XcalableMP講習会
～XcalableMPの概要～

理化学研究所　計算科学研究機構

石原誠

目次

- はじめに
- 並列プログラミング言語XcalableMP(XMP)
- XMPのキホンのキ
- XMPの文法解説
- まとめ

はじめに

- 大規模並列計算させたい時どんな計算機で行うか想像してみてください
  - 分散(共有)メモリ型の計算機を思い浮かべることでしょう・・・
- では分散メモリ型の計算機ではどうやって並列計算を実現しますか？
  - MPIが主流ではないでしょうか・・・
- ではMPIで高速な大規模プログラムが手早く書ける自信がありますか？
  - 首を傾げることでしょう・・・なぜならプログラムがスパゲッティコードと化すから
- そこでXcalebleMP(XMP)を使うのです!!

並列プログラミング言語XcalableMP (XMP)

- 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規格部会で検討中のMPIに代わる並列プログラミングモデル
- 目標:
  - Performance
  - Expressiveness
  - Optimizability
  - Education cost

XcalableMP

www.xcalablemp.org

XMPのキホンのキ

- XMPの特徴
- XMPの実行モデル
  - グローバルビューとローカルビュー
- XMPのメモリモデル
- XMPのお約束ごと
- XMPの記法例

XMPの特徴

- C/Fortranの拡張である
  - ディレクティブベースの並列化指示：無視すればただの逐次プログラム
  - 指示文によって並列化・同期・通信を明示的に記載できる
- ふたつのプログラミングモデルからなる
  - グローバルビュー
  - ローカルビュー

グローバルビューとローカルビュー

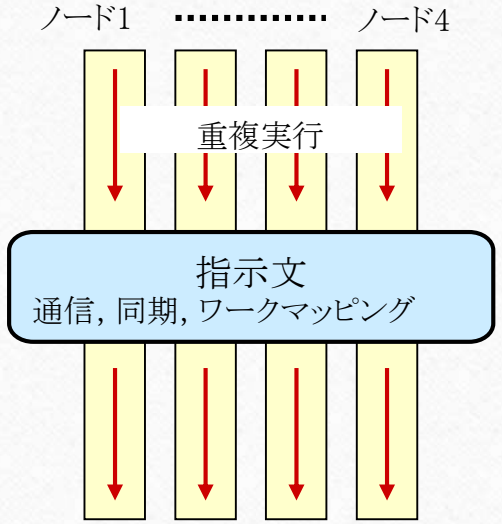
例題：1から100までの総和を4ノード使用して解け

- グローバルビューの場合
  - 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
    - 指示文ひとつで動作可能
- 命令方法：
  1から100までの総和を4ノードで分担して解け

- ローカルビューの場合
  - 各ノードが解くべき問題を個別に示す
    - 自由度は高いがテクニックが必要
- 命令方法：
  ノード1は1から25の総和を解け
  ノード2は26から50の…(以下略)

XMPの実行モデル

- SPMD(Single Program, Multiple Data)モデル
- 各ノードは同一のコードを独立に(重複して)実行
- 指示文の箇所では、全ノードが協調して動作(集団実行)
  - 通信・同期
  - ワークマッピング(並列処理)



XMPのメモリモデル

- 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)のみアクセス
- 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 (:Fortran2008から導入された記法)
- 「分散」されないデータは全ノードに重複して配置

XMPのお約束ごと

- XMPでは並列化などの通信指示文は以下のように書き始める
  #pragma xmp　…から始まるプリプロセス(C言語版)
  !$xmp …から始まる特殊なコメント文(Fortran自由形式版)
  C$xmp …から始まる特殊なコメント文(Fortran固定形式版)
- Coarrayの書き方は
  A[*配列の領域*]:[*目的ノード番号*](C言語版)
  A(*配列の領域*)[*目的ノード番号*](Fortran版)
  と書く

プログラム例（MPIとの比較）

```
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( );
}
```

MPIプログラム

```
int array[MAX];
#pragma xmp nodes p(*)
#pragma xmp template t(0:MAX-1)
#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];
   }
}
```

XMP/Cプログラム

XMPの文法解説(1)

- XMPにおけるプログラムの書き方はふたつに大別できる
  - グローバルビュー
    - 並列・同期・分散をできるだけシンプルな指示文によって実装できる仕組み
  - ローカルビュー(Coarray)
    - 代入文の形式で他のノードのデータを読み/書きできる仕組み

XMPの文法解説(2)

