HPF応用編2
HPF/SX V2プログラミング

平成19年7月

NEC

内容

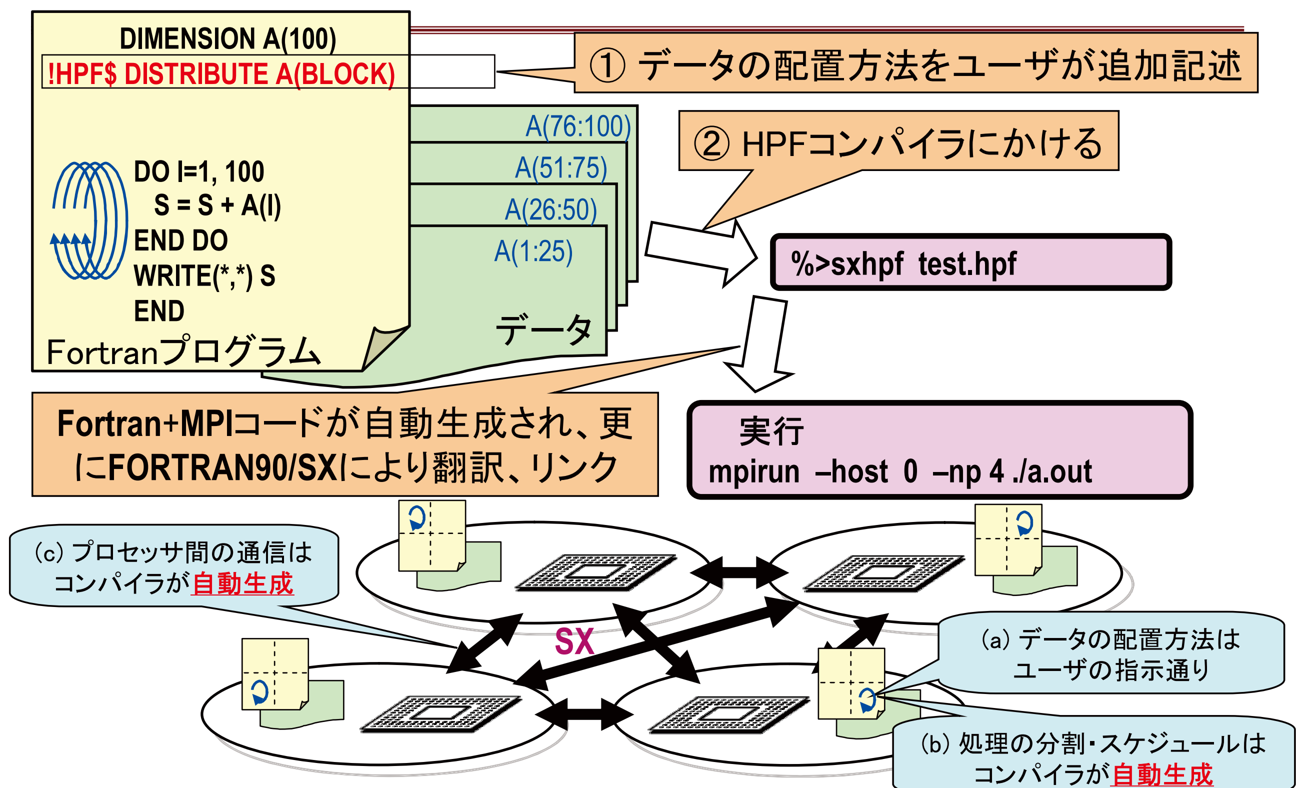
- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

HPF/SX V2の特徴

- NEC SXシリーズ向けHPFコンパイラ
- 自動分散並列化機能
  - リダクションを含む並列ループの自動認識
  - 配列に対する通信用バッファ領域の自動割付
- HPF2.0仕様に加え、さまざまな拡張仕様をサポート
  - 主要なHPF 公認拡張仕様、HPF/JA拡張仕様
  - 不規則問題(有限要素法等)向け独自拡張仕様
- HPFプログラム向けデバッグ、チューニング機能
  - 並列化情報リスト、並列化情報診断メッセージ
  - 様々なデバッグ用オプション
- 上級者向け機能
  - 部分的にMPIを利用したチューニングが可能
  - SXの持つ3階層の並列性をミックスして利用可能
    - ベクトル化
    - 共有メモリ並列化（ノード内）
    - 分散メモリ並列化（ノード間）

NEC

HPF/SX V2 によるプログラム開発イメージ



NEC

コンパイル方法

- コンパイラの起動
  - クロスコンパイラ環境（フロントエンド，会話形式）

```
%> sxhpf ［オプション］ ファイル名 …
```

  - セルフコンパイラ環境（SX本体）

```
%> hpf ［オプション］ ファイル名 …
```

- 入力ファイル名

	前処理なし	前処理あり
固定形式	*.hpf *.f *.for	*.F
自由形式	*.f90	*.F90

※ 翻訳時オプションで変更可能

※ .fの場合、-Mftnオプションは使用できない。

NEC

主なコンパイルオプション

オプション	意味
-Mftn	FORTRAN90/SXによる翻訳を省略(開発時コンパイル時間短縮)
-Mextend	ソースの1行を132文字までとする(固定形式の場合に有効)
-Mfixed/free	ソースを固定形式(既定値)／自由形式として扱う
-Minfo	診断メッセージを出力する
-Mlist2	並列化情報リストを出力する
-Moverlap=size:n	既定のシフト通信用バッファサイズ（n=4）を変更する
-Mscalarnew	定義されるスカラ変数をすべてNEW変数と扱う
-Mnomapnew	定義される、マップされていない配列をすべてNEW変数と扱う
-Msubchk	範囲外アクセス検出（デバッグ用）
-Mcommonchk	共通ブロックの矛盾検出（デバッグ用）
-Mnoindependent	INDEPENDENT指示文を無視(デバッグ用)
-Mnoentry	実行時のエラーチェックを省略(完成時の性能測定用)

NEC

実行方法

■ 実行（mpirunコマンドで指定）

```
mpirun [MPIオプション] 実行ファイル名 [引数等] [HPFオプション]
```

■ 例

```
#実行ファイルa.outを、4プロセスで並列実行
mpirun –np 4 ./a.out

#実行ファイルa.outを、ホスト0上で、4プロセス並列実行
mpirun –host 0 –np 4 ./a.out

#ホスト0上で8プロセス、ホスト1上で8プロセスの、16並列実行
mpirun –host 0 –np 8 –host 1 –np 8 ./a.out

#HPFオプション-commmsg付きで、4プロセス並列実行
mpirun –np 4 ./a.out -hpf -commmsg
```


内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - ◆ 例題コード1のHPF化
  - ◆ 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

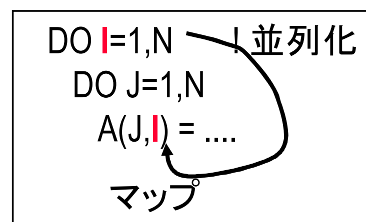
HPF/SX V2プログラミングの基本手順

1. コードのクリーンナップ
2. どのループを並列化するか決める
   - 一番実行コストの高いループ
3. 配列のマッピングを決める
   - 並列化するループに対応する次元でマップ
4. HPFコンパイラで翻訳してみる
5. 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
   - 配列のマッピングの追加、修正
   - 並列化可能であること(INDEPENDENT)や通信不要であること(ON, LOCAL) を明示する指示文の追加
   - その他の指示文やFORTRAN90/SXコンパイラ指示行追加
6. 上記の4、5を繰り返す。

NEC

配列のマッピングの決め方

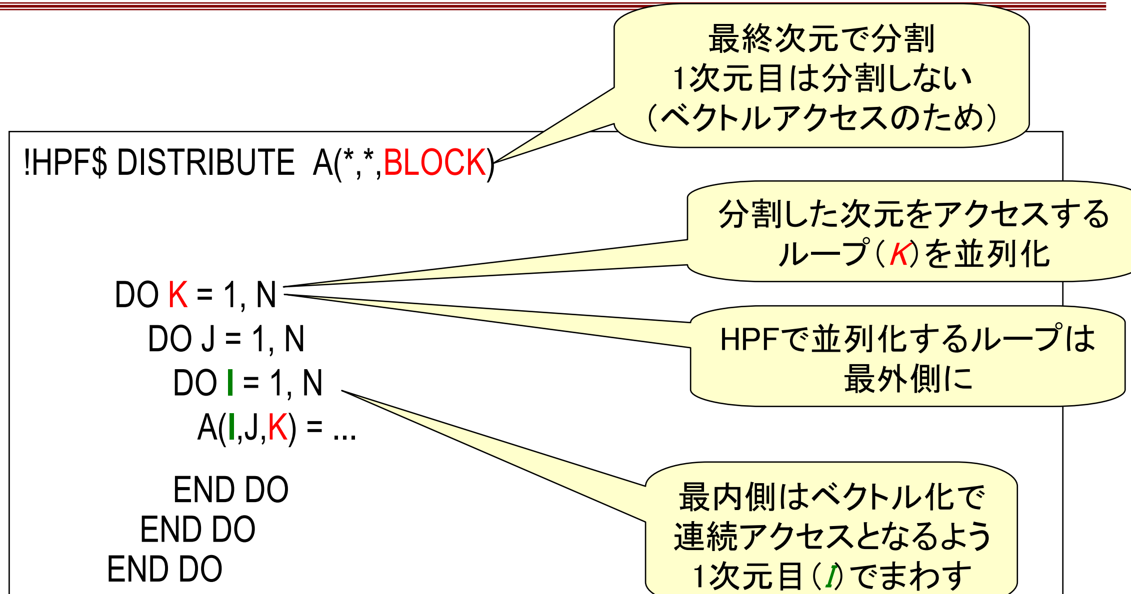
- どのループを並列化するか決める
  - ◆ もっとも実行コストの高いループ
  - ◆ 最終結果を計算しているループ
- 並列化するループでアクセスされる配列の次元をマップ
  - ◆ 多重ループの場合、通常は
    ー最内側をベクトル実行
    ー最外側を並列化
    するのがもっとも効率的



  - ◆ 多次元配列の場合、メモリアクセスの連続性から、通常は
    ー1次元目をベクトルアクセス
    ー最終次元をHPFにより分散
    するのがもっとも効率的

