FUJITSU THE POSSIBILITIES ARE INFINITE

HPF(High Performance Fortran)講習会
入門編その2

岩下 英俊
富士通(株)
次世代テクニカルコンピューティング開発本部
2009/07/16-17

(C) HPFPC & FUJITSU Ltd., 2009

その2の内容

- スカラ変数
  - スカラ変数の使い方(4.1)
  - 集計計算(4.2)
- 多次元配列と重複割付け(4.3)
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法
- 手続呼出し
  - 手続呼出しの種類(5.1)
  - グローバル手続の呼出し(5.2)
  - ローカル手続の呼出し(5.3)
- 分散配列に関する制限事項(5.4)
- HPFによる並列化の方法(5.5)

(C) HPFPC & FUJITSU Ltd., 2009

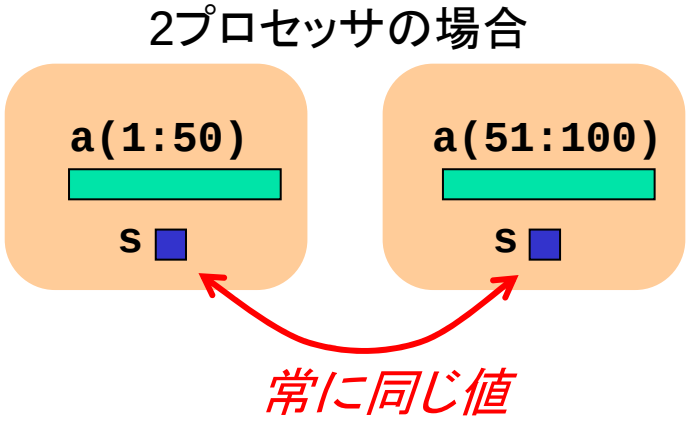
スカラ変数の割付け

- 分散と重複
  - 配列は（原則として）分散割付け
  - スカラは重複割付け
    - 値の一意性は、システムが保証

【例】

```
      real a(100),s
!HPF$ DISTRIBUTE a(block)

      ...
```



(C) HPFPC & FUJITSU Ltd., 2009

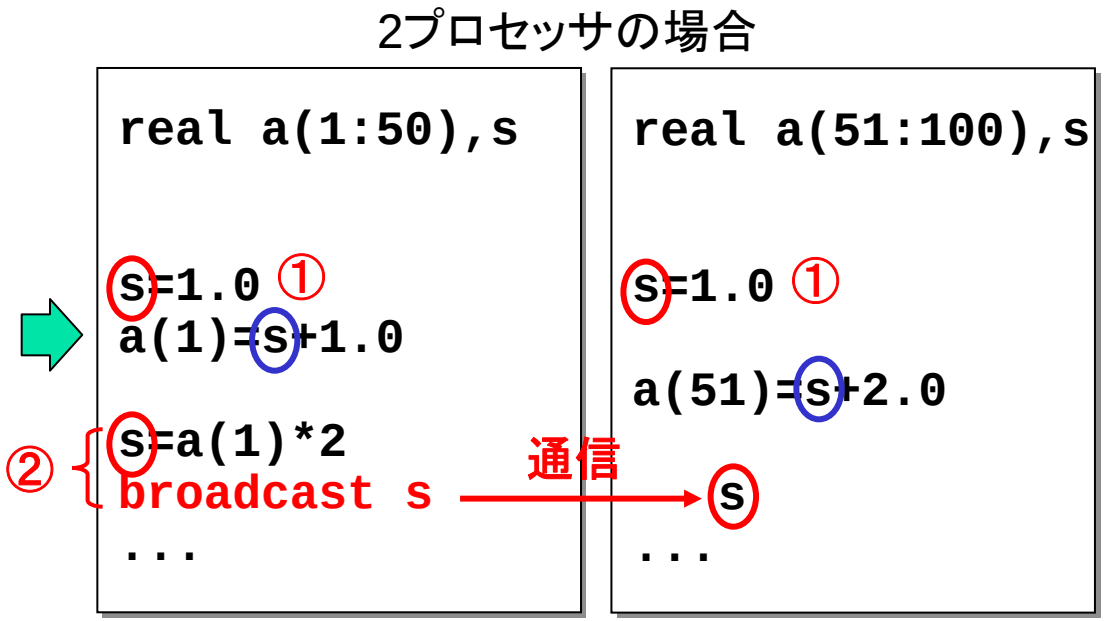
スカラ変数のアクセス(1/2)

- 並列実行の外では、値の一意性を自動保証
  - 読み出しは、常に自プロセッサから
  - 書き込みは
    - ① 全プロセッサが、それぞれ行う（冗長計算）。または、
    - ② 1プロセッサで書き込んだ値を、全プロセッサに放送

【例】

```
      real a(100),s
!HPF$ DISTRIBUTE a(block)

      s=1.0
      a(1)=s+1.0
      a(51)=s+2.0
      s=a(1)*2
      ...
```



(C) HPFPC & FUJITSU Ltd., 2009

スカラ変数の割付け(2/2)

- 並列実行中は、一時的なばらつきが起こる
  - 読むだけの変数なら、問題なし
  - 書き込みのある変数は、用途毎に分類
    - 自動並列では、自動的に識別

```
do i=...
 tmp=a(i-1)+a(i)
 b(i)=tmp
enddo
```

(e) ループ内局所変数
ループを繰り越さないで役目を終える変数。
通常これがほとんど。

NEW変数

```
do i=...
 if(...) ifound=i
enddo
```

(f) 選択的代入
ループ内で1度だけ代入される変数。
（複数回代入なら、並列化可能条件に反する。）

```
do i=...
 asum=asum+a(i)
 if(a(i)>ax) ax=a(i)
enddo
```

(g) 集計計算
プロセッサを跨ぐ総和、最大値など、特別なパターン。

集計変数

(C) HPFPC & FUJITSU Ltd., 2009

INDEPENDENT指示文のNEW節

NEW変数を含むループの並列化

!HPF$ INDEPENDENT [, *〈節〉*] …

〈節〉 はNEW節またはREDUCTION節

NEW節は **NEW(*v*$_1$, *v*$_2$, …)**

v$_1$ や *v*$_2$ はNEW変数. 変数名で指定.

```
!HPF$ INDEPENDENT,NEW(k,tmp,j)
      do i=m,n
       k=i+1
       do j=k,n
         tmp=a(i-1,j)+a(i,j)
         b(k)=b(k)+tmp
       enddo
      enddo
```

多くのコンパイラでは、無指定のスカラ変数をNEW変数と判断する最適化を行う。

(C) HPFPC & FUJITSU Ltd., 2009

集計計算の並列化の原理

- 厳密な意味では、並列化できない。
  - 演算順序の変更によって、並列化
    - 交換則と結合則が成り立つ演算（総和、最大値など）に限る

```
asum=0.0
do i=1,8
   asum=asum+a(i)
end do
```

並列化 ⇒

```
asum=0.0
do i=1,4
   asum=asum+a(i)
end do
asum=(asumのプロセッサ間総和)
```

```
asum=0.0
do i=5,8
   asum=asum+a(i)
end do
```

a(1) + a(2) + ・・・ + a(8)

(a(1)+a(2)+a(3)+a(4)) + (a(5)+a(6)+a(7)+a(8))

  - 浮動小数点演算では、誤差の範囲で逐次と結果が違う。

(C) HPFPC & FUJITSU Ltd., 2009

プログラム例:集計計算

- 分散配列の総和を返す関数
  - input: a(n), n, a0
  - output: $a0+\sum_{i=1}^{n} a(i)$

```
1         real function asum(a,n,a0)
2         real a(n),a0
3  !HPF$ DISTRIBUTE a(CYCLIC)
4
5         asum=a0
6  !HPF$ INDEPENDENT,REDUCTION(asum)
7         do i=1,n
8          asum=asum+a(i)
9         end do
10        return
11        end
```

INDEPENDENT指示を書くならREDUCTION節は必須

集計文
このパターンから加算のreductionであることを自動判定

(C) HPFPC & FUJITSU Ltd., 2009

INDEPENDENT指示文のREDUCTION節

集計(リダクション)計算

!HPF$ INDEPENDENT [, *〈節〉*] . . .

