Linux PCクラスタを用いた並列粒子シミュレーション

蔡 東生[†] 安室秀則[††,☆] 李 尭亭[†]
陸 全明[†] 肖 池階[†]

本研究では，安価な Linux OS を載せた 2 ウェイ PC16 台と 100 ベース・イーサネット・スイッチを使いベアウルフ・クラスタ・システム（以降 PC クラスタと呼ぶ）を構成し，これまで大型並列計算機上などで行われてきた UCLA の Decyk, V.K. 教授によって提案された「Skeleton Particle-In-Cell（PIC）コード」（以降 SPIC と呼ぶ）をクラスタ用に改良しクラスタ・システムの並列性能評価実験を行った．PC クラスタ上で：(1) Fortran 90 によるオブジェクト指向化，(2) PVM によるメッセージ・パッシングと OpenMP によるマルチタスク，スレッド分割を併用することで，通信遅延によるスピードアップの劣化を低くおさえる改良手法を SPIC に適用した．また，(1)，(2) により最適化したコードを使い実際の並列ベンチマークテストを行い大型計算機による結果と比較した．さらに HPF（High Performance Fortran）で並列化したコードと並列性能を本 PC クラスタ上でベンチマークテストを行い比較した．

Parallel Particle Simulation Using Linux PC Cluster

Dongsheng Cai,[†] Hidenori Yasumuro,[††,☆] Yaoting Li,[†]
Quanming Lu[†] and Chijie Xiao[†]

We build a 2-way PentiumPro Linux PC cluster using a 100Base-TX ethernet switch. In this report, using a Skeleton Particle-In-Cell (SPIC) code that was proposed for a testbed and benchmarking purposes to build a large scale parallel PIC code, we have first improved the code and then benchmarked the parallel performances of our Linux cluster using the skeleton-PIC-code that was optimized using both PVM and OpenMP. The code was improved by being rewritten in object-oriented-manner using Fortran 90 derived types. We measured the parallel performances of our code in our Linux PC cluster, and compared the parallel performances of our code with those of commercial parallel computers. We also compared our optimized code with High Perfomance Fortran (HPF) code on our Linux cluster.

1. はじめに

近年，PC（パーソナルコンピュータ）の発達によって，これまでスーパコンピュータによらなければ不可能だった大規模な粒子シミュレーションが安価な PC を使って可能となってきた．粒子シミュレーションの並列化は歴史的には新しい研究ではなく，並列計算機が登場してから，ほぼすべての並列計算機でその並列化が行われている．たとえば，コネクション・マシンでは Dagum [2] がデータ並列モデルで並列化を行っている．このとき，1 仮想プロセッサに 1 粒子，1 グリッドずつそれぞれ対応させている．グリッドを持つ粒子シミュレーションでは，グリッドデータと粒子データの CPU に対する対応付けが，並列化の最大の問題点になる．それに対して，Liewer ら [1] は粒子データとグリッドデータをそれぞれドメイン分割し，それぞれのドメインを対応づける手法をとった．このとき，粒子を 1 タイムステップ動かし，粒子がドメインから離れると，離れた粒子データをバッファにためておき，別のドメインに粒子データを粒子をすべて動かした後，一度に送る．グリッドデータもゴーストセルもしくはガードセルを使い場の情報をすべて更新してから，一度に足しあわせる．しかし，いずれの方法もどの程度の並列性能があるかは不明であった．その後，Liewer らの GCPIC コードに基づき並列性能の計測指標および並列アルゴリズム開発のためのテストコードとしての粒子シミュレーションコード Skeleton Particle-In-Cell

† 筑波大学電子情報工学系
Institute of Information Sciences and Electronics, University of Tsukuba
†† 筑波大学工学研究科
Doctoral Program in Engineering, University of Tsukuba
☆ 現在，日本ユニシス株式会社
Presently with Nihon UNISYS Ltd.