NEC

理想的な並列化パタン



◆ 可能ならば、この形に近づくよう
ループ交換などの書き換えを行ったほうがよい

NEC

並列化やマッピングの指示の追加(1)

■ 診断メッセージに注目 %>sxhpf -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化されるループとの対応の不整合
    → マッピングを見直す
  - コンパイラがループの並列化可能性を見切れていない
    → INDEPENDENT, NEW, REDUCTION等を指示
  - コンパイラの計算マッピングが不適当
    → ON HOME, LOCAL等を指示

NEC

並列化やマッピングの指示の追加(2)

■ 並列化情報リストから問題を抽出(1)

`%> sxhpf –Mlist2 filename.hpf` で並列化情報リストfilename.lstを生成

◆ 並列化できないと判定されたループの抽出

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

- 特に、コストの高い通信を伴う場合、チェックが必要
- 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、通信が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

- 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
```


並列化やマッピングの指示の追加(3)

■ 並列化情報リストから問題を抽出(2)

◆ 通信発生箇所の抽出

```
%> 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 HOME, LOCAL**指示文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。コストが低いので問題ない。
RED	リダクション。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
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。
CPY	配列のテンポラリへのコピー。通信対象と、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。

並列化やマッピングの指示の追加(4)

・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

↓

```
      SUBROUTINE SUB(A,inew,iold)
      REAL A(100,100,2)
!HPF$ DISTRIBUTE A(*,BLOCK,*)
!HPF$ INDEPENDENT,NEW(i)
      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　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
```

↓

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


内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- Fortranプログラムの HPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

例題コード1 主要部

```
! メインデータ: 1次元配列
      parameter(np=139264,ndim=np+2,isteps=3)
     real*8, dimension(ndim)  ::vx,vy,vz,am,fx,fy,fz,pot,presx,presy,presz,rx,ry,rz
! メインループ:  配列rx,ry,rzの各要素のペアの値に対して物理量を計算
! メインループはistepsサイクル反復実行される
      do j=1,np
      do i=1,np
         if(i.ne.j)then
           xij0 = rx(i) – rx(j)
             …
           xij = 0.5d0 * (sign(1.d0,0.5d0*xcel-abs(xij0))+1.d0)*xij0 ......
             …
           rij = dist(xij,yij,zij)
             …
           fx(j) = fx(j) – fijxp
             …
           pot(j) = pot(j) + p
           presx(j) = presx(j) + presijxp
             …
      enddo
      enddo
```


NEC

例題コード1(1) 配列のマッピングを決める

■ メインのループ

```
do j=1,np ！並列化対象
 do i=1,np
   if(i.ne.j)then
     xij0 = rx(i) – rx(j)
     :
     fx(j) = fx(j) – fijxp
     :
 enddo
 enddo
```

➡ 並列化対象ループに対応する次元で配列を分散

```
!hpf$  distribute (block) :: vx,vy,vz,am,fx,fy,fz,pot,presx,presy,presz
!hpf$& ,rx,ry,rz  ！継続行
```

DISTRIBUTE指示文: 配列をプロセッサ上にマップ

BLOCK : 均等に分割

例題コード1(2) 並列化、通信状況のチェック1

■ −Mftn、−Minfo、−Mlist2 オプション付きで翻訳

```
%>sxhpf   -Mftn -Minfo  -Mlist2 sample1.F
```

−Mftn ：HPFによる翻訳のみを行いFortranによる翻訳を省略（コンパイル時間節約のため）

−Minfo ：診断メッセージを発行

−Mlist2：並列化情報リストsample1.lstを出力

■ 出力される診断メッセージ

行番号

```
54,  Independent loop
     14 FORALLs generated
73,  Independent loop                           ！並列化可能と自動判定された
     Independent loop parallelized : am(n)      ！並列化された
        :
128, Independent loop
     Array rx not aligned with home array; array copied  ！テンポラリへのコピー
        :
     Independent loop parallelized : rx(j)
     communication is generated: array copy     ！定型的な通信が発生した
        :
```

例題コード1(3) 並列化、通信状況のチェック2

■ 並列化情報リストファイル sample1.lst によるチェック

◆ 並列化状況

```
%>grep "<S>" sample1.lst        ! 並列化できないと判定された
%>grep "<N>" sample1.lst        ! 並列化可能だが並列化されなかった
```

⇨ 何も出力されないので問題なし

◆ 通信状況

```
%>grep "COMM:" sample1.lst
COMM: CPY [rx] [LINO: 128 in sample1.F]        ! 配列rxのテンポラリへのコピー
COMM: CPY [ry] [LINO: 128 in sample1.F]        ! 配列ryのテンポラリへのコピー
COMM: CPY [rz] [LINO: 128 in sample1.F]        ! 配列rzのテンポラリへのコピー
COMM: RED [potsum] [LINO: 170 in sample1.F]    ! スカラpotsumに対するリダクション
        :
```

⇨ •スカラに対するリダクションは問題なし。
•通信発生のメッセージCPY(128行目)をチェック

NEC

例題コード1(4) 通信発生場所の調査

■テンポラリへのコピーが発生する128行目のループ（メインループ）

```
128, Independent loop                                   ! 並列化可能なループと判定された
     Array rx not aligned with home array; array copied ! テンポラリへのコピー
       :
     Independent loop parallelized : rx(j)              ! rx(j)を基準として並列化された
     communication is generated: array copy             ! 定型的な通信が発生した
       :
```

```
!hpf$ distribute (block) :: rx,fx,....
                  ループ直前にテンポラリへのコピー発生
128)  do j=1,np   ! 並列化
      do i=1,np   ! 各jに対しiは1～np
        if(i.ne.j)then
          xij0 = rx(i) – rx(j)
              :
          fx(j) = fx(j) – fijxp
              :
      enddo
```

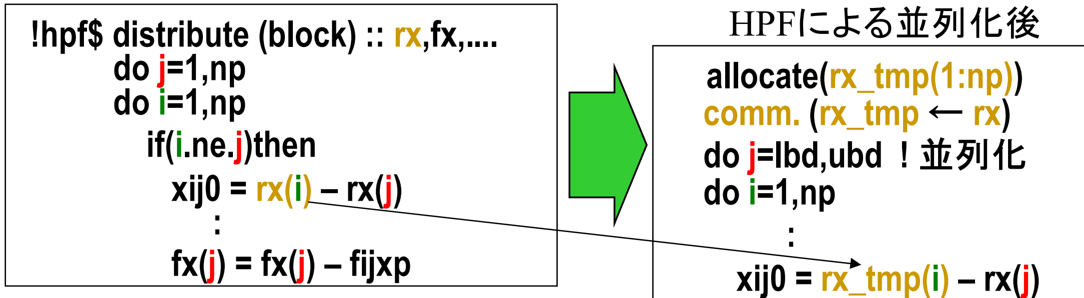
並列化されるループ(j) の各繰り返しでrxの全ての要素が参照されているため、各プロセッサがそれぞれ、rxの全体を必要とする。

⇨ rxは、DISTRIBUTE指示文でプロセッサ間に分散されているため、通信必要。

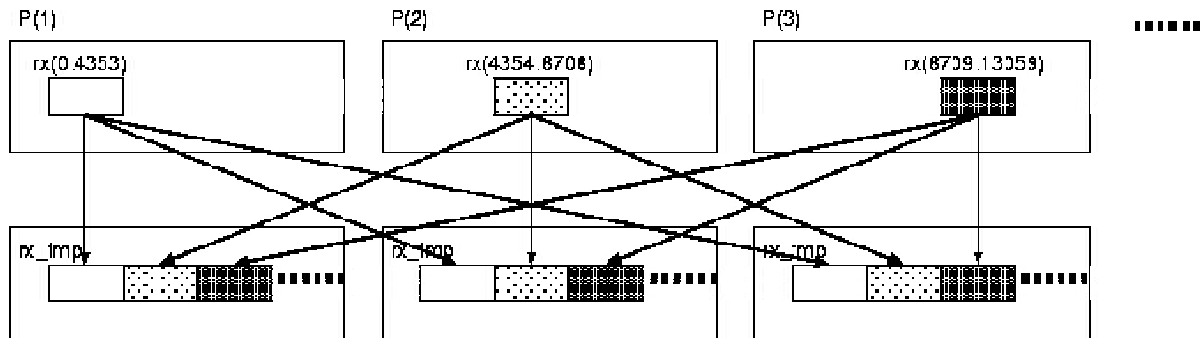
NEC

例題コード1(5) 参考: コンパイラの変形イメージ

■ 生成される通信とテンポラリ配列のイメージ



各プロセッサ上に、テンポラリが確保されるため、使用メモリ量は増大



NEC

例題コード1：性能値

Num. of Proc.	経過時間(s)	SpeedUp	GFLOPS	ベクトル化率(%)	ベクトル長	メモリ(GB)
(非並列)	802.1		9.00	99.74	256.0	0.064
1	806.4	0.99	8.96	99.72	256.0	0.074
2	402.6	1.99	17.92	99.72	256.0	0.164
4	201.6	3.98	35.81	99.72	256.0	0.437
8	102.4	7.83	70.68	99.72	255.6	1.358
16	52.5	15.28	139.47	99.72	255.3	2.954

(SX-8性能参考値)

·ソース規模　248行中
HPF指示文　　2行(0.8%)

行数	指示文内訳
2	DISTRIBUTE指示文

·DISTRIBUTE指示文は、継続行のため2行

NEC

内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

例題コード2 主要部

```
! メインデータ: 3次元配列
        parameter(ibar=636,ib=ibar+2,isteps=3)
        real*8, dimension(0:ib,0:ib,0:ib) :: u,v,w,p,un,vn,wn,co,cov

! メインループ(red-black法)
! 収束するまでの反復実行を1サイクルとし、それがisteps-1サイクル実行される
        do k=1,ibar,2
!CDIR NODEP   ! ベクトル化を阻害する依存がない事を明示するFORTRAN90/SX指示行
        do n=1,n1
           i=li(n)  ! 1次元目添字(間接アクセス)
           j=lj(n)  ! 2次元目添字(間接アクセス)
           div = (u(i,j,k)-u(i-1,j,k))*rdx+(v(i,j,k)-v(i,j-1,k))*rdy+(w(i,j,k)-w(i,j,k-1))*rdz
           dp = -co(i,j,k)*div
           u(i,j,k)= u(i,j,k) + cov(i) *dt*dp*rdx
           u(i,j,k) = u(i-1,j,k) –cov(i-1)*dt*dp*rdx
                   …
           p(i,j,k) = p(i,j,k) + dp
           dmax = max(dmax,abs(div))
        enddo
!CDIR NODEP
        do n=n1+1,n2
          .....
```