注意:CoarrayにおいてCとFortranでは記法が似ているが意味が違う

- Cの場合

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

if (xmp_node_num() == 1)
  a[0:3] = b[3:3]:[2];
```

配列bはCoarrayであると宣言するのは共通(Cには:が必要)

base　length
「0からの3要素」

コロンの後の[]はノード番号を表す

解釈:ノード2が持つbの3から3要素分のデータをaの0から3要素分に代入

- Fortranの場合

```
real b(100)[*]

if (xmp_node_num().eq.1)
  a(1:3) = b(3:5)[2]
```

lower bound　upper bound
「1からの3まで」

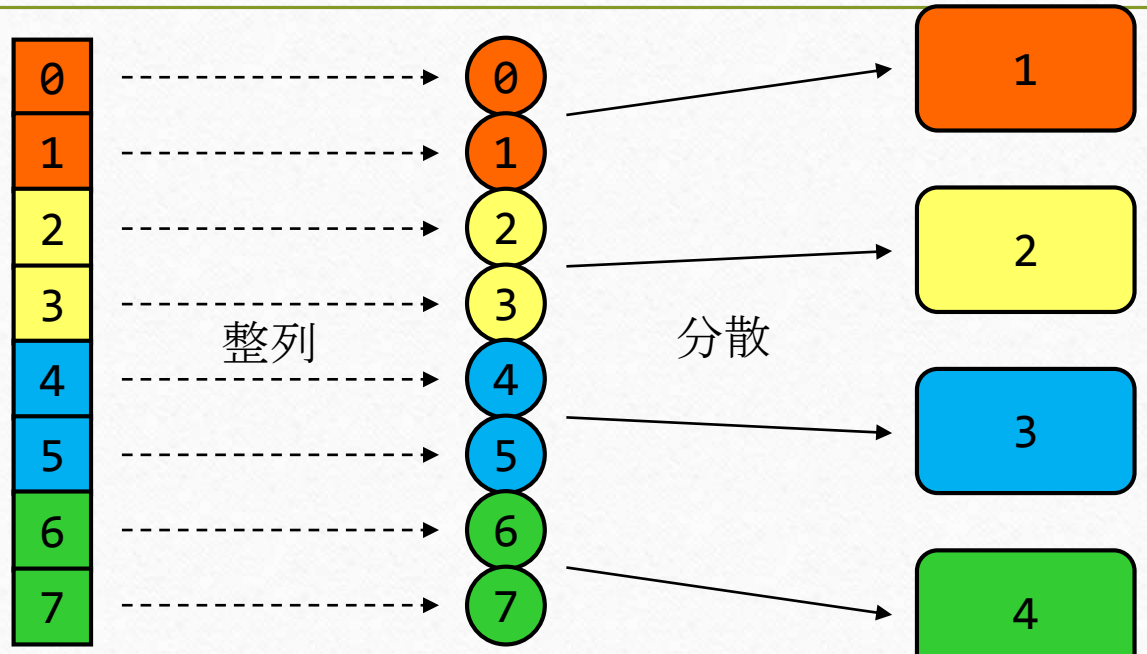
[]はノード番号を表す

解釈:ノード2が持つbの3から5番目の要素のデータをaの1から3番目の要素に代入

グローバルビューでの並列化命令(指示文)

- データマッピング
  - nodes指示文
  - template指示文
  - distribute指示文
  - align指示文
- ワークマッピング
  - loop指示文
  - task指示文
- 通信・同期
  - shadow/reflect指示文
  - gmove指示文
  - bcast指示文
  - barrier指示文

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
  - 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
  - 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

[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(0:19)
#pragma xmp distribute t(block) onto p
```

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

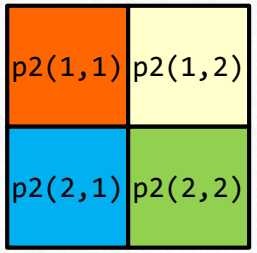
例2: サイクリック分散

```
#pragma xmp nodes p(4)
#pragma xmp template t(0:19)
#pragma xmp distribute t(cyclic) onto p
```

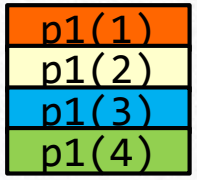
ノード	インデックス
p(1)	0, 4, 8, 12, 16
p(2)	1, 5, 9, 13, 17
p(3)	2, 6, 10, 14, 18
p(4)	3, 7, 11, 15, 19

