HPF応用編

2009年7月

NEC　第一コンピュータソフトウェア事業部
林　康晴

NEC

内容

- HPFによる並列化の基本手順
- 練習用プログラムのHPF化例
- fhpfを使用する場合のコツ
- Information

NEC

HPFによる並列化の基本手順

1. どのループを並列化するかを決める
   - 一番実行コストの高いループ(一般のチューニングと同様)
2. 配列のマッピングを決める
   - 並列化するループがアクセスする次元で分散する
3. HPFコンパイラで翻訳してみる
4. 並列化情報リスト、診断メッセージ等を見て、指示文を追加、修正する
   - 配列のマッピングの追加、修正
   - 並列化可能であること(**INDEPENDENT指示文**)や、通信不要であること(**ON HOME指示文＋LOCAL節**) を明示する指示文の追加
   - その他の指示文の追加
5. 実行性能を確認し、上記の3、4を繰り返す

NEC

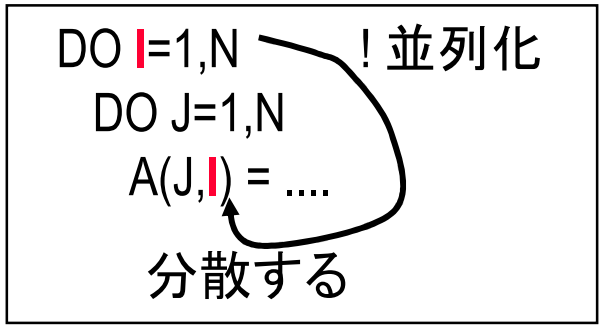
内容

- HPFによる並列化の基本手順
  1. どのループを並列化するかを決める
  2. 配列のマッピングを決める
  3. HPFコンパイラで翻訳してみる
  4. 並列化情報リスト、診断メッセージを見て、指示文を追加、修正する

NEC

配列のマッピングの決め方(1)

- どのループを並列化するか決める。
  - 最も実行コストの高いループ。
    - Fortranコンパイラのプロファイル機能などを利用する。
- 並列化するループがアクセスする次元をマップする。
  - 多重ループの場合、通常は
    ・最外側を並列化
    するのがもっとも効率的。
    - 並列化のオーバヘッドを最小にするため。



  - 多次元配列の場合、メモリアクセスの連続性から、通常は
    ・最終次元をHPF指示文により分散
    するのがもっとも効率的。

NEC

配列のマッピングの決め方(2) 指示文

- 配列宣言と指示文
  - 分散したい次元の上下限が、全ての配列で同一であることが翻訳時に分かるなら、多くの場合**DISTRIBUTE**指示文だけでOK。(マッピングの指定には、**ALIGN**指示文も使用可能)

例:

```
        PARAMETER (IX=100,IY=100)
        REAL*8  A(IX,IY),B(0:IX,IY),C(IX+1,IY)
!HPF$ DISTRIBUTE (*,BLOCK) :: A,B,C
                ...
        DO J=1,IY -1    ! 並列化
          DO I=1,IX
            A(I,J) = B(I,J)-B(I-1,J)+C(I,J+1)-C(I,J)
          END DO
        END DO
```

•DO Jで並列化 → 配列の2次元目でマップ → 2次元目の上下限は全て同じ
⇨ DISTRIBUTE指示文だけで十分

NEC

配列のマッピングの決め方(3) 分散形式

- 分散形式の選択方法
  - 規則的な問題なら、通常**BLOCK**分散または**GEN_BLOCK**分散を利用する(通信や参照時の効率がよい)。
  - 何次元分散するかは、何重並列化するかで決める（＝プロセッサ構成の次元数）。通常は1次元で十分。

例:

```
DO K=1,N ！並列化
  DO J=1,N
    DO I=1,N
      A(I,J,K)  = ....
```

```
1次元分散 （=1次元プロセッサ）
!HPF$ PROCESSORS P(4)
!HPF$ DISTRIBUTE  A(*,*,BLOCK) ONTO  P
```

```
DO K=1,N ！並列化
  DO J=1,N ！並列化
    DO I=1,N
      A(I,J,K)  = ....
```

```
2次元分散 （=2次元プロセッサ）
!HPF$ PROCESSORS P(2,2)
!HPF$ DISTRIBUTE  A(*,BLOCK,BLOCK) ONTO P
```

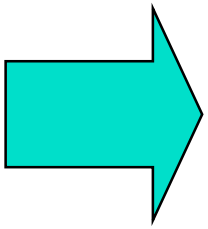

NEC

配列のマッピングの決め方(4)

- 既にマップした配列を基準に、他の配列をマップしてゆく
  - 分散配列が右辺に出現するループ中で定義される配列は、分散配列と同じプロセッサに配置する。

例:

```
real a(10),b(10)
!HPF$ distribute (block)  :: a
      do i=1,n
        b(i) = a(i)
      enddo
```



```
real a(10),b(10)
!HPF$ distribute (block) :: a,b
      do i=1,n
        b(i) = a(i)
      enddo
```

  - 分散配列が実引数になっている場合、特に事情がなければ、対応する仮引数にも同一のデータマッピングを指定する。
    - 場所により実引数の分散が異なる場合、手続のクローニングを利用する
  - 分散配列が仮引数の場合、特に事情がなければ、対応する実引数にも同一のデータマッピングを指定する。
  - 共通ブロック変数をマップする場合、その共通ブロックが出現する全ての手続で同一のマッピングを指定する必要がある。
    - 引数や大域変数に合わせて、呼ばれ側、呼び側手続中の配列もマップしていく。

