FDPS C言語インタフェース ユーザチュートリアル

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

理化学研究所 計算科学研究センター 粒子系シミュレータ研究チーム

0 目 次

1 変更記録

- 2018/11/7
  - 作成および初期リリース (FDPS バージョン 5.0 として)
- 2019/7/19
  - N 体/SPH サンプルコードの記述の改訂 (第 7.1 節))
- 2020/08/16
  - PIKG の利用法を追加 (第 3.4.1.8 節)

2 概要

本節では、Framework for Developing Particle Simulator (FDPS) および FDPS C 言語インターフェース の概要について述べる。FDPS は粒子シミュレーションのコード開発を支援するフレームワークである。FDPS が行うのは、計算コストの最も大きな粒子間相互作用の計算と、粒子間相互作用の計算のコストを負荷分散するための処理である。これらはマルチプロセス、マルチスレッドで並列に処理することができる。比較的計算コストが小さく、並列処理を必要としない処理 (粒子の軌道計算など) はユーザーが行う。

FDPS が対応している座標系は、2 次元直交座標系と 3 次元直交座標系である。また、境界条件としては、開放境界条件と周期境界条件に対応している。周期境界条件の場合、x、y、z 軸方向の任意の組み合わせの周期境界条件を課すことができる。

ユーザーは粒子間相互作用の形を定義する必要がある。定義できる粒子間相互作用の形には様々なものがある。粒子間相互作用の形を大きく分けると 2 種類あり、1 つは長距離力、もう 1 つは短距離力である。この 2 つの力は、遠くの複数の粒子からの作用を 1 つの超粒子からの作用にまとめるか (長距離力)、まとめないか (短距離力) という基準でもって分類される。

長距離力には、小分類があり、無限遠に存在する粒子からの力も計算するカットオフなし長距離力と、ある距離以上離れた粒子からの力は計算しないカットオフあり長距離力がある。前者は開境界条件下における重力やクーロン力に対して、後者は周期境界条件下の重力やクーロン力に使うことができる。後者のためには Particle Mesh 法などが必要となるが、これは FDPS の拡張機能として用意されている。

短距離力には、小分類が 4 つ存在する。短距離力の場合、粒子はある距離より離れた粒子からの作用は受けない。すなわち必ずカットオフが存在する。このカットオフ長の決め方によって、小分類がなされる。すなわち、全粒子のカットオフ長が等しいコンスタントカーネル、 カットオフ長が作用を受ける粒子固有の性質で決まるギャザーカーネル、カットオフ長が作用を与える粒子固有の性質で決まるスキャッタカーネル、カットオフ長が作用を受ける粒子と作用を与える粒子の両方の性質で決まるシンメトリックカーネルである。コンスタントカーネルは分子動力学における LJ 力に適用でき、その他のカーネルは SPH などに適用できる。

ユーザーは、粒子間相互作用や粒子の軌道積分などを、 C 言語 を用いて記述する。

3 入門：サンプルコードを動かしてみよう

本節では、まずはじめに、FDPS および FDPS C 言語インターフェース の動作環境、必要なソフトウェア、インストール方法などを説明し、その後、サンプルコードの使用方法を説明する。サンプルコードの中身に関しては、次節 (第 4 節) で詳しく述べる。

3.1 動作環境

FDPS は Linux, Mac OS X, Windows などの OS 上で動作する。

3.2 必要なソフトウェア

本節では、FDPS を使用する際に必要となるソフトウェアを記述する。まず標準機能を用いるのに必要なソフトウェア、次に拡張機能を用いるのに必要なソフトウェアを記述する。

3.2.1 標準機能

本節では、FDPS の標準機能のみを使用する際に必要なソフトウェアを記述する。最初に逐次処理機能のみを用いる場合（並列処理機能を用いない場合）に必要なソフトウェアを記述する。次に並列処理機能を用いる場合に必要なソフトウェアを記述する。

3.2.1.1 逐次処理

逐次処理の場合に必要なソフトウェアは以下の通りである。

- make
- C++コンパイラ (gcc バージョン 4.8.3 以降なら確実, K コンパイラバージョン 1.2.0 で動作確認済)
- C コンパイラ (上記 C++コンパイラと相互運用可能なもの。gcc 4.8.3 以降の gfortran なら確実)
- Python 2.7.5 以上、または、Python 3.4 以上 (これ以外での正常動作は保証しない。特に、Python 2.7 以前では動作しない)

3.2.1.2 並列処理

本節では、FDPS の並列処理機能を用いる際に必要なソフトウェアを記述する。まず、OpenMP を使用する際に必要なソフトウェア、次に MPI を使用する際に必要なソフトウェア、最後に OpenMP と MPI を同時に使用する際に必要なソフトウェアを記述する。

3.2.1.2.1 OpenMP

OpenMP を使用する際に必要なソフトウェアは以下の通り。

- make
- OpenMP 対応の C++コンパイラ (gcc version 4.8.3 以降なら確実, K コンパイラバージョン 1.2.0 で動作確認済)
- OpenMP 対応の C コンパイラ (上記 C++コンパイラと相互運用可能なもの。gcc 4.8.3 以降なら確実)
- Python 2.7.5 以上、または、Python 3.4 以上 (これ以外での正常動作は保証しない。特に、Python 2.7 以前では動作しない)

3.2.1.2.2 MPI

MPI を使用する際に必要なソフトウェアは以下の通り。

- make
- MPI version 1.3 対応の C++コンパイラ (Open MPI 1.6.4 で動作確認済, K コンパイラバージョン 1.2.0 で動作確認済)
- MPI version 1.3 対応の C コンパイラ (上記 C++コンパイラと相互運用可能なもの。Open MPI 1.6.4 で動作確認済み)
- Python 2.7.5 以上、または、Python 3.4 以上 (これ以外での正常動作は保証しない。特に、Python 2.7 以前では動作しない)

3.2.1.2.3 MPI+OpenMP

MPI と OpenMP を同時に使用する際に必要なソフトウェアは以下の通り。

- make
- MPI version 1.3 と OpenMP に対応の C++コンパイラ (Open MPI 1.6.4 で動作確認済, K コンパイラバージョン 1.2.0 で動作確認済)
- MPI version 1.3 と OpenMP に対応の C コンパイラ (上記 C++コンパイラと相互運用可能なもの。Open MPI 1.6.4 で動作確認ずみ)
- Python 2.7.5 以上、または、Python 3.4 以上 (これ以外での正常動作は保証しない。特に、Python 2.7 以前では動作しない)

3.2.2 拡張機能

本節では、FDPS の拡張機能を使用する際に必要なソフトウェアについて述べる。FDPS の拡張機能には Particle Mesh がある。以下では Particle Mesh を使用する際に必要なソフトウェアを述べる。

3.2.2.1 Particle Mesh

Particle Mesh を使用する際に必要なソフトウェアは以下の通りである。

- make
- MPI version 1.3 と OpenMP に対応の C++コンパイラ (Open MPI 1.6.4 で動作確認済)
- FFTW 3.3 以降

3.3 インストール

本節では、FDPS および C 言語インターフェース のインストールについて述べる。取得方法、ビルド方法について述べる。

3.3.1 取得方法

ここでは FDPS の取得方法を述べる。最初に最新バージョンの取得方法、次に過去のバージョンの取得方法を述べる。

3.3.1.1 最新バージョン

以下の方法のいずれかで FDPS の最新バージョンを取得できる。

- ブラウザから
  1. ウェブサイト https://github.com/FDPS/FDPS で”Download ZIP”をクリックし、ファイル FDPS-master.zip をダウンロード
  2. FDPS を展開したいディレクトリに移動し、圧縮ファイルを展開
- コマンドラインから
  - Subversion を用いる場合：以下のコマンドを実行するとディレクトリ trunk の下を Subversion レポジトリとして使用できる

```
$ svn co --depth empty https://github.com/FDPS/FDPS
$ cd FDPS
$ svn up trunk
```

– Git を用いる場合：以下のコマンドを実行するとカレントディレクトリにディレクトリ FDPS ができ、その下を Git のレポジトリとして使用できる

```
$ git clone git://github.com/FDPS/FDPS.git
```

3.3.1.2 過去のバージョン

以下の方法でブラウザから FDPS の過去のバージョンを取得できる。

- ウェブサイト https://github.com/FDPS/FDPS/releases に過去のバージョンが並んでいるので、ほしいバージョンをクリックし、ダウンロード
- FDPS を展開したいディレクトリに移動し、圧縮ファイルを展開

3.3.2 インストール方法

C++言語で記述された FDPS 本体はヘッダライブラリ[注 1)]のため、`configure` などを行う必要はない。基本的にはアーカイブを展開したあと、自分のソースファイルをコンパイルする時に適切なインクルードパスを設定すればよい。実際の手続きは第 3.4 節で説明するサンプルコードとその Makefile をみて欲しい。

C 言語の場合、コンパイル前に C 言語ソースファイルから FDPS とのインターフェースコードを生成する必要がある。その手順は仕様書 doc_spec_ftn_ja.pdf の第 6 章に記述されている。本サンプルコードの Makefile では、インターフェースコードが `make` コマンド実行中に自動的に生成されるようになっている。ユーザが自分のコードの Makefile を作る時にはサンプルコードの Makefile を参考にすることを推奨する。

3.4 サンプルコードの使用方法

本節ではサンプルコードの使用方法について説明する。サンプルコードには重力 N 体シミュレーションコードと、SPH シミュレーションコードがある。最初に重力 N 体シミュレーションコード、次に SPH シミュレーションコードの使用について記述する。サンプルコードは拡張機能を使用していない。

3.4.1 重力 N 体シミュレーションコード

本サンプルコードは、 FDPS C 言語インターフェース を用いて書かれた無衝突系の N 体計算コードである。このコードでは一様球のコールドコラプス問題を計算し、粒子分布のスナップショットを出力する。

注 1)ヘッダファイルだけで構成されるライブラリのこと

3.4.1.1 概要

以下の手順で本コードを使用できる。

- ディレクトリ$(FDPS)/sample/c/nbody に移動。これ以後、ディレクトリ$(FDPS) は FDPS の最も上の階層のディレクトリを指す ($(FDPS) は環境変数にはなっていない)。$(FDPS) は FDPS の取得によって異なり、ブラウザからなら FDPS-master, Subversion からなら trunk, Git からなら FDPS である。
- カレントディレクトリにある Makefile を編集
- コマンドライン上で make を実行
- nbody.out ファイルの実行
- 結果の解析

最後に x86 版 Phantom-GRAPE および PIKG を使う場合について述べる。

3.4.1.2 ディレクトリ移動

ディレクトリ$(FDPS)/sample/c/nbody に移動する。

3.4.1.3 Makefile の編集

サンプルコードのディレクトリには GCC 用に書かれた Makefile がある。以下、この Makefile について解説する。

まず、Makefile の初期設定について説明する。サンプルコードをコンパイルするにあたって、ユーザが設定すべき Makefile 変数は 4 つあり、C コンパイラを表す CC、C++コンパイラを表す CXX、それぞれのコンパイルオプションを表す CFLAGS, CXXFLAGS である。これらの初期設定値は次のようになっている:

```
CC=gcc
CXX=g++
CFLAGS = -O3 -ffast-math -funroll-loops -finline-functions $(FDPS_INC)
CXXFLAGS = -O3 -ffast-math -funroll-loops $(FDPS_INC)
```

ここで、$(FDPS_INC) は FDPS 本体をインクルードするために必要なインクルード PATH が格納された変数であり、Makefile 内で設定済みである。したがって、ここで変更する必要はない。

上記 4 つの Makefile 変数の値を適切に編集し、make コマンドを実行することで実行ファイルが得られる。OpenMP と MPI を使用するかどうかで編集方法が変わるため、以下でそれを説明する。

- OpenMP も MPI も使用しない場合

  - 変数 CC に C コンパイラを代入する
  - 変数 CXX に C++コンパイラを代入する
- OpenMP のみ使用の場合
  - 変数 CC に OpenMP 対応の C コンパイラを代入する
  - 変数 CXX に OpenMP 対応の C++コンパイラを代入する
  - CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp の行のコメントアウトを外す
  - CXXFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp の行のコメントアウトを外す
- MPI のみ使用の場合
  - 変数 CC に MPI 対応の C コンパイラを代入する
  - 変数 CXX に MPI 対応の C++コンパイラを代入する
  - CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL の行のコメントアウトを外す
  - CXXFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL の行のコメントアウトを外す
- OpenMP と MPI の同時使用の場合
  - 変数 CC に MPI 対応の C コンパイラを代入する
  - 変数 CXX に MPI 対応の C++コンパイラを代入する
  - CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp の行のコメントアウトを外す
  - CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL の行のコメントアウトを外す
  - CXXFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp の行のコメントアウトを外す
  - CXXFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL の行のコメントアウトを外す

次に、ユーザが本 Makefile をユーザコードで使用する場合に便利な情報を記述する。ユーザコードで使用する場合に最も重要となる Makefile 変数は、FDPS_LOC, HDR_USER_DEFINED_TYPE, SRC_USER の 3 つである。まず、変数 FDPS_LOC には、FDPS のトップディレクトリの PATH を格納する。本 Makefile では、FDPS のソースディレクトリの PATH や C 言語とのインターフェースコードを生成するスクリプトの PATH 等、FDPS に関連する各種な設定がこの変数の値に基いて自動的に設定されるようになっている。したがって、ユーザは適切に設定する必要がある。次に、変数 HDR_USER_DEFINED_TYPE, SRC_USER には、それぞれ、ユーザ定義型が記述された C 言語ヘッダーファイル名と、ユーザ定義型以外の部

分が記述されたC言語ファイル名を格納する。FDPSのC言語インターフェースコードはユーザーコードの構造体を記述する部分から生成されるので、その部分が記述されたファイルをHDR_USER_DEFINED_TYPEで、それ以外をSRC_USERで指定する。これにより、SRC_USERで指定したファイルが変更されてもFDPSの再コンパイルは起きなくなるので、コンパイル・リンクの時間が短くなる。但し、HDR_USER_DEFINED_TYPE、或いは、SRC_USERに格納された(複数の)ファイルの間に依存関係がある場合、依存関係を示すルールをMakefileに追記しなければならない点に注意して頂きたい。この記述方法に関しては、例えば、GNU makeのマニュアル等を読んで頂きたい。

3.4.1.4 make の実行

makeコマンドを実行する。 このとき、まずFDPSのC言語インターフェースプログラムが生成され、その後、インターフェースプログラムとサンプルコードが一緒にコンパイルされる。

3.4.1.5 実行

実行方法は以下の通りである。

- MPIを使用しない場合、コマンドライン上で以下のコマンドを実行する

```
$ ./nbody.out
```

- MPIを使用する場合、コマンドライン上で以下のコマンドを実行する

```
$ MPIRUN -np NPROC ./nbody.out
```

ここで、MPIRUNにはmpirunやmpiexecなどが、NPROCには使用するMPIプロセスの数が入る。

正しく終了すると、以下のようなログを出力する。energy errorは絶対値で1×10^{-3}のオーダーに収まっていればよい。

```
time:      7.000, energy error:  -3.8957312e-03
time:      8.000, energy error:  -3.7788873e-03
time:      9.000, energy error:  -3.7627744e-03
time:     10.000, energy error:  -3.7071071e-03
MemoryPool::finalize() is completed!
******** FDPS has successfully finished. ********
```

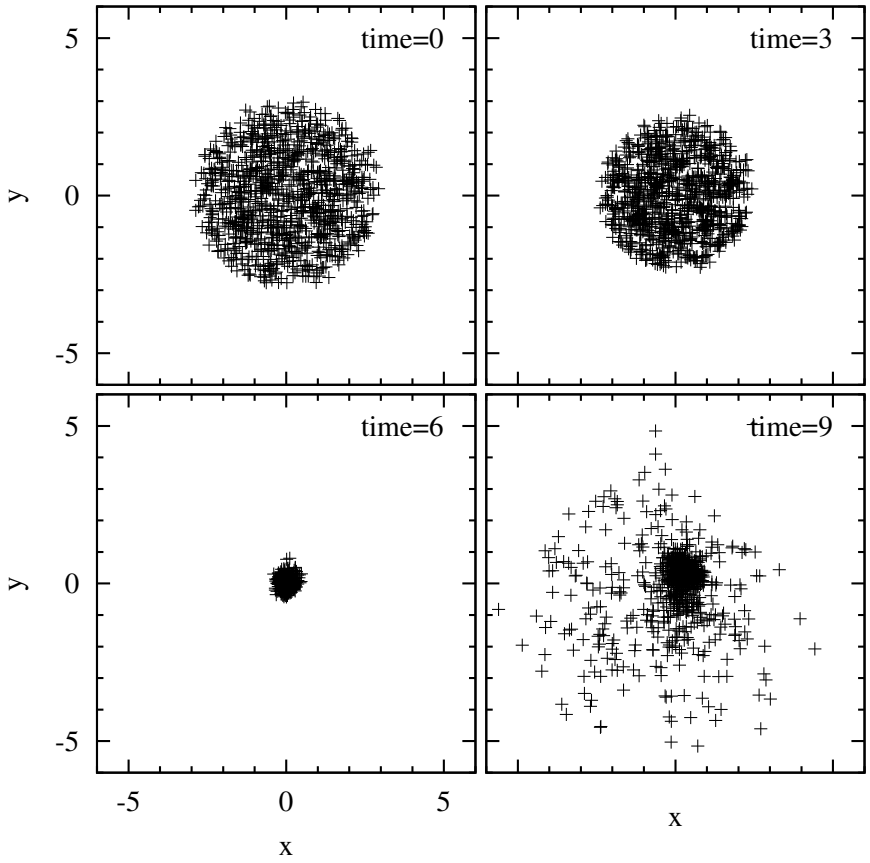

図 1:

3.4.1.6 結果の解析

ディレクトリ `result` に粒子分布を出力したファイル ”snap0000x-proc0000y.dat” ができている。ここで x は整数で時刻に対応している。y は MPI プロセス番号を表しており、MPI 実行しなければ常に y=0 である。

出力ファイルフォーマットは 1 列目から順に粒子の ID, 粒子の質量、位置の x, y, z 座標、粒子の x, y, z 軸方向の速度である。

ここで実行したのは、粒子数 1024 個からなる一様球 (半径 3) のコールドコラプスである。コマンドライン上で以下のコマンドを実行すれば、時刻 9 における xy 平面に射影した粒子分布を見ることができる。

```
$ cd result
$ cat snap00009-proc* > snap00009.dat
$ gnuplot
> plot "snap00009.dat" using 3:4
```

他の時刻の粒子分布をプロットすると、一様球が次第に収縮し、その後もう一度膨張する様子を見ることができる (図 1 参照)。

粒子数を10000個にして計算を行いたい場合には、ファイルc_main.cの中の関数c_main()の変数ntotを10000に設定し、再度、コンパイルした上で実行すればよい。

3.4.1.7 x86版Phantom-GRAPEを使う場合

Phantom-GRAPEはSIMD命令を効率的に活用することで重力相互作用の計算を高速に実行するライブラリである (詳細はTanikawa et al.[2012, New Astronomy, 17, 82] とTanikawa et al.[2012, New Astronomy, 19, 74] を参照のこと)。

まず、使用環境を確認する。SIMD命令セットAVXをサポートするIntel CPUまたはAMD CPUを搭載したコンピュータを使用しているならば、x86版Phantom-GRAPEを使用可能である。

次にディレクトリ$(FDPS)/src/phantom_grape_x86/G5/newton/libpg5に移動して、ファイルMakefileを編集し、コマンドmakeを実行してPhantom-GRAPEのライブラリlibpg5.aを作る。

最後に、ディレクトリ$(FDPS)/sample/c/nbodyに戻り、ファイルMakefile内の''#use_phantom_grape_x86 = yes''の''#''を消す。makeを実行してコンパイルする (OpenMP, MPIの使用・不使用どちらにも対応) と、x86版Phantom-GRAPEを使用したコードができている。上と同様の方法で実行・結果の確認を行うとさきほどと同様の結果が得られる。

Intel Core i5-3210M CPU @ 2.50GHz の2コアで性能テスト (OpenMP使用、MPI不使用) をした結果、粒子数8192の場合に、Phantom-GRAPEを使うと、使わない場合に比べて、最大で5倍弱ほど高速なコードとなる。

3.4.1.8 PIKGを使う場合

PIKGはDSL(ドメイン固有言語) 記述から様々なアーキテクチャ向けに最適化された粒子間相互作用カーネルを生成するカーネルジェネレータである.

ディレクトリ$(FDPS)/sample/c/nbodyのファイルMakefile内の''#use_pikg_x86 = yes''の''#''を消す。 この状態でmakeを実行してコンパイルする (OpenMP, MPIの使用・不使用どちらにも対応) と、PIKGから生成されたカーネルを使用した実行ファイルnbody.outができている。 上と同様の方法で実行・結果の確認を行うとさきほどと同様の結果が得られる。

デフォルトでは，通常のC言語コードと同等のreferenceモードの相互作用カーネルが生成される．Makefile内の#COVERSION_TYPEとその直後にある#CXXFLAGS の行のコメントアウトを外すと，別のアーキテクチャ(AVX2やAVX-512) 向けにコードを生成できる．AVX2モードを使う場合は，利用するCPUがAVX2及びFMAに対応していなくてはならない．さらにAVX-512モードを使う場合には利用するCPUがAVX-512FおよびAVX-512DQに対応していなくてはならない.

