CCPP Training

College Park, MD, March 12-13, 2019

Host-side coding

Dom Heinzeller

Global Model Test Bed

Outline of Talk

- When are changes required on the host model side
- Register an existing variable with CCPP
- Adding new variables for running physics
- Output new variables for diagnostics
- Output new variables for restarts
- Wrap up

Host-side coding – scenarios

- In many cases, physics developers do not need to change the host model code (exception: CCPP prebuild config)
- The following scenarios require making host-model changes:
  - An existing variable on the host model side is not yet exposed to the CCPP, i.e. there is no metadata for it
  - A new variable is required for physics computations
  - A new or existing variable must be added to the model output (for diagnostics or for restarts)

Adding an existing variable to CCPP

Case 1: the existing variable is a standard Fortran variable, not a member of a derived data type (constants, flags, …)

- locate the module in which the variable is defined
- add variable to module metadata table if existent
- if the module doesn't have a metadata table yet:
  - create metadata table from scratch
  - add Fortran file to CCPP prebuild config, option `VARIABLE_DEFINITION_FILES`
  - make sure that the module is compiled before the host model cap (e.g., `IPD_CCPP_driver.F90` for slow physics), and that the compiler flags have the necessary include statements

Adding an existing variable to CCPP

Case 1: the existing variable is a standard Fortran variable, not a member of a derived data type (constants, flags, …)

```
module GFS_typedefs
   ...
!> \section arg_table_GFS_typedefs
!! | local_name | standard_name   | long_name        ...
!! |------------|-----------------|-----------------...
!! | ...        | ...             | ...
!! | LTP        | extra_top_layer | extra layer rad ...
!!
   ...
   integer :: LTP
   ...
contains
   ...
```

Adding an existing variable to CCPP

Case 2: the existing variable is a member of a derived data type, (DDT; e.g. `GFS_Data(:)%Sfcprop%oro`), known to CCPP

- locate the module in which the DDT is defined
- add variable to the DDT's metadata table

```
contains
   ...
!! \section arg_table_GFS_sfcprop_type
!! | local_name                         | standard_name ...
!! |------------------------------------|-----------------
!! | ...                                | ...
!! | GFS_Data(cdata%blk_no)%Sfcprop%oro | orography     ...
!!
   type GFS_sfcprop_type
      ...
      real(kind=kind_phys), pointer :: oro(:) => null()
```

Developmental Testbed Center

Adding an existing variable to CCPP

Case 3: the existing variable is a member of a derived data type, which is not yet known to CCPP

- add metadata table for DDT, follow case 2 instructions above
- locate the module that holds the memory for the DDT
- add the DDT to this module's metadata table, do as for case 1

```
module CCPP_data
!! \section arg_table_CCPP_data Argument Table
!! | local_name                | standard_name                  ...
!! |---------------------------|-------------------------------
!! | ...                       | ...                            ...
!! | GFS_Control               | GFS_control_instance           ...
!! | GFS_Data(cdata%blk_no)    | GFS_data_instance              ...
!!
contains
```

Developmental Testbed Center

Adding a variable to the host+CCPP

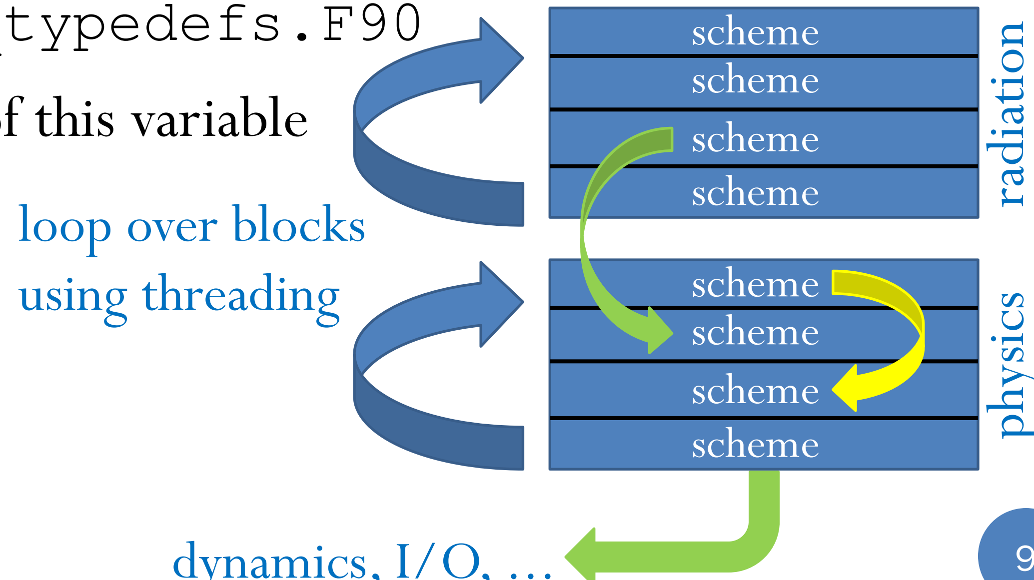
Case 1: the new variable is a member of a derived data type that already exists on the host model side – this is the easiest case.