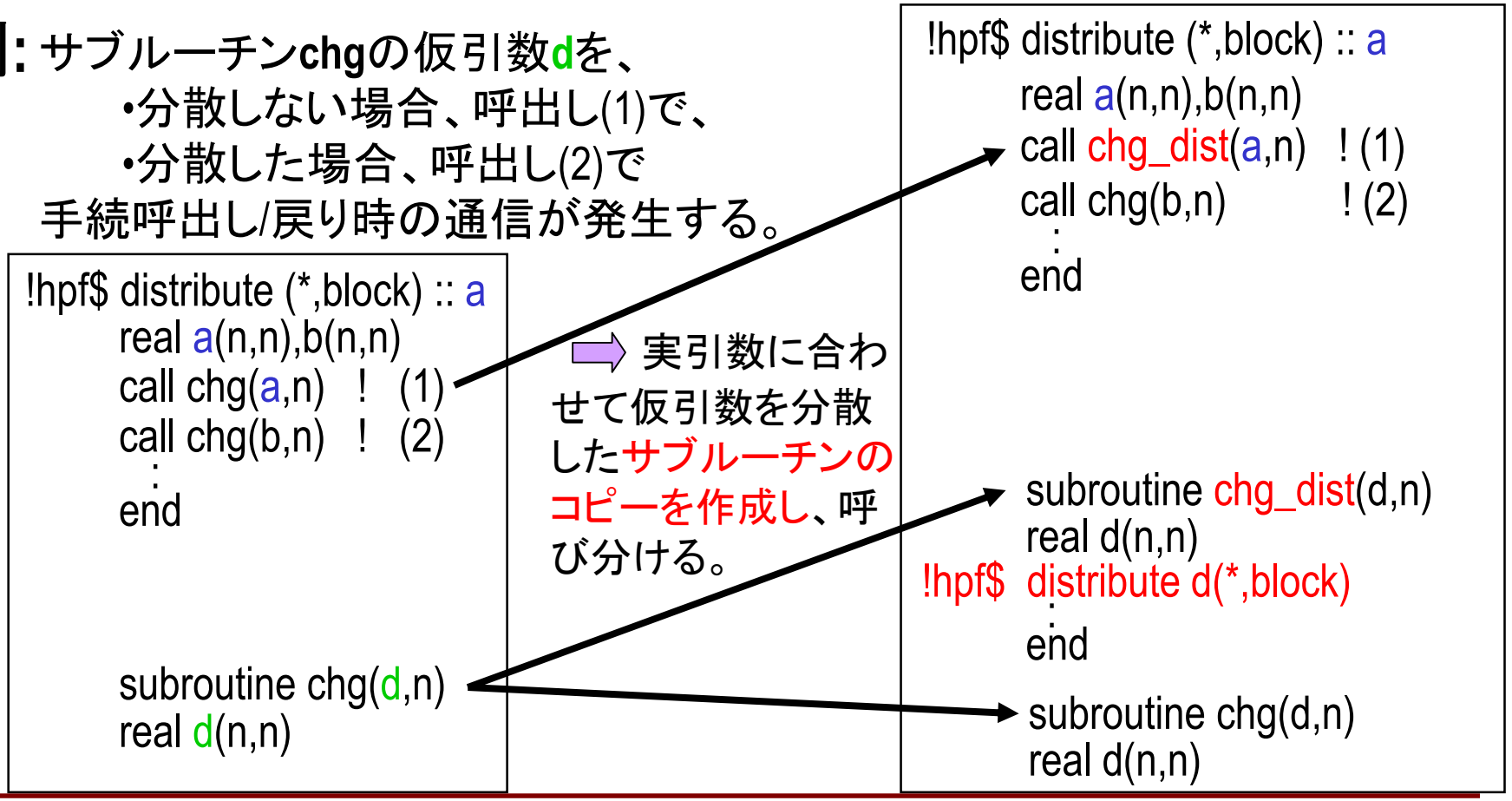
NEC

手続のクローニング

- 1つの手続呼出しに対する実引数のマッピングが、場所により異なる場合に実施する

例: サブルーチン**chg**の仮引数**d**を、
- 分散しない場合、呼出し(1)で、
- 分散した場合、呼出し(2)で

手続呼出し/戻り時の通信が発生する。



NEC

内容

- HPFによる並列化の基本手順
  1. どのループを並列化するかを決める
  2. 配列のマッピングを決める
  3. HPFコンパイラで翻訳してみる
  4. 並列化情報リスト、診断メッセージを見て、指示文を追加、修正する

NEC

HPFコンパイラで翻訳してみる(1)

- 診断メッセージに注目する `%>hpf -Minfo filename.hpf`
  - うまく並列化できたループには
    「independent loop parallelized …」のメッセージ
  - 通信が発生する箇所には
    「expensive communication …」　！コストが高い通信
    「forall is scalarized: complicated communication」！コストが高い通信
    「Array xx not aligned with home array; array copied」！テンポラリへのコピー
    等のメッセージ
  - 原因として考えられること:
    - 配列のマッピングと並列ループの処理分担に不整合がある。
      → データマッピングの指示文を追加、修正する。
    - 並列化できるループなのに、コンパイラが並列化できないと判断した。
      → INDEPENDENT指示文, NEW節, REDUCTION節等を指示する。
    - コンパイラが決めた処理分担が不適当。
      → ON HOME指示文, LOCAL節等を指示する。(境界処理ループや並列化できないループでコストの高い通信が発生する場合、特に有効)

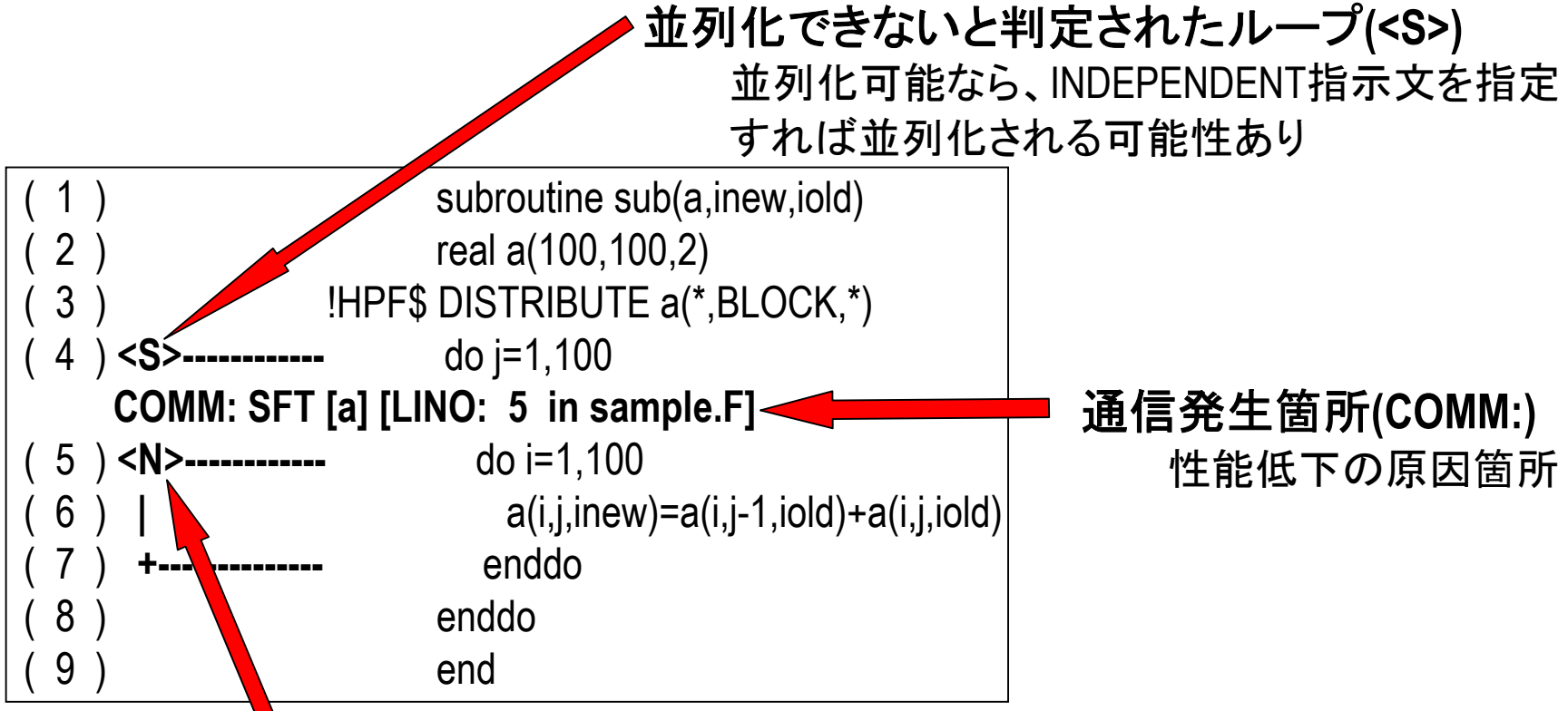
NEC

HPFコンパイラで翻訳してみる(2)