3.4.2 SPH シミュレーションコード

本サンプルコードには標準SPH法がFDPSを使って実装されている。簡単のため、smoothing lengthは一定値を取ると仮定している。コードでは、3次元の衝撃波管問題の初期条件を生成し、衝撃波管問題を実際に計算する。

3.4.2.1 概要

以下の手順で本コードを使用できる。

- ディレクトリ`$(FDPS)/sample/c/sph`に移動
- カレントディレクトリにある`Makefile`を編集 (後述)
- コマンドライン上で`make`を実行
- `sph.out`ファイルの実行 (後述)
- 結果の解析 (後述)

3.4.2.2 ディレクトリ移動

ディレクトリ`$(FDPS)/sample/c/sph`に移動する。

3.4.2.3 Makefile の編集

`Makefile`の編集の仕方はN体計算の場合と同一なので、 第3.4.1.3節を参照されたい。

3.4.2.4 `make` の実行

`make`コマンドを実行する。 N体計算のときと同様、このとき、まずFDPSのC言語インターフェースプログラムが生成され、その後、インターフェースプログラムとサンプルコードが一緒にコンパイルされる。

3.4.2.5 実行

実行方法は以下の通りである。

- MPIを使用しない場合、コマンドライン上で以下のコマンドを実行する

```
$ ./sph.out
```

- MPI を使用する場合、コマンドライン上で以下のコマンドを実行する

```
$ MPIRUN -np NPROC ./sph.out
```

ここで、MPIRUN には mpirun や mpiexec などが、NPROC には使用する MPI プロセスの数が入る。

正しく終了すると以下のようなログを出力する。

```
******** FDPS has successfully finished. ********
```

3.4.2.6 結果の解析

実行するとディレクトリ result にファイルが出力されている。 ファイル名は”snap0000x-proc0000y.dat” となっている。ここで、x,y は整数で、それぞれ、時刻と MPI プロセス番号を表す。MPI 実行でない場合には、常に y=0 である。 出力ファイルフォーマットは 1 列目から順に粒子の ID、粒子の質量、位置の x, y, z 座標、粒子の x, y, z 軸方向の速度、密度、内部エネルギー、圧力である。

コマンドライン上で以下のコマンドを実行すれば、横軸に粒子の x 座標、縦軸に粒子の密度をプロットできる (時刻は 40)。

```
$ cd result
$ cat snap00040-proc* > snap00040.dat
$ gnuplot
> plot "snap00040.dat" using 3:9
```

正しい答が得られれば、図 2 のような図を描ける。

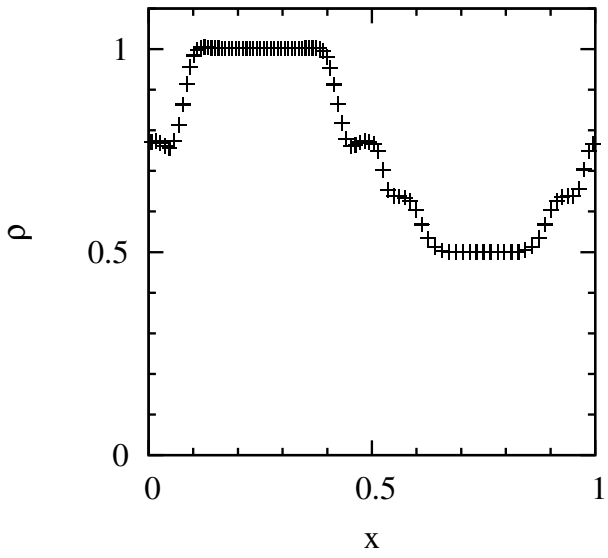

図 2: 衝撃波管問題の時刻 $t = 40$ における密度分布

4 サンプルコードの解説

本節では、前節 (第 3 節) で動かしたサンプルコードについての解説を行う。特に、ユーザが定義しなければならない構造体 (以後、**ユーザ定義型**と呼ぶ) や FDPS の各種 API の使い方について詳しく述べる。説明の重複を避けるため、いくつかの事項に関しては、その詳細な説明がN体シミュレーションコードの節でのみ行われている。そのため、SPH シミュレーションだけに興味があるユーザも、N 体シミュレーションコードの節に目を通して頂きたい。

4.1 N 体シミュレーションコード

4.1.1 ソースファイルの場所と構成

ソースファイルは`$(FDPS)/sample/c/nbody` 以下にある。 サンプルコードは、次節で説明するユーザ定義型が記述されたソースコード `user_defined.h` と、相互作用関数が定義された `user_defined.c`、N 体シミュレーションのメインループ等が記述されたソースコード `c_main.c` から構成される。この他に、GCC 用の `Makefile` がある。

4.1.2 ユーザー定義型・ユーザ定義関数

本節では、FDPS の機能を用いて N 体計算を行う際、ユーザーが記述しなければならない構造体と `void` 関数について記述する。

4.1.2.1 FullParticle 型

ユーザーはユーザ定義型の 1 つ FullParticle 型を記述しなければならない。FullParticle 型には、シミュレーションを行うにあたって、N 体粒子が持っているべき全ての物理量が含まれている。Listing 1 に本サンプルコードの FullParticle 型の実装例を示す (`user_defined.h` を参照)。

Listing 1: FullParticle 型

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

FDPS C 言語インターフェースを使ってユーザコードを開発する場合、ユーザは構造体がどのユーザ定義型 (FullParticle 型, EssentialParticleI 型, EssentialParticleJ 型, Force 型) に対応するかを FDPS に教えなければならない。本インターフェースにおいて、この指示は、構造体に決まった書式のコメント文を加えることによって行う (以後、この種のコメント文を **FDPS 指示文**と呼ぶ)。本サンプルコードでは、FullParticle 型が EssentialParticleI 型、EssentialParticleJ 型、そして、Force 型を兼ねている。そのため、構造体がすべてのユーザ定義型に対応すること指示する以下のコメント文を記述している:

```
typedef struct full_particle { //$fdps FP,EPI,EPJ,Force
```

また、FDPS は FullParticle 型のどのメンバ変数が質量や位置等の**必須物理量** (どの粒子計算でも必ず必要となる物理量、或いは、特定の粒子計算において必要とされる物理量と定義する) に対応するのかを知っていなければならない。この指示も決まった書式のコメント文をメンバ変数に対して記述することで行う。今回の例では、メンバ変数 mass, pos, vel が、それぞれ、質量、位置、速度に対応することを FDPS に指示するため、以下の指示文が記述されている:

```
double mass; //$fdps charge
fdps_f64vec pos; //$fdps position
fdps_f64vec vel; //$fdps velocity
```

ただし、メンバ変数が速度であることを指示する `//$fdps velocity` は予約語であり、指示は任意である (現時点で FDPS の振舞に一切影響しない)。

FullParticle 型は EssentialParticleI 型、EssentialParticleJ 型、Force 型との間でデータの移動 (データコピー) を行う。ユーザはこのコピーの仕方を指示する FDPS 指示文も記述しなければならない。本サンプルコードでは、以下のように記述している:

```
//$fdps copyFromForce full_particle (pot,pot) (acc,acc)
//$fdps copyFromFP full_particle (id,id) (mass,mass) (pos,pos)
```

ここで、キーワード `copyFromForce` を含む指示文は、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のかを指示するもので、FullParticle 型に**常に**記述しなければならない指示文である。一方、キーワード `copyFromFP` は FullParticle 型から EssentialParticleI 型および EssentialParticleJ 型へのデータコピーの仕方を指示するもので、EssentialParticleI 型と EssentialParticleJ 型には**必ず**記述しなければならない指示文である。今、FullParticle 型はこれら 2 つを兼ねているため、ここに記述している。

今、FullParticle 型は Force 型を兼ねている。Force 型にも必ず記述しなければならない指示文がある。それは、相互作用計算において、積算対象のメンバ変数をどのように 0 クリアするかを指示する指示文である。本サンプルコードでは、積算対象である加速度とポテンシャルのみを 0 クリアすることを指示するため、次の指示文を記述している:

```
//$fdps clear id=keep, mass=keep, pos=keep, vel=keep
```

ここで、キーワード clear の右に記述された構文 *mbr*=keep は、メンバ変数 *mbr* の値を変更しないことを指示する構文である。

FDPS 指示文の書式の詳細については、仕様書 doc_specs_ftn_ja.pdf をご覧頂きたい。

4.1.2.2 相互作用関数 calcForceEpEp

ユーザーは粒子間相互作用の仕方を記述した相互作用関数 calcForceEpEp を記述しなければならない。void 関数 calcForceEpEp には、粒子-粒子相互作用計算の具体的な内容を書く必要がある。Listing 2 に、本サンプルコードでの実装を示す (user_defined.c を参照)。

Listing 2: 関数 calcForceEpEp

```
1  void calc_gravity_ep_ep(Full_particle *ep_i,
2                          int n_ip,
3                          Full_particle *ep_j,
4                          int n_jp,
5                          Full_particle* f)
6  {
7      int i, j;
8      for (i=0; i<n_ip ;i++) {
9          Full_particle *pi = ep_i + i;
10         double eps2 = pi->eps * pi->eps;
11         double xi = pi->pos.x;
12         double yi = pi->pos.y;
13         double zi = pi->pos.z;
14         double ax, ay, az, pot;
15         ax = ay = az = pot = 0;
16         for (j=0; j<n_jp; j++) {
17             Full_particle *pj = ep_j + j;
18             double dx = xi - pj->pos.x;
19             double dy = yi - pj->pos.y;
20             double dz = zi - pj->pos.z;
21             double r2 = dx*dx+dy*dy+dz*dz+eps2;
22             double rinv = 1.0/sqrt(r2);
23             double mrinv = pj->mass* rinv;
24             double mr3inv = mrinv*rinv*rinv;
25             ax -= dx*mr3inv;
26             ay -= dy*mr3inv;
27             az -= dz*mr3inv;
28             pot = pot - mrinv;
29         }
30         Full_particle *pfi = f+i;
31         pfi->pot += pot;
32         pfi->acc.x += ax;
33         pfi->acc.y += ay;
34         pfi->acc.z += az;
35     }
36 }
```

本サンプルコードでは、void 関数 calc_gravity_ep_ep として実装されている。void 関数の仮引数は、EssentialParticleI の配列、EssentialParticleI の個数、EssentialParticleJ の配列、EssentialParticleJ の個数、Force 型の配列である。本サンプルコードでは、FullParticle

型がすべてのユーザ定義型を兼ねているため、引数のデータ型はすべて full_particle 型となっていることに注意して頂きたい。

4.1.2.3 相互作用関数 calcForceEpSp

ユーザーは粒子-超粒子間相互作用の仕方を記述した相互作用関数 calcForceEpSp を記述しなければならない。calcForceEpSp には、粒子-超粒子相互作用計算の具体的な内容を書く必要があり、void 関数として実装しなければならない。Listing 3 に、本サンプルコードでの実装を示す (user_defined.c を参照)。

Listing 3: 関数 calcForceEpSp

```
1  void calc_gravity_ep_sp(Full_particle *ep_i,
2                          int n_ip,
3                          fdps_spj_monopole *ep_j,
4                          int n_jp,
5                          Full_particle *f)
6  {
7      int i, j;
8      for (i=0; i<n_ip; i++) {
9          Full_particle *pi = ep_i + i;
10         double eps2 = pi->eps*pi->eps;
11         double xi = pi->pos.x;
12         double yi = pi->pos.y;
13         double zi = pi->pos.z;
14         double ax, ay, az, pot;
15         ax = ay = az = pot = 0;
16         for (j=0; j<n_jp; j++) {
17             fdps_spj_monopole *pj = ep_j + j;
18             double dx = xi - pj->pos.x;
19             double dy = yi - pj->pos.y;
20             double dz = zi - pj->pos.z;
21             double r2 = dx*dx+dy*dy+dz*dz+eps2;
22             double rinv = 1.0/sqrt(r2);
23             double mrinv = pj->mass* rinv;
24             double mr3inv = mrinv*rinv*rinv;
25             ax -= dx*mr3inv;
26             ay -= dy*mr3inv;
27             az -= dz*mr3inv;
28             pot = pot - mrinv;
29         }
30         Full_particle *pfi = f+i;
31         pfi->pot += pot;
32         pfi->acc.x += ax;
33         pfi->acc.y += ay;
34         pfi->acc.z += az;
35     }
36 }
```

本サンプルコードでは、void 関数 calc_gravity_ep_sp として実装されている。void 関数の仮引数は、EssentialParticleI の配列、EssentialParticleI の個数、超粒子の配列、超粒子の個数、Force 型の配列である。本サンプルコードでは、FullParticle 型がすべてのユーザ定義

型を兼ねているため、引数の Force 型は full_particle 型となっていることに注意して頂きたい。ここで指定する超粒子型はこの相互作用計算を実施するのに使用するツリーオブジェクトの種別と適合していなければならない。

4.1.3 プログラム本体

本節では、FDPS C 言語インターフェースを用いて N 体計算を行うにあたり、“メイン関数” c_main() に書かれるべき void 関数や関数に関して解説する。ここで、メイン関数とはっきり書かないのは、次の理由による: FDPS C 言語インターフェースを使用する場合、ユーザコードは必ずvoid 関数 c_main() の下に記述されなければならず、ユーザコードは正しい意味でのメイン関数を持たない (メイン関数はインターフェースプログラムの C++ ソースコード内にある)。しかし、実質的には void 関数 c_main() がメイン関数の役割を果たす。そのため、敢えて “メイン関数” という言葉を使った。メイン関数という言葉は、それがユーザコードの入り口であることを示すのに適しているので、以後、c_main() をメイン関数と呼ぶことにする。本サンプルコードのメイン関数は c_main.c に記述されている。

4.1.3.1 ヘッダーファイルのインクルード

FDPS の標準機能を利用できるようにするため、FDPS_c_if.h をインクルードする。

Listing 4: ヘッダーファイル FDPS_c_if.h のインクルード

```
1 #include "FDPS_c_if.h"
```

4.1.3.2 開始、終了

まずは、FDPS の初期化/開始を行う必要がある。次のように、メイン関数に記述する。

Listing 5: FDPS の開始

```
1 fdps_initialize();
```

FDPS は、開始したら明示的に終了させる必要がある。今回は、プログラムの終了と同時に FDPS も終了させるため、メイン関数の最後に次のように記述する。

Listing 6: FDPS の終了

```
1 fdps_finalize();
```

4.1.3.3 オブジェクトの生成・初期化

FDPS の初期化に成功した場合、ユーザーはコード中で用いるオブジェクトを作成する必要がある。本節では、オブジェクトの生成/初期化の仕方について解説する。

4.1.3.3.1 オブジェクトの生成

今回の計算では、粒子群オブジェクト、領域情報オブジェクトに加え、重力計算用のツリーオブジェクトを 1 個生成する必要がある。C 言語インターフェースでは、これらオブジェクトはすべて整数変数に格納された識別番号を使って操作する。したがって、まず識別番号を格納する整数変数を用意したあとに、オブジェクトを生成する API を呼び出す必要がある。以下にそのコードを記す。これらはサンプルコード `c_main.c` のメイン関数内に記述されている。

Listing 7: オブジェクトの生成

```
1  void c_main() {
2
3      // Create and initialize dinfo object
4      int dinfo_num;
5      fdps_create_dinfo(&dinfo_num);
6      // Create and initialize psys object
7      int psys_num;
8      fdps_create_psys(&psys_num,"full_particle");
9      // Create and initialize tree object
10     int tree_num;
11     fdps_create_tree(&tree_num,
12                      "Long,full_particle,full_particle,full_particle,
                              Monopole");
13
14 }
```

ここでも、実際のサンプルコードから該当部分だけを抜き出していることに注意して頂きたい。

上に示すように、粒子群オブジェクトを生成する際には FullParticle 型に対応する構造体名を文字列として API の引数に渡す必要がある。同様に、ツリーオブジェクト生成の際には、ツリーの種別を示す文字列を API の引数に渡す必要がある。両 API において、**構造体名は小文字で入力されなければならない**。

4.1.3.3.2 領域情報オブジェクトの初期化

ユーザーはオブジェクトを作成したら、そのオブジェクトの初期化を行う必要がある。本サンプルコードでは周期境界等は用いていないため、領域情報オブジェクトの初期化は API `fdps_init_dinfo` を実行するだけでよい:

Listing 8: 領域オブジェクトの初期化

```
1  fdps_init_dinfo(dinfo_num,coef_ema);
```

ここで、API `fdps_init_dinfo` の第 2 引数は領域分割に使用される指数移動平均の平滑化係数を表す。この係数の意味については仕様書に詳しい解説があるので、そちらを参照されたい。

4.1.3.3.3 粒子群オブジェクトの初期化

次に、粒子群オブジェクトの初期化を行う必要がある。粒子群オブジェクトの初期化は、API `fdps_init_psys` で行う:

Listing 9: 粒子群オブジェクトの初期化

```
1 fdps_init_psys(psys_num);
```

4.1.3.3.4 ツリーオブジェクトの初期化

次に、ツリーオブジェクトの初期化を行う必要がある。ツリーオブジェクトの初期化は API `fdps_init_tree` で行う。この API には、引数として大雑把な粒子数を渡す必要がある。今回は、全体の粒子数 (`ntot`) をセットしておく事にする:

Listing 10: ツリーオブジェクトの初期化

```
1 fdps_init_tree(tree_num,ntot,theta,
2                n_leaf_limit,n_group_limit);
```

この API の第 3 引数以降の引数の意味は次の通りである:

- `theta` — ツリー法で力の計算をする場合の見込み角についての基準
- `n_leaf_limit` — ツリーを切るのをやめる粒子数の上限
- `n_group_limit` — 相互作用リストを共有する粒子数の上限

4.1.3.4 粒子データの初期化

初期条件の設定を行うためには、粒子群オブジェクトに粒子データを入力する必要がある。(既に API `fdps_init_psys` で初期化済みの) 粒子群オブジェクトに、FullParticle 型粒子のデータを格納するには、粒子群オブジェクトの API `fdps_set_nptcl_loc` と `fdps_get_psys_cptr` を用いて、次のように行う:

Listing 11: 粒子データの初期化

```
1  void foo(psys_num) {
2     // Set # of local particles
3     nptcl_loc = 1024;
4     fdps_set_nptcl_loc(psys_num,nptcl_loc);
5
6     // Get the pointer to full particle data
7     Full_particle *ptcl = (Full_particle *)  fdps_get_psys_cptr(psys_num);
8
9     // Initialize particle data
10    int i;
11    for (i = 0; i < nptcl_loc; i++) {
12       ptcl[i].pos.x = /* Do something */ ;
13    }
14
15 }
```

まず、粒子群オブジェクトに粒子データを保存するのに必要なメモリを確保しなければならない。これを行うには API fdps_set_nptcl_loc を実行すればよい。この API は指定された粒子群オブジェクトのローカル粒子数 (自プロセスが管理する粒子数) の値を設定し、かつ、その粒子数を格納するのに必要なメモリを確保する。粒子データを初期化するためには、確保されたメモリのアドレスを取得しなければならない。これには API fdps_get_psys_cptr を使用する。 アドレスは void *ポインタとして返ってくるため、Full_particle *型にキャストしていること注意されたい。API fdps_get_psys_cptr によってポインタを設定した後は、ポインタを粒子配列のように使用することが可能である。

4.1.3.5 ループ

本節では、時間積分ループの中で行わなければならないことについて、解説する。

4.1.3.5.1 領域分割の実行

まずは、粒子分布に基いて、領域分割を実行する。本サンプルコードでは、これを領域情報オブジェクトの API fdps_decompose_domain_all を用いて行っている:

Listing 12: 領域分割の実行

```
1 if (num_loop % 4 == 0) {
2    fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
3 }
```

ここで、計算時間の節約のため、領域分割は 4 ループ毎に 1 回だけ行うようにしている。 第 3 引数は、領域分割する際に、自プロセスからどれだけ粒子をサンプリングするかを表す “重み”を表す変数である。この変数は > 0 である必要があるが、負値が指定された場合、FDPS のデフォルト値が採用される。詳細は仕様書を参照されたい。

4.1.3.5.2 粒子交換の実行

次に、領域情報に基いて、プロセス間の粒子の情報を交換する。これには、粒子群オブジェクトの API fdps_exchange_particle を用いる:

Listing 13: 粒子交換の実行

```
1 fdps_exchange_particle(psys_num,dinfo_num);
```

4.1.3.5.3 相互作用計算の実行

領域分割・粒子交換が終了したら、相互作用の計算を行う。これには、ツリーオブジェクトの API fdps_calc_force_all_and_write_back を用いる:

Listing 14: 相互作用計算の実行

```
1 void c_main() {
```

```
2
3     // Do somehting
4
5     fdps_calc_force_all_and_write_back(tree_num,
6                                        calc_gravity_ep_ep,
7                                        calc_gravity_ep_sp,
8                                        psys_num,
9                                        dinfo_num,
10                                       true,
11                                       FDPS_MAKE_LIST);
12
13    // Do something
14
15 }
```

ここで、API の第 2,3 引数には関数 calcForceEpEp, calcForceEpSp の関数ポインタを指定する。第 6 引数には、前回の相互作用計算の結果をクリアするかどうかを_Bool 値で指定する。第 7 引数は、相互作用リストの再利用を行うかどうかを指示するフラグで、ここでは新規に相互作用リストを作成して相互作用計算を行うモードを選択している。

4.1.3.5.4 時間積分

本サンプルコードでは、時間積分を Leapfrog 時間積分法によって行う。時間積分は形式的に、$K(\frac{\Delta t}{2})D(\Delta t)K(\frac{\Delta t}{2})$ と表される。ここで、Δt は時間刻み、$K(\cdot)$ は速度を指定された時間だけ時間推進するオペレータ、$D(\cdot)$ は位置を指定された時間だけ時間推進するオペレータである。本サンプルコードにおいて、これらのオペレータは、void 関数 kick と void 関数 drift として実装している。

時間積分ループの最初で、最初の $D(\Delta t)K(\frac{\Delta t}{2})$ の計算を行い、粒子の座標と速度の情報を更新している:

Listing 15: $D(\Delta t)K(\frac{\Delta t}{2})$ オペレータの計算

```
1 // Leapfrog: Kick-Drift
2 kick(psys_num,0.5*dt);
3 time_sys +=  dt;
4 drift(psys_num,dt);
```

時間積分ループの次の部分では、力の計算を行い、その後、最後の $K(\frac{\Delta t}{2})$ の計算を行っている:

Listing 16: $K(\frac{\Delta t}{2})$ オペレータの計算

```
1 // Leapfrog: Kick
2 kick(psys_num,0.5d0*dt);
```

4.1.3.6 粒子データの更新

上記で説明した kick や drift 等の void 関数で、粒子データを更新するためには、粒子群オブジェクトに格納されている粒子データにアクセスする必要がある。これは、第 4.1.3.4

節で説明した方法とほぼ同様に行う:

Listing 17: 粒子データの更新

```
1  void foo(psys_num) {
2     // Get # of local particles
3     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
4
5     // Get the pointer to full particle data
6     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
7
8     // Initialize or update particle data
9     int i;
10    for (i = 0; i < nptcl_loc; i++) {
11       ptcl[i].pos.x = /* Do something */ ;
12    }
13 }
```

API fdps_get_psys_cptr を使い、粒子群オブジェクトに格納された粒子データのアドレスをポインタとして受け取る。受け取ったポインタは要素数 nptcl_loc の粒子配列として振る舞うので、一般的な配列同様に値を更新すればよい。

4.1.4 ログファイル

計算が正しく開始すると、標準出力に、時間・エネルギー誤差の 2 つが出力される。以下はその出力の最も最初のステップでの例である。

Listing 18: 標準出力の例

```
1 time:     0.0000000000E+000, energy error:    -0.0000000000E+000
```

4.2 固定長 SPH シミュレーションコード

本節では、前節 (第 3 節) で使用した、固定 smoothing length での標準 SPH 法のサンプルコードの詳細について解説する。

4.2.1 ソースファイルの場所と構成

ソースファイルは$(FDPS)/sample/c/sph 以下にある。 サンプルコードは、次節で説明するユーザ定義型が記述されたソースコード user_defined.h、相互作用関数が記述された user_defined.c、数学定数が定義された mathematical_constants.*、SPH シミュレーションのメインループ等が記述されたソースコード c_main.c から構成される。この他に、GCC 用の Makefile がある。

4.2.2 ユーザー定義型・ユーザ定義関数

本節では、FDPS の機能を用いて SPH の計算を行う際に、ユーザーが記述しなければならない構造体と void 関数について記述する。

4.2.2.1 FullParticle 型

ユーザーはユーザ定義型の 1 つ FullParticle 型を記述しなければならない。FullParticle 型には、シミュレーションを行うにあたって、SPH 粒子が持っているべき全ての物理量が含まれている。Listing 19 に本サンプルコード中で用いる FullParticle 型の実装例を示す (`user_defined.h` を参照)。

Listing 19: FullParticle 型

```
1  typedef struct full_particle { //$fdps FP
2      //$fdps copyFromForce force_dens (dens,dens)
3      //$fdps copyFromForce force_hydro (acc,acc) (eng_dot,eng_dot) (dt,dt)
4      double mass; //$fdps charge
5      fdps_f64vec pos; //$fdps position
6      fdps_f64vec vel;
7      fdps_f64vec acc;
8      double dens;
9      double eng;
10     double pres;
11     double smth; //$fdps rsearch
12     double snds;
13     double eng_dot;
14     double dt;
15     long long int id;
16     fdps_f64vec vel_half;
17     double eng_half;
18 } Full_particle;
```

SPH サンプルコードでは N 体サンプルコードと異なり、FullPartice 型が他のユーザ定義型を兼ねることはない。したがって、この構造体が FullParticle 型であることを示すため、次の指示文を記述している:

```
typedef struct full_particle { //$fdps FP
```

SPH シミュレーションにおける相互作用は短距離力である。そのため、必須物理量として探索半径が加わる。粒子位置等の指定も含め、どのメンバ変数がどの必須物理量に対応しているかの指定を次の指示文で行っている:

```
double mass; //$fdps charge
fdps_f64vec pos; //$fdps position
double smth; //$fdps rsearch
```

N 体シミュレーションコードの節で述べたように、メンバ変数が粒子速度であることを指定するキーワード `velocity` は予約語でしかないため、本サンプルコードでは指定してない。

FullParticle 型は Force 型との間でデータコピーを行う。ユーザは指示文を使い、FDPS にデータコピーの仕方を教えなければならない。後述するように本 SPH サンプルコードには 2 つの Force 型が存在する。したがって、ユーザはそれぞれの Force 型に対して、指示文を記述する必要がある。本サンプルコードでは、以下のように記述している:

```
//$fdps copyFromForce force_dens (dens,dens)
//$fdps copyFromForce force_hydro (acc,acc) (eng_dot,eng_dot) (dt,dt)
```

4.2.2.2 EssentialParticleI 型

ユーザーは EssentialParticleI 型を記述しなければならない。EssentialParticleI 型には、Force の計算を行う際、i 粒子が持っているべき全ての物理量をメンバ変数として持っている必要がある。また、本サンプルコード中では、EssentialParticleJ 型も兼ねているため、j 粒子が持っているべき全ての物理量もメンバ変数として持っている必要がある。Listing 20 に、本サンプルコードの EssentialParticleI 型の実装例を示す (`user_defined.h` 参照):

Listing 20: EssentialParticleI 型

```
1  typedef struct essential_particle { //$fdps EPI,EPJ
2      //$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,
           mass) (smth,smth) (dens,dens) (pres,pres) (snds,snds)
3      long long int id;
4      fdps_f64vec pos; //$fdps position
5      fdps_f64vec vel;
6      double mass; //$fdps charge
7      double smth; //$fdps rsearch
8      double dens;
9      double pres;
10     double snds;
11 } Essential_particle;
```

まず、ユーザは指示文を用いて、この構造体が EssentialParticleI 型かつ EssentialParticleJ 型であることを FDPS に教えなければならない。本サンプルコードでは、以下のように記述している:

```
typedef struct essential_particle { //$fdps EPI,EPJ
```

次に、ユーザはこの構造体のどのメンバ変数がどの必須物理量に対応するのかを指示文によって指定しなければならない。今回は SPH シミュレーションを行うので探索半径の指定も必要である。本サンプルコードでは、以下のように記述している:

```
fdps_f64vec pos; //$fdps position
double mass; //$fdps charge
double smth; //$fdps rsearch
```

EssentialParticleI 型と EssentialParticleJ 型は FullParticle 型からデータを受け取る。ユーザは FullParticle 型のどのメンバ変数を EssentialParticle?型 (?=I,J) のどのメンバ変数にコピーするのかを、指示文を用いて指定する必要がある。本サンプルコードでは、以下のように記述している:

```
//$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,mass)
(smth,smth) (dens,dens) (pres,pres) (snds,snds)
```

