

FDPS 講習会の手引

谷川衝、岩澤全規、細野七月、似鳥啓吾、村主崇行、行方大輔、牧野淳一郎

平成 26 年 7 月 6 日

目次

1	プログラム	2
2	FDPS の実習	3
2.1	準備	3
2.1.1	FOCUS の計算機を用いる場合	3
2.1.2	自分で用意した計算機で実行する場合	4
2.2	実習本番	9
2.2.1	重力 N 体シミュレーションコード	9
2.2.1.1	概要	9
2.2.1.2	シリアルコード	9
2.2.1.2.1	ディレクトリ移動	10
2.2.1.2.2	make の実行	10
2.2.1.2.3	計算の実行	10
2.2.1.2.4	結果の解析	11
2.2.1.3	Phantom-GRAPE の利用	12
2.2.1.4	OpenMP/MPI の利用	13
2.2.2	SPH シミュレーションコード	15
2.2.2.1	概要	15
2.2.2.2	シリアルコード	15
2.2.2.2.1	ディレクトリ移動	15
2.2.2.2.2	make の実行	15
2.2.2.2.3	計算の実行	15
2.2.2.2.4	結果の解析	16
2.2.2.3	OpenMP/MPI の利用	16

1 プログラム

- 13:00 – 14:00 FDPS の講義
 - 13:00 – 13:05 イン트로ダクション (牧野淳一郎)
 - 13:05 – 13:15 概要説明 (行方大輔)
 - 13:15 – 13:25 FDPS で必要な C++ 解説 (似鳥啓吾)
 - 13:25 – 13:35 FDPS 詳細 1 – API と内部構造 (岩澤全規)
 - 13:35 – 13:45 FDPS 詳細 2 – サンプルコード解説 (細野七月)
 - 13:45 – 14:00 Q&A
- 14:00 – 15:30 FDPS の実習
 - FDPS のインストール
 - サンプルコードの使用 1 (重力 N 体シミュレーションコード)
 - サンプルコードの使用 2 (SPH シミュレーションコード)
- 15:30 – 17:00 FDPS 使用に関する相談

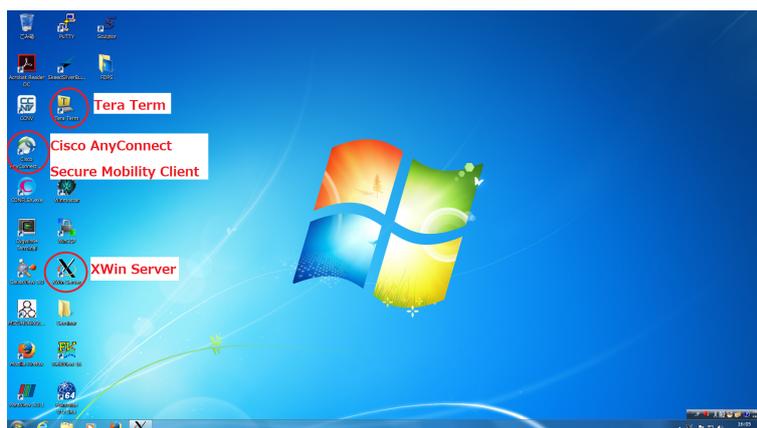


図 1:

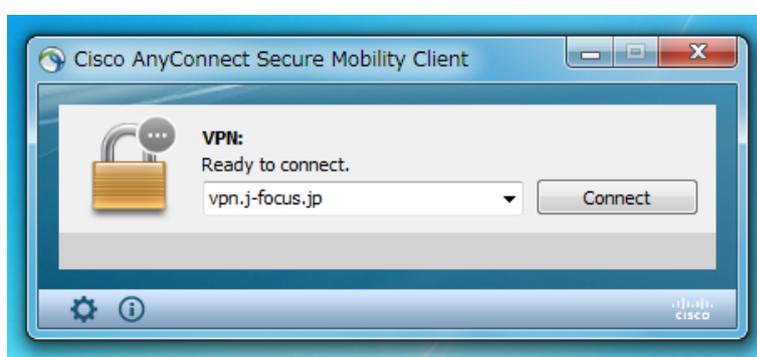


図 2:

2 FDPSの実習

2.1 準備

2.1.1 FOCUSの計算機を用いる場合

まずは、FOCUSの計算機にログインするまでの手順を紹介する。講習用PCの電源をつけたら、講習会当日に受けとったアカウント名とパスワードを使ってログインする。PCが起動したらデスクトップ上のXwin Severをダブルクリックし、X serverを立ち上げる(図1参照)。

次にVPN接続を行う為、デスクトップ上にあるCisco AnyConect Secure Mobility Clientをダブルクリックで起動する(図1参照)。すると図2の様な画面が現れるので、Connectをクリック。ユーザー名、パスワードを聞かれるので、事前に配布されたユーザー名、パスワードを入力する(図3参照)。

次にデスクトップ上にあるTera Termをダブルクリックして起動する(図1参照)。すると図4の様な画面が現れるので、一度キャンセルする。上のタブの設定(図5参照)からSSH

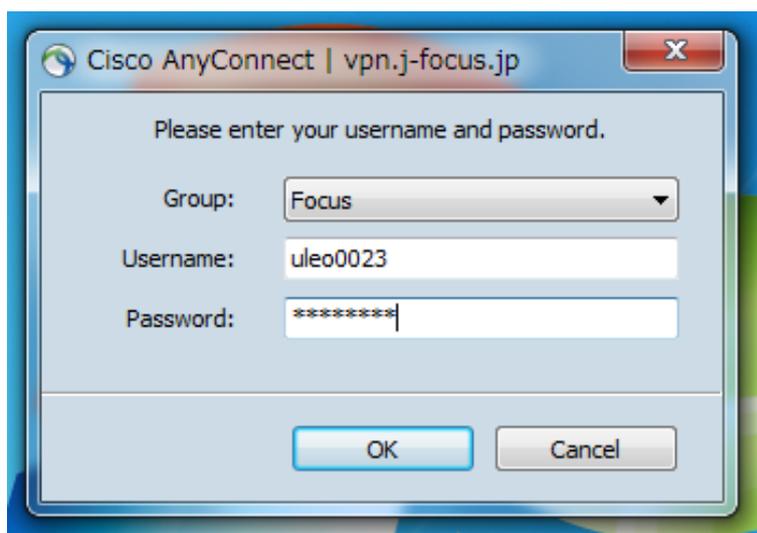


図 3:

転送をクリックすると図 6 の様な画面が現れるので、リモートのアプリケーションをローカルの X サーバーに表示するのチェックを入れる。

上のタブのファイルから新しい接続を選び (図 7 参照)、図 8 の画面になるので、ホスト名を # にして OK を押す。

図 9 の画面になったら、続行を押し、図 10 の画面になるので、事前に配布されたユーザー名、パスワードを入力する。

図 11 の画面になれば FOCUS の計算機にログイン成功である。

FOCUS の計算機にログインしたら、以下のコマンドを実行する。

```
$ module load gnu/openmpi165
```

以下のコマンドを実行し、FDPS を自分のカレントディレクトリにコピーする (「\$」はコマンドプロンプトであるので、「\$」を打ち込む必要はない)。

```
$ cp -r ../share/FDPS2.0 .
```

以上により、ディレクトリ FDPS2.0 がコピーされる。以下ではディレクトリ FDPS2.0 があるディレクトリの名前を fdps とする。

2.1.2 自分で用意した計算機で実行する場合

<https://github.com/FDPS/FDPS> から FDPS の最新版をダウンロードし、好きなディレクトリ下で解凍する。これによってディレクトリ FDPS-master が出来る。以下ではディレクトリ FDPS-master があるディレクトリの名前を fdps とし、文章中の FDPS2.0 を FDPS-master と読み替える。

