J. Plasma Fusion Res. Vol.83, No.3 (2007) 300-305

連載コラム High Performance Fortran で並列計算を始めよう

8．もっと活用しよう HPF(2)

林　康晴
NEC　第一コンピュータソフトウェア事業部

（原稿受付：2007年2月13日）

いよいよシリーズも最終回となりました．今回は，前回の続きとして，HPF プログラムの高速化やベクトル計算機上での並列化について解説します．

8.1　HPF プログラムの高速化(2)

HPF のような並列プログラムの効率を上げるには，並列化可能な部分を増やすとともに，オーバヘッド（分散並列の場合は主に通信）を減らし，負荷バランスの均等化を図ることが重要です．本節では複数のループや手続にまたがるプログラムの並列化や高速化について解説します．

8.1.1　1つの手続を異なる分散の実引数で呼び出す場合

既存の Fortran プログラムを HPF 化する際，1つの手続に対する実引数の分散が，Fig. 1(a)のように呼出しごとに異なってしまう場合があります．このような場合，実引数と仮引数の分散が異なる呼出しの入口と出口で自動再マッピングが発生してしまいます．そこで Fig. 1(b)のように，実引数のマッピングの種類分だけサブルーチンのコピーを作り，それぞれの仮引数の分散を実引数と一致させると，手続呼出し時の自動再マッピングによる通信の発生を避けることができます．

同様の手法は，Fig. 2(a)のように，同一の一時配列が複数箇所で利用されている場合も適用できます．この場合も，Fig. 2(b)のように一時配列のコピーを作成し，各ループに合わせて適切に分散された一時配列を利用すると，通信を抑制できます．

8.1.2　転置パターンの通信が繰返し発生する場合

連載の第6回で解説した2次元静電粒子コードのように，2次元目を block 分散した2次元配列を実引数として，以下のような順序で FFT，逆 FFT サブルーチンを呼び出す処理を考えてみます（Fig. 3(a)参照）．

1．X 方向の FFT（仮引数は2次元目を block 分散）
2．Y 方向の FFT（仮引数は1次元目を block 分散）
3．X 方向の逆 FFT（仮引数は2次元目を block 分散）
4．Y 方向の逆 FFT（仮引数は1次元目を block 分散）

この場合，Y 方向の FFT，逆 FFT サブルーチンの呼出し時と戻り時に，分散次元を変更し，元に戻すための自動再マッピングがそれぞれ発生するため，合計4回の通信が必要です．そこで，Fig. 3(b)のように，X 方向と Y 方向の逆 FFT の順序を入れ替えて1次元目の block 分散で行う処理を1ヶ所にまとめます．さらに，Y 方向の FFT/逆 FFT の前後で，1次元目で block 分散された一時配列 tmp との間でコピーを行って，Y 方向の処理は tmp に対して行うようにすると，手続呼出し時の自動再マッピングが発生しないため，通信回数をコピー時の2回だけに削減できます．

```
      dimension nodist(100,100)       ! 分散指定なし
      dimension dist(100,100)         ! 分散指定あり
!HPF$ DISTRIBUTE (*,BLOCK) :: dist
      interface
        subroutine sub(array)
        dimension array(100,100)      ! 分散指定なし
        end subroutine
      end interface
      call sub(nodist)   !自動再マッピングは起こらない
      call sub(dist)     !自動再マッピングが発生
      end
      subroutine sub(array)
      dimension array(100,100)
        << 略 >>
      end
```

(a)手続のクローニング前

```
      dimension nodist(100,100)       ! 分散指定なし
      dimension dist(100,100)         ! 分散指定あり
!HPF$ DISTRIBUTE (*,BLOCK) :: dist
        << 略 >>
      call sub(nodist)   !自動再マッピングは起こらない
      call subdist(dist) !自動再マッピングは起こらない
      end
      subroutine sub(array)
      dimension array(100,100)
          << 略 >>
      end subroutine
      subroutine subdist(array)
      dimension array(100,100)
!HPF$ DISTRIBUTE (*,BLOCK) :: array
          << 略 >>
      end subroutine
```

