HPFプログラミングの実際

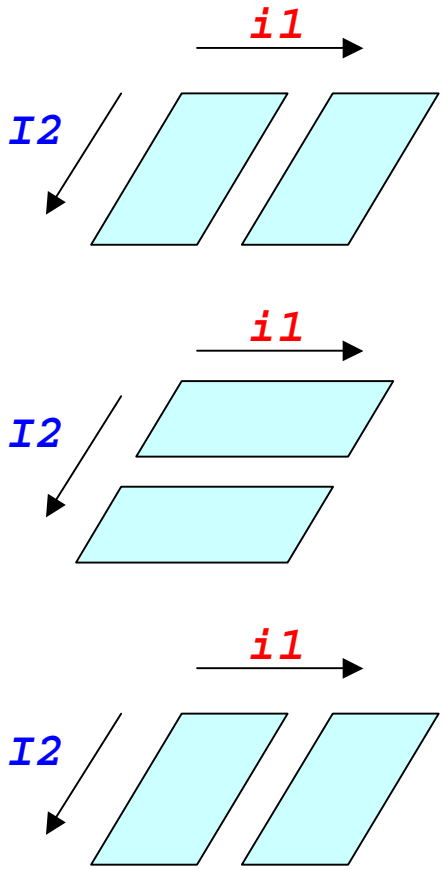
NEC　末広 謙二

2003年 6月 11日

MAXWELL

- Maxwell ソルバー(2次元 電磁波解析、FFT法)
- プログラムの概要（主要部）

```
subroutine ms_solver
  do i1 = ...
    jx, jy, rho   電流･電化密度
       ↓ FFT
    jxf, jyf, rohf   (波数)
  end do
  do i2 = ...
    jxf, jyf
       ↓
    rr, vv ← blnf, brnf   磁場(波数)
       ↓
    exf, eypf, eymf   電場(波数)
  end do
  do i1 = ...
    exf, eypf, eymf
       ↓ 逆FFT
    ex, eyp, eym   電場
  end do
```



MAXWELL

- 配列の分割を決める

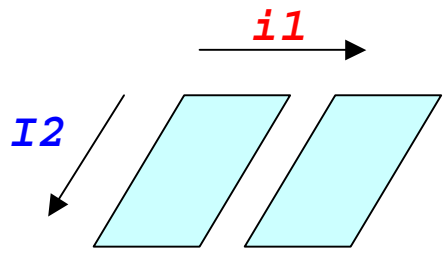
```
subroutine ms_solver
  do i1 = ...
    jx, jy, rho
      ↓
    jxf, jyf, rohf
  end do
  do i2 = ...
    jxf, jyf
      ↓
    rr, vv ← blnf,
      ↓
    exf, eypf, eymf
  end do
  do i1 = ...
    exf, eypf, eymf
      ↓
    ex, eyp, eym
  end do
```

② 右辺の配列をループの分割に合わせる
jx,jy,rohの形状は(*i1*,*i2*)なので

```
!HPF$ distribute(block,*)::
!HPF$+                jx, jv, roh
```

