MONAN
A estrutura de chamadas MPAS

Workshop Interno da DIMNT para início dos trabalhos com o MONAN-ATM/SFC de 2 a 3 de outubro de 2023.

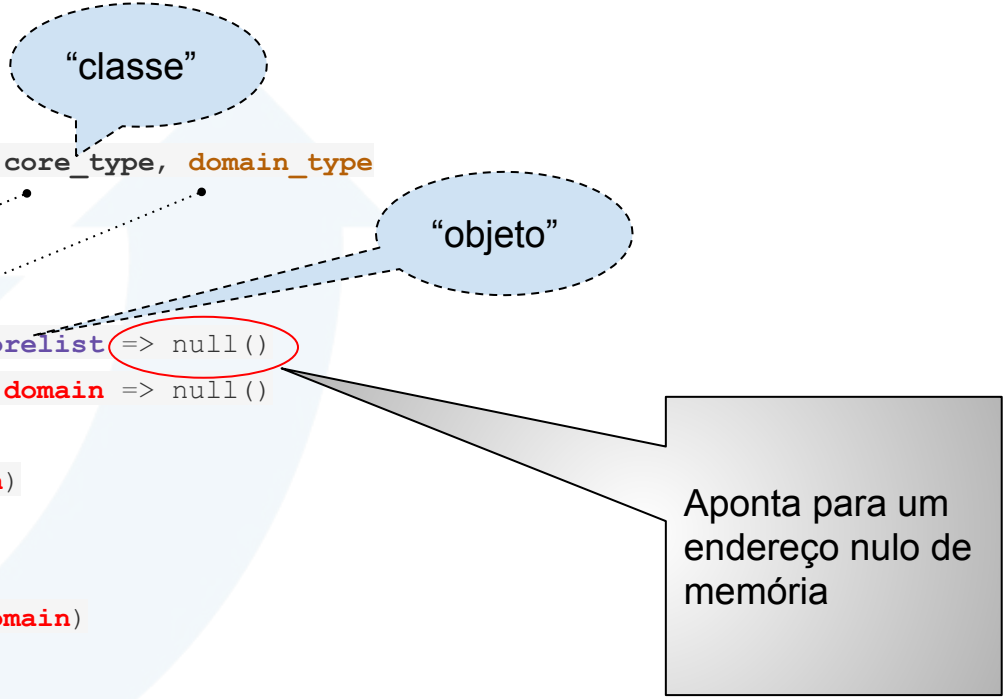
Luiz Flávio Rodrigues/Denis Eiras
CGCT/DIMNT/GCC
luiz.rodrigues@inpe.br/denis.eiras@inpe.br

0 Introdução

- O MPAS usa algo próximo de orientação de objetos - OO;
- Essa estrutura é nova para alguns e pode ser desafiador pensar na forma de OO;
- Uma classe tem propriedades (variáveis) e métodos (funções). Um objeto pode ser instanciado a essa classe e herdar seus atributos;
- Uma classe pode instanciar outra classe passando a ter internamente a mesma;
- Um objeto pode ser passado por chamada de funções e subrotinas;
- Iremos fazer uma pequena introdução, usando o próprio código, de como essa estrutura funciona;
- Não iremos abordar todo o problema. Apenas fazer uma introdução;
- Apontaremos onde entra a física e como acrescentar novos códigos;
- Apontaremos como colocar novas variáveis globais e como alocá-las;

1 Programa Principal

```
program mpas

  use mpas_subdriver
  use mpas_derived_types, only : core_type, domain_type

  implicit none

  type (core_type), pointer :: corelist => null()
  type (domain_type), pointer :: domain => null()

  call mpas_init(corelist, domain)
  call mpas_run(domain)

  call mpas_finalize(corelist, domain)

  stop
```