- 並列化情報リストを利用する(-Mlist2オプションにより生成)

```
%> hpf -Mlist2 filename.hpf
```

により並列化情報リストfilename.lstが生成される



NEC

並列化情報リストの使い方(1)

- 並列化情報リストから問題を抽出する(1)
  - 並列化できないと判定されたループの抽出方法

```
%>grep “<S>” filename.lst
(   57) <S>----------    do ij=2,ibar*2,2
```

57行目のdo ijのループは並列化できないと判定された

    - 特にコストの高い通信を伴う場合、注意が必要。INDEPENDENT指示文(+NEW節、REDUCTION節)を指定すれば、並列化される場合が多い。
    - 同じループネストに<P>（並列化された)マークのついたループがあり、通信(COMM:)が出力されていない場合は問題ない。

```
(131) <P>----------          do k=1,ibar
(132)  |  <S> ----------          do n=1,n
```

  - 並列化可能だが並列化されなかったループの抽出方法
    - ループ中の配列をマップすると、並列化される場合あり。
    - 通信を伴わない場合は、問題ないことが多い。

```
%>grep “<N>” filename.lst
(   44) <N>----------    do i=0,ib
```

44行目のdo iのループは並列化できるが並列化しなかった

NEC

並列化情報リストの使い方(2)

- 並列化情報リストから問題を抽出する(2)
  - 通信発生箇所の抽出方法

```
%> grep  COMM:  finename.lst
COMM: SCL [u] [LINO: 137 in filename.hpf]
```

  - 出力フォーマット

```
COMM:  通信パタン [変数名] [LINO:  行番号 in ファイル名]
```

  - 通信パタンとしては、以下のようなものがある
    - マッピング(DISTRIBUTE指示文等)の追加・修正やINDEPENDENT指示文による並列化、ON指示構文+LOCAL節による通信抑制などで削減できる場合も多い

SFT	シフト通信。コストが低いので通常は問題ない。
RED	集計計算。並列ループの外側に生成されているなら通常やむを得ない。並列ループの内側ならREDUCTION節付きINDEPENDENT指示文が有効。
SCL	一要素通信。配列が対象の場合は、通信コストが高いためチェックが必要。
CPY	配列のテンポラリへのコピー。通信対象の配列と、ループ並列化の基準配列との間に、マッピングの不整合がある場合等に発生する。
G/S	Gather/Scatter。間接アクセスループ等で発生し、通信コストが高い場合が多い

NEC

内容

- HPFによる並列化の基本手順
  1. どのループを並列化するかを決める
  2. 配列のマッピングを決める
  3. HPFコンパイラで翻訳してみる
  4. 並列化情報リスト、診断メッセージを見て、指示文を追加、修正する

NEC

指示文の追加によるチューニング 例1

■以下の例では、inew != ioldであることを知っていれば、
INDEPENDENT指示文を指定することにより並列化可能。

```
      SUBROUTINE SUB(A,inew,iold)
      REAL A(100,100,2)
!HPF$ DISTRIBUTE A(*,BLOCK,*)
      DO j=1,100
        DO i=1,100
          A(i,,j,inew) = A(i,j-1,iold)+A(i,j,iold)
        ENDDO
      ENDDO
```

並列化対象

●inew == ioldの時、DO jのループは並列化不可

j=k　: A(i,k,inew) = A(i,k−1,iold)+A(i,k,iold)

j=k+1: A(i,k+1,inew) = A(i,k,iold)+A(i,k+1,iold)

```
-Mlist2      %>grep "<S>" filename.lst
             (   4) <S>----------   do j=1,100
```

```
      SUBROUTINE SUB(A,inew,iold)
      REAL A(100,100,2)
!HPF$ DISTRIBUTE A(*,BLOCK,*)
!HPF$ INDEPENDENT,NEW(i,j)
      DO j=1,100
        DO i=1,100
          A(i,,j,inew) = A(i,j-1,iold)+A(i,j,iold)
        ENDDO
      ENDDO
```

並列化対象

```
-Mlist2      %>grep "<S>" filename.lst
             %>grep "<P>" filename.lst
             (   5) <P>----------   do j=1,100
```

並列化された

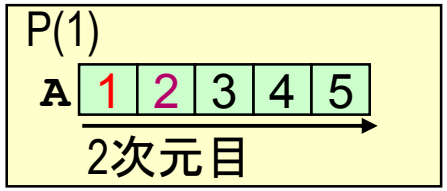
NEC

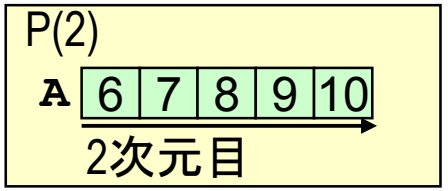
指示文の追加によるチューニング　例2

■以下の例では、A(:,1)を保持するプロセッサ(P(1))だけで実行すると、通信なしで実行可能。(例えば、2プロセッサで実行する場合)

➡ **ON指示構文とLOCAL節を指定して、コンパイラに教える。**

```
      real a(10,10)
!hpf$ processors p(2)
!hpf$ distribute (*,block) onto p :: a
      do i=1,10
        a(i,1)=a(i,2)
      enddo
```






```
-Mlist2     %>grep COMM: filename.lst
            COMM: SFT [a] [LINO: 4 in filename.hpf]
```

```
      real a(10,10)
!hpf$  processors p(2)
!hpf$  distribute (*,block) onto p :: a
!hpf$ on home(a(:,1)), local begin
      do i=1,10
        a(i,1)=a(i,2)
      enddo
!hpf$ endon
```