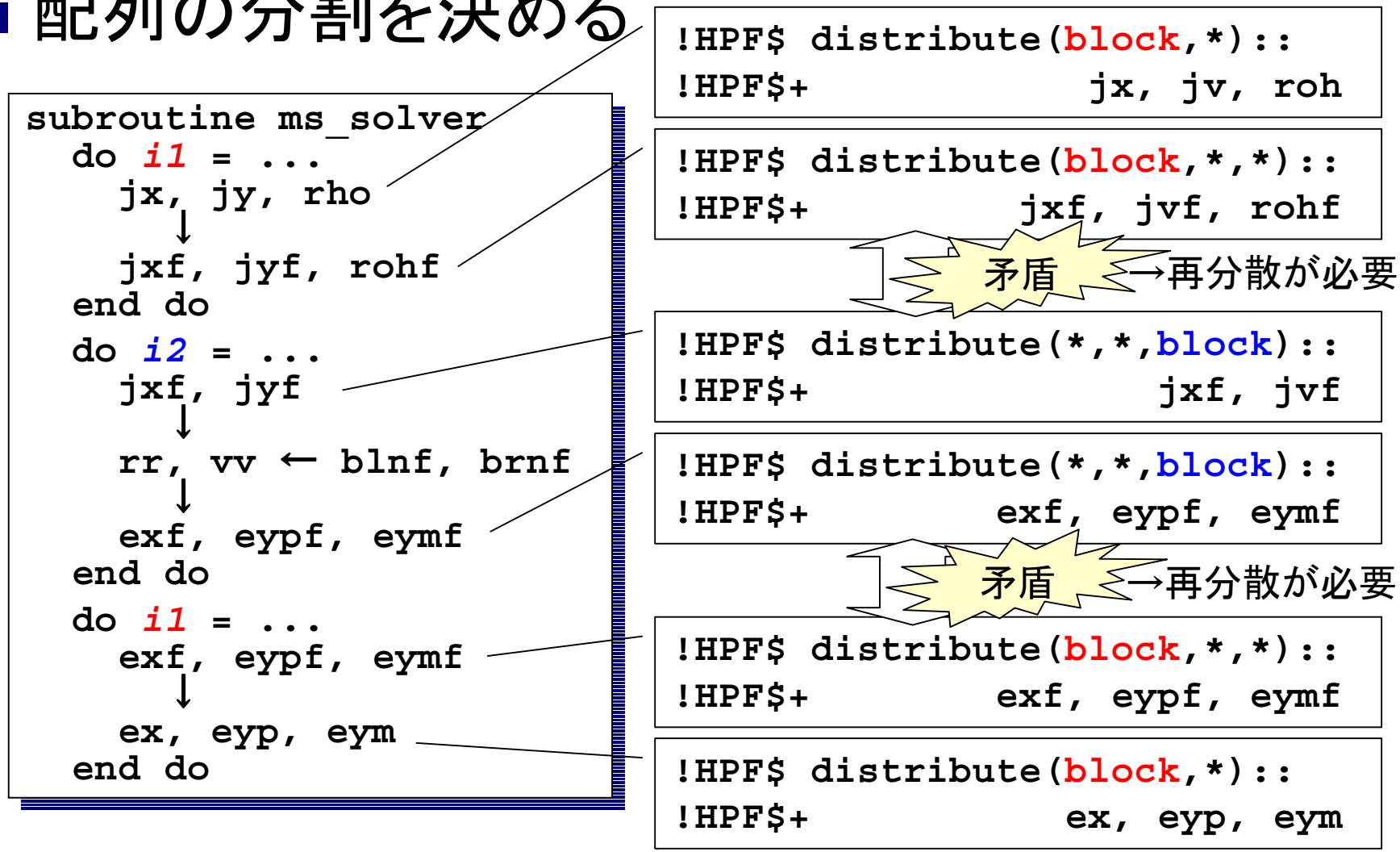
① ターゲットとなる配列をループの分割に合わせる
jxf,jyf,rohf の形状は(*i1*,2,*i2*)なので

```
!HPF$ distribute(block,*,*)::
!HPF$+                jxf, jvf, rohf
```



MAXWELL

- 配列の分割を決める



MAXWELL

- 再分散指示の挿入

```
subroutine ms_solver
  do i1 = ...
    jx, jy, rho
       ↓
    jxf, jyf, rohf
  end do
  do i2 = ...
    jxf, jyf
       ↓
    rr, vv ← blnf, brnf
       ↓
    exf, eypf, eymf
  end do
  do i1 = ...
    exf, eypf, eymf
       ↓
    ex, eyp, eym
  end do
```

```
!HPF$ dynamic:: jxf, jvf
!HPF$ distribute(block,*,*)::
!HPF$+              jxf, jvf, rohf
```

```
!HPF$ redistribute(*,*,block)::
!HPF$+                    jxf, jvf
```

```
!HPF$ dynamic:: exf,expf,eymf
!HPF$ distribute(*,*,block)::
!HPF$+              exf, eypf, eymf
```

```
!HPF$ redistribute(block,*,*)::
!HPF$+              exf, eypf, eymf
```