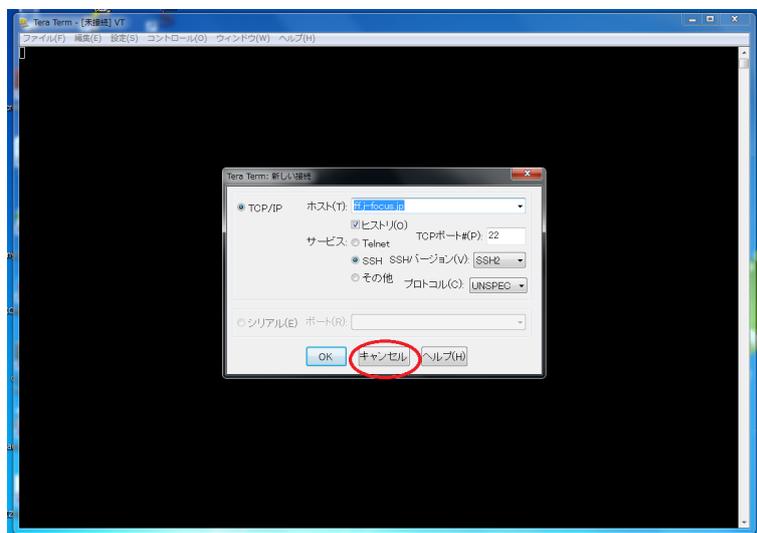


図 4:

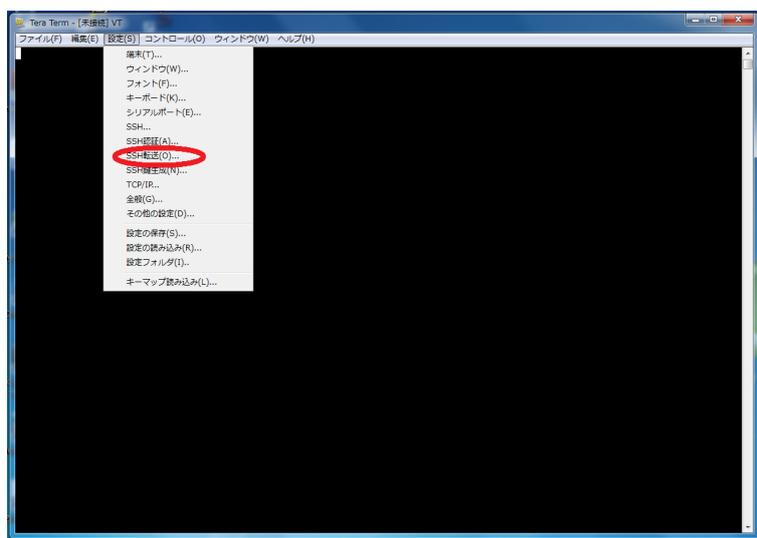


図 5:

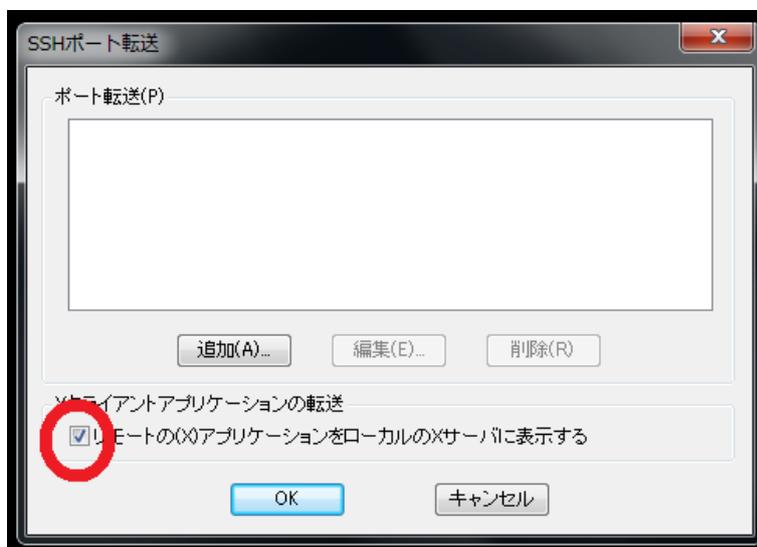


図 6:

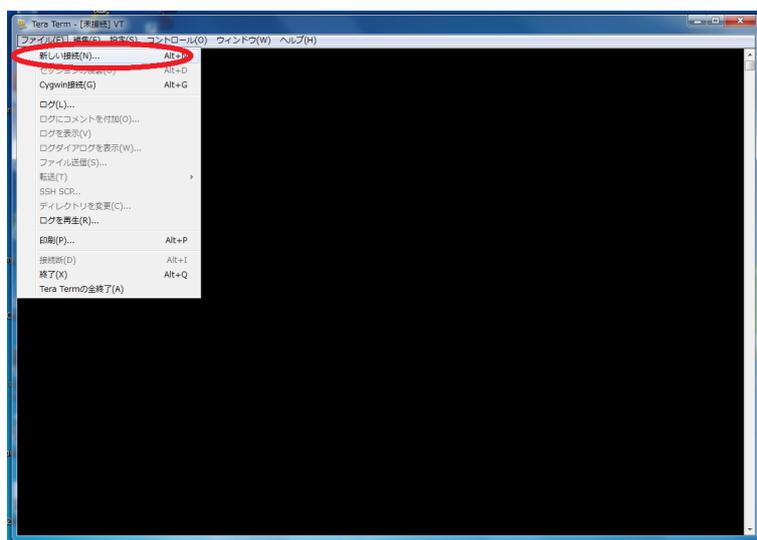


図 7:

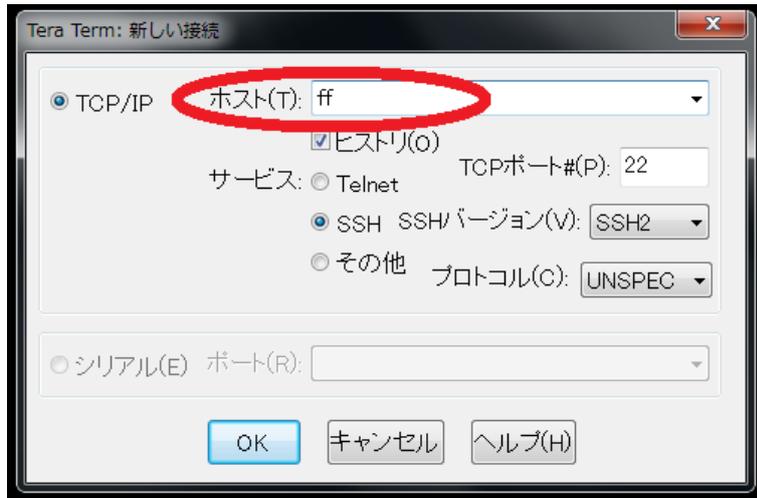


図 8:

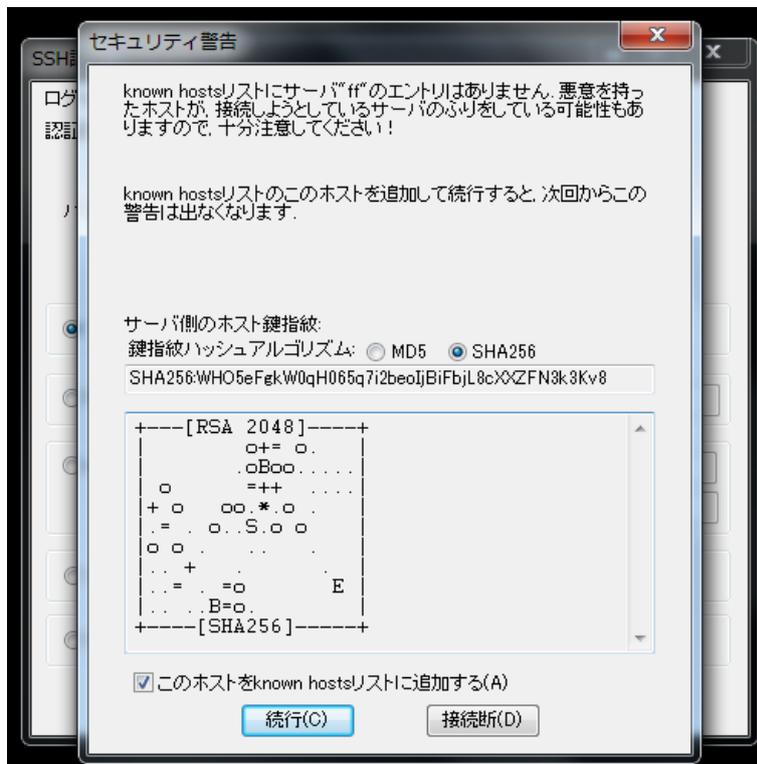


図 9:

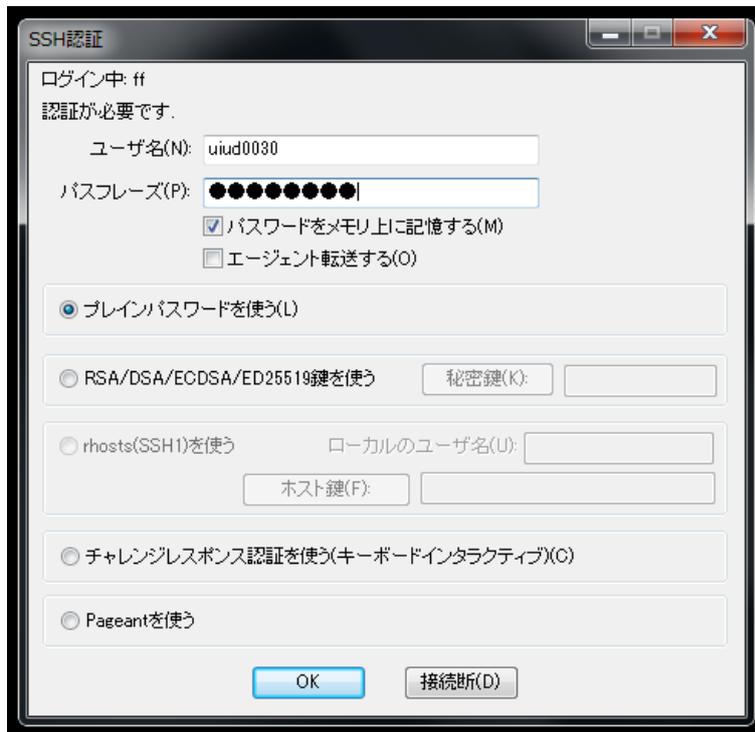


図 10:

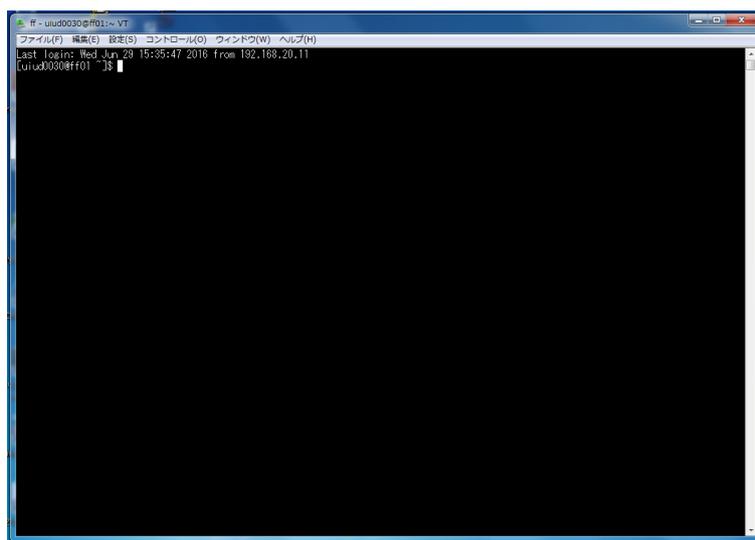


図 11:

2.2 実習本番

実習で行うことは、FDPS を使って実装された重力 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を使用することである。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を使用する。

なお、実習の際に Makefile を更新し、コンパイルし直すという作業を何回か行うが、ここで注意しなくてはならないのは、Makefile を編集しただけでは実行ファイルの再作成は行われないうことである。この場合、きちんと前回作った実行ファイルを明示的に “\$ rm ./nbody.out” もしくは “\$ make clean” などで消す必要がある。これを忘れた場合、「make: ‘nbody.out’ は更新済みです」と出る。

2.2.1 重力 N 体シミュレーションコード

ここでは、重力 N 体シミュレーションコードでの cold collapse を、並列環境無し、OpenMP を用いた並列計算環境、OpenMP + MPI を用いた並列計算環境の 3 つで行う。後者の 2 つに関しては FOCUS スパコンに計算を用いる。

2.2.1.1 概要

ここでは、用意された重力 N 体シミュレーションコードを動かしてみよう。このコードは、重力多体系のコールドコラプスを計算する。この節でまず行うことは、シリアルコードのコンパイルと実行、出て来た結果の解析である。次にシリアルコードを Phantom-GRAPe を用いて高速化して、その速さを体験しよう。最後に OpenMP や MPI を利用して、さらにコードを高速化する。