4.2.2.3 Force 型

ユーザーは Force 型を記述しなければならない。Force 型には、Force の計算を行った際にその結果として得られる全ての物理量をメンバ変数として持っている必要がある。また、本サンプルコード中では、Force の計算は密度の計算と流体相互作用計算の 2 つが存在するため、Force 型は 2 つ書く必要がある。Listing 21 に、本サンプルコード中で用いる Force 型の実装例を示す。

Listing 21: Force 型

```
1  typedef struct force_dens { //$fdps Force
2      //$fdps clear smth=keep
3      double dens;
4      double smth;
5  } Force_dens;
6
7  typedef struct force_hydro { //$fdps Force
8      //$fdps clear
9      fdps_f64vec acc;
10     double eng_dot;
11     double dt;
12 } Force_hydro;
```

まず、ユーザはこれらの構造体が Force 型であることを指示文によって指定する必要がある。この実装例では、それぞれの構造体に対して、以下のように記述している:

```
typedef struct force_dens { //$fdps Force
typedef struct force_hydro { //$fdps Force
```

これらの構造体は Force 型であるから、ユーザは**必ず**、相互作用計算における積算対象のメンバ変数の初期化方法を指定する必要がある。本サンプルコードでは、積算対象である密度、(圧力勾配による) 加速度、エネルギー密度の変化率、時間刻みのみを 0 クリアする指示を出している:

```
//$fdps clear smth=keep
//$fdps clear
```

この例において、Force 型 `force_dens` には、smoothing length を表すメンバ変数 `smth` が用意されている。本来、固定長 SPH では、Force 型に smoothing length に対応するメンバを持たせる必要はない。しかし、ここでは、ユーザが将来的に可変長 SPH に移行することを想定して用意してある。可変長 SPH の formulation の 1 つである Springel [2005,MNRAS,364,1105] の方法では、密度計算と同時に smoothing length を計算する必要がある。そのような formulation を実装する場合には、この例のように、Force 型に smoothing length を持たせる必要が生じ

る。本サンプルコードでは固定長 SPH を使うため、smth を 0 クリアされないようにしている (0 クリアされては 2 回目以降の密度計算が破綻するため)。

4.2.2.4 相互作用関数 calcForceEpEp

ユーザーは粒子間相互作用の仕方を記述した相互作用関数 calcForceEpEp を記述しなければならない。相互作用関数 calcForceEpEp には、各 Force 型に対応する粒子-粒子相互作用計算の具体的な内容を書く必要がある。Listing 22 に、本サンプルコードでの実装を示す (user_defined.c を参照)。

Listing 22: 関数 calcForceEpEp

```
1  void calc_density(Essential_particle *ep_i,
2                    int n_ip,
3                    Essential_particle *ep_j,
4                    int n_jp,
5                    Force_dens *f) {
6      int i,j;
7      fdps_f64vec dr;
8      for (i = 0; i< n_ip; i++) {
9          for (j = 0; j < n_jp; j++) {
10             dr.x = ep_j[j].pos.x - ep_i[i].pos.x;
11             dr.y = ep_j[j].pos.y - ep_i[i].pos.y;
12             dr.z = ep_j[j].pos.z - ep_i[i].pos.z;
13             f[i].dens += ep_j[j].mass * W(dr,ep_i[i].smth);
14         }
15     }
16
17 }
18
19 void calc_hydro_force(Essential_particle *ep_i,
20                       int n_ip,
21                       Essential_particle *ep_j,
22                       int n_jp,
23                       Force_hydro *f) {
24     // Local parameters
25     const double C_CFL=0.3;
26     // Local variables
27     int i,j;
28     double mass_i,mass_j,smth_i,smth_j,
29            dens_i,dens_j,pres_i,pres_j,
30            snds_i,snds_j;
31     double povrho2_i,povrho2_j,
32            v_sig_max,dr_dv,w_ij,v_sig,AV;
33     fdps_f64vec pos_i,pos_j,vel_i,vel_j,
34                 dr,dv,gradW_i,gradW_j,gradW_ij;
35
36     for (i = 0; i < n_ip; i++) {
37         // Zero-clear
38         v_sig_max = 0.0;
39         // Extract i-particle info.
40         pos_i.x = ep_i[i].pos.x;
41         pos_i.y = ep_i[i].pos.y;
```

```
42          pos_i.z = ep_i[i].pos.z;
43          vel_i.x = ep_i[i].vel.x;
44          vel_i.y = ep_i[i].vel.y;
45          vel_i.z = ep_i[i].vel.z;
46          mass_i  = ep_i[i].mass;
47          smth_i  = ep_i[i].smth;
48          dens_i  = ep_i[i].dens;
49          pres_i  = ep_i[i].pres;
50          snds_i  = ep_i[i].snds;
51          povrho2_i = pres_i/(dens_i*dens_i);
52          for (j = 0; j < n_jp; j++) {
53              // Extract j-particle info.
54              pos_j.x = ep_j[j].pos.x;
55              pos_j.y = ep_j[j].pos.y;
56              pos_j.z = ep_j[j].pos.z;
57              vel_j.x = ep_j[j].vel.x;
58              vel_j.y = ep_j[j].vel.y;
59              vel_j.z = ep_j[j].vel.z;
60              mass_j  = ep_j[j].mass;
61              smth_j  = ep_j[j].smth;
62              dens_j  = ep_j[j].dens;
63              pres_j  = ep_j[j].pres;
64              snds_j  = ep_j[j].snds;
65              povrho2_j = pres_j/(dens_j*dens_j);
66              // Compute dr & dv
67              dr.x = pos_i.x - pos_j.x;
68              dr.y = pos_i.y - pos_j.y;
69              dr.z = pos_i.z - pos_j.z;
70              dv.x = vel_i.x - vel_j.x;
71              dv.y = vel_i.y - vel_j.y;
72              dv.z = vel_i.z - vel_j.z;
73              // Compute the signal velocity
74              dr_dv = dr.x * dv.x + dr.y * dv.y + dr.z * dv.z;
75              if (dr_dv < 0.0) {
76                  w_ij = dr_dv / sqrt(dr.x * dr.x + dr.y * dr.y + dr.z * dr.z
                        );
77              } else {
78                  w_ij = 0.0;
79              }
80              v_sig = snds_i + snds_j - 3.0 * w_ij;
81              if (v_sig > v_sig_max) v_sig_max=v_sig;
82              // Compute the artificial viscosity
83              AV = - 0.5*v_sig*w_ij / (0.5*(dens_i+dens_j));
84              // Compute the average of the gradients of kernel
85              gradW_i  = gradW(dr,smth_i);
86              gradW_j  = gradW(dr,smth_j);
87              gradW_ij.x = 0.5 * (gradW_i.x + gradW_j.x);
88              gradW_ij.y = 0.5 * (gradW_i.y + gradW_j.y);
89              gradW_ij.z = 0.5 * (gradW_i.z + gradW_j.z);
90              // Compute the acceleration and the heating rate
91              f[i].acc.x -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.x;
92              f[i].acc.y -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.y;
93              f[i].acc.z -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.z;
94              f[i].eng_dot += mass_j * (povrho2_i + 0.5*AV)
95                              *(dv.x * gradW_ij.x
```

```
 96                                  +dv.y * gradW_ij.y
 97                                  +dv.z * gradW_ij.z);
 98          }
 99          f[i].dt = C_CFL*2.0*smth_i/(v_sig_max*kernel_support_radius);
100      }
101  }
```

本 SPH シミュレーションコードでは、2 種類の相互作用があるため、calcForceEpEp は 2 つ記述する必要がある。いずれの場合にも、void 関数の仮引数は、EssentialParticleI の配列、EssentialParticleI の個数、EssentialParticleJ の配列、EssentialParticleJ の個数、Force 型の配列である。

4.2.3 プログラム本体

本節では、FDPS を用いて SPH 計算を行う際に、メイン関数に書かれるべき関数に関して解説する (本文書におけるメイン関数の定義については、第 4.1.3 節を参照のこと) 。

4.2.3.1 ヘッダーファイルのインクルード

FDPS の標準機能を利用できるようにするため、FDPS_c_if.h をインクルードする。

Listing 23: ヘッダーファイル FDPS_c_if.h のインクルード

```
1 #include "FDPS_c_if.h"
```

4.2.3.2 開始、終了

まずは、FDPS の初期化/開始を行う必要がある。次のように、メイン関数に記述する。

Listing 24: FDPS の開始

```
1 fdps_initialize();
```

FDPS は、開始したら明示的に終了させる必要がある。今回は、プログラムの終了と同時に FDPS も終了させるため、メイン関数の最後に次のように記述する。

Listing 25: FDPS の終了

```
1 fdps_finalize();
```

4.2.3.3 オブジェクトの生成・初期化

FDPS の初期化に成功した場合、ユーザーはコード中で用いるオブジェクトを作成する必要がある。本節では、オブジェクトの生成/初期化の仕方について、解説する。

4.2.3.3.1 オブジェクトの生成

SPH では、粒子群オブジェクト、領域情報オブジェクトに加え、密度計算用に Gather 型の短距離力用ツリーを 1 本、流体相互作用計算用に Symmetry 型の短距離力用ツリーを 1 本生成する必要がある。以下にそのコードを記す。

Listing 26: オブジェクトの生成

```
1  void c_main() {
2      // Make an instance of ParticleSystem and initialize it
3      int psys_num;
4      fdps_create_psys(&psys_num,"full_particle");
5
6      // Make an instance of DomainInfo and initialize it
7      int dinfo_num;
8      fdps_create_dinfo(&dinfo_num);
9
10     // Make two tree structures
11     int tree_num_dens;
12     fdps_create_tree(&tree_num_dens,
13                      "Short,force_dens,essential_particle,
                               essential_particle,Gather");
14     int tree_num_hydro;
15     fdps_create_tree(&tree_num_hydro,
16                      "Short,force_hydro,essential_particle,
                               essential_particle,Symmetry");
17 }
```

ここでも、実際のサンプルコードから該当部分だけを抜き出していることに注意して頂きたい。API `fdps_create_psys` と `fdps_create_tree` には、それぞれ、粒子種別とツリー種別を示す文字列を渡す。**これら文字列の中のすべての構造体名は小文字で記述されなければならないことに注意して頂きたい**。

4.2.3.3.2 領域情報オブジェクトの初期化

ユーザーはオブジェクトを作成したら、そのオブジェクトの初期化を行う必要がある。ここでは、まず領域情報オブジェクトの初期化について、解説する。領域情報オブジェクトの初期化が終わった後、領域情報オブジェクトに周期境界の情報と、境界の大きさをセットする必要がある。今回のサンプルコードでは、x, y, z 方向に周期境界を用いる。

Listing 27: 領域情報オブジェクトの初期化

```
1 fdps_init_dinfo(dinfo_num,coef_ema);
2 fdps_set_boundary_condition(dinfo_num,FDPS_BC_PERIODIC_XYZ);
3 fdps_set_pos_root_domain(dinfo_num,&pos_ll,&pos_ul);
```

4.2.3.3.3 粒子群オブジェクトの初期化

次に、粒子群オブジェクトの初期化を行う必要がある。粒子群オブジェクトの初期化は、次の一文だけでよい。

Listing 28: 粒子群オブジェクトの初期化

```
1 fdps_init_psys(psys_num);
```

4.2.3.3.4 ツリーオブジェクトの初期化

次に、ツリーオブジェクトの初期化を行う必要がある。ツリーオブジェクトの初期化を行う関数には、引数として大雑把な粒子数を渡す必要がある。今回は、粒子数の 3 倍程度をセットしておく事にする。

Listing 29: 相互作用ツリークラスの初期化

```
1 fdps_init_tree(tree_num_dens,3*ntot,theta, &
2                n_leaf_limit,n_group_limit);
3 fdps_init_tree(tree_num_hydro,3*ntot,theta, &
4                n_leaf_limit,n_group_limit);
```

4.2.3.4 ループ

本節では、時間積分ループの中で行わなければならないことについて、解説する。

4.2.3.4.1 領域分割の実行

まずは、粒子分布に基いて、領域分割を実行する。これには、領域情報オブジェクトの API `fdps_decompose_domain_all` を用いる。

Listing 30: 領域分割の実行

```
1 fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
```

4.2.3.4.2 粒子交換の実行

次に、領域情報に基いて、プロセス間の粒子の情報を交換する。これには、粒子群オブジェクトの API `fdps_exchange_particle` を用いる。

Listing 31: 粒子交換の実行

```
1 fdps_exchange_particle(psys_num,dinfo_num);
```

4.2.3.4.3 相互作用計算の実行

領域分割・粒子交換が終了したら、相互作用の計算を行う。これには、ツリーオブジェクトの API `fdps_calc_force_all_and_write_back` を用いる。

Listing 32: 相互作用計算の実行

```
1  void c_main() {
2
3      // Do something
4
5      fdps_calc_force_all_and_write_back(tree_num_dens,
6                                         calc_density,
7                                         NULL,
8                                         psys_num,
9                                         dinfo_num,
10                                        true,
11                                        FDPS_MAKE_LIST);
12     call set_pressure(psys_num);
13     fdps_calc_force_all_and_write_back(tree_num_hydro,
14                                        calc_hydro_force,
15                                        NULL,
16                                        psys_num,
17                                        dinfo_num,
18                                        true,
19                                        FDPS_MAKE_LIST);
20
21     // Do something
22
23 }
```

4.2.4 コンパイル

作業ディレクトリで make コマンドを打てばよい。Makefile としては、サンプルコードに付属の Makefile をそのまま用いる事にする。

```
$ make
```

4.2.5 実行

MPI を使用しないで実行する場合、コマンドライン上で以下のコマンドを実行すればよい。

```
$ ./sph.out
```

もし、MPI を用いて実行する場合は、以下のコマンドを実行すればよい。

```
$ MPIRUN -np NPROC ./sph.out
```

ここで、MPIRUN には mpirun や mpiexec などの MPI 実行プログラムが、NPROC にはプロセス数が入る。

4.2.6 ログファイル

計算が終了すると、result フォルダ下にログが出力される。

4.2.7 可視化

ここでは、gnuplot を用いた可視化の方法について解説する。gnuplot で対話モードに入るために、コマンドラインから gnuplot を起動する。

```
$ gnuplot
```

対話モードに入ったら、gnuplot を用いて可視化を行う。今回は、50 番目のスナップショットファイルから、横軸を粒子の x 座標、縦軸を密度に取ったグラフを生成する。

```
gnuplot> plot "result/snap00050-proc00000.dat" u 3:9
```

ここで、文字列 proc の後の整数は MPI のプロセス番号を表す。

5 サンプルコード

5.1 N 体シミュレーション

N 体シミュレーションのサンプルコードを以下に示す。このサンプルは第 3, 4 節で用いた N 体シミュレーションのサンプルコードと同じものである。これをカット&ペーストしてコンパイルすれば、正常に動作する N 体シミュレーションコードを作ることができる。

Listing 33: N 体シミュレーションのサンプルコード (user_defined.h)

```
#pragma once
/* Standard headers */
#include <math.h>
/* FDPS headers */
#include "FDPS_c_if.h"

typedef struct full_particle { //$fdps FP,EPI,EPJ,Force
    //$fdps copyFromForce full_particle (pot,pot) (acc,acc)
    //$fdps copyFromFP full_particle (id,id) (mass,mass) (eps,eps) (pos,
        pos)
    //$fdps clear id=keep, mass=keep, eps=keep, pos=keep, vel=keep
    long long id; // $fdps id
    double  mass; //$fdps charge
    double  eps;
    fdps_f64vec pos; //$fdps position
    fdps_f64vec vel; //$fdps velocity
    double  pot;
    fdps_f64vec acc;
} Full_particle;

#ifndef USE_PIKG_KERNEL
void  calc_gravity_ep_ep(Full_particle *ep_i,
                         int n_ip,
                         Full_particle *ep_j,
                         int n_jp,
                         Full_particle *f);

void  calc_gravity_ep_sp(Full_particle *ep_i,
                         int n_ip,
                         fdps_spj_monopole *ep_j,
                         int n_jp,
                         Full_particle *f);
#endif
```

Listing 34: N 体シミュレーションのサンプルコード (user_defined.c)

```
#include "user_defined.h"

void calc_gravity_ep_ep(Full_particle *ep_i,
                        int n_ip,
                        Full_particle *ep_j,
                        int n_jp,
                        Full_particle* f)
```

```
 8 {
 9     int i, j;
10     for (i=0; i<n_ip ;i++) {
11         Full_particle *pi = ep_i + i;
12         double eps2 = pi->eps * pi->eps;
13         double xi = pi->pos.x;
14         double yi = pi->pos.y;
15         double zi = pi->pos.z;
16         double ax, ay, az, pot;
17         ax = ay = az = pot = 0;
18         for (j=0; j<n_jp; j++) {
19             Full_particle *pj = ep_j + j;
20             double dx = xi - pj->pos.x;
21             double dy = yi - pj->pos.y;
22             double dz = zi - pj->pos.z;
23             double r2 = dx*dx+dy*dy+dz*dz+eps2;
24             double rinv = 1.0/sqrt(r2);
25             double mrinv = pj->mass* rinv;
26             double mr3inv = mrinv*rinv*rinv;
27             ax -= dx*mr3inv;
28             ay -= dy*mr3inv;
29             az -= dz*mr3inv;
30             pot = pot - mrinv;
31         }
32         Full_particle *pfi = f+i;
33         pfi->pot += pot;
34         pfi->acc.x += ax;
35         pfi->acc.y += ay;
36         pfi->acc.z += az;
37     }
38 }
39
40 void calc_gravity_ep_sp(Full_particle *ep_i,
41                         int n_ip,
42                         fdps_spj_monopole *ep_j,
43                         int n_jp,
44                         Full_particle *f)
45 {
46     int i, j;
47     for (i=0; i<n_ip; i++) {
48         Full_particle *pi = ep_i + i;
49         double eps2 = pi->eps*pi->eps;
50         double xi = pi->pos.x;
51         double yi = pi->pos.y;
52         double zi = pi->pos.z;
53         double ax, ay, az, pot;
54         ax = ay = az = pot = 0;
55         for (j=0; j<n_jp; j++) {
56             fdps_spj_monopole *pj = ep_j + j;
57             double dx = xi - pj->pos.x;
58             double dy = yi - pj->pos.y;
59             double dz = zi - pj->pos.z;
60             double r2 = dx*dx+dy*dy+dz*dz+eps2;
61             double rinv = 1.0/sqrt(r2);
62             double mrinv = pj->mass* rinv;
```

```
63              double mr3inv = mrinv*rinv*rinv;
64              ax -= dx*mr3inv;
65              ay -= dy*mr3inv;
66              az -= dz*mr3inv;
67              pot = pot - mrinv;
68          }
69          Full_particle *pfi = f+i;
70          pfi->pot += pot;
71          pfi->acc.x += ax;
72          pfi->acc.y += ay;
73          pfi->acc.z += az;
74      }
75  }
```

Listing 35: N 体シミュレーションのサンプルコード (c_main.c)

```
1  /* Standard headers */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <math.h>
6  /* FDPS headers */
7  #include "user_defined.h"
8  #ifdef USE_PIKG_KERNEL
9  #include "kernel_epep.h"
10 #include "kernel_epsp.h"
11 #endif
12 #include "FDPS_c_if.h"
13
14 void dump_fullp(Full_particle p)
15 {
16     printf("%lld␣%15.7e␣%15.7e␣%15.7e␣%15.7e␣%15.7e␣%15.7e␣%15.7e",
17            p.id, p.mass, p.pos.x, p.pos.y, p.pos.z,
18            p.vel.x, p.vel.y, p.vel.z);
19     printf("%15.7e␣%15.7e␣%15.7e␣%15.7e\n",
20            p.acc.x, p.acc.y, p.acc.z, p.pot);
21 }
22 void dump_fullpsys(Full_particle *p, int n)
23 {
24     int i;
25     for (i=0;i<n;i++)dump_fullp(p[i]);
26 }
27
28 void dump_particles(int psys_num)
29 {
30     Full_particle *ptcl;
31     ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
32     int n = fdps_get_nptcl_loc(psys_num);
33     dump_fullpsys(ptcl, n);
34 }
35
36 void setup_IC(int psys_num,
37               int nptcl_glb)
38 {
39
40     double m_tot=1.0;
```

```
41      double rmax=3.0;
42      double r2max=rmax*rmax;
43      //   Get # of MPI processes and rank number
44      int nprocs = fdps_get_num_procs();
45      int myrank = fdps_get_rank();
46      // Make an initial condition at RANK 0
47      if (myrank == 0 ){
48          //Set # of local particles
49          fdps_set_nptcl_loc(psys_num,nptcl_glb);
50          Full_particle *ptcl;
51          ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
52          //** initialize Mersenne twister
53          int mtts_num;
54          fdps_create_mtts(&mtts_num);
55          fdps_mtts_init_genrand(mtts_num,0);
56          int i;
57          for (i=0; i < nptcl_glb; i++){
58              Full_particle *q = ptcl+i;
59              q->id = i;
60              q->mass = m_tot/nptcl_glb;
61              double r2 = r2max*2;
62              fdps_f64vec pos;
63              while (r2 >= r2max){
64                  pos.x= (2*fdps_mtts_genrand_res53(mtts_num)-1.0) * rmax;
65                  pos.y= (2*fdps_mtts_genrand_res53(mtts_num)-1.0) * rmax;
66                  pos.z= (2*fdps_mtts_genrand_res53(mtts_num)-1.0) * rmax;
67                  r2 = pos.x*pos.x + pos.y*pos.y + pos.z*pos.z;
68              }
69              q->pos = pos;
70              q->vel.x = 0.0;
71              q->vel.y = 0.0;
72              q->vel.z = 0.0;
73              q->eps = 1.0/32.0;
74          }
75          fdps_f64vec cm_pos;
76          fdps_f64vec cm_vel;
77          cm_pos.x = 0.0; cm_pos.y = 0.0; cm_pos.z = 0.0;
78          cm_vel.x = 0.0; cm_vel.y = 0.0; cm_vel.z = 0.0;
79          double cm_mass = 0;
80          for (i=0; i < nptcl_glb; i++){
81              Full_particle *pi = ptcl+i;
82              cm_pos.x +=  pi->pos.x* pi->mass;
83              cm_pos.y +=  pi->pos.y* pi->mass;
84              cm_pos.z +=  pi->pos.z* pi->mass;
85              cm_vel.x +=  pi->vel.x* pi->mass;
86              cm_vel.y +=  pi->vel.y* pi->mass;
87              cm_vel.z +=  pi->vel.z* pi->mass;
88              cm_mass += pi->mass;
89          }
90          cm_pos.x /= cm_mass;
91          cm_pos.y /= cm_mass;
92          cm_pos.z /= cm_mass;
93          cm_vel.x /= cm_mass;
94          cm_vel.y /= cm_mass;
95          cm_vel.z /= cm_mass;
```

```
96          for (i=0; i < nptcl_glb; i++){
97              Full_particle* q = ptcl+i;
98              q->pos.x -= cm_pos.x;
99              q->pos.y -= cm_pos.y;
100             q->pos.z -= cm_pos.z;
101             q->vel.x -= cm_vel.x;
102             q->vel.y -= cm_vel.y;
103             q->vel.z -= cm_vel.z;
104         }
105         //dump_fullpsys(ptcl, nptcl_glb);
106     } else{
107         fdps_set_nptcl_loc(psys_num,0);
108     }
109 }
110
111 void calc_energy(int psys_num,
112                  double *etot,
113                  double *ekin,
114                  double *epot)
115 {
116     *etot = *ekin = *epot = 0;
117     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
118     Full_particle *ptcl;
119     ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
120
121     double  ekin_loc = 0.0;
122     double  epot_loc = 0.0;
123     int i;
124     for (i=0;i < nptcl_loc; i++){
125         Full_particle *pi = ptcl+i;
126         fdps_f64vec v = pi->vel;
127         ekin_loc += pi->mass * (v.x*v.x+v.y*v.y+v.z*v.z);
128         epot_loc += pi->mass * (pi->pot + pi->mass/pi->eps);
129     }
130     ekin_loc *= 0.5;
131     epot_loc *= 0.5;
132     double etot_loc = ekin_loc + epot_loc;
133     *ekin = fdps_get_sum_f64(ekin_loc);
134     *epot = fdps_get_sum_f64(epot_loc);
135     *etot = fdps_get_sum_f64(etot_loc);
136 }
137
138 void  kick(int psys_num, double dt)
139 {
140     Full_particle *ptcl;
141     ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
142     int n = fdps_get_nptcl_loc(psys_num);
143     int i;
144     for (i=0;i < n; i++){
145         Full_particle *pi = ptcl+i;
146         fdps_f64vec *pv, *pa;
147         pv = &(pi->vel);
148         pa = &(pi->acc);
149         pv->x += pa->x * dt;
150         pv->y += pa->y * dt;
```

```
151         pv->z += pa->z * dt;
152     }
153 }
154
155 void  drift(int psys_num, double dt)
156 {
157     Full_particle *ptcl;
158     ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
159     int n = fdps_get_nptcl_loc(psys_num);
160     int i;
161     for (i=0;i < n; i++){
162         Full_particle *pi = ptcl+i;
163         fdps_f64vec *px, *pv;
164         pv = &(pi->vel);
165         px = &(pi->pos);
166         px->x += pv->x * dt;
167         px->y += pv->y * dt;
168         px->z += pv->z * dt;
169     }
170 }
171
172 int c_main()
173 {
174     fprintf(stderr, "FDPS␣on␣C␣test␣code\n");
175     fdps_initialize();
176     // Create and initialize dinfo object
177     int dinfo_num;
178     float coef_ema=0.3;
179     fdps_create_dinfo(&dinfo_num);
180     fdps_init_dinfo(dinfo_num,coef_ema);
181     // Create and initialize psys object
182     int psys_num;
183     fdps_create_psys(&psys_num,"full_particle");
184     fdps_init_psys(psys_num);
185     // Create and initialize tree object
186     int tree_num;
187     fdps_create_tree(&tree_num,
188                      "Long,full_particle,full_particle,full_particle,
                             Monopole");
189     int ntot=1024;
190     double theta = 0.5;
191     int n_leaf_limit = 8;
192     int n_group_limit = 64;
193     fdps_init_tree(tree_num, ntot, theta, n_leaf_limit, n_group_limit);
194     // Make an initial condition
195     setup_IC(psys_num,ntot);
196     // Domain decomposition and exchange particle
197     fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
198     fdps_exchange_particle(psys_num,dinfo_num);
199     // Compute force at the initial time
200     fdps_calc_force_all_and_write_back(tree_num,
201                                        calc_gravity_ep_ep,
202                                        calc_gravity_ep_sp,
203                                        psys_num,
204                                        dinfo_num,
```

```
205                                         true,
206                                         FDPS_MAKE_LIST);
207     //dump_particles(psys_num);
208     // Compute energies at the initial time
209     double etot0,ekin0,epot0;
210     calc_energy(psys_num, &etot0, &ekin0,&epot0);
211     printf("Energies = %21.14e  %21.14e  %21.14e\n",etot0,ekin0,epot0);
212     // Time integration
213     double time_diag = 0;
214     double time_snap = 0;
215     double time_sys  = 0;
216     double time_end = 10.0;
217     double dt = 1.0/128.0;
218     double dt_diag = 1.0;
219     double dt_snap = 1.0;
220     int    num_loop = 0;
221     while (time_sys <= time_end){
222         if (time_sys + dt/2 >= time_snap){
223             // output(psys_num)
224             time_snap += dt_snap;
225         }
226         double etot1, ekin1, epot1;
227         calc_energy(psys_num, &etot1,&ekin1,&epot1);
228         //printf( "Energies = %21.14e  %21.14e  %21.14e\n",etot1,ekin1,
                epot1);
229         //dump_particles(psys_num);
230         if (fdps_get_rank() == 0){
231             if (time_sys + dt/2 >= time_diag){
232                 printf ("time: %10.3f, energy error: %15.7e\n",
233                         time_sys, (etot1-etot0)/etot0);
234                 time_diag = time_diag + dt_diag;
235             }
236         }
237         kick(psys_num,0.5*dt);
238         time_sys +=  dt;
239         drift(psys_num,dt);
240         // Domain decomposition & exchange particle
241         if (num_loop %4  == 0) {
242             fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
243         }
244         fdps_exchange_particle(psys_num,dinfo_num);
245         // Force calculation
246         fdps_calc_force_all_and_write_back(tree_num,
247                                            calc_gravity_ep_ep,
248                                            calc_gravity_ep_sp,
249                                            psys_num,
250                                            dinfo_num,
251                                            true,
252                                            FDPS_MAKE_LIST);
253         kick(psys_num,0.5*dt);
254         num_loop += 1;
255     }
256     fdps_finalize();
257     return 0;
258 }
```