〈節〉 は、NEW節またはREDUCTION節

REDUCTION節は

REDUCTION([*〈種別1〉* :] r_1 , r_2, ...**)**

REDUCTION(*〈種別2〉* : r_1 / i_{11} , i_{12} , ... / , ...**)**

〈種別1〉 は　+ * .AND. .OR. .EQV. .NEQV.
MAX MIN IAND IOR または IEOR

〈種別2〉 は　FIRSTMAX FIRSTMIN LASTMAX
または LASTMIN

r_1 や r_2 は集計変数(変数名)

i_{11} や i_{12} は位置変数(スカラ変数名)

3つの書式
- 種別指定なし
- 種別1を指定
- 種別2を指定

(C) HPFPC & FUJITSU Ltd., 2009

REDUCTIONの書式1

- 書式1：演算種別指定なし（HPF2.0仕様）

  REDUCTION(asum, amax)

  - 集計変数は、集計文だけに表れてよい。
  - 集計文の形式は以下に限定

    （rは集計変数、expはrを含まない式）

    - r = r *op* exp　または　r = exp *op* r

      op は + * .AND. .OR. .EQV. .NEQV. のいずれか
    - r = *f* (r, exp)　または　r = *f* (exp, r)

      f は MAX MIN IAND IOR IEOR のいずれか

      【例】**amax = MAX(amax, a(i))**
  - 制限違反はエラー検出される。
    - 集計文以外での集計変数の参照
    - 集計文の形式の違反

(C) HPFPC & FUJITSU Ltd., 2009

REDUCTIONの書式1の例

分散配列 a から、以下の値を得る。

- aamax　a の配列要素の絶対値の最大値

```
1         parameter(m=100)
2         real a(m)
3  !HPF$ DISTRIBUTE a(BLOCK)
4
5         aamax = -1.0
6  !HPF$ INDEPENDENT, REDUCTION(aamax)
7         do i=1,m
8           aamax = max( aamax, abs(a(i)) )
9         end do
```


(C) HPFPC & FUJITSU Ltd., 2009

REDUCTIONの書式2

- 書式2：種別1の指定（HPF/JA拡張仕様）

```
REDUCTION(+:asum)
REDUCTION(MAX:amax)
```

  - 集合演算の演算子・関数名を明示
    - + * .AND. .OR. .EQV. .NEQV. MAX MIN IAND IOR IEOR のいずれか
    - OpenMPと同じ書式
  - 書式1の機能を包含し、制限を緩和
    - 集計変数の出現場所は自由。呼出し手続で更新してもよい。
    - 意味的に、集計計算でなければならない。
      - 利用者責任。コンパイラはエラー検出しない（できない）。
- 書式1と2の使い分け
  - 書式1の方が安全。エラー検出される。
  - 書式1で表現できなければ、注意して書式2を使う。

(C) HPFPC & FUJITSU Ltd., 2009

REDUCTIONの書式2の例

分散配列 a から、以下の値を得る。

- aamax　a の配列要素の絶対値の最大値

```
1        parameter(m=100)
2        real a(m)
3  !HPF$ DISTRIBUTE a(BLOCK)
4
5        aamax = -1.0
6  !HPF$ INDEPENDENT, REDUCTION(MAX:aamax)
7        do i=1,m
8          if ( aamax < abs(a(i)) ) then
9            aamax = abs(a(i))
10         end if
11       end do
```


(C) HPFPC & FUJITSU Ltd., 2009

REDUCTIONの書式3

- 書式3：種別2の指定（HPF/JA拡張仕様）

  REDUCTION(FIRSTMAX:amax/i,j/)

  - MAX, MINの機能拡張
    - FIRSTMAX, FIRSTMIN, LASTMAX, LASTMIN のいずれか
  - 最大（小）値を探すループ実行で、最大（小）値が得られた位置や、その位置での他の情報を、最大（小）値と同時に得る。

    【例】分散配列 a の絶対値最大の要素を探し、
    その位置 i1 と値 a1=a(i1) を求める。

```
      ...
      aamax = -1.0
!HPF$ INDEPENDENT, REDUCTION(FIRSTMAX:aamax/i1,a1/)
      do i=1,m
        if ( aamax < abs(a(i)) ) then
          i1 = i
          a1 = a(i1)
          aamax = abs(a1)
        end if
      end do
```


(C) HPFPC & FUJITSU Ltd., 2009

その2の内容

- スカラ変数
  - スカラ変数の使い方(4.1)
  - 集計計算(4.2)

- 多次元配列と重複割付け(4.3)
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法
- 手続呼出し
  - 手続呼出しの種類(5.1)
  - グローバル手続の呼出し(5.2)
  - ローカル手続の呼出し(5.3)
- 分散配列に関する制限事項(5.4)
- HPFによる並列化の方法(5.5)

(C) HPFPC & FUJITSU Ltd., 2009

HPFの基本（復習）

- 基本的な書き方3種

データの分散 DISTRIBUTE	ループ並列化 INDEPENDENT	ループ処理分担 ON HOME
書く	書かない	書かない
書く	書く	書かない
書く	書く	書く

  - 必要度に合わせて
    - 自動でやってみて、だめなら書く
    - 性能的にシビアな部分は、最初から全部書く

- まだ性能が不十分なら
  - 通信の明示
    - SHADOW+REFLECT+LOCALの使用を、「その1」で紹介
  - 並列アルゴリズムの変更
    - データ分散の選択

(C) HPFPC & FUJITSU Ltd., 2009

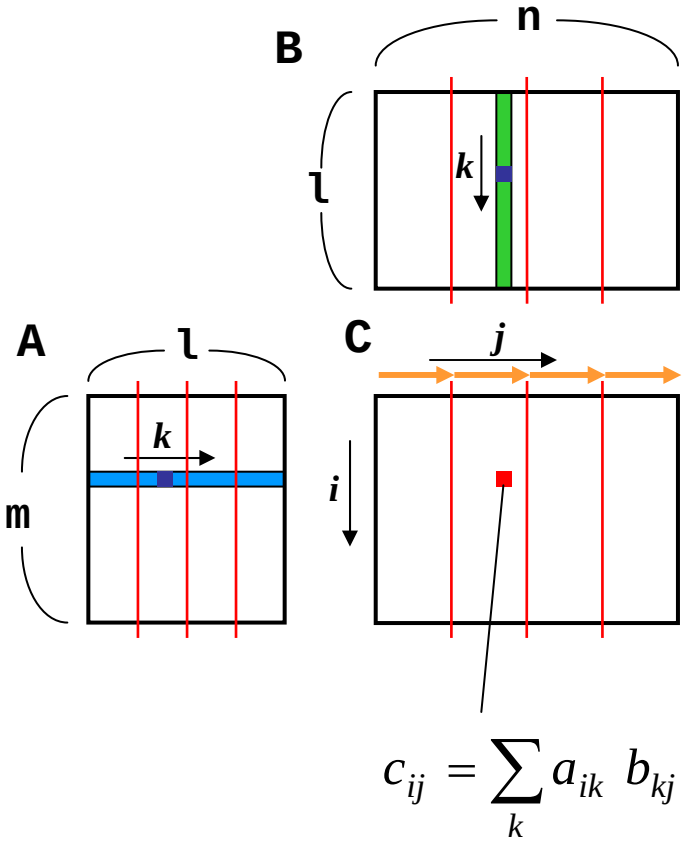
行列積プログラム（1）(1/2)

