XcalableMP Tutorial

Masahiro Nakao (RIKEN AICS, Japan)

Sep.13-15, 2017, Maison de la Simulation, Saclay, France

Agenda in the afternoon session

- Hands-on for Global-view programming
- Hands-on for Local-view programming

Agenda in the afternoon session

- Hands-on for Global-view programming
- Hands-on for Local-view programming

Objective

http://xcalablemp.org/lecture.html

Please download "Source Code" in Tutorial.

in ./tutorial/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

Exercise 1

- Distribute array a[] to execute loop iteration in parallel

[C] init.c

```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

```fortran
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
```

Exercise 1

These files are partly parallelized.

- To parallelize, XMP directives are added to xmp_init.c or xmp_init.f90.

[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

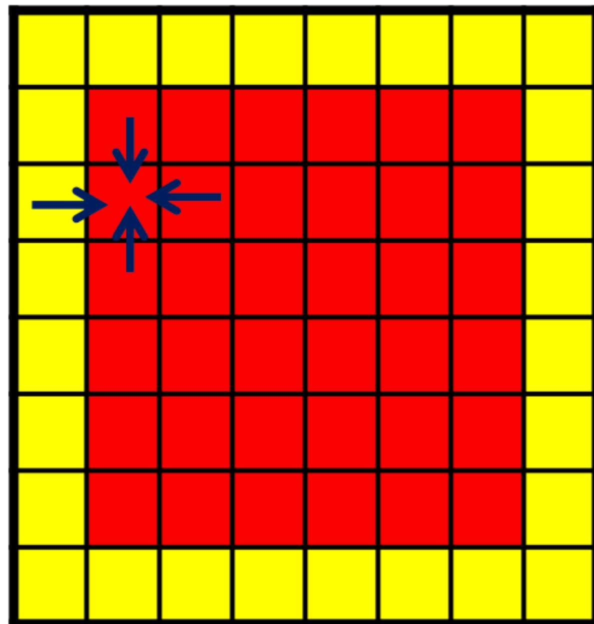
```
program init
!$xmp nodes p(2)
!$xmp template t(10)
!$xmp distribute t(block) onto p
  integer :: a(10), i
[align directive]

[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
```

Execute binary file

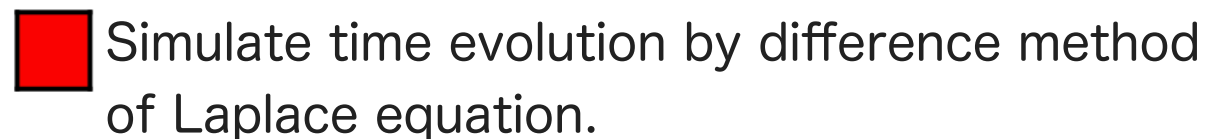
- Compiler
  - [C] $ xpmcc xmp_init_c -o xmp_init
  - [F] $ xmpf90 xmp_init.f90 -o xmp_init
- Execute
  - mpirun -np 2 ./xmp_init

Exercise 2 : Laplace equation

- Parallelization for 2-dimensional difference method



```
[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;
```

Each red cell is updated by top, bottom, left, right elements

Execute a serial code

- Please see laplace.c or laplace.f90 by editor
- Compile
  - [C] $ icc laplace.c
  - [F] $ ifort laplace.f90
- Execute
  - $ ./a.out
  - Verification =   5.54885...

Please memory the verification value to compare with the value of an XMP program.

Parallelization pattern in XMP

- **[Pattern 1]** Loop indices in both sides are the same
  - [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
```

Parallelization pattern in XMP

- **[Pattern 2]** Refer to neighborhood elements
  - [C] u[j][i] = uu[j-1][i] + uu[j][i-1] + ...;
  - [F] u(i,j) = uu(i,j-1) + uu(i-1,j) + ...

The shadow region must be added to distributed array by the shadow directive.

The shadow region must be updated before being referred by the reflect directive.

