FUJITSU Software
Technical Computing Suite V2.0

Fortran User's Guide
Additional Volume
COARRAY

Preface

Purpose of This Manual

Coarray is a specification based on Fortran 2008 standard. This manual explains how to use coarray on the Fortran system (hereafter, "this system").

In this manual, the following terms are used:

- COARRAY program
  Program using coarrays

- COARRAY feature
  Feature of compiling and executing COARRAY program. This is also simply called COARRAY.

Intended Readers

This manual is intended for readers who use the COARRAY feature on this system.

Readers of this manual are assumed to have knowledge of Fortran program, C program, C++ program, OpenMP, MPI, and Linux commands/file manipulation/shell programming.

Organization of This Manual

This manual is organized as follows:

- Chapter 1 Overview and Usage of COARRAY
- Chapter 2 Parallelization with COARRAY
- Chapter 3 Using MPI from COARRAY Programs
- Chapter 4 Debugging of COARRAY Programs
- Chapter 5 Tuning of COARRAY Programs
- Chapter 6 Notes

Related Manuals

The following manuals are related to this manual:

- Fortran Language Reference
- Fortran User's Guide
- Fortran Compiler Messages
- Fortran/C/C++ Runtime Messages
- C User's Guide
- C++ User's Guide
- MPI User's Guide

Notes of This Manual

The code examples of optimization in this manual are conceptual source that complements each explanation of functions. When the code examples are compiled and executed, optimizations may not work as expected. This is because optimizations depend on compiler options and other conditions.

Notation Used in This Manual

Syntax Description Symbols

A syntax description symbol is a symbol that has a specific meaning when used to describe syntax. The following symbols are used in this manual:

Symbol name	Symbol	Explanation
Selection symbols	{ }	Only one of the items enclosed in the braces must be selected (items are listed vertically).
	|	Multiple items are enumerated by this delimiter (items are listed horizontally).
Option symbol	[ ]	An item enclosed in brackets can be omitted. This symbol includes the meaning of the selection symbol "{ }".
Repeat symbol	...	The item immediately preceding the ellipsis can be specified repeatedly in the syntax.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Trademarks

- OpenMP is a trademark of OpenMP Architecture Review Board.

- Linux is the registered trademark of Linus Torvalds in the U.S. and other countries.

- Other trademarks and registered trademarks are trademarks or registered trademarks of their respective owners.

Date of Publication and Version

Version	Manual code
November 2015, 1st Version	J2UL-2066-01ENZ0(00)

Copyright

Copyright FUJITSU LIMITED 2015

Contents

Chapter 1 Overview and Usage of COARRAY

This chapter gives an overview and usage of COARRAY in this system.

1.1 Overview of COARRAY

The following describes an overview of COARRAY provided in this system.

COARRAY in this system supports only process-parallelization. Use OpenMP for thread-parallelization. For details of OpenMP, see the section "Parallelization by OpenMP Specification" in "Fortran User's Guide".

1.1.1 Specification of COARRAY in this System

The MPI library is used by COARRAY in this system.

MPI (Message Passing Interface) is the set of standards determined by the MPI Forum for regulating the library interface, which enables Fortran, C, and C++ languages to be used for parallel MPI programming in parallel computing systems with distributed memory.

In this system, COARRAY programs can be linked with MPI programs.

1.1.2 Image and Rank

COARRAY programs are copied and executed asynchronously. Each copy is termed "image". Images have a positive integer number which starts at 1. This number is termed "image index".

Since COARRAY programs are based on execution environments of MPI, "image" is sometimes called "rank". Ranks have a non-negative integer number which starts at 0. This number is termed "rank number".

The image index is associated with the rank number: "image index = rank number + 1".
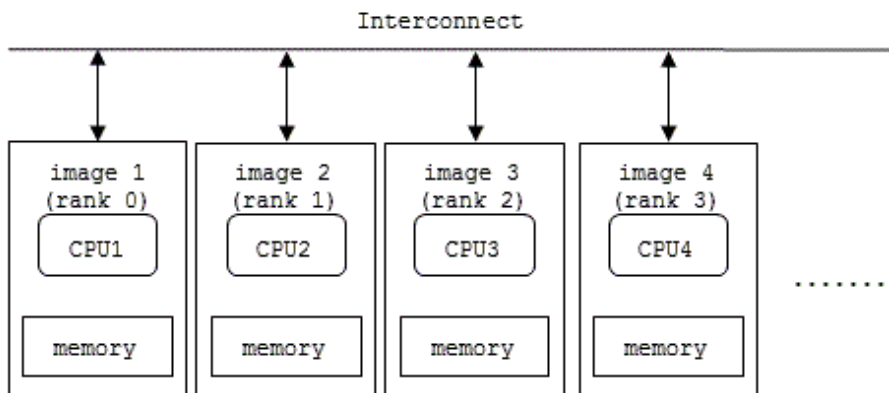
1.2 Usage of COARRAY

The following describes compilation and execution of COARRAY programs.

1.2.1 How to Compile

The following describes the format and meanings of compiler options for COARRAY.

For more details, see the section "Compiler Options" in "Fortran User's Guide".

-N{coarray|nocoarray}

These options specify whether or not to validate COARRAY specification.

If neither is specified, it is regarded that -Nnocoarray is specified.

coarray

This option validates COARRAY specification.

It is necessary to specify -Ncoarray if the compilation command line includes object programs which are compiled with -Ncoarray.

```
$ frtpx a.f90 -Ncoarray -c
$ frtpx a.o -Ncoarray
```

nocoarray

This option invalidates COARRAY specification.

If you compile COARRAY programs with this option, error messages are output and the compilation stops.

1.2.2 Linking with C/C++ Programs

You can link COARRAY programs with C/C++ programs and execute them.

Linking with C programs

Use Fortran compiler (frt or frtpx) and specify the -Ncoarray option at linking time.

Linking with C++ programs

Use C++ compiler (FCC or FCCpx) and specify the --linkcoarray option at linking time.

Refer to "C User's Guide" or "C++ User's Guide" for details of interlanguage linkage.

1.2.3 How to Execute

Use the mpiexec(1) command on compute nodes to run executable programs which are compiled and linked with the -Ncoarray option. Submit executable programs as jobs to the Job Operation Software instead of running executables directly on compute nodes.

See the Job Operation Software manual as for the way of submission. See "MPI User's Guide" for details of the mpiexec(1).

In the Job Operation Software manual, explanations on process-parallelization are described based on MPI process. Replace descriptions on MPI with COARRAY appropriately to read the manual.

1.2.3.1 Execution Command Formats

COARRAY programs are executed by using mpiexec(1) command, which is used to execute MPI programs.

Typical usage of mpiexec(1) is described in this document. See "MPI User's Guide" for details of the command and "1.1.2 Image and Rank" for the rank and rank number of mpiexec(1).

mpiexec(1) format

Command	Operands
mpiexec	*global_options local_options execfile execfile_arguments*

Specify "*global_options*" and "*local_options*" after the "mpiexec".

Specify "*execfile*" and "*execfile_arguments*" subsequently. Runtime options specified as "*execfile_arguments*" are passed to all images. For details of "*execfile*" and "*execfile_arguments*", see "MPI User's Guide".

Example

```
$ mpiexec -of-proc procfile ./a.out
```

global_options

The following describes options that can be specified as "*global_options*".

Option	Meaning
{ -h | --help }	Displays help messages for this command and ends mpiexec(1).
	Even if mpiexec(1) is executed with no arguments, the help message is displayed and mpiexec(1) ends.
{ -of | --of | -std | --std } *FILE*	The parallel process standard output and standard error output are saved in the file with the name specified at *FILE*. If just the filename or the relative path is specified, the relative path from the job execution current directory is used.
	If this option is specified more than once, the parameter specified last takes priority.
{ -oferr | --oferr | -stderr | --stderr } *ERR_FILE*	Saves the parallel process standard error output to the file with the name specified at *ERR_FILE*. If just the filename or the relative path is specified, the relative path from the job execution current directory is used.
	If this option is specified more than once, the parameter specified last takes priority.
{ -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc } *ERR_PROC_FILE*	Saves the parallel process standard error output to the file with the filename "*ERR_PROC_FILE*.rank-number". The character string for the "rank number" is the actual rank number under MPI_COMM_WORLD, expressed as a numeric character string of the same number of digits.
	If this option is specified more than once, the parameter specified last takes priority.
{ -ofout | --ofout | -stdout | --stdout } *OUT_FILE*	Saves the parallel process standard output to the file with the filename specified at *OUT_FILE*. If just the filename or the relative path is specified, the relative path from the job execution current directory is used.
	If this option is specified more than once, the parameter specified last takes priority.
{ -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc } *OUT_PROC_FILE*	Saves the parallel process standard output, separately for each process, to the file with the filename "*OUT_PROC_FILE*.rank-number". The character string for the "rank number" is the actual rank number under MPI_COMM_WORLD, expressed as a numeric character string of the same number of digits.
	If this option is specified more than once, the parameter specified last takes priority.
{ -of-proc | --of-proc | -std-proc | --std-proc } *PROC_FILE*	Saves the parallel process standard output and standard error output, separately for each process, to the file with the filename "*PROC_FILE*.rank-number". The character string for the "rank number" is the actual rank number under MPI_COMM_WORLD, expressed as a numeric character string of the same number of digits.
	If this option is specified more than once, the parameter specified last takes priority.
{ -ofprefix | --ofprefix | -stdprefix | --stdprefix } *PREFIX*	Outputs the character string corresponding to the keyword specified at *PREFIX* at the start of the parallel process standard output and standard error output lines.
	Any of the following keywords can be specified at *PREFIX*:
	- rank The rank number under MPI_COMM_WORLD is attached at the start of the output character string.
	- nid The node ID is attached at the start of the output character string.
	- rank,nid The rank number under MPI_COMM_WORLD and the node ID are both attached in sequence at the start of the output character string.
	- nid,rank The node ID and the rank number under MPI_COMM_WORLD are both attached in sequence at the start of the output character string.
	If this option is specified more than once, the parameter specified last takes priority.