(b)手続のクローニング後

Fig. 1　手続のクローニングによる自動再マッピング抑制．

Let Us Start Parallel Processing Using High Performance Fortran ! 8. Let Us Practice HPF (2)
HAYASHI Yasuharu
author's e-mail: hayashi@hpc.bs1.fc.nec.co.jp

```
      real a(n,n),b(n,n),tmp(n)
!HPF$ DISTRIBUTE (*,BLOCK) :: a,b
      do j=1,n
        tmp(j) = a(1,j)
        b(1,j) = tmp(j)
      enddo
!HPF$ ON HOME(a(:,1)), LOCAL
      do i=1,n
        tmp(i) = a(i,1)
        b(i,1) = tmp(i)
      enddo
```

(a) 一時配列のクローニング前

```
      real tmpdist(n),tmp(n)
!HPF$ DISTRIBUTE tmpdist(BLOCK)
      do j=1,n
        tmpdist(j) = a(1,j)
        b(1,j) = tmpdist(j)
      enddo
!HPF$ ON HOME(a(:,1)), LOCAL
      do i=1,n
        tmp(i) = a(i,1)
        b(i,1) = tmp(i)
      enddo
```

(b)一次配列のクローニング後

Fig. 2 一時配列のクローニングによる通信抑制.

```
      real rho(lx,ly)
!HPF$ DISTRIBUTE rho(*,BLOCK)
      call fftx(rho)     !X 方向 FFT
      call ffty(rho)     !Y 方向 FFT:入口・出口で通信
        << 略 >>
      call inv_fftx(rho)!X 方向逆 FFT
      call inv_ffty(rho)!Y 方向逆 FFT:入口・出口で通信
```

(a)処理順序変更前

```
      real rho(lx,ly),tmp(lx,ly)
!HPF$ DISTRIBUTE rho(*,BLOCK)
!HPF$ DISTRIBUTE tmp(BLOCK,*)
      call fftx(rho)     !X 方向 FFT
      tmp = rho          !一時配列へのコピー:通信発生
      call ffty(tmp)     !Y 方向 FFT:通信なし
        << 略 >>
      call inv_ffty(tmp)!Y 方向逆 FFT:通信なし
      rho = tmp          !一時配列からのコピー:通信発生
      call inv_fftx(rho)!X 方向逆 FFT
```

(b)処理順序変更後

Fig. 3 処理順序の変更十一時配列の利用による通信削減.

8.1.3 乱数生成の並列化

擬似乱数の生成処理は，通常漸化式の計算を行うため並列化することができません．しかし，モンテカルロシミュレーションなどで大量の乱数が必要となる場合，乱数生成処理の逐次実行時間がボトルネックになりがちなので，並列化したいところです．本項では，Mersenne Twister（以下 MT）を利用した並列乱数生成法をご紹介します[1]．MT は，松本眞・西村拓士両氏によって開発された擬似乱数生成アルゴリズムです[2]．HPF プログラムから MT を使った乱数生成手続を並列に呼び出すためには，公開されているソースコードに若干の修正が必要です．その一例

```
      integer, parameter :: np = 8
      integer aaa(np),maskB(np),maskC(np) !パラメタ
      double precision rnd(10000,np)      ! 乱数
!HPF$ PROCESSORS p(np)
!HPF$ DISTRIBUTE (*,BLOCK) onto p :: rnd
!HPF$ DISTRIBUTE (BLOCK) onto p :: aaa,maskB,maskC
      interface
       EXTRINSIC('Fortran','LOCAL')
     &  subroutine init_genrand(s,a,maskB,maskC)
        integer,intent(in) :: s,a,maskB,maskC
       end subroutine
       EXTRINSIC('Fortran','LOCAL')
     &  doubleprecision function genrand_real2()
       end function
      end interface

      read(99)aaa,maskB,maskC
!HPF$ INDEPENDENT
      do i=1,np
        call init_genrand(3241*i,
     &        aaa(i),maskB(i),maskC(i))
      enddo
!HPF$ INDEPENDENT
      do i=1,np
        do j=1,10000
          rnd(j,i)=genrand_real2()
        enddo
      enddo
         << 略 >>
      end
```

Fig. 4 並列乱数生成の例.