redistribute する変数には
dynamic の指定も忘れずに。

MAXWELL

- independent 指示文の挿入
  - 1回の繰り返しの中に「閉じて」使われる変数
    → new 節
  - リダクション演算（総和など）の結果を持つ変数
    → reduction 節

```
!HPF$ independent, new(i2,field1), reduction(sum1)
      do i1 = ...
        do i2 = ...
          field1(i2,1) = ...
          ...
          sum1 = sum1 + ...
        end do
      end do
```

i1 の値が変わるごとに
毎回新しく定義される
→ new変数

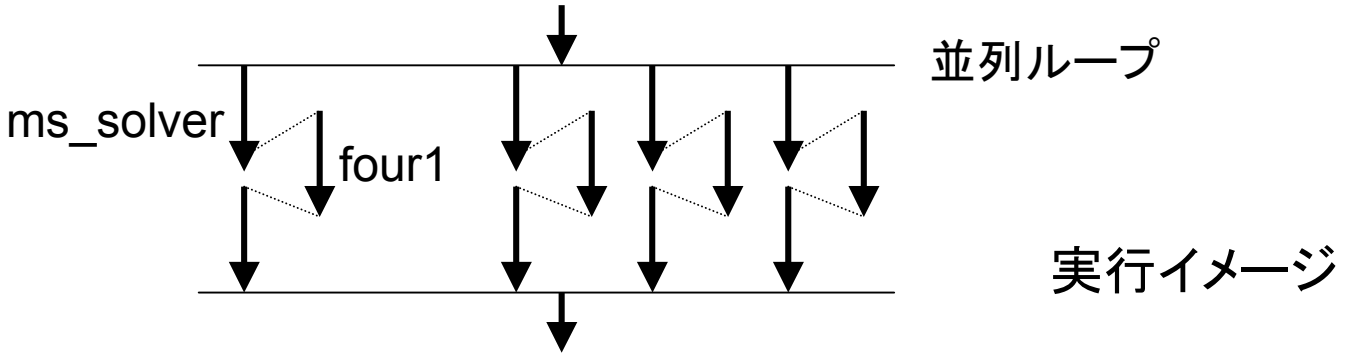
総和演算
→ reduction変数

MAXWELL

- 並列ループ内からのサブルーチン呼び出し
  - 1次元FFTルーチンを並列ループから呼びたい

```
!HPF$ independent, new(field1)
      do i1 = ...
        field1 = jx(i1,...)
        call four1(field1,...)
        jxf(i1,...) = field1
      end do
```

- 呼び出し間に依存・副作用がないことが条件



MAXWELL

- 並列ループ内からの手続き呼び出し（つづき）
  - 呼び出すルーチンの interface block を記述

```
interface
  extrinsic (fortran_local) pure
&          subroutine four1(field,...)
  real,dimension(...),intent(inout):: field
  ...
  end subroutine
end interface
```

    - extrinsic(fortran_local) :
      Localモデル（各並列プロセッサごとに閉じた計算を行う）
    - pure : 副作用がない（SAVE変数への書き込み、I/O操作等）
  - 呼び出されるルーチンにも同様の宣言を記述

ただし、こう書けば望みどおり並列化されるかどうかは、処理系に依存します。（NEC製コンパイラは並列化する）

MAXWELL

- 性能測定結果（NEC府中 SX-7 シングルノード）

台数	1	4	8
F90版（秒）	6.26		
HPF版（秒） （F90比）	6.42 （0.98）	1.81 （3.46）	1.20 （5.22）

MAXWELL

- もっと速くならないか？
  - dynamic な配列は効率が悪い...?
- コンパイルメッセージ（-Minfo）を見てみると

```
85, Array jxf not aligned with home array; array copied
    Array jyf not aligned with home array; array copied
    Array exf not aligned with home array; array copied
    Array eypf not aligned with home array; array copied
    Array eymf not aligned with home array; array copied
    Independent loop parallelized
    expensive communication: all-to-all communication (copy_section)
    expensive communication: all-to-all communication (copy_section)
    expensive communication: all-to-all communication (copy_section)
    expensive communication: all-to-all communication (copy_section)
    expensive communication: all-to-all communication (copy_section)
    communication is generated: array copy
    communication is generated: array copy
    communication is generated: array copy
       ...
```

