Vol.2012-ARC-202 No.34
Vol.2012-HPC-137 No.34
2012/12/14

粒子分布を考慮した粒子接触判定計算のMPIおよびOpenMPによる並列化

高木　翔[1,a)]　和田　直樹[1]　岡　大樹[2]　竹田　宏[2]　片桐　孝洋[3]　堀端　康善[1]

概要：粒子シミュレーションでは接触判定計算の計算負荷が最も大きく高速化が必要である．本研究では解析領域内の粒子分布に偏りがある場合を想定し, そのときの接触判定計算を MPI や OpenMP で並列化する際の問題点について, アルゴリズムとマルチコア最適化の観点から検証した．本論文では各 CPU の計算負荷を均一に分散するために格子番号毎ではなく粒子番号毎に接触判定計算を行うことで各 CPU が担当する粒子数を等しくした. また接触判定計算時のループ分割方式としてブロックサイクリック分割を併用して負荷分散を行った. さらにキャッシュの有効利用の為に粒子の並べ替えを行うことで逐次計算時間の高速化と台数効果の向上を図った. 以上の最適化を行う事で OpenMP による 16 スレッド並列で理想的な台数効果が得られた. また MPI 並列化では，メモリに関するスケーラビリティが無いが通信が不要な方法で並列化を行った．1024 プロセス（64 ノード× 16 コア）を使用して並列化した結果，ピュア MPI で 682 倍, ハイブリッド並列で 833 倍の台数効果が得られた.

キーワード：OpenMP 接触判定 粒子解析 MPI

1. はじめに

プラズマや銀河, 粉体などの振舞いは粒子シミュレーションによって表現できる．これらのシミュレーションでは接触している各粒子について, その速度や質量といった情報から粒子間作用力を計算する．そのため各粒子が他のどの粒子と接触しているかを判定する計算（以後, 接触判定計算）が必要となる．しかし全粒子 N 個の組合せについて総当たりで接触判定計算を行うと計算量が $O(N^2)$ となる．つまり粒子シミュレーションにおいて, 全体の計算時間の大きな割合を占める接触判定計算の高速化は重要である．

そのような理由から, 接触判定計算の計算量を減らすために全粒子について総当たりで計算せず, 近傍の粒子についてのみ計算を行うのが一般的な方法である．

近傍の粒子を絞り込むために接触判定格子を用いる方法がある [1]．2.1 節でも述べているが，この方法により接触判定計算を最良のケースで計算量 $O(N)$ で行う事ができる．ただし接触判定格子を用いることで粒子の座標配列に対してのストライドアクセスが発生する. このストライドアクセスが原因となりキャッシュミスヒットが増え，計算時間が増加するという問題があった. この問題を解決するために粒子の並べ替えを行い，キャッシュミスヒットを減らす手法が研究されている [2].

また接触判定格子を用いると接触判定計算に要する時間が粒子分布に依存してしまい, 特に粒子が空間の一部に密集していると並列化したときに高い台数効果が得られないことも課題となっている．

そこで本研究では粒子分布に偏りがある体系での接触判定計算において各 CPU 間の負荷バランスを均一にするため, 格子番号ではなく粒子番号毎に接触判定計算を行う方法を提案し性能評価を行う. またブロックサイクリック分割による負荷分散や粒子の並べ替えによるキャッシュ最適化が, 今回のように粒子分布に偏りがあるケースでの接触判定計算においても有効であるか検討を行う．

2. 接触判定計算

本研究における接触判定計算の手順を**図 1** に示す．

初めに解析領域を分割して，生成した各セルに対して全粒子を登録する．次に粒子番号の並べ替えを行う．最後に接触判定計算を行う．

[1] 法政大学大学院工学研究科
Graduate School of Engineering, Hosei University
[2] 株式会社アールフロー
R-flow Corporation, Ltd.
[3] 東京大学情報基盤センター
Information Technology Center, The University of Tokyo
[a)] sho.takagi.8p@stu.hosei.ac.jp

