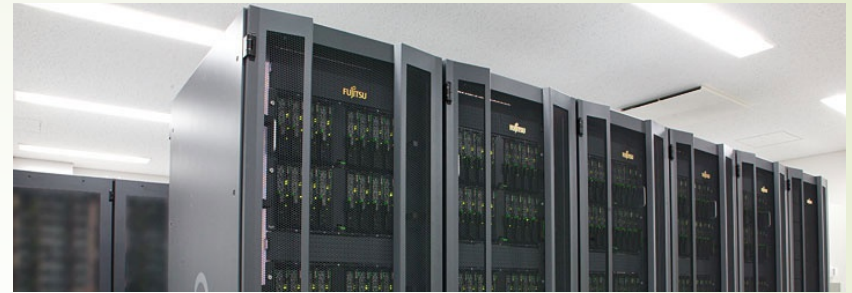


# XcalableMP講習会

村井均 (理研)

# はじめに



- ▶ 大規模シミュレーションなどの計算を行うためには、クラスタのような分散メモリシステムの利用が一般的
- ▶ 並列プログラミングの現状
  - ▶ 大半はMPI (Message Passing Interface) を利用
  - ▶ MPIはプログラミングコストが大きい
- ▶ 目標
  - ▶ 高性能と高生産性を兼ね備えた並列プログラミング言語の開発

# 並列プログラミング言語

## XcalableMP

- ▶ 次世代並列プログラミング言語検討委員会 / PCクラスタコンソーシアムXcalableMP規格部会で検討中。
- ▶ MPIに代わる並列プログラミングモデル
- ▶ 目標:
  - ▶ Performance
  - ▶ Expressiveness
  - ▶ Optimizability
  - ▶ Education cost

The logo for XcalableMP, featuring the word "XcalableMP" in a bold, blue, sans-serif font. The "X" is significantly larger than the other letters. Above the "calable" part, there are three horizontal lines of varying lengths, suggesting motion or a stylized "X".

[www.xcalablemp.org](http://www.xcalablemp.org)

# XcalableMPの特徴（1）

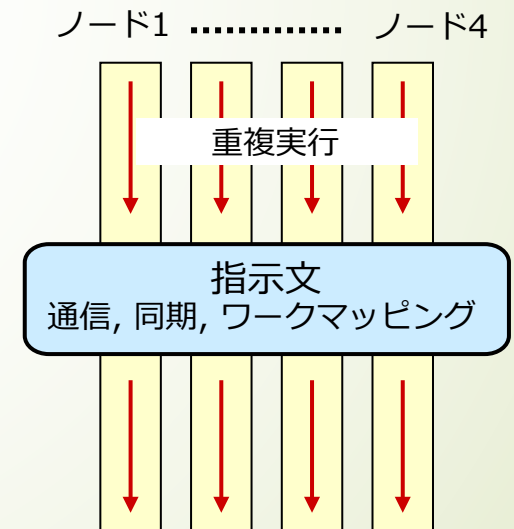
- ▶ Fortran/Cの拡張（指示文ベース）
  - 逐次プログラムからの移行が容易
- ▶ SPMDモデル
  - ▶ 各ノード（並列実行の主体）が独立に（重複して）実行を開始する。

# XcalableMPの特徴（2）

- ▶ 明示的な並列化と通信
  - ▶ ワークマッピング（並列処理）、通信、および同期は「集団的」な指示文によって明示される。
    - チューニングが容易
- ▶ 2つのプログラミングモデル
  - ▶ グローバルビュー
  - ▶ ローカルビュー

# XMPの実行モデル (SPMD)

- 各ノードは、同一のコードを独立に（重複して）実行する。
- 指示文の箇所では、全ノードが協調して動作する（集団実行）。
  - 通信・同期
  - ワークマッピング（並列処理）



# メモリモデル

- ▶ 各ノードは、自身のローカルメモリ上のデータ (ローカルデータ) のみをアクセスできる。
- ▶ 他のノード上のデータ (リモートデータ) にアクセスする場合は、特殊な記法による明示的な指定が必要。
  - ▶ 通信指示文
  - ▶ `coarray`
- ▶ 「分散」されないデータは、全ノードに重複して配置される。

