

FDPS Fortran 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, and 牧野淳一郎

理化学研究所 計算科学研究機構 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	9
1.1	文書の構成	9
1.2	ライセンス	10
1.3	ユーザーサポート	11
1.3.1	コンパイルできない場合	11
1.3.2	コードがうまく動かない場合	11
1.3.3	その他	11
第 2 章	FDPS 概要	13
2.1	開発目的	13
2.2	基本的な考えかた	13
2.2.1	大規模並列粒子シミュレーションの手順	13
2.2.2	ユーザーと FDPS の役割分担	14
2.2.3	ユーザーのやること	15
2.2.4	補足	15
2.3	コードの動作	15
2.3.1	FDPS 本体	15
2.3.2	Fortran インターフェース	16
第 3 章	Fortran インターフェースのファイル構成と概要	19
3.1	ファイル構成と概要	19
3.1.1	FDPS 本体	19
3.1.2	Fortran インターフェース	19
3.1.3	Fortran インターフェースを使ったコード開発の流れ	22
3.1.4	インターフェースプログラム生成の必要性	23
3.2	ドキュメント	24
3.3	サンプルコード	24
第 4 章	FDPS で提供される派生データ型	25
4.1	ベクトル型	25
4.2	対称行列型	26
4.3	超粒子型	28
4.4	時間プロファイル型	31
4.5	列挙型	32

4.5.1	境界条件型	32
4.5.2	相互作用リストモード型	33
第 5 章	ユーザー定義型・ユーザー定義関数	35
5.1	ユーザー定義型	35
5.1.1	共通規則	35
5.1.1.1	Fortran 文法に関する要請	35
5.1.1.2	FDPS 指示文 (共通項目のみ)	36
5.1.1.2.1	ユーザー定義型の種別を指定する FDPS 指示文	37
5.1.1.2.2	必須物理量を指定する FDPS 指示文	37
5.1.1.2.3	FDPS 指示文の記述例	38
5.1.2	FullParticle 型	39
5.1.2.1	常に必要な FDPS 指示文とその記述法	39
5.1.2.2	場合によっては必要な FDPS 指示文とその記述法	40
5.1.3	EssentialParticleI 型	41
5.1.3.1	常に必要な FDPS 指示文とその記述法	41
5.1.3.2	場合によっては必要な FDPS 指示文とその記述法	41
5.1.4	EssentialParticleJ 型	42
5.1.4.1	常に必要な FDPS 指示文とその記述法	42
5.1.4.2	場合によっては必要な FDPS 指示文とその記述法	42
5.1.5	Force 型	43
5.1.5.1	常に必要な FDPS 指示文とその記述法	43
5.1.5.2	場合によっては必要な FDPS 指示文とその記述法	45
5.2	ユーザー定義関数	45
5.2.1	共通規則	45
5.2.1.1	Fortran 文法に関する要請	45
5.2.1.2	FDPS 本体からの要請	46
5.2.2	関数 calcForceEpEp	47
5.2.3	関数 calcForceEpSp	48
第 6 章	Fortran インターフェースの生成	51
6.1	スクリプトの動作条件	51
6.2	スクリプトの使用方法	51
第 7 章	Fortran インターフェースのコンパイル	53
7.1	コンパイル	53
7.1.1	コンパイルの基本手順	53
7.1.2	GCC を用いたコンパイルの仕方	55
7.1.2.1	MPI を使用しない場合	55
7.1.2.2	MPI を使用する場合	55
7.2	コンパイル時マクロ定義	56

7.2.1	座標系の指定	56
7.2.1.1	直角座標系 3 次元	56
7.2.1.2	直角座標系 2 次元	56
7.2.2	並列処理の指定	56
7.2.2.1	OpenMP の使用	56
7.2.2.2	MPI の使用	56
7.2.3	データ型の精度の指定	57
7.2.3.1	超粒子型のメンバ変数の型の精度の指定	57
7.2.4	拡張機能 Particle Mesh の使用	57
7.2.5	デバッグ用出力の指定	57
第 8 章	API 仕様一覧	59
8.1	開始および終了処理に関わる API	59
8.1.1	PS_initialize	59
8.1.2	PS_finalize	60
8.1.3	PS_abort	61
8.2	粒子群オブジェクト用 API	62
8.2.1	create_psys	63
8.2.2	delete_psys	64
8.2.3	init_psys	65
8.2.4	get_psys_info	66
8.2.5	get_psys_memsize	67
8.2.6	get_psys_time_prof	68
8.2.7	clear_psys_time_prof	69
8.2.8	set_nptcl_smpl	70
8.2.9	set_nptcl_loc	71
8.2.10	get_nptcl_loc	72
8.2.11	get_nptcl_glb	73
8.2.12	get_psys_fptr	74
8.2.13	exchange_particle	75
8.2.14	add_particle	76
8.2.15	remove_particle	77
8.2.16	adjust_pos_into_root_domain	78
8.2.17	sort_particle	79
8.3	領域情報オブジェクト用 API	80
8.3.1	create_dinfo	81
8.3.2	delete_dinfo	82
8.3.3	init_dinfo	83
8.3.4	get_dinfo_time_prof	84
8.3.5	clear_dinfo_time_prof	85
8.3.6	set_nums_domain	86