Vol.2012-ARC-202 No.34
Vol.2012-HPC-137 No.34
2012/12/14

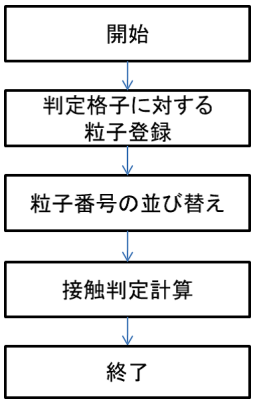

図 1 フローチャート

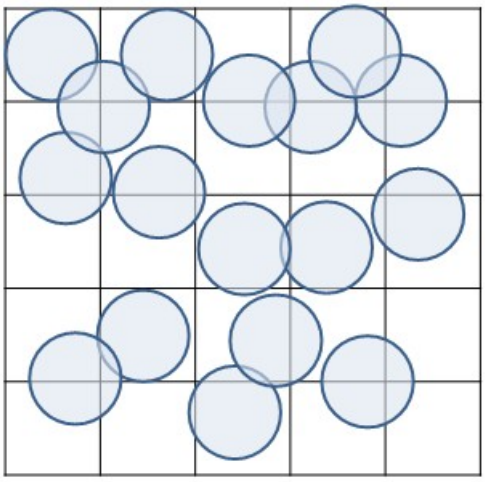

図 2 判定格子のイメージ

2.1 計算領域と粒子登録

本研究では，一辺の長さが 1 の立方体領域を計算の対象とする．この 3 次元領域に任意の数だけ乱数で粒子を発生させ, 各粒子について接触判定計算を行う. なお本研究では，全粒子について総当たりで接触判定計算することを避けるため接触判定格子を用いて接触判定計算を行う．

図 2 のように計算領域を格子状に分割し, 全粒子を一つの判定格子内に登録する. 接触判定格子の格子幅については粒子径と同じにすることにより接触する可能性のある粒子を同一格子内とその隣接格子（3 次元では 27 個の判定格子）内に存在する粒子に限定することができる。このように接触判定格子を用いることで接触判定計算の計算量は $O(N)$ となる．

2.2 2 点間距離の算出と各情報の更新

注目する粒子の番号を i, 相手粒子の番号を j, それぞれの粒子径を d_i, d_j, 粒子の中心座標を $(x_i, y_i, z_i), (x_j, y_j, z_j)$ とすると，以下の式を満たしたとき，i と j の粒子が接触しているみなす。

$$(d_i + d_j) \geq 2\sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2} \quad (1)$$

なお本研究では，粒子径は全粒子共通で D としている．また

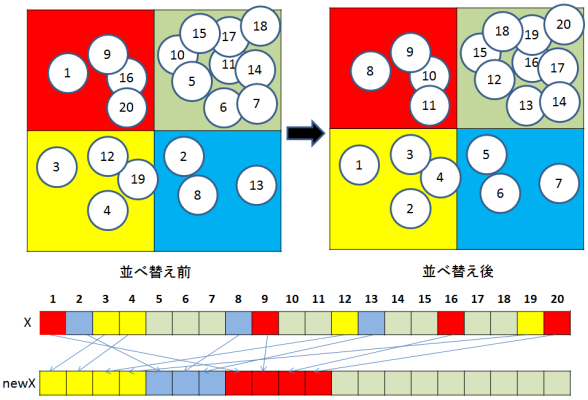

図 3 粒子番号の並べ替えの例

たプログラム内では (1) 式の両辺を 2 乗して (2) 式のようにすることで平方根の演算を減らす工夫をしている。

$$D^2 \geq (x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2 \quad (2)$$

2 粒子が接触していると判定したときは，”他の粒子との総接触数”と”接触している相手粒子の番号”の二つの情報を更新する．これらの計算を各粒子について同様に行う．

2.3 粒子番号の並べ替え

接触判定格子を用いた接触判定計算では, 同一格子内にバラバラな番号の粒子が登録される場合が多い．

例えば**図 3** のように

・粒子番号 3,4,12,19 が黄色のセル
・粒子番号 2,8,13 が水色のセル
・粒子番号 1,9,16,20 が赤色のセル
・粒子番号 5,6,7,10,11,14,15,17,18 が緑色のセル

とそれぞれの粒子が登録されているケースを想定する．

ここで図 3 中の X は, 並べ替え前の粒子の x 座標配列であり,newX は並べ替え後の粒子の x 座標配列である．1 から 20 まであるインデックスは粒子番号を意味している．