コード（以降，SPIC と呼ぶ）[3] が提唱された．本研究では SPIC を使い，デュアル Pentium Pro プロセッサ搭載の共有メモリ型 SMP-PC 16 台を 100BASE-TX イーサネットスイッチで接続して Linux PC クラスタすなわちベアウルフ・クラスタ・システム[4] を構成し，そのシステム上で SPIC による並列粒子シミュレーションを行い，PC クラスタ並列性能をベンチマークした．SPIC を PC クラスタ上で高速化するため，(1) Fortran 90 の derived types 定義[5] を使いコードのオブジェクト指向化をはかり，(2) PC ノード間通信はメッセージパッシングを PVM[6] により行い，(3) PC ノード内通信は OpenMP[7] を用いて行う．(1)〜(3) の機能を新たに加えることにより SPIC の最適化を試みた．

本論文では 2 章で SPIC コードの構成と並列ドメイン分割を議論し，3 章では PC クラスタ・システムの構成について報告する．4 章では単純にプロセッサにタスクを割り振った PVM による並列化ベンチマークとその結果について議論する．5 章では Fortran 90 の derived types 定義によるデータのカプセル化，OpenMP によるノード内スレッド分割による並列化について議論する．6 章では 5 章で用いた最適化 SPIC コードのベンチマークと PVM による単純並列化ベンチマーク結果との比較を行う．さらに 7 章では HPF コードとのベンチマーク比較を行う．8 章ではこれらの結果について議論しまとめを行う．

2. Skeleton PIC コード（SPIC）について

1, 2, 3 次元 Skeleton Particle-In-Cell（SPIC）コードは UCLA の Decyk 教授によって提案された大規模並列計算機のパフォーマンス計測用および新並列アルゴリズムのテスト用の粒子シミュレーションコード[3] で，並列計算機のための新しいアルゴリズムやアーキテクチャの性能評価のために開発されたコードである[3),9)]．現在，数値トカマクシミュレータ開発プロジェクト[3),9)] などに用いられている．SPIC は**図 1** のように粒子シミュレーションを行ううえで，必要最小限の機能のみが含まれているが，性能評価に必要な高速化のボトルネックとなる要素はすべて含まれている[3]．粒子は 1 種類，すなわち，電子のみ用いられている．粒子は静電力のみで動く静電近似を用い，電磁力項は無視し，マックスウェル方程式のうちポアッソン方程式のみを FFT を使い数値的に解いている．境界条件は周期境界条件を用いることとする．本並列シミュレーションでは**図 2** のように 2 次元の場合シミュレーション・ドメインを各プロセッサへ 1 次元ドメイン分割を行って計算することとする．3 次元の場合も同様である．

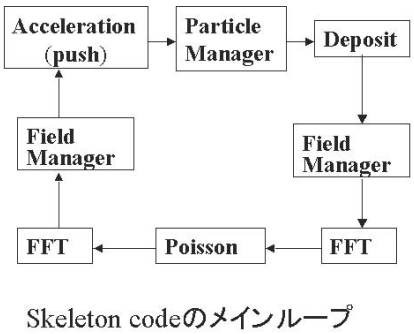

図 1 SPIC の構成
Fig. 1 Structure of SPIC code.

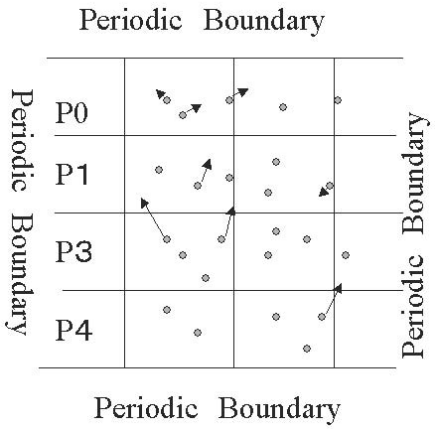

図 2 SPIC コードのドメイン分割．P1，P2，P3，P4 はそれぞれプロセッサ番号
Fig. 2 Domain decomposition of SPIC code. P1, P2, P3, and P4 denote the processor numbers, respectively.

