XMPによる並列化実装 2

中尾 昌広（理化学研究所 計算科学研究機構）

目次

- グローバルビュープログラミング
- ローカルビュープログラミング（時間があれば）

用いるコード

./xmp-sample/global-view

		C	Fortran
Exercise 1	Serial	init.c	init.f90
	XMP	xmp_init.c	xmp_init.f90
Exercise 2	Serial	laplace.c	laplace.f90
	XMP	xmp_laplace.c	xmp_laplace.f90

課題1

- 分散配列a[]を定義し，ループ文を2ノードで並列実行させよ

[C] init.c

```
#include <stdio.h>
int a[10];

int main(){

  for(int i=0;i<10;i++)
    a[i] = i+1;

  for(int i=0;i<10;i++)
    printf("%d¥n", a[i]);

  return 0;
}
```

[F] init.f90

```
program init
  integer :: a(10), i

  do i=1,10
    a(i)=i
  end do

  do i=1,10
    print *, a(i)
  end do

end program init
```

課題1

下記のファイルは部分的に並列化しています

- XMP指示文をxmp_init.c or xmp_init.f90に挿入してください

[C] xmp_init.c

```
#pragma xmp nodes p[2]
#pragma xmp template t[10]
#pragma xmp distribute t[block] onto p
int a[10];
[align directive]
int main(){
[loop directive]
  for(int i=0;i<10;i++)
    a[i] = i+1;

[loop directive]
  for(int i=0;i<10;i++)
    printf(”[%d] %d¥n", xmpc_node_num(), a[i]);
return 0;
}
```

[F] xmp_init.f90

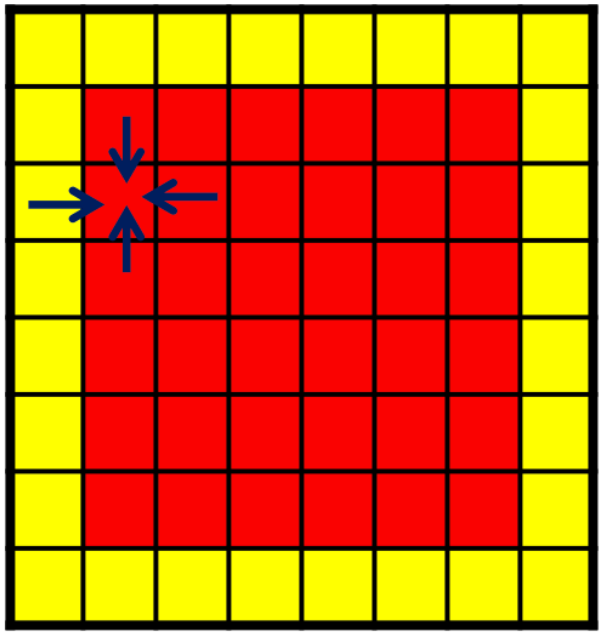
```
program init
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p
  integer :: a(10), i
[align directive]

[loop directive]
  do i=1,10
    a(i)=i
  end do
[loop directive]
  do i=1,10
    print *, xmp_node_num(), a(i)
  end do
end program init
```

実行手順

- コンパイル
  - [C] $ xmpcc xmp_init_c -o xmp_init
  - [F] $ xmpf90 xmp_init.f90 -o xmp_init
- 2ノードで実行

課題2：ラプラス関数



初期値は0

初期値は下記の通り

[C] sin((double)j/N2*M_PI)+cos((double)i/N1*M_PI)
[F] sin(dble(i-1)/N1*PI)+cos(dble(j-1)/N2*PI)

```
for(j = 1; j < N2-1; j++)
  for(i = 1; i < N1-1; i++)
    u[j][i] = (uu[j-1][i] + uu[j+1][i] + uu[j][i-1] + uu[j][i+1])/4.0;
```

赤のセルは上下左右のセルの値を使って更新される.

逐次コード

- まずはlaplace.c or laplace.f90をエディタで開いてください
- 下記のようにコンパイルしてください
  - [C] $ gcc laplace.c -lm
  - [F] $ gfortran laplace.f90
- フロントエンドノードで実行
  - $ ./a.out
  - Verification =　5.54885...

並列化のパターン

- **[Pattern 1]** 両辺で配列のインデックスが同じ場合
  - [C] u[j][i] = uu[j][i];
  - [F] u(i,j) = u(i,j)

[C]

```
#pragma xmp loop (j,i) on t[j][i]
for(int j=0;j<10;j++)
  for(int i=0;i<10;i++)
    u[j][i] = uu[j][i];
```

[F]

```
!$xmp loop (i,j) on t(i,j)
do j=1,10
  do i=1,10
    u(i,j) = u(i,j)
  end do
end do
```

並列化のパターン

- **[Pattern 2]** 隣接のセルを参照する場合
  - [C] u[j][i] = uu[j-1][i] + uu[j][i-1] + ...;
  - [F] u(i,j) = uu(i,j-1) + uu(i-1,j) + ...

shadow指示文を使って，袖領域を分散配列に追加する

reflect指示文を使って，ループ文の前で袖領域の更新を行う

[C]

```
#pragma xmp shadow uu[1:1][1:1]
   :
#pragma xmp reflect (uu)
   :
#pragma xmp loop (j,i) on t[j][i]
for(int j=0;j<10;j++)
  for(int i=0;i<10;i++)
    u[j][i] = uu[j-1][i] + uu[j][i-1] + ...
```

