SMP-PC クラスタにおける SPAM 粒子シミュレーションのハイブリッド並列化

吉川茂洋[†1,☆] 朴泰祐[†2] 佐藤三久[†2]
高橋大介[†2] Carol G. Hoover[†3] William G. Hoover[†4]

SMP クラスタにおいて，メッセージパッシングと共有メモリプログラミングを組み合わせたハイブリッド並列化により，メッセージパッシングのみ用いた場合を上回る性能が得られた事例は少ない．この原因を解明するため，SPAM（Smoothed Particle Applied Mechanics）粒子シミュレーションを MPI と OpenMP によりハイブリッド並列化し，SMP-PC クラスタ上で性能評価した．その結果，MPI のみのほうがハイブリッドの性能を上回る結果となった．性能解析により，ハイブリッドにおける L2 キャッシュミス率の増加が性能低下の原因であることが示された．また，この種の問題に OpenMP を適用した場合の問題点についても考察する．

Hybrid Parallelization for SPAM Particle Simulation on SMP-PC Clusters

Shigehiro Yoshikawa,[†1] Taisuke Boku,[†2] Mitsuhisa Sato,[†2]
Daisuke Takahashi,[†2] Carol G. Hoover[†3]
and William G. Hoover[†4]

There are few cases that hybrid parallelization using message-passing and shared-memory models simultaneously outperformed pure message-passing model on SMP clusters. To study this behavior, we evaluate our MPI+OpenMP hybrid model for the SPAM (Smoothed Particle Applied Mechanics) particle simulation on an SMP-PC cluster. As a result, pure MPI model outperformed hybrid one. Our performance analysis showed that low performance of our hybrid model was caused by high L2-cache miss-hit ratio. Also, we discuss issues of OpenMP on such a problem.

1. はじめに

近年，HPC 向けの多くのハイエンドマシンでは，複数のプロセッサを共有メモリ結合した SMP を各計算ノードに用いている．さらに，SMP 構成の PC も比較的安価に入手できるため，SMP-PC による大規模クラスタも多数構築されている[1)]．この傾向は今後も続くことが予想され，本稿でも SMP-PC クラスタをターゲットプラットフォームとした．一般に SMP クラスタでは，従来行われてきた MPI のみ用いたメッセージパッシングによる並列化が可能である．しかし，SMP ノード上では共有メモリプログラミング，ノード間はメッセージパッシングを行うハイブリッド並列化は，MPI のみに比べ，以下の点で有利なはずである．

- **ノード内通信**：共有メモリへの read，write で通信を行うため，関数呼出し，データコピーによる余分なオーバヘッドがない．
- **ノード間通信**：メッセージパッシングによる総通信回数が減るため，全体全通信等のオーバヘッドが小さい．
- **動的負荷分散**：共有メモリのアルゴリズムにより，低オーバヘッドで細粒度な負荷分散が可能．

しかし，このような定性的な予測に反し，ハイブリッド並列化を行っても MPI のみの場合を上回る性能が得られないという研究結果が報告されている[2),3)]．本

†1 筑波大学システム情報工学研究科
Graduate School of Systems and Information Engineering, University of Tsukuba
†2 筑波大学電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†3 Lawrence Livermore National Laboratory
†4 Department of Applied Science, University of California
☆ 現在，富士通株式会社
Presently with Fujitsu Limited

稿ではこの原因を解明するため，実機と実問題を用いた性能評価を行う．ターゲットアプリケーションには，ハイブリッド並列化が定性的に有効と考えられる，負荷分散が不可欠な問題として，SPAM 粒子シミュレーションを用いる．

SPAM（Smoothed Particle Applid Mechanics）は連続体の解析に有用な粒子シミュレーション手法である．基本的な概念は天文物理学等で用いられる SPH（Smoothed Particle Hydrodynamics）[4),5)]と同じであり，粒子の存在する点で物理量を代表し，その運動を追跡することにより解析を行う．SPAM では粒子間の相互作用に距離に応じて急速に変化するポテンシャルエネルギを用いており，作用力の計算には，分子動力学法シミュレーションでよく用いられるカットオフの手法を適用することができる．

本稿では 2 次元の衝撃波の解析問題に対し SPAM を適用する．このようなカットオフを用いる粒子シミュレーションを並列化する場合，粒子分割法（particle decomposition）と空間分割法（space decomposition）の 2 種類がよく用いられる[6),7)]．我々の問題では，問題空間がカットオフの半径に比べ非常に大きいので，空間分割法を採用した．空間分割法では，空間を固定サイズの部分空間（セル）に分割し，セルの一辺の長さをカットオフの半径に等しいかあるいは大きく設定する．これにより，相互作用計算時の通信を大幅に限定することができる．我々の問題ではシミュレーション過程でセル内の粒子の密度が大きく変化するため，プロセッサ間での計算の負荷分散が重要となる．

以下，2 章で並列 SPAM 粒子シミュレーションの概要を述べた後，3 章でハイブリッド並列化で用いたプログラミング手法について述べる．4 章では実機上での性能評価を示し，5 章でまとめを行う．