Option	Meaning
{ -stdin | --stdin } *STDIN_FILE*	Loads from the file with the filename specified at STDIN_FILE, the standard input for all parallel processes that were generated by executing the COARRAY program. If just the filename or the relative path is specified, the relative path from the job execution current directory is used. If this option is specified more than once, the parameter specified last takes priority.
{ -V | --version }	Outputs the mpiexec(1) version information. If this option is specified without specifying any other options, mpiexec(1) ends after the mpiexec(1) version information is output.

local_options

The following describes options that can be specified in *local_options*.

Option	Meaning

-x *NAME=VALUE*	Specifies the environment variable when executing a COARRAY program.
	
	NAME indicates the environment variable name. *VALUE* indicates the value to be set in that environment variable.
	
	If it is necessary to specify spaces, the following format is also allowed.
	
	`"NAME=VALUE "`
	
	Only one environment variable can be specified for this option. To set multiple environment variables, specify this option as often as needed.
	
	`-x OMP_NUM_THREADS=8 -x THREAD_STACK_SIZE=4096`
	
	However, if the environment variable name is specified more than once, the value specified last takes priority.

{ -c | -np | --np | -n | --n } *N*	Specifies the number (an integer) of parallel processes for the relevant COARRAY programs. If this option is omitted, the maximum number of parallel processes that can be generated is assumed. If this option is specified more than once, the parameter specified last takes priority.

1.2.3.2 Number of Processes

The number of processes is decided by the following priority levels.

1. The value specified for the mpiexec(1)

2. The value specified for the Job Operation Software.

Example

The following example is a job script to execute a COARRAY program on four nodes with one process per core in the PRIMEHPC FX100 system.

```
#!/bin/sh
#PJM -L node=4
#PJM --mpi proc=128
PATH=/opt/FJSVmxlang/bin:$PATH; export PATH
LD_LIBRARY_PATH=/opt/FJSVmxlang/lib64:$LD_LIBRARY_PATH; export LD_LIBRARY_PATH
mpiexec -n 128 ./a.out
```

1.2.3.3 Execution Profile File

The execution profile file is available for COARRAY programs in the same way as Fortran programs not using coarrays.

1.2.3.4 Standard Input, Output, and Error Output for Execution

Standard input of the mpiexec(1) is not standard input of each image by default. If COARRAY programs expect standard input, it is necessary to specify a filename containing what you want to pass to images with the --stdin option of the mpiexec(1).

You can also specify destinations of the standard output and error output by means of the respective options.

If '*' is specified as the UNIT specifier of READ statements, only the image whose image index is 1 is enable to execute the statements. A runtime error occurs if the statements are executed by other images.

1.2.3.5 Filename for Input and Output

In general, when a COARRAY program is executed, copied programs run similarly on multiple nodes. Therefore, when using files for input or output, it is necessary to be aware of whether file objects with the same filename actually mean the same one file or not. When you refer a file on the shared file system, the file means the same file. On the contrary, when you refer a file on local file systems, the file means respective files on each node.

Addition of image indices to filenames enables images to create respective files on the shared file system.

Processing filename described above is on your own responsibility.

In usual Fortran programs, the filename of a preconnected file is fort.{unit number}. In COARRAY programs, the filename of a preconnected file is fort.{unit number}.{image index} in order to create respective files for each node.

Chapter 2 Parallelization with COARRAY

This chapter gives a description of parallelization with the COARRAY.

2.1 Configuration of COARRAY Program

The following example describes the configuration of a COARRAY program.

The program demonstrates stencil operations, which is often used in Jacobi iterative method.

```
 1  PROGRAM TOY_STENCIL_SOLVER_CAF_1D_SIMPLE
 2   USE ISO_FORTRAN_ENV
 3   IMPLICIT NONE
 4   INTEGER::ITER,I,J,IMAX,JMAX,JMAX_ALL,JACCMAX,JLOC,NIMG,ID
 5   REAL(8)::DIFF
 6   REAL(8),SAVE::RSD[*]
 7   REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::FLD_A
 8   REAL(8),DIMENSION(:,:),ALLOCATABLE::FLD_B
 9   INTEGER,PARAMETER::SIZE=1026,NN=10
10   REAL(8),PARAMETER::PARAM=0.1666666666
11
12   NIMG=NUM_IMAGES()
13   ID=THIS_IMAGE()
14
15   IF(NIMG/=16)THEN
16     IF(ID==1)THEN
17       PRINT *,' THE NUMBER OF IMAGES IS NOT 16. '
18     END IF
19     STOP
20   END IF
21
22   IMAX=SIZE
23   JMAX_ALL=SIZE
24   JACCMAX=(JMAX_ALL-2)/NIMG
25   JMAX=JACCMAX+2
26
27   ALLOCATE(FLD_A(IMAX,JMAX)[*],FLD_B(IMAX,JMAX))
28
20   JLOC=JACCMAX*(ID-1)
30   DO J=1,JMAX
31     FLD_A(:,J)=1.0+DBLE(MOD((JLOC + J),16))/DBLE(JMAX_ALL)
32   END DO
33   FLD_B=0.0
34
35   RSD=0.0
36
37   DO ITER=1,NN
38
39     DO J=2,JMAX-1
40       DO I=2,IMAX-1
41         FLD_B(I,J)=PARAM*(FLD_A(I-1,J)+FLD_A(I+1,J)+FLD_A(I,J-1)+FLD_A(I,J+1))
42         DIFF=FLD_B(I,J)-FLD_A(I,J)
43         RSD=RSD+DIFF*DIFF
44       END DO
45     END DO
46
47     FLD_A(2:IMAX-1,2:JMAX-1)=FLD_B(2:IMAX-1,2:JMAX-1)
48
49     IF(NIMG>1)THEN
50       SYNC ALL
51       IF(ID==1)THEN
52           FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
```

```
53        ELSE IF(ID==NIMG)THEN
54            FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
55        ELSE
56            FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
57            FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
58        END IF
50        SYNC ALL
60     END IF
61
62    END DO
63
64    SYNC ALL
65    CALL CO_SUM(RSD,RESULT_IMAGE=1)
66    IF(ID==1)THEN
67      PRINT *,' RESIDUAL :',RSD
68    END IF
69
70    DEALLOCATE(FLD_A,FLD_B)
71
72    STOP
73  END PROGRAM TOY_STENCIL_SOLVER_CAF_1D_SIMPLE
```

2.1.1 Declaration of Coarrays

All variables in programs have objects on each image.

Coarrays are declared as following. They can be defined and referenced between images.

```
6    REAL(8),SAVE::RSD[*]
7    REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::FLD_A
```

Variables except coarrays are local variables on each image.

```
4    INTEGER::ITER,I,J,IMAX,JMAX,JMAX_ALL,JACCMAX,JLOC,NIMG,ID
5    REAL(8)::DIFF
...
8    REAL(8),DIMENSION(:,:),ALLOCATABLE::FLD_B
```

2.1.2 Image Index

Each image has an image index. It is also possible to acquire the entire number of images.

The entire number of images cannot be changed within a program.

The NUM_IMAGES at line 12 acquires the entire number of images, and then the THIS_IMAGE at line 13 acquires the image index in processing.

```
12    NIMG=NUM_IMAGES()
13    ID=THIS_IMAGE()
```

2.1.3 Acquisition of Local Work Area and Creating Data

The following describes the data structure of the above example. Calculated data in the entire program is a two-dimensional array with 1024 rows and 1024 columns. It is divided into strips and shared among 16 images.

Since this calculation requires neighboring data, data on each image overlaps each other by one column. That is, the both ends of each image are overlapped.

Thus, data block on each image is a two-dimensional array with 1024 rows and 66 columns.
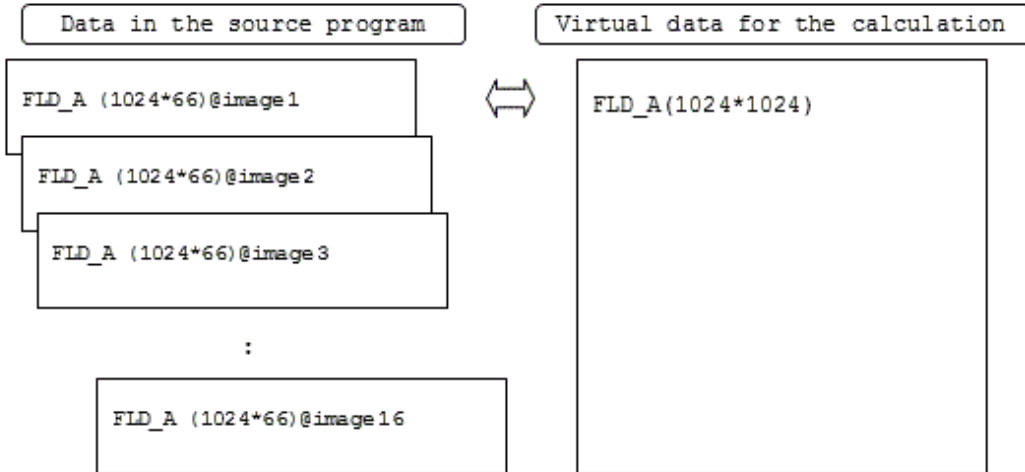
Each image acquires a local work area divided at the beginning.

SIZE and IMAX is 1026, the number of rows including neighbors. JACCMAX is 1024/16=64, the number of columns calculated on one image. JMAX is 64+2=66, the number of columns including neighbors.

