ハイパフォーマンス
コンピューティング 72－7
（１９９８．８．６）

Pentium Pro Cluster を使った並列粒子シュミレーション

安室 秀則
筑波大学 理工学研究科
蔡 東生
筑波大学 電子･情報工学系

概要 — 本実験では UCLA の Viktor K.Decyk 氏によって書かれた、大規模 並列計算機用の粒子シュミレーション コード skelton particle-in-cell codes を使い、Pentium Pro プロセッサー搭載の PC を Linux (PC UNIX)によってネットワークを組んだシステムで並列計算を行ない、並列計算機を用いた場合とパフォーマンス比較を行なった。また、Viktor K.Decyk 氏が行なった RISC プロセッサーへの skelton PIC code の最適化の論文をもとに、Pentium Pro プロセッサーのための、チューンナップについて実験した。

Skelton PIC Code using Pentium Pro Cluster

Hidenori Yasumuro
Master's Program in Science and Engineering at University of Tsukuba
Dong Sheng Cai
Institute of Information Science and Electronics, University of Tsukuba

Abstract - This paper discusses two main subjects . The first one is about parallel particle simulation using the skelton PIC code. I used the skelton particle-in-cell code which was written by Professor Viktor K.Decyk of UCLA . I run it on Pentium Pro Cluster and compared the result with other commercial parallel computer timing results .The second part is about the optimization of the skelton PIC code that is the single processor version on a Pentium Pro Processor .

1 まえがき

近年、PC(パーソナルコンピューター) 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ュミレーションがPC を使ってできるようになってきた。この実験ではUCLA の Viktor K.Decyk 氏によって書かれた、大規模並列計算機用の粒子シュミレーション コードskelton particle-in-cell code を使い、Pentium Pro プロセッサー搭載の PC を Linux(PC UNIX) によってネットワークを組んだシステムで並列計算を行ない、並列計算機を用いた場合とパフォーマンス比較を行なった。

また、Viktor K.Decyk 氏が行なった RISC プロセッサーへの skelton -PIC code の最適化の論文をもとに、Pentium Pro プロセッサーのための、チューンナップについて実験した。

これらの実験は、さらに大規模な並列粒子シュミレーションを Pentium Pro Cluster を使ってやるための予備実験となり、課題や可能性を見つけ出すことができると考えられる。

2 並列シュミレーションについて

2.1 並列粒子シュミレーション実験についての概要

この並列粒子シュミレーションでは、後述する2次元 Skelton PIC code を使い、2CPU Pentium Pro を搭載したパーソナル コンピュータを1～8 台使い、並列処理の実験をした。

この実験の目的としては、一般的な並列計算用コンピュータよりも安価で身近なパーソナル コンピュータを複数使い、今まで並列計算用コンピュータでしか実験されてこなかった大規模並列粒子シュミレーションを行い、並列計算用コンピュータを用いた場合と比較し、どの程度の結果が得られるか調べることである。

この実験で使用したパーソナルコンピュータは2CPU SMP なので、1～8台同時に使い 、1プロセッサーから16プロセッサの並列処理をおこなった。

最初に粒子数を変えながら 1p(processor),2p ,4p,8p,16p で PIC コードを走らせ比較し、さらに4p,8p,16p で倍精度計算のベンチマークテストを行い、Viktor K.Decyk 氏らが行った並列計算機でのベンチマークテストと比較した。

2.2 Skeleton PIC codes について

1、2、3次元 Skelton particle-in-cell codes は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ーの性能評価や開発のためのものである。この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。コードは粒子を動かし、電荷を分配し、電場について解く。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種類だけ使う。この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。本並列シュミレーションでは2次元の Skelton PIC code のシュミ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

このコードは2つの異なる部分とデータ構造をもつ。1つは下の式で示すニュートンの力学法則で表される、粒子が場から受ける作用である。

$$\frac{dv_i(t)}{dt} = \frac{q_i}{m_i} E(x_i(t))$$

$$\frac{dx_i(t)}{dt} = v_i(t)$$

i はi 番目の粒子を表し、上の式を中点差分で解くと

$$v_i(t + \frac{dt}{2}) = v_i(t - \frac{dt}{2}) + \frac{q_i}{m_i} E(x_i(t))$$

$$x_i(t + dt) = x_i(t) + v_i(t + \frac{dt}{2})$$

電場における粒子の位置である $E(x_i)$ は、前に格子上で算出された電場からの補間によって求める。補間はギャザー操作であるために、計算時間がかなりかかる。一般的に線形補間が用いられるが、Skelton PIC code ではスプライン2次補間を用いている。

