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 計算科学研究機構 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)

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 言語を用いて記述する。

3 入門：サンプルコードを動かしてみよう

本節では、まずはじめに、FDPS および FDPS 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 4.4.5 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 4.4.5 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 1.8.1 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 1.8.1 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 1.8.1 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

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ている。

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、標準出力は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。

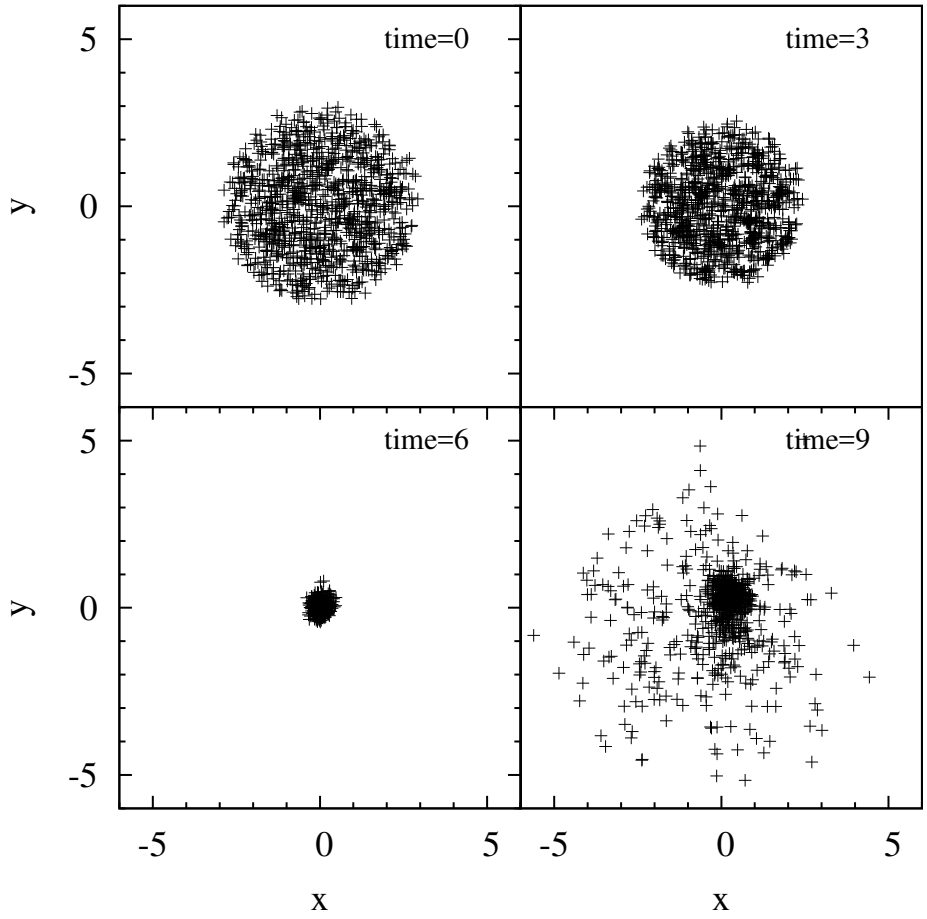

図 1:

```
$ 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。

3.4.2 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ている。

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、標準出力は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。

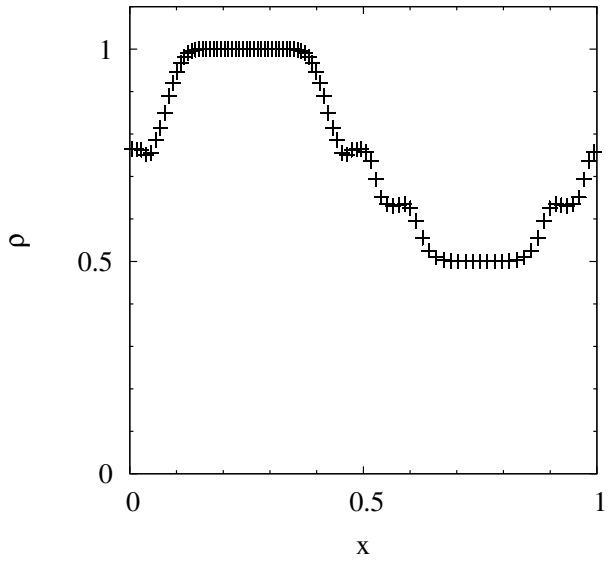

図 2: 衝撃波管問題の時刻 $t = 40$ における密度分布

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