2. 並列 SPAM 粒子シミュレーション

2.1 概 要

2 次元の衝撃波解析に SPAM を適用する．粒子 i，j 間のポテンシャルエネルギ wp，および粒子 i に働く力 f_x，f_y は次の式で表される．

$$wp(r_{ij}) = -\frac{\beta}{\pi R^3}\frac{r_{ij}}{R}\left(1-\frac{r_{ij}}{R}\right)^2 \quad (1)$$

$$f_x(i) = -\sum_{j\neq i} wp(r_{ij})\frac{x_i - x_j}{r_{ij}} \quad (2)$$

$$f_y(i) = -\sum_{j\neq i} wp(r_{ij})\frac{y_i - y_j}{r_{ij}} \quad (3)$$

β は物性に基づく定数，π は円周率，x_i，x_j，y_i，y_j は粒子 i，j の 2 次元平面上の座標，r_{ij} は粒子 i と粒子 j 間の距離，R はカットオフ半径を表す．粒子に働く力の計算は粒子 j がカットオフ半径 R 内に存在する場合にのみ行われる．力の計算を基に，シミュレーションでは適切な時間刻み幅 Δt を用いて，粒子の位置と速度を時間発展的に計算する．この時間積分における誤差を減らすため 4 次のルンゲクッタ（Runge-Kutta）法を適用する．

シミュレーションは，次のような手順で行われる．

(1) シミュレーションの初期条件（粒子の初期位置・速度等）を設定する．
(2) 粒子をセルに分散する．
(3) プロセスにセルをマッピングする．
(4) セル内の粒子 i と，同一セル内の他の粒子，および隣接したセル内の粒子との組 (i, j) を求め，リストを作成する．このステップを sorting と呼ぶ．
(5) カットオフ判定後，ルンゲクッタ法を用いて粒子に働く力と加速度を計算する．
(6) 粒子の位置を更新する．
(7) セル間での粒子の交換を行う．
(8) ステップ (4)〜(7) を繰り返す．

ステップ (5) とステップ (7) ではプロセス間で粒子データの明示的な交換が必要である．

ステップ (3) では負荷分散を考慮してセルのマッピングを工夫する必要がある．我々はプロセスへのセルマッピングを自由に記述できるよう，各セルにはそのセル内の粒子データだけでなく，近傍のセルの情報も保持するようにした．したがって，各セルは近傍のセルへのポインタを保持し，そのセルがどのプロセスに所有されているか知ることができる．このようにコーディングすると処理のオーバヘッドが比較的大きくなるが，セルマッピングを柔軟に記述できるため負荷分散を行いやすい．

事実，我々のコードでは，プロセスへのセルのマッピングと，シミュレーションのコア部分は独立に記述され，どのようなマッピングにも対応可能となっている．その反面，プロセス間の通信はセル単位で行われ，プロセスあたりの総通信量に対し，通信回数は比較的多くなる．しかし，後述するように全実行時間の大部分はプロセス内の粒子計算に費やされ，通信時間はそれほど影響しない．

2.2 初 期 条 件

今回用いる 2 次元の衝撃波解析のコードでは，粒子の初期速度は y 次元方向に関してはランダムに与えるが，x 次元方向には**図 1** のような勾配を与える．これは x 次元方向に大きな圧力がかけられていること

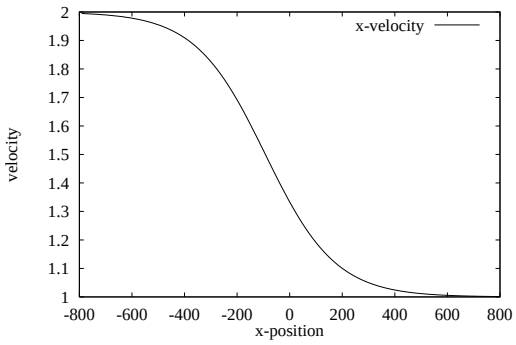

図 1 x 次元方向の初期速度
Fig. 1 Initial velocity in x-direction.

を想定している．実際には，図 1 の勾配に適当なゆらぎを与えて設定する．また，粒子の位置の初期位置は，粒子の密度が x 次元の正の方向に向かって大きくなるように設定する．したがって，左側のセルより右側のセルのほうがつねに粒子の密度が大きくなる．

この初期条件の下で長時間シミュレーションを行うと，粒子が右側の空間に集まってしまうため，実際に計算を必要とする空間は縮小する．このような状況で，シミュレーションの初期に設定したプロセスへのセルマッピングを使い続けると非常に効率が悪くなるので，あるタイムステップが経過した後にステップ (3) に戻り，プロセスへセルを再マッピングする．一般的にこのようなセルの再マッピングはコストが大きいので，適切なタイミングで実行させる．

3. プログラミング

我々が実装した SPAM コードは Fortran77 でコーディングされている．空間分割法による並列化を記述した MPI プログラミングでは特別なことは行っていないが，前述のようにセルマッピングに関しては詳細な制御が可能となっている．また，SMP ノードでより大きな並列性を得るために，1 つの MPI プロセスを OpenMP を用いてさらに並列化することを考える．このような並列化をハイブリッド並列化と呼ぶ．

3.1 力の計算の並列化

この問題で最も時間を消費する処理はルンゲクッタ法を用いて粒子に働く力と加速度を計算する部分である．たとえば，43,200 粒子のシミュレーションを 1 台の計算機で実行させると，力の計算で約 88%の CPU 時間を消費する（**表 1**）．我々はこの計算を行っているループを OpenMP のディレクティブを用いて並列化する．