コードの2つ目の部分は、粒子によって形成される場を求めることである。skeleton code は下の式のようなポアソン方程式を使っている。

$$\nabla \cdot E = 4\pi\rho(x)$$

この方程式はフーリエ変換を使い格子上で解く。一般的に、場を解くのに時間はあまりかからない。もととなる $\rho(x)$ は下のように逆補間によって粒子の位置から算出する

$$\rho(x) = \Sigma q_i S(x - x_i)$$

S とは粒子の状態の関数である。ここで1次元配列から2次元配列へスキャター操作するため、ここで計算時間のかなりをしめる。

Skeleton code のメインループについて

1 粒子の調整は acceleration サブルーチン で行う

2 粒子を隣のプロセッサーに移すのは particle manager
サブルーチンが行う

3 格子上の粒子の電荷を貯めるのは、deposit サブルーチンで行う

4 ガードセルを field manager サブルーチンで隣接プロセッサ間でたし合せる

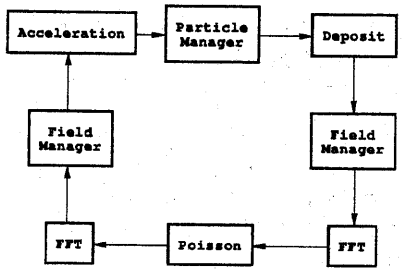

図 1: PIC code の main loop

5 電荷密度の実数から複素数への FFT は FFT サブルーチンが行う
6 電場はフーリエ空間で poisson solver サブルーチンで計算される
7 電場の複素数から実数への FFT は FFT サブルーチンが行う
8 ガードセルを field manager サブルーチンで分配する

2.3 この実験で使ったシステムについて

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1～8台使用した。
HP Vectra XU は200MHz の Pentium Pro プロセッサを2個、64MB のメインメモリーを搭載している。OS は Linux(PC UNIX) を使用してネットワークを構成している。
LAN の通信速度は、全二重 10Mbit/sec,switch でロードストア方式である。フォートランコンパイラは g77(GNU project Fortran Compiler v0.5.21) を使い、ベンチマークテストでは、倍精度計算のオプション-double, 速度向上のためのオプションとして-O3, さらに Pentium,Pentium Pro などの Intel386 アーキテクチャCPU 用の倍精度計算の速度向上のオプション-malign を使ってコンパイルを行った。
1つのシングル スレッド タスクを実行した場合、2つのプロセッサは、オペレーティング システムの仕事を分け合うことは出来るが、このタスクの全ての仕事は、1つのプロセッサによって遂行しなければならない。2つのシングル スレッド タスクを実行した場合、2つのプロセッサーは異なるタスクを実行する。
したがって、1つのホストで1つのタスクしか立ちあげなくても1つのタスクの実行時間が半分になるわではなく、また1プロセッサに1タスクとして実験するので、1つのホストで2つのタスクまでしか立ちあげない。

3 並列シュミレーション実験の結果について

3.1 実験の結果について

3.1.1 1～16プロセッサーでの比較

まず1～16プロセッサーで粒子数を変えて、実行時間を比較した。下の表に結果を示す。その際、SP とはシングル・プロセッサ・バージョンの事で、通信を全く行なわないコードである。1 Pは自分から自分へ、データを送る通信を行なっている。領域を32×64 グリッド、粒子数 A は粒子数 $(3 \times 3 + 1) \,/\, cell$ で、粒子数 B は $(6 \times 6 + 4) \,/\, cell$でタイムステップ325で実験した。

表1 :比較実験 単位 秒)

	粒子数 A	粒子数 B
SP	68.01	254.56
1p	72.33	259.32
2p	39.29	134.25
4p	22.41	71.58
8p	17.10	40.53
16P	15.47	28.70

