ハイパフォーマンス
コンピューティング 87−8
（２００１．７．２５）

SPAM 粒子シミュレーションコードのハイブリッド並列化

吉川茂洋[†] 朴泰祐[†] William G. Hoover[††]
Carol G. Hoover[†††] 佐藤三久[†]

2 次元の衝撃波解析に SPAM (Smooth Particle Applied Mechanics) を適用し、SMP クラスタをターゲットプラットフォームとしてコードを並列化した。SMP ノードでの並列性を引き出すために MPI と OpenMP を組み合わせたハイブリッド並列化による実装を行なった。SMP-PC クラスタ COSMO 上で 43,200 粒子のシミュレーションを行なった結果、実行性能は MPI のみを用いるほうが良い性能が得られた。キャッシュの利用効率に基づく性能解析を行ない、性能差の原因について考察する。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 code

Shigehiro YOSHIKAWA,[†] Taisuke BOKU,[†]
William G. Hoover,[††] Carol G. Hoover[†††]
and Mitsuhisa SATO[†]

We apply the SPAM (Smooth Particle Applied Mechanics) method for 2-dimensional shock-wave analysis and develop a parallelized SPAM code for SMP cluster. To exploit the parallelism in an SMP node, We implement the program with Hybrid method combining MPI and OpenMP. We evaluated the performance of palallelized SPAM code on an SMP-PC cluster named COSMO, and the results show MPI-Only method has higher performance than Hybrid one. We show a performance analysis based on the cache utilization on SMP processors, and also discuss issues of OpenMP on such a problem.

1. はじめに

SPAM (Smooth 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の一つである。基本的な概念は分子動力学法シミュレーション (MD) に非常に良く似ており、粒子間の相互作用には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いる。したがって、粒子間の作用力の計算には、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。カットオフとは、ある固定距離より離れた粒子間でのポテンシャルを打ち切ることにより、相互作用の計算量を大幅に削減する手法である。

我々は 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た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[1),2)]。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、プロセッサへのセルのマッピングを工夫することにより、計算の負荷分散を行なうことが重要である。

本稿では、2 次元の衝撃波を解析するための並列 SPAM コードを実装し、その性能解析を行なう。計算のプラットフォームには、近年 HPC 分野で関心の高い SMP クラスタをターゲットとする。SMP クラスタは高価な超並列計算機に比べ、より安価に高性能なシステムを構築できるため注目されている。我々は SMP ノードでの並列性を引き出すため、MPI のみによる実装だけでなく、MPI と OpenMP を組み合わせたハイブリッド並列化による実装を行なう。ハイブリッド並列化では、SMP ノード間は MPI によるセルのマッピングによる負荷分散、SMP ノード内では OpenMP のスレッドスケジューリングによる負荷分散を行なう。

以下、2 章で並列 SPAM 粒子シミュレーションの

† 筑波大学 電子・情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† Department of Applied Science, University of California
††† Lawrence Livermore National Laboratory

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 章でまとめを行なう。

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及び粒子 i に働く力は次の式で表される。

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

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

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

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なわれる。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・速度など) 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) 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。ここでカットオフの判定を行なう。
(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なわれ、プロセス当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常に粒子の密度が大きくなる。

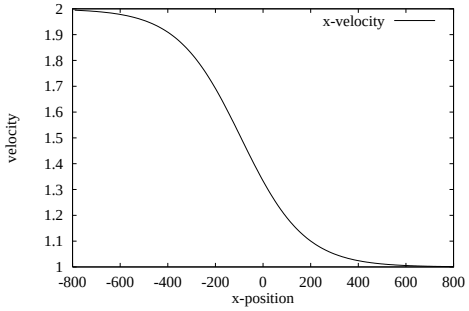

図 1 x 次元方向の初期速度

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。基本的な MPI プログラミングでは特別な事は行なっていないが、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、一つの 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。

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。例えば、43,200 粒子のシミュレーションを一台の計算機で実行させると、力の計算で約 70 % の CPU

時間を消費する。我々はこの計算を行なっているループを OpenMP のディレクティブを用いて並列化する。粒子間力の計算において、OpenMP を用いた並列化の粒度を大きく取れるように、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される。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取る。

sorting 処理も比較的処理は重いが、リストを作成する際に任意の粒子 i 及び粒子 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からである。SPAM コードでの粒子間ポテンシャル計算は複雑なため、このことは性能上重要であり省くことはできない。このように粒子間の対称性が崩れるため、sorting 処理を OpenMP を用いて効率的に並列化することは難しい。

sorting のステップで作成された (i, j) リストに対して、力の計算を行なう。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* によって並列化すると、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。力の計算を行なうコアループを OpenMP の *parallel do* で並列化したコードを図 3 に示す。`npair` は (i, j) リストの長さ、`rlucy` はカットオフ半径である。*critical* で囲まれた部分を 2 つ用意しているのは排他制御される部分を小さくするためである。

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なっている。以降の節では、各 MPI プロセスに一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、一つの 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