を Fig. 4 に示します．修正方法の概要は以下のとおりです．

- まず，上記[2]のページで公開されている Dynamic Creator（以下 DC）[3]によって，あらかじめ並列数個の異なる id 番号に対するパラメタを求めて，ファイルに格納しておきます．id 番号に依存して値が変わるパラメタは 3 つ（aaa，maskB，maskC）だけなので，これら 3 つの値を格納し，残りは乱数生成手続中で定数とすればよいでしょう．
- 異なる id 番号に対するパラメタから生成される乱数列は相関が少ないと考えられているため，各プロセッサは異なるパラメタに基づいて，並列ループ中でそれぞれ独立な乱数を生成することができます．そのために，上記 3 つのパラメタをそれぞれプロセッサ数分宣言し，block 分散してプロセッサごとに 1 組配置されるようにします．
- DC 中の初期化関数 sgenrand_mt と乱数生成関数 genrand_mt は，Tsuyoshi Tada 氏による Fortran 版の MT である mt19937ar.f を例にとると，init_genrand，genrand_int32（倍精度実数版は genrand_real2）にそれぞれ対応します．また上記のパラメタ aaa，maskB，maskC は，mt19937ar.f では，それぞれ MATRIX_A，T1_MASK，T2_MASK に対応します．両者を比較して他のパラメタも DC で生成したものを使うように修正してみてください．
- その上で，Fig. 4のように，並列ループ中で初期化手続（init_genrand）を呼び出し，各パラメタの値を手続内部の共通ブロック変数に設定した後，乱数生成手続

```
      real x(n)
!HPF$ DISTRIBUTE (BLOCK) :: x
      integer k
      k=0
      do i=1,n
        if(...)then
          k=k+1
          x(k) = x(i)
        endif
      enddo
```

(a)次元拡張前

```
      real x(n/2,2)
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE (*,BLOCK) onto p :: x
      integer k(2)
!HPF$ DISTRIBUTE (BLOCK) onto p :: k
!HPF$ INDEPENDENT,NEW(i)
      do ip = 1,2
        k(ip) = 0
        do i=1,n
          if(...)then
            k(ip)=k(ip)+1
            x(k(ip),ip) = x(i,ip)
          endif
        enddo
      enddo
```

(b)次元拡張後

Fig. 5 粒子データのパッキング.

(genrand_real2)を呼び出して，乱数を生成します．以前の連載で説明したEXTRINSIC機能を利用すると，Fortranプログラムとして並列に呼び出すことができます．ローカルモデルのFortran手続中では，共通ブロック変数の値はプロセッサごと異なる値を持つことに注意してください．

- 一度の呼出しで複数個の乱数を生成するように修正すればベクトル化も容易ですので，挑戦してみてください．

8.1.4 奥の手：次元拡張による並列化

Fig. 5(a)のように，1次元粒子データをパッキングするような場合，1次元配列のまま分散すると，x(i)とx(k)が配置されるプロセッサが異なる度に不規則な一要素ごとの通信が必要となり効率が悪くなります．そこで，Fig. 6のように，並列数分の寸法を持つ次元を付加して，その次元で分散を行う方法があります[4]．この場合Fig. 5(b)のように，分散次元をアクセスするループを新たに最外側に追加する必要がありますが，ループ中の処理自体は各プロセッサの中だけでローカルに行うことができます．このような修正により，パッキングやソートのように複数プロセッサ間にまたがるとコストの高い通信が発生する処理の通信コストを大幅に減らすことができます．