```
13    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 Fortran インターフェースにおいて、この指示は、派生データ型名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) (eps,eps) (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, eps=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 関数 calcForceEpEp

ユーザーは粒子間相互作用の仕方を記述した関数 calcForceEpEp を記述しなければならない。関数 calcForceEpEp には、粒子-粒子相互作用計算の具体的な内容を書く必要があり、Fortran のサブルーチンとして実装しなければならない。Listing 2 に、本サンプルコードの関数 calcForceEpEp を示す (user_defined.F90 を参照)。

Listing 2: 関数 calcForceEpEp

```
1    !**** Interaction function (particle-particle)
2    subroutine calc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
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      do i=1,n_ip
14         eps2 = ep_i(i)%eps * ep_i(i)%eps
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
```

本サンプルコードでは、サブルーチン calc_gravity_pp 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 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関数 calcForceEpSp を記述しなければならない。関数 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、Fortran のサブルーチンとして実装しなければならない。Listing 3 に、本サンプルコードの関数 calcForceEpSp を示す (user_defined.F90 を参照)。

Listing 3: 関数 calcForceEpSp

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

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

本サンプルコードでは、サブルーチン calc_gravity_psp 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 (メインプログラム) を持たない。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

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

```
subroutine foo(fdps_ctrl,psys_num)
   use fdps_vector
   use fdps_module
   use user_defined_types
   implicit none
   type(fdps_controller), intent(IN) :: fdps_ctrl
   integer, intent(IN) :: psys_num
   !* Local variables
   integer :: i,nptcl_loc
   type(full_particle), dimension(:), pointer :: ptcl

   !* Set # of local particles
   call fdps_ctrl%set_nptcl_loc(psys_num,nptcl_loc)

   !* Get the pointer to full particle data
   call fdps_ctrl%get_psys_fptr(psys_num,ptcl)

   !* Initialize particle data
   do i=1,nptcl_loc
      ptcl(i)%pos = ! Do something
   end do

   !* Release the pointer
   nullify(ptcl)

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_pp)
11    pfunc_ep_sp = c_funloc(calc_gravity_psp)
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_pp` と `calc_gravity_psp` 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、関数 `kick` と関数 `drift` として実装している。

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

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

```
1 !* Leapfrog: Kick-Drift
2 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

[注 4)] C 言語方式で記述されたアドレス情報のこと。

```
!* Leapfrog: Kick
call kick(fdps_ctrl,psys_num,0.5d0*dt)
```

4.1.3.6 粒子データの更新

上記で説明したkickやdrift等のサブルーチンで、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

```
subroutine foo(fdps_ctrl,psys_num)
   use fdps_vector
   use fdps_module
   use user_defined_types
   implicit none
   type(fdps_controller), intent(IN) :: fdps_ctrl
   integer, intent(IN) :: psys_num
   !* Local variables
   integer :: i,nptcl_loc
   type(full_particle), dimension(:), pointer :: ptcl

   !* Get # of local particles
   nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)

   !* Get the pointer to full particle data
   call fdps_ctrl%get_psys_fptr(psys_num,ptcl)

   !* Initialize or update particle data
   do i=1,nptcl_loc
      ptcl(i)%pos = ! Do something
   end do

   !* Release the pointer
   nullify(ptcl)

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

```
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 dens_force (dens,dens)
4       !$fdps copyFromForce hydro_force (acc,acc) (eng_dot,eng_dot) (dt,dt)
5       real(kind=c_double) :: mass !$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:

```
!$fdps copyFromForce dens_force (dens,dens)
!$fdps copyFromForce hydro_force (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
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) :: dens_force !$fdps Force
3        !$fdps clear smth=keep
4        real(kind=c_double) :: dens
5        real(kind=c_double) :: smth
6     end type dens_force
7
8     type, public, bind(c) :: hydro_force !$fdps Force
9        !$fdps clear
10       type(fdps_f64vec) :: acc
11       real(kind=c_double) :: eng_dot
12       real(kind=c_double) :: dt
13    end type hydro_force
```