[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
```

Parallelization pattern in XMP

- **[Pattern 3]** There is a reduction operation
  - [C] s += abs(uu[j][i]-u[j][i]);
  - [F] s = s + abs(uu(i,j)-u(i.j))

[C]

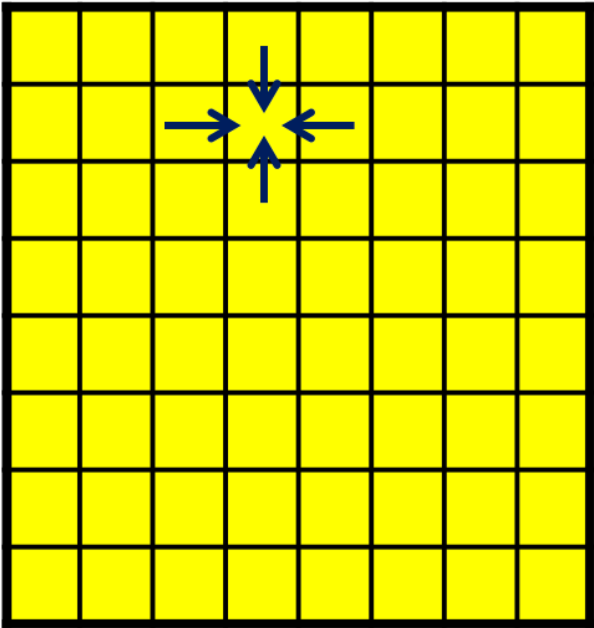
```
#pragma xmp loop (j,i) on t[j][i] reduction (+:s)
for(int j=0;j<10;j++)
  for(int i=0;i<10;i++)
    s += abs(uu[j][i] - u[j][i]);
```

[F]

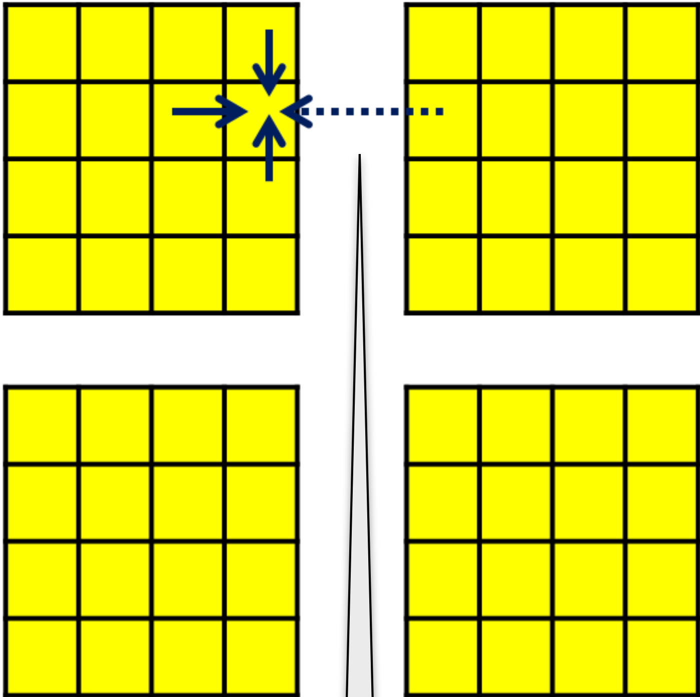
```
!$xmp loop (i,j) on t(i,j) reduction(+:s)
do j=1,10
  do i=1,10
    s = s + abs(uu(i,j) - u(i,j))
  end do
end do
```

Refer to remote elements

Serial execution



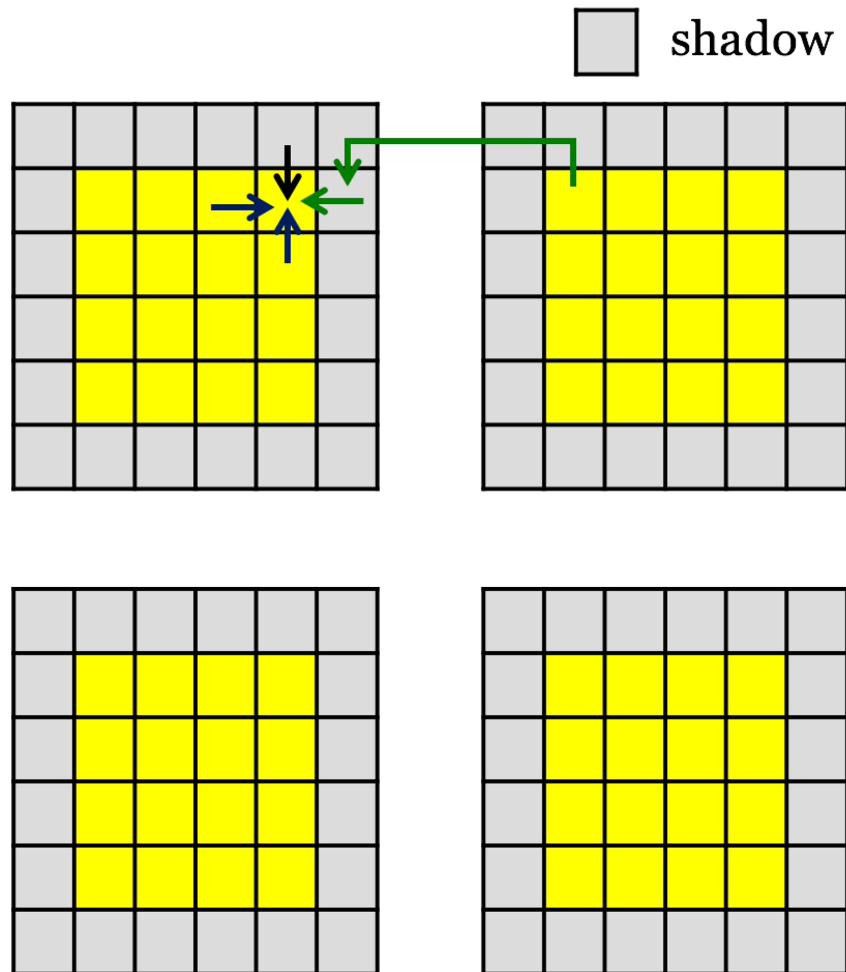
Parallel execution



Communication between nodes is needed !!

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

Refer to remote elements



1. The shadow directive defines shadow area to distributed array
2. The reflect directive updates shadow area from neighborhood node before loop calculation
3. The loop directive calculates loop statement with referring data in shadow area

Exercise 2

- Create xmp_laplace.c or xmp_laplace.f90, which are partly parallelized based on serial code (laplace.c or laplace.f90).
  - 2-dimensional node set, template-dimensional are used
  - execute four nodes and eight nodes
  - Confirm verification value is the same as that of serial version

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]          pattern 1
    for(j = 0; j < N2; j++){
        for(i = 0; i < N1; i++){
            u[j][i] = 0.0;
            uu[j][i] = 0.0;
        }
    }
```

```
[loop directive]                           pattern 1
    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]                           pattern 1
        for(j = 1; j < N2-1; j++)
          for(i = 1; i < N1-1; i++)
            uu[j][i] = u[j][i];

[reflect directive]                        pattern 2
[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]   pattern 3
    for(j = 1; j < N2-1; j++)
        for(i = 1; i < N1-1; i++)
          value += fabs(uu[j][i]-u[j][i]);
```