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に整列させる

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

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

- 多次元配列も整列可能

[C]
```
#pragma xmp align a[i][j] with t(i,j)
```

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

データマッピング指示文(4)
align指示文(2)

- 配列aの要素iを、テンプレートtの要素i+1に整列可能

[C]
```
#pragma xmp align a[i] with t(i+1)
```

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

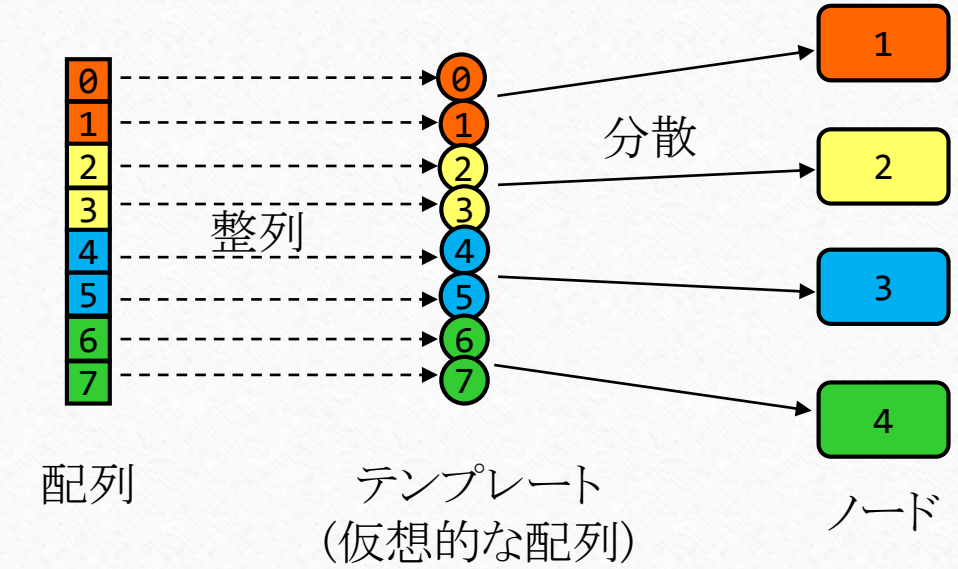
- 従って多次元配列でも同様に整列可能

[C]
```
#pragma xmp align a[i][j] with t(i+1,j)
```

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

データマッピング

- 整列 + 分散による2段階の処理




```
#pragma xmp nodes p(4)
#pragma xmp template t(0:7)
#pragma xmp distribute t(block) onto p
float a[8];
#pragma xmp align a[i] with t(i)
```

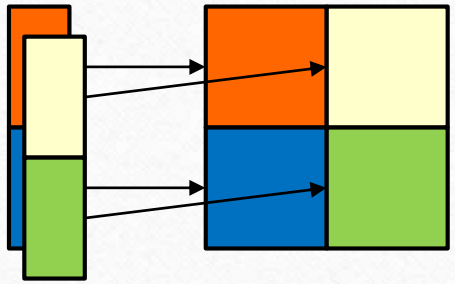
特殊な整列

- 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(1,1)とp2(1,2)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[j] with t(j)
...
a = (int *)xmp_malloc(xmp_desc_of(a), 100);
```

[F]
```
integer, allocatable :: a(:)
!$xmp align a(j) with t(j)
...
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」
  - 提供している演算は+, max, minなど

```
#pragma xmp loop (i) 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

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

```c
#pragma xmp task on p(1)
{
   func_a();
}

#pragma xmp task on p(2)
{
   func_b();
}
```

C

p(1)がfunc_aを実行する。

p(2)がfunc_bを実行する。

