

サンプルコードの解説

行方大輔

(理化学研究所 計算科学研究センター
粒子系シミュレータ研究チーム)

サンプルコード (1)

- 付属するサンプルコード
 - 重力 N 体計算コード
 - 流体計算コード (Smoothed Particle Hydrodynamics[SPH])
 - 重力 N 体/SPHコード
 - P^3M コード
- 本スライドでは、重力 N 体計算コードを取り上げて解説する。
 - 計算するのはcold collapse問題.
 - 初期条件はその場で生成 (ファイル読み込みではない).
 - 時間積分法はleap-frog法.
 - ファイル構成 (中身は、後で詳しく説明)
 - user_defined.F90
 - f_main.F90
 - Makefile

サンプルコード (2)

- ユーザが書くべきもの

- FDPS指示文付きの粒子クラス
 - 相互作用関数
 - 初期条件生成ルーチン
 - 時間積分ルーチン
 - I/Oルーチン
- } user_defined.F90
- } f_main.F90

- ユーザがすべきこと

- 付属のPythonスクリプトを使用して、Fortran インターフェイスを生成

↪ サンプルコード付属のMakefileでは、これを自動で行う。

user_defined.F90

■ 粒子クラスの定義

```
module user_defined_types
```

```
① use, intrinsic :: iso_c_binding
```

```
② use fdps_vector  
use fdps_super_particle  
implicit none
```

```
③
```

```
!**** Full particle type  
type, public, bind(c) :: full_particle !$fdps FP,EPI,EPJ,Force  
!$fdps copyFromForce full_particle (pot,pot) (acc,acc)  
!$fdps copyFromFP full_particle (id,id) (mass,mass) (eps,eps) (pos,pos)  
!$fdps clear id=keep, mass=keep, eps=keep, pos=keep, vel=keep  
integer(kind=c_long_long) :: id  
real(kind=c_double) mass !$fdps charge  
real(kind=c_double) :: eps  
type(fdps_f64vec) :: pos !$fdps position  
type(fdps_f64vec) :: vel !$fdps velocity  
real(kind=c_double) :: pot  
type(fdps_f64vec) :: acc  
end type full_particle
```

① C言語との相互運用に必要なモジュール (Fortran 2003から導入)

② ベクトル型と超粒子型が定義されたモジュール (FDPSから提供)
これらは、粒子クラスと相互作用関数の定義に必要.

③ 粒子クラスの定義

- C言語と相互運用の必要性から、(i) bind(c)属性、(ii) C言語と互換性のあるデータ型の使用、が必要.
- FDPS指示文を使って、構造体やメンバ変数が何を表す物理量かを指定する必要がある. また、ユーザ定義型の間のデータコピーの方法や、データを初期化する方法も指示する必要がある.

■ 相互作用関数の定義

```
!**** Interaction function (particle-particle) ①
subroutine calc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
integer(c_int), intent(in), value :: n_ip,n_jp
type(full_particle), dimension(n_ip), intent(in) :: ep_i
type(full_particle), dimension(n_jp), intent(in) :: ep_j
type(full_particle), dimension(n_ip), intent(inout) :: f
!* Local variables
integer(c_int) :: i,j
real(c_double) :: eps2,poti,r3_inv,r_inv
type(fdps_f64vec) :: xi,ai,rij

!* Compute force
do i=1,n_ip
  eps2 = ep_i(i)%eps * ep_i(i)%eps
  xi = ep_i(i)%pos
  ai = 0.0d0
  poti = 0.0d0
  do j=1,n_jp
    rij%x = xi%x - ep_j(j)%pos%x
    rij%y = xi%y - ep_j(j)%pos%y
    rij%z = xi%z - ep_j(j)%pos%z
    r3_inv = rij%x*rij%x &
             + rij%y*rij%y &
             + rij%z*rij%z &
             + eps2
    r_inv = 1.0d0/sqrt(r3_inv)
    r3_inv = r_inv * r_inv
    r_inv = r_inv * ep_j(j)%mass
    r3_inv = r3_inv * r_inv
    ai%x = ai%x - r3_inv * rij%x
    ai%y = ai%y - r3_inv * rij%y
    ai%z = ai%z - r3_inv * rij%z
    poti = poti - r_inv
  end do
  f(i)%pot = f(i)%pot + poti
  f(i)%acc = f(i)%acc + ai
end do
end subroutine calc_gravity_pp
```

① bind(c)属性が必要.