2 Tipos - O tipo core_type

Com lista encadeada

FRAMEWORK - **mpas_core_types.inc**

```
abstract interface
   function mpas_core_init_function(domain, timeStamp) result(iErr)
      import domain_type
      type (domain_type), intent(inout) :: domain
      character (len=*), intent(out) :: timeStamp
      integer :: iErr
   end function
end interface
…
type core_type
   type (domain_type), pointer :: domainlist => null()

   character (len=StrKIND) :: modelName !< Constant: Name of model
   character (len=StrKIND) :: coreName !< Constant: Name of core
   …

   ! Core init, run, and finalize function pointers
   procedure (mpas_core_init_function), pointer, nopass :: core_init => null()
   procedure (mpas_core_run_function), pointer, nopass :: core_run => null()
   procedure (mpas_core_finalize_function), pointer, nopass :: core_finalize => null()
   …

   ! core_type is a linked list
   type (core_type), pointer :: next => null()
end type core_type
```

A função ainda não existe. Apenas define a interface

"nopass" impede que o procedure (core_init) seja visto fora do "type"

Nome da Função

vou me lembrar disso!

MONAN Model for Ocean-laNd-Atmosphere prediction

INPE

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

3 Programa Principal
Invocando a inicialização

```
program mpas

  use mpas_subdriver
  use mpas_derived_types, only : core_type, domain_type

  implicit none

  type (core_type), pointer :: corelist => null()
  type (domain_type), pointer :: domain => null()

  call mpas_init(corelist, domain)

  call mpas_run(domain)

  call mpas_finalize(corelist, domain)

  stop
```

mpas_init
Inicializa as estruturas
(“objetos”)

4 Tipos

A finalização dos ponteiros para procedures

Driver - **mpas_subdriver.F**

```
subroutine mpas_init(corelist,domain_ptr,...
…
#ifdef CORE_ATMOSPHERE
     call atm_setup_core(corelist)
     call atm_setup_domain(domain_ptr)
#endif
```

```
abstract interface
      function mpas_core_init_function(domain, timeStamp) result(iErr)
```

core_atmosphere - **mpas_atm_core.F**

```
function atm_core_init(domain, startTimeStamp) result(ierr)
```

core_atmosphere - **atm_setup_core.F**

```
subroutine atm_setup_core(core)
…
type (core_type), pointer :: core
…
core % core_init => atm_core_init
core % core_run => atm_core_run
…
```

vou me lembrar disso!

FRAMEWORK - **mpas_domain_types.inc**

```
type domain_type
 …
    ! Back pointer to core
    type (core_type), pointer :: core => null()

    ! Domain_type is a linked list
    type (domain_type), pointer :: next => null()
 end type domain_type
```

