FDPS並列版実装記述書

粒子系シミュレータ研究チーム

目 次

1 変更記録

- 2014/09/01 作成
- 2014/09/23 「リアル粒子のサンプル」が領域クラスに属することに決定. 境界条件を考慮した動作概略の記述. コーディング規約の記述.
- 2014/09/30 「リアル粒子のサンプル」が粒子群クラスへ移動. 境界条件を考慮した動作概略の記述を整理.
- 2014/09/30 詳細記述のファイル入力、粒子交換のセクションを追加。
- 2014/10/07 詳細記述の領域クラスセクションを追加。

2 TODO リスト

- 実装関係 (1.0)
  - 周期境界の短距離力
  - 長距離力のツリー判定に幾何中心
  - 長距離力の quadrupole moment
  - PM の組み込み
  - 相互作用リストを返すメソッド
  - ツリーのコピー
  - ファイル IO
  - エラー処理
- 性能評価
- リリース版整備
  - サンプルコード
  - テストスィート
  - confiture, make, make check, make install, etc.
  - インストールに必要になることもあるソフトウェア (MPI, FFTW)
- 文書
  - 外部仕様書
  - チュートリアル
  - 論文
  - 英語版ドキュメント
- ユーザーサポート体制
- デモ (スパイラル, PPPT, cosmology, アルゴン, サンタバーバラクラスタ, ダムブレイク, 粉体なにか, MPI, 渦糸法)
- 実装関係 (1.0 –)
  - 可視化対応 (zindaiji, Yt, splash etc.)
  - 任意境界条件
  - アクセラレータ対応
  - fortran 対応

3　はじめに

この文書は, Framework for Developing Particle Simulator (FDPS) の並列版実装仕様書である. **??**節に動作概略, **??**節に詳細仕様, **??**節にテスト定義を記述する.

4　FDPS 並列版の動作概略

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. 粒子シミュレーションは, 相互作用の計算と軌道積分の繰り返しであるが, FDPS は相互作用の計算のみを行う. 軌道積分はユーザーが行う. また, FDPS が支援する相互作用は, 2 粒子間相互作用の重ね合わせで記述できるもののみである. この相互作用はユーザー定義である. これ以外の相互作用の計算はユーザーが行う. FDPS は, 直交座標系, D 次元空間 ($D = 2, 3$), 様々な境界条件に対応している. 相互作用の計算は, マルチプロセス, マルチスレッド, SIMD 演算を用いて, 並列に処理される. FDPS は C++言語にて記述される.

対応する 2 粒子間相互作用を具体的に述べる. 大きく分けて 2 種類の相互作用に対応し, 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.

短距離力にも, 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.

対応した境界条件には 3 種類があり, その選択はユーザーが行う. 1 つめは非周期境界である. これは開境界に対応する. ユーザーが粒子を適切に配置して壁を作れば, 閉境界を作ることもできる. 2 つめは並進対称の周期境界である. これによって, x, y, z 軸方向の周期境界やシアリングボックスなどに対応できる. ??? 3 つめはユーザー定義の周期境界である. 例えば, 球の一部を取ってマントル対流を表現することも可能だろう. また鏡面境界もできるはずである.

この節の構成は以下の通り. **??**節では, 相互作用を計算するための動作を記述する. **??**, **??**節では, これらを実現するためのモジュールとそれらモジュールのインターフェースを概説する.

4.1　動作

FDPS において, 相互作用の計算は, 大きく 3 つの動作に分かれる. 1 つは, ルートドメインを, 1 プロセスが担当するドメインに分割することである. 2 つめは, 各プロセスが, 担当するドメイン内に存在するリアル粒子を持つように, リアル粒子を交換することである. 3 つめは, 実際の相互作用の計算である.

これらを 3 つのモジュールに対応させる. 1 つめは領域クラスである. このモジュールは, ルートドメインの分割を行い, ドメイン情報を持つ. 2 つめは粒子群クラスである. これは粒子交換を行い, 粒子情報を持つ. 3 つめは相互作用ツリークラスである. これは相互作用計算を行い, それに必要なツリー情報を持つ.

以上 3 つのモジュールは, MPI を用いて並列に処理される. MPI で使用される変数は, 通信用データクラスというモジュールで管理される. 通信用データクラスはシングルトンパターンを用いて実装されるクラスである. どこからでもアクセスできるため, ここでは特に記述しない.

4.2 モジュール概略

この節では, **??**節で触れた 3 つのモジュール, すなわち領域クラス, 粒子群クラス, 相互作用ツリークラスの動作を記述する. 様々なデータ構造が現れるが, 断らないかぎり, そのデータ構造はそのクラスに属す.

4.2.1 領域クラス

このモジュールはルートドメインのドメインへの分割と, ドメイン情報の管理を行う. ルートドメインをドメインに分割する方法は, 各プロセスのリアル粒子のサンプルをすること, サンプルされたリアル粒子を参照して実際のドメイン分割すること, の 2 段階で行われる. ドメインは D 次元直方体である. 第 1 段階は粒子群クラスで行われる. ここでは第 2 段階の動作を概略する. なお, ルードドメインの分割に関わるパラメータの設定は, ここでは記述しない.

4.2.1.1 ルートドメインの分割

各プロセスは, 粒子群クラスからサンプル粒子の位置ベクトルの配列 pos_sample_ptcl_(この配列は C++ ベクタに変更される可能性がある. 以下で現れる配列も同様である) を受け取り, 配列 pos_sample_に保存する. ある 1 つのプロセス (この節の中でのみアルファプロセスと呼ぶ) は, 全プロセスの配列 pos_sample_を集めて, 配列 pos_sample_tot_に保存する. アルファプロセスは, 配列 pos_sample_tot_を使って, サンプル粒子が適切に配分されるように, ルートドメインをドメインに分割する. さらに, そのドメインを表す D 次元直方体のデータを配列 domain_glb_に保存する. 最後に, アルファプロセスは配列 domain_glb_を全プロセスに放送する. 各プロセスは, アルファプロセスから受け取った配列を, 各々の配列 domain_glb_に保存する.

4.2.2 粒子群クラス

このモジュールは 2 つのことを行う. 1 つめは, ルートドメインの分割に必要な粒子のサンプルである. 2 つめは, リアル粒子の交換である. なお, リアル粒子のフル粒子データをファ

イルから読み込むことや, ファイルへの書き込むことは, このモジュールで行われるが, ここでは記述しない. リアル粒子のフル粒子データは各プロセスの配列 ptcl_にすでに存在するものとする

4.2.2.1 リアル粒子のサンプル

各プロセスは, リアル粒子のフル粒子データが保存された配列 ptcl_からランダムにリアル粒子をサンプルする. サンプルされたリアル粒子の位置ベクトルを抜き出し, ローカルなサンプル粒子の位置ベクトルの配列 pos_sample_ptcl_に保存する.

4.2.2.2 リアル粒子の交換

各プロセスは, リアル粒子のフル粒子データの配列 ptcl_とドメインの配列 domain_glb_(領域クラスに属す) を比べる. もし自分の持つリアル粒子で自分のドメインからはみだしたリアル粒子があれば, 適切なドメインを担当するプロセスへ, そのリアル粒子を送信し, そのフル粒子データを配列 ptcl から削除する. 他のプロセスからリアル粒子をを受信したならば, そのフル粒子データを配列 ptcl_ へ加える.

4.2.3 相互作用ツリークラス

相互作用の計算は次の 6 段階で行われる. i) リアル粒子のフル粒子データから相互作用の計算に必要なデータを抜きとる. ii) 自分のプロセスが持つリアル粒子のみからなるローカルツリーを構築する. iii) 自分のプロセスが持つリアル粒子への作用の計算に必要なリアル粒子を他のプロセスから受け取る. iv) 自分のプロセスが持つリアル粒子と他のプロセスから受け取ったリアル粒子からなるグローバルツリーを構築する. v) グローバルツリーを用いて, 自分のプロセスが持つリアル粒子への作用を計算する. vi) 計算した作用をリアル粒子のフル粒子データへ書き込む. 以下では各節ごとに i) から vi) の概略を記述する.

4.2.3.1 相互作用計算に必要な粒子データの読込

各プロセスが, リアル粒子のフル粒子データの配列 ptcl_(粒子群クラスに属す) から別々のデータを抜きとって, 3 つの配列に保存する. a) ローカルツリーの作成に必要なデータを抜きとりツリー粒子データの配列 tp_buf_に保存する. b) 作用される粒子に必要なデータを抜きとり, i 粒子データの配列 ep_i_buf_に保存する. c) 作用する粒子に必要なデータを抜きとり, j 粒子データの配列 ep_j_buf_に保存する.

4.2.3.2 ローカルツリーの作成

各プロセスが, 配列 tp_buf_を使って, 2^D 分木構造であるローカルツリーを作り, 配列 tc_loc_[0] に保存する. ローカルツリーのリーフセルには, リーフセルに含まれるツリー粒

子データ, i 粒子データ, j 粒子データがそれぞれ, 配列 tp_loc_[0], ep_i_[0], ep_j_loc_[0](概念上はリスト) に保存されている.

ユーザー定義の境界条件の場合, これに追加してなされることがあるので, 記述する. ルートドメインに対して, 複数のイメージドメインが存在するはずである. これらそれぞれに対して, ローカルツリーを構築する. まず, ユーザープログラムから関数 pfunc_map_image_を受け取る. 各プロセスは自分が担当するリアル粒子に対するイメージ粒子を作る. これらのイメージ粒子を使って, ローカルツリーである 2^D 分木構造を作り, 配列 tc_loc_[id] に保存する (id はイメージドメインの ID). それぞれのリーフセルには, リーフセルに含まれるツリー粒子データ, i 粒子データ, j 粒子データがそれぞれ, 配列 tp_loc_[id], ep_i_[id], ep_j_loc_[id](概念上はリスト) に保存されている. この中には全くいらないローカルツリーもあるはずなので, そのようなものは作らない仕掛はある.

4.2.3.3 相互作用計算に必要な粒子の交換

各プロセスが, 自分のドメインに属するリアル粒子, リアル超粒子, イメージ粒子, イメージ超粒子のうち, 別ドメインの相互作用計算に必要なものを, ドメインデータの配列 domain_glb_(領域クラスに属す) とルートドメインのローカルツリーを表す配列 tc_loc_[0], イメージドメインのローカルツリーを表す配列 tc_loc_[id] を使って探す. 見つけたリアル粒子, リアル超粒子, イメージ粒子, イメージ超粒子をその別ドメインに送信する. ドメインに属するリアル粒子への作用を計算するのに必要な別ドメインのリアル粒子, リアル超粒子, イメージ粒子, イメージ超粒子を受信する.

受信したリアル粒子, リアル超粒子, イメージ粒子, イメージ超粒子のうち, グローバルツリーの作成に必要なデータを抜き出し, ツリー粒子データの配列 tp_buf_へ加える. さらに, 受信したリアル粒子とイメージ粒子は配列 ep_j_buf_に加え, 受信したリアル超粒子とイメージ超粒子は配列 sp_j_buf_に保存する.

4.2.3.4 グローバルツリーの作成

各プロセスが, ツリー粒子データの配列 tp_buf_を使って, 2^D 分木構造であるグローバルツリーを作り, 配列 tc_glb_に保存する. グローバルツリーのリーフセルには, リーフセルに含まれるツリー粒子データ, i 粒子データ, j 粒子データがそれぞれ, 配列 tp_glb_, ep_i_, ep_j_glb_(概念上はリスト) に保存されている.

4.2.3.5 相互作用の計算

各プロセスは, グローバルツリーである配列 tc_glb_を使って, 同じリアル粒子やリアル超粒子から作用を受けるリアル粒子のグループを作り, その i 粒子データを配列 ip_group_に保存する. グローバルツリーである配列 tc_glb_を使って, このリアル粒子のグループに作用するリアル粒子を探して配列 ep_j_に, リアル超粒子を探して配列 sp_j_に保存する. 粒子から粒子への作用を計算する関数 pfunc_ep_ep と, 超粒子から粒子への作用を計算する関数 pfunc_ep_sp をユーザーからもらう. このとき配列 ip_group_, ep_j_, sp_j_を使って作用を

計算し, 結果を配列 force_i_に書き込む. 自分のドメインに属する全リアル粒子への作用を計算し終わるまでこれを繰り返す.

4.2.3.6　相互作用の書込

各プロセスは, 作用の結果が保存された force_i_をリアル粒子のフル粒子データの配列 ptcl_(粒子群クラスに属す) にコピーする.

4.2.4　データ構造まとめ

この節で登場したデータ構造がどのクラスに属するか記述する.

領域クラスは, ドメインデータの配列 domain_glb_, 自分のプロセスからサンプルされたリアル粒子の位置ベクトルの配列 pos_sample_, 全プロセスからサンプルされたリアル粒子の位置ベクトルの配列 pos_sample_tot_を持つ.

粒子群クラスは, リアル粒子のフル粒子データの配列 ptcl_, サンプル粒子の位置ベクトルの配列 pos_sample_ptcl_を持つ.

相互作用ツリークラスは, ツリー粒子データの配列 tp_buf_, i 粒子データの配列 ep_i_buf_, j 粒子データの配列 ep_j_buf_, ローカルツリーの配列 tc_loc_[], ローカルツリーのリーフセルにぶらさがるツリー粒子データの配列 tp_loc_[](概念上はリスト), i 粒子データの配列 ep_i_[](概念上はリスト), j 粒子データの配列 ep_j_loc_[](概念上はリスト), グローバルツリーのである配列 tc_glb_, グローバルツリーのリーフセルにぶらさがるツリー粒子データのリスト tp_glb_(概念上はリスト), j 粒子データのリスト ep_j_glb_(概念上はリスト), 実際の相互作用をする際に使う i 粒子データの配列 ip_group_とその相互作用リストである j 粒子データの配列 ep_j_とリアル超粒子データの配列 sp_j_, 作用の結果の配列 force_i_を持つ.

4.3　モジュールインターフェース

この節では領域クラス, 粒子群クラス, 相互作用ツリークラスの 3 つのモジュールでどのようなデータ構造が入出力されるのかを記述する. またユーザープログラムから供給されるものも記述する. この模式図を図**??**に載せる.

4.3.1　領域クラス

このクラスへの入力は, 粒子群クラスから pos_sample_ptcl_である. このクラスからの出力は, 粒子群クラスへの domain_glb_, 相互作用ツリークラスへの domain_glb_である.

4.3.2　粒子群クラス

このクラスへの入力は, 領域クラスからの domain_glb_, 相互作用ツリークラスからの force_i_, ユーザープログラムからのフル粒子データである. このクラスからの出力は, 領域クラスへの pos_sample_ptcl_, 相互作用ツリークラスへの ptcl_である.

モジュール	名前	データ構造	要素の型
領域クラス	domain_glb_	配列	D 次元直方体
	pos_sample_	配列	位置ベクトル
	pos_sample_tot_	配列	位置ベクトル
粒子群クラス	pos_sample_ptcl_	配列	位置ベクトル
	ptcl_	配列	フル粒子データ
相互作用ツリークラス	tp_buf_	配列	ツリー粒子データ
	ep_i_buf_	配列	i 粒子データ
	ep_j_buf_	配列	j 粒子データ
	tc_loc_[]	配列	ツリーセル
	tp_loc_[]	配列	ツリー粒子データ
	ep_i_[]	配列	i 粒子データ
	ep_j_loc_[]	配列	j 粒子データ
	tc_glb_	配列	ツリーセル
	tp_glb_	配列	ツリー粒子データ
	ep_j_glb_	配列	j 粒子データ
	ip_group_	配列	i 粒子データ
	ep_j_	配列	j 粒子データ
	sp_j_	配列	超粒子データ
	force_i_	配列	作用の結果
ユーザープログラム	pfunc_map_image_	関数ポインタ (ファンクタ)	–
	pfunc_ep_ep_	関数ポインタ (ファンクタ)	–
	pfunc_ep_sp_	関数ポインタ (ファンクタ)	–

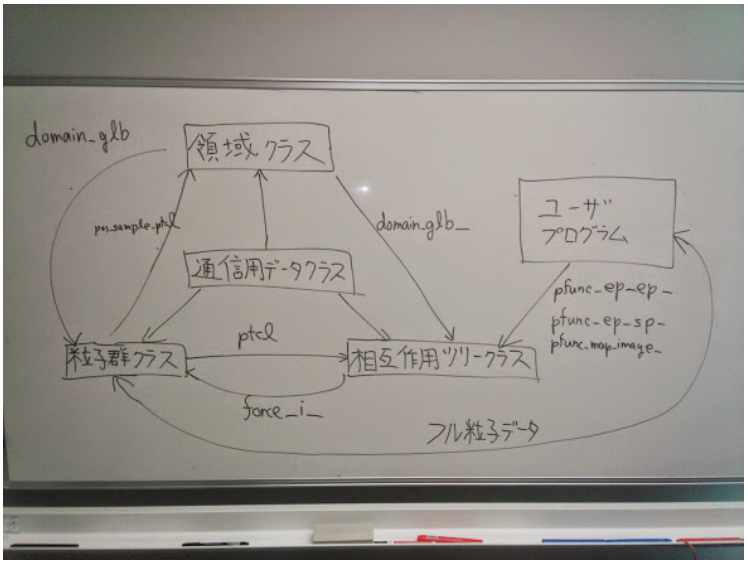

図 1: モジュールインターフェースの模式図

4.3.3 相互作用クラス

このクラスへの入力は, 領域クラスからの domain_glb_, 粒子群クラスから ptcl_である. このクラスからの出力は, 粒子群クラスへの force_i_, ユーザープログラムからの関数 pfunc_map_image_, pfunc_ep_ep_, pfunc_ep_sp_である.

4.3.4 ユーザープログラムから供給される関数

ユーザープログラムからの出力は次の通り. 粒子群クラスへのフル粒子データ, 相互作用ツリークラスへのイメージ粒子を与える関数 pfunc_map_image_と作用を計算する関数 pfunc_ep_ep, pfunc_ep_sp である.

5 FDPS 並列版詳細記述

5.1 メモリー管理

5.1.1 ReallocatableArray クラス

この節では, ReallocatableArray クラスの詳細記述を行う.

ReallocatableArray クラスは以下の様に記述される。メンバ data_は T 型の配列で、格納最大要素数を capacity_、実際の要素数を size_に格納する。

ソースコード 1: DomainInfo

```
1  namespace  ParticleSimulator{
2      template<class T>
3      class ReallocatableArray{
4      public:
5          T * data_;
6          int size_;
7          int capacity_;
8          ReallocatableArray();
9          ReallocatableArray(int capa);
10         ReallocatableArray(int size, const T & val);
11         ~ReallocatableArray();
12         void reserve(const int n);
13         int size() const;
14         int capacity() const;
15         const T & operator [] (const int i) const;
16         T & operator [] (const int i);
17         T & front(){ return data_[0]; }
18         T & back(){ return data_[size_-1]; }
19         T * data(){ return data_; }
20         const T * data() const { return data_; }
21         void push_back(const T & val);
22         void resizeNoInitialize(const int n)
23         T * getPointer(const int i=0) const;
24         void pushBackNoCheck(const T & val){
25         long getMemSize() const;
26         void dump(const std::string str="");
27
28         }
29     };
30 }
```

コンストラクタは以下の 3 種類。

```
ReallocatableArray();
```

空のクラスを作成する。std::vector と同じ振る舞いをする。

```
ReallocatableArray(int capa);
```

capacity_に引数 capa を代入する。この際、size_は 0 である。std::vector と振る舞いが異なる。

```
ReallocatableArray(int size, const T & val)
```

以下はデストラクタである。

```
~ReallocatableArray()
```

data_の領域が開放される。

capacity_と size_に引数 size を代入する。また size 個の要素を引数￥tt val で初期化する。std::vector と同じ振る舞いをする。

メソッド reserve(),size(),capacity(),front(),back(),data(),push_back(), 演算子 [] は std::vector のそれとほぼ同じである。

以下には、AllocatableArray クラス特有のメソッドを記述する。

```
public:
void PS::AllocatableArray::resizeNoinitialize(const int n);
```

size_に引数 n を代入する。n が capacity_を超えた場合は capacity_を n の 2 倍に変更する。std::vector::resize との違いは付け加えた要素の初期化はされない。

```
public:
void PS::AllocatableArray::puchBackNoCheck(const T & val);
```

val を data_の末尾に付け加え、size_を一つインクリメントする。std::vector::push_back と違い、配列の領域 (capacity_) のチェックを行わない。

```
public:
T * PS::AllocatableArray::getPointer(const int i=0);
```

メンバ data_の i 番目のポインタを返す。

```
public:
long getMemSize();
```

data_が使用しているバイト数を返す。

```
public:
void dump(const std::string str="");
```

標準出力に引数 str,size_,capacity_を表示する。

5.2 時間計測

FDPS では簡易的な時間計測クラス (Timer) を用意した。
Timer クラスは以下のように記述されている。

ソースコード 2: Timer

```
1  namespace ParticleSimulator{
2      class Timer{
3      private:
4          enum{
5              SPLIT_CAPACITY = 128,
6              STRING_SIZE = 1024,
7          };
8          double begin_time_;
9          double end_time_;
10         double split_time_[SPLIT_CAPACITY];
11         char func_name_[SPLIT_CAPACITY][STRING_SIZE];
12         size_t cnt_;
13     public:
14         void reset();
15         void start();
16         void stop(const char * str='\0');
17         void restart(const char * str='\0');
18         void dump(std::ostream & fout);
19     };
20 }
```

メソッド、reset でクラスの初期化を行い、start で現在の時刻をメンバ begin_time_に入れる。stop で時間計測を止め、現在時刻と begin_time_との差をメンバ split_time_[cnt_] に格納する。続けて測定を行いたいときは stop ではなく restart を使う。この関数はそこまでの時間を計測し、split_time_[cnt_] にその値を入れる。restart するたびにカウンター変数 cnt_はインクリメントされる。dump は全スプリットタイムや最もそのスプリットタイムのかかったプロセスのランクとその時間を引数 fout に表示する。

stop や restart にある引数でそのスプリットタイムを識別するための文字列をメンバ func_name_入れる事が出来る。この名前は dump を行った時に出力される。

5.3 標準モジュール詳細

この節では, 4 つのモジュール, すなわち通信用データクラス, 領域クラス, 粒子群クラス, 相互作用ツリークラスの動作の詳細、実装方法を記述する。様々なデータ構造が現れるが, 断らないかぎり, そのデータ構造はそのクラスに属す.

