計算機アーキテクチャ 137－22
ハイパフォーマンス 80－22
コンピューティング
（２０００．３．３）

PCクラスタを用いた並列粒子シミュレーション

蔡 東生† 李 尭亭† 陸 全明† 安室 秀則‡
†筑波大学 電子・情報工学系 ‡筑波大学 理工学研究科

概要

本研究では、安価なPCを１６台使いクラスタシステムを構成し、これまで大型並列計算機上で行われてきたUCLAのV.K.Decyk教授らによって書かれた、Skeleton particle-in-cell(PIC) codesによる実験を行なった。また、ＰＣクラスタ上でＰＶＭによるメッセージパッシングとＯｐｅｎＭＰによるスレッド分割を併用することで、通信遅延によるパフォーマンスの減少をより低く押さえる手法をＰＩＣコードに適用した。また、最適化したコードを使い、ベンチマークテストを行ない大型計算機による結果と比較した。さらにＨＰＦで並列化したコードとパフォーマンスを比較した。

Parallel Particle Simulation using PC Cluster

DongSheng Cai , Yaoting Li , Quanming Lu,
Institute of Information Science and Electoronics,University of Tsukuba
Hidenori Yasumuro
Science and Engineering,University of Tsukuba

Abstruct-We built a dual PentiumPro PC cluster. In this report, using a Skeleton particle-in-cell(PIC) codes that is developed by Professor V.K.Decyk of UCLA for a tested and benchmarking purposes, we have measured the performances of the Skeleton-PIC-code using PVM, and OpenMP. We have compared the optimized code with HPF code about performances.

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、これまでスーパーコンピューター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PCを使って可能となってきた。

この実験ではUCLAのViktor K.Decyk教授によって書かれた、大規模 並列計算機のパフォーマンス計測用の粒子シミュレーション コードskeleton particle-in-cell（PIC） codeを使い、Dual Pentium Pro プロセッサ搭載のSMP-PCを１６台、100BASE-TX イーサネットスイッチで接続してクラスタシステムを構成し、そのシステム上で並列粒子シミュレーションを行った。

2．Skeleton PIC codes について

１、２、３次元 Skeleton particle-in-cell(PIC) codes はUCLAのViktor K.Decyk教授によって書かれた、大規模 並列計算機のパフォーマンス計測用の粒子シミュレーション コードで、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や開発のためのものであ

る。数値トカマクシミュレータ開発等にも用いられている。PICコードは必要最小限に作られているが、性能評価に必要な要素はすべて含んでいる。粒子は運動や場のエネルギーを分析するためだけに用いるため1種類だけ（電子）使う。PICコードは静電気力は近似値を使い、磁場は無視する。境界条件は周期的とする。本並列シミュレーションでは図1の様に2次元のPICコードのシミュレーションドメインを各CPUへ1次元ドメイン分割を行って計算する。