力の計算において，OpenMP を用いた並列化の粒

表 1 1 タイムステップの実行時間の割合（43,200 粒子）
Table 1 Breakdown for the execution time per step.

force calc.	88.03%
sorting	6.50%
others	5.47%

```
!$OMP parallel do private(i,j,xij,yij,rr,rij,w,wp)
      do ij=0,npair-1
         i = ipairs(1,ij)
         j = ipairs(2,ij)
         xij = x(i) - x(j)
         yij = y(i) - y(j)
         if(yij.gt.+0.5*ny) yij = yij - ny
         if(yij.lt.-0.5*ny) yij = yij + ny
         rr = xij*xij + yij*yij
         if(rr.lt.rlucy*rlucy)then
            rij = sqrt(rr)
            call pot(rij,w,wp)
!$OMP critical
            fx(i) = fx(i) - wp*xij/rij
            fx(j) = fx(j) + wp*xij/rij
            fy(i) = fy(i) - wp*yij/rij
            fy(j) = fy(j) + wp*yij/rij
!$OMP end critical
         endif
      enddo
```

図 2 粒子に働く力の計算部分のコード
Fig. 2 Core part of force calculation code.

度を大きくとれるように，sorting において作成される粒子ペアのリストは，セル単位ではなく，プロセス単位で作成する．すなわち，各プロセス内ではまず各セルをスキャンして粒子ペアのリストを作成した後，そのリストを一気に処理する形をとる．なお，リストを作成する際に任意の粒子 i および粒子 j に関し，(i,j) のエントリがあれば，(j,i) のエントリは作成しない．この 2 つの粒子の組は作用反作用の関係にあり，(i,j) で計算したポテンシャルエネルギを使って (j,i) の力を計算できるため，ポテンシャルエネルギの計算を半分に減らすことができる．

sorting のステップで作成された (i,j) リストに対して，力の計算を行う．しかし，これらの (i,j) 粒子ペアにはカットオフ半径内のものと半径外のものが含まれているため，このリストを単純なブロック分割で OpenMP の *parallel do*を使って並列化すると，各スレッドの計算負荷に偏りが生じる可能性がある．この計算負荷の分散のために OpenMP のスレッドスケジューリングの利用が考えられる．

さらに，複数のスレッドが同時にある粒子へ働く力を更新してしまわないよう，*critical* ディレクティブを用いて粒子に働く力の更新を保護する必要がある．力の計算を行うコアループを OpenMP の *parallel do*で並列化したコードを**図 2** に示す．`npair` は (i,j) リス

表 2 シミュレーション条件
Table 2 Conditions for the simulation.

粒子数	43,200
空間サイズ	$1{,}600 \times 18$
カットオフ半径	3.0
セルサイズ	3.1
セル数	518×6
タイムステップ数	50

トの長さ，`rlucy` はカットオフ半径である．

3.2 sorting の並列化

上述した sorting 処理も OpenMP による並列化が可能である．プロセス内における (i, j) 粒子ペアのリストの作成はセル単位で行っているため，セル単位で処理をスレッドに割り当てればよい．ただし，リストに書き込む際に *critical* ディレクティブを用いて保護する必要がある．このとき，リストにはスレッドがランダムに書き込むため，リスト内の粒子列の空間的局所性が失われる．すなわち，セル単位で粒子列がまとまっていないため，キャッシュに当たりにくくなる．これを回避するためには，スレッドプログラミングにより処理をスレッド内に閉じ込め，局所性を確保する工夫が必要である．sorting の並列化はリストを使用する力の計算に影響するため，詳細は 4 章の性能評価で述べる．

我々のプログラムは MPI と OpenMP を混在させてプログラミングを行っている．以降の章では，各 MPI プロセスに 1 つのスレッドのみ割り当て，複数の MPI プロセスを用いてプログラムを実行した場合を MPI-Only と呼ぶ．一方，1 つの MPI プロセスに複数のスレッドを割り当てて実行した場合を Hybrid と呼ぶ．使用する物理的なプロセッサ数は MPI プロセスと OpenMP によるスレッド数を組み合わせて決定される．

4. 性能評価

表 2 に示す条件下でシミュレーションを行う．また，2.2 節で述べた初期条件とともに，y 次元方向には周期境界条件を，x 次元方向はすべての粒子が左から右へと移動することから，右端にのみミラー境界条件を設定する．

上記のシミュレーション条件下では，実装したコードではセルの再マッピングに約 105 秒のコストがかかるため，本稿では 50 タイムステップごとにセルの再マッピングを行う．実装したコードではセルの再マッピングは逐次処理で行っているため，再マッピングにかかる実行時間は使用するプロセス数を増やしても変わらない．つまり，ハイブリッド並列化と MPI の性能比較において本質的な問題とはならない．したがって，再マッピングのコストを除いた 50 タイステップの実行時間をもとに，1 タイムステップあたりの実行時間を評価指標とする．

4.1 評価環境

評価環境には 12 ノードからなる SMP-PC クラスタ *COSMO*（Cluster Of Symmetric Multi prOcessor）を用いた．各ノードは 4 つの Pentium-II Xeon 450 MHz（1 MB L2 キャッシュ）が共有メモリ結合されている．主記憶は 4-way インタリーブで構成され，高いスループットが得られる[10]．ノード間は 100base-TX Ethernet Switch と 800 Mbps Myrinet で接続されているが，今回の性能評価では Myrinet のみ使用した．