問題サイズとして 43,200 粒子のシミュレーションを行なう。また、2.2 節で述べた初期条件の下でシミュレーションを開始する。y 次元方向には周期境界条件を設定し、x 次元方向は全ての粒子が左から右へと移動するので、右端にのみミラー境界条件を設定する。

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

図 2 粒子に働く力の計算部分のコード

4.1 評価環境

並列 SPAM コードの評価には、我々の研究室で稼働している SMP-PC クラスタ COSMO (Cluster Of Symmetric Multi prOcessor) を用いる[5]。各ノードは Intel Pentium-II Xeon (450MHz, 512KB 4way L2 キャッシュ) を 4 台搭載した DELL PowerEdge6300 で、4 ノードで構成される。主記憶は 4-way インターリーブで構成され、高いスループットが得られる[5]。ノード間は 100base-TX Ethernet Switch と 800Mbps Myrinet で接続されている。なお、今回の性能評価では 100base-TX Ethernet のみ使用する。

OS には SMP 対応の Linux 2.2.16 を使用している。OpenMP コンパイラには PGI Fortran 3.1、MPI ライブラリには MPICH-1.2.0 を用いる。

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、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 12 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

我々がターゲットとしている問題は、x 次元方向に関して非常にロードバランスが悪くなっている。したがって、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) と負荷分散がうまく行なわれない。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。

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 3 の if 文内の計算) を行な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 1 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。

表 1 cyclic mapping による計算の負荷分散

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

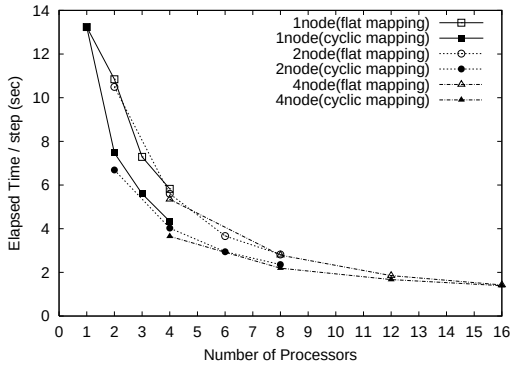

図 3 セルマッピングの効果

図 3 はセルのマッピングに flat mapping を用いた場合と cyclic mapping を用いた場合の 1 タイムステップ当たり実行時間である。cyclic mapping を用いることにより、最大で約 30 % の性能向上が得られた。しかし、使用するプロセッサ数が増加するにつれて、両者の性能差は小さくなっている。これは、cyclic mapping によりプロセス間のデータ通信が増えたためと考えられる。

以降の性能評価では MPI プロセスへのセルマッピングには cyclic mapping を用いる。

4.3 OpenMP による負荷分散

次に力の計算を行な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な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 3 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、COSMO の 1 台の SMP ノード上で 4 スレッド用いて実行する。表 2 に OpenMP におけるスレッドスケジューリングを static, cyclic, dynamic と変えた場合の 1 タイムステップ当たりの実行時間を示す。static では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。cyclic は *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。dynamic は cyclic と同様に細かくブロック分割し動的にスレッドに割り当てる。cyclic と dynamic についてはそれぞれ最も良い性能が出るようにブロックサイズを指定した。

表 2 スレッドスケジューリングの効果

スケジュール方法	実行時間 (sec)
static	6.80
cyclic	6.78
dynamic	6.77

表 3 スレッドスケジューリングによる計算の負荷分散

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

表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なわれる粒子 (i, j) の組み合わせが、(i, j) リスト内で偏り無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 3 の if 文内の計算) を行なった回数について表 3 に示す。このように単純なブロック分割である static スケジューリングでも 4 つのスレッド間での計算負荷にそれほど偏りがないことが分かる。したがって、ここではスレッドスケジューリングを工夫することによる効果はあまり現れない。

4.4 ハイブリッド並列 SPAM コードの性能

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 タイムステップ当たりの実行時間を図 4 に示す。なお、MPI-Only におけ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 4 から MPI-Only は Hybrid より常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 5 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[5]。以下では共有メモリバスへのアクセ

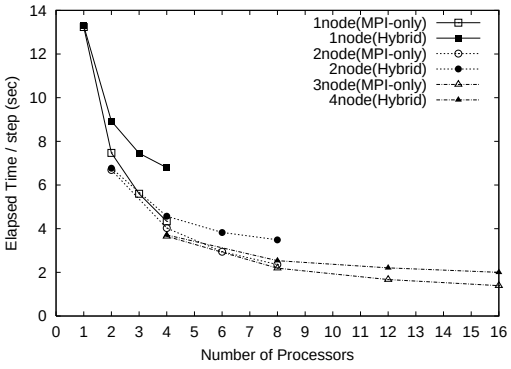

図 4 MPI-Only と Hybrid の実行時間

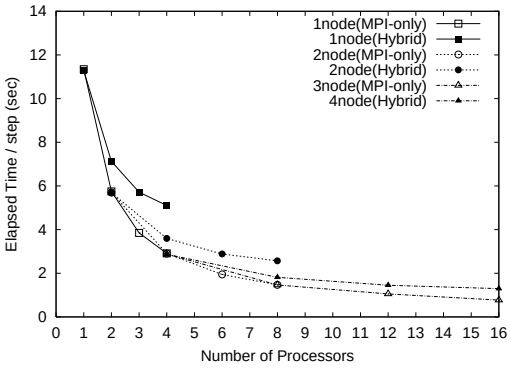

図 5 MPI-Only と Hybrid の内部処理時間 (力の計算)