Exercise 2

- Compile
  - [C] xmpcc xmp_laplace.c -o xmp_laplace.x
  - [F] xmpf90 xmp_laplace.f90 -o xmp_laplace.x
- Execute
  - mpirun -np 4 ./xmp_laplace.x
  - mpirun -np 8 ./xmp_laplace.x
- Check
  - Is the verification value the same as that of the serial code ?

    (There may be a little difference due to the order of addition values)

Agenda in the afternoon session

- Hands-on for Global-view programming
- Hands-on for Local-view programming

Agenda in the afternoon session

- About coarray features
- ex 1 : Put/Get scalar variables (Only execution)
- ex 2 : Put/Get arrays (Only execution)
- ex 3 : Parallelize matrix multiply (Write program and execution it)

Coarray features

- One-sided communication (Put/Get)

array[start:length[:step]]

b[3], b[4], b[5] which image 1 has are copied to a[0], a[1], a[2] at image s0

[C]

```
int a[10];
int b[10]:[*];   // b is a coarray
       :
if(xmpc_this_image() == 0)
  a[0:3] = b[3:3]:[1]       // Get
```

[] indicates image index
(Fortran is 1-origin, C is 0-origin)

image 0

image 1

a[10]

a[10]

b[10]

3 5

Array section in XMP/C (1/2)

- Syntax

```
array[base : length :step]
```

- Each of base, length, and step must be an integer expression.
- Base:
  - When base is omitted, it is assumed to be 0
- Length :
  - When length is omitted, it is assumed to account for the remainder of the array dimension.
  - length must be greater than zero
- Step :
  - When step is omitted, it is assumed to be 1.
  - step must not be zero

Array section in XMP/C (2/2)

Assuming that an array A is declared by the following statement,

int A[100];

some array sections can be specified as follows:

A[10:10]	10 elements from A[10] to A[19]
A[10:]	90 elements from A[10] to A[99]
A[:10]	10 elements from A[0] to A[9]
A[10:5:2]	5 elements, A[10], A[12], A[14], A[16], A[18]
A[:]	the whole of A (from A[0] to A[99])