5.3.1 通信用データクラス

5.3.1.1

5.3.2 領域クラス

この節では, 領域クラスの詳細記述を行う. 領域クラスの動作は, 初期化, ルートドメインの分割である.

領域クラスは以下の様に記述されている。このクラスはグローバルにすべきか？

ソースコード 3: DomainInfo

```
1     class DomainInfo{
2     private:
3         F32vec * pos_sample_tot_;
4         F32vec * pos_sample_loc_;
5         F32ort * pos_domain_;
6         F32ort * pos_domain_temp_;
7         F32 coef_ema_;
8         S32 target_number_of_sample_particle_;
9         S32 number_of_sample_particle_tot_;
10        S32 number_of_sample_particle_loc_;
11        S32 n_domain_[DIMENSION];
12        F32ort pos_root_domain_;
13        bool first_call_by_initialize;
14        bool first_call_by_decomposeDomain;
15        S32 boundary_condition_;
16        bool periodic_axis_[DIMENSION];
```

5.3.2.1 初期化

領域クラスの初期化は, 以下の関数を全プロセスで呼び出すことで行われる.

```
public:
void PS::DomainInfo::initialize(const PS::F32 coefficient_EMA);
```

coefficient_EMA は移動平均の係数である. ここでは移動平均の引数が設定される.
この関数は必ず呼び出す必要がある.

5.3.2.2 領域の分割数の設定

以下の関数を全プロセスで呼びだすことで, x, y, z 軸方向の分割数が設定される.

```
public:
void PS::DomainInfo::setDomain(const PS::S32 nx,
                               const PS::S32 ny,
                               const PS::S32 nz);
```

nx, ny, nz はそれぞれ, x, y, z 軸方向の分割数である. nx, ny, nz の積は MPI プロセス数でなければならない.

この関数は呼出さなくてもよい. この場合, nx, ny, nz のデフォルト値が採用される. デフォルト値は, 以下のようになっている. 3 つの積が MPI プロセス数であり, かつそれぞれが最も近い値である. 大小関係は, nx >= ny >= nz となっている.

5.3.2.2.1 境界条件の設定

境界条件の設定の設定は, 以下の関数を全プロセスで呼び出すことで行われる.

```
void setBoundaryCondition(enum BOUNDARY_CONDITION bc);
```

引数については、仕様書参照。この関数を呼び出すと、境界条件がメンバ boundary_condition_ に格納される。また、周期境界条件の場合は周期境界となる軸の periodic_axis_が true となる（開放境界の軸は false)。

以下の関数によって境界条件を得る。

```
S32 getBoundaryCondition();
```

5.3.2.2.2 ルートドメインの設定

ルートドメインの設定の詳細記述を行う. ここでは, ルートドメインの形が直方体であるとする. これは, 開放境界条件と周期境界条件の場合にのみ対応できる. 将来的には, この節の名前を「開放境界条件と周期境界条件の場合のルートドメインの設定」などに変更する必要があるかもしれない.

ルートドメインの設定は, 以下の関数を全プロセスで呼び出すことで行われる.

```
public:
void PS::DomainInfo::setRootDomain(const PS::F32ort particle_domain[]);
```

引数 particle_domain は, 直方体を表す変数である. この直方体がルートドメインとなる. ルートドメインは, 最小値側で閉境界, 最大値側で開境界とする.

この関数は呼出さなくてもよい. この場合, ルートドメインにはデフォルト値が採用される. ルートドメインのデフォルト値は, 単精度浮動小数の最大値と最小値である (それぞれ

std::numeric_limits<PS::F32>::max(), - std::numeric_limits<PS::F32>::max()) と与えられる).

5.3.2.3 ルートドメインの分割

この節では, ルートドメインの分割の詳細記述を行う. これは, ルートドメインの設定, サンプル粒子の回収, 分割の実行の順に行われる.

この動作は, ドメインの分割のために参照する粒子の種類が 1 つの場合, 以下の関数を呼び出すだけですむ.

```
public:
void PS::DomainInfo::decomposeDomainAll(not yet defined);
```

以下, 続く.

5.3.2.3.1 サンプル粒子の回収

サンプル粒子の回収について記述する.

サンプル粒子の回収をするには, 以下の関数を全プロセスで呼び出すだけでよい.

```
public:
void PS::DomainInfo::collectSampleParticle(const PS::ParticleSystem & psys,
                                           const PS::F32 weight=1.0,
                                           const bool clear=true);
```

第 1 引数 psys は, サンプルの対象となる粒子の粒子群クラスである. 第 2 引数 weight は, 各プロセスでサンプルする粒子のウェイトである. 第 3 引数 clear はすでにサンプルした粒子のデータを消去するかしないかを決めるものである. clear が, true ならば消去し, false ならば消去しない.

この関数の内部では, psys からサンプルした粒子の位置座標を, 領域クラスのメンバ変数である F32vec 型の配列 pos_sample_loc_に格納する. 各プロセスでサンプルする粒子の数の決定と, 実際のサンプルはこの関数が呼び出す関数 PS::ParticleSystem::getSampleParticle が行う.

第 3 引数が true だった場合は, pos_sample_loc_を空にしてから, 上の動作を行う. 第 2 引数が false だった場合は, pos_sample_loc_を空にせずに, 上の動作を行う.

サンプル粒子が全粒子のサブセットであることをテスト. あと, インデックスの平均が 3sigma くらいに入ってるかどうか

5.3.2.3.2 分割の実行

分割の実行に関する詳細記述を行う.

分割の実行を行うには, 全プロセスで, 以下の関数を呼び出すだけでよい.

```
public:
void PS::DomainInfo::decomposeDomain();
```

この関数の内部では以下のことを行う．これらはすべてある 1 つのプロセス (この節ではルートプロセスと呼ぶ) で行う．行うことは, Orthogonal Multi Section (Makino 2004) と同じである．

ルートプロセスに全プロセスから配列 pos_sample_loc_を集め, 配列 pos_sample_tot_に格納する．

この配列 pos_sample_tot_ を使ってルートドメインを分割する．まず, ルートドメインを x 軸方向に nx 個のスラブに分割する．次にこのスラブそれぞれを y 軸方向に ny 個のカラムに分割する．最後にこのカラムそれぞれを z 軸方向に nz 個のドメインに分割する．これらのドメインの境界を配列 pos_domain_temp_に格納する．

配列 pos_domain_temp_, 配列 pos_domain_, 変数 coefficient_EMA を使って, 移動平均を取り, ドメインの境界を確定する．この確定したドメインの境界を配列 pos_domain_に格納する．ルートプロセスは, 配列 pos_domain_を全プロセスに放送する．

decomposeDomain: 分割したあとの粒子の分布がサンプル粒子の数に合ってるかどうか

5.3.3 粒子群クラス

5.3.3.1 ファイル入力

ここでは、ユーザーが用意した粒子の初期条件ファイルの入力メソッドについて記述する。

ファイル入力を行う関数は以下の二つである。

```
void loadParticleSingle(
                        const char * filename,
                        const char * mode,
                        void (*(FullParticle::pfunc))(FILE *));
```

```
void loadParticle(
                  const char * format,
                  const char * mode,
                  void (*(FullParticle::pfunc))(FILE *))
```

前者は一つの初期条件ファイルを読み込む場合、後者はファイルが分割されている場合に使う。

これらの関数では、以下の様にメンバが設定、変更される。

- プロセスの持つ粒子数 n_ptcl_loc_を設定。
- 全プロセスが持つ粒子数の合計 n_ptcl_tot_を設定。
- ptcl_の配列サイズ n_ptcl_loc_max_を設定。

- ptcl_の配列の確保と初期条件ファイルからの代入。

まず、loadParticleSingle(...)。大まかな流れはプロセス番号0のプロセスがユーザーが用意した初期条件ファイルからユーザーが定義した粒子読み取り関数（フルパーティクルクラスのメソッド）を使って粒子データを読み取り、各プロセスの持つ粒子数が同じになるようにフルパーティクルデータを送信する。ユーザーが用意する初期条件ファイルはアスキーの場合は一行に一粒子の情報が書き込まれていなければならない。バイナリーの場合は一粒子ごとまとまって情報が書かれている必要がある。アスキー、バイナリーは第二引数で"r"、"rb"を選択する事で切り替える。これら以外のものが設定された場合は例外を送出する。粒子読み取り関数は以下の様に定義されている。

```
void FullParticle::loadOneParticleAscDec(FILE *fp) {
    fscanf(fp, "%lf%lf%lf%lf%lf%lf%lf",
                  &this->mass,
                  &this->pos[0], &this->pos[1], &this->pos[2],
                  &this->vel[0], &this->vel[1], &this->vel[2]);
}
```

初期条件ファイルには、粒子数の情報が無いので、最初にファイルを終りまで読み込み粒子数を数え、それをメンバn_ptcl_tot_に格納し、n_ptcl_tot_とプロセス数から、各プロセスが持つ粒子数n_ptcl_loc_とptcl_の配列の大きさn_ptcl_loc_max_を決定し、ptcl_の配列を確保する。

n_ptcl_loc_max_はデフォルトでは以下の様に決定する。

$$\text{n\_ptcl\_loc\_max\_} = \text{n\_ptcl\_tot\_}/\text{プロセス数} * 4 + 1000; \tag{1}$$

配列の大きさの決め方はユーザーも定義出来るようにしたいが、どの様にするかは決まっていない。

n_ptcl_loc_は各プロセスで同じになるようにする。n_ptcl_tot_がプロセス数で割り切れない場合はプロセス番号の若いプロセスから順に粒子を一つ多く持つようにする。ここまで実行したのち、再びファイルを頭から読み込み、データをptcl_に格納する。

次に、loadParticle(...)の詳細について述べる。ユーザーが用意する粒子の初期条件ファイルの形式や読み込み関数はloadParticleSingl(...)の場合と同じである。第一引数fileformatでファイル名のフォーマットを指定する。フォーマットの指定方法は標準Cライブラリの関数printfの第1引数と同じである。ただし変換指定は必ず2つであり、その指定子はどちらも整数である。1つ目の変換指定にはそのジョブの全プロセス数が、2つ目の変換指定にはプロセス番号が入る。例えば、引数がnbody_%03d_%03d.initならば、全プロセス数64のジョブのプロセス番号12のプロセスは、nbody_064_012.initというファイルを読み込む。フォーマットの指定正しくない場合は例外を送出する。

各プロセスは割り当てられた初期条件ファイルから粒子データを読み取りptcl_に格納する。初期条件ファイルには、粒子数の情報が無いので、最初にファイルを終りまで読み込み粒子数を数え、それをメンバn_ptcl_loc_に格納し、MPI::Allreduceを使ってn_ptcl_loc_の和を

n_ptcl_tot_に格納する。n_ptcl_tot_とプロセス数から、ptcl_の配列の大きさn_ptcl_loc_max_を loadParticleSingle(...) の場合と同じ方法で決定し、ptcl_の配列を確保する。もし、n_ptcl_loc_max_がn_ptcl_loc_より、小さくなってしまった場合は例外を送出する。

5.3.3.2 ファイル出力

5.3.3.3 初期設定

```
public:
void PS::ParticleSystem::initialize()
```

いまのところやることがないが, そのうちできるかもしれないので, とりあえず残しておく.

5.3.3.4 粒子配列のメモリ確保

各プロセスにおける粒子配列のメモリ確保は, 以下の関数を各プロセスで呼び出すことで行われる.

```
public:
void PS::ParticleSystem::createParticle(const PS::S32 n_limit)
```

引数 n_limit は粒子配列のサイズである.
この関数は必ず呼出されなければならない.

5.3.3.5 サンプル粒子数の設定

```
public:
void PS::ParticleSystem::setAvarageTargetNumberOfSampleParticlePerProcess
(const PS::S32 nsampleperprocess)
```

引数は, 領域分割のためにサンプルする粒子の目標数. この目標数は, 各プロセスがある粒子種で集める粒子数の平均である. この粒子数と全プロセス数から, 全プロセスでサンプルすべき粒子数を求めることができ, この数がn_smp_ptcl_tot_に格納される.

この関数は呼出さなくてもよい. この場合, デフォルト値が採用される. デフォルト値は 30 である.

5.3.3.6 リアル粒子のサンプル

各プロセスで以下の関数を呼び出すと, 各プロセスでその担当粒子から粒子のサンプルが行われる.

```
public:
void PS::ParticleSystem::getSampleParticle(PS::S32 & number_of_sample_particle,
                                           PS::F32vec pos_sample [],
                                           const PS::F32 weight)
```

第 1, 第 2 引数は出力である. 第 1 引数 number_of_sample_partile はサンプルする粒子の数, 第 2 引数 pos_sample はサンプルした粒子の位置座標である. 第 3 引数は入力であり, そのプロセスでサンプルすべき粒子の数のウェイトである.

サンプルする粒子の数は (PS::S32) (weight * n_smp_ptcl_tot_ / weight_all) として与えられる. weight_all は全プロセスの weight の和である.

サンプルする粒子はメルセンヌ・ツイスター法を用いたモンテカルロ法で選ばれる.

この関数は領域クラスのメンバ関数である collectSampleParticle で呼出されるだけであり, ユーザーに使用されることはない.

getSampleParticle: テスト DomainInfo::collectSampleParticle と同じような感じ

5.3.3.7 リアル粒子の交換

この節では, リアル粒子の交換に関する詳細記述を行う. この動作は以下の関数を全プロセスで呼び出すことで行われる.

```
public:
void PS::ParticleSystem::exchangeParticle(const PS::DomainInfo & dinfo)
```

手順は以下の 2 段階. 1 段階目では, 領域クラスのメンバ変数で PS::F32ort 型の配列 pos_domain_を使って, 各粒子がどのドメインに入るかを計算する. 2 段目では, 自プロセス以外が担当するドメインに入る粒子があれば, フルパーティクルデータごと該当プロセスへ送信する.

1 段階目を記述する. これは各プロセスで行われる. 行われることは以下の通り. 各粒子に対して, 自分のドメインに入るか, 入らないかを調べる. もし入らない場合, その粒子が入るドメインを探す. 探し方は以下の通り. まず, その粒子が入るスラブを探す. 次に, その粒子が, そのスラブ内のどのカラムに入るかを探す. 最後に, その粒子が, そのカラム内のどのドメインに入るかを探す. 上記のスラブ, カラム, ドメインの探し方は, 二分探索である.

2 段階目を記述する. 各プロセスは, 他のプロセスにフルパーティクルデータを送る. ptcl_には, 元々自プロセスが持っていた粒子が先頭, 他プロセスから送られてきた粒子は後に置くことにする.

5.3.4 相互作用ツリークラス

5.3.4.1 全体的な流れ

粒子群クラスから粒子情報をもらい、それを内部の EPI,EPJ クラスの配列に情報を格納する。さらに、EPJ(or I, 相互作用によって異なる) から TP(TreeParticle) クラスを作る。TP

クラスは粒子の位置座標のモートンキーと EPJ(or I) のアドレスを持つ。アドレスは配列のインデックスで S32 型で持つことにする。これにより、粒子が同じ順番で並んでいれば、EPI でも EPJ でも指すことが出来る。またポインタで持つ場合に比べてクラスが軽くなり、配列の再確保なども容易になる（実装するかは未定)。ここまでで、粒子群クラスの情報のコピーを持つので、力を計算するまでの間に粒子群クラスから情報をもらう必要はない。次に TP を使ってローカルツリーを作る。ツリー構造は TC(TreeCell) クラスの配列によって実現され、これは内部に子セルへの先頭アドレス、自セル内にある先頭の TP(EPI,EPJ) へのアドレスを持つ。さらに TC クラスはセルのモーメントやセルの外側境界の情報などを持つ MOMENT クラス (長距離力の場合のみユーザーが定義する事も出来る) を持つ。長距離力の場合は SPJ は MOMENT クラスから情報をもらう事になる。具体的なツリー構築法は決めていない (挿入か、それ以外か) が、ツリー構築時点で粒子はツリーを辿った順に並べる。これにより、力やモーメント計算などで起こる間接参照を少なく出来る。

ローカルツリー構築、モーメント計算後、相互作用に応じた通信を行い、EPJ（長距離力の場合は SPJ クラスも）を送る。長距離力の場合、送られて来た EPJ、SPJ クラスから TP クラスを作る。ローカルツリーの場合と同様に TP クラスからグローバルツリーを構築する。モーメント計算においては TC の指す粒子のアドレスが EPJ か SPJ かを区別する必要がある。これは TC の粒子のアドレスの MSB で区別しようと考えているがまだ未定。

以下は TP と TC の例。

ソースコード 4: TreeParticle

```
1 namespace ParticleSimulator{
2     class TreeParticle{
3     public:
4         U64 key_; //モートンキー
5         S32 adr_ptcl_; //対応する粒子のアドレス。
6     };
7 }
```

ソースコード 5: TreeCell

```
1 namespace ParticleSimulator{
2     template<class Tmom>
3     class TreeCell{
4     public:
5         S32 n_ptcl_; //持っている粒子数
6         S32 adr_tc_; //子セルの先頭アドレス
7         S32 adr_ptcl_; //粒子の先頭アドレス
8         Tmom mom_; //モーメント情報
9     };
10 }
```

5.3.4.2 初期化

以下の関数により初期化を行う。

```
void initialize(DomainInfo * dinfo,
                const F32 theta = 0.5,
                const S32 nleafmax = 8,
                const S32 ngroupmax = 64);
```

メンバ theta_、n_leaf_max_、n_group_max_を設定。さらに、FORCE, EPI, EPJ, TP, TC, SPJ の配列を確保する。配列サイズの決め方はまだ決めていない。

5.3.4.3 相互作用計算に必要な粒子データの読込

粒子群クラスから相互作用ツリークラスへの粒子データの読み込みは以下の関数によって行う。

```
void setParticleFromFullParticle(const Tpsys & psys, const bool clear);
```

引数の意味は仕様書参照。粒子群クラスのメンバ ptcl_から、epi_org_、epj_org_、に必要なデータを抜き出す。この時、epi_org_、epj_org_については、ユーザーが定義したメソッド EPI::copyFromFP(...), EPJ::copyFromFP(...) によって行われ、tp_loc_については、TP::setFromEP(...) によって行われる。

5.3.4.4 ツリーのルートセルの決定

以下の二つはツリーのルートセルを決定する為の関数である。この関数を呼ぶことで、ツリーのルートセルの情報が決まる (メンバ変数 pos_root_cell_, center_, length_に値が入る)。FDPS では、ローカルツリー、グローバルツリーで同じルートセルを使う。最初の関数は領域クラスから、境界条件の情報をもらい、適切なルートセルを決定する。開放境界の場合は、全ての粒子を内包する最少の立方体をルートセルにする。周期境界条件の場合は、全ての粒子を内包する最少の立方体に最大の探査半径のマージを取った立方体を使う。

```
void setRootCell(const DomainInfo & dinfo);
```

下の関数を使うと、直接ツリーのルートセルの中心 (cen) と一辺の長さ (len) を指定する事が出来る。ここで指定された範囲外に粒子がある場合の動作は未定義である。

```
void setRootCell(const F32 len, const F32vec & cen=F32vec(0.0));
```