スを伴なう 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 4 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) と全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。

表の結果から 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 との組を重複が無いように作成する。

(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) セル内の全ての粒子について、1 と 2 を繰り返す。

表 4 力の計算部分の L2 キャッシュミス率 (%)

#node	#processor	MPI-Only	Hybrid
1	1	0.118	0.119
	2	0.119	0.552
	3	0.121	0.723
	4	0.130	0.854
2	1	0.124	0.120
	2	0.122	0.555
	3	0.125	0.726
	4	0.132	0.861
4	1	0.122	0.123
	2	0.128	0.563
	3	0.138	0.747
	4	0.142	0.899

(4) そのプロセスが担当する全てのセルについて 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なうと、図 6 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

$$5 \times 100 \times 4 \times 8byte = 16000byte$$

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 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。

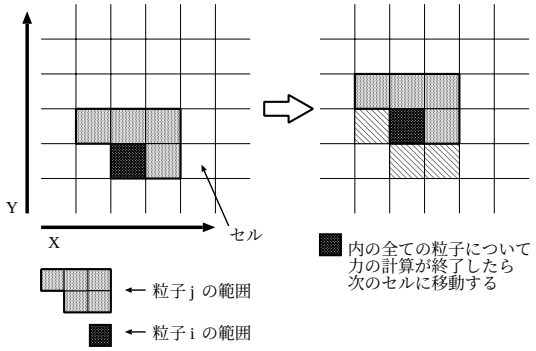

図 6 力の計算におけるデータアクセスパターン

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全く関係なく (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上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 と同じような割り当てを行なうのは困難であり、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、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、例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り当てを行なうことであるが、ここで取り挙げた問題のように、ダイナミックな負荷分散を行なう必要が生じると、これがうまく適用できない。もう一つの方法は、最初から複数のループに跨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なうようなプログラミングを行なうことである。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兼ねない。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、対応しづらい。

OpenMP を用いた Hybrid 並列プログラムでは、ノード内通信のオーバヘッドを減らし、またノード間通信の粒度を大きくし、同様に通信オーバヘッドを減らすというメリットがある。しかし今回の例のように、OpenMP 部分でのデータ局所性に基づくオーバヘッドが通信オーバヘッドを上回る場合、総合的な性能で MPI のみの場合を上回るのは難しい。この場合、naive な並列化はあまり意味がなく (それくらいなら MPI だけでプログラミングした方が簡単である)、Hybrid 並列化によって高性能が得られる例は数少ない。

5. ま と め

本稿では、連続体の解析に有用な手法である SPAM を 2 次元の衝撃波解析に適用し、SMP クラスタをターゲットプラットフォームとして並列化を行なった。SMP ノードでより高い性能向上を得るために、MPI だけでなく、MPI と OpenMP を組み合わせるハイブリッド並列化を行なったが、実行性能では MPI だけ用いた場合の方が良い結果となった。MPI と OpenMP によるハイブリッド並列化で高い性能を得ることは難しく、最近の研究でも報告されている[3),4)]。キャッシュミス率に基づく解析から、ハイブリッドは MPI のみ用いる場合に比べて、データの局所性を効率良く利用できないことが大きな原因と考えられる。

今回行なったハイブリッド並列化では、性能を改善するための余地が残されている。我々のコードでは粒子に働く力を更新する際に、スレッド同士が競合しないように *critical* ディレクティブを用いている。この排他制御のオーバヘッドは、多次元配列を別に用意し、スレッド毎に独立に力の更新を行なわせることにより削除可能だが、ループの最後に余分な加算処理が必要となる。それぞれの手法のオーバヘッドのトレードオフを調べるために、後者の手法についても実装及び評価が必要である。また、データの配置とアクセス順序制御に関しても、さらに考察を重ねる必要がある。

謝辞 本研究の初期段階において貴重な意見を頂いた、Lawrence Livermore National Laboratory の Antony DeGroot 博士に感謝する。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 12680327 による。

参 考 文 献

1) D.M.Beazley, P.S.Lomdahl : *Message-passing multi-cell molecular dynamics on the Connection Machine 5*, Parallel Computing 20, pp.173-195, 1994.
2) S.Plimpton : *Fast Parallel Algorithms for Short-Range Molecular Dynamics*, Journal of Computational Physics, vol.117, pp.1-19, 1995.
3) D.S.Henty : *Performance of Hybrid Message-Passing and Shared-Memory Parallelism for Discrete Element Modeling*, Proc. of SC'00, Dallas, USA, Nov. 2000.
4) F.Cappello, D.Etiemble : *MPI versus MPI + OpenMP on IBM SP for the NAS Benchmarks*, Proc. of SC'00, Dallas, USA, Nov. 2000.
5) 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, pp155-160, 2000.