クラスタ実行環境として RWC で開発された SCore-4.2.1[8] を使用した．OS には SMP 対応の Linux 2.4.10，コンパイラには PGI Fortran 3.2-4a[9]（最適化オプションに `-fast` を使用），MPI ライブラリには MPICH-SCore を用いた．

COSMO 上での性能評価において，実際に使用される物理的なプロセッサ数は，SMP ノード数と SMP ノード内のプロセッサ数との積である．以降の性能評価では，MPI-Only，Hybird ともに，SMP ノード数を 1，2，4，8，12，さらに各 SMP ノード内で使用するプロセッサ数を 1，2，3，4 と変化させて計 20 通りの場合について実行する．

4.2 セルマッピングによる負荷分散

今回ターゲットとした問題は，x 次元方向に関して非常にロードバランスが悪く，単純に x 次元方向を MPI プロセス数でブロック分割し，プロセスにセルを割り当てる（以下，flat mapping）と負荷に偏りが生じる．そこで，x 次元方向に関して MPI プロセス数の整数倍でより細かくブロック分割し，セルをプロセスにサイクリックにマッピングする（以下，cyclic mapping）．負荷分散では cyclic mapping が有利であるが，ブロックの境界が増加するため，プロセス間でのデータ通信が増加するという欠点がある．ここでは，両者のトレードオフを調べる．

両者の mapping 方法を実装し，負荷分散の状況を調べるために，1 タイムステップの間に実際に力の計算（図 2 の if 文内の計算）を行った回数について測定した．4 プロセスで実行した場合の結果を**表 3** に示す．cyclic mapping では計算負荷が各プロセスに均等に分散されていることが分かる．

図 3 はセルのマッピングに flat mapping を用いた

表 3 セルマッピングによる計算の負荷分散
Table 3 Load balancing with cell mapping.

プロセス ID	flat mapping	cyclic mapping
0	366,719	972,417
1	387,360	974,227
2	1562,349	962,881
3	1535,350	954,058

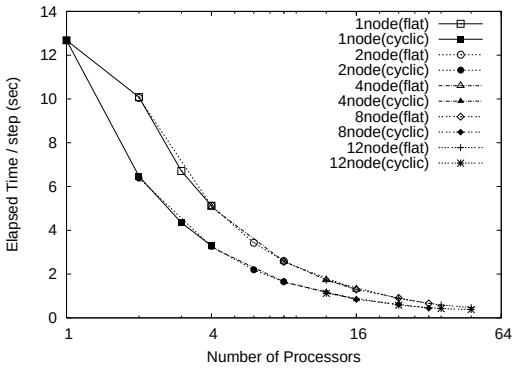

図 3 セルマッピングの効果
Fig. 3 Effects of cell mapping methods.

場合と cyclic mapping を用いた場合の 1 タイムステップあたりの実行時間である．cyclic mapping を用いることにより，最大で約 35%の性能向上が得られた．しかし，cyclic mapping ではプロセス数が増加するにつれ，プロセス間のデータ通信が増え，両者の性能差は小さくなっている．しかし，今回測定した 48 プロセッサではつねに cyclic mapping を用いた場合の性能が上回っており，以降の性能評価では MPI プロセスへのセルマッピングに cyclic mapping を用いる．

4.3 ハイブリッド並列化

4.3.1 OpenMP による負荷分散

力の計算を行うループの各イタレーションの計算負荷の分散のために，OpenMP のスレッドスケジューリングを用いた場合の効果について調べる．この負荷分散とセルマッピングによる負荷分散との違いは，セルマッピングによる負荷分散は (i, j) リストの長さ（`npair`）を均等にプロセスに割り当てることであり，ここでの負荷分散は (i, j) リスト内で実際に行われる力の計算の分散を目的としている．

スレッドスケジューリングは図 2 の *parallel do* ディレクティブに *schedule* (*SCHEDULE,chunk*) を指定することにより制御する．この効果を検証するために，OpenMP におけるスレッドスケジューリングについて，以下の 3 種類について実行した．

- **static**：スレッド数でループ長を単純にブロック分割し，静的にスレッドに割り当てる．

表 4 スレッドスケジューリングの効果
Table 4 Effects of thread scheduling methods.

スケジュール方法	実行時間 (sec)
static	7.67
cyclic	7.64
dynamic	7.59

表 5 スレッドスケジューリングによる計算の負荷分散
Table 5 Load balancing with thread scheduling.

