ハイパフォーマンス
コンピューティング 79−10
（1999．12．3）

HPFによる実用コードの並列化とその性能評価

坂上仁志，伊東英之，小川雄三，高橋豊

姫路工業大学工学部情報工学科

データ並列言語HPFが実用コードの並列化に対してどの程度有効であるのかを評価するため，陽的解法を用いた3次元流体コード，PIC法を用いた2次元静電粒子コードおよびCIP法を用いた2次元流体コードを取り上げ，NEC Cenju-3上のHPFコンパイラで実際に並列化を行って性能を測定した．その結果，3次元流体コードでは非常に少ないHPF指示行の挿入だけで高い並列性能が得られ，2次元静電粒子コードおよび2次元CIPコードでは若干のソースコード修正が必要であるものの比較的良好な並列性能が得られた．

Parallelization of Scientific Applications Using High Performance Fortran

Hitoshi SAKAGAMI, Hideyuki ITOH, Yuzo OGAWA, and Yutaka TAKAHASHI

Department of Computer Engineering, Himeji Institute of Technology

We are focusing on checking describability and simplicity of HPF to parallelize scientific applications and its performances. In this paper, we have selected three programs, 3D fluid code, 2D particle code and 2D CIP code, and parallel performances of them were evaluated using HPF that was implemented at the NEC distributed memory parallel computer, Cenju-3. We have found that inserting a few HPF directives could achieve excellent performance for 3D fluid code. We have also found that relatively good performance could be obtained for 2D particle code and 2D CIP code with some source code modifications.

1．はじめに

従来のFortran77/90で書かれたプログラムに最小限の付加的な指示行を追加するだけで並列実行を可能とするデータ並列言語HPF (High Performance Fortran)[1-3]が提案されている．基本的にHPFでは，データを各プロセッサ上にどのように配置するかのみをユーザが明示的に指示し，データ転送や実行の同期等のそれ以外のことはすべて処理系に任せる．このため，ユーザは煩わしいプロセッサ間の通信管理や実行制御を記述するプログラミングから解放され，比較的容易に並列計算機の高性能が享受できる．

本論文では，実用コードとして3次元流体コード，2次元静電粒子コードおよび2次元CIPコードを考え，NEC並列処理センターのCenju-3上でのHPF化と実際に得られた性能向上について述べる．

2．3次元流体コードの並列化

実用アプリケーションとして，3次元非粘性圧縮性流体方程式を固定グリッドのオイラー法を用いて離散化し，理想気体の状態方程式と共に解く3次元流体コードを考える[4]．このコードでは，カーテシアン座標系（直交座標系）を用いた立方格子状の計算グリッドを導入し，離散化された方程式の時間発展は陽的解法で解いているため，計算には近距離の絡合いしかなく，シミュレーション空間を部分領域に分割して，各々の部分領域を別々のプロセッサに割り振って並列計算を行うことが原則として容易である．多次元の時間発展は分ステップ法を用いて1次元問題の重ね合わせで解いており，空間の微分には5点差分スキームを使用している．

シミュレーション空間を分割する方法として，Z方向にのみ分割する場合，YZ方向に分割する場合およびXYZすべての方向に分割する場合を考える．部分領域境界上のデータは，分割方向に計算する時に交換する必要があるので，分割方向を増やすと通信回数は増加する．しかし，分割方向を増やすとデータ交換平面が分割されるので，1回当たりの通信すべきデータ量は減少する．このため，通信量と通信回数についてはトレードオフが存在する[5]．

それぞれの場合でのHPF指示文を以下に示す．

```
!HPF$ PROCESSORS cj1(4)
      real array(lx,ly,lz)
!HPF$ DISTRIBUTE array(*,*,BLOCK) ONTO cj1

!HPF$ PROCESSORS cj2(2,2)
      real array(lx,ly,lz)
!HPF$ DISTRIBUTE array(*,BLOCK,BLOCK) ONTO cj2

!HPF$ PROCESSORS cj3(2,2,2)
      real array(lx,ly,lz)
!HPF$ DISTRIBUTE array(BLOCK,BLOCK,BLOCK) ONTO cj3
```

なお，すべての場合において，コメント行を除く全体のソース1050行に対して44行のHPF指示文を挿入するだけで並列化ができた．実際には，Fortranのinclude文を使って，配列を各サブルーチン内に展開するので，エディタで挿入する行数はずっと少ない．