② 粒子数に対応する引数にはvalue属性が必要.

➤ これは、値渡しを指示するキーワードで、(FDPSで定義された)相互作用関数の仕様に対応させるため必要となる.

③ 相互作用の具体的な中身を実装.

➤ 今回は、重力計算なので逆2乗則の計算を行っている.

➤ 最も内側ループでは最適化の観点から、構造体の成分を直接使用して計算.

```

!**** Interaction function (particle-super particle)
subroutine calc_gravity_psp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
  integer(c_int), intent(in), value :: n_ip,n_jp
  type(full_particle), dimension(n_ip), intent(in) :: ep_i
  ① type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
  type(full_particle), dimension(n_ip), intent(inout) :: f
  !* Local variables
  integer(c_int) :: i,j
  real(c_double) :: eps2,poti,r3_inv,r_inv
  type(fdps_f64vec) :: xi,ai,rij

do i=1,n_ip
  eps2 = ep_i(i)%eps * ep_i(i)%eps
  xi = ep_i(i)%pos
  ai = 0.0d0
  poti = 0.0d0
  do j=1,n_jp
    rij%x = xi%x - ep_j(j)%pos%x
    rij%y = xi%y - ep_j(j)%pos%y
    rij%z = xi%z - ep_j(j)%pos%z
    r3_inv = rij%x*rij%x &
             + rij%y*rij%y &
             + rij%z*rij%z &
             + eps2
    r_inv = 1.0d0/sqrt(r3_inv)
    r3_inv = r_inv * r_inv
    r_inv = r_inv * ep_j(j)%mass
    r3_inv = r3_inv * r_inv
    ai%x = ai%x - r3_inv * rij%x
    ai%y = ai%y - r3_inv * rij%y
    ai%z = ai%z - r3_inv * rij%z
    poti = poti - r_inv
  end do
  f(i)%pot = f(i)%pot + poti
  f(i)%acc = f(i)%acc + ai
end do

end subroutine calc_gravity_psp

end module user_defined_types

```

① 粒子-超粒子相互作用の場合には、超粒子型を使用する必要がある。

➤ ユーザコードで使用されるツリーオブジェクトの種類に応じた超粒子型である必要がある。

f_main.F90

```
!-----  
!//////////////////// < M A I N R O U T I N E > //////////////////////  
!-----
```

① `subroutine f_main()` ① ユーザコードはすべてサブルーチン `f_main()` の中に実装.

② `use fdps_module` ② FDPSのFortran用APIを使用するためのモジュール.
`use user_defined_types`

`implicit none`

`!* Local parameters`

`integer, parameter :: ntot=2**10`

`!-(force parameters)`

`real, parameter :: theta = 0.5`

`integer, parameter :: n_leaf_limit = 8`

`integer, parameter :: n_group_limit = 64`

`!-(domain decomposition)`

`real, parameter :: coef_ema=0.3`

`!-(timing parameters)`

`double precision, parameter :: time_end = 10.0d0`

`double precision, parameter :: dt = 1.0d0/128.0d0`

`double precision, parameter :: dt_diag = 1.0d0/8.0d0`

`double precision, parameter :: dt_snap = 1.0d0`

`!* Local variables`

③ `type(fdps_controller) :: fdps_ctrl`

`type(c_funptr) :: pfunc_ep_ep, pfunc_ep_sp`

③ Fortran用APIを提供するクラスである `fdps_controller` クラスのオブジェクトを生成.

```
① !* Initialize FDPS  
call fdps_ctrl%PS_initialize()
```

① FDPSの初期化

```
② !* Create domain info object  
call fdps_ctrl%create_dinfo(dinfo_num)  
call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
```

② 領域情報オブジェクトの生成 & 初期化

```
③ !* Create particle system object  
call fdps_ctrl%create_psys(psys_num,'full_particle')  
call fdps_ctrl%init_psys(psys_num)
```

③ 粒子群オブジェクトの生成 & 初期化

```
④ !* Create tree object  
call fdps_ctrl%create_tree(tree_num, &  
                                "Long,full_particle,full_particle,full_particle,Monopole")  
call fdps_ctrl%init_tree(tree_num,ntot,theta, &  
                                n_leaf_limit,n_group_limit)
```