分散されていない次元は":"を指定する

```
-Mlist2     %>grep COMM: filename.lst
```

何も出力されない = 通信が削減された

- 但し、10プロセッサで実行するような場合は通信が必要なので、**LOCAL節**は指定できない

NEC

内容

- HPFによる並列化の基本手順
- 練習用プログラムのHPF化例
- fhpfを使用する場合のコツ
- Information

NEC

練習用プログラムsample.F

- 3つのプログラム単位：全86行(空白行とコメントを含む)
  - モジュールparam
    - 問題サイズと時間発展ループの繰返し数を大域定数として宣言
  - 主プログラムsample
    - 主要配列は2次元: a(n,n),b(n,n),c(n,n)
    - 初期化後、時間発展ループ中で計算を行い、実行時間とチェックsumを出力
  - サブルーチンbound
    - 境界処理ループだけを含む

  備考: 頻出パタンの練習です。計算内容に意味はありません。
- このままでもHPFコンパイラで翻訳、実行は可能（もちろん、何台で実行しても速くはならない)

NEC

まずは、並列化対象ループを選択する

- コストの高い手続、ループを調べる
  - 「main loop」とコメントのあるループ
    - 本来は、Fortranコンパイラのプロファイル機能などで調べる
- どのループを並列化するかを決める
  - もっともコストの高いループネストの最外側ループ(37行目)

```
12    double precision a(n,n),b(n,n),c(n,n),sum,ap
                              :
! main loop
37        do j=2, n-1
38           do i=2, n-1
39              ix = idxx(i)
40              iy = idxy(j)
41              a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
42           enddo
43        enddo
```


NEC

データマッピングを指定する(1)

- 並列化するループ(37行目)中で定義される配列をマップする
  - do jのループがアクセスする、配列aの2次元目を分散する
    - 通常は、BLOCK分散が効率が良い
    - 問題サイズの関係上、BLOCK分散では負荷バランスが不均等になる場合等、GEN_BLOCK分散が有効な場合もある。
    - 本ループだけを見れば、右辺の配列b,cはマップする必要はない。但し、マップしても通信が出ない場合は、マップしたほうがメモリの節約になる

```
!hpf$ distribute (*,block) :: a
                :
! main loop
37     do j=2, n-1
          do i=2, n-1
             ix = idxx(i)
             iy = idxy(j)
             a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
          enddo
        enddo
```


NEC

アクセスパタン表

- 各配列に対して、並列化対象ループがアクセスする次元の、添字の一覧表を作成すると、最適なデータマッピングの候補を決めるのに便利。
- 例:

```
do j=2, n-1  ！ 並列化対象
  do i=2, n-1
    ix = idxx(i)
    iy = idxy(j)
    a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
  enddo
enddo
```

参照種別＼配列名	a	b	c	idxx	idxy
定義	j				
使用		j, j-1, j+1	*	*	j

- 並列化対象ループ(do j)がアクセスする次元の添字を、定義と使用に分けて一覧にする。
- 配列cやidxxは、並列化対象DOループのDO変数jに対応する次元がないため、「＊」と表示。

既にデータマッピングを決定した配列aを基準とすると

- 配列bは、
  - ALIGN b(j) WITH a(j)
  - ALIGN b(j) WITH a(j-1)
  - ALIGN b(j) WITH a(j+1)
  - 非分散

のいずれかが最適

- 配列idxyは、
  - ALIGN idxy(j) WITH a(j)
  - 非分散

のいずれかが最適

- 配列c、idxxは、
  - 非分散が最適

※実際のALIGN指示文の指定にはテンプレートが必要

NEC

データマッピングを指定する(2)

- マップした配列aが右辺に出現する高コストなループを探す。
  - 52行目のループ → 左辺に出現する配列bを配列aに合わせてマップする。
    - 分散次元の宣言範囲が同一で、添字にもずれがない(共にj)ため、配列aと同様、DISTRIBUTE指示文だけでよい。
    - もちろん、ALIGN指示文やTEMPLATE指示文を利用して、同等のマッピングを指定することも可能。

```
        double precision a(n,n),b(n,n)
!hpf$ distribute (*,block) :: a,b
              :
52      do j=2, n-1
          do i=2, n-1
             ix = idxx(i)
             b(ix,j) = a(i,j) * c(i,j)
             ap = ap * a(i,j)
          enddo
        enddo
```

•ここで、配列bのデータマッピングは、前頁のアクセスパタン表から導出される候補にも適合していることに注意。

NEC

データマッピングを指定する(3)

- マップした配列が実引数となるサブルーチンbound (76行目)
  - サブルーチンboundの呼出し(29行目)における実引数bに対応する仮引数aに、実引数bと同じマッピングを指定する。
    - 特別な理由がない限り、実引数と仮引数のマッピングは同一にする。
    - 並列化できない等の理由で、呼ばれ側手続内で仮引数に対して何度も通信が出てしまう場合には、手続境界で再マッピングしてしまったほうがトータルの通信量が少なくなる場合もある。

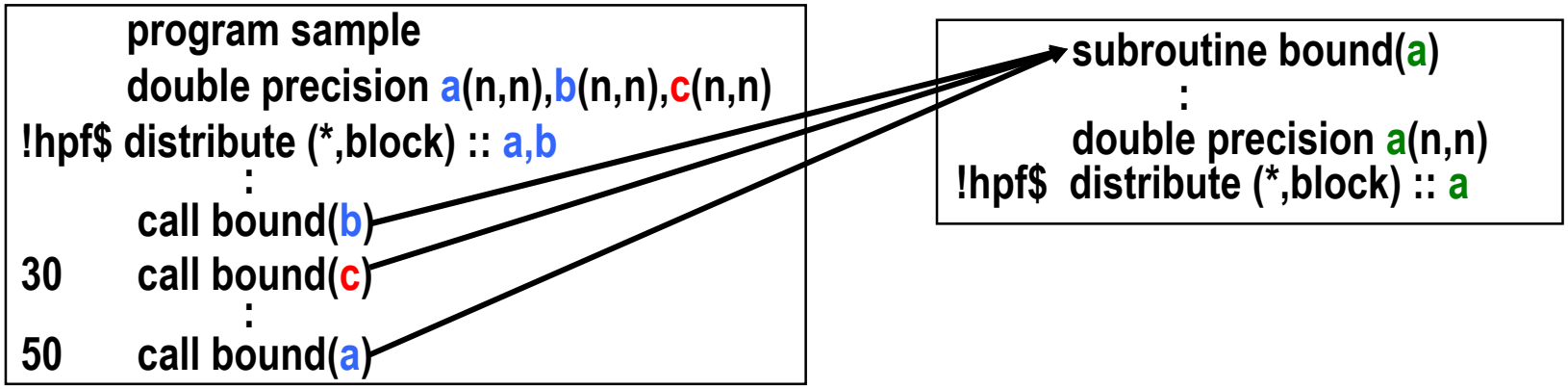
```
        program sample
        double precision a(n,n),b(n,n)
!hpf$ distribute (*,block) :: a,b
                :
29      call bound(b)
                :
        end

76      subroutine bound(a)
        double precision a(n,n)
!hpf$  distribute (*,block) :: a
```


