Subcolumns in CAM
Cheryl Craig1, Steve Goldhaber1, Andrew Gettelman1, Julio Bacmeister1, Brian Eaton1
National Center for Atmospheric Research, PO Box 3000, Boulder, CO 80307

Introduction
While CAM provides a grid for use by physics parameterizations, there are phenomena for which a smaller scale is desirable. In other cases, an ensemble approach within a single grid column can improve parameterization performance. The CAM subcolumn infrastructure provides the ability to create fields with several elements within a single grid column. It supports a variable number of subcolumns per grid column. While some physics parameterizations are currently run on a sub-grid scale (e.g., radiation, SPCAM), the new infrastructure provides a common subcolumn infrastructure for use by multiple parameterizations.

Please note that these changes are transparent to existing parameterizations. No code changes are required in order to work within the new infrastructure.

This poster provides an introduction to infrastructure changes made to support subcolumns, examples of subcolumn usage in CAM physics, and a brief description of how subcolumns are created and managed.

IF YOU PLAN ON IMPLEMENTING SUBCOLUMNS: CONTACT US – THIS IS A WORK IN PROGRESS

Infrastructure changes on CAM development trunk
Dynamic state/end/ptend changes
Changes to implement subcolumns
plot is still maximum number of grid columns / chunk state*sacol is still the number of columns to loop over, but may be larger than plots if using subcolumns
plots is the maximum number of subcolumns = 1 for grid
state*splots is the maximum number of total columns whether using grid or subcolumns = plots*subplots (replaces plots in a lot of places)

New Fields to support subcolumns

Microphysics and Subcolumns
Code snippets from micro_mg_cam

(right colorized, ( indicates function description at right)
if (use_subcol_microp) then

! Allocate subcolumn structures
  state_sc{splots} = state*{plots} * subplots call physics_state_alloc(state_sc, lchknk, state_sc*splots)
call physics_tend_alloc(tend_sc, state_sc*splots)

! Generate subcolumns using the requested scheme
call subcol_genstate(tend, state_sc, tend_sc, pfbuf) !
call microp_driver_tend(use_subcol_microp, state_sc*splots, & state_sc, tend_sc, ztoldt, pfbuf, cmelka)

! Average the subcolumn ptend for use in gridded update
call subcol_ptend_avg(ptend_sc, state_sc, tend_sc, pfbuf) !
call physics_update(state_sc, tend_sc, ztoldt, tend_sc) !
call check_energy_chng(state_sc, tend_sc, "microp_tend", nstep, & ztoldt, zero_sc, pre_str, snow_str, zero_sc) !
call physics_state_dealloc(state_sc)
call physics_tend_dealloc(tend_sc)
call microp_tend_dealloc(ptend_sc)
else
  call microp_driver_tend(use_subcol_microp, state_sc*splots, & state, tend_sc, ztoldt, pfbuf, cmelka)
end if

call physics_update(state, tend_sc, ztoldt, tend) !
call check_energy_chng(state, tend, "microp_tend", nstep, ztoldt, zero, & pre_str, snow_str, zero) !

Code snippets from micro_mg_cam
(right colorized, ( indicates function description at right)

if (use_subcol_microp) then

! Get pfbuf fields which are not on subcolumns and ! copy them to columns
  call pfbuf_get_field(pfbuf, naai_idx, temp_fld2dr) !
call subcol_field_copy(temp_fld2dr, state, naai) !
else
  call pfbuf_get_field(pfbuf, naai_idx, naai)
end if

! Retrieve fields which may or may not be on subcolumns
! dependent on whether using subcolumns
  call pfbuf_get_field(pfbuf, pre_str, pre_str, col_type=col_type)
call micro_mg_tend
  (Note that this routine does NOT need any modifications. The internal loop over neols will automatically loop over grids or subcolumns)
if (use_subcol_microp) then

! Output root field (subcolumns output as a separate dimension)
call subcol_outfld(ORAIN, SC, qroot, pgrid_pcols, lchknk, state) !
! Average the subcolumns back to the grid column
call subcol_avg(ORAIN, temp_2dr, pfbuf) !
call outfld(ORAIN, temp_2dr, pgrid_pcols, lchknk)
else
call outfld(ORAIN, qroot, pcols, lchknk)
end if