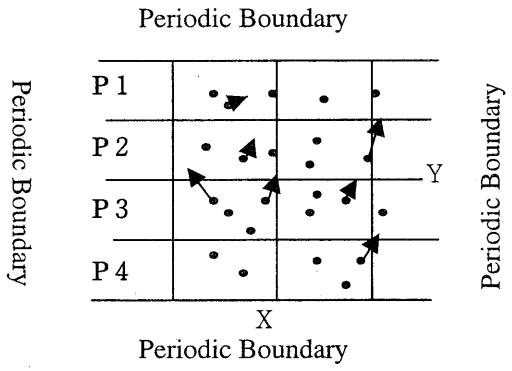


図1：PICコードのドメイン分割

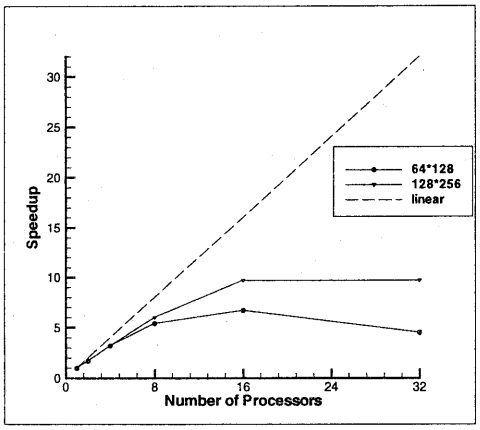
3．Clusterシステムについて

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(2CPU SMP)を1～16台使用した。HP Vectra XUは200MHzのPentium Proプロセッサを2個、128MBのメインメモリーを搭載している。OSはLinux(kernel 2.2.10)を使用。ネットワークは、100BASE-TX（全二重 100Mbit/sec,ロードストア方式）のスイッチ PLANEX FDX-242Cを使用して構成している。

フォートランコンパイラはpgf77,pgf90, pgHPF(Portrand Group,Inc)を使用している。

4．並列処理実験（1）

この並列粒子シミュレーションでは、2次元のＰＩＣコードを使い、2CPU Pentium Proを搭載したパーソナル コンピュータを1～16台使い、1プロセッサーから32プロセッサの並列処理をおこなった。問題サイズをかえ並列シミュレーションを行ない、その並列化効率と問題点などを検討するため以下の条件で実験を行なった。



（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

図2：実験（1）

領域サイズによらず、8プロセッサまでは、並列化効率が上がっているが、16、32プロセッサと増やしていくと、十分な台数効率が得られていない。特に、領域サイズの小さいときには顕著である。これは、問題規模が小さい時は通信遅延による影響が大きいためと思われる。これを改善するためには、通信コストを押さえる事が必要になる。その方法について、次節で述べる。

5．コードの最適化

5.1 OpenMPの使用

PCクラスタではイーサネットスイッチでネッ

トワークを構成するため、ノードの数を増やす上でスイッチの数などのハード的制限がある。そこで、SMP-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行ない、図３のようにノード内では共有メモリ型としてスレッド分割を行なう手法が有効である。本研究ではノード内の通信を削減により並列化の効果を高めるため OpenMP と PVM を同時に用いた。

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なう。PIC コードでスレッド分割する際もっとも重要となるのが、acceleration サブルーチンと deposit サブルーチンである。ＰＩＣコードの処理を大きく分けると３つある。１つは、電場についてフーリエ領域で場を解くセクション。２つめは、粒子を動かすセクション（acceleration)。３つめは、粒子から場へ電荷の分配を行なうセクション（deposit)。特に、acceleration で５０％、deposit で３０％のコストがかかる。また、他の粒子シミュレーションでも重要となる、一般性の高い部分である。そこで、その２つのサブルーチンの処理の最適化がもっとも重要となる。

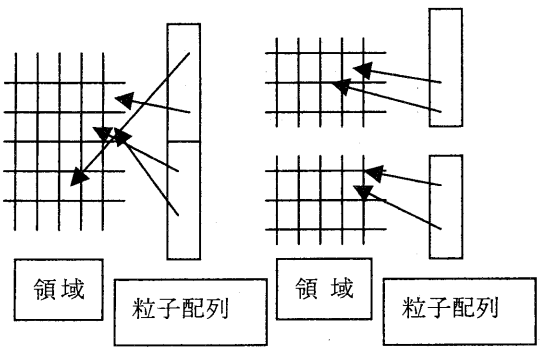


図３：タスク内領域分割

全体の処理の並列化の流れ

１．　フィールドのガードセルのコピー
２．　ＦＦＴ　⇦ メッセージによる通信
３．　場の計算　⇦ スレッド分割
４．　ＦＦＴ　⇦ メッセージによる通信
５．　粒子の移動　⇦ スレッド分割
６．　粒子の通信　⇦ メッセージによる通信
７．　電荷のチャージ　⇦ スレッド分割
（１～７を繰り返す）