また、1p～16p までの数値を
S:Speed up 指数

$$S = \frac{time_{Nproc}}{time_{1proc}}$$

という指標で表したグラフを、図2に示す。
まず、SP と 1p の違いについては、全く通信を行なわない SP より、自分から自分への通信を行なっている分だけ1 p が遅くなっていると考えられる。領域を固定して粒子数を増やした場合、通信の量の増加よりも計算量の増加が大きいため、粒子数が少ない時ほど差が開かなっかったと思われる。
1p～16p の違いは、粒子数が少ない時には、プロセッサの数を増やしていっても、通信の量はあまり減らず、逆に、より多くのタスクと同期を取ったりしなければならなくなる。したがって、計算量はへるが通信時間が減らないために、プロセッサを増やしていっても、あまり実行時間が減らないと考えられる。それに比べ、粒子数が多いときには、通信時間の全体に占める割合が、少ないので通信時間の変化よりも計算時間の減少が大きく実行時間に影響するので、プロセッサ数を2倍にすると実行時間は2分の1近くまで減少していると考えられる。

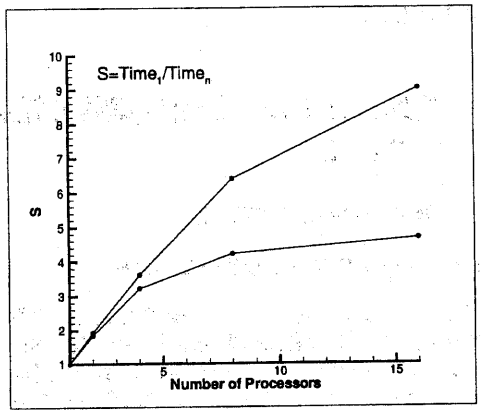

図 2:比較実験グラフ

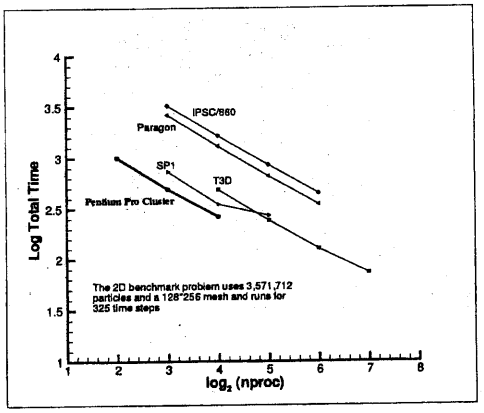

図 3:ベンチマークテスト

3.1.2 ベンチマーク・テスト

V.K.Decyk 氏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タイムステップで倍精度計算で４、８、１６プロセッサーで実験した。 領域、粒子数が大きいため、１、２プロセッサーではメモリー不足のため実行出来なかった）その結果を Decyk 氏らが行なった結果と合わせたものを、図３に示す。

この結果から考察すると、Cray T3D が 150MFlops の EV4 DEC Alpha processor を使っている事や、Pentium Pro processor が 250MFlops ということを考えると、マシンのアーキテクチャやコンパイラの性能の違いなどで、単純には比較できないが、妥当な結果が得られたと思う。

3.2 並列処理実験のまとめ

並列処理による粒子シュミレーションの実験は、当初、予想していた結果がだいたい得られたと思が、マシンのメモリー不足などにより、十分な実験が出来なかった。各マシンの状態を一定に保つことなど、大型計算機にはない問題もあると思う。しかし、身近なパーソナルコンピュータで大型並列計算機と同等の結果が出せたことには、満足出来ると思う。ただ１つの計算機ではなく、LAN 接続された複数の計算機を使うという形体を考えると、３２、６４とプロセッサーを増やしていくと、並列計算機に比べて通信の速度や、状態、信頼性などの問題が出てきて、パフォーマンスの大幅な向上は難しいと考えられる。しかし、コストパフォーマンスを考えると、かなりいい結果がでたと思う。

4 シングルプロセッサー用コードのチューンナップ

4.1 シングルプロセッサー用コードのチューンナップの概要

この実験で使った skelton-PIC code よりもさらに、大規模で複雑な粒子シュミレーションを行う場合、膨大な計算量が要求される。そのため計算機のアーキテクチャを考えた、コードのチューンナップが必要になる。この実験では、Viktor K.Decyk 氏らが行った、コードの RISC Processor への最適化の実験の論文をもとに、主にキャッシュメモリーの為の変数の配列化と、サブルーチン dpost のパイプライン処理の為の最適化に絞り、それぞれの Pentium Pro プロセッサへの有効性を検証した。

