Vol.2012-HPC-136 No.3
2012/10/3

粒子接触判定計算のOpenMPによる最適化

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

概要：粒子解析においては粒子同士の接触判定計算が最も計算負荷が大きく，接触判定計算時間の短縮が必須の課題である．本稿では，接触判定計算にOpenMP並列化を用いて，16スレッドまでの並列計算による高速化を試みた．本稿で行った高速化技法は，粒子番号の並び替え，ループ融合，および座標配列のアクセス性の改善である．性能評価の結果，16スレッド時の台数効果が最適化前は11倍であったが，最適化後は14倍まで性能向上ができた．

キーワード：OpenMP，接触判定，粒子解析

1. はじめに

粉体計算や荷電粒子のシミュレーション，銀河の衝突計算など粒子解析の分野では，対象とする問題ごとにさまざまな運動方程式を立てて計算を行う．その際に，どの問題に対しても共通して行われる処理は粒子の接触判定計算である．粒子の接触判定計算時間が，処理時間の大部分を占めると言われている．その理由としては，接触判定以外の計算は粒子数に比例する計算量となるが，接触判定計算は粒子径の大きさ次第で粒子数の2乗に比例する計算量となるためである．したがって，粒子解析を行う上で接触判定計算は最も計算負荷がかかる問題であり，その高速化は必須の課題となっている．

本研究では，高速化のためにOpenMP並列化による16スレッドまでの並列計算を行い，理想値に近い並列化効率を出すことを目標とする．しかし単純に並列化しただけでは並列数が上がるにしたがい並列化効率が低下し，理想の並列性能を出すことは難しい．この要因は，計算時のキャッシュミスヒット率が高いことが挙げられる．

そこで本稿では，粒子番号の並び替え，ループ融合，および，座標配列のアクセス性改善の最適化を行い，キャッシュミスヒット率を削減する最適化手法を提案することを目的とする．また，並列数を上げたときの接触判定計算時間について，どこまで理想値に近づけることが可能か見極めることを目的にする．

[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)] naoki.wada.9m@stu.hosei.ac.jp

2. 対象とする問題

2.1 領域と接触判定格子

一辺の長さが1の立方体が対象の領域であり，領域内部に存在する粒子に対して接触判定計算を行う．粒子の位置情報は，乱数発生させた外部のデータを読み込んで使用する．計算機を変更して性能評価する際も，同じ粒子位置情報を用いて行うことを想定する．

接触判定計算は全ての粒子に対して行う必要があり，単純な接触判定計算では一つの粒子に対して残り全ての粒子座標との距離を比較する方法がある．しかしこの方法では，明らかに接触していない粒子に対しても全て接触判定する必要があるため，粒子数が多くなると接触判定計算に時間を要する．

そこで本稿では，粒子解析の分野で一般的に用いられる接触判定格子を使用することで接触判定計算を行った．接触判定格子のイメージを**図 1** に示す．

図1では，対象とする立方体領域を小さい格子状に分割することで，全ての粒子を必ず一つの小格子に格納する方法である．格子の幅を粒子径の大きさと同じになるように設定すると，接触している可能性のある粒子は，同一格子内または隣接した格子内だけとなるため，判定計算を少なくすることができる．

2.2 リスト構造による粒子番号登録

接触判定格子を使用して領域内の各格子に粒子を格納する場合，複数の粒子が一つの格子内に格納されることがある．全格子内に最大で何個の粒子が存在するかという情報

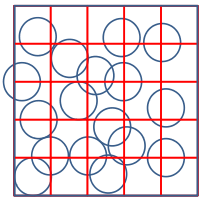
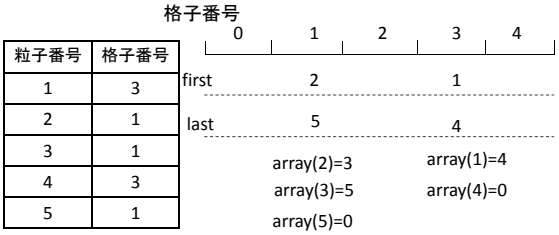


図 1 接触判定格子イメージ



粒子番号	格子番号
1	3
2	1
3	1
4	3
5	1

図 2 1 次元領域，粒子数 5 の場合のリスト構造例

が分かれば，最大の要素数分だけの大きさを確保した配列を各格子に用意して格納すればよい．しかし，全ての格子に最大分の粒子が入るということは一般的に無く，メモリ使用量が無駄に大きくなることがある．