8.2 ベクトル並列計算機上での分散並列化

最後に，ベクトル計算機上での並列化について述べます．ベクトル計算機で並列実行する場合，まずは逐次プログラムとしてベクトル化率と平均ベクトル長が高くなるようチューニングしておくことが重要です．

```
      real*8 x(n/2,2)
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE (*,BLOCK) ONTO p :: x
```

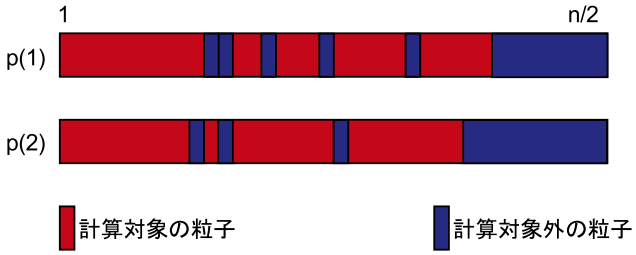

Fig. 6 次元拡張による並列化.

8.2.1 一般的な原則と注意点

ベクトル化は原則として最内側のループを対象として行われ，その際のメモリアクセスが連続の場合に最も効率が良くなります．そのため，配列の1次元目を連続にアクセスするループを最内側に記述しベクトル化するのが一般的です．一方並列化は，通信や処理の分担範囲の決定といったオーバヘッドを最小にするため，できるだけ外側のループで行うと効率が良くなります．また，配列はできるだけ最後の次元で分割した方が，通信時に他のプロセッサと交換する配列要素がメモリ上連続に配置されている場合が多くなるため，通信効率が良くなります．これらのことから，一般に配列は最終次元で分散し，多重ループの最外側で最終次元をアクセスして並列化を行い，最内側で1次元目をアクセスしてベクトル化を行うと最も効率が良くなります．

ベクトル化とHPFによる並列化を併用する場合，並列化対象ループのループ長や配列の分散次元の寸法が，元のFortranプログラムとは変わってしまうため，ベクトル化の効率に影響を与える可能性がある点に注意が必要です．

例えばFig. 7(a)のように，配列aの分散次元をアクセスする最外側ループで並列化を行う場合，HPFコンパイラによる並列化後には，Fig. 7(b)のように，並列ループの始値と終値が変数に置き換えられてしまいます．Fortranコンパイラは，プログラムのさまざまな記述を解析して，ベクトル化対象のループを選択しますので，並列化によるループ

```
      real a(129,256)
!HPF$ PROCESSORS p(4)
!HPF$ DISTRIBUTE a(BLOCK,*) onto p
!HPF$ INDEPENDENT
      do i=1,128      ! 並列化対象のループ
        do j=1,256
          a(i,j) = ...
```

(a)HPFソースプログラム

```
      do i=lbnd,ubnd ! 並列化後のループ
        do j=1,256
          a(i,j) =...
```

(b)並列化後のループの始値と終値

Fig. 7 並列化によるループの変形.

```
      real a(1023,1023)
!HPF$ PROCESSORS p(2)
!HPF$ DISTRIBUTE a(BLOCK,*) onto p
!HPF$ SHADOW a(0:1,0)
      do j=1,1023
        a(1,j)=0.0
      enddo
```

Fig. 8 シャドウ領域によるバンクコンフリクトの回避.

長の変化と，配列のアクセスパターンの組合せによっては，元のFortranプログラムと並列化後のプログラムとで，ベクトル化対象のループが変化する可能性があります．例えば，もともとはjのループがベクトル化対象であったのに，並列化後にはiのループがベクトル化対象となってしまい，ベクトル長が短すぎて効率が低下する，といったことが起こりえるわけです．どのループがベクトル化の対象となるかを事前に予測することは困難ですが，プログラムの実行時情報を参照し，平均ベクトル長が想定されるより短い場合，並列化されループ長が短くなったループがベクトル化されていないかチェックしてみて下さい．不適切なループがベクトル化されている場合，ベクトル化を抑止する指示行などを指定して，ベクトル化対象ループを変更するとよいでしょう．