4.2 Pentium Pro について

インテル Pentium Pro プロセッサの特性

Pentium Pro プロセッサは、６４ビット外部データ・バスおよび３６ビット・アドレス・バスを使用する。このプロセッサは２つの８ KB レベル１キャッシュ・メモリ（命令用とデータ用）と２５６ KB レベル２キャッシュ・メモリを内蔵している。命令の実行用に３つのパイプラインと５つの実行ユニットを使用する。

スーパーパイプラインのアーキテクチャについて

Pentium Pro プロセッサの性能を向上させている核心はスーパーパイプライン・アーキテクチャである。これにより命令をより迅速に実行できる。各命令は、パイプラインと呼ばれる機構によりいくつかのステップを経て実行される。実際にはPentiumPro プロセッサは３つのパイプラインを使用し、これによって同時に２つ以上の命令を実行

することができる。PentiumPro の各パイプラインは最大１４の個別のステップで構成される。

順序外実行について

Pentium Pro プロセッサは、x86 命令をマイクロオペ (Micro-op) と呼ばれる単純な操作に分解する事によって、命令を実行する順序を変更することができるようになった。実行に平均より長くかかる特別な命令の場合には、最初の命令が完了するのを待っている間に、Pentium Pro は別の命令をフェッチして実行する。新しい命令が実行されている間、最初の命令のステータスはレジスタに格納され、衝突が生じないことを保証する。この機能は順序外実行とよばれ、これによって Pentium Pro の３つのパイプラインが最大効率で使用される事を保証する。

見込み実行 について

すべてのプログラムは分岐を含んでいる。分岐は条件命令によって作り出され、命令実行のスレッドを変化させることができる。条件付命令が実行される条件によって、プログラムにとって必要な命令が異なる。Pentium Pro は分岐予測ユニットを備えており、プログラム分岐を識別し、それらの命令をレベル１命令キャッシュ・メモリにロードする
。PentiumPro はそれぞれが必要な場合にはそれらの命令を実行することもできる。この機能は見込み実行と呼ばれ、プログラム分岐にかかる時間を大幅に短縮することができる。

4.3 チューンナップの方法

4.3.1 変数配列の多次元配列化

最初に、粒子配列のメモリ参照の間隔を改善することを行う。４つの粒子の位置、速度を表す配列を別々にせずに、１つの２次元配列にまとめることによって、１つのキャッシュラインに並びキャッシュミスヒットが減少すると考えられる。したがって下のように、粒子の位置の配列,x(j),y(j) と速度の配列 vx(j),vy(j) を１つの２次元配列 PART(4,j) に変更した。

```
PART(1,j) = x(j)
PART(2,j) = y(j)
PART(3,j) = vx(j)
PART(4,j) = vy(j)
```

4.3.2 パイプライン処理について

パイプライン処理の最適化の有効性を確かめる為に、計算時間がかなりかかり、比較的構造の単純な格子上の粒子の電荷を貯めるサブルーチン :dpost を使って調べた。RISC プロセッサーのパイプライン処理では IF 文がネックとなるので、次のように計算前に配列を用意する。

```
dimension nad(nx+2)
nad(j)=j-1
if (j.eq.1) nad(j) = nx
if (j.eq.nx+2) nad(j) = 1
```

そして次のように、IF 文の代わりに配列を使用する。

```
nn = nn + 1
if (nn.gt.nx) nn = nn - nx
    ⇓
nn = nad(nn+2)

nl = nn - 1
if (nl.lt.1) nl = nl + nx
      ⇓
nl = nad(nn)
```

さらに、

```
nn = part(1,j) + .5
dxl = part(1,j) - float(nn)
```

のように計算結果をすぐ次の行で使うと、パイプラインが結果待ちの状態となり止まってしまうと考えられるので計算結果は何行か後で使うように、行の順番を組換えた。
そして、次のように５つのケースについて、サブルーチン dpost を改良して領域 32×64 、粒子数 $(15 \times 15 + 25)$ ／ *cell* でタイムステップ３２５で５回づつ実行してサブルーチン dpost にかかった時間を比較した。

- Case1 :オリジナルのコードのもの
- Case2 :オリジナルの行の順番を、２段のパイプライン処理に配慮した順番にかえたもの
- Case3 :配列によって IF 文をなくしたもの
- Case4 :配列によって IF 文をなくし、行の順番を、２段のパイプライン処理に配慮した順番にかえたもの
- Case5 :配列によって IF 文をなくし、行の順番を、３段のパイプライン処理に配慮した順番にかえたもの