スレッド ID	static	cyclic	dynamic
0	924,895	942,257	960,329
1	970,685	955,141	960,285
2	972,495	970,682	956,265
3	972,915	972,910	964,111

- **cyclic**：*chunk* にブロックサイズを指定し，より細かい粒度で分割しサイクリックにスレッドへ割り当てる．ブロックサイズは，リストの長さ `npair` をスレッド数の整数倍（ここでは 4）で分割したサイズを指定した．
- **dynamic**：cyclic と同様，*chunk* にブロックサイズを指定する．こちらは，スレッドに動的に割り当てる．なお，ブロックサイズは cyclic の場合と同じサイズを指定した．

cyclic と dynamic についてはそれぞれ最も良い性能が得られたブロックサイズを指定した．

COSMO の 1 台の SMP ノード上で 4 スレッド用いて実行し，各スケジューリング方法における 1 タイムステップあたりの実行時間を**表 4** に示す．

表 4 から分かるようにスレッドスケジューリングを変えても実行時間にあまり差は見られない．これは，カットオフの半径内に存在し，実際に力の計算が行われる粒子 (i, j) の組合せが，(i, j) リスト内で偏りなく存在しているためと考えられる．

各スケジューリングについて，1 タイムステップの間に実際に力の計算（図 2 の if 文内の計算）を行った回数について**表 5** に示す．このように単純なブロック分割である static でも 4 つのスレッド間での計算負荷にそれほど偏りがない．今回の測定では static でも十分といえるが，より長いステップ数実行した場合やシミュレーション条件によっては負荷の偏りが大きくなると考えられる．よって，以降の Hybrid の性能評価には dynamic によるスレッドスケジューリングを用いる．

4.3.2 OpenMP オーバヘッド

表 4 の実行時間と，図 3 の MPI-only の 1 ノードの実行時間を比較すると，Hybrid の性能が非常に悪いことが分かる．この原因の 1 つに粒子に働く力を更

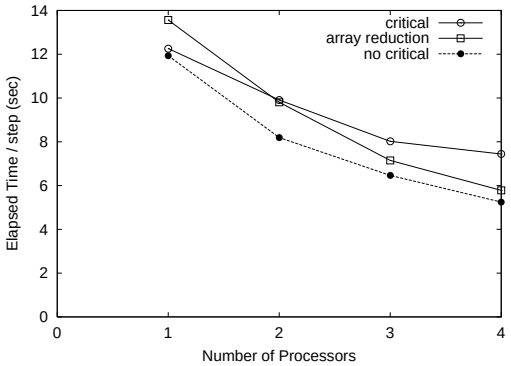

図 4 critical region のオーバヘッド
Fig. 4 Overheads in the critical region.

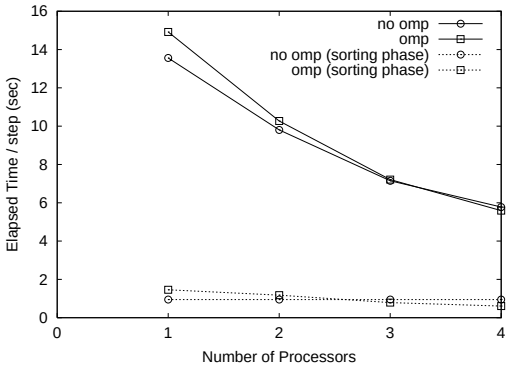

図 5 sorting の OpenMP 並列化
Fig. 5 OpenMP parallelization for the sorting part.

新する際の critical region のオーバヘッドが考えられる．このオーバヘッドの影響を調べるため，以下の 3 種類について評価する．

- **array reduction**：2 次元配列を用いて，各スレッドに個別に力を更新させ，最後に全スレッドの配列を加算する．critical region のオーバヘッドはなくなるが，余分な加算処理のオーバヘッドが加わる．
- **critical**：`!OMP critical` を用いた場合．図 2 のコード．
- **no critical**：図 2 のコードから `!OMP critical` を取り除いて実行した場合（シミュレーション結果は正しくない）．オーバヘッドのない理想的な場合を想定している．

1 ノード上で実行した結果を**図 4** に示す．

array reduction により 4 スレッド時に約 22%性能が向上した．また，オーバヘッドのない理想的な場合（no critical）でも，約 30%の性能向上が限界である．以降の Hybrid の性能評価には array reduction を使用する．

4.3.3 sorting の OpenMP 並列化

3.2 節で述べたように，sorting 処理も OpenMP による並列化が可能である．しかし，単純に *parallel do* で並列化してしまうと，(i, j) リスト内の空間的局所性が失われるだけでなく，前節と同様 critical region のオーバヘッドも大きく性能向上しない．したがって，ここでも 3 次元配列を用意し，各スレッドが個別にリストを更新するように並列化した．このとき，長さの異なるリストがスレッドの数だけ作成されるが，力の計算時に先頭のスレッドのリストから順に使用し，全スレッド分繰り返すようにした．1 ノード上で実行した結果を**図 5** に示す．

sorting の OpenMP 並列化により 4 スレッド時にわずかに性能向上が見られた．sorting にかかる実行時間では 3 スレッド時でも性能向上しているが，全体の実行時間では 4 スレッドの場合のみである．これは，スレッドごとに長さが異なる粒子ペアリストが作成され，力の計算時にはその短くなったリストをさらにスレッドで分割するため，力の計算部分で余分なオーバヘッドが発生してしまうためと考えられる．また，粒子リストの長さがスレッドごとに異なってしまうのは，セル単位でスレッドに処理を割り当てているためであり，上述したように *parallel do* を用いずスレッドプログラミングを行っているため，スレッド間で効率的な負荷分散ができない．

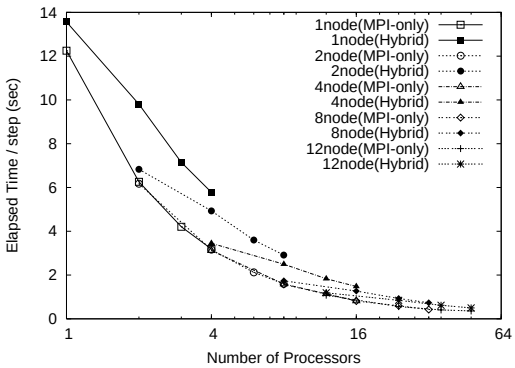

図 6 MPI-Only と Hybrid の実行時間
Fig. 6 Execution time for MPI-Only and Hybrid codes.