[F]

```
!$xmp shadow uu(1:1,1:1)
  :
!$xmp reflect (uu)
  :
!$xmp loop (i,j) on t(i,j)
do j=1,10
  do i=1,10
    u(i,j) = u(i,j-1) + uu(i-1,j) +
  end do
end do
```

並列化のパターン

- **[Pattern 3]** reductionが必要な場合
  - [C] s += abs(uu[j][i]-u[j][i]);
  - [F] s = s + abs(uu(i,j)-u(i,j))

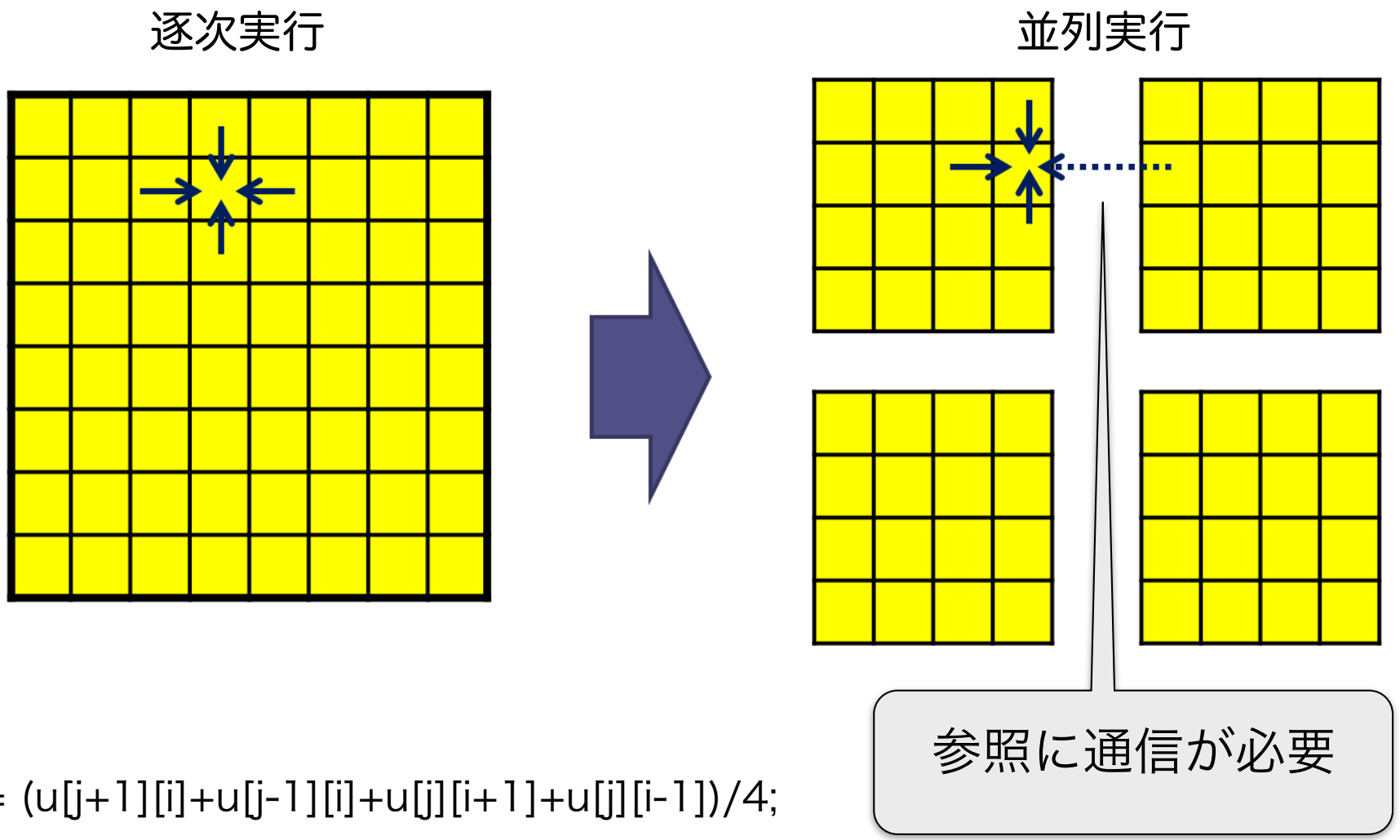
[C]

```
#pragma xmp loop (j,i) on t[j][i] reduction (+:s)
for(int j=0;j<10;j++)
  for(int i=0;i<10;i++)
    s += abs(uu[j][i] - u[j][i]);
```

[F]