黄色の格子内の粒子について接触判定計算を行う場合, 粒子の並べ替えを行っていないと X,Y 座標の配列に対して $3 \rightarrow 4 \rightarrow 12 \rightarrow 19$ という順でアクセスする．空間局所性を考慮すると $1 \rightarrow 2 \rightarrow 3 \rightarrow 4$・・・と順番にアクセスする事が理想であり, このように粒子番号がバラバラな状態で接触判定を行うことはキャッシュミスヒットを引き起こす要因となる. そこで同一格子内の粒子が配列内の近い位置に格納されるよう, 図 3 中の配列 X から配列 newX という形に座標データを並び替える．これを Y 座標についても同様に行う．(3 次元の場合は Z 座標も同様）このようにする事で粒子の座標配列に対するストライドアクセスを避ける事ができ, キャッシュミスヒットの低減が期待される．[2] では粒子が均一に分散しているケースについて, 粒子の並べ替えを行う事により L1 キャッシュミスヒット率を 25%から 2%にまで低減している．

Vol.2012-ARC-202 No.34
Vol.2012-HPC-137 No.34
2012/12/14

3. 負荷分散手法

3.1 格子並列および粒子並列

判定格子を用いて接触判定計算を行う場合, **図 4** のように格子番号毎に並列計算する方法（以後, 格子並列）と, **図 5** のように粒子番号毎に計算する方法（以後, 粒子並列）がある. なお図 4 における G は X,Y,Z 方向の各分割数であり, 図 5 における N は全粒子数である.

単一の格子内に大量の粒子が固まって存在しているケースでの接触判定計算の並列化について考える. 格子並列の場合，担当する格子数は各スレッド間で等しくなるが，各格子に登録されている粒子の数が違うので，各スレッドで担当する粒子数には大きな違いが生じる. それにより計算負荷の軽いスレッドについては，待ち状態が発生してしまう恐れがある.

一方，粒子並列の場合は図 5 のように粒子番号で並列化を行うので，各スレッドが担当する粒子数は一定となる. それにより，格子並列時よりも計算負荷を分散でき, 並列化したときに高い台数効果を期待できる.

```
!$ omp parallel do
   do k=1,G
      do j=1,G
         do i=1,G
            :
         end do
      end do
   end do
!$ omp end parallel do
```

図 4 格子番号による判定法

```
!$ omp parallel do
   do k=1,N


      :


   end do
!$ omp end parallel do
```

図 5 粒子番号による判定法

3.2 ループ分割の方式

粒子番号による並列化を行うことで各 CPU 間での負荷分散が期待される. ただし粒子の並べ替えを行う事で，粒子座標配列の中で同一格子内の粒子が固まって格納される. それによりブロック分割では，各 CPU 間で十分な負荷分散ができなくなる可能性がある. そこで本研究では，ループ分割の方式としてブロック分割以外にブロックサイクリック分割を用いる.

図 6 のように粒子の X 座標配列について，番号 1 から 10 までがスレッド 1, 番号 11 から 20 までがスレッド 2 に割り当てられている場合での接触判定計算を考える. ここでは説明を簡単にするために，隣接格子内の粒子については無視し，同一格子内の粒子についての接触判定計算のみを考える.

まずブロック分割の例では，スレッド 1 は計 12 回（黄:6 回, 水:3 回, 赤:3 回）, スレッド 2 は計 45 回の接触判定計算

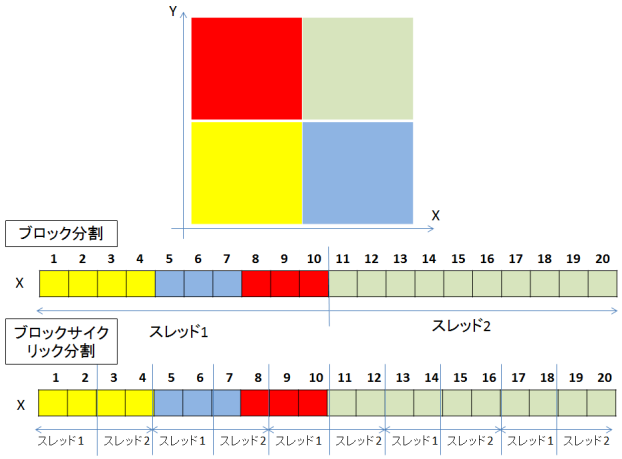

図 6 ブロックサイクリック分割の例（2 スレッド）

が必要となる. ブロック分割では，スレッド 2 の方が計算負荷が重いことがわかる.