また，1 スレッド時の性能低下がやや大きい．並列化オーバヘッド以外の原因として，スレッドごとの粒子ペアリストとして用いた 3 次元配列ために，キャッシュに当たりにくくなっている可能性も考えられる．

4.4 Hybrid と MPI-Only の性能比較

次に SPAM コードを Hybrid で実行した場合と MPI-Only で実行した場合の 1 タイムステップあたりの実行時間を**図 6** に示す．なお，MPI-Only におけ

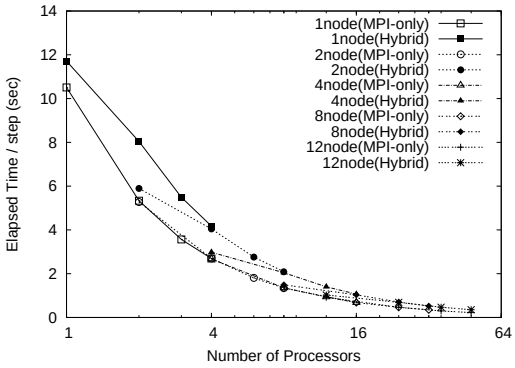

図 7 MPI-Only と Hybrid の内部処理時間（力の計算）
Fig. 7 Force calculation time for MPI-Only and Hybrid codes.

表 6 力の計算部分の L2 キャッシュミス率
Table 6 L2 cache miss-hit ratio in force calculation.

#node	#processor	MPI-Only	Hybrid
1	1	0.136	0.136
	2	0.139	0.305
	3	0.148	0.529
	4	0.146	0.642
2	1	0.142	0.139
	2	0.144	0.308
	3	0.154	0.531
	4	0.155	0.646
4	1	0.147	0.161
	2	0.157	0.316
	3	0.161	0.530
	4	0.152	0.641
8	1	0.159	0.161
	2	0.149	0.316
	3	0.142	0.530
	4	0.124	0.641
12	1	0.160	0.162
	2	0.146	0.316
	3	0.108	0.536
	4	0.109	0.644

るプロセスのノードへのマッピングは，通信を最適化するために隣接するプロセス（隣接するセルを担当するプロセス）が同じノードになるようにした．図 6 から MPI-Only は Hybrid よりつねに性能が良いことが分かる．

この両者の性能差の原因を調べるため，プロセス間の通信時間を除いた内部処理時間，すなわち粒子に働く力の計算にかかる時間を測定した．測定結果を**図 7** に示す．この結果から，通信時間ではなく，力の計算にかかる時間が性能差の主な原因であることが分かる．

SMP クラスタでは共有バスがボトルネックとなるため，キャッシュの利用効率の測定に基づく性能解析が有効である[10]．以下では共有メモリバスへのアクセスをともなう L2 キャッシュのミス率を考える．

力の計算部分について，L2 キャッシュミス率を測定した結果を**表 6** に示す．なお，Intel の Pentium-II Xeon には，プロセッサのパフォーマンスを測定するためのカウンタが用意されており，このカウンタを使用し，プロセッサのデータ系のメモリ参照回数（DATA_MEM_REFS）とすべてのバストランザクション回数（BUS_TRAN_ANY）を測定することにより，L2 キャッシュミス率を算出している（式 (4)）．

$$\frac{\text{BUS\_TRAN\_ANY}}{\text{DATA\_MEM\_REFS}} = \text{L2-cache miss-hit ratio} \tag{4}$$

表 6 から Hybrid では SMP ノード内で複数のプロセッサを用いた場合，L2 キャッシュミス率が大きくなることが分かる．以下，Hybrid で L2 キャッシュミス率が悪くなる原因について考察する．

Hybrid では，sorting のステップで作成される (i, j) リストに対しスレッドを用いて並列化している．この (i, j) リストがどのような構造になっているかを説明するために，リストの作成手順を以下に示す．

(1) あるセル内の粒子 i について，同一セル内の粒子 j との組を重複がないように作成する．
(2) 粒子 i と隣接した 8 つのセル内の粒子 j との組を作成する．セルが境界上にある場合以外は，重複を避けるため，東，北西，北，北東の 4 つのセル内の粒子との組を作成する．
(3) セル内のすべての粒子について，(1) と (2) を繰り返す．
(4) そのプロセスが担当するすべてのセルについて (1)〜(3) を繰り返す．

このように，(i, j) リストはセルを処理単位として作成される．なお，ステップ (4) では，プロセスが自分の担当しているセルを参照する場合，y 次元方向を下から上に連続にアクセスする．