2.2.1.2 シリアルコード

以下の手順で本コードを使用できる。

- ディレクトリ `fdps/FDPS2.0/sample/nbody` に移動
- `make` を実行
- ジョブの投入
- 結果の解析
- OpenMP/MPI の利用 (オプション)

2.2.1.2.1 ディレクトリ移動

ディレクトリ `fdps/FDPS2.0/sample/nbody` に移動する。

```
$ cd fdps/FDPS2.0/sample/nbody
```

2.2.1.2.2 `make` の実行

`make` コマンドを実行する。

```
$ make
```

2.2.1.2.3 計算の実行

まずは、インタラクティブ実行で計算を実行する。これは、生成された実行ファイルの名前をそのまま実行すればよい。

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

計算が開始されると以下の様な表示が出力される。

```

//=====\\
||                                     ||
|| :::::::::: :::::::::: :::::::::: :::::::::: ||
|| ::          ::          : ::          : ::          ||
|| :::::::::: ::          : ::::::::::' ':::::::::: ||
|| ::          ::::::::::' ::          '.....' ||
||      Framework for Developing      ||
||          Particle Simulator          ||
||      Version 2.0 (2016/06)          ||
\\=====//

```

```

Home   : https://github.com/fdps/fdps
E-mail : fdps-support@mail.jmlab.jp
Licence: MIT (see, https://github.com/FDPS/FDPS/blob/master/LICENSE)
Note   : Please cite Iwasawa et al. (in press.)

```

Copyright (C) 2015

Masaki Iwasawa, Ataru Tanikawa, Natsuki Hosono,
 Keigo Nitadori, Takayuki Muranushi, Daisuke Namekata
 Junichiro Makino and many others

***** FDPS has successfully begun. *****

./result/t-de.dat

Number of processes: 1

Number of threads per process: 1

(以下省略)

正しくジョブが終了すると、標準入出力の最後には以下のようなログが出力されるはずである。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: 9.5000000 energy error: -3.804653e-03
time: 9.6250000 energy error: -3.971175e-03
time: 9.7500000 energy error: -3.822343e-03
time: 9.8750000 energy error: -3.884310e-03
***** FDPS has successfully finished. *****

```

ただし、後述する Phantom-GRAPE を用いた場合、energy error 数値は変わるので注意する。

2.2.1.2.4 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”000*.dat”ができています。*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、

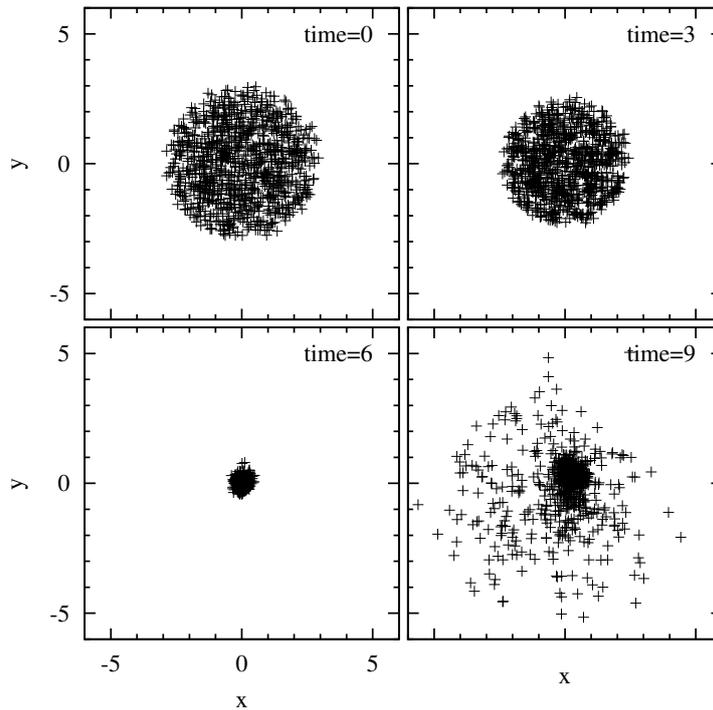


図 12:

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
$ set size square
$ plot [-5:5][-5:5]"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 12 参照)。

2.2.1.3 Phantom-GRAPe の利用

以下では、相互作用計算に Phantom-GRAPe を使う場合について、記述する。この場合、ユーザーはまずは Phantom-GRAPe のコンパイルを行わなければならない。今回使う Phantom-GRAPe のソースコードは、`fdps/FDPS2.0/src/phantom_grape_x86/G5/newton/libpg5/` 以下に存在するので、そこまで移動し、コンパイルを行う。以下は、現在 `sample/nbody/` に居る場合の例である。