8.3.7	<code>set_boundary_condition</code>	87
8.3.8	<code>set_pos_root_domain</code>	88
8.3.9	<code>collect_sample_particle</code>	89
8.3.10	<code>decompose_domain</code>	90
8.3.11	<code>decompose_domain_all</code>	91
8.4	ツリーオブジェクト用 API	92
8.4.1	ツリーの種別	93
8.4.1.1	長距離力用ツリーの種別	93
8.4.1.2	短距離力用ツリーの種別	93
8.4.2	<code>create_tree</code>	94
8.4.3	<code>delete_tree</code>	95
8.4.4	<code>init_tree</code>	96
8.4.5	<code>get_tree_info</code>	97
8.4.6	<code>get_tree_memsizes</code>	98
8.4.7	<code>get_tree_time_prof</code>	99
8.4.8	<code>clear_tree_time_prof</code>	100
8.4.9	<code>get_num_interact_ep_ep_loc</code>	101
8.4.10	<code>get_num_interact_ep_sp_loc</code>	102
8.4.11	<code>get_num_interact_ep_ep_glb</code>	103
8.4.12	<code>get_num_interact_ep_sp_glb</code>	104
8.4.13	<code>clear_num_interact</code>	105
8.4.14	<code>get_num_tree_walk_loc</code>	106
8.4.15	<code>get_num_tree_walk_glb</code>	107
8.4.16	<code>calc_force_all_and_write_back</code>	108
8.4.17	<code>calc_force_all</code>	111
8.4.18	<code>calc_force_making_tree</code>	113
8.4.19	<code>calc_force_and_write_back</code>	115
8.4.20	<code>get_neighbor_list</code>	117
8.4.21	<code>get_epj_from_id</code>	118
8.5	通信用 API	119
8.5.1	<code>get_rank</code>	120
8.5.2	<code>get_rank_multi_dim</code>	121
8.5.3	<code>get_num_procs</code>	122
8.5.4	<code>get_num_procs_multi_dim</code>	123
8.5.5	<code>get_logical_and</code>	124
8.5.6	<code>get_logical_or</code>	125
8.5.7	<code>get_min_value</code>	126
8.5.8	<code>get_max_value</code>	128
8.5.9	<code>get_sum</code>	130
8.5.10	<code>broadcast</code>	131

8.5.11	get_wtime	132
8.6	Particle Mesh 用 API	133
8.6.1	create_pm	134
8.6.2	delete_pm	135
8.6.3	get_pm_mesh_num	136
8.6.4	get_pm_cutoff_radius	137
8.6.5	set_dinfo_of_pm	138
8.6.6	set_psys_of_pm	139
8.6.7	get_pm_force	140
8.6.8	get_pm_potential	141
8.6.9	calc_pm_force_only	142
8.6.10	calc_pm_force_all_and_write_back	143
8.7	その他の API	144
8.7.1	MT_init_genrand	145
8.7.2	MT_genrand_int31	146
8.7.3	MT_genrand_real1	147
8.7.4	MT_genrand_real2	148
8.7.5	MT_genrand_real3	149
8.7.6	MT_genrand_res53	150
第 9 章	エラーメッセージ	151
9.1	FDPS 本体	151
9.1.1	概要	151
9.1.2	コンパイル時のエラー	151
9.1.3	実行時のエラー	151
9.1.3.1	PS_ERROR: can not open input file	152
9.1.3.2	PS_ERROR: can not open output file	152
9.1.3.3	PS_ERROR: Do not initialize the tree twice	152
9.1.3.4	PS_ERROR: The opening criterion of the tree must be ≥ 0.0	152
9.1.3.5	PS_ERROR: The limit number of the particles in the leaf cell must be > 0	153
9.1.3.6	PS_ERROR: The limit number of particles in ip groups must be \geq that in leaf cells	153
9.1.3.7	PS_ERROR: The number of particles of this process is beyond the FDPS limit number	154
9.1.3.8	PS_ERROR: The forces w/o cutoff can be evaluated only under the open boundary condition	154
9.1.3.9	PS_ERROR: A particle is out of root domain	154
9.1.3.10	PS_ERROR: The smoothing factor of an exponential moving average is must between 0 and 1.	154
9.1.3.11	PS_ERROR: The coodinate of the root domain is inconsistent.	155

9.1.3.12	PS_ERROR: Vector invalid accesse	155
9.2	FDPS Fortran インターフェース	155
9.2.1	コンパイル時のエラー検出	155
9.2.2	実行時のエラー検出	155
9.2.2.1	FullParticle ‘派生データ型名‘ does not exist	156
9.2.2.2	An invalid ParticleSystem number is received	156
9.2.2.3	cannot create Tree ‘ツリーの種類‘	156
9.2.2.4	An invalid Tree number is received	156
9.2.2.5	The combination psys_num and tree_num is invalid	156
9.2.2.6	tree_num passed is invalid	156
9.2.2.7	EssentialParticleJ specified does not have a member variable representing the search radius or Tree specified does not sup- port neighbor search	157
9.2.2.8	Unknown boundary condition is specified	157
第 10 章	限界と制約	159
10.1	FDPS 本体	159
10.2	FDPS Fortran インターフェース	159
第 11 章	変更履歴	161