5.2 固定長 SPH シミュレーション

固定長 SPH シミュレーションのサンプルコードを以下に示す。このサンプルは第 3, 4 節で用いた固定長 SPH シミュレーションのサンプルコードと同じものである。これをカット&ペーストしてコンパイルすれば、正常に動作する固定長 SPH シミュレーションコードを作ることができる。

Listing 36: 固定長 SPH シミュレーションのサンプルコード (user_defined.h)

```
1  #pragma once
2  /* Standard headers */
3  #include <math.h>
4  /* FDPS headers */
5  #include "FDPS_c_if.h"
6  /* User-defined headers */
7  #include "mathematical_constants.h"
8
9  /* Force types */
10 typedef struct force_dens { //$fdps Force
11     //$fdps clear smth=keep
12     double dens;
13     double smth;
14 } Force_dens;
15
16 typedef struct force_hydro { //$fdps Force
17     //$fdps clear
18     fdps_f64vec acc;
19     double eng_dot;
20     double dt;
21 } Force_hydro;
22
23 /* Full particle type */
24 typedef struct full_particle { //$fdps FP
25     //$fdps copyFromForce force_dens (dens,dens)
26     //$fdps copyFromForce force_hydro (acc,acc) (eng_dot,eng_dot) (dt,dt)
27     double mass; //$fdps charge
28     fdps_f64vec pos; //$fdps position
29     fdps_f64vec vel;
30     fdps_f64vec acc;
31     double dens;
32     double eng;
33     double pres;
34     double smth; //$fdps rsearch
35     double snds;
36     double eng_dot;
37     double dt;
38     long long int id;
39     fdps_f64vec vel_half;
40     double eng_half;
41 } Full_particle;
42
43 /* Essential particle type */
44 typedef struct essential_particle { //$fdps EPI,EPJ
45     //$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,
          mass) (smth,smth) (dens,dens) (pres,pres) (snds,snds)
```

```
46      long long int id;
47      fdps_f64vec pos; //$fdps position
48      fdps_f64vec vel;
49      double mass; //$fdps charge
50      double smth; //$fdps rsearch
51      double dens;
52      double pres;
53      double snds;
54  } Essential_particle;
55
56  /* Prototype declarations */
57  double W(fdps_f64vec dr, double h);
58  fdps_f64vec gradW(fdps_f64vec dr, double h);
59
60  void calc_density(Essential_particle *ep_i,
61                    int n_ip,
62                    Essential_particle *ep_j,
63                    int n_jp,
64                    Force_dens *f);
65  void calc_hydro_force(Essential_particle *ep_i,
66                        int n_ip,
67                        Essential_particle *ep_j,
68                        int n_jp,
69                        Force_hydro *f);
70
71  /* Gloabl variable */
72  extern const double kernel_support_radius;
```

Listing 37: 固定長 SPH シミュレーションのサンプルコード (user_defined.c)

```
1   #include "user_defined.h"
2
3   /* Global variable */
4   const double kernel_support_radius=2.5;
5
6   /* Kernel functions */
7   double W(fdps_f64vec dr, double h) {
8       double s,s1,s2,ret;
9       s = sqrt(dr.x * dr.x
10              +dr.y * dr.y
11              +dr.z * dr.z)/h;
12      s1 = 1.0 - s;
13      if (s1 < 0.0) s1 = 0.0;
14      s2 = 0.5 - s;
15      if (s2 < 0.0) s2 = 0.0;
16      ret = (s1*s1*s1) - 4.0*(s2*s2*s2);
17      ret = ret * 16.0e0/(pi*h*h*h);
18      return ret;
19  }
20
21  fdps_f64vec gradW(fdps_f64vec dr, double h) {
22      double dr_abs,s,s1,s2,coef;
23      fdps_f64vec ret;
24      dr_abs = sqrt(dr.x * dr.x
25                   +dr.y * dr.y
26                   +dr.z * dr.z);
```

```
27      s = dr_abs/h;
28      s1 = 1.0 - s;
29      if (s1 < 0.0) s1 = 0.0;
30      s2 = 0.5 - s;
31      if (s2 < 0.0) s2 = 0.0;
32      coef = - 3.0*(s1*s1) + 12.0*(s2*s2);
33      coef = coef * 16.0/(pi*h*h*h);
34      coef = coef / (dr_abs*h + 1.0e-6*h);
35      ret.x = dr.x * coef;
36      ret.y = dr.y * coef;
37      ret.z = dr.z * coef;
38      return ret;
39  }
40
41  /* Interaction functions */
42  void calc_density(Essential_particle *ep_i,
43                    int n_ip,
44                    Essential_particle *ep_j,
45                    int n_jp,
46                    Force_dens *f) {
47      int i,j;
48      fdps_f64vec dr;
49      for (i = 0; i< n_ip; i++) {
50          for (j = 0; j < n_jp; j++) {
51              dr.x = ep_j[j].pos.x - ep_i[i].pos.x;
52              dr.y = ep_j[j].pos.y - ep_i[i].pos.y;
53              dr.z = ep_j[j].pos.z - ep_i[i].pos.z;
54              f[i].dens += ep_j[j].mass * W(dr,ep_i[i].smth);
55          }
56      }
57
58  }
59
60  void calc_hydro_force(Essential_particle *ep_i,
61                        int n_ip,
62                        Essential_particle *ep_j,
63                        int n_jp,
64                        Force_hydro *f) {
65      // Local parameters
66      const double C_CFL=0.3;
67      // Local variables
68      int i,j;
69      double mass_i,mass_j,smth_i,smth_j,
70             dens_i,dens_j,pres_i,pres_j,
71             snds_i,snds_j;
72      double povrho2_i,povrho2_j,
73             v_sig_max,dr_dv,w_ij,v_sig,AV;
74      fdps_f64vec pos_i,pos_j,vel_i,vel_j,
75                  dr,dv,gradW_i,gradW_j,gradW_ij;
76
77      for (i = 0; i < n_ip; i++) {
78          // Zero-clear
79          v_sig_max = 0.0;
80          // Extract i-particle info.
81          pos_i.x = ep_i[i].pos.x;
```

```
82          pos_i.y = ep_i[i].pos.y;
83          pos_i.z = ep_i[i].pos.z;
84          vel_i.x = ep_i[i].vel.x;
85          vel_i.y = ep_i[i].vel.y;
86          vel_i.z = ep_i[i].vel.z;
87          mass_i  = ep_i[i].mass;
88          smth_i  = ep_i[i].smth;
89          dens_i  = ep_i[i].dens;
90          pres_i  = ep_i[i].pres;
91          snds_i  = ep_i[i].snds;
92          povrho2_i = pres_i/(dens_i*dens_i);
93          for (j = 0; j < n_jp; j++) {
94              // Extract j-particle info.
95              pos_j.x = ep_j[j].pos.x;
96              pos_j.y = ep_j[j].pos.y;
97              pos_j.z = ep_j[j].pos.z;
98              vel_j.x = ep_j[j].vel.x;
99              vel_j.y = ep_j[j].vel.y;
100             vel_j.z = ep_j[j].vel.z;
101             mass_j  = ep_j[j].mass;
102             smth_j  = ep_j[j].smth;
103             dens_j  = ep_j[j].dens;
104             pres_j  = ep_j[j].pres;
105             snds_j  = ep_j[j].snds;
106             povrho2_j = pres_j/(dens_j*dens_j);
107             // Compute dr & dv
108             dr.x = pos_i.x - pos_j.x;
109             dr.y = pos_i.y - pos_j.y;
110             dr.z = pos_i.z - pos_j.z;
111             dv.x = vel_i.x - vel_j.x;
112             dv.y = vel_i.y - vel_j.y;
113             dv.z = vel_i.z - vel_j.z;
114             // Compute the signal velocity
115             dr_dv = dr.x * dv.x + dr.y * dv.y + dr.z * dv.z;
116             if (dr_dv < 0.0) {
117                 w_ij = dr_dv / sqrt(dr.x * dr.x + dr.y * dr.y + dr.z * dr.z
                        );
118             } else {
119                 w_ij = 0.0;
120             }
121             v_sig = snds_i + snds_j - 3.0 * w_ij;
122             if (v_sig > v_sig_max) v_sig_max=v_sig;
123             // Compute the artificial viscosity
124             AV = - 0.5*v_sig*w_ij / (0.5*(dens_i+dens_j));
125             // Compute the average of the gradients of kernel
126             gradW_i  = gradW(dr,smth_i);
127             gradW_j  = gradW(dr,smth_j);
128             gradW_ij.x = 0.5 * (gradW_i.x + gradW_j.x);
129             gradW_ij.y = 0.5 * (gradW_i.y + gradW_j.y);
130             gradW_ij.z = 0.5 * (gradW_i.z + gradW_j.z);
131             // Compute the acceleration and the heating rate
132             f[i].acc.x -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.x;
133             f[i].acc.y -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.y;
134             f[i].acc.z -= mass_j*(povrho2_i+povrho2_j+AV)*gradW_ij.z;
135             f[i].eng_dot += mass_j * (povrho2_i + 0.5*AV)
```

```
136                                 *(dv.x * gradW_ij.x
137                                  +dv.y * gradW_ij.y
138                                  +dv.z * gradW_ij.z);
139         }
140         f[i].dt = C_CFL*2.0*smth_i/(v_sig_max*kernel_support_radius);
141     }
142 }
```

Listing 38: 固定長 SPH シミュレーションのサンプルコード (c_main.c)

```
1  /* Standard headers */
2  #include <stdio.h>
3  #include <stdlib.h>
4  #include <stdbool.h>
5  #include <math.h>
6  /* FDPS headers */
7  #include "FDPS_c_if.h"
8  /* user-defined headers */
9  #include "mathematical_constants.h"
10 #include "user_defined.h"
11
12 void setup_IC(int psys_num,
13               double *end_time,
14               fdps_f32vec *pos_ll,
15               fdps_f32vec *pos_ul) {
16     // Get # of MPI processes and rank number
17     int nprocs = fdps_get_num_procs();
18     int myrank = fdps_get_rank();
19
20     // Set the box size
21     pos_ll->x = 0.0;
22     pos_ll->y = 0.0;
23     pos_ll->z = 0.0;
24     pos_ul->x = 1.0;
25     pos_ul->y = pos_ul->x / 8.0;
26     pos_ul->z = pos_ul->x / 8.0;
27
28     // Make an initial condition at RANK 0
29     if (myrank == 0) {
30         // Set the left and right states
31         const double dens_L = 1.0;
32         const double eng_L  = 2.5;
33         const double dens_R = 0.5;
34         const double eng_R  = 2.5;
35         // Set the separation of particle of the left state
36         const double dx = 1.0 / 128.0;
37         const double dy = dx;
38         const double dz = dx;
39         // Set the number of local particles
40         int nptcl_glb = 0;
41         // (1) Left-half
42         const int nx_L = 0.5*pos_ul->x/dx;
43         const int ny_L = pos_ul->y/dy;
44         const int nz_L = pos_ul->z/dz;
45         nptcl_glb += nx_L * ny_L * nz_L;
46         printf("nptcl_glb(L)␣␣␣=␣%d\n",nptcl_glb);
```

```
47          // (2) Right-half
48          const int nx_R = 0.5*pos_ul->x/((dens_L/dens_R)*dx);
49          const int ny_R = ny_L;
50          const int nz_R = nz_L;
51          nptcl_glb += nx_R * ny_R * nz_R;
52          printf("nptcl_glb(L+R) = %d\n",nptcl_glb);
53          // Place SPH particles
54          fdps_set_nptcl_loc(psys_num,nptcl_glb);
55          Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(
                psys_num);
56          int id = -1;
57          // (1) Left-half
58          int i,j,k;
59          for (i = 0; i < nx_L; i++) {
60              for (j = 0; j < ny_L; j++) {
61                  for (k = 0; k < nz_L; k++) {
62                      id++;
63                      ptcl[id].id    = id;
64                      ptcl[id].pos.x = dx * i;
65                      ptcl[id].pos.y = dy * j;
66                      ptcl[id].pos.z = dz * k;
67                      ptcl[id].dens  = dens_L;
68                      ptcl[id].eng   = eng_L;
69                  }
70              }
71          }
72          // (2) Right-half
73          for (i = 0; i < nx_R; i++) {
74              for (j = 0; j < ny_R; j++) {
75                  for (k = 0; k < nz_R; k++) {
76                      id++;
77                      ptcl[id].id    = id;
78                      ptcl[id].pos.x = 0.5*pos_ul->x + ((dens_L/dens_R)*dx)*
                            i;
79                      ptcl[id].pos.y = dy * j;
80                      ptcl[id].pos.z = dz * k;
81                      ptcl[id].dens  = dens_R;
82                      ptcl[id].eng   = eng_R;
83                  }
84              }
85          }
86          printf("nptcl(L+R) = %d\n",id+1);
87          // Set particle mass and smoothing length
88          for (i = 0; i < nptcl_glb; i++) {
89              ptcl[i].mass = 0.5*(dens_L+dens_R)
90                           * (pos_ul->x*pos_ul->y*pos_ul->z)
91                           / nptcl_glb;
92              ptcl[i].smth = kernel_support_radius * 0.012;
93          }
94      } else {
95          fdps_set_nptcl_loc(psys_num,0);
96      }
97
98      // Set the end time
99      *end_time = 0.12;
```

```
100
101     // Inform to STDOUT
102     if (fdps_get_rank() == 0) printf("setup...completed!\n");
103     //fdps_finalize();
104     //exit(0);
105
106 }
107
108 double get_timestep(int psys_num) {
109     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
110     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
111     double dt_loc = 1.0e30;
112     int i;
113     for (i = 0; i < nptcl_loc; i++)
114         if (ptcl[i].dt < dt_loc)
115             dt_loc = ptcl[i].dt;
116     return fdps_get_min_value_f64(dt_loc);
117 }
118
119 void initial_kick(int psys_num, double dt) {
120     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
121     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
122     int i;
123     for (i = 0; i < nptcl_loc; i++) {
124         ptcl[i].vel_half.x = ptcl[i].vel.x + 0.5 * dt * ptcl[i].acc.x;
125         ptcl[i].vel_half.y = ptcl[i].vel.y + 0.5 * dt * ptcl[i].acc.y;
126         ptcl[i].vel_half.z = ptcl[i].vel.z + 0.5 * dt * ptcl[i].acc.z;
127         ptcl[i].eng_half = ptcl[i].eng + 0.5 * dt * ptcl[i].eng_dot;
128     }
129 }
130
131 void full_drift(int psys_num, double dt) {
132     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
133     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
134     int i;
135     for (i = 0; i < nptcl_loc; i++) {
136         ptcl[i].pos.x += dt * ptcl[i].vel_half.x;
137         ptcl[i].pos.y += dt * ptcl[i].vel_half.y;
138         ptcl[i].pos.z += dt * ptcl[i].vel_half.z;
139     }
140 }
141
142 void predict(int psys_num, double dt) {
143     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
144     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
145     int i;
146     for (i = 0; i < nptcl_loc; i++) {
147         ptcl[i].vel.x += dt * ptcl[i].acc.x;
148         ptcl[i].vel.y += dt * ptcl[i].acc.y;
149         ptcl[i].vel.z += dt * ptcl[i].acc.z;
150         ptcl[i].eng += dt * ptcl[i].eng_dot;
151     }
152 }
153
154 void final_kick(int psys_num, double dt) {
```

```
155     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
156     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
157     int i;
158     for (i = 0; i < nptcl_loc; i++) {
159         ptcl[i].vel.x = ptcl[i].vel_half.x + 0.5 * dt * ptcl[i].acc.x;
160         ptcl[i].vel.y = ptcl[i].vel_half.y + 0.5 * dt * ptcl[i].acc.y;
161         ptcl[i].vel.z = ptcl[i].vel_half.z + 0.5 * dt * ptcl[i].acc.z;
162         ptcl[i].eng = ptcl[i].eng_half + 0.5 * dt * ptcl[i].eng_dot;
163     }
164 }
165
166 void set_pressure(int psys_num) {
167     const double hcr=1.4;
168     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
169     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
170     int i;
171     for (i = 0; i < nptcl_loc; i++) {
172         ptcl[i].pres = (hcr - 1.0) * ptcl[i].dens * ptcl[i].eng;
173         ptcl[i].snds = sqrt(hcr * ptcl[i].pres / ptcl[i].dens);
174     }
175 }
176
177 void output(int psys_num, int nstep) {
178     int myrank = fdps_get_rank();
179     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
180     Full_particle *ptcl = (Full_particle *) fdps_get_psys_cptr(psys_num);
181     char filename[64] = {'\0'};
182     sprintf(filename,"./result/snap%05d-proc%05d.txt",nstep,myrank);
183     FILE *fp;
184     if ((fp = fopen(filename,"w")) == NULL) {
185         fprintf(stderr,"Cannot␣open␣file␣%s\n",filename);
186         exit(EXIT_FAILURE);
187     }
188     int i;
189     for (i = 0; i < nptcl_loc; i++) {
190         fprintf(fp,"%ld␣%15.7e␣%15.7e␣%15.7e␣%15.7e",
191                 ptcl[i].id,ptcl[i].mass,
192                 ptcl[i].pos.x,ptcl[i].pos.y,ptcl[i].pos.z);
193         fprintf(fp,"%15.7e␣%15.7e␣%15.7e␣%15.7e␣%15.7e␣%15.7e\n",
194                 ptcl[i].vel.x,ptcl[i].vel.y,ptcl[i].vel.z,
195                 ptcl[i].dens,ptcl[i].eng,ptcl[i].pres);
196     }
197     fclose(fp);
198 }
199
200 void check_cnsrvd_vars(int psys_num){
201     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
202     Full_particle *ptcl = fdps_get_psys_cptr(psys_num);
203     fdps_f64vec mom_loc;
204     mom_loc.x = 0.0; mom_loc.y = 0.0; mom_loc.z = 0.0;
205     double eng_loc = 0.0;
206     int i;
207     for (i = 0; i < nptcl_loc; i++) {
208         mom_loc.x += ptcl[i].vel.x * ptcl[i].mass;
209         mom_loc.y += ptcl[i].vel.y * ptcl[i].mass;
```

```
210         mom_loc.z += ptcl[i].vel.z * ptcl[i].mass;
211         eng_loc += ptcl[i].mass *(ptcl[i].eng
212                                   + 0.5*(ptcl[i].vel.x * ptcl[i].vel.x
213                                         +ptcl[i].vel.y * ptcl[i].vel.y
214                                         +ptcl[i].vel.z * ptcl[i].vel.z));
215     }
216     double eng = fdps_get_sum_f64(eng_loc);
217     fdps_f64vec mom;
218     mom.x = fdps_get_sum_f64(mom_loc.x);
219     mom.y = fdps_get_sum_f64(mom_loc.y);
220     mom.z = fdps_get_sum_f64(mom_loc.z);
221     if (fdps_get_rank() == 0) {
222         printf("eng␣␣␣=␣%15.7e\n",eng);
223         printf("mom.x␣=␣%15.7e\n",mom.x);
224         printf("mom.y␣=␣%15.7e\n",mom.y);
225         printf("mom.z␣=␣%15.7e\n",mom.z);
226     }
227 }
228
229 void c_main() {
230     // Initialize some global variables
231     setup_math_const();
232
233     // Initialize FDPS
234     fdps_initialize();
235
236     // Make an instance of ParticleSystem and initialize it
237     int psys_num;
238     fdps_create_psys(&psys_num,"full_particle");
239     fdps_init_psys(psys_num);
240
241     // Make an initial condition and initialize the particle system
242     double end_time;
243     fdps_f32vec pos_ll, pos_ul;
244     setup_IC(psys_num,&end_time,&pos_ll,&pos_ul);
245
246     // Make an instance of DomainInfo and initialize it
247     int dinfo_num;
248     fdps_create_dinfo(&dinfo_num);
249     float coef_ema = 0.3;
250     fdps_init_dinfo(dinfo_num,coef_ema);
251     fdps_set_boundary_condition(dinfo_num,FDPS_BC_PERIODIC_XYZ);
252     fdps_set_pos_root_domain(dinfo_num,&pos_ll,&pos_ul);
253
254     // Perform domain decomposition and exchange particles
255     fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
256     fdps_exchange_particle(psys_num,dinfo_num);
257
258     // Make two tree structures
259     int ntot = fdps_get_nptcl_glb(psys_num);
260     // tree_dens (used for the density calculation)
261     int tree_num_dens;
262     fdps_create_tree(&tree_num_dens,
263                      "Short,force_dens,essential_particle,
                               essential_particle,Gather");
```

```
264     float theta = 0.5;
265     int n_leaf_limit = 8;
266     int n_group_limit = 64;
267     fdps_init_tree(tree_num_dens,3*ntot,theta,n_leaf_limit,n_group_limit);
268
269     // tree_hydro (used for the force calculation)
270     int tree_num_hydro;
271     fdps_create_tree(&tree_num_hydro,
272                      "Short,force_hydro,essential_particle,
                               essential_particle,Symmetry");
273     fdps_init_tree(tree_num_hydro,3*ntot,theta,n_leaf_limit,n_group_limit)
              ;
274
275     // Compute density, pressure, acceleration due to pressure gradient
276     fdps_calc_force_all_and_write_back(tree_num_dens,
277                                        calc_density,
278                                        NULL,
279                                        psys_num,
280                                        dinfo_num,
281                                        true,
282                                        FDPS_MAKE_LIST);
283     set_pressure(psys_num);
284     fdps_calc_force_all_and_write_back(tree_num_hydro,
285                                        calc_hydro_force,
286                                        NULL,
287                                        psys_num,
288                                        dinfo_num,
289                                        true,
290                                        FDPS_MAKE_LIST);
291     // Get timestep
292     double dt = get_timestep(psys_num);
293
294     // Main loop for time integration
295     int nstep = 0; double time = 0.0;
296     for (;;) {
297         // Leap frog: Initial Kick & Full Drift
298         initial_kick(psys_num,dt);
299         full_drift(psys_num,dt);
300
301         // Adjust the positions of the SPH particles that run over
302         // the computational boundaries.
303         fdps_adjust_pos_into_root_domain(psys_num,dinfo_num);
304
305         // Leap frog: Predict
306         predict(psys_num,dt);
307
308         // Perform domain decomposition and exchange particles again
309         fdps_decompose_domain_all(dinfo_num,psys_num,-1.0);
310         fdps_exchange_particle(psys_num,dinfo_num);
311
312         // Compute density, pressure, acceleration due to pressure
                gradient
313         fdps_calc_force_all_and_write_back(tree_num_dens,
314                                            calc_density,
315                                            NULL,
```

```
316                                             psys_num,
317                                             dinfo_num,
318                                             true,
319                                             FDPS_MAKE_LIST);
320         set_pressure(psys_num);
321         fdps_calc_force_all_and_write_back(tree_num_hydro,
322                                             calc_hydro_force,
323                                             NULL,
324                                             psys_num,
325                                             dinfo_num,
326                                             true,
327                                             FDPS_MAKE_LIST);
328
329         // Get a new timestep
330         dt = get_timestep(psys_num);
331
332         // Leap frog: Final Kick
333         final_kick(psys_num,dt);
334
335         // Output result files
336         int output_interval = 10;
337         if (nstep % output_interval == 0) {
338            output(psys_num,nstep);
339            check_cnsrvd_vars(psys_num);
340         }
341
342         // Output information to STDOUT
343         if (fdps_get_rank() == 0) {
344            printf("================================\n");
345            printf("time  = %15.7e\n",time);
346            printf("nstep = %d\n",nstep);
347            printf("================================\n");
348         }
349
350         // Termination condition
351         if (time >= end_time) break;
352
353         // Update time & step
354         time  += dt;
355         nstep++;
356     }
357     fdps_finalize();
358 }
```