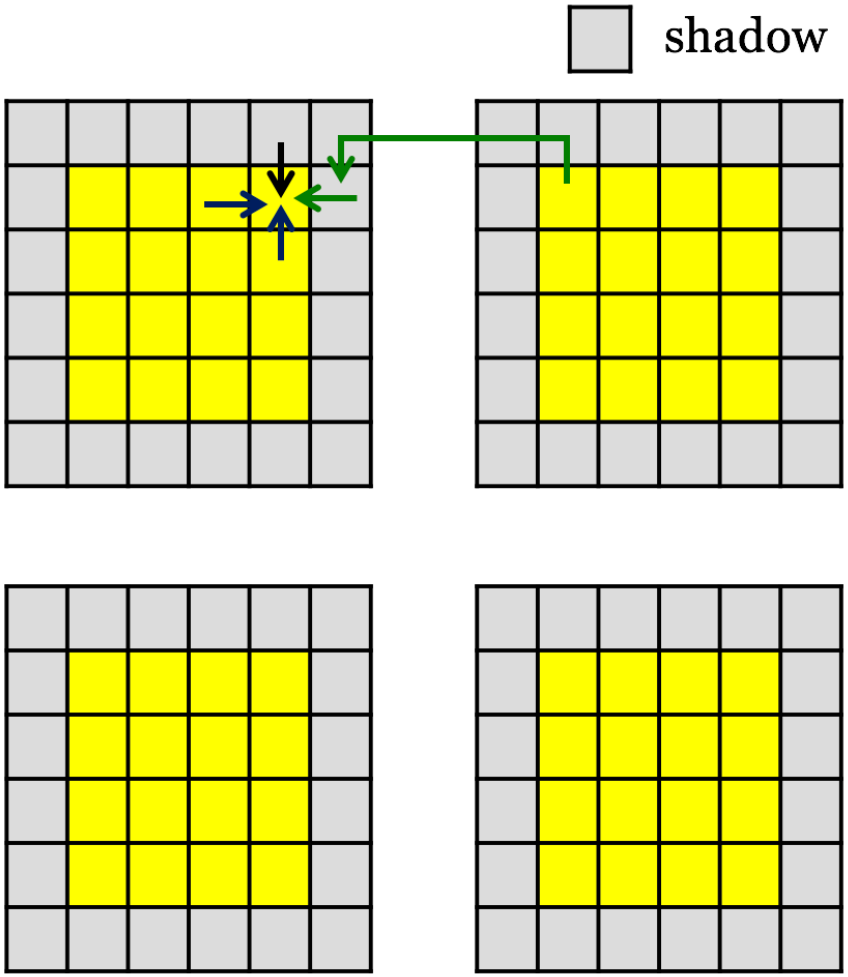
```
!$xmp loop (i,j) on t(i,j) reduction(+:s)
do j=1,10
  do i=1,10
    s = s + abs(uu(i,j) - u(i,j))
  end do
end do
```

リモートデータの参照




```
u[j][i] = (u[j+1][i]+u[j-1][i]+u[j][i+1]+u[j][i-1])/4;
```

リモートデータの参照



1. shadow指示文は袖領域を分散配列に追加する
2. reflect指示文は，隣接ノードから分散配列の端の値を，自ノードの袖領域にコピーする
3. loop指示文は，隣接セルの参照を含むfor文を処理する

課題2

- xmp_laplace.c or xmp_laplace.f90を元に，XMPで並列化せよ
  - 2次元ノード集合と2次元テンプレートを用いる
  - 4ノードと8ノードで実行させる
  - 逐次バージョンの結果と比べる

XMP version Laplace equation

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define N1 64
#define N2 64
double u[N2][N1],uu[N2][N1];

#pragma xmp nodes p[*][4]
#pragma xmp template t[N2][N1]
[distribute directive]
[align directive]
[align directive]
[shadow directive]

int main(int argc, char **argv)
{
   int j,i,k,niter = 100;
   double value = 0.0;

#pragma xmp loop (j,i) on t[j][i]
   for(j = 0; j < N2; j++){
      for(i = 0; i < N1; i++){
         u[j][i] = 0.0;
         uu[j][i] = 0.0;
      }
   }
```

pattern 1

```
[loop directive]
   for(j = 1; j < N2-1; j++)
      for(i = 1; i < N1-1; i++)
         u[j][i] = sin((double)i/N1*M_PI)
                 + cos((double)j/N2*M_PI);

   for(k = 0; k < niter; k++){
      /* old <- new */
[loop directive]
      for(j = 1; j < N2-1; j++)
        for(i = 1; i < N1-1; i++)
          uu[j][i] = u[j][i];

[reflect directive]
[loop directive]
      for(j = 1; j < N2-1; j++)
        for(i = 1; i < N1-1; i++)
          u[j][i] = (uu[j-1][i] + uu[j+1][i] +
                  uu[j][i-1] + uu[j][i+1])/4.0;
   }