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) :: dens_force !$fdps Force
type, public, bind(c) :: hydro_force !$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 `dens_force` 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、メンバ関数 `clear` で `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 関数 calcForceEpEp

ユーザーは粒子間相互作用の仕方を記述した関数 calcForceEpEp を記述しなければならない。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り、Fortran のサブルーチンとして実装しなければならない。Listing 22 に、本サンプルコードの関数 calcForceEpEp 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(dens_force), 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(hydro_force), 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.
```

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、関数 calcForceEpEp は 2 つ記述する必要がある。いずれの場合にも、関数の仮引数は、EssentialParticleI の配列、EssentialParticleI の個数、EssentialParticleJ の配列、EssentialParticleJ の個数、Force 型の配列である。

4.2.3 プログラム本体

本節では、FDPS を用いて SPH 計算を行う際に、メイン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、メイン関数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 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、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(dens_tree_num, &
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(hydro_tree_num, &
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/0040.txt" u 3:9
```

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)

```
1  !==============================
2  !   MODULE: User defined types
3  !==============================
4  module user_defined_types
5     use, intrinsic :: iso_c_binding
6     use fdps_vector
7     use fdps_super_particle
8     implicit none
9
10    !**** Full particle type
11    type, public, bind(c) :: full_particle !$fdps FP,EPI,EPJ,Force
12       !$fdps copyFromForce full_particle (pot,pot) (acc,acc)
13       !$fdps copyFromFP full_particle (id,id) (mass,mass) (eps,eps) (pos,
               pos)
14       !$fdps clear id=keep, mass=keep, eps=keep, pos=keep, vel=keep
15       integer(kind=c_long_long) :: id
16       real(kind=c_double)  mass !$fdps charge
17       real(kind=c_double) :: eps
18       type(fdps_f64vec) :: pos !$fdps position
19       type(fdps_f64vec) :: vel !$fdps velocity
20       real(kind=c_double) :: pot
21       type(fdps_f64vec) :: acc
22    end type full_particle
23
24    contains
25
26    !**** Interaction function (particle-particle)
27    subroutine calc_gravity_pp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
28       integer(c_int), intent(in), value :: n_ip,n_jp
29       type(full_particle), dimension(n_ip), intent(in) :: ep_i
30       type(full_particle), dimension(n_jp), intent(in) :: ep_j
31       type(full_particle), dimension(n_ip), intent(inout) :: f
32       !* Local variables
33       integer(c_int) :: i,j
34       real(c_double) :: eps2,poti,r3_inv,r_inv
35       type(fdps_f64vec) :: xi,ai,rij
36
37       !* Compute force
38       do i=1,n_ip
39          eps2 = ep_i(i)%eps * ep_i(i)%eps
40          xi = ep_i(i)%pos
41          ai = 0.0d0
42          poti = 0.0d0
43          do j=1,n_jp
```

```
44              rij%x  = xi%x - ep_j(j)%pos%x
45              rij%y  = xi%y - ep_j(j)%pos%y
46              rij%z  = xi%z - ep_j(j)%pos%z
47              r3_inv = rij%x*rij%x &
48                     + rij%y*rij%y &
49                     + rij%z*rij%z &
50                     + eps2
51              r_inv  = 1.0d0/sqrt(r3_inv)
52              r3_inv = r_inv * r_inv
53              r_inv  = r_inv * ep_j(j)%mass
54              r3_inv = r3_inv * r_inv
55              ai%x   = ai%x - r3_inv * rij%x
56              ai%y   = ai%y - r3_inv * rij%y
57              ai%z   = ai%z - r3_inv * rij%z
58              poti   = poti - r_inv
59              ! [IMPORTANT NOTE]
60              !   In the innermost loop, we use the components of vectors
61              !   directly for vector operations because of the following
62              !   reasion. Except for intel compilers with ‘-ipo‘ option,
63              !   most of Fortran compilers use function calls to perform
64              !   vector operations like rij = x - ep_j(j)%pos.
65              !   This significantly slow downs the speed of the code.