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) Fortran インターフェースの仕様書である。この文書は理化学研究所計算科学研究機構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 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 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 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 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派生データ型や関数について記述されている。第8章には、Fortran 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) 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, Fukushima & 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 Storage and Analysis, No. 5) のモジュールを使用している。GreeM コードは Yoshikawa & Fukushima (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。

Copyright (c) <2015-> <FDPS developer team>

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the “Software”), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

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](mailto:fdps-support@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減することである。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。

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

ここで \mathbf{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。 \mathbf{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 \mathbf{u}_i を持つように、他のプロセスと物理量ベクトル \mathbf{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 \mathbf{u}_j と超粒子の物理量ベクトル \mathbf{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、 $d\mathbf{u}_i/dt$ 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 \mathbf{u}_i とその時間導関数 $d\mathbf{u}_i/dt$ 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 \mathbf{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 \mathbf{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 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 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 インターフェースにも用意されている。したがって、Fortran 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。

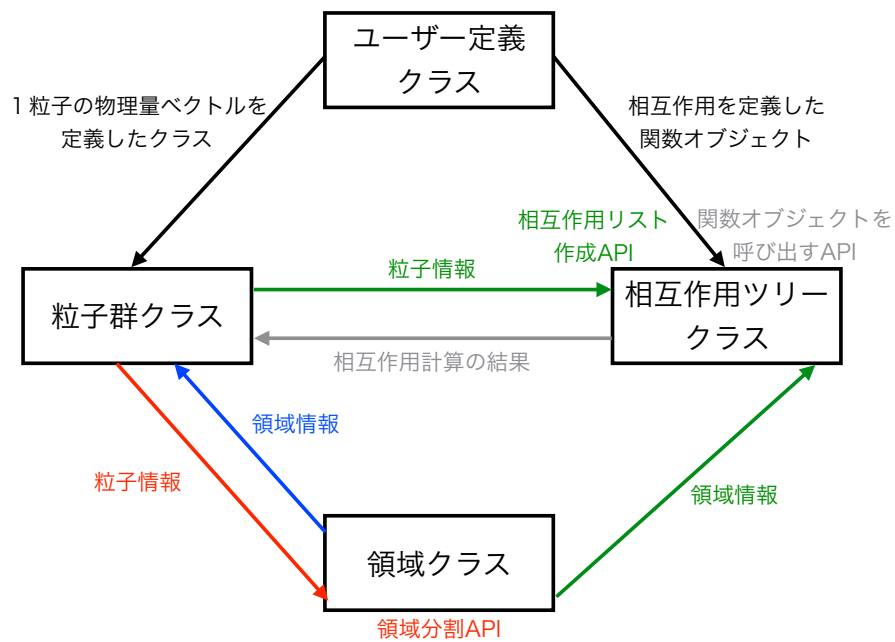


図 2.1: モジュールインターフェースと情報の流れの模式図。

第3章 Fortran 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 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、ユーザ自身が定義 (実装) する派生データ型 (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次節 3.1.4 で解説する。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、インターフェース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。