- 行列積 C = A × B を計算するサブルーチン
  - Input: [m, l] 行列 A, [l, n] 行列 B; サイズ m, n, l
  - Output: [m, n] 行列 C
  - A, B, C とも2次元目でblock分散

```
1        subroutine mmul(A,B,C,m,n,l)
2        real A(m,l),B(l,n),C(m,n)
3  !HPF$ DISTRIBUTE (*,BLOCK) :: A,B,C
4
5        C=0.0
6        do j=1,n
7         do i=1,m
8          do k=1,l
9           C(i,j)=C(i,j)+A(i,k)*B(k,j)
10         end do
11        end do
12       end do
13       return
14       end
```

j で自動並列化

4並列の場合



$$c_{ij} = \sum_{k} a_{ik}\ b_{kj}$$

(C) HPFPC & FUJITSU Ltd., 2009

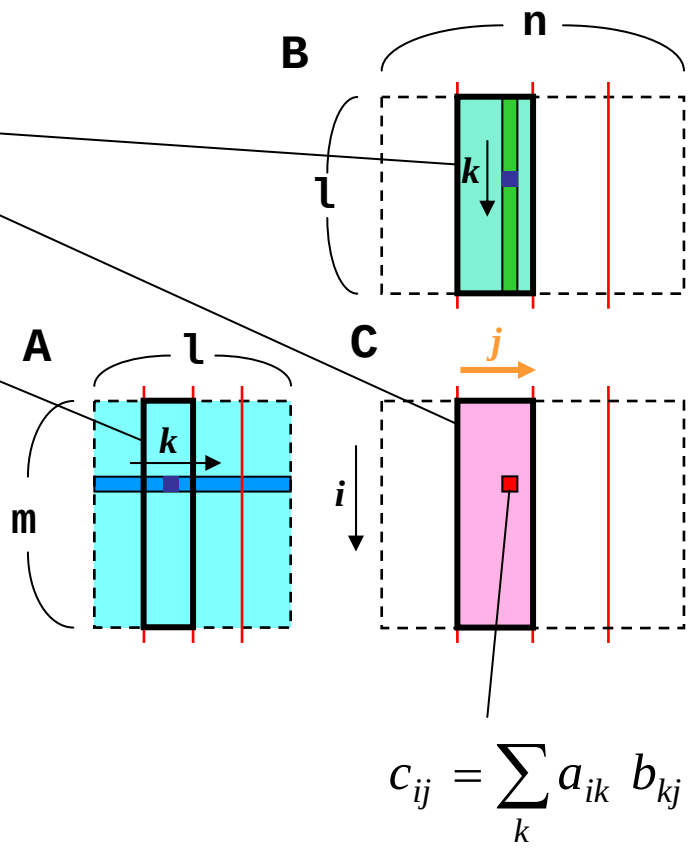
行列積プログラム（1）(2/2)

- 各プロセッサの動作イメージ
  - 並列化した結果、Aの参照で通信が必要
    → コンパイラが自動生成

```
1   subroutine mmul(A,B,C,m,n,l)
2   real A(m,･･･),B(l,･･･),C(m,･･･)
3
4
5   C=0.0
6   do j∈担当範囲
7    do i=1,m
8     do k=1,l
9      C(i,j)=C(i,j)+A(i,k)*B(k,j)
10    end do
11   end do
12  end do
13  return
14  end
```

通信発生
（自動生成）

4並列の場合



$$c_{ij} = \sum_{k} a_{ik}\ b_{kj}$$

(C) HPFPC & FUJITSU Ltd., 2009

行列積プログラム(2)

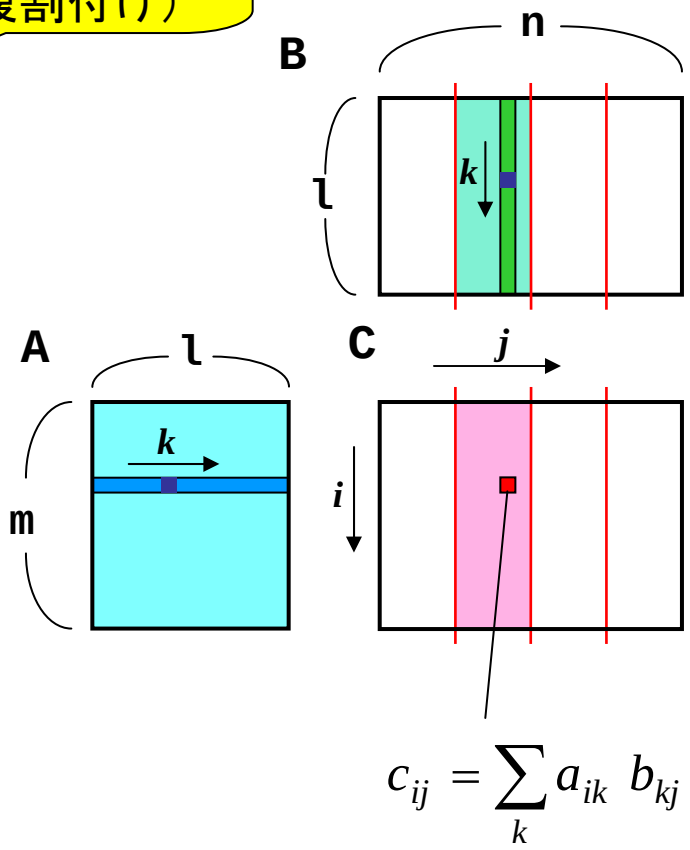
- 性能改善版
  - Aのコピーを全員が持つ ・・・重複配列

```
1         subroutine mmul(A,B,C,m,n,l)
2         real A(m,l),B(l,n),C(m,n)
3  !HPF$ DISTRIBUTE (*,BLOCK) :: B,C
4
5         C=0.0
6         do j=1,n
7          do i=1,m
8           do k=1,l
9            C(i,j)=C(i,j)+A(i,k)*B(k,j)
10          end do
11         end do
12        end do
13        return
14        end
```

Aは分散指示なし（重複割付け）

j で自動並列化

どれも通信なし

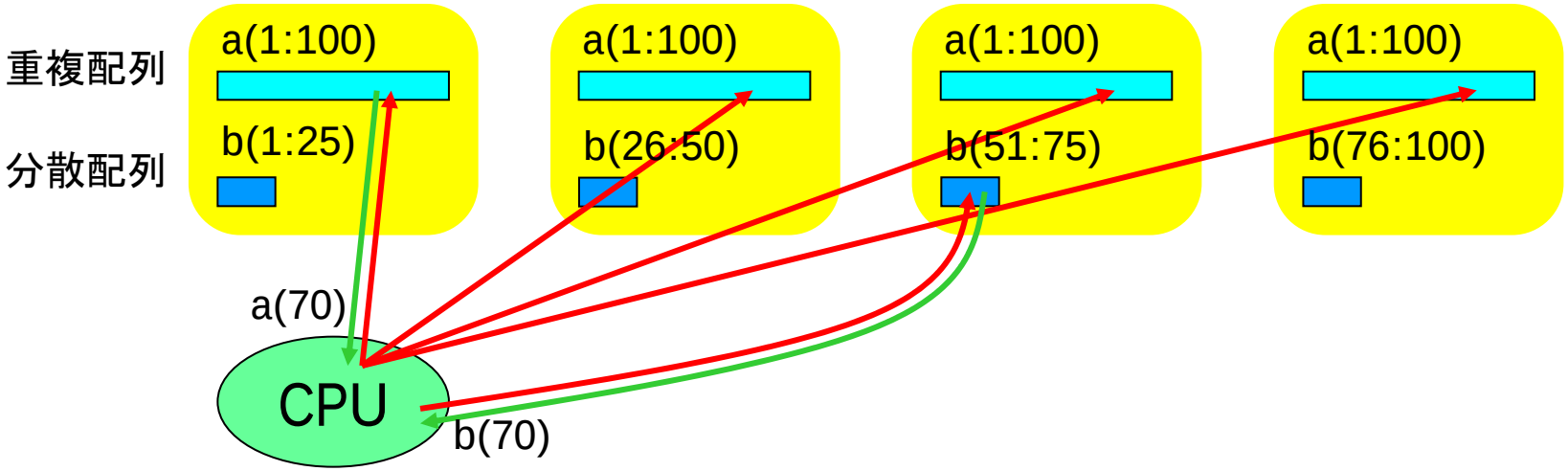


$$c_{ij} = \sum_{k} a_{ik}\ b_{kj}$$

(C) HPFPC & FUJITSU Ltd., 2009

重複配列

- 分散配列と比較して
  - 参照はいつも通信なしで高速。
  - 更新は遅い。1プロセッサからbroadcast、または全プロセッサの冗長実行（並列性なし）。
    - コンパイラは、常に同じ値を保証。
  - メモリを食う。メモリ消費量に台数効果なし。
- 用途
  - 値の更新が少ない、読むだけのデータ