3. PC クラスタ・システムについて

この実験ではヒューレット・パッカード社のパーソナルコンピュータ HP Vectra XU（デュアル PentiumPro による 2 ウェイ SMP 構成）を 1〜16 台使用した．HP Vectra XU は 200 MHz の Pentium Pro プロセッサを 2 個，128 MB のメインメモリを搭載している．OS は Linux（カーネル 2.210）を使用．ネットワークは，100 BASE-TX（全二重 100 Mbit/sec，ロードストア方式）のスイッチ PLANEX 社製 FDX-242C を使用して，クラスタを構成した．また，フォートランコンパイラは Portrand Group's 社製 pgf77，pgf90，pgHPF を使用した．

4. メッセージパッシングによる並列処理実験

この並列粒子シミュレーションでは，主に 2 次元の SPIC を使い，2 ウェイ・デュアル Pentium Pro を搭載した PC ノードを 1〜16 台使い，1 プロセッサから 32 プロセッサの並列処理を行った．ここで，1〜16 プロセッサまでは，1 台の PC ノードに 1 タスクずつ立ち上げるようにした．1 台の PC に 2 PVM タスクを立ち上げ得た場合，1 タスクのみ立ち上げた場合より，計算実行時間は平均約 8〜10%増えることを実験で確認している．以降この問題では，1 台の PC をノードもしくは PC ノードと呼ぶことにする．問題サイズを変え，PVM による並列シミュレーションを行い，その並列化効率と問題点などを検討するため以下の (1)，(2) の条件で実験を行った（**図 3**）．実験 (1) 領域 64×128 グリッド，粒子数 90/グリッド，(2) 領域 128×256 グリッド，粒子数 109/グリッド．

この実験では論文 3) にあるとおり現実にビームプラズマ不安定をシミュレーションしており，実際の物理シミュレーションである．ビームプラズマ不安定のため各プロセッサへの負荷バランスが計算量にして最大 20%程度ばらつく[3]．領域サイズによらず，8 プロセッサまでは，スピードアップ[10] がほぼ線形に上昇しているが，16，32 プロセッサと増やしていくと，十分な線形性が得られていない．特に，領域サイズの小さい実験 (1) のときに顕著である．これは，問題規模が小さい (1) のときは問題サイズに対する通信遅延によるオーバヘッドの割合[11),12)] がより大きいためと考えられる．もちろん，問題サイズを大きくしていくことにより，通信遅延によるオーバヘッド部分の割合をおさえることは可能であるが，並列効率[11),12)] は低くおさえられたままである．ここで，スピードアップをはかるには，根本的に相対通信コストを低くおさえることが必要になる．その方法について，次章で述べる．

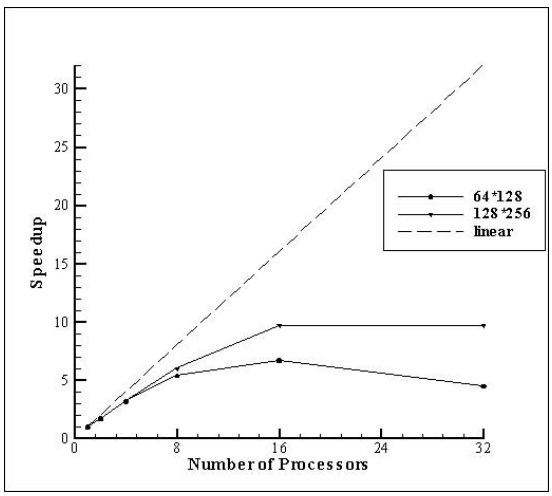

図 3 実験 (1) 領域 64×128 グリッド，粒子数 90/グリッド，(2) 領域 128×256 グリッド，粒子数 109/グリッド
Fig. 3 Experiments (1) 64×128 Grids, 90 particles/cell; (2) 128×256 Grids, 109 particles/cell.

5. SPIC の最適化

5.1 OpenMP の使用

PC クラスタではイーサネットスイッチなどでネットワークを構成するため，PC ノードの数を増やすうえでスイッチの数などのハードウェア上の制限がある．そこで，共有メモリ型の 2 ウェイ PC の利点を生かすためノード間では分散メモリ型の並列化としてメッセージパッシングを用い，**図 4** のようにノード内では共有メモリ型としてマルチタスク（スレッド分割）を行う手法が有効である．本研究ではノード内の通信を削減することにより並列化の効果を高めるため OpenMP と PVM を併用した．PVM と OpenMP で並列化を行う際，基本的には，まず PVM を使って並列化されたプログラムの各並列化可能なループを OpenMP を使ってスレッド分割して並列化を行う．SPIC でスレッド分割する際，最も重要となるのが，図 1 における「acceleration」サブルーチンと「deposit」サブルーチンである．SPIC の処理を大きく分けると 4 つある．1 つは，電場を周波数領域で解くサブルーチン「Poisson」，「FFT」．2 つめは，電場から生じる力をスプライン補間により粒子に伝え，粒子を加速し，動かすサブルーチン（「acceleration」）．3 つめは，粒子情報から場を解くためグリッドに電荷の分配を行うサブルーチン（「deposit」）．4 つめは FFT と逆 FFT サブルーチン（FFT）特に，「acceleration」で全体の約