ベクトル計算機においてバンクコンフリクトを避けるための手法として，配列の寸法を奇数にすることがよく行われます．例えば，Fig. 8のような，配列aの2次元目をアクセスするループでのベクトル化を考えてみます．プログラムの他の部分の都合上，配列aを1次元目で分散した場合，このまま2プロセッサ上にblock分散すると，分割サイズが512要素となってしまい，バンクコンフリクトが多発する可能性があります．このような場合，袖領域（シャドウ領域）を付加することによって分散次元の割付けサイズを調節できます．例えば，Fig. 8のように1次元目の上側に幅1のシャドウ領域を付加すると，1次元目の割付けサイズが513となり，バンクコンフリクトを避けることができます．

8.2.2 電荷密度計算部分のベクトル化と分割サイズ

ここでは第6回の連載で解説したプログラムを例題として，HPFにおけるベクトル化を考えます．3次元流体コードの方は，そのまま翻訳すればベクトル計算機上でも良好な結果が得られるので，今回は主に2次元静電粒子コード[5]のポイントを解説します．このプログラムで最もコストの高い電荷密度を計算するループ（Fig. 9）は，電荷密度を格納する配列rhotmpに対する間接参照ループとなっているため，そのままではベクトル化することができません．

このような場合よく行われるのは，Fig. 10のように，元の1重ループを，ベクトルレジスタ長ごとに区切って2重ループとする方法です．内側ループでベクトル化できるよう，配列rhotmpは内側ループで連続アクセスするための次元を付加します．またrhotmpが出現する前後のループに対しても次元拡張に伴う修正を行います．この際注意が必要なのは，普通のblock分散では，一般には分割サイズがベクトルレジスタ長の倍数にはなりませんから，内側ループの途中で粒子データが配置されているプロセッサ間の境目が来てしまい，外側のiiループで並列化する際通信が必要になる，ということです．この通信を抑制するために，Fig. 10の5行目のように粒子データの分割サイズがベクトルレジスタ長の倍数になるよう明示します．また並列化す

```
      dimension xe(no),ye(no),rhotmp(lx,ly),
     &          rho(lx,ly)
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (BLOCK) ONTO proc :: xe,ye
!HPF$ DISTRIBUTE (*,BLOCK) ONTO proc :: rho
          << rhotmpの初期化 >>
!HPF$ INDEPENDENT,NEW(ixe,dxe,ddxe,iye,dye,ddye)
!HPF$& ,REDUCTION(+:rhotmp)
      do i = 1, no
         ixe  = xe(i)
         dxe  = xe(i) - ixe
         ddxe = 1.0-dxe
         iye  = ye(i)
         dye  = ye(i) - iye
         ddye = 1.0 - dye
         rhotmp(ixe  ,iye  ) =
     &          rhotmp(ixe  ,iye  ) - ddxe * ddye
         rhotmp(ixe+1,iye  ) =
     &          rhotmp(ixe+1,iye  ) -  dxe * ddye
         rhotmp(ixe  ,iye+1) =
     &          rhotmp(ixe  ,iye+1) - ddxe *  dye
         rhotmp(ixe+1,iye+1) =
     &          rhotmp(ixe+1,iye+1) -  dxe *  dye
      enddo
          << rhotmpからrhoへのコピー >>
```

