[] 連載 FDPS 入門 (2)

理化学研究所　計算科学研究機構
岩澤全規　masaki.iwasawa@riken.jp
理化学研究所　計算科学研究機構
行方大輔　daisuke.namekata@riken.jp

概要
本連載では、私達が開発・公開している多体シミュレーションプログラム開発フレームワーク「FDPS (Framework for Developing Particle Simulators)」を紹介します。連載第二回の今回は、具体的なサンプルコードを例にとり、FDPS を使ってどのように粒子系シミュレーションプログラムを開発するのか、を詳しく説明していきます。開発に必要となる C++言語の文法、FDPS の内部実装についても合わせて解説します。本稿の後半では、Fortran からの使い方を紹介します。

キーワード：粒子シミュレーション、分子動力学、HPC、並列化、MPI

1　はじめに

テスト

2　FDPS の詳しい使い方と内部実装について

テスト

3　FDPS の Fortran からの使い方

本節では、どのようにして Fortran から FDPS を使うのか、について解説します。Fortran から FDPS を利用する場合、ユーザプログラムの開発は大まかには次の手順に沿って行うことになります。

1. 粒子構造体の定義
2. FDPS の Fortran 用インターフェースプログラムの生成
3. 相互作用関数の定義
4. ユーザプログラム本体の開発

C++から利用する場合の手順との最も大きな違いは、ステップ 2 の存在です。Fortran には C++言語のテンプレートに相当する機能がありませんので、FDPS が各種ライブラリ関数を提供するためには、事前に粒子のデータ構造を知っている必要があります。そのため、最初にユーザに粒子構造体を定義してもらい (ステップ 1)、その後、その粒子構造体用のライブラリ関数を生成する (ステップ 2)、という手順を踏む必要があります。このステップ 2 は、実際には、単に我々が提供する Python スクリプトを実行するだけの作業で、特に難しいことはありません。

ユーザはステップ 2 で生成された Fortran インターフェース (FDPS を操作するための Fortran サブルーチン/関数の集まり) を使って、ユーザプログラム本体を開発することとなります。Fortran インターフェースが提供する関数群は、C++のものとほぼ同じになっており、コード開発は、C++から FDPS を利用する場合とほぼ同じ要領で行えます。ただし、Fortran インターフェースは、Fortran 2003 から導入された C 言語との相互運用を可能にする文法を使用していますので、粒子構造体や相互作用関数の定義等は、いくつかの要請に従って実装する必要があります。

以下では、各手順について、より詳しく説明していきます。

3.1　粒子構造体の記述方法

粒子は、Fortran の構造体 (別名、派生データ型) と特別なコメント文を用いて定義します。といっても、わかりにくいと思いますので、まずは例を見て頂きましょう。以下は、第 x 節で紹介した C++の粒子クラスを Fortran で記述したものです。

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

同じものがディレクトリ `sample/fortran/vdw-test`

の user_defined.F90 に記述されています (これより後で示すプログラムのソースファイルも、すべてこのディレクトリにあります)。

この例で、type 〜 end type が 1 つの構造体を表しており、この間に定義されている変数 (id, mass 等) がメンバ変数となります。構造体の名称は、type 文の右側で指定でき、今の場合、fplj となっています。この構造体で表されるデータを、Fortran プログラムから C++で記述された FDPS 本体に渡したり、或いは逆に、FDPS 本体から受け取ったりするため、Fortran インターフェースプログラムでは、Fortran 2003 から言語仕様として導入された C 言語との相互運用を可能にする機能を利用しています。この機能を利用するためには、構造体が C 言語と相互運用可能である必要があります。C 言語と相互運用可能であるための条件の詳細は省きますが、主な条件は、

- 構造体が bind(c) 属性を持つこと
- すべてのメンバ変数が C 言語と相互運用可能なデータ型であること

です。最初の条件は、type 文に bind(c) キーワードをつけるだけクリアできます。2 番目の条件についてですが、Fortran 2003 以降には C 言語の基本データ型に対応するデータ型が Fortran 側に用意されており、それらを用いてメンバ変数を定義すれば満たすことができます。今回の例の integer(kind=c_int) などは、そのようなデータ型の 1 例です。C 言語と相互運用可能なデータ型の一覧は、例えば、GNU gfortran のオンラインドキュメント[1] 等で紹介されています。なお、これらのデータ型は組込モジュール iso_c_binding 内に定義されていますので、use 文で、このモジュールを参照可能にしておく必要があります。構造体には、C 言語と相互運用可能な構造体をメンバ変数として含めることも可能です。fdps_f64vec は FDPS から提供されるそのようなデータ型の 1 つです。

プログラムを見ると、!$fdps で始まるコメント文が多数あることに気づかれるかと思います。これらが本節冒頭で「特別なコメント文」と述べたものです。第 x 節で述べたように、FDPS は、粒子構造体のどのメンバ変数が何を表すのか、といったことを知っている必要があります。!$fdps から始まるコメント文は、このような情報を FDPS に教えるための指示文です。以降、単に FDPS 指示文と呼ぶことにします。FDPS 指示文には大きく分けて以下の 3 種類があります。