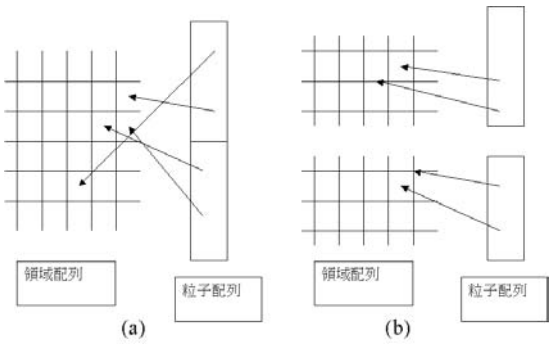

図 4 OpenMP のためのノード内領域分割を (a) 行わないケースと (b) 行ったケース
Fig. 4 OpenMP (a) nondecompositions and (b) decompositions in a node.

50%，「deposit」で全体の約 30%の計算コストがかかる．この 2 サブルールーチンの計算特性は，他の静電近似を使っていない完全電磁粒子シミュレーションでも本質的に同じで，重要度と一般性の高い部分である．そこで，この 2 つのサブルーチン「acceleration」，「deposite」の処理の最適化が最も重要となり，この部分に関して OpenMP によるノード内並列化を試みる．

先行研究[11),12)] にあるとおり我々の過去の実験では，下記の単純なループ分割によるノード内マルチタスク化では，ノード内通信の高速化，スピードアップ向上は，OpenMP を使うためのソフトウェウア・オーバヘッドによっておさえられ，良い結果は得られないことが知られている[11),12)]．以下は，OpenMP を用いたマルチタスク化である．

```
subroutine push
dimension part(idimp,npmax,nvp)
           ! nvp is the number of processors
           !  on SMP
sbufl(idimp,nbmax,nvp),sbufr(idimp,nbmax,nvp)
!$OMP PARALLEL
!$OMP DO
do i=1,nvp
do j=1,npp(i)   ! npp(i) is the number of particle
 …              ! here move the particles
!define particles which move to other processors

enddo
!$OMP END DO NOWAIT
!$OMP END PARALLEL
end

subroutine pmove  ! this subroutine move particles
                  ! to other processors
…
end
```

全体の並列化処理の流れの概略を**図 5** に示す．

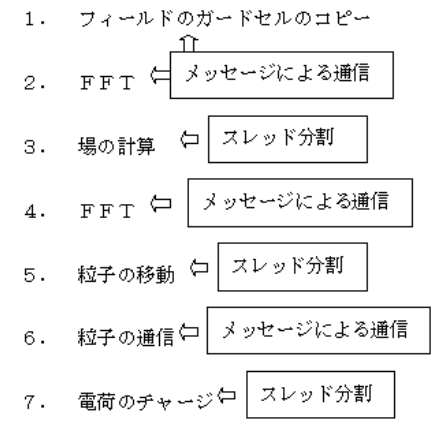

図 5 並列化処理の流れ
Fig. 5 Our parallelization procedure.