  - 確かに大量の通信が発生しているように見える。

MAXWELL

- 改善案（1）：dynamicはやめ、
  再分散を手続き境界で行う

```
subroutine ms_solver
 distribute(block,*,*)::jxf
 do i1 = ...
   ...
 end do

 redistribute(*,*,block)::jxf
 do i2 = ...
   ...
 end do
```

```
subroutine ms_solver
 distribute(block,*,*)::jxf
 do i1 = ...
   ...
 end do
 call ms_solver_2(jxf)


subroutine ms_solver_2(jxf)
 distribute(*,*,block)::jxf
 do i2 = ...
   ...
 end do
end
```

サブルーチンとして括り出す。

再分散する配列を引数で渡し、
仮引数を静的な分散で宣言する。

MAXWELL

- 性能測定結果（NEC府中 SX-7 シングルノード）

台数	1	4	8
F90版（秒）	6.26		
HPF版（秒） オリジナル	6.42 （0.98）	1.81 （3.46）	1.20 （5.22）
HPF版（秒） 括り出し	6.40 （0.98）	1.76 （3.56）	1.16 （5.40）

MAXWELL

- 改善案(2):ループのマッピングを明示し、
  データの再分散は処理系に任せる

```
subroutine ms_solver
 distribute(block,*,*)::jxf
 do i1 = ...
   ...
 end do

 redistribute(*,*,block)::jxf
 do i2 = ...
   ...
 end do
```

```
subroutine ms_solver
 distribute(block,*,*)::jxf
 distribute(*,block)::exf
 do i1 = ...
   ...
 end do
(jxfの再分散が自動挿入される)
 do i2 = ...
  on home(exf(:,i2)) begin
   ...
  end on
 end do
```

・ループのマッピングをON文で明示

・配列の再分散は、ループに合わせて
コンパイラが自動的に挿入する。

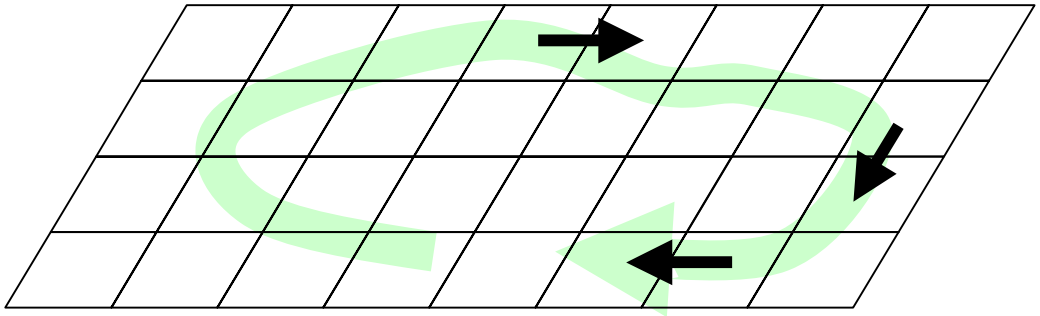
MAXWELL

- 性能測定結果（NEC府中 SX-7 シングルノード）

台数	1	4	8
F90版（秒）	6.26		
HPF版（秒） オリジナル	6.42 （0.98）	1.81 （3.46）	1.20 （5.22）
HPF版（秒） 括り出し	6.40 （0.98）	1.76 （3.56）	1.16 （5.40）
HPF版（秒） ON指示	6.41 （0.98）	1.77 （3.54）	1.16 （5.40）