5.3.4.5 ローカルツリーの作成

以下、2 種類の方法を考える。最初は後者の方法を実装する。

粒子挿入によるツリー作成

まず、tp_org_の粒子挿入によりツリーを作る。一つのセルにn_leaf_max_個以上の粒子が入ってきたら8個の子セルをtc_loc_から連続に取り、その最初の小セルの配列先頭からの相対アドレスをadr_tc_に格納。次に、ツリーを順に辿りその順番に粒子を並び替えtp_i_に格納する。

スレッド並列にする場合はまずn_leaf_max_より十分大きな粒子数(粒子数/スレッド数位?)のセルをリーフ(ここではブランチと呼ぶ)とするツリーを作り、各ブランチセルに対して各スレッドでツリーを独立に作れば良い。粒子の並び替えのスレッド並列では以下の様にする。各ブランチ粒子数のprefix sum(オフセット)を計算する。各ブランチを各スレッドで辿り粒子を並び替え、先程計算したオフセットを使ってtp_i_の適切な場所に格納すればよい。ep_i_も同じ順番に並べ替える。

モートンソートによるツリー作成

まず、epi_org_の位置座標からモートンキーを計算し、tp_loc_のkeyに値を代入する。tp_loc_のモートンキーを使ってソートを行い同じ順番にEPも並べ替える。この際並べ替えたEPはepi_sorted_,epj_sorted_に格納する。これら行う関数が以下である。

```
void mortonSortLocalTreeOnly();
```

ソートには粒子数の少ないところ(∼ 8000)ではsample sort, 粒子数の多いところではradix sortを使うつもりであるが現在はradix sortのみ実装。また、radix sortのprefix sumはまだスレッド並列化していない。ローカルツリーのモートンソートは以下の関数によって実行される。

図**??**は「京」1ノードでのstd::sort(1スレッド使用), sample sort(8スレッド使用), radix sort(8スレッド使用)の比較。ソートした構造体は64bitのkeyと32bit整数からなる。

sample sortはまず、各スレッドで各部分をソートする。その後ランダムに粒子を選び、ランダムサンプルされたものをシングルスレッドで再びソート。サンプルの値を使って、各スレッドで同じ粒子数位持つように、値の範囲を決め粒子を並べ替える。この時prefix sumを使って粒子の位置のオフセットを決める。並べ替えたら、各スレッドでソートをすればグローバルにソートされたことになる。ここのソートはマージソートを使う。

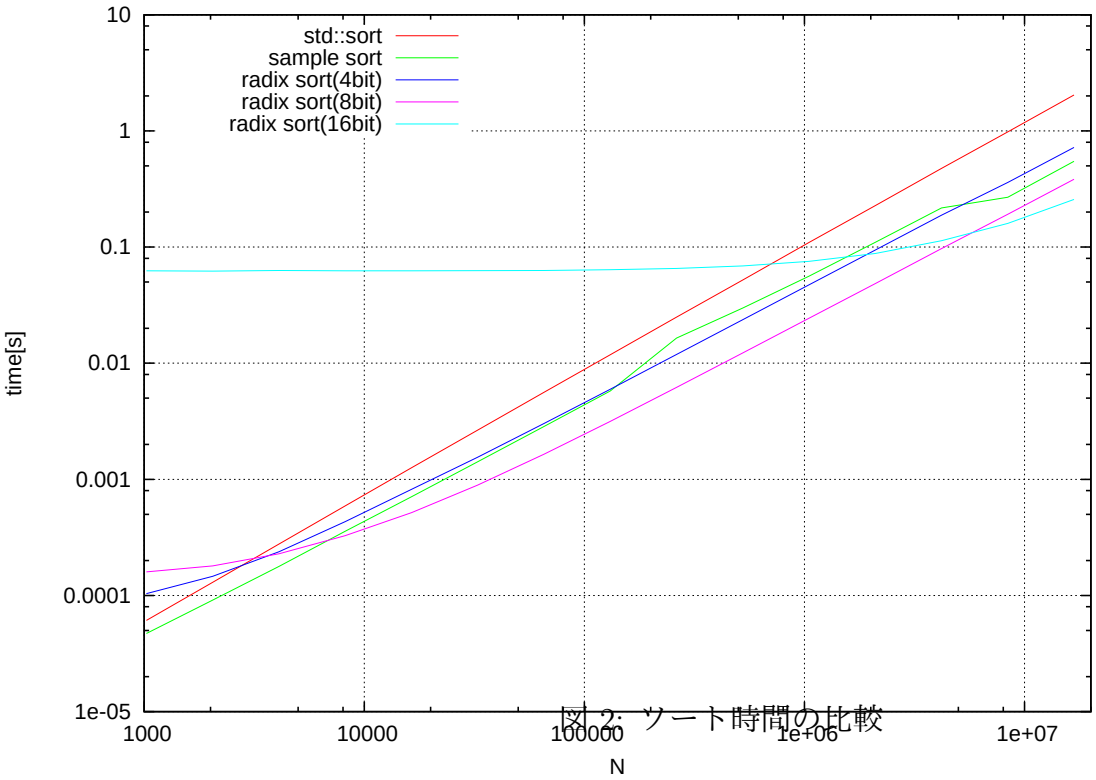

図 2: ソート時間の比較

次にツリー構造をトップからレベル毎に作っていく。ツリーのセルを作るのにはバイナリーサーチを使う。粒子数が少なくなってきたらリニアサーチを使った方が良いかもしれないが実装していない。子セルをアロケートする場所を決めるのに prefix sum を使っているが、スレッド並列化はしていない。以下の関数により、ツリー構造が作られる。この関数はモートンソート前に呼んではならない。

```
void linkCellLocalTreeOnly();
```

モーメントの計算は下のレベルから順に行う。ツリーセルはレベル毎に連続でメモリー上に並んでいるので、各レベルでスレッド並列で行う。モーメントの計算は次の関数で行う。

```
void calcMomentLocalTreeOnly();
```

図??は京で 8 スレッド使った時にローカルツリー作りにかかる時間。

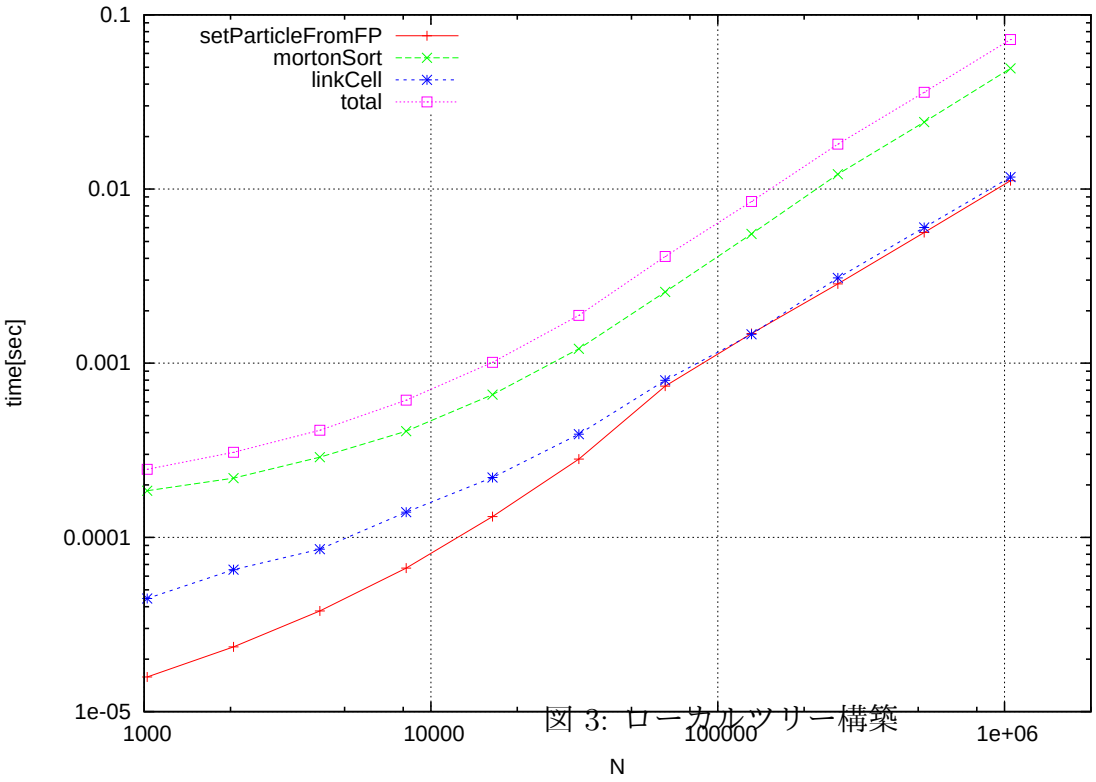

図 3: ローカルツリー構築

図??は Xeon(2.93GHz, Nehalem) で 6 スレッド使った時にローカルツリー作りにかかる時間。モーメント計算は重心の計算のみ。

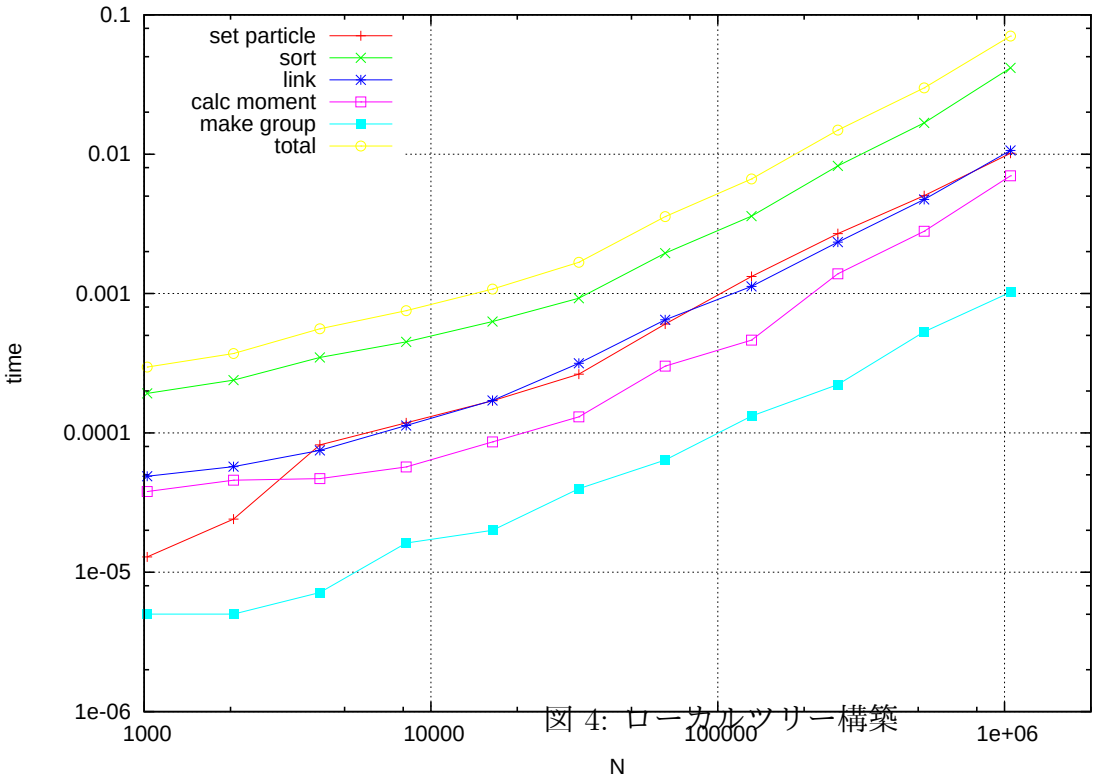

図 4: ローカルツリー構築

radix sort クラス

radix sort クラスは以下の様になっている。

ソースコード 6: RadixSort

```
1  namespace ParticleSimulator{
2      template<class T, int NBIT=8>
3      class RadixSort{
4      private:
5          int n_thread_;
6          int n_bucket_;
7          T mask_;
8          int ** bucket_size_;
9          int ** prefix_sum_;
10     public:
11         RadixSort();
12         template<class Tobj>
13         void lsdSort(Tobj * & val,
14                      Tobj * & val_buf,
15                      const int first,
```

```
16                          const int last);
17     };
18 }
```

注意として、このクラスは unsigned 型の整数のみに対応している。また、メソッド `lsdSort` で OpenMP を使っている。

クラステンプレートパラメータは最初がソートする対象の型 (U64)、`NBIT` は radix のビット数でデフォルトは 8bit。コンストラクタにより、メンバの領域や値が代入される。

```
template<class Tobj>
void lsdSort(Tobj * & val,
             Tobj * & val_buf,
             const int first,
             const int last);
```

- 引数

  `val`: 入力、出力。

  `val_buf`: 入力。

  `first`: 入力。

  `last`: 入力。

- 返り値

  なし。

- 機能

  `val` のメソッド `getKey()` により返される値の順に `val` をソートする。`val_buf` は `val` と同じ型、サイズの配列でソートする時のバッファー領域として使う。`first` はソートする配列の領域の最初のインデックス、`last` はソートする配列の領域の最後のインデックス。内部では OpenMP を用いて並列化されている。

使い方はまずオブジェクトを作り、メソッド `lsdSort()` を呼び出す。

ソースコード 7: RadixSort の例

```
1
2      PS::TreeParticle * data = new TreeParticle[n_size];
3      PS::TreeParticle * data_buf = new TreeParticle[n_size];
4      for(int j=0; j<n_size; j++){
5          data[j].setKey(PS::U64(abs(rand()))<<32 | PS::U64(abs(
               rand())));
6      }
```

```
7       PS::RadixSort<PS::U64, 8> RS;
8       RS.lsdSort(data, data_buf, 0, n_size-1);
```

5.3.4.6 相互作用計算に必要な粒子の交換

前提として、全プロセスのドメインの座標を全プロセスが持っている (領域分割時に全座標を放送する)。

5.3.4.6.1 開放境界、長距離力カットオフなし

他ドメインに対して自プロセスのツリーをたどり、他ドメインが力を計算するのに十分なEPJ、SPJをそれぞれ、ep_j_send_、sp_j_send_ に格納する。またその個数をn_ep_j_send_[]、n_sp_j_send_[] に格納する。配列のインデックスはプロセス番号と対応させること。まず、n_ep_j_send_[]、n_sp_j_send_[] を Alltoall し、受信した値をn_ep_j_recv_[]、n_sp_j_recv_[] に格納する。次に Alltoallv を使い ep_j_send_、sp_j_send_を送り、受信した値を ep_j_recv_、sp_j_recv_に格納する。

5.3.4.6.2 開放境界、長距離力カットオフあり

自ドメインと他ドメインの距離を測り、カットオフ半径より短い場合はローカルツリーをたどる。辿り方はほとんど”長距離力カットオフなし”の場合と同じだか、ツリーセルと他ドメインの距離がカットオフ長より長い場合はそのセルを辿る必要は無い。通信は粒子数を送る所は”長距離力カットオフなし”の場合と同じだが、粒子は全てのノードに送るわけではないので Alltoallv を使わずに Isend, Irecv で送る。

5.3.4.6.3 開放境界、短距離力散乱モード

他プロセスのドメインに対して自プロセスのツリーを辿る。この時ツリーの外側境界を使って、他プロセスのドメインとの距離を測る。通信パターンは”開放境界、長距離力カットオフあり”の場合と同じ。

5.3.4.6.4 開放境界、短距離力収集モード

最初に、ツリールートセルの外側境界の座標を Allgather する。他プロセスのツリールートセルの外側境界に対して自プロセスのツリーを辿る。この時ツリーの内側境界を使って、他プロセスのドメインとの距離を測る。

上記の方法だと、例えばプロセス内の粒子にカットオフ半径のばらつきがあった場合など、通信量が非常に増えてしまう場合も考えられる。そこで、以下の様な方法も考えられる。まず、”短距離力散乱モード”と同様の方法で、粒子を送る (Alltoall で粒子数を送り、Isend,Iredv で EPJ を送る)。次に送られて来た粒子のカットオフ半径内に入り、かつ送られ

て来た粒子の送信元プロセスのルートセルのドメインとの距離がそのカットオフ半径より長い粒子 (「送信元」プロセスには送られていない粒子) を ep_j_send_に格納する。

5.3.4.6.5　開放境界、短距離力対称モード

最初に、ツリールートセルの外側境界の座標を Allgather する。他プロセスのツリールートセルの外側境界に対して自プロセスのツリーを辿る。この時ツリーの外側境界を使って、他プロセスの外側境界との距離を測る。

収集モードの場合で述べた 2 段階通信の方法も考えられる。

5.3.4.6.6　開放境界、短距離力固定モード

短距離力散乱モードと同じ実装を行う。

5.3.4.6.7　直方体周期境界、長距離力カットオフあり

5.3.4.6.8　直方体周期境界、短距離力散乱モード

5.3.4.6.9　直方体周期境界、短距離力収集モード

5.3.4.6.10　直方体周期境界、短距離力対称モード

5.3.4.6.11　直方体周期境界、短距離力固定モード

5.3.4.7　グローバルツリーの作成

5.3.4.7.1　開放境界条件

ep_j_recv_,sp_j_recv_から tp_glb_を作る。この際 TP の adr_ptcl_は ep_j_recv_,sp_j_recv_の配列のインデックスを入れるが、EPJ と SPJ を区別するために SPJ の MSB は 1 にする。ツリー構築はローカルツリーと同様。最初はモートーンソートを作ってからのツリー構築を実装する。LET が少ない場合は挿入の方が速いかもしれない。

5.3.4.7.2　直方体周期境界条件

5.3.4.8　i 粒子グループ構築

PS ではいくつかの i 粒子群に対してツリーを辿るので、相互作用の計算の前に i 粒子グループの構築を行う。これは、ローカルツリーの構造を使うだけなので、ローカルツリー構築以降、相互作用計算の前ならどこでも行う事が出来るので、LET 交換にオーバーラップさせることも出来る。ただし、この計算コストは軽いのであまり意味がないかもしれない。

以下に i 粒子群を表すクラスを示す。

ソースコード 8: IPGroup

```
1 namespace ParticleSimulator{
2     template<int Tsmode>
3     class IPGroup{
4     public:
5         F32ort vertex_;
6         S32 adr_ptcl_;
7         S32 n_ptcl_;
8         template<class Ttc> void copyFromTC(const Ttc & tc);
9     };
```

vertex_はi粒子グループを囲む直方体の座標で、力の種類によって表現するものが異なる。力の種類はテンプレートパラメータとして与えられる。短距離力散乱モードでは内側境界、短距離力収集モードと短距離力対称モードと短距離力固定長モードでは外側境界を使う。長距離力では内側境界を使うと計算量がツリーセルがスパースなところでは計算量が減るが絶対必要というわけではないので、長距離量の場合の実装をどうするかは未定。adr_ptcl_はep_i_のインデクスで、n_ptcl_はそのi粒子群の個数である。ep_i_は順番に並んでいるのでインデックスと粒子数だけあれば計算可能である。

以下の関数により、i粒子グループを作る。

```
void makeIPGroup();
```

5.3.4.9 相互作用の計算

5.3.4.10 相互作用の書込

5.3.4.11 近傍粒子探査

SEARCH_MODEが$\mathrm{LONG}_S CATTER$、$\mathrm{LONG_CUTOFF_S}$CATTER、$\mathrm{LONG_S}$YMMETRY、$\mathrm{LONG_CUTOFF_S}$YMMETRYの場合には以下に述べる近傍粒子探査用の関数が使える。$\mathrm{LONG_S}$YMMETRY、$\mathrm{LONG_CUTOFF_S}$YMMETRY は未実装。

5.4 拡張モジュール詳細

5.4.1 Particle Mesh クラス

この節では、Particle Meshクラスの詳細記述を行う。Particle Meshクラスを以下のように記述した。

関数calcForceAllAndWriteBackはParticle Meshによる力を全粒子に対して計算し、その計算結果を書き込むことまで行う。その他の関数が必要となるのは、複数種類の粒子を扱う場合だけである。

ソースコード 9: ParticleMesh

```
1  namespace ParticleSimulator {
2      namespace ParticleMesh {
3          class ParticleMesh{
4          public:
5              template<class Tpsys,
6                       class Tdinfo>
7              void calcForceAllAndWriteBack(const Tpsys & psys,
8                                            const Tdinfo & dinfo
                                                  );
9              template<class Tdinfo>
10             void setDomainInfoParticleMesh(const Tdinfo & dinfo
                    );
11             template<class Tpsys>
12             void setParticleParticleMesh(const Tpsys & psys,
13                                           const bool clear=true
                                                 );
14             void calcMeshForceOnly();
15             F32vec getForce(F32vec pos);
16         }
17     namespace PM = ParticleMesh;
18 }
19 namespace PS = ParticleSimulator;
```