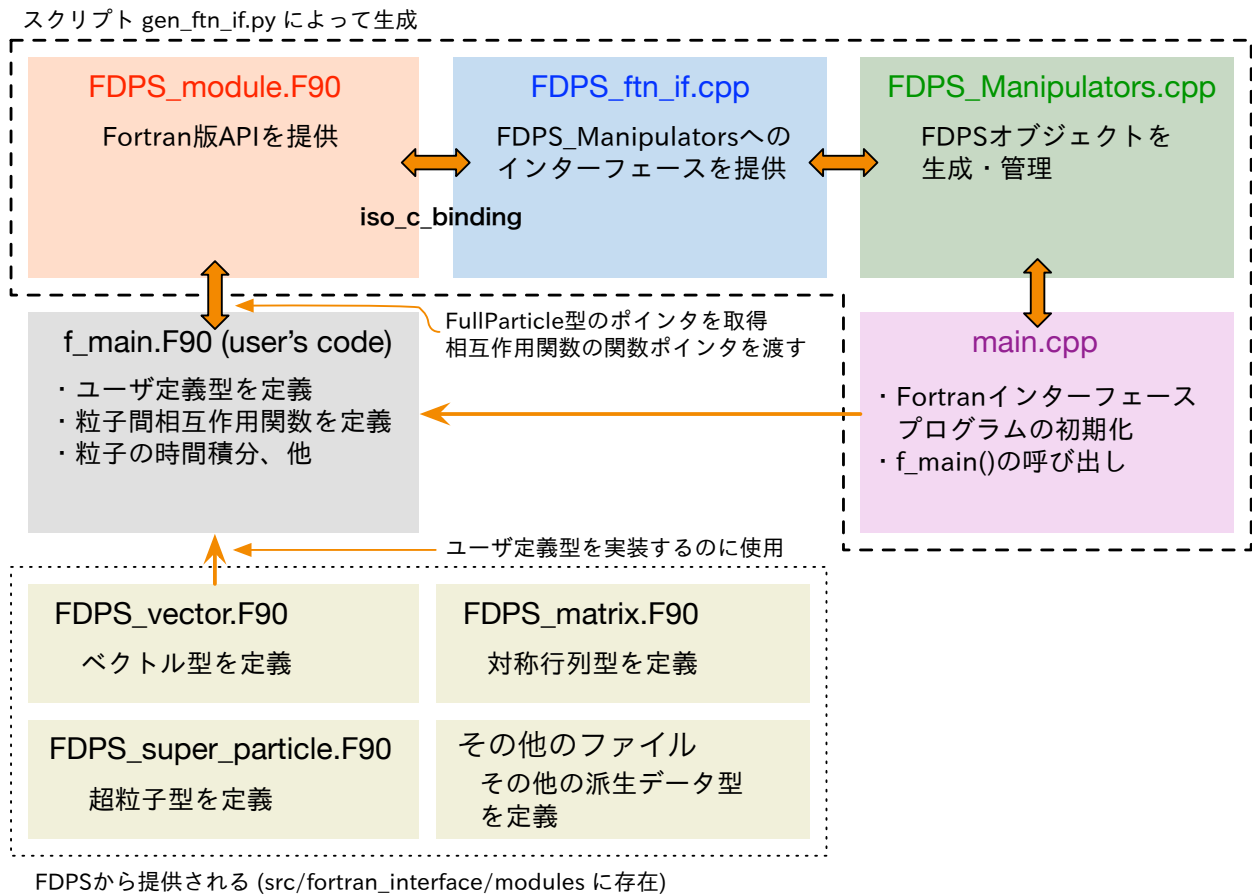


図 3.1: Fortran インターフェースとユーザコードの関係。

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 Fortran 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 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になる：

[1] ユーザ定義型の実装

前節で述べた通り、FDPS の Fortran 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 の派生データ型として実装する。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` 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 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 のサブルーチンとして実装する。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()` 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。

Fortran インターフェースを用いたコードの例に関しては、`doc/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)。

[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は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。

3.1.4 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 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けばよい)。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 の関数で使わ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 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 では不可能である。これが、Fortran 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 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 プログラムの間でデータをやり取りするためには、Fortran 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 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` であり、仕様書 (本文書) が `doc_specs_ftn_ja.pdf` である。

3.3 サンプルコード

サンプルコードはディレクトリ `sample/fortran` の下にある。サンプルコードは 3 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`)、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`)、 P^3M (Particle-Particle-Particle-Mesh) 計算用コード (`sample/fortran/p3m`) となっている。

第4章 FDPSで提供される派生データ型

FDPS Fortran インターフェースでは独自の派生データ型が定義されている。派生データ型には、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。これらの派生データ型は第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 ベクトル型

ベクトル型は `fdps_f32vec` と `fdps_f64vec` の2種類があり、`src/fortran_interface/modules/FDPS_vector.F90` 内において、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 4.1: ベクトル型

```

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

```

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

4.2 対称行列型

対称行列型には、`fdps_f32mat` と `fdps_f64mat` の 2 種類があり、`src/fortran_interface/modules/FDPS_matrix.F90` 内において、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 4.2: 対称行列型

```

1 module fdps_matrix
2   use, intrinsic :: iso_c_binding
3   implicit none
4
5   !**** PS::F32mat
6   type, public, bind(c) :: fdps_f32mat
7 #ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
8     real(kind=c_float) :: xx,yy,zz,xy,xz,yz
9 #else
10    real(kind=c_float) :: xx,yy,xy
11 #endif
12 end type fdps_f32mat
13
14 !**** PS::F64mat
15 type, public, bind(c) :: fdps_f64mat
16 #ifndef PARTICLE_SIMULATOR_TWO_DIMENSION
17     real(kind=c_double) :: xx,yy,zz,xy,xz,yz
18 #else
19     real(kind=c_double) :: xx,yy,xy
20 #endif
21 end type fdps_f64mat
22
23 end module fdps_matrix

```

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

4.3 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派生データ型である。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_cutoff`があり、`src/fortran_interface/modules/FDPS_super_particle.F90`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 4.3: 超粒子型

```

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::SPJMonopoleCutoff
58 type, public, bind(c) :: fdps_spj_monopole_cutoff
59     real(kind=c_double) :: mass
60     type(fdps_f64vec) :: pos
61 end type fdps_spj_monopole_cutoff
62
63 ! [TODO]
64 !     PS::SPJMonopolePeriodic
65 !     PS::SPJMonopoleCutoffScatter
66
67 end module fdps_super_particle