次にブロックサイクリック分割の例では, スレッド 1 は計 27 回の計算（黄：5 回, 水：3 回, 赤：1 回, 緑：18 回）, スレッド 2 は計 30 回（黄：1 回, 水：0 回, 赤：2 回, 緑：27 回）計算を行うが，スレッド 1,2 間での計算負荷はほぼ均等となっている. このようにブロックサイクリック分割を用いる事で，スレッド間で計算負荷を分散できる.

チャンクサイズ，ブロックサイズは，あまり小さくするとストライドアクセスが発生してキャッシュミスヒットが増えるが，逆にあまり大きくとり過ぎると負荷分散ができなくなるので，適切な値を決めるのが難しい.

そこで今回は，ブロックサイクリック分割を行う際は, 複数回プログラムを実行して最適なチャンクサイズやブロックサイズを見つけ, その都度値を設定している.

4. 性能評価

4.1 実験環境

実験では，東京大学に 2012 年から導入された富士通 PRIMEHPC FX10(以下 FX10) を用いた．FX10 の仕様を **表 1** に示す.

FX10 は全体で 4800 の計算ノードを持つ. なお今回の実験では，通常使えるキューの設定から，最高で 1440 ノードまでしか使えない. 各計算ノードは SPARC64^{TM}IXfx を搭載しており, 1 ノード内では最大 16 スレッド並列で計算可能である. 4.3 節では，1 ノード内での計算を OpenMP によって並列化し性能評価を行う. また 4.4 節では, ピュア MPI やハイブリッド MPI により，複数のノードを使って並列実行した際の性能評価を行う.

4.2 計算モデル

2.1 節でも述べたように，今回の測定では一辺の長さが 1 の立方体領域を計算の対象とする. この 3 次元領域に任意の数だけ乱数で粒子を発生させる. 粒子を発生させる際は,

表 1　FX10 の仕様

項目	仕様
ノード数	4800
1 ノードあたりの理論演算性能	236.5GFlops
主記憶容量	32GB
CPU	SPARC64^{TM}IXfx 16cores/node 1.848GHz , 14.78GFlops
OS	Red Hat Enterprise Linux Server 6.1
Compiler	Fujitsu Fortran Compiler
Compiler Option	-Kfast,openmp

領域の中心に近づくほど粒子密度が急激に増えるような粒子分布とした。**図 7** のように分布した各粒子について接触判定計算を行い，それに要する計算時間を測定する．なお今回の実験で扱う粒子数 N は 1024000 としており, 図 7 の例はその 1/400 にあたる．

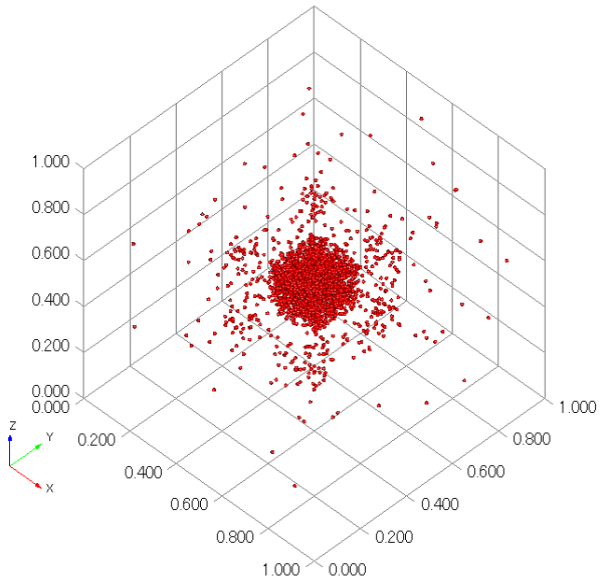

図 7　粒子数 N=2560 のときの粒子分布

4.3　OpenMP による並列化

FX10 で 1 ノードを占有し,OpenMP により最大 16 スレッドで並列実行したときの計算時間や台数効果について性能を評価する．ここで台数効果とは，並列処理の効果を示す一般的な指標である．(3) のように「逐次実行における計算時間」に対する「並列実行における実行時間」の比で表される．

$$\text{台数効果}=\frac{\text{逐次実行における計算時間}}{\text{並列実行における計算時間}} \qquad (3)$$

なお 4.3 節における計算時間とは，接触判定計算に要する計算時間 T_c と粒子の並べ替えに要する計算時間 T_s を合計した総計算時間 T_{ALL} の事を指す．