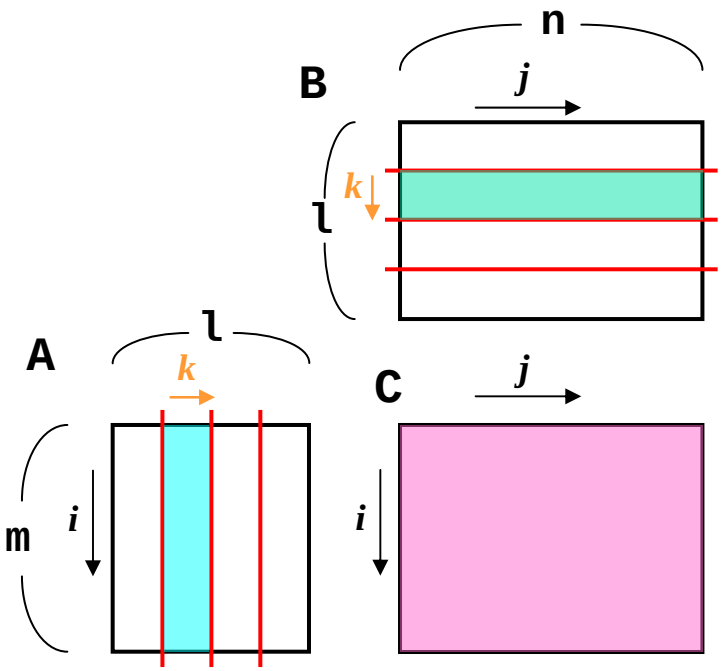


(C) HPFPC & FUJITSU Ltd., 2009

行列積プログラム(3)

- C を重複配列にする方法
  - 配列の集計計算

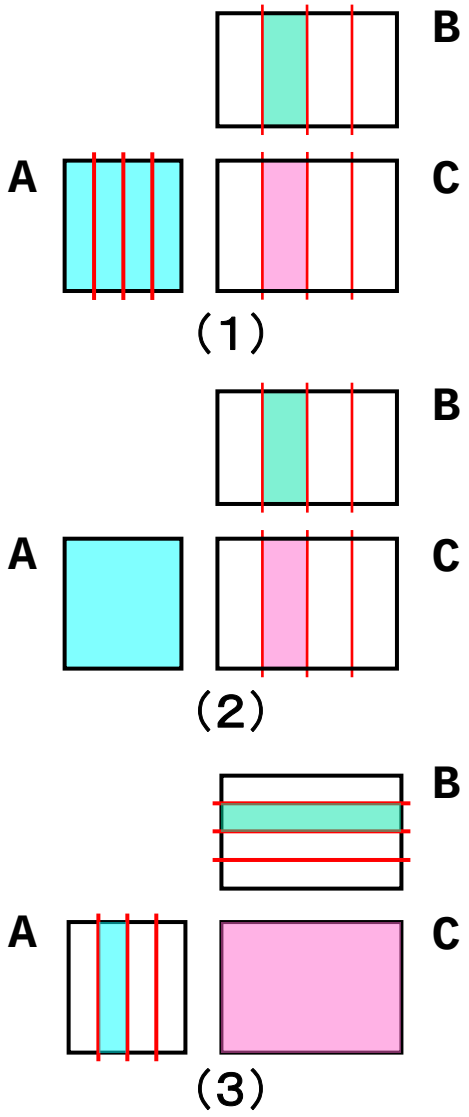
```
1         subroutine mmul(A,B,C,m,n,l)
2         real A(m,l),B(l,n),C(m,n)
3  !HPF$ DISTRIBUTE A(*,BLOCK)
4  !HPF$ DISTRIBUTE B(BLOCK,*)
5
6         C=0.0
7  !HPF$ INDEPENDENT,NEW(j,i),REDUCTION(C)
8         do k=1,l
9  !HPF$   ON HOME(A(:,k))
10         do j=1,n
11          do i=1,m
12           C(i,j)=C(i,j)+A(i,k)*B(k,j)
13          end do
14         end do
15        end do
16        return
17        end
```



(C) HPFPC & FUJITSU Ltd., 2009

分散方法の比較

- データ配置、通信コスト、メモリ効率
  - （1）A,B,Cとも2次元目で分散
    - Aの参照で通信が発生
    - 配置のメモリ使用量は最小
      - 実行時に大きな一時領域が必要
  - （2）Aは重複、B,Cは2次元目で分散
    - 通信が発生しないので、最も高速
    - Aのメモリ使用量が大きい
  - （3）A/Bは2/1次元目で分散、Cは重複
    - Cの全配列集計計算のコスト
    - Cのメモリ使用量が大きい
- 全体の性能のためには、プログラム全体を考慮して分散を選択



