,pver,pcnst) :: &
      q              ! consituent tendencies (kg/kg/s)

! boundary fluxes
   real(r8), dimension(pcols) ::&
      hflux_srf,     &! net heat flux at surface (W/m2)
      hflux_top,     &! net heat flux at top of model (W/m2)
      taux_srf,      &! net zonal stress at surface (Pa)
      taux_top,      &! net zonal stress at top of model (Pa)
      tauy_srf,      &! net meridional stress at surface (Pa)
      tauy_top        ! net meridional stress at top of model (Pa)
   real(r8), dimension(pcols,pcnst) ::&
      cflx_srf,     &! constituent flux at surface (kg/m2/s)
```

e.g.

```
subroutine convect_deep_tend(..., ptend,...

use physics_types, only: physics_ptend
type(physics_ptend), intent(out) :: ptend

ptend(:ncol,:) = new_tendency(:ncol,:)
```

Declare dummy variable named 'ptend', of type physics_ptend.
Intent(out) means it can be modified by the subroutine and is set to zero at the beginning.

Data structures: chunks

The fundamental data structure used by the physics driver contains an arbitrary collection of vertical columns, and is referred to as a **chunk**.

- There are no assumptions about the horizontal location of the columns, e.g., they are not necessarily neighbors in the global grid.)
- the module **phys grid.F90** provides query functions that return
  - the number of columns in each chunk
  - latitude, longitude coordinates of the individual columns in a chunk.
- SPMD parallelism is done over the chunk dimension
- A single call to either tphysbc or tphysac passes only a single chunk of the decomposed physics grid
- The data on a chunk is a variable of a derived type (structure) that contains multiple fields

CAM Array Dimensions

Physics

- pcols = PCOLS, &! maximum number of columns in a chunk
- pver = PLEV, &! number of vertical levels
- pcnst = PCNST, &! number of advected constituents (including water vapor)
- ppcnst= PCNST+PNATS ! number of constituents operated on by physics

Dynamics

- plon = PLON, &! number of longitudes in the global dynamics grid
- plev = PLEV, &! number of levels in the global dynamics grid
- plat = PLAT, &! number of latitudes in the global dynamics grid

Note that currently CAM uses the same number of vertical levels in both the dynamics (PLEV) and physics (PVER) grid but it doesn't have to.

Utility modules

- Physical constants: shr_const_mod module

```
real(r8), parameter ::&
    r_universal = 6.02214e26*1.38065e-23, &! Universal gas constant (J/K/kmol)
    mwdry = 28.966, &! molecular weight dry air
    mwco2 = 44., &! molecular weight co2
    …
```

- phys_control.F90

```
public :: &
  phys_ctl_readnl,   &! read namelist from file
  phys_getopts,      &! generic query method
  phys_deepconv_pbl, &! return true if deep convection is allowed in the PBL
  phys_do_flux_avg,  &! return true to average surface fluxes
  cam_physpkg_is,    &! query for the name of the physics package
  cam_chempkg_is      ! query for the name of the chemistry package
```

Constituents

The constituents module is responsible for managing the names and physical properties of all trace constituents in a model run.

- It assigns the index values in the constituent arrays,
- keeps track of whether or not the initial values of each constituent are to be read from the initial file.

The packages that implement constituents (e.g., chemistry packages) are responsiblefor registering the names and properties of the constituents with the constituents module, which can then make these values known to other packages that require them

Physics buffer

The module phys_buffer manages the physics buffer which stores fields

- that must be available across timesteps
  - automatically writes fields to the restart files
- OR that must be shared between physics packages
  - avoids passing the field through subroutine argument list from one parameterization up to tphys*c and back down to another parameterization
- Complicated format, follow an example in the code!
  - pbuf_register
  - pbuf_add('fld_name',scope, fdim, mdim, ldim, index)
  - fld_idx = pbuf_get_fld_idx('fld_name')
  - fld => pbuf(fld_idx%fld_ptrfield(fdim,pcols,mdim,begchunk:endchunk,ldim)

surface exchange types

```
> cd /home/s07hsu00/cesm_collection/cesm1_0_3/models/atm/cam/src/physics/cam
> less physpkg.F90
                    use camsrfexch_types, only: cam_out_t, cam_in_t
> cd /work/s07hsu31/cesm_run/case01/atm/obj
> ~s07hsu00/findccm camsrfexch_types.F90
12 directories
/home/s07hsu00/cesm_collection/cesm1_0_3/models/atm/cam/src/control
> less /home/s07hsu00/cesm_collection/cesm1_0_3/models/atm/cam/src/control/camsrfexch_types.F90
 public cam_out_t              ! Data from atmosphere
  public cam_in_t              ! Merged surface data
!-----------------------------------------------------------------------
! This is the data that is sent from the atmosphere to the surface models
!-----------------------------------------------------------------------
  type cam_out_t
     integer  :: lchnk            ! chunk index
     integer  :: ncol             ! number of columns in chunk
     real(r8) :: tbot(pcols)      ! bot level temperature
     real(r8) :: zbot(pcols)      ! bot level height above surface
     real(r8) :: ubot(pcols)      ! bot level u wind
     real(r8) :: vbot(pcols)      ! bot level v wind
     real(r8) :: qbot(pcols,pcnst)  ! bot level specific humidity
```

Stand-alone and Single-Column CAM

- Users who are developing CAM might be interested in using the stand-alone CAM configure/run instead of dealing with the entire CESM structure

  cesm_collection/cesm1_0_3/models/atm/cam/bld> ls

  build-namelist   configure   run-pc.csh   (and more)

- Single-column CAM is a good tool for developing physics parameterizations
  - no dynamics
  - runs with field-experiment data

  ? cesm_collection/cesm1_0_3/models/atm/cam/bld/run-scam.csh

Get help: CESM bulletin board

NCAR UCAR DiscussCESM Community Earth System Model

FORUMS REGISTER LOGIN

Home » Forums

FORUMS

View Forums Active topics Unanswered topics

CESM - General

The Community Earth System Model (CESM) is a fully coupled, global climate model that provides state-of-the-art computer simulations of the Earth's past, present, and future climate states.

Forum	Topics	Posts	Last post
Announcements	15	39	CESM1_0_5 issues by jedwards March 20, 2013 - 2:14pm
Bug reporting	103	287	problem with... by dbailey February 20, 2013 - 10:41am

Atmospheric Modeling with CAM

The Community Atmosphere Model (CAM) is the atmosphere model component of the CESM. Information about running CAM as the atmospheric component of the CESM is found in the CESM release documentation. For information on CAM microphysics, visit the CAM Microphysics Development Group. Please see the Whole Atmosphere Community Climate Model Forum and the Climate Chemistry Forum for topic discussions specific to these capabilities of CAM.

Forum	Topics	Posts	Last post
			What is the...