domain contém core

MONAN Model for Ocean-land-Atmosphere prediction

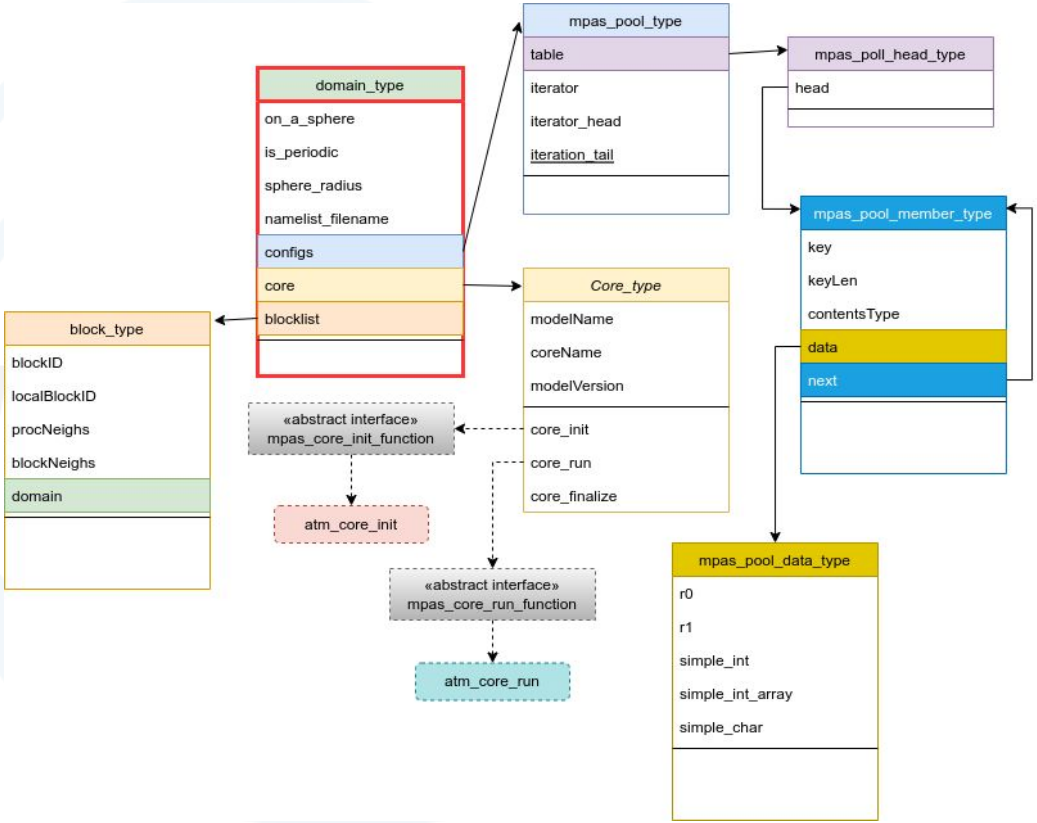
INPE

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Estrutura

Uma UML de parte dos tipos - MPAS



Observem que “domain_type” que vai criar o objeto “domain” carrega todas as estruturas (“classes”) e pode transportá-las por chamada.

Esse diagrama contém uma pequena parte de todos os tipos utilizados.

Programa Principal
Invocando o motor

```
program mpas

  use mpas_subdriver
  use mpas_derived_types, only : core_type, domain_type

  implicit none

  type (core_type), pointer :: corelist => null()
  type (domain_type), pointer :: domain => null()

  call mpas_init(corelist, domain)

  call mpas_run(domain)

  call mpas_finalize(corelist, domain)

  stop
```

Invoca passando apenas o tipo (“objeto”) inicializado

7 Ligando o motor

Invoca apenas a função passada no “objeto”

Driver - **mpas_subdriver.F**

```
subroutine mpas_run(domain_ptr)

    use mpas_log, only: mpas_log_info
    implicit none

    type (domain_type), intent(inout), pointer :: domain_ptr

    integer :: iErr

    if ( associated(domain_ptr % logInfo) ) mpas_log_info => domain_ptr % logInfo

    iErr = domain_ptr % core % core_run(domain_ptr)

end subroutine mpas_run
```

Veja que o tipo é usado na invocação. O tipo é passado como argumento e contém a função core_run que estava lá no tipo básico “core_type”

O objeto é carregado

Lembre-se que:

```
core % core_run => atm_core_run
```

8 Rodando a máquina
A função atm_core_run

core_atmosphere - **mpas_atm_core.F**

```
function atm_core_run(domain) result(ierr)
…
     type (domain_type), intent(inout) :: domain
…
     type (block_type), pointer :: block_ptr
     type (mpas_pool_type), pointer :: state, diag, mesh, diag_physics, tend, tend_physics

     do while (.not. mpas_is_clock_stop_time(
…
        currTime = mpas_get_clock_time(clock, MPAS_NOW, ierr)
…
        call mpas_log_write('Begin timestep '//trim(timeStamp))

#ifdef DO_PHYSICS
            call mpas_pool_get_subpool(block_ptr % structs, 'diag_physics', diag_physics)
…
```