④ ツリーオブジェクトの生成 & 初期化

```
!* Make an initial condition  
call setup_IC(fdps_ctrl,psys_num,ntot)
```

```
⑤ !* Domain decomposition and exchange particle  
call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)  
call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
```

⑤ 領域分割と粒子交換

```
⑥ !* Compute force at the initial time  
pfunc_ep_ep = c_funloc(calc_gravity_pp)  
pfunc_ep_sp = c_funloc(calc_gravity_psp)  
call fdps_ctrl%calc_force_all_and_write_back(tree_num, &  
                                                pfunc_ep_ep, &  
                                                pfunc_ep_sp, &  
                                                psys_num, &  
                                                dinfo_num)
```

⑥ 相互作用の計算

- 相互作用関数の関数ポインタを組込関数 `c_funloc` で取得し、それをAPIの引数に渡している。


```
!* Compute energies at the initial time
clear = .true.
call calc_energy(fdps_ctrl,psys_num,etot0,ekin0,epot0,clear)
```

```
!* Time integration
```

```
time_diag = 0.0d0
```

```
time_snap = 0.0d0
```

```
time_sys = 0.0d0
```

```
num_loop = 0
```

① do ① 時間積分ループの開始

```
!* Output
```

```
!if (fdps_ctrl%get_rank() == 0) then
```

```
! write(*,50)num_loop,time_sys
```

```
! 50 format('(num_loop, time_sys) = ',i5,1x,1es25.16e3)
```

```
!end if
```

```
if ( (time_sys >= time_snap) .or. &
```

```
((time_sys + dt) - time_snap) > (time_snap - time_sys)) ) then
```

```
call output(fdps_ctrl,psys_num)
```

```
time_snap = time_snap + dt_snap
```

```
end if
```

```
!* Compute energies and output the results
```

```
clear = .true.
```

```
call calc_energy(fdps_ctrl,psys_num,etot1,ekin1,epot1,clear)
```

```
if (fdps_ctrl%get_rank() == 0) then
```

```
if ( (time_sys >= time_diag) .or. &
```

```
((time_sys + dt) - time_diag) > (time_diag - time_sys)) ) then
```

```
write(*,100)time_sys,(etot1-etot0)/etot0
```

```
100 format("time: ",1es20.10e3," energy error: ",1es20.10e3)
```

```
time_diag = time_diag + dt_diag
```

```
end if
```

```
end if
```

①

```

!* Leapfrog: Kick-Drift
call kick(fdps_ctrl,psys_num,0.5d0*dt)
time_sys = time_sys + dt
call drift(fdps_ctrl,psys_num,dt)

!* Domain decomposition & exchange particle
if (mod(num_loop,4) == 0) then
  call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
end if
call fdps_ctrl%exchange_particle(psys_num,dinfo_num)

!* Force calculation
pfunc_ep_ep = c_funloc(calc_gravity_pp)
pfunc_ep_sp = c_funloc(calc_gravity_psp)
call fdps_ctrl%calc_force_all_and_write_back(tree_num,      &
                                             pfunc_ep_ep, &
                                             pfunc_ep_sp, &
                                             psys_num,      &
                                             dinfo_num)

!* Leapfrog: Kick
call kick(fdps_ctrl,psys_num,0.5d0*dt)

```

① 時間積分の主要部分.

```

!* Update num_loop
num_loop = num_loop + 1

```

```

!* Termination
if (time_sys >= time_end) then
  exit
end if

```

② end do ② 時間積分ループの終了

③ !* Finalize FDPS call fdps_ctrl%PS_Finalize() ③ FDPSの終了処理

```

end subroutine f_main

```

最後に

- ユーザが書かなければならないのは大体これくらい。
 - ➡ 重力計算の場合は、114行(user_defined.F90)+380行(f_main.F90)
= 約500行で書ける。
- コード内に並列化を意識するようなところは無かった。
 - ➡ コンパイル方法を切り替えるだけで、
OpenMP/MPIを使用するかどうかを切り替えられる。

実習の流れ

- 詳しくはFDPSに付属するチュートリアルを御覧ください。
((`$FDPS`)/doc/doc_tutorial_ftn_ja.pdf)
- 持参して頂いたパソコンにFDPSをダウンロードし、サンプルコードを
 - (1) 並列化無し
 - (2) スレッド並列 (OpenMP)
 - (3) ハイブリッド並列 (OpenMP + MPI)