そこで，粒子登録方法としてリスト構造 [1] による登録を用いた．これは格子内に最初に登録された粒子番号を first，最後に登録された粒子番号を last として，first から last まで順に辿ることで格子内に格納された全ての粒子を読み出せるデータ構造である．

使用する配列は，格子数分の大きさを確保した first と last，それらを辿るための配列 array の 3 つだけとなり，前者の方法よりメモリ使用量は少なくて済む．**図 2** は，長さ 1 の 1 次元領域内に 5 つの粒子が存在する場合のリスト構造についての例となっている．今回はこの構造を用いて，各格子内に格納した粒子を順に呼び出して接触判定計算を行う．

2.3 接触判定と接触情報

接触判定は，注目している格子と，そこから隣接している格子内に格納されている粒子の粒子中心間距離を見て行う．具体的には，「ある一つの粒子と別の一つの粒子との粒子中心間距離が粒子径以下になった場合」を接触とみなす．

実際の粒子解析においては，接触判定後の粒子情報を用いて別の計算を行うため，ここでは「接触している粒子数」「接触した相手の粒子番号」という情報を各粒子が持っているものと仮定し，接触とみなされた場合にはこれら 2 つの情報を更新する．

3. 最適化手法

3.1 粒子番号の並び替えによるキャッシュミスヒット率の削減

全ての粒子はリスト構造を用いて一つの判定格子に登録し，判定時に粒子番号が呼び出されることで粒子同士の接触判定計算が行われる．しかし，登録される格子を決める元となっている粒子座標は乱数で決定されるため，格子内の粒子番号は連続にならない．この状態で判定計算を行うと，粒子情報を取得するための配列アクセスに規則性が無い（連続アクセスでない）ことから，キャッシュメモリを有効に利用できない．その結果，判定を行う毎にメモリからのデータ読み込みが発生し，計算時間を増加させる．

そこで，接触判定計算に入る前に再度リストを呼び出し，リスト内に登録されている粒子番号の振り直しを行う処理を導入する．全ての格子に対してこの処理を行うと，領域内の粒子は昇順の番号に並び替えられて，接触判定時に周囲の格子内の粒子情報がキャッシュメモリに登録される．その結果，メモリからの粒子情報の読み込みが減ることによる高速化が期待できる．

3.2 ループ融合

判定は各格子を走査して格子内の粒子を呼び出すことによって行う．対象とする領域は 3 次元であり，単純に行うならば 3 次元配列を用いての 3 重ループ計算となる．この場合，判定計算を行う箇所は 3 重ループ内であることから，OpenMP の do 指示文をループに挟む形で挿入することとなる．

この do 指示文は，記述した直後にあるループ演算のみが並列化されるが，3 重ループ中の 1 ループの長さではループ長が短いために，高い並列化効率が得られない可能性がある．

そこで**図 3** の 3 重ループ以外に，**図 4** のように 2 重ループに書き換えたプログラム（最外および第 2 ループのループ融合をしたプログラム）を用意した．3 重ループでは並列化されるループ長は G であるのに対し，2 重ループではループ長は G^2 となる．この書き換えによって並列化の効率がどのように変わるか検討する．

3.3 座標配列の連続アクセス性の改善

判定を行う際の粒子座標は，x，y，z について乱数発生させた外部データを読み込み，粒子数を N とした場合 x(N)，y(N)，z(N) という 3 つの 1 次元配列で扱われる．この座標データは，粒子登録や判定計算といった箇所で頻繁に呼び出され，また x,y,z の座標は毎回ほぼ同時に呼び出されるという性質を持つ．

ここでは 1 次元配列で扱っていた座標配列を，**表 1** のよ

```
!$ 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 3 ループ融合前