O subdomínio de um "core" está dentro de um bloco

Dados estão em conjuntos com suas definições gerais

9 Rodando a máquina
Invocando o timestep (passando o domain)

core_atmosphere - **mpas_atm_core.F**

```
        …
       call mpas_timer_start("time integration")
        call mpas_dmpar_get_time(integ_start_time)
        call atm_do_timestep(domain, dt, itimestep)
        call mpas_dmpar_get_time(integ_stop_time)
        call mpas_timer_stop("time integration")
        call mpas_log_write(' Timing for integration step: $r s', realArgs=(/real(integ_stop_time -
integ_start_time, kind=RKIND)/))
…
```

Chama o timestep para a física

10 Fazendo um timestep
Invocando o driver geral da física e a dinâmica

core_atmosphere - **mpas_atm_core.F**

```
subroutine atm_do_timestep(domain, dt, itimestep)
    …
#ifdef DO_PHYSICS
    use mpas_atmphys_driver
#endif
    type (domain_type), intent(inout) :: domain
    real (kind=RKIND), intent(in) :: dt
   …
#ifdef DO_PHYSICS
    !proceed with physics if moist_physics is set to true:
    if(moist_physics) then
       call physics_timetracker(domain,dt,clock,itimestep,xtime_s)
       call physics_driver(domain,itimestep,xtime_s)
    endif
#endif

call atm_timestep(domain, dt, currTime, itimestep, exchange_halo_group)
```

Chama o driver "macro"

sempre carregando consigo o "domain"

Invocando a Física
Obtendo os configurações definidas

mpas_pool_get_config faz a busca em **domain%configs** pelo **esquema escolhido** e retorna no **config_xxx_scheme**

core_atmosphere/physics - **mpas_atmphys_driver.F**

```
…
call mpas_pool_get_config(domain%configs,'config_pbl_scheme'       ,config_pbl_scheme       )
call mpas_pool_get_config(domain%configs,'config_radt_lw_scheme'   ,config_radt_lw_scheme   )
call mpas_pool_get_config(domain%configs,'config_radt_sw_scheme'   ,config_radt_sw_scheme   )
…
```

Obter as configurações dos conjuntos (pool) sempre passam pela subrotina de busca - Essa estrutura é usada múltiplas vezes dentro do código

Invocando a Física

Driver geral chama driver específico

core_atmosphere/physics - **mpas_atmphys_driver.F**

```
   !call to long wave radiation scheme:
   if(l_radtlw) then
      time_lev = 1
      call allocate_radiation_lw(block%configs,xtime_s)
!$OMP PARALLEL DO
      do thread=1,nThreads
         call driver_radiation_lw(xtime_s,block%configs,mesh,state,time_lev,diag_physics, &
                                  atm_input,sfc_input,tend_physics, &
                                  cellSolveThreadStart(thread), cellSolveThreadEnd(thread))
      end do
!$OMP END PARALLEL DO
   endif
```

Observe que são resolvidos em paralelo com grupos de células em threads

Driver da Parametrização física
Chama a parametrização escolhida

core_atmosphere - **mpas_atmphys_driver_radiation_lw.F**

```
use mpas_atmphys_vars
…
subroutine driver_radiation_lw(xtime_s,configs,mesh,state,time_lev,diag_physics,atm_input, &
                               sfc_input,tend_physics,its,ite)
…
call mpas_pool_get_config(configs,'config_radt_lw_scheme',radt_lw_scheme      )

copy MPAS arrays to local arrays:
call radiation_lw_from_MPAS(xtime_s,configs,mesh,state,time_lev,diag_physics,atm_input,sfc_input,its,ite)

!call to longwave radiation scheme:
radiation_lw_select: select case (trim(radt_lw_scheme))
      case ("rrtmg_lw")
      …
      case ("cam_lw")
      …
```