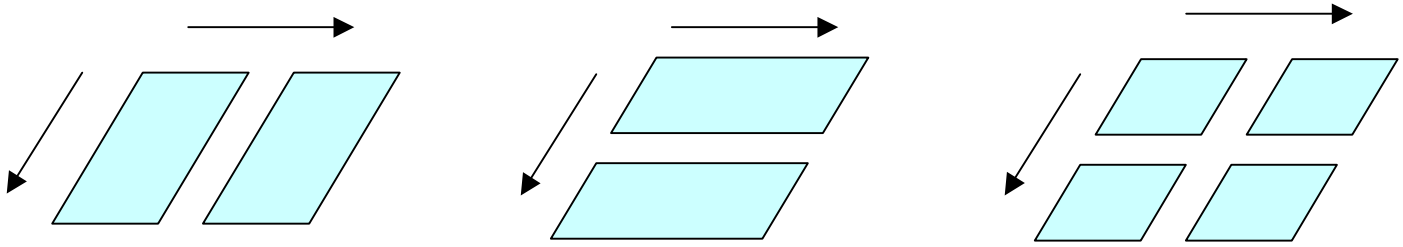
CIP2D

- CIP法（Cubic Interpolated Propagation法）
  - 流体方程式の差分解法（2次元）



格子点ごとに、
参照する「隣」の
方向が異なる

  - どの方向でも並列化可能



(block,*)　　(*,block)　　(block,block)

今回はこれを選択

CIP2D

- プログラムの主要部

```
do j=2, maxy-1
   do i=2, maxx-1
      isgn=1
      if(un(i,j).lt.0.d0)then
         isgn=-1
      endif
      jsgn=1
      if(vn(i,j).lt.0.d0)then
         jsgn=-1
      endif
      xsgn=isgn
      ysgn=jsgn
      im1=i-isgn
      jm1=j-jsgn

      a8=fs(i,j)-fs(im1,j)-fs(i,jm1)+fs(im1,jm1)
      ...
   end do
end do
```

IF文の結果により
j+1 または j-1 のいずれかになる。
すなわち、流速の正負により、
参照する「隣」が変わる。

CIP2D

- HPF化のポイント
  - 「隣」の参照があるものには、適切な SHADOW を指定

```
!hpf$ shadow(0:0,1:1)::fs,...
```

  - 並列化対象ループに independent を指定

```
!hpf$ independent, new(...)
      do j=2, maxy-1
         do i=2, maxx-1
            ...
         end do
      end do
```

CIP2D

- 性能測定結果（NEC府中 SX-6 シングルノード）

台数	1	4	8
F90版（秒）	23.33		
HPF版（秒） （F90比）	29.52 （0.79）	15.96 （1.46）	15.56 （1.50）

CIP2D

- もっと速くならないか？
- コンパイルメッセージ（-Minfo）を見てみると

```
18, Array fs not aligned with home array; array copied
    Array gx not aligned with home array; array copied
    Array gy not aligned with home array; array copied
    Independent loop parallelized
    communication is generated: array copy
    communication is generated: array copy
    communication is generated: array copy
19, Independent loop
     ...
```

  - メインループで大量の通信が発生している
    - jm1がj±1であることが処理系にはわからないため、SHIFT通信の代わりに全コピー通信を生成している

CIP2D

- 改善案(1):「j±1」を書き下す

```
do j=2, maxy-1
   do i=2, maxx-1
      ...
      jsgn=1
      fs_im1jm1=fs(im1,j-1)
      fs_ijm1=fs(i,j-1)
      if(vn(i,j).lt.0.d0)then
         jsgn=-1
         fs_im1jm1=fs(im1,j+1)
         fs_ijm1=fs(i,j+1)
      endif
      ...
      a8=fs(i,j)-fs(im1,j)-fs_ijm1+fs_im1jm1
      ...
   end do
end do
```