Coarray features

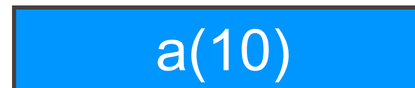
- One-sided communication (Put/Get)

b(3:5) which image 2 has
are copied to a(1:3) at image s0

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

[] indicates image index
(Fortran is 1-origin, C is 0-origin)

image 1



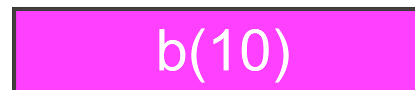
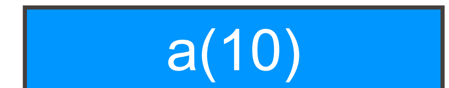


image 2



3 5

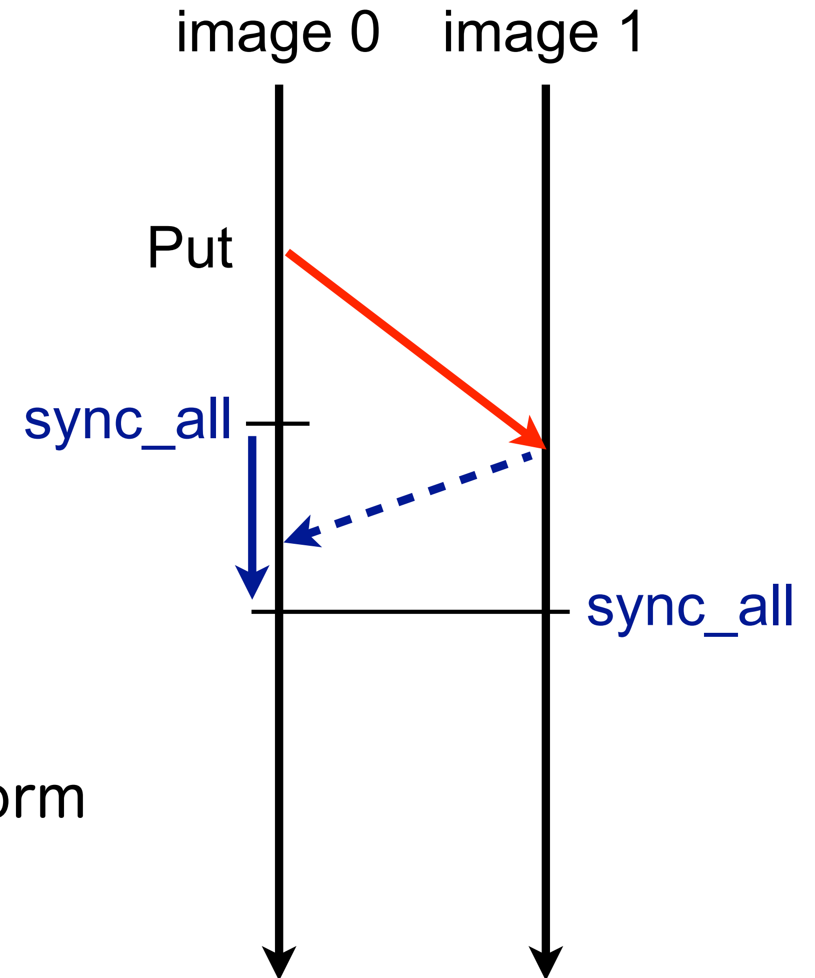
Synchronization

[C]

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

[F]

```
sync all
```



Complete all one-sided communication which a node has issued, and then perform barrier operation.

Exercise 1

- Put/Get scalar variables
  - [C] xmpcc coarray_scalar.c -o coarray_scalar.x
  - [F] xmpf90 coarray_scalar.f90 -o coarray_scalar.x
  - mpirun -np 2 ./coarray_scalar.x

[C]