このような手順で作成された (i, j) リストを用いて，粒子に働く力の計算を行うと，**図 8** のように，5 つのセル内のデータのみ繰り返しアクセスすることになる．力の計算で主に使われる倍精度実数データは，粒子の位置情報（図 2 の `x`，`y`）と力（図 2 の `fx`，`fy`）の 4 つである．また，この問題では 1 つのセル内の粒子は 100 を超えておらず，5 つのセル内でアクセスされるデータ量の最大値を見積もると

$$5 \times 100 \times 4 \times 8\,\text{byte} = 16{,}000\,\text{byte}$$

となり，約 16 KB で非常に小さい．また，この問題では，y 次元方向のセルは 6 つと少なく，$6 \times 3 = 18$ 個のセルをアクセスしてもデータ量は約 56 KB で L2

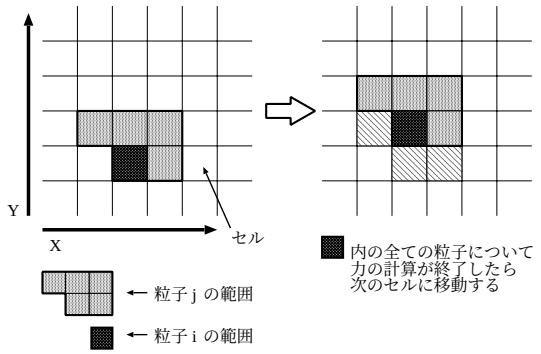

図 8 力の計算におけるデータアクセスパターン
Fig. 8 Data access pattern in force calculation.

キャッシュから追い出されることはない．したがって，隣の列のセル（x 次元方向で右隣の列のセル）の力を計算するときには，東と北東以外のセルはすでにキャッシュに入っていると考えられる．したがって，x 次元方向のセルも連続にアクセスすることによりキャッシュを有効的に利用できる．

次に複数のプロセッサ上で実行する場合について考える．MPI-Only では使用するプロセス数を増やしても，各プロセスがそれぞれ (i,j) リストを作成し，(i,j) リストの先頭から処理していくため，キャッシュを有効に利用できる．一方，Hybrid はこのセルの空間的な位置とはまったく関係なく (i,j) リストの要素を割り当てるので，MPI-Only ほどキャッシュに当たらないと考えられる．つまり，(i,j) リストは先頭から，あるいはセル単位の処理の変わり目から，計算を始めるのがキャッシュを利用するうえで最も効率が良いと考えられるが，Hybrid では (i,j) リストの任意の点から処理をスレッドに割り当ててしまうので，効率が悪くなると考えられる．

Hybrid で MPI-Only と同じような割当てを行うのは困難であり，余分な処理が必要になる．(i,j) リストでセル単位での処理の変わり目をスレッドに割り当てるためには，プログラムの広範囲を parallel region とし，スレッド ID を用いた詳細なスレッドプログラミングが必要になると考えられる．

4.5 OpenMP におけるデータローカライゼーション

以上をまとめると，Hybrid，すなわち OpenMP を用いるプログラミングにおいて，プロセス内のデータの生成とその処理というような関係を考える場合，それらが同一のスレッドで実行されることが重要である．すなわち，データのローカライゼーションが重要であり，たとえば最初のループでデータを生成し，次のループでそれを参照するような場合，各ループのイタレーションが同一スレッドに割り当てられないと，キャッシュのヒット率が大幅に低下し，性能を悪化させる大きな要因になる．

これを避ける有効な方法は，前半のループと後半のループの各範囲がぴったり一致するブロック割当てを行うことであるが，ここで取りあげた問題のように，ダイナミックな負荷分散を行う必要が生じると，これがうまく適用できない．もう 1 つの方法は，最初から複数のループにまたがる処理をスレッドに大きく分けてしまい，スレッド内で各計算フェーズを閉じ込めて行うようなプログラミングを行うことである．しかし，この方法はいわゆる naive な並列化であり，OpenMP の持つ，逐次プログラムからのインクリメンタルな並列化という，プログラムの簡便さを大きく損なうことになりかねない．また，動的負荷分散への対応については最初に述べた方法と同様，対応しにくい．

上記の点から，OpenMP を用いた Hybrid 並列プログラムでは，内部処理において MPI のみの場合よりもキャッシュを有効利用できないことが多く，内部処理に関してある程度の性能低下は避けられない．したがって今回の例のように，Hybrid 並列化を行うことで，内部処理の性能低下に比べて通信時間および負荷分散の面でより大きな性能向上が得られない場合，総合的な性能で MPI のみの場合より高性能を得ることは難しい．

5. ま と め

本稿では，最近の研究で報告されているハイブリッド並列化における性能低下の原因を調べるために，SPAM 粒子シミュレーションを MPI と OpenMP を用いてハイブリッド並列化した．SMP-PC クラスタ上で実行した結果，MPI のみ用いた場合の方がハイブリッドよりつねに良い性能となった．性能解析により，ハイブリッドは MPI のみに比べて，力の計算における L2 キャッシュミス率が高いことが分かった．この原因として，OpenMP による並列化においてデータの局所性を効率良く利用できないことが考えられる．

また，今回 OpenMP による並列化を試みた sorting 処理は，後に力の計算で使用するデータを扱っているため，OpenMP による並列化は難しい．ループで行っている処理によっては，MPI による粗粒度な並列化のほうが低オーバヘッドで効率が良い場合がある．特に，ループ間で扱うデータに依存性がある場合，OpenMP で一方のループを並列化すると，他方のループにも何らかの影響が出るため，性能チューニングが難しい．