Microphysics Buffer (pfbuf) changes
The physics buffer holds fields for use by the physics package.

pfbuf structure:
(buffer_field_type):: bgf – holds grid data
(buffer_field_type):: bgf_sc – holds subcolumn data – NEW

New Parameters to Identify grid/subcolumns
col_type: int – 0=grid, 1=subcolumns
grid_type: int(bifield_kind) – each bit is turned on/off to indicate which field(s) are required

pfbuf_add_field: Adds grid and/or subcolumn fields to pfbuf based on optional grid_types_flag – If not present adds grid field only; If present adds field(s) specified by grid_types_flag

pfbuf_set_field: Sets grid and/or subcolumn fields in pfbuf based on optional grid_types_flag – If not present sets grid field only; If present sets the field(s) specified by the grid_types_flag

pfbuf_get_field: Gets grid or subcolumn field based on optional col_type_flag – If not present then grid field is returned.

grid_flag_utils.F90: new module which contains routines to manipulate grid_type variables

Subcolumn scheme API
Different methods for generating and averaging subcolumn fields are called subcolumn schemes and are designated with the ‘subcol_scheme’ namelist variable.
The subcolumn scheme API is still undergoing development. The interfaces below are subject to change.

◆ Public interface functions which implement a subcolumn scheme
  ➢ subcol_register: Read the scheme name from namelist, initialize any scheme-global variables, and add scheme fields as necessary (typically using pbuf_add_field).
  ➢ subcol_init: Initialize any variables or fields specific to the active scheme (typically by calling pbuf_set_field).
  ➢ subcol_gen: Generate subcol fields from grid fields. Create copies of the physics state and the physics tend structures with subcolumns.
  ➢ subcol_field_avg: Average a field defined on subcolumns back into an averaged field for the grid cell. Averaging may be weighted based on scheme variables.
  ➢ subcol_ptend_avg: Average a physics tendency (ptend) defined on subcolumns to an averaged ptend for the grid cell.
  ➢ subcol_outfld: Unpack a subcolumn field before calling outfld.

◆ Public interface functions which do not depend on the subcolumn scheme
  ➢ subcol_field_copy: Copy a grid-average field into one defined on subcolumns. This function is used so that parameterizations can operate on a set of fields using the same equations whether or not they are using subcolumns.
  ➢ subcol_ptend_copy: Copy a physics_ptend object into one with subcolumns. This function is used so that physics_update can update the physics state and tendencies using the same equations whether or not they are using subcolumns.
  ➢ subcol_register_flags: Generate flags for use with pbuf_add_field, pbuf_get_field, and pbuf_set_field.

◆ Other physics package functions using subcolumns
  NB: These functions work on structures which all have subcolumns or which all have no subcolumns. Physics parameterizations using these functions do not have to make any changes.
  ➢ physics_update: Update the state and the dynamics tendency with the parameterization tendencies.
  ➢ check_energy_chng: Check that the energy and water change matches the boundary fluxes.

Summary
A subcolumn infrastructure has been developed for use by CAM physics parameterizations. This infrastructure:
  o is flexible and extensible and supports a variable number of subcolumns
  o introduces standard methods for building and working with subcolumns
  o dynamically allocates field storage to optimize memory use
  o can be used to support either sub-grid scale simulation or column ensembles
  o does not require any code changes to existing parameterizations
  o allows parameterizations to use the same subcolumn field data
  o supports multiple ‘schemes’ for subcolumn generation and data averaging
  o supports output of subcolumn data to CAM history files

While development is currently underway to implement subcolumn usage in CAM microphysics, the infrastructure has been committed to the CAM development trunk as of cam5_2_09. Note that this infrastructure is not in CESM 1.2 (except for the dynamic allocation of state/end/ptend which was committed in cam5_2_09).

Acknowledgements
We gratefully acknowledge useful discussions with William Collins, Steve Ghan, Phil Rasch, Vincent Larson, Cecile Hannay, Kate Thayer-Calder, Charles Bardeen, Chris Fischer, Joe McNerney, Sean Santos, John Truesdale, Jim Edwards, Mariana Vertenstein, and Francis Vitt.

This work has been supported by the National Science Foundation and by the US DOE Advanced Scientific Computing Research and Biological and Environmental Research Programs.