4.3.1　格子並列および粒子並列時の接触判定計算の性能評価

4.3.1 節では，粒子分布に偏りのあるケースでの粒子並列および格子並列時の計算時間について比較する．なおループ分割は通常のブロック分割で行い, 粒子の並べ替えは行っていない．ここで粒子数 $N = 1024000$，粒子径 $D = 1/200$ としている．格子並列時の結果を**表 2**，粒子並列時の結果を**表 3** に示す．

表 2　粒子並列時の結果（ブロック分割, 並べ替え無）

スレッド数	T_{ALL} (s)	台数効果
1	8.273	-
2	5.204	1.590
4	2.722	3.039
8	1.408	5.876
16	0.75	11.03

表 3　格子並列時の結果（ブロック分割, 並べ替え無）

スレッド数	T_{ALL} (s)	台数効果
1	5.193	-
2	2.6	1.997
4	2.519	2.062
8	2.462	2.109
16	1.612	3.221

表 2 及び表 3 より, 逐次による計算時間は格子並列時の方が速いことがわかる．これは，格子並列時の L1 キャッシュミスヒット率が 34.7%であるのに対し, 粒子並列時は 43.8%と約 9.2% 高いのが原因と考えられる．

また並列数を上げていくと，格子並列時は台数効果が上がらないが, 粒子並列時は理想的とは言えないものの台数効果が上がっている.4 スレッドまでは格子並列の方が速いが，8 スレッド以降では粒子並列の方が速くなっている．

4.3.2　粒子の並べ替えを適用した接触判定計算の性能評価

粒子の並べ替えを行い, 粒子番号で接触判定計算を並列化した結果を**表 4**，格子番号で並列化した結果を**表 5** に示す．なお粒子の並べ替えの有効性について評価を行うため, ループ分割方式は通常のブロック分割で行っている．また 4.3.1 節と同様に粒子数 $N = 1024000$，粒子径 $D = 1/200$ としている．

表 4　粒子並列時の結果（ブロック分割, 並べ替え有）

スレッド数	T_{ALL} (s)	台数効果
1	2.005	-
2	0.971	2.065
4	0.583	3.439
8	0.297	6.751
16	0.151	13.28

表 5 格子並列時の結果（ブロック分割, 並べ替え有）

スレッド数	T_{ALL} (s)	台数効果
1	2.04	-
2	0.996	2.048
4	0.946	2.156
8	0.898	2.272
16	0.575	3.548

表 6 キャッシュミスヒット率の比較

接触判定計算方法		L1 cache miss hit(%)	L2 cache miss hit(%)
粒子並列	並べ替え無	43.89	14
	並べ替え有	4.545	0.414
格子並列	並べ替え無	34.7	3.413
	並べ替え有	3.982	0.375

初めに逐次での計算時間について,4.3.1 節の並べ替えを行わないケースでは格子並列の方が粒子並列よりも速い結果となっていた. しかし粒子の並べ替えを行う事で，逐次での計算時間はほぼ等しくなっている. 特に粒子の並べ替えの効果は粒子並列時のときに大きく，並び替えの有, 無で 6 秒以上の差が生じている.

次に台数効果について，格子並列時は並べ替えを行った場合での台数効果が 16 スレッドで約 3.5 倍となり，並び替えを行った事による台数効果の向上は見られなかった. 一方粒子並列時に並べ替えを行った場合では，16 スレッドでの台数効果は 13.28 倍となり，向上が見られた.

並べ替えを行う前と後でのキャッシュミスヒットの違いを見てみると，並べ替え前は L1 キャッシュミスヒット率が 43.8%であったのに対し並べ替え後は 4.54%まで減っている．同様に L2 キャッシュミスヒット率も並べ替え前は 14%であったのに対し，並べ替え後は 0.414%まで減っている.

4.3.3 ループ分割方式変更時の接触判定計算の性能評価

粒子並列時のループ分割方式として，ブロックサイクリック分割を用いた場合の結果を**表 7** に示す. なお粒子数 $N = 1024000$，粒子径 $D = 1/200$ としており，ブロックサイクリック分割でのチャンクサイズは 200 としている.