NEC

例題コード2(1) 配列のマッピングを決める

■ メインのループ

```
          do k=1,ibar,2  ! 並列化対象
!CDIR NODEP
          do n=1,n1      ! ベクトル化対象
              i=li(n)
              j=lj(n)
               :
             u(i,j,k) = u(i,j,k)+cov(i) *dt*dp*rdx
               :
           enddo
               :
```

並列化対象ループに対応する次元で3次元配列を分散

```
!hpf$ distribute (*,*,block) :: u,v,w,p,un,vn,wn,co
```

DISTRIBUTE指示文: 配列をプロセッサ上にマップ
BLOCK : 均等に分割 、 ＊: 分散しない

NEC

例題コード2(2) 並列化、通信状況のチェック1

■ −Mftn、−Minfo、−Mlist2 オプション付きで翻訳

```
%>sxhpf  -Mftn -Minfo  -Mlist2 sample2.F
```

■ 出力される診断メッセージ

```
40,    Independent loop     ! 並列化可能なループと自動的に判定された
          :
147,   Distributing inner loop; 2 new loops
       expensive communication: scalar communication... ! 高コストな通信
       expensive communication: scalar communication... ! 高コストな通信
          :
181,   Distributing inner loop; 6 new loops
          :
       forall is scalarized: complicated communication     ! 非定型の通信
          :
260,   Distributing inner loop; 3 new loops
          :
       expensive communication: scalar communication... ! 高コストな通信
```

･主要なループ部分はIndependent loopであると判定できておらず、並列化もされていない。

NEC

例題コード2(3) 並列化、通信状況のチェック2

■ 並列化情報リストファイル sample2.lst によるチェック

```
%>grep "<S>" sample2.lst  ! 並列化できないと判定されたループを抽出
        (    67) <S>------------------      do ij=2,ibar*2,2
        (    68) <S>------------------        do i=1,min(ibar,ij-1)
                  :
        (   131) <S>------------------      do k=1,ibar
        (   132) <S>------------------      do n=1,n2
        (   147) <S>------------------      do n=1,nll
                  :
        (   257) <S>------------------      do k=1,ibar,2
        (   260) <S>------------------      do n=1,n1
        (   277) <S>------------------      do n=n1+1,n2
```

```
%>grep "COMM:" sample2.lst   !通信情報を抽出
        COMM: SCL [u] [LINO:  147 in sample2.F]    ! 配列uに対する1要素通信
                 :
        COMM: CPY [dp] [LINO:  260 in sample2.F]  ! テンポラリへのコピー
```


NEC

例題コード2(4) 並列化不可のループのチェック(1)

・並列化情報ファイルsample2.lst

```
(    67) <S>------       do ij=2,ibar*2,2
(    68) <S>------         do i=1,min(ibar,ij-1)
(    69)                     j=ij-i
(    70)                     if (j.le.ibar)then
(    71)                       n = n+1
(    72)                       Ii(n)=i
(    73)                       Ij(n)=ij-i
                  :
```

・(71) n=n+1は、ループの前の繰返しの値を使っているので並列化できない

・参照されている配列Ii,Ijは、共にマップされていないので、ループは並列化されず、通信も発生しない。主要部分でなければ並列化されていなくてもそれほど問題はない。(逆に並列化したい場合は、出現する配列をマップする必要がある)

•他にも2つ(78,79行目、95,96行目)、同パタンのループあり

NEC

例題コード2(5) 並列化不可のループのチェック(2)

```
(  131) <S>----         do k=1,ibar
(  132) <S>----           do n=1,n2
(  133)                       i=li(n)
(  134)                       j=lj(n)
(  135)                       co(i,j,k) = orc/(dt*
(  136)                 &         ( rdx2*(cov(i-1)+cov(i))
(  137)                 &         ( rdy2*(cov(j-1)+cov(j))
(  138)                 &         ( rdz2*(cov(k-1)+cov(k)) ) )
```

・マップされた配列coの分散次元(3次元目)に対応するループkで並列化可能

➡ 並列化可能と判定できていない(<S>)ので、INDEPENDENT指示文を追加

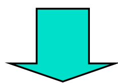
```
!hpf$ independent,new(n,i,j)
      do k=1,ibar
      do n=1,n2
         i=li(n)
         j=lj(n)
         co(i,j,k) = orc/(dt* ....
```

•他にも2つ(179,180行目、241,242行目)、同パタンのループあり

NEC

例題コード2(6) 並列化不可のループのチェック(3)

```
(257) <S>-----        do k=1,ibar,2          ! メインループ
            :
(260) <S>-----        do n=1,n1
       COMM: RED [p] [LINO: 261 in sample2.F]
(261)                    i=li(n)
(262)                    j=lj(n)
                          :
(267)                    u(i,j,k)= u(i,j,k) + cov(i) * dt*dp*rdx
                          :
(274)                    dmax=max(dmax,abs(div))
```



変数dmaxに対するリダクション(最大値)を含むが、並列化可能

➡REDUCTION節付きINDEPENDENT指示文を指定

```
!hpf$   independent,new(k,n,i,j,div,dp),reduction(dmax)
        do k=1,ibar,2
!CDIR NODEP
        do n=1,n1
           :
           u(i,j,k)= u(i,j,k) + cov(i) * dt*dp*rdx
           :
           dmax=max(dmax,abs(div))
```

•他にも2つ(297,299行目、346,347行目)、同パタンのループあり

NEC

例題コード2(7) 並列化不可のループのチェック(4)

```
(  147) <S>----        do n=1,nll
(  148)                    i=ll1(n)
(  149)                    j=ll2(n)
                            :
(  151)                      u(i,j,0)=-u(i,j,1)
(  152)                      v(i,j,0)=-v(i,j,1)
                               :
(  156)                      u(i,j,ibar1)=-u(i,j,ibar)      ! ibar1=ibar+1
```

·境界処理のループ。DO nのループの異なる繰り返しで、配列u,vの同じ要素が定義される可能性があるので、一般的には、並列化不可

·DO変数(n)と配列のマップされた次元(u,vの3次元目)が対応していない

⇨ 各反復を全プロセッサが実行　→　通信が発生

```
147,  Distributing inner loop; 2 new loops
      expensive communication: scalar communication...   ! 高コストな通信
            :
```

•u(:,:,0)とu(:,:,1)、u(:,:,ibar1)とu(:,:,ibar)とは、通常同一プロセッサ上にあるので、本当は通信不要（極端にプロセッサ数が多くない場合）

⇨ ON指示構文+LOCAL節の指定で通信の抑制が可能

例題コード2(8) 参考: コンパイラによる変形イメージ

■ 境界処理で頻出するパタン
  •各繰返しを全プロセッサが実行
    • 全プロセッサが計算対象のデータを必要とする
      ⇨ 各繰返し中で各代入文毎にブロードキャスト発生

```
do n=1,nll
   i=ll1(n)
   j=ll2(n)
  u(i,j,0) = -u(i,j,1)
      :
  u(i,j,ibar1)=-u(i,j,ibar)
      :
  k=j
  u(i,0,k) = -u(i,1,k)
      :
 enddo
```

⇨

HPFによる変形後ソース

```
do n=1,nll
   :
    broadcast. (u_tmp1 ← -u(i,j,1))
    if(u(i,j,0)を持っている) u(i,j,0) = u_tmp1
      :
    broadcast. (u_tmp2 ← -u(i,j,ibar))
    if(u(i,j,ibar1)を持っている)u(i,j,ibar1)=u_tmp2
      :
    broadcast. (u_tmp3 ← -u(i,1,k))
    if(u(i,0,k)を持っている) u(i,0,k) = u_tmp3
      :
enddo
```

例題コード2(9) ON指示構文の挿入

```
      do n=1,nll
         i=ll1(n)
         j=ll2(n)
!hpf$    on home(u(:,:,0)), local begin
         u(i,j,0) = -u(i,j,1)
            :
!hpf$    endon
!hpf$    on home(u(:,:,ibar1)),local begin
         u(i,j,ibar1)=-u(i,j,ibar)
            :
!hpf$    endon
         k=j
!hpf$    on home(u(:,:,k)), local begin
         u(i,0,k) = -u(i,1,k)
            :
!hpf$    endon
      enddo
```

- •ON指示構文により、実行プロセッサを明示
- •LOCAL節により通信が不要であることを明示
- •境界処理部分の効率が悪い場合は、ON指示構文+LOCAL節が有効な場合が多い

NEC

例題コード2(10) 並列化、通信状況のチェック3

- 再度−Mftn,−Minfo,−Mlist2オプション付で翻訳

```
%>sxhpf   -Mftn -Minfo  -Mlist2 sample2.F
```

- -Minfoオプションによる診断メッセージ

```
40,   Independent loop
        :
131,  Independent loop parallelized : co(:,:,k)
        :
241,  Independent loop parallelized : u(:,:,k)
257,  Array w not aligned with home array; array copied
      Reduction call .reduce_maxval emitted for variable dmax
      Independent loop parallelized : w(:,:,k)
        :
```