(C) HPFPC & FUJITSU Ltd., 2009

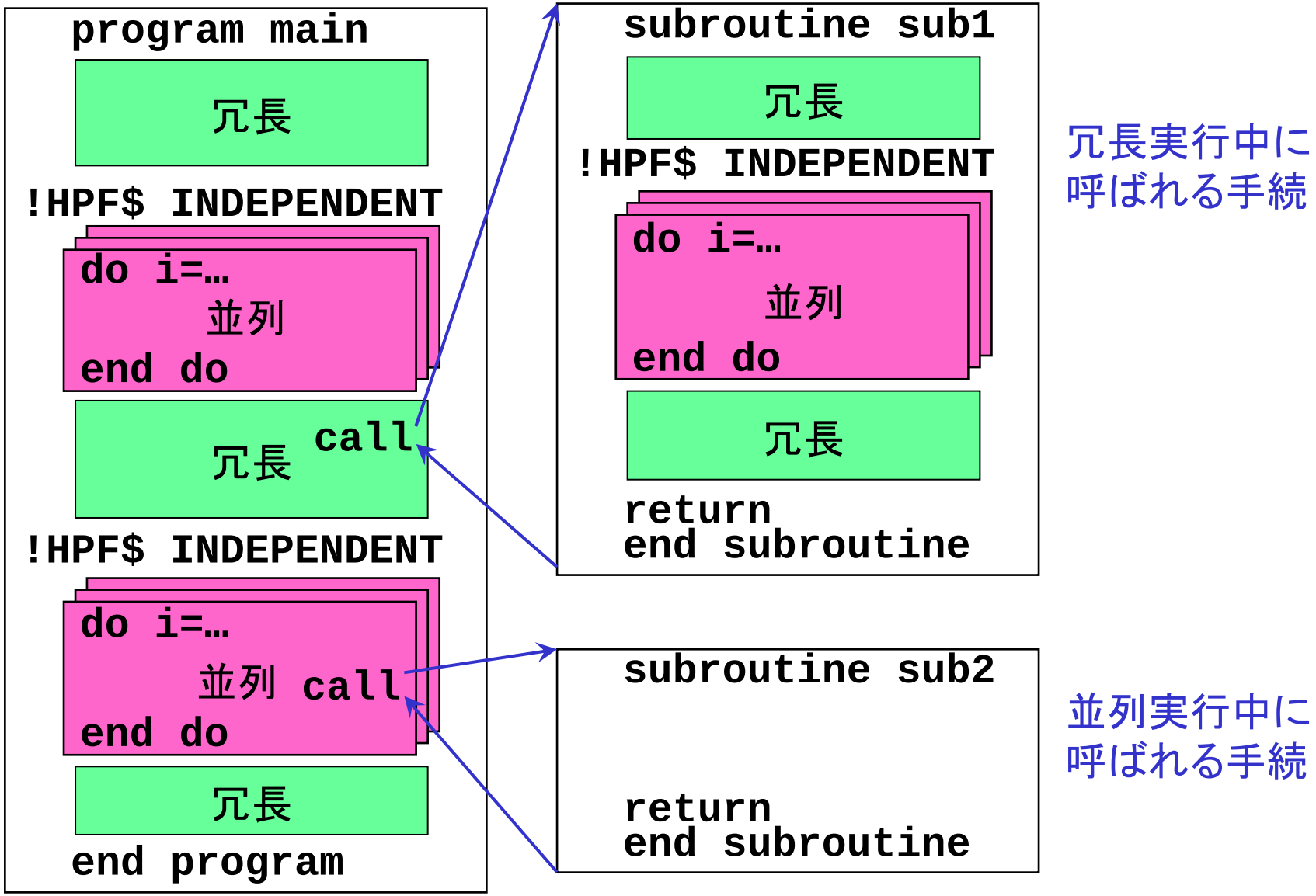
その2の内容

- スカラ変数
  - スカラ変数の使い方(4.1)
  - 集計計算(4.2)
- 多次元配列と重複割付け(4.3)
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法

- 手続呼出し
  - 手続呼出しの種類(5.1)
  - グローバル手続の呼出し(5.2)
  - ローカル手続の呼出し(5.3)
- 分散配列に関する制限事項(5.4)
- HPFによる並列化の方法(5.5)