```
!$ omp parallel do
do jk=1,G*G
  k=int(((jk-1)/G)+1)
  j=jk-(k-1)*G
  do i=1,G
  :
  end do
end do
!$ omp end parallel do
```

図 4 ループ融合後

うに 1 つの 2 次元配列で扱う形に書き換えた．このようにすることで，x,y,z の座標アドレスにアクセスする際に連続性を持つことができるため，キャッシュメモリの利用効率が向上して読み込みにかかる時間の短縮が期待される．今回は，座標配列に 1 次元配列を用いた場合と 2 次元配列に書き換えた場合の比較を行うことで，どのような変化が生じるか確認する．

表 1 座標配列の変更例

変更前	変更後
Ax(N)	A(0,N)
Ay(N)	A(1,N)
Az(N)	A(2,N)

4. 性能評価

4.1 計測環境・初期設定

実験は東京大学に 2012 年に導入された FX10 スーパーコンピュータシステム（富士通 PRIMEHPC FX10）の 1 ノードを用いて行った．1 ノード内のコア数は 16 であるため，OpenMP による並列化は最大の 16 スレッドまで行った．

表 2 FX10 構成

項目		仕様
ノード	理論演算性能	236.5GFlops
	プロセッサ数 (コア数)	16
	主記憶容量	32GB
プロセッサ	プロセッサ名	SPARC64^{TM}IXfx
	周波数	1.848GHz
	理論演算性能 (コア)	14.78GFlops
ソフトウェア	OS	Red Hat Enterprise Linux Server 6.1
	コンパイラ	Fujitsu Fortran Compiler
	コンパイラオプション	-Kfast,parallel,openmp

今回の問題では，粒子径は 1/200，粒子数は 800 万 (200×200×200=8,000,000) に設定した．また判定時に各粒子は接触数の情報を更新するが，1 粒子あたりの接触数情報の上限は 100 とする．

表 3 ケースによるプログラムの違い

ケース番号	粒子番号並び替え	ループ融合	座標配列変更
case1	×	×	×
case2	○	×	×
case3	×	○	×
case4	×	×	○
case5	○	○	○

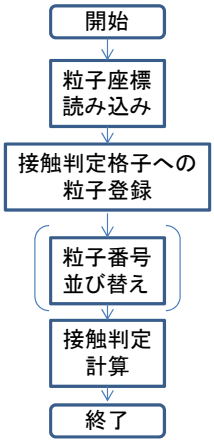


図 5 接触判定計算の流れ

4.2 測定結果

個別に行った方法での効果を見るために**表 3** のように 5 つのプログラムを用意した．具体的には，最適化を行っていないプログラムを基準の case1 として用意し，それに対してそれぞれ異なる最適化手法で記述した 3 種のプログラムを case2 から case4，また 3 つの方法を全て加えたプログラムを case5 として，接触判定計算時間及び台数効果（スピードアップ）[2] の比較を行う．スピードアップとは，並列計算の効果を表現する一般的な指標であり，(1) より求めたものを使用する．

$$S = \frac{\text{逐次実行に要する計算時間}}{\text{並列計算に要する計算時間}} \quad (1)$$