粒子クラスに粒子質量のゲッターとなるメンバ関数が必要。質量のメンバ変数が mass である場合は、以下のように書く。

```
void getChargeParticleMesh() {
    return this->mass;
}
```

5.4.1.1 全粒子への力を計算する関数

```
template<class Tpsys,
         class Tdinfo>
void PS::PM::calcForceAllAndWriteBack(const Tpsys & psys,
                                      const Tdinfo & dinfo);
```

上の関数で、全粒子への力を計算し、計算結果を粒子群クラスへ格納する。psys は力を計算すべき粒子群クラス、dinfo は領域クラスである。

計算結果を粒子群クラスへ格納するためのセッターが必要である。粒子クラスにメンバ関数 copyFromForceParticleMesh(const PS::F32vec & force) を設定する必要がある。もし、Particle Mesh による力のメンバ変数が PS::F32vec apm であるなら、以下のように記述する。

```
void copyFromForceParticleMesh(const PS::F32vec & force) {
    this->apm = force;
}
```

また、Particle--particle の力 (メンバ変数を acc とする) にそのまま足しこむ場合、以下のように記述する。

```
void copyFromForceParticleMesh(const PS::F32vec & force) {
    this->apc += force;
}
```

この関数を使わない場合は、このセッターを用意する必要はない。
この関数は、全プロセスで呼びだす必要がある。

5.4.1.2 領域クラスをセットする関数

```
template<class Tdinfo>
void PS::PM::setDomainInfoParticleMesh(const Tdinfo & dinfo);
```

上の関数で、このクラスに領域クラスの情報を与える。引数の dinfo は領域クラスである。
この関数は、全プロセスで呼びだす必要がある。

5.4.1.3 粒子情報をセットする関数

```
template<class Tpsys>
void PS::PM::setParticleParticleMesh(const Tpsys & psys,
                                     const bool clear=true);
```

上の関数で、このクラスに粒子群クラスの情報を与える。第 1 引数のは粒子群クラスである。第 2 引数はすでに与えられた粒子群クラスの情報をクリアするかどうかを決める。true のときはクリアし、false のときはクリアしない。
この関数は、全プロセスで呼びだす必要がある。

5.4.1.4 メッシュ上の力を計算する関数

```
void PS::PM::calcMeshForceOnly();
```

メッシュ上の力を計算する。
この関数は、全プロセスで呼び出す必要がある。

5.4.1.5 1つの粒子への力を計算する関数

```
PS::F32vec PS::PM::getForce(PS::F32vec pos);
```

1つの粒子への力を計算し、その計算結果を返す関数である。引数 pos は力を計算したい粒子の位置である。返す値は計算結果の力である。

5.4.1.6 Particle Mesh クラスの使い方

Particle Mesh クラスを使うには以下の4つのことを行う必要がある。

1. MPI と FFTW のインストール
2. Particle Mesh クラスのコンパイル
3. Particle Mesh クラスを使った FDPS コードの記述
4. FDPS コードのコンパイル

以下、詳細に記述する。

5.4.1.6.1 MPI と FFTW のインストール

ぐっどらっく

5.4.1.6.2 Particle Mesh クラスのコンパイル

以下のように行う。ディレクトリ src_parallel の下のディレクトリ particle_mesh の Makefile を適切に編集して make する。編集すべきことは以下の2点である。

- INCLUDE_FFTW に FFTW のヘッダファイルがあるディレクトリを記述する
- param_fdps.h の中の SIZE_OF_MESH（1次元方向のメッシュの数）を設定。推奨値は $N^{1/3}/2$（N は粒子数）。

うまく行けば、同じディレクトリにライブラリ libpm.a とヘッダファイル particle_mesh.hpp ができている。

5.4.1.6.3 FDPS コードを記述

以下のように行う。

- 上でできたヘッダファイルを include する
- PM を計算したい粒子クラスに以下のメソッドを加える
  - void copyFromForceParticleMesh(const PS::F32vec & force)。この中で force を好きなメンバ変数にセットする。
  - PS::F64 getChargeParticleMesh()。この中で質量を返す。
- このクラスのインスタンスを生成するときに、PS::PM::ParticleMesh とする

5.4.1.6.4 FDPS コードのコンパイル

上で記述した FDPS コードをコンパイルするには以下のことを行う必要がある。

- ヘッダファイル particle_mesh.hpp のあるディレクトリを記述すること
- ライブラリ libpm.a とのリンク
- FFTW のヘッダファイルがあるディレクトリを記述すること
- FFTW のライブラリとのリンク

6　テスト定義

この節では，最初の4節で4つのモジュール，すなわち通信用データクラス, 領域クラス，粒子群クラス，相互作用ツリークラスのテストについて記述する。様々なデータ構造が現れるが，断らないかぎり，そのデータ構造はそのクラスに属す。最後に結合テストについて記す。

6.1　通信用データクラス

6.2　領域クラス

6.3　粒子群クラス

6.4　リアル粒子の交換

交換前と交換後で粒子の個数が変わっていないかをチェック． 交換後に，プロセスが担当する粒子が適切かどうかチェック．

6.5　相互作用ツリークラス

6.5.1　ソートの単体テスト

ソースコード 10: radix sort のテスト

```
1  int main(int argc, char *argv[]){
2
3      if(argc < 3){
4          std::cerr<<"too␣few␣args.;␣arg1:␣problem␣size,␣arg2:␣
               repeat␣count"<<std::endl;
5          return 1;
6      }
7
8      PS::RadixSort<PS::U64, 8> RS;
9
10     PS::S32 n_size = std::atoi(argv[1]);
11     PS::S32 n_repeat = std::atoi(argv[2]);
12     PS::TreeParticle * data = new PS::TreeParticle[n_size];
13     PS::TreeParticle * data_buf = new PS::TreeParticle[n_size];
14     bool * flag = new bool[n_size];
15     PS::S32 err_comp = 0;
16     PS::S32 err_order = 0;
17
```

```
18      for(PS::S32 i=0; i<n_repeat; i++){
19          for(int j=0; j<n_size; j++){
20              data[j].setKey(PS::U64(abs(rand()))<<32 | PS::U64(
                    abs(rand())));
21              data[j].adr_ptcl_ = j;
22              flag[j] = false;
23          }
24          RS.lsdSort(data, data_buf, 0, n_size-1);
25          for(PS::S32 j=0; j<n_size; j++) flag[data[j].adr_ptcl_]
                = true;
26          for(PS::S32 j=0; j<n_size; j++){
27              if(flag[j] == false){
28                  err_comp++;
29              }
30          }
31          for(PS::S32 j=1; j<n_size; j++){
32              if(data[j].getKey() < data[j-1].getKey() ){
33                  err_order++;
34              }
35          }
36      }
37
38      if( err_comp || err_order){
39          std::cout<<"FAIL␣err_comp="<<err_comp<<"␣err_order="<<
                err_order<<std::endl;
40      }
41      else{
42          std::cout<<"PASS"<<std::endl;
43      }
44
45      return 0;
46  }
```

コマンドラインから第一引数が問題サイズ、第二引数が繰り返し回数。ソートされた配列を先頭から見ていき、常に次の値が現在の値より大きいかをチェックし、また粒子が消えたりしていないかもチェックする。

6.5.2 ローカルツリーのモートンソート

モートン順序に並べられたEPIから再びモートンキーを作り、モートンオーダーにならんでいるかをチェックする。また、モートンオーダーに並べたときに粒子が消えたりしていな

いかを TP の adr_ptc_を見ることでチェックする。このテストは以下の関数によって行われる。結果の出力は fout に行う。

```
void checkMortonSortLocalTreeOnly(std::ostream & fout = std::cout){
```

6.5.3 ローカルツリーの構築

ローカルツリー内の全てのセルについて、各セルの持つ粒子がそのセルの境界に入っているかをチェックする。セルの辿り方はツリーウォークと同じ方法で行う。このテストは以下の関数によって行われる。第一引数 tolerance は粒子がボックスから外れていた時に許容する大きさ。これは、ツリーの構築時に使ったセルの大きさの定義とこの関数内での定義が丸め誤差の範囲では違い、セル境界付近の粒子がボックスから出ていることがある為。結果の出力は fout に行う。

```
void checkMakeLocalTree(const F32 tolerance = 1e-6, std::ostream & fout = std::cout);
```

6.5.4 ローカルツリーのモーメント計算

モーメント計算のテストは以下の関数によって行う。サーチの種類により、行うテストは異なる。結果は fout に出力される。tolerance は許容する誤差である。

```
void checkCalcMomentLocalTree(const F32 tolerance = 1e-5, std::ostream & fout = std::c
```

6.5.4.1 長距離力、カットオフなし

各プロセスの持つ全粒子の質量、重心を直接求め、それをモーメント計算によって求めたツリーのルートセルのもつ質量、重心と比較する。tolerance は重心の質量、座標の差の許容誤差である。

6.5.4.2 長距離力、カットオフあり

6.5.4.3 短距離力

各プロセスの持つ全粒子の外側境界、内側境界を直接求め、それをモーメント計算によって求めたツリーのルートセルのもつ外側境界、内側境界と比較する。tolerance はそれぞれの方法で求めた境界の許容誤差である。

6.5.5 LET 交換

LET 交換のテストは以下の関数によって行う。サーチの種類により、行うテストは異なる。結果は fout に出力される。tolerance は許容する誤差で、長距離力の場合のみ必要である（短距離力の場合は無視される）。

```
void checkExchangeLocalEssentialTree(const DomainInfo & dinfo, const F32 tolerance = 1
```

6.5.5.1 長距離力、カットオフなし

自プロセスの EPJ と送られて来た EPJ、SPJ から、重心を計算し、その結果を全粒子の重心と比較する。全粒子の重心は、各プロセスで重心を計算し、Allreduce により求める。tolerance は重心の質量、座標の差の許容誤差である。

6.5.5.2 長距離力、カットオフあり

6.5.5.3 短距離力、全モード

送られて来た EPJ とツリーを使わずに通信した場合に送られてきた EPJ とを比較する。比較は粒子の x 座標を使ってソートし、二つの方法で送られて来た粒子の座標を比較する。

6.5.6 グローバルツリーのモートンソート

ローカルツリーのモートンソートのテストと同様に、モートン順序に並べられた EPI から再びモートンキーを作り、モートンオーダーにならんでいるかをチェックする。また、モートンオーダーに並べたときに粒子が消えたりしていないかを TP の adr_ptc_を見ることでチェックする。このテストは以下の関数によって行われる。結果の出力は fout に行う。

```
void checkMortonSortGlobalTreeOnly(std::ostream & fout = std::cout){
```

6.5.7 グローバルツリーの構築

ローカルツリーのテストと同様に、グローバルツリー内の全てのセルについて、各セルの持つ粒子がそのセルの境界に入っているかをチェックする。セルの辿り方はツリーウォークと同じ方法である。これは以下の関数で行う。

```
void checkMakeLocalTree(const F32 tolerance = 1e-6, std::ostream & fout = std::cout);
```

6.5.8 グローバルツリーのモーメント計算

モーメント計算のテストは以下の関数によって行う。サーチの種類により、行うテストは異なる。結果は fout に出力される。tolerance は許容する誤差である。

```
void checkCalcMomentGlobalTree(const F32 tolerance = 1e-5, std::ostream & fout = std::
```

6.5.8.1 長距離力、カットオフなし

各プロセスの持つ全粒子の質量、重心を直接求め、Allreduce し全体の重心質量、座標を求め、それをモーメント計算によって求めたツリーのルートセルのもつ質量、重心と比較する。tolerance は重心の質量、座標の差の許容誤差である。

6.5.8.2 長距離力、カットオフあり

6.5.8.3 短距離力

各プロセスの持つ全粒子の外側境界、内側境界を直接求め、それをモーメント計算によって求めたツリーのルートセルのもつ外側境界、内側境界と比較する。tolerance はそれぞれの方法で求めた境界の許容誤差である。

6.5.9 相互作用計算

FDPS で実際に求めた力と全粒子から直接求めた力を比較する。

```
template<class Tfunc_ep_ep, class Tfunc_compare>
void checkForce(Tfunc_ep_ep pfunc_ep_ep, Tfunc_compare func_compare,
                std::ostream & fout=std::cout);
```

ここで、pfunc_ep_ep は計算で使う EPI-EPJ 相互作用の関数、func_compare は比較用の関数で、ユーザーが関数ポインタもしくはファンクタによって与えることが出来る。結果は fout に出力する。構造体を返せた方が良い？

func_compare は以下の様に定義出来る。

ソースコード 11: 力の比較用ファンクタ

```
1 struct CompareGrav{
2     void operator () (ForceGrav * grav0, ForceGrav * grav1,
3                       const PS::S32 n, std::ostream & fout){
4         bool err = false;
5         for(PS::S32 i=0; i<n; i++){
6             PS::F64 dpot = std::abs( (grav0[i].pot - grav1[i].
                  pot) / grav0[i].pot );
```

```
7               PS::F64vec dacc_vec = grav0[i].acc - grav1[i].acc;
8               PS::F64 dacc = sqrt( (dacc_vec*dacc_vec) / (grav0[i
                    ].acc*grav0[i].acc) );
9               if( dpot > 1e-1 || dacc > 1e-1){
10                  fout<<"Compare␣Grav:␣FAIL"<<std::endl;
11                  fout<<"grav0[i].pot="<<grav0[i].pot<<"␣grav1[i
                        ].pot="<<grav1[i].pot<<std::endl;
12                  fout<<"grav0[i].acc="<<grav0[i].acc<<"␣grav1[i
                        ].acc="<<grav1[i].acc<<std::endl;
13                  err = true;
14              }
15          }
16          if(!err) fout<<"Compare␣Grav:␣PASS"<<std::endl;
17      }
18  };
```

[caption=]
また、ユーザーは以下の関数によって直接計算した力の値を得ることが出来る。

```
template<class Tfunc_ep_ep>
void calcForceDirect(Tfunc_ep_ep pfunc_ep_ep, Tforce force[], const bool clear=true);
```

force は直接計算した力を格納する配列でユーザーが配列を確保する。clear は force の値を初期化するかを決める。

これと FDPS で計算した力を返す関数を使う事で比較を行う事も出来る。

```
Tforce getForce(const S32 i);
```

以下に開放境界の場合の各関数のテストを記述する。

ソースコード 12: 開放境界、モートンソート、ローカルツリー構築、モーメント計算、LET 交換、グローバルツリー構築、相互作用計算のテスト

```
1  int main(int argc, char *argv[]){
2      std::cout<<std::setprecision(15);
3      std::cerr<<std::setprecision(15);
4      PS::Initialize(argc, argv);
5
6      char sinput[1024];
7      int c;
8      while((c=getopt(argc,argv,"i:h")) != -1){
9          switch(c){
10         case 'i':
11             sprintf(sinput,optarg);
```

```
12              break;
13          case 'h':
14              std::cerr<<"i: input file name (nemo ascii)"<<std::
                    endl;
15              return 0;
16          }
17      }
18
19      PS::ParticleSystem<FPGrav> system_grav;
20      system_grav.initialize();
21      PS::S32 n_grav_glb, n_grav_loc;
22      PS::F32 time_sys;
23      ReadNemoAscii(system_grav, n_grav_glb, n_grav_loc, time_sys
            , sinput);
24
25      PS::DomainInfo dinfo;
26      dinfo.initialize();
27      dinfo.setBoundaryCondition(PS::BOUNDARY_CONDITION_OPEN);
28      dinfo.collectSampleParticle(system_grav);
29      dinfo.decomposeDomain();
30
31      system_grav.exchangeParticle(dinfo);
32      n_grav_loc = system_grav.getNumberOfParticleLocal();
33
34  #if 1
35      PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav,
            MomentGrav, SPJ>::Normal tree_grav;
36
37      PS::F32 theta = 0.4;
38      tree_grav.initialize(n_grav_glb, theta);
39  #if 0
40
41  #else
42      //tree_grav.initializeLocalTree(half_len_grav_glb);
43      tree_grav.setParticleLocalTree(system_grav);
44
45      //tree_grav.mortonSortLocalTreeOnly(dinfo);
46      tree_grav.setRootCell(dinfo);
47      tree_grav.mortonSortLocalTreeOnly();
48      tree_grav.checkMortonSortLocalTreeOnly();
```

```
49
50      tree_grav.linkCellLocalTreeOnly();
51      tree_grav.checkMakeLocalTree();
52
53      tree_grav.calcMomentLocalTreeOnly();
54      tree_grav.checkCalcMomentLocalTree();
55
56      tree_grav.exchangeLocalEssentialTree(dinfo);
57      tree_grav.checkExchangeLocalEssentialTree(dinfo);
58
59      tree_grav.setLocalEssentialTreeToGlobalTree();
60
61      tree_grav.mortonSortGlobalTreeOnly();
62      tree_grav.checkMortonSortGlobalTreeOnly();
63
64      tree_grav.linkCellGlobalTreeOnly();
65      tree_grav.checkMakeGlobalTree();
66
67      tree_grav.calcMomentGlobalTreeOnly();
68      tree_grav.checkCalcMomentGlobalTree();
69
70      tree_grav.makeIPGroup();
71      tree_grav.checkMakeIPGroup();
72
73      PS::S32 n_ipg_grav = tree_grav.getNumberOfIPG();
74      bool err_grav = false;
75      for(PS::S32 i=0; i<n_ipg_grav; i++){
76          tree_grav.makeInteractionList(i, err_grav);
77          //tree_grav.checkMakeInteractionList();
78          tree_grav.calcForceOnly(CalcForceEpEp(), CalcForceSpEp
                ), i);
79      }
80      tree_grav.copyForceOriginalOrder();
81      tree_grav.checkForce( CalcForceEpEp(), CompareGrav(), dinfo
             );
82
83  #endif
84  #endif
85
86
```

```
87      ///////////////
88      //// SHORT ////
89      PS::ParticleSystem<FPSPH> system_sph;
90      system_sph.initialize();
91      PS::S32 n_sph_glb, n_sph_loc;
92      ReadNemoAscii(system_sph, n_sph_glb, n_sph_loc, time_sys,
            sinput);
93      PS::F64 half_len_sph_glb = system_sph.getHalfLength();
94      std::cout<<"half_len_sph_glb="<<half_len_sph_glb<<std::endl
            ;
95
96      for(PS::S32 i=0; i<n_sph_loc; i++){
97          system_sph[i].r_search = pow( (n_sph_glb/(
                half_len_sph_glb*half_len_sph_glb*
                half_len_sph_glb)), -0.333333) * (system_sph[i].
                id%10)*0.2;
98      }
99
100     system_sph.exchangeParticle(dinfo);
101     n_sph_loc = system_sph.getNumberOfParticleLocal();
102
103 #if 0
104     ////////////////////////
105     //// SCATTER MODE ////
106     PS::TreeForForceShort<ResultDens, EPIScatter, EPJScatter>::
            Scatter tree_scatter;
107     tree_scatter.initialize(n_sph_glb);
108 #if 0
109     tree_scatter.calcForceAllAndWriteBack(CalcDensityScatter(),
             system_sph, dinfo);
110     tree_scatter.checkForce( CalcDensityScatter(),
            CompareDensity(), dinfo);
111 #else
112
113     tree_scatter.setParticleLocalTree(system_sph);
114
115     tree_scatter.setRootCell(dinfo);
116     tree_scatter.mortonSortLocalTreeOnly();
117     tree_scatter.checkMortonSortLocalTreeOnly();
118
```

```
119     tree_scatter.linkCellLocalTreeOnly();
120     tree_scatter.checkMakeLocalTree();
121
122     tree_scatter.calcMomentLocalTreeOnly();
123     tree_scatter.checkCalcMomentLocalTree();
124
125     tree_scatter.exchangeLocalEssentialTree(dinfo);
126     tree_scatter.checkExchangeLocalEssentialTree(dinfo);
127
128     tree_scatter.setLocalEssentialTreeToGlobalTree();
129
130     tree_scatter.mortonSortGlobalTreeOnly();
131     tree_scatter.checkMortonSortGlobalTreeOnly();
132
133     tree_scatter.linkCellGlobalTreeOnly();
134     tree_scatter.checkMakeGlobalTree();
135
136     tree_scatter.calcMomentGlobalTreeOnly();
137     tree_scatter.checkCalcMomentGlobalTree();
138
139     tree_scatter.makeIPGroup();
140     tree_scatter.checkMakeIPGroup();
141
142
143     PS::S32 n_ipg_sph = tree_scatter.getNumberOfIPG();
144     bool err_sph = false;
145     for(PS::S32 i=0; i<n_ipg_sph; i++){
146         tree_scatter.makeInteractionList(i, err_sph);
147         tree_scatter.calcForceOnly( CalcDensityScatter(), i);
148     }
149     tree_scatter.copyForceOriginalOrder();
150
151     tree_scatter.checkForce( CalcDensityScatter(),
          CompareDensity(), dinfo);
152
153 #endif
154 #endif
155
156 #if 0
157     ////////////////////
```

```
158     //// GATHER MODE ////
159     PS::TreeForForceShort<ResultDens, EPIGather, EPJGather>::
            Gather tree_gather;
160     tree_gather.initialize(n_sph_glb);
161 #if 0
162     tree_gather.calcForceAllAndWriteBack(CalcDensityGather(),
            system_sph, dinfo);
163     tree_gather.checkForce( CalcDensityGather(), CompareDensity
            (), dinfo);
164 #else
165
166     tree_gather.setParticleLocalTree(system_sph);
167
168     tree_gather.setRootCell(dinfo);
169     tree_gather.mortonSortLocalTreeOnly();
170     tree_gather.checkMortonSortLocalTreeOnly();
171
172     tree_gather.linkCellLocalTreeOnly();
173     tree_gather.checkMakeLocalTree();
174
175     tree_gather.calcMomentLocalTreeOnly();
176     tree_gather.checkCalcMomentLocalTree();
177
178     tree_gather.exchangeLocalEssentialTree(dinfo);
179     tree_gather.checkExchangeLocalEssentialTree(dinfo);
180
181     tree_gather.setLocalEssentialTreeToGlobalTree();
182
183     tree_gather.mortonSortGlobalTreeOnly();
184     tree_gather.checkMortonSortGlobalTreeOnly();
185
186     tree_gather.linkCellGlobalTreeOnly();
187     tree_gather.checkMakeGlobalTree();
188
189     tree_gather.calcMomentGlobalTreeOnly();
190     tree_gather.checkCalcMomentGlobalTree();
191
192     tree_gather.makeIPGroup();
193     tree_gather.checkMakeIPGroup();
194
```

```
195     PS::S32 n_ipg_sph_gather = tree_gather.getNumberOfIPG();
196     bool err_sph_gather = false;
197     for(PS::S32 i=0; i<n_ipg_sph_gather; i++){
198         tree_gather.makeInteractionList(i, err_sph_gather);
199         //tree_gather.checkMakeInteractionList(i);
200         tree_gather.calcForceOnly( CalcDensityGather(), i);
201     }
202     tree_gather.copyForceOriginalOrder();
203     tree_gather.checkForce( CalcDensityGather(), CompareDensity
            (), dinfo);
204
205 #endif
206 #endif
207
208 #if 0
209     ////////////////////////
210     //// SYMMETRY MODE ////
211     PS::TreeForForceShort<ResultDens, EPISymmetry, EPJSymmetry
            >::Symmetry tree_symmetry;
212     tree_symmetry.initialize(n_sph_glb);
213 #if 0
214     tree_symmetry.calcForceAllAndWriteBack(CalcDensitySymmetry
            (), system_sph, dinfo);
215     tree_symmetry.checkForce( CalcDensitySymmetry(),
            CompareDensity(), dinfo);
216 #else
217
218     tree_symmetry.setParticleLocalTree(system_sph);
219
220     tree_symmetry.setRootCell(dinfo);
221     tree_symmetry.mortonSortLocalTreeOnly();
222     tree_symmetry.checkMortonSortLocalTreeOnly();
223
224     tree_symmetry.linkCellLocalTreeOnly();
225     tree_symmetry.checkMakeLocalTree();
226
227     tree_symmetry.calcMomentLocalTreeOnly();
228     tree_symmetry.checkCalcMomentLocalTree();
229
230     tree_symmetry.exchangeLocalEssentialTree(dinfo);
```

```
231     tree_symmetry.checkExchangeLocalEssentialTree(dinfo);
232
233     tree_symmetry.setLocalEssentialTreeToGlobalTree();
234
235     tree_symmetry.mortonSortGlobalTreeOnly();
236     tree_symmetry.checkMortonSortGlobalTreeOnly();
237
238     tree_symmetry.linkCellGlobalTreeOnly();
239     tree_symmetry.checkMakeGlobalTree();
240
241
242     tree_symmetry.calcMomentGlobalTreeOnly();
243     tree_symmetry.checkCalcMomentGlobalTree();
244
245
246     tree_symmetry.makeIPGroup();
247     tree_symmetry.checkMakeIPGroup();
248
249     PS::S32 n_ipg_sph_symmetry = tree_symmetry.getNumberOfIPG
            ();
250     bool err_sph_symmetry = false;
251     for(PS::S32 i=0; i<n_ipg_sph_symmetry; i++){
252         tree_symmetry.makeInteractionList(i, err_sph_symmetry);
253         //tree_symmetry.checkMakeInteractionList(i);
254         tree_symmetry.calcForceOnly( CalcDensitySymmetry(), i);
255     }
256     tree_symmetry.copyForceOriginalOrder();
257     tree_symmetry.checkForce( CalcDensitySymmetry(),
            CompareDensity(), dinfo);
258
259 #endif
260 #endif
261
262     PS::Finalize();
263     return 0;
264 }
```