6 拡張機能の解説

6.1 P^3M コード

本節では、FDPS の拡張機能 Particle Mesh (以下、PM と省略する) の使用方法について、P^3M(Particle-Particle-Particle-Mesh) 法のサンプルコードを用いて解説を行う。このサンプルコードでは、塩化ナトリウム (NaCl) 結晶の系全体の結晶エネルギーを P^3M 法で計算し、結果を解析解と比較する。P^3M 法では、力、及び、ポテンシャルエネルギーの計算を、Particle-Particle(PP) パートと Particle-Mesh(PM) パートに split して行われる。このサンプルコードでは PP パートを FDPS 標準機能を用いて計算し、PM パートを FDPS 拡張機能を用いて計算する。なお、拡張機能 PM の仕様の詳細は、仕様書 9.2 節で説明されているので、そちらも参照されたい。

6.1.1 サンプルコードの場所と作業ディレクトリ

サンプルコードの場所は、`$(FDPS)/sample/c/p3m` である。まずは、そこに移動する。

```
$ cd (FDPS)/sample/c/p3m
```

サンプルコードはユーザ定義型が実装された `user_defined.h`、相互作用関数が実装された `user_defined.c`、ユーザコードのそれ以外の部分が実装された `c_main.c`、及び、GCC 用の Makefile である `Makefile` から構成される。

6.1.2 ユーザー定義型

本節では、FDPS の機能を用いて P^3M 法の計算を行うにあたって、ユーザーが記述しなければならない構造体について記述する。

6.1.2.1 FullParticle 型

ユーザーは FullParticle 型を記述しなければならない。Listing 39 に、サンプルコードの FullParticle 型を示す。FullParticle 型には、計算を行うにあたって、粒子が持っているべき全ての物理量が含まれている必要がある。

Listing 39: FullParticle 型

```
1 typedef struct fp_nbody { //$fdps FP
2    //$fdps copyFromForce force_pp (pot,pot) (acc,acc)
3    //$fdps copyFromForcePM acc_pm
4    long long int id;
5    double mass; //$fdps charge
6    double rcut; //$fdps rsearch
7    fdps_f64vec pos; //$fdps position
8    fdps_f64vec acc;
9    double pot;
```

```
10     fdps_f32vec acc_pm;
11     float pot_pm;
12 } FP_nbody;
```

この 構造体が FullParticle 型であることを示すため、次の指示文を記述している:

```
typedef struct fp_nbody { //$fdps FP
```

P^3M シミュレーションにおける相互作用はカットオフを持つ長距離力である。そのため、必須物理量としてカットオフ半径が加わる。現在の FDPS の仕様では、カットオフ半径の指定は探索半径の指定 (§ 4.2 参照) と同様に行う。次の指示文は、ど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ものである:

```
double mass; //$fdps charge
double rcut; //$fdps rsearch
fdps_f64vec pos; //$fdps position
```

FullParticle 型は Force 型との間でデータコピーを行う。ユーザは指示文を使い、FDPS にデータコピーの仕方を教えなければならない。また、拡張機能 PM を用いて相互作用計算を行う場合、FullParticle 型には PM モジュールで計算した相互作用計算の結果をどのメンバ変数で受け取るのか指示する指示文を記述する必要がある。本サンプルコードでは、以下のように記述している:

```
//$fdps copyFromForce force_pp (pot,pot) (acc,acc)
//$fdps copyFromForcePM acc_pm
```

6.1.2.2 EssentialParticleI 型

ユーザーは EssentialParticleI 型を記述しなければならない。EssentialParticleI 型には、PP パートの Force 計算を行う際、i 粒子が持っているべき全ての物理量をメンバ変数として持っている必要がある。また、本チュートリアル中では、EssentialParticleJ 型も兼ねているため、j 粒子が持っているべき全ての物理量もメンバ変数として持っている必要がある。Listing 40 に、サンプルコードの EssentialParticleI 型を示す。

Listing 40: EssentialParticleI 型

```
1 typedef struct ep_nbody { //$fdps EPI,EPJ
2     //$fdps copyFromFP fp_nbody (id,id) (mass,mass) (rcut,rcut) (pos,pos)
3     long long int id;
4     double mass; //$fdps charge
5     double rcut; //$fdps rsearch
6     fdps_f64vec pos; //$fdps position
7 } EP_nbody;
```

まず、ユーザは指示文を用いて、この構造体が EssentialParticleI 型かつ EssentialParticleJ 型であることを FDPS に教えなければならない。本サンプルコードでは、以下のように記述

している:

```
typedef struct ep_nbody { //$fdps EPI,EPJ
```

次に、ユーザはこの構造体のどのメンバ変数がどの必須物理量に対応するのかを指示文によって指定しなければならない。今回は相互作用がカットオフ有りの長距離力であるため、カットオフ半径の指定も必要である。本サンプルコードでは、以下のように記述している:

```
double mass; //$fdps charge
double rcut; //$fdps rsearch
fdps_f64vec pos; //$fdps position
```

EssentialParticleI 型と EssentialParticleJ 型は FullParticle 型からデータを受け取る。ユーザは FullParticle 型のどのメンバ変数を EssentialParticle?型 (?=I,J) のどのメンバ変数にコピーするのかを、指示文を用いて指定する必要がある。本サンプルコードでは、以下のように記述している:

```
//$fdps copyFromFP fp_nbody (id,id) (mass,mass) (rcut,rcut) (pos,pos)
```

6.1.2.3 Force 型

ユーザーは Force 型を記述しなければならない。Force 型は、PP パートの Force の計算を行った際にその結果として得られる全ての物理量をメンバ変数として持っている必要がある。本サンプルコードの Force 型を Listing 41 に示す。このサンプルコードでは、Force は Coulomb 相互作用計算のみであるため、Force 型が 1 つ用意されている。

Listing 41: Force 型

```
1 typedef struct force_pp { //$fdps Force
2    //$fdps clear
3    double pot;
4    fdps_f64vec acc;
5 } Force_pp;
```

まず、ユーザはこの構造体が Force 型であることを指示文によって指定する必要がある。本サンプルコードでは、以下のように記述している:

```
typedef struct force_pp { //$fdps Force
```

この構造体は Force 型であるから、ユーザは**必ず**、相互作用計算における積算対象のメンバ変数の初期化方法を指定する必要がある。本サンプルコードでは Force 型のすべてのメンバ変数にデフォルトの初期化方法を指定するため、単に、キーワード `clear` の指示文を記述している:

```
//$fdps clear
```

6.1.2.4 相互作用関数 calcForceEpEp

ユーザーは相互作用関数 calcForceEpEp を記述しなければならない。calcForceEpEp には、PP パートの Force の計算の具体的な内容を書く必要がある。calcForceEpEp は、`void` 関数として実装されなければならない。引数は、EssentialParticleI の配列、EssentialParticleI の個数、EssentialParticleJ の配列、EssentialParticleJ の個数、Force 型の配列である。本サンプルコードの calcForceEpEp の実装を Listing 42 に示す。

Listing 42: 相互作用関数 calcForceEpEp

```
1  void calc_force_ep_ep(EP_nbody *ep_i,
2                        int n_ip,
3                        EP_nbody *ep_j,
4                        int n_jp,
5                        Force_pp *f) {
6      int i,j;
7      for (i = 0; i < n_ip; i++) {
8          for (j = 0; j < n_jp; j++) {
9              fdps_f64vec dr;
10             dr.x = ep_i[i].pos.x - ep_j[j].pos.x;
11             dr.y = ep_i[i].pos.y - ep_j[j].pos.y;
12             dr.z = ep_i[i].pos.z - ep_j[j].pos.z;
13             double rij  = sqrt(dr.x * dr.x
14                                +dr.y * dr.y
15                                +dr.z * dr.z);
16             if ((ep_i[i].id == ep_j[j].id) && (rij == 0.0)) continue;
17             double rinv = 1.0/rij;
18             double rinv3 = rinv*rinv*rinv;
19             double xi = 2.0*rij/ep_i[i].rcut;
20             f[i].pot   += ep_j[j].mass * S2_pcut(xi) * rinv;
21             f[i].acc.x += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.x;
22             f[i].acc.y += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.y;
23             f[i].acc.z += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.z;
24         }
25         // Self-interaction term
26         f[i].pot -= ep_i[i].mass * (208.0/(70.0*ep_i[i].rcut));
27     }
28 }
```

P^{3}M 法の PP パートは、(距離に関する) カットオフ付きの 2 体相互作用である。そのため、ポテンシャルと加速度の計算にカットオフ関数 (`S2_pcut()`, `S2_fcut()`) が含まれていることに注意されたい。ここで、各カットオフ関数は、粒子の形状関数 $S(r)$ が $S2(r)$ のときのカットオフ関数である必要がある。ここで、$S2(r)$ は Hockney & Eastwood (1988) の式 (8.3) で定義される形状関数で、以下の形を持つ:

$$S2(r) = \begin{cases} \dfrac{48}{\pi a^4}\left(\dfrac{a}{2} - r\right) & r < a/2, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

ここで、r は粒子からの距離、a は形状関数のスケール長である。粒子の電荷量を q とすれば、この粒子が作る電荷密度分布 $\rho(r)$ は $\rho(r) = q\,S2(r)$ と表現される。これは r に関して線形な密度分布を仮定していることを意味する。PP パートのカットオフ関数が $S2(r)$ を仮定

したものでなければならない理由は、PM パートのカットオフ関数が S2 型形状関数を仮定して実装されているためである (PM パートと PP パートのカットオフ関数は同じ形状関数に基づく必要がある)。

カットオフ関数はユーザが定義する必要がある。サンプルコードの冒頭に `S2_pcut()` と `S2_fcut()` の実装例がある。これらの関数では、Hockney & Eastwood (1988) の式 (8-72),(8-75) が使用されている。カットオフ関数は、PP 相互作用が以下の形となるように定義されている:

$$\Phi_{\rm PP}(\boldsymbol{r}) = \frac{m}{|\boldsymbol{r}-\boldsymbol{r}'|}\texttt{S2\_pcut}(\xi) \tag{2}$$

$$\boldsymbol{f}_{\rm PP}(\boldsymbol{r}) = \frac{m(\boldsymbol{r}-\boldsymbol{r}')}{|\boldsymbol{r}-\boldsymbol{r}'|^3}\texttt{S2\_fcut}(\xi) \tag{3}$$

ここで、$\xi = 2|\boldsymbol{r}-\boldsymbol{r}'|/a$ である。本サンプルコードでは a を変数 `rcut` で表現している。

Hockney & Eastwood (1988) の式 (8-75) を見ると、$r = 0$ のとき、メッシュポテンシャル ϕ^m が次の有限値を持つことがわかる (ここで、$1/4\pi\varepsilon_0$ の因子は省略した):

$$\phi^m(0) = \frac{208}{70a} \tag{4}$$

この項はサンプルコードの i 粒子のループの最後で考慮されている:

```
1 f[i].pot -= ep_i[i].mass * (208.0/(70.0*ep_i[i].rcut));
```

この項を考慮しないと解析解と一致しないことに注意する必要がある。

6.1.2.5 相互作用関数 calcForceEpSp

ユーザーは相互作用関数 calcForceEpSp を記述しなければならない[注 2)]。calcForceEpSp には、粒子-超粒子間相互作用計算の具体的な内容を書く必要がある。calcForceEpSp は、`void` 関数 として実装されなければならない。引数は、EssentialParticleI の配列、EssentialParticleI の個数、超粒子型の配列、超粒子型の個数、Force 型の配列である。本サンプルコードの calcForceEpSp の実装を Listing 43 に示す。

Listing 43: 相互作用関数 calcForceEpSp

```
1 void calc_force_ep_sp(EP_nbody *ep_i,
2                        int n_ip,
3                        fdps_spj_monopole_cutoff *ep_j,
4                        int n_jp,
5                        Force_pp *f) {
6    int i,j;
7    for (i = 0; i < n_ip; i++) {
8       for (j = 0; j < n_jp; j++) {
9          fdps_f64vec dr;
```

注 2) 冒頭で述べたように、本サンプルコードでは相互作用計算に $\mathrm{P^3M}$ 法を用いる。FDPS の枠組み内で、これを実現するため、後述するように、見込み角の基準値 θ を 0 に指定して相互作用計算を行う。このため、粒子-超粒子相互作用は発生しないはずである。しかしながら、API `fdps_calc_force_all_and_write_back` に、粒子-超粒子間相互作用を計算する関数のアドレスを渡す必要があるため、相互作用関数 calcForceEpSp を定義する必要がある。

```
10          dr.x = ep_i[i].pos.x - ep_j[j].pos.x;
11          dr.y = ep_i[i].pos.y - ep_j[j].pos.y;
12          dr.z = ep_i[i].pos.z - ep_j[j].pos.z;
13          double rij = sqrt(dr.x * dr.x
14                            +dr.y * dr.y
15                            +dr.z * dr.z);
16          double rinv = 1.0/rij;
17          double rinv3 = rinv*rinv*rinv;
18          double xi = 2.0*rij/ep_i[i].rcut;
19          f[i].pot   += ep_j[j].mass * S2_pcut(xi) * rinv;
20          f[i].acc.x += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.x;
21          f[i].acc.y += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.y;
22          f[i].acc.z += ep_j[j].mass * S2_fcut(xi) * rinv3 * dr.z;
23      }
24   }
25 }
```

6.1.3 プログラム本体

本節では、サンプルコード本体について解説を行う。詳細な説明に入る前に、サンプルコードの内容と全体構造について説明を与える。6.1 節で述べたように、このサンプルコードでは NaCl 結晶の結晶エネルギーを P^3M 法によって計算し解析解と比較する。NaCl 結晶は一様格子状に並んだ粒子として表現される。Na と Cl は互い違いに並んでおり、Na に対応する粒子は正の電荷を、Cl に対応する粒子は負の電荷を持っている。この粒子で表現された結晶を、大きさが $[0,1)^3$ の周期境界ボックスの中に配置し、結晶エネルギーを計算する。結晶エネルギーの計算精度は周期境界ボックスの中の粒子数や粒子の配置に依存するはずなので、サンプルコードでは、これらを変えてエネルギーの相対誤差を調べ、結果をファイルに出力する内容となっている。

コードの全体構造は以下のようになっている:

(1) FDPS で使用するオブジェクトの生成と初期化
(2) 指定された粒子数と配置の結晶を生成 (メイン関数の `setup_NaCl_crystal()`)
(3) 各粒子のポテンシャルエネルギーを P^3M 法で計算 (メイン関数内)
(4) 系全体のエネルギーを計算し、解析解と比較 (メイン関数の `calc_energy_error()`)
(5) (2)〜(4) を繰り返す

以下で、個々について詳しく説明を行う。

6.1.3.1 ヘッダーファイルのインクルード

FDPS の標準機能を利用できるようにするため、`FDPS_c_if.h` をインクルードする。

Listing 44: ヘッダーファイル `FDPS_c_if.h` のインクルード

```
1 #include "FDPS_c_if.h"
```

6.1.3.2 開始、終了

まずは、FDPS の初期化/開始を行う必要がある。次のように、メイン関数に記述する。

Listing 45: FDPS の開始

```
1 fdps_initialize();
```

FDPS は、開始したら明示的に終了させる必要がある。今回は、プログラムの終了と同時に FDPS も終了させるため、メイン関数の最後に次のように記述する。

Listing 46: FDPS の終了

```
1 fdps_finalize();
```

6.1.3.3 オブジェクトの生成と初期化

FDPS の初期化に成功した場合、ユーザーはコード中で用いるオブジェクトを作成する必要がある。本節では、オブジェクトの生成/初期化の仕方について、解説する。

6.1.3.3.1 オブジェクトの生成

P^3M 法の計算では、粒子群クラス、領域クラスに加え、PP パートの計算用の tree を 1 本、さらに PM パートの計算に必要な ParticleMesh オブジェクトの生成が必要である。

Listing 47: オブジェクトの生成

```
1 fdps_create_psys(&psys_num,"fp_nbody");
2 fdps_create_dinfo(&dinfo_num);
3 fdps_create_pm(&pm_num);
4 fdps_create_tree(&tree_num,
5                  "Long,force_pp,ep_nbody,ep_nbody,MonopoleWithCutoff");
```

ここに示したコードは実際にサンプルコードから該当箇所だけを取り出したものであることに注意して頂きたい。

6.1.3.3.2 オブジェクトの初期化

ユーザーはオブジェクトを生成したら、そのオブジェクトを使用する前に、初期化を行う必要がある。以下で、各オブジェクトの初期化の仕方について解説を行う。

(i) 粒子群オブジェクトの初期化 粒子群オブジェクトの初期化は、以下のように行う:

Listing 48: 粒子群オブジェクトの初期化

```
1 fdps_init_psys(psys_num);
```

サンプルコードではメイン関数の冒頭で呼び出されている。

(ii) 領域オブジェクトの初期化 領域オブジェクトの初期化は、以下のように行う:

Listing 49: 領域オブジェクトの初期化

```
1 fdps_init_dinfo(dinfo_num,coef_ema);
```

サンプルコードではメイン関数の冒頭で呼び出されている。

初期化が完了した後、領域オブジェクトには境界条件と境界の大きさをセットする必要がある。サンプルコードでは、この作業は粒子分布を決定する void 関数 setup_NaCl_crystal の中で行われている:

```
1 fdps_set_boundary_condition(dinfo_num,FDPS_BC_PERIODIC_XYZ);
2 fdps_f32vec pos_ll, pos_ul;
3 pos_ll.x = 0.0; pos_ll.y = 0.0; pos_ll.z = 0.0;
4 pos_ul.x = 1.0; pos_ul.y = 1.0; pos_ul.z = 1.0;
5 fdps_set_pos_root_domain(dinfo_num,&pos_ll,&pos_ul);
```

(iii) ツリーオブジェクトの初期化 相互作用ツリーオブジェクトの初期化も、API fdps_init_tree を使って、以下のように行う:

Listing 50: ツリーオブジェクトの初期化

```
1 fdps_init_tree(tree_num,3*nptcl_loc,theta,
2                n_leaf_limit,n_group_limit);
```

ツリーオブジェクトの API fdps_init_tree には引数として、大雑把な粒子数を渡す必要がある。これは API の第 2 引数として指定する。上記の例では、API が呼ばれた時点でのローカル粒子数の 3 倍の値がセットされるようになっている。一方、API の第 3 引数は 、tree 法で力を計算するときの opening angle criterion θ を指定する。本サンプルコードでは PP パートの計算で粒子-超粒子相互作用を発生させないようにするため、$\theta = 0$ を指定している。

(iv) ParticleMesh オブジェクトの初期化 特に明示的に初期化を行う必要はない。

6.1.3.4 粒子分布の生成

本節では、粒子分布を生成する void 関数 setup_NaCl_crystal の動作とその中で呼ばれている FDPS の API について解説を行う。この void 関数は、周期境界ボックスの 1 次元あたりの粒子数と、原点 $(0, 0, 0)$ に最も近い粒子の座標を引数として、3 次元粒子分布を生成する。サンプルコードでは、これらのパラメータは構造体 crystal_parameters のオブジェクト NaCl_params を使って渡されている:

```
1 // In user_defined.h
2 typedef struct crystal_parameters {
3    int nptcl_per_side;
4    fdps_f64vec pos_vertex;
5 } Crystal_parameters;
6 // In c_main.c
7 Crystal_parameters NaCl_params;
8 setup_NaCl_crystal(psys_num, dinfo_num, NaCl_params);
```

setup_NaCl_crystal の前半部分において、渡されたパラメータを使って粒子分布を生成している。この結晶の系全体のエネルギーは

$$E = -\frac{N\alpha m^2}{R_0} \tag{5}$$

と解析的に書ける。ここで、N は分子の総数 (原子の数は $2N$ 個)、m は粒子の電荷量、R_0 は最隣接原子間距離、α はマーデリング (Madelung) 定数である。NaCl 結晶の場合、$\alpha \approx 1.747565$ である (例えば、キッテル著 「固体物理学入門 (第 8 版)」を参照せよ)。計算結果をこの解析解と比較するとき、系全体のエネルギーが粒子数に依存しては不便である。そこで、サンプルコードでは、系全体のエネルギーが粒子数に依存しないように、粒子の電荷量 m を

$$\frac{2Nm^2}{R_0} = 1 \tag{6}$$

となるようにスケールしていることに注意されたい。

粒子分布の生成後、FDPS の API を使って、領域分割と粒子交換を行っている。以下で、これらの API について解説する。

6.1.3.4.1 領域分割の実行

粒子分布に基いて領域分割を実行するには、領域オブジェクトの API fdps_decompose_domain_all を使用する:

Listing 51: 領域分割の実行

```
1 fdps_decompose_domain_all(dinfo_num,psys_num);
```

6.1.3.4.2 粒子交換の実行

領域情報に基いてプロセス間の粒子の情報を交換するには、粒子群オブジェクトの API fdps_exchange_particle を使用する:

Listing 52: 粒子交換の実行

```
1 fdps_exchange_particle(psys_num,dinfo_num);
```

6.1.3.5 相互作用計算の実行

粒子分布を決定し、領域分割・粒子交換が終了したら、相互作用の計算を行う。サンプルコードでは、この作業をメイン関数で行っている:

Listing 53: 相互作用計算の実行

```
1 // [4] Compute force and potential with P^{3}M method
2 // [4-1] Get the pointer to FP and # of local particles
3 int nptcl_loc = fdps_get_nptcl_loc(psys_num);
```

```
 4 FP_nbody *ptcl = (FP_nbody *) fdps_get_psys_cptr(psys_num);
 5 // [4-2] PP part
 6 fdps_calc_force_all_and_write_back(tree_num,
 7                                     calc_force_ep_ep,
 8                                     calc_force_ep_sp,
 9                                     psys_num,
10                                     dinfo_num,
11                                     true,
12                                     FDPS_MAKE_LIST);
13 // [4-3] PM part
14 fdps_calc_pm_force_all_and_write_back(pm_num,
15                                        psys_num,
16                                        dinfo_num);
17 int i;
18 for (i = 0; i < nptcl_loc; i++) {
19    fdps_f32vec pos32;
20    pos32.x = ptcl[i].pos.x;
21    pos32.y = ptcl[i].pos.y;
22    pos32.z = ptcl[i].pos.z;
23    fdps_get_pm_potential(pm_num,&pos32,&ptcl[i].pot_pm);
24 }
25 // [4-4] Compute the total acceleration and potential
26 for (i = 0; i < nptcl_loc; i++) {
27    ptcl[i].pot -= ptcl[i].pot_pm;
28    ptcl[i].acc.x -= ptcl[i].acc_pm.x;
29    ptcl[i].acc.y -= ptcl[i].acc_pm.y;
30    ptcl[i].acc.z -= ptcl[i].acc_pm.z;
31 }
```