５．２ タスク内領域分割

前節で述べた、並列化コードの部分、部分を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。一つは、Deposit サブルーチンで一旦、二つの配列から一つに足しあわせなければならない点である。これは、ループをする時、粒子配列について分割するためで、通常の行列演算では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の配列をシェアで定義すれば良い。しかし、粒子シミュレーションでは読み込まれたデータ（粒子の位置情報）と書き込みのデータの場所（領域上の位置）が動的に変化するため書き込みの配列をシェアで定義することではアクセスの衝突がおこる。現在使用している OpenMP 対応のコンパイラでは配列に対するリダクション演算（スレッド分割終了時に指定の演算子で同期）をサポートしていな

いため、領域の配列をスレッドの数作り、ループ終了後に足しあわせる作業が必要となる。このためタスクごとに分割された粒子配列をさらに分割して、スレッドの数作る事とで問題が解決できる。配列をスレッドの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、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時にメモリバス容量か、メモリコンフリクトが原因と思われるが不明である。

データをスレッドの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、一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。データの読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事が必要となるが、非常に煩雑になる。そこで、Fortran90 によるオブジェクト指向化を行い、領域ごとのオブジェクトを定義し、それをスレッドの数、生成して処理の効率化を計っている。

その際、粒子の移動の通信で、ノード内では通信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やり取りする。この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。また、領域データに関してもガードセルのコピーの際、同様の処理を行う。

スレッド分割で並列に処理する時は、下の様に、SECTION 構文を使いタスク並列で行なった。

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

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

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

6. 並列処理実験（2）

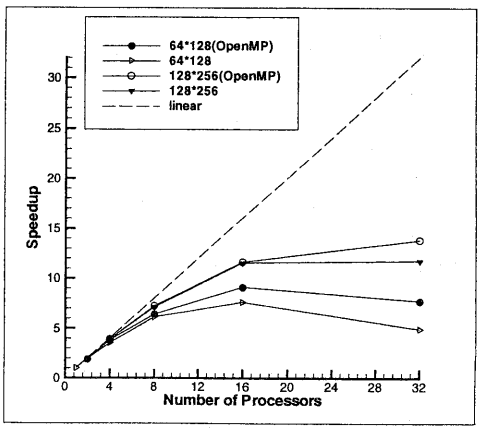
6.1 最適化コードのシミュレーション

オブジェクト指向化とＯｐｅｎＭＰを使い最適化したコードを使用し

（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 でスレッド分割をせずに行ったコードの結果と合わせて示す。また、スレッド分割したコードは各ノードにつき１タスク、スレッド分割をしていないコードは各ノードに２タスクを配置した。



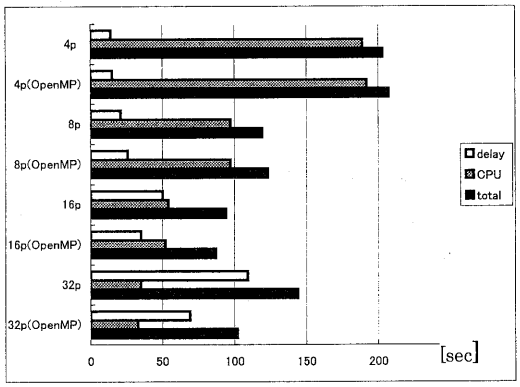


図５：条件（１）の結果

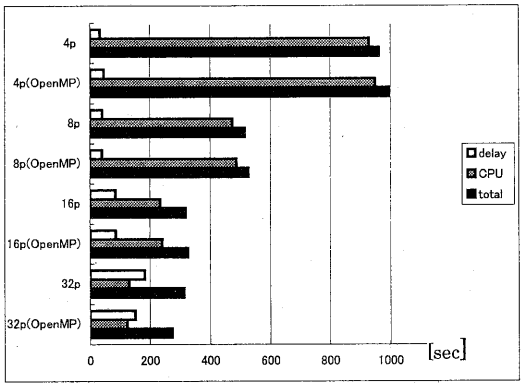


図６：条件（２）の結果

横軸は時間（秒）を表し、CPU は CPU の実行時間、delay は通信の遅延時間、
Total は CPU と delay を合わせたものを表す。

考察

OpenMP を使用した場合、図４、５、６の様に予想されたとおりノードを増やしていた時に、ノード内通信の削減により、通信の遅延を押さえる事で OpenMP を使用しないコードよりも高いパフォーマンスが得られた。

