FDPS Fortran 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, 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

- 2016/12/22
  - 作成および初期リリース (FDPS バージョン 3.0 として)
- 2018/07/11
  - 第 4 節の以下の記述の修正・改善
    - ユーザ定義型のソースコードの一部が端切れしていた (第 4.1 節, 第 4.2 節)
    - 一部のディレクトリ名に誤植
- 2018/08/29
  - N 体/SPH サンプルコードの解説を追加 (第 7.1 節)
- 2018/08/31
  - x86 用 Phantom-GRAPE ライブラリの節を追加 (第 3.4.1.7 節)

2 概要

本節では、Framework for Developing Particle Simulator (FDPS) および FDPS Fortran 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、 Fortran 2003 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 Fortran 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)
- Fortran コンパイラ (Fortran 2003 標準をサポートし、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 Fortran コンパイラ (Fortran 2003 標準をサポートし、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 Fortran コンパイラ (Fortran 2003 標準をサポートし、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 Fortran コンパイラ (Fortran 2003 標準をサポートし、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 FDPS Fortran 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。

Fortran の場合、コンパイル前に Fortran 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 Fortran 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`$(FDPS)/sample/fortran/nbody`に移動。これ以後、ディレクトリ`$(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/fortran/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2つのMakefileがある。1つはGCC用に書かれた`Makefile`であり、もう1つはIntelコンパイラ用に書かれた`Makefile.intel`である。ここでは`Makefile`について詳しく解説し、`Makefile.intel`に関しては使用上の注意点を本節最後で述べるのみとする。

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、Fortranコンパイラを表す`FC`、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`FCFLAGS`, `CXXFLAGS`である。これらの初期設定値は次のようになっている:

```
FC=gfortran
CXX=g++
FCFLAGS = -std=f2003 -O3 -ffast-math -funroll-loops -finline-functions
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 FC に Fortran 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 FC に OpenMP 対応の Fortran 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
  - FCFLAGS += -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 FC に MPI 対応の Fortran 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
  - FCFLAGS += -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 FC に MPI 対応の Fortran 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
  - FCFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp の行のコメントアウトを外す
  - FCFLAGS += -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, SRC_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 Fortran 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 SRC_USER_DEFINED_TYPE, SRC_USER には、それぞれ、ユーザ定義型が記述された Fortran ファイル名と、ユーザ定義型以外の部分が記述された Fortran ファイル名を格納する。FDPS の Fortran インターフェースコードはユーザーコードのクラス (派生型) 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 SRC_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、SRC_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。

最後に、Makefile.intel を使用する上での注意点について説明する。変数の初期値が異なる点を除き、Makefile.intel の構造は Makefile と同じである。したがって、変数の値をユーザが利用する計算機システムにおける値に適切に設定すれば、Makefile と同様に利用可能である。以下に変更する上での注意点を述べる:

- /opt/intel/bin を、利用する計算機システムにおける Intel コンパイラの格納ディレクトリの PATH に変更する。

- /opt/intel/include を、Intel コンパイラに付属するヘッダファイル群を格納したディレクトリの PATH に変更する。

- Makefile.intel の LDFLAGS は、-L/opt/intel/lib/intel64 -L/usr/lib64 -lifport -lifcore -limf -lsvml -lm -lipgo -lirc -lirc_s となっている。
  この中の -lifcore [注 2)] は、C++コンパイラで C++オブジェクトと Fortran オブジェクトをリンクするため必要である[注 3)]。計算機システムのライブラリパスに、Intel コンパイラのライブラリ群が登録されていない場合、さらに、-L/opt/intel/lib/intel64 -L/usr/lib64 -lifport -limf -lsvml -lm -lipgo -lirc -lirc_s のような指定が必要である。
  ここで、/opt/intel/lib/intel64 は、Intel コンパイラのライブラリ群が格納されたディレクトリの PATH で、/usr/lib64 はライブラリ libm を格納したディレクトリの PATH である。これらは利用する計算機システムに合わせて修正する必要がある。コンパイルに必要なライブラリ群 (-l*) は、Intel コンパイラのバージョンによって変わる可能性があるので確認して頂きたい。

- 本書を執筆時点 (2016/12/26) で、Intel コンパイラで OpenMP を有効にするオプションは-openmp、或いは、-qopenmp である。これは Intel コンパイラのバージョンによって異なり、より新しいバージョンのコンパイラは後者を使用する (前者を使用した場合、廃止予定の警告が出る)。

- 利用する計算機システムによっては、-lifcore の指定以外の設定が環境変数 (PATH, CPATH, LD_LIBRARY_PATH 等として) で既に行われていることもありえる。

注 2) libifcore は、Fortran ランタイムライブラリである。
注 3) Intel コンパイラ (バージョン 17.0.0 20160721) において確認。

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 Fortran 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:    9.5000000000E+000, energy error:   -3.8046534069E-003
time:    9.6250000000E+000, energy error:   -3.9711750200E-003
time:    9.7500000000E+000, energy error:   -3.8223429428E-003
time:    9.8750000000E+000, energy error:   -3.8843099298E-003
******** FDPS has successfully finished. ********
```

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。

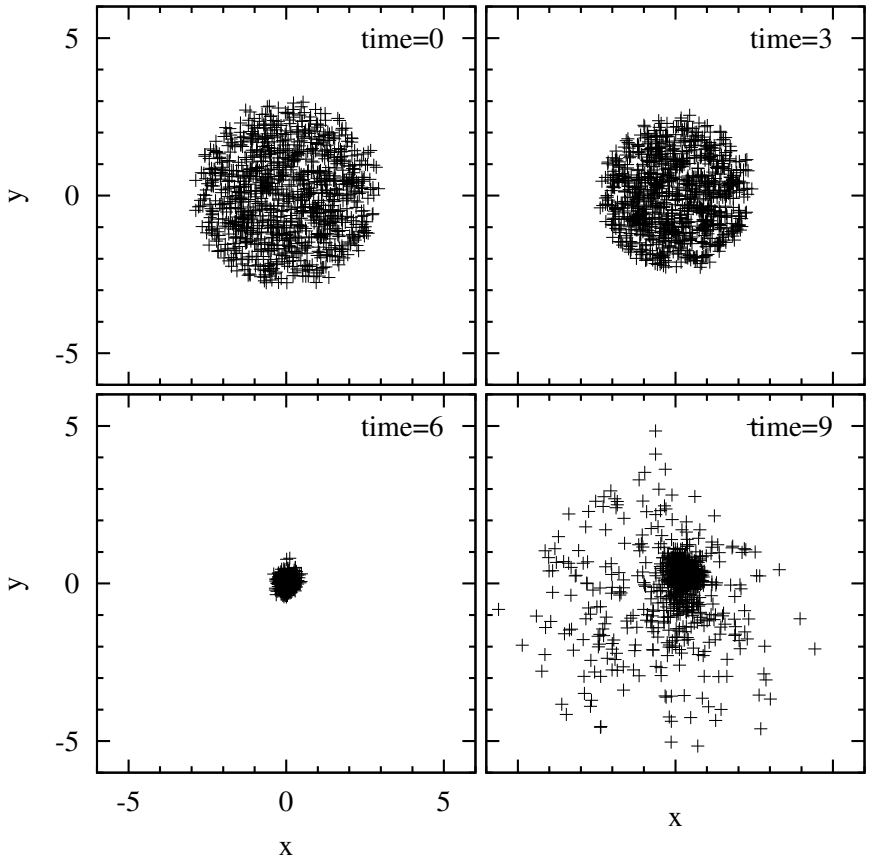

図 1:

```
$ 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 `f_main.F90` の中のサブルーチン `f_main()` のパラメータ変数 `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/fortran/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。

3.4.2 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/fortran/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/fortran/sph` に移動する。

3.4.2.3 Makefile の編集

SPH サンプルコードにも、N 体計算のサンプルコードの場合と同様、GCC と Intel コンパイラ用に 2 種類の Makefile が用意されている。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 Fortran 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。

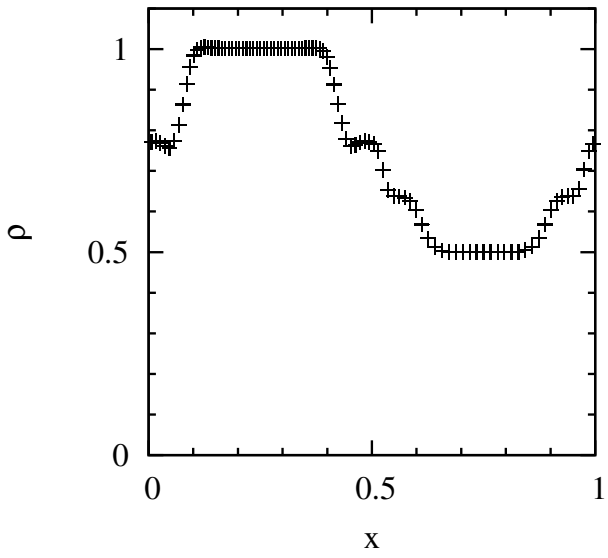

図 2: 衝撃波管問題の時刻 $t = 40$ における密度分布

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派生データ型 (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/fortran/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.F90` と、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 `f_main.F90` から構成される。この他に、GCC と Intel コンパイラ用の Makefile である `Makefile` と `Makefile.intel` 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派生データ型と サブルーチンについて記述する。

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.F90` を参照)。

Listing 1: FullParticle 型

```
1    type, public, bind(c) :: full_particle !$fdps FP,EPI,EPJ,Force
2       !$fdps copyFromForce full_particle (pot,pot) (acc,acc)
3       !$fdps copyFromFP full_particle (id,id) (mass,mass) (pos,pos)
4       !$fdps clear id=keep, mass=keep, pos=keep, vel=keep
5       integer(kind=c_long_long) :: id
6       real(kind=c_double)  mass !$fdps charge
7       type(fdps_f64vec) :: pos !$fdps position
8       type(fdps_f64vec) :: vel !$fdps velocity
9       real(kind=c_double) :: pot
10      type(fdps_f64vec) :: acc
11   end type full_particle
```

FDPS Fortran 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、ユーザは派生データ型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、派生データ型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、派生データ型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:

```
type, public, bind(c) :: 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:

```
real(kind=c_double) :: mass !$fdps charge
type(fdps_f64vec) :: pos !$fdps position
type(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。サブルーチン 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.F90 を参照)。

Listing 2: 関数 calcForceEpEp

```
1    subroutine calc_gravity_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2       implicit none
3       integer(c_int), intent(in), value :: n_ip,n_jp
4       type(full_particle), dimension(n_ip), intent(in) :: ep_i
5       type(full_particle), dimension(n_jp), intent(in) :: ep_j
6       type(full_particle), dimension(n_ip), intent(inout) :: f
7       !* Local variables
8       integer(c_int) :: i,j
9       real(c_double) :: eps2,poti,r3_inv,r_inv
10      type(fdps_f64vec) :: xi,ai,rij
11
12      !* Compute force
13      eps2 = eps_grav * eps_grav
14      do i=1,n_ip
15         xi = ep_i(i)%pos
16         ai = 0.0d0
17         poti = 0.0d0
18         do j=1,n_jp
19            rij%x  = xi%x - ep_j(j)%pos%x
20            rij%y  = xi%y - ep_j(j)%pos%y
21            rij%z  = xi%z - ep_j(j)%pos%z
22            r3_inv = rij%x*rij%x &
23                   + rij%y*rij%y &
24                   + rij%z*rij%z &
25                   + eps2
26            r_inv  = 1.0d0/sqrt(r3_inv)
27            r3_inv = r_inv * r_inv
28            r_inv  = r_inv * ep_j(j)%mass
29            r3_inv = r3_inv * r_inv
30            ai%x   = ai%x - r3_inv * rij%x
31            ai%y   = ai%y - r3_inv * rij%y
32            ai%z   = ai%z - r3_inv * rij%z
33            poti   = poti - r_inv
34            ! [IMPORTANT NOTE]
35            !   In the innermost loop, we use the components of vectors
36            !   directly for vector operations because of the following
37            !   reasion. Except for intel compilers with `-ipo` option,
38            !   most of Fortran compilers use function calls to perform
39            !   vector operations like rij = x - ep_j(j)%pos.
40            !   This significantly slow downs the speed of the code.
```

```
41              !   By using the components of vector directly, we can avoid
42              !   these function calls.
43           end do
44           f(i)%pot = f(i)%pot + poti
45           f(i)%acc = f(i)%acc + ai
46        end do
47
48     end subroutine calc_gravity_ep_ep
```

本サンプルコードでは、サブルーチン calc_gravity_ep_ep として実装されている。サブルーチンの仮引数は、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、サブルーチン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.F90 を参照)。

Listing 3: 関数 calcForceEpSp

```
1     subroutine calc_gravity_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2        implicit none
3        integer(c_int), intent(in), value :: n_ip,n_jp
4        type(full_particle), dimension(n_ip), intent(in) :: ep_i
5        type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
6        type(full_particle), dimension(n_ip), intent(inout) :: f
7        !* Local variables
8        integer(c_int) :: i,j
9        real(c_double) :: eps2,poti,r3_inv,r_inv
10       type(fdps_f64vec) :: xi,ai,rij
11
12       eps2 = eps_grav * eps_grav
13       do i=1,n_ip
14          xi = ep_i(i)%pos
15          ai = 0.0d0
16          poti = 0.0d0
17          do j=1,n_jp
18             rij%x  = xi%x - ep_j(j)%pos%x
19             rij%y  = xi%y - ep_j(j)%pos%y
20             rij%z  = xi%z - ep_j(j)%pos%z
21             r3_inv = rij%x*rij%x &
22                    + rij%y*rij%y &
23                    + rij%z*rij%z &
24                    + eps2
25             r_inv  = 1.0d0/sqrt(r3_inv)
26             r3_inv = r_inv * r_inv
27             r_inv  = r_inv * ep_j(j)%mass
28             r3_inv = r3_inv * r_inv
```

```
29              ai%x    = ai%x - r3_inv * rij%x
30              ai%y    = ai%y - r3_inv * rij%y
31              ai%z    = ai%z - r3_inv * rij%z
32              poti    = poti - r_inv
33           end do
34           f(i)%pot = f(i)%pot + poti
35           f(i)%acc = f(i)%acc + ai
36        end do
37
38     end subroutine calc_gravity_ep_sp
```

本サンプルコードでは、サブルーチン calc_gravity_ep_sp として実装されている。サブルーチンの仮引数は、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 Fortran インターフェースを用いて N 体計算を行うにあたり、“メインルーチン” f_main() に書かれるべきサブルーチンや関数に関して解説する。ここで、メインルーチンとはっきり書かないのは、次の理由による: FDPS Fortran インターフェースを使用する場合、ユーザコードは必ずサブルーチン f_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ルーチンを持たない (メインルーチン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、実質的にはサブルーチン f_main() がメインルーチンの役割を果たす。そのため、敢えて “メインルーチン” という言葉を使った。メインルーチンという言葉は、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、f_main() をメインルーチンと呼ぶことにする。本サンプルコードのメインルーチンは f_main.F90 に記述されている。

4.1.3.1 fdps_controller 型オブジェクトの生成

FDPS Fortran インターフェースにおいて、FDPS の API はすべて Fortran 2003 のクラス FDPS_controller のメンバ関数として提供される。このクラスは、インターフェースプログラムの 1 つである FDPS_module.F90 の中の、モジュール fdps_module 内で定義されている。したがって、ユーザは FDPS の API を使用するために、FDPS_controller 型オブジェクトを生成しなければならない。本サンプルコードでは、FDPS_controller 型オブジェクト fdps_ctrl をメインルーチンで生成している:

Listing 4: fdps_controller 型オブジェクトの生成

```
1 subroutine f_main()
2    use fdps_module
3    implicit none
4    !* Local variables
```

```
5     type(fdps_controller) :: fdps_ctrl
6
7     ! Do something
8
9  end subroutine f_main
```

ここに示したコードは実際にサンプルコードから必要な部分だけを取り出したものであることに注意して頂きたい。

上記の理由から、以下の説明において、FDPS の API はこのオブジェクトのメンバ関数として呼び出されていることに注意されたい。

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、メインルーチンに記述する。

Listing 5: FDPS の開始

```
1  call fdps_ctrl%ps_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、メインルーチンの最後に次のように記述する。

Listing 6: FDPS の終了

```
1  call fdps_ctrl%ps_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。Fortran 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 `f_main.F90` のメインルーチン内に記述されている。

Listing 7: オブジェクトの生成

```
1  subroutine f_main()
2     use fdps_module
3     use user_defined_types
4     implicit none
5     !* Local variables
6     integer :: psys_num,dinfo_num,tree_num
```

```
7
8     !* Create FDPS objects
9     call fdps_ctrl%create_dinfo(dinfo_num)
10    call fdps_ctrl%create_psys(psys_num,'full_particle')
11    call fdps_ctrl%create_tree(tree_num, &
12                               "Long,full_particle,full_particle,
                                       full_particle,Monopole")
13
14  end subroutine f_main
```

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 型に対応する派生データ型名を文字列として 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、**派生データ型 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 `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  call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
```

ここで、API `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 `init_psys` で行う:

Listing 9: 粒子群オブジェクトの初期化

```
1  call fdps_ctrl%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 `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  call fdps_ctrl%init_tree(tree_num,ntot,theta, &
```

```
2                                   n_leaf_limit,n_group_limit)
```

この API には 3 つの省略可能引数が存在し、サンプルコードではこれらを省略せずに指定している:

- 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 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 set_nptcl_loc と get_psys_fptr を用いて、次のように行う:

Listing 11: 粒子データの初期化

```
1  subroutine foo(fdps_ctrl,psys_num)
2     use fdps_vector
3     use fdps_module
4     use user_defined_types
5     implicit none
6     type(fdps_controller), intent(IN) :: fdps_ctrl
7     integer, intent(IN) :: psys_num
8     !* Local variables
9     integer :: i,nptcl_loc
10    type(full_particle), dimension(:), pointer :: ptcl
11
12    !* Set # of local particles
13    call fdps_ctrl%set_nptcl_loc(psys_num,nptcl_loc)
14
15    !* Get the pointer to full particle data
16    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
17
18    !* Initialize particle data
19    do i=1,nptcl_loc
20       ptcl(i)%pos = ! Do something
21    end do
22
23    !* Release the pointer
24    nullify(ptcl)
25
26 end subroutine foo
```

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 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 get_psys_fptr を使用す