NEC並列処理センターのCenju-3上で実装されているHPFを用いて3次元流体コードの並列性能を評価した．Lx=Ly=Lz=64としたときの並列実行によるスピードアップ率をZ方向分割（○），YZ方向分割（□），XYZ方向分割（▲）について図1に示す．

図1より，Z方向のみに分割した場合には32台で27倍の，YZ方向およびXYZ方向に分割した場合には64台で50倍という高い並列性能が得られている．また，スケーラビリティも良好であり，更にプロセッサ台数が多い場合でも高性能が期待できる．

Cenju-3のHPFコンパイラの既定メモリ配置では，各プロセッサは分散されて自分に割り当てられた配列の一部だけではなく，元々の配列すべてをメモリ上に確保する．

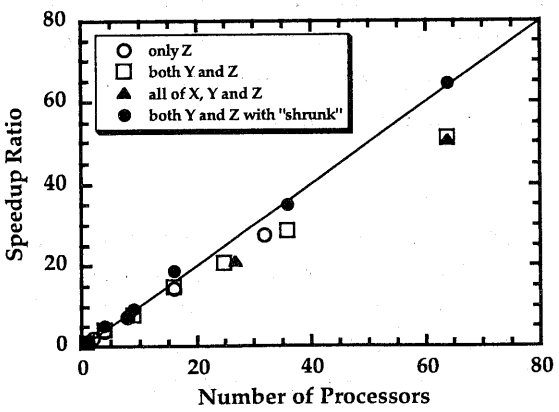

図1：3次元流体コードのスピードアップ率．

一方shrunkオプションを指定すると，自分に割り当てられた配列の一部だけをメモリ上に確保する．このため，メモリを有効に利用でき，特に大規模な問題を高並列で実行するときに有用であるが，配列参照時にはグローバルからローカルへのインデックス変換が必要となり計算負荷が増える．多次元配列を連続的に参照するループが並列化されると，各プロセッサはインデックスの一部のみを受け持つため，配列要素に対するアクセスは，既定配置の場合アドレス的に連続とはならないことがある．しかし，shrunkの場合には連続アクセスとなり，割り当てメモリ量が少なくなっていることも含めてキャッシュヒット率の向上が期待できる．効率のよいキャッシュ利用が期待できるYZ方向分割の場合に，shrunkを指定して性能を評価した．このときの並列実行によるスピードアップ率を図1に●で示す．この場合には，64台で64倍という極めて高い並列性能が得られている．

このコードの場合，複数のDOループにおいて同じ配列要素にアクセスしている．HPF2.0のSHADOW指示文を使えば，必要なデータ転送をDOループ外で行い，効率がよくなる可能性があるが，複数のDOループについて同じ通信を行うかもしれず，まだ改善の余地が残る．JAHPF[6]で議論されたHPF/JA1.0仕様[7]であるREFLECT指示文を使えば，1回の通信だけが必要なことをユーザが明示できて通信の最適化が促進され，更なる高並列性能が期待できる．

3．2次元静電粒子コードの並列化

次の実用アプリケーションとして，古典的Particle-In-Cell法を用いた2次元静電粒子コードを考える[8]．このコードでは，2次元FFTを用いてポアソン方程式を解いて静電場を求め，その静電場内における電子と陽子の運動をすべての粒子について計算している．電荷密度を計算する部分に不規則な配列参照／代入があり，一般にベクトル型スーパーコンピュータでも高い性能は得にくい．

並列化の手法として領域分割法を用いると，各プロセッサは自身が受け持つ領域内に存在する粒子に関する計算のみを行えばよいことになる．この場合，粒子がある領域から別の領域に移動すると，その粒子の計算を受け持つプロセッサが替わるので，粒子情報をプロセッサ間でやり取りしなければならない．しかし，このような通信パターンをHPFで記述するためには，かなりトリッキーなコーディングとなり，付加的な指示行を追加するだけでプログラムの並列化を行うHPF本来の趣旨から逸脱する．そこで，領域分割法ではなく，粒子情報を納めている配列を単純にBLOCK分散し，各プロセッサが受け持つ粒子群を固定して並列化を行う．このため，粒子の運動を計算するDOループは，無理なく効率よく並列化される．