(a) 構造体の種別を表す指示文

(b) メンバ変数が何の物理量を表すかを指定する指示文

(c) FDPS 内部で行われるデータ操作の仕方を指示する指示文

以下、これらについて簡単に説明したいと思います。

指示文 (a) は、構造体が 4 つのユーザ定義型 (FullParticle 型、EssentialParticleI 型、EssentialParticleJ 型、Force 型) のどれに対応するかを FDPS に指示するためのもので、今回の例では、

```
!$fdps FP,EPI,EPJ,Force
```

となっています。この例では、この粒子構造体がすべてのユーザ定義型を兼ねることを FDPS に指示しています。

指示文 (b) は、粒子の質量 (電荷) や位置等、FDPS が知っておかなければならない物理量が、どのメンバ変数に対応するかを指定します。今回の例では、

```
real(kind=c_double) :: mass !$fdps charge
type(fdps_f64vec) :: pos !$fdps position
type(fdps_f64vec) :: vel !$fdps velocity
real(kind=c_double) :: search_radius !$fdps rsearch
```

となっています。これによって、FDPS は、mass が粒子の質量 (電荷)、pos が粒子の位置、vel が粒子の速度、search_radius が探索半径であると解釈します。

指示文 (c) は、今回の例では、type 文のすぐ下の

```
!$fdps copyFromForce fplj (pot,pot) (acc,acc)
!$fdps copyFromFP fplj (id,id) (mass,mass) (pos,pos) \
   (search_radius,search_radius)
!$fdps clear id=keep, mass=keep, pos=keep, vel=keep, \
   search_radius=keep
```

の部分に対応します。ここで、\ は行が下の行に継続していることを示す記号で、実際には指示文は 1 行で記述しなければなりません。キーワード copyFromForce を含む指示文は Force 型から FullParticle 型へのデータコピーを、copyFromFP を含む指示文は FullParticle 型から EssentialParticleI 型および EssentialParticleJ 型へのデータコピーの仕方を指定しています。キーワード clear を含む指示文は相互作用計算時に Force 型を初期化する方法を指定しています。

以上、Fortran における粒子構造体の定義方法を簡単に説明しました。紙面の都合上、指示文の文法に関する詳細な説明は省きましたが、詳しい情報は仕様書[2]でご覧頂けます。

3.2 Fortran インターフェースの生成方法

Fortran インターフェースは、以下のコマンドを実行するだけで自動的に生成されます。

```
./$(FDPS_LOC)/scripts/gen_ftn_if.py user_defined.F90
```

[1] https://gcc.gnu.org/onlinedocs/gfortran/ISO_005fC_005fBINDING.html#ISO_005fC_005fBINDING

[2] https://github.com/FDPS/FDPS/blob/master/doc/doc_specs_ftn_ja.pdf

ここで、$(FDPS_LOC) は FDPS のトップディレクトリのパスを表します。また、粒子構造体は、ファイル user_defined.F90 に定義されているとしています。インターフェースの生成が成功した場合、カレントディレクトリに、FDPS_Manipulators.cpp、FDPS_ftn_if.cpp、FDPS_module.F90、main.cpp という 4 つのファイルが作成されているはずです。Fortran インターフェースは、ファイル FDPS_module.F90 内に定義されるモジュール fdps_module 内に実装されています。

3.3 相互作用関数の記述方法

相互作用関数は Fortran のサブルーチンとして定義します。以下に、第 x 節で紹介した C++の相互作用関数を Fortran で記述したもの (インターフェース部分のみ) を示します。

```
subroutine calc_force_fpfp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
   integer(c_int), intent(in), value :: n_ip,n_jp
   type(fplj), dimension(n_ip), intent(in) :: ep_i
   type(fplj), dimension(n_jp), intent(in) :: ep_j
   type(fplj), dimension(n_ip), intent(inout) :: f

   ! 中身は省略

end subroutine calc_force_fpfp
```

相互作用関数もまた C 言語と相互運用可能でなければなりません。そのためには、次の条件

- サブルーチンが bind(c) 属性を持つこと
- 関数内で使用される変数のデータ型が C 言語と相互運用可能なデータ型であること

を満たす必要があります。最初の条件は、サブルーチン名の右側に bind(c) キーワードを付けることでクリアできます。これらの条件に加え、FDPS 側からの要請として、EssentialParticleI 型 配列と EssentialParticleJ 型 配列の個数を示す n_ip と n_jp は値渡しでなければなりません。このため、value 属性を指定しています。