る。 アドレスは Fortran ポインタで受け取る必要がある。そのため、上記の例では、ポインタを以下のように用意している:

```
type(full_particle), dimension(:), pointer :: ptcl
```

API get_psys_fptr 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。上の例では、粒子データの設定が完了した後、ポインタを組込関数 nullify によって解放している。

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 decompose_domain_all を用いて行っている:

Listing 12: 領域分割の実行

```
1 if (mod(num_loop,4) == 0) then
2    call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
3 end if
```

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。

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 exchange_particle を用いる:

Listing 13: 粒子交換の実行

```
1 call fdps_ctrl%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 calc_force_all_and_write_back を用いる:

Listing 14: 相互作用計算の実行

```
1 subroutien f_main()
2    use, intrinsic :: iso_c_binding
3    use user_defined_types
4    implicit none
5    !* Local variables
6    type(c_funptr) :: pfunc_ep_ep,pfunc_ep_sp
```

```
7
8     ! Do somehting
9
10    pfunc_ep_ep = c_funloc(calc_gravity_ep_ep)
11    pfunc_ep_sp = c_funloc(calc_gravity_ep_sp)
12    call fdps_ctrl%calc_force_all_and_write_back(tree_num,     &
13                                                 pfunc_ep_ep, &
14                                                 pfunc_ep_sp, &
15                                                 psys_num,    &
16                                                 dinfo_num)
17
18    ! Do something
19
20 end subroutine f_main
```

ここで、API の第 2,3 引数には関数 calcForceEpEp, calcForceEpSp の (C 言語アドレス[注 4)]としての) 関数ポインタを指定する。関数の C 言語アドレスは、Fortran 2003 で導入された組込関数 `c_funloc` を使って取得する (この組込み関数はモジュール `iso_c_binding` で提供されるため、`use` 文を使い、このモジュールを利用可能にしている)。C 言語アドレスを格納するためには、同じく Fortran 2003 で導入された派生データ型 `c_funptr` の変数が必要である。そのため、本サンプルコードでは、`c_funptr` 型変数として、`pfunc_ep_ep` と `pfunc_ep_sp` を用意している。ここに、`calc_gravity_ep_ep` と `calc_gravity_ep_sp` の C 言語アドレスを格納した上で、API に渡している。

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、サブルーチン `kick` とサブルーチン `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 call kick(fdps_ctrl,psys_num,0.5d0*dt)
3 time_sys = time_sys + dt
4 call drift(fdps_ctrl,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 call kick(fdps_ctrl,psys_num,0.5d0*dt)
```

[注 4)] C 言語方式で記述されたアドレス情報のこと。

4.1.3.6 粒子データの更新

上記で説明した kick や drift 等のサブルーチンで、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  subroutine foo(fdps_ctrl,psys_num)
2     use fdps_vector
3     use fdps_module
4     use user_defined_types
5     implicit none
6     type(fdps_controller), intent(IN) :: fdps_ctrl
7     integer, intent(IN) :: psys_num
8     !* Local variables
9     integer :: i,nptcl_loc
10    type(full_particle), dimension(:), pointer :: ptcl
11
12    !* Get # of local particles
13    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
14
15    !* Get the pointer to full particle data
16    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
17
18    !* Initialize or update particle data
19    do i=1,nptcl_loc
20       ptcl(i)%pos = ! Do something
21    end do
22
23    !* Release the pointer
24    nullify(ptcl)
25
26 end subroutine foo
```

API get_psys_fptr 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/fortran/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.F90` と、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 `f_main.F90` から構成される。この他に、GCC と Intel コンパイラ用の Makefile である `Makefile` と `Makefile.intel` 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派生データ型とサブルーチンについて記述する。

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.F90` を参照)。

Listing 19: FullParticle 型

```
1    !**** Full particle type
2    type, public, bind(c) :: full_particle !$fdps FP
3       !$fdps copyFromForce force_dens (dens,dens)
4       !$fdps copyFromForce force_hydro (acc,acc) (eng_dot,eng_dot) (dt,dt)
5       real(kind=c_double) :: mass !$fdps charge
6       type(fdps_f64vec) :: pos !$fdps position
7       type(fdps_f64vec) :: vel
8       type(fdps_f64vec) :: acc
9       real(kind=c_double) :: dens
10      real(kind=c_double) :: eng
11      real(kind=c_double) :: pres
12      real(kind=c_double) :: smth !$fdps rsearch
13      real(kind=c_double) :: snds
14      real(kind=c_double) :: eng_dot
15      real(kind=c_double) :: dt
16      integer(kind=c_long_long) :: id
17      type(fdps_f64vec) :: vel_half
18      real(kind=c_double) :: eng_half
19   end type 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、この派生データ型が FullParticle 型であることを示すため、次の指示文を記述している:

```
type, public, bind(c) :: 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:

```
real(kind=c_double) :: mass !$fdps charge
type(fdps_f64vec) :: pos !$fdps position
real(kind=c_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.F90` 参照):

Listing 20: EssentialParticleI 型

```
1    !**** Essential particle type
2    type, public, bind(c) :: essential_particle !$fdps EPI,EPJ
3       !$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,
              mass) (smth,smth) (dens,dens) (pres,pres) (snds,snds)
4       integer(kind=c_long_long) :: id !$fdps id
5       type(fdps_f64vec) :: pos !$fdps position
6       type(fdps_f64vec) :: vel
7       real(kind=c_double) :: mass !$fdps charge
8       real(kind=c_double) :: smth !$fdps rsearch
9       real(kind=c_double) :: dens
10      real(kind=c_double) :: pres
11      real(kind=c_double) :: snds
```

```
12    end type essential_particle
```

まず、ユーザは指示文を用いて、この派生データ型が 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:

```
type, public, bind(c) :: essential_particle !$fdps EPI,EPJ
```

次に、ユーザはこの派生データ型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:

```
type(fdps_f64vec) :: pos !$fdps position
real(kind=c_double) :: mass !$fdps charge
real(kind=c_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     !**** Force types
2     type, public, bind(c) :: force_dens !$fdps Force
3        !$fdps clear smth=keep
4        real(kind=c_double) :: dens
5        real(kind=c_double) :: smth
6     end type force_dens
7
8     type, public, bind(c) :: force_hydro !$fdps Force
9        !$fdps clear
10       type(fdps_f64vec) :: acc
11       real(kind=c_double) :: eng_dot
12       real(kind=c_double) :: dt
13    end type force_hydro
```

まず、ユーザはこれらの派生データ型が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、それぞれの派生データ型に対して、以下のように記述している:

```
type, public, bind(c) :: force_dens !$fdps Force
type, public, bind(c) :: force_hydro !$fdps Force
```

これらの派生データ型は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.F90` を参照)。

Listing 22: 関数 calcForceEpEp

```
1    !**** Interaction function
2    subroutine calc_density(ep_i,n_ip,ep_j,n_jp,f) bind(c)
3       integer(kind=c_int), intent(in), value :: n_ip,n_jp
4       type(essential_particle), dimension(n_ip), intent(in) :: ep_i
5       type(essential_particle), dimension(n_jp), intent(in) :: ep_j
6       type(force_dens), dimension(n_ip), intent(inout) :: f
7       !* Local variables
8       integer(kind=c_int) :: i,j
9       type(fdps_f64vec) :: dr
10
11      do i=1,n_ip
12         f(i)%dens = 0.0d0
13         do j=1,n_jp
14            dr%x = ep_j(j)%pos%x - ep_i(i)%pos%x
15            dr%y = ep_j(j)%pos%y - ep_i(i)%pos%y
16            dr%z = ep_j(j)%pos%z - ep_i(i)%pos%z
```

```
17              f(i)%dens = f(i)%dens &
18                        + ep_j(j)%mass * W(dr,ep_i(i)%smth)
19           end do
20        end do
21
22     end subroutine calc_density
23
24     !**** Interaction function
25     subroutine calc_hydro_force(ep_i,n_ip,ep_j,n_jp,f) bind(c)
26        integer(kind=c_int), intent(in), value :: n_ip,n_jp
27        type(essential_particle), dimension(n_ip), intent(in) :: ep_i
28        type(essential_particle), dimension(n_jp), intent(in) :: ep_j
29        type(force_hydro), dimension(n_ip), intent(inout) :: f
30        !* Local parameters
31        real(kind=c_double), parameter :: C_CFL=0.3d0
32        !* Local variables
33        integer(kind=c_int) :: i,j
34        real(kind=c_double) :: mass_i,mass_j,smth_i,smth_j, &
35                               dens_i,dens_j,pres_i,pres_j, &
36                               snds_i,snds_j
37        real(kind=c_double) :: povrho2_i,povrho2_j, &
38                               v_sig_max,dr_dv,w_ij,v_sig,AV
39        type(fdps_f64vec) :: pos_i,pos_j,vel_i,vel_j, &
40                             dr,dv,gradW_ij
41
42        do i=1,n_ip
43           !* Zero-clear
44           v_sig_max = 0.0d0
45           !* Extract i-particle info.
46           pos_i = ep_i(i)%pos
47           vel_i = ep_i(i)%vel
48           mass_i  = ep_i(i)%mass
49           smth_i  = ep_i(i)%smth
50           dens_i  = ep_i(i)%dens
51           pres_i  = ep_i(i)%pres
52           snds_i  = ep_i(i)%snds
53           povrho2_i = pres_i/(dens_i*dens_i)
54           do j=1,n_jp
55              !* Extract j-particle info.
56              pos_j%x = ep_j(j)%pos%x
57              pos_j%y = ep_j(j)%pos%y
58              pos_j%z = ep_j(j)%pos%z
59              vel_j%x = ep_j(j)%vel%x
60              vel_j%y = ep_j(j)%vel%y
61              vel_j%z = ep_j(j)%vel%z
62              mass_j  = ep_j(j)%mass
63              smth_j  = ep_j(j)%smth
64              dens_j  = ep_j(j)%dens
65              pres_j  = ep_j(j)%pres
66              snds_j  = ep_j(j)%snds
67              povrho2_j = pres_j/(dens_j*dens_j)
68              !* Compute dr & dv
69              dr%x = pos_i%x - pos_j%x
70              dr%y = pos_i%y - pos_j%y
71              dr%z = pos_i%z - pos_j%z
```

```
72              dv%x = vel_i%x - vel_j%x
73              dv%y = vel_i%y - vel_j%y
74              dv%z = vel_i%z - vel_j%z
75              !* Compute the signal velocity
76              dr_dv = dr%x * dv%x + dr%y * dv%y + dr%z * dv%z
77              if (dr_dv < 0.0d0) then
78                 w_ij = dr_dv / sqrt(dr%x * dr%x + dr%y * dr%y + dr%z * dr%z
                       )
79              else
80                 w_ij = 0.0d0
81              end if
82              v_sig = snds_i + snds_j - 3.0d0 * w_ij
83              v_sig_max = max(v_sig_max, v_sig)
84              !* Compute the artificial viscosity
85              AV = - 0.5d0*v_sig*w_ij / (0.5d0*(dens_i+dens_j))
86              !* Compute the average of the gradients of kernel
87              gradW_ij = 0.5d0 * (gradW(dr,smth_i) + gradW(dr,smth_j))
88              !* Compute the acceleration and the heating rate
89              f(i)%acc%x = f(i)%acc%x - mass_j*(povrho2_i+povrho2_j+AV)*
                       gradW_ij%x
90              f(i)%acc%y = f(i)%acc%y - mass_j*(povrho2_i+povrho2_j+AV)*
                       gradW_ij%y
91              f(i)%acc%z = f(i)%acc%z - mass_j*(povrho2_i+povrho2_j+AV)*
                       gradW_ij%z
92              f(i)%eng_dot = f(i)%eng_dot &
93                           + mass_j * (povrho2_i + 0.5d0*AV) &
94                            *(dv%x * gradW_ij%x &
95                              +dv%y * gradW_ij%y &
96                              +dv%z * gradW_ij%z)
97           end do
98           f(i)%dt = C_CFL*2.0d0*smth_i/(v_sig_max*kernel_support_radius)
99        end do
100       ! [IMPORTANT NOTE]
101       !   In the innermost loop, we use the components of vectors
102       !   directly for vector operations because of the following
103       !   reasion. Except for intel compilers with `-ipo` option,
104       !   most of Fortran compilers use function calls to perform
105       !   vector operations like rij = x - ep_j(j)%pos.
106       !   This significantly slow downs the speed of the code.
107       !   By using the components of vector directly, we can avoid
108       !   these function calls.
109
110    end subroutine calc_hydro_force
```

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、サブルーチンの仮引数は、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、メインルーチン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ルーチンの定義については、第 4.1.3 節を参照のこと)

。

4.2.3.1 fdps_controller 型オブジェクトの生成

ユーザは FDPS の API を使用するために、FDPS_controller 型オブジェクトを生成しなければならない。本サンプルコードでは、FDPS_controller 型オブジェクト fdps_ctrl をメインルーチンで生成している:

Listing 23: fdps_controller 型オブジェクトの生成

```
1 subroutine f_main()
2    use fdps_module
3    implicit none
4    !* Local variables
5    type(fdps_controller) :: fdps_ctrl
6
7    ! Do something
8
9 end subroutine f_main
```

ここに示したコードは実際にサンプルコードから必要な部分だけを取り出したものであることに注意して頂きたい。

上記の理由から、以下の説明において、FDPS の API はこのオブジェクトのメンバ関数として呼び出されていることに注意されたい。

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、メインルーチンに記述する。

Listing 24: FDPS の開始

```
1 call fdps_ctrl%ps_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、メインルーチンの最後に次のように記述する。

Listing 25: FDPS の終了

```
1 call fdps_ctrl%PS_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  subroutine f_main()
2     use fdps_vector
3     use fdps_module
4     use user_defined_types
5     implicit none
6     !* Local variables
7     integer :: psys_num,dinfo_num
8     integer :: dens_tree_num,hydro_tree_num
9
10    !* Create FDPS objects
11    call fdps_ctrl%create_psys(psys_num,'full_particle')
12    call fdps_ctrl%create_dinfo(dinfo_num)
13    call fdps_ctrl%create_tree(dens_tree_num, &
14                               "Short,dens_force,essential_particle,
                                      essential_particle,Gather")
15    call fdps_ctrl%create_tree(hydro_tree_num, &
16                               "Short,hydro_force,essential_particle,
                                      essential_particle,Symmetry")
17
18 end subroutine f_main
```

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 create_psys と 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派生データ型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 call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
2 call fdps_ctrl%set_boundary_condition(dinfo_num,fdps_bc_periodic_xyz)
3 call fdps_ctrl%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 call fdps_ctrl%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 call fdps_ctrl%init_tree(dens_tree_num,3*ntot,theta, &
2                          n_leaf_limit,n_group_limit)
3 call fdps_ctrl%init_tree(hydro_tree_num,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 `decompose_domain_all` を用いる。

Listing 30: 領域分割の実行

```
1 call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
```

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 `exchange_particle` を用いる。

Listing 31: 粒子交換の実行

```
1 call fdps_ctrl%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 `calc_force_all_and_write_back` を用いる。

Listing 32: 相互作用計算の実行

```
1  subroutine f_main()
2     use, intrinsic :: iso_c_binding
3     use user_defined_types
4     implicit none
5     !* Local variables
6     type(c_funptr) :: pfunc_ep_ep
7
8     ! Do something
9
10    pfunc_ep_ep = c_funloc(calc_density)
11    call fdps_ctrl%calc_force_all_and_write_back(tree_num_dens, &
12                                                 pfunc_ep_ep,   &
13                                                 psys_num,      &
14                                                 dinfo_num)
15    call set_pressure(fdps_ctrl,psys_num)
16    pfunc_ep_ep = c_funloc(calc_hydro_force)
17    call fdps_ctrl%calc_force_all_and_write_back(tree_num_hydro, &
18                                                 pfunc_ep_ep,   &
19                                                 psys_num,      &
20                                                 dinfo_num)
21
22    ! Do something
23
24 end subroutine f_main
```