NEC

データマッピングを指定する(4)

- 仮引数をマップした手続boundの呼出箇所をチェックする。
  - 実引数が配列aの場合(50行目)は、仮引数とマッピングが同一。



  - 実引数が配列cの場合(30行目)は、配列cがマップされていないので、サブルーチンboundの呼出し時と戻り時に通信が発生。
    - 実配列引数cを仮引数に合わせてマップするか、サブルーチンboundのクローニングを行うかを決める。
    - この作業を忘れた場合、実行時オプション-hpf –commmsgを付けて実行すると、手続呼出し時に通信/コピーが発生する際、メッセージが出力されるのでチェック可能。

NEC

データマッピングの指定/手続のクローニング

- 配列cはメインループ中で間接アクセスされているので、サブルーチンboundの仮引数に合わせてマップすると、配列aやbとアクセスパタンが一致せず通信が発生する。
  - 配列cはマップしない（非分散。p.22のアクセスパタン表から導出されるマッピングの候補にも適合していることに注意）。
  - サブルーチンboundのクローニングを行う。

```
        double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
              :
! main loop
        do j=2, n-1
          do i=2, n-1
            ix = idxx(i)
            iy = idxy(j)
            a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
          enddo
        enddo
```


NEC

手続のクローニングを行う

- 仮引数をマップしていないこと以外、サブルーチンboundと同一のサブルーチンbound_nodistを作成し、boundの実引数が配列cの場合(30行目)、bound_nodistの呼出しに置き換える。

```
        double precision a(n,n),b(n,n),c(n,n)
!hpf$   distribute (*,block) :: a,b
          :
        call bound(b)
        call bound_nodist(c) ──────→ クローニングした手続を呼び出す
          :
        call bound(a)
          :
        end
        subroutine bound(a)
        double precision a(n,n)
!hpf$   distribute (*,block) :: a
          :
        end
        suboutine bound_nodist(a)
        double precision a(n,n)
          :
        end
```

元の手続

クローニングした手続

NEC

並列化状況のチェック(1)

- –Mlist2、–Minfoオプション付きで翻訳する。
  - 並列化情報リストをチェックする(-Mlist2オプションで生成される)。

```
%> hpf –Mlist2 –Minfo sample.F
                :
%> grep “<S>” sample.lst
(   35) <S>----------     do iter=1,maxiter
(   53) <S>----------       do j=1,n
(   54) <S>----------         do i=1,n
%> grep COMM: sample.lst
      COMM: SFT [b] [LINO: 38 in sample.F]
      COMM: CPY [a] [LINO: 54 in sample.F]
      COMM: SCL [a] [LINO: 54 in sample.F]
      COMM: RED [bp] [LINO: 55 in sample.F]
      COMM: RED [sum] [LINO: 64 in sample.F]
      COMM: SFT [a] [LINO: 83 in sample.F]
      COMM: SFT [a] [LINO: 83 in sample.F]
```

並列化できないと判定されたループの抽出。

時間発展ループ。並列化できない。

53,54行目のループネストは要チェック。

通信発生箇所の抽出。
- SFT(シフト)は多くの場合問題なし。
- RED(集計)は多くの場合やむを得ない。

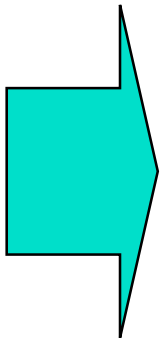
要チェック。上記の53、54行目の<S>に対応して出ている可能性あり。
- CPY(コピー)は、テンポラリへの置き換え(通常通信を伴う)。
- SCL(要素毎)は、配列が対象の場合、高コスト。

NEC

並列化不可と判定されたループのチェック(<S>)

- 53行目のループは、配列a,bの分散次元に対応するdo jで並列化可能だが、コンパイラは自動判定できていない。
  - do jのループにINDEPENDENT指示文を指定する。
  - 但し、変数apに対する集計計算(総積)を含むため、REDUCTION節も必要。
    - REDUCTION節を忘れると結果不正となる。

```
・並列化情報リスト
                        !hpf$  distribute (*,block) :: a,b
                                          :
(    53) <S>------------          do j=1,n
         COMM:  CPY [a] [LINO:  54  in sample.F]
         COMM:  SCL [a] [LINO:  54  in sample.F]
(    54) <S>------------              do i=1,n
         COMM:  RED [ap] [LINO:  55  in sample.F]
(    55)                                ix = idxx(i)
(    56)                                b(ix,j) = a(i,j)*c(i,j)
(    57)                                ap = ap * a(i,j)
(    58)                              enddo
(    59)                            enddo
```



```
!hpf$  distribute (*,block) :: a,b
                :
!hpf$ independent, reduction(ap)
          do j=1,n
             do i=1,n
                ix = idxx(i)
                b(ix,j) = a(i,j)*c(i,j)
                ap = ap * a(i,j)
              enddo
          enddo
```


NEC

並列化状況のチェック(2)

- 再度、–Mlist2、–Minfoオプション付きで翻訳する。
  - 並列化情報リストをチェックする。

```
%> hpf  –Mlist2  –Minfo  sample.F
                :
%> grep  “<S>”  sample.lst
(   35) <S>----------     do iter=1,maxiter
(   55) <S>----------          do i=1,n
%> grep “<P>”  sample.lst
(   38) <P>----------          do j=2,n-1
(   47) <P>----------          do i=1,n
(   54) <P>----------          do j=1,n
(   65) <P>----------       do  j=1,n
%> grep  COMM:  sample.lst
        COMM:  SFT [b]  [LINO:  38  in sample.F]
        COMM:  RED [bp]  [LINO:  54  in sample.F]
        COMM:  RED [sum]  [LINO:  65  in sample.F]
        COMM:  SFT [a]  [LINO:  84  in sample.F]
        COMM:  SFT [a]  [LINO:  84  in sample.F]
```

並列化できないと判定されたループの抽出。

時間発展ループ。並列化できない。

同じループネストの54行目の<S>が消え、<P>に変わっており、重い通信も出ていないので問題なし。

並列化されたループの抽出。

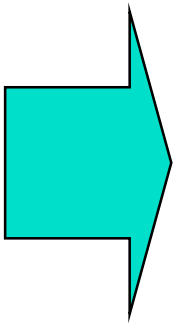
通信発生箇所の抽出。
- CPY/SCLマークは消滅。

NEC

更なる高速化： 境界処理ループの通信削減

- 84行目のループは、シフト通信(SFT)が発生している境界処理ループ。
  - 配列aの分散次元の寸法はn=2047なので、例えば16プロセッサで並列実行する場合、a(:,1)とa(:,2)、およびa(:,n)とa(:,n-1)は同一のプロセッサ上にある。
    - ON HOME指示構文+LOCAL節を指定すれば通信の抑制が可能。
    - 境界処理で頻出のパタン。

```
・並列化情報リスト
                                  double precision a(n,n)
                        !hpf$  distribute (*,block) :: a
                                        :
        COMM:  SFT [a] [LINO:  84  in sample.F]
        COMM:  SFT [a] [LINO:  84  in sample.F]
(    84) <N>----------          do i=1,n
(    85)   |                      a(i,1) = a(i,2)
(    86)   |                      a(i,n) = a(i,n-1)
(    87)   +--------------      enddo
```



```
          double precision a(n,n)
!hpf$  distribute (*,block) :: a
                :
          do i=1,n
!hpf$       on home(a(:,1)), local begin
            a(i,1) = a(i,2)
!hpf$       endon
!hpf$       on home(a(:,n)), local begin
            a(i,n) = a(i,n-1)
!hpf$       endon
          enddo
```