粒子は2次元のシミュレーション空間を自由に移動するため，各プロセッサは場の情報を納めている配列に関しては，全空間について重複して持つ必要がある．各プロセッサは，自身が受け持つ粒子の位置情報から全空間に分布するローカルな電荷密度を計算し，そのローカル電荷密度に対してREDUCTION演算を行うことにより，トータルな電荷密度を求める．電荷密度が求まると，それを2次元FFTして電荷密度のフーリエ係数を求めるが，2次元FFTは1次元FFTの重ね合わせで行っている．そこで，FFTそのものを並列実行するため，1次元FFTの方向にあわせてFFTされる配列をREDISTRIBUTEする．電荷密度のフーリエ係数が求まると，それにフォームファクタを乗じて電位のフーリエ係数を計算し，それを2次元逆フーリエ変換することで電位を求める．このときも，1次元逆フーリエ変換の方向にあわせて配列を同様にREDISTRIBUTEする．各プロセッサは，電位について全空間の情報が必要なため，求まった電位をREPLICATEする．

電荷密度を計算する典型的なDOループは，十分なベクトル長を確保できる一時配列を用意して，その一時配列にローカルな電荷密度をベクトル演算で求め，最後に総和計算を行ってトータルな電

荷密度を求める．現在Cenju-3上に実装されているHPFコンパイラは，上記のベクトル化可能な電荷密度の計算ループを効率よく並列化することができない．そこで，ソースコードを修正して粒子情報の配列を1次元から2次元に拡張し，配列rhotmpのインデックスを入れ替えて各プロセッサが連続アクセスできるようにした．このときの典型的な修正前，修正後のソースコードを簡単化のため1次元にして図2に示す．この結果，各プロセッサは自身の受け持つ粒子の情報からローカルな電荷密度を効率よく並列計算できるようになった．ただし，粒子情報の配列を1次元から2次元に拡張したため，粒子情報を参照する他のDOループにおいても，同様なソースコードの修正が必要となった．

```
      real xe(no), rho(lx), rhotmp(lvec,lx)
      do iv = 1, lvec
         do i = 1, no, lvec
            ixe = xe(i+iv-1)
            dxe = xe(i+iv-1) - ixe
            rhotmp(iv,ixe) = rhotmp(iv,ixe) +
     $                                 (1.0-dxe)
         end do
      end do
c.... reduction rhotmp to rho
      do iv = 1, lvec
         do i = 1, lx
            rho(i) = rho(i) + rhotmp(iv,i)
         end do
      end do
```

⇒

```
      real xe(no/lpara,lpara), rho(lx),
     $     rhotmp(lx,lpara)
!HPF$ DISTRIBUTE xe(*,BLOCK), rhotmp(*,BLOCK)
      do ip = 1, lpara
         do i = 1, no/lpara
            ixe = xe(i,ip)
            dxe = xe(i,ip) - ixe
            rhotmp(ixe,ip) = rhotmp(ixe,ip) +
     $                                 (1.0-dxe)
         end do
      end do
c.... reduction rhotmp to rho
      do ip = 1, lpara
         do i = 1, lx
            rho(i) = rho(i) + rhotmp(i,ip)
         end do
      end do
```

図2：2次元静電粒子コードにおけるソースコードの修正．

このコードでは，2次元FFTは1次元FFTを繰り返し呼ぶことで実現している．1次元FFTを呼び出すときの実引数は2次元配列であるが，仮引数は1次元配列である．HPFでは，このようなコーディングは問題を発生させる．そこで，2次元配列のデータを1次元の一時配列にコピーするようソースコードを修正した．Fortran77で書かれたプログラムでは，実引数と仮引数の配列次元が一致していないことが多いため，そのままではHPF化が困難になる可能性があり，このようなソースコードの修正を支援するシステムの開発が望まれる．