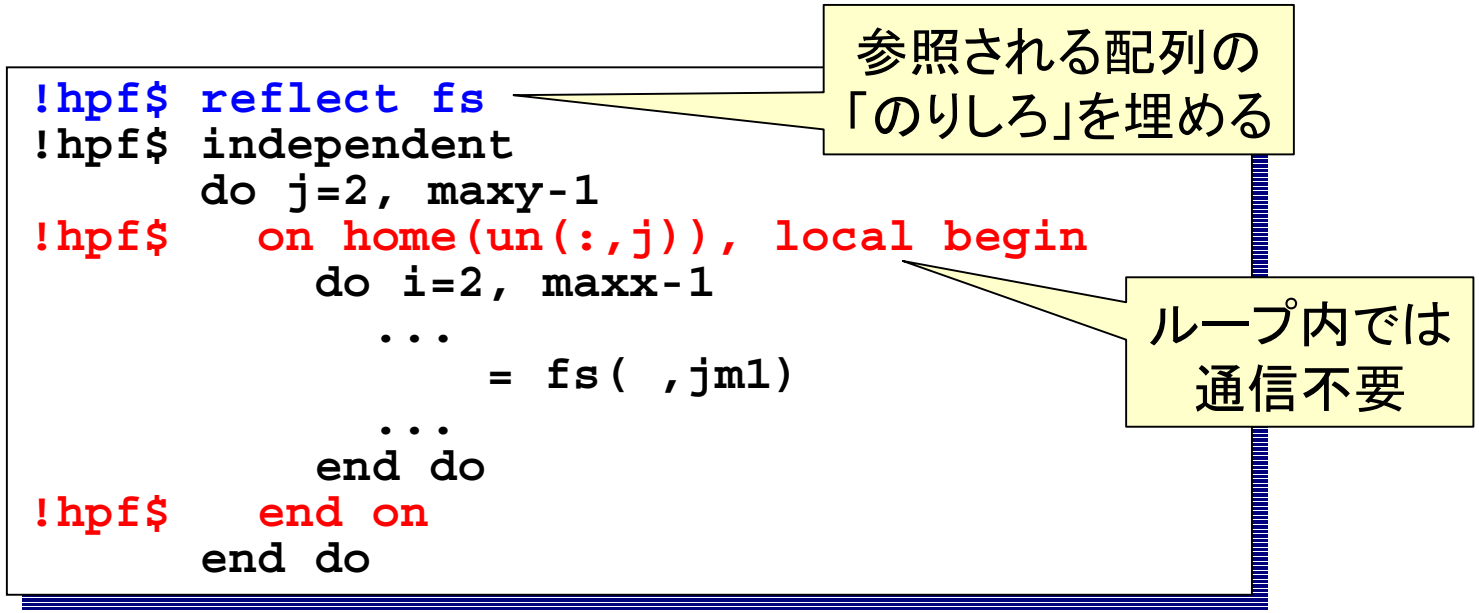
CIP2D

- 性能測定結果（NEC府中 SX-6 シングルノード）

台数	1	4	8
F90版（秒）	23.33		
HPF版（秒） オリジナル	29.52 （0.79）	15.96 （1.46）	15.56 （1.50）
HPF版（秒） 書き下し	26.15 （0.89）	9.83 （2.37）	7.15 （3.26）

CIP2D

- 改善案(2):local／reflect 指示文を使う



  - コード本文の修正は不要

CIP2D

- 性能測定結果（NEC府中 SX-6 シングルノード）

台数	1	4	8
F90版（秒）	23.33		
HPF版（秒） オリジナル	29.52 （0.79）	15.96 （1.46）	15.56 （1.50）
HPF版（秒） 書き下し	26.15 （0.89）	9.83 （2.37）	7.15 （3.26）
HPF版（秒） local/reflect	27.26 （0.86）	9.85 （2.37）	6.87 （3.40）

HPF化の実際

- まとめ
  - プログラムのどこで、どういう通信が起こるかを常にイメージしながら、配列の分割方法を考える。
  - 通信を最小化するためには、どこで、どういう通信を「起こさせる」のが最適かを考える。
  - 最適な通信を起こさせるための記述方法はいろいろあるが、まずは楽な（自分のわかりやすい）方法を選べばよい。
  - コンパイル時の情報メッセージはよく吟味。
  - HPF/JA拡張は便利です（笑）。