Fortran Module Template

Weishuo Liu[a]

[a]School of Energy and Power Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, 100191, China

The module template to integrate the predictor into a Fortran CFD program can be found in the public repository. The design of this module allows it to be used as an absolute external plug-in component, which means the module maintains all the user-defined variables (e.g., predictor instances, additional field) by itself, and only interacts with the program's originally allocated arrays. In this way, no extra memory is allocated in the main program, and the risk of memory leak caused by mutual contamination between the legacy-code-created global variables (normally in F77 format) and newly created objects can be minimized to the greatest extent.

The structural template can be found in Listing. 2. Basically, all the managements of the ML-associated variables (e.g., predictor objects and data arrays) are governed by the module itself. whereas the subroutine for updating field is in charge of interacting with the main program. Users only need to define their needed arrays in the definition of pd_pack, the initialization/finalization functions for such arrays, and the update-field subroutine by themselves.

The read-in text file for predictor settings is also shown below as a reference (Listing 1), the initialization and the format of this setting file also can be modified according to the user's demand.

```
# Whether it is from PB (0) or from SavedModel (1) format? !n_ml_options;
0
# This is model dir
ML_inputs/output_graph.pb
# This is the tag string for SavedModel(1) format, will not be used if n_ml_options = 0
serve
# This is input node name
Lamda_in/Placeholder
# This is output node name
layer_out/Wx_plus_b/Add
```

Listing 1: Setting file for Fortran predictor module

```
module ML_module_selfdef
use iso_c_binding, only: c_ptr, c_char
implicit none

type :: pd_pack
  type(c_ptr) :: pd_ml
  character(kind=c_char, len=:), allocatable :: in_node_name, out_node_name

  ! Status Vars: 1 for created, 0 for not created
  integer :: n_pd_created = 0, n_arr_created = 0

  ! User Modified Part: -------------------------------------------------------------
  ! ML Arrays (same shape for the input and output nodes)
  ! Assumed a basic NN, which is 2D input (n * 6) and 2D output (n * 2),
  ! the dim of containers should be (6, n) and (2, n) because of Fortran's ColMajor
  real(8),allocatable,dimension(:,:) :: ml_input_container, ml_output_container
  !
  ! arrays for calculating input/output
  real(8),allocatable,dimension(:,:,:) :: SomeArray1,  SomeArray2,  SomeArray3
  ! End of Modified Part: -----------------------------------------------------------
end type pd_pack
```

