FDPSユーザーチュートリアル

FDPS開発者

目 次

1 TODO

2　変更記録

- 2015/01/25 作成

3 概要

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

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

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

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

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

ユーザーは、粒子間相互作用や粒子の軌道積分などを、C++言語を用いて記述する。将来的には Fortran 言語でも記述できるように検討する。

4 入門

本節では、FDPS の入門について記述する。FDPS を使用する環境、FDPS に必要なソフトウェア、FDPS のインストール方法、サンプルコードの使用方法、の順で記述する。

4.1 動作環境

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

4.2 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。

4.2.1 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。

4.2.1.1 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)

4.2.1.2 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。

4.2.1.2.1 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)

4.2.1.2.2　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)

4.2.1.2.3　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)

4.2.2　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。

4.2.2.1　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

4.3　インストール

本節では、FDPS 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。

4.3.1　取得方法

以下の方法のいずれかで FDPS 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 のしたを 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
```

4.3.2 ビルド方法

configure などをする必要はない。

4.4 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記述する。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。

4.4.1 重力 N 体シミュレーションコード

4.4.1.1 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/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

。

4.4.1.2 ディレクトリ移動

ディレクトリ$(FDPS)/sample/nbody に移動に移動する。

4.4.1.3 Makefile の編集

Makefile の編集項目は以下の通りである。OpenMP と MPI を使用するかどうかで編集方法が変ることに注意。

- OpenMP も MPI も使用しない場合
  - マクロ CC に C++コンパイラを代入する
- OpenMP のみ使用の場合
  - マクロ CC に OpenMP 対応の C++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp” の行のコメントアウトを外す (インテルコンパイラの場合は-fopenmp を外す)
- MPI のみ使用の場合
  - マクロ CC に MPIC++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL”の行のコメントアウトを外す
- OpenMP と MPI の同時使用の場合
  - マクロ CC に MPI 対応の C++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp” の行のコメントアウトを外す (インテルコンパイラの場合は-fopenmp を外す)
  - ”CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL”の行のコメントアウトを外す

4.4.1.4 make の実行

make コマンドを実行する。

4.4.1.5 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、標準エラー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:  9.6250000 energy error: -4.512836e-03
time:  9.7500000 energy error: -4.440746e-03
time:  9.8750000 energy error: -4.652358e-03
time: 10.0000000 energy error: -4.605855e-03
******** FDPS has successfully finished. ********
```

4.4.1.6 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”000x.dat”ができている。x は 0 から 9 の値で、時刻を表す。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。

```
$ gnuplot
$ plot "result/0009.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以下のように実行すればよい (MPI を使用しない場合)。

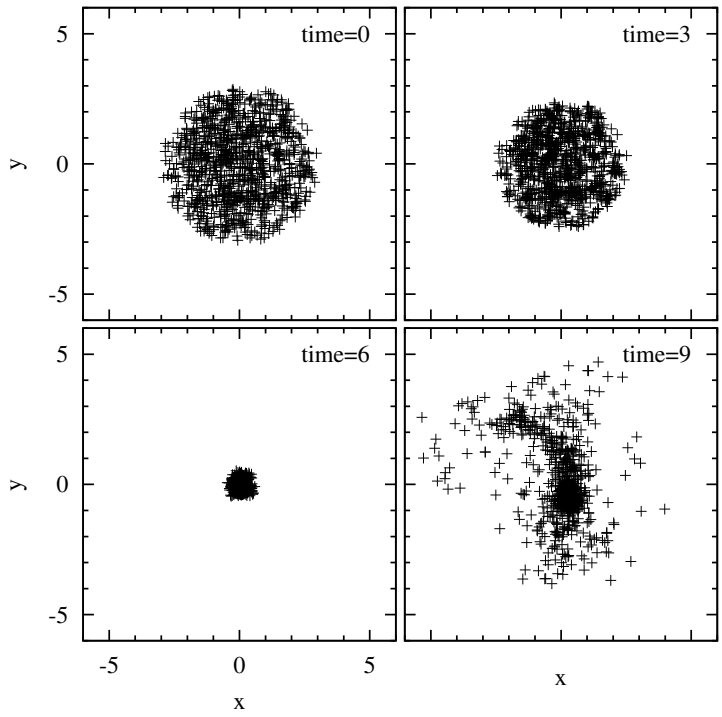

図 1:

```
$ ./nbody.out -N 10000
```

4.4.2 SPH シミュレーションコード

4.4.2.1 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/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)

4.4.2.2 ディレクトリ移動

ディレクトリ$(FDPS)/sample/nbody に移動に移動する。

4.4.2.3 Makefile の編集

Makefile の編集項目は以下の通りである。OpenMP と MPI を使用するかどうかで編集方法が変ることに注意。

- OpenMP も MPI も使用しない場合
  - マクロ CC に C++コンパイラを代入する
- OpenMP のみ使用の場合
  - マクロ CC に OpenMP 対応の C++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp” の行のコメントアウトを外す (インテルコンパイラの場合は-fopenmp を外す)
- MPI のみ使用の場合
  - マクロ CC に MPIC++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL”の行のコメントアウトを外す
- OpenMP と MPI の同時使用の場合
  - マクロ CC に MPI 対応の C++コンパイラを代入する
  - ”CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL -fopenmp” の行のコメントアウトを外す (インテルコンパイラの場合は-fopenmp を外す)
  - ”CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL”の行のコメントアウトを外す

4.4.2.4 make の実行

make コマンドを実行する。

4.4.2.5 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
```