5.2 タスク内における領域分割

前節で述べた，並列化を行ううえでコードの各部分を単純に文献 11)，12) のようにスレッド分割していく方法では，いくつかの問題点が残る．1 つは，「deposit」サブルーチン（以降 deposit とする）でいったんドメイン分割された 2 つの配列から 1 つに足しあわせなければならない点である．これは，ループ内で粒子配列について分割するためで，通常の行列演算ではドメイン分割されたデータを読み込む配列の場所と，書き込む配列の場所が，静的に決まるためその関係を考慮してループを分割すれば，アクセスの衝突が起こらず書き込み配列を OpenMP シェア（Share）[7)] 定義すればよい．しかし，粒子シミュレーションで粒子はシミュレーションの時間経過につれて，粒子は分割されたドメインを越えて動的に移動し，結果的に読み込まれたデータ（粒子の位置情報）と deposit 書き込みのデータの場所（領域上の位置）がドメインを越えて動的に変化する．そのため書き込みの配列を OpenMP のシェア（Share）[7)] で定義すると分割された領域外へのアクセスが起こり，計算時間が遅くなる．現在使用している OpenMP ではこのような配列に対するスレッド分割終了時に指定の演算子で同期して適当なリダクション演算を行う機能をサポートしていないため，領域の配列をスレッドもしくはタスクの数作り，ループ終了後にガードセルもしくはゴーストセル[3)] を足しあわせる作業が必要となる．このためメッセージパッシングのタスクごとにドメイン分割された粒子配列をさらに分割して，各ノードで OpenMP スレッド数作ることで問題が解決できる．配列をスレッド数の次元だけ多重化し，領域分割を行う方法も考えられるが，データ構造が複雑で我々の実験では十分にパフォーマンスが得られていない．その原因として，SMP 内でデータ共有したときにコンパイラがそれぞれのスレッドの占有データであると認識できないためとも考えられるが，原因は不明である．データをスレッド数で多次元化した場合，データ書き込みの衝突は起こらないので計算の安全性は保証されるが，1 つの配列に同時にアクセスするため，パフォーマンスの減少が見られる．また，データの読み込みだけならばあまり問題とならないが，書き込みが何度も起こる配列を共有化した場合，問題となる．これを避けるためには，スレッドごとにそれぞれ該当する配列や変数について複数定義していくことが必要となるが，非常に煩雑になる．そこで，Fortran90 の derived types 定義[5)] を用い，領域ごとのオブジェクトを定義し，それをスレッド数生成してデータの局所性と処理の効率化を図る．各ノード

で OpenMP を用いないとき，配列は生成されない．

粒子が動的にノード間を移動する通信では，ノード内ではメッセージ・パッシングせずにバッファを交換し，ノード外とはメッセージパッシングでバッファをやりとりする．このバッファ交換処理は並列性が高く，通信の量も増えず，大きなコストとはならない[3]．また，ドメイン領域データ処理に関してもガードセルもしくはゴーストセル[3] のコピーの際，同様の処理を行う．

OpenMP によるスレッド分割で並列処理するときは，下記のとおり，SECTION 構文を使いタスク並列を行う．

```
!$OMP PARALLEL
!$OMP SECTIONS
!$OMP SECTION
do j=1,particle1% number_of_particles
field1%q=

!$OMP SECTION
do j=1,particle2% number_of_particles
field2%q=

!$OMP END SECTIONS
!$OMP END PARALLEL
```

上記で number_of_particles，q はそれぞれ下記のように定義された F90 の derived types である[5] スレッドの数に応じて必要な変数を初期化して生成する．

```
type species_descriptor
        integer :: number_of_particles
        ..............
end type species_descriptor

type grid
     real, dimension(nx,ny)::q,fx,fy
end type grid
```

6. PVM と OpenMP を併用した並列処理実験

本章ではまず，PVM と OpenMP を併用し，5 章で最適化された SPIC コードを用いて PC クラスタ上でベンチマークテストを行い，4 章の単純タスク並列化コードのベンチマークテスト結果との比較を行う．

6.1 最適化コードのシミュレーション

オブジェクト指向化と PVM と OpenMP を併用した最適化コードを使用し，(1) 領域：64 × 128 グリッド，粒子数：90/グリッド，(2) 領域：128×256 グリッド，粒子数：109/グリッド，の 2 つの条件でタイムステップ 325 でシミュレーションした．その際，比較のため，OpenMP でスレッド分割せずに行ったコードの結果とあわせて示す．また，比較のためスレッド分割したコードは各ノードにつき 1 PVM タスク 2 スレッド，スレッド分割をしていないコードは各ノードに 2 PVM タスクを配置した．