スレッド間でのデータの流れに基づく解析手法も提案されている[11]が，まだ発展段階にある．ハイブリッド並列化においても，そのような解析を簡単に行えることが重要である．

謝辞 本研究の初期段階において貴重なご意見をいただいた，Lawrence Livermore National Laboratory の Antony DeGroot 博士に感謝いたします．また，実験環境の構築にご協力をいただいた日本原子力研究所の板倉憲一氏に感謝いたします．なお，本研究の一部は日本学術振興会科学研究費補助金基盤研究（C）課題番号 14580360 による．

参 考 文 献

1) Clusters@TOP500. http://clusters.top500.org
2) Henty, D.S.: Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling, *Proc. SC'00*, Dallas, USA (Nov. 2000).
3) Cappello, F. and Etiemble, D.: MPI versus MPI + OpenMP on IBM SP for the NAS Benchmarks, *Proc. SC'00*, Dallas, USA (Nov. 2000).
4) Lucy, L.B.: A numerical approach to the testing of the fission hypothesis, *The Astronomical Journal*, Vol.82, pp.1013–1024 (1977).
5) Gingold, R.A. and Monaghan, J.J.: Smoothed particle hydrodynamics: Theory and application to non-spherical starts, *Mon. Not. R. Astr. Soc.*, 181, pp.375–389 (1977).
6) Beazley, D.M. and Lomdahl, P.S.: Message-passing multi-cell molecular dynamics on the Connection Machine 5, *Parallel Computing*, Vol.20, pp.173–195, (1994).
7) Plimpton, S.: Fast Parallel Algorithms for Short-Range Molecular Dynamics, *Journal of Computational Physics*, Vol.117, pp.1–19 (1995).
8) SCore Cluster System Software. http://www.pccluster.org
9) PGI Workstation, The Portland Group, Inc. http://www.pgroup.com
10) 吉川茂洋，早川秀利，近藤正章，板倉憲一，朴泰祐，佐藤三久：SMP-PC クラスタにおける OpenMP+MPI の性能評価，情報処理学会研究報告，2000-HPC-80-27, pp.155–160 (2000).
11) 佐藤茂久，草野和寛，佐藤三久：OpenMP 並列プログラムのデータフロー解析手法，情報処理学会研究報告，2000-HPC-82-13, pp.71–76 (2000).

(平成 14 年 1 月 29 日受付)
(平成 14 年 5 月 17 日採録)

吉川 茂洋

昭和 51 年生．平成 12 年筑波大学第三学群情報学類卒業．平成 14 年同大学大学院システム情報工学研究科博士課程中退．同年，富士通株式会社に入社．クラスタリングによる高信頼性システムの研究開発に従事．

朴 泰祐（正会員）

昭和 59 年慶應義塾大学工学部電気工学科卒業．平成 2 年同大学大学院理工学研究科電気工学専攻後期博士課程修了．工学博士．昭和 63 年慶應義塾大学理工学部物理学科助手．平成 4 年筑波大学電子・情報工学系講師，平成 7 年同助教授，現在に至る．超並列処理ネットワーク，超並列計算機アーキテクチャ，ハイパフォーマンスコンピューティング，並列処理システム性能評価の研究に従事．電子情報通信学会，日本応用数理学会，IEEE 各会員．

佐藤 三久（正会員）

昭和 34 年生．昭和 57 年東京大学理学部情報科学科卒業．昭和 61 年同大学大学院理学系研究科博士課程中退．同年新技術事業団後藤磁束量子情報プロジェクトに参加．平成 3 年通産省電子技術総合研究所入所．平成 8 年新情報処理開発機構並列分散システムパフォーマンス研究室室長．平成 13 年より，筑波大学電子情報工学系教授．同大学計算物理学研究センター勤務．理学博士．並列処理アーキテクチャ，言語およびコンパイラ，計算機性能評価技術，グリッドコンピューティング等の研究に従事．日本応用数理学会会員．

高橋　大介（正会員）

昭和 45 年生．平成 3 年呉工業高等専門学校電気工学科卒業．平成 5 年豊橋技術科学大学工学部情報工学課程卒業．平成 7 年同大学大学院工学研究科情報工学専攻修士課程修了．平成 9 年東京大学大学院理学系研究科情報科学専攻博士課程中退．同年同大学大型計算機センター助手．平成 11 年同大学情報基盤センター助手．平成 12 年埼玉大学大学院理工学研究科助手．平成 13 年筑波大学電子・情報工学系講師．博士（理学）．並列数値計算アルゴリズムに関する研究に従事．平成 10 年度情報処理学会山下記念研究賞，平成 10 年度情報処理学会論文賞各受賞．日本応用数理学会，ACM，IEEE，SIAM 各会員．

Carol G. Hoover

She attended Auburn University and the University of Virginia, receiving her Ph.D. degree from the University of California in 1977. She has worked at the Lawrence Livermore National Laboratory in a variety of research and management positions ever since. Her specialty is large-scale parallel computation.

William G. Hoover

He attended Oberlin College and received both M.S. Chem. and Ph.D. degrees from the University of Michigan in 1961. After a postdoctoral year at Duke University he became a staff physicist at Lawrence Radiation Laboratory. In 1972 he took a joint professorship in the University of California and wrote 250 research papers on statistical mechanics and computational physics, as well as three books. Now he is a professor emeritus and pursuing chaos and computation fulltime.