# プログラム例 (MPIとの比較)

## XMP/Cプログラム

```
int array[MAX];
#pragma xmp nodes p[*]
#pragma xmp template t[MAX]
#pragma xmp distribute t[block] onto p
#pragma xmp align array[i] with t[i]

main(){
#pragma xmp loop on t[i] reduction(+:res)
  for (i = 0; i < MAX; i++){
    array[i] = func(i);
    res += array[i];
  }
}
```

シンプル

## MPIプログラム

```
int array[MAX];

main(int argc, char **argv){
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  dx = MAX/size;
  llimit = rank * dx;
  if (rank != (size -1)) ulimit = llimit + dx;
  else ulimit = MAX;
  temp_res = 0;

  for (i = llimit; i < ulimit; i++){
    array[i] = func(i);
    temp_res += array[i];
  }

  MPI_Allreduce(&temp_res, &res, 1, MPI_INT,
               MPI_SUM, MPI_COMM_WORLD);

  MPI_Finalize( );
}
```



# XMPのグローバルレビュー・プログラミング



ローカルビュー: 各ノードが解くべき問題を個別に示す。「ノード1は問題1~25を解け。ノード2は.....」

- ▶ 解くべき問題全体を記述し、それをN個のノードが分担する方法を示す。
  - ▶ 「問題1~100を4人で分担して解け」
- ▶ 分かりやすい（基本的に指示文を挿入するだけ）。
- ▶ 「分担」を指定する方法
  - ▶ データマッピング
  - ▶ ワークマッピング
  - ▶ 通信・同期

# XcalableMP指示文の記法

- ➡ XMPの指示文は、「#pragma xmp」または「!\$xmp」から始まる。

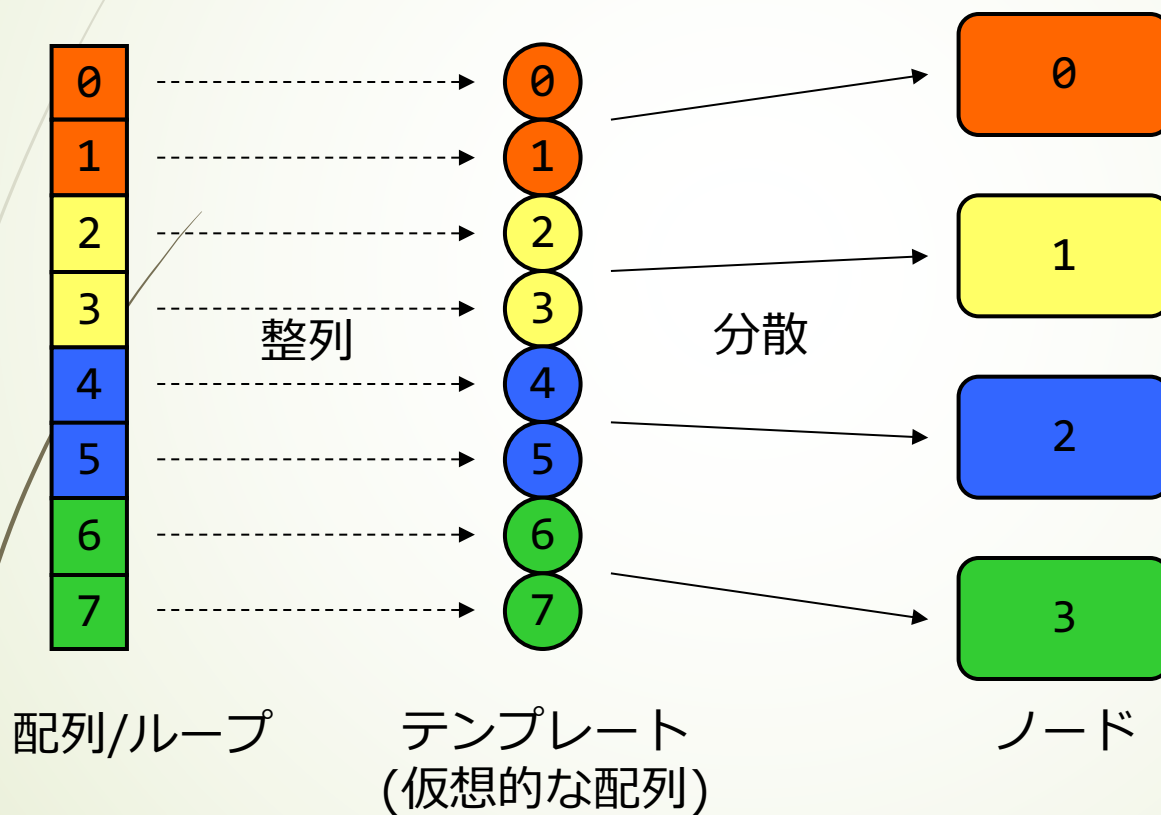
例

```
[C] #pragma xmp align a[i] with t[i]
```

```
[F] !$xmp align a(i) with t(i)
```