OpenMP を使用した場合，**図 6**，**7**，**8** のようにノードを増やしていたときに，ノード内通信の削減により，通信の遅延をおさえることで OpenMP を使用しないコードよりも線形性の高いスピードアップが得られた．しかし，問題規模があまり大きくない (2) のケースで 16 プロセッサに比べて，32 プロセッサの全体のスピードアップ性能が落ちるなどの問題がある．これは，全計算時間に対する通信オーバヘッドの割合が大きいためである．問題規模を大きくしたとき (2) のケースでは，16 よりも 32 プロセッサで若干良いスピードアップ・パフォーマンスとなったが，OpenMP を使用した効果が十分とはいえない．さらにプロセッサ数が少ないときには，OpenMP を使用しないときよりもスピードアップ性能が低くなっている．これは，問題が大きいときには，よりスレッド分割していない電場についての処理時間の割合が全体に対して大きくなり，その部分の並列化の効果が十分でないためと OpenMP のオーバヘッドの割合が小さくなるためと考えられる．問題規模の大きいときと小さいときを比較すると，小さいときは計算に対して大きな通信遅延があり，より OpenMP による効果が顕著に現れるが，

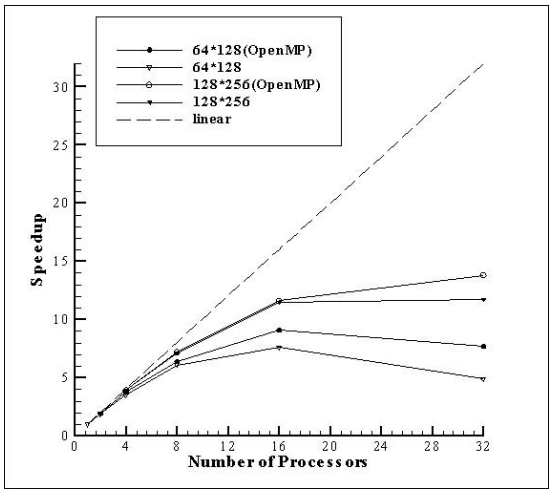

図 6 台数効果
Fig. 6 Linearity of speedup.

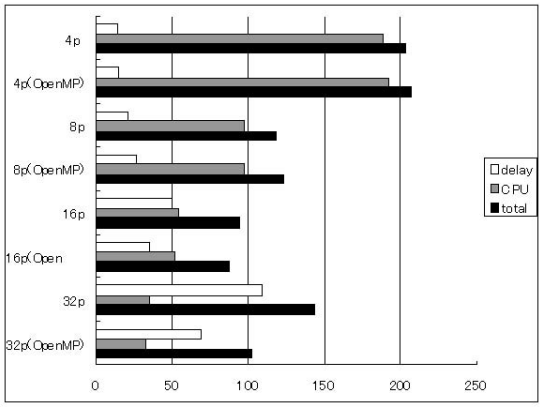

図 7 条件 (1) の結果：横軸は時間（秒）を表し，CPU はプロセッサの実行時間，delay は通信の遅延時間，Total は CPU と delay をあわせたものを表す

Fig. 7 Benchmark results of case (1): x axis denotes time (sec.). “CPU”, “delay”, and “Total” are denoted by processor execution time, communication delay time, and total time of both “CPU” and “delay”, respectively.

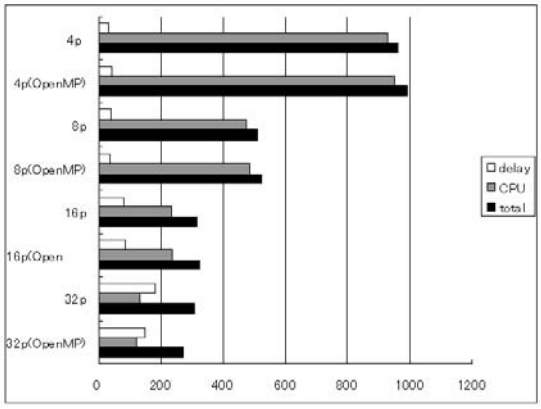

図 8 条件 (2) の結果：横軸は時間（秒）を表し，CPU はプロセッサの実行時間，delay は通信の遅延時間，Total は CPU と delay をあわせたものを表す

Fig. 8 Benchmark results of case (2): x axis denotes time (sec.). “CPU”, “delay”, and “Total” are denoted by processor execution time, communication delay time, and total time of both “CPU” and “delay”, respectively.