の3パターンについてコンパイル・実行

【計算内容】 重力: cold collapse 問題、流体: 衝撃波管問題

その後、結果を確認

付録

FDPS指示文

■ FDPS指示文の種類

- ① 派生データ型がどのユーザ定義型(FullParticle型[FP], EssentialParticleI型[EPI], EssentialParticleJ型[EPJ], Force型[Force])に対応するかを指定する指示文。
- ② 派生データ型のメンバ変数がどの必須物理量(粒子の電荷/質量[charge], 粒子の位置[position], 粒子の速度[velocity], 粒子のサイズ/相互作用半径[rsearch])に対応するかを指定する指示文。
- ③ ユーザ定義型同士のデータ移動の方法を指定する指示文。

ここで、赤文字で示された英字はFDPS指示文で使用されるキーワード文字列である。

FDPS指示文

(1)ユーザ定義型の種別を指定する指示文 (指示文①)

■ 書式

```
type, public, bind(c) :: type_name !$fdps keyword  
end type [type_name]
```

或いは

```
!$fdps keyword  
type, public, bind(c) :: type_name  
end type [type_name]
```

■ 機能

派生データ型 *type_name* が *keyword* で指定されたユーザ定義型であることをFDPSに教える。可能なキーワードはFP, EPI, EPJ, Forceであり、それぞれ、FullParticle型, EssentialParticleI型, EssentialParticleJ型, Force型に対応する。詳細は仕様書doc_spec_ftn_ja.pdfの第5.1.1.2.2節を参照。

FDPS指示文

(2) 必須物理量を指定する指示文 (指示文②)

■ 書式

```
type, public, bind(c) :: type_name  
  data_type :: mbr_name !$fdps keyword  
end type [type_name]
```

或いは

```
type, public, bind(c) :: type_name  
  !$fdps keyword  
  data_type :: mbr_name  
end type [type_name]
```

■ 機能

派生データ型 *type_name* のメンバ変数 *mbr_name* が *keyword* で指定された必須物理量であることをFDPSに教える。可能なキーワードは、charge, position, velocity, rsearch であり、それぞれ、粒子の電荷(質量), 位置, 速度, 探索半径(相互作用半径)に対応している。詳細は仕様書の第5.1.1.2.2節を参照のこと。

FDPS指示文

(3) 各ユーザ定義型に固有の指示文 (指示文③)

□ FullParticle型

■ 書式

```
type, public, bind(c) :: FP
    !$fdps copyFromForce force (src_mbr, dst_mbr) (src_mbr, dst_mbr) ...
end type FP
```

■ 機能

相互作用計算後に Force型に対応する派生データ型 *force* から、FullParticle型 FP にデータ(相互作用計算の結果)をコピーする方法を指定する。*src_mbr* がForce型のメンバ変数で、*dst_mbr* がFullParticle型のメンバ変数である。詳細は仕様書の第5.1.2.1節を参照のこと。

なお、拡張機能 Particle Mesh を使用する場合には、別な指示文も必要になるが、割愛する。詳細は、仕様書の第5.1.2.2節を参照のこと。

□ EssentialParticleI型, EssentialParticleJ型

■ 書式

```
type, public, bind(c) :: EPI
  !$fdps copyFromFP fp (src_mbr, dst_mbr) (src_mbr, dst_mbr) ...
end type EPI
```

■ 機能

FullParticle型 *fp* からEssentialParticle?型(?=I, J)にデータをコピーする方法を指定する。*src_mbr*はFullParticle型のメンバ変数で、*dst_mbr*がEssentialParticle?型(?=I, J)のメンバ変数である。詳細は、仕様書の第5.1.3.1節を参照のこと。

□ Force型

Force型に固有の必須指示文は複数の書式をサポートしているが、ここではサンプルコードで使用されているものを紹介する。

■ 書式

```
type, public, bind(c) :: Force
  !$fdps clear [mbr=val, mbr=keep, ...]
end type Force
```

■ 機能

相互作用の計算結果を初期化する方法を指示する。メンバ変数 *mbr* の値を *val* に初期化する。もしメンバ変数の値を変更したくない場合にはキーワード *keep* を指定する。詳細は仕様書の第5.1.5.1節を参照のこと。