NEC

並列化状況のチェック(3)

- 再度、–Mlist2、–Minfoオプション付きで翻訳する。
  - ◆ 並列化情報リストをチェックする。

```
%> hpf –Mlist2 –Minfo sample.F
                :
%> grep “<S>” sample.lst
(   35) <S>--------------    do iter=1,maxiter
(   55) <S>--------------        do i=1,n
%> grep “<P>” sample.lst
(   38) <P>--------------        do j=2,n-1
(   47) <P>--------------        do i=1,n
(   54) <P>--------------        do j=1,n
(   65) <P>--------------      do  j=1,n
%> grep  COMM:  sample.lst
        COMM:  SFT [b]  [LINO:  38  in sample.F]
        COMM:  RED [bp]  [LINO:  54  in sample.F]
        COMM:  RED [sum]  [LINO:  65  in sample.F]
```

並列化できないと判定されたループの抽出

時間発展ループ。並列化できない。

同じループネストの54行目の<S>が消えて、並列化(<P>)されており、重い通信も出ていないので問題なし。

並列化されたループの抽出。

通信発生箇所の抽出。
- CPY/SCLマークは消滅。

84行目(境界処理部分)のSFT(シフト通信)が消滅。

NEC

完成したら

- 完成したら、翻訳・実行してみる

```
%> hpf  -Minfo  -Mlist2  sample.F  -o  sample    ← 実行ファイルsampleを生成する
%> cat run.sh                                     ←  ジョブスクリプトの例
#PBS -l cputim_job=00:10:00    ← 使用CPU時間を指定する
#PBS -l memsz_job=1gb          ← 使用メモリ量を指定する
#PBS -T vltmpi                 ←  Voltaire MPIを指定する
#PBS -l cpunum_job=1           ← 各ノードの使用CPU(コア)数を指定する
#PBS -b  4                     ← 使用ノード数を指定する
#PBS -q PCS-A                  ← 使用するキューを指定する
cd  sample                     ← 実行ディレクトリへ移動する
mpirun_rsh -np 4 ${NQSII_MPIOPTS} ./sample     ← 実行する(4並列実行の場合）
%>qsub run.sh                                  ←  ジョブを投入する
```

- mpif90の最適化オプション(-fastsse等)も利用可能
- 必要なら、さらにチューニングを行う

NEC

まとめ

- 練習用プログラムのHPF化
  1. 並列化するループを選択する
     - 高コストな手続の高コストなループ
  2. データマッピングを指定する
     - 並列化対象ループ中で定義される配列
     - すでにマップされた配列に合わせて他の配列をマップしていく
     - 仮引数と実引数のマッピングは、原則として同一にする
  3. 手続のクローニングを行う
     - 同じ手続に対する実引数のマッピングが呼出し場所により異なる場合
  4. 自動並列化できないループにINDEPENDENT指示文を指定する
     - 集計計算を含む場合、REDUCTION節も付ける
  5. 境界処理部分には、ON HOME指示構文+LOCAL節を指定する
     - 頻出パタンの通信削減による高速化

NEC

内容

- HPFによる並列化の基本手順
- 練習用プログラムのHPF化例
- fhpfを使用する場合のコツ
- Information

NEC

fhpfを利用する場合のコツ

- まず、一通り以下の作業を済ませてしまう。
  1. 並列化するループには**INDEPENDENT指示文**を指定する
     - 作業変数は、**NEW節**に指定したほうが効率がよくなる。以下のような翻訳時メッセージが出力されたら、NEW変数かどうかチェックしてみる
       - ✓ “line:行番号 内側のDOループのDO変数iがNEW節に指定されていません”
       - ✓ “line:行番号 DOループ内にNEW節にもREDUCTION節にも現れていないスカラ変数に対する代入文があります. (名前:ix)”
     - 集計計算を行っている場合は、**REDUCTION節**の指定も忘れずに。
  2. 隣接要素参照パタンのループがあれば、**SHADOW指示文、REFLECT指示文、ON指示文+LOCAL節**で通信を最適化する
     - 性能の大幅アップが期待できる。
  3. 境界処理部分は、可能なら**ON指示文+LOCAL節**を指定する
     - hpfコマンドの場合と同様。

NEC

隣接要素参照パタンの最適化例(1)

- 隣接要素間の演算を含む場合(37行目のループ)
  - 隣接要素間の参照がある配列bは、**INDEPENDENT指示文だけの指定で並列化すると、通信のため一時的に配列全体がバッファに置き換えられる。**
    - ✓ “line:行番号　変数bのための通信が並列ループの外に生成されます”
  - ➢ **シフト通信が使えるパタンなので、SHADOW指示文、REFLECT指示文、ON指示構文＋LOCAL節を明示的に指定して、ユーザが通信を制御したほうが効率がよい。**

```
!hpf$  distribute (*,block) :: a,b
            :
!hpf$ independent,new(i,j,ix,y)
37    do j=2, n-1
         do i=2, n-1
            ix = idxx(i)
            iy = idxy(j)
            a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
         enddo
      enddo
```


NEC

隣接要素参照パタンの最適化例(2)

- 隣接要素参照パタンに対するシフト通信を指定する
  1. SHADOW指示文でシフト通信用バッファ(袖領域)を確保する
  2. REFLECT指示文で、シフト通信を行う
  3. ON-HOME-LOCAL指示文で通信なしでの実行を指示

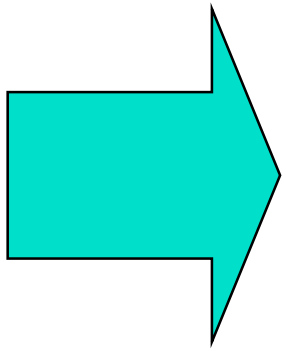
```
!hpf$  distribute (*,block) :: a,b
!hpf$  shadow (0,1) :: b          ! 1. 配列bの分散次元の上方向と下方向に幅1のシャドウ領域
          :
!hpf$ reflect b                   ! 2. 配列bのシャドウ領域へシフト通信
!hpf$ independent,new(i,j,ix,y)
      do j=2, n-1
!hpf$    on home(a(:,j)), local begin   ! 3. 配列a(:,j)に合わせて並列化すると通信が不要
         do i=2, n-1
            ix = idxx(i)
            iy = idxy(j)
            a(i,j) = (b(i,j)+b(i-1,j)+b(i+1,j)+b(i,j-1)+b(i,j+1))*0.2d0*c(ix,iy)*ap
         enddo
!hpf$    endon
      enddo
```