·並列化可能なループは全て並列化された。
·高コストな通信(expensive communication)も消滅
·257行目の配列コピーをチェック

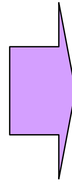
NEC

例題コード2(11) 参考：HPFによる変形イメージ

HPFによる並列化後

■ 257行目の配列コピーのメッセージに対応するループ

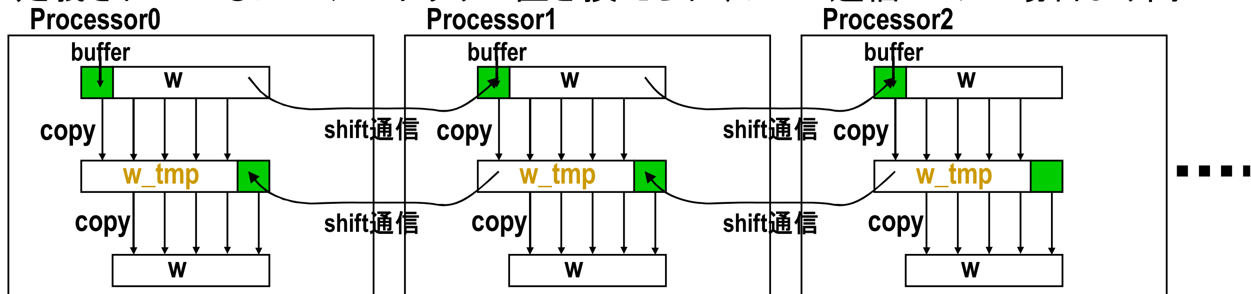
```
!hpf$  independent,....
       do k=1,ibar,2
       do n=1,n1
          i=li(n)
          j=lj(n)
          :
          w(i,j,k)=w(i,j,k) + ...
          w(i,j,k-1)=w(i,j,k-1) – ....
          :
        enddo
        enddo
```



```
comm_shift. (w)
w_tmp = w
do k=lbnd,ubnd,2
   :
   w(i,j,k)=w(i,j,k) + ....
   w_tmp(i,j,k-1)=w_tmp(i,j,k-1) –....
   :
enddo
comm_shift (w_tmp)
w = w_tmp
```

•配列wの分散されている次元の参照が1つずれており、通信は必要

•定義されているためテンポラリに置き換えられ、シフト通信のみの場合より高コスト



NEC

例題コード2(12) 並列化、通信状況のチェック4

■ 並列化情報リストファイル sample2.lst

```
%>grep "<S>" sample2.lst  ! 並列化できないと判定されたループを抽出
  (     67) <S>---------      do ij=2,ibar*2,2  ! 分散配列を含まない初期化ループ
  (     68) <S>---------        do i=1,min(ibar,ij-1) ! 分散配列を含まない初期化ループ
              :
  (   132) | <S>---------       do n=1,n2
```

·並列化可能なループは、全て並列化されている。

·132行目は、同一のループネストに並列化されたループがあるので問題なし。

```
(130)                              !hpf$ independent,new(k,n,i,j)
(131) <P>----------------------          do k=1,ibar
(132)  |  <S> ----------------             do n=1,n
```

```
%>grep "COMM:" sample2.lst  ! 通信情報を抽出
COMM: SFT [u] [LINO:  180 in sample2.F]   ! 配列uに対するシフト通信
        :
COMM: RED [dmax] [LINO:  257 in sample2.F]   ! スカラdmaxに対するリダクション
```

◆ シフトとリダクション以外の通信は消滅

NEC

例題コード2：性能値

num. of proc.	経過時間(s)	SpeedUp	GFLOPS	ベクトル化率(%)	ベクトル長	メモリ(GB)
(非並列)	841.0		3.95	99.93	255.7	16.0
1	855.3	0.98	3.89	99.93	255.0	18.0
2	444.3	1.89	7.49	99.93	254.8	17.7
4	236.6	3.55	14.06	99.93	254.7	18.1
8	136.8	6.15	24.41	99.92	254.0	19.1
16	71.6	11.75	47.04	99.91	252.9	20.9

(SX-8性能参考値)

・ソース規模　366行中
HPF指示文　13行(3.6%)

行数	指示文内訳
1	DISTRIBUTE指示文
6	INDEPENDENT指示文
6	ON指示構文

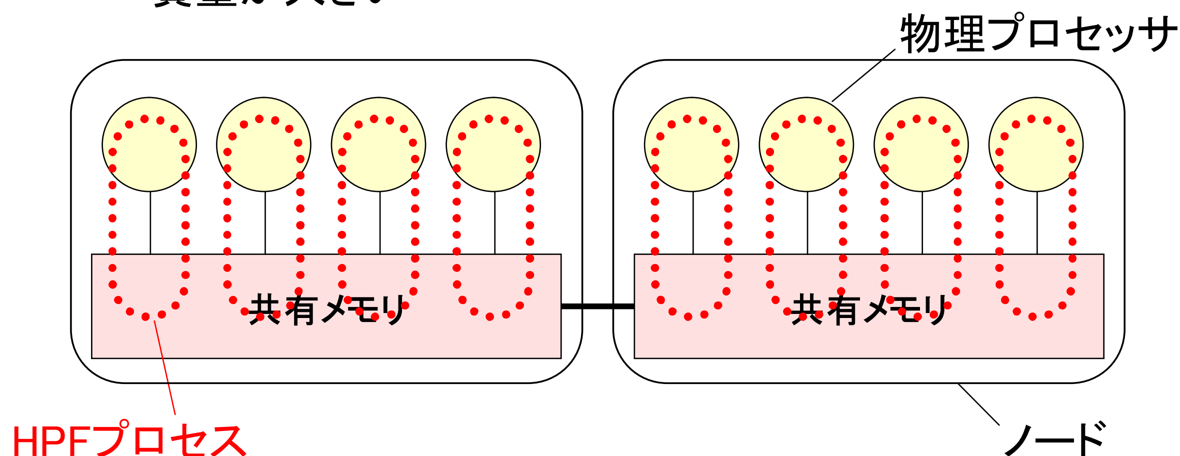
内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

SX上でのHPFプログラム実行(1) Flat並列化

（a） 1物理プロセッサ＝1 HPFプロセス(Flat並列化）

- ノード内並列・ノード間並列がシームレスなので簡単
- プロセスごとにメモリが別空間となるため、メモリの消費量が大きい

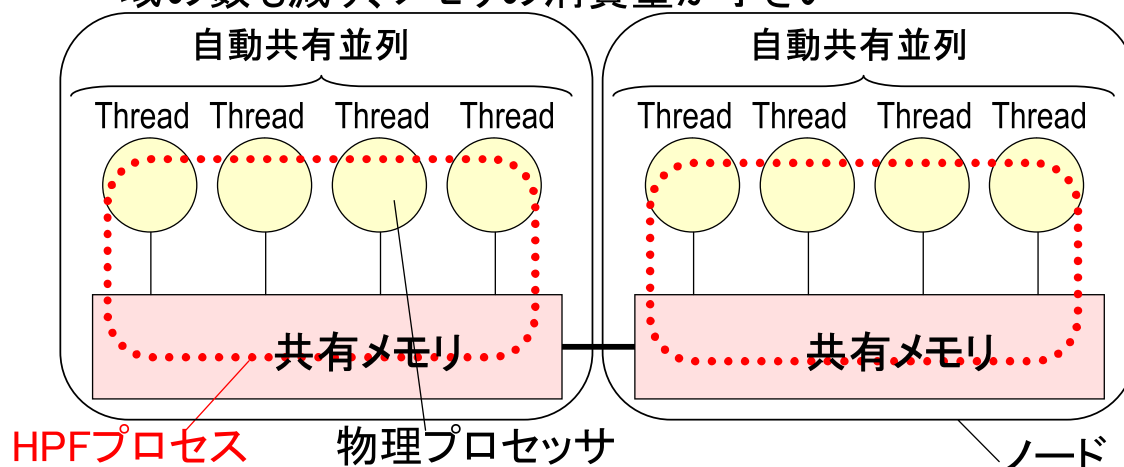


NEC

SX上でのHPFプログラム実行(2) Hybrid並列化

（b） 1ノード＝1 HPFプロセス(Hybrid並列化）

- ノード内は、FORTRAN90/SXの自動共有並列化(Multi Thread並列化)を行う
- プロセス数が減るため、プロセス毎に割付けられる領域の数も減り、メモリの消費量が小さい



NEC

Hybrid並列化の方法

(a) Flat並列化の場合

- ◆ コンパイル

```
%> sxhpf sample.hpf
```

- ◆ 実行(ジョブスクリプト)

```
#!/bin/sh
mpirun –np 8 ./a.out
```

(b) Hybrid並列化の場合

- ◆ コンパイル

```
%> sxhpf –P auto sample.hpf
```

•F90/SXのコンパイルメッセージや編集リストを参照し、必要に応じてF90/SXのコンパイラ指示行を追加する

- ◆ 実行(ジョブスクリプト)

```
#!/bin/sh
F_RSVTASK=8
export  F_RSVTASK
MPIEXPORT="F_RSVTASK"
export MPIEXPORT
mpirun –np 1 ./a.out
```

•環境変数**F_RSVTASK**で共有並列数を指定し、さらに**MPIEXPORT**に、**F_RSVTASK**を指定する
•HPFプロセス数(上記の場合1)と共有並列数(上記の場合8)の積が、CPU数を越えないように!!。
•**F_RSVTASK**を指定しない場合、ノード内の全CPUが共有並列に利用される。

NEC

例題コード1のHybrid並列化(1)

■ FORTRAN90/SXのメッセージで共有並列化状況をチェック

```
%> sxhpf  -P auto sample1.F

f90: vec(1) : sample1.F, line 54: ループ全体をベクトル化する。
            :
f90: mul(1) : sample1.F, line 128 : PARDOオプションによりDOループが並列化された
f90: vec(1) : sample1.F, line 129: ループ全体をベクトル化する。
f90: mul(10) : sample1.F, line 167: 自動並列化により、並列手続main_$1が生成された
f90: vec(1) : sample1.F, line 170: ループ全体をベクトル化する。
```

·128行目のループ(初期化部分)以外は共有並列化されていない

NEC

例題コード1のHybrid並列化(2)

- 共有並列化対象のループは、全て一重ループであり、ループ長が長いので、共有並列化とベクトル化の両方行うことが可能
- HPFの並列化によりループ長が変数になり、翻訳時に不明となるため、ベクトル化のみ行われ、共有並列化されない

Fortranコード

```
do n=1,np    ! ループ長である定数npは非常に大きい。
  vx(n) = vx(n) + dt*fx(n)/am(n)
```

HPFによる変形

```
do n=lbnd,ubnd    ! ループ長はnp/プロセッサ数 に変形される
  vx(n) = vx(n) + dt*fx(n)/am(n)
```

■FORTRAN90/SXのコンパイラ指示行!CDIR SELECT(CONCUR)を挿入し、共有並列化を優先することを指示

```
!CDIR  SELECT(CONCUR)
      do n=1,np
        vx(n) = vx(n) + dt*fx(n)/am(n)
```


