IPD
Interoperable Physics Driver

NEMSfv3gfs Forecast System Training and Tutorial
12-14 June, 2018

IPD

Designed to be lightweight and simple

Works with different physics packages

Name is a bit of a misnomer

Not really a driver, but an aliasing layer

- radiation and physics aliased to generic “steps”
- data grouped into containers based on purpose

Self-describing data for I/O-related elements

- diagnostic data
- restart data

GFS physics library

GFS physics:

- GFS_typedefs.F90
- GFS_diagnostics.F90
- GFS_abstraction_layer.F90
- GFS_driver.F90
- GFS_restart.F90

GFS_abstraction_layer.F90

Defines a generically named module:

physics_abstraction_layer

use statement creates generically named routines via Fortran aliasing:

initialize	=>	GFS_initialize
time_vary_step	=>	GFS_time_vary_step
radiation_step1	=>	GFS_radiation_driver
physics_step1	=>	GFS_physics_driver
physics_step2	=>	GFS_stochastic_physics

GFS_abstraction_layer.F90

Same concept for the typedef containers:

Statein	=>	GFS_statein_type
Stateout	=>	GFS_stateout_type
Sfcprop	=>	GFS_sfcprop_type
Coupling	=>	GFS_coupling_type
Grid	=>	GFS_grid_type
Tbd	=>	GFS_tbd_type
CldProp	=>	GFS_cldprop_type
Radtend	=>	GFS_radtend_type
IntDiag	=>	GFS_diag_type

IPD

IPD_typdefs.F90

container definitions

IPD_driver.F90

interface to physics routines

IPD_typedefs.F90

Relies upon a set of standardized types provided by an abstraction layer defined within the physics

Standardized types are used to define various IPD types

- IPD_control_type
- IPD_diag_type
- IPD_restart_type
- IPD_data_type

Defines F90 abstract interface procedures

- IPD_func0d_proc
- IPD_func1d_proc

IPD_typedefs.F90

```
IPD_control_type   =>  control_type
IPD_diag_type      =>  diagnostic_type
IPD_restart_type   =>  restart_type

type IPD_data_type
   public
   type(statein_type)      :: Statein
   type(stateout_type)     :: Stateout
   type(sfcprop_type)      :: Sfcprop
   type(coupling_type)     :: Coupling
   type(grid_type)         :: Grid
   type(tbd_type)          :: Tbd
   type(cldprop_type)      :: Cldprop
   type(radtend_type)      :: Radtend
   type(intdiag_type)      :: Intdiag
 end type IPD_data_type
```

IPD_driver.F90

subroutine IPD_initialize (IPD_Control, IPD_Data, IPD_Diag, &
IPD_Restart, Init_parm)

```
call initialize (IPD_Control,        IPD_Data%Statein,   &
                 IPD_Data%Stateout,  IPD_Data%Sfcprop,   &
                 IPD_Data%Coupling,  IPD_Data%Grid,      &
                 IPD_Data%Tbd,       IPD_Data%Cldprop,   &
                 IPD_Data%Radtend,   IPD_Data%IntDiag,   &
                 IPD_Diag,      IPD_Restart,     Init_parm)
```

call diagnostic_populate (IPD_Diag, …)

call restart_populate (IPD_Restart, …)

IPD_driver.F90

subroutine IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart,
IPD_func0d, *IPD_func1D*)

```
call IPD_func0d  (IPD_Control,           IPD_Data%Statein, &
                  IPD_Data%Stateout,  IPD_Data%Sfcprop, &
                  IPD_Data%Coupling,  IPD_Data%Grid,    &
                  IPD_data%Tbd,       IPD_Data%Cldprop, &
                  IPD_Data%Radtend,   IPD_Data%IntDiag)

call IPD_func1d  (IPD_Control,              IPD_Data(:)%Statein, &
                  IPD_Data(:)%Stateout,  IPD_Data(:)%Sfcprop, &
                  IPD_Data(:)%Coupling,  IPD_Data(:)%Grid,    &
                  IPD_data(:)%Tbd,       IPD_Data(:)%Cldprop, &
                  IPD_Data(:)%Radtend,   IPD_Data(:)%IntDiag)
```

IPD_funcNd is defined via Fortran pointers by the calling routine

Using IPD

Begin Integration Loop

call atmos_dynamics ()

call populate_state (IPD_Data)

func1d => time_vary_step

call IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart, IPD_func1d=func1d)

func1d => radiation_step1

call IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart, IPD_func1d=func1d)

func1d => physics_step1

call IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart, IPD_func1d=func1d)