しかし、問題規模があまり大きくないときに１６プロセッサに比べて、３２プロセッサのトータルのパフォーマンスが落ちるなどの問題がある。これは、通信オーバーヘッドの為である。

問題規模を大きくした時は、１６よりも３２プロセッサで若干、良いパフォーマンスとなったが、OpenMP を使用した時の効果が十分とはいえなくなった。さらにプロセッサ数が少ない時には、OpenMP を使用しない時よりもパフォーマンスが低くなっている。これは、問題が大きい時には、より場についての処理が大きくなり、その部分の並列化の効果が十分でないためと思われる。

問題規模の大きい時と小さい時を比較すると、小さい時は計算よりも大きな通信遅延があり、より OpenMP による効果が現れるが、問題規模の大きい時は通信遅延の時間が全体の時間に占める割合が少なく、効果が若干落ちたと考えられる。

通信の遅延を改善する方法としては、メッセージを送るタイミングなどによるソフトウェアによる方法と、ハードによる方法がある。

コードの中でメッセージを送るタイミングの調整などを行ったが、性能の改善は見られ無かった。

ハードでの改善策として、スイッチを更に高性能のものに変える。ノード内のプロセッサ数を増やしノードの数を減らし、OpenMP による効果を更に引き出す方法などが考えられる。

6.ベンチマークテスト

V.K.Decyk 教授らが行なったのと同じ条件、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。その結果を V.K.Decyk 教授らが行なった結果と合わせたものを、図７に示す。

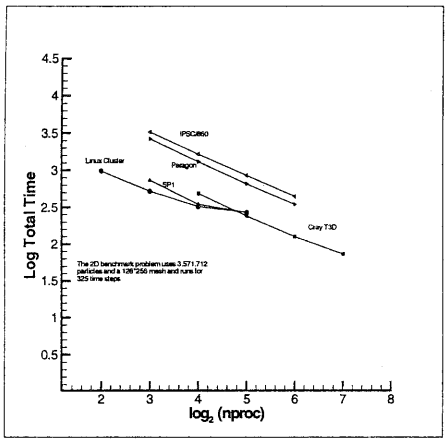


図７：ベンチマークテスト

7. HPF コードとの比較

前節では、PVM と OpenMP を使うことで、より台数効果が高まる事を示したが、メッセージパッシングとマルチスレッドを同時に使う事はプログラムが複雑で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)を使う方法が考えられる。そこで、2 次元の PIC コードを HPF を使い並列化し、先程の PVM と OpenMP とを同時に使ったコードと比較した。

領域 64×１２８グリッド、粒子数 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。

表１：ＨＰＦとの比較

Total [sec]
(CPU/delay)

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

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。

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事も実証出来たと思われる。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 node 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ーヘッドを押さえる必要がある事を確認した。また、HPF を使ったコードとの比較により、パフォーマンスの面では、HPF は必ずしも有効ではない結果を得た。しかしメッセージパッシングとスレッド分割でパフォーマンスを得るのは容易ではないため、どちらが優れているかは一概に言うことはできず目的により検討が必要と思われる。これは、大規模プラズマ粒子シュミレーションコード TristanCode を PC クラスタ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] VictorK.Decyk,
SkeltonPIC codes for parallel computers,
Computer PhysicsCommunications87,1995
[2] V.K.Decyk,C.D.Norton,and B.K.Symanski
Object-Oriented Concepts Using Fortran90.
Technical Report PPG-1560, July 1996
[3]DongSheng Cai,Quanming Lu,Yaoting Li,
Scalability in Particle-in-Cell codeusing both
PVM and OpenMP on PC Cluster,
Changsha,China,1999,pp.69-73.