NEC

隣接要素参照パタンの最適化例(3)

- 配列bにSHADOW指示文を指定したので調整；
  1. 配列bが実引数となっているサブルーチンboundの対応する仮引数aに、実引数bと同じSHADOW指示文を指定する。
  2. さらに、サブルーチンboundの実引数となっている配列aにも、配列bと同じSHADOW指示文を指定する。
  - 手続呼出し時/戻り時に、袖領域を一致させるためだけに引数のコピーが発生するのを防止するため。

```
        double precision a(n,n),b(n,n)
!hpf$ distribute (*,block) :: a,b
!hpf$ shadow (0,1) :: b
                :
        call bound(b)
        call bound_nodist(c)
                :
        call bound(a)
        end
        subroutine bound(a)
        double precision a(n,n)
!hpf$  distribute (*,block) :: a
```



```
        double precision a(n,n),b(n,n)
!hpf$ distribute (*,block) :: a,b
!hpf$ shadow (0,1) :: b,a
                :
        call bound(b)
        call bound_nodist(c)
                :
        call bound(a)
        end
        subroutine bound(a)
        double precision a(n,n)
!hpf$  distribute (*,block) :: a
!hpf$  shadow (0,1) :: a
```

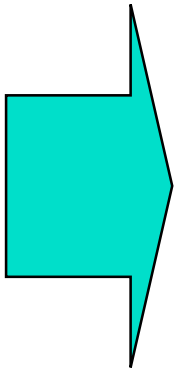

NEC

INDEPENDENT指示文とNEW節の指定例

- 以下のループ(25行目)は自動並列化可能だが、INDEPENDENT指示文とNEW節を指定した方が効率が良い。(自動に任せると、DO変数i,jに対する終値保障のオーバヘッドが発生する)

```
      double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :

      do j=1,n   ! 並列化
        do i=1,n
          b(i,j) = 1.0d0 ! 2次元目で分散
          c(i,j) = 1.0d0 ! 非分散
        enddo
      enddo
```



```
      double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :
!hpf$ independent, new(i,j)
      do j=1,n   ! 並列化
        do i=1,n
          b(i,j) = 1.0d0
          c(i,j) = 1.0d0
        enddo
      enddo
```

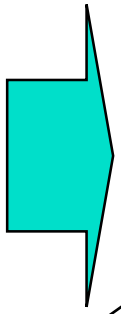
- ✓ ここで、並列ループ中で定義される配列b(2次元目BLOCK分散)と配列c(非分散)のマッピングが異なることに注目!!
- ✓ このまま並列化すると非分散の配列cに対して通信が発生する。

NEC

ループの分割による通信の削減例

- マッピングの異なる配列が1つのループ中で定義されている場合、ループを分割すると、通信の削減が可能。

```
      double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :
!hpf$ independent, new(i,j)
      do j=1,n
        do i=1,n
          b(i,j) = 1.0d0 ！2次元目で分散
          c(i,j) = 1.0d0 ！非分散
        enddo
      enddo
```



```
      double precision a(n,n),b(n,n),c(n,n)
!hpf$ distribute (*,block) :: a,b
        :
!hpf$ independent, new(i,j)
      do j=1,n
        do i=1,n
          b(i,j) = 1.0d0 ！2次元目で分散
        enddo
      enddo
!hpf$ independent, new(i,j)
      do j=1,n
        do i=1,n
          c(i,j) = 1.0d0 ！ 非分散
        enddo
      enddo
```

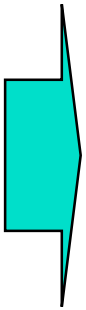
1つのループネスト中で定義される配列のマッピングを一種類にする。

NEC

fhpfの出力ソース(～.mpi.f)簡易解析法

- MPI_Allreduceの出現箇所とその第一引数に注目する
  - fhpfにおいて、コストの高い通信には、MPI_Allreduceが使われている場合が多い。
  - 第一引数が、元のソースにはなかった変数なら要注意。(コンパイラによってテンポラリが生成されたことを意味している)
  - 通信の発生箇所を調べるには、第一引数が使われているループを探す。
  - そのままでは、元のソースとの対応が取りづらいので、あらかじめ元のソースに文番号として行番号をつけておけば、対応がとりやすい。以下の例では、27行目からのループ中の、配列cに対して通信が出力されていると推測できる。

```
        double precision a(n,n),b(n,n),c(n,n)
!hpf$  distribute (*,block) :: a,b
             :
!hpf$ independent, new(i,j)
27      do j=1,n
28        do i=1,n
29          b(i,j) = 1.0d0
30          c(i,j) = 1.0d0
31        enddo
32      enddo
```

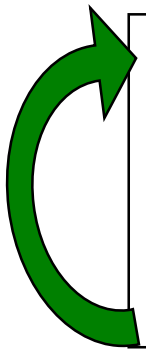


```
        do j=spmd_start, spmd_end ! 並列化
28        DO i=1,2047,1
29          b(i-1,j) = 1.0D0
30          cX0(i,j+spmdX1+1) = 1.0D0
            maskX1(i,j+spmdX1+1) = ....
            workX0(i,j+spmdX1+1) = cX0(i,...)
31        CONTINUE
        ENDD
        CALL MPI_Allreduce(workX0,.....
        CALL MPI_Allreduce(maskX1,.....
```


NEC

行番号追加スクリプトln.plの使い方

- 固定形式のFortran(またはHPF)ソースに、文番号として行番号を付加する簡易的なperlスクリプト
  - 入力ファイルのサフィックスは、～.fまたは～.F
  - 出力ファイルのサフィックスは、～.hpf



```
%> perl   ./ln.pl  source.F          ← 行番号が付加されたsource.hpfを生成
%> fhpf  source.hpf                  ← MPIプログラムsource.mpi.fを生成
%> grep  MPI_Allreduce  source.mpi.f  ← 重い通信がないかチェック
%> less  source.mpi.f                ← 通信発生箇所と通信対象配列を確認
%> vi  source.F                      ← 通信発生箇所を修正
```