   /* check value */
   value = 0.0;
#pragma xmp loop (j,i) on t[j][i] [reduction clause]
   for(j = 1; j < N2-1; j++)
      for(i = 1; i < N1-1; i++)
        value += fabs(uu[j][i]-u[j][i]);
```

pattern 1

pattern 2

pattern 3

目次

- グローバルビュープログラミング
- ローカルビュープログラミング（時間があれば）

ローカルビュープログラミング

- Coarray記法について
  - 課題1：スカラー値のPut/Get（実行のみ）
  - 課題2：配列のPut/Get（実行のみ）
  - 課題3：行列積の作成

Coarray記法

- 片側通信 (Put/Get)

array[start:length[:step]]

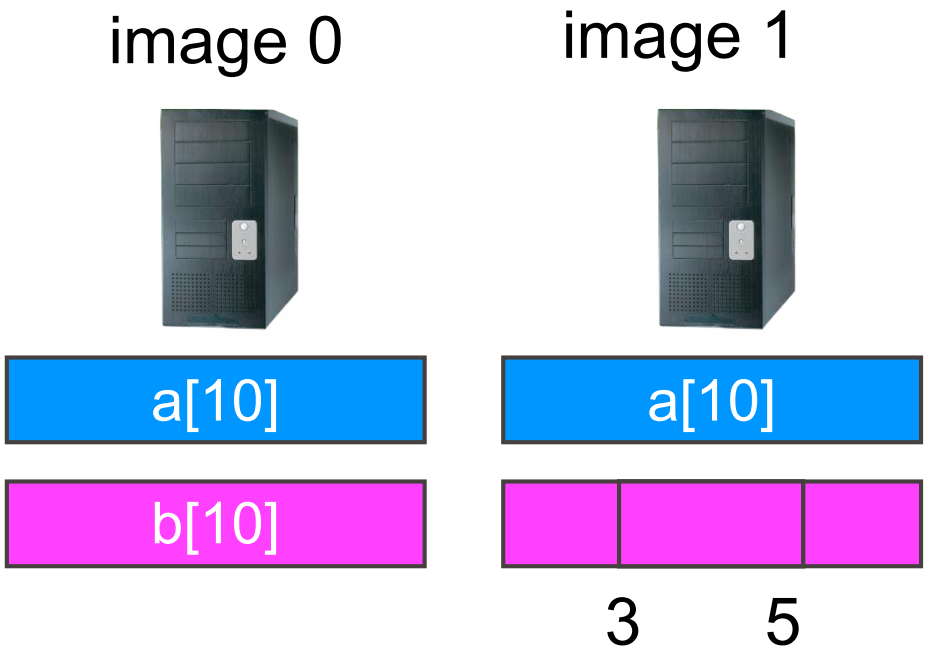
ノード1が持っている b[3], b[4], b[5]
の値が，ノード番号0のa[0], a[1], a[2]
にコピーされる

[C]

```
int a[10];
int b[10]:[*];   // b is a coarray

        :

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



角括弧はノード番号を示す
（Fortran is 1-origin, C is 0-origin）

Array section in XMP/C (1/2)

- シンタックス

  array[base : length :step]

- base, length, stepはint型
- Base:
  - 省略された場合は0になる
- Length :
  - 省略された場合は，残りの要素数になる
- Step :
  - 省略された場合は1になる

Array section in XMP/C (2/2)

int A[100]:[*]のCoarrayが宣言されている場合

A[10:10]	A[10] から A[19]の10要素
A[10:]	A[10] から A[99]の90要素
A[:10]	A[0] から A[9]の10要素
A[10:5:2]	A[10], A[12], A[14], A[16], A[18]の5要素
A[:]	Aの全要素 (A[0] から A[99])

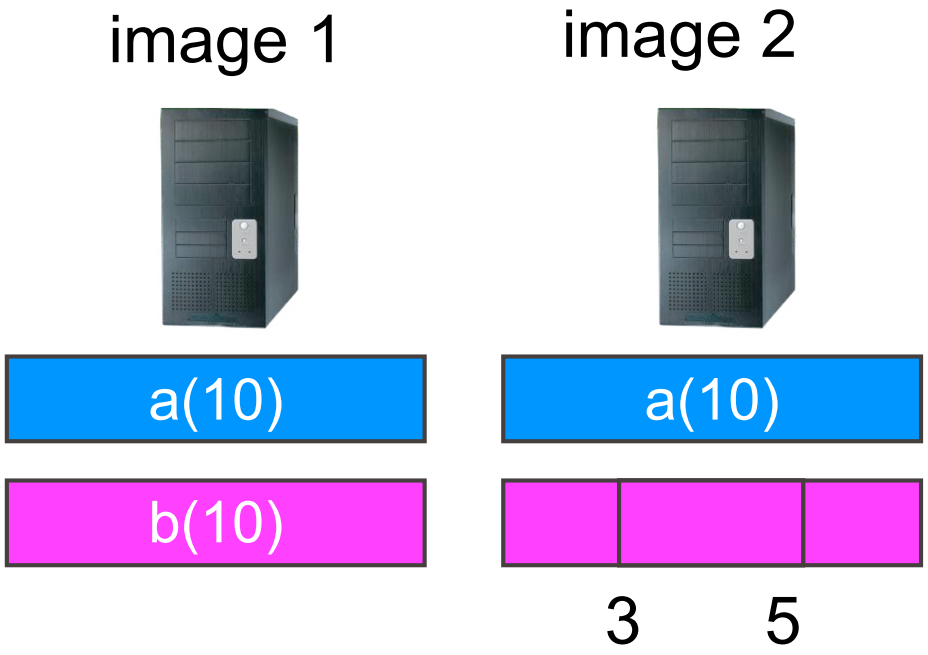
Coarray記法

- 片側通信 (Put/Get)

ノード2が持っているb(3:5)は
ノード1のa(1:3)にコピーされる

[F]
```
integer :: a(10)
integer :: b(10)[*]   // b is a coarray
      :
if(this_image() == 1) then
  a(1:3) = b(3:5)[2]        // Get
```

