Gather モード並列化の送信粒子リスト生成について

似鳥啓吾

2015 年 4 月 29 日

1　はじめに

FDPS では（主に SPH 用として）並列近傍粒子探索の機能として、Gather モード、Scatter モード、あるいはその両方の Symmetric モードをサポートしているが、送信粒子リストを作る部分の計算負荷が Gather モードでは Scater モードに対して桁で重いという問題が顕在化している。

やるべきことは

- ある体積 V に含まれる粒子セットを見つけなさい

というだけのことで、V が単純な直方体だとか、あるいは重複のない直方体数個の和集合ならどうということはない。実際問題として Gather モード探索で用いる V は、

- 半径も中心もバラバラの沢山の球体からなる和集合で、
- しかも球体同士のの体積重複も多い

というとても厄介な不定形状となっている。

1.1　単純な実装と問題点

確実に答えを得るには、

1. 個々の球体に対して近傍粒子探索を行い含まれる粒子リストを作る
2. 全ての球体の近傍粒子リストを「重複を排除しながら」結合する

特に 2. が面倒なのだけれど、「とりあえず動くもの」なら、`std::vector`, `std::sort`, `std::unique` の組み合わせか、もしくは `std::set` でも使うことで作ることができる。STL が速ければこれでもいい、という気になるかもしれないが、大きな問題は、「重複を排除する前の結合リスト」が「重複を排除した後の結合リスト」に比べて 100 倍とかに大きくなってしまうことで、これでは近傍粒子探索のコストが送信すべき粒子数の 100 倍になってしまっている（とりあえず動くものができてから発覚した似鳥の設計ミスです）。

1.2　可能な枝切り

Scatter→Gather の二段階通信になるのは避けられないのだが、

- 一段回目の通信で送信済みの粒子は探索の対象外とする
- 二段階目の送信粒子リストの作成途中であっても、既に送信リストに登録済みの粒子は探索の対象外とする

といった工夫が可能であるように思われる。しかし、一度ツリーに登録された粒子を「削除」する手続きは不可能ではなくても煩雑になる（どのタイミングでセルの持つ境界をアップデートするか？）のでできればやりたくない。

2 記号の定義

プロセス A とプロセス B の (直方体の領域 A と領域 B の) 間の通信を考える。それぞれ N_A 個と N_B 個の粒子を持っている。一段目の Scatter モードの通信で領域 A から領域 B へ $N_{A\to B}$ 個の粒子が、二段目の Gather モードの通信で領域 B から領域 A へ $N_{B\to A}$ 個の粒子が送られるものとする。正反対の通信も実際には発生するが、記号を簡略化するためにここでは考えないことにする。こうすることで、考えるべきは、領域 B において（プロセス B が)、受け取った $N_{A\to B}$ 個の粒子情報（座標と半径）から送信すべき $N_{B\to A}$ 個の粒子を決定する方法になる。

受け取った $N_{A\to B}$ 個の粒子から（外側境界を持った）ツリーを作ることになると思うが、こちらは「ツリー A」と呼んでしまうことにする。また、領域 B にいる N_B 個の粒子からは既に内側外側両方の境界を持ったツリーが構築済みのはずで、こちらは普通に「ツリー B」と呼ぶ。

球体を沢山重ねた複雑な形状の領域は $V_{A\to B}$ とでも呼んでおき、プロセス B の持つ粒子のうち未送信でかつこの領域の内側に入るものを列挙したい。

3 探索アルゴリズム

3.1 オリジナルのもの

オーダーは $O(N_{A\to B}\log N_B)$ となる。なぜなら、$N_{A\to B}$ 個の粒子に対してツリー B を辿って近傍粒子探索を行うから。実際には $\log N_B$ というよりは近傍粒子数程度のファクターがかかっていて Scatter モードに比べて桁で遅い。

3.2 重複の排除が不要なもの

これ自体はとても効率の良いアルゴリズムとはいえないのだけど、こらをベースに最適化を考えて行こうと思う。プロセス B の全ての N_B 粒子、ないしこれから一段回目で送信済みなものをスキップしたうえで、ツリー A を辿ることで $V_{A\to B}$ に含まれるものを抽出する。この場合は $O(N_B\log N_{A\to B})$ になる。明らかに $V_{A\to B}$ に含まれない粒子に対してもループが回るので効率は悪いが、殆どの粒子は再帰に入ることもなくスキップされる。

ツリー A 側に次のようなメソッドがあればいい（擬似コード）：

```
bool TreeA.does_include(const Vec3 pos);
```

自身の外側境界の外に粒子があれば単に `false` を返し、内側にあった場合は子ノードを再帰呼び出しし、ひ

とつでも true があれば true を、そうでなければ false を返す。

```
for(int ic=0; ic<8; ic++){
    if(child[ic].does_include(pos)) return true;
}
return false;
```

直方体バージョンも後で使うかも。

```
bool TreeA.has_overlap_with(const Orthotope ot);
```

3.3 領域 B 側のツリーも使う

領域 B 側のツリーも使えば、セル単位で $V_{A\to B}$ とオーバーラップの無いものをスキップできて効率が改善できる。いわゆる FMM で用いるような double traverse になるが、次のような場合は注意が必要：
ツリー B のひとつのセルが、ツリー A の複数のセルと相互作用する場合、同じ粒子が重複して送信リストに登録されるおそれがある。

ダブルトラバースそのものは、再帰を使えば簡単に書ける。

1. ツリー A は外側境界を、ツリー B は内側境界を使う
2. 最初は双方のツリーのルートセル同士を「相互作用」させる
3. 直方体同士に重なりが無ければそれ以上の探索は不要
4. 重なりがある場合は、どちらかのツリーを下に降り、8 つの子セルを相手と「相互作用」させる

実際問題としてどちらのツリーを下に降りるべきかにある程度の任意性があって、階層の浅い方、または体積の大きい方を優先して下るのがいいだろう。

これだとやっぱりリスト重複が避けられないので、ツリー A を辿るときは常にルートからということにして、3.2 から辿る粒子数を N_B から減らしてみる。

1. ツリー B をルートから下へ降りて行く
   (a) if セルの外側境界が領域 A とオーバーラップしていたら下へ降りる（セル内に送信済み粒子が混ざっているから）
   (b) else セルの内側境界が $V_{A\to B}$ とオーバーラップしていたら下へ降りる（送るべき粒子が混ざっているから)、この判定のためには内側境界でツリー A を辿る
   (c) else どちらのオーバーラップも無ければ探索終了、return
2. リーフまで来たら、粒子でもやることは同じ。先ずは送信済みでないことを領域 A との距離計算で確認し、未送信だったらツリー A を辿ることで $V_{A\to B}$ に含まれるかを確認、含まれていれば送信リストに push back

これでオーダーは $O((N_{B\to A}+\log N_B)\log N_{A\to B})$ ぐらいになる。速いかはわからないけど、実装するのは簡単そう。

http://ja.wikipedia.org/wiki/粒子法