表 7 粒子並列時の結果（ブロックサイクリック分割, 並べ替え有）

スレッド数	T_{ALL} (s)	台数効果
1	2.005	-
2	0.958	2.093
4	0.481	4.168
8	0.242	8.285
16	0.124	16.17

ブロックサイクリック分割を行う事で，16 スレッドで 16.17 倍というスーパーリニアな台数効果が得られている.

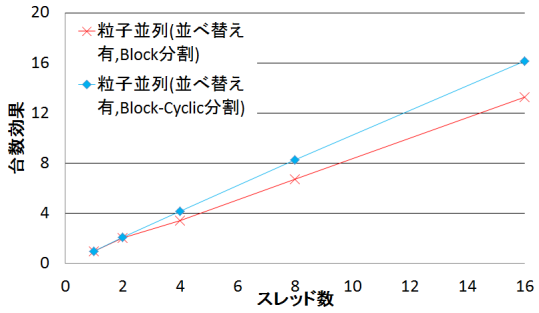

図 8 ループ分割方式変更時の台数効果の比較

4.4 MPI による並列化

MPI 並列化ではメモリを節約するために各ノードにデータを分散するのが一般的である. しかし本研究のプログラムで扱うデータ量は各ノードに分散しなければならないほど膨大ではないので, 全ノードに共通の粒子座標データを与えることとした. この観点で分散メモリ環境を考慮しておらず通信が発生しないプログラムとなっている.

また粒子の並び替え部分について MPI 並列化を実装していないため，4.4 節では並べ替えに要する計算時間は無視して接触判定に要する計算時間のみで性能評価を行う. MPI による並列化は，ピュア MPI（ノード内，ノード間ともに MPI）によるものと，ハイブリッド MPI（ノード間 MPI, ノード内 OpenMP）によるものの 2 パターンで行う.

4.4.1 測定結果

粒子分布に偏りがある体系での接触判定計算に対し, 粒子の並べ替え及びブロックサイクリック分割を適用し粒子番号で並列化したときの台数効果と並列化効率を**図 9** および **図 10** に示す. ここで並列化効率とは，台数効果をプロセス数で割った値である.

$$\text{並列化効率} = \frac{\text{台数効果}}{\text{使用したプロセス数}} \tag{4}$$

なお粒子数 N=1024000，粒子径 D=1/200，ブロックサイクリック分割したときのブロックサイズは全て 200 である．使用した最大ノード数は 64 ノードである. またハイブリッド並列では，1 ノード内を全て OpenMP により 16 スレッドで並列化している.

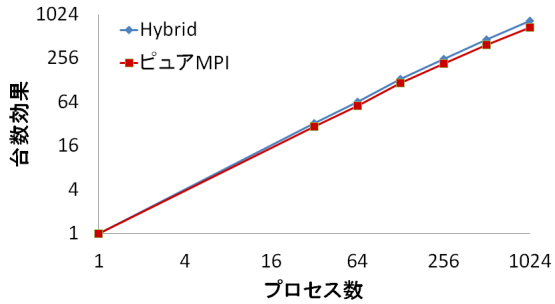

図 9 台数効果（ハイブリッド MPI 及びピュア MPI）

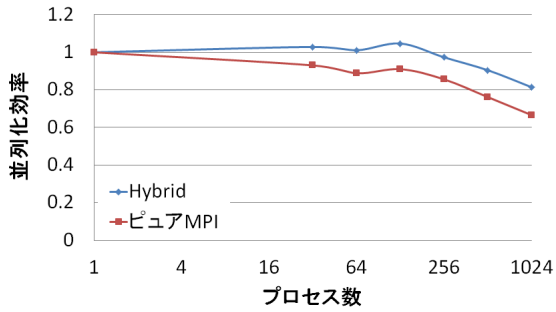

図 10 並列化効率（ハイブリッド MPI 及びピュア MPI）

1024 プロセスを用いて並列化したときの台数効果は, ハイブリッド並列で 833 倍, ピュア MPI で 682 倍であった. 図 9 および図 10 の結果のとおりハイブリッド並列の方がピュア MPI 並列よりも高い台数効果が得られている.