Email address: liuweishuo@buaa.edu.cn (Weishuo Liu)

```fortran

! This module maintains an array of predictors for multi-block or grid sequencing usage.
! It is recommended to create a single predictor for a block or mesh sequence to avoid
! massive allocate-reallocate expenses.
type(pd_pack), allocatable, dimension(:) :: pd_pack_arr

contains

! Fixed Part (Basically no need to modify): -------------------------------------------
! Allocate total number of predictors
subroutine AllocatePredictors(npredictor)
  integer :: npredictor
  allocate(pd_pack_arr(npredictor))
end subroutine AllocatePredictors


! Initialize the i-th predictor from the setting file:
subroutine Init_ML_Predictor(i_pd, file_name)
  !
  use ML_Predictor
  use ISO_C_Binding, only: c_char

  implicit none
  integer, intent(in) :: i_pd
  character(kind=c_char, len=:), intent(in) :: file_name

  ! Vars to store setting and read-in text
  integer :: n_ml_options = 99
  character(kind=C_char, len=256) :: buffer_string
  character(kind=C_char, len=:), allocatable :: file_name, tags_name

  if ( pd_pack_arr(i_pd)%n_pd_created /= 0) then
    print*, 'Predictor has already been created'
    return
  end if

  open(unit=1220, file=file_name,status="OLD",action="READ")
  read(1220,*)
  read(1220,*)  n_ml_options

  ! Read settings from file
  read(1220,*)
  read(1220,'(a256)')  buffer_string
  print*, "buffer_string is: ", trim(buffer_string)
  file_name = trim(buffer_string)

  read(1220,*)
  read(1220,'(a256)')  buffer_string
  tags_name = trim(buffer_string)

  read(1220,*)
  read(1220,'(a256)')  buffer_string
  pd_pack_arr(i_pd)%in_node_name = trim(buffer_string)

  read(1220,*)
  read(1220,'(a256)')  buffer_string
  pd_pack_arr(i_pd)%out_node_name = trim(buffer_string)

  close(unit=1220)

  ! Initialize the predictor according to the setttings

  if ( n_ml_options == 0) then
      print*, "Reading models from PB graph:"
      pd_pack_arr(i_pd)%pd_ml = C_CreatePredictor(file_name)
      call C_PredictorRegisterInputNode(pd_pack_arr(i_pd)%pd_ml, &
                                        pd_pack_arr(i_pd)%in_node_name)
      call C_PredictorRegisterOutputNode(pd_pack_arr(i_pd)%pd_ml, &
                                         pd_pack_arr(i_pd)%out_node_name)
      print*, "ML predictor created."

  else if (n_ml_options == 1) then
      print*, "Reading models from  SavedModel format:"
      pd_pack_arr(i_pd)%pd_ml = C_CreatePredictor(file_name, tags_name)
      call C_PredictorRegisterInputNode(pd_pack_arr(i_pd)%pd_ml,  &
                                        pd_pack_arr(i_pd)%in_node_name)
      call C_PredictorRegisterOutputNode(pd_pack_arr(i_pd)%pd_ml, &
                                         pd_pack_arr(i_pd)%out_node_name)
      print*, "ML predictor created."

  else
      print*, "unsupported n_ml_options, the value is: ", n_ml_options
      stop
  end if

  pd_pack_arr(i_pd)%n_pd_created = 1

end subroutine Init_ML_Predictor

! Delete the i-th predictor before program ends:
subroutine Finalize_ML_Predictor(i_pd)
  use ML_Predictor
  ! use iso_c_binding, only: c_ptr
  implicit none
  integer, intent(in) :: i_pd
  if ( pd_pack_arr(i_pd)%n_pd_created == 0) then
      print*, 'Predictor has not been created, no need to finalize'
      return
  end if
  call C_DeletePredictor(pd_pack_arr(i_pd)%pd_ml)
end subroutine Finalize_ML_Predictor
! End of Fixed Part: -----------------------------------------------------------------

! User Modified Part: ----------------------------------------------------------------

! Allocate Arrays in the i-th predictor:
subroutine Allocate_ML_Arrays(i_pd, jdim, kdim, idim)
  use ML_Predictor
  ! use iso_c_binding, only: c_ptr
  implicit none
  integer, intent(in) :: i_pd
  integer, intent(in) :: jdim, kdim, idim
  ! please modify it according to your need

end subroutine Allocate_ML_Arrays

! Delete Arrays in the i-th predictor:
subroutine Deallocate_ML_Arrays(i_pd)
  use ML_Predictor
  ! use iso_c_binding, only: c_ptr
  implicit none
  integer, intent(in) :: i_pd
  ! Delete Arrays, please modify it according to your need

end subroutine Deallocate_ML_Arrays

! Use i-th predictor to call ML prediction
subroutine updateSomeField(i_pd, jdim, kdim, idim, ..., arr_in_1, ..., arr_out_1, ...)
  use ML_Predictor
  ! use iso_c_binding, only: c_ptr
  implicit none
  integer, intent(in) :: i_pd
  integer, intent(in) :: jdim, kdim, idim

  !array references from the main program
  real, intent(in) :: arr_in_1(jdim,kdim,idim), arr_in_2(jdim,kdim,idim), ...

  !array references to be modified in the main program
  real, intent(in) :: arr_out_1(jdim,kdim,idim), arr_out_2(jdim,kdim,idim), ...

  ! Some code to assemble input data from arr_in_1, arr_in_2, ...
  ! ...

  ! run the prediction:
  call C_PredictorSetNodeData(pd_pack_arr(i_pd)%pd_ml, &
                              pd_pack_arr(i_pd)%in_node_name, &
                              pd_pack_arr(i_pd)%ml_input_container, &
                              size(pd_pack_arr(i_pd)%ml_input_container))

  call C_PredictorRun(pd_pack_arr(i_pd)%pd_ml)

  call C_PredictorGetNodeData(pd_pack_arr(i_pd)%pd_ml,&
                              pd_pack_arr(i_pd)%out_node_name, &
                              pd_pack_arr(i_pd)%ml_output_container, &
                              size(pd_pack_arr(i_pd)%ml_output_container))

  ! Some code calculate arr_out_1, arr_out_2 ...
  ! ...

end subroutine updateSomeField
! End of Modified Part: -----------------------------------------------------------

end module ML_module_selfdef
```

Listing 2: Minimal example to run A + B model in Fortran