FDPS Fortran/C 言語 インタフェース 仕様書

行方大輔, 岩澤全規, 似鳥啓吾, 谷川衝, 村主崇行, Long Wang, 細野七月, 野村昴太郎, and 牧野淳一郎

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

目 次

第1章　この文書について

1.1　文書の構成

この文書は大規模並列粒子シミュレーションの開発を支援する Framework for Developing Particle Simulator (FDPS) の Fortran 及び C 言語 インターフェース (以降、単に Fortran/C 言語インターフェースと略記する) の仕様書である。この文書は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の行方大輔、岩澤全規、似鳥啓吾、谷川衝、細野七月、村主崇行、Long Wang、牧野淳一郎によって記述された。

この文書は以下のような構成となっている。

第 2、3、7 章には、FDPS Fortran/C 言語 インターフェースを使ってプログラムを書く際に前提となる情報が記述されている。第 2 章には、FDPS の概要として、FDPS の基本的な考えかたや動作が記述されている。第 3 章には、FDPS Fortran/C 言語 インターフェースのファイル構成とその概要が記述されている。第 7 章には、FDPS Fortran/C 言語 インターフェースを使用したコードをコンパイルする時にどのようなマクロを用いればよいかが記述されている。

第 4、5、8 章には、FDPS Fortran/C 言語 インターフェースを使ってプログラムを書く際に必要となる情報が提供されている。第 4 章には、FDPS で独自に定義されている派生データ型 (Fortran)、或いは、構造体 (C 言語) が記述されている。第 5 には、FDPS の API を使用する際にユーザーが定義する必要がある Fortran の派生データ型やサブルーチン (C 言語では構造体や関数) について記述されている。第 8 章には、Fortran、或いは、C 言語から FDPS を操作するための API について記述されている。

第 9、10 章には、FDPS の API を使用したコードを記述したがコードが思ったように動作しない場合に有用な情報が記載されている。第 9 章にはエラーメッセージが記述されている。第 10 章には、FDPS の限界について記述されている。

最後に第 11 章にはこの文書の変更履歴が記述されている。

1.2 ライセンス

MIT ライセンスに準ずる。標準機能のみ使用する場合は、Iwasawa et al. (2016, Publications of the Astronomical Society of Japan, 68, 54)、及び、Namekata et al. (2018, Publications of the Astronomical Society of Japan, 70, 70) の引用をお願いします。

拡張機能の Particle Mesh クラスは GreeM コード (開発者 : 石山智明、似鳥啓吾) (Ishiyama, Fukushige & Makino 2009, Publications of the Astronomical Society of Japan, 61, 1319; Ishiyama, Nitadori & Makino, 2012 SC'12 Proceedings of the International Conference on High Performance Computing, Networking Stroage and Analysis, No. 5) のモジュールを使用している。GreeM コードは Yoshikawa & Fukushige (2005, Publications of the Astronomical Society of Japan, 57, 849) で書かれたコードをベースとしている。Particle Mesh クラスを使用している場合は、上記 3 つの文献の引用をお願いします。

拡張機能のうち x86 版 Phantom-GRAPE を使用する場合は Tanikawa et al.(2012, New Astronomy, 17, 82) と Tanikawa et al.(2012, New Astronomy, 19, 74) の引用をお願いします。

1.3 ユーザーサポート

FDPS を使用したコード開発に関する相談は fdps-support<at>mail.jmlab.jp で受け付けています (<at>は@に変更お願い致します)。以下のような場合は各項目毎の対応をお願いします。

1.3.1 コンパイルできない場合

ユーザーには以下の情報提供をお願いします。

- コンパイル環境
- コンパイル時に出力されるエラーメッセージ
- ソースコード (可能ならば)

1.3.2 コードがうまく動かない場合

ユーザーには以下の情報提供をお願いします。

- 実行環境
- 実行時に出力されるエラーメッセージ
- ソースコード (可能ならば)

1.3.3 その他

思い通りの性能がでない場合やその他の相談なども、上のメールアドレスにお知らせください。

第2章　FDPS概要

この章ではFDPSの概要を記述する。FDPSの開発目的、FDPSの基本的な考えかた、FDPSを使用して作成したコードの動作について概説する。

2.1　開発目的

粒子シミュレーションは、重力N体シミュレーション、SPHシミュレーション、渦糸法、MPS法、分子動力学シミュレーションなど理学工学の様々な分野で使用されている。より大きい空間スケール、より高い空間分解能 (または質量分解能)、より長い時間スケールの物理現象を追跡するために、高性能な粒子シミュレーションコードへの要請はますます強くなっている。

高性能な粒子シミュレーションコードを組むためには、シミュレーションコードの大規模並列化を避けることはできない。粒子シミュレーションコードの大規模並列化をする際には、ロードバランスのため動的領域分割、領域分割に合わせた粒子交換、ノード間通信の削減と最適化、キャッシュ利用効率の向上、SIMDユニット利用効率の向上、アクセラレータへの対応など、数多くの困難な処理を行う必要がある。現在、研究グループは個別にこれらの処理へ対応している。

しかし、上記の処理は粒子シミュレーション共通のものである。FDPSの開発目的は、これらの処理を高速に行うライブラリを提供し、大規模並列化への対応に追われていた研究者の負担を軽くすることである。FDPSを使うことで、研究者がよりクリエイティブな仕事に専念できるようになれば、幸いである。

2.2　基本的な考えかた

ここではFDPSの基本的な考えかたについて記述する。

2.2.1　大規模並列粒子シミュレーションの手順

まずFDPSにおいて、大規模並列粒子シミュレーションがどのような手順で行われることを想定しているかを記述する。粒子シミュレーションは、以下のような微分方程式を時間発展させるものである。

$$\frac{d\boldsymbol{u}_i}{dt} = \sum_j f(\boldsymbol{u}_i, \boldsymbol{u}_j) + \sum_s g(\boldsymbol{u}_i, \boldsymbol{v}_s) \tag{2.1}$$

ここで $\boldsymbol{u}_i$ は粒子 i の物理量ベクトルであり、この物理量には質量、位置、速度など粒子が持つあらゆる物理量が含まれる。関数 f は粒子 j から粒子 i への作用を規定する。以後、作用を受ける粒子を i 粒子、作用を与える粒子を j 粒子と呼ぶことにする。$\boldsymbol{v}_s$ は i 粒子から十分遠方にある粒子を 1 つの粒子としてまとめた粒子 (以後、この粒子を超粒子と呼ぶ) の物理量ベクトルである。関数 g は超粒子から i 粒子への作用を規定する。式 (2.1) の第 2 項は、重力やクーロン力など無限遠まで到達する長距離力の場合はゼロではない。しかし流体の圧力のような短距離力はゼロである。

大規模並列化された粒子シミュレーションコードは以下の手順で式 (2.1) を時間発展させる。ここではデータの入出力や初期化は省略している。

1. 以下の 2 段階の手順でどのプロセスがどの粒子の式 (2.1) を時間発展させるか決める。
   (a) プロセスの間でロードバランスを取れるように、シミュレーションで扱っている空間の領域を分割し、各プロセスの担当領域を決める (領域分割)。
   (b) 各プロセスが、自分の担当する領域に存在する全粒子の物理量ベクトル $\boldsymbol{u}_i$ を持つように、他のプロセスと物理量ベクトル $\boldsymbol{u}_i$ を交換する (粒子交換)。
2. 各プロセスは、自分の担当する全粒子の式 (2.1) の右辺を計算するのに必要な j 粒子の物理量ベクトル $\boldsymbol{u}_j$ と超粒子の物理量ベクトル $\boldsymbol{v}_s$ を他のプロセスと通信することで集めて、j 粒子のリストと超粒子のリスト (まとめて相互作用リストと呼ぶ) を作る (相互作用リストの作成)。
3. 各プロセスは自分の担当する全粒子に対して、式 (2.1) の右辺を計算し、$d\boldsymbol{u}_i/dt$ を求める (相互作用の計算)。
4. 各プロセスは、自分の担当する全粒子の物理量ベクトル $\boldsymbol{u}_i$ とその時間導関数 $d\boldsymbol{u}_i/dt$ を使って、全粒子の時間積分を実行し、次の時刻の物理量ベクトル $\boldsymbol{u}_i$ を求める (時間積分)。
5. 手順 1 に戻る。

2.2.2 ユーザーと FDPS の役割分担

FDPS は、プロセス間の通信が発生する処理は FDPS が担当し、プロセス間の通信の発生しない処理はユーザーが担当するという役割分担を基本としている。従って、前節に挙げた、領域分割・粒子交換 (項目 1)・相互作用リストの作成 (項目 2) を FDPS が、相互作用の計算 (項目 3)・時間積分 (項目 4) をユーザーが担当することになる。ユーザーは FDPS の API を呼び出すだけで、大規模並列化に関わる煩雑な処理を避けつつ、高性能な任意の相互作用の粒子シミュレーションコードを手に入れることができる。

2.2.3 ユーザーのやること

ユーザーが FDPS を使って粒子シミュレーションコードを作成するときにやることは以下の項目である。

- 粒子の定義 (第 5 章)。粒子の持つ物理量 (式 (2.1) で言えば $\boldsymbol{u}_i$) の指定。例えば質量、位置、速度、加速度、元素組成、粒子サイズ、など。
- 相互作用の定義 (第 5 章)。粒子間の相互作用 (式 (2.1) で言えばで関数 f, g) を指定。例えば、重力、クーロン力、圧力、など。
- FDPS の API の呼出 (第 8 章)

2.2.4 補足

式 (2.1) の右辺は 2 粒子間相互作用の重ね合わせである。従って、FDPS の API を呼ぶだけでは、3 つ以上の粒子の間の相互作用の計算を行うことはできない。しかし、FDPS はネイバーリストを返す API を用意している。ネイバーリストを用いれば、ユーザーはプロセス間の通信の処理をすることなく、このような相互作用の計算をできる。

第 2.2.1 節で示した手順は、全粒子が同じ時間刻みを持っている。そのため、FDPS の API を呼び出すだけでは、独立時間刻みで時間積分を効率的に行うことができない。しかし、上と同じくネイバーリストを返す API があるため、Particle Particle Particle Tree 法を用いて独立時間刻みを実装することは可能であろう。

2.3 コードの動作

ここでは FDPS を使用して作成したコードの動作の概略を記述する。まずはじめに C++ で記述された FDPS 本体の動作の概略を説明し、その後、Fortran/C 言語 インターフェースの動作を解説する。

2.3.1 FDPS 本体

FDPS 本体のコードには 4 つのモジュール[注 1)]がある。3 つは FDPS のモジュールで、1 つはユーザー定義のモジュールである。まとめると以下のようになる。

- 領域クラス：全プロセスが担当する領域の情報と、領域分割を行う API を持つ
- 粒子群クラス：全粒子の情報と、プロセスの間での粒子交換を行う API を持つ

注 1) 1 つの大きな機能を提供するためのデータと手続きの集まりという意味。FDPS ではモジュールを C++のクラス機能によって実現している。

- 相互作用ツリークラス：粒子分布から作られたツリー構造と、相互作用リストを作成する API を持つ
- ユーザー定義クラス：ある 1 粒子を定義するクラス、粒子間の相互作用を定義する関数オブジェクトを持つ

これら 4 つのモジュールの間で情報がやり取りされる。これは図 2.1 で概観できる。図 2.1 に示された情報のやりとりは、第 2.2.1 節に記述された手順 1 から 3 と、これらの手順以前に行われる手順 (手順 0 とする) に対応する。以下はこれらの手順の詳細な記述である。

0. ユーザー定義クラスのうち 1 粒子を定義するクラスが粒子群クラスへ、粒子間の相互作用を定義する関数オブジェクトが相互作用ツリークラスへ渡される。これはクラスの継承ではなく、粒子を定義するクラスは粒子群クラスのテンプレート引数として、粒子間の相互作用を定義する関数オブジェクトは相互作用ツリークラスの API の引数として渡される

1. 以下の 2 段階でロードバランスを取る
   (a) 領域クラスが持つ領域分割の API が呼ばれる。このとき粒子情報が粒子群クラスから領域クラスへ渡される (赤字と赤矢印)
   (b) 粒子群クラスが持つ粒子交換の API が呼ばれる。このとき領域情報が領域クラスから粒子群クラスへ渡される (青字と青矢印)

2. 相互作用ツリークラスが持つ相互作用リストを作成する API が呼ばれる。このとき領域情報が領域クラスから相互作用ツリークラスへ、粒子情報が粒子群クラスから相互作用ツリークラスへ渡される (緑字と緑矢印)

3. 相互作用ツリーククラスが持つ相互作用を定義した関数オブジェクトを呼び出す API が呼ばれる。相互作用計算が実行され、相互作用計算の結果が相互作用ツリークラスから粒子群クラスへ渡される (灰色の字と灰色矢印)

2.3.2 Fortran/C 言語 インターフェース

次章以降で詳しく述べるが、前節で述べた API は Fortran インターフェース及び C 言語インターフェースにも用意されている。したがって、Fortran 或いは C 言語においても、第 2.2.1 節に記述された手順は、対応する API の適切な呼び出しにより実現できる。

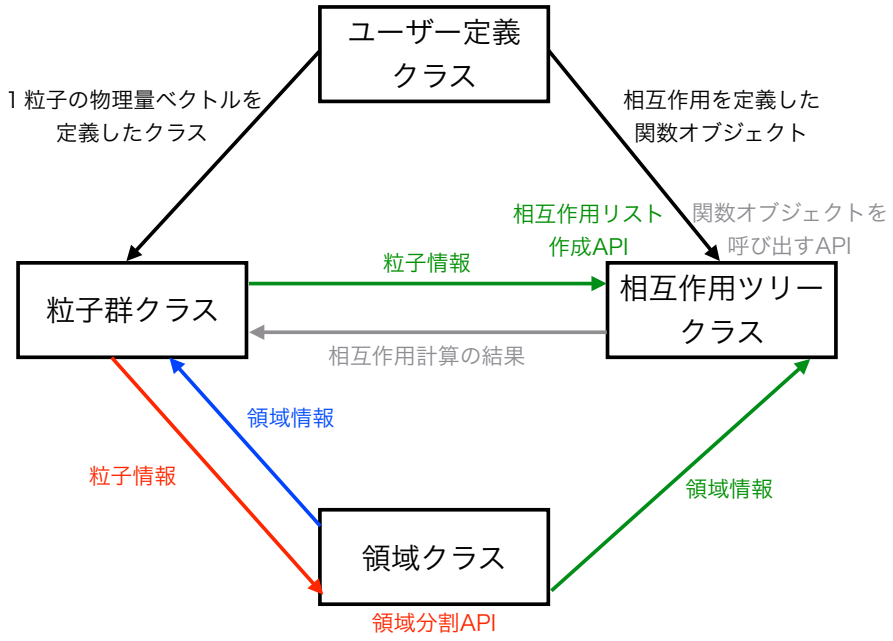

図 2.1: モジュールインターフェースと情報の流れの模式図。

第3章　Fortran/C言語 インターフェースのファイル構成と概要

この章では、FDPS Fortran/C 言語 インターフェースのファイル構成と概要について記述する。はじめにソースファイルの構成とインターフェース概要について記述し、その後、ドキュメントとサンプルコードについて記述する。

3.1　ファイル構成と概要

3.1.1　FDPS 本体

FDPS 本体のソースファイルはディレクトリ `src` の下にある。FDPS 本体は C++ で記述されており、FDPS の標準機能関係のソースファイルはすべて `src` の 直下にある。FDPS には拡張機能が用意されており、現時点では、Particle Mesh と x86 版 Phantom-GRAPE が実装されている。それぞれのソースファイルが、`src/particle_mesh` と `src/phantom_GRAPE_x86` にある。これら拡張機能は静的ライブラリとして使用される。そのため、各ディレクトリにおいて、ユーザ自身の手で、静的ライブラリを作成する必要がある。詳細は FDPS 本体の仕様書 (`doc/doc_specs_cpp_ja.pdf`) をご覧頂きたい。

3.1.2　Fortran インターフェース

前節で述べた機能の内、Fortran から利用可能なのは、FDPS 標準機能 (一部 API は除く) と拡張機能 Particle Mesh である。ユーザは Fortran インターフェースプログラムを通して、これらの機能を使用することとなる。この Fortran インターフェースは、ユーザがディレクトリ `scripts` の下に置かれたスクリプト `gen_ftn_if.py` を実行することで生成される (スクリプトの仕様は第 6 章で解説する)。このインターフェース生成用スクリプトは、FDPS を利用するにあたってユーザ自身が定義 (実装) しなければならない派生データ型 (**ユーザ定義型**; 第 5 章参照) を解析して、インターフェースプログラムを生成する。**したがって、ユーザ最初にしなければならないことはユーザ定義型の実装である**。FDPS の Fortran 用のインターフェースがライブラリの形ではなく、このようなインターフェースプログラムの生成という形で提供される理由については別途第 3.1.5 節で解説する。Fortran でユーザ定義型を実装するのに必要となる Fortran ファイルが `src/fortran_interface/modules` に、インターフェース生成時に設計図として使用されるファイル群が `src/fortran_interface/blueprints` に配置されている。

図 3.1 に、Fortran 用インターフェースプログラムの生成が正常に行われた場合の Fortran インターフェースのファイル構成とその役割を示している。図の破線で囲まれた 4 つのファイル (FDPS_module.F90, FDPS_ftn_if.cpp, FDPS_Manipulators.cpp, main.cpp) がスクリプトによって生成される Fortran インターフェースプログラムであり、f_main.F90 がユーザ側が用意するプログラムである。図の点線で囲まれたファイル (FDPS_vector.F90、FDPS_matrix.F90、FDPS_super_particle.F90 等) は、前述したように、ユーザがユーザ定義型およびユーザ定義関数 (第 2 章参照) を記述するのに必要な派生データ型の定義を与える。以下、それぞれのインターフェースプログラムの役割について説明を行う。

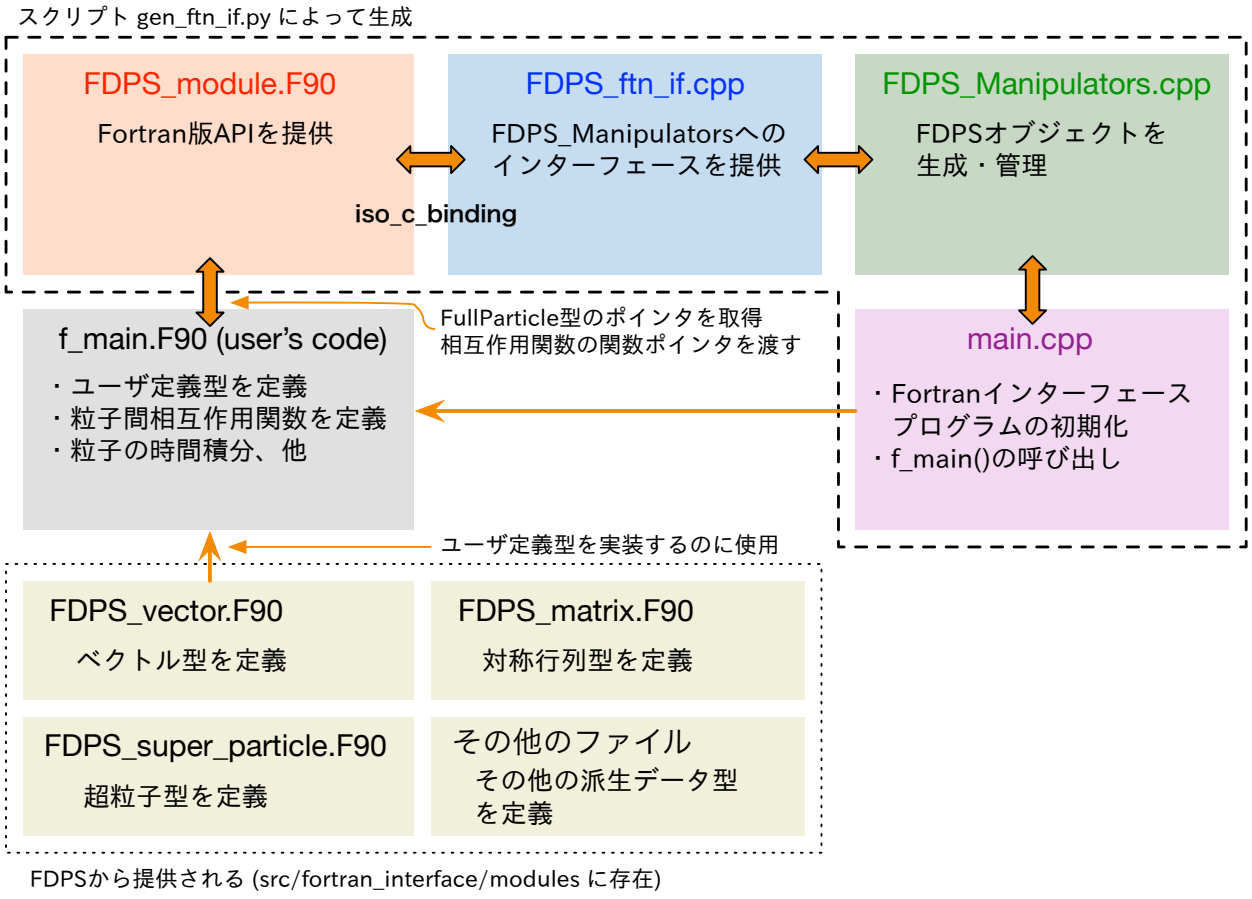

図 3.1: Fortran インターフェースとユーザコードの関係。

まず FDPS_Manipulators.cpp と main.cpp について説明する。FDPS 本体は C++で記述されているため、第 2 章「FDPS 概要」で説明した領域クラス、粒子群クラス、相互作用ツリークラスの C++オブジェクトは、すべて C++ファイル内で生成し、管理する必要がある。これを行うのが、FDPS_Manipulators.cpp である。同様の理由によって、実行プログラムの main 関数は C++ファイルに置く必要がある。そのため、main.cpp が生成される。この main.cpp では f_main() という名称の Fortran のサブルーチンを呼び出す。したがって、ユーザは Fortran サブルーチン f_main() を用意し、その中にユーザコードを実装する必要がある。詳細は第 8 章「API 仕様一覧」に譲るが、FDPS_Manipulators.cpp で生成される C++オブジェクトは、Fortran の整数変数に割り当てられる。したがって、ユーザはこれら

のオブジェクトを整数変数を使って管理することとなる。

次に FDPS_ftn_if.cpp について説明する。Fortran は C++の関数を直接呼び出して使用することはできないが、Fortran 2003 の機能 (Fortran モジュール iso_c_binding で提供される機能のこと) を使用することで、C 言語の関数を呼び出すことが可能になる。そこで、本 FDPS Fortran インターフェースでは、FDPS_Manipulators.cpp 内で定義される各種の C++関数の C 言語インターフェースを別途用意し、これらを Fortran から呼び出して、FDPS を操作する仕組みとした。これら C 言語インターフェースが FDPS_ftn_if.cpp に実装されている。

最後に、FDPS_module.F90 について説明する。FDPS_module.F90 は、C 言語インターフェースを呼び出すための派生データ型 FDPS_controller をユーザに提供する。この FDPS_controller は、Fortran 2003 のクラス (メンバ関数を持つ派生データ型のこと) であり、そのメンバ関数が FDPS の Fortran 用インターフェースを与える。メンバ関数、すなわち、Fortran インターフェースの一覧は第 8 章「API 仕様一覧」で記述する。FDPS_controller は、FDPS_module.F90 において、以下のように定義されている (リスト 3.1) :

Listing 3.1: FDPS_module.F90 の構造

```
1  module FDPS_module
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     !**** FDPS controller
6     type, public :: FDPS_controller
7     contains
8        !
9        ! APIs are defined here.
10       !
11    end type FDPS_controller
12
13 end module FDPS_module
```

見やすさのため、上記のリストにおいて、メンバ関数の宣言部の記述は省略している。実際には、各メンバ関数の宣言が、文字列 contains と文字列 end type FDPS_controller の間の領域に記述される。このような仕様のため、ユーザはユーザコードにおいて、以下の手順で Fortran インターフェースを使用する必要がある :

(1) モジュール FDPS_module を use する
(2) クラス FDPS_controller のオブジェクトを生成する
(3) 生成した FDPS_controller オブジェクトのメンバ関数を呼び出す

最も単純な使用例をリスト 3.2 に示す :

Listing 3.2: Fortran インターフェースの使用例

```
1  subroutine f_main()
2     use FDPS_module ! Step (1)
3     implicit none
4     type(FDPS_controller) :: fdps_ctrl ! Step (2)
5
6     ! Call Fortran interface
```

```
7     call fdps_ctrl%PS_initialize() ! Step (3)
8
9  end subroutine f_main
```

リスト中にコメントで示された番号は、上の手順の番号に対応している。

3.1.3 C 言語 インターフェース

C 言語の場合も Fortran の場合と全く同様に、C 言語 インターフェースプログラムを通して、FDPS の機能を使用することになる。C 言語 インターフェースプログラムの生成は、ユーザがディレクトリ `scripts` の下に置かれたスクリプト `gen_c_if.py` を実行することで行われる。このスクリプトは、FDPS を利用するにあたってユーザ自身が定義 (実装) しなければならない構造体 (同様に**ユーザ定義型**と呼称) を解析して、インターフェースプログラムを生成する。C 言語でユーザ定義型を実装するのに必要となる C 言語ヘッダーファイルが `src/c_interface/headers` に、インターフェース生成時に設計図として使用されるファイル群が `src/c_interface/blueprints` に配置されている。

図 3.2 に、C 言語用インターフェースプログラムの生成が正常に行われた場合の C 言語インターフェースのファイル構成とその役割を示している。図の破線で囲まれた 4 つのファイル (`FDPS_c_if.h`, `FDPS_ftn_if.cpp`, `FDPS_Manipulators.cpp`, `main.cpp`) がスクリプトによって生成される C 言語インターフェースプログラムであり、`c_main.c` がユーザ側が用意するプログラムである。図の点線で囲まれたファイル (`FDPS_basic.h`、`FDPS_enum.h`、`FDPS_vector.h`、`FDPS_matrix.h`、`FDPS_super_particle.h` 等) は、前述したように、ユーザがユーザ定義型およびユーザ定義関数 (第 2 章参照) を記述するのに必要な構造体の定義を与えるものである。図からわかるように、ファイル構成は、Fortran インターフェースプログラムと非常に類似した構成となっており (図 3.1 参照)、特に同名のファイルの役割は Fortran インターフェースで説明したファイルと全く同じである。以下、異なる部分についての注意書きのみを記す。

- 実行ファイルの `main` 関数は `main.cpp` にある。この `main.cpp` では `c_main()` という名称の `void` 関数を呼び出す。したがって、ユーザは `void` 関数 `c_main()` を用意し、その中にユーザコードを実装する必要がある。

- FDPS の C 言語用 API のプロトタイプ宣言は `FDPS_c_if.h` に記述されている。したがって、ユーザは FDPS の機能を利用するため、このファイルをインクルードする必要がある。`FDPS_c_if.h` では、FDPS が提供する構造体の定義が記述されたヘッダーファイル群 (`FDPS_basic.h` 等) がインクルードされているため、ユーザはこのファイルのみをインクルードすれば、これら構造体をユーザコードの中で利用することができる。

3.1.4 Fortran/C 言語 インターフェースを使ったコード開発の流れ

本節では、FDPS の Fortran/C 言語 インターフェースを使ったユーザコード開発の流れについて記述する。大まかな流れは以下のようになる:

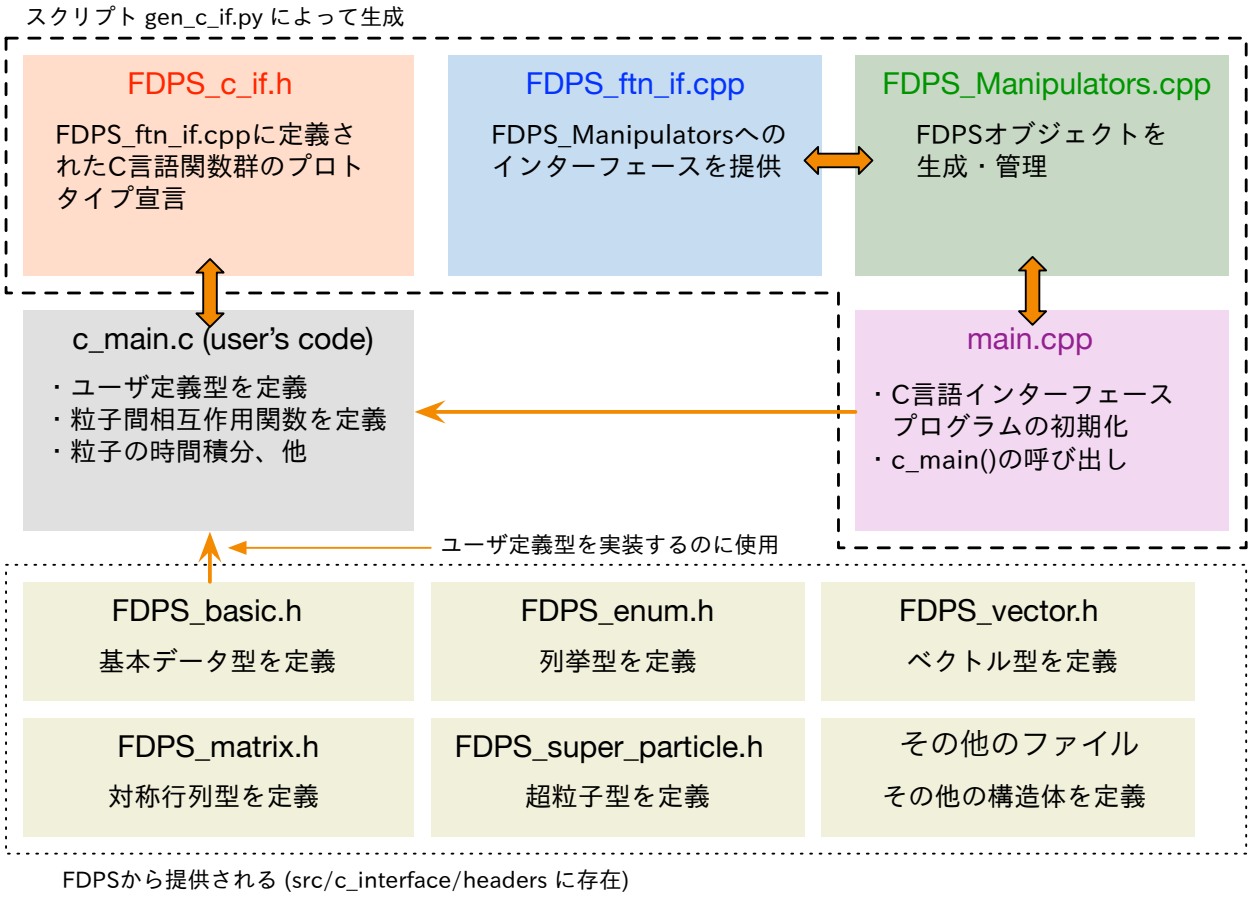

図 3.2: C 言語 インターフェースとユーザコードの関係。

[1] **ユーザ定義型の実装**
前節で述べた通り、FDPS の Fortran/C 言語 インターフェースを生成するためには、はじめにユーザ定義型を実装しなければならない。ユーザ定義型は Fortran で FDPS を利用する場合には派生データ型で、C 言語で FDPS を利用する場合には構造体として実装する。ユーザ定義型の記述方法の詳細は、第 5 章で説明する。

[2] **インターフェースプログラムの生成**
ユーザ定義型の実装が完了したら、インターフェース生成用スクリプト `gen_ftn_if.py` または `gen_c_if.py` を使って、インターフェースプログラムを生成する。生成が完了した時点で、ユーザは FDPS の Fortran/C 言語用インターフェースをユーザコードの中で使用することができるようになる。スクリプトの使用法と仕様については第 6 章で説明する。

[3] **ユーザ定義関数の実装**
ユーザは相互作用を記述する関数 (ユーザ定義関数) を実装しなければならない。ユーザ定義関数は、Fortran ではサブルーチン、C 言語では `void` 関数として実装する。ユーザ定義関数の記述方法の詳細は、第 5 章で説明する。

[4] **ユーザコードの開発**
ユーザ定義型、ユーザ定義関数、FDPS API を用いて、ユーザが行いたい粒子シミュレーションコードを開発する。この際、次の点に注意して開発を行う必要がある:

- ユーザコードは Fortran のサブルーチン `f_main()`、或いは、C 言語の `void` 関数 `c_main()` の中に実装しなければならない。
- FDPS Fortran インターフェースの API は、クラス `FDPS_controller` のメンバ関数として提供される。したがって、FDPS API はメンバ関数を呼び出して使用する。一方、FDPS C 言語 インターフェースの API のプロトタイプ宣言は、`FDPS_c_if.h` でなされているため、このファイルをインクルードすることで API を呼び出すことが可能となる。

Fortran インターフェースを用いたコードの例に関しては、`sample/fortran` の下で提供されているサンプルコードを参照して頂きたい (第 3.3 節も参照のこと)。一方、C 言語インターフェースを用いたサンプルコードは、`sample/c` の下に用意されている。

[5] **コンパイル**
ユーザコードの実装が完了したら、コンパイルを行い、実行プログラムを得る。前節で述べたように、インターフェースプログラムは C++言語と、Fortran 言語或いは C 言語のソースファイルが混在した構成となっており、単一の言語のみで構成されたプログラムとは異なる仕方でコンパイルする必要がある。この点に関しての詳細は、第 7 章で解説する。FDPS ではコンパイル時のマクロ定義を使い、いくつかの設定を行うことが可能である。これに関しても、第 7 章で解説する。拡張機能 Particle Mesh を使用する場合には、事前に必要なライブラリをインストールし、コンパイル時に適切にライブラリを指定することが必要である。

[6] **実行**
コンパイルして得られる実行ファイルは、通常の実行ファイルと違いはない。ユーザが利用している計算機環境の利用規則に則って、実行ファイルを実行する。

3.1.5 インターフェースプログラム生成の必要性

前々節で述べたように、FDPS の Fortran/C 言語用インターフェースはライブラリの形で提供されるものではなく、インターフェースプログラムのソースコードの形で提供される。本節では、この理由について解説を行う。

まず、準備として、C++での FDPS の使用について概説する。第 2 章 2.2.3 節で述べた通り、FDPS ではユーザは粒子や相互作用の定義を自由に行うことができ、これによって、FDPS は様々なタイプの粒子シミュレーションに対して適用可能となっている。この自由度を実現するため、FDPS 本体の関数は C++のテンプレート機能を用いて記述されている。ここで、テンプレート機能とは、Fortran でいうサブルーチンや関数 (或いは C 言語の関数) に、(変数ではなく) データ型を引数として受け取れるようにする機能のことである。この機能によって、C++では仮のデータ型を使用して関数を記述することが可能となる (この仮の

データ型はコンパイル時に具体的なデータ型になってさえいればよい)。また、FDPS 本体は C++のヘッダファイルの形で提供される。したがって、C++で FDPS を使用する場合、ユーザは FDPS のヘッダファイルをユーザコードの中でインクルードし、FDPS API のテンプレート引数にユーザが定義した粒子型を指定して使用する。ユーザコードのコンパイル時には、FDPS API の関数で使用されるすべての変数のデータ型が決定されているため、コンパイラは問題なくユーザプログラムをコンパイルすることが可能となっているのである。

Fortran や C 言語にはテンプレート機能に相当するものは存在しないため、仮の (或いは、未定の) データ型を用いてサブルーチンや関数を実装する、ということは Fortran や C 言語では不可能である。これが、Fortran/C 言語用インターフェースをライブラリの形で提供できない 1 つの理由である。我々は、Fortran や C 言語においてもユーザが粒子や相互作用の定義を自由に行えるようにするため、ユーザが実装した粒子の派生データ型/構造体等を調べ、それに応じて適切な API を自動的に生成する方法を採用している。

もう 1 つの理由は、C++で実装された FDPS をそのまま使用しているからである。C++で記述された FDPS と Fortran 或いは C 言語のプログラムの間でデータをやり取りするためには、Fortran や C 言語で記述された粒子型と同等な粒子クラスを C++側に用意する必要がある。これにもユーザが実装した派生データ型や構造体を解析して生成するという作業が必要となる。

以上の理由により、FDPS Fortran/C 言語 インターフェースは、ソースコードで提供される形となっている。

3.2 ドキュメント

ドキュメント関係のファイルはディレクトリ `doc` の下にある。サンプルコードを使って FDPS Fortran インターフェースの基本的な使用法を解説するチュートリアル文書が `doc_tutorial_ftn_ja.pdf` である。C 言語 インターフェースの基本的な使用方法についてのチュートリアル文書は `doc_tutorial_c_ja.pdf` である。仕様書 (本文書) が `doc_specs_ftn_ja.pdf` である。

3.3 サンプルコード

Fortran と C 言語のサンプルコードが、それぞれ、ディレクトリ `sample/fortran` 及び `sample/c` の下にある。サンプルコードは 4 つ用意されており、それぞれ、無衝突系の重力 N 体シミュレーションコード (`sample/fortran/nbody`, `sample/c/nbody`)、固定長カーネルを使った SPH シミュレーションコード (`sample/fortran/sph`, `sample/c/sph`)、P^3M(Particle-Particle-Particle-Mesh) 計算用コード (`sample/fortran/p3m`, `sample/c/p3m`)、円盤銀河の N 体/SPH シミュレーションコード (`sample/fortran/nbody+sph`, `sample/c/nbody+sph`) となっている。

第4章　FDPSで提供されるデータ型

FDPS Fortran/C 言語 インターフェースでは独自のデータ型が定義されている。データ型には、ベクトル型、対称行列型、超粒子型、時間プロファイル型、列挙型がある。これらに加え、C 言語では基本データ型もある。これらのデータ型は第 5 章で説明するユーザ定義型やユーザ定義関数の実装に必要となる他、いくつかの API の引数に指定したり、返り値を受け取る際に必要となる。

4.1　基本データ型 (C 言語のみ)

基本データ型は C 言語インターフェースでのみ提供されるデータ型で、fdps_s32, fdps_u32, fdps_f32, fdps_s64, fdps_u64, fdps_f64 の 6 種類がある。これらは、FDPS が提供する他の構造体の定義に使用される。src/c_interface/headers/FDPS_basic.h において、以下のように定義されている。

Listing 4.1: 基本データ型 (C 言語のみ)

```
1  #pragma once
2
3  /* 32 bit data types */
4  typedef int           fdps_s32;
5  typedef unsigned int  fdps_u32;
6  #ifdef PARTICLE_SIMULATOR_ALL_64BIT_PRECISION
7  typedef double        fdps_f32;
8  #else
9  typedef float         fdps_f32;
10 #endif
11
12 /* 64 bit data types */
13 typedef long long int          fdps_s64;
14 typedef unsigned long long int fdps_u64;
15 typedef double                 fdps_f64;
```

ただし、マクロ PARTICLE_SIMULATOR_ALL_64BIT_PRECISION は現時点では正式にはサポートしておらず、C 言語インターフェースが正しく動作するのは、このマクロが未定義の場合のみである。

4.2　ベクトル型

ベクトル型は fdps_f32vec と fdps_f64vec の 2 種類がある。これらは、Fortran では src/fortran_interface/modules/FDPS_vector.F90、C 言語では src/c_interface/headers/

FDPS_vecotr.h で、以下のように定義される。それぞれ、32 bit と 64 bit の浮動小数点数をメンバ変数として持つベクトルを表す。ベクトルの空間次元はデフォルトでは 3 であり、コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合のみ 2 となる。

Listing 4.2: ベクトル型 (Fortran)

```
1  module fdps_vector
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     type, public, bind(c) :: fdps_f32vec
6  #ifdef PARTICLE_SIMULATOR_TWO_DIMENSION
7        real(kind=c_float) :: x,y
8  #else
9        real(kind=c_float) :: x,y,z
10 #endif
11    end type fdps_f32vec
12
13    type, public, bind(c) :: fdps_f64vec
14 #ifdef PARTICLE_SIMULATOR_TWO_DIMENSION
15       real(kind=c_double) :: x,y
16 #else
17       real(kind=c_double) :: x,y,z
18 #endif
19    end type fdps_f64vec
20
21 end module fdps_vector
```

Listing 4.3: ベクトル型 (C 言語)

```
1  #pragma once
2  #include "FDPS_basic.h"
3
4  //**** PS::F32vec
5  typedef struct  {
6  #ifdef PARTICLE_SIMULATOR_TWO_DIMENSION
7     fdps_f32 x,y;
8  #else
9     fdps_f32 x,y,z;
10 #endif
11 } fdps_f32vec;
12
13 //**** PS::F64vec
14 typedef struct  {
15 #ifdef PARTICLE_SIMULATOR_TWO_DIMENSION
16    fdps_f64 x,y;
17 #else
18    fdps_f64 x,y,z;
19 #endif
20 } fdps_f64vec;
```

Fortran においては、これらベクトル型に対して、代入 (=) と演算子 (+,-,*,/) が表 4.1 のように拡張されている。詳細に関しては、FDPS_vector.F90 を参照して頂きたい。

記号	左辺	右辺	定義
=	ベクトル	スカラー†	左辺に右辺を代入する。但し、右辺がスカラーの場合、左辺の各成分すべてに右辺を代入し、右辺が配列の場合、配列の先頭から順に、左辺ベクトルの x,y(,z) 成分に配列要素を代入。
	ベクトル	スカラー値の配列‡	
	ベクトル	ベクトル	
+	ベクトル	スカラー値の配列	左辺と右辺を加算する。但し、オペランドの 1 つが配列の場合、配列の各要素は先頭から順番に、ベクトル成分 x,y(,z) に対応するものとする。
	スカラー値の配列	ベクトル	
	ベクトル	ベクトル	
	なし	ベクトル	何も行わない
-	ベクトル	スカラー値の配列	左辺から右辺を減算する。但し、オペランドの 1 つが配列の場合、配列の各要素は先頭から順番に、ベクトル成分 x,y(,z) に対応するものとする。
	スカラー値の配列	ベクトル	
	ベクトル	ベクトル	
	なし	ベクトル	ベクトルの各成分の符号反転
*	ベクトル	スカラー	スカラーベクトル積
	スカラー	ベクトル	
	ベクトル	スカラー値の配列	内積。但し、オペランドの 1 つが配列の場合、配列の各要素は先頭から順番に、ベクトル成分 x,y(,z) に対応するものとする。
	スカラー値の配列	ベクトル	
	ベクトル	ベクトル	
/	ベクトル	スカラー	左辺を右辺で除算する

† ここでスカラー型は Fortran の基本データ型である必要がある。
‡ 配列の要素数は、コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合には 2、それ以外の場合には 3 である必要がある。

表 4.1: ベクトル型に対して拡張された代入と演算子

4.3 対称行列型

対称行列型には、fdps_f32mat と fdps_f64mat の 2 種類がある。Fortran では src/fortran_interface/modules/FDPS_matrix.F90、C 言語では src/c_interface/headers/FDPS_matrix.h において、以下のように定義される。それぞれ、32 bit と 64 bit の浮動小数点数をメンバ変数として持つ対称行列を表す。行列の次元はデフォルトでは 3 であり、コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合のみ 2 となる。

Listing 4.4: 対称行列型 (Fortran)

```
module fdps_matrix
   use, intrinsic :: iso_c_binding
   implicit none

   !**** PS::F32mat
   type, public, bind(c) :: fdps_f32mat
#ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
      real(kind=c_float) :: xx,yy,zz,xy,xz,yz
#else
      real(kind=c_float) :: xx,yy,xy
#endif
   end type fdps_f32mat

   !**** PS::F64mat
   type, public, bind(c) :: fdps_f64mat
#ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
      real(kind=c_double) :: xx,yy,zz,xy,xz,yz
#else
      real(kind=c_double) :: xx,yy,xy
#endif
   end type fdps_f64mat

end module fdps_matrix
```

Listing 4.5: 対称行列型 (C 言語)

```
#pragma once
#include "FDPS_basic.h"

//**** PS::F32mat
typedef struct {
#ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
   fdps_f32 xx,yy,zz,xy,xz,yz;
#else
   fdps_f32 xx,yy,xy;
#endif
} fdps_f32mat;

//**** PS::F64mat
typedef struct {
#ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
   fdps_f64 xx,yy,zz,xy,xz,yz;
#else
   fdps_f64 xx,yy,xy;
#endif
} fdps_f64mat;
```

Fortran では、これら対称行列型に対して、代入 (=) と演算子 (+,-,*,-) が表 4.2 のように拡張されている。詳細に関しては、FDPS_matrix.F90 を参照して頂きたい。

記号	左辺	右辺	定義
=	対称行列	スカラー†	左辺に右辺を代入。但し、右辺がスカラーの場合、左辺の各成分すべてに右辺が代入される。
	対称行列	対称行列	
+	対称行列	対称行列	左辺に右辺を加算する
	なし	対称行列	何も行わない
-	対称行列	対称行列	左辺から右辺を減算する
	なし	対称行列	行列の各成分の符号反転
*	対称行列	スカラー	スカラー行列積
	スカラー	対称行列	
	対称行列	対称行列	行列積
/	対称行列	スカラー	左辺を右辺で除算する

† ここでスカラー型は Fortran の基本データ型である必要がある。

表 4.2: 対称行列型に対して拡張された代入と演算子

4.4 超粒子型

超粒子型は、粒子-超粒子間の相互作用計算を記述するのに必要となるデータ型である。ここで超粒子とは、FDPS 本体で長距離力計算の方法として採用しているツリー法において、力を計算する対象の粒子に対して、十分に遠くにある複数の粒子を 1 つの粒子として表現したものである。これら超粒子型は、FDPS 本体から超粒子のデータを受け取るのに使用される。

超粒子型には、`fdps_spj_monopole`、`fdps_spj_quadrupole`、`fdps_spj_monopole_geomcen`、`fdps_spj_dipole_geomcen`、`fdps_spj_quadrupole_geomcen`、`fdps_spj_monopole_scatter`、`fdps_spj_quadrupole_scatter`、`fdps_spj_monopole_symmetry`、`fdps_spj_quadrupole_symmetry`、`fdps_spj_monopole_cutoff` がある。Fortran では `src/fortran_interface/modules/FDPS_super_particle.F90`、C 言語では `src/c_interface/headers/FDPS_super_particle.h` において、以下のように定義される。ここで、各超粒子型のメンバ変数には前述したベクトル型および対称行列型が使用されていることに注意されたい。

Listing 4.6: 超粒子型 (Fortran)

```
1 module fdps_super_particle
2    use, intrinsic :: iso_c_binding
3    use fdps_vector
4    use fdps_matrix
5    implicit none
6
```

```
 7     !**** PS::SPJMonopole
 8     type, public, bind(c) :: fdps_spj_monopole
 9        real(kind=c_double) :: mass
10        type(fdps_f64vec) :: pos
11     end type fdps_spj_monopole
12
13     !**** PS::SPJQuadrupole
14     type, public, bind(c) :: fdps_spj_quadrupole
15        real(kind=c_double) :: mass
16        type(fdps_f64vec)  :: pos
17        type(fdps_f64mat)  :: quad
18     end type fdps_spj_quadrupole
19
20     !**** PS::SPJMonopoleGeometricCenter
21     type, public, bind(c) :: fdps_spj_monopole_geomcen
22        integer(kind=c_long_long) :: n_ptcl
23        real(kind=c_double) :: charge
24        type(fdps_f64vec) :: pos
25     end type fdps_spj_monopole_geomcen
26
27     !**** PS::SPJDipoleGeometricCenter
28     type, public, bind(c) :: fdps_spj_dipole_geomcen
29        integer(kind=c_long_long) :: n_ptcl
30        real(kind=c_double) :: charge
31        type(fdps_f64vec) :: pos
32        type(fdps_f64vec) :: dipole
33     end type fdps_spj_dipole_geomcen
34
35     !**** PS::SPJQuadrupoleGeometricCenter
36     type, public, bind(c) :: fdps_spj_quadrupole_geomcen
37        integer(kind=c_long_long) :: n_ptcl
38        real(kind=c_double) :: charge
39        type(fdps_f64vec) :: pos
40        type(fdps_f64vec) :: dipole
41        type(fdps_f64mat) :: quadrupole
42     end type fdps_spj_quadrupole_geomcen
43
44     !**** PS::SPJMonopoleScatter
45     type, public, bind(c) :: fdps_spj_monopole_scatter
46        real(kind=c_double) :: mass
47        type(fdps_f64vec) :: pos
48     end type fdps_spj_monopole_scatter
49
50     !**** PS::SPJQuadrupoleScatter
51     type, public, bind(c) :: fdps_spj_quadrupole_scatter
52        real(kind=c_double) :: mass
53        type(fdps_f64vec)  :: pos
54        type(fdps_f64mat)  :: quad
55     end type fdps_spj_quadrupole_scatter
56
57     !**** PS::SPJMonopoleSymmetry
58     type, public, bind(c) :: fdps_spj_monopole_symmetry
59        real(kind=c_double) :: mass
60        type(fdps_f64vec) :: pos
61     end type fdps_spj_monopole_symmetry
```

```
62
63    !**** PS::SPJQuadrupoleSymmetry
64    type, public, bind(c) :: fdps_spj_quadrupole_symmetry
65       real(kind=c_double) :: mass
66       type(fdps_f64vec) :: pos
67       type(fdps_f64mat) :: quad
68    end type fdps_spj_quadrupole_symmetry
69
70
71    !**** PS::SPJMonopoleCutoff
72    type, public, bind(c) :: fdps_spj_monopole_cutoff
73       real(kind=c_double) :: mass
74       type(fdps_f64vec) :: pos
75    end type fdps_spj_monopole_cutoff
76
77 end module fdps_super_particle
```

Listing 4.7: 超粒子型 (C 言語)

```
1  #pragma once
2  #include "FDPS_basic.h"
3  #include "FDPS_vector.h"
4  #include "FDPS_matrix.h"
5
6  #ifdef PARTICLE_SIMULATOR_SPMOM_F32
7  typedef fdps_s32    fdps_sSP;
8  typedef fdps_f32    fdps_fSP;
9  typedef fdps_f32vec fdps_fSPvec;
10 typedef fdps_f32mat fdps_fSPmat;
11 #else
12 typedef fdps_s64    fdps_sSP;
13 typedef fdps_f64    fdps_fSP;
14 typedef fdps_f64vec fdps_fSPvec;
15 typedef fdps_f64mat fdps_fSPmat;
16 #endif
17
18 //**** PS::SPJMonopole
19 typedef struct {
20    fdps_fSP mass;
21    fdps_fSPvec pos;
22 } fdps_spj_monopole;
23
24 //**** PS::SPJQuadrupole
25 typedef struct {
26    fdps_fSP mass;
27    fdps_fSPvec pos;
28    fdps_fSPmat quad;
29 } fdps_spj_quadrupole;
30
31 //**** PS::SPJMonopoleGeometricCenter
32 typedef struct {
33    fdps_sSP n_ptcl;
34    fdps_fSP charge;
35    fdps_fSPvec pos;
36 } fdps_spj_monopole_geomcen;
37
```

```
38 //**** PS::SPJDipoleGeometricCenter
39 typedef struct {
40    fdps_sSP n_ptcl;
41    fdps_fSP charge;
42    fdps_fSPvec pos;
43    fdps_fSPvec dipole;
44 } fdps_spj_dipole_geomcen;
45
46 //**** PS::SPJQuadrupoleGeometricCenter
47 typedef struct {
48    fdps_sSP n_ptcl;
49    fdps_fSP charge;
50    fdps_fSPvec pos;
51    fdps_fSPvec dipole;
52    fdps_fSPmat quadrupole;
53 } fdps_spj_quadrupole_geomcen;
54
55 //**** PS::SPJMonopoleScatter
56 typedef struct {
57    fdps_fSP mass;
58    fdps_fSPvec pos;
59 } fdps_spj_monopole_scatter;
60
61 //**** PS::SPJQuadrupoleScatter
62 typedef struct {
63    fdps_fSP mass;
64    fdps_fSPvec pos;
65    fdps_fSPmat quad;
66 } fdps_spj_quadrupole_scatter;
67
68 //**** PS::SPJMonopoleSymmetry
69 typedef struct {
70    fdps_fSP mass;
71    fdps_fSPvec pos;
72 } fdps_spj_monopole_symmetry;
73
74 //**** PS::SPJQuadrupoleSymmetry
75 typedef struct {
76    fdps_fSP mass;
77    fdps_fSPvec pos;
78    fdps_fSPmat quad;
79 } fdps_spj_quadrupole_symmetry;
80
81 //**** PS::SPJMonopoleCutoff
82 typedef struct {
83    fdps_fSP mass;
84    fdps_fSPvec pos;
85 } fdps_spj_monopole_cutoff;
```

それぞれの超粒子型は、FDPS 本体の相互作用ツリークラスの種類と対応している。したがって、ユーザは生成した相互作用ツリーオブジェクトの種類に応じて、対応する超粒子型を用いる必要がある。相互作用ツリーオブジェクトの種類と超粒子型の対応関係を表 4.3 に示す。超粒子は長距離力の計算でのみ使用されるため、短距離力用のツリーはこの表に含まれていないことに注意されたい。他の種類の相互作用ツリーおよび相互作用ツリーオブジェ

クトを生成する方法に関しては、第 8 章 8.4 節のツリー用 API の説明とともに行う。

ツリーの種別	モーメント情報の計算方法†	相互作用範囲	超粒子型
Long-Monopole 型	単極子 (重心)	計算領域全域	`fdps_spj_monopole`
Long-Quadrupole 型	四重極子 (重心) まで	計算領域全域	`fdps_spj_quadrupole`
Long-MonopoleGeometricCenter 型	単極子 (幾何中心)	計算領域全域	`fdps_spj_monopole_geomcen`
Long-DipoleGeometricCenter 型	双極子 (幾何中心) まで	計算領域全域	`fdps_spj_dipole_geomcen`
Long-QuadrupoleGeometricCenter 型	四重極子 (幾何中心) まで	計算領域全域	`fdps_spj_quadrupole_geomcen`
Long-MonopoleWithScatterSearch 型‡	単極子 (重心)	計算領域全域	`fdps_spj_monopole_scatter`
Long-QuadrupoleWithScatterSearch 型‡	四重極子 (重心) まで	計算領域全域	`fdps_spj_quadrupole_scatter`
Long-MonopoleWithSymmetrySearch 型‡	単極子 (重心)	計算領域全域	`fdps_spj_monopole_symmetry`
Long-QuadrupoleWithSymmetrySearch 型‡	四重極子 (重心) まで	計算領域全域	`fdps_spj_quadrupole_symmetry`
Long-MonopoleWithCutoff 型	単極子 (重心)	カットオフ半径内	`fdps_spj_monopole_cutoff`

† モーメントを粒子の重心を中心として計算する場合には「(重心)」、幾何中心を中心として計算する場合には「(幾何中心)」を付けて表している。
‡ ユーザ指定された半径を用いた近傍粒子探索が可能。相互作用計算においては、近傍粒子は超粒子に含めず通常の粒子として扱われるようになる。

表 4.3: 長距離力計算用ツリーの種類と対応する超粒子型

4.5 時間プロファイル型

時間プロファイル型は、FDPS 内部で行われる各種計算に要した時間を取得するのに使用される。時間プロファイル型は fdps_time_profile の 1 種類が存在し、Fortran では src/fortran_interface/modules/FDPS_time_profile.F90、C 言語では src/c_interface/headers/FDPS_time_profile.h において、以下のように定義される。このデータ型はもっぱら時間取得用 API で使用される (詳細は第 8 章参照)。

Listing 4.8: 時間プロファイル型 (Fortran)

```
1  module fdps_time_profile
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     !**** PS::TimeProfile
6     type, public, bind(c) :: fdps_time_prof
7        real(kind=c_double) :: collect_sample_particle
8        real(kind=c_double) :: decompose_domain
9        real(kind=c_double) :: exchange_particle
10       real(kind=c_double) :: set_particle_local_tree
11       real(kind=c_double) :: set_particle_global_tree
12       real(kind=c_double) :: make_local_tree
13       real(kind=c_double) :: make_global_tree
14       real(kind=c_double) :: set_root_cell
15       real(kind=c_double) :: calc_force
16       real(kind=c_double) :: calc_moment_local_tree
17       real(kind=c_double) :: calc_moment_global_tree
18       real(kind=c_double) :: make_LET_1st
19       real(kind=c_double) :: make_LET_2nd
20       real(kind=c_double) :: exchange_LET_1st
21       real(kind=c_double) :: exchange_LET_2nd
22
23       real(kind=c_double) :: morton_sort_local_tree
24       real(kind=c_double) :: link_cell_local_tree
25       real(kind=c_double) :: morton_sort_global_tree
26       real(kind=c_double) :: link_cell_global_tree
27
28       real(kind=c_double) :: make_local_tree_tot
29       ! = make_local_tree + calc_moment_local_tree
30       real(kind=c_double) :: make_global_tree_tot
31       real(kind=c_double) :: exchange_LET_tot
32       ! = make_LET_1st + make_LET_2nd + exchange_LET_1st +
               exchange_LET_2nd
33
34       real(kind=c_double) :: calc_force__core__walk_tree
35
36       real(kind=c_double) :: calc_force__make_ipgroup
37       real(kind=c_double) :: calc_force__core
38       real(kind=c_double) :: calc_force__copy_original_order
39
40       real(kind=c_double) :: exchange_particle__find_particle
41       real(kind=c_double) :: exchange_particle__exchange_particle
42
43       real(kind=c_double) :: decompose_domain__sort_particle_1st
```

```
44        real(kind=c_double) :: decompose_domain__sort_particle_2nd
45        real(kind=c_double) :: decompose_domain__sort_particle_3rd
46        real(kind=c_double) :: decompose_domain__gather_particle
47
48        real(kind=c_double) :: decompose_domain__setup
49        real(kind=c_double) :: decompose_domain__determine_coord_1st
50        real(kind=c_double) :: decompose_domain__migrae_particle_1st
51        real(kind=c_double) :: decompose_domain__determine_coord_2nd
52        real(kind=c_double) :: decompose_domain__determine_coord_3rd
53        real(kind=c_double) :: decompose_domain__exchange_pos_domain
54
55        real(kind=c_double) :: exchange_LET_1st__a2a_n
56        real(kind=c_double) :: exchange_LET_1st__icomm_sp
57        real(kind=c_double) :: exchange_LET_1st__a2a_sp
58        real(kind=c_double) :: exchange_LET_1st__icomm_ep
59        real(kind=c_double) :: exchange_LET_1st__a2a_ep
60    end type fdps_time_prof
61
62 end module fdps_time_profile
```