Driver da Parametrização física
Chama a parametrização escolhida

core_atmosphere - **mpas_atmphys_driver_radiation_lw.F**

```
call rrtmg_lwrad( &
          p3d        = pres_hyd_p     , p8w        = pres2_hyd_p , pi3d     = pi_p     , &
          t3d        = t_p            , t8w        = t2_p        , dz8w     = dz_p     , &
!          qv3d       = qv_p           , qc3d       = qc_p        , qi3d     = qi_p     , &
!          qs3d       = qs_p           , cldfra3d   = cldfrac_p   , tsk      = tsk_p    , &
          qv3d       = qvrad_p        , qc3d       = qcrad_p     , qi3d     = qirad_p  , &
          qs3d       = qsrad_p        , cldfra3d   = cldfrac_p   , tsk      = tsk_p    , &
          emiss      = sfc_emiss_p    , xland      = xland_p     , xice     = xice_p   , &
          snow       = snow_p         , icloud     = icloud      , o3input  = o3input  , &
          noznlevels = num_oznlevels  , pin        = pin_p       , o3clim   = o3clim_p , &
          glw        = glw_p          , olr        = olrtoa_p    , lwcf     = lwcf_p   , &
          rthratenlw = rthratenlw_p   , has_reqc   = has_reqc    , has_reqi = has_reqi ,
...
```

Nos próximos slide trataremos dessas variáveis

Memória das variáveis físicas

As variáveis físicas GLOBAIS! Onde estão?

core_atmosphere/physics - mpas_atmphys_vars.F

```
module mpas_atmphys_vars
use mpas_kind_types
…
real(kind=RKIND),dimension(:,:,:),allocatable:: &
   pres_hyd_p,        &!pressure located at theta levels                    [Pa]
   pres_hydd_p,       &!"dry" pressure located at theta levels              [Pa]
   pres2_hyd_p,       &!pressure located at w-velocity levels               [Pa]
   pres2_hydd_p,      &!"dry" pressure located at w-velocity levels         [Pa]
   znu_hyd_p           !(pres_hyd_p / P0) needed in the Tiedtke convection scheme [Pa]

real(kind=RKIND),dimension(:,:,:),allocatable:: &
   o3clim_p            !climatological ozone volume mixing ratio          [???]
```

A alocação não é feita no próprio módulo!

Ela depende da estrutura!!!!

Memória das variáveis físicas

As variáveis físicas globais! Onde são alocadas?

core_atmosphere/physics - mpas_atmphys_interface.F

```
module mpas_atmphys_interface
…
use mpas_atmphys_vars
…
subroutine allocate_forall_physics(configs)
…
!... arrays used for calculating the hydrostatic pressure and exner function:
if(.not.allocated(psfc_hyd_p)   ) allocate(psfc_hyd_p(ims:ime,jms:jme)          )
if(.not.allocated(psfc_hydd_p) ) allocate(psfc_hydd_p(ims:ime,jms:jme)          )
if(.not.allocated(pres_hyd_p)   ) allocate(pres_hyd_p(ims:ime,kms:kme,jms:jme)  )
if(.not.allocated(pres_hydd_p) ) allocate(pres_hydd_p(ims:ime,kms:kme,jms:jme) )
if(.not.allocated(pres2_hyd_p) ) allocate(pres2_hyd_p(ims:ime,kms:kme,jms:jme) )
if(.not.allocated(pres2_hydd_p)) allocate(pres2_hydd_p(ims:ime,kms:kme,jms:jme))
if(.not.allocated(znu_hyd_p)    ) allocate(znu_hyd_p(ims:ime,kms:kme,jms:jme)   )
```

Memória das variáveis físicas

As variáveis específicas! Onde são alocadas?

core_atmosphere - **mpas_atmphys_driver_radiation_lw.F**

```
subroutine allocate_radiation_lw(configs,xtime_s)
…
if(.not.allocated(xice_p)        ) allocate(xice_p(ims:ime,jms:jme)                 )
if(.not.allocated(xland_p)       ) allocate(xland_p(ims:ime,jms:jme)                )
…
radiation_lw_select: select case (trim(radt_lw_scheme))

   case("rrtmg_lw")
      …
      if(.not.allocated(o3clim_p)      ) allocate(o3clim_p(ims:ime,1:num_oznlevels,jms:jme))
```

Observe que somente na primeira chamada os valores serão alocados.

Obrigado

CGCT/DIMNT/GCC