3.4 ユーザプログラム本体の記述方法

最後にユーザプログラム本体が Fortran でどのように記述されるのかを解説します。以下に、第 x 節で紹介した C++のサンプルコードを Fortran で書いたものを示します。

```
subroutine f_main()
   use fdps_module
   use user_defined_types
   implicit none
   !* 局所変数宣言 (重要な変数のみ示す)
   type(fdps_controller) :: fdps_ctrl
   type(c_funptr) :: pfunc_ep_ep

   !* FDPS を初期化
   call fdps_ctrl%PS_Initialize()
   !* 粒子群オブジェクトを生成
   call fdps_ctrl%create_psys(psys_num,'fplj')
   call fdps_ctrl%init_psys(psys_num)
   !* 初期粒子分布を設定
   call setup_IC(fdps_ctrl,psys_num,ntot,boxdh)
   !* 領域情報オブジェクトを生成
   call fdps_ctrl%create_dinfo(dinfo_num)
   call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
   call fdps_ctrl%set_boundary_condition(dinfo_num, &
                                         fdps_bc_periodic_xyz)
   call fdps_ctrl%set_pos_root_domain(dinfo_num,pos_ll,pos_ul)
   !* 領域分割と粒子交換
   call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
   call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
   !* ツリーオブジェクトを生成
   call fdps_ctrl%create_tree(tree_num, &
                              "Short,fplj,fplj,fplj,Scatter")
   call fdps_ctrl%init_tree(tree_num,3*ntot,theta, &
                            n_leaf_limit,n_group_limit)
   !* 相互作用計算
   pfunc_ep_ep = c_funloc(calc_force_fpfp)
   call fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
                                                pfunc_ep_ep, &
                                                psys_num,    &
                                                dinfo_num)

   !(時間積分ループはここに記述)

   !* FDPS の終了処理を実行
   call fdps_ctrl%PS_Finalize()

end subroutine f_main
```

プログラム中のコメント文は、各場所で何をしているかを説明しています。コメント文を追うと、プログラム全体の構造がわかります。すなわち、FDPS の初期化から始まって、各種のオブジェクトを作成し、それらを使って相互作用計算を行う、というものになっています。このように、FDPS の使い方は C++の場合とほぼ同じことがわかります。しかし、いくつか Fortran に特有な部分がありますので、以下にそれらをまとめます。

- ユーザプログラムはサブルーチン f_main() の中に記述しなければなりません。これは、プログラムのメイン関数は C++側にあって、そこから f_main() が呼び出されているためです。
- FDPS の Fortran インターフェースは、モジュール fdps_module 内で定義されるクラス fdps_controller のメンバ関数として提供されます。Fortran のクラスは C++のクラスと同じようにメンバ変数とメンバ関数を持つデータ構造です。サブルーチン f_main() 内で、このクラスのメンバ関数を呼び出すため、use 文でモジュール fdps_module にアクセス可能にし (プログラム 2 行目)、局所変数として、fdps_controller クラスのオブジェクト fdps_ctrl を宣言しています (6 行目)。メンバ関数の呼出は、

  ```
  call fdps_ctrl%member_func()
  ```

  のように行います。
- 粒子群オブジェクト等の FDPS 固有のオブジェクトはすべて整数変数で管理します。
- 相互作用関数を FDPS に渡すためには関数の C 言語アドレスが必要となりますが、それは Fortran

2003 で追加された組込関数 `c_funloc` で取得できます。関数の返り値は、`c_funptr` 型です。

各 API の機能に関しては、API の名称が C++のものと類似していることから予想がつくと思いますが、詳しくは仕様書をご参照して頂ければと思います。

最後に、Fortran で記述されたサンプルコードの実行例をお見せしたいと思います。サンプルコードはディレクトリ `sample/fortran/vdw-test` にありますので、そこに移動して、`make; ./vdwtest.out` を実行すれば、以下のような出力が得られるはずです。

```
******** FDPS has successfully begun. ********
====================================
 Paralleization infomation:
   # of processes is 1
   # of thread is    1
====================================
 nptcl_1d =            8
np_ave=1536
time:    7.8125000000E-003, energy error:    1.5220343903E-010
time:    6.2500000000E-002, energy error:    2.5923769738E-009
(途中省略)
time:    9.9375000000E+000, energy error:    4.0086395482E-005
time:    1.0000000000E+001, energy error:    6.2026013003E-005
******** FDPS has successfully finished. ********
```

以上、簡単ではありますが、Fortran で FDPS を利用する方法について解説を行いました。

次回は、より現実的な分子動力学のシミュレーションプログラムの実装例について解説します。原子結合等の取扱もそこで扱う予定です。

参考文献

[1] M. Iwasawa, A. Tanikawa, N. Hosono, K. Nitadori, T. Muranushi, and J. Makino, *Publ. Astron. Soc. J.*, **68**, 54 (2016).

著者紹介

岩澤全規 (学術博):〔経歴〕1989 年東京大学大学院総合文化研究科博士課程修了，同年東京大学大学教養学部助手．2016 年から現所属．〔専門〕天体物理、計算科学．

行方大輔 (博士 (理学)) :〔経歴〕2010 年北海道大学大学院理学院博士課程修了．2016 年から現所属.〔専門〕天体物理、計算科学．