ここで、API の第 2 引数には関数 calcForceEpEp の (C 言語アドレスとしての) 関数ポインタを指定する。

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.F90)

```
!================================
!   MODULE: User defined types
!================================
module user_defined_types
   use, intrinsic :: iso_c_binding
   use fdps_vector
   use fdps_super_particle
   implicit none

   !* Public variables
   real(kind=c_double), public :: eps_grav ! gravitational softening

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

   contains

   !**** Interaction function (particle-particle)
#if defined(ENABLE_PHANTOM_GRAPE_X86)
   subroutine calc_gravity_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
#if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
      use omp_lib
#endif
      use phantom_grape_g5_x86
      implicit none
      integer(c_int), intent(in), value :: n_ip,n_jp
      type(full_particle), dimension(n_ip), intent(in) :: ep_i
      type(full_particle), dimension(n_jp), intent(in) :: ep_j
      type(full_particle), dimension(n_ip), intent(inout) :: f
      !* Local variables
      integer(c_int) :: i,j
      integer(c_int) :: nipipe,njpipe,devid
      real(c_double), dimension(3,n_ip) :: xi,ai
      real(c_double), dimension(n_ip) :: pi
```

```
45        real(c_double), dimension(3,n_jp) :: xj
46        real(c_double), dimension(n_jp) :: mj
47
48        nipipe = n_ip
49        njpipe = n_jp
50        do i=1,n_ip
51           xi(1,i) = ep_i(i)%pos%x
52           xi(2,i) = ep_i(i)%pos%y
53           xi(3,i) = ep_i(i)%pos%z
54           ai(1,i) = 0.0d0
55           ai(2,i) = 0.0d0
56           ai(3,i) = 0.0d0
57           pi(i)   = 0.0d0
58        end do
59        do j=1,n_jp
60           xj(1,j) = ep_j(j)%pos%x
61           xj(2,j) = ep_j(j)%pos%y
62           xj(3,j) = ep_j(j)%pos%z
63           mj(j)   = ep_j(j)%mass
64        end do
65  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
66        devid = omp_get_thread_num()
67        ! [IMPORTANT NOTE]
68        !   The subroutine calc_gravity_pp is called by a OpenMP thread
69        !   in the FDPS. This means that here is already in the parallel
                region.
70        !   So, you can use omp_get_thread_num() without !$OMP parallel
                directives.
71        !   If you use them, a nested parallel resions is made and the
                gravity
72        !   calculation will not be performed correctly.
73  #else
74        devid = 0
75  #endif
76        call g5_set_xmjMC(devid, 0, n_jp, xj, mj)
77        call g5_set_nMC(devid, n_jp)
78        call g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip)
79        do i=1,n_ip
80           f(i)%acc%x = f(i)%acc%x + ai(1,i)
81           f(i)%acc%y = f(i)%acc%y + ai(2,i)
82           f(i)%acc%z = f(i)%acc%z + ai(3,i)
83           f(i)%pot   = f(i)%pot   - pi(i)
84        end do
85     end subroutine calc_gravity_ep_ep
86
87     subroutine calc_gravity_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
88  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
89        use omp_lib
90  #endif
91        use phantom_grape_g5_x86
92        implicit none
93        integer(c_int), intent(in), value :: n_ip,n_jp
94        type(full_particle), dimension(n_ip), intent(in) :: ep_i
95        type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
96        type(full_particle), dimension(n_ip), intent(inout) :: f
```

```
 97        !* Local variables
 98        integer(c_int) :: i,j
 99        integer(c_int) :: nipipe,njpipe,devid
100        real(c_double), dimension(3,n_ip) :: xi,ai
101        real(c_double), dimension(n_ip) :: pi
102        real(c_double), dimension(3,n_jp) :: xj
103        real(c_double), dimension(n_jp) :: mj
104
105        nipipe = n_ip
106        njpipe = n_jp
107        do i=1,n_ip
108           xi(1,i) = ep_i(i)%pos%x
109           xi(2,i) = ep_i(i)%pos%y
110           xi(3,i) = ep_i(i)%pos%z
111           ai(1,i) = 0.0d0
112           ai(2,i) = 0.0d0
113           ai(3,i) = 0.0d0
114           pi(i)   = 0.0d0
115        end do
116        do j=1,n_jp
117           xj(1,j) = ep_j(j)%pos%x
118           xj(2,j) = ep_j(j)%pos%y
119           xj(3,j) = ep_j(j)%pos%z
120           mj(j)   = ep_j(j)%mass
121        end do
122 #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
123        devid = omp_get_thread_num()
124        ! [IMPORTANT NOTE]
125        !   The subroutine calc_gravity_psp is called by a OpenMP thread
126        !   in the FDPS. This means that here is already in the parallel
                region.
127        !   So, you can use omp_get_thread_num() without !$OMP parallel
                directives.
128        !   If you use them, a nested parallel resions is made and the
                gravity
129        !   calculation will not be performed correctly.
130 #else
131        devid = 0
132 #endif
133        call g5_set_xmjMC(devid, 0, n_jp, xj, mj)
134        call g5_set_nMC(devid, n_jp)
135        call g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip)
136        do i=1,n_ip
137           f(i)%acc%x = f(i)%acc%x + ai(1,i)
138           f(i)%acc%y = f(i)%acc%y + ai(2,i)
139           f(i)%acc%z = f(i)%acc%z + ai(3,i)
140           f(i)%pot   = f(i)%pot   - pi(i)
141        end do
142     end subroutine calc_gravity_ep_sp
143 #else
144     subroutine calc_gravity_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
145        implicit none
146        integer(c_int), intent(in), value :: n_ip,n_jp
147        type(full_particle), dimension(n_ip), intent(in) :: ep_i
148        type(full_particle), dimension(n_jp), intent(in) :: ep_j
```

```
149       type(full_particle), dimension(n_ip), intent(inout) :: f
150       !* Local variables
151       integer(c_int) :: i,j
152       real(c_double) :: eps2,poti,r3_inv,r_inv
153       type(fdps_f64vec) :: xi,ai,rij
154
155       !* Compute force
156       eps2 = eps_grav * eps_grav
157       do i=1,n_ip
158          xi = ep_i(i)%pos
159          ai = 0.0d0
160          poti = 0.0d0
161          do j=1,n_jp
162             rij%x  = xi%x - ep_j(j)%pos%x
163             rij%y  = xi%y - ep_j(j)%pos%y
164             rij%z  = xi%z - ep_j(j)%pos%z
165             r3_inv = rij%x*rij%x &
166                    + rij%y*rij%y &
167                    + rij%z*rij%z &
168                    + eps2
169             r_inv  = 1.0d0/sqrt(r3_inv)
170             r3_inv = r_inv * r_inv
171             r_inv  = r_inv * ep_j(j)%mass
172             r3_inv = r3_inv * r_inv
173             ai%x   = ai%x - r3_inv * rij%x
174             ai%y   = ai%y - r3_inv * rij%y
175             ai%z   = ai%z - r3_inv * rij%z
176             poti   = poti - r_inv
177             ! [IMPORTANT NOTE]
178             !   In the innermost loop, we use the components of vectors
179             !   directly for vector operations because of the following
180             !   reasion. Except for intel compilers with '-ipo' option,
181             !   most of Fortran compilers use function calls to perform
182             !   vector operations like rij = x - ep_j(j)%pos.
183             !   This significantly slow downs the speed of the code.
184             !   By using the components of vector directly, we can avoid
185             !   these function calls.
186          end do
187          f(i)%pot = f(i)%pot + poti
188          f(i)%acc = f(i)%acc + ai
189       end do
190
191    end subroutine calc_gravity_ep_ep
192
193    !**** Interaction function (particle-super particle)
194    subroutine calc_gravity_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
195       implicit none
196       integer(c_int), intent(in), value :: n_ip,n_jp
197       type(full_particle), dimension(n_ip), intent(in) :: ep_i
198       type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
199       type(full_particle), dimension(n_ip), intent(inout) :: f
200       !* Local variables
201       integer(c_int) :: i,j
202       real(c_double) :: eps2,poti,r3_inv,r_inv
203       type(fdps_f64vec) :: xi,ai,rij
```

```
204
205         eps2 = eps_grav * eps_grav
206         do i=1,n_ip
207            xi = ep_i(i)%pos
208            ai = 0.0d0
209            poti = 0.0d0
210            do j=1,n_jp
211               rij%x  = xi%x - ep_j(j)%pos%x
212               rij%y  = xi%y - ep_j(j)%pos%y
213               rij%z  = xi%z - ep_j(j)%pos%z
214               r3_inv = rij%x*rij%x &
215                      + rij%y*rij%y &
216                      + rij%z*rij%z &
217                      + eps2
218               r_inv  = 1.0d0/sqrt(r3_inv)
219               r3_inv = r_inv * r_inv
220               r_inv  = r_inv * ep_j(j)%mass
221               r3_inv = r3_inv * r_inv
222               ai%x   = ai%x - r3_inv * rij%x
223               ai%y   = ai%y - r3_inv * rij%y
224               ai%z   = ai%z - r3_inv * rij%z
225               poti   = poti - r_inv
226            end do
227            f(i)%pot = f(i)%pot + poti
228            f(i)%acc = f(i)%acc + ai
229         end do
230
231      end subroutine calc_gravity_ep_sp
232   #endif
233
234   end module user_defined_types
```

Listing 34: N 体シミュレーションのサンプルコード (f_main.F90)

```
1  !-------------------------------------------------------------------------
2  !////////////////////// < M A I N  R O U T I N E > ////////////////////
3  !-------------------------------------------------------------------------
4  subroutine f_main()
5     use fdps_module
6  #if defined(ENABLE_PHANTOM_GRAPE_X86)
7     use phantom_grape_g5_x86
8  #endif
9     use user_defined_types
10    implicit none
11    !* Local parameters
12    integer, parameter :: ntot=2**10
13    !-(force parameters)
14    real, parameter :: theta = 0.5
15    integer, parameter :: n_leaf_limit = 8
16    integer, parameter :: n_group_limit = 64
17    !-(domain decomposition)
18    real, parameter :: coef_ema=0.3
19    !-(timing parameters)
20    double precision, parameter :: time_end = 10.0d0
21    double precision, parameter :: dt = 1.0d0/128.0d0
22    double precision, parameter :: dt_diag = 1.0d0/8.0d0
```

```
23     double precision, parameter :: dt_snap = 1.0d0
24     !* Local variables
25     integer :: i,j,k,num_loop,ierr
26     integer :: psys_num,dinfo_num,tree_num
27     integer :: nloc
28     logical :: clear
29     double precision :: ekin0,epot0,etot0
30     double precision :: ekin1,epot1,etot1
31     double precision :: time_diag,time_snap,time_sys
32     double precision :: r,acc
33     type(fdps_controller) :: fdps_ctrl
34     type(full_particle), dimension(:), pointer :: ptcl
35     type(c_funptr) :: pfunc_ep_ep,pfunc_ep_sp
36     !-(IO)
37     character(len=64) :: fname
38     integer(c_int) :: np
39
40     !* Initialize FDPS
41     call fdps_ctrl%PS_Initialize()
42
43     !* Create domain info object
44     call fdps_ctrl%create_dinfo(dinfo_num)
45     call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
46
47     !* Create particle system object
48     call fdps_ctrl%create_psys(psys_num,'full_particle')
49     call fdps_ctrl%init_psys(psys_num)
50
51     !* Create tree object
52     call fdps_ctrl%create_tree(tree_num, &
53                                "Long,full_particle,full_particle,
                                       full_particle,Monopole")
54     call fdps_ctrl%init_tree(tree_num,ntot,theta, &
55                              n_leaf_limit,n_group_limit)
56
57     !* Make an initial condition
58     call setup_IC(fdps_ctrl,psys_num,ntot)
59
60     !* Domain decomposition and exchange particle
61     call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
62     call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
63
64 #if defined(ENABLE_PHANTOM_GRAPE_X86)
65      call g5_open()
66      call g5_set_eps_to_all(eps_grav);
67 #endif
68
69     !* Compute force at the initial time
70     pfunc_ep_ep = c_funloc(calc_gravity_ep_ep)
71     pfunc_ep_sp = c_funloc(calc_gravity_ep_sp)
72     call fdps_ctrl%calc_force_all_and_write_back(tree_num,     &
73                                                  pfunc_ep_ep, &
74                                                  pfunc_ep_sp, &
75                                                  psys_num,    &
76                                                  dinfo_num)
```

```
77     !* Compute energies at the initial time
78     clear = .true.
79     call calc_energy(fdps_ctrl,psys_num,etot0,ekin0,epot0,clear)
80
81     !* Time integration
82     time_diag = 0.0d0
83     time_snap = 0.0d0
84     time_sys  = 0.0d0
85     num_loop = 0
86     do
87        !* Output
88       !if (fdps_ctrl%get_rank() == 0) then
89       !   write(*,50)num_loop,time_sys
90       !   50 format('(num_loop, time_sys) = ',i5,1x,1es25.16e3)
91       !end if
92        if ( (time_sys >= time_snap) .or. &
93             (((time_sys + dt) - time_snap) > (time_snap - time_sys)) ) then
94           call output(fdps_ctrl,psys_num)
95           time_snap = time_snap + dt_snap
96        end if
97
98        !* Compute energies and output the results
99        clear = .true.
100       call calc_energy(fdps_ctrl,psys_num,etot1,ekin1,epot1,clear)
101       if (fdps_ctrl%get_rank() == 0) then
102          if ( (time_sys >= time_diag) .or. &
103               (((time_sys + dt) - time_diag) > (time_diag - time_sys)) )
                     then
104             write(*,100)time_sys,(etot1-etot0)/etot0
105             100 format("time: ",1es20.10e3,", energy error: ",1es20.10e3)
106             time_diag = time_diag + dt_diag
107          end if
108       end if
109
110       !* Leapfrog: Kick-Drift
111       call kick(fdps_ctrl,psys_num,0.5d0*dt)
112       time_sys = time_sys + dt
113       call drift(fdps_ctrl,psys_num,dt)
114
115       !* Domain decomposition & exchange particle
116       if (mod(num_loop,4) == 0) then
117          call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
118       end if
119       call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
120
121       !* Force calculation
122       pfunc_ep_ep = c_funloc(calc_gravity_ep_ep)
123       pfunc_ep_sp = c_funloc(calc_gravity_ep_sp)
124       call fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
125                                                    pfunc_ep_ep, &
126                                                    pfunc_ep_sp, &
127                                                    psys_num,    &
128                                                    dinfo_num)
129       !* Leapfrog: Kick
130       call kick(fdps_ctrl,psys_num,0.5d0*dt)
```

```
131
132         !* Update num_loop
133         num_loop = num_loop + 1
134
135         !* Termination
136         if (time_sys >= time_end) then
137            exit
138         end if
139      end do
140
141 #if defined(ENABLE_PHANTOM_GRAPE_X86)
142      call g5_close()
143 #endif
144
145      !* Finalize FDPS
146      call fdps_ctrl%PS_Finalize()
147
148 end subroutine f_main
149
150 !-------------------------------------------------------------------------
151 !////////////////////////// S U B R O U T I N E ////////////////////////////
152 !////////////////////////// < S E T U P _ I C > ////////////////////////////
153 !-------------------------------------------------------------------------
154 subroutine setup_IC(fdps_ctrl,psys_num,nptcl_glb)
155      use fdps_vector
156      use fdps_module
157      use user_defined_types
158      implicit none
159      type(fdps_controller), intent(IN) :: fdps_ctrl
160      integer, intent(IN) :: psys_num,nptcl_glb
161      !* Local parameters
162      double precision, parameter :: m_tot=1.0d0
163      double precision, parameter :: rmax=3.0d0,r2max=rmax*rmax
164      !* Local variables
165      integer :: i,j,k,ierr
166      integer :: nprocs,myrank
167      double precision :: r2,cm_mass
168      type(fdps_f64vec) :: cm_pos,cm_vel,pos
169      type(full_particle), dimension(:), pointer :: ptcl
170      character(len=64) :: fname
171
172      !* Get # of MPI processes and rank number
173      nprocs = fdps_ctrl%get_num_procs()
174      myrank = fdps_ctrl%get_rank()
175
176      !* Make an initial condition at RANK 0
177      if (myrank == 0) then
178         !* Set # of local particles
179         call fdps_ctrl%set_nptcl_loc(psys_num,nptcl_glb)
180
181         !* Create an uniform sphere of particles
182         !** get the pointer to full particle data
183         call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
184         !** initialize Mersenne twister
185         call fdps_ctrl%MT_init_genrand(0)
```

```
186       do i=1,nptcl_glb
187          ptcl(i)%id   = i
188          ptcl(i)%mass = m_tot/nptcl_glb
189          do
190             ptcl(i)%pos%x = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
191             ptcl(i)%pos%y = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
192             ptcl(i)%pos%z = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
193             r2 = ptcl(i)%pos*ptcl(i)%pos
194             if ( r2 < r2max ) exit
195          end do
196          ptcl(i)%vel = 0.0d0
197       end do
198
199       !* Correction
200       cm_pos  = 0.0d0
201       cm_vel  = 0.0d0
202       cm_mass = 0.0d0
203       do i=1,nptcl_glb
204          cm_pos  = cm_pos  + ptcl(i)%mass * ptcl(i)%pos
205          cm_vel  = cm_vel  + ptcl(i)%mass * ptcl(i)%vel
206          cm_mass = cm_mass + ptcl(i)%mass
207       end do
208       cm_pos = cm_pos/cm_mass
209       cm_vel = cm_vel/cm_mass
210       do i=1,nptcl_glb
211          ptcl(i)%pos = ptcl(i)%pos - cm_pos
212          ptcl(i)%vel = ptcl(i)%vel - cm_vel
213       end do
214
215       !* Output
216      !fname = 'initial.dat'
217      !open(unit=9,file=trim(fname),action='write',status='replace', &
218      !     form='unformatted',access='stream')
219      !open(unit=9,file=trim(fname),action='write',status='replace')
220      !    do i=1,nptcl_glb
221      !       !write(9)ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z
222      !       write(9,'(3es25.16e3)')ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos
                %z
223      !    end do
224      !close(unit=9)
225
226       !* Release the pointer
227       nullify( ptcl )
228
229    else
230       call fdps_ctrl%set_nptcl_loc(psys_num,0)
231    end if
232
233    !* Set the gravitational softening
234    eps_grav = 1.0d0/32.0d0
235
236 end subroutine setup_IC
```

```
237
238 !------------------------------------------------------------------------
239 !////////////////////////// S U B R O U T I N E ////////////////////////
240 !//////////////////////////     < K I C K >     ////////////////////////
241 !------------------------------------------------------------------------
242 subroutine kick(fdps_ctrl,psys_num,dt)
243    use fdps_vector
244    use fdps_module
245    use user_defined_types
246    implicit none
247    type(fdps_controller), intent(IN) :: fdps_ctrl
248    integer, intent(IN) :: psys_num
249    double precision, intent(IN) :: dt
250    !* Local variables
251    integer :: i,nptcl_loc
252    type(full_particle), dimension(:), pointer :: ptcl
253
254    !* Get # of local particles
255    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
256
257    !* Get the pointer to full particle data
258    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
259    do i=1,nptcl_loc
260       ptcl(i)%vel = ptcl(i)%vel + ptcl(i)%acc * dt
261    end do
262    nullify(ptcl)
263
264 end subroutine kick
265
266 !------------------------------------------------------------------------
267 !////////////////////////// S U B R O U T I N E ////////////////////////
268 !//////////////////////////    < D R I F T >    ////////////////////////
269 !------------------------------------------------------------------------
270 subroutine drift(fdps_ctrl,psys_num,dt)
271    use fdps_vector
272    use fdps_module
273    use user_defined_types
274    implicit none
275    type(fdps_controller), intent(IN) :: fdps_ctrl
276    integer, intent(IN) :: psys_num
277    double precision, intent(IN) :: dt
278    !* Local variables
279    integer :: i,nptcl_loc
280    type(full_particle), dimension(:), pointer :: ptcl
281
282    !* Get # of local particles
283    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
284
285    !* Get the pointer to full particle data
286    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
287    do i=1,nptcl_loc
288       ptcl(i)%pos = ptcl(i)%pos + ptcl(i)%vel * dt
289    end do
290    nullify(ptcl)
291
```

```
292 end subroutine drift
293
294 !------------------------------------------------------------------------
295 !//////////////////////    S U B R O U T I N E    ////////////////////////
296 !////////////////////// < C A L C _ E N E R G Y > ////////////////////////
297 !------------------------------------------------------------------------
298 subroutine calc_energy(fdps_ctrl,psys_num,etot,ekin,epot,clear)
299    use fdps_vector
300    use fdps_module
301    use user_defined_types
302    implicit none
303    type(fdps_controller), intent(IN) :: fdps_ctrl
304    integer, intent(IN) :: psys_num
305    double precision, intent(INOUT) :: etot,ekin,epot
306    logical, intent(IN) :: clear
307    !* Local variables
308    integer :: i,nptcl_loc
309    double precision :: etot_loc,ekin_loc,epot_loc
310    type(full_particle), dimension(:), pointer :: ptcl
311
312    !* Clear energies
313    if (clear .eqv. .true.) then
314       etot = 0.0d0
315       ekin = 0.0d0
316       epot = 0.0d0
317    end if
318
319    !* Get # of local particles
320    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
321    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
322
323    !* Compute energies
324    ekin_loc = 0.0d0
325    epot_loc = 0.0d0
326    do i=1,nptcl_loc
327       ekin_loc = ekin_loc + ptcl(i)%mass * ptcl(i)%vel * ptcl(i)%vel
328       epot_loc = epot_loc + ptcl(i)%mass * (ptcl(i)%pot + ptcl(i)%mass/
                eps_grav)
329    end do
330    ekin_loc = ekin_loc * 0.5d0
331    epot_loc = epot_loc * 0.5d0
332    etot_loc = ekin_loc + epot_loc
333    call fdps_ctrl%get_sum(ekin_loc,ekin)
334    call fdps_ctrl%get_sum(epot_loc,epot)
335    call fdps_ctrl%get_sum(etot_loc,etot)
336
337    !* Release the pointer
338    nullify(ptcl)
339
340 end subroutine calc_energy
341
342 !------------------------------------------------------------------------
343 !////////////////////////// S U B R O U T I N E ////////////////////////////
344 !//////////////////////////   < O U T P U T >   ////////////////////////////
345 !------------------------------------------------------------------------
```

```
346 subroutine output(fdps_ctrl,psys_num)
347    use fdps_vector
348    use fdps_module
349    use user_defined_types
350    implicit none
351    type(fdps_controller), intent(IN) :: fdps_ctrl
352    integer, intent(IN) :: psys_num
353    !* Local parameters
354    character(len=16), parameter :: root_dir="result"
355    character(len=16), parameter :: file_prefix_1st="snap"
356    character(len=16), parameter :: file_prefix_2nd="proc"
357    !* Local variables
358    integer :: i,nptcl_loc
359    integer :: myrank
360    character(len=5) :: file_num,proc_num
361    character(len=64) :: cmd,sub_dir,fname
362    type(full_particle), dimension(:), pointer :: ptcl
363    !* Static variables
364    integer, save :: snap_num=0
365
366    !* Get the rank number
367    myrank = fdps_ctrl%get_rank()
368
369    !* Get # of local particles
370    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
371
372    !* Get the pointer to full particle data
373    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
374
375    !* Output
376    write(file_num,"(i5.5)")snap_num
377    write(proc_num,"(i5.5)")myrank
378    fname =  trim(root_dir) // "/" &
379          // trim(file_prefix_1st) // file_num // "-" &
380          // trim(file_prefix_2nd) // proc_num // ".dat"
381    open(unit=9,file=trim(fname),action='write',status='replace')
382       do i=1,nptcl_loc
383          write(9,100)ptcl(i)%id,ptcl(i)%mass, &
384                      ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z, &
385                      ptcl(i)%vel%x,ptcl(i)%vel%y,ptcl(i)%vel%z
386          100 format(i8,1x,7e25.16e3)
387       end do
388    close(unit=9)
389    nullify(ptcl)
390
391    !* Update snap_num
392    snap_num = snap_num + 1
393
394 end subroutine output
```