Listing 4.9: 時間プロファイル型 (C 言語)

```
1  //**** PS::TimeProfile
2  typedef struct {
3     double collect_sample_particle;
4     double decompose_domain;
5     double exchange_particle;
6     double set_particle_local_tree;
7     double set_particle_global_tree;
8     double make_local_tree;
9     double make_global_tree;
10    double set_root_cell;
11    double calc_force;
12    double calc_moment_local_tree;
13    double calc_moment_global_tree;
14    double make_LET_1st;
15    double make_LET_2nd;
16    double exchange_LET_1st;
17    double exchange_LET_2nd;
18    double write_back;
19
20    double morton_sort_local_tree;
21    double link_cell_local_tree;
22    double morton_sort_global_tree;
23    double link_cell_global_tree;
24
25    double make_local_tree_tot; // = make_local_tree +
            calc_moment_local_tree
26    double make_global_tree_tot;
27    double exchange_LET_tot; // = make_LET_1st + make_LET_2nd +
            exchange_LET_1st + exchange_LET_2nd
28
29    double calc_force__core__walk_tree;
30    double calc_force__core__keep_list;
31    double calc_force__core__copy_ep;
32    double calc_force__core__dispatch;
```

```
33     double calc_force__core__retrieve;
34
35     double calc_force__make_ipgroup;
36     double calc_force__core;
37     double calc_force__copy_original_order;
38
39     double exchange_particle__find_particle;
40     double exchange_particle__exchange_particle;
41
42     double decompose_domain__sort_particle_1st;
43     double decompose_domain__sort_particle_2nd;
44     double decompose_domain__sort_particle_3rd;
45     double decompose_domain__gather_particle;
46
47     double decompose_domain__setup;
48     double decompose_domain__determine_coord_1st;
49     double decompose_domain__migrae_particle_1st;
50     double decompose_domain__determine_coord_2nd;
51     double decompose_domain__determine_coord_3rd;
52     double decompose_domain__exchange_pos_domain;
53
54     double exchange_LET_1st__a2a_n;
55     double exchange_LET_1st__icomm_sp;
56     double exchange_LET_1st__a2a_sp;
57     double exchange_LET_1st__icomm_ep;
58     double exchange_LET_1st__a2a_ep;
59
60     double add_moment_as_sp_local;
61     double add_moment_as_sp_global;
62 } fdps_time_prof;
```

4.6 列挙型

本節では、FDPS Fortran/C 言語 インターフェースで定義されている列挙型について記述する。

4.6.1 境界条件型

境界条件型は、境界条件を指定する API `set_boundary_condition` (Fortran) または `fdps_set_boundary_condition` (C 言語) で使用される (第 8 章 8.3 節「領域情報オブジェクト用 API」参照)。Fortran では `FDPS_module.F90` において、C 言語では `src/c_interface/headers/FDPS_enum.h` において、以下のように定義されている。

Listing 4.10: 境界条件型 (Fortran)

```
1 module FDPS_module
2    use, intrinsic :: iso_c_binding
3    implicit none
4
5    !* Enum types
```

```
6     !**** PS::BOUNDARY_CONDITION
7     enum, bind(c)
8        enumerator :: fdps_bc_open
9        enumerator :: fdps_bc_periodic_x
10       enumerator :: fdps_bc_periodic_y
11       enumerator :: fdps_bc_periodic_z
12       enumerator :: fdps_bc_periodic_xy
13       enumerator :: fdps_bc_periodic_xz
14       enumerator :: fdps_bc_periodic_yz
15       enumerator :: fdps_bc_periodic_xyz
16       enumerator :: fdps_bc_shearing_box
17       enumerator :: fdps_bc_user_defined
18    end enum
19
20 end module FDPS_module
```

Listing 4.11: 境界条件型 (C 言語)

```
1  typedef enum {
2     FDPS_BC_OPEN,
3     FDPS_BC_PERIODIC_X,
4     FDPS_BC_PERIODIC_Y,
5     FDPS_BC_PERIODIC_Z,
6     FDPS_BC_PERIODIC_XY,
7     FDPS_BC_PERIODIC_XZ,
8     FDPS_BC_PERIODIC_YZ,
9     FDPS_BC_PERIODIC_XYZ,
10    FDPS_BC_SHEARING_BOX,
11    FDPS_BC_USER_DEFINED,
12 } FDPS_BOUNDARY_CONDITION;
```

表 4.4 に、Fortran の各列挙子に対応する境界条件を示す。C 言語の列挙子名は文字の大小を除き、Fortran の列挙子名と同じである。したがって、適切に読み替えて頂きたい。

4.6.2 相互作用リストモード型

相互作用リストモード型は、相互作用計算時に相互作用リストを使い回すかどうかを決定するためのデータ型である。これは、Fortran では、ツリーオブジェクト用 API `calc_force_all_and_write_back` 及び `calc_force_all` において、C 言語では、API `fdps_calc_force_all_and_write_back` 及び `fdps_calc_force_all` において、引数として使われる (第 8 章 8.4 節「ツリーオブジェクト用 API」参照)。このデータ型は、Fortran では `FDPS_module.F90` において、C 言語では `src/c_interface/headers/FDPS_enum.h` において、以下のように定義されている。

Listing 4.12: 相互作用リストモード型 (Fortran)

```
1  module FDPS_module
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     !* Enum types
```

列挙子	境界条件
fdps_bc_open	開放境界となる。**デフォルトではこの境界条件となる。**
fdps_bc_periodic_x	x 軸方向のみ周期境界、その他の軸方向は開放境界となる。周期の境界の下限は閉境界、上限は開境界となっている。この境界の規定はすべての軸方向にあてはまる。
fdps_bc_periodic_y	y 軸方向のみ周期境界、その他の軸方向は開放境界となる。
fdps_bc_periodic_z	z 軸方向のみ周期境界、その他の軸方向は開放境界となる。
fdps_bc_periodic_xy	x,y 軸方向のみ周期境界、その他の軸方向は開放境界となる。
fdps_bc_periodic_xz	x,z 軸方向のみ周期境界、その他の軸方向は開放境界となる。
fdps_bc_periodic_yz	y,z 軸方向のみ周期境界、その他の軸方向は開放境界となる。
fdps_bc_periodic_xyz	全方向周期境界条件。
fdps_bc_shearing_box	シアリングボックス境界条件 (現時点では未実装)。
fdps_bc_user_defined	ユーザ定義の境界条件 (現時点で未実装)。

表 4.4: 境界条件型の列挙子に対応する境界条件

```
6     !**** PS::INTERACTION_LIST_MODE
7     enum, bind(c)
8        enumerator :: fdps_make_list
9        enumerator :: fdps_make_list_for_reuse
10       enumerator :: fdps_reuse_list
11    end enum
12
13 end module FDPS_module
```

Listing 4.13: 相互作用リストモード型 (C 言語)

```
1 typedef enum {
2    FDPS_MAKE_LIST,
3    FDPS_MAKE_LIST_FOR_REUSE,
4    FDPS_REUSE_LIST,
5 } FDPS_INTERACTION_LIST_MODE;
```

表 4.5 に、Fortan の各列挙子に対応する動作モードを示す。C 言語の列挙子名は Fortran の列挙子名と文字の大小を除き一致している。したがって、適切に読み替えて頂きたい。

4.6.3 CALC_DISTANCE_TYPE 型

CALC_DISTANCE_TYPE 型は粒子間の距離の計算方法を変更するための型である．Fortran では FDPS_module.F90 において、C 言語では src/c_interface/headers/FDPS_enum.h において、以下のように定義されている。

列挙子	機能
fdps_make_list	相互作用リストを毎回作り相互作用計算を行う場合に用いる。相互作用リストの再利用はできない。**デフォルトではこの動作が仮定される。**
fdps_make_list_for_reuse	相互作用リストを再利用し相互作用計算を行いたい場合に用いる。このオプションを選択する事で FDPS は相互作用リストを作りそれを保持する。作成した相互作用リストは fdps_make_list_for_reuse、もしくは、fdps_make_list を用いて相互作用計算を行った際に破棄される。
fdps_reuse_list	事前に作成された相互作用リストを再利用して相互作用計算を行う。再利用される相互作用リストは fdps_make_list_for_reuse を選択時に作成した相互作用リストである。相互作用リストに含まれる超粒子のモーメント情報は最新の粒子情報で再計算されたものが使用される。

表 4.5: 相互作用リストモード型の列挙子に対応する動作モード

Listing 4.14: CALC_DISTANCE_TYPE 型 (Fortran)

```
1  module FDPS_module
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     !**** PS::CALC_DISTANCE_TYPE
6     enum, bind(c)
7       enumerator :: fdps_calc_distance_type_normal
8       enumerator :: fdps_calc_distance_type_nearest_x
9       enumerator :: fdps_calc_distance_type_nearest_y
10      enumerator :: fdps_calc_distance_type_nearest_xy
11      enumerator :: fdps_calc_distance_type_nearest_z
12      enumerator :: fdps_calc_distance_type_nearest_xz
13      enumerator :: fdps_calc_distance_type_nearest_yz
14      enumerator :: fdps_calc_distance_type_nearest_xyz
15    end enum
16
17 end module FDPS_module
```

Listing 4.15: CALC_DISTANCE_TYPE 型 (C 言語)

```
1  typedef enum {
2      FDPS_CALC_DISTANCE_TYPE_NORMAL = 0,
3      FDPS_CALC_DISTANCE_TYPE_NEAREST_X = 1,
4      FDPS_CALC_DISTANCE_TYPE_NEAREST_Y = 2,
5      FDPS_CALC_DISTANCE_TYPE_NEAREST_XY = 3,
6      FDPS_CALC_DISTANCE_TYPE_NEAREST_Z = 4,
7      FDPS_CALC_DISTANCE_TYPE_NEAREST_XZ = 5,
8      FDPS_CALC_DISTANCE_TYPE_NEAREST_YZ = 6,
9      FDPS_CALC_DISTANCE_TYPE_NEAREST_XYZ = 7,
10 } FDPS_CALC_DISTANCE_TYPE;
```

デフォルトの NORMAL では、 tree walk での距離判定に通常のデカルト距離 (L2 ノルム) が使われる。 X, Y, Z の指定がはいると、それぞれの座標軸について周期境界であるとしてもっとも近い粒子イメージとのデカルト距離を計算する。

このパラメータは、周期境界を指定せず、(BOUNDARY_CONDITION_OPEN)、相互作用が LONG である時に、tree walk は周期境界であるように動作することを可能にする。

4.6.4 EXCHANGE_LET_MODE 型

EXCHANGE_LET_MODE 型は LET 交換の方法を決定するためのデータ型である。

Fortran では FDPS_module.F90 において、C 言語では src/c_interface/headers/FDPS_enum.h において、以下のように定義されている。

Listing 4.16: EXCHANGE_LET_MODE 型 (Fortran)

```
1  module FDPS_module
2     use, intrinsic :: iso_c_binding
3     implicit none
4
5     !**** PS::EXCHANGE_LET_MODE
6     enum, bind(c)
7       enumerator :: fdps_exchange_let_a2a
8       enumerator :: fdps_exchange_let_p2p_exact
9       enumerator :: fdps_exchange_let_p2p_fast
10    end enum
11
12 end module FDPS_module
```

Listing 4.17: EXCHANGE_LET_MODE 型 (C 言語)

```
1 typedef enum {
2     EXCHANGE_LET_A2A,
3     EXCHANGE_LET_P2P_EXACT,
4     EXCHANGE_LET_P2P_FAST,
5 }EXCHANGE_LET_MODE;
```

EXCHANGE_LET_A2A では、LET の交換に MPI_Alltoall を使用する。

EXCHANGE_LET_P2P_EXACT では、LET の交換に MPI_Alltoall を使用せず、MPI_Allgather と MPI_Isend/recv を使用する。MPI_Alltoall が効率的に動かない計算機では、こちらの方が速い場合がある。結果は丸め誤差の範囲で PS::EXCHANGE_LET_A2A を用いた場合と一致する。

EXCHANGE_LET_P2P_FAST では、LET の交換に MPI_Alltoall を使用せず、MPI_Allgather と MPI_Isend/recv を使用する。結果は PS::EXCHANGE_LET_P2P_EXACT を用いた場合と異なるが、より通信量が減っているため、高速に動作する可能性がある。

第5章　ユーザー定義型・ユーザー定義関数

本章では、ユーザーが定義しなければならない派生データ型 (Fortran) 或いは 構造体 (C 言語) (**ユーザ定義型**) と相互作用関数 (**ユーザ定義関数**) について記述する。ユーザー定義型には、FullParticle 型、EssentialParticleI 型、EssentialParticleJ 型、Force 型がある。また、ユーザー定義関数には、粒子-粒子相互作用を記述する calcForceEpEp と、粒子-超粒子間相互作用を記述する calcForceEpSp がある。本章で記述するのは、ユーザ定義型やユーザ定義関数を定義する際の規定である。FDPS は、ユーザ定義型が粒子の位置等、粒子計算に必須の物理量を持つことを仮定する。したがって、ユーザは FDPS に必須物理量がユーザ定義型のどのメンバ変数に対応するかを教える必要がある。また、FDPS 内部では、ユーザ定義型の間でデータのやりとりを行うが、それをユーザが指定した方法で行うことになる。したがって、ユーザはその方法をコードに書く必要がある。これら FDPS への指示や方法の記述はすべてコード内に特別な指示文 (**FDPS 指示文**) を記述することによって行う。以下、まずはじめにユーザ定義型について記述し、その後ユーザ定義関数について記述する。

5.1　ユーザ定義型

まず概要を述べる。FullParticle 型は、ある 1 粒子の情報すべてを持つ派生データ型 (Fortran) 或いは 構造体 (C 言語) であり、粒子群クラスのオブジェクトの生成に使用されるものである (第 2 章 2.3 節の手順 0)。EssentialParticleI 型、EssentialParticleJ 型、Force 型は粒子間の相互作用の定義を補助するものであり、それぞれ、相互作用を計算する際に i 粒子に必要な情報、相互作用を計算する際に j 粒子に必要な情報、相互作用の結果の情報を持つ派生データ型 (Fortran) 或いは 構造体 (C 言語) である。これらは FullParticle 型のサブセットであるため、これらを FullParticle 型で代用することも可能である。しかし、FullParticle 型は相互作用の定義に必要のないデータを多く含む場合も考えられるため、計算コストを軽減したいならば、これらの型を使用することを検討するべきである。

以下では、はじめにユーザ定義型を記述する上での共通の規則について記述する。その後、FullParticle 型、EssentialParticleI 型、EssentialParticleJ 型、Force 型の順で記述する。

5.1.1　共通規則

5.1.1.1　Fortran 文法に関する要請

本節では、ユーザ定義型となるために派生データ型が満たすべき最低限の Fortran 文法について記述する。第 3 章で述べた通り、本 FDPS Fortran インターフェースは、FDPS の C

言語インターフェースを通して、FDPS 本体とデータをやり取りする。このため、すべてのユーザ定義型は C 言語と (Fortran 2003 標準で)**interoperable** である必要がある。具体的には、ユーザ定義型となる派生データ型は次の条件を満たしている必要がある：

(1) 派生データ型は `bind(c)` 属性を持たなければならない。
(2) すべてのメンバ変数が interoperable なデータ型である。Fortran 2003 標準 (ISO/IEC 1539-1:2004(E)) で定義される「C 言語と interoperable な」データ型の一覧は本書表 8.1 や言語仕様書の第 15 節「Interoperability with C」で確認できる[注 1)]ほか、GCC Wiki のページ GFortranStandards で紹介されている各種非公式文書 (ドラフト段階の言語仕様書) や GNU gfortran のオンラインドキュメントでも解説されている。「C 言語と interoperable」な派生データ型をメンバ変数として持つことは可能である。
(3) すべてのメンバ変数は `allocatable` 属性を持たない。
(4) すべてのメンバ変数は `pointer` 属性を持たない。
(5) メンバ関数を持たない。

加えて、FDPS 側からの要請として、次の条件を満たす必要がある：

(6) 派生データ型はモジュール内で定義されている。
(7) 派生データ型は `public` 属性を持つ。
(8) メンバ変数として持たせることが可能な派生データ型はベクトル型と対称行列型 (第 4 章参照) のみである。
(9) 派生データ型は多次元配列をメンバ変数として持てない (これは将来のバージョンにおいて対応する予定である)。
(10) メンバ変数の (1 次元) 配列の形状を指定する場合、`dimension` 文で指定するか、変数名に (配列要素数) を付けるかの、どちらか片方の方法でなければならない。

以上の条件が、派生データ型がユーザ定義型となるために満たす必要がある Fortran 文法である。これに加え、次節 5.1.1.3 で説明する FDPS 指示文によって、どのユーザ定義型 (FullParticle 型,EssentialParticleI 型,EssentialParticleJ 型,Force 型) に対応するかや、必須物理量がどのメンバ変数に対応しているか等を指定してはじめてユーザ定義型となる。

5.1.1.2 C 言語 文法に関する要請

本節では、ユーザ定義型となるために構造体が満たすべき最低限の C 言語文法について記述する。第 3 章で述べた通り、C 言語インターフェースプログラムの 1 つ `FDPS_ftn_if.cpp` は、Fortran インターフェースでも共用される。このため、自由に構造体を定義できるわけではなく、前節 5.1.1.1 で述べたような制限を受ける。具体的には、ユーザ定義型となる構造体は次の条件を満たさなければならない：

(1) 構造体はタグ名を持つ必要がある。タグ名はすべて小文字でなければならない。

注 1) Fortran の言語仕様書を販売している ISO (International Organization for Standardization) からは Fortran 2008 Standard (ISO/IEC 1539-1:2010(E)) のみ購入可能である。

(2) メンバ変数のデータ型として利用できるのは、(i) Fortran 2003 標準 (ISO/IEC 1539-1:2004(E)) と相互運用可能なデータ型 (詳細は第 5.1.1.1 節の (2)、及び本書表 8.1 を参照のこと)、(ii) ベクトル型、(iii) 対称行列型のみである。**特に、符号なし整数、及び、あらゆるポインタは持てないことに注意して頂きたい。**
(3) 構造体は多次元配列をメンバ変数として持てない (これは将来のバージョンにおいて対応する予定である)。
(4) メンバ変数名はすべて小文字でなければならない。

Fortran の場合と同様、これに加え、次節 5.1.1.3 で説明する FDPS 指示文を付け加えて、はじめてユーザ定義型の資格を得る。

5.1.1.3 FDPS 指示文 (共通項目のみ)

本節では、すべてのユーザ定義型に共通して使用可能な FDPS 指示文の概要と記述方法について解説する。各ユーザ定義型に固有の指示文に関しては、第 5.1.2〜5.1.5 節で解説する。

FDPS 指示文には以下の 3 つの種類がある：

(a) 派生データ型/構造体がどのユーザ定義型に対応するかを指定する指示文。
(b) 派生データ型/構造体のメンバ変数がどの必須物理量に対応するかを指定する指示文。
(c) 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。

これらの指示文の内、以下で、最初の 2 つ (a),(b) について解説する。以降の解説では、まず、Fortran における FDPS 指示文の書き方について説明し、その後、C 言語の場合を説明する。

5.1.1.3.1 ユーザ定義型の種別を指定する FDPS 指示文

派生データ型 *type_name* がどのユーザ定義型に対応するかを指定するには、次の書式の指示文を記述する：

```
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]
```

ここで [] は、その中身が省略可能であることを示す記号である。FDPS 指示文は必ず文字列`!$fdps` で開始される。英字はすべて小文字でなければならない。!で始まることからわかるように、FDPS 指示文は単なるコメント文であり、Fortran プログラムの動作に影響を与えるものではない。インターフェース生成スクリプトだけが、このコメント文を指示文として

解釈する。!$fdps に半角スペースを置いて続く *keyword* は、ユーザ定義型を指定するための文字列である。可能なキーワードは、FP, EPI, EPJ, Force であり、大文字・小文字を区別する。それぞれ FullParticle 型、EssentialParticleI 型、EssentialParticleJ 型、Force 型に対応している。FDPS 指示文は派生データ型名の右側か、1 つ前の行に記述しなければならない。FDPS 指示文の中で改行を行うことはできない。第 5.1 節で述べた通り、EssentialParticleI 型、EssentialParticleJ 型、Force 型は FullParticle 型のサブセットでり、FullParticle 型がこれら 3 つを兼ねることが可能である。その場合には、以下のリスト 5.1 に示されるように、キーワードをカンマで区切って並べればよい：

Listing 5.1: FullParticle 型が他を兼ねる場合の例

```
1 type, public, bind(c) :: full_particle !$fdps FP,EPI,EPJ,Force
2 end type full_particle
```

FullParticle 型が EssentialParticleI 型だけを兼ねるといったことも可能である。

構造体 *tag_name* がどのユーザ定義型に対応するかを指定するには、次の書式の指示文を記述する：

```
struct tag_name { //$fdps keyword
};
```

或いは、

```
//$fdps keyword
struct tag_name {
};
```

指示文の書き方は Fortran の場合とほぼ同じである。違いのみ以下に示す。

- コメント文からコメント記号と先頭の空白文字をすべて取り除いたときに得られる文字列が、文字列$fdps で開始される場合 (後続の文字列と空白文字で区切られている必要がある)、そのコメント文は指示文として解釈される。上の例では、コメント記号//を使っているが、別のコメント記号/*,*/を使って、/* $fdps */のように記述してもよい。
- 指示文 (a) と解釈されるのは、struct の直前の指示文、或いは、*tag_name* の後の最初の指示文である。指示文はどちらか一方の位置に記述しなければならない。

5.1.1.3.2 必須物理量を指定する FDPS 指示文

次に、必須物理量に対応するメンバ変数を指定する指示文 (b) について解説する。FDPS では必須物理量として、粒子の電荷量 (質量)、粒子の位置が必要である。また、ある種の粒子シミュレーションでは探索半径も必要となる。派生データ型 *type_name* のメンバ変数 *mbr_name* がどの必須物理量に対応するかを指定するには、次の書式の指示文を記述する：

```
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]
```

ここでは、見やすさのため、ユーザ定義型の種別を指定する指示文は省略している。指示文は、指示文の開始を示す文字列!$fdps で始まり、半角スペースを置いて *keyword* が続く。可能なキーワードは、id、charge、position、rsearch、velocity である[注 2)]。それぞれ、粒子の識別番号、粒子の電荷量 (質量)、粒子の位置、粒子の探索半径、粒子の速度に対応している。キーワードはすべて小文字でなければならない。また、1 つのメンバ変数に対して 1 つの指示文を対応させなければならない。指示文は、変数名の右側か 1 つ前の行に記述しなければならない。

メンバ変数のデータ型 *data_type* は、対応する必須物理量が持つべきデータ型に一致していなければならない。以下に、各必須物理量が持つべきデータ型をまとめる:

物理量名	可能なデータ型
粒子の識別番号	integer(kind=c_long_long)
電荷 (質量) および探索半径	real(kind=c_float) real(kind=c_double)
位置および速度	type(fdps_f32vec) real(kind=c_float), dimension(space_dim)† type(fdps_f64vec) real(kind=c_double), dimension(space_dim)†

† space_dim は空間次元を表す。コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合は 2、それ以外は 3 である必要がある (第 7 章参照)。

表 5.1: 各必須物理量が持つべきデータ型。

構造体 *tag_name* のメンバ変数 *mbr_name* がどの必須物理量に対応するかを指定するには、次の書式の指示文を記述する：

[注 2)] 但し、velocity は予約語であり、現時点で生成されるインターフェースプログラムの内容に影響しない

```
struct tag_name {
   data_type mbr_name; //$fdps keyword
};
```

或いは、

```
struct tag_name {
   //$fdps keyword
   data_type mbr_name;
};
```

指示文の書き方は Fortran の場合と同じである。各必須物理量が持つべきデータ型は、上の表に記載された Fortran のデータ型を C 言語のデータ型に適切に読み替えることで得られる (表 8.1 を参照のこと)。

5.1.1.3.3 FDPS 指示文の記述例

最後に、FullParticle 型の実装例を示す (リスト 5.2 および 5.3)。この例ではここで説明しなかった FDPS 指示文 (c) が使用されているが、FDPS 指示文 (a),(b) がどのように使用されているかに注意してほしい。

Listing 5.2: ユーザ定義型の例 (Fortran)

```
1  module user_defined_types
2     use, intrinsic :: iso_c_binding
3     use :: fdps_vector
4     implicit none
5
6     !**** Full particle type
7     type, public, bind(c) :: full_particle !$fdps FP,EPI,EPJ,Force
8        !$fdps copyFromForce full_particle (pot,pot) (acc,acc)
9        !$fdps copyFromFP full_particle (id,id) (mass,mass) (eps,eps) (pos,
               pos)
10       integer(kind=c_long_long) :: id !$fdps id
11       real(kind=c_double) :: mass !$fdps charge
12       real(kind=c_double) :: eps
13       type(fdps_f64vec) :: pos !$fdps position
14       type(fdps_f64vec) :: vel !$fdps velocity
15       real(kind=c_double) :: pot
16       type(fdps_f64vec) :: acc
17    end type full_particle
18
19 end module user_defined
```

Listing 5.3: ユーザ定義型の例 (C 言語)

```
1 #include "FDPS_c_if.h"
2
3 //**** Full particle type
```

```
4  typedef struct full_particle { //$fdps FP,EPI,EPJ,Force
5     //$fdps copyFromForce full_particle (pot,pot) (acc,acc)
6     //$fdps copyFromFP full_particle (id,id) (mass,mass) (eps,eps) (pos,pos
          )
7     long long int id; //$fdps id
8     double mass; //$fdps charge
9     double eps;
10    fdps_f64vec pos; //$fdps position
11    fdps_f64vec vel; //$fdps velocity
12    double pot;
13    fdps_f64vec acc;
14 } Full_particle;
```

以下では、各ユーザ定義型に個別の指示文も含めて、記述の規則を解説していく。

5.1.2 FullParticle 型

FullParticle 型は粒子情報すべてを持つ派生データ型 (Fortran) 或いは 構造体 (C 言語) であり、第 2 章 2.3 節の手順 0 に対応して、粒子群オブジェクトを生成するのに必要なユーザー定義型である。ユーザーはこの派生データ型/構造体に対して、どのようなメンバ変数を定義してもかまわない。ただし、ユーザーは、FDPS 指示文を用いて、必須物理量に対応するメンバ変数と、FullParticle 型と他のユーザ定義型の間のデータ移動の方法を記述する必要がある。以下、常に必要な FDPS 指示文と、場合によっては必要な FDPS 指示文について記述する。

5.1.2.1 常に必要な FDPS 指示文とその記述法

常に必要な FDPS 指示文は、以下である：

- 粒子の電荷量 (質量) に対応するメンバ変数を指定する指示文
- 粒子の位置に対応するメンバ変数を指定する指示文
- 計算された相互作用の結果を Force 型から FullParticle 型に書き戻す方法を指定する指示文

最初の 2 つに関しては、第 5.1.1.3 節で説明した方法で記述すればよい。最後のものは、Fortran では、次の書式で指示文を記述する必要がある：

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

FDPS 指示文は文字列`!$fdps`で開始される。その後、1 個以上の半角スペースを挟み、キーワード`copyFromForce`を記述する。このキーワードによって、この FDPS 指示文が Force 型から FullParticle 型へのデータコピーの仕方を記述する指示文であるとみなされる。キーワード`copyFromForce`の後には、Force 型に対応する派生データ型名 *force* を記述する。キーワードとの間には 1 個以上の半角スペースが必要である。続いて、1 個以上の変数ペア (*src_*

mbr,*dst_mbr*) が半角スペースを区切り文字として並ぶ。これは Force 型のどのメンバ変数を FullParticle 型のどのメンバ変数にコピーするかを示している。*src_mbr* が Force 型のメンバ変数であり、*dst_mbr* が FullParticle 型のメンバ変数である。FDPS 指示文は途中で改行することはできない。

C 言語では、次の書式で指示文を記述する必要がある：

```
struct fp {
   //$fdps copyFromForce force (src_mbr,dst_mbr) (src_mbr,dst_mbr) ...
}
```

記述方法は Fortran の場合と同じである。

粒子シミュレーションによっては、1 つの FullParticle 型に対し、複数種の相互作用を定義する必要がある場合が想定される。その場合には、各々の Force 型に対して、この FDPS 指示文を記述する必要がある。

本指示文の記述例がリスト 5.2 及び 5.3 に示されているので、そちらも参照されたい。

5.1.2.2 場合によっては必要な FDPS 指示文とその記述法

本節では、以下に示す場合に必要となる指示文について記述する：

(i) 次の種別の相互作用ツリーオブジェクトを使用する場合：

- Long-MonopoleWithScatterSearch 型
- Long-QuadrupoleWithScatterSearch 型
- Long-MonopoleWithSymmetrySearch 型
- Long-QuadrupoleWithSymmetrySearch 型
- Long-MonopoleWithCutoff 型
- Short 型に分類されるすべてのツリー