処理の一連の流れは**図 5** のようになっている．接触判定計算の時間とは，この図中「接触判定計算」の実行時間を指す．なお，最適化手法で粒子番号の並び替えを行ったものに対しては，接触判定計算時間は「粒子番号並び替え時間＋接触判定計算時間」で求めたものとする．これは接触判定計算の時間に対して並び替えにかかる時間が大きい割合を占めることがあり，この時間を無視することができなかったためである．

4.2.1 接触判定計算時間比較

図 6 に，各ケース別に比較を行った際の並列計算時間を示す．case3 と case4 に関しては，何も書き換えを行っていない case1 の計算時間とほぼ変わらない結果が得られた．粒子径の並び替えを行った case2 と case5 に関しては，どのスレッド数でも計算時間が短縮されていることがわ

かった．

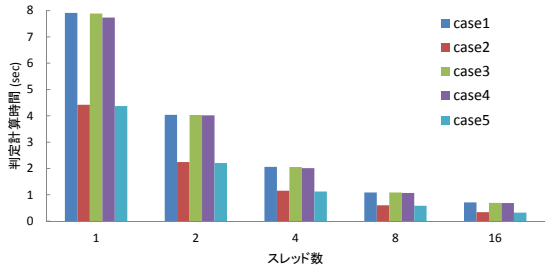


図 6 各ケースについての接触判定計算時間

4.2.2 スピードアップ比較

図 7 から**図 10** に，各ケース別のスピードアップの違いを示した．いずれの場合も 8 スレッドまでは理想値に近い値を取っているが，8 スレッド以上の並列数になるとき，スピードアップが悪くなることがわかる．

特に case3,case4 については，16 スレッド時の場合で 11 倍前後のスピードアップに留まっていることが確認できる．それに対し，case2 については 13 倍，case4 については 14 倍程度までスピードアップが向上している．したがって，粒子番号の並び替えの手法は，スピードアップ向上に関し効果的である．

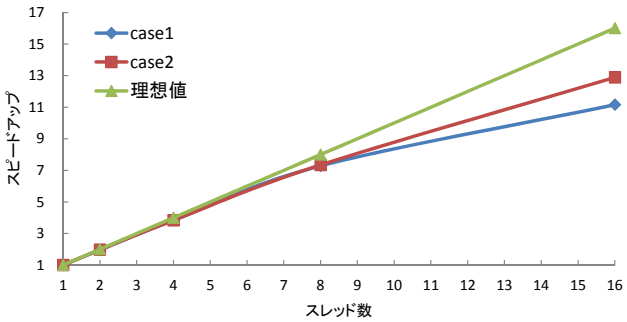


図 7 case2 でのスピードアップの変化

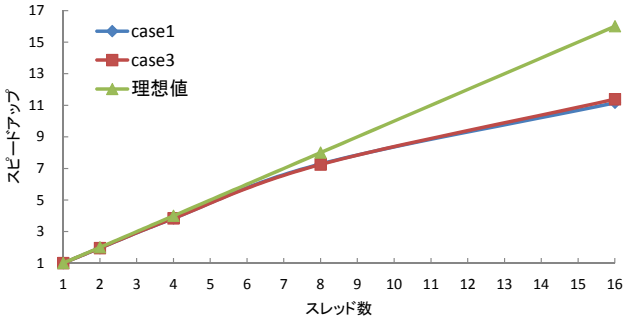


図 8 case3 のスピードアップの変化

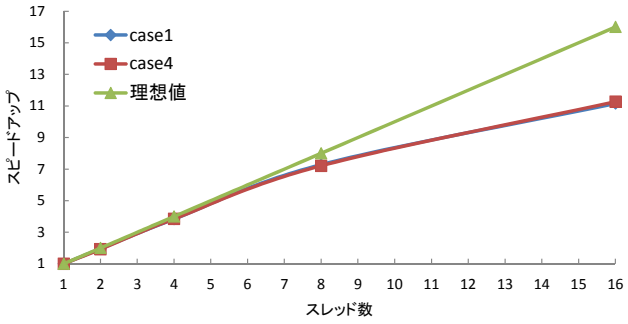


図 9 case4 でのスピードアップの変化

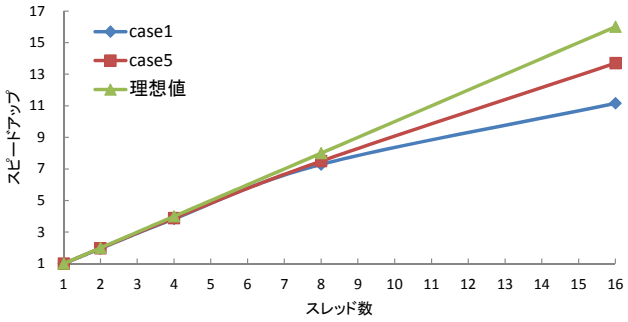


図 10 case5 でのスピードアップの変化

4.2.3 パフォーマンスモニタによるキャッシュミスヒット率の解析

ここでは，東京大学情報基盤センターの FX10 で提供されているプログラミング支援ツールの FUJITSU Software Development Tools Version 1.2.1 for Windows を利用して，キャッシュミスヒット率の解析を行う．

図 11 は，各ケースでの接触判定計算時の L1 キャッシュミスヒット率と L2 キャッシュミスヒット率を表したものである．最適化していない case1 や，ループ融合を行った case3 では，L1 キャッシュミスヒット率が 25%程度である．これに対し，case2 や case5 では，2%前後まで L1 キャッシュミスヒット率が低下している．また，座標配列の変更を行った際にもキャッシュミスヒット率の低下が確認できる．

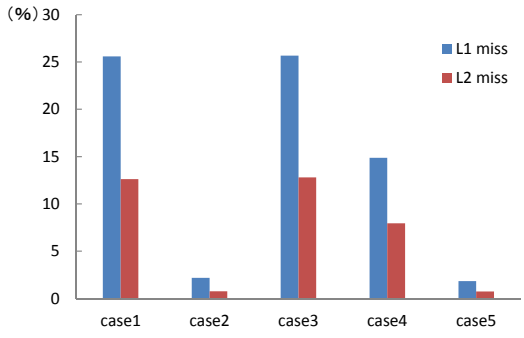


図 11 各ケースでの L1 キャッシュミスヒット率と L2 キャッシュミスヒット率

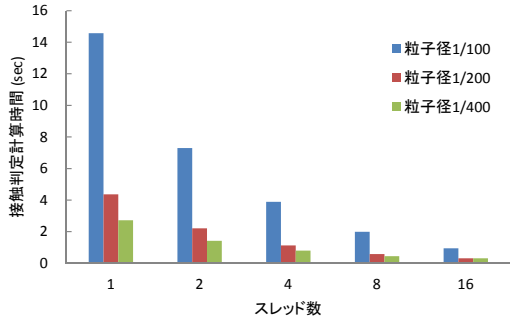


図 12 粒子径変更時の case5 実行についての接触判定計算時間