66              !   By using the components of vector directly, we can avoid
67              !   these function calls.
68           end do
69           f(i)%pot = f(i)%pot + poti
70           f(i)%acc = f(i)%acc + ai
71        end do
72
73     end subroutine calc_gravity_pp
74
75     !**** Interaction function (particle-super particle)
76     subroutine calc_gravity_psp(ep_i,n_ip,ep_j,n_jp,f) bind(c)
77        integer(c_int), intent(in), value :: n_ip,n_jp
78        type(full_particle), dimension(n_ip), intent(in) :: ep_i
79        type(fdps_spj_monopole), dimension(n_jp), intent(in) :: ep_j
80        type(full_particle), dimension(n_ip), intent(inout) :: f
81        !* Local variables
82        integer(c_int) :: i,j
83        real(c_double) :: eps2,poti,r3_inv,r_inv
84        type(fdps_f64vec) :: xi,ai,rij
85
86        do i=1,n_ip
87           eps2 = ep_i(i)%eps * ep_i(i)%eps
88           xi = ep_i(i)%pos
89           ai = 0.0d0
90           poti = 0.0d0
91           do j=1,n_jp
92              rij%x  = xi%x - ep_j(j)%pos%x
93              rij%y  = xi%y - ep_j(j)%pos%y
94              rij%z  = xi%z - ep_j(j)%pos%z
95              r3_inv = rij%x*rij%x &
96                     + rij%y*rij%y &
97                     + rij%z*rij%z &
98                     + eps2
```

```
 99              r_inv  = 1.0d0/sqrt(r3_inv)
100              r3_inv = r_inv * r_inv
101              r_inv  = r_inv * ep_j(j)%mass
102              r3_inv = r3_inv * r_inv
103              ai%x   = ai%x - r3_inv * rij%x
104              ai%y   = ai%y - r3_inv * rij%y
105              ai%z   = ai%z - r3_inv * rij%z
106              poti   = poti - r_inv
107           end do
108           f(i)%pot = f(i)%pot + poti
109           f(i)%acc = f(i)%acc + ai
110        end do
111
112     end subroutine calc_gravity_psp
113
114 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    use user_defined_types
 7    implicit none
 8    !* Local parameters
 9   !integer, parameter :: ntot=2**10
10    integer, parameter :: ntot=2**18
11    !-(force parameters)
12    real, parameter :: theta = 0.5
13    integer, parameter :: n_leaf_limit = 8
14    integer, parameter :: n_group_limit = 64
15    !-(domain decomposition)
16    real, parameter :: coef_ema=0.3
17    !-(timing parameters)
18    double precision, parameter :: time_end = 10.0d0
19    double precision, parameter :: dt = 1.0d0/128.0d0
20    double precision, parameter :: dt_diag = 1.0d0/8.0d0
21    double precision, parameter :: dt_snap = 1.0d0
22    !* Local variables
23    integer :: i,j,k,num_loop,ierr
24    integer :: psys_num,dinfo_num,tree_num
25    integer :: nloc
26    logical :: clear
27    double precision :: ekin0,epot0,etot0
28    double precision :: ekin1,epot1,etot1
29    double precision :: time_diag,time_snap,time_sys
30    double precision :: r,acc
31    type(fdps_controller) :: fdps_ctrl
32    type(full_particle), dimension(:), pointer :: ptcl
33    type(c_funptr) :: pfunc_ep_ep,pfunc_ep_sp
34    !-(IO)
35    character(len=64) :: fname
36    integer(c_int) :: np
37
```

```
38    !* Initialize FDPS
39    call fdps_ctrl%PS_Initialize()
40
41    !* Create domain info object
42    call fdps_ctrl%create_dinfo(dinfo_num)
43    call fdps_ctrl%init_dinfo(dinfo_num,coef_ema)
44
45    !* Create particle system object
46    call fdps_ctrl%create_psys(psys_num,'full_particle')
47    call fdps_ctrl%init_psys(psys_num)
48
49    !* Create tree object
50    call fdps_ctrl%create_tree(tree_num, &
51                               "Long,full_particle,full_particle,
                                       full_particle,Monopole")
52    call fdps_ctrl%init_tree(tree_num,ntot,theta, &
53                             n_leaf_limit,n_group_limit)
54
55    !* Make an initial condition
56    call setup_IC(fdps_ctrl,psys_num,ntot)
57
58    !* Domain decomposition and exchange particle
59    call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
60    call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
61
62    !* Compute force at the initial time
63    pfunc_ep_ep = c_funloc(calc_gravity_pp)
64    pfunc_ep_sp = c_funloc(calc_gravity_psp)