(ii) 拡張機能 Particle Mesh を用いる場合

(iii) FullParticle 型が他のユーザ定義型を兼ねる場合

(i) の場合、ユーザは派生データ型 或いは 構造体のメンバ変数のどれが探索半径であるかを指定しなければならない (相互作用ツリーの種別に関しては、第 8 章で解説する)。これは、第 5.1.1.3 節で説明した方法で記述すればよい。

(ii) の場合には、ユーザは FDPS の Particle Mesh モジュールで計算された力を FullParticle 型に書き戻す方法を指示する必要がある。これは、Fortran の場合、次の書式の FDPS 指示文を使って指定する：

```
type, public, bind(c) :: FP
   !$fdps copyFromForcePM mbr_name
end type FP
```

FDPS 指示文は文字列`!$fdps` で開始される。その後、1 個以上の半角スペースを挟み、キー

ワード copyFromForcePM が続く。これによって、この指示文が Particle Mesh モジュールから FullParticle 型への力のコピーの仕方を指定する指示文であると解釈される。キーワードの次に 1 個以上の半角スペースをおいて、コピー先である FullParticle 型のメンバ変数名 *mbr_name* が続く。コピー先のメンバ変数は第 4 章で説明したベクトル型でなければならない。FDPS 指示文は途中で改行することはできない。

C 言語の場合には、指示文は次の書式で記述する必要がある：

```
struct FP {
   //$fdps copyFromForcePM mbr_name
};
```

Fortran と全く同じ書式なので説明は省略する。

(iii) の場合には、他のユーザ定義型で常に必要となる FDPS 指示文のすべてと、場合によっては必要となる FDPS 指示文を必要なだけ記述する必要がある。これらに関しては、対応するユーザ定義型の節を参照して頂きたい。

5.1.3 EssentialParticleI 型

EssentialParticleI 型は相互作用の計算に必要な i 粒子の情報を持つ派生データ型 (Fortran) 或いは 構造体 (C 言語) であり、相互作用関数 (ユーザ定義関数) の定義に必要となるほか、相互作用ツリーオブジェクトの生成に必要となる。EssentialParticleI 型は FullParticle 型 (第 5.1.2 節) のサブセットである。ユーザは FDPS 指示文を用いて、必須物理量に対応するメンバ変数と、FullParticle 型との間のデータ移動の方法を記述する必要がある。以下、常に必要な FDPS 指示文と、場合によっては必要な FDPS 指示文について記述する。

5.1.3.1 常に必要な FDPS 指示文とその記述法

常に必要となる FDPS 指示文は、以下である：

- 粒子の電荷量 (質量) に対応するメンバ変数を指定する指示文
- 粒子の位置に対応するメンバ変数を指定する指示文
- FullParticle 型から相互作用計算に必要な粒子データをコピーするための方法を指定する指示文

最初の 2 つに関しては、第 5.1.1.3 節で説明した方法で記述すればよい。最後のものは、Fortran の場合、次の書式で指示文を記述する必要がある：

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

書式は、(i) !$fdps に続く文字列が copyFromFP である点、(ii) *fp* がコピー元となる FullParticle 型の派生データ型名である点、の 2 つ除き、第 5.1.2.1 節に記述した copyFromForce 指

示文と同じである。この場合、*src_mbr* が FullParticle 型のメンバ変数名であることに注意されたい。

C 言語の場合、次の書式で記述する必要がある：

```
struct epi {
   //$fdps copyFromFP fp (src_mbr,dst_mbr) (src_mbr,dst_mbr) ...
};
```

Fortran と全く同じ書式なので説明は省略する。

5.1.3.2 場合によっては必要な FDPS 指示文とその記述法

本節では、以下に示す場合に必要となる指示文について記述する：

(i) 次の種別の相互作用ツリーオブジェクトを使用する場合：

- Long-MonopoleWithSymmetrySearch 型
- Long-QuadrupoleWithSymmetrySearch 型
- Short-Gather 型
- Short-Symmetry 型

(ii) EssentialParticleI 型が他のユーザ定義型を兼ねる場合

(i) の場合、ユーザは派生データ型 (Fortran) 或いは 構造体 (C 言語) のメンバ変数のどれが探索半径であるかを指定しなければならない (相互作用ツリーの種別に関しては、第 8 章で解説する)。これは、第 5.1.1.3 節で説明した方法で記述すればよい。

(ii) の場合には、他のユーザ定義型で常に必要となる FDPS 指示文のすべてと、場合によっては必要となる FDPS 指示文を必要なだけ記述する必要がある。これらに関しては、対応するユーザ定義型の節を参照して頂きたい。

5.1.4 EssentialParticleJ 型

EssentialParticleJ 型は相互作用の計算に必要な j 粒子の情報を持つ派生データ型 (Fortran) 或いは 構造体 (C 言語) であり、相互作用関数 (ユーザ定義関数) の定義に必要となるほか、相互作用ツリーオブジェクトの生成に必要となる。EssentialParticleJ 型は FullParticle 型 (第 5.1.2 節) のサブセットである。ユーザは FDPS 指示文を用いて、必須物理量に対応するメンバ変数と、FullParticle 型との間のデータ移動の方法を記述する必要がある。以下、常に必要な FDPS 指示文と、場合によっては必要な FDPS 指示文について記述する。

5.1.4.1 常に必要な FDPS 指示文とその記述法

常に必要となる FDPS 指示文は、以下である：

- 粒子の電荷量 (質量) に対応するメンバ変数を指定する指示文
- 粒子の位置に対応するメンバ変数を指定する指示文
- FullParticle 型から相互作用計算に必要な粒子データをコピーするための方法を指定する指示文

最初の 2 つに関しては、第 5.1.1.3 節で説明した方法で記述すればよい。最後のものは、第 5.1.3.1 節で説明した `copyFromFP` 指示文を記述すればよい。

5.1.4.2 場合によっては必要な FDPS 指示文とその記述法

本節では、以下に示す場合に必要となる指示文について記述する：

(i) 次の種別の相互作用ツリーオブジェクトを使用する場合：

- Long-MonopoleWithScatterSearch 型
- Long-QuadrupoleWithScatterSearch 型
- Long-MonopoleWithSymmetrySearch 型
- Long-QuadrupoleWithSymmetrySearch 型
- Long-MonopoleWithCutoff 型
- Short 型に分類されるすべてのツリー

(ii) EssentialParticleJ 型が他のユーザ定義型を兼ねる場合

(iii) 粒子の識別番号から対応する EPJ を取得したい場合 (API `(fdps_)get_epj_from_id` を使用する場合; 接頭辞 `fdps_`がつくのは C 言語用 API のみ)

(i) の場合、ユーザは派生データ型/構造体のメンバ変数のどれが探索半径であるかを指定しなければならない (相互作用ツリーの種別に関しては、第 8 章で解説する)。これは、第 5.1.1.3 節で説明した方法で記述すればよい。

(ii) の場合には、他のユーザ定義型で常に必要となる FDPS 指示文のすべてと、場合によっては必要となる FDPS 指示文を必要なだけ記述する必要がある。これらに関しては、対応するユーザ定義型の節を参照して頂きたい。

(iii) の場合には、ユーザは派生データ型/構造体のメンバ変数のどれが粒子の識別番号であるかを指定しなければならない。これは、第 5.1.1.3 節で説明した方法で記述すればよい。

5.1.5 Force 型

Force 型は相互作用の結果を保持する派生データ型 (Fortran) 或いは 構造体 (C 言語) であり、相互作用関数の定義に必要となるほか、相互作用ツリーオブジェクトの生成に必要となる。以下、常に必要な FDPS 指示文と、場合によっては必要な FDPS 指示文について記述する。

5.1.5.1 常に必要な FDPS 指示文とその記述法

常に必要な FDPS 指示文は、相互作用の計算結果を初期化する方法を指示する指示文である。この指示文の書式は、初期化の仕方に応じて3通りある。ユーザはいずれか 1 つの方法で初期化を指示しなければならない。以下、各書式について解説する。

(1) すべてのメンバ変数をデフォルト初期化する場合

Force 型のすべてのメンバ変数に対して、デフォルトの初期化を行う場合には、何も記述しない。ここで、デフォルトの初期化とは、整数と浮動小数点数は 0 に、論理型は.false.(Fortran)/false(C 言語) に、第 4 章のベクトル型と対称行列型はその各成分を 0 にする初期化のことである。

(2) メンバ変数の初期化を個別に指定したい場合

Force 型のメンバ変数を個別に、ある決まった値に初期化したい場合には、Fortran では、以下のように記述する:

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

ここで、Force は Force 型の派生データ型名である。見やすさのため、この派生データ型が Force 型であることを示す指示文は省略していることに注意されたい。文字列!$fdps が指示文の開始を示す。その後、1 個以上の半角スペースを挟み、キーワード clear が続く。このキーワードによって、この FDPS 指示文が Force 型の初期化の方法を指示する文であるとみなされる。キーワード clear の後の [] はその中身が省略可能であることを示す記号であり、実際には [] を記述してはならないことに注意して頂きたい。

個別指定の内容はキーワード clear の後に記述する。個別指定のないメンバ変数には自動的にデフォルト初期化が適用される。個別指定の方法は 2 種類あり、以下でそれを説明する。

まず、特定のメンバ変数 *mbr* を特定の値 *val* に初期化したい場合には、*mbr*=*val* と記述する。ここで、記号=の前後に 0 個以上の半角スペースを入れることが可能である。初期値はメンバ変数の型と矛盾してはならず、Fortran の言語仕様に従って記述されなければならない。例えば、メンバ変数が論理型の場合には.true. か.false. のいずれかでなければならない。メンバ変数がベクトル型や対称行列型の場合、全成分を同じ値に初期化する初期化だけが指定可能であり、*val* はスカラー値である必要がある。各成分を異なる値に初期化したい場合には、次項の初期化方法を使用して頂きたい。

次に、特定のメンバ変数 *mbr* を初期化したくない場合には、*mbr*=keep と記述する。右辺の keep が初期化しないことを指示するキーワードである。同様、記号=の前後に 0 個以上の半角スペースを入れることが可能である。

複数の個別指定を並べることが可能で、その場合には、それらをカンマで区切って並べる。

C 言語では、次の書式で記述する：

```
struct force {
   //$fdps clear [mbr=val , mbr=keep, ...]
};
```

Fortran と全く同じ書式なので説明は省略する。

(3) **複雑な初期化を行いたい場合**

より複雑な初期化を行いたい場合には、初期化を Fortran ではサブルーチン、C 言語では関数、を用いて行うことができる。この場合には、Fortran では、以下のように記述する:

```
type, public, bind(c) :: Force
   !$fdps clear subroutine subroutine_name
end type Force
```

ここで、*subroutine_name* は初期化に使用するサブルーチン名である。このサブルーチンはグローバル領域に定義されていなければならない。言い換えれば、Fortran のモジュール内や他のサブルーチンや関数の内部手続として定義されてはならない。初期化を行うサブルーチンは以下のインターフェースを持たなければならない:

```
subroutine subroutine_name(f) bind(c)
   use, intrinsic :: iso_c_binding
   implicit none
   type(Force), intent(inout) :: f

   ! Initialize Force

end subroutine [subroutine_name]
```

ここで、[] はその中身が省略可能であることを示す記号である。

C 言語では、次の書式で記述する：

```
struct force {
   //$fdps clear function function_name
};
```

書式は Fortran の場合とほぼ同じである。初期化に使用する関数はグローバル領域に定義されていなければならない。初期化を行う関数は以下のインターフェースを持つ必要がある：

```
void function_name(struct force *f) {

   // Initialize Force

}
```

5.1.5.2 場合によっては必要な FDPS 指示文とその記述法

なし

5.2 ユーザ定義関数

まず概要を述べる。関数 calcForceEpEp と calcForceEpSp は、それぞれ j 粒子から i 粒子への作用を計算する関数と超粒子から i 粒子への作用を計算する関数である。これらの関数ポインタは、相互作用ツリー用の API の引数として渡される。相互作用が短距離力の場合には超粒子を必要としない。その場合、関数 calcForceEpSp を定義する必要はない。

5.2.1 共通規則

まず最初に Fortran で関数 calcForceEpEp と calcForceEpSp を定義するための規則について説明を行い、その後、C 言語での規則について説明を行う。

5.2.1.1 Fortran 文法に関する要請 および FDPS 本体の仕様による要請

本節では、Fortran のサブルーチンがユーザ定義関数となるために満たすべき最低限の Fortran 文法について記述する。これを説明するため、FDPS Fortran インターフェースを使って相互作用計算を行うまでにユーザが踏むべき手順について述べる。インターフェースプログラムの生成が成功したと仮定すると、次のようになる：

(I) 相互作用計算の内容を Fortran サブルーチンとして実装する。
(II) ユーザプログラムにおいて、関数の C 言語アドレスを格納するための変数を用意する。これは Fortran 2003 のモジュール `iso_c_binding` で提供される派生データ型 `type(c_funloc)` の変数を用意すればよい。
(III) ユーザプログラムにおいて、相互作用計算に使用する Fortran サブルーチンの C 言語アドレスを、モジュール `iso_c_binding` で提供される関数 `c_funloc` によって取得し、前項の変数に代入する。
(IV) FDPS API の引数に関数ポインタが格納された変数を渡して、API を呼び出す。
(V) FDPS 本体は受け取った C 言語アドレスを C 言語で記述された関数と理解して、実行する。

上記手順の (III) において、関数 c_funloc でユーザ定義関数の C 言語アドレスを取得するためには、ユーザ定義関数が C 言語と interoperable でなければならない。具体的には、ユーザ定義関数となる Fortran サブルーチンは次の条件を満たしている必要がある：

(1) 関数は bind(c) 属性を持たなければならない。
(2) すべての仮引数が interoperable なデータ型である。interoperable なデータ型に関しては、第 5.1.1 節の記述を参照されたい。

上に述べた条件に加え、FDPS 本体の仕様に起因する条件がある。それは以下の条件である：

(3) ユーザ定義関数の仮引数の内、i 粒子と j 粒子の粒子数に対応する仮引数は value 属性が付いていなければならない。value 属性は、いわゆる値渡しであることを指示するものである。

以上がユーザ定義関数が満たすべき最低限の条件である。理解を助ける目的で、N 体計算のサンプルコードの粒子-粒子相互作用に対応したユーザ定義関数の例をリスト 5.4 に示しておく。相互作用関数の記述方法の詳細はまだ解説していないので、ここでは、bind(c) 属性と value 属性の位置だけを確認して頂きたい。

Listing 5.4: 粒子-粒子相互作用に対応したユーザ定義関数の実装例

```
1  subroutine calc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2     integer(c_int), intent(in), value :: n_ip,n_jp
3     type(full_particle), dimension(n_ip), intent(in) :: ep_i
4     type(full_particle), dimension(n_jp), intent(in) :: ep_j
5     type(full_particle), dimension(n_ip), intent(inout) :: f
6     !* Local variables
7     integer(c_int) :: i,j
8     real(c_double) :: eps2,poti,r3_inv,r_inv
9     type(fdps_f64vec) :: xi,ai,rij
10
11    do i=1,n_ip
12       eps2 = ep_i(i)%eps * ep_i(i)%eps
13       xi%x = ep_i(i)%pos%x
14       xi%y = ep_i(i)%pos%y
15       xi%z = ep_i(i)%pos%z
16       ai%x = 0.0d0
17       ai%y = 0.0d0
18       ai%z = 0.0d0
19       poti = 0.0d0
20       do j=1,n_jp
21          rij%x  = xi%x - ep_j(j)%pos%x
22          rij%y  = xi%y - ep_j(j)%pos%y
23          rij%z  = xi%z - ep_j(j)%pos%z
24          r3_inv = rij%x*rij%x &
25                 + rij%y*rij%y &
26                 + rij%z*rij%z &
27                 + eps2
28          r_inv  = 1.0d0/sqrt(r3_inv)
29          r3_inv = r_inv * r_inv
30          r_inv  = r_inv * ep_j(j)%mass
```

```
31            r3_inv = r3_inv * r_inv
32            ai%x   = ai%x - r3_inv * rij%x
33            ai%y   = ai%y - r3_inv * rij%y
34            ai%z   = ai%z - r3_inv * rij%z
35            poti   = poti - r_inv
36         end do
37         f(i)%pot   = f(i)%pot   + poti
38         f(i)%acc%x = f(i)%acc%x + ai%x
39         f(i)%acc%y = f(i)%acc%y + ai%y
40         f(i)%acc%z = f(i)%acc%z + ai%z
41      end do
42
43  end subroutine calc_gravity_pp
```

5.2.1.2 C 言語 文法に関する要請 および FDPS 本体の仕様による要請

Fortran の場合と異なり (第 5.2.1.1 節参照)、C 言語から FDPS を利用する場合、ユーザは任意の関数の C 言語アドレスを自由に取得することが可能である。そのため、void 関数を使ってユーザ定義関数を実装する際、C 言語文法的な制限は存在しない。しかしながら、FDPS 本体ではユーザ定義関数の仮引数仕様が決まっているため、C 言語でもそれを満たす必要がある。これについては次節以降に説明する。

5.2.2 関数 calcForceEpEp

関数 calcForceEpEp は粒子同士の相互作用を記述するものであり、相互作用の定義に必要となる。関数 calcForceEpEp は、以下の書式で記述しなければならない。

Fortran 書式

```
subroutine calc_force_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
   use, intrinsic :: iso_c_binding
   implicit none
   integer(kind=c_int), intent(in), value :: n_ip,n_jp
   type(essential_particle_i), dimension(n_ip), intent(in) :: ep_i
   type(essential_particle_j), dimension(n_jp), intent(in) :: ep_j
   type(force), dimension(n_ip), intent(inout) :: f

end subroutine calc_force_ep_ep
```

C 言語 書式

```
void calc_force_ep_ep(struct essential_particle_i *ep_i,
                      int n_ip,
                      struct essential_particle_j *ep_j,
                      int n_jp,
                      force *f) {

}
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
n_ip	integer(kind=c_int) または int	入力	i 粒子の粒子数を格納した変数。
n_jp	integer(kind=c_int) または int	入力	j 粒子の粒子数を格納した変数。
ep_i	essential_particle_i 型 †	入力	i 粒子情報を持つ配列。
ep_j	essential_particle_j 型 †	入力	j 粒子情報を持つ配列。
f	force 型 †	入出力	i 粒子の相互作用結果を返す配列。

‡ それぞれ EssentialParticleI 型、EssentialParticleJ 型、Force 型の派生データ型名 (Fortran) または 構造体名 (C 言語) である。Fortran においては、もしこれらが本サブルーチンと別なモジュールで定義されている場合には、そのモジュールを use する必要がある点に注意されたい。同様に、C 言語の場合でも必要なヘッダーファイルをインクルードする必要がある。

返り値

なし

機能

j 粒子から i 粒子への作用を計算する。

5.2.3 関数 calcForceEpSp

関数 calcForceEpSp は超粒子から粒子への作用を記述するものであり、相互作用の定義に必要となる。関数 calcForceEpEp は以下の書式で記述しなければならない。

Fortran 書式

```
subroutine calc_force_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
   use, intrinsic :: iso_c_binding
   use :: fdps_super_particle
   implicit none
   integer(kind=c_int), intent(in), value :: n_ip,n_jp
   type(essential_particle_i), dimension(n_ip), intent(in) :: ep_i
   type(super_particle_j), dimension(n_jp), intent(in) :: ep_j
   type(force), dimension(n_ip), intent(inout) :: f

end subroutine calc_force_ep_sp
```

C 言語 書式

```
void calc_force_ep_sp(struct essential_particle_i *ep_i,
                      int n_ip,
                      struct super_particle_j *ep_j,
                      int n_jp,
                      force *f) {

}
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
n_ip	integer(kind=c_int) または int	入力	i 粒子の粒子数を格納した変数。
n_jp	integer(kind=c_int) または int	入力	超粒子の粒子数を格納した変数。
ep_i	essential_particle_i 型 †	入力	i 粒子情報を持つ配列。
ep_j	super_particle_j 型 ‡	入力	超粒子情報を持つ配列。
f	force 型 †	入出力	i 粒子の相互作用結果を返す配列。

† それぞれ EssentialParticleI 型と Force 型の派生データ型名 (Fortran) または 構造体名 (C 言語) である。Fortran では、これらが本サブルーチンと別なモジュールで定義されている場合には、そのモジュールを use する必要がある点に注意されたい。同様に、C 言語でも必要なヘッダーファイルをインクルードする必要がある。
‡ 第 4 章 4.4 節で定義されるいずれかの超粒子型でなければならない。

返り値

なし

機能

超粒子から i 粒子への作用を計算する。

第6章　Fortran/C言語 インターフェースの生成

本章では、FDPS Fortran/C 言語 インターフェース生成スクリプトの動作条件と使用方法について記述する。

6.1　スクリプトの動作条件

本節では、インターフェース生成スクリプト `gen_ftn_if.py` (Fortran インターフェース生成用) 及び `gen_c_if.py` (C 言語インターフェス生成用) の動作条件について記述する。2 つのインターフェース生成スクリプトはディレクトリ `scripts` の直下に配置されている。本スクリプトは、プログラミング言語 Python で実装されており、正常な動作のためには、Python 2.7.5 以上、或いは、Python 3.4 以上が必要である。ユーザの環境に合わせて、スクリプト第 1 行目の

```
#!/usr/bin/env python
```

の部分を適宜修正して使用されたい。ここで、確認すべき点は `env` コマンドの PATH と Python インタープリタの名称である (利用する計算機システムによっては `python` が存在せず、`python2.7` や `python3.4` のように名称にバージョン名が付いたもののみが用意されている場合がある)。もし Python インタープリタに PATH が通っていない場合には、PATH を通すか、以下のように、絶対 PATH で Python インタープリタを指定する:

```
#!/path/to/python
```

上記に加え、正常な動作のためには、以下の条件を満たす必要がある:

- スクリプト `gen_ftn_if.py` に入力されるすべての Fortran コードが Fortran 2003 標準 (ISO/IEC 1539-1:2004(E)) の文法に従って記述されていること。本スクリプトに言語の自動判別機能や詳細な文法チェック機能は実装されておらず、誤った文法が使用された場合の動作は不定である。
- スクリプト `gen_c_if.py` に入力されるすべての C 言語コードは C99 (ISO/IEC 9899:1999(E))、または、それよりも新しい規格に従って記述されていること。本スクリプトに言語の自動判別機能や詳細な文法チェック機能は実装されておらず、誤った文法が使用された場合の動作は不定である。

6.2 スクリプトの使用方法

本スクリプトを使ってインターフェースプログラムを生成するためには、コマンドライン上で、以下のようにしてスクリプトを実行すればよい。Fortran インターフェースを生成する場合には

```
$ gen_ftn_if.py -o output_directory user1.F90 user2.F90...
```

C 言語インターフェースを生成する場合には

```
$ gen_c_if.py -o output_directory user1.h user2.h...
```

ここで、環境変数 PATH にディレクトリ scripts が追加されていると仮定している。PATH を通さずにスクリプトを使用する場合には、絶対 PATH か相対 PATH でスクリプトを実行する必要がある。スクリプトの引数には、ユーザ定義型が記述された Fortran ファイル (gen_ftn_if.py を使用する場合) 或いは C 言語ヘッダーフェイル (gen_c_if.py を使用する場合) を指定する。複数のファイルを指定する場合には、1 個以上の半角スペースを空けて、ファイル名を並べる。この際、並べる順番は任意でよい。

オプション「-o」でインターフェースプログラムを出力するディレクトリを指定することができる。オプション「-o」の代わりに、「--output」あるいは「--output_dir」を使用することもできる。指定がない場合には、カレントディレクトリに出力される。

オプション「-DPARTICLE_SIMULATOR_TWO_DIMENSION」を指定した場合、シミュレーションの空間次元数が 2 と仮定される。このオプションに引数はない。本オプションが無指定の場合、3 が仮定される。空間次元数は、ユーザ定義型の位置と速度に対応するメンバ変数のデータ型のチェックに使用される。ユーザコードのコンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義される場合には、必ずこのオプションを指定しなければならない (無指定の場合、インターフェースプログラムの正常な動作は保証されない)。

本スクリプトの使用方法はオプション「-h」あるいは「--help」でも確認することできる。以下は、gen_ftn.if.py の場合の例である。

```
[user@hostname somedir]$ gen_ftn_if.py -h
[namekata@jenever0 scripts]$ ./gen_ftn_if.py --help
usage: gen_ftn_if.py [-h] [-o DIRECTORY] [-DPARTICLE_SIMULATOR_TWO_DIMENSION]
                     FILE [FILE ...]

Analyze user's Fortran codes and generate C++/Fortran source files
required to use FDPS from the user's Fortran code.

positional arguments:
  FILE                  The PATHs of input Fortran files

optional arguments:
  -h, --help            show this help message and exit
  -o DIRECTORY, --output DIRECTORY, --output_dir DIRECTORY
                        The PATH of output directory
  -DPARTICLE_SIMULATOR_TWO_DIMENSION
                        Indicate that simulation is performed
                        in the 2-dimensional space (equivalent
                        to define the macro
                        PARTICLE_SIMULATOR_TWO_DIMENSION)
```

生成されたインターフェースプログラムをユーザプログラムと一緒にコンパイルすることで、実行ファイルが得られる。コンパイルの仕方に関しては、次の第 7 章を参照されたい。

第7章　Fortran/C言語 インターフェースのコンパイル

ここまでの章で、FDPSの Fortran/C 言語 インターフェースの生成に必要な情報に関して解説を行ってきた。本章では、インターフェースプログラムのコンパイルに関連したトピックを扱う。第3章の図3.1に示されるように、FDPS Fortran インターフェースプログラムは、C++言語のソースファイルと Fortran 言語のソースファイルから構成される。同様、FDPS C 言語インターフェースプログラムも C++言語のソース・ファイルと C 言語のソースファイルから構成される (図3.2)。本章のはじめに、複数の言語で構成されるインターフェースプログラムをコンパイルする際の注意点について記述する。次に、FDPS Fortran/C 言語 インターフェースで使用可能なマクロ定義について記述する。FDPSでは、コンパイル時のマクロ定義によって、座標系の指定や並列処理の有無等を選択することができる。使用可能なマクロとその機能について解説する。

7.1　コンパイル

本節では、Fortran/C 言語 インターフェースプログラムを含むユーザコードをコンパイルする方法に関して記述する。はじめにコンパイラ依存しない事項に関して記述した後に、例として GCC (The GNU Compiler Collection) を使った場合のコンパイル方法を示す。

7.1.1　コンパイルの基本手順

ここでは、(コンパイラ依存しない) コンパイルの一般的な手順について説明する。まずはじめに Fortran インターフェースを利用する場合を述べ、その後、C 言語インターフェースを利用する場合を述べる。

7.1.1.1　Fortran インターフェースを利用する場合

前提条件として、C++言語と Fortran 言語で記述された複数のソースファイルをコンパイルして実行ファイルを得るためには、相互運用可能な C++コンパイラ、C++リンカー、および、Fortran コンパイラが必要である。今日では、通常、C++コンパイラは C++リンカーとして動作するため、事実上必要となるのは相互運用可能な C++コンパイラと Fortran コンパイラである。Fortran コンパイラは Fortran 2003 標準 (ISO/IEC 1539-1:2004(E)) に対応し

ていなければならない。また、FDPS 本体のコンパイルのため、C++コンパイラは C++03 標準 (ISO/IEC 14882:2003) に対応している必要がある。

第 3 章で述べた通り、Fortran インターフェースを用いたコードでは `main` 関数が C++側に存在する。したがって、コンパイルは、まずコンパイラで Fortran と C++のソースファイルからオブジェクトファイルを生成し、その後、C++リンカーでオブジェクトファイルをリンクするという手順となる。より詳細には、以下の手順でコンパイルを行う:

[1] **Fortran ソースのコンパイル**
ユーザが記述したすべての Fortran ソースコードの他、インターフェースプログラムの 1 つである `FDPS_module.F90`、FDPS から提供される Fortran ファイル群 (`src/fortran_interface/modules/*.F90`) を、Fortran コンパイラでコンパイルし、オブジェクトファイルを生成する。多くの場合、オブジェクトファイルの生成はコンパイラオプション「`-c`」を付けてコンパイルすることよってなされる。

コンパイル時に注意しなければならないのは、コンパイラに渡すファイルの順序である。多くのコンパイラでは、あるファイル `foo.F90` でモジュール `bar` を使用している場合 (`use` している場合)、モジュール `bar` が記述されたファイルは先にコンパイルされていなければならない。コンパイラは引数に渡されたファイルを先頭から順に処理するため、独立なモジュールを先に記述し、その後、依存関係の順にファイルを並べる必要がある。すなわち、Fortran コンパイラを `FC` とすれば、以下のようにコンパイルする:

```
$ FC -c \
  FDPS_time_profile.F90 \
  FDPS_vector.F90 \
  FDPS_matrix.F90 \
  FDPS_super_particle.F90 \
  user_defined_1.F90 ... user_defined_n.F90 \
  FDPS_module.F90 \
  user_code_1.F90 ... user_code_n.F90
```

ここで、\ はコマンドラインが次の行に継続することを表す。これは、本文書のスペースの都合上導入したものであり、実際には不要である。サブルーチン `f_main()` はユーザコード (`user_code_*.F90`) のどれかに実装されていると仮定する。この例におけるファイルの依存関係は次のようになっている:

- `FDPS_super_particle.F90` は `FDPS_vector.F90` と `FDPS_matrix.F90` に依存
- `FDPS_module.F90` はユーザ定義型が記述された n 個のファイル `user_defined_`i`.F90` ($i = 1$-n) に依存
- n 個のユーザコード `user_code_`i`.F90` ($i = 1$-n) は、`FDPS_module.F90` に依存

[2] **C++ソースのコンパイル**
インターフェースプログラムのすべての C++ファイル (`main.cpp`, `FDPS_Manipulators.cpp`, `FDPS_ftn_if.cpp`) を、C++コンパイラでコンパイルし、オブジェクトファイルを

生成する。C++はヘッダファイルが存在するため、ファイルの順番を気にする必要はない。したがって、コンパイラを CXX とすれば、以下のようにコンパイルする:

```
$ CXX -c FDPS_Manipulators.cpp FDPS_ftn_if.cpp main.cpp
```

[3] **オブジェクトファイルのリンク**
[1], [2] で作成したオブジェクトファイル (*.o) を、C++のリンカーでリンクし、実行ファイルを作成する。コンパイラによって、C++リンカーで Fortran のオブジェクトファイル C++のオブジェクトにリンクするために、特別なコンパイルオプションが必要となる場合がある。これを LDFLAGS とすると、リンクは以下のようにすればよい:

```
$ CXX *.o [LDFLAGS]
```

ここで、[] はその中身がコンパイラによっては省略可能であることを示す記号である。リンクが成功すれば、実行ファイルが作成されるはずである。

上に示した基本手順では、言語仕様を指定するコンパイラオプション等は省略している。また、並列計算や拡張機能を使う際に必要となるライブラリ等もすべて省略している。これらはコンパイラ依存する部分であり、使用するコンパイラに応じて適切に指定する必要がある。

7.1.1.2 C 言語インターフェースを利用する場合

C++言語と C 言語で記述された複数のソースファイルをコンパイルして実行ファイルを得るためには、相互運用可能な C++コンパイラ、C++リンカー、および、C コンパイラが必要である。前節で述べた理由により、事実上必要となるのは相互運用可能な C++コンパイラと C コンパイラである。C コンパイラは C99 規格 (ISO/IEC 9899:1999(E)) に対応していなければならない。また、FDPS 本体のコンパイルのため、C++コンパイラは C++03 標準 (ISO/IEC 14882:2003) に対応している必要がある。

第 3 章で述べた通り、C 言語 インターフェースを用いたコードでは main 関数が C++側に存在する。したがって、コンパイルは、まずコンパイラで C 言語と C++のソースファイルからオブジェクトファイルを生成し、その後、C++リンカーでオブジェクトファイルをリンクするという手順となる。より詳細には、以下の手順でコンパイルを行う:

[1] **C 言語ソースのコンパイル**
ユーザが記述したすべての C 言語ソースコードを、C コンパイラでコンパイルし、オブジェクトファイルを生成する。多くの場合、オブジェクトファイルの生成はコンパイラオプション「-c」を付けてコンパイルすることよってなされる。すなわち、C コンパイラを CC とすれば、以下のようにコンパイルする:

```
$ CC -c user_code_1.c ... user_code_n.c
```

[2] **C++ソースのコンパイル**
インターフェースプログラムのすべての C++ファイル (`main.cpp`, `FDPS_Manipulators.cpp`, `FDPS_ftn_if.cpp`) を、C++コンパイラでコンパイルし、オブジェクトファイルを生成する。したがって、コンパイラを `CXX` とすれば、以下のようにコンパイルする:

```
$ CXX -c FDPS_Manipulators.cpp FDPS_ftn_if.cpp main.cpp
```

[3] **オブジェクトファイルのリンク**
[1], [2] で作成したオブジェクトファイル (`*.o`) を、C++のリンカーでリンクし、実行ファイルを作成する。場合によっては、ライブラリをリンクする必要がある。リンクオプションを `LDFLAGS` とすると、リンクは以下のようにすればよい:

```
$ CXX *.o [LDFLAGS]
```

ここで、[] はその中身が場合によっては省略可能であることを示す記号である。リンクが成功すれば、実行ファイルが作成されるはずである。

上に示した基本手順では、言語仕様を指定するコンパイラオプション等は省略している。また、並列計算や拡張機能を使う際に必要となるライブラリ等もすべて省略している。これらはコンパイラ依存する部分であり、使用するコンパイラに応じて適切に指定する必要がある。

7.1.2 GCC を用いたコンパイルの仕方

本節では、例として、GCC (バージョン 4.8.3 以上) の場合のコンパイルの仕方を記述する。本節を通して、C++コンパイラ、Fortran コンパイラ、C コンパイラをそれぞれ `g++`、`gfortran`、`gcc` とする。また、MPI に対応した GCC コンパイラをぞれぞれ `mpic++`、`mpif90`、`mpicc` とし、使用する MPI ライブラリは OpenMPI (バージョン 1.6.4 以上) であるとする。以下、はじめに Fortran インターフェースを利用する場合について述べ、その後、C 言語インターフェースを利用する場合について述べる。

7.1.2.1 Fortran インターフェースを利用する場合

7.1.2.1.1 MPI を使用しない場合

`gfortran` で Fortran のソースコードを Fortran 2003 標準としてコンパイルするためには、コンパイルオプション `-std=f2003` が必要である。また、GCC の場合には、C++のオブジェクトファイルと Fortran のオブジェクトファイルをリンクするためには、リンク時にオプション `-lgfortran` が必要となる。したがって、第 7.1.1 節で説明した手順において、`FC`、`CXX`、`LDFLAGS` を、以下のように設定すればよい:

```
FC      = gfortran -std=f2003
CXX     = g++
LDFLAGS = -lgfortran
```

7.1.2.1.2 MPI を使用する場合

MPI を使用する場合で問題となるのは、Fortran で記述されたユーザコードの中で MPI を使用する場合である。この場合、C++用の MPI ライブラリだけでなく、Fortran 用の MPI ライブラリをリンクする必要がある。それぞれのライブラリの名称が `libmpi` と `libmpi_f90` であるとすれば、コンパイルは、第 7.1.1 節で説明した手順において、`FC`、`CXX`、`LDFLAGS` を、以下のように設定して行えばよい:

```
FC      = mpif90 -std=f2003
CXX     = mpic++
LDFLAGS = -lgfortran -L PATH -lmpi -lmpi_f90
```

ここで、*PATH* は MPI ライブラリがインストールされているディレクトリの絶対 PATH である。

MPI ライブラリの名称は当然ユーザの計算機環境ごとに異なりうる。詳細は、ユーザの利用している計算機システムの管理者に問い合わせて確認して頂きたい。

7.1.2.2 C 言語インターフェースを利用する場合

7.1.2.2.1 MPI を使用しない場合

コンパイルは、第 7.1.1 節で説明した手順において、`CC`、`CXX`、`LDFLAGS` を以下のように設定して行えばよい:

```
CC      = gc
CXX     = g++
LDFLAGS =
```

7.1.2.2.2 MPI を使用する場合

コンパイルは、第 7.1.1 節で説明した手順において、`CC`、`CXX`、`LDFLAGS` を以下のように設定して行えばよい:

```
CC      = mpicc
CXX     = mpic++
LDFLAGS = -LPATH -lmpi
```

ここで、*PATH* は MPI ライブラリがインストールされているディレクトリの絶対 PATH である。

MPI ライブラリの名称は当然ユーザの計算機環境ごとに異なりうる。詳細は、ユーザの利用している計算機システムの管理者に問い合わせて確認して頂きたい。

7.2 コンパイル時マクロ定義

7.2.1 座標系の指定

座標系は直角座標系 3 次元と直角座標系 2 次元の選択ができる。以下、それらの選択方法について述べる。

7.2.1.1 直角座標系 3 次元

デフォルトは直角座標系 3 次元である。なにも行わなくても直角座標系 3 次元となる。

7.2.1.2 直角座標系 2 次元

コンパイル時に PARTICLE_SIMULATOR_TWO_DIMENSION をマクロ定義すると直交座標系 2 次元となる。

7.2.2 並列処理の指定

並列処理に関しては、OpenMP の使用/不使用、MPI の使用/不使用を選択できる。以下、選択の仕方について記述する。

7.2.2.1 OpenMP の使用

デフォルトは OpenMP 不使用である。使用する場合は、PARTICLE_SIMULATOR_THREAD_PARALLEL をマクロ定義すればよい。GCC コンパイラの場合はコンパイラオプションに-fopenmp をつける必要がある。

7.2.2.2 MPI の使用

デフォルトは MPI 不使用である。使用する場合は、PARTICLE_SIMULATOR_MPI_PARALLEL をマクロ定義すればよい。

7.2.3 データ型の精度の指定

超粒子型 (第 4.3 節参照) のメンバ変数の型の精度を選択できる。以下、選択の仕方について記述する。

7.2.3.1 超粒子型のメンバ変数の型の精度の指定

デフォルトはすべてのメンバ変数が 64 ビットである。32 ビットにしたい場合、PARTICLE_SIMULATOR_SPMOM_F32 をマクロ定義すればよい。

7.2.4 拡張機能 Particle Mesh の使用

拡張機能 Particle Mesh を使用するためには、PARTICLE_SIMULATOR_USE_PM_MODULE をマクロ定義すればよい。デフォルトでは Particle Mesh 機能は使用できない。

7.2.5 デバッグ用出力の指定

デバッグ作業のため、マクロ PARTICLE_SIMULATOR_DEBUG_PRINT が用意されている。このマクロが定義済みの場合、FDPS の動作ログが出力されるようになる。

7.2.6 粒子のソートの方法の変更

TreeForForce クラスの内部では粒子はモートンキーの順でソートされている．デフォルトでは並列ソートアルゴリズムとしてマージソートが使われているが,
PARTICLE_SIMULATOR_USE_RADIX_SORT
をマクロ定義することでソートアルゴリズムを基数ソートに、また、
PARTICLE_SIMULATOR_USE_SAMPLE_SORT
をマクロ定義することで並列サンプルソートに変更できる。

アーキテクチャによっては基数ソートが若干速いかもしれない。また、 A64fx では並列サンプルソートが速いことが確認されている。

第8章　API 仕様一覧

この章では、Fortran/C 言語 インターフェースの各 API の仕様について記述する。第 3 章で述べた通り、Fortran では、各 API は派生データ型 `fdps_controller` のオブジェクトのメンバ関数として用意されている。以下では、このオブジェクトの名称が `fdps_ctrl` であるとして説明を行う。また、説明を簡略化するため、多くの場合、仮引数のデータ型は Fortran の場合のみを示す。C 言語のデータ型との対応は以下の表 8.1 を参考にして頂きたい。

Fortran のデータ型	C 言語でのデータ型
integer(kind=c_int)	int
integer(kind=c_short)	short int
integer(kind=c_long)	long int
integer(kind=c_long_long)	long long int
integer(kind=c_signed_char)	signed char/unsigned char
integer(kind=c_size_t)	size_t
integer(kind=c_int8_t)	int8_t
integer(kind=c_int16_t)	int16_t
integer(kind=c_int32_t)	int32_t
integer(kind=c_int64_t)	int64_t
integer(kind=c_int_least8_t)	int_least8_t
integer(kind=c_int_least16_t)	int_least16_t
integer(kind=c_int_least32_t)	int_least32_t
integer(kind=c_int_least64_t)	int_least64_t
integer(kind=c_int_fast8_t)	int_fast8_t
integer(kind=c_int_fast16_t)	int_fast16_t
integer(kind=c_int_fast32_t)	int_fast32_t
integer(kind=c_int_fast64_t)	int_fast64_t
integer(kind=c_intmax_t)	intmax_t
integer(kind=c_intptr_t)	intptr_t
real(kind=c_float)	float
real(kind=c_double)	double
real(kind=c_long_double)	long double
complex(kind=c_float_complex)	float _Complex
complex(kind=c_double_complex)	double _Complex
complex(kind=c_long_double_complex)	long double _Complex
logical(kind=c_bool)	_Bool
character(kind=c_char)	char

表 8.1: C 言語と相互運用可能な Fortran のデータ型と対応する C 言語のデータ型

8.1 開始および終了処理に関わる API

この節では、FDPS の初期化および終了処理に関わる API について記述する。
関連する全 API の名称の一覧を以下に示す:

```
ps_initialize (Fortran のみ)
fdps_initialize (C 言語のみ)
ps_finalize (Fortran のみ)
fdps_finalize (C 言語のみ)
ps_abort (Fortran のみ)
fdps_abort (C 言語のみ)
```

以下、順に、各 API の仕様を記述する。ただし、Fortran 版と C 言語版で同じ機能を持つ API については、まとめて一つの節で説明する。以降、本文書では、簡単のため、FortranとC言語のAPIの仕様を1つの節で説明する場合、その節の名前はFortranのAPI名を使用することにする。

8.1.1 ps_initialize

Fortran 構文

```
subroutine fdps_ctrl%ps_initialize()
```

C 言語 構文

```
void fdps_initialize();
```

仮引数仕様

なし

返値

なし

機能

FDPS の初期化を行う。FDPS の API のうち最初に呼び出さなければならない。

8.1.2 ps_finalize

Fortran 構文

```
subroutine fdps_ctrl%ps_finalize()
```

C 言語 構文

```
void fdps_finalize();
```

仮引数仕様

なし

返値

なし

機能

FDPS の終了処理を行う。

8.1.3 ps_abort

Fortran 構文

```
subroutine fdps_ctrl%ps_abort(err_num)
```

C 言語 構文

```
void fdps_abort(const int err_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
err_num	integer(kind=c_int)	入力	プログラムの終了ステータスを与える。Fortran では、この引数は省略可能であり、省略した場合、デフォルト値 -1 が使用される。

返り値

なし

機能

FDPS の異常終了処理を行う。引数はプログラムの終了ステータスである。この引数は、MPI を使用していない場合は C++の std::exit 関数に渡され、MPI を使用している場合は MPI の MPI_Abort 関数に渡される。

8.2 粒子群オブジェクト用 API

本節では、第 2 章で説明した粒子群クラスのオブジェクト (以後、**粒子群オブジェクト**と呼ぶ) に関する API について説明する。FDPS 本体において、粒子群オブジェクトは FullParticle 型に記述された粒子の情報のすべてを持ち、粒子交換を行う API を提供する。ユーザは、粒子群オブジェクトを通じて、粒子情報の初期化・更新を行うこととなる。Fortran/C 言語 インターフェースを用いたプログラムでは、粒子群オブジェクトを識別番号で管理する。

粒子群オブジェクトを操作する全 API の名称の一覧を以下に示す:

```
(fdps_)create_psys
(fdps_)delete_psys
(fdps_)init_psys
(fdps_)get_psys_info
(fdps_)get_psys_memsize
(fdps_)get_psys_time_prof
(fdps_)clear_psys_time_prof
(fdps_)set_nptcl_smpl
(fdps_)set_nptcl_loc
(fdps_)get_nptcl_loc
(fdps_)get_nptcl_glb
get_psys_fptr (Fortran のみ)
fdps_get_psys_cptr (C 言語のみ)
(fdps_)exchange_particle
(fdps_)add_particle
(fdps_)remove_particle
(fdps_)adjust_pos_into_root_domain
(fdps_)sort_particle
```

ここで、(fdps_) は C 言語の場合のみ API 名に接頭辞 fdps_ が付くことを示している。以下、順に、各 API の仕様を記述する。

8.2.1 create_psys

Fortran 構文

```
subroutine fdps_ctrl%create_psys(psys_num,psys_info_in)
```

C 言語 構文

```
void fdps_create_psys(int *psys_num,
                      char *psys_info);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入出力	粒子群オブジェクトの識別番号を受け取るための変数。C 言語では変数のアドレスを引数として指定する必要があることに注意。
psys_info_in	character(len=*,kind=c_char)	入力	FullParticle 型の派生データ型名を格納した文字列。
psys_info	char *	入力	FullParticle 型の構造体名を格納した文字列定数。

返り値

なし

機能

文字列 psys_info_in (Fortran の場合) 或いは 文字列定数 psys_info (C 言語の場合) で指定される名称の FullParticle 型に対応した粒子群オブジェクトを生成し、そのオブジェクトの識別番号を返す。FullParticle型の派生データ型名 (Fortran の場合) 或いは 構造体名 (C 言語の場合) はすべて小文字で入力されなければならない。

8.2.2 delete_psys

Fortran 構文

```
subroutine fdps_ctrl%delete_psys(psys_num)
```

C 言語 構文

```
void fdps_delete_psys(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

なし

機能

メモリー上から、識別番号 psys_num を持つ粒子群オブジェクトを削除する。

8.2.3 init_psys

Fortran 構文

```
subroutine fdps_ctrl%init_psys(psys_num)
```

C 言語 構文

```
void fdps_init_psys(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトを初期化する。以降に記述する粒子群オブジェクト用 API を使用する前に、必ず 1 度呼び出す必要がある。

8.2.4 get_psys_info

Fortran 構文

```
subroutine fdps_ctrl%get_psys_info(psys_num,psys_info)
```

C 言語 構文

```
void fdps_get_psys_info(const int psys_num,
                        char *psys_info,
                        size_t *charlen);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
psys_info	character(len=*,kind=c_char)	入出力	粒子群オブジェクトに対応した FullParticle 型の派生データ型名 (Fortran の場合) 或いは 構造体名 (C 言語の場合)。
charlen	size_t *	入出力	変数psys_infoに書き込まれた文字列の長さ。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトに対応した FullParticle 型の派生データ型名 (Fortran の場合) 或いは 構造体名 (C 言語の場合) を取得する。これは粒子群オブジェクト生成時に指定した文字列そのものである。

8.2.5 get_psys_memsize

Fortran 構文

```
function fdps_ctrl%get_psys_memsize(psys_num)
```

C 言語 構文

```
long long int fdps_get_psys_memsize(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型のスカラー値。

機能

識別番号 psys_num の粒子群オブジェクトが消費しているメモリー量を Byte 単位で返す。

8.2.6 get_psys_time_prof

Fortran 構文

```
subroutine fdps_ctrl%get_psys_time_prof(psys_num,prof)
```

C 言語 構文

```
void fdps_get_psys_time_prof(const int psys_num,
                             fdps_time_prof *prof);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
prof	type(fdps_time_prof)	入出力	粒子群オブジェクト用の API でかかった時間を受け取るための変数。C 言語では変数のアドレスを引数として指定する必要があることに注意。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトで粒子交換 (API (fdps_)exchange_particle) にかかった時間 (ミリ秒単位) を fdps_time_prof 型のメンバ変数 exchange_particle に格納する。

8.2.7 clear_psys_time_prof

Fortran 構文

```
subroutine fdps_ctrl%clear_psys_time_prof(psys_num)
```

C 言語 構文

```
void fdps_clear_psys_time_prof(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

なし

機能

FDPS 本体に用意された識別番号 psys_num の粒子群オブジェクトの TimeProfile 型プライベートメンバ変数のメンバ変数 exchange_particles_の値を 0 クリアする。ここで、TimeProfile 型は Fortran/C 言語インターフェースで用意された fdps_time_prof 型に対応する C++のデータ型のことである (詳細は、FDPS 本体の仕様書を参照)。本 API は時間計測をリセットするために使用する。

8.2.8 set_nptcl_smpl

Fortran 構文

```
subroutine fdps_ctrl%set_nptcl_smpl(psys_num,nptcl)
```

C 言語 構文

```
void fdps_set_nptcl_smpl(const int psys_num,
                         const int nptcl);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
nptcl	integer(kind=c_int)	入力	1 つの MPI プロセスでサンプルする粒子数目標。

返り値

なし

機能

1 つの MPI プロセスでサンプルする粒子数の目標を設定する。呼び出さなくてもよいが、呼び出さないとこの目標数が 30 となる。

8.2.9 set_nptcl_loc

Fortran 構文

```
subroutine fdps_ctrl%set_nptcl_loc(psys_num,nptcl)
```

C 言語 構文

```
void fdps_set_nptcl_loc(const int psys_num,
                        const int nptcl);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
nptcl	integer(kind=c_int)	入力	粒子数。

返り値

なし

機能

1 つの MPI プロセスの持つ粒子数を設定する。MPI プロセスごとに異なる粒子数を指定してもよい。

8.2.10 get_nptcl_loc

Fortran 構文

```
function fdps_ctrl%get_nptcl_loc(psys_num)
```

C 言語 構文

```
int fdps_get_nptcl_loc(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

integer(kind=c_int) 型のスカラー値。

機能

自分の MPI プロセスの持つ粒子数を返す。

8.2.11 get_nptcl_glb

Fortran 構文

```
function fdps_ctrl%get_nptcl_glb(psys_num)
```

C 言語 構文

```
int fdps_get_nptcl_glb(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

integer(kind=c_int) 型のスカラー値。

機能

全粒子数を返す。

8.2.12 get_psys_fptr (Fortran のみ)

Fortran 構文

```
subroutine fdps_ctrl%get_psys_fptr(psys_num,fptr_to_FP)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
fptr_to_FP	FullParticle 型, dimension(:), pointer	入出力	粒子群オブジェクトで管理されている FullParticle 型の粒子配列へのポインタ。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトで管理されている FullParticle 型の粒子配列へのポインタを取得する。配列サイズは本 API の呼出時のローカル粒子数 (API get_nptcl_loc の返り値) にセットされる。正常にアクセス可能なのは、fptr_to_FP(i) ($i = 1$〜ローカル粒子数) である。本 API は粒子群オブジェクトで管理される FullParticle 型の粒子配列への唯一のアクセス方法を提供する。本 API の使用例を以下に示す。この例では、粒子群オブジェクトで管理されている full_particle 型 の粒子配列のポインタを取得し、値を設定している:

Listing 8.1: API get_psys_fptr の使用例

```
!* Local variables
type(full_particle), dimension(:), pointer :: ptcl
!* Get the pointer to full particle data
call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
!* Set particle data
do i=1,nptcl_loc
   ptcl(i)%mass =  ! do something
   ptcl(i)%pos%x = ! do something
   ptcl(i)%pos%y = ! do something
   ptcl(i)%pos%z = ! do something
end do
```

8.2.13 fdps_get_psys_cptr (C 言語のみ)

C 言語 構文

```
void * fdps_get_psys_cptr(const int psys_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	const int	入力	粒子群オブジェクトの識別番号。

返り値

void *型。

機能

識別番号 psys_num の粒子群オブジェクトで管理されている FullParticle 型の粒子配列の先頭アドレスを取得する。正常にアクセス可能なのは、配列要素が 0 から $n_{\mathrm{ptcl,loc}} - 1$ の間である。ここで、$n_{\mathrm{ptcl,loc}}$ は API fdps_get_nptcl_loc の返り値である。本 API は粒子群オブジェクトで管理される FullParticle 型の粒子配列への唯一のアクセス方法を提供する。本 API の使用例を以下に示す。この例では、粒子群オブジェクトで管理されている full_particle 型の粒子配列のポインタを取得し、値を設定している:

Listing 8.2: API fdps_get_psys_cptr の使用例

```
1  // Local variables
2  struct full_particle *ptcl;
3  // Get the pointer to full particle data
4  ptcl = (struct full_particle *) fdps_get_psys_cptr(psys_num);
5  // Set particle data
6  for (i = 0; i < nptcl_loc; i++) {
7     ptcl[i].mass =  // do something
8     ptcl[i].pos.x = // do something
9     ptcl[i].pos.y = // do something
10    ptcl[i].pos.z = // do something
11 }
```

8.2.14 exchange_particle

Fortran 構文

```
subroutine fdps_ctrl%exchange_particle(psys_num,dinfo_num)
```

C 言語 構文

```
void fdps_exchange_particle(const int psys_num,
                            const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。

返り値

なし

機能

粒子が適切なドメインに配置されるように、粒子の交換を行う。

8.2.15 add_particle

Fortran 構文

```
subroutine fdps_ctrl%add_particle(psys_num,ptcl)
```

C 言語 構文

```
void fdps_add_particle(const int psys_num,
                       const void *cptr_to_fp);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
ptcl	FullParticle 型	入力	追加したい粒子のデータ。
cptr_to_fp	void *型	入力	追加したい粒子のデータのアドレス。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトで管理されている FullParticle 型の粒子配列の末尾に、粒子 ptcl (Fortran の場合) 或いは ポインタ cptr_to_fp が指す粒子データのコピー (C 言語の場合) を追加する。

8.2.16 remove_particle

Fortran 構文

```
subroutine fdps_ctrl%remove_particle(psys_num,nptcl,ptcl_indx)
```

C 言語 構文

```
void fdps_remove_particle(const int psys_num,
                          const int nptcl,
                          int *ptcl_indx);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
nptcl	integer(kind=c_int)	入力	配列 ptcl_indx のサイズ。
ptcl_indx	integer(kind=c_int), dimension(nptcl)	入力	消去する粒子の配列インデックス (配列要素番号) を格納した配列。C 言語では配列の先頭アドレスを引数に指定することに注意。

返り値

なし

機能

配列 ptcl_indx に格納されている配列インデックスの粒子を削除する。配列インデックスの最小値は Fortran では 1、C 言語では 0 とする。この関数を呼ぶ前後で、粒子の配列インデックスが同じである事は保証されない。

8.2.17 adjust_pos_into_root_domain

Fortran 構文

```
subroutine fdps_ctrl%adjust_pos_into_root_domain(psys_num,dinfo_num)
```

C 言語 構文

```
void fdps_adjust_pos_into_root_domain(const int psys_num,
                                      const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。

返り値

なし

機能

周期境界条件の場合に、計算領域からはみ出した粒子を計算領域に適切に戻す。

8.2.18 sort_particle

Fortran 構文

```
subroutine fdps_ctrl%sort_particle(psys_num,pfunc_comp)
```

C 言語 構文

```
void fdps_sort_particle(const int psys_num,
                        _Bool (*pfunc_comp)(const void *, const void
*));
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
pfunc_comp	type(c_funptr)	入力	比較関数の関数ポインタ。

返り値

なし

機能

粒子群オブジェクトが保持する FullParticle の配列を比較関数 comp (この関数ポインタが pfunc_comp) で指示したように並べ替える。比較関数は返り値を logical(kind=c_bool) 型とし、引数は 識別番号 psys_num に対応した FullParticle 型を 2 つ取るものである必要がある (比較関数の引数のデータ型が、識別番号 psys_num で指定される粒子群オブジェクトの生成に使用した FullParticle 型と異なる場合の動作は不定である)。例として以下に FullParticle がメンバ変数 id を持っており id の昇順に並べ替える場合の比較関数を示す (Fortran の場合)。

Listing 8.3: 比較関数の例

```
1 function comp(left, right) bind(c)
2    use, intrinsic :: iso_c_binding
3    use user_defined_types
4    implicit none
5    logical(kind=c_bool) :: comp
6    type(full_particle), intent(in) :: left, right
```

```
7     comp = (left%id < right%id)
8 end function comp
```

ここで、構造体 full_particle は、モジュール user_defined_types 内で定義されているものとする。

8.2.19 set_psys_comm_info

Fortran 構文

```
subroutine fdps_ctrl%set_psys_comm_info(psys_num, ci)
```

C 言語 構文

```
void fdps_fdps_set_psys_comm_info(int psys_num,
                                  int ci);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
ci	integer(kind=c_int)	入力	コミュニケータクラスに対応する番号。

返り値

なし

機能

通信に使うコミュニケータを指定する。

8.3 領域情報オブジェクト用 API

本節では、第 2 章で説明した領域情報クラスのオブジェクト (以後、**領域情報オブジェクト**と呼ぶ) に関する API について説明する。FDPS 本体において、領域情報オブジェクトは、領域情報を保持し、領域分割を行う API を提供する。Fortran/C 言語 インターフェースを用いたプログラムでは、領域情報オブジェクトを識別番号で管理する。

領域情報オブジェクトを操作する全 API の名称の一覧を以下に示す:

```
(fdps_)create_dinfo
(fdps_)delete_dinfo
(fdps_)init_dinfo
(fdps_)set_dinfo_comm_info
(fdps_)get_dinfo_time_prof
(fdps_)clear_dinfo_time_prof
(fdps_)set_nums_domain
(fdps_)set_boundary_condition
(fdps_)get_boundary_condition
(fdps_)set_pos_root_domain
(fdps_)set_pos_root_domain_x
(fdps_)set_pos_root_domain_y
(fdps_)set_pos_root_domain_z
(fdps_)collect_sample_particle
(fdps_)decompose_domain
(fdps_)decompose_domain_all
```

ここで、(fdps_) の意味は前節冒頭で述べた通りである。

以下、順に、各 API の仕様を記述する。

8.3.1 create_dinfo

Fortran 構文

```
subroutine fdps_ctrl%create_dinfo(dinfo_num)
```

C 言語 構文

```
void fdps_create_dinfo(int *dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入出力	領域情報オブジェクトの識別番号を受け取るための変数。C 言語では変数のアドレスを引数に指定する必要があることに注意。

返り値

なし

機能

領域情報オブジェクトをメモリ上に生成し、そのオブジェクトの識別番号を返す。

8.3.2 delete_dinfo

Fortran 構文