角括弧はノード番号を示す
（Fortran is 1-origin, C is 0-origin）



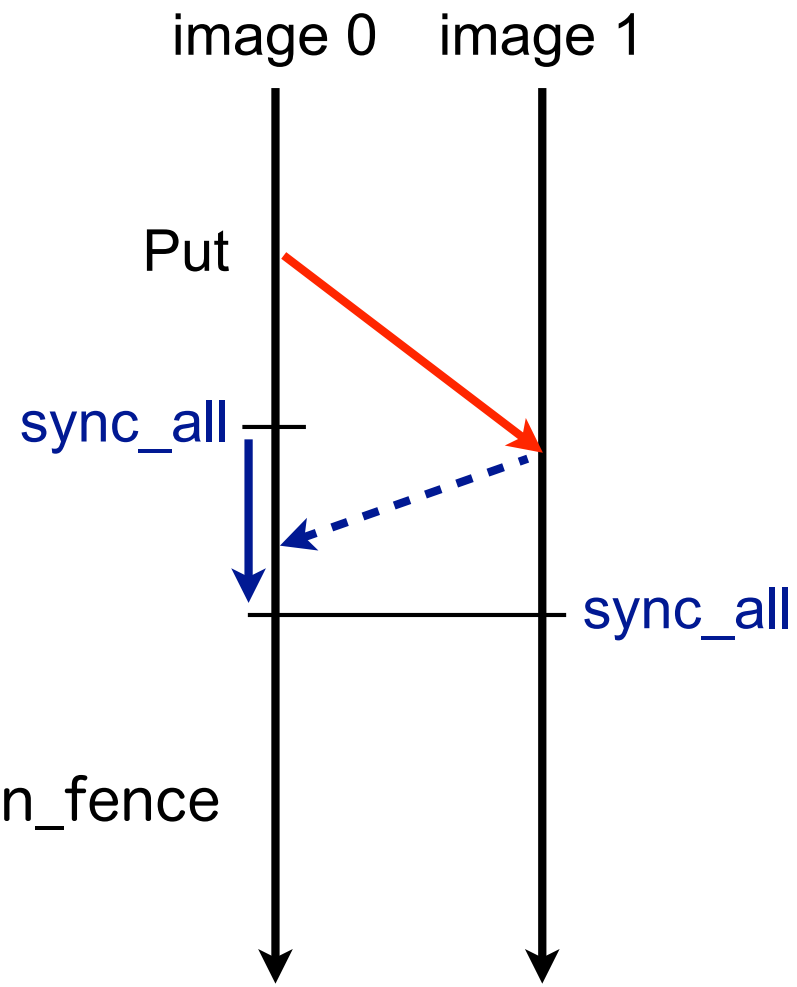
同期

[C]

```
void xmp_sync_all(int *status)
```

[F]

```
sync all
```



これまでに実行した片側通信が完了させ，
かつノード間でバリア同期を行う（MPI_Win_fence
+ MPI_Barrierに相当）

課題1

- スカラ変数に対するPut/Get
  - [C] xmpcc coarray_scalar.c -o coarray_scalar.x
  - [F] xmpf90 coarray_scalar.f90 -o coarray_scalar.x
  - 2プロセスで実行せよ

[C]

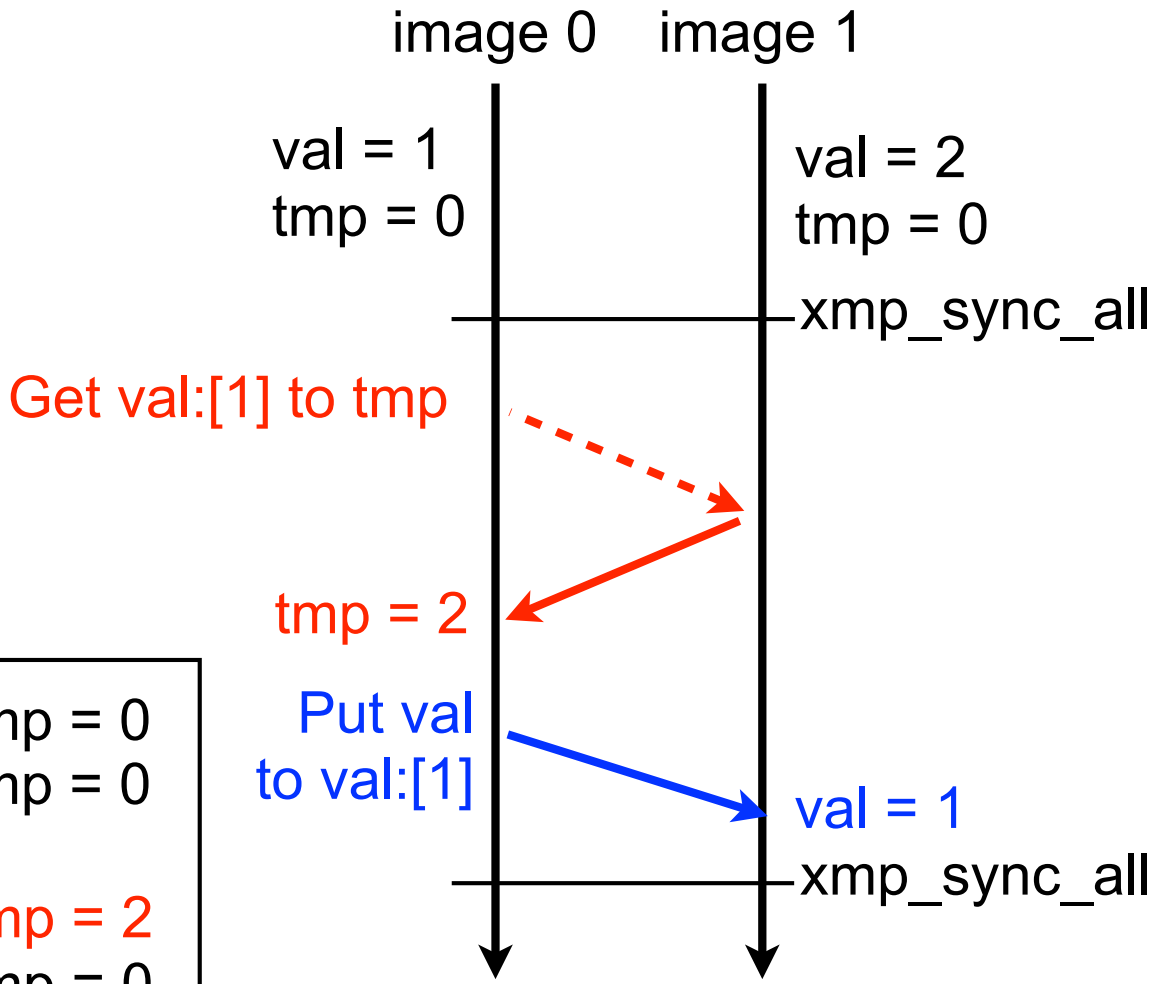
```
if(xmpc_this_image() == 0){
    tmp = val:[1];  // Get
    val:[1] = val;   // Put
 }
xmp_sync_all(NULL);
```