以下に周期境界の場合の各関数のテストを記述する。

ソースコード 13: 開放境界、モートンソート、ローカルツリー構築、モーメント計算、LET 交換、グローバルツリー構築、相互作用計算のテスト

```
1
2 int main(int argc, char *argv[]){
3     std::cout<<std::setprecision(15);
4     std::cerr<<std::setprecision(15);
5     PS::Initialize(argc, argv);
6
7     PS::S32 my_rank = PS::Comm::getRank();
8     //PS::S32 n_proc = PS::Comm::getNumberOfProc();
9
10    char sinput[1024];
11    int c;
12    while((c=getopt(argc,argv,"i:h")) != -1){
13        switch(c){
14        case 'i':
15            sprintf(sinput,optarg);
16            break;
17        case 'h':
18            std::cerr<<"i: input file name (nemo ascii)"<<std::
                  endl;
19            return 0;
20        }
21    }
22
23    ///////////////
24    //// SHORT ////
25    PS::ParticleSystem<FPSPH> system_sph;
26    system_sph.initialize();
27    PS::S32 n_sph_glb, n_sph_loc;
28    PS::F32 time_sys;
29    ReadNemoAscii(system_sph, n_sph_glb, n_sph_loc, time_sys,
          sinput);
30
31    PS::DomainInfo dinfo;
32    dinfo.initialize();
33    dinfo.setNumberOfDomainMultiDimension(PS::Comm::
          getNumberOfProc(), 1, 1);
34    dinfo.setBoundaryCondition(PS::
          BOUNDARY_CONDITION_PERIODIC_X);
35    PS::F32vec low_pos_domain(-50.0, 0.0, 0.0);
36    PS::F32vec high_pos_domain(50.0, 0.0, 0.0);
```

```
37      dinfo.setPosRootDomain(low_pos_domain, high_pos_domain);
38      dinfo.collectSampleParticle(system_sph);
39      dinfo.decomposeDomain();
40
41      bool pa[3];
42      dinfo.getPeriodicAxis(pa);
43      PS::F32ort pos_root_domain = dinfo.getPosRootDomain();
44      PS::F32ort pos_my_domain = dinfo.getPosDomain(my_rank);
45      PS::F64 half_len_sph_glb = system_sph.getHalfLength();
46      if(PS::Comm::getRank() == 0){
47          std::cout<<"pa[0]="<<pa[0]<<"␣pa[1]="<<pa[1]<<"␣pa[2]="
                <<pa[2]<<std::endl;
48          std::cout<<"pos_root_domain="<<pos_root_domain<<std::
                endl;
49          std::cout<<"half_len_sph_glb="<<half_len_sph_glb<<std::
                endl;
50      }
51
52
53      for(PS::S32 i=0; i<n_sph_loc; i++){
54          system_sph[i].r_search = pow( (n_sph_glb/(
                half_len_sph_glb*half_len_sph_glb*
                half_len_sph_glb)), -0.333333) * (system_sph[i].
                id%10)*0.2*10 * PS::MT::genrand_real2() * 2.0 *
                0.01;
55      }
56
57      system_sph.exchangeParticle(dinfo);
58
59      n_sph_loc = system_sph.getNumberOfParticleLocal();
60
61  #if 1
62      PS::TreeForForceShort<ResultDens, EPIScatter, EPJScatter>::
            Scatter tree_scatter;
63      tree_scatter.initialize(n_sph_glb);
64
65  #if 0
66      tree_scatter.calcForceAllAndWriteBack(CalcDensityScatter(),
            system_sph, dinfo);
67      tree_scatter.checkForce( CalcDensityScatter(),
```

```
            CompareDensity(), dinfo);
#else

    tree_scatter.setParticleLocalTree(system_sph);

    tree_scatter.setRootCell(dinfo);

    tree_scatter.mortonSortLocalTreeOnly();
    tree_scatter.checkMortonSortLocalTreeOnly();

    tree_scatter.linkCellLocalTreeOnly();
    tree_scatter.checkMakeLocalTree();

    tree_scatter.calcMomentLocalTreeOnly();
    tree_scatter.checkCalcMomentLocalTree();

    tree_scatter.exchangeLocalEssentialTree(dinfo);
    tree_scatter.checkExchangeLocalEssentialTree(dinfo);

    tree_scatter.setLocalEssentialTreeToGlobalTree();

    tree_scatter.mortonSortGlobalTreeOnly();
    tree_scatter.checkMortonSortGlobalTreeOnly();

    tree_scatter.linkCellGlobalTreeOnly();
    tree_scatter.checkMakeGlobalTree();

    tree_scatter.calcMomentGlobalTreeOnly();
    tree_scatter.checkCalcMomentGlobalTree();

    tree_scatter.makeIPGroup();
    tree_scatter.checkMakeIPGroup();

    PS::S32 n_ipg_sph_scatter = tree_scatter.getNumberOfIPG();
    bool err_sph_scatter = false;
    for(PS::S32 i=0; i<n_ipg_sph_scatter; i++){
        tree_scatter.makeInteractionList(i, err_sph_scatter);
        //tree_scatter.checkMakeInteractionList(dinfo, i);
        tree_scatter.calcForceOnly( CalcDensityScatter(), i);
    }
```

```
107     tree_scatter.copyForceOriginalOrder();
108     tree_scatter.checkForce( CalcDensityScatter(),
            CompareDensity(), dinfo);
109
110 #endif
111 #endif
112
113 #if 0
114     PS::TreeForForceShort<ResultDens, EPIGather, EPJGather>::
            Gather tree_gather;
115     tree_gather.initialize(n_sph_glb);
116 #if 0
117     tree_gather.calcForceAllAndWriteBack(CalcDensityGather(),
            system_sph, dinfo);
118     tree_gather.checkForce( CalcDensityGather(), CompareDensity
            (), dinfo);
119 #else
120     tree_gather.setParticleLocalTree(system_sph);
121
122     tree_gather.setRootCell(dinfo);
123     tree_gather.mortonSortLocalTreeOnly();
124     tree_gather.checkMortonSortLocalTreeOnly();
125
126
127     tree_gather.linkCellLocalTreeOnly();
128     tree_gather.checkMakeLocalTree();
129
130     tree_gather.calcMomentLocalTreeOnly();
131     tree_gather.checkCalcMomentLocalTree();
132
133     tree_gather.exchangeLocalEssentialTree(dinfo);
134     tree_gather.checkExchangeLocalEssentialTree(dinfo);
135
136     tree_gather.setLocalEssentialTreeToGlobalTree();
137
138     tree_gather.mortonSortGlobalTreeOnly();
139     tree_gather.checkMortonSortGlobalTreeOnly();
140
141     tree_gather.linkCellGlobalTreeOnly();
142     tree_gather.checkMakeGlobalTree();
```

```
143
144     tree_gather.calcMomentGlobalTreeOnly();
145     tree_gather.checkCalcMomentGlobalTree();
146
147     tree_gather.makeIPGroup();
148     tree_gather.checkMakeIPGroup();
149
150     PS::S32 n_ipg_sph_gather = tree_gather.getNumberOfIPG();
151     bool err_sph_gather = false;
152     for(PS::S32 i=0; i<n_ipg_sph_gather; i++){
153         tree_gather.makeInteractionList(i, err_sph_gather);
154         //tree_gather.checkMakeInteractionList(dinfo, i);
155         tree_gather.calcForceOnly( CalcDensityGather(), i);
156     }
157     tree_gather.copyForceOriginalOrder();
158     tree_gather.checkForce( CalcDensityGather(), CompareDensity
          (), dinfo);
159
160 #endif
161 #endif
162
163 #if 0
164     PS::TreeForForceShort<ResultDens, EPISymmetry, EPJSymmetry
          >::Symmetry tree_symmetry;
165     tree_symmetry.initialize(n_sph_glb);
166
167 #if 0
168     tree_symmetry.calcForceAllAndWriteBack(CalcDensitySymmetry
          (), system_sph, dinfo);
169     tree_symmetry.checkForce( CalcDensitySymmetry(),
          CompareDensity(), dinfo);
170 #else
171     tree_symmetry.setParticleLocalTree(system_sph);
172
173     tree_symmetry.setRootCell(dinfo);
174     tree_symmetry.mortonSortLocalTreeOnly();
175     tree_symmetry.checkMortonSortLocalTreeOnly();
176
177     tree_symmetry.linkCellLocalTreeOnly();
178     tree_symmetry.checkMakeLocalTree();
```

```
179
180     tree_symmetry.calcMomentLocalTreeOnly();
181     tree_symmetry.checkCalcMomentLocalTree();
182
183     tree_symmetry.exchangeLocalEssentialTree(dinfo);
184     tree_symmetry.checkExchangeLocalEssentialTree(dinfo, 1e-4);
185
186
187     tree_symmetry.setLocalEssentialTreeToGlobalTree();
188
189     tree_symmetry.mortonSortGlobalTreeOnly();
190     tree_symmetry.checkMortonSortGlobalTreeOnly();
191
192     tree_symmetry.linkCellGlobalTreeOnly();
193     tree_symmetry.checkMakeGlobalTree();
194
195     tree_symmetry.calcMomentGlobalTreeOnly();
196     tree_symmetry.checkCalcMomentGlobalTree();
197
198     tree_symmetry.makeIPGroup();
199     tree_symmetry.checkMakeIPGroup();
200
201
202     PS::S32 n_ipg_sph_symmetry = tree_symmetry.getNumberOfIPG
            ();
203     bool err_sph_symmetry = false;
204     for(PS::S32 i=0; i<n_ipg_sph_symmetry; i++){
205         tree_symmetry.makeInteractionList(i, err_sph_symmetry);
206         //tree_symmetry.checkMakeInteractionList(i);
207         tree_symmetry.calcForceOnly( CalcDensitySymmetry(), i);
208     }
209
210     tree_symmetry.copyForceOriginalOrder();
211     tree_symmetry.checkForce( CalcDensitySymmetry(),
            CompareDensity(), dinfo);
212 #endif
213 #endif
214     PS::Finalize();
215     return 0;
216
```

```
217 }
```

6.6 結合テスト

6.6.1 Tree-PM

宇宙論 N 体シミュレーションの 1 スナップショットの加速度を計算する． FDPS で作ったコードと GreeM を比較する． PM パートは GreeM から借用した． FFT は fftw-3.3.4 を使った．

条件設定

$N = 32^3$ の粒子を $\mathbf{0} \leq \boldsymbol{x} < \mathbf{1}$ の箱にランダムに分布させる． $\theta = 0.5$（比較のために $\theta = 0.0$ も），$N_{\rm leaf} = 10$，$N_{\rm crit} = 300$． $\Omega_0 = 0.3$． $\epsilon = 2.5 \times 10^{-4}$． メッシュ長 $1/16 (= 2/N^{1/3})$． カットオフ半径 $3/16$． プロセス数 $2 \times 2 \times 2$． 各プロセスで粒子配列の順番が FDPS と GreeM で全く同じになるようにした．

結果

PM force は GreeM と FDPS で完全に一致した．

図??は PP force の結果 ($\theta = 0.0$)． GreeM と FDPS の相対誤差の最大は 0.1%を越えるくらい． これは，Phantom-GRAPE の誤差によるもの． Tanikawa et al. (2012, NewA, 19, 74) の fig. 8 を見るとわかるが，力が最も大きいところで最も相対誤差が大きく 0.1%程度 (ここで使っている E と F はこの論文のものと同じ)．

図??はツリーの検証． GreeM と FDPS の $\theta = 0.5$ の相対誤差分布はほぼ同じ． GreeM と FDPS の $\theta = 0.5$ 同士の差の方が小さい．

図??はツリーの検証． GreeM と FDPS の $\theta = 1.0$ の相対誤差分布はほぼ同じ． GreeM と FDPS の $\theta = 1.0$ 同士の差の方が小さい．

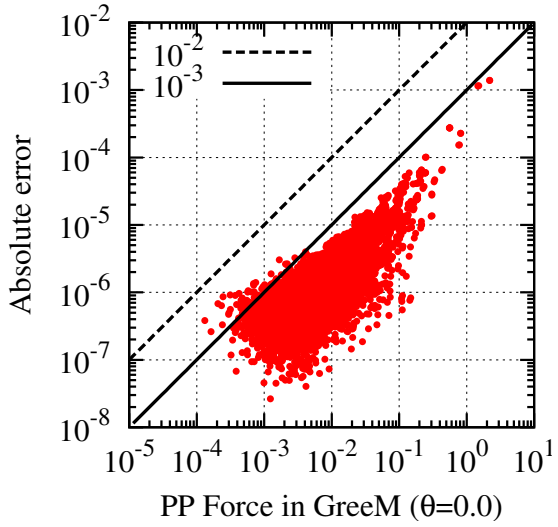

図 5: GreeM の PP force の絶対値 (横軸) と, GreeM と FDPS の PP force の絶対誤差 (縦軸). ツリーの精度は $\theta = 0.0$.

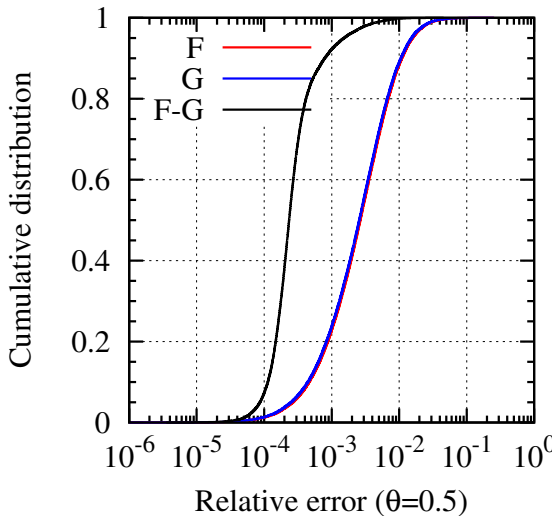

図 6: (赤)FDPS の PP force の相対誤差 ($\theta = 0.0$ と 0.5 を比較). (青)GreeM の PP force の相対誤差 ($\theta = 0.0$ と 0.5 を比較). (黒)FDPS と GreeM の PP force の相対誤差 (どちらも $\theta = 0.5$).

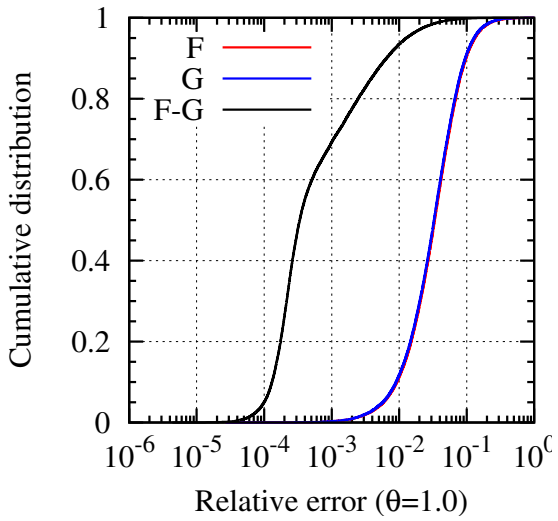

図 7: (赤)FDPS の PP force の相対誤差 ($\theta = 0.0$ と 1.0 を比較). (青)GreeM の PP force の相対誤差 ($\theta = 0.0$ と 1.0 を比較). (黒)FDPS と GreeM の PP force の相対誤差 (どちらも $\theta = 1.0$).