```
subroutine fdps_ctrl%delete_dinfo(dinfo_num)
```

C 言語 構文

```
void fdps_delete_dinfo(const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。

返り値

なし

機能

識別番号 dinfo_num の領域情報オブジェクトをメモリから消去する。

8.3.3 init_dinfo

Fortran 構文

```
subroutine fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
```

C 言語 構文

```
void fdps_init_dinfo(const int dinfo_num,
                     const float coef_ema);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
coef_ema	real(kind=c_float)	入力	指数移動平均の平滑化係数。Fortran の場合、この引数は省略可能で、省略された場合、デフォルト値は 1 が使用される。C 言語の場合、変数の値が < 0 または > 1 の場合、自動的に Fortran のデフォルト値が使用される。

返り値

なし

機能

領域情報オブジェクトを初期化し、指数移動平均の平滑化係数を設定する。この係数の許される値は 0 から 1 である。それ以外の値を入れた場合はエラーメッセージを送出しプログラムは終了する。大きくなるほど、最新の粒子分布の情報が領域分割に反映されやすい。1 の場合、最新の粒子分布の情報のみ反映される。0 の場合、最初の粒子分布の情報のみ反映される。 1 度は呼ぶ必要がある。過去の粒子分布の情報を領域分割に反映する必要がある理由については、Ishiyama, Fukushige & Makino (2009, Publications of the Astronomical Society of Japan, 61, 1319) を参照のこと。

8.3.4 get_dinfo_time_prof

Fortran 構文

```
subroutine fdps_ctrl%get_dinfo_time_prof(dinfo_num,prof)
```

C 言語 構文

```
void fdps_get_dinfo_time_prof(const int dinfo_num,
                              fdps_time_prof *prof);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
prof	type(fdps_time_prof)	入出力	領域情報オブジェクトの API でかかった時間を受け取るための変数。C 言語では引数に変数のアドレスを指定する必要があることに注意。

返り値

なし

機能

領域情報オブジェクトの API である (fdps_)collect_sample_particle と (fdps_)decompose_domain にかかった時間 (ミリ秒単位) を fdps_time_prof 型変数のメンバ変数である collect_sample_particles と decompose_domain に格納する。

8.3.5 clear_dinfo_time_prof

Fortran 構文

```
subroutine fdps_ctrl%clear_dinfo_time_prof(dinfo_num)
```

C 言語 構文

```
void fdps_clear_dinfo_time_prof(const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。

返り値

なし

機能

FDPS 本体に用意された識別番号 dinfo_num の領域情報オブジェクトの TimeProfile 型プライベートメンバ変数のメンバ変数 collect_sample_particles と decompose_domain の値を 0 クリアする。ここで、TimeProfile 型は Fortran/C 言語インターフェースで用意された fdps_time_prof 型に対応する C++のデータ型のことである (詳細は、FDPS 本体の仕様書を参照)。本 API は時間計測をリセットするために使用する。

8.3.6 set_nums_domain

Fortran 構文

```
subroutine fdps_ctrl%set_nums_domain(dinfo_num,nx,ny,nz)
```

C 言語 構文

```
void fdps_set_nums_domain(const int dinfo_num,
                          const int nx,
                          const int ny,
                          const int nz);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
nx	integer(kind=c_int)	入力	x 軸方向のルートドメインの分割数。
ny	integer(kind=c_int)	入力	y 軸方向のルートドメインの分割数。
nz	integer(kind=c_int)	入力	z 軸方向のルートドメインの分割数で、デフォルトは 1。

返り値

なし

機能

計算領域の分割する方法を設定する。nx, ny, nz はそれぞれ x 軸、y 軸、z 軸方向の計算領域の分割数である。呼ばなければ自動的に nx, ny, nz が決まる。呼んだ場合に入力する nx, ny, nz の総積が MPI プロセス数と等しくなければ、FDPS はエラーメッセージを送り、プログラムを止める。

8.3.7 set_boundary_condition

Fortran 構文

```
subroutine fdps_ctrl%set_boundary_condition(dinfo_num,bc)
```

C 言語 構文

```
void fdps_set_boundary_condition(const int dinfo_num,
                                 const int bc);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
bc	integer(kind=c_int)	入力	境界条件を与えるための変数。

返り値

なし

機能

境界条件の設定をする。許される入力は、第 4.6 節で説明した境界条件型である。すなわち、Fortran では、fdps_bc_open(開境界)、fdps_bc_periodic_x と fdps_bc_periodic_y と fdps_bc_periodic_z(それぞれ x, y, z 軸のみ周期境界でそれ以外が開境界)、fdps_bc_periodic_xy と fdps_bc_periodic_xz と fdps_bc_periodic_yz(それぞれ xy, xz, yz 軸のみ周期境界でそれ以外が開境界)、fdps_bc_periodic_xyz(xyz 軸すべて周期境界)、fdps_bc_shearing_box(シアリングボックス)、fdps_bc_user_defined(ユーザー定義の境界条件) である。ただし、fdps_bc_shearing_box と fdps_bc_user_defined は未実装である。C 言語でも上記に対応する境界条件型が用意されているので、それらのみ指定可能である。

8.3.8 get_boundary_condition

Fortran 構文

```
function fdps_ctrl%set_boundary_condition(dinfo_num)
```

C 言語 構文

```
int fdps_get_boundary_condition(const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。

返り値

integer(kind=c_int) 型のスカラー値。

機能

現在設定されている境界条件の情報を整数値として返す。取りうる値は境界条件型 (第 4.6 節参照) の各列挙子に対応する整数である。

8.3.9 set_pos_root_domain

Fortran 構文

```
subroutine fdps_ctrl%set_pos_root_domain(dinfo_num,low,high)
```

C 言語 構文

```
void fdps_set_pos_root_domain(const int dinfo_num,
                              const fdps_f32vec *low,
                              const fdps_f32vec *high);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
low	Fortran の場合、以下のいずれか: real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec) C 言語では fdps_f32vec *型のみ	入力	ルートドメインの下限 (閉境界)。
high	low と同じ	入力	ルートドメインの上限 (開境界)。

返り値

なし

機能

計算領域の下限と上限を設定する。開放境界条件の場合は呼ぶ必要はない。それ以外の境界条件の場合は、呼ばなくても動作するが、その結果が正しいことは保証できない。high の座標の各値は low の対応する座標よりも大きくなければならない。そうでない場合は、FDPS はエラーメッセージを送出し、ユーザープログラムを終了させる。

8.3.10 collect_sample_particle

Fortran 構文

```
subroutine fdps_ctrl%collect_sample_particle(dinfo_num, &
                                             psys_num,  &
                                             clear,     &
                                             weight)
```

C 言語 構文

```
void fdps_collect_sample_particle(const int dinfo_num,
                                  const int psys_num,
                                  const _Bool clear,
                                  const float weight);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
psys_num	integer(kind=c_int)	入力	領域分割のためのサンプル粒子を提供する粒子群オブジェクトの識別番号を与えるための変数。
clear	logical(kind=c_bool)	入力	前にサンプルされた粒子情報をクリアするかどうかを決定するフラグ。.true. (Fortran の場合)/true (C 言語の場合) でクリアする。Fortran では、この引数は省略可能であり、省略された場合のデフォルト値は.true. である。
weight	real(kind=c_float)	入力	領域分割のためのサンプル粒子数を決めるためのウェイト。Fortran では、この引数は省略可能であり、省略された場合のデフォルト値はこの API を呼び出したプロセスが担当する粒子数となる。C 言語では負の値が入力された場合、自動的に Fortran におけるデフォルト値が設定される。プロセス i のウェイトを w_i、API (fdps_)set_nptcl_smpl で設定されたプロセスあたりのサンプル粒子数を $n_{\rm smpl}$、プロセス数を $n_{\rm proc}$ とすると、プロセス i からは $n_{\rm smpl} n_{\rm proc} (w_i / \sum_k w_k)$ 個の粒子数がサンプルされる。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトから粒子をサンプルする。clear によってこれより前にサンプルした粒子の情報を消すかどうか決める。weight によってその MPI プロセスからサンプルする粒子の量を調整する (weight が大きいほどサンプル粒子数が多い)。

8.3.11 decompose_domain

Fortran 構文

```
subroutine fdps_ctrl%decompose_domain(dinfo_num)
```

C 言語 構文

```
void fdps_decompose_domain(const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。

返り値

なし

機能

計算領域の分割を実行する。

8.3.12 decompose_domain_all

Fortran 構文

```
subroutine fdps_ctrl%decompose_domain_all(dinfo_num, &
                                          psys_num,  &
                                          weight)
```

C 言語 構文

```
void fdps_decompose_domain_all(const int dinfo_num,
                               const int psys_num,
                               const float weight);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
psys_num	integer(kind=c_int)	入力	領域分割のためのサンプル粒子を提供する粒子群オブジェクトの識別番号を与えるための変数。
weight	real(kind=c_float)	入力	領域分割のためのサンプル粒子数を決めるためのウェイト。ウェイトの意味とデフォルト値については、API (fdps_)collect_sample_particle を参照。

返り値

なし

機能

識別番号 psys_num の粒子群オブジェクトから粒子をサンプルし、続けてルートドメインの分割を行う。すなわち、領域情報オブジェクトの API である (fdps_)collect_sample_particle と (fdps_)decompose_domain が行うことをこの API は一度に行う。

8.3.13 set_dinfo_comm_info

Fortran 構文

```
subroutine fdps_ctrl%set_dinfo_comm_info(dinfo_num, ci)
```

C 言語 構文

```
void fdps_fdps_set_dinfo_comm_info(int dinfo_num,
                                   int ci);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号を与えるための変数。
ci	integer(kind=c_int)	入力	コミュニケータクラスに対応する番号。

返り値

なし

機能

通信に使うコミュニケータを指定する。

8.4 ツリーオブジェクト用 API

本節では、第 2 章で説明した相互作用ツリークラスのオブジェクト (以後、単に**ツリーオブジェクト**と呼ぶ) に関する API について説明する。FDPS 本体において、ツリーオブジェクトは粒子間相互作用の計算を行う API を提供する。Fortran/C 言語 インターフェースを用いたプログラムでは、ツリーオブジェクトを識別番号で管理する。

ツリーオブジェクトを操作する全 API の名称の一覧を以下に示す:

```
(fdps_)create_tree
(fdps_)delete_tree
(fdps_)init_tree
(fdps_)get_tree_info
(fdps_)get_tree_memsize
(fdps_)get_tree_time_prof
(fdps_)clear_tree_time_prof
(fdps_)get_num_interact_ep_ep_loc
(fdps_)get_num_interact_ep_sp_loc
(fdps_)get_num_interact_ep_ep_glb
(fdps_)get_num_interact_ep_sp_glb
(fdps_)clear_num_interact
(fdps_)get_num_tree_walk_loc
(fdps_)get_num_tree_walk_glb
(fdps_)set_particle_local_tree
(fdps_)get_force
(fdps_)calc_force_all_and_write_back
(fdps_)calc_force_all
(fdps_)calc_force_making_tree
(fdps_)calc_force_and_write_back
(fdps_)get_neighbor_list
(fdps_)get_epj_from_id
(fdps_)set_tree_comm_info
(fdps_)set_exchange_let_mode
```

以下、ツリーの種類に関して記述した後に、順に、各 API の仕様を記述していく。

8.4.1 ツリーの種別

本節では FDPS Fortran/C 言語 インターフェースで使用可能なツリーの種類とその定義について説明する。自然界のほとんどの相互作用は、長距離力と短距離力に分類することができる。これに応じて、FDPS では長距離力計算と短距離力計算で異なるツリーを用いる。ここでは、簡単のため、それぞれ、**長距離力用ツリー**と**短距離力用ツリー**と呼ぶことにする。FDPS ではこれら 2 種類のツリーが、さらに動作モードに応じて細分される。以下、長距離力用ツリーと短距離力用ツリーに分けて、記述する。

8.4.1.1 長距離力用ツリーの種別

長距離用ツリーは、モーメントの計算方法別に 10 種類に細分される。粒子の重心を中心として単極子まで計算する場合を Monopole 型、同じく四重極子までのモーメントを計算する場合を Quadrupole 型と呼ぶ。粒子の幾何中心を中心として単極子まで、双極子まで、そして、四重極子までのモーメントを計算する場合を、それぞれ、MonopoleGeometricCenter 型、DipoleGeometricCenter 型、QuadrupoleGeometricCenter 型と呼ぶ。

P^3T(Particle-Particle-Particle-Tree) 法等、一部の相互作用計算法では、近傍粒子探索が必要となる場合がある。そのような方法を使うユーザ用に、近傍粒子探索を可能とした Monopole 型と Quadrupole 型も用意している。近傍粒子探索を j 粒子の探索半径を用いて行う場合を、それぞれ、MonopoleWithScatterSearch 型、QuadrupoleWithScatterSearch 型と呼ぶ。近傍粒子探索を i 粒子の j 粒子の探索半径の大きい方を用いて行う場合を、それぞれ、MonopoleWithSymmetrySearch 型、QuadrupoleWithSymmetrySearch 型と呼ぶ。相互計算時には近傍粒子は超粒子に含まれず、通常の粒子として計算される。探索半径の持たせ方に関しては、§ 5.1.3.2 及び § 5.1.4.2 を参照のこと。

さらに、P^3M(Particle-Particle-Particle-Mesh) 法や TreePM 法などでは、長距離力をカットオフ半径によって遠方成分と近傍成分に分け、遠方成分は PM 法で、近傍成分は直接計算かツリー法で計算する。このような場合、カットオフ半径に含まれるツリー構造だけを考慮すればよく、この点における最適化を行える。これを Monopole 型に適用したものが、MonopoleWithCutoff 型である。この MonopoleWithCutoff 型ではカットオフ半径はすべての粒子で同一である必要がある。カットオフ半径は相互作用する粒子を見つけるための探索半径として使われ、探索半径は EssentialParticleJ 型が持っている必要がある (詳細は § 5.1.4.2 を参照のこと)。

以上が、本 Fortran/C 言語 インターフェースで使用可能な長距離力用ツリーである。一覧は、第 4.4 節の表 4.3 に示している。

8.4.1.2 短距離力用ツリーの種別

短距離用ツリーは、相互作用の仕方別に以下の 3 種類に細分される:

1. **Gather 型**
   相互作用の到達距離が有限で、かつ、その到達距離が i 粒子の大きさ、或いは、i 粒子が持つ探索半径で決まる場合。

2. **Scatter 型**
   相互作用の到達距離が有限で、かつ、その到達距離が j 粒子の大きさ、或いは、j 粒子が持つ探索半径で決まる場合。

3. **Symmetry 型**
   相互作用の到達距離が有限で、かつ、その到達距離が i,j 粒子の大きさ (i,j 粒子が持つ探索半径) のどちらか大きい方で決まる場合。

8.4.2 create_tree

Fortran 構文

```
subroutine fdps_ctrl%create_tree(tree_num,tree_info_in)
```

C 言語 構文

```
void fdps_create_tree(int *tree_num,
                      char *tree_info);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入出力	ツリーオブジェクトの識別番号を受け取るための変数。C 言語では引数に変数のアドレスを指定する必要があることに注意。
tree_info_in	character (len=*,kind=c_char)	入力	生成するツリーの種別を指定するための文字列。
tree_info	char *	入力	生成するツリーの種別を指定するための文字列定数。

返り値

なし

機能

メモリ上にツリーオブジェクトを生成し、そのオブジェクトの識別番号を返す。ツリーオブジェクトの種類は、文字列 tree_info_in (Fortran の場合) 或いは 文字列定数 tree_info (C 言語の場合) により指定される。長距離力用ツリーオブジェクトを生成する場合、文字列を以下のように指定する:

```
"Long,<force_type>,<epi_type>,<epj_type>,<tree_mode>"
```

ここで、<tree_mode>として取れるのは、Monopole, Quadrupole, MonopoleGeometricCenter, DipoleGeometricCenter, QuadrupoleGeometricCenter, MonopoleWithScatterSearch, QuadrupoleWithScatterSearch, MonopoleWithSymmetrySearch, QuadrupoleWithSymmetrySearch, MonopoleWithCutoff のいずれかである。Long も含め、これらのキーワードは大文字・小文字が区別される。さらに、角括弧<>は入力してはならない。これらは第 8.4.1.1 節で説明した長距離力用ツリーの種別に対応している。短距離力用ツリーオブジェクトを生成する場合、文字列を以下のように指定する:

```
"Short,<force_type>,<epi_type>,<epj_type>,<search_mode>"
```

ここで、<search_mode>として取れるのは、Gather, Scatter, Symmetry のいずれかである。同様に、大文字・小文字が区別される。これらは第 8.4.1.2 節で説明した短距離用ツリーの種別に対応している。

長距離力用ツリーと短距離力用ツリーに共通して、<force_type>, <epi_type>, <epj_type> にはユーザー定義型の派生データ型名 (Fortran の場合) 或いは 構造体名 (C 言語の場合) を指定する。各カンマの前後に空白があってはならない。また、文字列はすべて小文字で入力されなければならない。

8.4.3 delete_tree

Fortran 構文

```
subroutine fdps_ctrl%delete_tree(tree_num)
```

C 言語 構文

```
void fdps_delete_tree(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号を受け取るための変数。

返り値

なし

機能

識別番号 tree_num のツリーオブジェクトをメモリ上から削除する。

8.4.4 init_tree

Fortran 構文

```
subroutine fdps_ctrl%init_tree(tree_num,     &
                               nptcl,theta,  &
                               n_leaf_limit, &
                               n_group_limit)
```

C 言語 構文

```
void fdps_init_tree(const int tree_num,
                    const int nptcl,
                    const float theta,
                    const int n_leaf_limit,
                    const int n_group_limit);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
nptcl	integer(kind=c_int)	入力	粒子配列の上限。
theta	real(kind=c_float)	入力	見こみ角に対する基準。Fortran の場合、この引数は省略可能であり、省略された場合のデフォルト値は 0.7 である。C 言語では負値が入力された場合、自動的に Fortran におけるデフォルト値が設定される。
n_leaf_limit	integer(kind=c_int)	入力	ツリーを切るのをやめる粒子数の上限。Fortran の場合、この引数は省略可能であり、省略された場合のデフォルト値は 8 である。C 言語では負値が入力された場合、自動的に Fortran におけるデフォルト値が設定される。
n_group_limit	integer(kind=c_int)	入力	相互作用リストを共有する粒子数の上限。Fortran では省略可能であり、省略された場合のデフォルト値は 64 である。C 言語では負値が入力された場合、自動的に Fortran におけるデフォルト値が設定される。

返り値

なし

機能

識別番号 tree_num のツリーオブジェクトを初期化する。

8.4.5 get_tree_info

Fortran 構文

```
subroutine fdps_ctrl%get_tree_info(tree_num,tree_info)
```

C 言語 構文

```
void fdps_get_tree_info(const int tree_num,
                        char *tree_info,
                        size_t *charlen);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
tree_info	character(len=*,kind=c_char)	入出力	ツリーの種別を示す文字列を受け取るための変数。C 言語では引数に変数のアドレスを指定する必要があることに注意。
charlen	size_t *	入出力	tree_info に書き込まれた文字列の長さ。

返り値

なし

機能

識別番号 tree_num のツリーの種別を示す文字列を取得する。この文字列はツリー生成時に指定した文字列である。

8.4.6 get_tree_memsize

Fortran 構文

```
function fdps_ctrl%get_tree_memsize(tree_num)
```

C 言語 構文

```
long long int fdps_get_tree_memsize(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

対象のオブジェクトが使用しているメモリー量を Byte 単位で返す。

8.4.7 get_tree_time_prof

Fortran 構文

```
subroutine fdps_ctrl%get_tree_time_prof(tree_num,prof)
```

C 言語 構文

```
void fdps_get_tree_time_prof(const int tree_num,
                             fdps_time_prof *prof);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
prof	type(fdps_time_prof)	入出力	ツリーオブジェクトの API でかかった時間を受け取るための変数。C 言語の場合、変数のアドレスを引数に指定する必要があることに注意。

返り値

なし

機能

ローカルツリー構築、グローバルツリー構築、力の計算 (walk 込)、ローカルツリーのモーメント計算、グローバルツリーのモーメント計算、LET(Local Essential Tree) 構築、LET 交換にかかった時間（ミリ秒単位）を fdps_time_prof 型のメンバ変数の該当部分 make_local_tree, make_global_tree, calc_force, calc_moment_local_tree, calc_moment_global_tree, make_LET_1st_, make_LET_2nd, exchange_LET_1st, exchange_LET_2nd に格納する。長距離力や Short-Scatter 型ツリーの様に LET 交換が 1 段階通信の場合は make_LET_2nd, exchange_LET_2nd に値は格納されない。

8.4.8 clear_tree_time_prof

Fortran 構文

```
subroutine fdps_ctrl%clear_tree_time_prof(tree_num)
```

C 言語 構文

```
void fdps_clear_tree_time_prof(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

なし

機能

FDPS 本体に用意された識別番号 tree_num のツリーオブジェクトの TimeProfile 型プライベートメンバ変数のメンバ変数 make_local_tree, make_global_tree, calc_force, calc_moment_local_tree, calc_moment_global_tree, make_LET_1st, make_LET_2nd, exchange_LET_1st, exchange_LET_2nd の値を 0 クリアする。ここで、TimeProfile 型は Fortran/C 言語インターフェースで用意された fdps_time_prof 型に対応する C++のデータ型のことである (詳細は、FDPS 本体の仕様書を参照)。本 API は時間計測をリセットするために使用する。

8.4.9 get_num_interact_ep_ep_loc

Fortran 構文

```
function fdps_ctrl%get_num_interact_ep_ep_loc(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_interact_ep_ep_loc(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

自プロセス内で計算した EPI と EPJ の相互作用数を返す。

8.4.10 get_num_interact_ep_sp_loc

Fortran 構文

```
function fdps_ctrl%get_num_interact_ep_sp_loc(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_interact_ep_sp_loc(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

自プロセス内で計算した EPI と SPJ の相互作用数を返す。

8.4.11 get_num_interact_ep_ep_glb

Fortran 構文

```
function fdps_ctrl%get_num_interact_ep_ep_glb(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_interact_ep_ep_glb(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

全プロセス内で計算した EPI と EPJ の相互作用数を返す。

8.4.12 get_num_interact_ep_sp_glb

Fortran 構文

```
function fdps_ctrl%get_num_interact_ep_sp_glb(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_interact_ep_sp_glb(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

全プロセスで計算した EPI と SPJ の相互作用数を返す。

8.4.13 clear_num_interact

Fortran 構文

```
subroutine fdps_ctrl%clear_num_interact(tree_num)
```

C 言語 構文

```
void fdps_clear_num_interact(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

なし

機能

EP-EP,EP-SP の local,global の相互作用数を 0 クリアする。

8.4.14 get_num_tree_walk_loc

Fortran 構文

```
function fdps_ctrl%get_num_tree_walk_loc(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_tree_walk_loc(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

自プロセスでの相互作用計算時の tree walk 数を返す。

8.4.15 get_num_tree_walk_glb

Fortran 構文

```
function fdps_ctrl%get_num_tree_walk_glb(tree_num)
```

C 言語 構文

```
long long int fdps_get_num_tree_walk_glb(const int tree_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。

返り値

integer(kind=c_long_long) 型。

機能

全プロセスでの相互作用計算時の tree walk 数を返す。

8.4.16 set_particle_local_tree

Fortran 構文

```
subroutine fdps_ctrl%set_particle_local_tree(tree_num, &
                                             psys_num, &
                                             clear)
```

C 言語 構文

```
void fdps_set_particle_local_tree(const int tree_num,
                                  const int psys_num,
                                  const _Bool clear);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
clear	logical(kind=c_bool)	入力	前に読込んだ粒子データをクリアするかどうか決定するフラグ。.true. (Fortran の場合)/true (C 言語の場合) でクリアする。Fortran の場合、この引数は省略可能引数で、デフォルト値は.true. である。

返り値

なし

機能

識別番号 tree_num のツリーオブジェクトに、識別番号 psys_num の粒子群オブジェクトが保持する粒子データを読み込ませる。引数 clear が.true. (Fortran の場合)/true (C 言語の場合) ならば前に読込んだ粒子情報をクリアし、.false. (Fortran の場合)/false (C 言語の場合) ならクリアしない。.false./false の場合、新しく読み込む粒子データは、これまで読み込まれた粒子データの後に (メモリ上連続して) 格納される。

8.4.17 get_force

Fortran 構文

```
subroutine fdps_ctrl%get_force(tree_num, &
                               i, &
                               force)
```

C 言語 構文

```
void fdps_get_force(const int tree_num,
                    const fdps_s32 i,
                    const void *cptr_to_force);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
i	integer(kind=c_int)	入力	粒子配列のインデックス。
force	Force 型	入出力	i 番目に読み込まれた粒子の相互作用計算の結果を格納する変数。
cptr_to_force	void *	入出力	i 番目に読み込まれた粒子の相互作用計算の結果を格納する変数のアドレス。

返り値

なし

機能

識別番号 tree_num のツリーオブジェクトが API (fdps_)set_particle_local_tree で i 番目に読み込んだ粒子の受ける作用を返す。i が取りうる最小値は、Fortran では 1、C 言語では 0 である。force のデータ型は、当該ツリーオブジェクトを生成するときに使用した派生データ型と同じでなければならない。同様、cptr_to_force が指す先のデータの型は、当該ツリーオブジェクトを生成するときに仕様した構造体と同じでなければならない。

8.4.18 calc_force_all_and_write_back

Fortran 構文 (短距離力の場合)

```
subroutine fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
                                                   pfunc_ep_ep, &
                                                   psys_num,    &
                                                   dinfo_num,   &
                                                   list_mode)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。
list_mode	integer(kind=c_int)	入力	相互作用リストを使い回すかを決定する変数 (詳細は「機能」の欄を参照のこと)。

返り値

なし

機能

短距離版。識別番号 psys_num で指定された粒子群オブジェクトの粒子すべての相互作用を計算し、その計算結果を粒子群オブジェクトに書き戻す。関数ポインタとして渡される関数は第 5.2 節で述べたインターフェースとなっている必要がある。

引数 list_mode は第 4.6.2 節で説明した相互作用リストモード型で、相互作用リストの使い回し (再利用) に関する振舞を制御するための変数である。値は、fdps_make_list、fdps_make_list_for_reuse、fdps_reuse_list のいずれかでなければならない。これ以外が指定された場合の動作は不定である。引数の値が fdps_make_list ならば、新たに相互作用リストを作成し、相互作用計算を行う。この際に作成した相互作用リストの情報は FDPS 内部に保持されず、次の相互作用計算時に再利用することはできない。値が fdps_make_list_for_

reuse のときは、新たに相互作用リストを作成し相互作用計算を行う。作成した相互作用リストを FDPS 内部に保持するため、次回の相互作用計算時に、今回作った相互作用リストを再利用して相互作用計算を行うことができる。値が fdps_reuse_list ならば、前回 fdps_make_list_for_reuse を選んだ際に作成した相互作用リストを再利用して相互作用計算を行う。引数が省略された場合、デフォルト値 fdps_make_list が採用される。

Fortran 構文 (長距離力の場合)

```
subroutine fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
                                                   pfunc_ep_ep, &
                                                   pfunc_ep_sp, &
                                                   psys_num,    &
                                                   dinfo_num,   &
                                                   list_mode)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	type(c_funptr)	入力	EPI と SPJ 間の相互作用を計算する関数ポインタ。
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。
list_mode	integer(kind=c_int)	入力	相互作用リストを使い回すかを決定する変数 (詳細は Fortran 構文 (短距離力の場合) の「機能」の欄を参照のこと)。

返り値

なし

機能

長距離版。関数ポインタを 2 つ取る点を除いて短距離版と同じ。

C 言語 構文 (短距離力・長距離力共用)

```
void fdps_calc_force_all_and_write_back(const int tree_num,
                                        void *(pfunc_ep_ep)(void *, int,
void *, int, void *),
                                        void *(pfunc_ep_sp)(void *, int,
void *, int, void *),
                                        const int psys_num,
                                        const int dinfo_num,
                                        const _Bool clear,
                                        const int list_mode);
```

仮引数仕様

仮引数名	定義
tree_num	ツリーオブジェクトの識別番号。
pfunc_ep_ep	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	EPI と SPJ 間の相互作用を計算する関数ポインタ。識別番号tree_num のツリーオブジェクトが短距離力用のツリーの場合には使用されない。その場合、NULL ポインタを指定しておけばよい。
psys_num	粒子群オブジェクトの識別番号。
dinfo_num	領域情報オブジェクトの識別番号。
clear	前回の相互作用計算の結果をクリアするかを指定するためのフラグ。true の場合、クリアする。
list_mode	相互作用リストを使い回すかを決定する変数 (詳細は Fortran 構文 (短距離力の場合) の「機能」の欄を参照のこと)。ただし、Fortran の場合と次の相違点がある。(i) 引数は省略可能ではない。もし負の整数値が指定された場合、自動的に Fortran におけるデフォルト値に設定される。(ii) C 言語の相互作用リストモード型を使って指定する必要がある。

返り値

なし

機能

Fortran 版の API の説明を参照のこと。

8.4.19 calc_force_all

Fortran 構文 (短距離力の場合)