PP パートの相互作用計算には API `fdps_calc_force_all_and_write_back` 、PM パートの相互作用計算には API `fdps_calc_pm_force_all_and_write_back` を用いている。PM パートの計算の後に、加速度とポテンシャルの総和を計算している。この総和計算を**引き算で**行っていることに注意して頂きたい。引き算を使用する理由は、FDPS の拡張機能 PM は重力を想定してポテンシャルを計算するからである。すなわち、拡張機能 PM では、電荷 $m(>0)$ は正のポテンシャルを作るべきところを、質量 $m>0$ の重力ポテンシャルとして計算する。このポテンシャルは負値である。したがって、拡張機能 PM を Coulomb 相互作用で使用するには**符号反転が必要**となる。

6.1.3.6 エネルギー相対誤差の計算

エネルギーの相対誤差の計算は関数 `calc_energy_error` で行っている。ここでは、解析解の値としては $E_0 \equiv 2E = -1.7475645946332$ を採用した。これは、PM^3(Particle-Mesh Multipole Method) で数値的に求めた値である。

6.1.4 コンパイル

本サンプルコードでは FFTW ライブラリ (http://www.fftw.org) を使用するため、ユーザ環境に FFTW3 をインストールする必要がある。コンパイルは、付属の Makefile 内の変

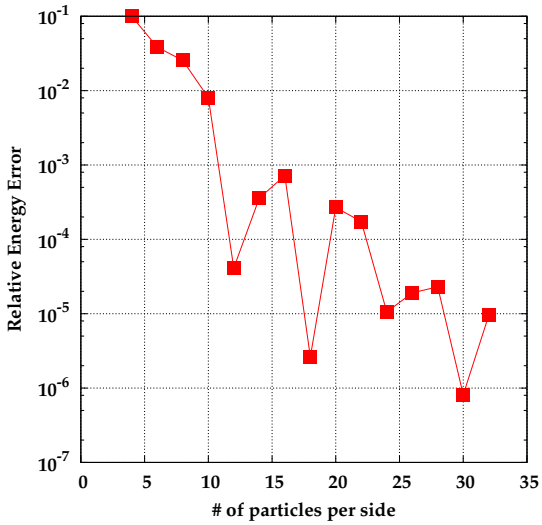

図 3: 1 辺あたりの粒子数とエネルギー相対誤差の関係 (メッシュ数は 16^3、カットオフ半径は 3/16)

数 FFTW_LOC と FDPS_LOC に、FFTW と FDPS のインストール先の PATH をそれぞれ指定し、make コマンドを実行すればよい。

```
$ make
```

コンパイルがうまく行けば、work ディレクトリに実行ファイル p3m.x が作成されているはずである。

6.1.5 実行

FDPS 拡張機能の仕様から、本サンプルコードはプロセス数が 2 以上の MPI 実行でなければ、正常に動作しない。そこで、以下のコマンドでプログラムを実行する:

```
$ MPIRUN -np NPROC ./p3m.x
```

ここで、“MPIRUN” には mpirun や mpiexec などの mpi 実行プログラムが、“NPROC” にはプロセス数が入る。

6.1.6 結果の確認

計算が終了すると、work フォルダ下にエネルギーの相対誤差を記録したファイルが出力される。結果をプロットすると、図 3 のようになる。

7 より実用的なアプリケーションの解説

前節までは、比較的単純なサンプルコードを用いて、FDPS の基本的な機能について解説を行ってきた。しかしながら、実際の研究では、複数の粒子種の取り扱いが必要になる等、より複雑なアプリケーションを書く必要がある。そこで、本節では、より実用的なサンプルコードを使って、FDPS の他の機能について解説を行う。記述を簡潔にするため、読者は前節までの内容を理解しているものと仮定する。

7.1 N 体/SPH コード

本節では、より実用的なアプリケーションの一例として円盤銀河の時間発展を計算する N 体/SPH サンプルコードの解説を行う。このサンプルコードは、重力のみで相互作用するダークマターと星を N 体粒子、重力及び流体相互作用を行うガスを SPH 粒子として表現する。重力計算は Tree 法で、流体計算は SPH 法を用いて行う。SPH 法は Springel & Hernquist [2002, MNRAS, 333, 649] 及び Springel [2005, MNRAS, 364, 1105] で提案された方法 (以下、簡単のため、Springel の方法と呼ぶ) を使用している。本解説を読むことにより、ユーザは、FDPS で複数の粒子種を扱う方法を学ぶことができる。

以下では、まずはじめにコードの使用方法について説明を行う。次に Springel の方法について簡単な解説を行った後、サンプルコードの中身について具体的に解説する。

7.1.1 コードの使用方法

前節で述べた通り、本コードは円盤銀河の N 体/SPH シミュレーションを行うコードである。ダークマターと星の初期条件は、銀河の初期条件を作成するソフトウェア MAGI (Miki & Umemura [2018, MNRAS, 475, 2269]) で作成したファイルを読み込んで設定する。一方、ガスの初期条件は本コードの内部で作成する。したがって、以下の手順で本コードを使用する。

- ディレクトリ `$(FDPS)/sample/c/nbody+sph` に移動
- カレントディレクトリにある `Makefile` を編集
- MAGI を使って初期条件に対応する粒子データを生成し、`./magi_data/dat` 以下に配置
- コマンドライン上で `make` を実行
- `nbodysph.out` ファイルの実行
- 結果の解析

以下、順に説明していく。

7.1.1.1 ディレクトリ移動

サンプルコードの場所は、$(FDPS)/sample/c/nbody+sph である。まずは、そこに移動する。

7.1.1.2 サンプルコードのファイル構成

以下はサンプルコードのファイル構成である。

```
$ ls | awk '{print $0}'
Makefile
Makefile.ofp
c_main.c
ic.c
ic.h
job.ofp.sh
leapfrog.c
leapfrog.h
macro_defs.h
magi_data/
mathematical_constants.c
mathematical_constants.h
physical_constants.c
physical_constants.h
tipsy_file_reader.cpp
tipsy_file_reader.h
user_defined.c
user_defined.h
```

各ソースファイルの内容について簡単に解説を行う。まず、c_main.c はアプリケーションのメイン関数が実装されている。ic.*には初期条件を作成する関数が実装されている。初期条件は円盤銀河の他、複数用意されている (後述)。leapfrog.*には粒子の軌道の時間積分を Leapfrog 法を用いて行う関数が実装されている。macro_defs.h には計算を制御するためのマクロが定義されている。mathematical_constants.*には数学定数が、physical_constants.*には物理定数が定義されている。tipsy_file_reader.*には MAGI が出力した粒子データを読むための関数が定義されている。user_defined.*には、ユーザ定義型や相互作用関数が実装されている。

ディレクトリ magi_data には、銀河の初期条件を作成するソフトウェア MAGI に入力するパラメータファイル (magi_data/cfg/*) と MAGI を動作させるスクリプト (magi_data/sh/run.sh) が格納されている。

7.1.1.3 Makefile の編集

Makefile の編集項目は以下の通りである。

- 変数 CXX に使用する C++コンパイラを代入する。
- 変数 CC に使用する C コンパイラを代入する。
- 変数 CXXFLAGS に C++コンパイラのコンパイルオプションを指定する。
- 変数 CFLAGS に C コンパイラのコンパイルオプションを指定する。
- 本コードでは計算を制御するためにいくつかのマクロを用意している。表 1 にマクロ名とその定義の対応を示した。また、本コードには、INITIAL_CONDITION の値に応じて自動的に設定されて使用されるマクロも存在する。これらは一般に変更する必要はないが、詳細は macro_defs.h を参照して頂きたい。
- 本コードでは重力計算に x86 版 Phantom-GRAPE ライブラリを使用することができる。Phantom-GRAPE ライブラリを使用する場合、Makefile の変数 use_phantom_grape_x86 の値を yes にする。

OpenMP や MPI の使用/不使用の指定に関しては、第 3 節を参照して頂きたい。

マクロ名	定義
INITIAL_CONDITION	初期条件の種類の指定、或いは、コードの動作の指定に使用されるマクロ。0 から 3 までのいずれかの値を取る必要がある。値に応じて、次のように指定される。 0:円盤銀河の初期条件を選択、1:Cold collapse 問題の初期条件を選択、2:Evrard test 問題の初期条件を選択、3:ガラス状に分布した SPH 粒子データを生成するモードで実行ファイルを作成
ENABLE_VARIABLE_SMOOTHING_LENGTH	smoothing length が可変/固定を制御するマクロ。定義されている場合、可変となり Springel の方法で SPH 計算が行われる。未定義の場合、固定長カーネルの SPH コードとなる。
USE_ENTROPY	流体の熱力学状態を記述する独立変数としてエントロピーを使うか単位質量あたりの内部エネルギーを使用するかを指定するマクロ。定義されている場合エントロピーが用いられる。但し、後述するマクロ ISOTHERMAL_EOS が定義されている場合には、単位質量あたりの内部エネルギーが強制的に使用される (圧力の計算に内部エネルギーを使用する)。
USE_BALSARA_SWITCH	Balsara switch (Balsara [1995, JCP, 121, 357]) の使用/不使用を制御するマクロ。定義されている場合は使用する。
USE_PRESCR_OF_THOMAS_COUCHMAN_1992	Thomas & Couchman [1992, MNRAS,257, 11] で提案された SPH 計算の tensile 不安定を防ぐ簡便な方法を使用するかを制御するマクロ。定義されている場合は使用する。
ISOTHERMAL_EOS	流体を等温で取り扱うかどうかを指定するマクロ。定義されている場合は等温で扱い、未定義の場合にはエントロピー方程式、或いは、内部エネルギー方程式が解いて、熱力学的状態を時間発展させる。
READ_DATA_WITH_BYTESWAP	MAGI の粒子データを読み込む際に基本データ型単位で byteswap してデータを読み込むかを制御するマクロ。定義されている場合は byteswap する。

表 1: コンパイル時マクロの種類と定義

7.1.1.4 MAGI を使った粒子データの生成

前述した通り、ユーザは事前に銀河の初期条件を作成するソフトウェア MAGI を使い、以下に指定する手順でデータを作成する必要がある。MAGI を利用できないユーザは、指定するサイトからこちらが用意したデータをダウンロードすることも可能である。以下、各場合について詳しく述べる。

MAGI を使ってデータ作成を行う場合 以下の手順でデータ作成を行う。

1. https://bitbucket.org/ymiki/magi から MAGI をダウンロードし、Web の “How to compile MAGI” に記載された手順に従って、適当な場所にインストールする。但し、本サンプルコードは TIPSY ファイルの粒子データ読み込みしかサポートしていないため、MAGI は `USE_TIPSY_FORMAT=ON` の状態でビルドされている必要がある。
2. `./magi_data/sh/run.sh` を開き、変数 `MAGI_INSTALL_DIR` にコマンド `magi` がインストールされたディレクトリを、変数 `NTOT` に希望する N 体粒子の総数をセットする (ダークマターと星への振り分けは MAGI が自動的に行う)。
3. `./magi_data/cfg/*`を編集し、ダークマターと銀河のモデルを指定する。指定方法の詳細は上記サイトか、或いは、Miki & Umemura [2018, MNRAS, 475, 2269] の第 2.4 節を参照のこと。デフォルトの銀河モデル (以下、**デフォルトモデル**) は次の 4 成分から構成される:
   (i) ダークマターハロー (NFW profile, $M = 10^{12}$ $\mathrm{M}_\odot$, $r_s = 21.5$ kpc, $r_c = 200$ kpc, $\Delta_c = 10$ kpc)
   (ii) バルジ (King モデル, $M = 5 \times 10^{10}$ $\mathrm{M}_\odot$, $r_s = 0.7$ kpc, $W_0 = 5$)
   (iii) thick disk (Sérsic profile, $M = 2.5 \times 10^{10}$ $\mathrm{M}_\odot$, $r_s = 3.5$ kpc, $n = 1.5$, $z_d = 1$ kpc, $Q_{T,\mathrm{min}} = 1.0$)
   (iv) thin disk (exponential disk, $M = 2.5 \times 10^{10}$ $\mathrm{M}_\odot$, $r_s = 3.5$ kpc, $z_d = 0.5$ kpc, $Q_{T,\mathrm{min}} = 1.0$)

   デフォルトモデルでは、2 つの星円盤は bar モードに対してわずかに不安定であるため、弱い棒状構造を持つ渦巻き銀河になることが期待される初期条件となっている。MAGI の最新のリリース (version 1.1.1 [2019 年 7 月 19 日時点]) では、従来のリリースとデフォルトの動作モードが変更されたため、thick disk と thin disk のパラメータ指定の仕方が FDPS 5.0d 以前から変わっていることに注意されたい。具体的には、従来の MAGI では disk 成分の速度分散をパラメータ f を通して指定する形になっていたが (FDPS 5.0d 以前に付属するサンプルコードでは、$f = 0.125$)、最新のリリースでは Toomore Q value の最小値 $Q_{T,\mathrm{min}}$ を通して指定する方式になっている。
4. ディレクトリ `magi_data` に移動し、以下のコマンドを実行:

```
$ ./sh/run.sh
```

5. MAGI が正しく終了しているなら、magi_data/dat 以下に、拡張子が tipsy の粒子データが生成されているはずである。

データをダウンロードする場合 以下のサイトからダウンロードし、./magi_data/dat/以下に置く。各粒子データの銀河モデルはすべてデフォルトモデルで、粒子数だけ異なる。

- $N = 2^{21}$: http://particle.riken.jp/~fdps/magi_data/Galaxy/21/Galaxy.tipsy
- $N = 2^{22}$: http://particle.riken.jp/~fdps/magi_data/Galaxy/22/Galaxy.tipsy
- $N = 2^{23}$: http://particle.riken.jp/~fdps/magi_data/Galaxy/23/Galaxy.tipsy
- $N = 2^{24}$: http://particle.riken.jp/~fdps/magi_data/Galaxy/24/Galaxy.tipsy

7.1.1.5 make の実行

make コマンドを実行する。

7.1.1.6 実行

実行方法は以下の通りである。

- MPI を使用しない場合、コマンドライン上で以下のコマンドを実行する

```
$ ./nbodysph.out
```

- MPI を使用する場合、コマンドライン上で以下のコマンドを実行する

```
$ MPIRUN -np NPROC ./nbodysph.out
```

ここで、MPIRUN には mpirun や mpiexec などが、NPROC には使用する MPI プロセスの数が入る。

7.1.1.7 結果の解析

ディレクトリ result に N 体粒子と SPH 粒子の粒子データファイル “nbody0000x-proc0000y.dat” と “sph0000x-proc0000y.dat” が出力される。ここで x は時刻に対応する整数、y は MPI のプロセス番号 (ランク番号) を表す。 N 体粒子データの出力フォーマットは、1 列目から順に粒子の ID, 粒子の質量、位置の x, y, z 座標、粒子の x, y, z 軸方向の速度となっている。一方、SPH 粒子データの出力フォーマットは、1 列目から順に粒子の ID, 粒子の質量、位置の x, y, z 座標、粒子の x, y, z 軸方向の速度、密度、単位質量あたりの内部エネルギー、エントロピー、圧力となっている。

図 4 は、N 体粒子数 2^{21}、SPH 粒子数 2^{18} で円盤銀河のシミュレーションを行ったときの $T = 0.46$ における星分布とガス分布である。

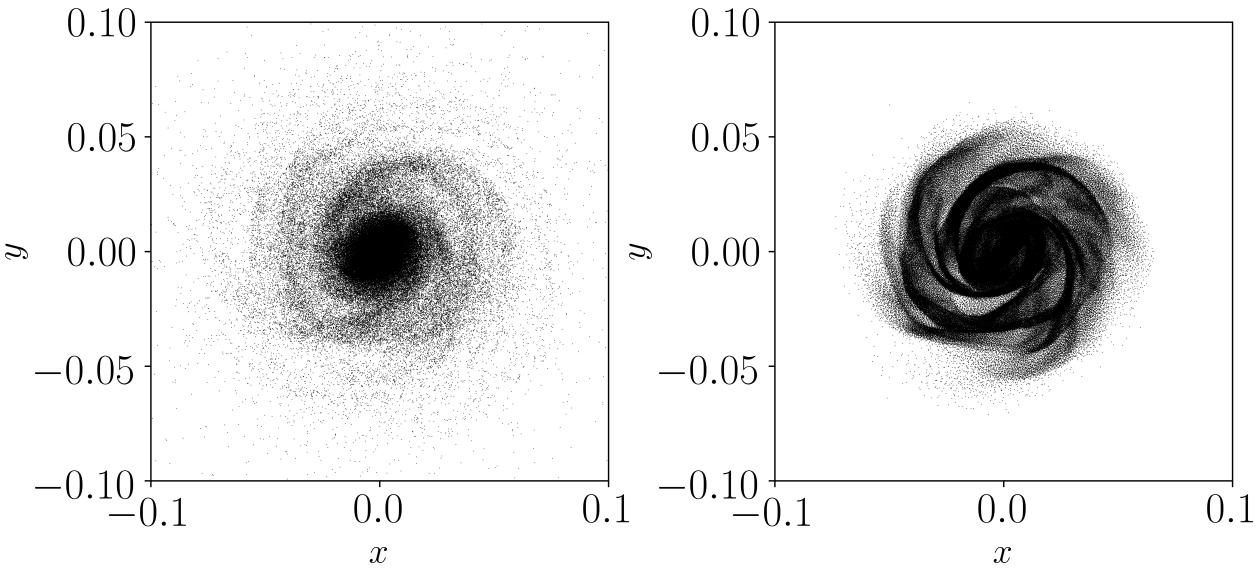

図 4: $T = 0.46$ における星分布 (左) とガス分布 (右) (計算は以下の条件で実施した: N 体粒子数 2^{21}、SPH 粒子数 2^{18}、等温、ガス温度 10^4 K、平均分子量 $\mu = 0.5$)

以下では、まず Springel の方法について解説し、その後、サンプルコードの実装について説明していく。

7.1.2 Springel の方法

Springel & Hernquist [2002, MNRAS, 333, 649] では、smoothing length が可変な場合でも、系のエネルギーとエントロピーが保存するようなスキーム (具体的には運動方程式) を定式化した。以下、彼らの定式化を手短に説明する。導出方針としては、smoothing length も独立変数とみて系の Lagrangian を立て、それを粒子数個の拘束条件の下、Euler-Lagrange 方程式を解く、というものである。

具体的には、彼らは系の Lagrangian として次のようなもの選んだ:

$$L(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \frac{1}{2}\sum_{i=1}^{N} m_i \dot{\boldsymbol{r}}_i^2 - \frac{1}{\gamma - 1}\sum_{i=1}^{N} m_i A_i \rho_i^{\gamma-1} \tag{7}$$

ここで、$\boldsymbol{q} = (\boldsymbol{r}_1, ..., \boldsymbol{r}_N, h_1, ...h_N)$ であり、下付きの整数はすべて粒子番号を表す。$\boldsymbol{r}_i$ は位置、h_i は smoothing length、m_i は質量、γ は比熱比、ρ_i は密度、A_i はエントロピー関数と呼ばれ、単位質量あたりの内部エネルギー u_i と次の関係がある:

$$u_i = \frac{A_i}{\gamma - 1}\rho_i^{\gamma-1} \tag{8}$$

式 (7) の第 1 項目は運動エネルギー、第 2 項目は内部エネルギーを表す。この Lagrangian をそのまま Euler-Lagrangian 方程式を使って解くと、$4N$ 個の方程式になってしまうので、彼らは次の N 個の拘束条件を導入した。

$$\phi_i = \frac{4\pi}{3} h_i^3 \rho_i - \overline{m} N_{\mathrm{neigh}} = 0 \tag{9}$$

ここで、$\overline{m}$ は SPH 粒子の平均質量[注 3)]、$N_{\rm neigh}$ は近傍粒子数 (定数) である。この拘束条件の下、Lagrange の未定乗数法を使って、Euler-Lagrange 方程式をとけば、以下の運動方程式が得られる:

$$\frac{{\rm d}\boldsymbol{v}_i}{{\rm d}t} = -\sum_{j=1}^{N} m_j \left[f_i \frac{P_i}{\rho_i^2} \nabla_i W(r_{ij}, h_i) + f_j \frac{P_j}{\rho_j^2} \nabla_i W(r_{ij}, h_j) \right] \tag{10}$$

ここで、P_i は圧力、$r_{ij} = |\boldsymbol{r}_i - \boldsymbol{r}_j|$、$W$ はカーネル関数、f_i は ∇h term と呼ばれる量で、

$$f_i = \left(1 + \frac{h_i}{3\rho_i} \frac{\partial \rho_i}{\partial h_i} \right)^{-1} \tag{11}$$

と定義される。

系の熱力学的状態はエントロピー A_i を独立変数として記述される。断熱過程の場合、エントロピーは衝撃波以外のところでは流れに沿って一定である。Springel [2005, MNRAS, 364, 1105] では、衝撃波でのエントロピー増加と速度の変化を人工粘性を使って次のようにモデル化している:

$$\frac{{\rm d}A_i}{{\rm d}t} = \frac{1}{2} \frac{\gamma - 1}{\rho_i^{\gamma - 1}} \sum_{j=1}^{N} m_j \Pi_{ij} \boldsymbol{v}_{ij} \cdot \nabla_i \overline{W}_{ij} \tag{12}$$

$$\left. \frac{{\rm d}\boldsymbol{v}_i}{{\rm d}t} \right|_{\rm visc} = -\sum_{j=1}^{N} m_j \Pi_{ij} \nabla_i \overline{W}_{ij} \tag{13}$$

ここで、$\boldsymbol{v}_{ij} = \boldsymbol{v}_i - \boldsymbol{v}_j$、$\boldsymbol{v}_i$ は速度、$\overline{W}_{ij} = \frac{1}{2}(W(r_{ij}, h_i) + W(r_{ij}, h_j))$ である。Π_{ij} に関しては、原論文を参照して頂きたい。

したがって、SPH 計算の手順は次のようになる:

(1) 式 (9) と以下の式を無矛盾に解き、密度 ρ_i と h_i を決定する。

$$\rho_i = \sum_{j=1}^{N} m_j W(r_{ij}, h_i) \tag{14}$$

(2) 式 (11) で定義される ∇h term を計算する。
(3) 式 (10)、(12)、(13) の右辺を計算する。
(4) SPH 粒子の位置、速度、エントロピーを時間積分する。

以下、まずユーザ定義クラスと相互作用関数の実装について解説を行い、次にメインルーチンの実装について解説を行う。複数粒子種の取扱は後者で解説する。

7.1.3 ユーザー定義型

本サンプルコードのユーザ定義型はすべて `user_defined.h` に定義されている。はじめに用意されているユーザ定義型の種類について簡単に説明しておく。冒頭で述べたように、本

[注 3)] 拘束条件に使用していることから、定数扱いであることに注意。

サンプルコードは 2 種類の粒子 (N 体粒子, SPH 粒子) を扱う。そのため、FullParticle 型も 2 種類用意している (N 体粒子用に 構造体 `fp_nbody` を, SPH 粒子用に 構造体 `fp_sph`)。相互作用は重力相互作用と流体相互作用の 2 種類ある。そのため、Force 型を 3 種類用意している (重力計算用に 構造体 `force_grav` を、密度計算用に 構造体 `force_dens` を、そして、圧力勾配による加速度 (以下、単に圧力勾配加速度) の計算用に 構造体 `force_hydro` を; 第 4 節も参照のこと)。本サンプルコードでは、簡単のため、EssentialParticleI 型と EssentialParticleJ 型を 1 つの構造体で兼ねることにし (以下、単に EssentialParticle 型)、密度計算と圧力勾配加速度計算に同じ EssentialParticle 型を使用する。したがって、EssentialParticle 型の種類は 2 種類となっている (重力計算用に 構造体 `ep_grav` を、SPH 計算用に 構造体 `ep_hydro`)。

以下、各ユーザ定義型の実装について説明する。

7.1.3.1 FullParticle 型

まず、N 体粒子用の FullParticle 型である 構造体 `fp_nbody` について解説する。この構造体には、N 体粒子が持っているべき全ての物理量が含まれている。Listing 54 に、この構造体の実装を示す。メンバ変数 の構成は、第 3-4 節で紹介した N 体計算サンプルコードとほぼ同じであり、詳細はそちらを参照されたい。

Listing 54: FullParticle 型 (構造体 `fp_nbody`)

```
1 typedef struct fp_nbody { //$fdps FP
2    //$fdps copyFromForce force_grav (acc,acc) (pot,pot)
3    long long int id; //$fdps id
4    double mass; //$fdps charge
5    fdps_f64vec pos; //$fdps position
6    fdps_f64vec vel;
7    fdps_f64vec acc;
8    double pot;
9 } FP_nbody;
```