NEC

例題コード1：Flat並列化 v.s. Hybrid並列化

16並列	経過時間(s)	ベクトル化率	ベクトル長	メモリ(GB)
Flat(16HPF)	52.5	99.72	255.3	2.954
Hybrid(2HPF*8共有)	52.1	99.72	255.5	0.438

（SX-8性能参考値）

·実行性能はほとんど同じ

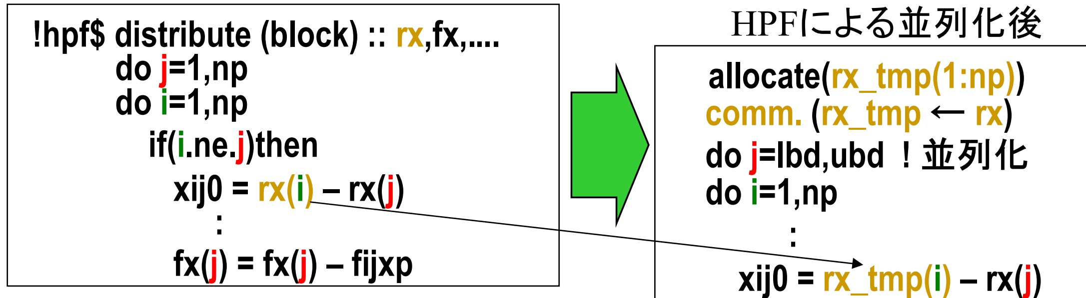
·メモリ量は、Hybrid並列化の方が圧倒的に有利

·プロセス毎に確保されるテンポラリ配列の数が減るため(次ページ参照)

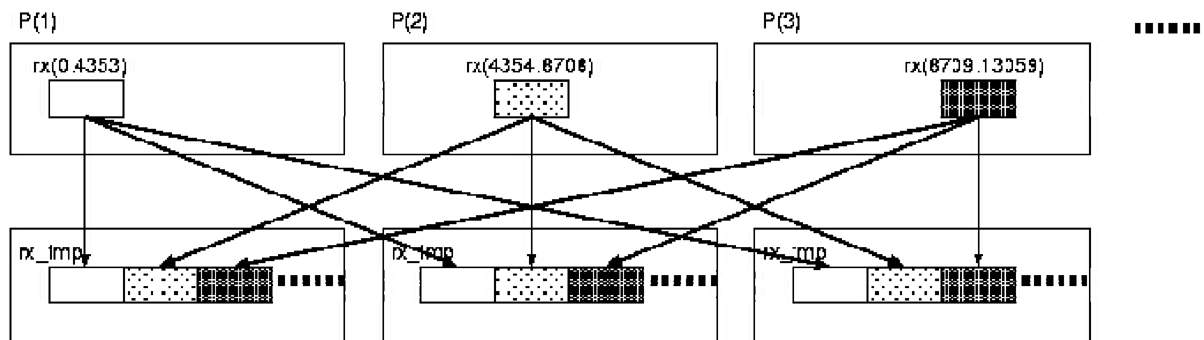
NEC

再掲：例題コード1の変形イメージ

■ 生成される通信とテンポラリ配列のイメージ



各プロセッサ上に、テンポラリが確保されるため、使用メモリ量は増大



NEC

例題コード2 : Flat並列化 v.s. Hybrid並列化

16並列	経過時間(s)	ベクトル化率	ベクトル長	メモリ(GB)
Flat(16HPF)	71.6	99.91	252.9	20.88
Hybrid(2HPF*8共有)	75.7	99.88	244.9	18.00

（SX-8性能参考値）

･−P autoオプションだけで全て共有並列化される
･実行性能はFlat並列化のほうが若干良好
･メモリ量は、Hybrid並列化の方が若干有利

NEC

Flat並列化とHybrid並列化の使い分け

- Hybrid並列化により性能向上が期待できる場合
  - 通信コストが高く、かつプロセス数の増加に従って、通信コストも増加する
    例)主要部に配列のリダクションを含む

```
!HPF$ DISTRIBUTE W(*,*,BLOCK)
         :
        DO J=1,N          ! 共有並列 :完全並列
           DO K=1,N    ! 分散並列 :リダクション: 共有並列化すると性能低下
              DO I=1,N  ! ベクトル化
                 A(I,J) = A(I,J) + W(I,J,K)
```

- Hybrid並列化によりメモリ削減が期待できる場合
  - 各プロセス上に複製されるサイズの大きな配列が(ユーザ配列又はテンポラリ配列として)出現する
    - 配列リダクションの場合、複製されたユーザ配列が出現
    - 例題コード1の場合、複製されたテンポラリ配列が出現
- それ以外の場合は、より容易なFlat並列化がお勧め

NEC

内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

デバッグ機能1 引数結合

例1) アドレス渡し

```
      real a(100,100)
!hpf$ distribute a(*,block)
      call sub(a(1,i))
        :
      subroutine sub(a)
      real a(100)
```

例2) 実引数と仮引数の形状が異なる

```
      real a(10000)
      call sub(a,10)
        :
      subroutine sub(a,n)
      real a(n,n)
!hpf$ distribute a(*,block)
```

•実行時にエラーが発生した手続や対象引数の情報を表示

例1

″仮引数名″: Nonsequential dummy array is associated with array element or scalar actual. PROG=″手続名″ ELN=″行番号″

例2

″仮引数名″: Dummy argument rank differs from actual. PROG=″手続名″ ELN=″行番号″

NEC

デバッグ機能2 共通ブロック

例1) マッピング指定漏れ

```
      common /com/ a(100,100)
!hpf$ distribute a(*,block)
      end
      subroutine sub
      common /com/a(100,100)
```

例2)使わない手続で宣言省略

```
      common /com/a(100),b(100)
!hpf$ distribute (block) :: a,b
      end
      subroutine sub
      common/com/ a(100)
!hpf$ distribute (block) ::a
```

·翻訳時オプション−Mcommonchkにより、実行時にエラーを検出し、手続名や配列名を表示
·本オプションは、全ての手続に対して指定する必要あり。

例1

Inconsistency detected in the mapping attribute of common variable between “手続名” and “手続名” : “配列名” in /共通ブロック名/

例2

Inconsistency detected in the number of explicitly mapped arrays of common block between “手続名” and “手続名” : “配列名” in /共通ブロック名/

NEC

デバッグ機能3　範囲外アクセス

■ 範囲外アクセス検出機能

```
real a(100,100)
!hpf$ distribute a(*,block)
do i=1,10000
  a(i,1) = ...
enddo
```

·分散次元より前の場合は可能

```
real a(100,100,100)
!hpf$ distribute a(*,*,block)
do k=1,100
do i=1,10000
  a(i,1,k) = ...
enddo
enddo
```

- ◆ 翻訳時オプション-Msubchkを指定すると実行時に範囲外アクセスの有無をチェック
- ◆ 並列化、ベクトル化の阻害を可能な限り抑制
- ◆ 範囲外アクセス検出時、配列名、行番号等の情報を警告又は致命的エラーメッセージとして出力(既定値は警告)。実行時オプション-hpf –subchk fatal　を指定すると、範囲外アクセス検出時に実行終了させることもできる

```
mpirun –np 8 ./a.out –hpf –subchk fatal
```

```
"配列名" is accessed out of declared bounds along %d th dim
PROG="手続名" ELN="行番号"
```


NEC

デバッグ機能4 INDEPENDENT指示文誤指定

例1) 依存のあるループへのINDEPENDENT指示文の指定

```
!hpf$ independent
      do i=1,n
        L= L+1  ! Lは前の繰返しの値を使用しているのでdoループは並列化できない。
        a(i) = L
      enddo
```

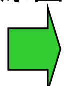
·Lを消去し、ループ本体を"a(i) = Lの初期値 + i" と書き直せば並列化可能

例2) REDUCTION節の指定もれ

```
!hpf$ independent
      do i=1,n
       do j=1,n
        a(j)=a(j)+b(j,i)  ! 配列aはリダクションの依存があるのでREDUCTION節の指定が必要
       enddo
      enddo
```

·INDEPENDENT指示文を消すか、INDEPENDENT, REDUCTION(a)に修正する

- ■ 翻訳時オプション**-Mnoindependent**によりINDEPENDENT指示文は無視される。
- ■ 本オプションにより結果が正常になった場合は、INDEPENDENT指示文の誤指定が原因の可能性が大きい。


翻訳時メッセージや並列化情報リストで、並列化不可と判定されたループやリダクションループをチェックすることで、誤ったINDEPENDENT指示文を突き止められる。

NEC

デバッグ機能5 その他

- デバッグ行

FORTRAN90/SXと同様に、!!(自由形式)、*(固定形式)で始まる行をデバッグ行と認識し、オプションにより注釈行として扱うか、有効な文として扱うかを切り替えられる。

既定値では注釈行、翻訳時オプション-Mdlinesで有効な文

```
*      write(*,*)A
```

- 手続境界での通信発生

例) サブルーチン側の通信指定漏れによりブロードキャスト発生

```
      real a(100)
!hpf$ distribute a(block)
      call sub(a)
          :
      subroutine sub(a)
      real a(100)
```

·実行時オプション−hpf −commmsg を指定すると、手続境界で通信が発生した場合、対象配列と手続名を表示

```
mpirun –host 0 –np 8 ./a.out –hpf -commmsg
```

"仮引数名": Communication occurs at procedure boundary PROG="手続名" ELN="行番号"

NEC

内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

実行時性能情報の取得

主に3つの手段

- MPIPROGINF ：プログラム全体の情報
- ftrace ：手続毎の情報
- 計時手続 ：任意の区間の実行時間(経過時間)

NEC

MPIPROGINF(1)

- プログラム全体の特性情報を出力
（実行時間、ベクトル化率、メモリ使用量等）
- 翻訳・リンク時に特別なオプション指定不要
- ハードウェアの情報を利用するのでオーバヘッドがない
- 実行時に環境変数 MPIPROGINFを指定

sh形式： MPIPROGINF= {YES|DETAIL|ALL|ALL_DETAIL}
export MPIPROGINF

csh形式：setenv MPIPROGINF {YES|DETAIL|ALL|ALL_DETAIL}

YES ：基本情報（集約形式:最大・最小・平均値）
DETAIL ：詳細情報（集約形式:最大・最小・平均値）
ALL ：基本情報（拡張形式:全プロセスの情報）
ALL_DETAIL：詳細情報（拡張形式:全プロセスの情報）

NEC

MPIPROGINF(2)　表示例

◆MPIPPROGINF=DETAIL（詳細情報（集約形式））

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.