[F]

```
if(this_image() == 1) then
    tmp = val[2]  ! Get
    val[2] = val   ! Put
end if
sync all
```

課題1：結果 (XMP/C)

```
xmp_sync_all(NULL);
if(xmpc_this_image() == 0){
    tmp = val:[1];  // Get
    val:[1] = val;   // Put
  }
xmp_sync_all(NULL);
```



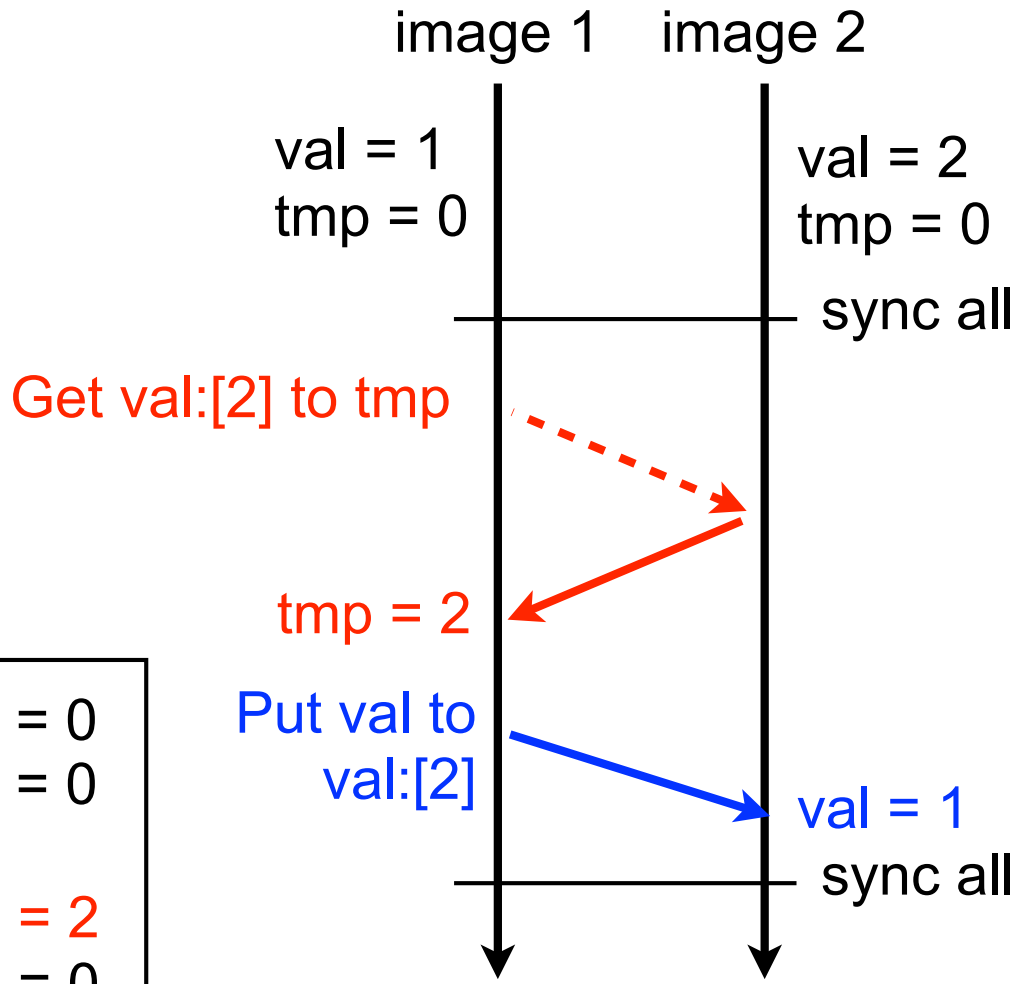
結果

```
[START] My image is 0, val = 1 tmp = 0
[START] My image is 1, val = 2 tmp = 0

[END]    My image is 0, val = 1 tmp = 2
[END]    My image is 1, val = 1 tmp = 0
```