```
subroutine fdps_ctrl%calc_force_all(tree_num,    &
                                    pfunc_ep_ep, &
                                    psys_num,    &
                                    dinfo_num,   &
                                    list_mode)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。
list_mode	integer(kind=c_int)	入力	相互作用リストを使い回すかを決定する変数 (詳細は API (fdps_)calc_force_all_and_write_back の Fortran 構文 (短距離力の場合) の「機能」の欄を参照のこと)。

返り値

なし

機能

短距離版。API calc_force_all_and_write_back から計算結果の書き戻しがなくなったもの。

Fortran 構文 (長距離力の場合)

```
subroutine fdps_ctrl%calc_force_all(tree_num,    &
                                    pfunc_ep_ep, &
                                    pfunc_ep_sp, &
                                    psys_num,    &
                                    dinfo_num,   &
                                    list_mode)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	type(c_funptr)	入力	EPI と SPJ 間の相互作用を計算する関数ポインタ。
psys_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。
list_mode	integer(kind=c_int)	入力	相互作用リストを使い回すかを決定する変数 (詳細は API (fdps_)calc_force_all_and_write_back の Fortran 構文 (短距離力の場合) の「機能」の欄を参照のこと)。

返り値

なし

機能

長距離版。関数ポインタを 2 つ取る点を除いて短距離版と同じ。

C 言語 構文 (短距離力・長距離力共用)

```
void fdps_calc_force_all(const int tree_num,
                         void *(pfunc_ep_ep)(void *, int, void *, int,
void *),
                         void *(pfunc_ep_sp)(void *, int, void *, int,
void *),
                         const int psys_num,
                         const int dinfo_num,
                         const _Bool clear,
                         const int list_mode);
```

仮引数仕様

仮引数名	定義
tree_num	ツリーオブジェクトの識別番号。
pfunc_ep_ep	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	EPI と SPJ 間の相互作用を計算する関数ポインタ。識別番号tree_num のツリーオブジェクトが短距離力用のツリーの場合には使用されない。その場合、NULL ポインタを指定しておけばよい。
psys_num	粒子群オブジェクトの識別番号。
dinfo_num	領域情報オブジェクトの識別番号。
clear	前回の相互作用計算の結果をクリアするかを指定するためのフラグ。true でクリアする。
list_mode	相互作用リストを使い回すかを決定する変数 (詳細は API (fdps_)calc_force_all_and_write_back の C 言語構文の記述を参照のこと)。

返り値

なし

機能

Fortran 版の API の説明を参照のこと。

8.4.20 calc_force_making_tree

Fortran 構文 (短距離力の場合)

```
subroutine fdps_ctrl%calc_force_making_tree(tree_num,    &
                                            pfunc_ep_ep, &
                                            dinfo_num)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	type(c_funptr)	入力	EPI と SPJ 間の相互作用を計算する関数ポインタ。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。

返り値

なし

機能

短距離版。ツリーオブジェクトに読み込まれた粒子群オブジェクトの粒子すべての相互作用を計算する。API calc_force_all_and_write_back に対して、粒子群オブジェクトからの粒子読み込みと計算結果の書き戻しがなくなったもの。

Fortran 構文 (長距離力の場合)

```
subroutine fdps_ctrl%calc_force_making_tree(tree_num,    &
                                            pfunc_ep_ep, &
                                            pfunc_ep_sp, &
                                            dinfo_num)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	type(c_funptr)	入力	EPI と SPJ 間の相互作用を計算する関数ポインタ。
dinfo_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

なし

機能

長距離版。関数ポインタを 2 つ取る点を除いて短距離版と同じ。

C 言語 構文 (短距離力・長距離力共用)

```
void fdps_calc_force_making_tree(const int tree_num,
                                 void *(pfunc_ep_ep)(void *, int, void
*, int, void *),
                                 void *(pfunc_ep_sp)(void *, int, void
*, int, void *),
                                 const int dinfo_num,
                                 const _Bool clear);
```

仮引数仕様

仮引数名	定義
tree_num	ツリーオブジェクトの識別番号。
pfunc_ep_ep	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	EPI と SPJ 間の相互作用を計算する関数ポインタ。識別番号tree_num のツリーオブジェクトが短距離力用のツリーの場合には使用されない。その場合、NULL ポインタを指定しておけばよい。
dinfo_num	粒子群オブジェクトの識別番号。
clear	前回の相互作用計算の結果をクリアするかを指定するためのフラグ。true でクリアする。

返り値

なし

機能

Fortran 版の API の説明を参照のこと。

8.4.21 calc_force_and_write_back

Fortran 構文 (短距離力の場合)

```
subroutine fdps_ctrl%calc_force_and_write_back(tree_num,   &
                                               func_ep_ep, &
                                               psys_num)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
dinfo_num	integer(kind=c_int)	入力	粒子群オブジェクトの識別番号。

返り値

なし

機能

短距離版。calc_force_all_and_write_back に対して、粒子群オブジェクトからの粒子読込、ローカルツリーの構築、グローバルツリーの構築、グローバルツリーのモーメントの計算がなくなったもの。

Fortran 構文 (長距離力の場合)

```
subroutine fdps_ctrl%calc_force_and_write_back(tree_num,    &
                                               pfunc_ep_ep, &
                                               pfunc_ep_sp, &
                                               psys_num)
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pfunc_ep_ep	type(c_funptr)	入力	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	type(c_funptr)	入力	EPI と SPJ 間の相互作用を計算する関数ポインタ。
dinfo_num	integer(kind=c_int)	入力	領域情報オブジェクトの識別番号。

返り値

なし

機能

長距離版。関数ポインタを 2 つ取る点を除いて短距離版と同じ。

C 言語 構文 (短距離力・長距離力共用)

```
void fdps_calc_force_and_write_back(const int tree_num,
                                    void *(pfunc_ep_ep)(void *, int,
void *, int, void *),
                                    void *(pfunc_ep_sp)(void *, int,
void *, int, void *),
                                    const int psys_num,
                                    const _Bool clear);
```

仮引数仕様

仮引数名	定義
tree_num	ツリーオブジェクトの識別番号。
pfunc_ep_ep	EPI と EPJ 間の相互作用を計算する関数ポインタ。
pfunc_ep_sp	EPI と SPJ 間の相互作用を計算する関数ポインタ。識別番号tree_num のツリーオブジェクトが短距離力用のツリーの場合には使用されない。その場合、NULL ポインタを指定しておけばよい。
dinfo_num	領域情報オブジェクトの識別番号。
clear	前回の相互作用計算の結果をクリアするかを指定するためのフラグ。true でクリアする。

返り値

なし

機能

Fortran 版の API の説明を参照のこと。

8.4.22 get_neighbor_list

Fortran 構文

```
subroutine fdps_ctrl%get_neighbor_list(tree_num, &
                                       pos,      &
                                       r_search, &
                                       num_epj,  &
                                       fptr_to_EPJ)
```

C 言語 構文

```
void fdps_get_neighbor_list(const int tree_num,
                            const fdps_f64vec *pos,
                            const fdps_f64 r_search,
                            int *num_epj,
                            void **cptr_to_epj);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
pos	type(fdps_f64vec)	入力	近傍粒子を求めたい粒子の位置。C 言語では引数に変数のアドレスを指定する必要があることに注意。
r_search	real(kind=c_double)	入力	近傍粒子を求めたい粒子の探索半径。
num_epj	integer(kind=c_int)	入出力	探索して求めた近傍粒子数を格納するための変数。C 言語では引数に変数のアドレスを指定する必要があることに注意。
fptr_to_EPJ	EssentialParticleJ 型, dimension(:), pointer	入出力	近傍粒子として同定された EssentialParticleJ 型粒子へのポインタ。
cptr_to_epj	void **	入出力	近傍粒子として同定された EssentialParticleJ 型粒子の配列の先頭アドレスを格納する変数のアドレス。近傍粒子配列の先頭アドレスは void *型として返ってくるため、void *型を格納できる変数のアドレスを指定する必要がある。

返り値

なし

機能

識別番号 tree_num のツリーオブジェクトを使って、位置 pos、探索半径 r_search の粒子に対して近傍粒子探索を行い、近傍粒子数および近傍粒子の粒子配列へのポインタを返す。この粒子配列のデータ型は、ツリーオブジェクト作成時に指定した EssentialParticleJ 型である必要がある。

8.4.23 get_epj_from_id

Fortran 構文

```
subroutine fdps_ctrl%get_epj_from_id(tree_num, &
                                     id,       &
                                     fptr_to_EPJ)
```

C 言語 構文

```
void * fdps_get_epj_from_id(const int tree_num,
                            const fdps_s64 id);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
id	type(kind=c_long_long)	入力	取得したい粒子の id。
fptr_to_EPJ	EssentialParticleJ 型, pointer	入出力	EssentialParticleJ 型へのポインタ。

返り値

Fortran の場合にはなし、C 言語の場合には void *型。

機能

識別番号 tree_num のツリーオブジェクトの生成時に指定された EssentialParticleJ(EPJ) 型が、メンバ変数に粒子 id を持つ場合に使用可能 (対応するメンバ変数には id であることを示す FDPS 指示文が必要)。Fortran では、引数 fptr_to_EPJ に、引数 id で指定された粒子 id を持つ EPJ のポインタをセットする。対応する EPJ がない場合は、fptr_to_EPJ は未結合状態 (NULL() の状態) となる (組み込み関数 associated で結合状態を判定可能)。また、複数の EPJ が同じ id を持つ場合結果は保証されない。メンバ変数が粒子 id であることを指示する指示文については、第 5 章を参照。以下に使用例を示す。

Listing 8.4: 例

```
integer(kind=c_long_long) :: id
type(essential_particle_j), pointer :: epj

call fdps_ctrl%get_epj_from_id(tree_num,id,epj)
if (associated(epj)) then
   ! Do something using epj
   write(*,*)'id␣=␣',epj%id
else
   write(*,*)'epj␣is␣NULL'
end if
```

C 言語の場合、引数 id で指定された粒子 id を持つ EPJ のアドレスが返り値として返ってくる。対応する EPJ がない場合の振る舞いは NULL ポインタが返ってくる。

8.4.24 set_tree_comm_info

Fortran 構文

```
subroutine fdps_ctrl%set_tree_comm_info(tree_num, ci)
```

C 言語 構文

```
void fdps_fdps_set_tree_comm_info(int tree_num,
                                  int ci);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
ci	integer(kind=c_int)	入力	コミュニケータクラスに対応する番号。

返り値

なし

機能

通信に使うコミュニケータを指定する。

8.4.25 set_exhange_let_mode

Fortran 構文

```
subroutine fdps_ctrl%set_exhange_let_mode(tree_num, ci)
```

C 言語 構文

```
void fdps_fdps_set_exhange_let_mode(int tree_num,
                                    int mode);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
tree_num	integer(kind=c_int)	入力	ツリーオブジェクトの識別番号。
ci	integer(kind=c_int)	入力	enum EXCHANGE_LET_MODE

返り値

なし

機能

LET 交換の方法を決定する。

8.5 コミュニケータ操作用 API

本節では、MPI コミュニケータを操作する API について説明する。
コミュニケータを操作する API の名称の一覧を以下に示す:

```
(fdps_)ci_initialize
(fdps_)ci_set_communicator
(fdps_)ci_delete
(fdps_)ci_create
(fdps_)ci_split
```

これらの関数群は、 MPI コミュニケータに対応したテーブルをもち、そのテーブルインデックスを通して MPI コミュニケータを操作する。このインデックスを ci_がついた通信関数、さらに set_dinfo_comm_info 等の関数で FDPS のデータクラスに与えることで、FDPS で指定したコミュニケータを使うことができる。
この機能により、単一プログラムの中で複数の FDPS インスタンスの時間積分を並行して行なうことができる。
以下、順に各 API の仕様を記述していく。

8.5.1 ci_initialize

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%ci_initialize(comm)
```

C 言語 構文

```
int fdps_ci_initialize(int comm);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
comm	integer(kind=c_int)	入力	MPI コミュニケータ (Fortran API)

返り値

integer(kind=c_int) 型。コミュニケータに対応するインデックスを返す。

機能

コミュニケータに対応するインデックスを返す。入力の MPI コミュニケータは Fortran API であることに注意する。すなわち、 引数は C 言語 API における MPI コミュニケータ (MPI_Comm 型) ではなく、それを MPI_Comm_c2f 関数で変換した Fortran 言語でのコミュニケータでなければならない。

8.5.2 ci_set_communicator

Fortran 構文

```
subroutine fdps_ctrl%ci_set_communicator(ci, comm)
```

C 言語 構文

```
void fdps_ci_set_communicator(int ci, int comm);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
ci	integer(kind=c_int)	入力	コミュニケータインデックス
comm	integer(kind=c_int)	入力	MPI コミュニケータ (Fortran API)

返り値

なし。

機能

あるインデックスの MPI コミュニケータを変更する。

8.5.3 ci_delete

Fortran 構文

```
subroutine fdps_ctrl%ci_delete(ci, comm)
```

C 言語 構文

```
void fdps_ci_delete(int ci, int comm);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
ci	integer(kind=c_int)	入力	コミュニケータインデックス
comm	integer(kind=c_int)	入力	MPI コミュニケータ (Fortran API)

返り値

なし。

機能

あるインデックスの MPI コミュニケータを削除 (MPI_Comm_free) する。このインデックスは「未使用」状態となり、ci_initialize によって新しいコミュニケータが割り当てられるまで使えない。

8.5.4 ci_create

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%ci_create(ci, n, rank)
```

C 言語 構文

```
int fdps_ci_create(int ci, int n, int rank[]);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
ci	integer(kind=c_int)	入力	コミュニケータインデックス
n	integer(kind=c_int)	入力	生成されるコミュニケータに所属するプロセスの数
rank	integer(kind=c_int), dimension(n)	入力	生成されるコミュニケータに所属するプロセスのランクの配列。

返り値

integer(kind=c_int) 型。生成されたコミュニケータに対応するインデックスを返す。

機能

呼び出しもとの ci に対応する MPI コミュニケータから新たなコミュニケータを作成する。配列 rank で表されるプロセスが所属するコミュニケータを作成し対応するインデックスを返す。

8.5.5 ci_split

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%ci_split(ci, n, rank)
```

C 言語 構文

```
int fdps_ci_split(int ci, int color, int key);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
ci	integer(kind=c_int)	入力	コミュニケータインデックス
color	integer(kind=c_int)	入力	これが同じプロセスは同一のコミュニケータに属する
key	integer(kind=c_int)	入力	同一コミュニケータの中での順序を与える

返り値

integer(kind=c_int) 型。生成されたコミュニケータに対応するインデックスを返す。

機能

呼び出しもとの ci に対応する MPI コミュニケータを分割する．同じ color のプロセスは同一コミュニケータに所属し、key の小さいものから順にそのコミュニケータでのランクが割り振られる．

8.6 通信用 API

通信関係の全 API の名称の一覧を以下に示す:

```
(fdps_)(ci_)get_rank
(fdps_)(ci_)get_rank_multi_dim
(fdps_)(ci_)get_num_procs
(fdps_)(ci_)get_num_procs_multi_dim
(fdps_)(ci_)get_logical_and
(fdps_)(ci_)get_logical_or
(ci_)get_min_value (Fortran のみ)
fdps_(ci_)get_min_value_s32 (C 言語のみ)
fdps_(ci_)get_min_value_s64 (C 言語のみ)
fdps_(ci_)get_min_value_u32 (C 言語のみ)
fdps_(ci_)get_min_value_u64 (C 言語のみ)
fdps_(ci_)get_min_value_f32 (C 言語のみ)
fdps_(ci_)get_min_value_f64 (C 言語のみ)
fdps_(ci_)get_min_value_w_id_f32 (C 言語のみ)
fdps_(ci_)get_min_value_w_id_f64 (C 言語のみ)
(ci_)get_max_value (Fortran のみ)
fdps_(ci_)get_max_value_s32 (C 言語のみ)
fdps_(ci_)get_max_value_s64 (C 言語のみ)
fdps_(ci_)get_max_value_u32 (C 言語のみ)
fdps_(ci_)get_max_value_u64 (C 言語のみ)
fdps_(ci_)get_max_value_f32 (C 言語のみ)
fdps_(ci_)get_max_value_f64 (C 言語のみ)
fdps_(ci_)get_max_value_w_id_f32 (C 言語のみ)
fdps_(ci_)get_max_value_w_id_f64 (C 言語のみ)
```

```
(ci_)get_sum (Fortran のみ)
fdps_(ci_)get_sum_s32 (C 言語のみ)
fdps_(ci_)get_sum_s64 (C 言語のみ)
fdps_(ci_)get_sum_u32 (C 言語のみ)
fdps_(ci_)get_sum_u64 (C 言語のみ)
fdps_(ci_)get_sum_f32 (C 言語のみ)
fdps_(ci_)get_sum_f64 (C 言語のみ)
(ci_)broadcast (Fortran のみ)
fdps_(ci_)broadcast_s32 (C 言語のみ)
fdps_(ci_)broadcast_s64 (C 言語のみ)
fdps_(ci_)broadcast_u32 (C 言語のみ)
fdps_(ci_)broadcast_u64 (C 言語のみ)
fdps_(ci_)broadcast_f32 (C 言語のみ)
fdps_(ci_)broadcast_f64 (C 言語のみ)
(fdps_)(ci_)get_wtime
(fdps_)(ci_)barrier
```

以下、順に各 API の仕様を記述していく。ただし、API 名が次の正規表現パターンにマッチするものは単一の節でまとめて説明を行う：*get_min_value*、*get_max_value*、*get_sum*、*broadcast*。

関数名に ci_ があるものは、MPI コミュニケータに対応するインデックスを引数にとることができる。この記述は煩雑になるため (fdps_)ci_get_rank についてのみ示す。

8.6.1 get_rank

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%get_rank()
```

C 言語 構文

```
int fdps_get_rank();
```

仮引数仕様

なし。

返り値

integer(kind=c_int) 型。全プロセス中でのランクを返す。

機能

全プロセス中でのランクを返す。

8.6.2 ci_get_rank

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%ci_get_rank(ci)
```

C 言語 構文

```
int fdps_ci_get_rank(int ci);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
ci	integer(kind=c_int)	入力	コミュニケータ番号。

返り値

integer(kind=c_int) 型。コミュニケータ中でのランクを返す。

機能

コミュニケータ中でのランクを返す。

8.6.3 get_rank_multi_dim

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%get_rank_multi_dim(id)
```

C 言語 構文

```
int fdps_get_rank_multi_dim(const int id);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
id	integer(kind=c_int)	入力	軸の番号。x 軸:0, y 軸:1, z 軸:2。

返り値

integer(kind=c_int) 型。id 番目の軸でのランクを返す。2 次元の場合、id=2 は 1 を返す。

機能

id 番目の軸でのランクを返す。2 次元の場合、id=2 は 1 を返す。

8.6.4 get_num_procs

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%get_num_procs()
```

C 言語 構文

```
int fdps_get_num_procs();
```

仮引数仕様

なし。

返り値

integer(kind=c_int) 型。全プロセス数を返す。

機能

全プロセス数を返す。

8.6.5 get_num_procs_multi_dim

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%get_num_procs_multi_dim(id)
```

C 言語 構文

```
int fdps_get_num_procs_multi_dim(const int id);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
id	integer(kind=c_int)	入力	軸の番号。x 軸:0, y 軸:1, z 軸:2。

返り値

integer(kind=c_int) 型。id 番目の軸のプロセス数を返す。2 次元の場合、id=2 は 1 を返す。

機能

id 番目の軸のプロセス数を返す。2 次元の場合、id=2 は 1 を返す。

8.6.6 get_logical_and

Fortran 構文

```
subroutine fdps_ctrl%get_logical_and(f_in, &
                                     f_out)
```

C 言語構文

```
_Bool fdps_get_logical_and(const _Bool in);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	logical(kind=c_bool)	入力	入力の論理値
f_out	logical(kind=c_bool)	入出力	出力の論理値
in	const _Bool	入力	入力の論理値

返り値

Fortran の場合はなし。C 言語の場合は _Bool 型。

機能

Fortran の場合、全プロセスでの f_in の論理積をとり f_out にいれる。C 言語の場合、全プロセスでの in の論理積をとり、その結果を返す。

8.6.7 get_logical_or

Fortran 構文

```
subroutine fdps_ctrl%get_logical_or(f_in, &
                                    f_out)
```

C 言語 構文

```
_Bool fdps_get_logical_or(const _Bool in);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	logical(kind=c_bool)	入力	入力の論理値
f_out	logical(kind=c_bool)	入出力	出力の論理値
in	const _Bool	入力	入力の論理値

返り値

Fortran の場合はなし。C 言語の場合は_Bool 型。

機能

Fortran の場合、全プロセスでの f_in の論理和をとり f_out にいれる。C 言語の場合、全プロセスでの in の論理和をとり、その結果を返す。

8.6.8 get_min_value

Fortran 構文 (1)

```
subroutine fdps_ctrl%get_min_value(f_in, &
                                   f_out)
```

C 言語 構文 (1)

```
fdps_s32 fdps_get_min_value_s32(const fdps_s32 f_in);
fdps_s64 fdps_get_min_value_s64(const fdps_s64 f_in);
fdps_u32 fdps_get_min_value_u32(const fdps_u32 f_in);
fdps_u64 fdps_get_min_value_u64(const fdps_u64 f_in);
fdps_f32 fdps_get_min_value_f32(const fdps_f32 f_in);
fdps_f64 fdps_get_min_value_f64(const fdps_f64 f_in);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	Fortran の場合、以下のいずれか: integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double) C 言語の場合、以下のいずれか: fdps_s32, fdps_s64 fdps_u32, fdps_u64 fdps_f32, fdps_f64	入力	入力値
f_out	入力と同じ	入出力	出力値

返り値

Fortran の場合はなし。C 言語の場合は入力値と同じデータ型。

機能

Fortran の場合、全プロセスで f_in の最小値を取り、結果を f_out に代入する。C 言語の場合、全プロセスで f_in の最小値を取り、その結果を返す。

最小値の他、最小値に対応したインデックスも返す API もある。これは以下のようになる。

Fortran 構文 (2)

```
subroutine fdps_ctrl%get_min_value(f_in, &
                                   i_in, &
                                   f_out,&
                                   i_out)
```

C 言語 構文 (2)

```
void fdps_get_min_value_w_id_f32(const fdps_f32 f_in,
                                 const int i_in,
                                 fdps_f32 *f_out,
                                 int *i_out);
void fdps_get_min_value_w_id_f64(const fdps_f64 f_in,
                                 const int i_in,
                                 fdps_f64 *f_out,
                                 int *i_out);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	real(kind=c_float) real(kind=c_double)	入力	入力値
i_in	integer(kind=c_int)	入力	入力値に対応するインデックス
f_out	f_in と同じ	入出力	出力値
i_out	integer(kind=c_int)	入出力	出力値に対応するインデックス

返り値

なし。

機能

全プロセスで f_in の最小値を取り、結果を f_out に格納する。さらに、その値に対応する i_in の値を i_out に格納する。

8.6.9 get_max_value

Fortra 構文 (1)

```
subroutine fdps_ctrl%get_max_value(f_in, &
                                   f_out)
```

C 言語 構文 (1)

```
fdps_s32 fdps_get_max_value_s32(const fdps_s32 f_in);
fdps_s64 fdps_get_max_value_s64(const fdps_s64 f_in);
fdps_u32 fdps_get_max_value_u32(const fdps_u32 f_in);
fdps_u64 fdps_get_max_value_u64(const fdps_u64 f_in);
fdps_f32 fdps_get_max_value_f32(const fdps_f32 f_in);
fdps_f64 fdps_get_max_value_f64(const fdps_f64 f_in);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	Fortran の場合、以下のいずれか: integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double) C 言語の場合、以下のいずれか: fdps_s32, fdps_s64 fdps_u32, fdps_u64 fdps_f32, fdps_f64	入力	入力値
f_out	入力と同じ	入出力	出力値

返り値

Fortran の場合はなし。C 言語の場合は入力値と同じデータ型。

機能

全プロセスで `f_in` の最大値を取り、結果を返す。

最大値の他、最大値に対応したインデックスも返す API もある。これは以下のようになる。

Fortran 構文 (2)

```
subroutine fdps_ctrl%get_max_value(f_in, &
                                   i_in, &
                                   f_out,&
                                   i_out)
```

C 言語 構文 (2)

```
void fdps_get_max_value_w_id_f32(const fdps_f32 f_in,
                                 const int i_in,
                                 fdps_f32 *f_out,
                                 int *i_out);
void fdps_get_max_value_w_id_f64(const fdps_f64 f_in,
                                 const int i_in,
                                 fdps_f64 *f_out,
                                 int *i_out);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	real(kind=c_float) real(kind=c_double)	入力	入力値
i_in	integer(kind=c_int)	入力	入力値に対応するインデックス
f_out	f_in と同じ	入出力	出力値
i_out	integer(kind=c_int)	入出力	出力値に対応するインデックス

返り値

Fortran の場合はなし。C 言語の場合は入力値と同じデータ型。

機能

全プロセスで f_in の最大値を取り、結果を f_out に格納する。さらに、その値に対応する i_in の値を i_out に格納する。

8.6.10 get_sum

Fortran 構文

```
subroutine fdps_ctrl%get_sum(f_in, &
                             f_out)
```

C 言語 構文

```
fdps_s32 fdps_get_sum_s32(const fdps_s32 f_in);
fdps_s64 fdps_get_sum_s64(const fdps_s64 f_in);
fdps_u32 fdps_get_sum_u32(const fdps_u32 f_in);
fdps_u64 fdps_get_sum_u64(const fdps_u64 f_in);
fdps_f32 fdps_get_sum_f32(const fdps_f32 f_in);
fdps_f64 fdps_get_sum_f64(const fdps_f64 f_in);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	Fortran の場合、以下のいずれか: integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double) C 言語の場合、以下のいずれか: fdps_s32, fdps_s64 fdps_u32, fdps_u64 fdps_f32, fdps_f64	入力	入力値
f_out	入力と同じ	入出力	出力値

返り値

Fortran の場合はなし。C 言語の場合は入力値と同じデータ型。

機能

全プロセスで `f_in` の総和を取り、結果を返す。

8.6.11 broadcast

Fortran 構文

```
subroutine fdps_ctrl%broadcast(val, &
                               n,   &
                               src)
```

C 言語 構文

```
void fdps_broadcast_s32(fdps_s32 *val, int n, int src);
void fdps_broadcast_s64(fdps_s64 *val, int n, int src);
void fdps_broadcast_u32(fdps_u32 *val, int n, int src);
void fdps_broadcast_u64(fdps_u64 *val, int n, int src);
void fdps_broadcast_f32(fdps_f32 *val, int n, int src);
void fdps_broadcast_f64(fdps_f64 *val, int n, int src);
```

仮引数仕様

返り値

なし。

機能

ランク番号 src のプロセスが n 個の val で指定される n 個の変数を全プロセスに放送する。結果は val に格納される。

仮引数名	データ型	入出力属性	定義
val	Fortran の場合、以下のいずれかの型の変数または配列: integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double) C 言語の場合、以下のいずれかの型の変数または配列 fdps_s32, fdps_s64 fdps_u32, fdps_u64 fdps_f32, fdps_f64	入出力	入力値。C 言語では引数に変数のアドレスを指定する必要があることに注意。
n	integer(kind=c_int)	入力	入力値の数。スカラー変数の場合には 1 を、配列の場合には配列サイズを指定する。
src	integer(kind=c_int)	入力	放送するプロセスのランク番号

8.6.12 get_wtime

Fortran 構文

```
real(kind=c_double) fdps_ctrl%get_wtime()
```

C 言語 構文

```
double fdps_get_wtime();
```

仮引数仕様

なし。

返り値

real(kind=c_double) 型。ウォールクロックタイムを返す。単位は秒。

機能

ウォールクロックタイムを返す。単位は秒。

8.6.13 barrier

Fortran 構文

```
subroutine fdps_ctrl%barrier()
```

C 言語 構文

```
void fdps_barrier();
```

仮引数仕様

なし。

返り値

なし。

機能

プロセス間の同期を取る。

8.7 Particle Mesh 用 API

本節では、FDPS 拡張機能 Particle Mesh を使用するための API を記述する。FDPS 本体において、Particle Mesh 計算に必要なデータは ParticleMesh オブジェクト (以後、単に **PM** オブジェクト) で管理される。他のオブジェクトと同様、Fortran/C 言語 インターフェースでは、PM オブジェクトを識別番号で管理する。

PM オブジェクトを操作する全 API の名称の一覧を以下に示す:

```
(fdps_)create_pm
(fdps_)delete_pm
(fdps_)get_pm_mesh_num
(fdps_)get_pm_cutoff_radius
(fdps_)set_dinfo_of_pm
(fdps_)set_psys_of_pm
(fdps_)get_pm_force
(fdps_)get_pm_potential
(fdps_)calc_pm_force_only
(fdps_)calc_pm_force_all_and_write_back
```

以下、順に、各 API の仕様を記述していく。

8.7.1 create_pm

Fortran 構文

```
subroutine fdps_ctrl%create_pm(pm_num)
```

C 言語 構文

```
void fdps_create_pm(int *pm_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入出力	PM オブジェクトの識別番号を受け取るための変数。C 言語では変数のアドレスを引数に指定する必要があることに注意。

返り値

なし。

機能

メモリ上に、Particle Mesh 計算で使用される PM オブジェクトを生成し、そのオブジェクトの識別番号を返す。

8.7.2 delete_pm

Fortran 構文