```fortran
!$xmp task on p(1)
   func_a
!$xmp end task

!$xmp task on p(2)
   func_b
!$xmp end task
```

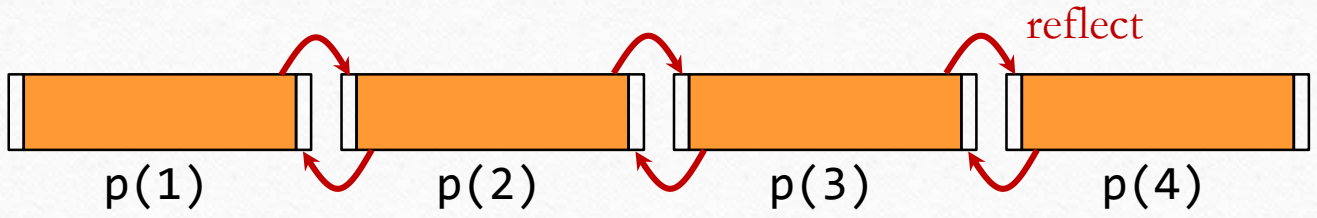
Fortran

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

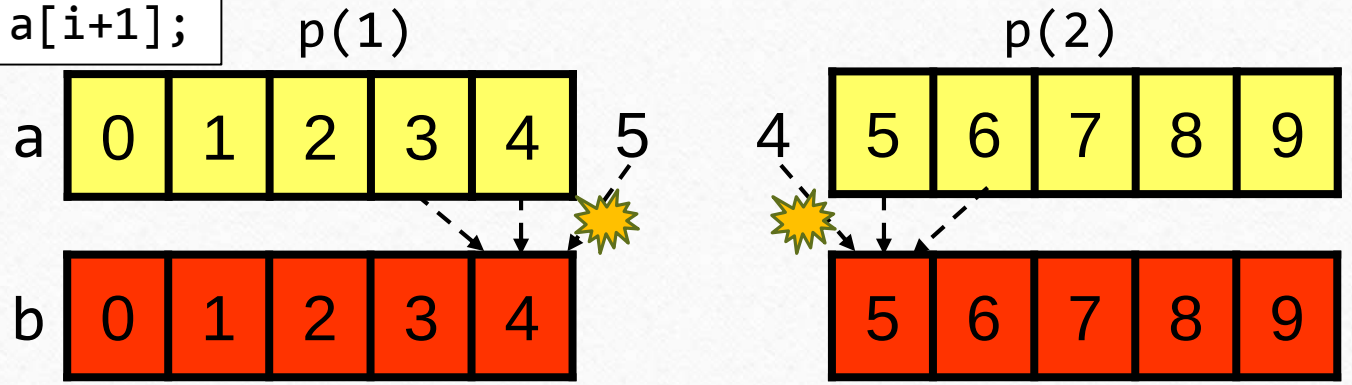
```
#pragma xmp distribute t(block) onto p
#pragma xmp align a[i] with t(i-1)
#pragma xmp shadow a[1:1]
...
#pragma xmp reflect (a)
```

aに対する隣接通信を実行



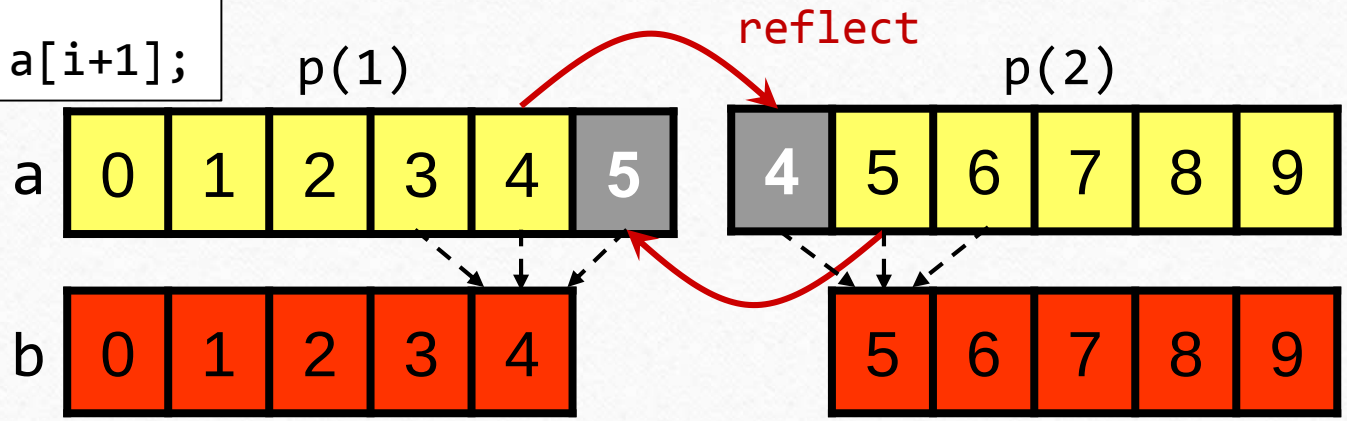
shadow/reflect指示文の例