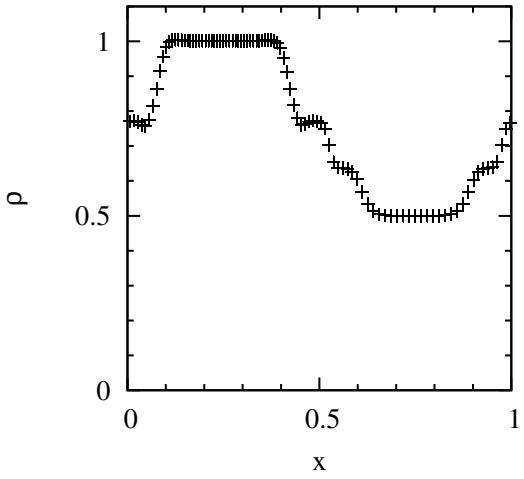

図 2:

- 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、標準エラー出力は以下のようなログを出力する。

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

4.4.2.6 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”00xx.dat”(x には数字が入る) となっている。ファイル名は時刻を表す。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。

これは 3 次元の衝撃波管問題である。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)。

```
$ gnuplot
$ plot "result/0040.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。

5 使ってみよう

5.1 サンプルコードのコンパイルと実行

5.2 前提知識

5.2.1 Vector 型

5.3 固定長 SPH シミュレーションコード

5.3.1 作業ディレクトリ

5.3.2 インクルード

ソースコード 1: Include files

```
1 //FDPSのヘッダをインクルード
2 #include <particle_simulator.hpp>
3 //コード中で使うヘッダをインクルード
4 #include <cmath>
5 #include <cstdio>
6 #include <iostream>
7 #include <vector>
```

5.3.3 ユーザー定義クラス

5.3.3.1 概要

5.3.3.2 FullParticle 型

ソースコード 2: FullParticle 型

```
1  struct FP{
2          PS::F64 mass;
3          PS::F64vec pos;
4          PS::F64vec vel;
5          PS::F64vec acc;
6          PS::F64 dens;
7          PS::F64 eng;
8          PS::F64 pres;
9          PS::F64 smth;
10         PS::F64 snds;
11         PS::F64 eng_dot;
12         PS::F64 dt;
```

```
13          PS::S64 id;
14          PS::F64vec vel_half;
15          PS::F64 eng_half;
16          void copyFromForce(const Dens& dens){
17                  this->dens = dens.dens;
18          }
19          void copyFromForce(const Hydro& force){
20                  this->acc     = force.acc;
21                  this->eng_dot = force.eng_dot;
22                  this->dt      = force.dt;
23          }
24          PS::F64 getCharge() const{
25                  return this->mass;
26          }
27          PS::F64vec getPos() const{
28                  return this->pos;
29          }
30          PS::F64 getRSearch() const{
31                  return kernelSupportRadius * this->smth;
32          }
33          void setPos(const PS::F64vec& pos){
34                  this->pos = pos;
35          }
36          void writeAscii(FILE* fp) const{
37                  fprintf(fp, "%ld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\
                        t%lf\t%lf\t%lf\t%lf\n", this->id,  this->
                        mass,  this->pos.x,  this->pos.y,  this->
                        pos.z,  this->vel.x,  this->vel.y,  this
                        ->vel.z,  this->dens,  this->eng,  this->
                        pres);
38          }
39          void readAscii(FILE* fp){
40                  fscanf(fp, "%ld\t%lf\t%lf\t%lf\t%lf\t%lf\t%lf\t
                        %lf\t%lf\t%lf\t%lf\n", &this->id, &this->
                        mass, &this->pos.x, &this->pos.y, &this->
                        pos.z, &this->vel.x, &this->vel.y, &this
                        ->vel.z, &this->dens, &this->eng, &this->
                        pres);
41          }
42          void setPressure(){
```

```
43                  const PS::F64 hcr = 1.4;
44                  pres = (hcr - 1.0) * dens * eng;
45                  snds = sqrt(hcr * pres / dens);
46          }
47  };
```

5.3.3.3 EssentialParticleI 型

ソースコード 3: EssentialParticle 型