# XMPのデータマッピング

## ➡ 整列 + 分散による2段階の処理



- 配列はテンプレートに整列され、
- テンプレートはノードに分散される。

# データマッピング指示文 (1)

## nodes指示文

- ▶ XMPプログラムの実行者である「ノード」のサイズと形状を宣言する。
  - ▶ データやワークを割り当てる対象。
  - ▶ プロセッサ（マルチコア可）とローカルメモリから成る。

[C] `#pragma xmp nodes p[4][4]`

[F] `!$xmp nodes p(4,4)`

# 動的なnodes指示文

- ➡ 実際のpのサイズは、実行時に決まる。
  - ➡ mpiexec コマンドの引数など。

```
[C] #pragma xmp nodes p[*]  
#pragma xmp nodes p[*][4]
```

```
[F] !$xmp nodes p(*)  
!$xmp nodes p(4,*)
```

最後の次元に「\*」を指定できる。

# データマッピング指示文 (2)

## template指示文

- ▶ XMPプログラムの並列処理の基準である「テンプレート」のサイズと形状を宣言する。
  - ▶ データやワークの整列の対象。

```
[C] #pragma xmp template t[64][64]
```

```
[F] !$xmp template t(64,64)
```

# データマッピング指示文 (3)

## distribute指示文

- ノード集合pに、テンプレートtを分散する。

```
[C] #pragma xmp distribute t[block] onto p
```

```
[F] !$xmp distribute t(block) onto p
```

- 分散形式として、ブロック、サイクリック、ブロックサイクリック、不均等ブロックを指定できる。

# データマッピングの例

## 例1: ブロック分散

```
#pragma xmp nodes p[4]
#pragma xmp template t[20]
#pragma xmp distribute t[block] onto p
```



ノード	インデックス
P[0]	0, 1, 2, 3, 4
P[1]	5, 6, 7, 8, 9
P[2]	10, 11, 12, 13, 14
P[3]	15, 16, 17, 18, 19

## 例2: サイクリック分散

```
#pragma xmp nodes p[4]
#pragma xmp template t[20]
#pragma xmp distribute t[cyclic] onto p
```

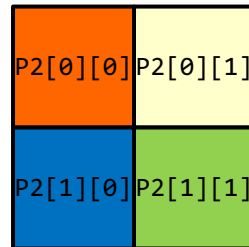


ノード	インデックス
P[0]	0, 4, 8, 12, 16
P[1]	1, 5, 9, 13, 17
P[2]	2, 6, 10, 14, 18
P[3]	3, 7, 11, 15, 19

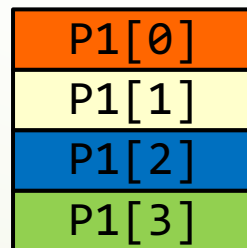


# 多次元テンプレートの分散

```
#pragma xmp nodes p2[2][2]  
#pragma xmp distribute t[block][block] onto p2
```



```
#pragma xmp nodes p1[4]  
#pragma xmp distribute t[block][*] onto p1
```