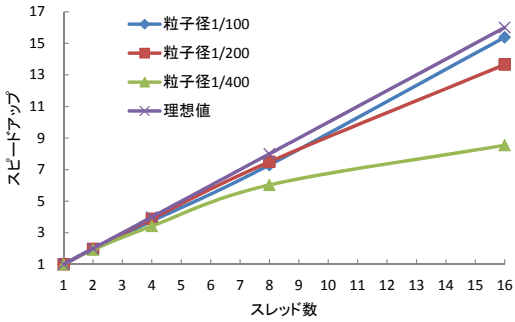


図 13 粒子径変更時の case5 実行についてのスピードアップ変化

4.2.4 接触数と接触判定計算時間の関係

ここまでの比較は，粒子径 1/200 で固定して行った．この粒子径を変えることにより，1 粒子あたりの平均接触数が変化して，接触判定にかかる時間が変わる．**表 4** は粒子径による総接触数の変化を表している．

ここでは，粒子接触数が変わることによる並列化効率の変化を確認するため，粒子径を変更した時の比較を行った．粒子数は 800 万として，粒子径は 1/200 を基準に，2 倍の大きさの 1/100 と，半分の大きさの 1/400 の場合を評価する。最適化は case5 のものを用いた．

表 4 粒子数 800 万のとき総接触数と 1 粒子あたりの平均接触数

粒子径	総接触数	1 粒子あたりの平均接触数
1/100	265109534	33.13
1/200	33321894	4.165
1/400	4174666	0.521

図 12 は，粒子径変更時の接触判定計算時間の変化である．この図では，粒子径が大きい順に接触判定計算に時間を要することがわかる．これは粒子径が大きいほど接触数が多くなり，接触判定時間が増えたことによる．

図 13 は粒子径変更時のスピードアップ変化である．この図からは，粒子径が大きいほどスピードアップが理想値に近づき，逆に粒子径が小さい場合はスピードアップが極端に悪くなることが確認できる．

この結果から，粒子径が大きい 1 粒子あたりの接触数が多いほど接触判定計算には時間がかかり，スピードアップは理想値に近づくことがわかる．

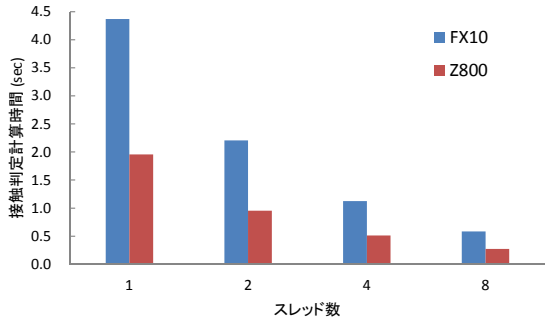


図 14 FX10 と Z800 での接触判定計算時間比較

4.3 異なる計算機での比較

4.3.1 計測環境・初期設定

各最適化手法の比較は FX10 上で行ったが，ここでは比較検討のため，FX10 とは異なる計算機上で性能評価した結果を示す．

ここで使用した計算機は，HP 社製 Z800 Workstation（以下，Z800 とする）である．Z800 の仕様は**表 5** に示した．

表 5 Z800 構成

項目		仕様
ハードウェア	プロセッサ名	Intel® Xeon® Processor X5677 ×2
	周波数	3.46GHz
	コア数	4
	主記憶容量	48GHz
ソフトウェア	OS	Red Hat Enterprise Linux 5.6
	コンパイラ	Intel® Fortran Compiler XE 12.1
	コンパイラオプション	-fast -openmp

表 5 の Z800 を使用して，FX10 で行った case5 についての比較を行う．なお Z800 では 8 並列まで可能であるため，この比較を行う際の OpenMP による並列数は 8 スレッドを上限とした．

4.3.2 接触判定計算時間比較

Z800 と FX10 との接触判定計算時間の比較を行った結果を**図 14** に示す．Z800 で実行した場合についても FX10 での結果と同様に，接触判定計算時間は各スレッドごとに約 1/2 ずつ減少することが確認できる．またこの結果では，どのスレッド数の場合でも FX10 に対して Z800 の実行時間が少ないことが確認できる．この理由として，整数演算性能などハードウェア構成の違いによるものと推察されるが，詳細解析は今後の予定である．