65    call fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
66                                                 pfunc_ep_ep, &
67                                                 pfunc_ep_sp, &
68                                                 psys_num,    &
69                                                 dinfo_num)
70    !* Compute energies at the initial time
71    clear = .true.
72    call calc_energy(fdps_ctrl,psys_num,etot0,ekin0,epot0,clear)
73
74    !* Time integration
75    time_diag = 0.0d0
76    time_snap = 0.0d0
77    time_sys  = 0.0d0
78    num_loop = 0
79    do
80       !* Output
81      !if (fdps_ctrl%get_rank() == 0) then
82      !   write(*,50)num_loop,time_sys
83      !   50 format('(num_loop, time_sys) = ',i5,1x,1es25.16e3)
84      !end if
85       if ( (time_sys >= time_snap) .or. &
86            (((time_sys + dt) - time_snap) > (time_snap - time_sys)) ) then
87          call output(fdps_ctrl,psys_num)
88          time_snap = time_snap + dt_snap
89       end if
90
91       !* Compute energies and output the results
```

```
92        clear = .true.
93        call calc_energy(fdps_ctrl,psys_num,etot1,ekin1,epot1,clear)
94        if (fdps_ctrl%get_rank() == 0) then
95           if ( (time_sys >= time_diag) .or. &
96                (((time_sys + dt) - time_diag) > (time_diag - time_sys)) )
                     then
97              write(*,100)time_sys,(etot1-etot0)/etot0
98              100 format("time: ",1es20.10e3,", energy error: ",1es20.10e3)
99              time_diag = time_diag + dt_diag
100          end if
101       end if
102
103       !* Leapfrog: Kick-Drift
104       call kick(fdps_ctrl,psys_num,0.5d0*dt)
105       time_sys = time_sys + dt
106       call drift(fdps_ctrl,psys_num,dt)
107
108       !* Domain decomposition & exchange particle
109       if (mod(num_loop,4) == 0) then
110          call fdps_ctrl%decompose_domain_all(dinfo_num,psys_num)
111       end if
112       call fdps_ctrl%exchange_particle(psys_num,dinfo_num)
113
114       !* Force calculation
115       pfunc_ep_ep = c_funloc(calc_gravity_pp)
116       pfunc_ep_sp = c_funloc(calc_gravity_psp)
117       call fdps_ctrl%calc_force_all_and_write_back(tree_num,    &
118                                                    pfunc_ep_ep, &
119                                                    pfunc_ep_sp, &
120                                                    psys_num,    &
121                                                    dinfo_num)
122       !* Leapfrog: Kick
123       call kick(fdps_ctrl,psys_num,0.5d0*dt)
124
125       !* Update num_loop
126       num_loop = num_loop + 1
127
128       !* Termination
129      !if (time_sys >= time_end) then
130       if (num_loop == 32) then
131          exit
132       end if
133    end do
134
135    !* Finalize FDPS
136    call fdps_ctrl%PS_Finalize()
137
138 end subroutine f_main
139
140 !-------------------------------------------------------------------------
141 !////////////////////////// S U B R O U T I N E ////////////////////////////
142 !////////////////////////// < S E T U P _ I C > ////////////////////////////
143 !-------------------------------------------------------------------------
144 subroutine setup_IC(fdps_ctrl,psys_num,nptcl_glb)
145    use fdps_vector
```

```
146     use fdps_module
147     use user_defined_types
148     implicit none
149     type(fdps_controller), intent(IN) :: fdps_ctrl
150     integer, intent(IN) :: psys_num,nptcl_glb
151     !* Local parameters
152     double precision, parameter :: m_tot=1.0d0
153     double precision, parameter :: rmax=3.0d0,r2max=rmax*rmax
154     !* Local variables
155     integer :: i,j,k,ierr
156     integer :: nprocs,myrank
157     double precision :: r2,cm_mass
158     type(fdps_f64vec) :: cm_pos,cm_vel,pos
159     type(full_particle), dimension(:), pointer :: ptcl
160     character(len=64) :: fname
161
162     !* Get # of MPI processes and rank number
163     nprocs = fdps_ctrl%get_num_procs()
164     myrank = fdps_ctrl%get_rank()
165
166     !* Make an initial condition at RANK 0
167     if (myrank == 0) then
168        !* Set # of local particles
169        call fdps_ctrl%set_nptcl_loc(psys_num,nptcl_glb)