課題1：結果 (XMP/Fortran)

```
sync all
if(this_image() == 1) then
    tmp = val:[2];  // Get
    val:[2] = val;   // Put
end if
sync all
```



結果

```
[START] My image is 1, val = 1 tmp = 0
[START] My image is 2, val = 2 tmp = 0

[END]   My image is 1, val = 1 tmp = 2
[END]   My image is 2, val = 1 tmp = 0
```

課題2

- 配列に対するPut/Get
  - [C] xmpcc coarray_vector.c -o coarray_vector.x
  - [F] xmpf90 coarray_vector.f90 -o coarray_vector.x
  - 2プロセスで実行せよ

[C] ノード 0, [F] ノード 1

```
a[10] = {0, 1, .., 9};
b[10] = {0, 1, .., 9};
c[10][10] = {{0, 1, .., 9},
             {10, 11, .., 19},
             ...
             {90, 91, .., 99}};
```

[C] ノード 1, [F] ノード 2

```
a[10] = {10, 11, .., 19};
b[10] = {10, 11, .., 19};
c[10][10] = {{100, 101, .., 109},
             {110, 111, .., 119},
             ...
             {190, 191, .., 199}};
```

課題２：結果

[C]

```
if(xmpc_this_image() == 0){
    a[0:3] = a[5:3]:[1];   // Get
}
```

[F]

```
if(this_image() == 1) then
    a(1:3) = a(6:8)[2]   ! Get
end if
```

[C]

```
a[0] = 15
a[1] = 16
a[2] = 17
a[3] = 3
a[4] = 4
a[5] = 5
a[6] = 6
a[7] = 7
a[8] = 8
a[9] = 9
```

[F]

```
a(1) = 15
a(2) = 16
a(3) = 17
a(4) = 3
a(5) = 4
a(6) = 5
a(7) = 6
a(8) = 7
a(9) = 8
a(10) = 9
```

課題 2：結果

[C]

```
if(xmpc_this_image() == 0){
    b[0:5:2] = b[0:5:2]:[1];      // Get
}
```

[F]

```
if(this_image() == 1) then
    b(1:10:2) = b(1:10:2)[2]      ! Get
end if
```

[C]

b[0] = 10
b[1] = 1
b[2] = 12
b[3] = 3
b[4] = 14
b[5] = 5
b[6] = 16
b[7] = 7
b[8] = 18
b[9] = 9

[F]

b(1) = 10
b(2) = 1
b(3) = 12
b(4) = 3
b(5) = 14
b(6) = 5
b(7) = 16
b(8) = 7
b(9) = 18
b(10) = 9

課題 2：結果

[C]

```
if(xmpc_this_image() == 0){
    c[0:5][0:5]:[1] = c[0:5][0:5];   // Put
}
```

[F]

```
if(this_image() == 1) then
    c(1:5,1:5)[2] = c(1:5,1:5)   // Put
end if
```

```
  0   1   2   3   4 105 106 107 108 109
 10  11  12  13  14 115  116 117 118 119
 20  21  22  23  24 125 126 127 128 129
 30  31  32  33  34 135 136 137 138 139
 40  41  42  43  44 145 146 147 148 149
150 151 152 153 154 155 156 157 158 159
160 161 162 163 164 165 166 167 168 169
170 171 172 173 174 175 176 177 178 179
180 181 182 183 184 185 186 187 188 189
190 191 192 193 194 195 196 197 198 199
```