```
$ cd ../../src/phantom_grape_x86/G5/newton/libpg5/  
$ make
```

コンパイルが成功したら、元のディレクトリに戻る。次に、Makefileの修正を行う。Makefileの

```
#use_phantom_grape_x86 = yes
```

となっているところのコメントアウトを外しコンパイルを行う。

無事にコンパイルが通れば、以降の実行・解析の手順は同様である。実行直後の表示に”rsqrt: MSE = ...” という行が追加されていれば正しく実行ができています。

(省略)

```
***** FDPS has successfully begun. *****  
./result/t-de.dat  
Number of processes: 1  
Number of threads per process: 1  
rsqrt: MSE = 1.158186e-04, Bias = 8.375360e-08  
(省略)
```

2.2.1.4 OpenMP/MPIの利用

OpenMP や MPI を利用する場合について以下に記述する。

以降では、計算の実行の際にはジョブの投入を行うことにする。以下の様なコマンドでジョブの投入を行う。これは OpenMP 実行の際の例である。

```
$ sbatch runOMP.sh
```

正しくジョブが投入されている場合、squeue コマンドを実行すると、例えば以下のようにログが出力される。

JOBID	PARTITION	NAME	USER	ST	TIME	NODES	NODELIST(REASON)
200402	d006h	nbody	uleo0016	PD	0:00	1	(None)

ジョブのキャンセルは以下のコマンドで実行できる。

```
$ scancel JOBID
```

JOBID は上の squeue コマンドで表示された JOBID の番号である (上で言えば、200402)。正しくジョブが終了すると、ファイル stdout.log の最後にログが出力される。

- OpenMP のみ使用の場合

- Makefile の編集

- * マクロ CC に OpenMP 対応の C++ コンパイラを代入する。今回の実習の環境では変更する必要は無い。
 - * “CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL” と “CFLAGS += -fopenmp” の 2 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 のコメントアウトははずさない)。
- make コマンドを実行する。
 - sbatch runOMP.sh と打って実行する。正しく実行された場合、stdout.log に以下のように表示されるはずである。

```

This is a sample program of N-body simulation on FDPS!
Number of processes: 1
Number of threads per process: 4
time: 0.000000 energy error: -0.000000e+00
(省略)
```

見て分かる通り、Number of threads per process: 4 となっている。これで、4 スレッドでの並列計算が行われている事が確認できた。

- OpenMP と MPI の同時使用の場合

- Makefile の編集
 - * マクロ CC に MPI 対応の C++ コンパイラを代入する。今回の実習の環境では mpiCC にする。
 - * “CFLAGS += -DPARTICLE_SIMULATOR_THREAD_PARALLEL” と “CFLAGS += -fopenmp” の 2 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 のコメントアウトははずさない)。
 - * “CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL” の行のコメントアウトを外す
- make コマンドを実行する。
- sbatch runMPI.sh と打って実行する。正しく実行された場合、stdout.log に以下のように表示されるはずである。

```

This is a sample program of N-body simulation on FDPS!
Number of processes: 2
Number of threads per process: 4
time: 0.000000 energy error: -0.000000e+00
(省略)
```

Number of processes: 2 となっている事がわかる。これで、2 プロセス 4 スレッドでの並列計算が行われている事が確認できた。

2.2.2 SPH シミュレーションコード

2.2.2.1 概要

ここでは、SPH シミュレーションコードを動かす。用意されているコードは、断熱スフィアコラプスの計算を行う。この節でまず行うことは、シリアルコードのコンパイルと実行、出て来た結果の解析である。最後に OpenMP や MPI を利用して、さらにコードを高速化する。

2.2.2.2 シリアルコード

以下の手順で本コードを使用できる。

- ディレクトリ `fdps/FDPS2.0/sample/nbody sph` に移動
- `make` を実行
- ジョブの投入
- 結果の解析
- OpenMP/MPI の利用 (オプション)

2.2.2.2.1 ディレクトリ移動

ディレクトリ `fdps/FDPS2.0/sample/nbody sph` に移動に移動する。

2.2.2.2.2 `make` の実行

`make` コマンドを実行する。

2.2.2.2.3 計算の実行

まずは、インタラクティブ実行で計算を実行する。これは、生成された実行ファイルの名前をそのまま実行すればよい。

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

正しくジョブが終了すると、最後に以下のようなログが出力されるはずである。

(省略)