Data block required on each image is allocated as two-dimensional arrays with 1024 rows and 66 columns.

```
22      IMAX=SIZE
23      JMAX_ALL=SIZE
24      JACCMAX=(JMAX_ALL-2)/NIMG
25      JMAX=JACCMAX+2
26
27      ALLOCATE(FLD_A(IMAX,JMAX)[*],FLD_B(IMAX,JMAX))
```

Sets the input data of the calculation.

```
29      JLOC=JACCMAX*(ID-1)
30      DO J=1,JMAX
31        FLD_A(:,J)=1.0+DBLE(MOD((JLOC+J),16))/DBLE(JMAX_ALL)
32      END DO
33      FLD_B=0.0
```

JLOC starts at 0 on image 1, because it is calculated as (64-1)*(1-1). It starts at 63 on image 2, because it is calculated as (64-1)*(2-1). The others are calculated in the same way.

Puts the same data into where is overlapped in this assignment.

The coarray FLD_A does not have an image selector ("[") above, data is assigned to local variables for each image. This means that each image sets necessary data respectively.

2.1.4 Local Calculation of Each Image

Each image performs assigned calculations. In this example, calculation is over the range of 1024 * 64 with inputting arrays with 1026 * 66 elements.

The calculation results are saved in the FLD_B with the input FLD_A. Then, the results in FLD_B are written back to the FLD_A.

```
39      DO J=2,JMAX-1
40        DO I=2,IMAX-1
41          FLD_B(I,J)=PARAM*(FLD_A(I-1,J)+FLD_A(I+1,J)+FLD_A(I,J-1)+FLD_A(I,J+1))
42          DIFF=FLD_B(I,J)-FLD_A(I,J)
43          RSD=RSD+DIFF*DIFF
44        END DO
45      END DO
46
47      FLD_A(2:IMAX-1,2:JMAX-1)=FLD_B(2:IMAX-1,2:JMAX-1)
```

2.1.5 Transfer of the Calculation Results to Neighbors

Each image transfers calculation results to other images.

At line 49, there is a consideration of the case that the number of images is one. The conditional expression is always true in this case.

At line 50, the program waits for local calculations on each image to be completed. Unless image control statements such as the SYNC ALL are stated, assignments to coarrays are not guaranteed.

On image 1, the FLD_A(2:1025,2) on image 2 is assigned to FLD_A(2:1025,66).

On image 16, the FLD_A(2:1025,65) on image 15 is assigned to FLD_A(2:1025,1).

On the other images 2-15, the edges of data on neighboring images are assigned to each local array.

At line 59, the program waits for assignments between each image to be completed.

```
49    IF(NIMG>1)THEN
50      SYNC ALL
51      IF(ID==1)THEN
52          FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
53      ELSE IF(ID==NIMG)THEN
54          FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
55      ELSE
56          FLD_A(2:IMAX-1,JMAX)=FLD_A(2:IMAX-1,2)[ID+1]
57          FLD_A(2:IMAX-1,1)=FLD_A(2:IMAX-1,JMAX-1)[ID-1]
58      END IF
59      SYNC ALL
60    END IF
```

2.1.6 Aggregation of Calculation Results

Each image computes as described above and waits for the completion at line 64.

The CO_SUM intrinsic subroutine sums the calculation results on each image. The sum is returned to image 1.

The result is displayed by image 1 on behalf of all the images.

```
64    SYNC ALL
65    CALL CO_SUM(RSD,RESULT_IMAGE=1)
66    IF(ID==1)THEN
67      PRINT *,' RESIDUAL :',RSD
68    END IF
```

2.2 Start and Termination of COARRAY Programs

The start and termination of COARRAY programs are described in this section.

2.2.1 Start of Program

- A COARRAY program is started with the asynchronous initiation of one or more images decided at the start of program.

- Each image is started as an individual process and has individual environment.

- An image is identified by the image index. It is an integer value from 1 to the number of images.

2.2.2 Termination of Program

- The termination of the executing program is either normal termination or error termination.

- Error termination is started on all images if any one of images starts an error termination.

- The program is ended when all images are terminated.

- Normal termination is started on the image when the STOP or END PROGRAM statement is executed on it.

- Error termination is started on the image when the ERROR STOP statement is executed or an error condition occurs on it.

- Normal termination is started on the image when the following FUJITSU extended service subroutines or functions are executed on it:

    - SETRCD service subroutine

    - EXIT service subroutine

    - exit(3) function in the C language (Mixed language programming with C/C++ is used.)

- Error termination is started on the image when the following FUJITSU extended service subroutines or functions are executed on it, or when the program specified with the following runtime option is terminated:

    - ABORT service subroutine

    - abort(3) function in the C language (Mixed language programming with C/C++ is used.)

    - Runtime option -a

2.2.3 Return Code

The return code of the program depends on the behavior of mpiexec(1) command if values specified for the return code on multiple images are not the same.

See "MPI User's Guide" for details.

2.2.4 Definition and Reference to Coarray

The following errors may occur when a coarray is defined or referred.

Return value	Meaning
1711	An error was detected in the data transfer process for the coarray area.
1721	An image index should be a positive integer and not exceed the number of images.
1790	Transfer where neither the source nor destination image is self-image is not supported.

2.3 Image Control Statements and Intrinsic Subroutines

This section describes the image control statements and the intrinsic subroutines.

2.3.1 SYNC ALL Statement

The SYNC ALL statement performs synchronization among all images.

The following example shows how to use the SYNC ALL statement.

Image 1 reads data and transfers it to the other images.

The first SYNC ALL statement controls the execution sequence so that the initialization of the coarray Z on images 2 and 3 is executed before updated by the data from image 1.

The second SYNC ALL statement controls the execution sequence so that the reference to the coarray Z on images 2 and 3 is executed after updated by the data from image 1.

```
REAL(8),SAVE::Z[*]
...
Z=0.0                    ! initialize the coarray Z by value 0.0 on each image
SYNC ALL                 ! First SYNC ALL statement
IF(THIS_IMAGE()==1)THEN
   READ(*,*)Z

   Z[2]=Z
```

```
    Z[3]=Z

END IF
SYNC ALL                 ! Second SYNC ALL statement
...
                         ! refer the coarray Z
```

The following figure shows the behavior of each image.
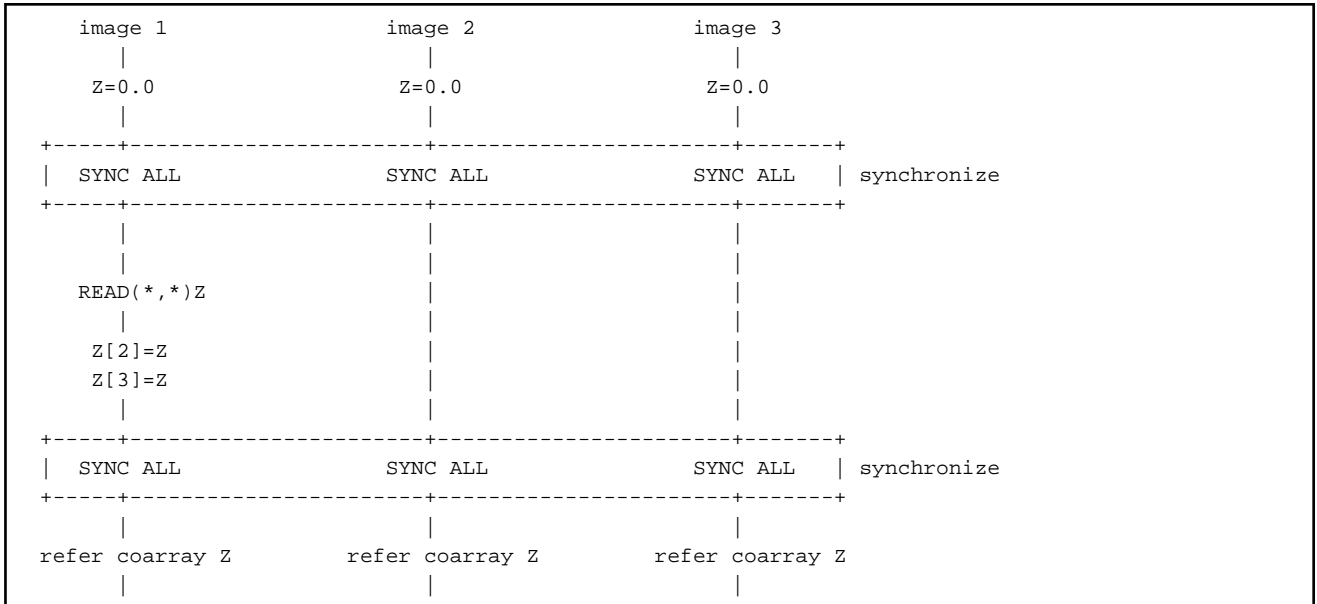
```
      image 1                  image 2                  image 3
        |                        |                        |
      Z=0.0                    Z=0.0                    Z=0.0
        |                        |                        |
  +-----+--------------------+--------------------+------+
  |  SYNC ALL                SYNC ALL                SYNC ALL   | synchronize
  +-----+--------------------+--------------------+------+
        |                        |                        |
        |                        |                        |
      READ(*,*)Z                 |                        |
        |                        |                        |
      Z[2]=Z                     |                        |
      Z[3]=Z                     |                        |
        |                        |                        |
  +-----+--------------------+--------------------+------+
  |  SYNC ALL                SYNC ALL                SYNC ALL   | synchronize
  +-----+--------------------+--------------------+------+
        |                        |                        |
   refer coarray Z         refer coarray Z         refer coarray Z
        |                        |                        |
```