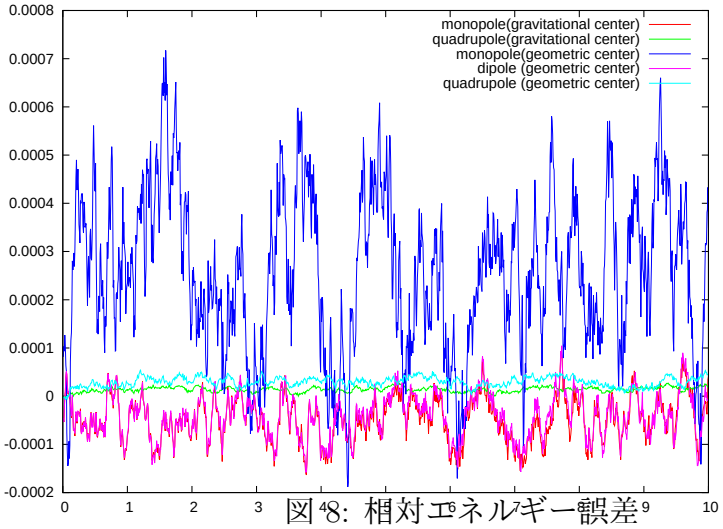

図 8: 相対エネルギー誤差

7 サンプル

7.1 重力 N 体計算

7.1.1 計算 1

以下は様々な多重局展開を使って計算を行った場合の結果である。

初期条件

N=16384、プラマーモデル、非等質量。質量の範囲は一番重い粒子の質量が一番軽いものの 3 倍。計算機は core-i5。並列化は 4 プロセス*2 スレッド。モーメントの計算を、重心展開の単極子と四重極子、さらに幾何中心展開の単極子、双極子、四重極子まで使った、5 つの計算を行った。ツリーのオープニングクライテリオンは 0.5。最大の i 粒子グループ数は 64。最大のリーフ粒子数は 8。PhantomGRAPE は使用していない。

結果

7.1.2 計算 2

以下は本サンプルコードの計算時間である。

初期条件

N=8M(M=2^{20}) のプラマーモデル。粒子はすべて等質量。計算機は京。並列化は 64-2048 プロセス*8 スレッドの。モーメントの計算は、重心展開の単極子を使った。ツリーのオープニングクライテリオンは 0.5。最大の i 粒子グループ数は 64。最大のリーフ粒子数は 8。PhantomGRAPE は使用していない。

結果

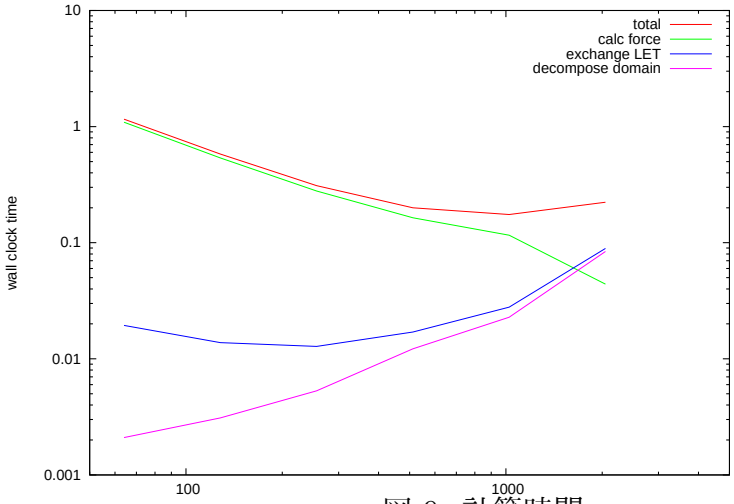

図 9: 計算時間

7.1.3 計算 3

以下は本サンプルコードの計算時間である。

初期条件

N=64M(M=2^{20}) のプラマーモデル。粒子はすべて等質量。計算機は京。並列化は 512-2048 プロセス*8 スレッドの。モーメントの計算は、重心展開の単極子を使った。ツリーのオープニングクライテリオンは 0.5。最大の i 粒子グループ数は 256。最大のリーフ粒子数は 8。PhantomGRAPE は使用していない。

結果

7.1.4 使用コード

ソースコード 14: 重力 N 体計算

```
1  #include<iostream>
2  #include<fstream>
3  #include<unistd.h>
4  #include<particle_simulator.hpp>
5
6  #ifdef USEPHANTOMGRAPE
7  #include"phantomgrape.hpp"
8  #endif
9
10 class ForceGrav{
11 public:
12     PS::F64vec acc;
```

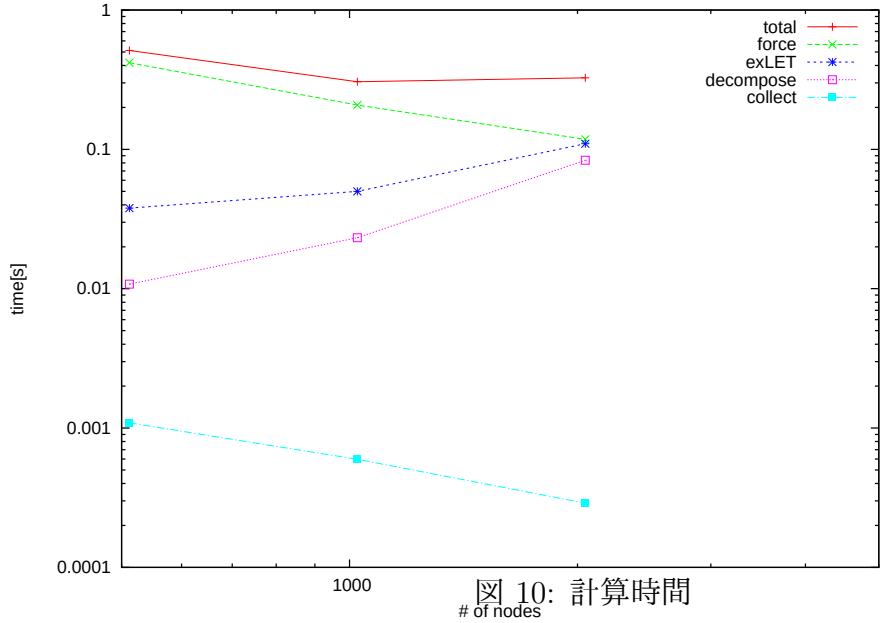

図 10: 計算時間

```
13      PS::F64 pot;
14      void clear(){
15          acc = 0.0;
16          pot = 0.0;
17      }
18  };
19
20  class FPGrav{
21  public:
22      PS::S64 id;
23      PS::F64 mass;
24      PS::F64vec pos;
25      PS::F64vec vel;
26      PS::F64vec acc;
27      PS::F64 pot;
28      PS::F64vec getPos() const { return pos; }
29      void copyFromForce(const ForceGrav & force){
30          acc = force.acc;
31          pot = force.pot;
32      }
33  };
34
```

```
35 class EPIGrav{
36 public:
37     PS::S64 id;
38     PS::F64vec pos;
39     static PS::F64 eps;
40     PS::F64vec getPos() const { return pos;}
41     void copyFromFP(const FPGrav & fp){
42         pos = fp.pos;
43         id = fp.id;
44     }
45 };
46
47 PS::F64 EPIGrav::eps = 1.0/32.0;
48
49 class EPJGrav{
50 public:
51     PS::S64 id;
52     PS::F64 mass;
53     PS::F64vec pos;
54     void copyFromFP(const FPGrav & fp){
55         mass = fp.mass;
56         pos = fp.pos;
57         id = fp.id;
58     }
59     PS::F64vec getPos() const { return pos; }
60     void setPos(const PS::F64vec & pos_new){ pos = pos_new;}
61     PS::F64 getCharge() const { return mass; }
62 };
63
64 #ifdef USEPHANTOMGRAPE
65
66 struct CalcForceEpEp{
67     void operator () (const EPIGrav * ep_i,
68                       const PS::S32 n_ip,
69                       const EPJGrav * ep_j,
70                       const PS::S32 n_jp,
71                       ForceGrav * force){
72         double xi[PhantomGRAPE::PG_NIMAX][3];
73         double mxj[PhantomGRAPE::PG_NJMAX][4];
74         double ai[PhantomGRAPE::PG_NIMAX][3];
```

```
75          double pi[PhantomGRAPE::PG_NIMAX];
76          const PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
77          const PS::F64 pot_corr = 1.0 / sqrt(eps2) * ep_j[0].
                getCharge(); // potential correction
78          for(PS::S32 i=0; i<n_ip; i++){
79              xi[i][0] = ep_i[i].getPos().x;
80              xi[i][1] = ep_i[i].getPos().y;
81              xi[i][2] = ep_i[i].getPos().z;
82
83          }
84          for(PS::S32 j=0; j<n_jp; j++){
85              mxj[j][0] = ep_j[j].getPos().x;
86              mxj[j][1] = ep_j[j].getPos().y;
87              mxj[j][2] = ep_j[j].getPos().z;
88              mxj[j][3] = ep_j[j].getCharge();
89          }
90          PhantomGRAPE pg;
91          pg.set_eps2(eps2);
92          pg.set_xj(n_jp, mxj);
93          pg.set_xi(n_ip, xi);
94          pg.run(n_ip, n_jp);
95          pg.get_ai(n_ip, ai, pi);
96
97          for(PS::S32 i=0; i<n_ip; i++){
98              force[i].acc.x += ai[i][0];
99              force[i].acc.y += ai[i][1];
100             force[i].acc.z += ai[i][2];
101             force[i].pot += pi[i] + pot_corr;
102         }
103     }
104 };
105
106 #else
107 struct CalcForceEpEp{
108     void operator () (const EPIGrav * ep_i,
109                       const PS::S32 n_ip,
110                       const EPJGrav * ep_j,
111                       const PS::S32 n_jp,
112                       ForceGrav * force){
113
```

```
114         PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
115         for(PS::S32 i=0; i<n_ip; i++){
116             PS::F64vec xi = ep_i[i].pos;
117             PS::F64vec ai = 0.0;
118             PS::F64 poti = 0.0;
119             PS::S64 idi = ep_i[i].id;
120             for(PS::S32 j=0; j<n_jp; j++){
121                 if( idi == ep_j[j].id ) continue;
122                 PS::F64vec rij = xi - ep_j[j].pos;
123                 PS::F64 r3_inv = rij * rij + eps2;
124                 PS::F64 r_inv = 1.0/sqrt(r3_inv);
125                 r3_inv = r_inv * r_inv;
126                 r_inv *= ep_j[j].mass;
127                 r3_inv *= r_inv;
128                 ai -= r3_inv * rij;
129                 poti -= r_inv;
130             }
131             force[i].acc += ai;
132             force[i].pot += poti;
133         }
134     }
135 };
136 #endif // USEPHANTOMGRAPE
137
138 #ifdef MONOPOLE
139
140 #ifdef USEPHANTOMGRAPE
141 struct CalcForceSpEp{
142     void operator () (const EPIGrav * ep_i,
143                       const PS::S32 n_ip,
144                       const PS::SPJMonopole * sp_j,
145                       const PS::S32 n_jp,
146                       ForceGrav * force){
147         double xi[PhantomGRAPE::PG_NIMAX][3];
148         double mxj[PhantomGRAPE::PG_NJMAX][4];
149         double ai[PhantomGRAPE::PG_NIMAX][3];
150         double pi[PhantomGRAPE::PG_NIMAX];
151         const PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
152         for(PS::S32 i=0; i<n_ip; i++){
153             xi[i][0] = ep_i[i].pos.x;
```

```
154             xi[i][1] = ep_i[i].pos.y;
155             xi[i][2] = ep_i[i].pos.z;
156         }
157         for(PS::S32 j=0; j<n_jp; j++){
158             mxj[j][0] = sp_j[j].getPos().x;
159             mxj[j][1] = sp_j[j].getPos().y;
160             mxj[j][2] = sp_j[j].getPos().z;
161             mxj[j][3] = sp_j[j].getCharge();
162         }
163         PhantomGRAPE pg;
164         pg.set_eps2(eps2);
165         pg.set_xj(n_jp, mxj);
166         pg.set_xi(n_ip, xi);
167         pg.run(n_ip, n_jp);
168         pg.get_ai(n_ip, ai, pi);
169 
170         for(PS::S32 i=0; i<n_ip; i++){
171             force[i].acc.x += ai[i][0];
172             force[i].acc.y += ai[i][1];
173             force[i].acc.z += ai[i][2];
174             force[i].pot += pi[i];
175         }
176 
177     }
178 };
179 #else
180 struct CalcForceSpEp{
181     void operator () (const EPIGrav * ep_i,
182                       const PS::S32 n_ip,
183                       const PS::SPJMonopole * sp_j,
184                       const PS::S32 n_jp,
185                       ForceGrav * force){
186         PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
187         for(PS::S32 i=0; i<n_ip; i++){
188             PS::F64vec xi = ep_i[i].pos;
189             PS::F64vec ai = 0.0;
190             PS::F64 poti = 0.0;
191             for(PS::S32 j=0; j<n_jp; j++){
192                 PS::F64vec rij = xi - sp_j[j].pos;
193                 PS::F64 r3_inv = rij * rij + eps2;
```

```
194                 PS::F64 r_inv = 1.0/sqrt(r3_inv);
195                 r3_inv = r_inv * r_inv;
196                 r_inv *= sp_j[j].mass;
197                 r3_inv *= r_inv;
198                 ai -= r3_inv * rij;
199                 poti -= r_inv;
200             }
201             force[i].acc += ai;
202             force[i].pot += poti;
203         }
204     }
205 };
206 #endif // #ifdef USEPHANTOMGRAPE
207
208 #elif QUADRUPOLE
209 struct CalcForceSpEp{
210     void operator () (const EPIGrav * ep_i,
211                       const PS::S32 n_ip,
212                       const PS::SPJQuadrupole * sp_j,
213                       const PS::S32 n_jp,
214                       ForceGrav * force){
215         PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
216         for(PS::S32 ip=0; ip<n_ip; ip++){
217             PS::F64vec xi = ep_i[ip].pos;
218             PS::F64vec ai = 0.0;
219             PS::F64 poti = 0.0;
220             for(PS::S32 jp=0; jp<n_jp; jp++){
221                 PS::F64 mj = sp_j[jp].mass;
222                 PS::F64vec xj= sp_j[jp].pos;
223                 PS::F64vec rij= xi - xj;
224                 PS::F64 r2 = rij * rij + eps2;
225                 PS::F64mat qj = sp_j[jp].quad;
226                 PS::F64 tr = qj.getTrace();
227                 PS::F64vec qr( (qj.xx*rij.x + qj.xy*rij.y + qj.
                        xz*rij.z),
228                                (qj.yy*rij.y + qj.yz*rij.z + qj.
                                    xy*rij.x),
229                                (qj.zz*rij.z + qj.xz*rij.x + qj.
                                    yz*rij.y) );
230                 PS::F64 qrr = qr * rij;
```

```
231                 PS::F64 r_inv = 1.0f/sqrt(r2);
232                 PS::F64 r2_inv = r_inv * r_inv;
233                 PS::F64 r3_inv = r2_inv * r_inv;
234                 PS::F64 r5_inv = r2_inv * r3_inv * 1.5;
235                 PS::F64 qrr_r5 = r5_inv * qrr;
236                 PS::F64 qrr_r7 = r2_inv * qrr_r5;
237                 PS::F64 A = mj*r3_inv - tr*r5_inv + 5*qrr_r7;
238                 PS::F64 B = -2.0*r5_inv;
239                 ai -= A*rij + B*qr;
240                 poti -= mj*r_inv - 0.5*tr*r3_inv + qrr_r5;
241             }
242             force[ip].acc += ai;
243             force[ip].pot += poti;
244         }
245     }
246 };
247 #elif MONOPOLEGEOMETRICCENTER
248 struct CalcForceSpEp{
249     void operator () (const EPIGrav * ep_i,
250                       const PS::S32 n_ip,
251                       const PS::SPJMonopoleGeometricCenter *
                              sp_j,
252                       const PS::S32 n_jp,
253                       ForceGrav * force){
254         PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
255         for(PS::S32 i=0; i<n_ip; i++){
256             PS::F64vec xi = ep_i[i].pos;
257             PS::F64vec ai = 0.0;
258             PS::F64 poti = 0.0;
259             for(PS::S32 j=0; j<n_jp; j++){
260                 PS::F64vec rij = xi - sp_j[j].pos;
261                 PS::F64 r3_inv = rij * rij + eps2;
262                 PS::F64 r_inv = 1.0/sqrt(r3_inv);
263                 r3_inv = r_inv * r_inv;
264                 r_inv *= sp_j[j].charge;
265                 r3_inv *= r_inv;
266                 ai -= r3_inv * rij;
267                 poti -= r_inv;
268             }
269             force[i].acc += ai;
```

```
270             force[i].pot += poti;
271         }
272     }
273 };
274 #elif DIPOLEGEOMETRICCENTER
275 struct CalcForceSpEp{
276     void operator () (const EPIGrav * ep_i,
277                       const PS::S32 n_ip,
278                       const PS::SPJDipoleGeometricCenter * sp_j
                              ,
279                       const PS::S32 n_jp,
280                       ForceGrav * force){
281         const PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
282         for(PS::S32 i=0; i<n_ip; i++){
283             const PS::F64vec xi = ep_i[i].pos;
284             PS::F64vec ai = 0.0;
285             PS::F64 poti = 0.0;
286             for(PS::S32 j=0; j<n_jp; j++){
287                 const PS::F64vec di = sp_j[j].dipole;
288                 const PS::F64vec rij = xi - sp_j[j].pos;
289                 const PS::F64 r2 = rij * rij + eps2;
290                 const PS::F64 r_inv = 1.0/sqrt(r2);
291                 const PS::F64 r2_inv = r_inv * r_inv;
292                 const PS::F64 r3_inv = r_inv * r2_inv;
293                 const PS::F64vec hij = rij * r_inv;
294                 const PS::F64 dihij = di * hij;
295                 const PS::F64 mj = sp_j[j].charge;
296                 poti -= mj * r_inv + dihij* r2_inv;
297                 ai -= (mj*r2_inv + 3.0*dihij*r3_inv) * hij - di
                        ;
298             }
299             force[i].acc += ai;
300             force[i].pot += poti;
301         }
302     }
303 };
304 #elif QUADRUPOLEGEOMETRICCENTER
305 struct CalcForceSpEp{
306     void operator () (const EPIGrav * ep_i,
307                       const PS::S32 n_ip,
```

```
308                     const PS::SPJQuadrupoleGeometricCenter *
                            sp_j,
309                     const PS::S32 n_jp,
310                     ForceGrav * force){
311         const PS::F64 eps2 = EPIGrav::eps * EPIGrav::eps;
312         for(PS::S32 ip=0; ip<n_ip; ip++){
313             const PS::F64vec xi = ep_i[ip].pos;
314             PS::F64vec ai = 0.0;
315             PS::F64 poti = 0.0;
316             for(PS::S32 jp=0; jp<n_jp; jp++){
317                 PS::F64 mj = sp_j[jp].charge;
318                 PS::F64vec xj= sp_j[jp].pos;
319                 PS::F64vec rij= xi - xj;
320                 PS::F64 r2 = rij * rij + eps2;
321                 PS::F64mat qj = sp_j[jp].quadrupole;
322                 PS::F64 tr = qj.getTrace();
323                 PS::F64vec qr( (qj.xx*rij.x + qj.xy*rij.y + qj.
                        xz*rij.z),
324                                (qj.yy*rij.y + qj.yz*rij.z + qj.
                                    xy*rij.x),
325                                (qj.zz*rij.z + qj.xz*rij.x + qj.
                                    yz*rij.y) );
326                 PS::F64 qrr = qr * rij;
327                 PS::F64 r_inv = 1.0f/sqrt(r2);
328                 PS::F64 r2_inv = r_inv * r_inv;
329                 PS::F64 r3_inv = r2_inv * r_inv;
330                 PS::F64 r5_inv = r2_inv * r3_inv * 1.5;
331                 PS::F64 qrr_r5 = r5_inv * qrr;
332                 PS::F64 qrr_r7 = r2_inv * qrr_r5;
333                 PS::F64 A = mj*r3_inv - tr*r5_inv + 5*qrr_r7;
334                 PS::F64 B = -2.0*r5_inv;
335                 ai -= A*rij + B*qr;
336                 poti -= mj*r_inv - 0.5*tr*r3_inv + qrr_r5;
337                 const PS::F64vec di = sp_j[jp].dipole;
338                 const PS::F64 dirij = di * rij;
339                 poti -= dirij* r3_inv;
340                 ai -= 3.0*dirij*r5_inv*rij - di;
341             }
342             force[ip].acc += ai;
343             force[ip].pot += poti;
```

```
344         }
345     }
346 };
347 #endif
348 
349 template<class Tpsys>
350 void ReadNemoAscii(Tpsys & psys,
351                    PS::S32 & n_glb,
352                    PS::S32 & n_loc,
353                    PS::F32 & t_sys,
354                    const char * ifile){
355     std::ifstream finput;
356     finput.open(ifile);
357     assert(finput);
358     PS::S32 dim;
359     finput>>n_glb>>dim>>t_sys;
360     std::cerr<<"ifile:"<<ifile<<std::endl;
361     std::cerr<<"n_glb="<<n_glb<<std::endl;
362     std::cerr<<"dim="<<dim<<std::endl;
363     std::cerr<<"t_sys="<<t_sys<<std::endl;
364 
365     PS::S32 my_rank = PS::Comm::getRank();
366     PS::S32 n_proc = PS::Comm::getNumberOfProc();
367     n_loc = n_glb/n_proc;
368     if( n_glb % n_proc > my_rank) n_loc++;
369 
370     psys.createParticle((n_glb/n_proc)*8+10000);
371     psys.setNumberOfParticleLocal(n_loc);
372 
373     PS::F32vec pos_shift = 0.0;
374 
375     PS::S32 i_h = n_glb/n_proc*my_rank;
376     if( n_glb % n_proc  > my_rank) i_h += my_rank;
377     else i_h += n_glb % n_proc;
378     const PS::S32 i_t = i_h+n_loc;
379     PS::F32 xf32;
380     PS::F32vec vf32;
381 
382     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++)psys[n].id = i;
383 
```

```
384     for(PS::S32 i=0; i<i_h; i++) finput>>xf32;
385     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++){
386         finput>>psys[n].mass;
387         //psys[n].mass *= (PS::MT::genrand_real1()+0.5);
388     }
389     for(PS::S32 i=i_t; i<n_glb; i++) finput>>xf32;
390
391     for(PS::S32 i=0; i<i_h; i++) finput>>vf32;
392     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++){
393         finput>>psys[n].pos;
394         psys[n].pos += pos_shift;
395     }
396     for(PS::S32 i=i_t; i<n_glb; i++) finput>>vf32;
397
398     for(PS::S32 i=0; i<i_h; i++) finput>>vf32;
399     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++) finput>>psys[n].
          vel;
400     for(PS::S32 i=i_t; i<n_glb; i++) finput>>vf32;
401     finput.close();
402 }
403
404 template<class Tpsys>
405 void WriteNemoAscii(const Tpsys & psys,
406                     const PS::F32 time_sys,
407                     const PS::S32 snp_id,
408                     const char * dir_name){
409     const PS::S32 n_loc = psys.getNumberOfParticleLocal();
410     PS::S32 n_glb = 0;
411     FPGrav * fp;
412     PS::AllGatherParticle(fp, n_glb, &psys[0], n_loc);
413     if(PS::Comm::getRank () == 0){
414         const PS::S32 STRINGSIZE = 1024;
415         char sout[STRINGSIZE];
416         sprintf(sout,"%s/snap%5d.dat", dir_name, snp_id);
417         for(int i=0;i<STRINGSIZE;i++)if(sout[i]==' ')sout[i]='0
              ';
418         std::ofstream foutput;
419         foutput.open(sout);
420         foutput<<std::setprecision(15);
421         foutput<<n_glb<<std::endl;
```

```
422         foutput<<"3"<<std::endl;
423         foutput<<time_sys<<std::endl;
424         for(PS::S32 i=0; i<n_glb; i++) foutput<<fp[i].mass<<std
                ::endl;
425         for(PS::S32 i=0; i<n_glb; i++) foutput<<fp[i].pos<<std
                ::endl;
426         for(PS::S32 i=0; i<n_glb; i++) foutput<<fp[i].vel<<std
                ::endl;
427         foutput.close();
428     }
429 }
430
431 template<class Tpsys>
432 void Kick(Tpsys & system,
433           const PS::F64 dt){
434     PS::S32 n = system.getNumberOfParticleLocal();
435     for(int i=0; i<n; i++){
436         system[i].vel  += system[i].acc * dt;
437     }
438 }
439
440 template<class Tpsys>
441 void Drift(Tpsys & system,
442            const PS::F64 dt){
443     PS::S32 n = system.getNumberOfParticleLocal();
444     for(int i=0; i<n; i++){
445         system[i].pos  += system[i].vel * dt;
446     }
447 }
448
449 template<class Tpsys>
450 void CalcEnergy(const Tpsys & system,
451                 PS::F64 & etot,
452                 PS::F64 & ekin,
453                 PS::F64 & epot,
454                 const bool clear=true){
455     if(clear){
456         etot = ekin = epot = 0.0;
457     }
458     PS::F64 etot_loc = 0.0;
```

```
459     PS::F64 ekin_loc = 0.0;
460     PS::F64 epot_loc = 0.0;
461     const PS::S32 nbody = system.getNumberOfParticleLocal();
462     for(PS::S32 i=0; i<nbody; i++){
463         ekin_loc += system[i].mass * system[i].vel * system[i].
                vel;
464         epot_loc += system[i].mass * system[i].pot;
465     }
466     ekin_loc *= 0.5;
467     epot_loc *= 0.5;
468     etot_loc = ekin_loc + epot_loc;
469     MPI::COMM_WORLD.Allreduce(&etot_loc, &etot, 1, PS::
            GetDataType<PS::F64>(), MPI::SUM);
470     MPI::COMM_WORLD.Allreduce(&epot_loc, &epot, 1, PS::
            GetDataType<PS::F64>(), MPI::SUM);
471     MPI::COMM_WORLD.Allreduce(&ekin_loc, &ekin, 1, PS::
            GetDataType<PS::F64>(), MPI::SUM);
472 }
473
474 int main(int argc, char *argv[]){
475     PS::F64 Tbegin = PS::GetWtime();
476     std::cout<<std::setprecision(15);
477     std::cerr<<std::setprecision(15);
478     PS::Initialize(argc, argv);
479     PS::F32 theta = 0.5;
480     const PS::S32 n_leaf_limit = 8;
481     PS::S32 n_group_limit = 64;
482     PS::F32 time_end = 10.0;
483     char sinput[1024];
484     char dir_name[1024];
485     int c;
486     while((c=getopt(argc,argv,"i:o:t:T:n:h")) != -1){
487         switch(c){
488         case 'i':
489             sprintf(sinput,optarg);
490             break;
491         case 'o':
492             sprintf(dir_name,optarg);
493             break;
494         case 't':
```

```
495             theta = atof(optarg);
496             std::cerr<<"theta="<<theta<<std::endl;
497             break;
498         case 'T':
499             time_end = atof(optarg);
500             std::cerr<<"time_end="<<time_end<<std::endl;
501             break;
502         case 'n':
503             n_group_limit = atoi(optarg);
504             std::cerr<<"n_group_limit="<<n_group_limit<<std::
                    endl;
505             break;
506         case 'h':
507             std::cerr<<"i: input file name (nemo ascii)"<<std::
                    endl;
508             std::cerr<<"o: dir name of output"<<std::endl;
509             std::cerr<<"t: theta (dafult: 0.5)"<<std::endl;
510             std::cerr<<"T: time_end (dafult: 10.0)"<<std::endl;
511             std::cerr<<"n: n_group_limit (dafult: 64.0)"<<std::
                    endl;
512             return 0;
513         }
514     }
515
516     std::ofstream fout_eng;
517     std::ofstream fout_tcal;
518
519     char sout_de[1024];
520     char sout_tcal[1024];
521     sprintf(sout_de, "%s/t-de.dat", dir_name);
522     sprintf(sout_tcal, "%s/t-tcal.dat", dir_name);
523     std::cerr<<sout_de<<std::endl;
524     std::cerr<<sout_tcal<<std::endl;
525     fout_eng.open(sout_de);
526     fout_tcal.open(sout_tcal);
527
528     PS::ParticleSystem<FPGrav> system_grav;
529     system_grav.initialize();
530     PS::S32 n_grav_glb, n_grav_loc;
531     PS::F32 time_sys;
```

```
532     ReadNemoAscii(system_grav, n_grav_glb, n_grav_loc, time_sys
            , sinput);
533     const PS::F32 coef_ema = 0.7;
534     PS::DomainInfo dinfo;
535     dinfo.initialize(coef_ema);
536     dinfo.collectSampleParticle(system_grav);
537     dinfo.decomposeDomain();
538     system_grav.exchangeParticle(dinfo);
539     n_grav_loc = system_grav.getNumberOfParticleLocal();
540 #ifdef MONOPOLE
541     PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav>::Monopole
             tree_grav;
542 #elif QUADRUPOLE
543     PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav>::
            Quadrupole tree_grav;
544 #elif MONOPOLEGEOMETRICCENTER
545     PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav>::
            MonopoleGeometricCenter tree_grav;
546 #elif DIPOLEGEOMETRICCENTER
547     PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav>::
            DipoleGeometricCenter tree_grav;
548 #elif QUADRUPOLEGEOMETRICCENTER
549     PS::TreeForForceLong<ForceGrav, EPIGrav, EPJGrav>::
            QuadrupoleGeometricCenter tree_grav;
550 #endif
551
552     tree_grav.initialize(n_grav_glb, theta, n_leaf_limit,
            n_group_limit);
553
554     tree_grav.calcForceAllAndWriteBack(CalcForceEpEp(),
            CalcForceSpEp(), system_grav, dinfo);
555 #ifdef CHECKFORCE
556     tree_grav.checkForce( CalcForceEpEp(), CompareGrav(), dinfo
            );
557 #else
558
559     PS::F64 Epot0, Ekin0, Etot0, Epot1, Ekin1, Etot1;
560     CalcEnergy(system_grav, Etot0, Ekin0, Epot0);
561
562     const PS::F32 dt = 1.0/128.0;
```

```
563
564     Kick(system_grav, dt*0.5);
565
566     PS::F64 Tloop = 0.0;
567
568     PS::S32 snp_id = 0;
569     while(time_sys < time_end){
570
571         PS::Timer timer;
572         timer.reset();
573         timer.start();
574         if( fmod(time_sys, 10.0) == 0.0){
575             WriteNemoAscii(system_grav, time_sys, snp_id,
                    dir_name);
576             snp_id++;
577         }
578         timer.restart("WriteNemoAscii");
579
580         time_sys += dt;
581         Drift(system_grav, dt);
582
583         timer.restart("Drift");
584
585         if( fmod(time_sys, 1.0/32.0) == 0.0){
586             dinfo.collectSampleParticle(system_grav, Tloop);
587             dinfo.decomposeDomain();
588         }
589
590         timer.restart("collect+decompose");
591
592         system_grav.exchangeParticle(dinfo);
593
594         timer.restart("exchangeParticle");
595
596         Tloop = PS::GetWtime();
597
598         tree_grav.calcForceAllAndWriteBackWithTimer
599             (CalcForceEpEp(), CalcForceSpEp(), system_grav,
                    dinfo, timer, true);
600
```

```
601         Tloop = PS::GetWtime() - Tloop;
602
603
604         Kick(system_grav, dt*0.5);
605
606         timer.stop("Kick");
607
608         fout_tcal<<"time_sys=␣"<<time_sys<<std::endl;
609         fout_tcal<<"tree_grav.getMemSizeUsed()=␣"<<tree_grav.
                getMemSizeUsed()*1e-9<<"␣[Gbyte]";
610         fout_tcal<<"␣system_grav.getMemSizeUsed()=␣"<<
                system_grav.getMemSizeUsed()*1e-9<<"␣[Gbyte]"<<
                std::endl;
611         tree_grav.dump_calc_cost(PS::Comm::getMaxValue(Tloop),
                fout_tcal);
612         fout_tcal<<"Tloop=␣"<<Tloop<<"␣Ttot="<<PS::GetWtime()-
                Tbegin<<std::endl;
613         timer.dump(fout_tcal);
614         fout_tcal<<std::endl;
615
616         CalcEnergy(system_grav, Etot1, Ekin1, Epot1);
617         if(PS::Comm::getRank() == 0){
618             fout_eng<<time_sys<<"␣␣␣"<<(Etot1-Etot0)/Etot0<<std
                    ::endl;
619         }
620         Kick(system_grav, dt*0.5);
621     }
622
623 #endif
624
625     PS::Finalize();
626     return 0;
627 }
```