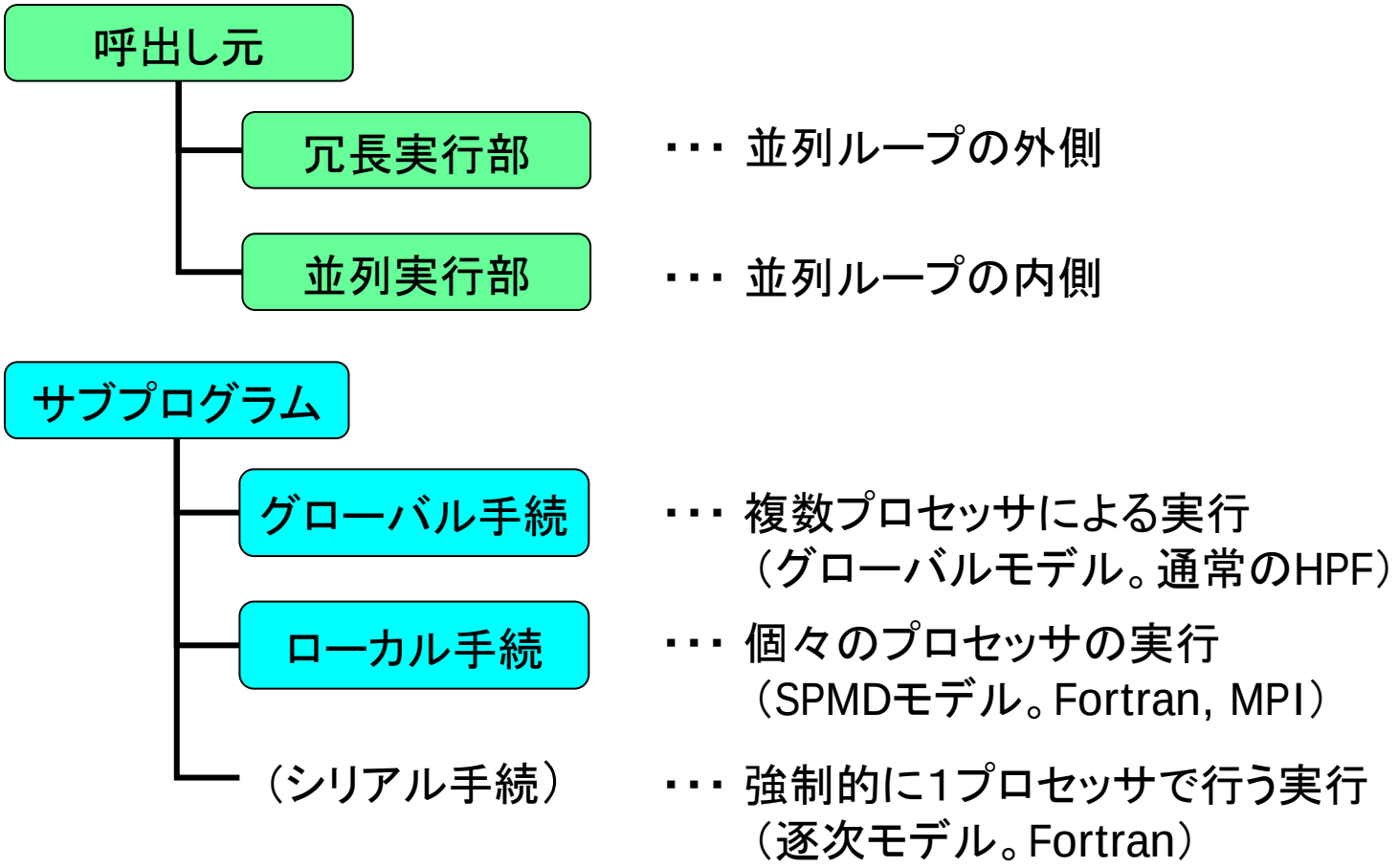
(C) HPFPC & FUJITSU Ltd., 2009

手続呼出しの場所による違い



(C) HPFPC & FUJITSU Ltd., 2009

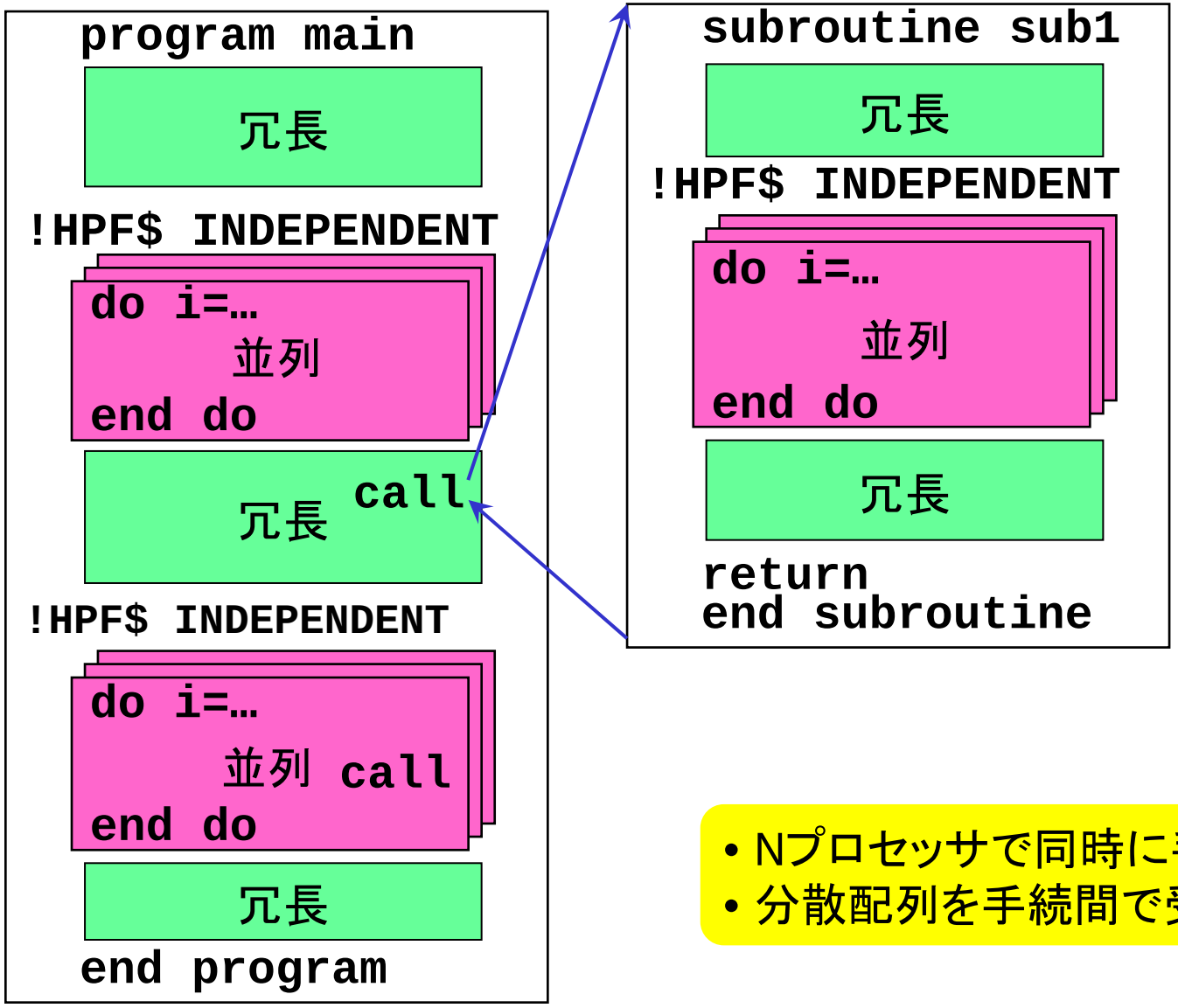
手続呼出しの分類



- 実用的には、2つのパターンで十分
  - 冗長実行部で、グローバル手続（HPF）を呼び出す。
  - 並列実行部で、ローカル手続（Fortran）を呼び出す。

(C) HPFPC & FUJITSU Ltd., 2009

グローバル手続の呼出し

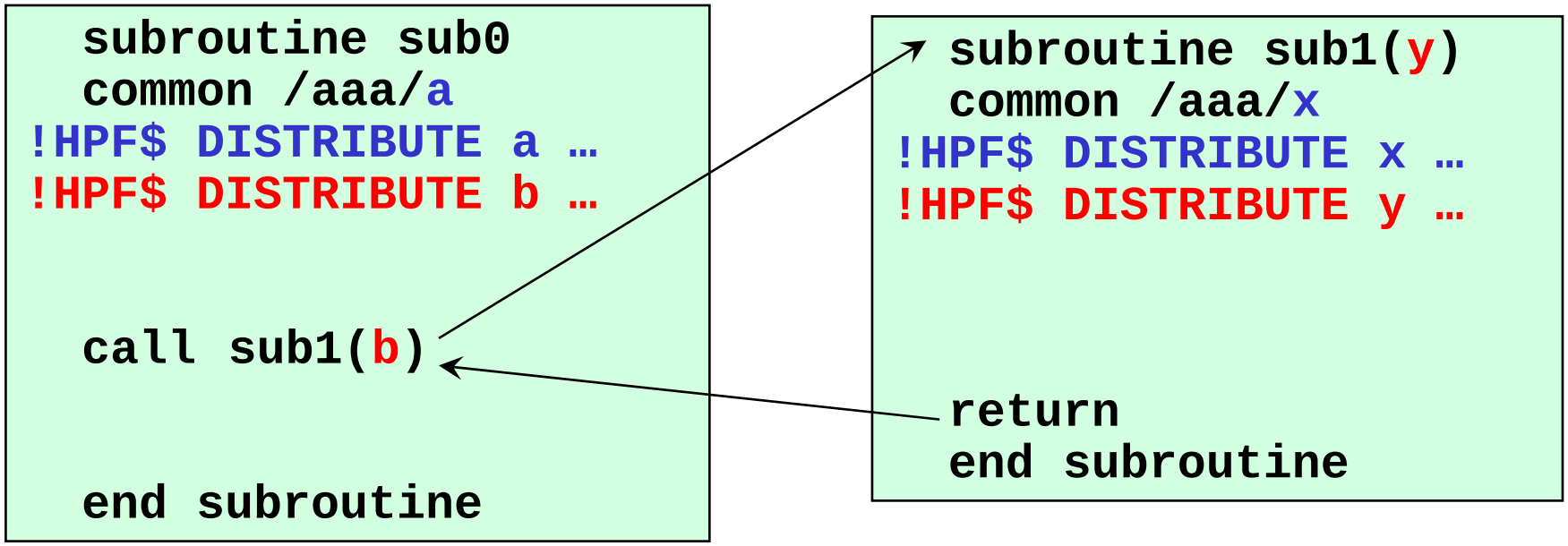


冗長実行中に
呼ばれる手続
||
グローバル手続
（HPF手続）

- Nプロセッサで同時に手続を呼び出す。
- 分散配列を手続間で受け渡しできる。

(C) HPFPC & FUJITSU Ltd., 2009

手続間のデータ渡し



- COMMON変数
  - 分散は利用者責任で一致。
- モジュール変数
  - 記述は一箇所なので、当然一致。
- 実引数と仮引数
  - 分散は一致していなくてもよい。→ 自動再マッピング

(C) HPFPC & FUJITSU Ltd., 2009

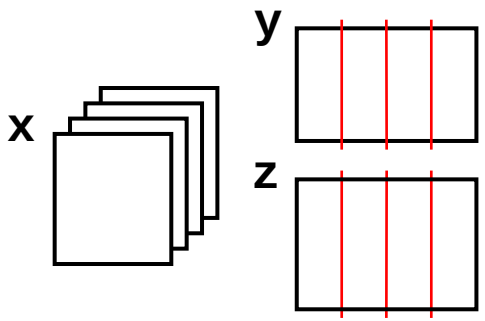
引数の分散が一致する場合

```
      real x(n1,m),y(m,n2),z(n1,n2)
!HPF$ DISTRIBUTE (*,BLOCK) :: y,z
      ...
      call mmul(x,y,z,n1,n2,m)
      ...
```

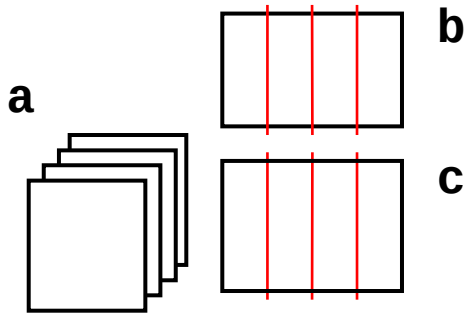
通信なし
（アドレス渡し）

```
      subroutine mmul(a,b,c,m,n,l)
      real a(m,l),b(l,n),c(m,n)
!HPF$ DISTRIBUTE (*,BLOCK) :: b,c

      c=0.0
      do j=1,n
       do i=1,m
        do k=1,l
         c(i,j)=c(i,j)+a(i,k)*b(k,j)
        end do
       end do
      end do
      return
      end
```



分散の有無、
分散次元・形式の
完全一致が条件



(C) HPFPC & FUJITSU Ltd., 2009

引数の分散が一致しない場合

```
      real x(n1,m),y(m,n2),z(n1,n2)
!HPF$ DISTRIBUTE (*,BLOCK) :: x,y,z
      ...
      call mmul(x,y,z,n1,n2,m)
      ...
```