次に、SPH 粒子用の FullParticle 型である 構造体 `fp_sph` について解説する。この構造体には、SPH 粒子が持っているべき全ての物理量が含まれている。Listing 55 に、この構造体の実装を示す。メンバ変数の内、主要な変数の意味は次の通りである: `id` (識別番号)、`mass` (質量)、`pos` (位置 $[\boldsymbol{r}_i]$)、`vel` (速度 $[\boldsymbol{v}_i]$)、`acc_grav` (重力加速度)、`pot_grav` (重力ポテンシャル)、`acc_hydro` (圧力勾配加速度)、`dens` (密度 $[\rho_i]$)、`eng` (単位質量あたりの内部エネルギー $[u_i]$)、`ent` (エントロピー関数 [以下、単にエントロピー]$[A_i]$)、`pres` (圧力 $[P_i]$)、`smth` (smoothing length[注 4)] $[h_i]$)、`gradh` (∇h term$[f_i]$)、`divv` ($(\nabla \cdot \boldsymbol{v})_i$、ここで下付きの i は粒子 i の位置での微分を示している)、`rotv` ($(\nabla \times \boldsymbol{v})_i$)、 `balsw` (Balsara switch のための係数で、定義式は Balsara [1995, JCP, 121, 357] の $f(a)$)、`snds` (音速)、`eng_dot` (`eng` の時間変化率)、`ent_dot` (`ent` の時間変化率)、`dt` (この粒子の軌道を時間積分するときに許される最大の時間刻み幅)。

以下の点に注意して頂きたい。

- SPH 粒子が関わる相互作用計算は、重力計算、密度計算、圧力勾配加速度計算の 3 種類あ

[注 4)] カーネル関数が 0 になる距離と定義。

るので、それに応じて copyFromForce 指示文も 3 つ用意されている。

Listing 55: FullParticle 型 (構造体 fp_sph)

```
1  typedef struct fp_sph { //$fdps FP
2     //$fdps copyFromForce force_grav (acc,acc_grav) (pot,pot_grav)
3     //$fdps copyFromForce force_dens (flag,flag) (dens,dens) (smth,smth) (
            gradh,gradh) (divv,divv) (rotv,rotv)
4     //$fdps copyFromForce force_hydro (acc,acc_hydro) (eng_dot,eng_dot) (
            ent_dot,ent_dot) (dt,dt)
5     long long id; //$fdps id
6     double mass; //$fdps charge
7     fdps_f64vec pos; //$fdps position
8     fdps_f64vec vel;
9     fdps_f64vec acc_grav;
10    double pot_grav;
11    fdps_f64vec acc_hydro;
12    int flag;
13    double dens;
14    double eng;
15    double ent;
16    double pres;
17    double smth;
18    double gradh;
19    double divv;
20    fdps_f64vec rotv;
21    double balsw;
22    double snds;
23    double eng_dot;
24    double ent_dot;
25    double dt;
26    fdps_f64vec vel_half;
27    double eng_half;
28    double ent_half;
29 } FP_sph;
```

7.1.3.2 EssentialParticle 型

まず、重力計算用の EssentialParticle 型である 構造体 ep_grav について解説する。この構造体には、重力計算を行う際、i 粒子と j 粒子が持っているべき全ての物理量をメンバ変数として持たせている。Listing 56 に、この構造体の実装を示す。 ユーザは EssentialParticle 型の定義部に、FullParticle 型からのデータコピーの方法を指示するため指示文 (copyFromFP 指示文) を書く必要があるが、本サンプルコードでは粒子種が 2 種類のため、2 つの copyFromFP 指示文が実装されていることに注意されたい。

Listing 56: EssentialParticle 型 (構造体 ep_grav)

```
1  typedef struct ep_grav { //$fdps EPI,EPJ
2     //$fdps copyFromFP fp_nbody (id,id) (mass,mass) (pos,pos)
3     //$fdps copyFromFP fp_sph (id,id) (mass,mass) (pos,pos)
4     long long id; //$fdps id
5     double mass; //$fdps charge
```

```
6     fdps_f64vec pos; //$fdps position
7 } EP_grav;
```

次に、密度計算と圧力勾配加速度計算用の EssentialParticle 型である 構造体 ep_hydro について解説する。この構造体には、密度計算と圧力勾配加速度計算を行う際、i 粒子と j 粒子が持つべき全ての物理量をメンバ変数として持たせている。Listing 57 に、この構造体の実装を示す。

Listing 57: EssentialParticle 型 (構造体 ep_hydro)

```
1  typedef struct ep_hydro { //$fdps EPI,EPJ
2     //$fdps copyFromFP fp_sph (id,id) (pos,pos) (vel,vel) (mass,mass) (smth
           ,smth) (dens,dens) (pres,pres) (gradh,gradh) (snds,snds) (balsw,
           balsw)
3     long long id; //$fdps id
4     fdps_f64vec pos; //$fdps position
5     fdps_f64vec vel;
6     double mass; //$fdps charge
7     double smth; //$fdps rsearch
8     double dens;
9     double pres;
10    double gradh;
11    double snds;
12    double balsw;
13 } EP_hydro;
```

7.1.3.3 Force 型

まず、重力計算用の Force 型である 構造体 force_grav について解説する。この構造体は、重力計算を行った際にその結果として得られる全ての物理量をメンバ変数として持っている必要がある。Listing 58 にこの構造体の実装を示す。

Listing 58: Force 型 (構造体 force_grav)

```
1 typedef struct force_grav { //$fdps Force
2     //$fdps clear
3     fdps_f64vec acc;
4     double pot;
5 } Force_grav;
```

次に、密度計算用の Force 型である 構造体 force_dens について解説する。この構造体は、密度計算を行った際にその結果として得られる全ての物理量をメンバ変数として持っている必要がある。Listing 59 に、この構造体の実装を示す。本サンプルコードの SPH 法では、smoothing length は固定ではなく、密度に応じて変化する。そのため、メンバ変数として smth を持つ。また、密度計算と同時に、∇h term および $(\nabla \cdot \boldsymbol{v})_i$、$(\nabla \times \boldsymbol{v})_i$ の計算を同時行うため、メンバ変数に gradh, divv, rotv を持つ。メンバ変数 flag は ρ_i と h_i を決定するイテレーション計算が収束したかどうかの結果を格納する変数である (詳細は、相互作用関数の節を参照のこと)。

Listing 59: Force 型 (構造体 force_dens)

```
typedef struct force_dens { //$fdps Force
   //$fdps clear smth=keep
   int flag;
   double dens;
   double smth;
   double gradh;
   double divv;
   fdps_f64vec rotv;
} Force_dens;
```

最後に、圧力勾配加速度計算用の Force 型である 構造体 force_hydro について解説する。この構造体は、圧力勾配加速度計算を行った際にその結果として得られる全ての物理量をメンバ変数として持っている必要がある。Listing 60 に、この構造体の実装を示す。

Listing 60: Force 型 (構造体 force_hydro)

```
typedef struct force_hydro { //$fdps Force
   //$fdps clear
   fdps_f64vec acc;
   double eng_dot;
   double ent_dot;
   double dt;
} Force_hydro;
```

7.1.4 相互作用関数

本サンプルコードで使用する相互作用関数はすべて user_defined.c に実装されている。全部で 4 種類あり、重力計算 (粒子間相互作用及び粒子-超粒子間相互作用)、密度計算、圧力勾配加速度計算に使用される。以下、順に説明していく。

7.1.4.1 重力計算

重力計算用の相互作用関数は void 関数 calc_gravity_ep_ep 及び calc_gravity_ep_sp として実装されている。Listing 61 にこれらの実装を示す。実装は第 3-4 節で紹介した N 体計算サンプルコードのものとほぼ同じであり、詳細はそちらを参照されたい。

Listing 61: 相互作用関数 (重力計算用)

```
#if defined(ENABLE_PHANTOM_GRAPE_X86)
void calc_gravity_ep_ep(struct ep_grav *ep_i,
                        int n_ip,
                        struct ep_grav *ep_j,
                        int n_jp,
                        struct force_grav *f) {
    int i,j;
    int nipipe = n_ip;
    int njpipe = n_jp;
    double (*xi)[3] = (double (*)[3])malloc(sizeof(double) * nipipe * 3);
    double (*ai)[3] = (double (*)[3])malloc(sizeof(double) * nipipe * 3);
    double  *pi     = (double  *    )malloc(sizeof(double) * nipipe);
    double (*xj)[3] = (double (*)[3])malloc(sizeof(double) * njpipe * 3);
```

```
14      double  *mj     = (double  *    )malloc(sizeof(double) * njpipe);
15      for (i = 0; i < n_ip; i++) {
16          xi[i][0] = ep_i[i].pos.x;
17          xi[i][1] = ep_i[i].pos.y;
18          xi[i][2] = ep_i[i].pos.z;
19          ai[i][0] = 0.0;
20          ai[i][1] = 0.0;
21          ai[i][2] = 0.0;
22          pi[i]    = 0.0;
23      }
24      for (j = 0; j < n_jp; j++) {
25          xj[j][0] = ep_j[j].pos.x;
26          xj[j][1] = ep_j[j].pos.y;
27          xj[j][2] = ep_j[j].pos.z;
28          mj[j]    = ep_j[j].mass;
29      }
30  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
31      int devid = omp_get_thread_num();
32      // [IMPORTANT NOTE]
33      //   The function calc_gravity_ep_ep is called by a OpenMP thread
34      //   in the FDPS. This means that here is already in the parallel
              region.
35      //   So, you can use omp_get_thread_num() without !$OMP parallel
              directives.
36      //   If you use them, a nested parallel resions is made and the
              gravity
37      //   calculation will not be performed correctly.
38  #else
39      int devid = 0;
40  #endif
41      g5_set_xmjMC(devid, 0, n_jp, xj, mj);
42      g5_set_nMC(devid, n_jp);
43      g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip);
44      for (i = 0; i < n_ip; i++) {
45         f[i].acc.x += ai[i][0];
46         f[i].acc.y += ai[i][1];
47         f[i].acc.z += ai[i][2];
48         f[i].pot   -= pi[i];
49      }
50      free(xi);
51      free(ai);
52      free(pi);
53      free(xj);
54      free(mj);
55  }
56
57  void calc_gravity_ep_sp(struct ep_grav *ep_i,
58                          int n_ip,
59                          fdps_spj_monopole *ep_j,
60                          int n_jp,
61                          struct force_grav *f) {
62      int i,j;
63      int nipipe = n_ip;
64      int njpipe = n_jp;
65      double (*xi)[3] = (double (*)[3])malloc(sizeof(double) * nipipe * 3);
```

```
66      double (*ai)[3] = (double (*)[3])malloc(sizeof(double) * nipipe * 3);
67      double  *pi     = (double  *    )malloc(sizeof(double) * nipipe);
68      double (*xj)[3] = (double (*)[3])malloc(sizeof(double) * njpipe * 3);
69      double  *mj     = (double  *    )malloc(sizeof(double) * njpipe);
70      for (i = 0; i < n_ip; i++) {
71          xi[i][0] = ep_i[i].pos.x;
72          xi[i][1] = ep_i[i].pos.y;
73          xi[i][2] = ep_i[i].pos.z;
74          ai[i][0] = 0.0;
75          ai[i][1] = 0.0;
76          ai[i][2] = 0.0;
77          pi[i]    = 0.0;
78      }
79      for (j = 0; j < n_jp; j++) {
80          xj[j][0] = ep_j[j].pos.x;
81          xj[j][1] = ep_j[j].pos.y;
82          xj[j][2] = ep_j[j].pos.z;
83          mj[j]    = ep_j[j].mass;
84      }
85  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
86      int devid = omp_get_thread_num();
87      // [IMPORTANT NOTE]
88      //   The function calc_gravity_ep_ep is called by a OpenMP thread
89      //   in the FDPS. This means that here is already in the parallel
              region.
90      //   So, you can use omp_get_thread_num() without !$OMP parallel
              directives.
91      //   If you use them, a nested parallel resions is made and the
              gravity
92      //   calculation will not be performed correctly.
93  #else
94      int devid = 0;
95  #endif
96      g5_set_xmjMC(devid, 0, n_jp, xj, mj);
97      g5_set_nMC(devid, n_jp);
98      g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip);
99      for (i = 0; i < n_ip; i++) {
100        f[i].acc.x += ai[i][0];
101        f[i].acc.y += ai[i][1];
102        f[i].acc.z += ai[i][2];
103        f[i].pot   -= pi[i];
104     }
105     free(xi);
106     free(ai);
107     free(pi);
108     free(xj);
109     free(mj);
110 }
111 #else
112 void calc_gravity_ep_ep(struct ep_grav *ep_i,
113                         int n_ip,
114                         struct ep_grav *ep_j,
115                         int n_jp,
116                         struct force_grav *f) {
117     int i,j;
```

```
118     double eps2 = eps_grav * eps_grav;
119     for (i = 0; i < n_ip; i++) {
120         fdps_f64vec xi,ai;
121         double poti;
122         xi.x = ep_i[i].pos.x;
123         xi.y = ep_i[i].pos.y;
124         xi.z = ep_i[i].pos.z;
125         ai.x = 0.0;
126         ai.y = 0.0;
127         ai.z = 0.0;
128         poti = 0.0;
129         for (j = 0; j < n_jp; j++) {
130             fdps_f64vec rij;
131             rij.x  = xi.x - ep_j[j].pos.x;
132             rij.y  = xi.y - ep_j[j].pos.y;
133             rij.z  = xi.z - ep_j[j].pos.z;
134             double r3_inv = rij.x * rij.x
135                           + rij.y * rij.y
136                           + rij.z * rij.z
137                           + eps2;
138             double r_inv  = 1.0/sqrt(r3_inv);
139             r3_inv = r_inv * r_inv;
140             r_inv  = r_inv * ep_j[j].mass;
141             r3_inv = r3_inv * r_inv;
142             ai.x -= r3_inv * rij.x;
143             ai.y -= r3_inv * rij.y;
144             ai.z -= r3_inv * rij.z;
145             poti -= r_inv;
146         }
147         f[i].acc.x += ai.x;
148         f[i].acc.y += ai.y;
149         f[i].acc.z += ai.z;
150         f[i].pot   += poti;
151     }
152 }
153
154 void calc_gravity_ep_sp(struct ep_grav *ep_i,
155                         int n_ip,
156                         fdps_spj_monopole *ep_j,
157                         int n_jp,
158                         struct force_grav *f) {
159     int i,j;
160     double eps2 = eps_grav * eps_grav;
161     for (i = 0; i < n_ip; i++) {
162         fdps_f64vec xi,ai;
163         double poti;
164         xi.x = ep_i[i].pos.x;
165         xi.y = ep_i[i].pos.y;
166         xi.z = ep_i[i].pos.z;
167         ai.x = 0.0;
168         ai.y = 0.0;
169         ai.z = 0.0;
170         poti = 0.0;
171         for (j = 0; j < n_jp; j++) {
172             fdps_f64vec rij;
```

```
173             rij.x  = xi.x - ep_j[j].pos.x;
174             rij.y  = xi.y - ep_j[j].pos.y;
175             rij.z  = xi.z - ep_j[j].pos.z;
176             double r3_inv = rij.x * rij.x
177                           + rij.y * rij.y
178                           + rij.z * rij.z
179                           + eps2;
180             double r_inv  = 1.0/sqrt(r3_inv);
181             r3_inv = r_inv * r_inv;
182             r_inv  = r_inv * ep_j[j].mass;
183             r3_inv = r3_inv * r_inv;
184             ai.x -= r3_inv * rij.x;
185             ai.y -= r3_inv * rij.y;
186             ai.z -= r3_inv * rij.z;
187             poti -= r_inv;
188         }
189         f[i].acc.x += ai.x;
190         f[i].acc.y += ai.y;
191         f[i].acc.z += ai.z;
192         f[i].pot   += poti;
193     }
194 }
195 #endif
```

7.1.4.2 密度計算

密度計算用の相互作用関数は void 関数 calc_density として実装されている。Listing 62 に、実装を示す。実装はマクロ ENABLE_VARIABLE_SMOOTHING_LENGTH が定義されているか否かで分かれる。このマクロが未定義の場合には、固定長カーネルコードとなり、実装は第 3-4 節で紹介した SPH サンプルコードとほぼ同じであるので、そちらを参照されたい。以下、このマクロが定義されている場合の実装について解説する。

第 7.1.2 節で説明したように、密度 ρ_i と smoothing length h_i は式 (14) と式 (9) を無矛盾に解いて決定する必要がある。これには 2 つの方程式を反復的に解く必要がある。このイテレーションを無限 for ループの中で行っている。本サンプルコードでは ρ_i と h_i の計算を効率的に行うため、smoothing length の値を定数 SCF_smth 倍してから密度計算を実行している 。このため、定数倍する前の smoothing length の値を $h_{i,0}$ とすると、このイテレーションの間に h_i を 0 から $h_{\mathrm{max,alw}} \equiv \mathtt{scf\_smth} \times h_{i,0}$ までの間なら変化させてもよいことになる。なぜなら、j 粒子リストの取りこぼしは発生しないからである。逆にこの範囲でイテレーションが収束しなければ、求めたい h_i は $h_{\mathrm{max,alw}}$ よりも大きいということになり、既存の j 粒子リストでは ρ_i と h_i を決定できないということになる。この場合、$h_{i,0}$ を大きくした上で、密度計算をやり直す必要がある。この外側のイテレーションは c_main.c の void 関数 calc_density_wrapper で行われている。この void 関数の詳細は第 7.1.5 節で行う。

無限 for ループの後には、∇h term の計算、$(\nabla \cdot \boldsymbol{v})_i$ 及び $(\nabla \times \boldsymbol{v})_i$ の計算を行っている。

Listing 62: 相互作用関数 (密度計算用)

```
1 void calc_density(struct ep_hydro *ep_i,
2                   int n_ip,
```

```
3                   struct ep_hydro *ep_j,
4                   int n_jp,
5                   struct force_dens *f) {
6 #if defined(ENABLE_VARIABLE_SMOOTHING_LENGTH)
7     // Local parameters
8     const double eps=1.0e-6;
9     // Local variables
10    int i,j;
11    int n_unchanged;
12    double M,M_trgt;
13    double dens,drho_dh;
14    double h,h_max_alw,h_L,h_U,dh,dh_prev;
15    fdps_f64vec dr,dv,gradW_i;
16    double *mj  = (double *)malloc(sizeof(double) * n_jp);
17    double *rij = (double *)malloc(sizeof(double) * n_jp);
18    M_trgt = mass_avg * N_neighbor;
19    for (i = 0; i < n_ip; i++) {
20        dens = 0.0;
21        h_max_alw = ep_i[i].smth; // maximum allowance
22        h = h_max_alw / SCF_smth;
23        // Note that we increase smth by a factor of scf_smth
24        // before calling calc_density().
25        h_L = 0.0;
26        h_U = h_max_alw;
27        dh_prev = 0.0;
28        n_unchanged = 0;
29        // Software cache
30        for (j = 0; j < n_jp; j++) {
31            mj[j] = ep_j[j].mass;
32            dr.x = ep_i[i].pos.x - ep_j[j].pos.x;
33            dr.y = ep_i[i].pos.y - ep_j[j].pos.y;
34            dr.z = ep_i[i].pos.z - ep_j[j].pos.z;
35            rij[j] = sqrt(dr.x * dr.x
36                          +dr.y * dr.y
37                          +dr.z * dr.z);
38        }
39        for(;;) {
40            // Calculate density
41            dens = 0.0;
42            for (j = 0; j < n_jp; j++)
43               dens += mj[j] * W(rij[j], h);
44            // Check if the current value of the smoohting length
                  satisfies
45            // Eq.(5) in Springel (2005).
46            M = 4.0 * pi * h * h * h * dens / 3.0;
47            if ((h < h_max_alw) && (fabs(M/M_trgt - 1.0) < eps)) {
48                // In this case, Eq.(5) holds within a specified accuracy.
49                f[i].flag = 1;
50                f[i].dens = dens;
51                f[i].smth = h;
52                break;
53            }
54            if (((h == h_max_alw) && (M < M_trgt)) || (n_unchanged == 4))
                  {
55                // In this case, we skip this particle forcibly.
```

```
56                  // In order to determine consistently the density
57                  // and the smoohting length for this particle,
58                  // we must re-perform calcForceAllAndWriteBack().
59                  f[i].flag = 0;
60                  f[i].dens = dens;
61                  f[i].smth = h_max_alw;
62                  break;
63              }
64              // Update h_L & h_U
65              if (M < M_trgt) {
66                  if (h_L < h) h_L = h;
67              } else if (M_trgt < M) {
68                  if (h < h_U) h_U = h;
69              }
70              dh = h_U - h_L;
71              if (dh == dh_prev) {
72                 n_unchanged++;
73              } else {
74                 dh_prev = dh;
75                 n_unchanged = 0;
76              }
77              // Update smoothing length
78              h = pow((3.0 * M_trgt)/(4.0 * pi * dens),1.0/3.0);
79              if ((h <= h_L) || (h == h_U)) {
80                 // In this case, we switch to the bisection search.
81                 // The inclusion of '=' in the if statement is very
82                 // important to escape a limit cycle.
83                 h = 0.5 * (h_L + h_U);
84              } else if (h_U < h) {
85                 h = h_U;
86              }
87          }
88          // Calculate grad-h term
89          if (f[i].flag == 1) {
90              drho_dh = 0.0;
91              for (j = 0; j < n_jp; j++)
92                 drho_dh += mj[j] * dWdh(rij[j], h);
93              f[i].gradh = 1.0 / (1.0 + (h * drho_dh) / (3.0 * dens));
94          } else {
95              f[i].gradh = 1.0; // dummy value
96          }
97          // Compute \div v & \rot v for Balsara switch
98  #if defined(USE_BALSARA_SWITCH)
99          for (j = 0; j < n_jp; j++) {
100            dr.x = ep_i[i].pos.x - ep_j[j].pos.x;
101            dr.y = ep_i[i].pos.y - ep_j[j].pos.y;
102            dr.z = ep_i[i].pos.z - ep_j[j].pos.z;
103            dv.x = ep_i[i].vel.x - ep_j[j].vel.x;
104            dv.y = ep_i[i].vel.y - ep_j[j].vel.y;
105            dv.z = ep_i[i].vel.z - ep_j[j].vel.z;
106            gradW_i = gradW(dr, f[i].smth);
107            f[i].divv -= mj[j] * (dv.x * gradW_i.x
108                                  +dv.y * gradW_i.y
109                                  +dv.z * gradW_i.z);
110            f[i].rotv.x -= mj[j] * (dv.y * gradW_i.z - dv.z * gradW_i.y);
```

```
111             f[i].rotv.y -= mj[j] * (dv.z * gradW_i.x - dv.x * gradW_i.z);
112             f[i].rotv.z -= mj[j] * (dv.x * gradW_i.y - dv.y * gradW_i.x);
113         }
114         f[i].divv   /= f[i].dens;
115         f[i].rotv.x /= f[i].dens;
116         f[i].rotv.y /= f[i].dens;
117         f[i].rotv.z /= f[i].dens;
118 #endif
119     }
120     free(mj);
121     free(rij);
122 #else
123     int i,j;
124     for (i = 0; i < n_ip; i++) {
125         f[i].dens = 0.0;
126         for (j = 0; j < n_jp; j++) {
127             fdps_f64vec dr;
128             dr.x = ep_j[j].pos.x - ep_i[i].pos.x;
129             dr.y = ep_j[j].pos.y - ep_i[i].pos.y;
130             dr.z = ep_j[j].pos.z - ep_i[i].pos.z;
131             double rij = sqrt(dr.x * dr.x
132                               +dr.y * dr.y
133                               +dr.z * dr.z);
134             f[i].dens += ep_j[j].mass * W(rij,ep_i[i].smth);
135         }
136         f[i].smth = ep_i[i].smth;
137         f[i].gradh = 1.0;
138         // Compute \div v & \rot v for Balsara switch
139 #if defined(USE_BALSARA_SWITCH)
140         for (j = 0; j < n_jp; j++) {
141             double mj = ep_j[j].mass;
142             fdps_f64vec dr,dv,gradW_i;
143             dr.x = ep_i[i].pos.x - ep_j[j].pos.x;
144             dr.y = ep_i[i].pos.y - ep_j[j].pos.y;
145             dr.z = ep_i[i].pos.z - ep_j[j].pos.z;
146             dv.x = ep_i[i].vel.x - ep_j[j].vel.x;
147             dv.y = ep_i[i].vel.y - ep_j[j].vel.y;
148             dv.z = ep_i[i].vel.z - ep_j[j].vel.z;
149             gradW_i = gradW(dr, f[i].smth);
150             f[i].divv -= mj * (dv.x * gradW_i.x
151                                +dv.y * gradW_i.y
152                                +dv.z * gradW_i.z);
153             f[i].rotv.x -= mj * (dv.y * gradW_i.z - dv.z * gradW_i.y);
154             f[i].rotv.y -= mj * (dv.z * gradW_i.x - dv.x * gradW_i.z);
155             f[i].rotv.z -= mj * (dv.x * gradW_i.y - dv.y * gradW_i.x);
156         }
157         f[i].divv   /= f[i].dens;
158         f[i].rotv.x /= f[i].dens;
159         f[i].rotv.y /= f[i].dens;
160         f[i].rotv.z /= f[i].dens;
161 #endif
162     }
163 #endif
164 }
```