「\*」は非分散を意味する。

# データマッピング指示文 (4)

## align指示文 (1)

- ➡ 配列aの要素iを、テンプレートtの要素i-1に整列させる。

```
[C] #pragma xmp align a[i] with t[i-1]
```

```
[F] !$xmp align a(i) with t(i-1)
```

- ➡ 多次元配列も整列可能。

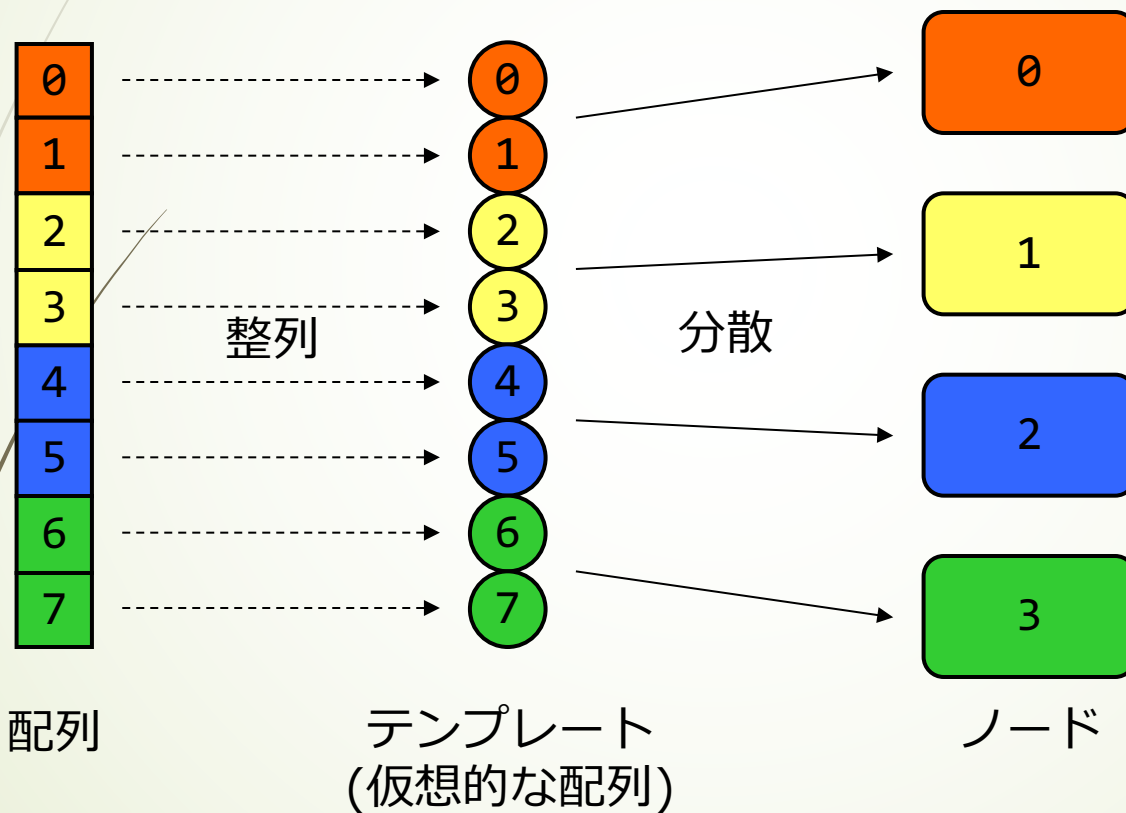
```
[C] #pragma xmp align a[i][j] with t[i-1][j]
```

```
[F] !$xmp align a(i,j) with t(i-1,j)
```

# データマ

```
#pragma xmp nodes p[4]
#pragma xmp template t[8]
#pragma xmp distribute t[block] onto p
float a[8];
#pragma xmp align a[i] with t[i]
```