Fig. 9 電荷密度の並列計算.

```
      parameter(lv=256)          ! ベクトルレジスタ長
      dimension xe(no),ye(no),rhotmp(lv+1,lx,ly),
     &          rho(lx,ly)
!HPF$ PROCESSORS proc(lpara)
!HPF$ DISTRIBUTE (BLOCK(((no-1)/(lpara*lv)+1)*lv))
!HPF$& ONTO proc :: xe,ye
!HPF$ DISTRIBUTE (*,BLOCK) ONTO proc :: rho
        << rho, rhotmpの初期化等 >>
!HPF$ INDEPENDENT,NEW(i,ixe,dxe,ddxe,iye,dye,ddye)
!HPF$& ,REDUCTION(+:rhotmp)
      do ii = 1, no, lv                ! 並列化
!HPF$    ON HOME(x(ii)), LOCAL(xe,ye) BEGIN
         do ivec = 1, min(lv,no-ii+1) ! ベクトル化
           i = ii + ivec - 1
           ixe  = xe(i)
           dxe  = xe(i) - ixe
           ddxe = 1.0-dxe
           iye  = ye(i)
           dye  = ye(i) - iye
           ddye = 1.0 - dye
           rhotmp(ivec,ixe,iye) =
     &        rhotmp(ivec,ixe,iye) - ddxe * ddye
           rhotmp(ivec,ixe+1,iye) =
     &        rhotmp(ivec,ixe+1,iye) - dxe * ddye
           rhotmp(ivec,ixe,iye+1) =
     &        rhotmp(ivec,ixe,iye+1) - ddxe * dye
           rhotmp(ivec,ixe+1,iye+1) =
     &        rhotmp(ivec,ixe+1,iye+1) - dxe * dye
         enddo
!HPF$    ENDON
      enddo
          << rhotmpの値をrhoに足し込む >>
```

Fig. 10 次元拡張とループの2重化によるベクトル化.

```
!HPF$ INDEPENDENT,NEW(ixe,dxe,ddxe,iye,dye,ddye)
!HPF$& ,REDUCTION(+:rhotmp)
      do iy=0,1
!HPF$ INDEPENDENT
      do ix=0,1
!HPF$ INDEPENDENT
!CDIR LISTVEC
      do i = 1, no
         ixe  = xe(i)
         dxe  = xe(i) - ixe
         ddxe = (1-ix)*1.0 - (-2*ix+1)*dxe
         iye  = ye(i)
         dye  = ye(i) - iye
         ddye = (1-iy)*1.0 - (-2*iy+1)*dye
         ixe = ixe + ix
         iye = iye + iy
         rhotmp(ixe  ,iye ) =
     &          rhotmp(ixe  ,iye ) - ddxe * ddye
      enddo
      enddo
      enddo
```

Fig. 11 LISTVEC 指示行によるベクトル化.

るループのDO変数iiと粒子データの添字iが対応しないため，コンパイラが自動的に通信の必要がないことを判定するのは難しいので，ON-HOME-LOCAL 指示文を指定して，粒子データに対する通信なしで実行可能であることを明示します(Fig. 10)．このような修正を行うことで，通信なしで外側ループを並列化し，内側ループをベクトル化できます．

8. 2. 3 間接参照のままでのベクトル化と REDUCTION 節

地球シミュレータ[6]等の Fortran コンパイラでサポートされている LISTVEC 指示行を，Fig. 11 のように指定すると，ベクトル化の際，依存がある箇所だけスカラ演算を行って補正することにより，間接参照の集計演算ループのままでもベクトル化可能となります[7]．ただし，LISTVEC 指示行を指定したループ中で間接参照される配列は，1種類につき1つの文にしか出現できない，という規則があるため，4回の rhotmp の出現をループ長2の2重ループに変換します（それぞれ別名の配列に置き換える方法もありますが，分散並列の場合，集計演算のための通信が配列の数だけ必要になり通信オーバヘッドが高くなります)．この際，中で参照される変数の値は，新たに加えた2重ループの DO 変数を使ってうまく調整します．

Fig. 11 の3重ループはいずれも配列 rhotmp に対する足込みのループとなっていますが，このように多重ループで集計演算を行う場合，REDUCTION 節は一番外側だけに指定することに注意してください．このような書換えにより，間接参照ループのままでもベクトル化が行えます．