4.3.3 スピードアップ比較

図 15 に，case5 実行時のスピードアップの変化を示した．8 スレッド実行での比較であるため FX10 では理想値

に近いということは確認済みであったが，Z800 の場合でも同様に，スピードアップは理想値に近くなることがこの結果よりわかった．

これらの結果から，他の計算機で実行を行った場合では，接触判定計算時間に差が生じることはあるが，スピードアップに関しては同様の効率が得られるものと考えられる．

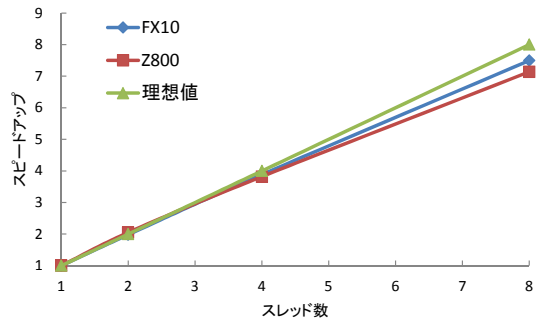


図 15 FX10 と Z800 でのスピードアップ比較

5. まとめ・課題

本稿で行った最適化方法は，粒子番号の並び替え，ループ融合，座標配列の変更である．

この中でスピードアップ変化が最も大きかったものは，粒子番号の並び替えであった．並び替え前のプログラムでは，16 スレッド並列時で 11 倍程度のスピードアップしか得られなかったが，並び替え後のプログラムでは 13 倍までスピードアップの向上が確認できた．

座標配列の変更については個別の方法では接触判定計算時間やスピードアップに大きな変化は見られなかった．しかし，接触判定計算時のキャッシュミスヒット率が低下したことや，他の最適化方法全てと組み合わせた場合の結果においてスピードアップが 14 倍まで上がったことから，最適化の効果はあるものと考えられる．

粒子径を変更させることで 1 粒子あたりの平均接触数を変えた場合のスピードアップの変化については，接触数が多いほど接触判定計算時間は増加するがスピードアップは理想値に近くなり，逆に接触数が少ないほど接触判定計算時間は少なくなるがスピードアップは理想値から遠くなる結果が得られた．このことから，並列化性能の変化は最適化だけでなく粒子の接触数にも影響されることが確認できた．

計算機構成の違いによる並列性能確認について，Z800 の 8 スレッドまでのスピードアップにおいて，理想値に近い値を得ることができた．これは，FX10 での測定結果とも近い値であることから，今回の実験条件では，ほぼ同様の並列性能が達成できる．しかし，FX10 でのスピードアップ比較である図 7 から図 10 を確認すると，8 スレッド以降でスピードアップが悪くなることも考えられる．したがって，計算機構成の違いによる並列性能については，より多くの並列実行ができる計算機で調査する必要がある．

今回の評価では，ループ融合の効果があまり得られていない．この要因の一つとしては，16 並列という並列数が低いものであるために，ループ融合による速度向上の効果が弱いことが考えられる．今後の課題として，MPI 並列化を行うことで 16 並列以上の高並列計算を行い，今回行った最適化の効率がどの程度まで向上するのか調査する予定である．

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, 707-716(2010)
[2] 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）