```

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-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。

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

4.4 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、`src/fortran_interface/modules/FDPS_time_profile.F90` 内において、以下のように定義される。この派生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 4.4: 時間プロファイル型

```

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 +
33      exchange_LET_2nd
34
35    real(kind=c_double) :: calc_force__core__walk_tree
36
37    real(kind=c_double) :: calc_force__make_ipgroup
38    real(kind=c_double) :: calc_force__core
39    real(kind=c_double) :: calc_force__copy_original_order
40
41    real(kind=c_double) :: exchange_particle__find_particle
42    real(kind=c_double) :: exchange_particle__exchange_particle
43
44    real(kind=c_double) :: decompose_domain__sort_particle_1st
45    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

```

4.5 列挙型

本節では、FDPS Fortran 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。

4.5.1 境界条件型

境界条件型は、境界条件を指定する API `set_boundary_condition` で使用され (第 8 章 8.3 節「領域情報オブジェクト用 API」参照)、FDPS_module.F90 において、以下のように定義されている。

Listing 4.5: 境界条件型

```

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

```

表 4.4 に、各列挙子に対応する境界条件を示す。

列挙子	境界条件
<code>fdps_bc_open</code>	開放境界となる。デフォルトではこの境界条件となる。
<code>fdps_bc_periodic_x</code>	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。
<code>fdps_bc_periodic_y</code>	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。
<code>fdps_bc_periodic_z</code>	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。
<code>fdps_bc_periodic_xy</code>	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。
<code>fdps_bc_periodic_xz</code>	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。
<code>fdps_bc_periodic_yz</code>	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。
<code>fdps_bc_periodic_xyz</code>	全方向周期境界条件。
<code>fdps_bc_shearing_box</code>	シアリングボックス境界条件 (現時点では未実装)。
<code>fdps_bc_user_defined</code>	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

4.5.2 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、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、`FDPS_module.F90` において、以下のように定義されている。

Listing 4.6: 相互作用リストモード型

```

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

```

表 4.5 に、各列挙子に対応する動作モードを示す。

列挙子	機能
<code>fdps_make_list</code>	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。
<code>fdps_make_list_for_reuse</code>	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 <code>fdps_make_list_for_reuse</code> 、もしくは、 <code>fdps_make_list</code> を用いて相互作用計算を行った際に破棄される。
<code>fdps_reuse_list</code>	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 <code>fdps_make_list_for_reuse</code> 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

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(ユーザー定義型)と相互作用関数(ユーザー定義関数)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、ユーザー定義型やユーザー定義関数を定義する際の規定である。FDPS は、ユーザー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、ユーザーは FDPS に必須物理量がユーザー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、ユーザー定義型の間でデータのやりとりを行うが、それをユーザーが指定した方法で行うことになる。したがって、ユーザー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ー定義型について記述し、その後ユーザー定義関数について記述する。

5.1 ユーザー定義型

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る。これらの派生データ型のうち EssentialParticleI 型、EssentialParticleJ 型、Force 型は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。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な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、はじめにユーザー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、ユーザー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 5.1.1.2 で説明する FDPS 指示文によって、どのユーザ定義型 (Full-Particle 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 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。

^{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。

5.1.1.2.1 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 型のサブセットであり、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。

5.1.1.2.2 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。

5.1.1.2.3 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)。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

^{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


```

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

```

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 型と他のユーザー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 5.1.1.2 節で説明した方法で記述すればよい。最後のものは、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。

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 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-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 5.1.1.2 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、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。

(iii) の場合には、他のユーザー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、対応するユーザー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であり、相互作用関数 (ユーザー定義関数) の定義に必要なほか、相互作用ツリーオブジェクトの生成に必要な。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 5.1.1.2 節で説明した方法で記述すればよい。最後のものは、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。

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：

- Short 型に分類されるすべてのツリー

(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 5.1.1.2 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な 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であり、相互作用関数(ユーザ定義関数)の定義に必要なほか、相互作用ツリーオブジェクトの生成に必要な。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
- 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 5.1.1.2 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-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 `get_epj_from_id` を使用する場合)

(i) の場合、ユーザは派生データ型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 5.1.1.2 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 5.1.1.2 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であり、相互作用関数の定義に必要なとなるほか、相互作用ツリーオブジェクトの生成に必要なとなる。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、初期化の仕方に依拠して 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` 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、以下のよう記述する:

```
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。

(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 のサブルーチンを用いて行うことができる。この場合には、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。

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

5.2.1.1 Fortran 文法に関する要請

本節では、Fortran のサブルーチンがユーザー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` でユーザー定義関数の C 言語アドレスを取得するためには、ユーザー定義関数が C 言語と interoperable でなければならない。具体的には、ユーザー定義関数となる 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。

5.2.1.2 FDPS 本体からの要請

前節で述べた条件に加え、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 5.3 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 5.3: 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

```

5.2.2 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。

```

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