170
171        !* Create an uniform sphere of particles
172        !** get the pointer to full particle data
173        call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
174        !** initialize Mersenne twister
175        call fdps_ctrl%MT_init_genrand(0)
176        do i=1,nptcl_glb
177           ptcl(i)%id   = i
178           ptcl(i)%mass = m_tot/nptcl_glb
179           do
180              ptcl(i)%pos%x = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
181              ptcl(i)%pos%y = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
182              ptcl(i)%pos%z = (2.0d0*fdps_ctrl%MT_genrand_res53()-1.0d0) *
                    rmax
183              r2 = ptcl(i)%pos*ptcl(i)%pos
184              if ( r2 < r2max ) exit
185           end do
186           ptcl(i)%vel = 0.0d0
187           ptcl(i)%eps = 1.0d0/32.0d0
188        end do
189
190        !* Correction
191        cm_pos  = 0.0d0
192        cm_vel  = 0.0d0
193        cm_mass = 0.0d0
194        do i=1,nptcl_glb
195           cm_pos  = cm_pos  + ptcl(i)%mass * ptcl(i)%pos
196           cm_vel  = cm_vel  + ptcl(i)%mass * ptcl(i)%vel
197           cm_mass = cm_mass + ptcl(i)%mass
```

```
198         end do
199         cm_pos = cm_pos/cm_mass
200         cm_vel = cm_vel/cm_mass
201         do i=1,nptcl_glb
202            ptcl(i)%pos = ptcl(i)%pos - cm_pos
203            ptcl(i)%vel = ptcl(i)%vel - cm_vel
204         end do
205
206         !* Output
207        !fname = 'initial.dat'
208        !open(unit=9,file=trim(fname),action='write',status='replace', &
209        !     form='unformatted',access='stream')
210        !open(unit=9,file=trim(fname),action='write',status='replace')
211        !   do i=1,nptcl_glb
212        !     !write(9)ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z
213        !      write(9,'(3es25.16e3)')ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos
               %z
214        !   end do
215        !close(unit=9)
216
217         !* Release the pointer
218         nullify( ptcl )
219
220      else
221         call fdps_ctrl%set_nptcl_loc(psys_num,0)
222      end if
223
224   end subroutine setup_IC
225
226   !-------------------------------------------------------------------------
227   !////////////////////////// S U B R O U T I N E ////////////////////////////
228   !//////////////////////////     < K I C K >     ////////////////////////////
229   !-------------------------------------------------------------------------
230   subroutine kick(fdps_ctrl,psys_num,dt)
231      use fdps_vector
232      use fdps_module
233      use user_defined_types
234      implicit none
235      type(fdps_controller), intent(IN) :: fdps_ctrl
236      integer, intent(IN) :: psys_num
237      double precision, intent(IN) :: dt
238      !* Local variables
239      integer :: i,nptcl_loc
240      type(full_particle), dimension(:), pointer :: ptcl
241
242      !* Get # of local particles
243      nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
244
245      !* Get the pointer to full particle data
246      call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
247      do i=1,nptcl_loc
248         ptcl(i)%vel = ptcl(i)%vel + ptcl(i)%acc * dt
249      end do
250      nullify(ptcl)
251
```

```
252 end subroutine kick
253
254 !-------------------------------------------------------------------------
255 !////////////////////////// S U B R O U T I N E ////////////////////////////
256 !//////////////////////////    < D R I F T >    ////////////////////////////
257 !-------------------------------------------------------------------------
258 subroutine drift(fdps_ctrl,psys_num,dt)
259    use fdps_vector
260    use fdps_module
261    use user_defined_types
262    implicit none
263    type(fdps_controller), intent(IN) :: fdps_ctrl
264    integer, intent(IN) :: psys_num
265    double precision, intent(IN) :: dt
266    !* Local variables
267    integer :: i,nptcl_loc
268    type(full_particle), dimension(:), pointer :: ptcl
269
270    !* Get # of local particles
271    nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
272
273    !* Get the pointer to full particle data