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 35: 固定長 SPH シミュレーションのサンプルコード (user_defined.F90)

```
!================================
!   MODULE: User defined types
!================================
module user_defined_types
   use, intrinsic :: iso_c_binding
   use fdps_vector
   implicit none

   !* Private parameters
   real(kind=c_double), parameter, private :: pi=datan(1.0d0)*4.0d0
   !* Public parameters
   real(kind=c_double), parameter, public :: kernel_support_radius=2.5d0

   !**** Force types
   type, public, bind(c) :: force_dens !$fdps Force
      !$fdps clear smth=keep
      real(kind=c_double) :: dens
      real(kind=c_double) :: smth
   end type force_dens

   type, public, bind(c) :: force_hydro !$fdps Force
      !$fdps clear
      type(fdps_f64vec) :: acc
      real(kind=c_double) :: eng_dot
      real(kind=c_double) :: dt
   end type force_hydro

   !**** Full particle type
   type, public, bind(c) :: full_particle !$fdps FP
      !$fdps copyFromForce force_dens (dens,dens)
      !$fdps copyFromForce force_hydro (acc,acc) (eng_dot,eng_dot) (dt,dt)
      real(kind=c_double) :: mass !$fdps charge
      type(fdps_f64vec) :: pos !$fdps position
      type(fdps_f64vec) :: vel
      type(fdps_f64vec) :: acc
      real(kind=c_double) :: dens
      real(kind=c_double) :: eng
      real(kind=c_double) :: pres
      real(kind=c_double) :: smth !$fdps rsearch
      real(kind=c_double) :: snds
      real(kind=c_double) :: eng_dot
      real(kind=c_double) :: dt
      integer(kind=c_long_long) :: id
      type(fdps_f64vec) :: vel_half
      real(kind=c_double) :: eng_half
   end type full_particle
```

```
47
48    !**** Essential particle type
49    type, public, bind(c) :: essential_particle !$fdps EPI,EPJ
50       !$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,
             mass) (smth,smth) (dens,dens) (pres,pres) (snds,snds)
51       integer(kind=c_long_long) :: id !$fdps id
52       type(fdps_f64vec) :: pos !$fdps position
53       type(fdps_f64vec) :: vel
54       real(kind=c_double) :: mass !$fdps charge
55       real(kind=c_double) :: smth !$fdps rsearch
56       real(kind=c_double) :: dens
57       real(kind=c_double) :: pres
58       real(kind=c_double) :: snds
59    end type essential_particle
60
61    !* Public routines
62    public :: W
63    public :: gradW
64    public :: calc_density
65    public :: calc_hydro_force
66
67    contains
68
69    !-------------------------------------------------------------------
70    pure function W(dr,h)
71       implicit none
72       real(kind=c_double) :: W
73       type(fdps_f64vec), intent(in) :: dr
74       real(kind=c_double), intent(in) :: h
75       !* Local variables
76       real(kind=c_double) :: s,s1,s2
77
78       s = dsqrt(dr%x*dr%x &
79                +dr%y*dr%y &
80                +dr%z*dr%z)/h
81       s1 = 1.0d0 - s
82       if (s1 < 0.0d0) s1 = 0.0d0
83       s2 = 0.5d0 - s
84       if (s2 < 0.0d0) s2 = 0.0d0
85       W = (s1*s1*s1) - 4.0d0*(s2*s2*s2)
86       W = W * 16.0d0/(pi*h*h*h)
87
88    end function W
89
90    !-------------------------------------------------------------------
91    pure function gradW(dr,h)
92       implicit none
93       type(fdps_f64vec) :: gradW
94       type(fdps_f64vec), intent(in) :: dr
95       real(kind=c_double), intent(in) :: h
96       !* Local variables
97       real(kind=c_double) :: dr_abs,s,s1,s2,coef
98
99       dr_abs = dsqrt(dr%x*dr%x &
100                     +dr%y*dr%y &
```

```
101                       +dr%z*dr%z)
102        s = dr_abs/h
103        s1 = 1.0d0 - s
104        if (s1 < 0.0d0) s1 = 0.0d0
105        s2 = 0.5d0 - s
106        if (s2 < 0.0d0) s2 = 0.0d0
107        coef = - 3.0d0*(s1*s1) + 12.0d0*(s2*s2)
108        coef = coef * 16.0d0/(pi*h*h*h)
109        coef = coef / (dr_abs*h + 1.0d-6*h)
110        gradW%x = dr%x * coef
111        gradW%y = dr%y * coef
112        gradW%z = dr%z * coef
113
114     end function gradW
115
116     !**** Interaction function
117     subroutine calc_density(ep_i,n_ip,ep_j,n_jp,f) bind(c)
118        integer(kind=c_int), intent(in), value :: n_ip,n_jp
119        type(essential_particle), dimension(n_ip), intent(in) :: ep_i
120        type(essential_particle), dimension(n_jp), intent(in) :: ep_j
121        type(force_dens), dimension(n_ip), intent(inout) :: f
122        !* Local variables
123        integer(kind=c_int) :: i,j
124        type(fdps_f64vec) :: dr
125
126        do i=1,n_ip
127           f(i)%dens = 0.0d0
128           do j=1,n_jp
129              dr%x = ep_j(j)%pos%x - ep_i(i)%pos%x
130              dr%y = ep_j(j)%pos%y - ep_i(i)%pos%y
131              dr%z = ep_j(j)%pos%z - ep_i(i)%pos%z
132              f(i)%dens = f(i)%dens &
133                        + ep_j(j)%mass * W(dr,ep_i(i)%smth)
134           end do
135        end do
136
137     end subroutine calc_density
138
139     !**** Interaction function
140     subroutine calc_hydro_force(ep_i,n_ip,ep_j,n_jp,f) bind(c)
141        integer(kind=c_int), intent(in), value :: n_ip,n_jp
142        type(essential_particle), dimension(n_ip), intent(in) :: ep_i
143        type(essential_particle), dimension(n_jp), intent(in) :: ep_j
144        type(force_hydro), dimension(n_ip), intent(inout) :: f
145        !* Local parameters
146        real(kind=c_double), parameter :: C_CFL=0.3d0
147        !* Local variables
148        integer(kind=c_int) :: i,j
149        real(kind=c_double) :: mass_i,mass_j,smth_i,smth_j, &
150                               dens_i,dens_j,pres_i,pres_j, &
151                               snds_i,snds_j
152        real(kind=c_double) :: povrho2_i,povrho2_j, &
153                               v_sig_max,dr_dv,w_ij,v_sig,AV
154        type(fdps_f64vec) :: pos_i,pos_j,vel_i,vel_j, &
155                             dr,dv,gradW_ij
```

```
156
157         do i=1,n_ip
158            !* Zero-clear
159            v_sig_max = 0.0d0
160            !* Extract i-particle info.
161            pos_i = ep_i(i)%pos
162            vel_i = ep_i(i)%vel
163            mass_i  = ep_i(i)%mass
164            smth_i  = ep_i(i)%smth
165            dens_i  = ep_i(i)%dens
166            pres_i  = ep_i(i)%pres
167            snds_i  = ep_i(i)%snds
168            povrho2_i = pres_i/(dens_i*dens_i)
169            do j=1,n_jp
170               !* Extract j-particle info.
171               pos_j%x = ep_j(j)%pos%x
172               pos_j%y = ep_j(j)%pos%y
173               pos_j%z = ep_j(j)%pos%z
174               vel_j%x = ep_j(j)%vel%x
175               vel_j%y = ep_j(j)%vel%y
176               vel_j%z = ep_j(j)%vel%z
177               mass_j  = ep_j(j)%mass
178               smth_j  = ep_j(j)%smth
179               dens_j  = ep_j(j)%dens
180               pres_j  = ep_j(j)%pres
181               snds_j  = ep_j(j)%snds
182               povrho2_j = pres_j/(dens_j*dens_j)
183               !* Compute dr & dv
184               dr%x = pos_i%x - pos_j%x
185               dr%y = pos_i%y - pos_j%y
186               dr%z = pos_i%z - pos_j%z
187               dv%x = vel_i%x - vel_j%x
188               dv%y = vel_i%y - vel_j%y
189               dv%z = vel_i%z - vel_j%z
190               !* Compute the signal velocity
191               dr_dv = dr%x * dv%x + dr%y * dv%y + dr%z * dv%z
192               if (dr_dv < 0.0d0) then
193                  w_ij = dr_dv / sqrt(dr%x * dr%x + dr%y * dr%y + dr%z * dr%z
                        )
194               else
195                  w_ij = 0.0d0
196               end if
197               v_sig = snds_i + snds_j - 3.0d0 * w_ij
198               v_sig_max = max(v_sig_max, v_sig)
199               !* Compute the artificial viscosity
200               AV = - 0.5d0*v_sig*w_ij / (0.5d0*(dens_i+dens_j))
201               !* Compute the average of the gradients of kernel
202               gradW_ij = 0.5d0 * (gradW(dr,smth_i) + gradW(dr,smth_j))
203               !* Compute the acceleration and the heating rate
204               f(i)%acc%x = f(i)%acc%x - mass_j*(povrho2_i+povrho2_j+AV)*
                        gradW_ij%x
205               f(i)%acc%y = f(i)%acc%y - mass_j*(povrho2_i+povrho2_j+AV)*
                        gradW_ij%y
206               f(i)%acc%z = f(i)%acc%z - mass_j*(povrho2_i+povrho2_j+AV)*
                        gradW_ij%z
```

```
207              f(i)%eng_dot = f(i)%eng_dot &
208                           + mass_j * (povrho2_i + 0.5d0*AV) &
209                            *(dv%x * gradW_ij%x &
210                              +dv%y * gradW_ij%y &
211                              +dv%z * gradW_ij%z)
212           end do
213           f(i)%dt = C_CFL*2.0d0*smth_i/(v_sig_max*kernel_support_radius)
214        end do
215        ! [IMPORTANT NOTE]
216        !   In the innermost loop, we use the components of vectors
217        !   directly for vector operations because of the following
218        !   reasion. Except for intel compilers with `-ipo` option,
219        !   most of Fortran compilers use function calls to perform
220        !   vector operations like rij = x - ep_j(j)%pos.
221        !   This significantly slow downs the speed of the code.
222        !   By using the components of vector directly, we can avoid
223        !   these function calls.
224
225     end subroutine calc_hydro_force
226
227  end module user_defined_types
```