```
//=====
time = 7.6861124696015781e-01, dt = 3.5727841270539857e-03
step = 88
//=====
9.999999999997635e-01
6.6872858041836647e-01
9.7700092080239072e-04
9.7700092080317243e-04
9.7700092080318761e-04
***** FDPS has successfully finished. *****
```

2.2.2.2.4 結果の解析

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”00**`.dat`”(* 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Evrard sphereである。以下のコマンドを実行すれば、横軸に r 、縦軸に密度の `logscale` での図が作成される。

```
$ gnuplot
$ set logscale
$ plot "result/0040.dat" using (sqrt($3**2 + $4**2 + $5**2)):9
```

正しい答が得られれば、図13のような図を描ける。

2.2.2.3 OpenMP/MPIの利用

OpenMP や MPI を利用する場合を以下に示す。

- OpenMP のみ使用の場合

- Makefile の編集

- * 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`” と “`CFLAGS += -fopenmp`” の2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` のコメントアウトははずさない)。

- `make` コマンドを実行する。

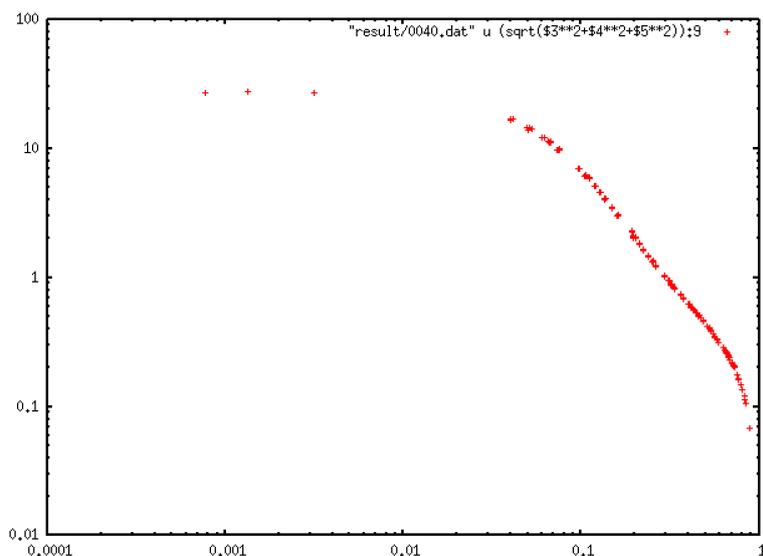


図 13:

- sbatch runOMP.sh と打って実行する。正しく実行された場合、stdout.log 以下のように表示されるはずである。

```

This is a sample program of Smoothed Particle Hydrodynamics on FDPS!
# of proc is 1
# of thread is 4
//=====
setup...
time = 0.0000000000000000e+00, dt = 8.5419983049046022e-02
//=====
(省略)

```

● OpenMP と MPI の同時使用の場合

- Makefile の編集
 - * 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” と “CFLAGS += -fopenmp” の 2 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 のコメントアウトははずさない)。
 - * “CFLAGS += -DPARTICLE_SIMULATOR_MPI_PARALLEL” の行のコメントアウトを外す
- make コマンドを実行する。
- sbatch runMPI.sh と打って実行する。正しく実行された場合、stdout.log 以下のように表示されるはずである。

This is a sample program of Smoothed Particle Hydrodynamics on FDPS!

```
# of proc is 2
```

```
# of thread is 4
```

```
//=====
```

```
setup...
```

```
time = 0.0000000000000000e+00, dt = 8.5419983049046022e-02
```

```
time = 0.0000000000000000e+00, dt = 8.5419983049046022e-02
```

```
//=====
```

```
(省略)
```