問題規模が大きいときは通信遅延の時間が全体の時間に占める割合が少なく，OpenMP を使った効果がそれほど顕著でない．通信の遅延を改善する方法としては，メッセージ送信時にパイプライン化を行うソフトウェア的方法と，スイッチのスループット改善などハードによる方法がある．さらに，ハードでの改善策として，ノード内のプロセッサ数を 4 ウェイ，8 ウェイと増やし，OpenMP による効果をさらに引き出す方法などが考えられる．

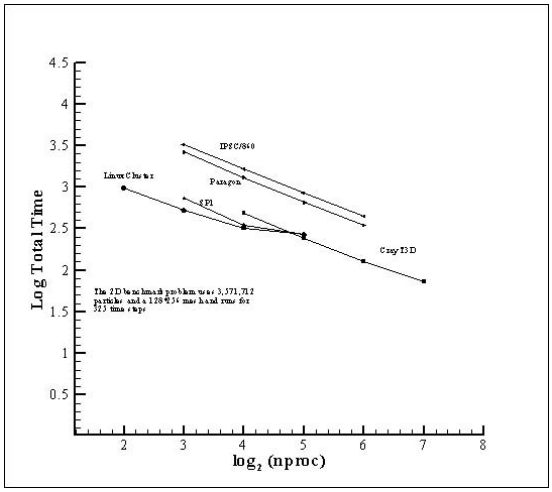

図 9 ベンチマークテスト

Fig. 9 Benchmark test.

6.2 ベンチマークテスト

Decyk 教授らが行ったのと同じ条件，粒子数 3，571，712 個，領域 128×256 グリッド，325 タイムスッテプで倍精度計算により 4，8，16，32 プロセッサで実験した[3]．その結果を Decyk 教授らが行った結果[3] とあわせたものを，**図 9** に示す．比較対照となっているのは主に PentiumPro 発表時に稼動していた商用の並列計算機である．

7. HPF コードとの比較

前章では，PVM と OpenMP を併用することで，CPU を多く利用してもスピードアップの線形性を下げないなど，ノードの台数効果が高まることを示したが，メッセージパッシングとマルチスレッドを同時に使うことはプログラムを複雑にし非常にプログラミングコストが高くなる．そこで，より簡単にクラスタ上で並列化する方法として HPF（High Performance Fortran）[8] を使う方法が考えられる．文献 11) によれば，PVM に比べ HPF コードは 5〜10%計算速度が遅くなる．そこで，2 次元の SPIC を HPF を使い並列化し，先ほどの PVM と OpenMP とを同時に使ったコードと比較した（**表 1**）．領域 64×128 グリッド，粒子・40/グリッドで 2，4，8，16 プロセッサでシミュレーションを行った．

実験の結果，HPF コードは OpenMP と PVM を使ったものより 10%から 60% CPU 時間が遅くなる．HPF を使うことで，メッセージパッシングを使うよりも容易に並列化が可能だが，パフォーマンスの面では不満が残る．これらの結果より，ノード内の通信処

表 1 HPF コードとの比較（Total [sec.] (CPU/Delay)）
Table 1 Compariosns with HPF code (Total [sec.] (CPU/Delay)).

	2PE	4PE	8PE	16PE
HPF	902	477	293	243
	(882/20)	(450/27)	(223/70)	(121/122)
OpenMP	793	409	227	158
+ PVM	(780/13)	(377/32)	(187/40)	(96/62)

理はやはり，6 章のようにメッセージパッシングにせよ HPF にせよ PC クラスタでは OpenMP の併用が望ましいことを示している．

8. ま と め

並列処理実験では，比較的安価な PC クラスタを用いて並列粒子シミュレーションを行っても，32 プロセッサ程度であれば，大型並列計算機に匹敵する実験が行えることも実証できた．しかし，ネットワークにイーサネットスイッチを用いた場合，16 ノード以上で行う場合 SMP-PC クラスタ上でよりパフォーマンスを引き出すために，OpenMP を使い通信のオーバヘッドをおさえ，スイッチへの負荷を減らす必要があることを確認した．また，HPF を使ったコードとの比較により，スピードアップの線形性では，HPF は必ずしも有効ではないとの結果を得た．メッセージパッシングとスレッド分割の併用で効果的なスピードアップを得るのはオブジェクト指向言語を用いてもプログラムコストがかかり容易ではない．並列性能とプログラムの平易さを考え，HPF とメッセージパッシングと OpenMP の併用どちらが優れているかは一概にいうことは難しい．開発プログラムの目的により検討が必要と思われる．これらの性能比較の結論は，数値トカマク，宇宙気象シミュレータなどの大規模プラズマ粒子シュミレーションコードを PC クラスタ，もしくは類似のアーキテクチュアの計算機で最適化する際に有効な指針となると考えられる．