Listing 36: 固定長 SPH シミュレーションのサンプルコード (f_main.F90)

```
1  !-------------------------------------------------------------------------
2  !////////////////////  < M A I N   R O U T I N E >  ////////////////////
3  !-------------------------------------------------------------------------
4  subroutine f_main()
5     use fdps_vector
6     use fdps_module
7     use user_defined_types
8     implicit none
9     !* Local parameters
10    !-(force parameters)
11    real, parameter :: theta = 0.5
12    integer, parameter :: n_leaf_limit = 8
13    integer, parameter :: n_group_limit = 64
14    !-(domain decomposition)
15    real, parameter :: coef_ema=0.3
16    !-(IO)
17    integer, parameter :: output_interval=10
18    !* Local variables
19    integer :: i,j,k,ierr
20    integer :: nstep
21    integer :: psys_num,dinfo_num
22    integer :: tree_num_dens,tree_num_hydro
23    integer :: ntot,nloc
24    logical :: clear
25    double precision :: time,dt,end_time
26    type(fdps_f64vec) :: pos_ll,pos_ul
27    type(fdps_controller) :: fdps_ctrl
28    type(full_particle), dimension(:), pointer :: ptcl
29    type(c_funptr) :: pfunc_ep_ep
30    !-(IO)
31    character(len=64) :: filename
32    !* External routines
```

```
33    double precision, external :: get_timestep
34
35    !* Initialize FDPS
36    call fdps_ctrl%PS_Initialize()
37
38    !* Make an instance of ParticleSystem and initialize it
39    call fdps_ctrl%create_psys(psys_num,'full_particle')
40    call fdps_ctrl%init_psys(psys_num)
41
42    !* Make an initial condition and initialize the particle system
43    call setup_IC(fdps_ctrl,psys_num,end_time,pos_ll,pos_ul)
44
45    !* Make an instance of DomainInfo and initialize it
46    call fdps_ctrl%create_dinfo(dinfo_num)
47    call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
48    call fdps_ctrl%set_boundary_condition(dinfo_num,fdps_bc_periodic_xyz)
49    call fdps_ctrl%set_pos_root_domain(dinfo_num,pos_ll,pos_ul)
50
51    !* Perform domain decomposition and exchange particles
52    call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
53    call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
54
55    !* Make two tree structures
56    ntot = fdps_ctrl%get_nptcl_glb(psys_num)
57    !** dens_tree (used for the density calculation)
58    call fdps_ctrl%create_tree(tree_num_dens, &
59                               "Short,force_dens,essential_particle,
                                     essential_particle,Gather")
60    call fdps_ctrl%init_tree(tree_num_dens,3*ntot,theta, &
61                             n_leaf_limit,n_group_limit)
62
63    !** hydro_tree (used for the force calculation)
64    call fdps_ctrl%create_tree(tree_num_hydro, &
65                               "Short,force_hydro,essential_particle,
                                     essential_particle,Symmetry")
66    call fdps_ctrl%init_tree(tree_num_hydro,3*ntot,theta, &
67                             n_leaf_limit,n_group_limit)
68
69    !* Compute density, pressure, acceleration due to pressure gradient
70    pfunc_ep_ep = c_funloc(calc_density)
71    call fdps_ctrl%calc_force_all_and_write_back(tree_num_dens, &
72                                                 pfunc_ep_ep,   &
73                                                 psys_num,      &
74                                                 dinfo_num)
75    call set_pressure(fdps_ctrl,psys_num)
76    pfunc_ep_ep = c_funloc(calc_hydro_force)
77    call fdps_ctrl%calc_force_all_and_write_back(tree_num_hydro, &
78                                                 pfunc_ep_ep,   &
79                                                 psys_num,      &
80                                                 dinfo_num)
81    !* Get timestep
82    dt = get_timestep(fdps_ctrl,psys_num)
83
84    !* Main loop for time integration
85    nstep = 0; time = 0.0d0
```

```
86      do
87         !* Leap frog: Initial Kick & Full Drift
88         call initial_kick(fdps_ctrl,psys_num,dt)
89         call full_drift(fdps_ctrl,psys_num,dt)
90
91         !* Adjust the positions of the SPH particles that run over
92         !  the computational boundaries.
93         call fdps_ctrl%adjust_pos_into_root_domain(psys_num,dinfo_num)
94
95         !* Leap frog: Predict
96         call predict(fdps_ctrl,psys_num,dt)
97
98         !* Perform domain decomposition and exchange particles again
99         call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
100        call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
101
102        !* Compute density, pressure, acceleration due to pressure gradient
103        pfunc_ep_ep = c_funloc(calc_density)
104        call fdps_ctrl%calc_force_all_and_write_back(tree_num_dens, &
105                                                     pfunc_ep_ep,   &
106                                                     psys_num,      &
107                                                     dinfo_num)
108        call set_pressure(fdps_ctrl,psys_num)
109        pfunc_ep_ep = c_funloc(calc_hydro_force)
110        call fdps_ctrl%calc_force_all_and_write_back(tree_num_dens, &
111                                                     pfunc_ep_ep,   &
112                                                     psys_num,      &
113                                                     dinfo_num)
114
115        !* Get a new timestep
116        dt = get_timestep(fdps_ctrl,psys_num)
117
118        !* Leap frog: Final Kick
119        call final_kick(fdps_ctrl,psys_num,dt)
120
121        !* Output result files
122        if (mod(nstep,output_interval) == 0) then
123           call output(fdps_ctrl,psys_num,nstep)
124           call check_cnsrvd_vars(fdps_ctrl,psys_num)
125        end if
126
127        !* Output information to STDOUT
128        if (fdps_ctrl%get_rank() == 0) then
129           write(*,200)time,nstep
130           200 format("================================"/ &
131                      "time␣␣=␣",1es25.16e3/               &
132                      "nstep␣=␣",i6/                       &
133                      "================================")
134        end if
135
136        !* Termination condition
137        if (time >= end_time) exit
138
139        !* Update time & step
140        time  = time + dt
```

```
141         nstep = nstep + 1
142
143      end do
144      call fdps_ctrl%ps_finalize()
145      stop 0
146
147      !* Finalize FDPS
148      call fdps_ctrl%PS_Finalize()
149
150  end subroutine f_main
151
152  !-------------------------------------------------------------------------
153  !////////////////////////// S U B R O U T I N E ////////////////////////////
154  !////////////////////////// < S E T U P _ I C > ////////////////////////////
155  !-------------------------------------------------------------------------
156  subroutine setup_IC(fdps_ctrl,psys_num,end_time,pos_ll,pos_ul)
157      use fdps_vector
158      use fdps_module
159      use user_defined_types
160      implicit none
161      type(fdps_controller), intent(IN) :: fdps_ctrl
162      integer, intent(IN) :: psys_num
163      double precision, intent(inout) :: end_time
164      type(fdps_f64vec) :: pos_ll,pos_ul
165      !* Local variables
166      integer :: i
167      integer :: nprocs,myrank
168      integer :: nptcl_glb
169      double precision :: dens_L,dens_R,eng_L,eng_R
170      double precision :: x,y,z,dx,dy,dz
171      double precision :: dx_tgt,dy_tgt,dz_tgt
172      type(full_particle), dimension(:), pointer :: ptcl
173      character(len=64) :: fname
174
175      !* Get # of MPI processes and rank number
176      nprocs = fdps_ctrl%get_num_procs()
177      myrank = fdps_ctrl%get_rank()
178
179      !* Set the box size
180      pos_ll%x = 0.0d0
181      pos_ll%y = 0.0d0
182      pos_ll%z = 0.0d0
183      pos_ul%x = 1.0d0
184      pos_ul%y = pos_ul%x / 8.0d0
185      pos_ul%z = pos_ul%x / 8.0d0
186
187      !* Make an initial condition at RANK 0
188      if (myrank == 0) then
189         !* Set the left and right states
190         dens_L = 1.0d0
191         eng_L  = 2.5d0
192         dens_R = 0.5d0
193         eng_R  = 2.5d0
194         !* Set the separation of particle of the left state
195         dx = 1.0d0 / 128.0d0
```

```
196         dy = dx
197         dz = dx
198         !* Set the number of local particles
199         nptcl_glb = 0
200         !** (1) Left-half
201         x = 0.0d0
202         do
203            y = 0.0d0
204            do
205               z = 0.0d0
206               do
207                  nptcl_glb = nptcl_glb + 1
208                  z = z + dz
209                  if (z >= pos_ul%z) exit
210               end do
211               y = y + dy
212               if (y >= pos_ul%y) exit
213            end do
214            x = x + dx
215            if (x >= 0.5d0*pos_ul%x) exit
216         end do
217         write(*,*)'nptcl_glb(L)␣␣␣=␣',nptcl_glb
218         !** (2) Right-half
219         x = 0.5d0*pos_ul%x
220         do
221            y = 0.0d0
222            do
223               z = 0.0d0
224               do
225                  nptcl_glb = nptcl_glb + 1
226                  z = z + dz
227                  if (z >= pos_ul%z) exit
228               end do
229               y = y + dy
230               if (y >= pos_ul%y) exit
231            end do
232            x = x + (dens_L/dens_R)*dx
233            if (x >= pos_ul%x) exit
234         end do
235         write(*,*)'nptcl_glb(L+R)␣=␣',nptcl_glb
236         !* Place SPH particles
237         call fdps_ctrl%set_nptcl_loc(psys_num,nptcl_glb)
238         call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
239         i = 0
240         !** (1) Left-half
241         x = 0.0d0
242         do
243            y = 0.0d0
244            do
245               z = 0.0d0
246               do
247                  i = i + 1
248                  ptcl(i)%id    = i
249                  ptcl(i)%pos%x = x
250                  ptcl(i)%pos%y = y
```

```
251                ptcl(i)%pos%z = z
252                ptcl(i)%dens  = dens_L
253                ptcl(i)%eng   = eng_L
254                z = z + dz
255                if (z >= pos_ul%z) exit
256             end do
257             y = y + dy
258             if (y >= pos_ul%y) exit
259          end do
260          x = x + dx
261          if (x >= 0.5d0*pos_ul%x) exit
262       end do
263       write(*,*)'nptcl(L)␣␣␣=␣',i
264       !** (2) Right-half
265       x = 0.5d0*pos_ul%x
266       do
267          y = 0.0d0
268          do
269             z = 0.0d0
270             do
271                i = i + 1
272                ptcl(i)%id    = i
273                ptcl(i)%pos%x = x
274                ptcl(i)%pos%y = y
275                ptcl(i)%pos%z = z
276                ptcl(i)%dens  = dens_R
277                ptcl(i)%eng   = eng_R
278                z = z + dz
279                if (z >= pos_ul%z) exit
280             end do
281             y = y + dy
282             if (y >= pos_ul%y) exit
283          end do
284          x = x + (dens_L/dens_R)*dx
285          if (x >= pos_ul%x) exit
286       end do
287       write(*,*)'nptcl(L+R)␣=␣',i
288       !* Set particle mass and smoothing length
289       do i=1,nptcl_glb
290          ptcl(i)%mass = 0.5d0*(dens_L+dens_R)         &
291                       * (pos_ul%x*pos_ul%y*pos_ul%z) &
292                       / nptcl_glb
293          ptcl(i)%smth = kernel_support_radius * 0.012d0
294       end do
295
296       !* Check the initial distribution
297      !fname = "initial.dat"
298      !open(unit=9,file=trim(fname),action='write',status='replace')
299      !    do i=1,nptcl_glb
300      !       write(9,'(3es25.16e3)')ptcl(i)%pos%x, &
301      !                              ptcl(i)%pos%y, &
302      !                              ptcl(i)%pos%z
303      !    end do
304      !close(unit=9)
305
```

```
306     else
307        call fdps_ctrl%set_nptcl_loc(psys_num,0)
308     end if
309
310     !* Set the end time
311     end_time = 0.12d0
312
313     !* Inform to STDOUT
314     if (fdps_ctrl%get_rank() == 0) then
315        write(*,*)"setup..."
316     end if
317    !call fdps_ctrl%ps_finalize()
318    !stop 0
319
320 end subroutine setup_IC
321
322 !-------------------------------------------------------------------------
323 !//////////////////////     S U B R O U T I N E     ////////////////////
324 !////////////////////// < G E T _ T I M E S T E P > ////////////////////
325 !-------------------------------------------------------------------------
326 function get_timestep(fdps_ctrl,psys_num)
327     use fdps_vector
328     use fdps_module
329     use user_defined_types
330     implicit none
331     real(kind=c_double) :: get_timestep
332     type(fdps_controller), intent(in) :: fdps_ctrl
333     integer, intent(in) :: psys_num
334     !* Local variables
335     integer :: i,nptcl_loc
336     type(full_particle), dimension(:), pointer :: ptcl
337     real(kind=c_double) :: dt_loc
338
339     !* Get # of local particles
340     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
341
342     !* Get the pointer to full particle data
343     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
344     dt_loc = 1.0d30
345     do i=1,nptcl_loc
346        dt_loc = min(dt_loc, ptcl(i)%dt)
347     end do
348     nullify(ptcl)
349
350     !* Reduction
351     call fdps_ctrl%get_min_value(dt_loc,get_timestep)
352
353 end function get_timestep
354
355 !-------------------------------------------------------------------------
356 !//////////////////////     S U B R O U T I N E     ////////////////////
357 !////////////////////// < I N I T I A L _ K I C K > ////////////////////
358 !-------------------------------------------------------------------------
359 subroutine initial_kick(fdps_ctrl,psys_num,dt)
360     use fdps_vector
```

```
361     use fdps_module
362     use user_defined_types
363     implicit none
364     type(fdps_controller), intent(in) :: fdps_ctrl
365     integer, intent(in) :: psys_num
366     double precision, intent(in) :: dt
367     !* Local variables
368     integer :: i,nptcl_loc
369     type(full_particle), dimension(:), pointer :: ptcl
370
371     !* Get # of local particles
372     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
373
374     !* Get the pointer to full particle data
375     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
376     do i=1,nptcl_loc
377        ptcl(i)%vel_half = ptcl(i)%vel + 0.5d0 * dt * ptcl(i)%acc
378        ptcl(i)%eng_half = ptcl(i)%eng + 0.5d0 * dt * ptcl(i)%eng_dot
379     end do
380     nullify(ptcl)
381
382 end subroutine initial_kick
383
384 !-------------------------------------------------------------------------
385 !////////////////////   S U B R O U T I N E   ////////////////////////////
386 !//////////////////////// < F U L L _ D R I F T > ////////////////////////
387 !-------------------------------------------------------------------------
388 subroutine full_drift(fdps_ctrl,psys_num,dt)
389     use fdps_vector
390     use fdps_module
391     use user_defined_types
392     implicit none
393     type(fdps_controller), intent(in) :: fdps_ctrl
394     integer, intent(in) :: psys_num
395     double precision, intent(in) :: dt
396     !* Local variables
397     integer :: i,nptcl_loc
398     type(full_particle), dimension(:), pointer :: ptcl
399
400     !* Get # of local particles
401     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
402
403     !* Get the pointer to full particle data
404     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
405     do i=1,nptcl_loc
406        ptcl(i)%pos = ptcl(i)%pos + dt * ptcl(i)%vel_half
407     end do
408     nullify(ptcl)
409
410 end subroutine full_drift
411
412 !-------------------------------------------------------------------------
413 !////////////////////////// S U B R O U T I N E ////////////////////////////
414 !//////////////////////////  < P R E D I C T >  ////////////////////////////
415 !-------------------------------------------------------------------------
```

```
416 subroutine predict(fdps_ctrl,psys_num,dt)
417    use fdps_vector
418    use fdps_module
419    use user_defined_types
420    implicit none
421    type(fdps_controller), intent(in) :: fdps_ctrl
422    integer, intent(in) :: psys_num
423    double precision, intent(in) :: dt
424    !* Local variables
425    integer :: i,nptcl_loc
426    type(full_particle), dimension(:), pointer :: ptcl
427
428    !* Get # of local particles
429    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
430
431    !* Get the pointer to full particle data
432    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
433    do i=1,nptcl_loc
434       ptcl(i)%vel = ptcl(i)%vel + dt * ptcl(i)%acc
435       ptcl(i)%eng = ptcl(i)%eng + dt * ptcl(i)%eng_dot
436    end do
437    nullify(ptcl)
438
439 end subroutine predict
440
441 !-------------------------------------------------------------------------
442 !////////////////////////   S U B R O U T I N E   ////////////////////////
443 !//////////////////////// < F I N A L _ K I C K > ////////////////////////
444 !-------------------------------------------------------------------------
445 subroutine final_kick(fdps_ctrl,psys_num,dt)
446    use fdps_vector
447    use fdps_module
448    use user_defined_types
449    implicit none
450    type(fdps_controller), intent(in) :: fdps_ctrl
451    integer, intent(in) :: psys_num
452    double precision, intent(in) :: dt
453    !* Local variables
454    integer :: i,nptcl_loc
455    type(full_particle), dimension(:), pointer :: ptcl
456
457    !* Get # of local particles
458    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
459
460    !* Get the pointer to full particle data
461    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
462    do i=1,nptcl_loc
463       ptcl(i)%vel = ptcl(i)%vel_half + 0.5d0 * dt * ptcl(i)%acc
464       ptcl(i)%eng = ptcl(i)%eng_half + 0.5d0 * dt * ptcl(i)%eng_dot
465    end do
466    nullify(ptcl)
467
468 end subroutine final_kick
469
470 !-------------------------------------------------------------------------
```

```
471 !//////////////////////     S U B R O U T I N E     ////////////////////
472 !////////////////////// < S E T _ P R E S S U R E > ////////////////////
473 !-------------------------------------------------------------------------
474 subroutine set_pressure(fdps_ctrl,psys_num)
475    use fdps_vector
476    use fdps_module
477    use user_defined_types
478    implicit none
479    type(fdps_controller), intent(in) :: fdps_ctrl
480    integer, intent(in) :: psys_num
481    !* Local parameters
482    double precision, parameter :: hcr=1.4d0
483    !* Local variables
484    integer :: i,nptcl_loc
485    type(full_particle), dimension(:), pointer :: ptcl
486
487    !* Get # of local particles
488    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
489
490    !* Get the pointer to full particle data
491    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
492    do i=1,nptcl_loc
493       ptcl(i)%pres = (hcr - 1.0d0) * ptcl(i)%dens * ptcl(i)%eng
494       ptcl(i)%snds = dsqrt(hcr * ptcl(i)%pres / ptcl(i)%dens)
495    end do
496    nullify(ptcl)
497
498 end subroutine set_pressure
499
500 !-------------------------------------------------------------------------
501 !////////////////////////// S U B R O U T I N E ////////////////////////
502 !//////////////////////////   < O U T P U T >   ////////////////////////
503 !-------------------------------------------------------------------------
504 subroutine output(fdps_ctrl,psys_num,nstep)
505    use fdps_vector
506    use fdps_module
507    use user_defined_types
508    implicit none
509    type(fdps_controller), intent(IN) :: fdps_ctrl
510    integer, intent(IN) :: psys_num
511    integer, intent(IN) :: nstep
512    !* Local parameters
513    character(len=16), parameter :: root_dir="result"
514    character(len=16), parameter :: file_prefix_1st="snap"
515    character(len=16), parameter :: file_prefix_2nd="proc"
516    !* Local variables
517    integer :: i,nptcl_loc
518    integer :: myrank
519    character(len=5) :: file_num,proc_num
520    character(len=64) :: cmd,sub_dir,fname
521    type(full_particle), dimension(:), pointer :: ptcl
522
523    !* Get the rank number
524    myrank = fdps_ctrl%get_rank()
525
```

```
526     !* Get # of local particles
527     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
528
529     !* Get the pointer to full particle data
530     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
531
532     !* Output
533     write(file_num,"(i5.5)")nstep
534     write(proc_num,"(i5.5)")myrank
535     fname =  trim(root_dir) // "/" &
536           // trim(file_prefix_1st) // file_num // "-" &
537           // trim(file_prefix_2nd) // proc_num // ".dat"
538     open(unit=9,file=trim(fname),action='write',status='replace')
539        do i=1,nptcl_loc
540           write(9,100)ptcl(i)%id,ptcl(i)%mass, &
541                       ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z, &
542                       ptcl(i)%vel%x,ptcl(i)%vel%y,ptcl(i)%vel%z, &
543                       ptcl(i)%dens,ptcl(i)%eng,ptcl(i)%pres
544           100 format(i8,1x,10e25.16e3)
545        end do
546     close(unit=9)
547     nullify(ptcl)
548
549 end subroutine output
550
551 !-------------------------------------------------------------------------
552 !////////////////          S U B R O U T I N E          ////////////////
553 !//////////////// < C H E C K _ C N S R V D _ V A R S > ////////////////
554 !-------------------------------------------------------------------------
555 subroutine check_cnsrvd_vars(fdps_ctrl,psys_num)
556     use fdps_vector
557     use fdps_module
558     use user_defined_types
559     implicit none
560     type(fdps_controller), intent(in) :: fdps_ctrl
561     integer, intent(in) :: psys_num
562     !* Local variables
563     integer :: i,nptcl_loc
564     type(full_particle), dimension(:), pointer :: ptcl
565     type(fdps_f64vec) :: mom_loc,mom
566     real(kind=c_double) :: eng_loc,eng
567
568     !* Get # of local particles
569     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
570
571     !* Get the pointer to full particle data
572     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
573     mom_loc = 0.0d0; eng_loc = 0.0d0
574     do i=1,nptcl_loc
575        mom_loc = mom_loc + ptcl(i)%vel * ptcl(i)%mass
576        eng_loc = eng_loc + ptcl(i)%mass &
577                           *(ptcl(i)%eng &
578                            +0.5d0*ptcl(i)%vel*ptcl(i)%vel)
579     end do
580     nullify(ptcl)
```

```
581
582     !* Reduction & output
583     call fdps_ctrl%get_sum(eng_loc,eng)
584     call fdps_ctrl%get_sum(mom_loc%x,mom%x)
585     call fdps_ctrl%get_sum(mom_loc%y,mom%y)
586     call fdps_ctrl%get_sum(mom_loc%z,mom%z)
587     if (fdps_ctrl%get_rank() == 0) then
588        write(*,100)eng
589        write(*,100)mom%x
590        write(*,100)mom%y
591        write(*,100)mom%z
592        100 format(1es25.16e3)
593     end if
594
595 end subroutine check_cnsrvd_vars
```

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/fortran/p3m` である。まずは、そこに移動する。

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

サンプルコードはユーザ定義型と相互作用関数が実装された `user_defined.F90`、ユーザコードのそれ以外の部分が実装された `f_main.F90`、GCC と intel コンパイラ用の Makefile である `Makefile` と `Makefile.intel` 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派生データ型について記述する。

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 37 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 37: FullParticle 型

```
1    type, public, bind(c) :: nbody_fp !$fdps FP
2       !$fdps copyFromForce nbody_pp_results (pot,pot) (agrv,agrv)
3       !$fdps copyFromForcePM agrv_pm
4       integer(kind=c_long_long) :: id
5       real(kind=c_double) :: m !$fdps charge
6       real(kind=c_double) :: rc !$fdps rsearch
7       type(fdps_f64vec) :: x !$fdps position
```

```
8       type(fdps_f64vec) :: v,v_half
9       type(fdps_f64vec) :: agrv
10      real(kind=c_double) :: pot
11      type(fdps_f32vec) :: agrv_pm
12      real(kind=c_float) :: pot_pm
13    end type nbody_fp
```