以上により並列化を行ったが，コメント行を除く全体のソース1500行に対して，54行のHPF指示文を挿入し，ソースコードを58行追加／修正する必要があった．

同様にNEC並列処理センターのCenju-3上で実装されているHPFを用いて2次元静電粒子コードの並列性能を評価した．Lx=Ly=64としたときの並列実行によるスピードアップ率を図3に示す．ただし，メッシュ当たりの粒子数を50（○），100（□），200（▲）としている．粒子に関する計算は，通信なしに並列実行できるため，粒子数が多いほど粒子計算の粒度が大きくなり，トータルな並列性能はよくなる．プロセッサ台数が12台まででは，粒子数が大きいほど良好な，比較的よい並列性能が得られているが，16台を越えると粒子数に関係なく急激に性能向上が低下し，更に台数が増えるとかえって性能が悪化している．

この原因を詳細に調べると，各プロセッサが計算したローカルな電荷密度から

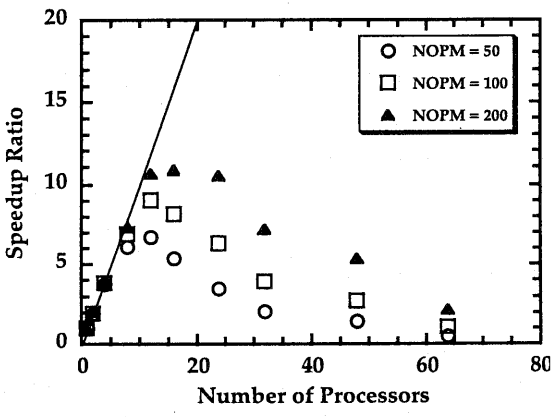

図3：2次元静電粒子コードのスピードアップ率．

REDUCTION演算によりトータルな電荷密度を求めるDOループにおける並列実行性能の著しい低下が支配的であることが分かった．プロセッサ台数が増えるほど，このループの実行時間が全体の性能に大きく影響し，急激に性能が低下したと考えられる．これは，REDUCTION演算の対象が配列要素であるため，解析能力の不足からCenju-3上に実装されている現在のHPFコンパイラは，効率のよいREDUCTIONコードを生成していないためと思われる．

4．2次元CIPコードの並列化

最後の実用アプリケーションとして，CIP法を用いた2次元流体子コードを考える[9]．このコードでは流体方程式をCIP法により離散化しているが，CIP法で必要な1次導関数を計算するため，異なった変数を実引数として同一のサブルーチンを何度も呼び出している．

領域分割の方法としてはX方向，Y方向およびXY方向が考えられるが，ここでは，メモリへ連続アクセスできるY方向分割，即ち2次元配列の2次元目のみをBLOCK分散する方法を用いる．Y方向計算時には，部分領域境界上のデータをプロセッサ間で通信して交換する必要がある．

このコードでは，流速の向きによってアクセスする配列のインデックスが異なっている．即ち，DOインデックスをiとすると，流速の向きによって配列に対してi±1のインデックスで参照している．実際のコーディングでは，流速の向きによって補助変数に±1を代入し，配列の参照にはDOインデックスiとこの補助変数の和を用いている．HPFコンパイラは，参照がi±1の範囲に限定されることを解析できず，DOインデックスiと補助変数の和が任意のインデックスになると判断するため，配列全体をREPLICATEする通信が発生する．このため，並列実行されるものの通信オーバーヘッドが大きく，並列性能は悪い．そこで，配列参照がi±1の範囲に限定されることを明確にするために，流速の向きを判断するブロックif文内に処理を重複して記述し，補助変数を使わないようにした．このときの典型的な修正前，修正後のソースコードを簡単化のため1次元にして図4に示す．この結果，REPLICATEという無駄な通信なしに並列実行できるようになった．なお，並列化のためには，コメント行を除く全体のソース507行に対して，19行のHPF指示文を挿入し，ソースコードを25行追加／修正する必要があった．

```
do i = ...
   if( ... ) then
      isgn = 1
   else
      isgn = -1
   end if
   ... = ... fs(i+isgn) ...
end do
```

```
do i = ...
   if( ... ) then
      isgn = 1
      ... = ... fs(i+1)
   else
      isgn = -1
      ... = ... fs(i-1)
   end if
end do
```