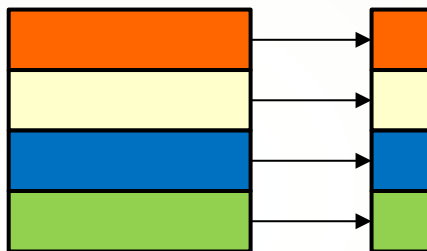
## ➡ 整列 + 分散による



# 特殊な整列

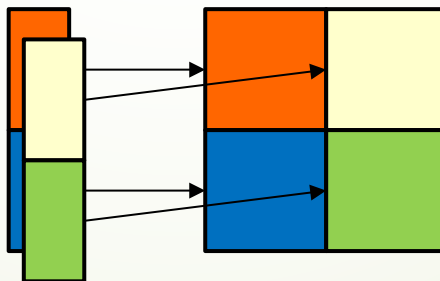
## 縮退

```
#pragma xmp distribute t[block] onto p1
#pragma xmp align a[i][*] with t[i]
```



## 複製

```
#pragma xmp distribute t[block][block] onto p2
#pragma xmp align a[i] with t[i][*]
```



a[0]の実体は、  
p2[0][0]とp2[0][1]  
に存在する。値の一致  
は保証されない。

# 動的な配列の整列

- ▶ ポインタまたは割付け配列として宣言。
- ▶ 実際の「整列」の処理は、続く `xmp_malloc` または `allocate` 文において実行される。

```
[C] int *a;  
    #pragma xmp align a[i] with t[i]  
    ...  
    a = (int *)xmp_malloc(xmp_desc_of(a), 100);
```

```
[F] integer, allocatable :: a(:)  
    !$xmp align a(i) with t(i)  
    ...  
    allocate (a(100))
```

# ワークマッピング指示文 (1)

## loop指示文 (1)

- ▶ ループの並列化を指示する。
  - ▶  $t[i][j]$ を持つノードが、繰り返し $i, j$ において $a[i][j]$ への代入を実行する。

```
#pragma xmp loop (i,j) on t[i][j]
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[i][j] = ...;
```

## loop指示文（2）

- ▶ アクセスされるデータが、その繰り返しを実行するノードに割り当てられていなければならない。
  - ▶ 下の例では、`t[i][j]`を持つノードが、`a[i][j]`を持たなければならない。
  - ▶ そうでない場合、事前に通信を行っておく。

```
#pragma xmp loop (i,j) on t[i][j]
for (i = 0; i < n; i++)
  for (j = 0; j < n; j++)
    a[i][j] = ...;
```

# loop指示文 (3)

## ▶ reduction節

- ▶ 並列ループの終了時に、各ノードの値を「集計」する。
- ▶ 提供している演算は+, max, minなど。

```
#pragma xmp loop on t[i] reduction(+:sum)
  for (i = 0; i < 20; i++)
    sum += i;
```

各ノード上のsumの値を合計した値で、  
各ノード上のsumを更新する。



# ワークマッピング指示文 (2)

## task指示文

- ➡ 直後の処理を、指定したノードが実行する。

```
#pragma xmp task on p[0]
```

```
{  
    func_a();  
}
```

P[0]がfunc\_aを実行する。

```
#pragma xmp task on p[1]
```

```
{  
    func_b();  
}
```

p[1]がfunc\_bを実行する。

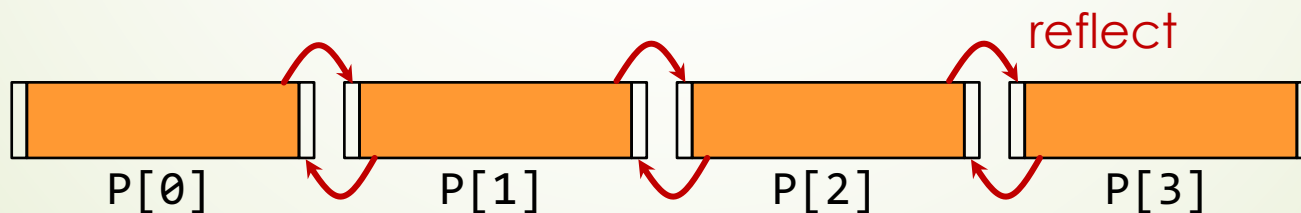
# 通信指示文 (1)