The following figure shows the behavior of each image without the first SYNC ALL statement.

```
      image 1                  image 2                  image 3
        |                        |                        |
      Z=0.0                      |                        |
        |                        |                        |
      READ(*,*)Z                 |                        |
        |                        |                        |
      Z[2]=Z                     |                        |
      Z[3]=Z                     |                        |
        |                        |                        |
        |                      Z=0.0                    Z=0.0
        |                        |                        |
  +-----+--------------------+--------------------+------+
  |  SYNC ALL                SYNC ALL                SYNC ALL   | synchronize
  +-----+--------------------+--------------------+------+
        |                        |                        |
   refer coarray Z        refer coarray Z(0.0)    refer coarray Z(0.0)
        |                        |                        |
```
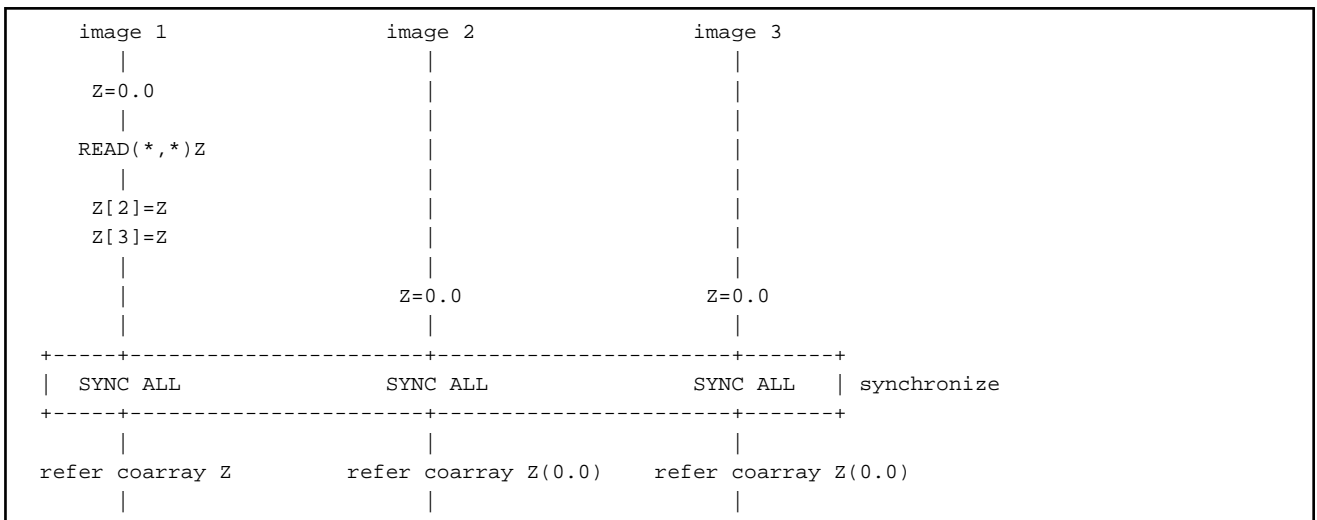
The program could run in the above sequence without the first SYNC ALL statement. In this case, images 2 and 3 initialize Z by the value 0.0 after image 1 defines Z by the READ statement and updates Z on images 2 and 3.

The following figure shows the behavior of each image without the second SYNC ALL statement.

```
      image 1                  image 2                  image 3
        |                        |                        |
      Z=0.0                    Z=0.0                    Z=0.0
        |                        |                        |
  +-----+--------------------+--------------------+------+
```

```
  |   SYNC ALL               SYNC ALL               SYNC ALL   | synchronize
  +-----+-----------------------+-----------------------+------+
     |                          |                       |
    READ(*,*)Z                  |                       |
                      refer coarray Z(0.0)   refer coarray Z(0.0)
                                |                       |
     Z[2]=Z                     |                       |
     Z[3]=Z                     |                       |
        |                       |                       |
  refer coarray Z               |                       |
        |                       |                       |
```
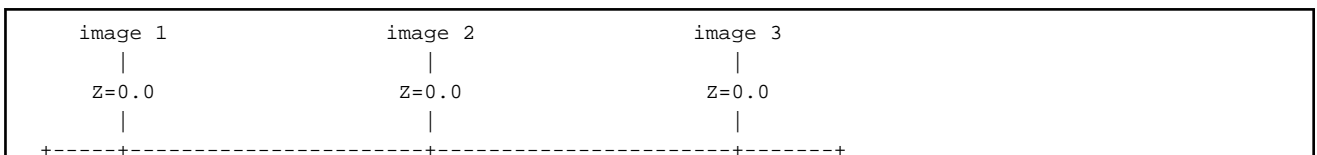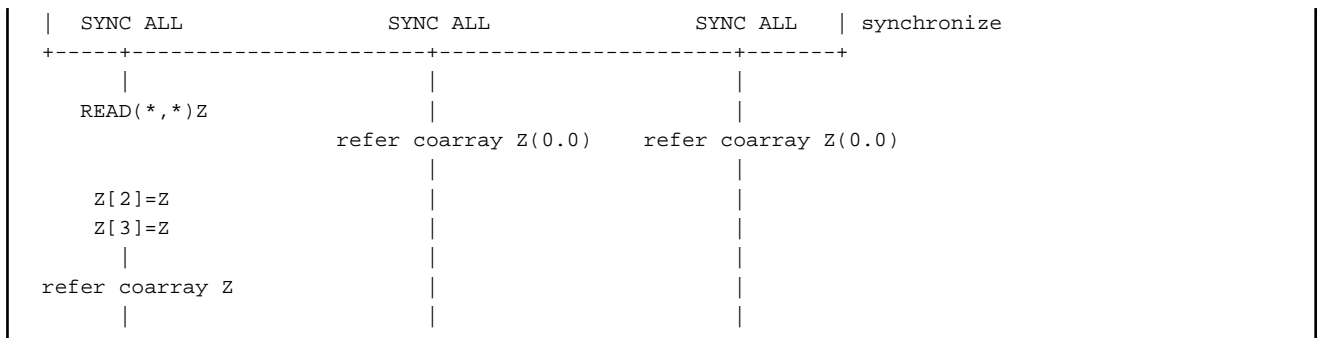
This example shows that images 2 and 3 refer the coarray Z before Z is updated by image 1.

STAT= specifier of SYNC ALL statement

If the STAT= specifier is specified for the SYNC ALL statement, the following value is returned.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.
1713	The image control statement cannot be executed in the CRITICAL construct.
1723	A deadlock was detected on this image.
1731	The SYNC ALL statement was terminated because the termination process was started on other images.
	(The return value is STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)
1751	An error was detected in the synchronization process for the SYNC ALL statement.
1752	An error was detected in the communication process for the SYNC ALL statement.

2.3.2 SYNC IMAGES Statement

The SYNC IMAGES statement performs synchronization among specified images.

The following example shows that image 1 has the other images wait for preparing data. Images except for image 1 are waiting for image 1 preparing the data but not for the other images. Therefore, they resume execution after synchronization with image 1. Image 1 resumes execution after synchronization with all the other images.

```
REAL(8),SAVE::Z[*]
...
IF(THIS_IMAGE()==1)THEN
   !
   ! prepare data for image 2 and 3
   !
   READ(*,*)Z

   Z[2]=Z
   Z[3]=Z
   SYNC IMAGES(*)  ! image 1 synchronizes with image 2 and 3
ELSE
   SYNC IMAGES(1)  ! image 2 and 3 synchronize with image 1
END IF
   !
   ! use data which is prepared by image 1
   !
```

The following figure shows the behavior of each image.

```
        image 1           image 2           image 3
          |                 |                 |
          |                 |                 |
      preparing         SYNC IMAGES(1)        |
      coarray data       |  |           SYNC IMAGES(1)
          |              |  |             |   |
      SYNC IMAGES(*) ---+  |             |   |
          |          -----------------+     |
          |                 |                 |
          |                 |                 |
          |                 |                 |
```
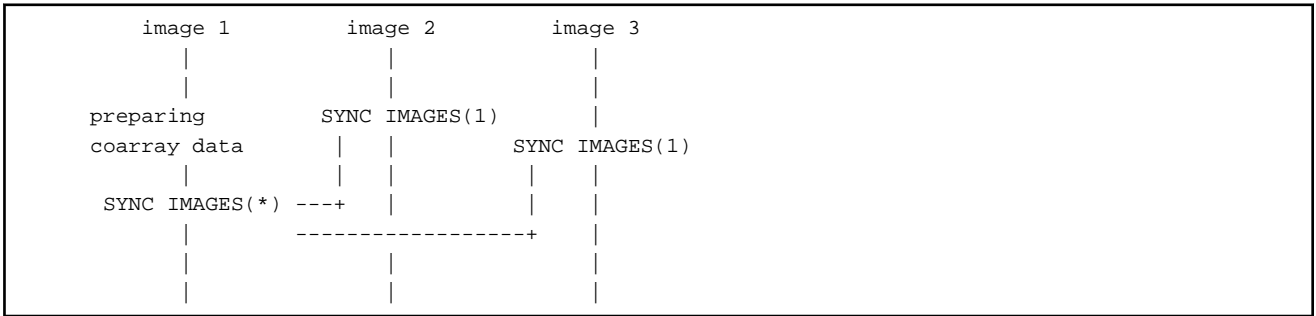
The correct result is obtained without synchronization between images 2 and 3 because it is required that images preparing data, image 1, should synchronize with images using data, images 2 and 3.