図4：2次元CIPコードにおけるソースコードの修正．

Lx=Ly=384としたときの並列実行によるスピードアップ率を図5に示す．ただし，●は全体での値，□，▲，○はそれぞれサブルーチンcip, fixed, periodicのみでの値である．全体については，16台では13倍と良好な並列性能が得られているが，64台では36倍とやや効率が低下する．サブルーチンcipは，計算のほとんどの部分を占め，効率よく並列実行されている．サブルーチンfixedは，Y方向の境界条件を計算する部分であり，全体の計算領域の両端を受け持つプロセッサしか実行しないため並列効率は本質的に悪い．サブルーチンperiodicは，X方向の境界条件を計算する部分であり，並列実行できるものの計算の粒度が極めて小さいため，低い並列効率に留まっている．このため，台数が増えるとサブルーチンcipの計算比重が相対的に小さくなり，並列性能が悪化していると考えられる．しかし，計算規模が大きくなれば台数が多くても良好な並列性能を期待できる．

このコードの場合，HPF/JA1.0仕様であるSHADOW, LOCAL指示文を使えば，補助変数を使った配列参照について通信の必要がないことを明示でき，ソースコードを修正することなく効率のよい並

列化ができる．

5．むすび

並列計算機を一部の人のみが利用できる特殊な機械から一般ユーザでも活用できる高性能なツールとするためには，HPFのような高レベルで並列性を記述できる言語が不可欠である．

そこで，HPFによる実用コードの並列性能を評価するため，陽的解法を用いた3次元流体コード，PIC法を用いた2次元静電粒子コードおよびCIP法を用いた2次元流体コードを取り上げた．Cenju-3上のHPFを使って評価を行った結果，3次元流体コードでは全ソース行数の数%程度のHPF指示文を挿入するだけで高い並列性能が得られることが分かった．2次元静電粒子コードでは，若干のソースコードの修正が必要であり，プロセッサ台数が12台程度までなら比較的良好な結果が得られた．また，2次元CIPコードでも，若干のソースコードの修正が必要であったものの，良好な結果が得られた．

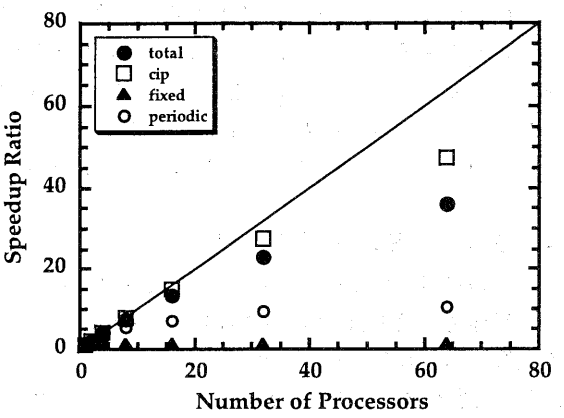

図5：2次元CIPコードのスピードアップ率．

更に，すべての場合についてHPF/JA1.0仕様が利用可能であれば，更に良好な並列性能を得られる可能性が高く，ソースコードを修正することなく並列化できることがわかった．

謝辞

本研究を実施するに当たり，並列計算機Cenju-3の利用環境を提供いただいたNEC並列処理センターに謝意を表する．また，HPF/JA仕様策定のため精力的に活動していただいたJAHPF参加会員ならびにJAHPFの種々の活動を積極的に援助していただいている（財）高度情報科学技術研究機構に対して謝辞を述べる．

参考文献

[1] High Performance Fortran Forum:High Performance Fortran language specification, version 1.0, Technical Report CRPC-TR92225, Rice University (1992).

[2] Koelbel, C. et al.: The High Performance Fortran Handbook, The MIT Press, Cambridge, MA (1992).

[3] High Performance Fortran Forum:High Performance Fortran language specification, version 2.0 (1996).

[4] H. Sakagami and K. Nishihara: Three Dimensional Rayleigh-Taylor Instability of Spherical Systems, Phys. Rev. Lett., Vol.65, pp.432-435 (1990).

[5] 坂上仁志，高橋豊: 3 次元流体コードの領域分割法における並列効率, 電情通学論, Vol.J80-D-I, pp.683-690 (1997).

[6] http://www.tokyo.rist.or.jp/jahpf/.

[7] HPF2.0公式マニュアル, スプリンガーフェラーグ東京 (1999).

[8] C.K.Birdsall and A.B.Langdon:Plasma Physics via Computer Simulation, McGraw-Hill, NY (1982).

[9] T.Yabe: Unified Solver CIP for Solid, Liquid and Gas, Comp. Fluid Dynamics Review ,Wiley (1997).