## shadow/reflect指示文

aの上下端に幅1のシャドウを付加する。

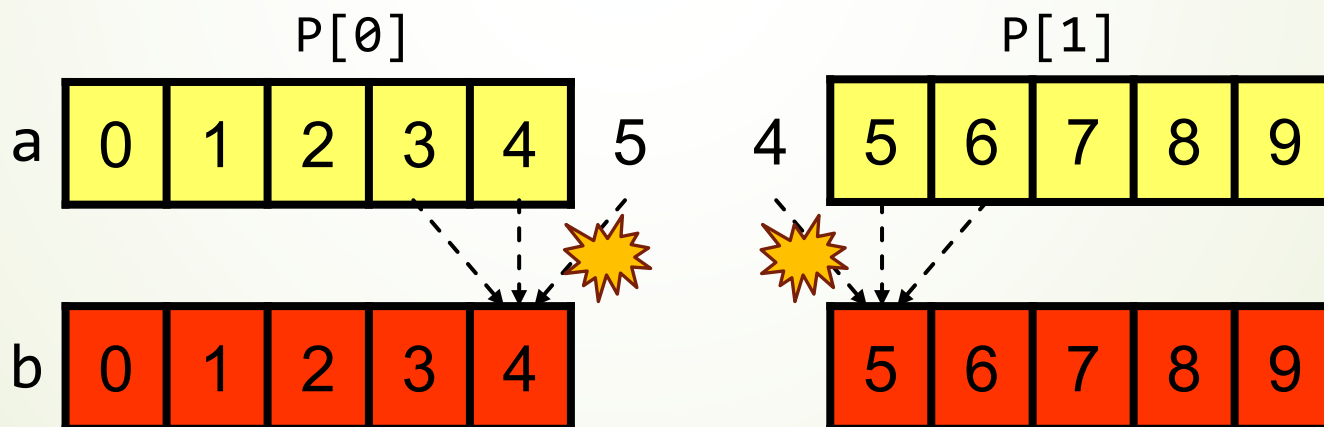
```
#pragma xmp distribute t[block] onto p
#pragma xmp align a[i] with t[i]
#pragma xmp shadow a[1:1]
...
#pragma xmp reflect (a)
```

aに対する隣接通信を実行する。



# shadow/reflect指示文の例

```
#pragma xmp loop on t[i]  
for (i = 1; i < 9; i++)  
    b[i] = a[i-1] + a[i] + a[i+1];
```



# shadow/reflect指示文の例

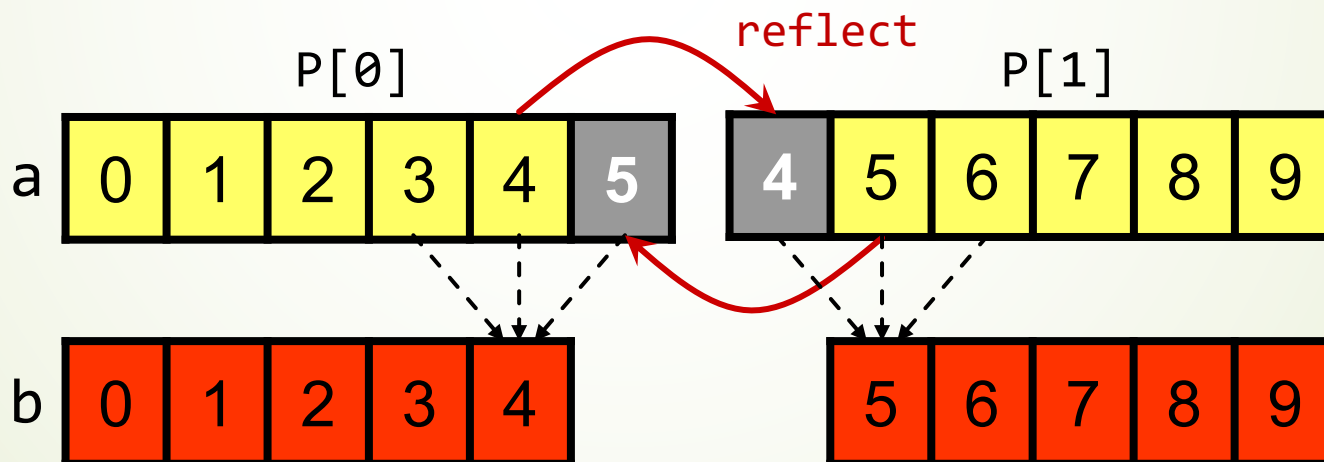
```
#pragma xmp shadow a[1:1]
```

```
#pragma xmp reflect (a)
```

```
#pragma xmp loop on t[i]
```

```
for (i = 1; i < 9; i++)
```

```
    b[i] = a[i-1] + a[i] + a[i+1];
```