274    call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
275    do i=1,nptcl_loc
276       ptcl(i)%pos = ptcl(i)%pos + ptcl(i)%vel * dt
277    end do
278    nullify(ptcl)
279
280 end subroutine drift
281
282 !-------------------------------------------------------------------------
283 !//////////////////////    S U B R O U T I N E    ////////////////////////
284 !////////////////////// < C A L C _ E N E R G Y > ////////////////////////
285 !-------------------------------------------------------------------------
286 subroutine calc_energy(fdps_ctrl,psys_num,etot,ekin,epot,clear)
287    use fdps_vector
288    use fdps_module
289    use user_defined_types
290    implicit none
291    type(fdps_controller), intent(IN) :: fdps_ctrl
292    integer, intent(IN) :: psys_num
293    double precision, intent(INOUT) :: etot,ekin,epot
294    logical, intent(IN) :: clear
295    !* Local variables
296    integer :: i,nptcl_loc
297    double precision :: etot_loc,ekin_loc,epot_loc
298    type(full_particle), dimension(:), pointer :: ptcl
299
300    !* Clear energies
301    if (clear .eqv. .true.) then
302       etot = 0.0d0
303       ekin = 0.0d0
304       epot = 0.0d0
305    end if
306
```

```
307     !* Get # of local particles
308     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
309     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
310
311     !* Compute energies
312     ekin_loc = 0.0d0
313     epot_loc = 0.0d0
314     do i=1,nptcl_loc
315        ekin_loc = ekin_loc + ptcl(i)%mass * ptcl(i)%vel * ptcl(i)%vel
316        epot_loc = epot_loc + ptcl(i)%mass * (ptcl(i)%pot + ptcl(i)%mass/
                ptcl(i)%eps)
317     end do
318     ekin_loc = ekin_loc * 0.5d0
319     epot_loc = epot_loc * 0.5d0
320     etot_loc = ekin_loc + epot_loc
321     call fdps_ctrl%get_sum(ekin_loc,ekin)
322     call fdps_ctrl%get_sum(epot_loc,epot)
323     call fdps_ctrl%get_sum(etot_loc,etot)
324
325     !* Release the pointer
326     nullify(ptcl)
327
328 end subroutine calc_energy
329
330 !-------------------------------------------------------------------------
331 !////////////////////////// S U B R O U T I N E ////////////////////////////
332 !//////////////////////////   < O U T P U T >   ////////////////////////////
333 !-------------------------------------------------------------------------
334 subroutine output(fdps_ctrl,psys_num)
335     use fdps_vector
336     use fdps_module
337     use user_defined_types
338     implicit none
339     type(fdps_controller), intent(IN) :: fdps_ctrl
340     integer, intent(IN) :: psys_num
341     !* Local parameters
342     character(len=16), parameter :: root_dir="result"
343     character(len=16), parameter :: file_prefix_1st="snap"
344     character(len=16), parameter :: file_prefix_2nd="proc"
345     !* Local variables
346     integer :: i,nptcl_loc
347     integer :: myrank
348     character(len=5) :: file_num,proc_num
349     character(len=64) :: cmd,sub_dir,fname
350     type(full_particle), dimension(:), pointer :: ptcl
351     !* Static variables
352     integer, save :: snap_num=0
353
354     !* Get the rank number
355     myrank = fdps_ctrl%get_rank()
356
357     !* Get # of local particles
358     nptcl_loc = fdps_ctrl%get_nptcl_loc(psys_num)
359
360     !* Get the pointer to full particle data
```

```
361     call fdps_ctrl%get_psys_fptr(psys_num,ptcl)
362
363     !* Output
364     write(file_num,"(i5.5)")snap_num
365     write(proc_num,"(i5.5)")myrank
366     fname =  trim(root_dir) // "/" &
367           // trim(file_prefix_1st) // file_num // "-" &
368           // trim(file_prefix_2nd) // proc_num // ".dat"
369     open(unit=9,file=trim(fname),action='write',status='replace')
370        do i=1,nptcl_loc
371           write(9,100)ptcl(i)%id,ptcl(i)%mass, &
372                       ptcl(i)%pos%x,ptcl(i)%pos%y,ptcl(i)%pos%z, &
373                       ptcl(i)%vel%x,ptcl(i)%vel%y,ptcl(i)%vel%z
374           100 format(i8,1x,7e25.16e3)
375        end do
376     close(unit=9)
377     nullify(ptcl)
378
379     !* Update snap_num
380     snap_num = snap_num + 1
381
382 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)