7.1.4.3 圧力勾配加速度計算

圧力勾配加速度用の相互作用関数は void 関数 calc_hydro_force として実装されている。Listing 63 に、実装を示す。この void 関数では、式 (10)、(12)、(13) の右辺の計算、及び、Springel [2005, MNRAS, 364, 1105] の式 (16) に従って dt の計算を行っている (dt については 構造体 fp_sph の説明を参照のこと)。

Listing 63: 相互作用関数 (圧力勾配加速度計算用)

```
1  void calc_hydro_force(struct ep_hydro *ep_i,
2                        int n_ip,
3                        struct ep_hydro *ep_j,
4                        int n_jp,
5                        struct force_hydro *f) {
6      // Local variables
7      int i,j;
8      double mass_i,mass_j,smth_i,smth_j,
9             dens_i,dens_j,pres_i,pres_j,
10            gradh_i,gradh_j,balsw_i,balsw_j,
11            snds_i,snds_j;
12     double povrho2_i,povrho2_j,
13            v_sig_max,dr_dv,w_ij,v_sig,AV;
14     fdps_f64vec pos_i,pos_j,vel_i,vel_j,
15                 dr,dv,gradW_i,gradW_j,gradW_ij;
16     for (i = 0; i < n_ip; i++) {
17         // Zero-clear
18         v_sig_max = 0.0;
19         // Extract i-particle info.
20         pos_i = ep_i[i].pos;
21         vel_i = ep_i[i].vel;
22         mass_i  = ep_i[i].mass;
23         smth_i  = ep_i[i].smth;
24         dens_i  = ep_i[i].dens;
25         pres_i  = ep_i[i].pres;
26         gradh_i = ep_i[i].gradh;
27         balsw_i = ep_i[i].balsw;
28         snds_i  = ep_i[i].snds;
29         povrho2_i = pres_i/(dens_i*dens_i);
30         for (j = 0; j < n_jp; j++) {
31             // Extract j-particle info.
32             pos_j.x = ep_j[j].pos.x;
33             pos_j.y = ep_j[j].pos.y;
34             pos_j.z = ep_j[j].pos.z;
35             vel_j.x = ep_j[j].vel.x;
36             vel_j.y = ep_j[j].vel.y;
37             vel_j.z = ep_j[j].vel.z;
38             mass_j  = ep_j[j].mass;
39             smth_j  = ep_j[j].smth;
40             dens_j  = ep_j[j].dens;
41             pres_j  = ep_j[j].pres;
42             gradh_j = ep_j[j].gradh;
43             balsw_j = ep_j[j].balsw;
44             snds_j  = ep_j[j].snds;
45             povrho2_j = pres_j/(dens_j*dens_j);
46             // Compute dr & dv
```

```
47              dr.x = pos_i.x - pos_j.x;
48              dr.y = pos_i.y - pos_j.y;
49              dr.z = pos_i.z - pos_j.z;
50              dv.x = vel_i.x - vel_j.x;
51              dv.y = vel_i.y - vel_j.y;
52              dv.z = vel_i.z - vel_j.z;
53              // Compute the signal velocity
54              dr_dv = dr.x * dv.x + dr.y * dv.y + dr.z * dv.z;
55              if (dr_dv < 0.0) {
56                 w_ij = dr_dv / sqrt(dr.x * dr.x + dr.y * dr.y + dr.z * dr.z
                        );
57              } else {
58                 w_ij = 0.0;
59              }
60              v_sig = snds_i + snds_j - 3.0 * w_ij;
61              if (v_sig > v_sig_max) v_sig_max = v_sig;
62              // Compute the artificial viscosity
63              AV = - 0.5*v_sig*w_ij / (0.5*(dens_i+dens_j)) * 0.5*(balsw_i+
                      balsw_j);
64              // Compute the average of the gradients of kernel
65              gradW_i  = gradW(dr,smth_i);
66              gradW_j  = gradW(dr,smth_j);
67              gradW_ij.x = 0.5 * (gradW_i.x + gradW_j.x);
68              gradW_ij.y = 0.5 * (gradW_i.y + gradW_j.y);
69              gradW_ij.z = 0.5 * (gradW_i.z + gradW_j.z);
70              // Compute the acceleration and the heating rate
71              f[i].acc.x -= mass_j*(gradh_i * povrho2_i * gradW_i.x
72                                    +gradh_j * povrho2_j * gradW_j.x
73                                    +AV * gradW_ij.x);
74              f[i].acc.y -= mass_j*(gradh_i * povrho2_i * gradW_i.y
75                                    +gradh_j * povrho2_j * gradW_j.y
76                                    +AV * gradW_ij.y);
77              f[i].acc.z -= mass_j*(gradh_i * povrho2_i * gradW_i.z
78                                    +gradh_j * povrho2_j * gradW_j.z
79                                    +AV * gradW_ij.z);
80              f[i].eng_dot += mass_j * gradh_i * povrho2_i * (dv.x * gradW_i
                      .x
81                                                             +dv.y * gradW_i
                                                                    .y
82                                                             +dv.z * gradW_i
                                                                    .z);
83                           + mass_j * 0.5 * AV * (dv.x * gradW_ij.x
84                                                 +dv.y * gradW_ij.y
85                                                 +dv.z * gradW_ij.z);
86              f[i].ent_dot += 0.5 * mass_j * AV * (dv.x * gradW_ij.x
87                                                   +dv.y * gradW_ij.y
88                                                   +dv.z * gradW_ij.z);
89          }
90          f[i].ent_dot *= ((specific_heat_ratio - 1.0)
91                           /pow(dens_i,specific_heat_ratio - 1.0));
92          f[i].dt = CFL_hydro*2.0*smth_i/v_sig_max;
93      }
94  }
```

7.1.5 プログラム本体

本節では、主に c_main.c に実装されたサンプルコード本体について解説を行う。詳細な説明に入る前に、サンプルコードの内容と全体構造について説明を与える。7.1 節冒頭で述べたように、このサンプルコードでは円盤銀河の N 体/SPH シミュレーションを行うものであるが、初期条件としては円盤銀河の他、簡単なテスト計算用の初期条件も用意されている。具体的に以下の 4 つの場合に対応している:

(a) 円盤銀河用の初期条件。この初期条件はコンパイルオプション時に-DINITIAL_CONDITION=0 が指定された場合に選択される。初期条件作成は ic.c の void 関数 galaxy_IC で行われる。ダークマターと星の分布は事前に MAGI で作成されたファイルを読み込んで設定される。一方、ガスの初期分布はこの void 関数内部で生成される。デフォルトでは粒子数 2^{18} で exponential disk ($M = 10^{10}$ M$_\odot$, $R_s = 7$ kpc [scale radius], $R_t = 12.5$ kpc [truncation radius], $z_d = 0.4$ kpc [scale height], $z_t = 1$ kpc [truncation height]) が生成される。

(b) Cold collapse 問題用の初期条件。この初期条件はコンパイルオプション時に-DINITIAL_CONDITION=1 が指定された場合に選択される。初期条件作成は ic.c の void 関数 cold_collapse_test_IC で行われる。

(c) Evrard test (Evrard [1988,MNRAS,235,911] の第 3.3 節) 用の初期条件。この初期条件はコンパイルオプション時に-DINITIAL_CONDITION=2 が指定された場合に選択される。初期条件作成は ic.c の void 関数 Evrard_test_IC で行われる。作成方法は 2 つあり、void 関数の最後の引数の値を手動で 0 か 1 にして指定する。0 の場合、格子状に並んだ SPH 粒子から Evrard 球の密度分布を作成する。1 の場合、ガラス状に分布した SPH 粒子から Evrard 球の密度分布を作成する。1 を選択するためには、事前に次項で説明するモードで SPH 粒子のデータを作成しておく必要がある。

(d) $[-1, 1)^3$ の立方体中に一様密度のガラス状の SPH 粒子分布を作成するための初期条件/動作モード。この初期条件はコンパイルオプション時に-DINITIAL_CONDITION=3 が指定された場合に選択される。初期条件作成は ic.c の void 関数 make_glass_IC で行われる。

コード全体の構造は以下のようになっている:

(1) FDPS で使用するオブジェクトの生成と初期化
(2) (必要であれば)Phantom-GRAPE ライブラリの初期化
(3) 初期条件ファイルの読み込み、或いは、初期条件の作成
(4) 終了時刻まで粒子の運動を計算

以下で、個々について詳しく説明を行う。

7.1.5.1 ヘッダファイルのインクルード

FDPS の C 言語用 API にアクセスできるようにするため、c_main.c のファイル冒頭部分で、FDPS_c_if.h をインクルードしている。

Listing 64: FDPS の C 言語用 API のヘッダーファイルのインクルード

```
1 #include "FDPS_c_if.h"
```

7.1.5.2 開始、終了

まずは、FDPS の初期化/開始を行う必要がある。次のように、メイン関数に記述する。

Listing 65: FDPS の開始

```
1 fdps_initialize();
```

FDPS は、開始したら明示的に終了させる必要がある。今回は、プログラムの終了と同時に FDPS も終了させるため、メイン関数の最後に次のように記述する。

Listing 66: FDPS の終了

```
1 fdps_finalize();
```

7.1.5.3 オブジェクトの生成と初期化

FDPS の初期化に成功した場合、ユーザーはコード中で用いるオブジェクトを作成する必要がある。本節では、オブジェクトの生成/初期化の仕方について、解説する。

7.1.5.3.1 粒子群オブジェクトの生成と初期化

本サンプルコードでは、N 体粒子と SPH 粒子のデータを異なる粒子群オブジェクトを用いて管理する。 2 つの整数 psys_num_nbody と psys_num_sph は、それぞれ、N 体粒子と SPH 粒子の粒子群オブジェクトの識別番号を格納する変数である。これら 2 つの整数を使い、粒子群オブジェクトを生成・初期化を以下のように行っている。

Listing 67: 粒子群オブジェクトの生成・初期化

```
1 fdps_create_psys(&psys_num_nbody,"fp_nbody");
2 fdps_init_psys(psys_num_nbody);
3 fdps_create_psys(&psys_num_sph,"fp_sph");
4 fdps_init_psys(psys_num_sph);
```

7.1.5.3.2 領域情報オブジェクトの生成と初期化

本サンプルコードでは、計算領域の分割を、N 体粒子と SPH 粒子を合わせた粒子全体が等分割されるように行うこととする。この場合、必要な領域情報オブジェクトは 1 つである。したがって、本コードでは領域情報オブジェクトの識別番号を格納する整数変数 dinfo_num を用意し、それを用いて生成・初期化を次のように行っている。

Listing 68: 領域情報オブジェクトの生成・初期化

```
fdps_create_dinfo(&dinfo_num);
fdps_init_dinfo(dinfo_num,coef_ema);
```

7.1.5.3.3 ツリーオブジェクトの生成と初期化

本サンプルコードでは、重力計算用、密度計算、圧力勾配加速度計算のそれぞれに 1 つずつツリーを用意している。ツリーオブジェクトの初期化の際には、API fdps_init_tree の第 2 引数に計算で使用する大雑把な粒子数を渡す必要がある。重力計算用のツリーオブジェクト (変数 tree_num_grav を介して制御される) では、ローカル粒子数の 3 倍の値を渡している。一方、密度計算と圧力勾配加速度計算に使用されるツリーオブジェクト (それぞれ変数 tree_num_dens と tree_num_hydro を介して制御される) では、ローカルの SPH 粒子数の 3 倍の値を渡している。

Listing 69: ツリーオブジェクトの生成・初期化

```
    // Make three tree structures
    int nptcl_loc_sph = 1;
    if (fdps_get_nptcl_loc(psys_num_sph) > 1)
        nptcl_loc_sph = fdps_get_nptcl_loc(psys_num_sph);
    int nptcl_loc_nbody = fdps_get_nptcl_loc(psys_num_nbody);
    int nptcl_loc_all   = nptcl_loc_nbody + nptcl_loc_sph;
    // tree for gravity calculation
    int tree_num_grav;
    fdps_create_tree(&tree_num_grav,
                     "Long,force_grav,ep_grav,ep_grav,Monopole");
    const float theta=0.5;
    const int n_leaf_limit=8, n_group_limit=64;
    fdps_init_tree(tree_num_grav, 3*nptcl_loc_all, theta,
                   n_leaf_limit, n_group_limit);
    // tree for the density calculation
    int tree_num_dens;
    fdps_create_tree(&tree_num_dens,
                     "Short,force_dens,ep_hydro,ep_hydro,Gather");
    fdps_init_tree(tree_num_dens, 3*nptcl_loc_sph, theta,
                   n_leaf_limit, n_group_limit);
    // tree for the hydrodynamic force calculation
    int tree_num_hydro;
    fdps_create_tree(&tree_num_hydro,
                     "Short,force_hydro,ep_hydro,ep_hydro,Symmetry");
    fdps_init_tree(tree_num_hydro, 3*nptcl_loc_sph, theta,
                   n_leaf_limit, n_group_limit);
```

7.1.5.4 初期条件の設定

初期条件の設定は void 関数 setup_IC で行われる。この void 関数はマクロ INITIAL_CONDITION の値に応じて、内部でさらに別の void 関数を呼び出しており、呼び出される void 関数とマクロの値の対応は、既に述べた通りである。引数の time_dump, dt_dump, time_end は、データ出力の最初の時刻、出力時間間隔、シミュレーション終了時間を表す変数であり、

個々の初期条件作成関数の中で設定すべきものである。また、境界条件、重力ソフトニングの値 (eps_grav)、系に許される最大の時間刻み (dt_max) も設定する必要がある (dt_max に関しては必ずしも設定する必要はない)。

Listing 70: 初期条件の設定

```
1 setup_IC(psys_num_nbody, psys_num_sph, dinfo_num,
2          &time_dump, &dt_dump, &time_end);
```

以下、円盤銀河の初期条件を設定する void 関数 galaxy_IC について、留意事項を述べておく。

- MAGI が作成する粒子データは MAGI のコード内単位系で出力される。単位系の情報は MAGI を実行したときに出力されるファイル doc/unit.txt に記述されている。このファイルに記載された単位質量、単位長さ、単位時間の値と、定数 magi_unit_mass, magi_unit_leng, magi_unit_time は一致させなければならない。
- 関数が読み込むファイルは ./magi_data/dat/Galaxy.tipsy である。別なファイルを読み込ませたい場合、手動でソースコードを変更する必要がある。
- 関数が生成するガス分布は R ($\equiv \sqrt{x^2+y^2}$) 方向と z 方向に exponential な密度分布を持つガス円盤である。それぞれの方向のスケール長が変数 Rs, zd で、分布を打ち切る距離は変数 Rt, zt である。
- 初期のガスの熱力学的状態はガス温度 temp と水素原子に対する平均分子量 mu を与えて指定する。コンパイル時マクロ USE_ENTROPY が定義済み/未定義に関わらず、粒子の熱力学的状態は単位質量あたりの内部エネルギーとして与える必要がある (fp_sph のメンバ変数 eng)。USE_ENTROPY が定義済みの場合、メイン関数 c_main() で呼び出されている void 関数 set_entropy によって、計算された密度と内部エネルギーの初期値から初期エントロピーが自動的に決定される。未定義の場合、ここで設定した eng の値がそのまま内部エネルギーの初期値となる。

7.1.5.5 領域分割の実行

複数の粒子種がある場合に、これらを合わせた粒子分布に基づいて領域分割を実行するには、領域情報オブジェクト用の 2 つの API fdps_collect_sample_particle と fdps_decompose_domain を併用する必要がある。まず、API fdps_collect_sample_particle でそれぞれの粒子群オブジェクトからサンプル粒子を集める。**このとき、2種類目以降の粒子種に対する呼び出しでは、第 3 引数に false を指定する必要がある。この指定がないと、1種類目の粒子群オブジェクトの情報がクリアされてしまうからである**。すべての粒子群オブジェクトに対して、この API の呼び出しが終わったら、API fdps_decompose_domain で領域分割を実行する。

Listing 71: 領域分割の実行

```
1 fdps_collect_sample_particle(dinfo_num, psys_num_nbody, true, -1.0);
```

```
2 fdps_collect_sample_particle(dinfo_num, psys_num_sph, false, -1.0);
3 fdps_decompose_domain(dinfo_num);
```

7.1.5.6 粒子交換の実行

先程計算した領域情報に基いてプロセス間の粒子の情報を交換するには、粒子群オブジェクト用 API `fdps_exchange_particle` を使用する:

Listing 72: 粒子交換の実行

```
1 fdps_exchange_particle(psys_num_nbody,dinfo_num);
2 fdps_exchange_particle(psys_num_sph,dinfo_num);
```

7.1.5.7 相互作用計算の実行

領域分割・粒子交換が完了したら、計算開始時の加速度を決定するため、相互作用計算を行う必要がある。以下に、本サンプルコードにおける初期条件作成後最初の相互作用計算の実装を示す。最初に重力計算をし、その後、密度計算・圧力勾配加速度計算を行っている。

Listing 73: 相互作用計算の実行

```
1     // Gravity calculation
2     double t_start = fdps_get_wtime();
3 #if defined(ENABLE_GRAVITY_INTERACT)
4     fdps_set_particle_local_tree(tree_num_grav, psys_num_nbody, true);
5     fdps_set_particle_local_tree(tree_num_grav, psys_num_sph, false);
6     fdps_calc_force_making_tree(tree_num_grav,
7                                 calc_gravity_ep_ep,
8                                 calc_gravity_ep_sp,
9                                 dinfo_num,
10                                true);
11    nptcl_loc_nbody = fdps_get_nptcl_loc(psys_num_nbody);
12    FP_nbody *ptcl_nbody = (FP_nbody *) fdps_get_psys_cptr(psys_num_nbody)
          ;
13    for (i = 0; i < nptcl_loc_nbody; i++) {
14        Force_grav f_grav;
15        void *pforce = (void *) &f_grav;
16        fdps_get_force(tree_num_grav, i, pforce);
17        ptcl_nbody[i].acc.x = f_grav.acc.x;
18        ptcl_nbody[i].acc.y = f_grav.acc.y;
19        ptcl_nbody[i].acc.z = f_grav.acc.z;
20        ptcl_nbody[i].pot   = f_grav.pot;
21    }
22    int offset = nptcl_loc_nbody;
23    nptcl_loc_sph = fdps_get_nptcl_loc(psys_num_sph);
24    FP_sph *ptcl_sph = (FP_sph *) fdps_get_psys_cptr(psys_num_sph);
25    for (i = 0; i < nptcl_loc_sph; i++) {
26        Force_grav f_grav;
27        fdps_get_force(tree_num_grav, i + offset, (void *)&f_grav);
28        ptcl_sph[i].acc_grav.x = f_grav.acc.x;
29        ptcl_sph[i].acc_grav.y = f_grav.acc.y;
```

```
30          ptcl_sph[i].acc_grav.z = f_grav.acc.z;
31          ptcl_sph[i].pot_grav   = f_grav.pot;
32      }
33  #endif
34      double t_grav = fdps_get_wtime() - t_start;
35      // SPH calculations
36      t_start = fdps_get_wtime();
37  #if defined(ENABLE_HYDRO_INTERACT)
38      calc_density_wrapper(psys_num_sph, dinfo_num, tree_num_dens);
39      set_entropy(psys_num_sph);
40      set_pressure(psys_num_sph);
41      fdps_calc_force_all_and_write_back(tree_num_hydro,
42                                         calc_hydro_force,
43                                         NULL,
44                                         psys_num_sph,
45                                         dinfo_num,
46                                         true,
47                                         FDPS_MAKE_LIST);
48  #endif
49      double t_hydro = fdps_get_wtime() - t_start;
```

まず重力計算の方法について説明する。重力計算は、N 体粒子と SPH 粒子の両方が関わる。このような複数の粒子種の間で 1 つの相互作用計算を行うには、ツリーオブジェクト用の API `fdps_set_particle_local_tree` と `fdps_calc_force_making_tree` を合わせて使用する必要がある。まず、各粒子群オブジェクトに対して、API `fdps_set_particle_local_tree` を使って、粒子情報をツリーオブジェクトに渡す。**このとき、2種類目以降の粒子群オブジェクトに対する呼び出しでは、第 3 引数に `false` を指定する必要がある。この指定が無いと、これまでツリーオブジェクトに渡した粒子情報がクリアされてしまうからである。**重力計算に関係するすべての粒子群オブジェクトに対して、この API の呼び出しが完了したら、API `fdps_calc_force_making_tree` で相互作用計算を行う。相互作用計算の結果を取得するためには、API `fdps_get_force` を使う。この API は引数に整数 i を取り、API `fdps_set_particle_local_tree` で i 番目に読み込んだ粒子が受ける相互作用を返す。したがって、2 種類目以降の粒子種の相互作用の結果を取得する場合、適切にオフセット値を指定する必要があることに注意されたい。

次に密度計算と圧力勾配加速度計算について説明する。これらの計算は 1 粒子種しか関わらないため、本チュートリアルでこれまで使ってきた API `fdps_calc_force_all_and_write_back` が使用できる。圧力勾配加速度に関しては、`void` 関数 `c_main` 内でこの API を直接呼び出している。一方、密度計算は、第 7.1.4 節でも述べた通り、ρ_i と h_i のイテレーション計算が収束しなかったときのための対処が必要であり、これを `void` 関数 `calc_density_wrapper` の中で行っている。実装は次のようになっている。実装はマクロ `ENABLE_VARIABLE_SMOOTHING_LENGTH` が定義済みか未定義かで分岐しており、未定義の場合には固定長カーネルの SPH コードとなるので、単に、API `fdps_calc_force_all_and_write_back` を 1 回だけ実行している。一方、上記マクロが定義済みの場合、すべての粒子の ρ_i と h_i が無矛盾に決定されるまで、API `fdps_calc_force_all_and_write_back` を繰り返し実行する。各粒子が収束したかの情報は構造体 `fp_sph` のメンバ変数 `flag` に格納されており、値が 1 のときに収束していることを示す。`flag` が 1 を取る粒子数が全 SPH 粒子数に一致したときに計算を

終わらせている。

Listing 74: void 関数 calc_density_wrapper の実装

```
1  void calc_density_wrapper(int psys_num,
2                            int dinfo_num,
3                            int tree_num) {
4  #if defined(ENABLE_VARIABLE_SMOOTHING_LENGTH)
5     int nptcl_loc = fdps_get_nptcl_loc(psys_num);
6     int nptcl_glb = fdps_get_nptcl_glb(psys_num);
7     FP_sph *ptcl = (FP_sph *) fdps_get_psys_cptr(psys_num);
8     // Determine the density and the smoothing length
9     // so that Eq.(6) in Springel (2005) holds within a specified accuracy.
10    for (;;) {
11        // Increase smoothing length
12        int i;
13        for (i = 0; i < nptcl_loc; i++) ptcl[i].smth *= SCF_smth;
14        // Compute density, etc.
15        fdps_calc_force_all_and_write_back(tree_num,
16                                           calc_density,
17                                           NULL,
18                                           psys_num,
19                                           dinfo_num,
20                                           true,
21                                           FDPS_MAKE_LIST);
22        // Check convergence
23        int n_compl_loc = 0;
24        for (i = 0; i < nptcl_loc; i++)
25            if (ptcl[i].flag == 1) n_compl_loc++;
26        int n_compl = fdps_get_sum_s32(n_compl_loc);
27        if (n_compl == nptcl_glb) break;
28    }
29 #else
30    fdps_calc_force_all_and_write_back(tree_num,
31                                       calc_density,
32                                       NULL,
33                                       psys_num,
34                                       dinfo_num,
35                                       true,
36                                       FDPS_MAKE_LIST);
37 #endif
38 }
```

void 関数 set_entropy は、初期条件作成後 1 回だけ呼び出される void 関数で、エントロピーの初期値をセットする。式 (8) から、エントロピーを計算するには初期密度が必要である。そのため、void 関数 calc_density_wrapper の後に配置されている。void 関数 set_entropy では、計算された密度と u_i の初期値を使って、エントロピーをセットする。これ以降は、エントロピーが独立変数となる。

7.1.5.8 時間積分ループ

本サンプルコードでは、時間積分を Leapfrog 時間積分法によって行っている (この方法に関しては、第 4.1.3.5.4 節を参照されたい)。粒子位置を時間推進する $D(\cdot)$ オペレータは void 関

数 full_drift、粒子速度を時間推進する $K(\cdot)$ オペレータは void 関数 initial_kick, final_kick として実装されている。

8 ユーザーサポート

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

8.1 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)

8.2 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)

8.3 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。

9 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. (PASJ, 68, 54)、Namekata et al. (PASJ, 70, 70) の引用をお願いします。

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

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