NEC

完成したら

- 完成したら翻訳、実行してみる

```
%> fhpf  sample.F                              ← MPIプログラムsample.F.mpi.f90に変換する
fhpf V1.4.4/MPI -- HPF translator for Linux system
sample.F -> sample.F.mpi.f90
%> mpif90  sample.F.mpi.f90  -o  sample        ← 実行ファイルsampleを生成する
%> cat run.sh                                  ← ジョブスクリプトの例
#PBS -l cputim_job=00:10:00    ← 使用CPU時間を指定する
#PBS -l memsz_job=1gb          ← 使用メモリ量を指定する
#PBS -T vltmpi                 ← Voltaire MPIを指定する
#PBS -l cpunum_job=1           ← 各ノードの使用CPU(コア)数を指定する
#PBS -b  4                     ← 使用ノード数を指定する
#PBS -q PCS-A                  ← 使用するキューを指定する
cd  sample                     ← 実行ディレクトリへ移動する
mpirun_rsh -np 4 ${NQSII_MPIOPTS} ./sample     ← 実行する(4並列実行の場合)
%>qsub run.sh                                  ← ジョブを投入する
```

- mpif90の最適化オプション(-fastsse等)も利用可能

NEC

fhpfを利用する場合のまとめ

1. 並列化したいループには、基本的にINDEPENDENT指示文を指定する
   - DO変数・作業変数は、NEW節中に指定したほうが効率が良い。
   - 集計計算を含む場合、REDUCTION節も指定する。
2. 通信の削減
   - 隣接要素参照パタンのループには、SHADOW指示文、REFLECT指示文、ON-HOME-LOCAL指示文を指定する。
   - 境界処理部分には、可能ならON-HOME-LOCAL指示文を指定する。
   - 1つのループで定義される配列の分散が異なる場合、分散を変更したり、ループを分割すると良い。

NEC

内容

- HPFによる並列化の基本手順
- 練習用プログラムのHPF化例
- fhpfを使用する場合のコツ
- Information

NEC

Information

- HPF推進協議会 (HPFPC)
  - HPFによる並列計算の普及促進を目的とし、HPCユーザとベンダーが参加
  - HPF講習会開催、HPFプログラミングガイドの公開、サンプルコードの公開
  - HPF2.0仕様書の和訳、HPF/JA拡張仕様の策定
- HPFPCのホームページ: http://www.hpfpc.org/

  以下のドキュメントやプログラム、フリーのコンパイラfhpfのダウンロードが可能
  - HPFプログラミングのテキストや、過去の講習会資料
  - HPF2.0仕様書(日本語訳)、HPF/JA仕様書
  - サンプルコード
  - 最新版fhpfコンパイラのダウンロードページ(http://www.hpfpc.org/fhpf/fhpf.html)
  - 参加(会費無料)を希望される方は右記参照 (http://www.hpfpc.org/nyuukai.html)
- 大阪大学レーザーエネルギー学研究センター公開テキスト

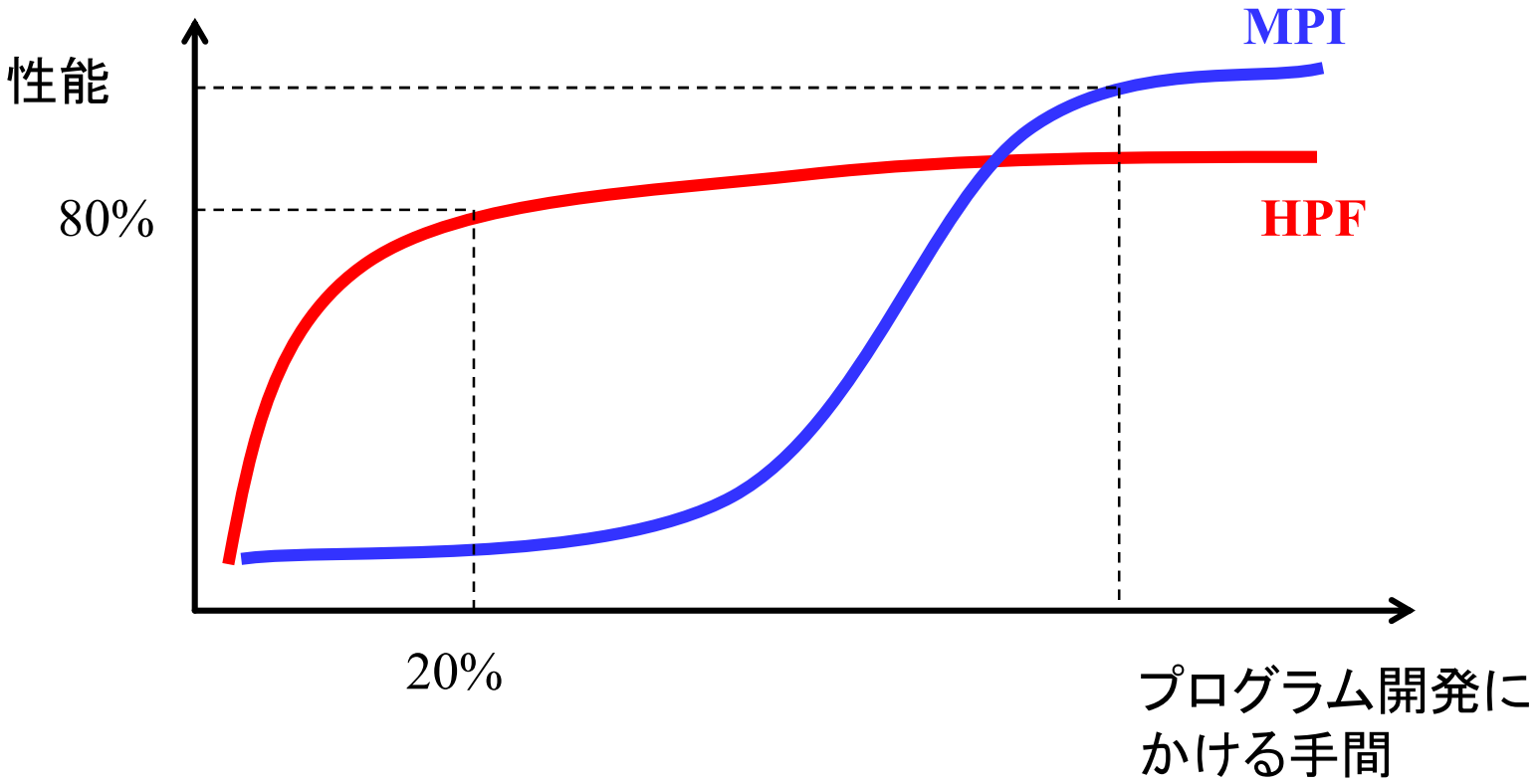
  (http://www.ile.osaka-u.ac.jp/research/comp/text.html)
- 核融合科学研究所のQ&A

  (http://www.dss.nifs.ac.jp/workgr/QandA/QandA-f90hpf.html)

NEC

最後に

- 「MPIの 20%の手間 で 80%の性能」
- プログラム開発、保守を効率化し、本来の研究に注力できる



NEC