7.2 衝撃波菅問題

7.2.1 計算 1

以下では一次元衝撃波菅問題を 3 次元ツリー構造を使って解く。
初期条件

図 11: 密度

図 12: 圧力

粒子数は1000で、$0 \leq x < 1$に等間隔で分布させる。$(\rho, p, v_x) = (1.0, 2.5, 0.0), (0.25, 1.795, 0.0)$。前者が$0 \leq x < 0.5$、後者が$0.5 \leq x < 1$の場合。$\alpha = 0.9$、$\gamma = 1.4$、時間刻みの精度パラメータ0.1。

結果

図 13: 速度

7.2.2 計算 2

以下では粒子をランダムにばら撒いた時の計算である。

初期条件

粒子数は6.4e7で、$0 \leq x < 1$にランダムに分布させる。$P = 1$、$\alpha = 1.0$、$\gamma = 1.4$。

結果

以下は京512-4096ノードで行った場合の1ステップの計算時間である。

使用コード

ソースコード 15: SPH サンプル使用

```
1
2 #include<iostream>
3 #include<fstream>
4 #include<unistd.h>
5 #include<particle_simulator.hpp>
6
7 PS::F64 CubicSpline(const PS::F64 r_sq,
8                     const PS::F64 h_inv){
9     PS::F64 xi = sqrt(r_sq) * h_inv;
10    PS::F64 xi10 = (1.0-xi > 0.0) ? 1.0-xi : 0.0;
11    PS::F64 xi05 = (0.5-xi > 0.0) ? 0.5-xi : 0.0;
12    return xi10*xi10*xi10 - 4.0*xi05*xi05*xi05;
13 }
14
15 PS::F64vec CubicSpline_ri(const PS::F64vec & rij,
```