STAT= specifier of SYNC IMAGES statement

If the STAT= specifier is specified for a SYNC IMAGES statement, the following value is returned.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.
1713	The image control statement cannot be executed in the CRITICAL construct.
1720	The same image index cannot be specified more than once in the SYNC IMAGES statement.
1722	An image index which is specified for the SYNC IMAGES statement should be a positive integer and not exceed the number of images.
1723	A deadlock was detected on this image.
1731	The SYNC IMAGES statement was terminated because the termination process was started on other images. (The return value is STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)
1752	An error was detected in the communication process for the SYNC IMAGES statement.

2.3.3 SYNC MEMORY Statement

The SYNC MEMORY statement divides statements into segments and completes memory operations in the proceeding segment on the image.

You need to do things such as constructing synchronization mechanism by means of atomic subroutine so as to order the execution sequence of segments on different images which are divided by the SYNC MEMORY statement.

STAT= specifier of SYNC MEMORY statement

If the STAT= specifier is specified for the SYNC MEMORY statement, the following value is returned.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.
1713	The image control statement cannot be executed in the CRITICAL construct.
1752	An error was detected in the communication process for the SYNC MEMORY statement.

2.3.4 ALLOCATE/DEALLOCATE Statement

If coarrays are specified in the ALLOCATE or DEALLOCATE statement, it is necessary that the same coarrays should be specified in the statement on all images in the same format in terms of the size, the order and the number of the coarrays. During the execution of the ALLOCATE or DEALLOCATE statement which contains coarrays, implicit synchronization is performed among all images.

The ALLOCATE and DEALLOCATE statements have the same effect of the SYNC MEMORY statement.

Deallocation by the DEALLOCATE statement or implicit deallocation should not be executed for locked LOCK_TYPE coarrays without unlocking operations. It is necessary that the lock variable should be unlocked by the UNLOCK statement before deallocation.

The following example shows that the ALLOCATE and DEALLOCATE statements synchronize implicitly and have the same effect of the SYNC MEMORY statement.

```
INTEGER::MY
REAL,SAVE::X[*]
REAL,ALLOCATABLE::Y[:]
...                              |
MY=THIS_IMAGE()                  | segment (1)
IF(MY==1)X=1.0                   |
ALLOCATE(Y[*])                   +------------------
IF(MY==2)PRINT *,X[1]            |
   !                             |
   ! process using coarray Y     | segment (2)
   !                             |
IF(MY==3)X=3.0                   |
DEALLOCATE(Y)                    +------------------
IF(MY==1)PRINT *,X[3]            | segment (3)
...                              |
```

The following figure shows the behavior of each image.

```
   image 1                 image 2                 image 3
     |                       |                       |                       |
     |                       |                       |                       |
     |                       |                       |                       |
   X=1.0                     |                       |                       | segment (1)
     |                       |                       |                       |
+-------+---------------------+---------------------+-------+                 |
| ALLOCATE(Y[*])         ALLOCATE(Y[*])          ALLOCATE(Y[*]) | synchronize +------------
+-------+---------------------+---------------------+-------+                 |
     |                       |                       |                       |
     |                 PRINT *,X[1]                   |                       |
     |                       |                       |                       |
 process using coarray Y  process using coarray Y  process using coarray Y  | segment (2)
     |                       |                       |                       |
     |                       |                     X=3.0                     |
     |                       |                       |                       |
+-------+---------------------+---------------------+-------+                 |
| DEALLOCATE(Y)          DEALLOCATE(Y)           DEALLOCATE(Y) | synchronize +------------
+-------+---------------------+---------------------+-------+                 |
     |                       |                       |                       |
   PRINT *,X[3]              |                       |                       | segment (3)
     |                       |                       |                       |
```

The program is divided into three segments by ALLOCATE and DEALLOCATE statements on each image, and they are ordered as segment (1), segment (2) and segment (3).

Since the ALLOCATE and DEALLOCATE statements are implicitly synchronized among all images, segments in different images are also ordered as segment (1), segment (2) and segment (3). Therefore, the PRINT statement in segment (2) on image 2 outputs "1.0" and the PRINT statement in segment (3) on image 1 outputs "3.0".

The following example shows that ALLOCATE and DEALLOCATE statements are executed while the STOP statement is executed on image 3 by mistake. Then the variable which is specified with STAT= specifier of the ALLOCATE and DEALLOCATE statements is set to the value STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTAN_ENV.

```
INTEGER::MY,ST
REAL,SAVE::X[*]=0.0
REAL,ALLOCATABLE::Y[:],Z[:]
...
MY=THIS_IMAGE()
ALLOCATE(Y[*])
IF(MY==3)THEN
  STOP
END IF
ALLOCATE(Z[*],STAT=ST)
IF(ST.EQ.STAT_STOPPED_IMAGE)THEN
  PRINT *,MY,' ALLOC ERROR'
END IF
DEALLOCATE(Y,STAT=ST)
IF(ST.EQ.STAT_STOPPED_IMAGE)THEN
  PRINT *,MY,' DEALLOC ERROR'
END IF
...
STOP
```

The following figure shows the behavior of each image.
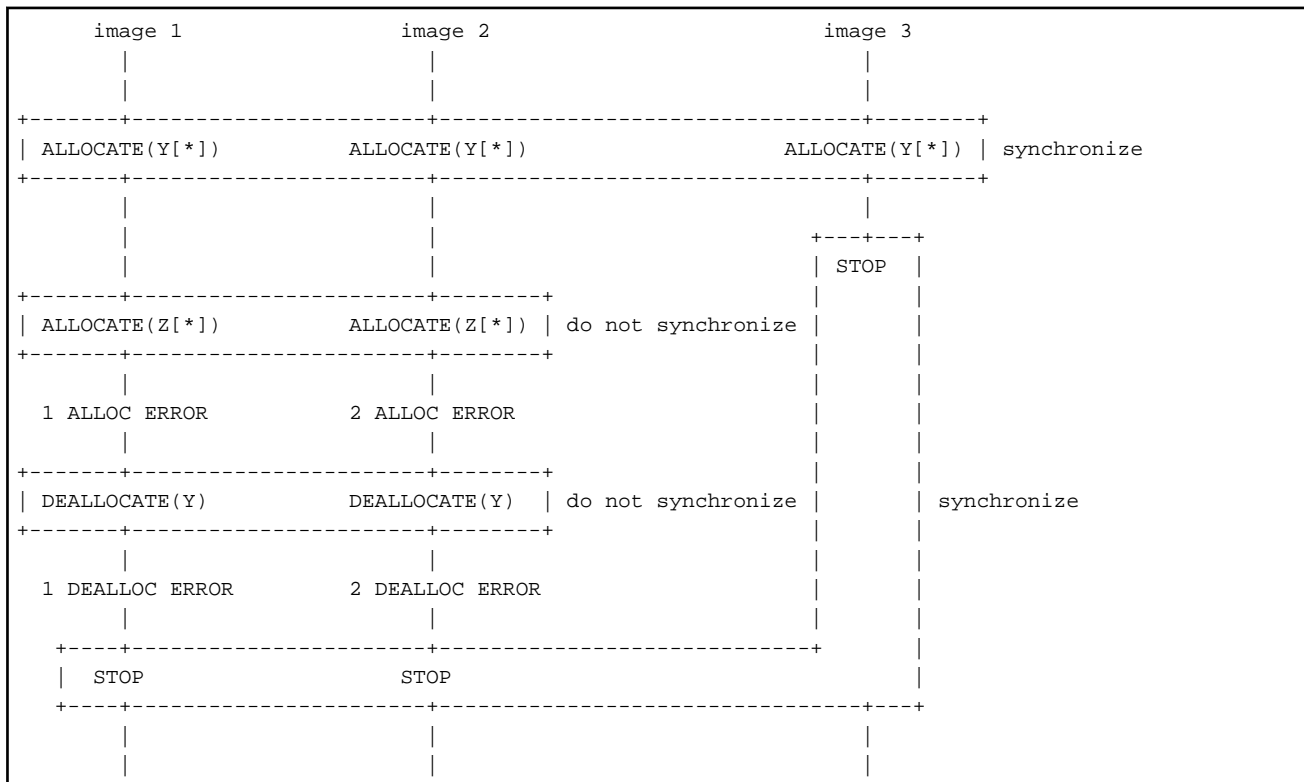
The first ALLOCATE statement is executed on all the images correctly. The second ALLOCATE statement is executed on images 1 and 2 while the STOP statement is executed on image 3. Then, the variable specified with STAT= specifier in the ALLOCATE statement on images 1 and 2 is set to the STAT_STOPPED_IMAGE.

Similarly, the variable specified with STAT= specifier in the DEALLOCATE statement on images 1 and 2 is set to the STAT_STOPPED_IMAGE.

After that, the STOP statements on images 1 and 2 are executed and synchronized with the STOP statement which has been already executed on image 3. Then the program is terminated.

STAT= specifier of ALLOCATE and DEALLOCATE statements

The following value is returned in the variable specified with the STAT= specifier in the ALLOCATE and DEALLOCATE statements for coarray in addition to the values output in the case of the statements for noncoarray.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning	
0	Normal end.	
1713	The image control statement cannot be executed in the CRITICAL construct.	
1723	A deadlock was detected on this image.	
1731	The ALLOCATE or DEALLOCATE statement was terminated because the termination process was started on other images. (The return value is STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)	
1705	ALLOCATE statement	The image control statement or different ALLOCATE statement cannot be executed on the other images during the allocation process for the coarray area.
1706	ALLOCATE statement	A synchronization error was detected in the allocation process for the coarray area.
1707	ALLOCATE statement	An error was detected in the allocation process for the coarray area.
1741	DEALLOCATE statement	The image control statement or different DEALLOCATE statement cannot be executed on the other images during the deallocation process for the coarray area.
1742	DEALLOCATE statement	A synchronization error was detected in the deallocation process for the coarray area.
1743	DEALLOCATE statement	An error was detected in the deallocation process for the coarray area.
1744	DEALLOCATE statement	An error was detected in the deallocation process for the coarray area.
1745	DEALLOCATE statement	The deallocation process for the lock variable cannot be executed without unlocking it.