Global Data of 4 processes  :        Min [Rank]        Max [Rank]    Average
==========================
```

Item		Min	[Rank]	Max	[Rank]	Average
Real Time (sec)	:	348.203	[3]	348.220	[0]	348.212
User Time (sec)	:	347.763	[1]	348.126	[2]	348.027
System Time (sec)	:	0.053	[3]	0.204	[1]	0.097
Vector Time (sec)	:	36.361	[3]	42.796	[0]	39.107
Instruction Count	:	24394030648	[0]	25395808156	[2]	24952758629
Vector Instruction Count	:	576518089	[3]	604610478	[0]	588600877
Vector Element Count	:	44875213685	[3]	44998595135	[0]	44936956672
FLOP Count	:	22349659842	[3]	22350882341	[2]	22350354587
MOPS	:	197.613	[0]	200.309	[2]	199.126
MFLOPS	:	64.200	[3]	64.270	[1]	64.220
Average Vector Length	:	74.426	[0]	77.838	[3]	76.375
Vector Operation Ratio (%)	:	64.410	[2]	65.416	[0]	64.846
Memory size used (MB)	:	81.862	[0]	81.862	[0]	81.862
MIPS	:	70.079	[0]	72.950	[2]	71.698
Instruction Cache miss (sec)	:	0.413	[0]	0.438	[1]	0.419
Operand Cache miss (sec)	:	86.153	[0]	86.207	[3]	86.179
Bank Conflict Time (sec)	:	1.595	[1]	1.600	[3]	1.597

NEC

ftrace(1)　Flat並列化の場合

- 手続毎に、実行回数，実行時間，ベクトル演算率等の情報をプロセス別に表示
- 手続別にロードバランスを確認可能
- 手続別に通信情報も採取表示可能

各プロセス毎に、負荷の高い手続順に並べて表示可能

```
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFL        OP   AVER.  I-CACHE
                      TIME[sec]( % )   [msec]                  RATIO  V.LEN  MISS      ...

buts          248000   141.163( 23.5)   0.569       .1  110.2  20.22  15.0   1.0273
blts          248000   136.544( 22.7)   0.5    219.5  113.8  28.85  19.1   0.1232
        ....
        <略>
PROG.UNIT  FREQUENCY  EXCLUSIVE       AVER.TIME   MOPS MFLOPS V.OP   AVER.  I-CACHE
                      TIME[sec]( % )   [msec]                  RATIO  V.LEN  MISS
buts          248000   141.163( 23.5)   0.569  207.1  110.2  20.22  15.0   1.0273
blts          248000   136.544( 22.7)   0.551  219.5  113.8  28.85  19.1   0.1232
        ....
```


NEC

ftrace(2) Hybrid並列化の場合

- 自動共有並列化されたループ単位に、実行回数，実行時間，ベクトル演算率等の情報をスレッド別と合計で表示

共有並列化されたループは、
”ループを含む手続名$番号”という
名前の1つの手続として表示される

PROG.UNIT	FREQUENCY	EXCLUSIVE TIME[sec](%)	AVER.TIME [msec]	MOPS	MFLOPS	V.OP RATIO	AVER. V.LEN	I-CACHE MISS	...
buts$2	248000	141.163(23.5)	0.569	207.1	110.2	20.22	15.0	1.0273	
-micro1	15500	8.823(1.5)	0.569	207.2	110.3	20.22	15.0	0.0163	
-micro2	15500	9.402(1.6)	0.607	207.0	110.4	20.26	16.0	0.0946	
-micro3	15500	9.416(1.6)	0.608	206.7	110.2	20.26	16.0	0.1089	
-micro4	15500	8.962(1.5)	[illegible]	204.0	108.6	20.22	15.0	0.0997	
....									
blts$3	248000	136.544(22.7)	[illegible]	[illegible]	113.8	28.85	19.1	0.1232	
-micro1	15500	8.540(1.4)	0.551	[illegible]	113.9	28.85	19.1	0.0037	
-micro2	15500	8.985(1.5)	0.580	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
-micro3	15500	9.002(1.5)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	
-micro4	15500	8.674(1.4)	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	

〈略〉

スレッド別に測定結果を並べて表示。

NEC

ftrace(3)　利用方法

- 翻訳時
  - 翻訳時オプション –ftraceを指定

例：
```
%>sxhpf –ftrace test.hpf
```

- 表示
  - 実行後、出力されたファイルftrace.out.*に対して、sxftraceコマンドを(クロスコンパイル環境にて)実行

```
%>sxftrace –f ftrace.out.*
```

or

  - 実行時に、環境変数 F_FTRACEを設定

NEC

計時関数

- FORTRAN90/SXの拡張組込み手続etime、又はHPF専用計時サブルーチンHPF_WCLOCKが利用可能

```
          double precision t1,t2

        call etime(t1)
!HPF$ independent
        do i=1, N
           ...
        end do
        call etime(t2)
        write(*,*) t2-t1
```

- 引数の型はDOUBLE PRECISION
- 任意のポイントで情報を得られる。
- 並列ループ内には書けない。
- HPF_WCLOCKの使用法も同様(但し、HPF_WCLOCKの場合、全プロセッサの戻り値を一致させるため、通信が発生する。)

NEC

内容

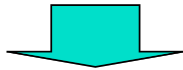
- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

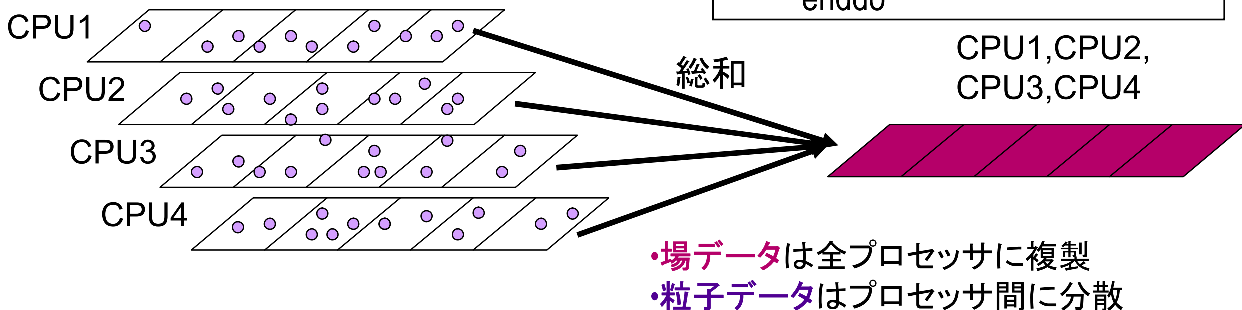
粒子コードのベクトル化と分散ブロック幅(1)

•粒子コードで頻出する、以下のような(1次元の例)総和計算のループは、間接アクセスのため依存が不明であり、そのままではベクトル化・並列化不可

```
!hpf$ processors p(npe)
!hpf$ distribute x(block) onto p
      double precision x(n),rho(m)
      rho = 0.0
      do i=1, n
         ix = x(i)
         rho(ix)= rho(ix) + f(ix)
         rho(ix+1)=rho(ix+1)+g(ix+1)
      enddo
```



ベクトル長毎にループを区切った、2重ループに変形し、内側ループでベクトル化する手法がよく用いられる

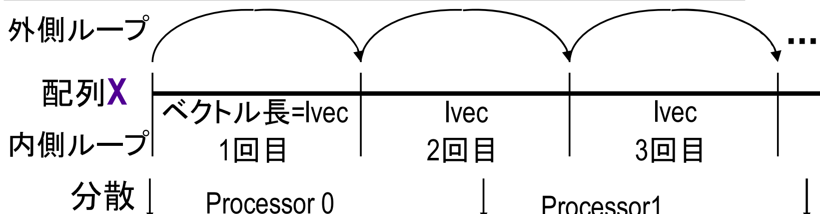


•場データは全プロセッサに複製
•粒子データはプロセッサ間に分散

NEC

粒子コードのベクトル化と分散ブロック幅(2)

```
!hpf$ processors p(npe)
!hpf$ distribute x(block) onto p
      double precision x(n),rho(m),w(lvec+1,m)
      w = 0.0 ; rho = 0.0
      do ii=1, n, lvec
        do ivec = 1, min(lvec,n-(ii-1)*lvec) ! ベクトル化
           i = ii + ivec -1
           ix = x(i)
           w(ivec,ix)= w(ivec,ix) + f(ix)
           w(ivec,ix+1)=w(ivec,ix+1)+g(ix+1)
        enddo
      enddo
      do ivec =1,lvec
       do i=1,m
         rho(i) = rho(i) + w(ivec,i)
       enddo
      enddo
```



•ループをベクトル長毎に切って2重化し、内側ループでベクトル化する。

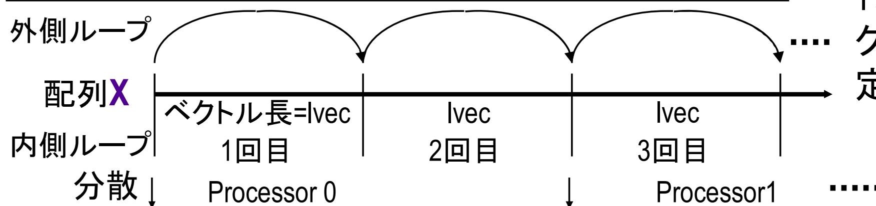
•そのため、場のデータrhoを、次元拡張した作業配列wに置き換える。

•但し、このままでは粒子データxの分散ブロック幅がベクトル長の倍数でないため、外側ループで並列化する際に、リモートアクセスが発生する(左図の例では2回目の内側ループの途中でProcessor0とProcessor1の境目が来る)

NEC

粒子コードのベクトル化と分散ブロック幅(3)