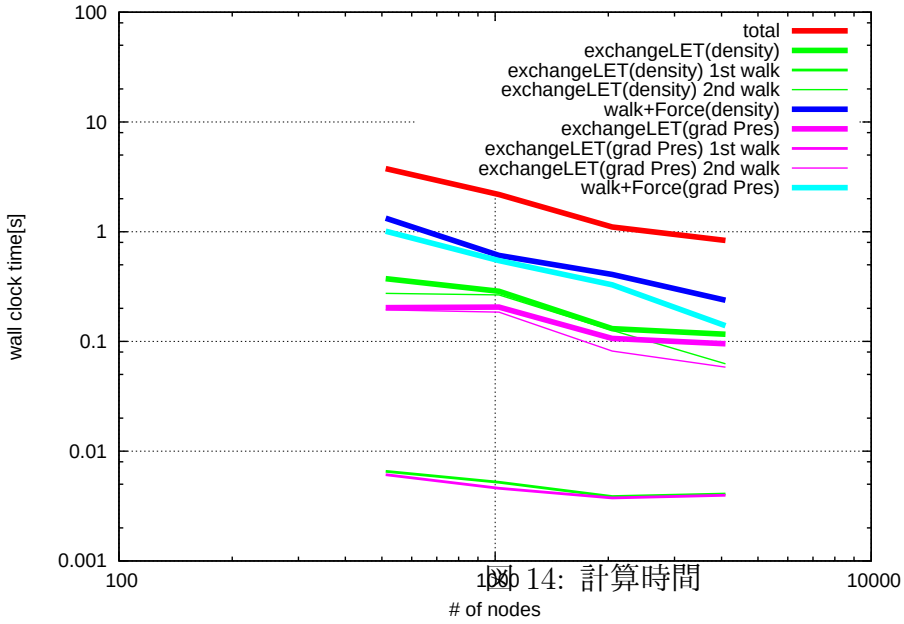

図 14: 計算時間

```
16                                  const PS::F64 h_inv){
17      PS::F64 r_sq = rij * rij;
18      PS::F64 r = sqrt(r_sq);
19      PS::F64 xi = r * h_inv;
20      PS::F64 xi10 = (1.0-xi > 0.0) ? 1.0-xi : 0.0;
21      PS::F64 xi05 = (0.5-xi > 0.0) ? 0.5-xi : 0.0;
22      PS::F64 C = (3.0*xi10*xi10 - 12.0*xi05*xi05) * h_inv / r;
23      return -C * rij;
24  }
25
26  class ResultDens{
27  public:
28      PS::F64 dens;
29      PS::F64 divv;
30      void clear(){
31          dens = divv = 0.0;
32      }
33  };
34
35  class ResultForce{
36  public:
37      PS::F64 eng_dot;
```

```
38      PS::F64vec acc;
39      void clear(){
40          acc = 0.0;
41          eng_dot = 0.0;
42      }
43  };
44
45  class FullPtcl{
46  public:
47
48      PS::F64 getCharge() const {
49          return this->mass;
50      }
51      PS::F64vec getPos() const {
52          return this->pos;
53      }
54      PS::F64 getRSearch() const {
55          return this->kernel_length;
56      }
57      void copyFromForce(const ResultDens & rd){
58          dens = rd.dens;
59          divv = rd.divv;
60      }
61      void copyFromForce(const ResultForce & rf){
62          acc = rf.acc;
63          eng_dot = rf.eng_dot;
64      }
65
66      PS::F64 mass;
67      PS::F64vec pos;
68      PS::F64 kernel_length;
69      PS::F64 dens;
70      PS::F64 divv;
71      PS::F64 pres;
72      PS::F64 vel_sound;
73      PS::F64vec acc;
74      PS::F64 eng_dot;
75      PS::S64 id;
76      PS::F64vec vel;
77      PS::F64 eng;
```

```
78      PS::F64vec vel_half;
79      PS::F64 eng_half;
80      PS::S64 n_neighbour;
81      void dump(std::ostream & fout=std::cout) const {
82          fout<<"pos="<<pos<<std::endl;
83          fout<<"dens="<<dens<<std::endl;
84          fout<<"kernel_length="<<kernel_length<<std::endl;
85      }
86  };
87
88
89  class EPIDens{
90  public:
91      PS::S64 id;
92      PS::F64vec pos;
93      PS::F64vec vel;
94      PS::F64vec getPos() const { return pos;}
95      void copyFromFP(const FullPtcl & fp){
96          id = fp.id;
97          pos = fp.getPos();
98      }
99  };
100
101 class EPJDens{
102 public:
103     PS::S64 id;
104     PS::F64 mass;
105     static PS::F64 kernel_length;
106     PS::F64vec pos;
107     PS::F64vec vel;
108     PS::F64 getCharge() const { return mass; }
109     PS::F64vec getPos() const { return pos; }
110     PS::F64 getRSearch() const { return kernel_length; }
111     void copyFromFP(const FullPtcl & fp){
112         id = fp.id;
113         mass = fp.getCharge();
114         kernel_length = fp.getRSearch();
115         pos = fp.getPos();
116     }
117 };
```

```
118
119 PS::F64 EPJDens::kernel_length;
120
121 struct CalcDensEpEp{
122     void operator () (const EPIDens * ep_i,
123                       const PS::S32 n_ip,
124                       const EPJDens * ep_j,
125                       const PS::S32 n_jp,
126                       ResultDens * dens){
127         static const PS::F64 h_inv = 1.0/ep_j[0].kernel_length;
128         static const PS::F64 r_crit_sq = ep_j[0].kernel_length
                * ep_j[0].kernel_length;
129         static const PS::F64 Cnorm = 8.0/3.0 * h_inv; // 1D
130         //static PS::F64 Cnorm = 80.0/(7.0*M_PI); // 2D
131         //static PS::F64 Cnorm = 16.0/M_PI; // 3D
132         for(PS::S32 i=0; i<n_ip; i++){
133             dens[i].clear();
134             for(PS::S32 j=0; j<n_jp; j++){
135                 const PS::F64vec rij = ep_i[i].getPos() - ep_j[
                        j].getPos();
136                 const PS::F64 r_sq = rij * rij;
137                 if( r_crit_sq > r_sq ){
138                     dens[i].dens += ep_j[j].getCharge() *
                            CubicSpline(r_sq, h_inv);
139                 }
140             }
141             dens[i].dens *= Cnorm;
142             PS::F64 divv = 0.0;
143             PS::F64 inv_dens = 1.0/dens[i].dens;
144             for(PS::S32 j=0; j<n_jp; j++){
145                 if(ep_i[i].id == ep_j[j].id) continue;
146                 const PS::F64vec rij = ep_i[i].getPos() - ep_j[
                        j].getPos();
147                 const PS::F64vec vij = ep_i[i].vel - ep_j[j].
                        vel;
148                 const PS::F64 mj = ep_j[j].mass;
149                 divv -= mj * vij * CubicSpline_ri(rij, h_inv);
150             }
151             dens[i].divv = divv * Cnorm * inv_dens;
152         }
```

```
153     }
154 };
155
156
157 template<class Tpsys>
158 void CalcPressureAndSoundVelocity(Tpsys & system, const PS::F32
        gamma){
159     const PS::F32 C = gamma - 1.0;
160     const PS::S32 n_ptcl = system.getNumberOfParticleLocal();
161     for(PS::S32 i=0; i<n_ptcl; i++){
162         system[i].pres = C * system[i].dens * system[i].eng;
163         system[i].vel_sound = sqrt(gamma * system[i].pres /
                system[i].dens);
164     }
165 }
166
167 class EPIForce{
168 public:
169     PS::F64vec pos;
170     PS::F64vec vel;
171     PS::F64 divv;
172     PS::F64 vel_sound;
173     PS::F64 dens;
174     PS::F64 pres;
175     PS::F64 kernel_length;
176     PS::S64 id;
177     static PS::F64 alpha;
178     void copyFromFP(const FullPtcl & fp){
179         pos = fp.pos;
180         vel = fp.vel;
181         divv = fp.divv;
182         vel_sound = fp.vel_sound;
183         dens = fp.dens;
184         pres = fp.pres;
185         kernel_length = fp.kernel_length;
186         id = fp.id;
187     }
188 };
189
190 PS::F64 EPIForce::alpha = 0.9;
```

```
191
192 class EPJForce{
193 public:
194     PS::F64vec pos;
195     PS::F64vec vel;
196     PS::F64 divv;
197     PS::F64 vel_sound;
198     PS::F64 pres;
199     PS::F64 dens;
200     PS::F64 kernel_length;
201     PS::F64 mass;
202     PS::S64 id;
203     PS::F64vec getPos() const {return pos;}
204     PS::F64 getRSearch() const {return kernel_length;}
205     void copyFromFP(const FullPtcl & fp){
206         pos = fp.pos;
207         vel = fp.vel;
208         divv = fp.divv;
209         vel_sound = fp.vel_sound;
210         pres = fp.pres;
211         dens = fp.dens;
212         kernel_length = fp.kernel_length;
213         mass = fp.mass;
214         id = fp.id;
215     }
216 };
217
218 struct CalcForceEpEp{
219     void operator () (const EPIForce * ep_i,
220                       const PS::S32   n_ip,
221                       const EPJForce * ep_j,
222                       const PS::S32   n_jp,
223                       ResultForce    * force){
224         for(PS::S32 i=0; i<n_ip; i++){
225             PS::F64 cs_i = ep_i[i].vel_sound;
226             PS::F64 h_i = ep_i[i].kernel_length;
227             PS::F64 inv_h_i = 1.0 / h_i;
228             PS::F64 Cnorm_i = (8.0/3.0) * inv_h_i; // 1D
229             PS::F64 dens_i = ep_i[i].dens;
230             PS::F64 pres_i = ep_i[i].pres;
```

```
231             PS::F64 f_grad_i = 1.0;
232             PS::F64 fp_dens_i = f_grad_i * pres_i / (dens_i *
                    dens_i);
233             PS::F64 alpha = EPIForce::alpha;
234             PS::F64vec acc = 0.0;
235             PS::F64 udot0 = 0.0;
236             PS::F64 udot1 = 0.0;
237             for(PS::S32 j=0; j<n_jp; j++){
238                 PS::F64 mj = ep_j[j].mass;
239                 PS::F64vec r_ij = ep_i[i].pos - ep_j[j].pos;
240                 if(ep_i[i].id == ep_j[j].id) continue;
241                 PS::F64vec v_ij = ep_i[i].vel - ep_j[j].vel;
242                 PS::F64 r_sq = r_ij * r_ij;
243                 PS::F64 h_j = ep_j[j].kernel_length;
244                 PS::F64 inv_h_j = 1.0 / h_j;
245                 PS::F64 Cnorm_j = (8.0/3.0) * inv_h_j; // 1D
246                 PS::F64 h_ij = (h_i + h_j) * 0.5;
247                 PS::F64 inv_h_ij = 1.0 / h_ij;
248                 PS::F64 Cnorm_ij = (8.0/3.0) * inv_h_ij; // 1D
249                 PS::F64 dens_j = ep_j[j].dens;
250                 PS::F64 pres_j = ep_j[j].pres;
251                 PS::F64 f_grad_j = 1.0;
252                 PS::F64 fp_dens_j = f_grad_j * pres_j / (dens_j
                        * dens_j);
253                 PS::F64 dens_ij = (dens_i + dens_j) * 0.5;
254                 PS::F64 w_ij = v_ij * r_ij / sqrt(r_sq);
255                 PS::F64 vel_sig = cs_i + ep_j[j].vel_sound - 3*
                        w_ij;
256                 PS::F64 pi_ij = -alpha*0.5*vel_sig*w_ij /
                        dens_ij;
257                 pi_ij = ( r_ij * v_ij <= 0.0 ) ? pi_ij : 0.0;
258                 PS::F64vec grad_w_i = Cnorm_i * CubicSpline_ri(
                        r_ij, inv_h_i);
259                 PS::F64vec grad_w_j = Cnorm_j * CubicSpline_ri(
                        r_ij, inv_h_j);
260                 PS::F64vec grad_w_ij = Cnorm_ij *
                        CubicSpline_ri(r_ij, inv_h_ij);
261                 acc -= mj * ( fp_dens_i * grad_w_i + fp_dens_j
                        * grad_w_j
262                                 + pi_ij * grad_w_ij );
```

```
263                 PS::F64vec mjvij = mj * v_ij;
264                 udot0 += mjvij * grad_w_i;
265                 udot1 += pi_ij * mjvij * grad_w_ij;
266             }
267             force[i].acc = acc;
268             force[i].eng_dot = fp_dens_i * udot0 + 0.5 * udot1;
269         }
270     }
271 };
272
273 template<class Tpsys>
274 void Kick1( Tpsys & system,
275             const PS::F64 dt){
276     PS::S32 n = system.getNumberOfParticleLocal();
277     for(PS::S32 i=0; i<n; i++){
278         system[i].vel_half  = system[i].vel + system[i].acc *
                dt;
279         system[i].eng_half  = system[i].eng + system[i].eng_dot
                * dt;
280     }
281 }
282
283 template<class Tpsys>
284 void Drift( Tpsys & system,
285             const PS::F64 dt){
286     PS::S32 n = system.getNumberOfParticleLocal();
287     for(PS::S32 i=0; i<n; i++){
288         system[i].pos  += system[i].vel_half * dt; // corrected
                value
289         system[i].vel  += system[i].acc * dt;           //
                predicted value
290         system[i].eng  += system[i].eng_dot * dt;        //
                predicted value
291     }
292 }
293
294 template<class Tpsys>
295 void Kick2( Tpsys & system,
296             const PS::F64 dt){
297     PS::S32 n = system.getNumberOfParticleLocal();
```

```
298     for(PS::S32 i=0; i<n; i++){
299         system[i].vel  = system[i].vel_half + system[i].acc *
                dt;
300         system[i].eng  = system[i].eng_half + system[i].eng_dot
                 * dt;
301     }
302 }
303
304 template<class Tpsys>
305 PS::F64 CalcDt(Tpsys & system,
306                const PS::F64 Ccfl){
307     PS::S32 n = system.getNumberOfParticleLocal();
308     PS::F64 dt = 99999.9;
309     for(PS::S32 i=0; i<n; i++){
310         dt = (dt < (system[i].kernel_length/system[i].vel_sound
                )) ? dt : (system[i].kernel_length/system[i].
                vel_sound);
311     }
312     PS::F64 dt_glb = PS::GetMinValue(dt);
313     return dt_glb * Ccfl;
314 }
315
316 template<class Tpsys>
317 void ReadSPHFormat(Tpsys & psys,
318                    PS::S32 & n_glb,
319                    PS::S32 & n_loc,
320                    PS::F64 & t_sys,
321                    const char * ifile){
322     std::ifstream finput;
323     finput.open(ifile);
324     assert(finput);
325     PS::S32 dim;
326     finput>>n_glb>>dim>>t_sys;
327
328     PS::S32 my_rank = PS::Comm::getRank();
329     PS::S32 n_proc = PS::Comm::getNumberOfProc();
330     n_loc = n_glb/n_proc;
331     if( n_glb % n_proc > my_rank) n_loc++;
332
333     psys.createParticle((n_glb/n_proc)*8+10000);
```

```
334     psys.setNumberOfParticleLocal(n_loc);
335
336     PS::S32 i_h = n_glb/n_proc*my_rank;
337     if( n_glb % n_proc  > my_rank) i_h += my_rank;
338     else i_h += n_glb % n_proc;
339     const PS::S32 i_t = i_h+n_loc;
340     PS::S32 xs32;
341     PS::F32 xf32;
342     PS::F32vec vf32;
343
344     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++) psys[n].id = i;
345
346     for(PS::S32 i=0; i<i_h; i++){
347         finput>>xs32>>xf32>>xf32>>vf32>>vf32>>xf32;
348     }
349     for(PS::S32 i=i_h, n=0; i<i_t; i++, n++){
350         finput>>psys[n].id>>psys[n].mass>>psys[n].eng>>psys[n].
                pos>>psys[n].vel>>psys[n].kernel_length;
351         psys[n].kernel_length = 1.0/n_glb * 3.0;
352     }
353 }
354
355 struct CompareDens{
356     void operator () (ResultDens * dens0, ResultDens * dens1,
357                       const PS::S32 n, std::ostream & fout){
358         bool err = false;
359         for(PS::S32 i=0; i<n; i++){
360             if( std::abs(dens0[i].dens - dens1[i].dens) > 1e
                    -5){
361                 fout<<"CompareDensity:␣FAIL"<<std::endl;
362                 fout<<"desn0[i].dens="<<dens0[i].dens<<"␣dens1[
                        i].dens="<<dens1[i].dens<<std::endl;
363                 err = true;
364             }
365             if( std::abs(dens0[i].divv - dens1[i].divv) > 1e
                    -5){
366                 fout<<"CompareDensity:␣FAIL"<<std::endl;
367                 fout<<"desn0[i].divv="<<dens0[i].divv<<"␣dens1[
                        i].divv="<<dens1[i].divv<<std::endl;
368                 err = true;
```

```
369             }
370         }
371         if(!err) fout<<"CompareDensity: PASS"<<std::endl;
372     }
373 };
374
375 struct CompareForce{
376     void operator () (ResultForce * force0, ResultForce *
            force1,
377                       const PS::S32 n, std::ostream & fout){
378         bool err = false;
379         for(PS::S32 i=0; i<n; i++){
380             if( std::abs(force0[i].eng_dot - force1[i].eng_dot)
                      > 1e-5){
381                 fout<<"CompareForce: FAIL"<<std::endl;
382                 fout<<"force0[i].eng_dot="<<force0[i].eng_dot<<
                        " force1[i].eng_dot="<<force1[i].eng_dot
                        <<std::endl;
383                 err = true;
384             }
385             PS::F64 dacc = sqrt( (force0[i].acc - force1[i].acc
                     )*(force0[i].acc - force1[i].acc)/ (force1[i
                     ].acc*force1[i].acc) );
386             if( dacc > 1e-5){
387                 fout<<"CompareForce: FAIL"<<std::endl;
388                 fout<<"force0[i].acc="<<force0[i].acc<<" force1
                        [i].acc="<<force1[i].acc<<std::endl;
389                 err = true;
390             }
391         }
392         if(!err) fout<<"CompareForce: PASS"<<std::endl;
393     }
394 };
395
396
397 template<class Tpsys>
398 void WriteSPHFormat(const Tpsys & psys,
399                     const PS::F32 time_sys,
400                     const PS::S32 snp_id,
401                     const char * dir_name){
```

```
402     const PS::S32 n_loc = psys.getNumberOfParticleLocal();
403     PS::S32 n_glb = 0;
404     FullPtcl * fp;
405     PS::AllGatherParticle(fp, n_glb, &psys[0], n_loc);
406     if(PS::Comm::getRank () == 0){
407         const PS::S32 STRINGSIZE = 1024;
408         char sout[STRINGSIZE];
409         sprintf(sout,"%s/snap%5d.dat", dir_name, snp_id);
410         for(int i=0;i<STRINGSIZE;i++)if(sout[i]==' ')sout[i]='0
                ';
411         std::ofstream foutput;
412         foutput.open(sout);
413         foutput<<std::setprecision(15);
414         foutput<<n_glb<<std::endl;
415         foutput<<"3"<<std::endl;
416         foutput<<time_sys<<std::endl;
417         for(PS::S32 i=0; i<n_glb; i++){
418             foutput<<fp[i].pos.x<<"   "<<fp[i].dens<<"   "<<fp[
                    i].pres<<"   "<<fp[i].vel.x<<std::endl;
419         }
420         foutput.close();
421     }
422 }
423
424
425 int main(int argc, char *argv[]){
426     std::cout<<std::setprecision(15);
427     std::cerr<<std::setprecision(15);
428     PS::Initialize(argc, argv);
429     char sinput[1024];
430     char dir_name[1024];
431     int c;
432     while((c=getopt(argc,argv,"i:o:h")) != -1){
433         switch(c){
434         case 'i':
435             sprintf(sinput,optarg);
436             break;
437         case 'o':
438             sprintf(dir_name,optarg);
439             break;
```

```
440         case 'h':
441             std::cerr<<"i: input file name (nemo ascii)"<<std::
                    endl;
442             std::cerr<<"o: dir name of output"<<std::endl;
443             return 0;
444         }
445     }
446
447     const PS::F64 gamma = 1.4;
448     const PS::F64 Ccfl = 0.1;
449     const PS::F64 time_end = 10.0;
450     PS::ParticleSystem<FullPtcl> system_sph;
451     system_sph.initialize();
452     PS::S32 n_sph_glb, n_sph_loc;
453     PS::F64 time_sys;
454     ReadSPHFormat(system_sph, n_sph_glb, n_sph_loc, time_sys,
            sinput);
455     PS::F64 half_len_sph_glb = system_sph.getHalfLength();
456     std::cout<<"half_len_sph_glb="<<half_len_sph_glb<<std::endl
            ;
457
458     PS::DomainInfo dinfo;
459     dinfo.initialize();
460     dinfo.setDomain(PS::Comm::getNumberOfProc(), 1, 1);
461     dinfo.collectSampleParticle(system_sph);
462     dinfo.decomposeDomain();
463
464     system_sph.exchangeParticle(dinfo);
465
466     PS::TreeForForce<PS::SEARCH_MODE_SCATTER, ResultDens,
            EPIDens, EPJDens, PS::MomentSearch, PS::
            SuperParticleBase> tree_dens;
467     tree_dens.initialize(n_sph_glb);
468     tree_dens.initializeLocalTree(half_len_sph_glb);
469     tree_dens.setParticleLocalTree(system_sph);
470
471     tree_dens.mortonSortLocalTreeOnly();
472     tree_dens.linkCellLocalTreeOnly();
473     tree_dens.calcMomentLocalTreeOnly();
474     tree_dens.exchangeLocalEssentialTree(dinfo);
```

```
475     tree_dens.setLocalEssentialTreeToGlobalTree();
476     tree_dens.mortonSortGlobalTreeOnly();
477     tree_dens.linkCellGlobalTreeOnly();
478     tree_dens.calcMomentGlobalTreeOnly();
479     tree_dens.makeIPGroup();
480     tree_dens.calcForceAndWriteBack(CalcDensEpEp(), system_sph
            );
481     //tree_dens.checkForce( CalcDensEpEp(), CompareDens());
482
483     // To get pres and vel_sound
484     CalcPressureAndSoundVelocity(system_sph, gamma);
485
486     PS::TreeForForce<PS::SEARCH_MODE_SCATTER, ResultForce,
            EPIForce, EPJForce, PS::MomentSearch, PS::
            SuperParticleBase> tree_force;
487
488     tree_force.initialize(n_sph_glb);
489     tree_force.initializeLocalTree(half_len_sph_glb);
490     tree_force.setParticleLocalTree(system_sph);
491
492     tree_force.mortonSortLocalTreeOnly();
493     tree_force.linkCellLocalTreeOnly();
494     tree_force.calcMomentLocalTreeOnly();
495     tree_force.exchangeLocalEssentialTree(dinfo);
496     tree_force.setLocalEssentialTreeToGlobalTree();
497     tree_force.mortonSortGlobalTreeOnly();
498     tree_force.linkCellGlobalTreeOnly();
499     tree_force.calcMomentGlobalTreeOnly();
500     tree_force.makeIPGroup();
501     tree_force.calcForceAndWriteBack(CalcForceEpEp(),
            system_sph);
502
503     PS::S32 snp_id = 0;
504     PS::S64 n_loop = 0;
505     while(time_sys < time_end){
506         if(0){
507             WriteSPHFormat(system_sph, time_sys, snp_id,
                    dir_name);
508             snp_id++;
509         }
```

```
510
511         PS::F64 dt = CalcDt(system_sph, Ccfl);
512         time_sys += dt;
513         Kick1(system_sph, dt*0.5);
514         Drift(system_sph, dt);
515
516         half_len_sph_glb = system_sph.getHalfLength();
517         //std::cout<<"half_len_sph_glb="<<half_len_sph_glb<<std
                ::endl;
518         dinfo.collectSampleParticle(system_sph);
519         dinfo.decomposeDomain();
520         system_sph.exchangeParticle(dinfo);
521
522         tree_dens.initializeLocalTree(half_len_sph_glb);
523         tree_dens.setParticleLocalTree(system_sph);
524
525         tree_dens.mortonSortLocalTreeOnly();
526         tree_dens.linkCellLocalTreeOnly();
527         tree_dens.calcMomentLocalTreeOnly();
528         tree_dens.exchangeLocalEssentialTree(dinfo);
529         tree_dens.setLocalEssentialTreeToGlobalTree();
530         tree_dens.mortonSortGlobalTreeOnly();
531         tree_dens.linkCellGlobalTreeOnly();
532         tree_dens.calcMomentGlobalTreeOnly();
533         tree_dens.makeIPGroup();
534         tree_dens.calcForceAndWriteBack(CalcDensEpEp(),
                system_sph);
535
536         // To get pres and vel_sound
537         CalcPressureAndSoundVelocity(system_sph, gamma);
538
539
540         tree_force.initializeLocalTree(half_len_sph_glb);
541         tree_force.setParticleLocalTree(system_sph);
542         tree_force.mortonSortLocalTreeOnly();
543         tree_force.linkCellLocalTreeOnly();
544         tree_force.calcMomentLocalTreeOnly();
545         tree_force.exchangeLocalEssentialTree(dinfo);
546         tree_force.setLocalEssentialTreeToGlobalTree();
547         tree_force.mortonSortGlobalTreeOnly();
```

```
548         tree_force.linkCellGlobalTreeOnly();
549         tree_force.calcMomentGlobalTreeOnly();
550         tree_force.makeIPGroup();
551         tree_force.calcForceAndWriteBack(CalcForceEpEp(),
                system_sph);
552         Kick2(system_sph, dt*0.5);
553         n_loop++;
554     }
555     PS::Finalize();
556     return 0;
557 }
```

8 コーディング規約

8.1 使用言語

基本的に C++実装。C++11 以降、MPI2 以降の機能は使わない。

8.2 命名規則

8.2.1 原則

名前はその意味が分かるように英語で付ける。単語は全て単数形にする。

8.2.2 ファイル名

ファイル名は全て小文字で単語と単語の間はアンダースコアでつなぐ。particle_system.hpp。

8.2.3 型名

型名は大文字で始まり単語の一番最初の文字も大文字とし、それ以外は小文字にする。DomainInfo。

8.2.4 変数名

クラスのメンバ変数名は全て小文字で単語と単語の間はアンダースコアでつなぎ、一番最後もアンダースコアで終わらす。pos_sample_。

8.2.5 関数名

クラスのメンバ関数名は最初の文字を小文字とし、単語の一番最初の文字は大文字とする。アクセサーは単語の最初を get、set で始める。loadParticle。

グローバル関数は最初の文字を大文字とし、単語の一番最初の文字も大文字とする。Initialize。

8.2.6 マクロ名

全て大文字とし、単語と単語の間はアンダースコアでつなぐ。PARTICLE_SIMULATOR_TWO_DIMENSION。

9　用語集

イメージ粒子 リアル粒子を境界に映した粒子

イメージ超粒子 イメージ粒子をまとめた粒子

イメージドメイン ルートドメインを境界に映したドメイン

リアル粒子 実際に存在する粒子．　対となるのがイメージ粒子

リアル超粒子 リアル粒子をまとめた粒子

ツリー粒子データ フル粒子データのサブセットで，ツリー作成に必要なデータ

ドメイン 1 プロセスが担当する計算領域

プロセス MPI プロセスの略

フル粒子データ リアル粒子の持つ全データ

リーフセル ツリーセルであり，子セルを持たないセル

ルートドメイン 全計算領域

i **粒子データ** フル粒子データのサブセットで，作用を計算する際に，作用される側の粒子が必要とするデータ

j **粒子データ** フル粒子データのサブセットで，作用を計算する際に，作用する側の粒子が必要とするデータ