この 派生データ型が FullParticle 型であることを示すため、次の指示文を記述している:

```
type, public, bind(c) :: nbody_fp !$fdps FP
```

P^{3}M 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:

```
real(kind=c_double) :: m !$fdps charge
real(kind=c_double) :: rc !$fdps rsearch
type(fdps_f64vec) :: x !$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:

```
!$fdps copyFromForce nbody_pp_results (pot,pot) (agrv,agrv)
!$fdps copyFromForcePM agrv_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 38 に、サンプルコードの EssentialParticleI 型を示す。

Listing 38: EssentialParticleI 型

```
1   type, public, bind(c) :: nbody_ep !$fdps EPI,EPJ
2       !$fdps copyFromFP nbody_fp (id,id) (m,m) (rc,rc) (x,x)
3       integer(kind=c_long_long) :: id
4       real(kind=c_double) :: m !$fdps charge
5       real(kind=c_double) :: rc !$fdps rsearch
6       type(fdps_f64vec) :: x !$fdps position
7   end type nbody_ep
```

まず、ユーザは指示文を用いて、この派生データ型が 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:

```
type, public, bind(c) :: nbody_ep !$fdps EPI,EPJ
```

次に、ユーザはこの派生データ型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:

```
real(kind=c_double) :: m !$fdps charge
real(kind=c_double) :: rc !$fdps rsearch
type(fdps_f64vec) :: x !$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 nbody_fp (id,id) (m,m) (rc,rc) (x,x)
```

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 39 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 39: Force 型

```
1    type, public, bind(c) :: nbody_pp_results !$fdps Force
2       !$fdps clear
3       real(kind=c_double) :: pot
4       type(fdps_f64vec) :: agrv
5    end type nbody_pp_results
```

まず、ユーザはこの派生データ型が 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:

```
type, public, bind(c) :: nbody_pp_results !$fdps Force
```

この派生データ型は 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、サブルーチン 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 40 に示す。

Listing 40: 相互作用関数 calcForceEpEp

```
1   subroutine calc_force_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2      integer(c_int), intent(in), value :: n_ip,n_jp
3      type(nbody_ep), dimension(n_ip), intent(in) :: ep_i
4      type(nbody_ep), dimension(n_jp), intent(in) :: ep_j
5      type(nbody_pp_results), dimension(n_ip), intent(inout) :: f
6      !* Local variables
7      integer(c_int) :: i,j
8      real(c_double) :: rij,rinv,rinv3,xi
9      type(fdps_f64vec) :: dx
10
11     do i=1,n_ip
12        do j=1,n_jp
13           dx%x = ep_i(i)%x%x - ep_j(j)%x%x
14           dx%y = ep_i(i)%x%y - ep_j(j)%x%y
15           dx%z = ep_i(i)%x%z - ep_j(j)%x%z
16           rij  = dsqrt(dx%x * dx%x &
17                        +dx%y * dx%y &
18                        +dx%z * dx%z)
19           if ((ep_i(i)%id == ep_j(j)%id) .and. (rij == 0.0d0)) cycle
20           rinv = 1.0d0/rij
21           rinv3 = rinv*rinv*rinv
22           xi = 2.0d0*rij/ep_i(i)%rc
23           f(i)%pot    = f(i)%pot    + ep_j(j)%m * S2_pcut(xi) * rinv
24           f(i)%agrv%x = f(i)%agrv%x + ep_j(j)%m * S2_fcut(xi) * rinv3 *
                 dx%x
25           f(i)%agrv%y = f(i)%agrv%y + ep_j(j)%m * S2_fcut(xi) * rinv3 *
                 dx%y
26           f(i)%agrv%z = f(i)%agrv%z + ep_j(j)%m * S2_fcut(xi) * rinv3 *
                 dx%z
27        end do
28        !* Self-interaction term
29        f(i)%pot = f(i)%pot - ep_i(i)%m * (208.0d0/(70.0d0*ep_i(i)%rc))
30     end do
31
32  end subroutine calc_force_ep_ep
```

P^3M 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 `rc` 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 = f(i)%pot - ep_i(i)%m * (208.0d0/(70.0d0*ep_i(i)%rc))
```

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 5)]。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、サブルーチン 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 41 に示す。

[注 5)] 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 `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。

Listing 41: 相互作用関数 calcForceEpSp

```
1    subroutine calc_force_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2       integer(c_int), intent(in), value :: n_ip,n_jp
3       type(nbody_ep), dimension(n_ip), intent(in) :: ep_i
4       type(fdps_spj_monopole_cutoff), dimension(n_jp), intent(in) :: ep_j
5       type(nbody_pp_results), dimension(n_ip), intent(inout) :: f
6       !* Local variables
7       integer(c_int) :: i,j
8       real(c_double) :: rij,rinv,rinv3,xi
9       type(fdps_f64vec) :: dx
10
11      do i=1,n_ip
12         do j=1,n_jp
13            dx%x = ep_i(i)%x%x - ep_j(j)%pos%x
14            dx%y = ep_i(i)%x%y - ep_j(j)%pos%y
15            dx%z = ep_i(i)%x%z - ep_j(j)%pos%z
16            rij  = dsqrt(dx%x * dx%x &
17                         +dx%y * dx%y &
18                         +dx%z * dx%z)
19            rinv = 1.0d0/rij
20            rinv3 = rinv*rinv*rinv
21            xi = 2.0d0*rij/ep_i(i)%rc
22            f(i)%pot    = f(i)%pot    + ep_j(j)%mass * S2_pcut(xi) * rinv
23            f(i)%agrv%x = f(i)%agrv%x + ep_j(j)%mass * S2_fcut(xi) * rinv3
                    * dx%x
24            f(i)%agrv%y = f(i)%agrv%y + ep_j(j)%mass * S2_fcut(xi) * rinv3
                    * dx%y
25            f(i)%agrv%z = f(i)%agrv%z + ep_j(j)%mass * S2_fcut(xi) * rinv3
                    * dx%z
26         end do
27      end do
28
29   end subroutine calc_force_ep_sp
```

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 (メインルーチンの `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 (メインルーチン内)
(4) 系全体のエネルギーを計算し、解析解と比較 (メインルーチンの 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 fdps_controller 型オブジェクトの生成

FDPS Fortran インターフェースにおいて、FDPS の API はすべて Fortran 2003 のクラス FDPS_controller のメンバ関数として提供される。このクラスは、インターフェースプログラムの 1 つである FDPS_module.F90 の中の、モジュール fdps_module 内で定義されている。したがって、ユーザは FDPS の API を使用するために、FDPS_controller 型オブジェクトを生成しなければならない。本サンプルコードでは、FDPS_controller 型オブジェクト fdps_ctrl をメインルーチンで生成している:

Listing 42: fdps_controller 型オブジェクトの生成

```
1 subroutine f_main()
2    use fdps_module
3    implicit none
4    !* Local variables
5    type(fdps_controller) :: fdps_ctrl
6
7    ! Do something
8
9 end subroutine f_main
```

ここに示したコードは実際にサンプルコードから必要な部分だけを取り出したものであることに注意して頂きたい。

上記の理由から、以下の説明において、FDPS の API はこのオブジェクトのメンバ関数として呼び出されていることに注意されたい。

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、メインルーチンに記述する。

Listing 43: FDPS の開始

```
1 fdps_ctrl%ps_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、メインルーチンの最後に次のように記述する。

Listing 44: FDPS の終了

```
1 fdps_ctrl%ps_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 45: オブジェクトの生成

```
1 call fdps_ctrl%create_psys(psys_num,'nbody_fp')
2 call fdps_ctrl%create_dinfo(dinfo_num)
3 call fdps_ctrl%create_pm(pm_num)
4 call fdps_ctrl%create_tree(tree_num, &
5                            "Long,nbody_pp_results,nbody_ep,nbody_ep,
                                 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 46: 粒子群オブジェクトの初期化

```
1 call fdps_ctrl%init_psys(psys_num)
```

サンプルコードではメインルーチン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 47: 領域オブジェクトの初期化

```
1 call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
```

サンプルコードではメインルーチン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サブルーチン `setup_NaCl_crystal` の中で行われている:

```
1 call fdps_ctrl%set_boundary_condition(dinfo_num,fdps_bc_periodic_xyz)
2 pos_ll%x = 0.0d0; pos_ll%y = 0.0d0; pos_ll%z = 0.0d0
3 pos_ul%x = 1.0d0; pos_ul%y = 1.0d0; pos_ul%z = 1.0d0
4 call fdps_ctrl%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 `init_tree` を使って、以下のように行う:

Listing 48: ツリーオブジェクトの初期化

```
call fdps_ctrl%init_tree(tree_num,3*nptcl_loc,theta, &
                         n_leaf_limit,n_group_limit)
```

ツリーオブジェクトの API `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 省略可能引数で 、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サブルーチン `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。このサブルーチンは、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派生データ型 `crystal_parameters` のオブジェクト `NaCl_params` を使って渡されている:

```
! In user_defined.F90
type, public, bind(c) :: crystal_parameters
   integer(kind=c_int) :: nptcl_per_side
   type(fdps_f64vec) :: pos_vertex
end type crystal_parameters
! In f_main.F90
type(crystal_parameters) :: NaCl_params
call setup_NaCl_crystal(fdps_ctrl, &
                        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 `decompose_domain_all` を使用する:

Listing 49: 領域分割の実行

```
1 call fdps_ctrl%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 `exchange_particle` を使用する:

Listing 50: 粒子交換の実行

```
1 call fdps_ctrl%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、この作業をメインルーチンで行っている:

Listing 51: 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  nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
4  call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
5  !* [4-2] PP part
6  pfunc_ep_ep = c_funloc(calc_force_ep_ep)
7  pfunc_ep_sp = c_funloc(calc_force_ep_sp)
8  call fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
9                                               pfunc_ep_ep, &
10                                              pfunc_ep_sp, &
11                                              psys_num,    &
12                                              dinfo_num)
13 !* [4-3] PM part
14 call fdps_ctrl%calc_pm_force_all_and_write_back(pm_num,   &
15                                                 psys_num, &
16                                                 dinfo_num)
17 do i=1,nptcl_loc
18    pos32 = ptcl(i)%x
19    call fdps_ctrl%get_pm_potential(pm_num,pos32,ptcl(i)%pot_pm)
20 end do
21 !* [4-4] Compute the total acceleration and potential
```

```
22 do i=1,nptcl_loc
23    ptcl(i)%pot  = ptcl(i)%pot  - ptcl(i)%pot_pm
24    ptcl(i)%agrv = ptcl(i)%agrv - ptcl(i)%agrv_pm
25 end do
```

PP パートの相互作用計算には API `calc_force_all_and_write_back` 、PM パートの相互作用計算には API `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。

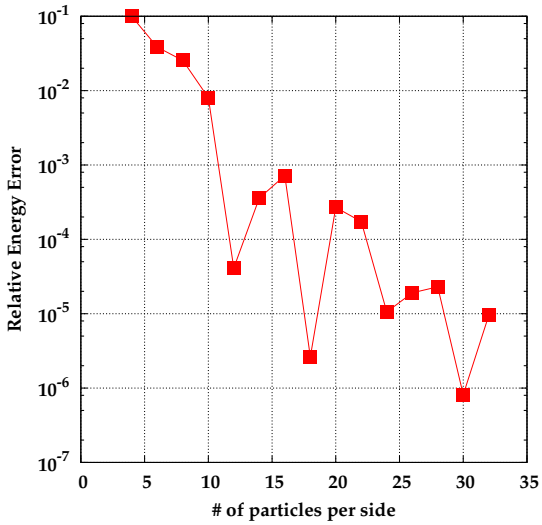

図 3: 1 辺あたりの粒子数とエネルギー相対誤差の関係 (メッシュ数は 16^3、カットオフ半径は 3/16)

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/fortran/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/fortran/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.intel
f_main.F90
ic.F90
leapfrog.F90
macro_defs.h
magi_data/
mathematical_constants.F90
physical_constants.F90
test.py
user_defined.F90
```

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、ic.F90 には初期条件を作成するサブルーチンが実装されている。初期条件は円盤銀河の他、複数用意されている (後述)。leapfrog.F90 には粒子の軌道の時間積分を Leapfrog 法を用いて行うサブルーチンが実装されている。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。f_main.F90 はアプリケーションのメインルーチンが実装されている。mathematical_constants.F90 には数学定数が、physical_constants.F90 には物理定数が定義されている。user_defined.F90 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 FC に使用する Fortran 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 FCFLAGS に Fortran 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。

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, $M = 10^{12}$ M$_\odot$, $r_s = 21.5$ kpc, $r_c = 200$ kpc, $\Delta_c = 10$ kpc)
   (ii) バルジ (King モデル, $M = 5 \times 10^{10}$ M$_\odot$, $r_s = 0.7$ kpc, $W_0 = 5$)
   (iii) thick disk (Sérsic profile, $M = 2.5 \times 10^{10}$ M$_\odot$, $r_s = 3.5$ kpc, $n = 1.5$, $z_d = 1$ kpc, $f = 0.125$)
   (iv) thin disk (exponential disk, $M = 2.5 \times 10^{10}$ M$_\odot$, $r_s = 3.5$ kpc, $z_d = 0.5$ kpc, $f = 0.125$)
   デフォルトモデルでは、2 つの星円盤は bar モードに対して不安定であるため、棒状銀河或いは棒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。
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。

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

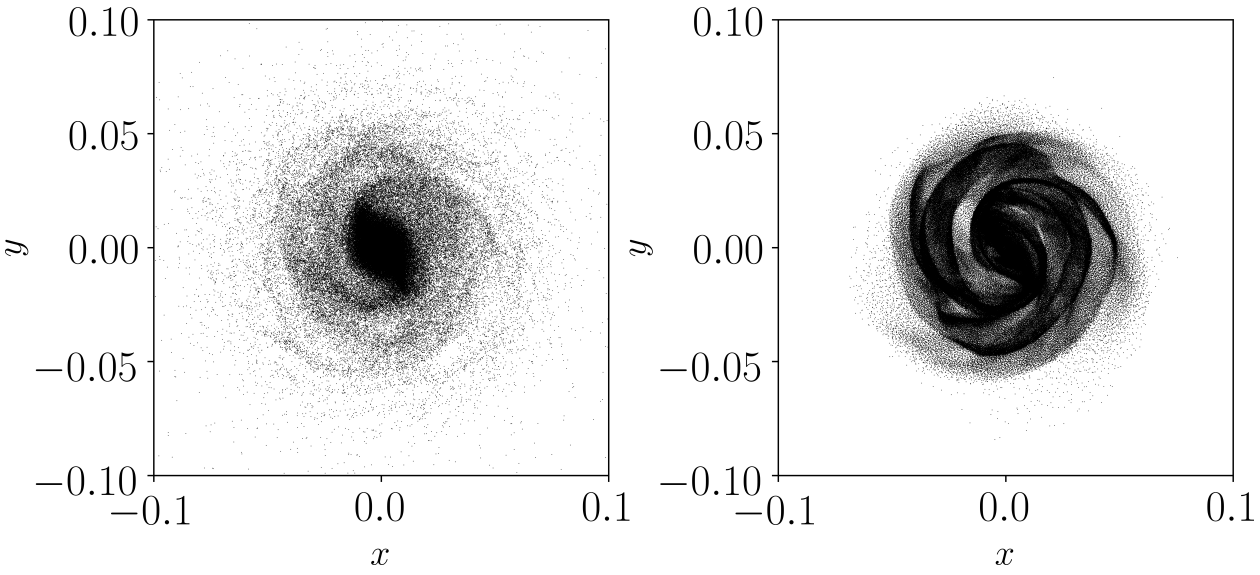

図 4: $T = 0.46$ における星分布 (左) とガス分布 (右) (計算は以下の条件で実施した: N 体粒子数 2^{21}、SPH 粒子数 2^{18}、等温、ガス温度 10^4 K、平均分子量 $\mu = 0.5$)

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 6)]、N_{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:

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

注 6) 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。

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:

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

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

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.F90` 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 派生データ型 `fp_nbody` を, SPH 粒子用に 派生データ型 `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 派生データ型 `force_grav` を、密度計算用に 派生データ型 `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) の計算用に派生データ型 `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 つの派生データ型で兼ねることにし (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 派生データ型 ep_grav を、SPH 計算用に 派生データ型 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 派生データ型 fp_nbody について解説する。この派生データ型には、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 52 に、この派生データ型の実装を示す。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 52: FullParticle 型 (派生データ型 fp_nbody)

```
1    !**** Full particle type
2    type, public, bind(c) :: fp_nbody !$fdps FP
3       !$fdps copyFromForce force_grav (acc,acc) (pot,pot)
4       integer(kind=c_long_long) :: id !$fdps id
5       real(kind=c_double) :: mass !$fdps charge
6       type(fdps_f64vec) :: pos !$fdps position
7       type(fdps_f64vec) :: vel
8       type(fdps_f64vec) :: acc
9       real(kind=c_double) :: pot
10   end type fp_nbody
```

次に、SPH 粒子用の FullParticle 型である 派生データ型 fp_sph について解説する。この派生データ型には、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 53 に、この派生データ型の実装を示す。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 7)] $[h_i]$)、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 種類あるので、それに応じて copyFromForce 指示文も 3 つ用意されている。

Listing 53: FullParticle 型 (派生データ型 fp_sph)

```
1    type, public, bind(c) :: fp_sph !$fdps FP
```

注 7) カーネル関数が 0 になる距離と定義。

```
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       !$fdps copyFromForce force_hydro (acc,acc_hydro) (eng_dot,eng_dot) (
              ent_dot,ent_dot) (dt,dt)
5       integer(kind=c_long_long) :: id !$fdps id
6       real(kind=c_double) :: mass !$fdps charge
7       type(fdps_f64vec) :: pos !$fdps position
8       type(fdps_f64vec) :: vel
9       type(fdps_f64vec) :: acc_grav
10      real(kind=c_double) :: pot_grav
11      type(fdps_f64vec)   :: acc_hydro
12      integer(kind=c_int) :: flag
13      real(kind=c_double) :: dens
14      real(kind=c_double) :: eng
15      real(kind=c_double) :: ent
16      real(kind=c_double) :: pres
17      real(kind=c_double) :: smth
18      real(kind=c_double) :: gradh
19      real(kind=c_double) :: divv
20      type(fdps_f64vec)   :: rotv
21      real(kind=c_double) :: balsw
22      real(kind=c_double) :: snds
23      real(kind=c_double) :: eng_dot
24      real(kind=c_double) :: ent_dot
25      real(kind=c_double) :: dt
26      type(fdps_f64vec)   :: vel_half
27      real(kind=c_double) :: eng_half
28      real(kind=c_double) :: ent_half
29    end type fp_sph
```