```
!hpf$ processors p(npe)
!hpf$ distribute x(block(((n-1)/(lvec*npe)+1)*lvec)) onto p
      double precision x(n),rho(m),w(lvec+1,m)
      w = 0.0 ; rho = 0.0
      do ii=1, n, lvec
        do ivec = 1, min(lvec,n-(ii-1)*lvec) ! ベクトル化
          i = ii + ivec -1
          ix = x(i)
          w(ivec,ix)= w(ivec,ix) + f(ix)
          w(ivec,ix+1)=w(ivec,ix+1)+g(ix+1)
        enddo
      enddo
      do ivec =1,lvec
       do i=1,m
         rho(i) = rho(i) + w(i,ivec)
       enddo
      enddo
```



•そこで、分散ブロック幅がベクトル長lvecの倍数になるように、DISTRIBUTE指示文中で分散ブロック幅を指定する。(下図の例では、分散ブロック幅がlvecの2倍)

•但し、このままでは、外側ループのDO変数iiと分散配列xの添字iが対応していないので、自動的には並列化できず、また内側ループ実行時にxに対するリモートアクセスがないことも自動判定できない。

NEC

粒子コードのベクトル化と分散ブロック幅(4)

```
!hpf$ processors p(npe)
!hpf$ distribute x(block(((n-1)/(lvec*npe)+1)*lvec)) onto p
      double precision x(n),rho(m),w(lvec+1,m)
      w = 0.0 ; rho = 0.0
!hpf$ independent, new(i,ix,ivec),reduction(w)
      do ii=1, n, lvec
!hpf$    on home(x(ii)), local(x)
        do ivec = 1, min(lvec,n-(ii-1)*lvec) ! ベクトル化
          i = ii + ivec -1
          ix = x(i)
          w(ivec,ix)= w(ivec,ix) + f(ix)
          w(ivec,ix+1)=w(ivec,ix+1)+g(ix+1)
        enddo
      enddo
      do ivec =1,lvec
       do i=1,m
         rho(i) = rho(i) + w(i,ivec)
       enddo
      enddo
```

•そこで、外側ループにREDUCTION節付きのINDEPENDENT指示文を指定して、作業配列wに対する集計演算を含む、並列化可能なループであることを明示する。

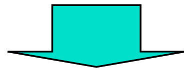
•さらに、外側ループの各繰返しを粒子データx(ii)がマップされているプロセッサで実行すれば、xに対する通信は不要であることをON指示構文+LOCAL節で明示する。

NEC

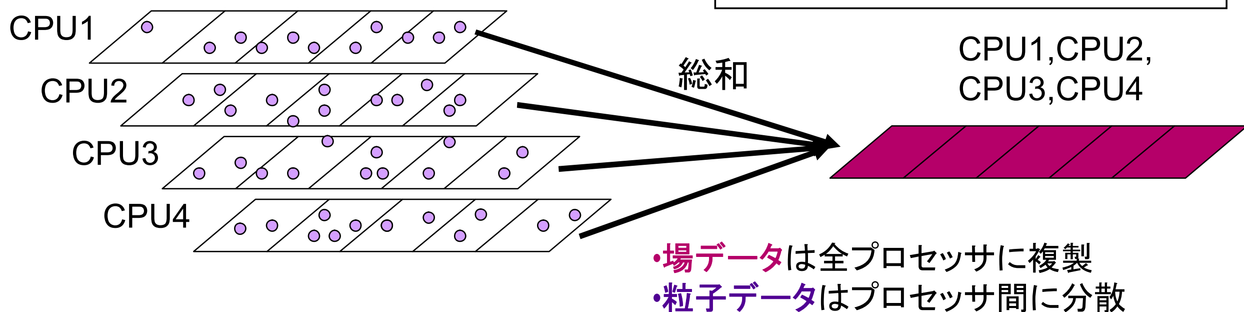
粒子コードのLISTVEC指示行によるベクトル化(1)

•粒子コードで頻出する、以下のような(1次元の例)総和計算のループは、間接アクセスのため依存が不明であり、そのままではベクトル化・並列化不可

```
!hpf$ processors p(npe)
!hpf$ distribute x(block) onto p
      double precision x(n),rho(m)
      rho = 0.0
      do i=1, n
         ix = x(i)
         rho(ix)= rho(ix) + f(ix)
         rho(ix+1)=rho(ix+1)+g(ix+1)
      enddo
```



•F90/SXコンパイラ指示行**LISTVEC**を指定すればベクトル化可能



•**場データ**は全プロセッサに複製
•**粒子データ**はプロセッサ間に分散

NEC

粒子コードのLISTVEC指示行によるベクトル化(2)

```
!hpf$ processors p(npe)
!hpf$ distribute x(block) onto p
      double precision x(n),rho(m)
      rho = 0.0
      do ii=0,1
!cdir listvec
      do i=1, n
         ix = x(i)
         ix = ix + ii
         rho(ix)= rho(ix) + (1-ii)*f(ix)+ii*g(ix)
      enddo
      enddo
```

•**LISTVEC**指示行を指定すると、ベクトル化可能だが、間接アクセスによる集計演算が行われる配列**rho**は、1つの行にしか記述できない。

•そこで、外側にループ長2のループを付加して、**rho**の2回の出現をループに変換する。

•ループ中で参照されている変数は、外側ループのDO変数**ii**の値を使ってうまく修正する。

•ただし、このままではループが並列化可能であることを自動判定できない。

NEC

粒子コードのLISTVEC指示行によるベクトル化(3)

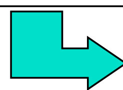
```
!hpf$ processors p(npe)
!hpf$ distribute x(block) onto p
      double precision x(n),rho(m)
      rho = 0.0
!hpf$ independent,new(i), reduction(rho)
      do ii=0,1
!hpf$ independent,new(ix)
!cdir  listvec
      do i=1, n
         ix = x(i)
         ix = ix + ii
         rho(ix)= rho(ix) + (1-ii)*f(ix)+ii*g(ix)
      enddo
      enddo
```

•そこで、外側ループにREDUCTION節付きのINDEPENDENT指示文を、内側ループにINDEPENDENT指示文を指定し、並列化可能であることを明示する。(2つのループは、共に配列rhoに対する集計演算を行うが、このように複数のループが集計演算を行う場合、REDUCTION節は一番外側のループだけに指定する。)

NEC

2つのベクトル化方法の比較

	演算効率	演算量	メモリ量
ループ2重化と次元拡張	○		
LISTVEC指示行		○	○



メモリ量と性能のバランスを考慮して、どちらの方法にするか決める。

•LISTVEC指示行の場合、1回のベクトル演算(ベクトル長の範囲)で、場のデータに関する依存があると、その部分はスカラ演算になり効率が低下するので、必要ならば(ベクトル化率が低い場合)、あらかじめ粒子データの並べ替えを行って、できるだけ依存を減らすことが必要。

NEC

内容

- HPF/SX V2概要
- HPF/SX V2プログラミング
- FortranプログラムのHPF化例
  - 例題コード1のHPF化
  - 例題コード2のHPF化
- Hybrid並列化の利用
- デバッグ機能
- 実行時性能情報の取得
- 粒子コード主要部のベクトル化例
- 付録

NEC

内容

- 不規則問題のための拡張機能

NEC

不規則問題のための拡張機能一覧

- 有限要素法など、間接アクセスを含むプログラム向けの機能

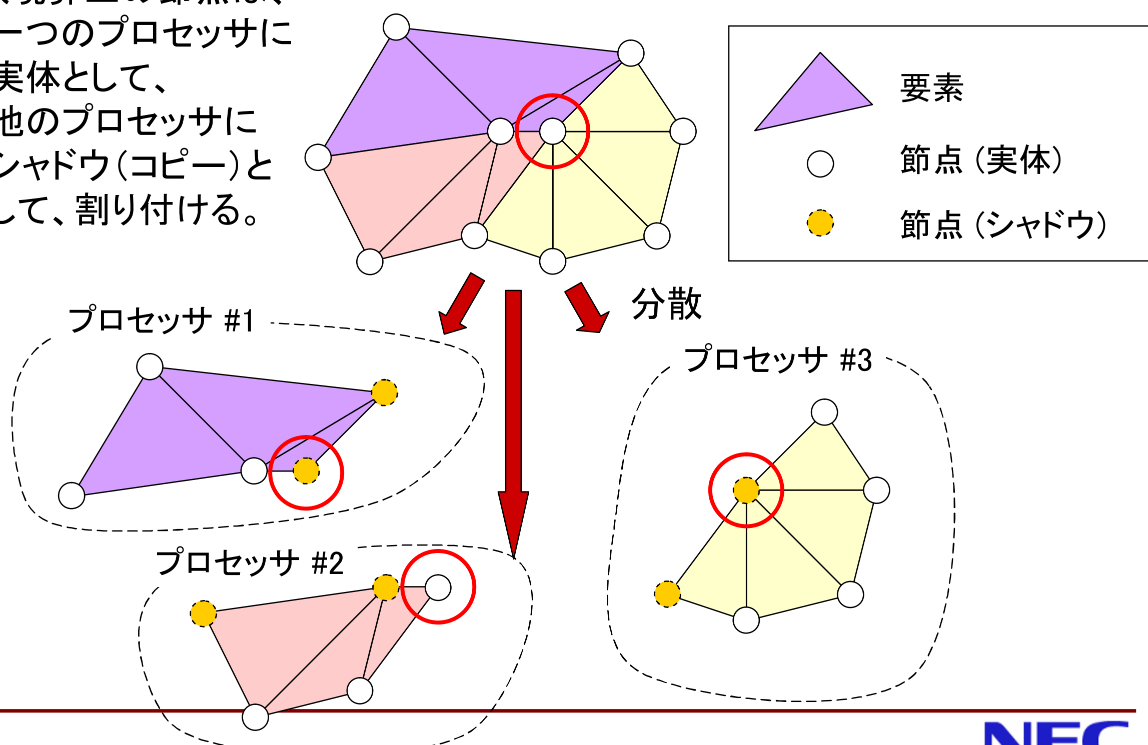
```
DO i=1,N
  .... A(IDX(i)) .....
```

機能	規則問題	不規則問題
分散形式	BLOCK (負荷均等) GEN_BLOCK(負荷不均等)	GEN_BLOCK (性能) INDIRECT(記述性)
シャドウ領域宣言	SHADOW指示文	不規則SHADOW指示文
通信制御	REFLECT指示文	REFLECT指示文
計算マッピング指定	ON指示文	ON指示文
通信不要を明示	LOCAL節	LOCAL節
集計演算明示	REDUCTION節	ローカルREDUCTION節
アクセス情報再利用		INDEX_REUSE指示文

NEC

不規則シャドウ(HALO) 概要

分散境界上の節点は、
- 一つのプロセッサに実体として、
- 他のプロセッサにシャドウ(コピー)として、割り付ける。



NEC

不規則シャドウ(HALO） 利用法概説

- 宣言 : SHADOW指示文
- 値の設定 : REFLECT指示文
- 値の参照 : ON指示構文+LOCAL節
- 参照オーバヘッド削減 : INDEX_REUSE指示文
- 集計演算 : ローカルREDUCTION節

•参考: 通常のシャドウの利用法

- 宣言 : SHADOW指示文
- 値の設定 : REFLECT指示文
- 値の参照 : ON指示構文+LOCAL節

NEC

不規則シャドウの宣言

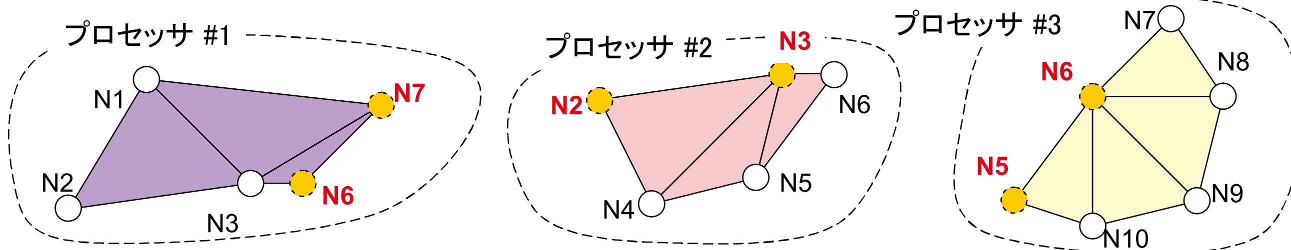
■SHADOW指示文に構造型配列を指定する。