2.3.5 LOCK/UNLOCK Statement

The LOCK and UNLOCK statements are used for making an exclusive control mechanism.

One of the locking resources is assigned to a lock variable when it is allocated.

The number of the locking resources is decided at the start of the program and cannot be changed at runtime.

You need to specify the number of lock variables by the environment variable FLIB_COARRAY_LOCKNO appropriately if the estimated number of lock variables required in the program exceeds the default value.

The following diagnostic message is output and an error termination process is started if the lock resources are insufficient at the allocation of lock variables.

```
jwe1738i-s The lock resources were insufficient.
```

If an exclusive control mechanism among multiple images is made by means of the LOCK or UNLOCK statement with a single lock variable, the same lock variable in the same image should be specified on all images because the lock variable is a coarray.

If the lock variable which is specified in the LOCK or UNLOCK statement on each image is the lock variable not on the same image but on each image, the exclusive control mechanism cannot be established among those images.

The following example shows that an exclusive control mechanism is established by a single lock variable.

```
USE,INTRINSIC::ISO_FORTRAN_ENV
TYPE(LOCK_TYPE)::X[*]
INTEGER,SAVE::SUM[*]=0
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

LOCK(X[NUM])
  SUM[NUM]=SUM[NUM]+1
UNLOCK(X[NUM])

SYNC ALL
IF(MY.EQ.NUM)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

The following figure shows the behavior of each image when the number of images is three.

```
      image 1                   image 2                   image 3
         |                         |                         |
       SUM=0                     SUM=0                     SUM=0
         |                         |                         |
   lock variable X           lock variable X  +----> lock variable X <--+
         |                         |           | +-->         |         |
         |                         |           | |            |         |
       MY=1                      MY=2          | |          MY=3        |
       NUM=3                     NUM=3         | |          NUM=3       |
         |                         |           | |            |         |
         |                         |           | |            |         |
         |          +-----------------------+ |            |         |
         |          |               |         | |            |         |
         |          |               |         | |            |         |
         |          |               |         | |            |         |
   LOCK(X[3]) --+       LOCK(X[3]) ------+       LOCK(X[3])------+
         :                         :                         |
   waiting for lock          waiting for lock          acquired lock
         :                         :                         |
         :                         :                       SUM=1
         :                         :                         |
         :                 acquired lock <--------- UNLOCK(X[3])
         :                         |                         |
         :                       SUM[3]=2       +---------+---------+
         :                         |            |      SYNC ALL     |
   acquired lock <--------- UNLOCK(X[3])        |                   |
         |                         |            |                   |
       SUM[3]=3          +---------+-------------+                   |
         |               |      SYNC ALL                            |
   UNLOCK(X[3])          |                                          |
         |               |                                          |
+-------+------------+                                          |
|    SYNC ALL        |                                          | synchronize
+-------+--------------------+--------------------+---------+
         |                         |                         |
         |                         |               NUMBER OF IMAGES = 3
         |                         |                         |
         |                         |                         |
```

The exclusive control mechanism works correctly among all images because the lock variable specified in the LOCK and UNLOCK statements is X[NUM] of the last image. The three images request for the single lock at the same time. It is indeterminate which image acquires the lock. The above example shows that image 3, image 2 and image 1 acquire the lock in this order.

The PRINT statement on image 3 outputs "NUMBER OF IMAGES = 3".

The following example shows that each image runs without the exclusive control by a lock variable.
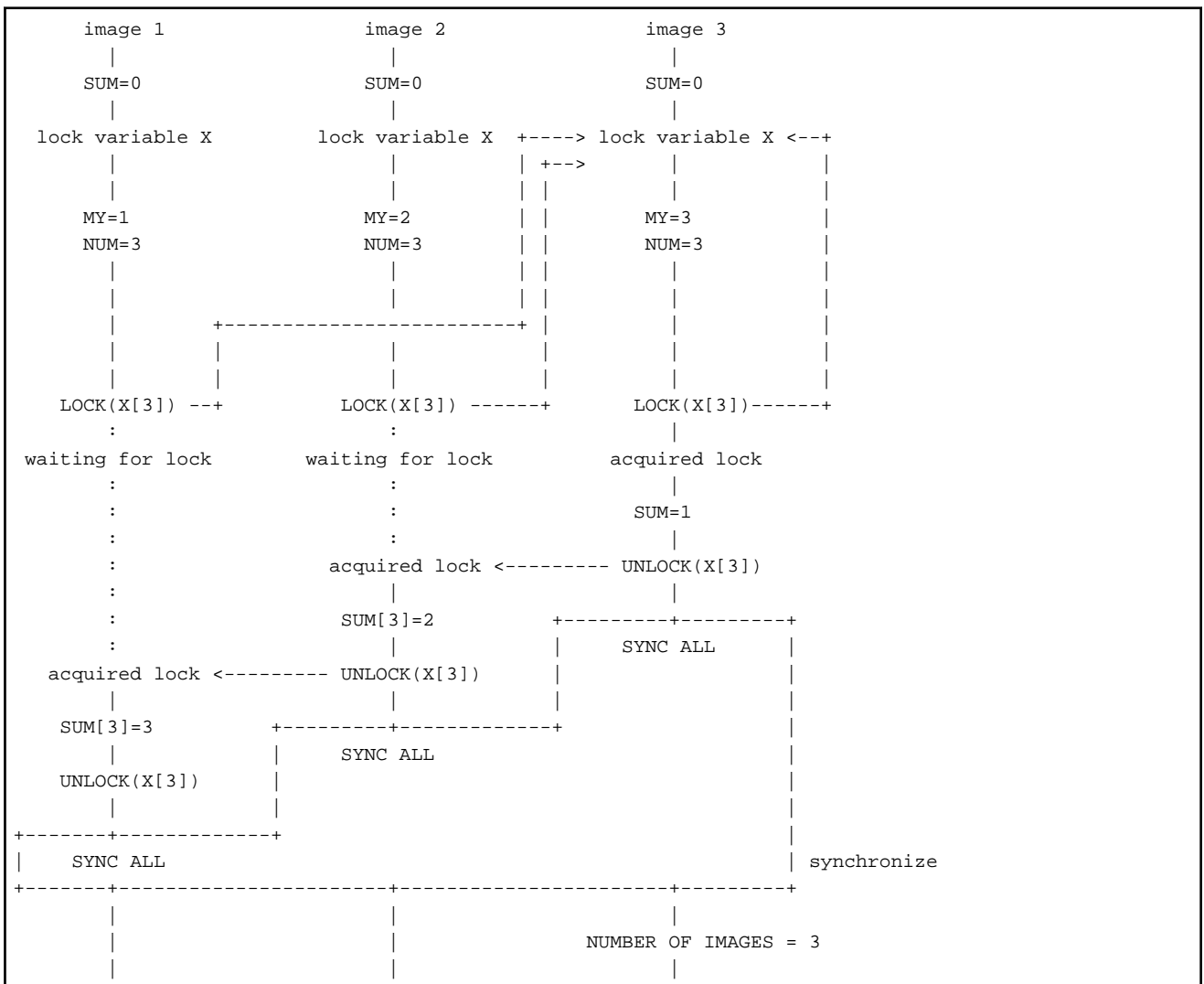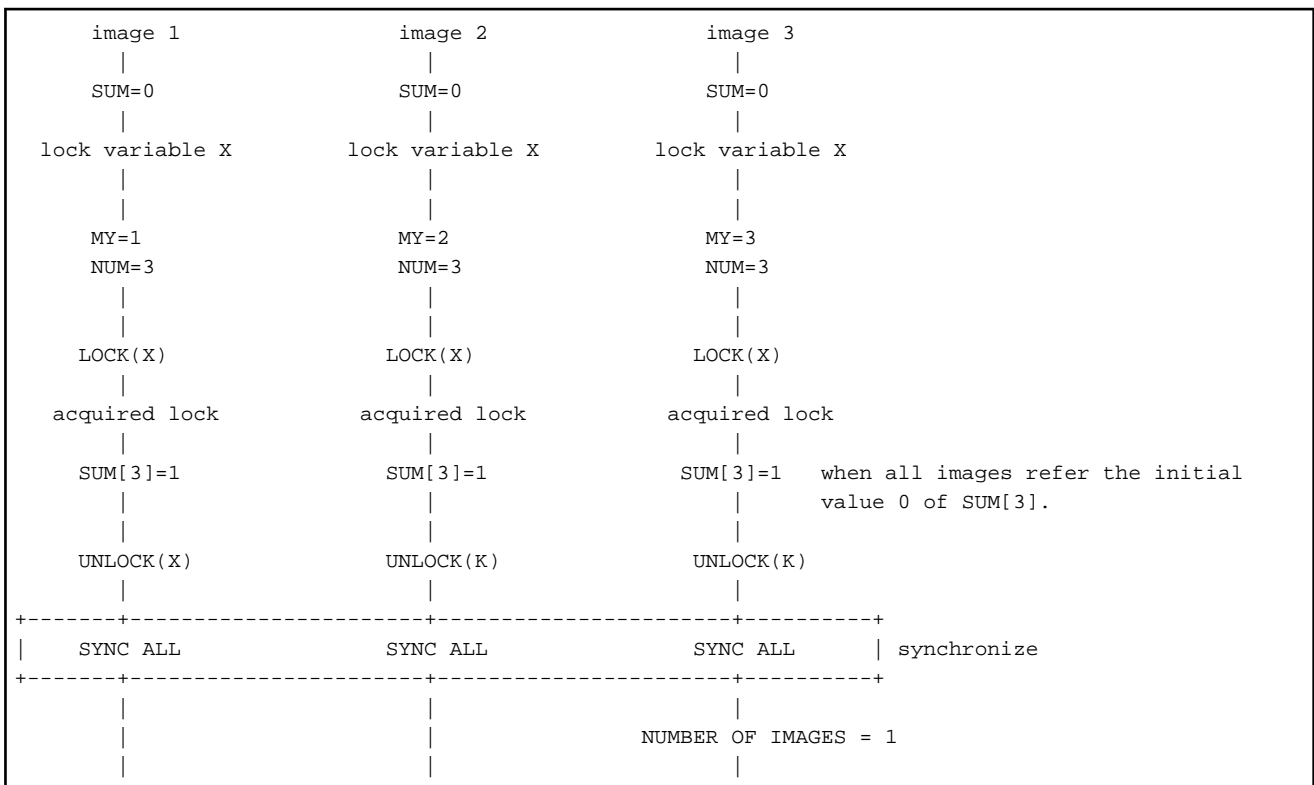
```
USE,INTRINSIC::ISO_FORTRAN_ENV
TYPE(LOCK_TYPE)::X[*]
INTEGER,SAVE::SUM[*]=0
INTEGER::MY,NUM
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