func1d => physics_step2

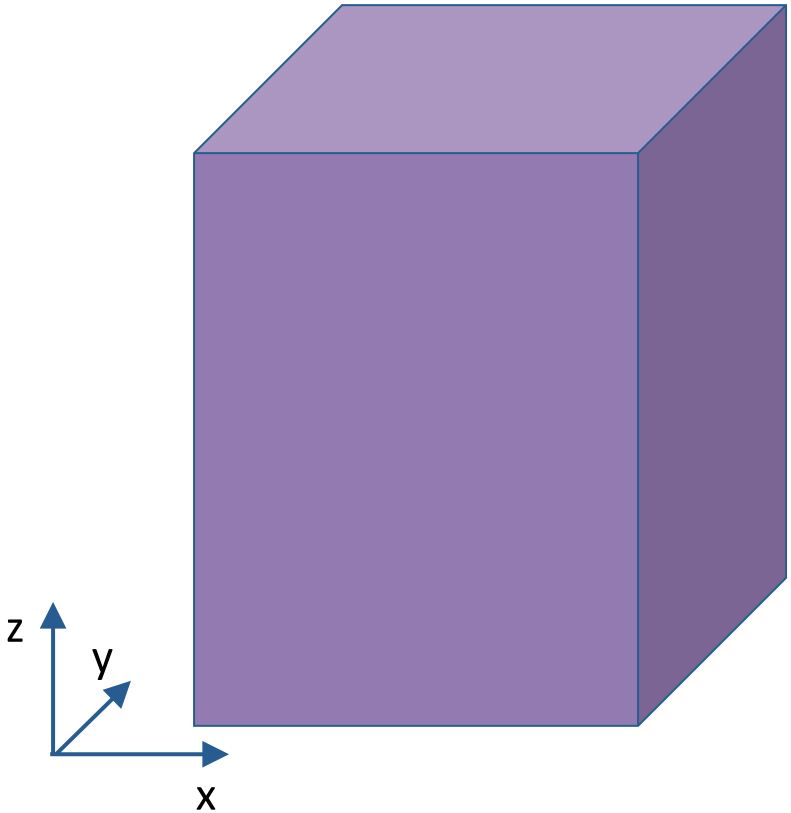
call IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart, IPD_func1d=func1d)

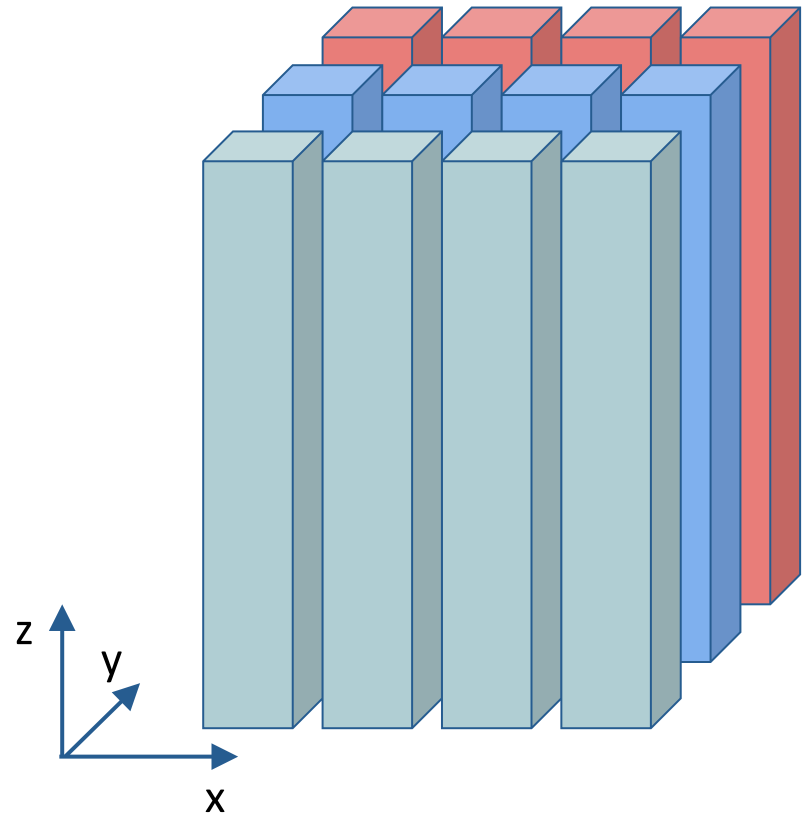
call update_prognostic_state (IPD_Data)

call output_diagnostics (IPD_Diag)

End Integration Loop

Blocking





Blocking

Defines containers to abstract the the specific variables needed by a given physics suite

```
type(IPD_control_type) ::                IPD_Control
type(IPD_data_type), allocatable ::      IPD_Data(:)
type(IPD_diag_type), allocatable ::      IPD_Diag(:)
type(IPD_restart_type) ::                IPD_Restart

allocate (IPD_Data(nblks))
```

Blocking → Threading

Serial

```
func1d => XXXX
call IPD_step (IPD_Control, IPD_Data, IPD_Diag, IPD_Restart, IPD_func1d=func1d)
```

Becomes

```
   func0d => XXXXX
!$OMP parallel do default (none) &
!$OMP        shared  (Atm_block, IPD_Control) &
!$OMP        shared  (IPD_Data, IPD_Diag, IPD_Restart) &
!$OMP        private (nb)
   do nb = 1,Atm_block%nblks
      call IPD_step (IPD_Control, IPD_Data(nb:nb), IPD_Diag, IPD_Restart, &
                     IPD_func0d=func0d)
   enddo
```

Dynamics-Physics Coupling

Hydrostatic vs. non-hydrostatic mismatch has implications on pressure, temperature, geometric heights, and their derivatives

Updating the post-physics dynamical state uses inverse procedure to that used prior to the physics

Documentation of the science or research assumptions for a given parameterization is key

Examples

Requires knowledge of physical parameterizations to populate IPD_Data%Statein in a consistent manner

Tracer mixing ratio depends on definition of total air mass

Total air mass – is it dry? does it include vapor? what about condensates?

total air mass (FV3 dycore) = dry + vapor + condensates

total air mass (GFS physics) = dry + vapor