7.1.3.2 EssentialParticle 型

まず、重力計算用の EssentialParticle 型である 派生データ型 ep_grav について解説する。この派生データ型には、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 54 に、この派生データ型の実装を示す。 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 54: EssentialParticle 型 (派生データ型 ep_grav)

```
1     type, public, bind(c) :: ep_grav !$fdps EPI,EPJ
2       !$fdps copyFromFP fp_nbody (id,id) (mass,mass) (pos,pos)
3       !$fdps copyFromFP fp_sph (id,id) (mass,mass) (pos,pos)
4       integer(kind=c_long_long) :: id !$fdps id
5       real(kind=c_double) :: mass !$fdps charge
6       type(fdps_f64vec) :: pos !$fdps position
7     end type 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 派生データ型 ep_hydro について解説する。この派生データ型には、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 55 に、この派生データ型の実装を示す。

Listing 55: EssentialParticle 型 (派生データ型 ep_hydro)

```
1    type, public, bind(c) :: 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       integer(kind=c_long_long) :: id !$fdps id
4       type(fdps_f64vec) :: pos !$fdps position
5       type(fdps_f64vec) :: vel
6       real(kind=c_double) :: mass !$fdps charge
7       real(kind=c_double) :: smth !$fdps rsearch
8       real(kind=c_double) :: dens
9       real(kind=c_double) :: pres
10      real(kind=c_double) :: gradh
11      real(kind=c_double) :: snds
12      real(kind=c_double) :: balsw
13   end type ep_hydro
```

7.1.3.3 Force 型

まず、重力計算用の Force 型である 派生データ型 force_grav について解説する。この派生データ型は、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 56 にこの派生データ型の実装を示す。

Listing 56: Force 型 (派生データ型 force_grav)

```
1    type, public, bind(c) :: force_grav !$fdps Force
2       !$fdps clear
3       type(fdps_f64vec) :: acc
4       real(kind=c_double) :: pot
5    end type force_grav
```

次に、密度計算用の Force 型である 派生データ型 force_dens について解説する。この派生データ型は、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 57 に、この派生データ型の実装を示す。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 57: Force 型 (派生データ型 force_dens)

```
1    type, public, bind(c) :: force_dens !$fdps Force
2       !$fdps clear smth=keep
3       integer(kind=c_int) :: flag
4       real(kind=c_double) :: dens
5       real(kind=c_double) :: smth
```

```
6       real(kind=c_double) :: gradh
7       real(kind=c_double) :: divv
8       type(fdps_f64vec) :: rotv
9    end type force_dens
```

最後に、圧力勾配加速度計算用の Force 型である 派生データ型 force_hydro について解説する。この派生データ型は、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 58 に、この派生データ型の実装を示す。

Listing 58: Force 型 (派生データ型 force_hydro)

```
1    type, public, bind(c) :: force_hydro !$fdps Force
2       !$fdps clear
3       type(fdps_f64vec) :: acc
4       real(kind=c_double) :: eng_dot
5       real(kind=c_double) :: ent_dot
6       real(kind=c_double) :: dt
7    end type 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.F90 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 サブルーチン calc_gravity_ep_ep 及び calc_gravity_ep_sp として実装されている。Listing 59 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 59: 相互作用関数 (重力計算用)

```
1  #if defined(ENABLE_PHANTOM_GRAPE_X86)
2     subroutine calc_gravity_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
3  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
4        use omp_lib
5  #endif
6        use phantom_grape_g5_x86
7        implicit none
8        integer(c_int), intent(in), value :: n_ip,n_jp
9        type(ep_grav), dimension(n_ip), intent(in) :: ep_i
10       type(ep_grav), dimension(n_jp), intent(in) :: ep_j
11       type(force_grav), dimension(n_ip), intent(inout) :: f
12       !* Local variables
13       integer(c_int) :: i,j
14       integer(c_int) :: nipipe,njpipe,devid
15       real(c_double), dimension(3,n_ip) :: xi,ai
16       real(c_double), dimension(n_ip) :: pi
```

```
17        real(c_double), dimension(3,n_jp) :: xj
18        real(c_double), dimension(n_jp) :: mj
19
20        nipipe = n_ip
21        njpipe = n_jp
22        do i=1,n_ip
23           xi(1,i) = ep_i(i)%pos%x
24           xi(2,i) = ep_i(i)%pos%y
25           xi(3,i) = ep_i(i)%pos%z
26           ai(1,i) = 0.0d0
27           ai(2,i) = 0.0d0
28           ai(3,i) = 0.0d0
29           pi(i)   = 0.0d0
30        end do
31        do j=1,n_jp
32           xj(1,j) = ep_j(j)%pos%x
33           xj(2,j) = ep_j(j)%pos%y
34           xj(3,j) = ep_j(j)%pos%z
35           mj(j)   = ep_j(j)%mass
36        end do
37  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
38        devid = omp_get_thread_num()
39        ! [IMPORTANT NOTE]
40        !   The subroutine calc_gravity_ep_ep is called by a OpenMP thread
41        !   in the FDPS. This means that here is already in the parallel
                region.
42        !   So, you can use omp_get_thread_num() without !$OMP parallel
                directives.
43        !   If you use them, a nested parallel resions is made and the
                gravity
44        !   calculation will not be performed correctly.
45  #else
46        devid = 0
47  #endif
48        call g5_set_xmjMC(devid, 0, n_jp, xj, mj)
49        call g5_set_nMC(devid, n_jp)
50        call g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip)
51        do i=1,n_ip
52           f(i)%acc%x = f(i)%acc%x + ai(1,i)
53           f(i)%acc%y = f(i)%acc%y + ai(2,i)
54           f(i)%acc%z = f(i)%acc%z + ai(3,i)
55           f(i)%pot   = f(i)%pot   - pi(i)
56        end do
57     end subroutine calc_gravity_ep_ep
58
59     subroutine calc_gravity_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
60  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
61        use omp_lib
62  #endif
63        use phantom_grape_g5_x86
64        implicit none
65        integer(c_int), intent(in), value :: n_ip,n_jp
66        type(ep_grav), dimension(n_ip), intent(in) :: ep_i
67        type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
68        type(force_grav), dimension(n_ip), intent(inout) :: f
```

```
69        !* Local variables
70        integer(c_int) :: i,j
71        integer(c_int) :: nipipe,njpipe,devid
72        real(c_double), dimension(3,n_ip) :: xi,ai
73        real(c_double), dimension(n_ip) :: pi
74        real(c_double), dimension(3,n_jp) :: xj
75        real(c_double), dimension(n_jp) :: mj
76
77        nipipe = n_ip
78        njpipe = n_jp
79        do i=1,n_ip
80           xi(1,i) = ep_i(i)%pos%x
81           xi(2,i) = ep_i(i)%pos%y
82           xi(3,i) = ep_i(i)%pos%z
83           ai(1,i) = 0.0d0
84           ai(2,i) = 0.0d0
85           ai(3,i) = 0.0d0
86           pi(i)   = 0.0d0
87        end do
88        do j=1,n_jp
89           xj(1,j) = ep_j(j)%pos%x
90           xj(2,j) = ep_j(j)%pos%y
91           xj(3,j) = ep_j(j)%pos%z
92           mj(j)   = ep_j(j)%mass
93        end do
94  #if defined(PARTICLE_SIMULATOR_THREAD_PARALLEL) && defined(_OPENMP)
95        devid = omp_get_thread_num()
96        ! [IMPORTANT NOTE]
97        !   The subroutine calc_gravity_ep_sp is called by a OpenMP thread
98        !   in the FDPS. This means that here is already in the parallel
                region.
99        !   So, you can use omp_get_thread_num() without !$OMP parallel
                directives.
100       !   If you use them, a nested parallel resions is made and the
                gravity
101       !   calculation will not be performed correctly.
102 #else
103       devid = 0
104 #endif
105       call g5_set_xmjMC(devid, 0, n_jp, xj, mj)
106       call g5_set_nMC(devid, n_jp)
107       call g5_calculate_force_on_xMC(devid, xi, ai, pi, n_ip)
108       do i=1,n_ip
109          f(i)%acc%x = f(i)%acc%x + ai(1,i)
110          f(i)%acc%y = f(i)%acc%y + ai(2,i)
111          f(i)%acc%z = f(i)%acc%z + ai(3,i)
112          f(i)%pot   = f(i)%pot   - pi(i)
113       end do
114    end subroutine calc_gravity_ep_sp
115 #else
116    subroutine calc_gravity_ep_ep(ep_i,n_ip,ep_j,n_jp,f) bind(c)
117       integer(kind=c_int), intent(in), value :: n_ip,n_jp
118       type(ep_grav), dimension(n_ip), intent(in) :: ep_i
119       type(ep_grav), dimension(n_jp), intent(in) :: ep_j
120       type(force_grav), dimension(n_ip), intent(inout) :: f
```

```
121         !* Local variables
122         integer(kind=c_int) :: i,j
123         real(kind=c_double) :: eps2,poti,r3_inv,r_inv
124         type(fdps_f64vec) :: xi,ai,rij
125         !* Compute force
126         eps2 = eps_grav * eps_grav
127         do i=1,n_ip
128            xi%x = ep_i(i)%pos%x
129            xi%y = ep_i(i)%pos%y
130            xi%z = ep_i(i)%pos%z
131            ai%x = 0.0d0
132            ai%y = 0.0d0
133            ai%z = 0.0d0
134            poti = 0.0d0
135            do j=1,n_jp
136               rij%x  = xi%x - ep_j(j)%pos%x
137               rij%y  = xi%y - ep_j(j)%pos%y
138               rij%z  = xi%z - ep_j(j)%pos%z
139               r3_inv = rij%x*rij%x &
140                      + rij%y*rij%y &
141                      + rij%z*rij%z &
142                      + eps2
143               r_inv  = 1.0d0/dsqrt(r3_inv)
144               r3_inv = r_inv * r_inv
145               r_inv  = r_inv * ep_j(j)%mass
146               r3_inv = r3_inv * r_inv
147               ai%x   = ai%x - r3_inv * rij%x
148               ai%y   = ai%y - r3_inv * rij%y
149               ai%z   = ai%z - r3_inv * rij%z
150               poti   = poti - r_inv
151            end do
152            f(i)%acc%x = f(i)%acc%x + ai%x
153            f(i)%acc%y = f(i)%acc%y + ai%y
154            f(i)%acc%z = f(i)%acc%z + ai%z
155            f(i)%pot   = f(i)%pot   + poti
156         end do
157      end subroutine calc_gravity_ep_ep
158
159      subroutine calc_gravity_ep_sp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
160         integer(kind=c_int), intent(in), value :: n_ip,n_jp
161         type(ep_grav), dimension(n_ip), intent(in) :: ep_i
162         type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
163         type(force_grav), dimension(n_ip), intent(inout) :: f
164         !* Local variables
165         integer(kind=c_int) :: i,j
166         real(kind=c_double) :: eps2,poti,r3_inv,r_inv
167         type(fdps_f64vec) :: xi,ai,rij
168         !* Compute force
169         eps2 = eps_grav * eps_grav
170         do i=1,n_ip
171            xi%x = ep_i(i)%pos%x
172            xi%y = ep_i(i)%pos%y
173            xi%z = ep_i(i)%pos%z
174            ai%x = 0.0d0
175            ai%y = 0.0d0
```

```
176           ai%z = 0.0d0
177           poti = 0.0d0
178           do j=1,n_jp
179              rij%x  = xi%x - ep_j(j)%pos%x
180              rij%y  = xi%y - ep_j(j)%pos%y
181              rij%z  = xi%z - ep_j(j)%pos%z
182              r3_inv = rij%x*rij%x &
183                     + rij%y*rij%y &
184                     + rij%z*rij%z &
185                     + eps2
186              r_inv  = 1.0d0/dsqrt(r3_inv)
187              r3_inv = r_inv * r_inv
188              r_inv  = r_inv * ep_j(j)%mass
189              r3_inv = r3_inv * r_inv
190              ai%x   = ai%x - r3_inv * rij%x
191              ai%y   = ai%y - r3_inv * rij%y
192              ai%z   = ai%z - r3_inv * rij%z
193              poti   = poti - r_inv
194           end do
195           f(i)%acc%x = f(i)%acc%x + ai%x
196           f(i)%acc%y = f(i)%acc%y + ai%y
197           f(i)%acc%z = f(i)%acc%z + ai%z
198           f(i)%pot   = f(i)%pot   + poti
199        end do
200     end subroutine calc_gravity_ep_sp
201 #endif
```

7.1.4.2 密度計算

密度計算用の相互作用関数は サブルーチン `calc_density` として実装されている。Listing 60 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 do-enddo 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 $h_{\mathrm{max,alw}} \equiv$ `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 `f_main.F90` の サブルーチン `calc_density_wrapper` で行われている。このサブルーチンの詳細は第 7.1.5 節で行う。

無限 do-enddo 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 60: 相互作用関数 (密度計算用)

```
1     subroutine calc_density(ep_i,n_ip,ep_j,n_jp,f) bind(c)
2        integer(kind=c_int), intent(in), value :: n_ip,n_jp
3        type(ep_hydro), dimension(n_ip), intent(in) :: ep_i
4        type(ep_hydro), dimension(n_jp), intent(in) :: ep_j
5        type(force_dens), dimension(n_ip), intent(inout) :: f
6        !* Local parameters
7        real(kind=c_double), parameter :: eps=1.0d-6
8        !* Local variables
9        integer(kind=c_int) :: i,j
10       integer(kind=c_int) :: n_unchanged
11       real(kind=c_double) :: M,M_trgt
12       real(kind=c_double) :: dens,drho_dh
13       real(kind=c_double) :: h,h_max_alw,h_L,h_U,dh,dh_prev
14       type(fdps_f64vec) :: dr,dv,gradW_i
15
16 #if defined(ENABLE_VARIABLE_SMOOTHING_LENGTH)
17       real(kind=c_double), dimension(n_jp) :: mj,rij
18       M_trgt = mass_avg * N_neighbor
19       do i=1,n_ip
20           dens = 0.0d0
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 = 0.0d0
26           h_U = h_max_alw
27           dh_prev = 0.0d0
28           n_unchanged = 0
29           ! Software cache
30           do j=1,n_jp
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) = dsqrt(dr%x * dr%x &
36                             +dr%y * dr%y &
37                             +dr%z * dr%z)
38           end do
39           iteration_loop: do
40               ! Calculate density
41               dens = 0.0d0
42               do j=1,n_jp
43                  dens = dens + mj(j) * W(rij(j), h)
44               end do
45               ! Check if the current value of the smoohting length
                       satisfies
46               ! Eq.(5) in Springel (2005).
47               M = 4.0d0 * pi * h * h * h * dens / 3.0d0
48               if ((h < h_max_alw) .and. (dabs(M/M_trgt - 1.0d0) < eps))
                       then
49                   ! In this case, Eq.(5) holds within a specified accuracy
                           .
```

```
50                      f(i)%flag = 1
51                      f(i)%dens = dens
52                      f(i)%smth = h
53                      exit iteration_loop
54                  end if
55                  if (((h == h_max_alw) .and. (M < M_trgt)) .or. (n_unchanged
                        == 4)) then
56                      ! In this case, we skip this particle forcibly.
57                      ! In order to determine consistently the density
58                      ! and the smoohting length for this particle,
59                      ! we must re-perform calcForceAllAndWriteBack().
60                      f(i)%flag = 0
61                      f(i)%dens = dens
62                      f(i)%smth = h_max_alw
63                      exit iteration_loop
64                  end if
65                  ! Update h_L & h_U
66                  if (M < M_trgt) then
67                     if (h_L < h) h_L = h
68                  else if (M_trgt < M) then
69                     if (h < h_U) h_U = h
70                  end if
71                  dh = h_U - h_L
72                  if (dh == dh_prev) then
73                     n_unchanged = n_unchanged + 1
74                  else
75                     dh_prev = dh
76                     n_unchanged = 0
77                  end if
78                  ! Update smoothing length
79                  h = ((3.0d0 * M_trgt)/(4.0d0 * pi * dens))**(1.0d0/3.0d0)
80                  if ((h <= h_L) .or. (h == h_U)) then
81                     ! In this case, we switch to the bisection search.
82                     ! The inclusion of '=' in the if statement is very
83                     ! important to escape a limit cycle.
84                     h = 0.5d0 * (h_L + h_U)
85                  else if (h_U < h) then
86                     h = h_U
87                  end if
88              end do iteration_loop
89              ! Calculate grad-h term
90              if (f(i)%flag == 1) then
91                  drho_dh = 0.0d0
92                  do j=1,n_jp
93                     drho_dh = drho_dh + mj(j) * dWdh(rij(j), h)
94                  end do
95                  f(i)%gradh = 1.0d0 / (1.0d0 + (h * drho_dh) / (3.0d0 * dens)
                        )
96              else
97                  f(i)%gradh = 1.0d0 ! dummy value
98              end if
99              ! Compute \div v & \rot v for Balsara switch
100 #if defined(USE_BALSARA_SWITCH)
101             do j=1,n_jp
102                dr%x = ep_i(i)%pos%x - ep_j(j)%pos%x
```

```
103               dr%y = ep_i(i)%pos%y - ep_j(j)%pos%y
104               dr%z = ep_i(i)%pos%z - ep_j(j)%pos%z
105               dv%x = ep_i(i)%vel%x - ep_j(j)%vel%x
106               dv%y = ep_i(i)%vel%y - ep_j(j)%vel%y
107               dv%z = ep_i(i)%vel%z - ep_j(j)%vel%z
108               gradW_i = gradW(dr, f(i)%smth)
109               f(i)%divv = f(i)%divv - mj(j) * (dv%x * gradW_i%x &
110                                               +dv%y * gradW_i%y &
111                                               +dv%z * gradW_i%z)
112               f(i)%rotv%x = f(i)%rotv%x - mj(j) * (dv%y * gradW_i%z - dv%z
                        * gradW_i%y)
113               f(i)%rotv%y = f(i)%rotv%y - mj(j) * (dv%z * gradW_i%x - dv%x
                        * gradW_i%z)
114               f(i)%rotv%z = f(i)%rotv%z - mj(j) * (dv%x * gradW_i%y - dv%y
                        * gradW_i%x)
115            end do
116            f(i)%divv   = f(i)%divv   / f(i)%dens
117            f(i)%rotv%x = f(i)%rotv%x / f(i)%dens
118            f(i)%rotv%y = f(i)%rotv%y / f(i)%dens
119            f(i)%rotv%z = f(i)%rotv%z / f(i)%dens
120 #endif
121         end do
122 #else
123         double precision :: mj,rij
124         do i=1,n_ip
125            f(i)%dens = 0.0d0
126            do j=1,n_jp
127               dr%x = ep_j(j)%pos%x - ep_i(i)%pos%x
128               dr%y = ep_j(j)%pos%y - ep_i(i)%pos%y
129               dr%z = ep_j(j)%pos%z - ep_i(i)%pos%z
130               rij = dsqrt(dr%x * dr%x &
131                           +dr%y * dr%y &
132                           +dr%z * dr%z)
133               f(i)%dens = f(i)%dens &
134                         + ep_j(j)%mass * W(rij,ep_i(i)%smth)
135            end do
136            f(i)%smth = ep_i(i)%smth
137            f(i)%gradh = 1.0d0
138            ! Compute \div v & \rot v for Balsara switch
139 #if defined(USE_BALSARA_SWITCH)
140            do j=1,n_jp
141               mj = ep_j(j)%mass
142               dr%x = ep_i(i)%pos%x - ep_j(j)%pos%x
143               dr%y = ep_i(i)%pos%y - ep_j(j)%pos%y
144               dr%z = ep_i(i)%pos%z - ep_j(j)%pos%z
145               dv%x = ep_i(i)%vel%x - ep_j(j)%vel%x
146               dv%y = ep_i(i)%vel%y - ep_j(j)%vel%y
147               dv%z = ep_i(i)%vel%z - ep_j(j)%vel%z
148               gradW_i = gradW(dr, f(i)%smth)
149               f(i)%divv = f(i)%divv - mj * (dv%x * gradW_i%x &
150                                            +dv%y * gradW_i%y &
151                                            +dv%z * gradW_i%z)
152               f(i)%rotv%x = f(i)%rotv%x - mj * (dv%y * gradW_i%z - dv%z *
                        gradW_i%y)
153               f(i)%rotv%y = f(i)%rotv%y - mj * (dv%z * gradW_i%x - dv%x *
```

```
                    gradW_i%z)
154             f(i)%rotv%z = f(i)%rotv%z - mj * (dv%x * gradW_i%y - dv%y *
                    gradW_i%x)
155          end do
156          f(i)%divv   = f(i)%divv   / f(i)%dens
157          f(i)%rotv%x = f(i)%rotv%x / f(i)%dens
158          f(i)%rotv%y = f(i)%rotv%y / f(i)%dens
159          f(i)%rotv%z = f(i)%rotv%z / f(i)%dens
160 #endif
161       end do
162 #endif
```