Case5 の３段のパイプライン処理が行なわれるようにするため、単に順番を組み換えるだけでは無理なので、１行ごく簡単な実際の計算とは関係のない行を加えた。その際、キャッシュ・ヒットに影響しないよう、この関数で頻繁に使われる変数を使用した。

5 チューンナップ実験の結果について

5.1 粒子配列の多次元化の結果

粒子配列を多次元化した結果、実行時間の改善は見られなかった。倍精度、単精度、粒子数や領域の大

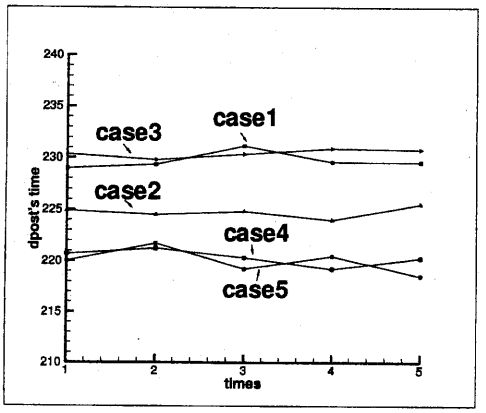

図 4:チューンナップの結果

小など、さまざな条件で比較したが、実行時間が減少しなかった。これは、このシステムのアーキテクチャによるものなのか、コンパイラによるものなのか、判然としない。

5.2 パイプライン処理の最適化の結果

Case1～Case5 までの結果は図 4 のようになった。
実験の結果から考察する。まず case1 と case3 の結果から単に IF 文をなくしただけではあまり効果がない事がわかる。これは、Pentium Pro の説明の所で書いたが、Pentium Pro が IF 文の見込み実行という処理をしているために、IF 文が他の RISC プロセッサのパイプライン処理のようにあまりネックとならないと考えられる。しかし、これは行の順番をパイプラインに配慮して組換えていないために、IF 文をなくしただけでは効果がないとも考えられる。次に case1 と case2 の結果から行の順番をパイプラインに配慮して組換えただけで IF 文をなくさなくても、ある程度効果があると考えられる。さらに case1 と case2、case4 の結果から IF 文をなくし、しかも行の順番をパイプラインに配慮して組換えると、明らかな効果がある。ただ 3～4 %と、他の計算機に対する効果が 5～10 %あるという事を考えると少し物足りない感じがする。これは、Pentium Pro プロセッサの順序外実行や、見込み実行が効果を発揮しているためと考えられる。
最後に case5 はあまり効果が得られなかった。ダミーの行の効果がなかったと、思われる。ループの中の 1 部が、本当にボトルネックとなっていたのか、ダミーを 1 行加えることにより本当にボトルネックが解消されるのかなど、コンパイラやプロセッサの問題となり、改善は難しいと考えられる。

5.3 チューンナップ実験のまとめ

この実験によって Pentium Pro プロセッサにおいての、パイプライン処理の最適化による効果の検証は、ある程度の成果を上げられたと思う。しかし、キャッシュメモリ使用の効率化のための実験は、思うような成果を、あげる事ができなっかった。

6 あとがき

6.1 まとめ

この実験により、Pentium Pro を搭載したパーソナル コンピュータをクラスタにして使う事によって、並列計算機に劣らない、シュミレーションが行なえることが、実証されたと思われる。

6.2 今後の課題

今後の課題としては、さらに大規模な実験を行なうには、コードの最適化が重要となってくるため、さらにプロセッサやコンパイラの性質について考慮した、他の RISC プロセッサへの最適化を参考にするだけではなく、Pentium Pro の特殊性に即した最適化の方法を考える必要があると思われる。

References

[1] Skelton PIC codes for parallel computers,Viktor K.Decyk,Computer Physics Communications 87,1995

[2] Optimization of Particle-in-cell Codes on RISC Processors ,Viktor K.Decyk ,Steve Roy Karmesin,Aeint de Boer,Paulett C.Liewer,1995

[3] Hight Performance Software on Intel Pentium Pro Processors or Micro-Ops to TeraFLOPS,Bruce Greer,Greg Henry,1997

[4] PVM-A User's Guide and Tutorial for Networked Parallel Computing,The MIT Press

[5] HP Vectra XU パフォーマンスガイド