```
module global
type shadow
  integer,pointer::index(:)
end type
end module
```

※ あらかじめ、メッシュの分割と順番の並べ替えを済ませておく必要がある。

```
use global
type(shadow) halo(3)
do i=1,3
    allocate(halo(i)%index(2))
enddo
halo(1)%index = (/6,7/)
halo(2)%index = (/2,3/)
halo(3)%index = (/5,6/)
call main(halo)
```

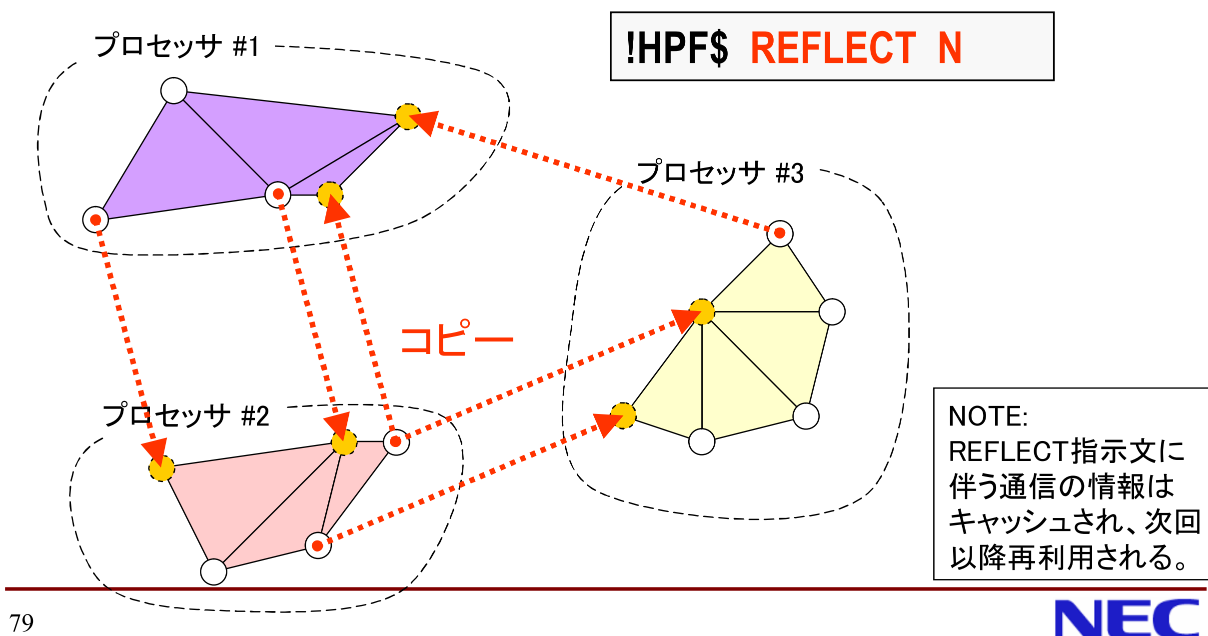
```
subroutine main(halo)
 use global
 type(shadow) halo(3)
 real n(10)
 integer :: m(3) =(/3,3,4/)
!hpf$ distribute n(gen_block(m))
!hpf$ shadow n(halo)
```



NEC

不規則シャドウへの値の設定

■ REFLECT指示文により、実体の値をシャドウへコピー



NEC

INDEX_REUSE指示文

- 不規則シャドウ領域は、参照オーバヘッドが高い。
- 不規則シャドウを持つ配列へアクセスするループが繰返し実行される場合、配列要素へのアクセス順序が毎回同一なら、INDEX_REUSE指示文を指定すると、最初のアクセス情報がキャッシュされ、次回の実行時に再利用される。

形式 ([ ]内は省略可能)
INDEX_REUSE [(論理式)] *v1,v2,...*

- INDEX_REUSE指示文を指定すると;

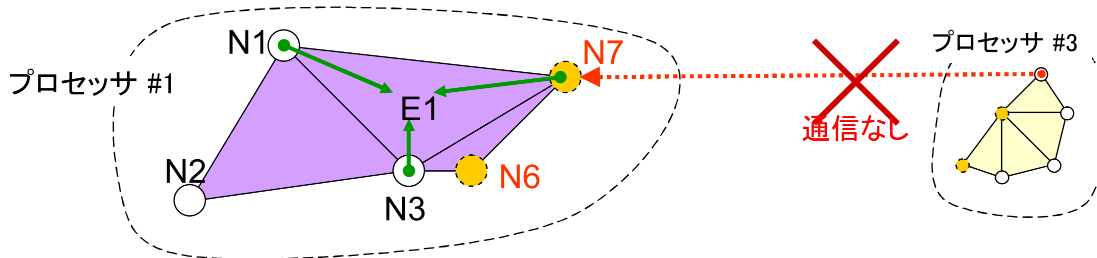
不規則シャドウを持つ配列*v1,v2,...*がON指示構文+LOCAL節の指定下で参照されるとき、論理式が.TRUE.であれば、前回のアクセス情報を再利用する。(省略時は.TRUE.と仮定)

NEC

不規則シャドウの参照

·ON指示構文+LOCAL節により、LOCAL節中に指定された配列は、他のプロセッサ上の実体ではなく、自プロセッサ上の(不規則)シャドウ実体を参照
·INDEX_REUSE指示文により参照オーバヘッド削減

```
!HPF$ INDEX_REUSE (.true.) N
!HPF$ INDEPENDENT
       DO i=1, ENUM
!HPF$    ON HOME(E(i)), LOCAL(N) BEGIN
         DO j=1, 3
            E(i) = E(i) + N(idx(i,j)) + 1
         END DO
!HPF$    END ON
       ND DO
```

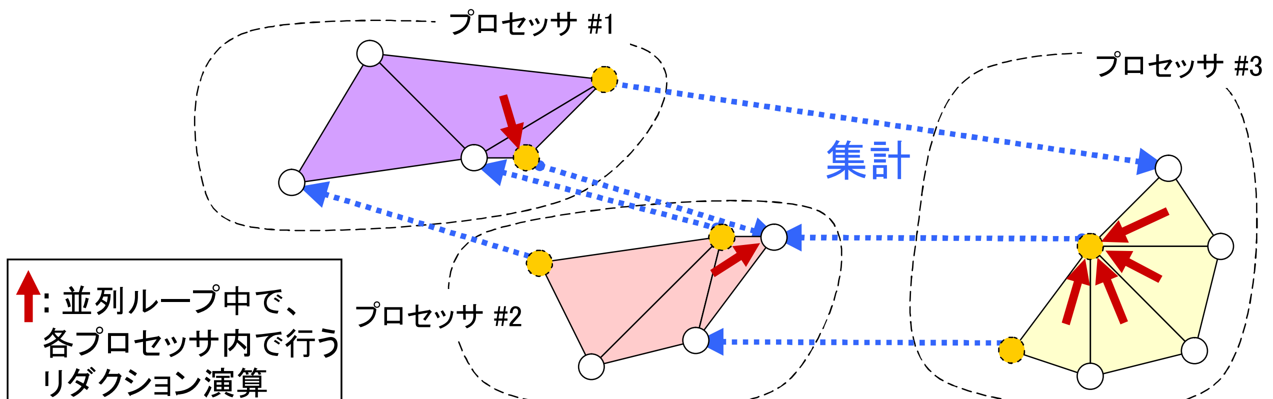


NEC

不規則シャドウを利用したリダクション

■ “LOCAL:”を指定したREDUCTION節により、並列ループ実行中、他のプロセッサ上の実体に対するリダクション演算は、自プロセッサ上の不規則シャドウ領域に対して行い、並列ループ実行後に、不規則シャドウ領域から実体へ集計する。

```
!HPF$ INDEX_REUSE (.TRUE.) N
!HPF$ INDEPENDENT, NEW(I), REDUCTION(LOCAL: N)
       DO I=1,ENUM
!HPF$    ON HOME(ELM(I)) BEGIN
         DO J=1,3
           N(IDX(I,J))=N(IDX(I,J))+ELM(I)....
         ENDDO
!HPF$    ENDON
       ENDDO
```



NEC