```

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)	入力	<i>i</i> 粒子の粒子数を格納した変数。
n_jp	integer(kind=c_int)	入力	<i>j</i> 粒子の粒子数を格納した変数。
ep_i	essential_particle_i 型 [†]	入力	<i>i</i> 粒子情報を持つ配列。
ep_j	essential_particle_j 型 [†]	入力	<i>j</i> 粒子情報を持つ配列。
f	force 型 [†]	入出力	<i>i</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である。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。

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。

```
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
```

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)	入力	i 粒子の粒子数を格納した変数。
n_jp	integer(kind=c_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である。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。

[‡] 第 4 章 4.3 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 インターフェースの生成

本章では、FDPS Fortran 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の動作条件について記述する。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:

- 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。

6.2 スクリプトの使用法

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:

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

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 `gen_ftn_if.py` の引数には、ユーザ定義型が記述された Fortran ファイルを指定する。複数の Fortran 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、Fortran 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`」でも確認することができる：

```
[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 インターフェースのコンパイル

ここまでの章で、FDPS の Fortran 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。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 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 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。

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によってなされる。

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。

7.1.2 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++コンパイラと Fortran コンパイラをそれぞれ `g++` と `gfortran` とする。また、MPI に対応した GCC コンパイラをそれぞれ `mpic++` と `mpif90` 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、MPI を使用しない場合と MPI を使用する場合に分けて記述する。

7.1.2.1 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
```

7.1.2.2 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++
LD_FLAGS = -lgfortran -LPATH -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。

7.2 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。

第8章 API 仕様一覧

この章では、Fortran 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、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。

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。

8.1.1 PS_initialize

```
subroutine fdps_ctrl%ps_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

```
subroutine fdps_ctrl%ps_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

```
subroutine fdps_ctrl%ps_abort(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。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 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:

```
create_psys
delete_psys
init_psys
get_psys_info
get_psys_memsize
get_psys_time_prof
clear_psys_time_prof
set_nptcl_smpl
set_nptcl_loc
get_nptcl_loc
get_nptcl_glb
get_psys_fptr
exchange_particle
add_particle
remove_particle
adjust_pos_into_root_domain
sort_particle
```

以下、順に、各 API の仕様を記述する。

8.2.1 create_psys

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

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。
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。

返り値

なし

機能

文字列 psys_info_in で指定された 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

```
subroutine fdps_ctrl%delete_psys(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

```
subroutine fdps_ctrl%init_psys(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

```
subroutine fdps_ctrl%get_psys_info(psys_num,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。

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

```
function fdps_ctrl%get_psys_memsize(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

```
subroutine fdps_ctrl%get_psys_time_prof(psys_num,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。

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 `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

```
subroutine fdps_ctrl%clear_psys_time_prof(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 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

```
subroutine fdps_ctrl%set_nptcl_smpl(psys_num,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

```
subroutine fdps_ctrl%set_nptcl_loc(psys_num,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

```
function fdps_ctrl%get_nptcl_loc(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

```
function fdps_ctrl%get_nptcl_glb(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

```
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)` ($i = 1 \sim$ 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

```
1  !* Local variables
2  type(full_particle), dimension(:), pointer :: ptcl
3  !* Get the pointer to full particle data
4  call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
5  !* Set particle data
6  do i=1,nptcl_loc
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 end do
```

8.2.13 exchange_particle

```
subroutine fdps_ctrl%exchange_particle(psys_num,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。

8.2.14 add_particle

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

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。

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` を追加する。

8.2.15 remove_particle

```
subroutine fdps_ctrl%remove_particle(psys_num,nptcl,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。

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。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。

8.2.16 adjust_pos_into_root_domain

```
subroutine fdps_ctrl%adjust_pos_into_root_domain(psys_num,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。

8.2.17 sort_particle

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

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。

Listing 8.2: 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。

8.3 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 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:

```
create_dinfo
delete_dinfo
init_dinfo
get_dinfo_time_prof
clear_dinfo_time_prof
set_nums_domain
set_boundary_condition
set_pos_root_domain
collect_sample_particle
decompose_domain
decompose_domain_all
```

以下、順に、各 API の仕様を記述する。

8.3.1 create_dinfo

```
subroutine fdps_ctrl%create_dinfo(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。

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

```
subroutine fdps_ctrl%delete_dinfo(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

```
subroutine fdps_ctrl%init_dinfo(dinfo_num,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で、デフォルトは 1.0。

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, Fukushima & Makino (2009, Publications of the Astronomical Society of Japan, 61, 1319) を参照のこと。

8.3.4 get_dinfo_time_prof

```
subroutine fdps_ctrl%get_dinfo_time_prof(dinfo_num,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。

返り値

なし

機能

領域情報オブジェクトの API である `collect_sample_particle` と `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

```
subroutine fdps_ctrl%clear_dinfo_time_prof(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 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

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

```
subroutine fdps_ctrl%set_boundary_condition(dinfo_num,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 4.5 節で説明した境界条件型である。すなわち、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。

8.3.8 set_pos_root_domain