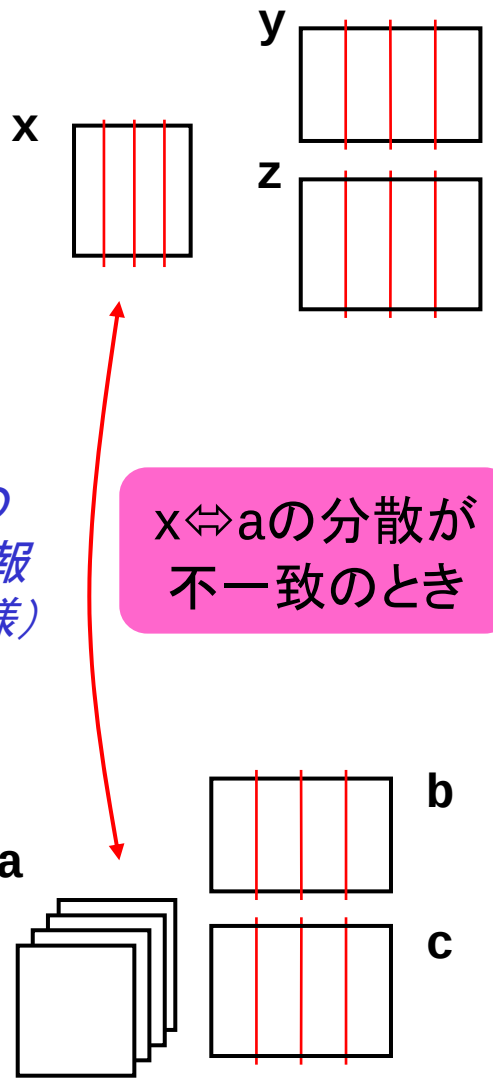
a(m,l) を割付け
x→a のコピー

a→x のコピー
aの開放

ただし文法では、
呼出し側に
サブプログラムの
引数に関する情報
（明示的引用仕様）
が必要

```
      subroutine mmul(a,b,c,m,n,l)
      real a(m,l),b(l,n),c(m,n)
!HPF$ DISTRIBUTE (*,BLOCK) :: b,c

      ...
```



x⇔aの分散が
不一致のとき

(C) HPFPC & FUJITSU Ltd., 2009

明示的引用仕様とは（F90文法）

- 呼出し側のコンパイル時に参照される、呼ばれ側の仕様

(a) INTERFACE文を使う方法

```
subroutine yyy

 interface
 subroutine xxx
   宣言部
 end subroutine
 end interface

 call xxx

end subroutine
```

```
subroutine xxx
   宣言部
   実行部
end subroutine
```

(b) モジュールを使う方法

```
module mmm
contains
 subroutine xxx
   宣言部
   実行部
 end subroutine
end module
```

```
subroutine yyy

 use mmm


 call xxx

end subroutine
```

(c) 内部手続にする方法

```
subroutine yyy

 call xxx

contains
 subroutine xxx
   宣言部
   実行部
 end subroutine
end subroutine
```

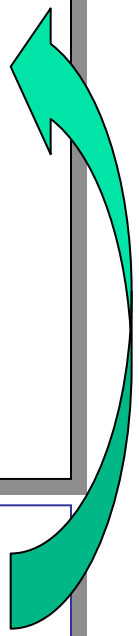

(C) HPFPC & FUJITSU Ltd., 2009

インタフェースブロックの記述例

```
      real x(n1,m),y(m,n2),z(n1,n2)
!HPF$ DISTRIBUTE (BLOCK,*) :: x,y,z
      interface
        subroutine mmul(a,b,c,m,n,l)
        real A(m,l),B(l,n),C(m,n)
!HPF$   DISTRIBUTE (*,BLOCK) :: b,c
        end subroutine
      end interface
      ...
      call mmul(x,y,z,n1,n2,m)
      ...
```

```
      subroutine mmul(a,b,c,m,n,l)
      real a(m,l),b(l,n),c(m,n)
!HPF$ DISTRIBUTE (*,BLOCK) :: b,c
      ...

      end subroutine
```