```
subroutine fdps_ctrl%delete_pm(pm_num)
```

C 言語 構文

```
void fdps_delete_pm(const int pm_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。

返り値

なし

機能

メモリ上から、識別番号 pm_num の PM オブジェクトを削除する。

8.7.3 get_pm_mesh_num

Fortran 構文

```
integer(kind=c_int) fdps_ctrl%get_pm_mesh_num()
```

C 言語 構文

```
int fdps_get_pm_mesh_num();
```

仮引数仕様

なし。

返り値

Particle Mesh 計算で使用されるメッシュの 1 次元方向当たりのメッシュ数。integer(kind=c_int) 型。

機能

Particle Mesh 計算に使用されるメッシュの 1 次元方向あたりのメッシュ数を返す。

8.7.4 get_pm_cutoff_radius

Fortran 構文

```
real(kind=c_double) fdps_ctrl%get_pm_cutoff_radius()
```

C 言語 構文

```
double fdps_get_pm_cutoff_radius();
```

仮引数仕様

なし。

返り値

Particle Mesh 計算に使用されるカットオフ半径。カットオフ半径はメッシュの格子間隔で規格化されている。real(kind=c_double) 型。

機能

Particle Mesh 計算で使われるカットオッフ半径を、メッシュ間隔で規格化された値として返す。

8.7.5 set_dinfo_of_pm

Fortran 構文

```
subroutine fdps_ctrl%set_dinfo_of_pm(pm_num,dinfo_num)
```

C 言語 構文

```
void fdps_set_dinfo_of_pm(const int pm_num,
                          const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	Particle Mesh 計算の対象となる粒子群オブジェクトに関連した領域情報オブジェクトの識別番号。

返り値

なし。

機能

識別番号 pm_num を持つ PM オブジェクトに、領域情報オブジェクトの識別番号をセットする。ここでセットされる領域情報オブジェクトは、FDPS が領域情報を取得するのに使用される。そのため、Particle Mesh 計算の対象となる粒子群オブジェクトと関連付けられたものである必要がある。

8.7.6 set_psys_of_pm

Fortran 構文

```
subroutine fdps_ctrl%set_psys_of_pm(pm_num,psys_num,clear)
```

C 言語 構文

```
void fdps_set_psys_of_pm(const int pm_num,
                         const int psys_num,
                         const _Bool clear);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。
psys_num	integer(kind=c_int)	入力	Particle Mesh 計算の対象となる粒子群オブジェクトの識別番号。
clear	logical(kind=c_bool)	入力	これまで読込んだ粒子情報をクリアするかどうか決定するフラグ。.true. ならばクリアする。Fortran の場合、引数は省略可能で、省略された場合、デフォルト値 .true. が使用される。

返り値

なし。

機能

識別番号 pm_num を持つ PM オブジェクトに、粒子群オブジェクトの識別番号をセットする。ここでセットされる粒子群オブジェクトの粒子情報を使って、FDPS は Particle Mesh 計算を行うことになる。

8.7.7 get_pm_force

Fortran 構文

```
subroutine fdps_ctrl%get_pm_force(pm_num,pos,f)
```

C 言語 構文

```
void fdps_get_pm_force(const int pm_num,
                       const fdps_f32vec *pos,
                       fdps_f32vec *force);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。
pos	Fortran の場合、以下のいずれか: real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec) C 言語では fdps_f32vec *型のみ	入力	メッシュからの力の計算に使用する位置座標。C 言語の場合、引数に変数のアドレスを指定する必要がある。
f	pos と同じデータ型	入出力	位置 pos におけるメッシュからの力。
force	fdps_f32vec *	入出力	位置 pos におけるメッシュからの力。引数に変数のアドレスを指定する必要がある。

コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合は space_dim は 2。それ以外は 3 である。

返り値

なし。

機能

位置 pos でのメッシュからの力を返す。この関数はスレッドセーフである。本 API 実行前に、識別番号 pm_num の PM オブジェクトを使い、後述する API (fdps_)calc_pm_force_only か (fdps_)calc_pm_force_all_and_write_back が少なくとも 1 回は実行されている必要がある。

8.7.8 get_pm_potential

Fortran 構文

```
subroutine fdps_ctrl%get_pm_potential(pm_num,pos,pot)
```

C 言語 構文

```
void fdps_get_pm_potential(const int pm_num,
                           const fdps_f32vec *pos,
                           fdps_f32 *pot);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。
pos	Fortran の場合、以下のいずれか: real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec) C 言語では fdps_f32vec *型のみ	入力	メッシュからのポテンシャルの計算に使用する位置座標。
pot	Fortran では real(kind=c_float) C 言語では fdps_f32	入出力	位置 pos におけるメッシュポテンシャル値。

コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION が定義されている場合は space_dim は 2。それ以外は 3 である。C 言語では引数 pos と pot は、変数のアドレスである。

返り値

なし。

機能

位置 pos でのメッシュポテンシャルの値を返す。この関数はスレッドセーフ である。本 API でポテンシャルの値を取得するためには、事前に、識別番号 pm_num の PM オブジェクトを使い、後述する API (fdps_)calc_pm_force_only か (fdps_)calc_pm_force_all_and_write_back が少なくとも 1 回は実行されている必要がある。

8.7.9 calc_pm_force_only

Fortran 構文

```
subroutine fdps_ctrl%calc_pm_force_only(pm_num)
```

C 言語 構文

```
void fdps_calc_pm_force_only(const int pm_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。

返り値

なし。

機能

識別番号 pm_num の PM オブジェクトを使い、メッシュ上の力を計算する。正しく機能するには、事前に粒子情報や領域情報が PM オブジェクトにセットされている必要がある。

8.7.10 calc_pm_force_all_and_write_back

Fortran 構文

```
subroutine fdps_ctrl%calc_pm_force_all_and_write_back(pm_num,   &
                                                      psys_num, &
                                                      dinfo_num)
```

C 言語 構文

```
void fdps_calc_pm_force_all_and_write_back(const int pm_num,
                                           const int psys_num,
                                           const int dinfo_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
pm_num	integer(kind=c_int)	入力	PM オブジェクトの識別番号。
psys_num	integer(kind=c_int)	入力	Particle Mesh 計算に使用する粒子群オブジェクトの識別番号。
dinfo_num	integer(kind=c_int)	入力	Particle Mesh 計算に使用する領域情報オブジェクトの識別番号。

返り値

なし。

機能

指定された識別番号を持つ粒子群オブジェクト, 領域情報オブジェクト, PM オブジェクトを使って、メッシュ上のポテンシャルおよび力を計算した上で、力のみを粒子群オブジェクトに書き戻す。

8.8 その他の API

本節では Fortran/C 言語 インターフェースに用意されている他の API について記述する。本節で説明する API の名称の一覧を以下に示す:

```
(fdps_)create_mtts
(fdps_)delete_mtts
(fdps_)mtts_init_genrand
(fdps_)mtts_genrand_int31
(fdps_)mtts_genrand_real1
(fdps_)mtts_genrand_real2
(fdps_)mtts_genrand_res53
(fdps_)mt_init_genrand
(fdps_)mt_genrand_int31
(fdps_)mt_genrand_real1
(fdps_)mt_genrand_real2
(fdps_)mt_genrand_res53
```

ここに示された API の内、名称に mt が含まれる API は擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を操作・使用するための API である。

以下、順に各 API の仕様について記述していく。

8.8.1 create_mtts

Fortran 構文

```
subroutine fdps_ctrl%create_mtts(mtts_num)
```

C 言語 構文

```
void fdps_create_mtts(int * mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入出力	疑似乱数生成用オブジェクトの識別番号を受け取る変数。C 言語では変数のアドレスを引数に指定する必要があることに注意。

返り値

なし。

機能

メモリ上にメルセンヌ・ツイスタ (Mersenne twister) を使って疑似乱数を生成するオブジェクトを 1 つ生成し、そのオブジェクトの識別番号を返す。

8.8.2 delete_mtts

Fortran 構文

```
subroutine fdps_ctrl%delete_mtts(mtts_num)
```

C 言語 構文

```
void fdps_delete_mtts(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

なし。

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトをメモリ上から削除する。

8.8.3 mtts_init_genrand

Fortran 構文

```
subroutine fdps_ctrl%mtts_init_genrand(mtts_num,s)
```

C 言語 構文

```
void fdps_mtts_init_genrand(const int mtts_num,
                            const int s);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。
s	integer(kind=c_int)	入力	疑似乱数生成に使用するシード。

返り値

なし

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを初期化する。

8.8.4 mtts_genrand_int31

Fortran 構文

```
function fdps_ctrl%mtts_genrand_int31(mtts_num)
```

C 言語 構文

```
int fdps_mtts_genrand_int31(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

integer(kind=c_int) 型スカラー値

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを使って、[0,0x7fffffff] の範囲で一様な整数乱数を生成する。

8.8.5 mtts_genrand_real1

Fortran 構文

```
function fdps_ctrl%mtts_genrand_real1(mtts_num)
```

C 言語 構文

```
double fdps_mtts_genrand_real1(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

real(kind=c double) 型スカラー値。

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを使って、[0.0,1.0] の範囲で一様な浮動小数点数乱数を生成する。

8.8.6 mtts_genrand_real2

Fortran 構文

```
function fdps_ctrl%mtts_genrand_real2(mtts_num)
```

C 言語 構文

```
double fdps_mtts_genrand_real2(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

real(kind=c double) 型スカラー値。

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを使って、[0.0,1.0) の範囲で一様な浮動小数点数乱数を生成する。

8.8.7 mtts_genrand_real3

Fortran 構文

```
function fdps_ctrl%mtts_genrand_real3(mtts_num)
```

C 言語 構文

```
double fdps_mtts_genrand_real3(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

real(kind=c double) 型スカラー値。

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを使って、(0.0,1.0) の範囲で一様な浮動小数点数乱数を生成する。

8.8.8 mtts_genrand_res53

Fortran 構文

```
function fdps_ctrl%mtts_genrand_res53(mtts_num)
```

C 言語 構文

```
double fdps_mtts_genrand_res53(const int mtts_num);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
mtts_num	integer(kind=c_int)	入力	疑似乱数生成用オブジェクトの識別番号を格納した変数。

返り値

real(kind=c double) 型スカラー値。

機能

識別番号 mtts_num の疑似乱数生成用オブジェクトを使って、[0.0,1.0) の範囲で一様な浮動小数点乱数を生成する。前述した mtts_genrand_real$x$ (x=1-3) は浮動小数点数へ変換するのに 32 ビット整数乱数を使用しているのに対し、本 API では 53 ビット整数乱数を使用している。

8.8.9 mt_init_genrand

Fortran 構文

```
subroutine fdps_ctrl%mt_init_genrand(s)
```

C 言語 構文

```
void fdps_mt_init_genrand(const int s);
```

仮引数仕様

仮引数名	データ型	入出力属性	定義
s	integer(kind=c_int)	入出力	擬似乱数生成に使用するシード。

返り値

なし。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) のオブジェクトを生成し初期化を行う。

8.8.10 mt_genrand_int31

Fortran 構文

```
function fdps_ctrl%mt_genrand_int31()
```

C 言語 構文

```
int fdps_mt_genrand_int31();
```

仮引数仕様

なし。

返り値

integer(kind=c_int) 型スカラー値。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を使って、[0,0x7fffffff] の範囲で一様な整数乱数を生成する。

8.8.11 mt_genrand_real1

Fortran 構文

```
function fdps_ctrl%mt_genrand_real1()
```

C 言語 構文

```
double fdps_mt_genrand_real1();
```

仮引数仕様

なし

返り値

real(kind=c_double) 型スカラー値。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を使って、[0.0,1.0] の範囲で一様な浮動小数点数乱数を生成する。

8.8.12 mt_genrand_real2

Fortran 構文

```
function fdps_ctrl%mt_genrand_real2()
```

C 言語 構文

```
double fdps_mt_genrand_real2();
```

仮引数仕様

なし

返り値

real(kind=c_double) 型スカラー値。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を使って、[0.0,1.0) の範囲で一様な浮動小数点数乱数を生成する。

8.8.13 mt_genrand_real3

Fortran 構文

```
function fdps_ctrl%MT_genrand_real3()
```

C 言語 構文

```
double fdps_mt_genrand_real3();
```

仮引数仕様

なし。

返り値

real(kind=c_double) 型スカラー値。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を使って、(0.0,1.0) の範囲で一様な浮動小数点乱数を生成する。

8.8.14 mt_genrand_res53

Fortran 構文

```
function fdps_ctrl%MT_genrand_res53()
```

C 言語 構文

```
double fdps_mt_genrand_res53();
```

仮引数仕様

なし。

返り値

real(kind=c_double) 型スカラー値。

機能

擬似乱数列生成器メルセンヌ・ツイスタ (Mersenne twister) を使って、[0.0,1.0) の範囲で一様な浮動小数点乱数を生成する。前述した mt_genrand_real$x$ (x=1-3) は浮動小数点数へ変換するのに 32 ビット整数乱数を使用しているのに対し、本 API では 53 ビット整数乱数を使用している。

第9章　エラーメッセージ

本章では、FDPS Fortran インターフェースを用いたプログラムを実行した際に出力されるエラーメッセージ (エラー検出) について記述する。Fortran インターフェースは FDPS 本体を使用しているため、まず FDPS 本体が検出するエラーについて記述する。その後、Fortran インターフェースに固有のエラー検出について記述する。

9.1　FDPS 本体

ここでは、FDPS 本体に関するエラーメッセージを記述するが、以下の点に関しては注意して頂きたい:

- 簡単のため、FDPS 本体を FDPS と略して記述する。
- FDPS 本体で定義された C++のデータ型、関数、API 名を使用する。
- Fortran インターフェースを使用する限り発生しないエラーに関しても記述されている。

9.1.1　概要

FDPS ではのコンパイル時もしくは実行時のエラー検出機能を備えている。ここでは、FDPS で検出可能なエラーとその場合の対処について記述する。ただし、ここに記述されていないエラーも起こる可能性がある。(その場合は開発者に報告していただけると助かります。)

9.1.2　コンパイル時のエラー

9.1.3　実行時のエラー

FDPS が実行時エラーを検出すると標準エラー出力に以下のような書式でメッセージを出力し、PS::Abort(-1) によってプログラムを終了する。

PS_ERROR: *ERROR MESSAGE*
function: *FUNCTION NAME*, line: *LINE NUMBER*, file: *FILE NAME*

- *ERROR MESSAGE*

  エラーメッセージ

- *FUNCTION NAME*
  エラーが起こった関数の名前
- *LINE NUMBER*
  エラーが起こった行番号
- *FILE NAME*
  エラーが起こったファイルの名前

以下、FDPS で用意されている実行時エラーメッセージを列挙していく。

9.1.3.1 PS_ERROR: can not open input file

ユーザーが FDPS のファイル入力関数を使っており、ユーザーが指定した入力ファイルがなかった場合に表示される。
エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

```
input file: “入力ファイル名”
```

9.1.3.2 PS_ERROR: can not open output file

ユーザーが FDPS のファイル出力関数を使っており、ユーザーが指定した出力ファイルがなかった場合に表示される。
エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

```
output file: “出力ファイル名”
```

9.1.3.3 PS_ERROR: Do not initialize the tree twice

同一のツリーオブジェクトに対して関数 PS::TreeForForce::initialize(...) を 2 度呼び出した場合に表示される。同一のツリーオブジェクトに対して PS::TreeForForce::initialize(...) の呼び出しを一回にする。

9.1.3.4 PS_ERROR: The opening criterion of the tree must be >= 0.0

長距離力モードでツリーのオープニングクライテリオンに負の値が入力された場合に表示される。関数 PS::TreeForForce::initialize(...) を使ってオープニングクライテリオンに 0 以上の値を指定する必要がある。
エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

theta_= “入力されたオープニングクライテリオンの値”
SEARCH_MODE: “対象となるツリーのサーチモードの型名”
Force: “対象となるツリーのフォースの型名”
EPI: “対象となるツリーの EPI の型名”
EPJ: “対象となるツリーの EPJ の型名”
SPJ: “対象となるツリーの SPJ の型名”

9.1.3.5 PS_ERROR: The limit number of the particles in the leaf cell must be > 0

長距離力モードでツリーのリーフセルの最大粒子数に負の値が入力された場合に表示される。関数 PS::TreeForForce::initialize(...) を使ってリーフセルの最大粒子数に正の整数を指定する必要がある。

エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

n_leaf_limit_=” 入力されたリーフセルの最大粒子数”
SEARCH_MODE: “対象となるツリーのサーチモードの型名”
Force: “対象となるツリーのフォースの型名”
EPI: “対象となるツリーの EPI の型名”
EPJ: “対象となるツリーの EPJ の型名”
SPJ: “対象となるツリーの SPJ の型名”

9.1.3.6 PS_ERROR: The limit number of particles in ip groups msut be >= that in leaf cells

長距離力モードでツリーのリーフセルの最大粒子数が i 粒子グループの粒子の最大数より大きかった場合に表示される。関数 PS::TreeForForce::initialize(...) を使って i 粒子グループの最大粒子数をリーフセルの最大粒子数以上にする必要がある。

エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

n_leaf_limit_=” 入力されたリーフセルの最大粒子数”
n_grp_limit_=” 入力された i 粒子グループの内の最大粒子数”
SEARCH_MODE: “対象となるツリーのサーチモードの型名”
Force: “対象となるツリーのフォースの型名”
EPI: “対象となるツリーの EPI の型名”
EPJ: “対象となるツリーの EPJ の型名”
SPJ: “対象となるツリーの SPJ の型名”

9.1.3.7 PS_ERROR: The number of particles of this process is beyond the FDPS limit number

FDPS では 1 プロセスあたりに扱える粒子数は 2G(G=2^{30}) であり、それ以上の粒子を確保しようとした場合に表示される。この場合、プロセス数を増やすなどして、1 プロセスあたりの粒子数を減らす必要がある。

9.1.3.8 PS_ERROR: The forces w/o cutoff can be evaluated only under the open boundary condition

開放境界以外の条件下でカットオフなし長距離力を設定した場合に表示される。カットオフなし長距離力の計算では必ず、開放境界条件を使う。無限遠までの粒子からの力を計算したい場合はカットオフあり長距離力の計算を FDPS で行い、カットオフ外からの力の計算は外部モジュールである Particle Mesh を使う事ができる。

9.1.3.9 PS_ERROR: A particle is out of root domain

ユーザーが *PS::DomainInfo::setRootDomain(...)* 関数を用いてルートドメインを設定しており、粒子がそのルートドメインからはみ出していた場合に表示される。周期境界条件の場合はユーザーは粒子をルートドメイン内に収まるように位置座標をシフトする必要がある。FDPS では粒子をルートドメイン内にシフトする関数
PS::ParticleSystem::adjustPositionIntoRootDomain(...) を用意しており、それを使うこともできる。

エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

position of the particle=” 粒子の座標”
position of the root domain=” ルートドメインの座標”

9.1.3.10 PS_ERROR: The smoothing factor of an exponential moving average is must between 0 and 1.

ユーザーが PS::DomainInfo::initialize(...) 関数を用いて平滑化係数に 0 未満もしくは 1 を超える値を設定した場合に表示される。エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

The smoothing factor of an exponential moving average=” 平滑化係数の値”

9.1.3.11 PS_ERROR: The coodinate of the root domain is inconsistent.

ユーザーが PS::DomainInfo::setPosRootDomain(...) 関数を用いてルートドメインを設定した時に、ユーザーが設定した小さい側の頂点の座標の任意の成分が大きい側の頂点の対応する座標の値よりも大きかった場合に表示される。エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

```
The coordinate of the low vertex of the rood domain=" 小さい側の頂点の座標"
The coordinate of the high vertex of the rood domain=" 大きい側の頂点の座標"
```

9.1.3.12 PS_ERROR: Vector invalid accesse

`Vector` 型の `[]` 演算子で定義されている範囲外の成分にアクセスを行った場合に表示される。エラーメッセージのあとに以下のメッセージも標準エラー出力に表示される。

```
Vector element=" 指定した成分" is not valid
```

9.2 FDPS Fortran/C 言語 インターフェース

本節では、FDPS Fortran/C 言語 インターフェース固有のエラーメッセージについて記述する。

9.2.1 コンパイル時のエラー検出

FDPS Fortran/C 言語 インターフェースにコンパイルエラーを検出する機能はない。

9.2.2 実行時のエラー検出

FDPS Fortran/C 言語 インターフェースが実行時エラーを検出すると、標準出力に以下のような書式でメッセージを出力し、`PS_abort(-1)` によってプログラムを終了する。

```
*** PS_FTN_IF_ERROR ***
message:  error_message
function: function_name
file:     file_name
```

ここで、

パラメータ名	定義
error_message	エラーメッセージ
function_name	エラーを検出したサブルーチン、或いは、関数の名前
file_name	上記のサブルーチン、或いは、関数が定義されているファイルの名前

である。

以下、本 Fortran/C 言語 インターフェースで用意されている実行時エラーメッセージを列挙していく。

9.2.2.1 FullParticle '派生データ型名' does not exist

これは、粒子群オブジェクトを生成する API `create_psys` に、FullParticle 型ではない派生データ型名が指定された場合に表示される。

9.2.2.2 An invalid ParticleSystem number is received

これは、不正な粒子群オブジェクト識別番号が指定された場合に表示される。

9.2.2.3 cannot create Tree 'ツリーの種類'

これは、ツリーオブジェクトを生成する API `create_tree` に、不正なツリーの種類が指定された場合に表示される。このエラーは、例えば、探索半径を持たない EssentialParticleJ 型で短距離力用ツリーを生成しようとしたとき等に起こる。

9.2.2.4 An invalid Tree number is received

これは、不正なツリーオブジェクト識別番号が指定された場合に表示される。

9.2.2.5 The combination psys_num and tree_num is invalid

これは、相互作用計算を行う次の API `calc_force_all_and_write_back`,`calc_force_all`, `calc_force_and_write_back` において、次の条件が満たされた場合に表示される:

- 粒子群オブジェクトとツリーオブジェクトの識別番号の組み合わせが不適切な場合
- 識別番号で指定された粒子群オブジェクトとツリーオブジェクトが存在しない場合

9.2.2.6 tree_num passed is invalid

これは API に不正なツリーオブジェクトの識別番号が渡された場合に表示される。

9.2.2.7 EssentialParticleJ specified does not have a member variable representing the search radius or Tree specified does not support neighbor search

これは近傍粒子リストを取得する API `get_neighbor_list` において、次の条件が満たされた場合に表示される:

- 識別番号で指定されたツリーオブジェクトを生成する際に、探索半径を持たない EssentialParticleJ 型が指定されている場合
- 識別番号で指定されたツリーオブジェクトが近傍粒子探索をサポートしないタイプのツリーの場合

エラーメッセージの後に、以下の情報も標準出力に表示される：

```
Please check the definitions of EssentialParticleJ
and tree object:
 - EssentialParticleJ: EPJ_name
 - TreeInfo: tree_info
```

ここで、

パラメータ名	定義
EPJ_name	ツリーオブジェクト生成時に指定した EssentialParticleJ 型として指定した派生データ型名
tree_info	ツリーオブジェクト生成時に指定したツリーの種類を示す文字列 (第 8 章 8.4 節参照)

である。

9.2.2.8 Unknown boundary condition is specified

これは境界条件を指定する API `set_boundary_condition` に、不正な列挙型が渡された場合に表示される。

第10章　限界と制約

本章では、FDPS および FDPS Fortran/C 言語 インターフェースの限界と制約について記述する。FDPS Fortran/C 言語 インターフェースは FDPS 本体の仕様による制限を無条件に受けるため、まずはじめに、FDPS 本体の限界について記述する。次に、FDPS Fortran/C 言語 インターフェース固有の限界およびユーザが受ける制約について記述する。

10.1　FDPS 本体

- FDPS 独自の整数型を用いる場合、GCC コンパイラと K コンパイラでのみ正常に動作することが保証されている。

10.2　FDPS Fortran/C 言語 インターフェース

現時点で、本 Fortran/C 言語 インターフェースには次の制約・限界がある。

- FDPS 本体の一部の低レベル API、および、入出力用 API はサポートしていない。
- GPU (Graphics Processing Unit) 上での実行はまだサポートしていない。
- ユーザが C++言語で記述されたユーザコードから FDPS 本体を直接使用する場合、ユーザは超粒子が持つべきモーメント情報を自由にカスタマイズすることが可能である。ここで、モーメント情報とは、粒子-超粒子間相互作用を計算する上で必要となる量で、超粒子を構成する粒子の持つ物理量から計算されるものである。例としては、単極子や双極子、高次の多重極子等がある。本 Fortran/C 言語 インターフェースでは、FDPS 本体が予めで用意しているモーメント情報のみをサポートする (第 4 章 4.4 節および第 8 章 8.4 節参照)。

第11章　変更履歴

本章では、本仕様書の変更履歴を記述する。

- 2016/12/26
  - Fortran インターフェース 初リリース (FDPS 3.0 として)
- 2017/08/23
  - FDPS に予め用意された超粒子型のデフォルトの精度を 64 ビットに変更 (FDPS 3.0a)。
- 2017/11/01
  - 粒子群オブジェクト用 API に粒子の並び替えを行う API `sort_particle` を追加。
  - ツリーオブジェクト用 API `calc_force_all_and_write_back` と `calc_force_all` に相互作用リストを再利用する機能を追加。
  - ツリーオブジェクト用 API に粒子 ID からそれに対応する EssentialParticleJ を取得する API `get_epj_from_id` を追加。
- 2017/11/08
  - FDPS 4.0 リリース
- 2017/11/17
  - API `broadcast` の不具合を修正 (FDPS 4.0a)
- 2018/8/1
  - Fortran インターフェス生成スクリプト `get_ftn_if.py` の以下の不具合を修正 (FDPS 4.1b)
    - 従来のスクリプトでは、`copyFromForce` 指示文の処理を正しく行っていなかった。具体的には、`$!fdps copyFromForce (src_mbr, dst_mbr) ...` と処理すべきところを、`$!fdps copyFromForce (dst_mbr, src_mbr)...` として処理していた。このバグのため、スクリプトがエラーで停止する場合があった。
    - 従来のスクリプトでは、内部の処理で、与えられたユーザ定義型からは生成できないはずの tree クラスを生成する場合があった。この場合、コンパイルエラーが発生する問題があった。
- 2018/8/2

  - API get_boundary_condition を追加
  - API collect_sample_particle の引数 weight のデフォルト値を 1 からローカル粒子数に変更。
  - API decompose_domain_all の引数 weight のデフォルト値が 1 と記述されていたが、実際にはローカル粒子数だったため、記述を修正。
- 2018/8/31
  - API barrier, set_particle_local_tree, get_force を追加
  - スレッドセーフな疑似乱数生成用 API を追加 (API に “mtts” がつくもの)
- 2018/11/8
  - C 言語インターフェースの記述を追加 (FDPS 5.0 としてリリース)
- 2018/12/7
  - 長距離力用ツリーの種別に MonopoleWithSymmetrySearch 型 及び QuadrupoleWithSymmetrySearch 型を追加 (FDPS 5.0a としてリリース)
- 2019/1/25
  - FDPS v5.0a で gen_ftn_if.py に入ったバグを修正 (FDPS 5.0c としてリリース)
- 2019/3/1
  - FDPS 5.0d リリース
    * EssentialParticleI 型が探索半径を保持している場合に、 gen_ftn_if.py が停止してしまうバグを修正。
    * 今回のリリースから FDPS の C++コア部分の実装で C++11 の機能を使用している。したがって、使用している C++コンパイラに適切なオプション (gcc の場合、-std=c++11) をつける必要がある。
- 2019/3/7
  - Long-MonopoleWithCutoff 型に関する記述を改善
- 2019/7/11
  - API remove_particle の仕様を明確化し、この API の実装を仕様に沿ったものに修正
- 2019/9/06
  - FDPS 5.0f リリース
    * コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION を定義した場合、コンパイルエラーになる問題を修正
    * C 言語から FDPS を使う場合、ユーザ定義型のメンバ変数名が 1 文字だと gen_c_if.py が正しく動作しない問題を修正

    * シンボリックリンク doc/doc_specs_c_ja.pdf 及び doc/doc_specs_c_en.pdf を追加

- 2019/9/10
  - FDPS 5.0g リリース
    * コンパイル時にマクロ PARTICLE_SIMULATOR_TWO_DIMENSION を定義した場合、実行時エラーになる問題を修正
- 2020/8/16
  - FDPS 6.0 リリース
    * PIKG を導入
- 2020.8.18
  - FDPS 6.0a リリース
    * 付属の PIKG のバージョンを v0.1b に更新
- 2020.8.19
  - FDPS 6.0b リリース
    * N 体シミュレーションサンプルコード (sample/*/nbody) の実装を PIKG で生成したカーネルを使った場合に性能が出るように改善
- 2020.8.28
  - FDPS 6.0b1 リリース
    * サンプルコードで使用する初期条件配布先が変更になったため、チュートリアルを修正
- 2020.9.02
  - FDPS 6.0b2 リリース
    * サンプルコードのツリーオブジェクトを初期化する関数の第一引数を修正
    * 対応するチュートリアルの記述も修正