5. 関連研究

今回用いた判定格子ベースの接触判定手法は，MPS 法や SPH 法といった流体シミュレーションや離散要素法 (DEM) などの粉体シミュレーションでよく用いられている．[3] では，粒子の並べ替え（リナンバリング）を行っていない場合の台数効果は 1024 コア (64 ノード× 16 コア) で数十倍にとどまっている．本研究では通信を考慮していないため単純な比較はできないがピュア MPI で 682 倍, ハイブリッド並列で 833 倍まで台数効果が出ており十分な並列化が行えていると考えられる.

また GPU 上での粒子シミュレーションに関する研究 [4] も行われている．[4] では，計算領域内に粒子が一様に存在する場合で最も性能が良く，粒子が 1 つのプロセッサに固まってしまう場合には，並列化性能が上がらないとしている．[4] で行われている計算は，近傍粒子探索だけではなく力の計算なども含まれているが, 多くの粒子シミュレーションでは近傍粒子探索が全体の計算時間の大きな割合を占めるとされている. 従って今回粒子分布に偏りがある体系での接触判定計算について, 特に共有メモリ環境においてリニアな台数効果を達成できたことは，粒子シミュレーション全体の計算時間の短縮にもつながると考えられる.

6. まとめ

本研究では，解析領域内の粒子分布に偏りがあるときの接触判定計算を MPI や OpenMP を用いて並列化した.

初めに，FX10 の 1 ノードを占有し OpenMP により最大 16 スレッドで並列実行したケースでは, 逐次計算と比べたときの台数効果は格子並列で最大約 3.6 倍, 粒子並列で最大約 16.2 倍となった. この結果から共有メモリ環境において粒子分布に偏りがある体系では, 格子番号による並列化よりも粒子番号による並列化の方が負荷分散の観点で有効と言える.

粒子の並べ替えについては今回のように粒子分布に偏りがある体系でも逐次計算時間の高速化や台数効果の向上という点で有効であった.

またブロック分割よりもブロックサイクリック分割でループを分割する方が計算時間が速く, 高い台数効果が得られた. ただしブロックサイクリック分割で設定するチャンクサイズの大きさは，その都度最適な値をプログラム上で設定しているので, 今後は最適な値を計算によって求められるようにする必要がある.

次にピュア MPI による並列化について, 粒子並列, 粒子の並べ替え, ブロックサイクリック分割の全てを適用することで 128 プロセス並列まではほぼリニアな台数効果が得られた. 一方ハイブリッドによる並列化では 256 プロセスまで理想的な台数効果が得られた.256 プロセス以降についてもピュア MPI より高い台数効果が得られた.

今回の問題サイズでは全てのノードに同じデータを持たせることでノード間の通信が発生しないプログラムを実装できた. しかし更に大きい問題サイズで計算を行うためにはメモリが不足する事が予想されるので分散メモリを考慮したデータ構造とする必要がある. そうした通信が発生するプログラムとなった場合にブロックサイクリック分割を用いて並列化すると通信が多発する事が予測される．分散メモリ環境を想定したときにどこまで高い台数効果が得られるかは今後の課題である.

参考文献

[1] Yusuke Shigeto & Mikio Sakai : Parallel Computing in Computational Granular Dynamics by Using Multi-Core Processors -Practical Usage of the DEM in Complicated Powder Systems in Industries-, J. Soc. Powder Technol, Japan, 47, pp.707-716 (2010)
[2] 和田直樹, 高木翔, 岡大樹, 竹田宏, 片桐孝洋, 堀端康善: 粒子接触判定計算の OpenMP による最適化，情報処理学会研究報告，Vol.2012-HPC-136, No.3(2012)
[3] 樫山和男，牛島省，寺田賢二郎，岡澤重信，木村一郎，中畑和之，浅井光輝，松本純一，岩下武史，小山田耕二:マルチフィジックスおよび最適化問題に向けたハイパフォーマンス計算力学, 学際大規模情報基盤共同利用・共同研究拠点，平成 23 年度共同研究中間報告書 (2012)
[4] 原田隆宏, 政家一誠, 越塚誠一, 河口洋一郎:複数の GPU を用いた粒子法シミュレーションの並列化，情報処理学会論文誌，Vol.49，pp.4067-4079 (2008)
[5] Peter S. Pacheco(原著), 秋葉 博 (翻訳) : MPI 並列プログラミング (2001)
[6] 牛島 省 : OpenMP による並列プログラミングと数値計算法，丸善株式会社（2006）