LOCK(X)
  SUM[NUM]=SUM[NUM]+1
UNLOCK(X)

SYNC ALL
IF(MY.EQ.NUM)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

The following figure shows the behavior of each image.

```
      image 1                    image 2                    image 3
        |                          |                          |
      SUM=0                      SUM=0                      SUM=0
        |                          |                          |
  lock variable X            lock variable X            lock variable X
        |                          |                          |
        |                          |                          |
      MY=1                       MY=2                       MY=3
      NUM=3                      NUM=3                      NUM=3
        |                          |                          |
        |                          |                          |
      LOCK(X)                    LOCK(X)                    LOCK(X)
        |                          |                          |
   acquired lock              acquired lock              acquired lock
        |                          |                          |
      SUM[3]=1                   SUM[3]=1                   SUM[3]=1    when all images refer the initial
        |                          |                          |                value 0 of SUM[3].
        |                          |                          |
      UNLOCK(X)                  UNLOCK(K)                  UNLOCK(K)
        |                          |                          |
+-------+--------------------------+--------------------------+---------+
|     SYNC ALL                   SYNC ALL                   SYNC ALL   | synchronize
+-------+--------------------------+--------------------------+---------+
        |                          |                          |
        |                          |              NUMBER OF IMAGES = 1
        |                          |                          |
```

Three images can acquire the individual lock at the same time because the lock variable specified in the LOCK and UNLOCK statements is a local variable X on each image. Therefore, the exclusive control mechanism cannot work among them.

Image 3 outputs "NUMBER OF IMAGES = 1" in the above sequence because the expected exclusive control could not be established.

STAT= specifier of the LOCK and UNLOCK statements

The following value is returned if the STAT= specifier is specified in the LOCK and UNLOCK statements.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.

Return value		Meaning
1713		The image control statement cannot be executed in the CRITICAL construct.
1722		An image index which is specified for the LOCK or UNLOCK statement should be a positive integer and not exceed the number of images.
1735		The variable which is not a lock variable cannot be specified for the LOCK or UNLOCK statement.
1732	LOCK statement	A lock variable which has been locked by an image cannot be locked by the same image. (The return value is STAT_LOCKED in the intrinsic module ISO_FORTRAN_ENV.)
1736	LOCK statement	An error was detected in the lock process.
1733	UNLOCK statement	A lock variable which has been locked by other images cannot be unlocked. (The return value is STAT_LOCKED_OTHER_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)
1734	UNLOCK statement	Unlocked lock variable cannot be unlocked. (The return value is STAT_UNLOCKED in the intrinsic module ISO_FORTRAN_ENV.)
1737	UNLOCK statement	An error was detected in the unlock process.

2.3.6 CRITICAL Construct

A block enclosed by the CRITICAL statement and the END CRITICAL statement should not be executed by multiple images at the same time. The following diagnostic message is output and the error termination process is started when an image control statement inhibited in the CRITICAL construct is executed.

```
jwe1713i-s The image control statement cannot be executed in the CRITICAL construct.
```

When the CRITICAL construct is executed on multiple images, it is indeterminate on which image it is executed first. All the intended execution on the images, however, is completed finally.

The following example shows that the number of active images is calculated by adding 1 to the coarray on image 3 by each image.

While an image executes the addition operation, the other image is prohibited from executing it by using the CRITICAL construct.

```
USE,INTRINSIC::ISO_FORTRAN_ENV
INTEGER,SAVE::SUM[*]=0
INTEGER::MY,NUM
MY=THIS_IMAGE()
NUM=NUM_IMAGES()

CRITICAL
  SUM[3]=SUM[3]+1
END CRITICAL

SYNC ALL
IF(MY.EQ.3)THEN
  PRINT *,'NUMBER OF IMAGES = ',SUM
END IF
```

The following figure shows the behavior of each image.

```
    image 1                 image 2                 image 3
       |                       |                       |
    SUM=0                   SUM=0                   SUM=0
```

```
        |                          |                          |
     CRITICAL                    CRITICAL                    CRITICAL
      waiting                     waiting                 possible to run
        :                          :                          |
        :                          :                        SUM=1
        :                          :                          |
        :                 possible to run <--------- END CRITICAL
        :                          |                          |
        :                       SUM[3]=2        +---------+---------+
        :                          |            |     SYNC ALL      |
 possible to run <--------- END CRITICAL       |                  |
        |                          |            |                  |
     SUM[3]=3          +--------+------------+  |                  |
        |              |     SYNC ALL        |  |                  |
   END CRITICAL        |                     |  |                  |
        |              |                     |  |                  |
+-------+------------+ |                     |  |                  |
|     SYNC ALL                               |                     | synchronize
+-------+--------------------+----------------------+---------+
        |                          |                   |
        |                          |           NUMBER OF IMAGES = 3
        |                          |                   |
```
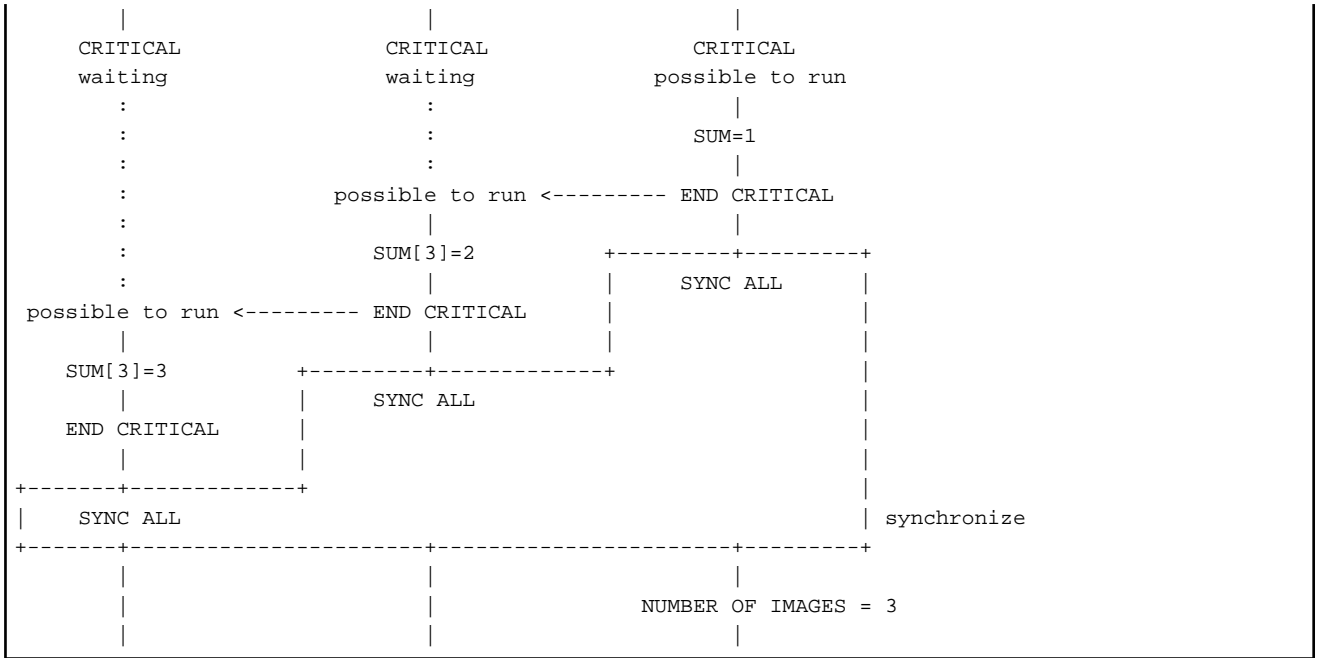
2.3.7 Atomic Subroutines

The atomic subroutines define or refer to variables atomically.

STAT argument of atomic subroutines

The following value is returned if the STAT argument is specified when it is called.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.
1722	An image index which is specified for the atomic subroutine should be a positive integer and not exceed the number of images.

2.3.8 Collective Subroutines

The following collective subroutines are provided:

- CO_MAX intrinsic subroutine

- CO_MIN intrinsic subroutine

- CO_SUM intrinsic subroutine

The collective subroutines should be called on all images in the same format. That is, the specified coarray should have the same name, type, and size. The value of the RESULT_IMAGE argument should be the same if specified.