```
subroutine fdps_ctrl%set_pos_root_domain(dinfo_num,low,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	real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec)	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。low と high は 32 ビット浮動小数点数型のサイズ space_dim の配列で与えられる。

8.3.9 collect_sample_particle

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

仮引数仕様

仮引数名	データ型	入出力属性	定義
<code>dinfo_num</code>	<code>integer(kind=c_int)</code>	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。
<code>psys_num</code>	<code>integer(kind=c_int)</code>	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。
<code>clear</code>	<code>logical(kind=c_bool)</code>	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。 <code>.true.</code> でクリアする。デフォルトは <code>.true.</code> 。
<code>weight</code>	<code>real(kind=c_float)</code>	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。デフォルトは 1。

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)。

8.3.10 decompose_domain

```
subroutine fdps_ctrl%decompose_domain(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。

8.3.11 decompose_domain_all

```
subroutine fdps_ctrl%decompose_domain_all(dinfo_num, &
                                          psys_num, &
                                          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。デフォルトは 1。

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 collect_sample_particle と decompose_domain が行うことをこの API は一度に行う。

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 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:

```
create_tree
delete_tree
init_tree
get_tree_info
get_tree_memsized
get_tree_time_prof
clear_tree_time_prof
get_num_interact_ep_ep_loc
get_num_interact_ep_sp_loc
get_num_interact_ep_ep_glb
get_num_interact_ep_sp_glb
clear_num_interact
get_num_tree_walk_loc
get_num_tree_walk_glb
calc_force_all_and_write_back
calc_force_all
calc_force_making_tree
calc_force_and_write_back
get_neighbor_list
get_epj_from_id
```

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 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類

長距離用ツリーは、モーメントの計算方法別に 6 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。P³T(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。これらを、それぞれ、MonopoleWithScatterSearch 型、QuadrupoleWithScatterSearch 型と呼ぶ。さらに、P³M(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。以上が、本 Fortran 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 4.3 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類

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

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

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。
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。

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` 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, 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

```
subroutine fdps_ctrl%delete_tree(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

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

仮引数仕様

仮引数名	データ型	入出力属性	既定値	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	0.7	見こみ角に対する基準。
n_leaf_limit	integer(kind=c_int)	入力	8	ツリーを切るのをやめる粒子数の上限。
n_group_limit	integer(kind=c_int)	入力	64	相互作用リストを共有する粒子数の上限。

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

```
subroutine fdps_ctrl%get_tree_info(tree_num,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。

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

```
function fdps_ctrl%get_tree_memsize(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

```
subroutine fdps_ctrl%get_tree_time_prof(tree_num,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。

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 `type(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

```
subroutine fdps_ctrl%clear_tree_time_prof(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 インターフェイスで用意された `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

```
function fdps_ctrl%get_num_interact_ep_ep_loc(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

```
function fdps_ctrl%get_num_interact_ep_sp_loc(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

```
function fdps_ctrl%get_num_interact_ep_ep_glb(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

```
function fdps_ctrl%get_num_interact_ep_sp_glb(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

```
subroutine fdps_ctrl%clear_num_interact(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

```
function fdps_ctrl%get_num_tree_walk_loc(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

```
function fdps_ctrl%get_num_tree_walk_glb(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 calc_force_all_and_write_back

```
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 4.5.2 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。

```
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「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。

8.4.17 calc_force_all

```
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「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。

```

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「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。

8.4.18 calc_force_making_tree

```
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。

```
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。

8.4.19 calc_force_and_write_back

```
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。

```
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。

8.4.20 get_neighbor_list

```
subroutine fdps_ctrl%get_neighbor_list(tree_num, &
                                     pos,      &
                                     r_search, &
                                     num_epj,  &
                                     fptr_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。
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	近傍粒子数。
fptr_to_EPJ	EssentialParticleJ 型, dimension(:), pointer	入出力	EssentialParticleJ 型へのポインタ。

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。

8.4.21 get_epj_from_id

```
subroutine fdps_ctrl%get_epj_from_id(tree_num, &
                                     id,          &
                                     fptr_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。
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

なし

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)。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 8.3: 例

```
1 integer(kind=c_long_long) :: id
2 type(essential_particle_j), pointer :: epj
3
4 call fdps_ctrl%get_epj_from_id(tree_num,id,epj)
5 if (associated(epj)) then
6   ! Do something using epj
7   write(*,*) 'id = ', epj%id
8 else
9   write(*,*) 'epj is NULL'
10 end if
```

8.5 通信用 API

本節では、MPI プロセス間通信に関係した API について説明を行う。通信関係の全 API の名称の一覧を以下に示す:

```
get_rank  
get_rank_multi_dim  
get_num_procs  
get_num_procs_multi_dim  
get_logical_and  
get_logical_or  
get_min_value  
get_max_value  
get_sum  
broadcast  
get_wtime
```

以下、順に各 API の仕様を記述していく。

8.5.1 get_rank

```
integer(kind=c_int) fdps_ctrl%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。

8.5.2 get_rank_multi_dim

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

8.5.3 get_num_procs

```
integer(kind=c_int) fdps_ctrl%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。