```
#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

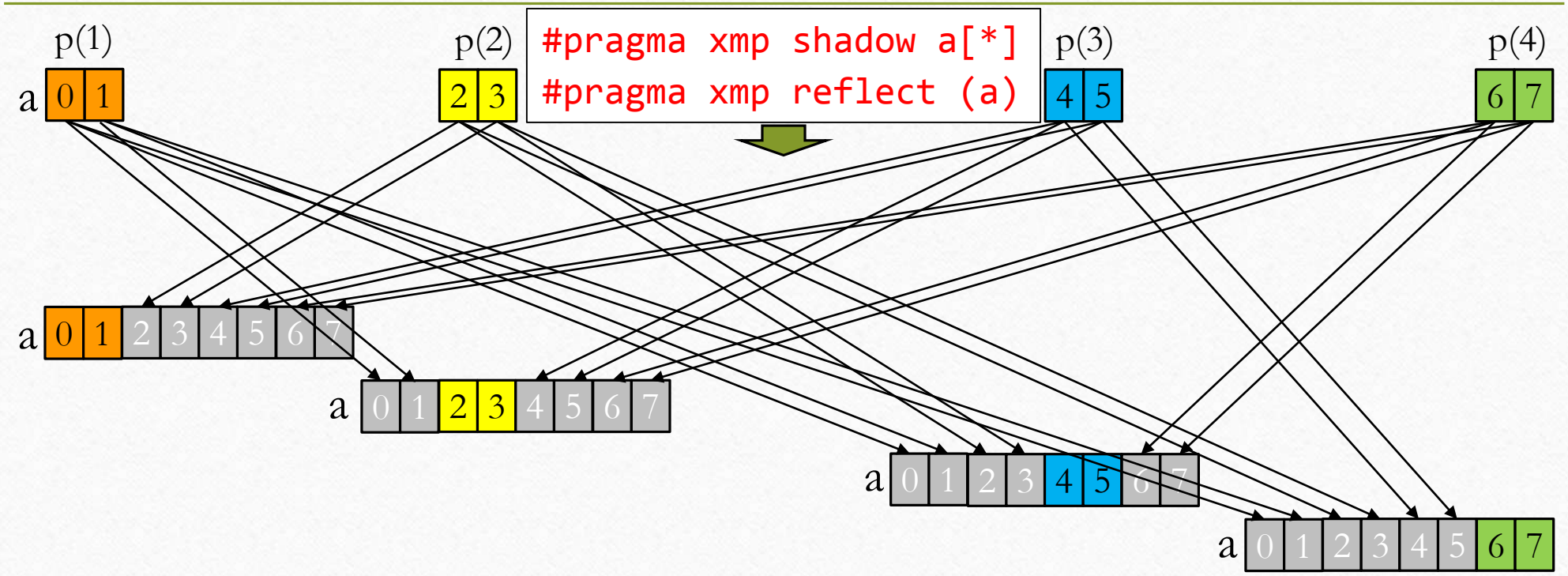
```
#pragma xmp shadow a[1:1]

#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];
```



full shadowという考え方

- 各ノードが持っているローカルデータの計算に、他のノードが持っている全てのデータも必要となったとき
  - full shadow という考え方を導入する
- どういうときに有効か？
  - 計算オーダーが通信オーダーよりはるかに大きいとき

full shadowという考え方

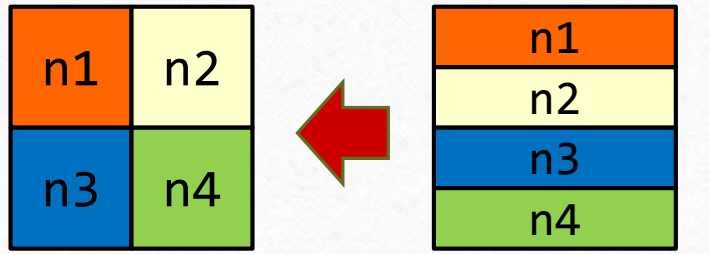


通信指示文(2)
gmove指示文

- 通信を伴う任意の代入文を実行

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

※ Cで「部分配列」も記述できる。



a[block][block]　　b[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(1)
```

※ from p(1)
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

まとめ

- XcalableMPは
  - 指示文を用いて十分な性能を発揮するプログラミングモデル
    - 可読性と可搬性の高いプログラムを生成する
    - インクリメンタルなプログラムの並列化を可能にする
  - Coarrayを用いてローカルビューのプログラミングを実現した
    - より緻密なプログラムをより容易に作成することを可能にする
- ぜひ実習でその使いやすさを体験してください