```
1  struct EP{
2          PS::F64vec pos;
3          PS::F64vec vel;
4          PS::F64    mass;
5          PS::F64    smth;
6          PS::F64    dens;
7          PS::F64    pres;
8          PS::F64    snds;
9          void copyFromFP(const FP& rp){
10                 this->pos  = rp.pos;
11                 this->vel  = rp.vel;
12                 this->mass = rp.mass;
13                 this->smth = rp.smth;
14                 this->dens = rp.dens;
15                 this->pres = rp.pres;
16                 this->snds = rp.snds;
17         }
18         PS::F64vec getPos() const{
19                 return this->pos;
20         }
21         PS::F64 getRSearch() const{
22                 return kernelSupportRadius * this->smth;
23         }
24         void setPos(const PS::F64vec& pos){
25                 this->pos = pos;
26         }
27 };
```

5.3.3.4 Force 型

ソースコード 4: Force 型

```
class Dens{
        public:
        PS::F64 dens;
        PS::F64 smth;
        void clear(){
                dens = 0;
        }
};

class Hydro{
        public:
        PS::F64vec acc;
        PS::F64 eng_dot;
        PS::F64 dt;
        void clear(){
                acc = 0;
                eng_dot = 0;
        }
};
```

5.3.3.5 calcForceEpEp 型

ソースコード 5: calcForceEpEp 型

```
class CalcDensity{
        public:
        void operator () (const EP* const ep_i, const PS::S32
             Nip, const EP* const ep_j, const PS::S32 Njp,
             Dens* const dens){
                for(PS::S32 i = 0 ; i < Nip ; ++ i){
                        dens[i].clear();
                        for(PS::S32 j = 0 ; j < Njp ; ++ j){
                                const PS::F64vec dr = ep_j[j].
                                      pos - ep_i[i].pos;
                                dens[i].dens += ep_j[j].mass *
                                      W(dr, ep_i[i].smth);
                        }
                }
        }
```

```
12 };
13
14 class CalcHydroForce{
15         public:
16         void operator () (const EP* const ep_i, const PS::S32
               Nip, const EP* const ep_j, const PS::S32 Njp,
               Hydro* const hydro){
17                for(PS::S32 i = 0; i < Nip ; ++ i){
18                        hydro[i].clear();
19                        PS::F64 v_sig_max = 0.0;
20                        for(PS::S32 j = 0; j < Njp ; ++ j){
21                                const PS::F64vec dr = ep_i[i].
                                      pos - ep_j[j].pos;
22                                const PS::F64vec dv = ep_i[i].
                                      vel - ep_j[j].vel;
23                                const PS::F64 w_ij = (dv * dr <
                                       0) ? dv * dr / sqrt(dr *
                                       dr) : 0;
24                                const PS::F64 v_sig = ep_i[i].
                                      snds + ep_j[j].snds - 3.0
                                       * w_ij;
25                                v_sig_max = std::max(v_sig_max,
                                       v_sig);
26                                const PS::F64 AV = - 0.5 *
                                      v_sig * w_ij / (0.5 * (
                                      ep_i[i].dens + ep_j[j].
                                      dens));
27                                const PS::F64vec gradW_ij = 0.5
                                       * (gradW(dr, ep_i[i].
                                      smth) + gradW(dr, ep_j[j
                                      ].smth));
28                                hydro[i].acc     -= ep_j[j].
                                      mass * (ep_i[i].pres / (
                                      ep_i[i].dens * ep_i[i].
                                      dens) + ep_j[j].pres / (
                                      ep_j[j].dens * ep_j[j].
                                      dens) + AV) * gradW_ij;
29                                hydro[i].eng_dot += ep_j[j].
                                      mass * (ep_i[i].pres / (
                                      ep_i[i].dens * ep_i[i].
```

```
                                        dens) + 0.5 * AV) * dv *
                                        gradW_ij;
30                          }
31                          hydro[i].dt = C_CFL * 2.0 * ep_i[i].
                                  smth / v_sig_max;
32                  }
33          }
34 };
```

6　サンプルコード

6.1　N 体シミュレーション

6.2　SPH シミュレーション

6.3　N 体+SPH シミュレーション

7　エラーメッセージ

7.1　概要

FDPS はいくつかのエラーメッセージを用意している。１つはコンパイル時のエラーメッセージであり、もう１つは実行時のエラーメッセージである。以下、この順に記述する。

7.2　コンパイル時のエラーメッセージ

7.3　実行時のエラーメッセージ

標準エラー出力に以下のような書式でメッセージが出力される。

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

- *ERROR MESSAGE*
  エラーメッセージ
- *FUNCTION NAME*
  エラーが起こった関数の名前
- *LINE NUMBER*
  エラーが起こった行番号
- *FILE NAME*
  エラーが起こったファイルの名前

8　ユーザーサポート

FDPS を使用したコード開発に関する相談は以下のメールアドレス fdps-support@mail.jmlab.jp で受け付けています。以下のような場合は各項目毎の対応をお願いします。

8.1　コンパイルできない場合

ユーザーには以下の情報提供をお願いします。

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

8.2　コードがうまく動かない場合

ユーザーには以下の情報提供をお願いします。

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

9　ライセンス

MIT ライセンスに準ずる。標準機能のみ使用する場合は、Iwasawa et al. (2015 in prep), Tanikawa et al. (2016 in prep) の引用を義務とする。拡張機能のうち Particle Mesh クラスを使用する場合は、上記に加え、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) の引用を義務とする。