8.5.4 get_num_procs_multi_dim

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

8.5.5 get_logical_and

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

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

返り値

なし

機能

全プロセスでの f_in の論理積をとり f_out にいれる。

8.5.6 get_logical_or

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

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

返り値

なし

機能

全プロセスでの f_in の論理和をとり f_out にいれる。

8.5.7 get_min_value

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

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double)	入力	入力値
f_out	入力と同じ	入出力	出力値

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。これは以下のようなになる。

```
subroutine fdps_ctrl%get_min_value(f_in, &
                                   i_in, &
                                   f_out, &
                                   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。

8.5.8 get_max_value

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

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double)	入力	入力値
f_out	入力と同じ	入出力	出力値

返り値

なし。

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。これは以下のようなになる。

```
subroutine fdps_ctrl%get_max_value(f_in, &
                                   i_in, &
                                   f_out, &
                                   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。

8.5.9 get_sum

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

仮引数仕様

仮引数名	データ型	入出力属性	定義
f_in	integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double)	入力	入力値
f_out	入力と同じ	入出力	出力値

返り値

なし。

機能

全プロセスで f_in の総和を取り、結果を返す。

8.5.10 broadcast

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

仮引数仕様

仮引数名	データ型	入出力属性	定義
val	integer(kind=c_int) integer(kind=c_long_long) real(kind=c_float) real(kind=c_double) またはそれらの配列	入力	入力値
n	integer(kind=c_int)	入力	入力値の数
src	integer(kind=c_int)	入力	放送するプロセスランク数

返り値

なし。

機能

プロセスランク **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。

8.5.11 get_wtime

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

```
create_pm  
delete_pm  
get_pm_mesh_num  
get_pm_cutoff_radius  
set_dinfo_of_pm  
set_psys_of_pm  
get_pm_force  
get_pm_potential  
calc_pm_force_only  
calc_pm_force_all_and_write_back
```

以下、順に、各 API の仕様を記述していく。

8.6.1 create_pm

```
subroutine fdps_ctrl%create_pm(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。

返り値

なし

機能

メモリ上に、Particle Mesh 計算で使用する 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。

8.6.2 delete_pm

```
subroutine fdps_ctrl%delete_pm(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。

8.6.3 get_pm_mesh_num

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

仮引数仕様

なし

返り値

Particle Mesh 計算で使用するメッシュの 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。

8.6.4 get_pm_cutoff_radius

```
real(kind=c_double) fdps_ctrl%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フ半径を、メッシュ間隔で規格化された値として返す。

8.6.5 set_dinfo_of_pm

```
subroutine fdps_ctrl%set_dinfo_of_pm(pm_num,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。

8.6.6 set_psys_of_pm

```
subroutine fdps_ctrl%set_psys_of_pm(pm_num,psys_num,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で省略可能である。 <code>.true.</code> ならばクリアする。デフォルトは <code>.true.</code> 。

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。

8.6.7 get_pm_force

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

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	real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec)	入力	メッシュからの力の計算に使用する位置座標。
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。

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 calc_pm_force_only か 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。

8.6.8 get_pm_potential

```
subroutine fdps_ctrl%get_pm_potential(pm_num,pos,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	real(kind=c_float), dimension(space_dim) real(kind=c_double), dimension(space_dim) type(fdps_f32vec) type(fdps_f64vec)	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	real(kind=c_float)	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。

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 `calc_pm_force_only` か `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。

8.6.9 calc_pm_force_only

```
subroutine fdps_ctrl%calc_pm_force_only(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。

8.6.10 calc_pm_force_all_and_write_back

```
subroutine fdps_ctrl%calc_pm_force_all_and_write_back(pm_num,    &
                                                    psys_num, &
                                                    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。

8.7 その他の API

本節では Fortran 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:

```
MT_init_genrand  
MT_genrand_int31  
MT_genrand_real1  
MT_genrand_real2  
MT_genrand_res53
```

ここに示された API の内、名称が MT で始まる 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。

8.7.1 MT_init_genrand

```
subroutine fdps_ctrl%MT_init_genrand(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。

8.7.2 MT_genrand_int31

```
function fdps_ctrl%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。

8.7.3 MT_genrand_real1

```
function fdps_ctrl%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。

8.7.4 MT_genrand_real2

```
function fdps_ctrl%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。

8.7.5 MT_genrand_real3

```
function fdps_ctrl%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。

8.7.6 MT_genrand_res53

```
function fdps_ctrl%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_realx` ($x=1-3$) 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 インターフェース

本節では、FDPS Fortran 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 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 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
<i>error_message</i>	エラーメッセージ
<i>function_name</i>	エラーを検出したサブルーチン、或いは、関数の名前
<i>file_name</i>	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 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
<i>EPJ_name</i>	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
<i>tree_info</i>	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 インターフェースの限界と制約について記述する。FDPS Fortran 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 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 インターフェース

現時点で、本 Fortran 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 インターフェースでは、FDPS 本体がデフォルト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 4.3 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)