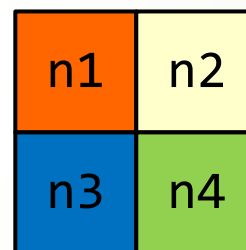
## 通信指示文 (2)

### gmove指示文

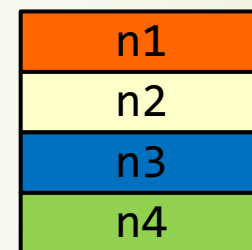
- 通信を伴う任意の代入文を実行する。

```
#pragma xmp gmove  
a[:, :] = b[:, :];
```

※ Cで「部分配列」も記述できる。



$a[\text{block}][\text{block}]$



$b[\text{block}][*]$

# 通信指示文（3）

## ▶ bcast指示文

- ▶ 特定のノードが、指定したデータを他のノードへブロードキャストする（ばらまく）。

```
#pragma xmp bcast (s) from p[0]
```

※ from p[0]  
は省略可

## ▶ barrier指示文

- ▶ ノードが互いに待ち合わせる（バリア同期）。

```
#pragma xmp barrier
```

# XcalableMPプログラムの例

```
!$xmp nodes p(npx,npy,npz)
```

```
!$xmp template (lx,ly,lz) :: t
```

```
!$xmp distribute (block,block,block) onto p :: t
```

```
!$xmp align (ix,iy,iz) with t(ix,iy,iz) ::
```

```
!$xmp&      sr, se, sm, sp, sn, sl, ...
```

```
!$xmp shadow (1,1,1) ::
```

```
!$xmp&      sr, se, sm, sp, sn, sl, ...
```

```
lx = 1024
```

```
!$xmp reflect (sr, sm, sp, se, sn, sl)
```

```
!$xmp loop on t(ix,iy,iz)
```

```
do iz = 1, lz-1
```

```
do iy = 1, ly
```

```
do ix = 1, lx
```

```
  wu0 = sm(ix,iy,iz ) / sr(ix,iy,iz )
```

```
  wu1 = sm(ix,iy,iz+1) / sr(ix,iy,iz+1)
```

```
  wv0 = sn(ix,iy,iz ) / sr(ix,iy,iz )
```

```
  ...
```

ノード集合の宣言

テンプレートの宣言と  
分散の指定

整列の指定

シャドウの指定

重複実行される

隣接通信の指定

ループの並列化の指定

# XMPのローカルビュー・プログラミング

- ▶ 各ノードが解くべき問題を個別に示す。
  - ▶ 「ノード1は問題1~25を解け。ノード2は.....」
- ▶ 自由度が高いが、やや難しい。
- ▶ ローカルビューのための機能として、Fortran 2008から導入したcoarrayをサポート。



# coarray機能

- ▶ 「coarray」として宣言されたデータは、代入文の形式で他のノードからもアクセスできる。

ノード2が持つb[3:3]のデータをa[0:3]に代入

```
float b[100]:[*];
```

```
if (xmpc_this_image() == 0)  
    a[0:3] = b[3:3]:[1];
```

配列bはcoarrayであると宣言

コロンの後の[]はノード番号を表す

base    length 「0からの3要素」