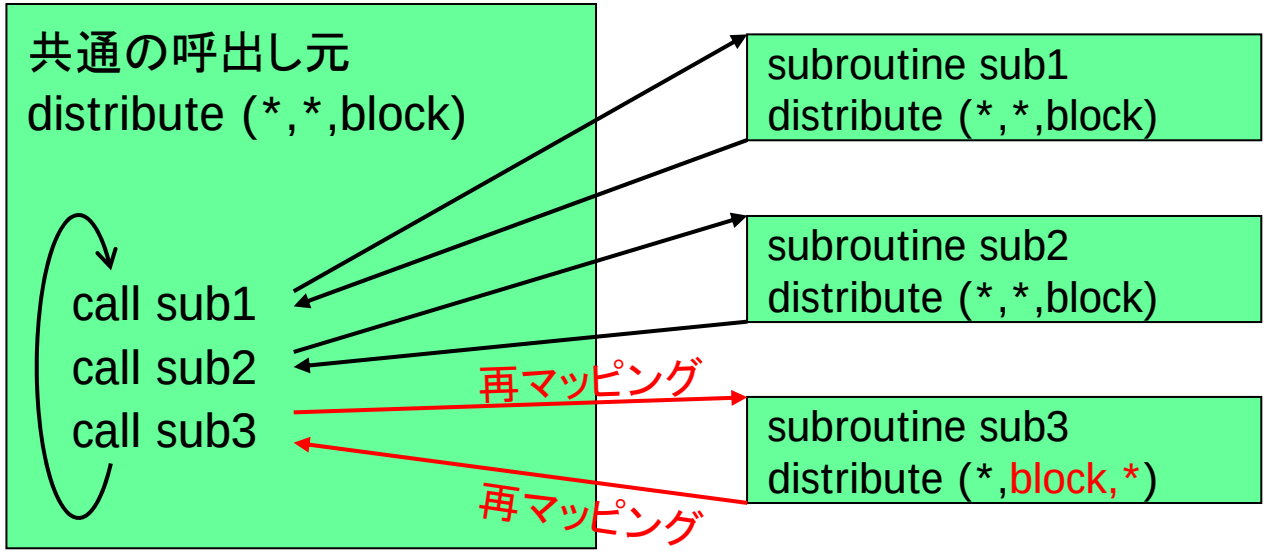


宣言部を
そのままコピー

(C) HPFPC & FUJITSU Ltd., 2009

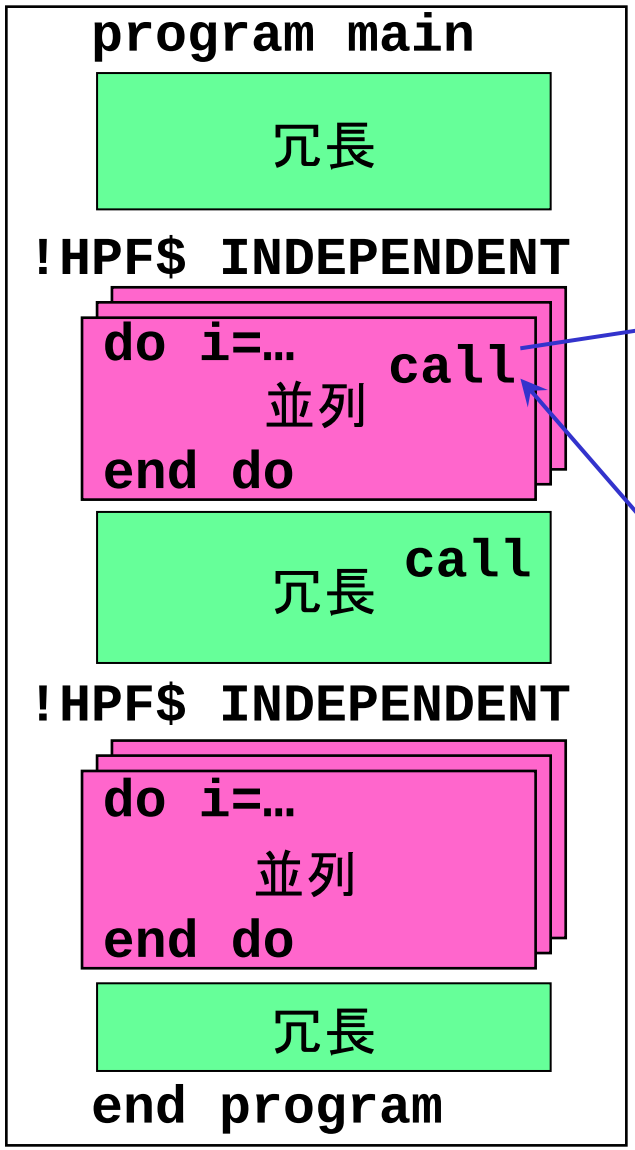
手続きを跨ぐデータ分散のコツ

- アプリケーション全体で、通信が少なくなるデータ分散を考える。
  - 再マッピングを避ける
    - コストの高いサブプログラムで分散を決める。
    - 呼出し元や、他のサブプログラムも、同じ分散にする。
  - 再マッピングを積極的に利用
    - 再マッピング回数を減らす工夫を。



(C) HPFPC & FUJITSU Ltd., 2009

ローカル手続の呼出し




```
subroutine sub2


return
end subroutine
```

並列実行中に
呼ばれる手続
||
逐次手続
（Fortran手続）

- 各プロセッサは、個々独立に手続を呼び出す。
- 手続内では、ローカルなデータしか扱えない。

(C) HPFPC & FUJITSU Ltd., 2009

ローカル手続の書き方・使い方

- Fortranコンパイラでコンパイルしてリンク
  - 可能。詳細手順は実装による。
- HPFコンパイラでコンパイルする方法
  - EXTRINSIC接頭辞（指示文ではない）を使用
    - その手続が（HPFでなく）Fortranであることの宣言

言語と言語モデルの宣言（Fortran仕様の拡張）

EXTRINSIC(〈*言語*〉**,** <*モデル*>**)** SUBROUTINE …

EXTRINSIC(〈*言語*〉**,** <*モデル*>**)** FUNCTION …

EXTRINSIC(〈*言語*〉**,** <*モデル*>**)** MODULE …

〈*言語*〉は　'HPF' または 'FORTRAN'

〈*モデル*〉は　'GLOBAL' 'LOCAL' または 'SERIAL'

デフォルトは ('HPF', 'GLOBAL')

許される組合せは処理系による.

手続本体と明示的引用仕様に必要.

(C) HPFPC & FUJITSU Ltd., 2009

ローカル手続をHPFで記述する例

```
      real a(2,n), c(n)
!HPF$ DISTRIBUTE a(*,BLOCK)
!HPF$ DISTRIBUTE c(BLOCK)

      interface
       EXTRINSIC('Fortran','LOCAL') subroutine foolocal(x,r)
       real x(2),r
       end subroutine
      end interface

!HPF$ INDEPENDENT
      do i=1,n
        call foolocal(a(:,i),c(i))
      enddo
```

```
      EXTRINSIC('Fortran','LOCAL') subroutine foolocal(x,r)
      real x(2),r
      r = sqrt(x(1)**2 + x(2)**2)
      return
      end subroutine
```


(C) HPFPC & FUJITSU Ltd., 2009

その2の内容

- スカラ変数
  - スカラ変数の使い方(4.1)
  - 集計計算(4.2)
- 多次元配列と重複割付け(4.3)
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法
- 手続呼出し
  - 手続呼出しの種類(5.1)
  - グローバル手続の呼出し(5.2)
  - ローカル手続の呼出し(5.3)

- 分散配列に関する制限事項(5.4)
- HPFによる並列化の方法(5.5)

(C) HPFPC & FUJITSU Ltd., 2009

分散配列の制限事項

- 分散配列に関して、Fortran仕様の一部が制限される。
  - 理由：メモリの連続性（記憶列結合、順序結合）が保証できない
  - 例えば、a(1) の次のアドレスに a(2) があるとは限らない。次のプロセッサかもしれない。
- 主な制限事項
  - EQUIVALENCE文中の指定は不可。
  - COMMON文で指定するとき、形状（次元数と大きさ）と分散がすべて一致すること。
  - 実引数と仮引数は、形状が一致すること。
    - 分散不一致は再マッピング条件を満たせば可。
  - 大きさ引継ぎ配列（仮引数 a(*) など）は不可。
- このような配列を含む手続の並列化は、
  - 重複配列として扱う。
  - あきらめて、Fortran手続として呼び出す。

(C) HPFPC & FUJITSU Ltd., 2009

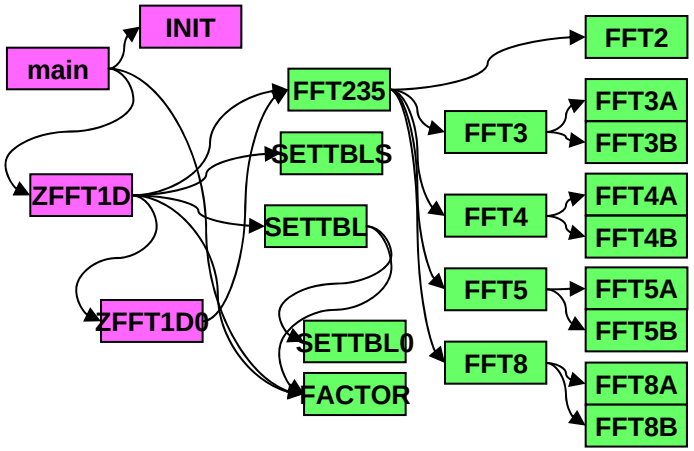
その2の内容

- スカラ変数
  - スカラ変数の使い方(4.1)
  - 集計計算(4.2)
- 多次元配列と重複割付け(4.3)
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法
- 手続呼出し
  - 手続呼出しの種類(5.1)
  - グローバル手続の呼出し(5.2)
  - ローカル手続の呼出し(5.3)
- 分散配列に関する制限事項(5.4)
- HPFによる並列化の方法(5.5)

(C) HPFPC & FUJITSU Ltd., 2009

並列化の手順の提案

1. プログラム全体の手続呼出し関係を把握。
   - 呼出しグラフを書いてみる。
2. 主要変数を見つけて、分散を決める。
   - 分散する次元を決める。
   - 分散形式はblockに仮決めでもよい。
     いつでも変更できる。（ここがHPF！）
3. 並列化の対象と、逐次手続を分ける。
   - 主要変数が引数かCOMMONで渡る手続が、
     並列化の対象
   - 並列ループから呼ばれる手続は、
     逐次手続（ローカル手続）
4. 方針は決まった。後はチューニング。
   - 主要変数以外の変数の分散
   - 計算マッピングが自動で不十分なら、INDEPENDENT、REDUCTON、
     ON HOME、・・・



(C) HPFPC & FUJITSU Ltd., 2009

その2の内容　おしまい

- スカラ変数
  - スカラ変数の使い方（4.1）
  - 集計計算（4.2）
- 多次元配列と重複割付け（4.3）
  - 行列積のプログラム例
  - 重複配列の使い方
  - 多次元配列の分散方法
- 手続呼出し
  - 手続呼出しの種類（5.1）
  - グローバル手続の呼出し（5.2）
  - ローカル手続の呼出し（5.3）
- 分散配列に関する制限事項（5.4）
- HPFによる並列化の方法（5.5）

(C) HPFPC & FUJITSU Ltd., 2009