```
1  !================================
2  !   MODULE: User defined types
3  !================================
4  module user_defined_types
5     use, intrinsic :: iso_c_binding
6     use fdps_vector
7     implicit none
8
9     !* Private parameters
10    real(kind=c_double), parameter, private :: pi=datan(1.0d0)*4.0d0
11    !* Public parameters
12    real(kind=c_double), parameter, public :: kernel_support_radius=2.5d0
13
14    !**** Force types
15    type, public, bind(c) :: dens_force !$fdps Force
16       !$fdps clear smth=keep
17       real(kind=c_double) :: dens
18       real(kind=c_double) :: smth
19    end type dens_force
20
21    type, public, bind(c) :: hydro_force !$fdps Force
```

```
22        !$fdps clear
23        type(fdps_f64vec) :: acc
24        real(kind=c_double) :: eng_dot
25        real(kind=c_double) :: dt
26    end type hydro_force
27
28    !**** Full particle type
29    type, public, bind(c) :: full_particle !$fdps FP
30        !$fdps copyFromForce dens_force (dens,dens)
31        !$fdps copyFromForce hydro_force (acc,acc) (eng_dot,eng_dot) (dt,dt)
32        real(kind=c_double) :: mass !$fdps charge
33        type(fdps_f64vec) :: pos !$fdps position
34        type(fdps_f64vec) :: vel
35        type(fdps_f64vec) :: acc
36        real(kind=c_double) :: dens
37        real(kind=c_double) :: eng
38        real(kind=c_double) :: pres
39        real(kind=c_double) :: smth !$fdps rsearch
40        real(kind=c_double) :: snds
41        real(kind=c_double) :: eng_dot
42        real(kind=c_double) :: dt
43        integer(kind=c_long_long) :: id
44        type(fdps_f64vec) :: vel_half
45        real(kind=c_double) :: eng_half
46    end type full_particle
47
48    !**** Essential particle type
49    type, public, bind(c) :: essential_particle !$fdps EPI,EPJ
50        !$fdps copyFromFP full_particle (id,id) (pos,pos) (vel,vel) (mass,
              mass) (smth,smth) (dens,dens) (pres,pres) (snds,snds)
51        integer(kind=c_long_long) :: id
52        type(fdps_f64vec) :: pos !$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(dens_force), 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(hydro_force), 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 :: dens_tree_num,hydro_tree_num
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(dens_tree_num, &
59                                "Short,dens_force,essential_particle,
                                       essential_particle,Gather")
60     call fdps_ctrl%init_tree(dens_tree_num,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(hydro_tree_num, &
65                               "Short,hydro_force,essential_particle,
                                       essential_particle,Symmetry")
66    call fdps_ctrl%init_tree(hydro_tree_num,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(dens_tree_num, &
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(hydro_tree_num, &
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(dens_tree_num, &
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(hydro_tree_num, &
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 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。

6.1 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)

6.2 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)

6.3 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。

7 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. (in prep) 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。