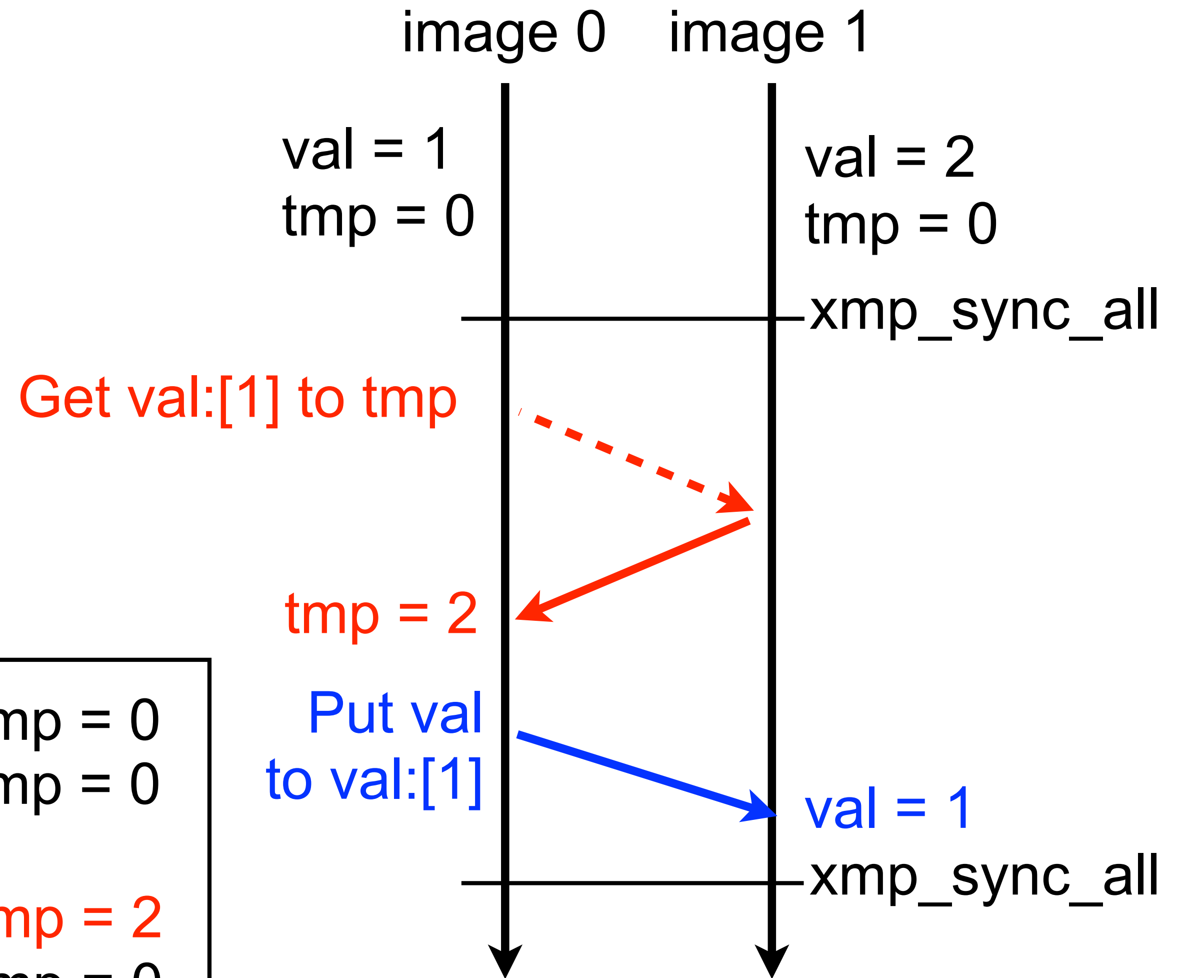
```
if(xmpc_this_image() == 0){
    tmp = val:[1];  // Get
    val:[1] = val;   // Put
 }
 xmp_sync_all(NULL);
```

[F]

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

Exercise 1 : result (XMP/C)