- most likely this will be in `GFS_typedefs.F90`
- add constituent array to type definition, allocate and initialize
- reset if applicable (diagnostic or interstitial variables), follow existing code in `GFS_typedefs.F90`

Adding a variable to the host+CCPP

Case 1: the new variable is a member of a derived data type that already exists on the host model side – this is the easiest case.

- most likely this will be in `GFS_typedefs.F90`
- add constituent array to type definition, allocate and initialize
- reset if applicable (diagnostic or interstitial variables), follow existing code in `GFS_typedefs.F90`
- key question: purpose of this variable
  - interstitial variable
  - persistent variable (also: restart and diagnostic variable)



Adding a variable to the host+CCPP

Case 1: the new variable is a member of a derived data type that already exists on the host model side – this is the easiest case.

- most likely this will be in `GFS_typedefs.F90`
- add constituent array to type definition, allocate and initialize
- reset if applicable (diagnostic or interstitial variables), follow existing code in `GFS_typedefs.F90`
- key question: purpose of this variable
  - interstitial variable: use thread-dependent GFS_interstitial DDT
  - persistent variable: use other GFS DDTs (Diag, Sfcprop, Tbd, …)

Adding a variable to the host+CCPP

Case 2: the new variable is a member of a derived data type that doesn't exist yet in the host model, or the new variable is a derived data type itself that doesn't exist yet in the host model

- most developers will not encounter this situation
  - except if the new variable is a DDT that should become a member of an existing DDT (e.g. `sfcflw_type` in `GFS_radtend_type`)
  - in this case, follow the previous instructions to add a member to a DDT and to add metadata for a new DDT
  - other scenarios are not covered here (most complicated cases), contact GMTB and NEMSfv3gfs developers if this is really needed

Adding a diagnostic variable

- best to use the `GFS_Intdiag_type` in `GFS_typedefs.F90`
- other persistent DDTs will work as well
- follow above instructions for adding a new variable to a DDT
- add code to `GFS_diagnostics.F90` for outputting the data (use an existing entry closest to your needs), for example:

```
idx = idx + 1
ExtDiag(idx)%axes = 2
ExtDiag(idx)%name = 'maxmf'
ExtDiag(idx)%desc = 'maximum mass-flux in column'
ExtDiag(idx)%unit = 'm s-1'
ExtDiag(idx)%mod_name = 'gfs_sfc'
allocate (ExtDiag(idx)%data(nblks))
do nb = 1,nblks
  ExtDiag(idx)%data(nb)%var2 => IntDiag(nb)%maxmf(:)
enddo
```

Adding a diagnostic variable

- best to use the `GFS_Intdiag_type` in `GFS_typedefs.F90`
- other persistent DDTs will work as well
- follow above instructions for adding a new variable to a DDT
- add code to `GFS_diagnostics.F90` for outputting the data (use an existing entry closest to your needs)
- this outputs the data when quilting is on, if you want that or not
- without quilting, output can be controlled using `diag_table`
- workaround: use if-my-scheme-is-on in `GFS_typedefs.F90`

```
if (Model%do_mynnedmf) then
   ! output maxmf
end if
```

Adding a restart variable

- more complicated than adding a diagnostic variable, because there are several ways to do that
  - add to `GFS_restart_type` in `GFS_restart.F90`
  - modify code in `FV3GFS_io.F90`
  - both require adjusting indices and dimensions, possibly more
- without understanding the full picture, it seems to me that certain variables go into `FV3GFS_io.F90` (surface properties, phy_var2, phy_var3), while others go in `GFS_restart.F90`
- contact FV3 developers if this is required (some may be sitting in the audience?)

Wrap up

- adding new variables can range anywhere between easy and highly complicated, depending on the situation
- in most cases it is straightforward for physics developers
- outputting diagnostic variables is an easy task, too
- adding restart variables is more complicated

At this point, we have covered everything you need to know to become a seasoned NEMSfv3gfs+CCPP developer!