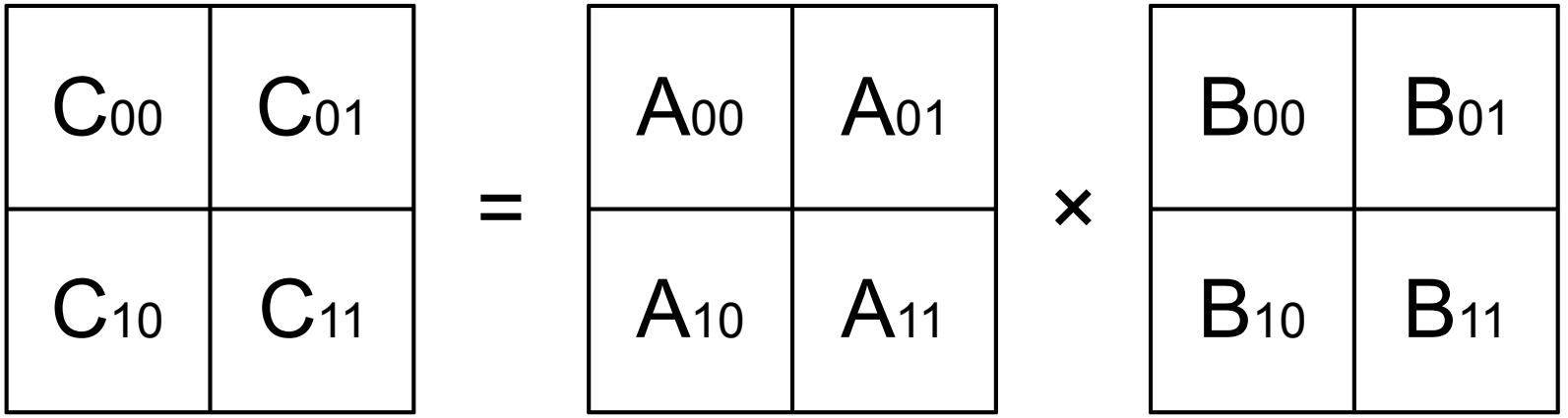
課題 3

- 行列積の並列化
  - C = A x B
- 並列化の手順
  - 4ノードで実行する
  - まず，ノード 0 in C（ノード1 in Fortran）はa[][]とb[][]のすべてのデータを代入する
  - 次に，ノード 0 in C（ノード1 in Fortran）はa[][]とb[][]の一部を他のノードにPutする
  - そして，4ノードは a[][]とb[][]を使ってc[][]の計算を行う
  - 最後に，ノード 0 in C（ノード1 in Fortran）は，他のノードからc[][]の一部を自分のc[][]に集める

```
for(i=0;i<N;i++)
  for(j=0;j<N;j++)
    for(k=0;k<N;k++)
      c[i][k] += a[i][j] * b[j][k];
```

課題 3

- Calculation by sub-matrix



XMP/C　XMP/Fortran

$C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10}$ ⟵ ノード 0, ノード 1

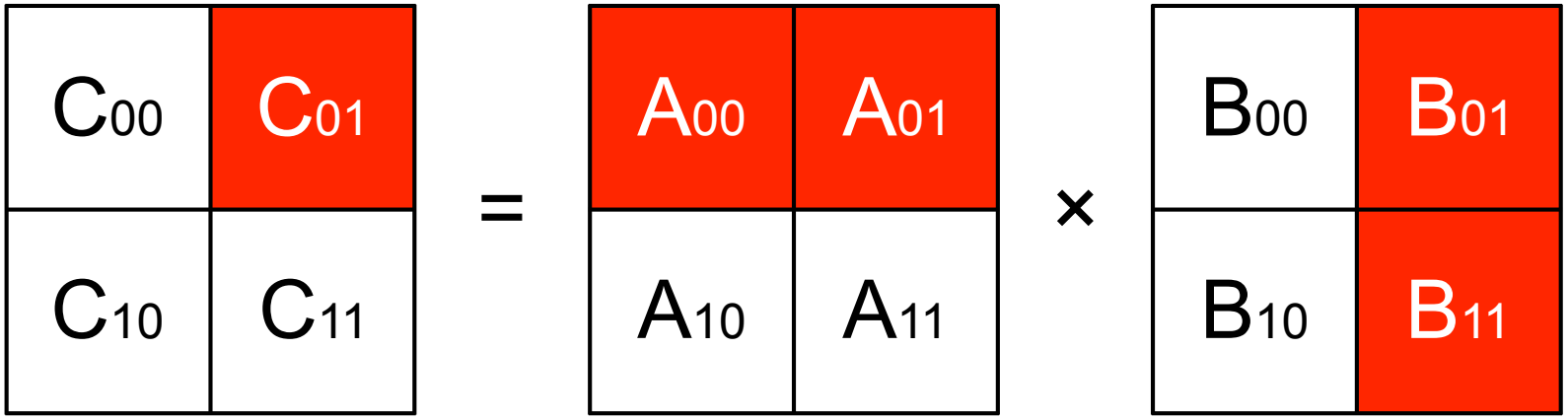
$C_{01} = A_{00} \times B_{01} + A_{01} \times B_{11}$ ⟵ ノード 1, ノード 2

$C_{10} = A_{10} \times B_{00} + A_{11} \times B_{10}$ ⟵ ノード 2, ノード 3

$C_{11} = A_{10} \times B_{01} + A_{11} \times B_{11}$ ⟵ ノード 3, ノード 4

課題 3

- Calculation by sub-matrix



XMP/C　XMP/Fortran

$C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10}$ ← ノード 0, ノード 1

$C_{01} = A_{00} \times B_{01} + A_{01} \times B_{11}$ ← ノード 1, ノード 2

$C_{10} = A_{10} \times B_{00} + A_{11} \times B_{10}$ ← ノード 2, ノード 3

$C_{11} = A_{10} \times B_{01} + A_{11} \times B_{11}$ ← ノード 3, ノード 4

課題 3

matmul.c or matmul.f90を元に，XMPで並列化を行う.

下記の赤字の箇所だけ，実装を行う.

1. (To compare time and verification) serial matrix multiply
2. In the function init_dmat(), Initialize arrays
3. In the function move_data(),
   - The root image puts sub-matrix A[][] and B[][] to another images
     - [C] Image 1 requires A[0:N/2][0:N] and B[0:N][N/2:N/2]
     - [F] Image 2 requires A(1:N,1:N/2) and B(N/2+1:N,1:N)
4. In the function mul_dmat(), execute a matrix multiply (C = A x B)
5. In the function gather_data(), gather results from each node
6. In the function verify(), perform verification
   - If the value is the same result as the sequential version, the result is OK
   - Please compare the measurement time