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

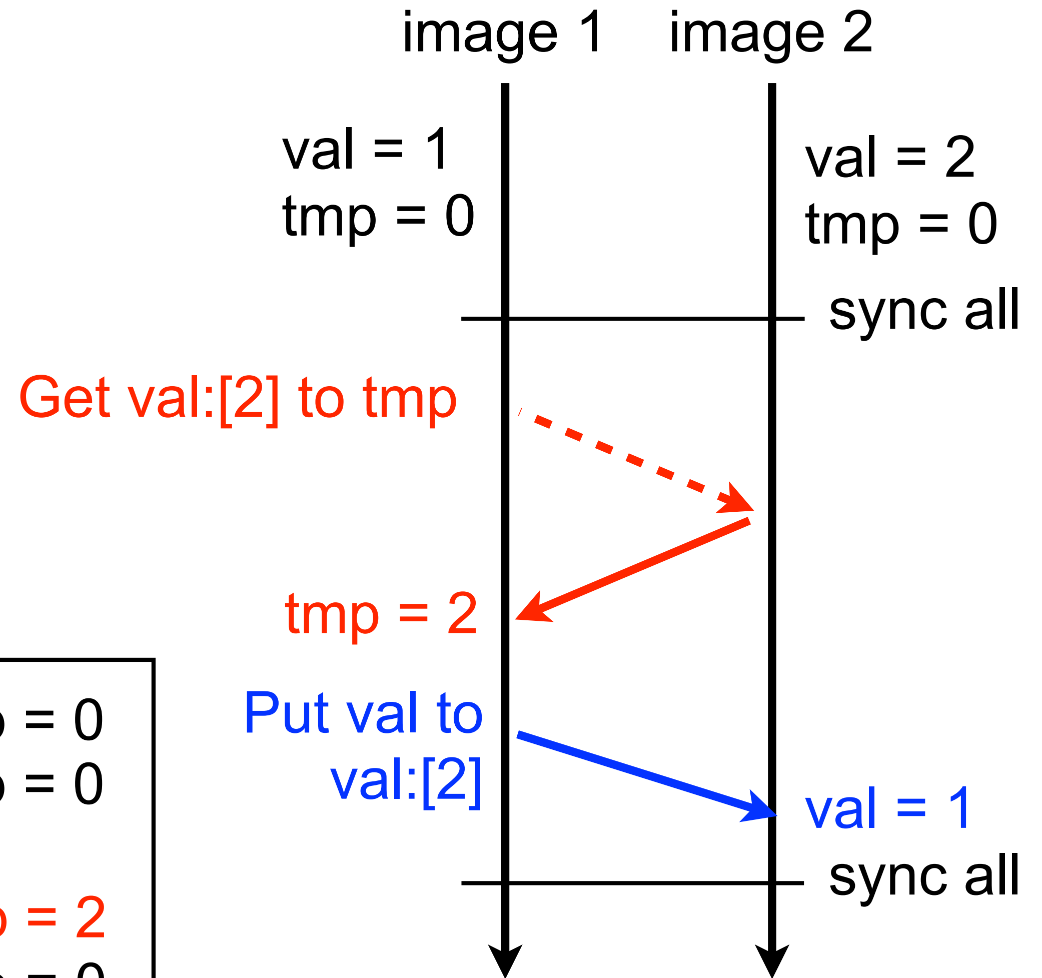


Result

```
[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
```

Exercise 1 : result (XMP/Fortran)

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



Result

```
[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
```

Exercise 2

- Put/Get arrays
  - [C] xmpcc coarray_vector.c -o coarray_vector.x
  - [F] xmpf90 coarray_vector.f90 -o coarray_vector.x
  - mpirun -np 2 ./coarray_vector.x

[C] image 0, [F] image 1

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

[C] image 1, [F] image 2

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

Exercise 2 : results

[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
```

Exercise 2 : results

[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

Exercise 2 : results

[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
```

Exercise 3

- Parallelize matrix multiply
  - C = A x B
- Flow of parallelization
  - Execute 4 nodes
  - At first, a root image (image 0 in C, image 1 in Fortran) has all data of arrays a[][] and b[][].
  - Next, the root image transfer a part of a[][] and b[][] to another images, and each node calculate c[][] in parallel.
  - In the end, the root image gathers c[][] which is calculated by each node.

```
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];
```

Exercise 3

- Calculation by sub-matrix

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

	XMP/C	XMP/Fortran
$C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10}$ ←	image 0,	image 1
$C_{01} = A_{00} \times B_{01} + A_{01} \times B_{11}$ ←	image 1,	image 2
$C_{10} = A_{10} \times B_{00} + A_{11} \times B_{10}$ ←	image 2,	image 3
$C_{11} = A_{10} \times B_{01} + A_{11} \times B_{11}$ ←	image 3,	image 4

handles

Exercise 3

- Calculation by sub-matrix

$$\begin{bmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{bmatrix} = \begin{bmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{bmatrix} \times \begin{bmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{bmatrix}$$

	XMP/C	XMP/Fortran
$C_{00} = A_{00} \times B_{00} + A_{01} \times B_{10}$ ←	image 0,	image 1
$C_{01} = A_{00} \times B_{01} + A_{01} \times B_{11}$ ←	image 1,	image 2
$C_{10} = A_{10} \times B_{00} + A_{11} \times B_{10}$ ←	image 2,	image 3
$C_{11} = A_{10} \times B_{01} + A_{11} \times B_{11}$ ←	image 3,	image 4

handles

Exercise 3

Based on matmul.c or matmul.f90
The implementation part is only the red part as following.

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(), a node 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