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 サブルーチン calc_hydro_force として実装されている。Listing 61 に、実装を示す。このサブルーチンでは、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 派生データ型 fp_sph の説明を参照のこと)。

Listing 61: 相互作用関数 (圧力勾配加速度計算用)

```
 1    !**** Interaction function
 2    subroutine calc_hydro_force(ep_i,n_ip,ep_j,n_jp,f) bind(c)
 3       integer(kind=c_int), intent(in), value :: n_ip,n_jp
 4       type(ep_hydro), dimension(n_ip), intent(in) :: ep_i
 5       type(ep_hydro), dimension(n_jp), intent(in) :: ep_j
 6       type(force_hydro), dimension(n_ip), intent(inout) :: f
 7       !* Local variables
 8       integer(kind=c_int) :: i,j
 9       real(kind=c_double) :: mass_i,mass_j,smth_i,smth_j, &
10                              dens_i,dens_j,pres_i,pres_j, &
11                              gradh_i,gradh_j,balsw_i,balsw_j, &
12                              snds_i,snds_j
13       real(kind=c_double) :: povrho2_i,povrho2_j, &
14                              v_sig_max,dr_dv,w_ij,v_sig,AV
15       type(fdps_f64vec) :: pos_i,pos_j,vel_i,vel_j, &
16                            dr,dv,gradW_i,gradW_j,gradW_ij
17       do i=1,n_ip
18          !* Zero-clear
19          v_sig_max = 0.0d0
20          !* Extract i-particle info.
21          pos_i = ep_i(i)%pos
22          vel_i = ep_i(i)%vel
23          mass_i  = ep_i(i)%mass
24          smth_i  = ep_i(i)%smth
25          dens_i  = ep_i(i)%dens
26          pres_i  = ep_i(i)%pres
27          gradh_i = ep_i(i)%gradh
28          balsw_i = ep_i(i)%balsw
29          snds_i  = ep_i(i)%snds
30          povrho2_i = pres_i/(dens_i*dens_i)
31          do j=1,n_jp
32             !* Extract j-particle info.
```

```
33              pos_j%x = ep_j(j)%pos%x
34              pos_j%y = ep_j(j)%pos%y
35              pos_j%z = ep_j(j)%pos%z
36              vel_j%x = ep_j(j)%vel%x
37              vel_j%y = ep_j(j)%vel%y
38              vel_j%z = ep_j(j)%vel%z
39              mass_j  = ep_j(j)%mass
40              smth_j  = ep_j(j)%smth
41              dens_j  = ep_j(j)%dens
42              pres_j  = ep_j(j)%pres
43              gradh_j = ep_j(j)%gradh
44              balsw_j = ep_j(j)%balsw
45              snds_j  = ep_j(j)%snds
46              povrho2_j = pres_j/(dens_j*dens_j)
47              !* Compute dr & dv
48              dr%x = pos_i%x - pos_j%x
49              dr%y = pos_i%y - pos_j%y
50              dr%z = pos_i%z - pos_j%z
51              dv%x = vel_i%x - vel_j%x
52              dv%y = vel_i%y - vel_j%y
53              dv%z = vel_i%z - vel_j%z
54              !* Compute the signal velocity
55              dr_dv = dr%x * dv%x + dr%y * dv%y + dr%z * dv%z
56              if (dr_dv < 0.0d0) then
57                 w_ij = dr_dv / sqrt(dr%x * dr%x + dr%y * dr%y + dr%z * dr%z
                        )
58              else
59                 w_ij = 0.0d0
60              end if
61              v_sig = snds_i + snds_j - 3.0d0 * w_ij
62              v_sig_max = max(v_sig_max, v_sig)
63              !* Compute the artificial viscosity
64              AV = - 0.5d0*v_sig*w_ij / (0.5d0*(dens_i+dens_j)) * 0.5d0*(
                       balsw_i+balsw_j)
65              !* Compute the average of the gradients of kernel
66              gradW_i  = gradW(dr,smth_i)
67              gradW_j  = gradW(dr,smth_j)
68              gradW_ij%x = 0.5d0 * (gradW_i%x + gradW_j%x)
69              gradW_ij%y = 0.5d0 * (gradW_i%y + gradW_j%y)
70              gradW_ij%z = 0.5d0 * (gradW_i%z + gradW_j%z)
71              !* Compute the acceleration and the heating rate
72              f(i)%acc%x = f(i)%acc%x - mass_j*(gradh_i * povrho2_i *
                       gradW_i%x &
73                                             +gradh_j * povrho2_j *
                                                      gradW_j%x &
74                                             +AV * gradW_ij%x)
75              f(i)%acc%y = f(i)%acc%y - mass_j*(gradh_i * povrho2_i *
                       gradW_i%y &
76                                             +gradh_j * povrho2_j *
                                                      gradW_j%y &
77                                             +AV * gradW_ij%y)
78              f(i)%acc%z = f(i)%acc%z - mass_j*(gradh_i * povrho2_i *
                       gradW_i%z &
79                                             +gradh_j * povrho2_j *
                                                      gradW_j%z &
```

```
80                                                  +AV * gradW_ij%z)
81              f(i)%eng_dot = f(i)%eng_dot
                                                                    &
82                           + mass_j * gradh_i * povrho2_i * (dv%x * gradW_i%
                                    x  &
83                                                                 +dv%y * gradW_i%
                                                                         y  &
84                                                                 +dv%z * gradW_i%
                                                                         z) &
85                           + mass_j * 0.5d0 * AV * (dv%x * gradW_ij%x
                                                  &
86                                                                   +dv%y * gradW_ij%y
                                                                           &
87                                                                   +dv%z * gradW_ij%z)
88              f(i)%ent_dot = f(i)%ent_dot                                  &
89                           + 0.5 * mass_j * AV * (dv%x * gradW_ij%x &
90                                                 +dv%y * gradW_ij%y &
91                                                 +dv%z * gradW_ij%z)
92           end do
93           f(i)%ent_dot = f(i)%ent_dot                        &
94                        * (specific_heat_ratio - 1.0d0) &
95                        / dens_i**(specific_heat_ratio - 1.0d0)
96           f(i)%dt = CFL_hydro*2.0d0*smth_i/v_sig_max
97         end do
```

7.1.5 プログラム本体

本節では、主に f_main.F90 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.F90 のサブルーチン 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サブルーチン内部で生成される。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.F90 のサブルーチン 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.F90 のサブルーチン Evrard_test_IC で行われる。作成方法は 2 つあり、サブルーチン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.F90` のサブルーチン `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 `fdps_controller` 型オブジェクトの生成

ユーザはFDPSのAPIを使用するために、`FDPS_controller` 型オブジェクトを生成しなければならない。本サンプルコードでは、`FDPS_controller` 型オブジェクト `fdps_ctrl` をメインルーチンで生成している:

Listing 62: `fdps_controller` 型オブジェクトの生成

```
1 subroutine f_main()
2    use fdps_module
3    implicit none
4    !* Local variables
5    type(fdps_controller) :: fdps_ctrl
6
7    ! Do something
8
9 end subroutine f_main
```

ここに示したコードは実際にサンプルコードから必要な部分だけを取り出したものであることに注意して頂きたい。上記の理由から、以下の説明において、FDPSのAPIはこのオブジェクトのメンバ関数として呼び出されていることに注意されたい。

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、メインルーチンに記述する。

Listing 63: FDPSの開始

```
1 call fdps_ctrl%ps_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、メインルーチンの最後に次のように記述する。

Listing 64: FDPS の終了

```
1 call fdps_ctrl%ps_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 65: 粒子群オブジェクトの生成・初期化

```
1 call fdps_ctrl%create_psys(psys_num_nbody,'fp_nbody')
2 call fdps_ctrl%init_psys(psys_num_nbody)
3 call fdps_ctrl%create_psys(psys_num_sph,'fp_sph')
4 call fdps_ctrl%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 66: 領域情報オブジェクトの生成・初期化

```
1 call fdps_ctrl%create_dinfo(dinfo_num)
2 call fdps_ctrl%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 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 67: ツリーオブジェクトの生成・初期化

```
1     nptcl_loc_sph    = max(fdps_ctrl%get_nptcl_loc(psys_num_sph),1)
2     nptcl_loc_nbody = fdps_ctrl%get_nptcl_loc(psys_num_nbody)
3     nptcl_loc_all    = nptcl_loc_nbody + nptcl_loc_sph
4     !** tree for gravity calculation
5     call fdps_ctrl%create_tree(tree_num_grav, &
6                                "Long,force_grav,ep_grav,ep_grav,Monopole")
7     call fdps_ctrl%init_tree(tree_num_grav, 3*nptcl_loc_all, theta, &
8                              n_leaf_limit, n_group_limit)
9     !** tree for the density calculation
10    call fdps_ctrl%create_tree(tree_num_dens, &
11                               "Short,force_dens,ep_hydro,ep_hydro,Gather")
12    call fdps_ctrl%init_tree(tree_num_dens, 3*nptcl_loc_sph, theta, &
13                             n_leaf_limit, n_group_limit)
14
15    !** tree for the hydrodynamic force calculation
16    call fdps_ctrl%create_tree(tree_num_hydro, &
17                               "Short,force_hydro,ep_hydro,ep_hydro,
                                     Symmetry")
18    call fdps_ctrl%init_tree(tree_num_hydro, 3*nptcl_loc_sph, theta, &
19                             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 サブルーチン setup_IC で行われる。このサブルーチンはマクロ INITIAL_CONDITION の値に応じて、内部でさらに別のサブルーチンを呼び出しており、呼び出されるサブルーチン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 68: 初期条件の設定

```
1 call 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 サブルーチン 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、メインルーチン f_main() で呼び出されている サブルーチン 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 collect_sample_particle と decompose_domain を併用する必要がある。まず、API 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 decompose_domain で領域分割を実行する。

Listing 69: 領域分割の実行

```
1 call fdps_ctrl%collect_sample_particle(dinfo_num, psys_num_nbody, clear)
2 call fdps_ctrl%collect_sample_particle(dinfo_num, psys_num_sph, unclear)
3 call fdps_ctrl%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 exchange_particle を使用する:

Listing 70: 粒子交換の実行

```
1 call fdps_ctrl%exchange_particle(psys_num_nbody,dinfo_num)
2 call fdps_ctrl%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 71: 相互作用計算の実行

```
1     !** Gravity calculation
2     t_start = fdps_ctrl%get_wtime()
3  #if defined(ENABLE_GRAVITY_INTERACT)
4     call fdps_ctrl%set_particle_local_tree(tree_num_grav, psys_num_nbody)
5     call fdps_ctrl%set_particle_local_tree(tree_num_grav, psys_num_sph,
            unclear)
6     pfunc_ep_ep = c_funloc(calc_gravity_ep_ep)
7     pfunc_ep_sp = c_funloc(calc_gravity_ep_sp)
8     call fdps_ctrl%calc_force_making_tree(tree_num_grav, &
9                                           pfunc_ep_ep,   &
10                                          pfunc_ep_sp,   &
11                                          dinfo_num)
12    nptcl_loc_nbody = fdps_ctrl%get_nptcl_loc(psys_num_nbody)
13    call fdps_ctrl%get_psys_fptr(psys_num_nbody, ptcl_nbody)
14    do i=1,nptcl_loc_nbody
15        call fdps_ctrl%get_force(tree_num_grav, i, f_grav)
16        ptcl_nbody(i)%acc%x = f_grav%acc%x
17        ptcl_nbody(i)%acc%y = f_grav%acc%y
18        ptcl_nbody(i)%acc%z = f_grav%acc%z
19        ptcl_nbody(i)%pot   = f_grav%pot
20    end do
21    offset = nptcl_loc_nbody
22    nptcl_loc_sph = fdps_ctrl%get_nptcl_loc(psys_num_sph)
23    call fdps_ctrl%get_psys_fptr(psys_num_sph, ptcl_sph)
24    do i=1,nptcl_loc_sph
25        call fdps_ctrl%get_force(tree_num_grav, i + offset, f_grav)
26        ptcl_sph(i)%acc_grav%x = f_grav%acc%x
27        ptcl_sph(i)%acc_grav%y = f_grav%acc%y
28        ptcl_sph(i)%acc_grav%z = f_grav%acc%z
29        ptcl_sph(i)%pot_grav   = f_grav%pot
30    end do
31 #endif
32    t_grav = fdps_ctrl%get_wtime() - t_start
33    !** SPH calculations
34    t_start = fdps_ctrl%get_wtime()
35 #if defined(ENABLE_HYDRO_INTERACT)
36    call calc_density_wrapper(psys_num_sph, dinfo_num, tree_num_dens)
37    call set_entropy(psys_num_sph)
38    call set_pressure(psys_num_sph)
39    pfunc_ep_ep = c_funloc(calc_hydro_force)
40    call fdps_ctrl%calc_force_all_and_write_back(tree_num_hydro, &
41                                                 pfunc_ep_ep,    &
42                                                 psys_num_sph,   &
43                                                 dinfo_num)
44 #endif
45    t_hydro = fdps_ctrl%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 `set_particle_local_tree` と `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 `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 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 get_force を使う。この API は引数に整数 i を取り、API 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 calc_force_all_and_write_back が使用できる。圧力勾配加速度に関しては、サブルーチン f_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、これをサブルーチン 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 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 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派生データ型 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 72: サブルーチン calc_density_wrapper の実装

```
1  subroutine calc_density_wrapper(psys_num,dinfo_num,tree_num)
2     use fdps_vector
3     use fdps_module
4     use user_defined_types
5     implicit none
6     integer, intent(in) :: psys_num,dinfo_num,tree_num
7     !* Local variables
8     integer :: i,nptcl_loc,nptcl_glb
9     integer :: n_compl_loc,n_compl
10    type(fdps_controller) :: fdps_ctrl
11    type(fp_sph), dimension(:), pointer :: ptcl
12    type(c_funptr) :: pfunc_ep_ep
13
14 #if defined(ENABLE_VARIABLE_SMOOTHING_LENGTH)
15    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
16    nptcl_glb = fdps_ctrl%get_nptcl_glb(psys_num)
17    call fdps_ctrl%get_psys_fptr(psys_num, ptcl)
18    pfunc_ep_ep = c_funloc(calc_density)
19    ! Determine the density and the smoothing length
20    ! so that Eq.(6) in Springel (2005) holds within a specified accuracy.
21    do
22        ! Increase smoothing length
23        do i=1,nptcl_loc
24            ptcl(i)%smth = scf_smth * ptcl(i)%smth
25        end do
```

```
26          ! Compute density, etc.
27          call fdps_ctrl%calc_force_all_and_write_back(tree_num,     &
28                                                       pfunc_ep_ep, &
29                                                       psys_num,    &
30                                                       dinfo_num)
31          ! Check convergence
32          n_compl_loc = 0; n_compl = 0
33          do i=1,nptcl_loc
34              if (ptcl(i)%flag == 1) n_compl_loc = n_compl_loc + 1
35          end do
36          call fdps_ctrl%get_sum(n_compl_loc, n_compl)
37          if (n_compl == nptcl_glb) exit
38      end do
39      !* Release the pointer
40      nullify(ptcl)
41  #else
42      pfunc_ep_ep = c_funloc(calc_density)
43      call fdps_ctrl%calc_force_all_and_write_back(tree_num,     &
44                                                   pfunc_ep_ep, &
45                                                   psys_num,    &
46                                                   dinfo_num)
47  #endif
48
49  end subroutine calc_density_wrapper
```

サブルーチン set_entropy は、初期条件作成後 1 回だけ呼び出されるサブルーチンで、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、サブルーチン calc_density_wrapper の後に配置されている。サブルーチン 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 $D(\cdot)$ オペレータはサブルーチン full_drift、粒子速度を時間推進する $K(\cdot)$ オペレータはサブルーチン 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。