8. 2. 4 2つのベクトル化方法の比較

ここで紹介した⑴次元拡張による方法，⑵LISTVEC 指示行による方法，を比較すると，前者は線形アクセスによる効率の良いベクトル化が可能である，という利点があります．一方，後者は次元拡張による余分なメモリを必要とせず，その分演算量も少ない，という利点があります．メモリ量に余裕のない場合は後者の方がよいですし，余裕のある場合はプログラムによって，性能が良い方を採用するとよいでしょう．

参 考 文 献

[1] Shinsuke Satake, Masao Okamoto, Noriyoshi Nakajima and Hisanori Takamaru : Development of Three-Dimensional Neoclassical Transport Simulation Code with High Performance Fortran on a Vector-Parallel Computer, NIFS Report NIFS-828 (2005).

[2] Mersenne Twister Home Page (http://www.math.sci.hiroshima-u.ac.jp/~m-mat/MT/mt.html).

[3] Makoto Matsumoto and Takuji Nishimura : Dynamic Creation of Pseudorandom Number Generators, Monte Carlo and Quasi-Monte Carlo Methods 1998, Springer, 2000, pp 56-69.

[4] Hiroaki Ohtani, Seiji Ishiguro, Ritoku Horiuchi, Yasuharu Hayashi, and Nobutoshi Horiuchi : Development of Electromagnetic Particle Simulation Code in an Open System, Lecture Notes in Computer Science in press.

[5] 坂上仁志，水野貴夫：国産 HPF コンパイラの性能評価と互換性検証，情報処理学会研究報告，2001-HPC-87, 73-78 (2001).

[6] 地球シミュレータセンター（http://www.es.jamstec.go.jp/esc/jp/）.

[7] 杉山徹，寺田直樹，村田健史，大村善治，臼井英之，松本紘：LISTVEC指示行を使った多粒子シミュレーションの大規模化，情報処理学会論文誌：コンピューティングシステム，Vol.45, NO.SIG6 (ACS6), pp.171-175, May. 2004.

●連載の終わりに代えて（HPF 推進協議会）

このシリーズを終えるにあたって，HPF 推進協議会として，HPF をここまで育て上げるきっかけを与えてくださった，故三好甫氏に感謝の意を捧げたいと思います．

三好氏は，「地球シミュレータ」の生みの親として多くの人に知られていますが，高度情報科学技術研究機構副理事長であった1990年代半ばに，シミュレーション研究など大規模計算を，地球シミュレータのような将来の高並列計算機で実行するには，もっとユーザ（人間）に近い並列処理言語，HPF が重要であると指摘されました．しかしながら，当時，HPF があまりにもまだ実験的段階で，実用に耐えないということから，実用に耐えうるための仕様を検討・実装化するため，全国のコンピュータベンダ，ユーザに呼びかけ，HPF 合同検討会（JAHPF）[1]を1997年１月に立ち上げられました．ここでは，月に１回ないし２回というハイペースで議論が行われ，三好氏も時間が許す限り，後見人というような雰囲気で参加され，ともすれば小手先の技術論に陥りそうになる私どもを軌道修正してくださいました．議論の結果は，2001年１月に HPF/JA 言語仕様として公表され，日本電気株式会社および富士通株式会社の HPF コンパイラに採用されています．JAHPF の組織そのものは，仕様の検討という所定の目的を達したため，この時点で解散し，三好氏の強い意志は同年６月，HPF の普及・啓蒙を目指して設立された現在の HPF 推進協議会（HPFPC）[2]に引き継がれました．

三好氏は，残念ながら，地球シミュレータが稼働開始するわずか２ヶ月半前の2001年11月17日に亡くなられました．しかし，三好氏の「それは正しいの？（ベンダの）ご都合じゃないの？」という声が，今でも私どもには聞こえてきます．三好甫氏のご冥福を心からお祈りいたします．

［１］HPF 合同検討会（JAHPF）（http://www.hpfpc.org/jahpf）．
［２］HPF 推進協議会（HPFPC）（http://www.hpfpc.org/）．