参 考 文 献

1) Liewer, P.C. and Decyk, V.K.: A General Concurrent Algorithm for Plasma Particle-in-Cell Simulation Codes, *J. Comput. Phys. 85*, pp.302–322 (1985).
2) Dagum, L.: *Parallel CFD: Implementations and Results Using Parallel Computers*, MIT Press, Cambridge, Mass. (1992).
3) Decyk, V.K.: Skelton PIC codes for parallel computers, *Comput. Physcs. Comm.* 87, pp.87–94 (1995).
4) Stering, T.L.: *How to build a beowulf*, MIT Press (1998).
5) Decyk, V.K., Norton, C.D. and Symanski, B.K.: Object-Oriented Concepts Using Fortran90, Technical Report PPG-1560, 1996; Norton, C.D., Szymanski, B.K. and Decyk, V.K., Object Oriented Parallel Computation for Plasma PIC Simulation, *Comm. ACM*, Vol.38, No.10, p.88 (1995).
6) Geist, A., et al.: *PVM*, MIT Press (1994).
7) OpenMP specifications (Fortran version2), http://www.openmp.org/specs/ (2000).
8) Koelbel, C.: *The high performance fortran handbook*, MIT Press (1993).
9) Dawson, J.M., Decyk, V.K., Sydora, R. and Liewer, P.: High-Performance Computing and Plasma Physics. *Physics Today*, 46.64 (1993).
10) Kumar, V., Grama, A., Guputa, A. and Karypis, G.: *Introduction to parallel computing*, Benjamin/Cummings Publishing (1994).
11) Cai, D., Lu, Q.M. and Li, Y.T.: Scalability in Particle-in-Cell code using both PVM and OpenMP on PC Cluster, *Proc. 3rd Workshop on Advanced Parallel Processing Technologies* pp.69–73 (1999); Akarsu, E., et al.: Particle-in-cell simulation codes in High Performance Fortran. http://www.bib.informatik.th-darmstadt.de/sc96/AKARSU/.
12) Cai, D. and Quanming Lu: Parallel PIC code using Java on PC Cluster. *Proc. HPC Asia 2000*, pp.495–500 (2000).

(平成 13 年 4 月 27 日受付)
(平成 13 年 9 月 17 日採録)

蔡 東生（正会員）

昭和 34 年生．平成元年東京大学大学院工学研究科博士課程航空学専攻修了．昭和 60 年スタンフォード大学電気工学科研究助手．同年神戸大学工学部助手．平成 4 年より筑波大学電子・情報工学系助手．平成 5 年より同講師．平成 9 年より同助教授．並列シミュレーション，可視化，非線形数理の研究に従事．PhD.，工学博士．AGU，ACM，日本応用数理学会各会員．

安室 秀則（正会員）

昭和 50 年生．平成 11 年筑波大学大学院工学研究科中退．同年日本ユニシス入社．並列シミュレーションの研究に従事．

李 尭亭（正会員）

昭和 37 年生．平成 4 年北京大学修士課程修了．平成 7 年中国科学院博士課程修了．平成 7 年〜平成 9 年中国科学院物理研究所 Post Doctor．平成 9 年〜平成 10 年筑波大学ベンチャービジネスラボ特別研究員．平成 10 年〜平成 12 年日本学術振興会特別研究員．平成 12 年〜平成 13 年筑波大学外国人研究者．並列シミュレーションの研究に従事．中国空間科学学会．

陸 全明（正会員）

昭和 44 年生．平成 5 年中国科技大学地球和空間学系修士課程修了．平成 8 年中国科技大学地球和空間学系博士課程修了．平成 8 年〜平成 9 年中国科学院上海光学和精密機械研究所．Post Doctor．平成 9 年〜平成 12 年筑波大学ベンチャービジネスラボ特別研究員．並列シミュレーションの研究に従事．中国空間科学学会．

肖 池階（正会員）

昭和 50 年生．平成 13 年中国北京大学大学院博士課程空間物理専攻卒業，博士学位取得．現在筑波大学ベンチャービジネスラボ特別研究員．太陽風・地球磁気圏の数値シミュレーションに関する研究に従事．中国空間科学学会．