The following example shows that the total value of elements of the coarray on all images is set on all images by the CO_SUM intrinsic subroutine.
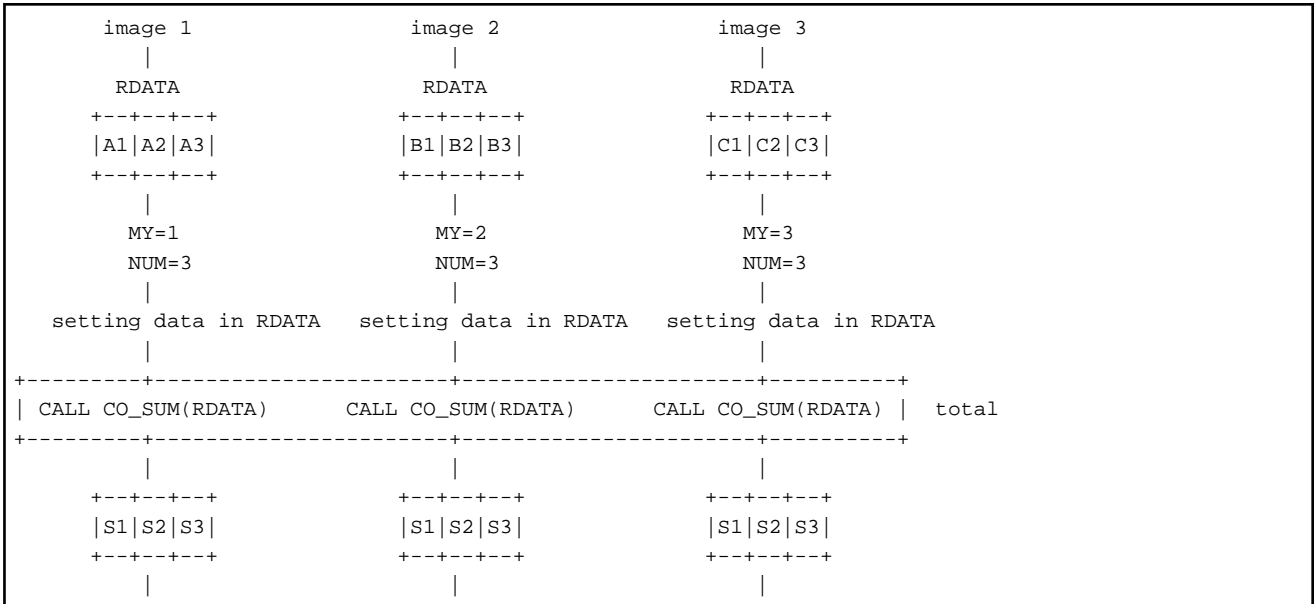
```
PROGRAM MAIN
  USE ISO_FORTRAN_ENV
  IMPLICIT NONE
  REAL(8),SAVE::RDATA(3)[*]
  INTEGER::MY,NUM,I,S
```

```
   MY=THIS_IMAGE()
   NUM=NUM_IMAGES()
         ...          ! setting data in coarray RDATA
   CALL CO_SUM(RDATA)
         ...
END
```

The following figure shows the behavior of each image.

```
      image 1                  image 2                  image 3
        |                        |                        |
      RDATA                    RDATA                    RDATA
    +--+--+--+               +--+--+--+               +--+--+--+
    |A1|A2|A3|               |B1|B2|B3|               |C1|C2|C3|
    +--+--+--+               +--+--+--+               +--+--+--+
        |                        |                        |
      MY=1                     MY=2                     MY=3
      NUM=3                    NUM=3                    NUM=3
        |                        |                        |
   setting data in RDATA   setting data in RDATA   setting data in RDATA
        |                        |                        |
+---------+--------------------+--------------------+----------+
| CALL CO_SUM(RDATA)      CALL CO_SUM(RDATA)      CALL CO_SUM(RDATA) |  total
+---------+--------------------+--------------------+----------+
        |                        |                        |
    +--+--+--+               +--+--+--+               +--+--+--+
    |S1|S2|S3|               |S1|S2|S3|               |S1|S2|S3|
    +--+--+--+               +--+--+--+               +--+--+--+
        |                        |                        |
```

The total value of all elements of RDATA is set in each RDATA on all images by calling the CO_SUM intrinsic subroutine.

The following example shows that the total value of all elements of the coarray on all images is set on the image which is specified as the value of the RESULT_IMAGE argument.
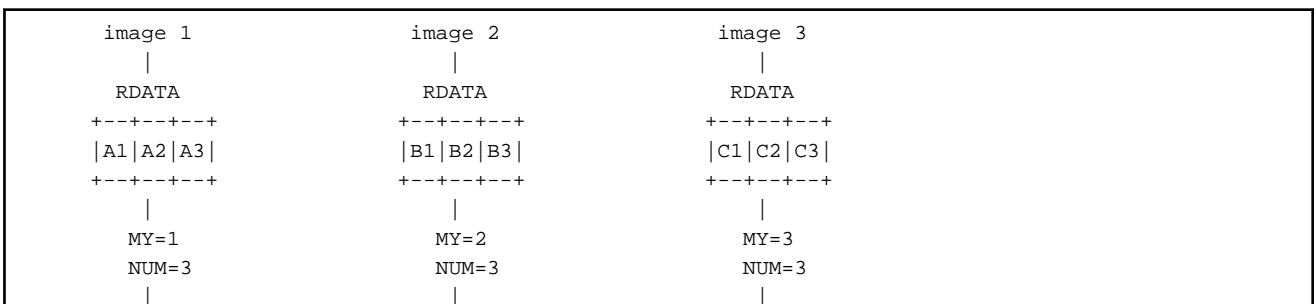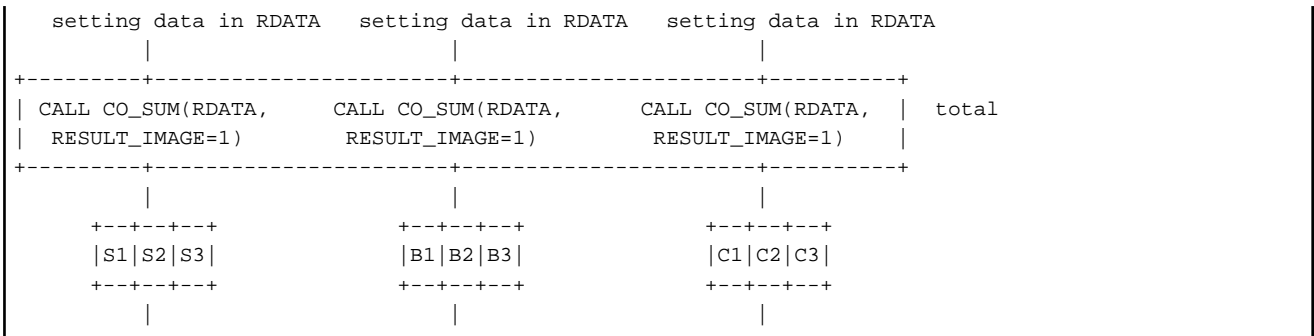
```
PROGRAM MAIN
   USE ISO_FORTRAN_ENV
   IMPLICIT NONE
   REAL(8),SAVE::RDATA(3)[*]
   INTEGER::MY,NUM,I,S

   MY=THIS_IMAGE()
   NUM=NUM_IMAGES()
         ...          ! setting data in RDATA
   CALL CO_SUM(RDATA,RESULT_IMAGE=1)
         ...
END
```

The following figure shows the behavior of each image.

```
      image 1                  image 2                  image 3
        |                        |                        |
      RDATA                    RDATA                    RDATA
    +--+--+--+               +--+--+--+               +--+--+--+
    |A1|A2|A3|               |B1|B2|B3|               |C1|C2|C3|
    +--+--+--+               +--+--+--+               +--+--+--+
        |                        |                        |
      MY=1                     MY=2                     MY=3
      NUM=3                    NUM=3                    NUM=3
        |                        |                        |
```

```
    setting data in RDATA    setting data in RDATA    setting data in RDATA
           |                         |                        |
+---------+----------------------+-----------------------+----------+
| CALL CO_SUM(RDATA,      CALL CO_SUM(RDATA,       CALL CO_SUM(RDATA,  |  total
|   RESULT_IMAGE=1)         RESULT_IMAGE=1)          RESULT_IMAGE=1)   |
+---------+----------------------+-----------------------+----------+
           |                         |                        |
      +--+--+--+                +--+--+--+               +--+--+--+
      |S1|S2|S3|                |B1|B2|B3|               |C1|C2|C3|
      +--+--+--+                +--+--+--+               +--+--+--+
           |                         |                        |
```

After the call of the CO_SUM subroutine, the total value of all elements of RDATA on all images is set on image 1, which is specified as the value of RESULT_IMAGE, and the coarrays on images 2 and 3 are not changed.

STAT argument of the collective subroutines

The following value is returned if the STAT argument is specified in the collective subroutines.

Values except for 0 are runtime diagnostic message numbers. Note that you could not get a correct value if you use a 1-byte integer variable to save a return value.

Return value	Meaning
0	Normal end.
1713	The image control statement cannot be executed in the CRITICAL construct.
1722	An image index which is specified for the collective subroutine should be a positive integer and not exceed the number of images.
1723	A deadlock was detected on this image.
1731	The collective subroutine was terminated because the termination process was started on other images. (The return value is STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)
1751	An error was detected in the synchronization process for the collective subroutine.
1752	An error was detected in the communication process for the collective subroutine.

2.3.9 MOVE_ALLOC Intrinsic Subroutine

The call of the MOVE_ALLOC intrinsic subroutine for coarrays should be performed from the same statement on all images. Each coarray specified as the FROM or TO argument should have the same allocation status on all images.

An implicit synchronization among all images is performed when the MOVE_ALLOC intrinsic subroutine with coarrays is called.

Execution of the MOVE_ALLOC intrinsic subroutine without unlocking by the UNLOCK statement is prohibited if the coarray specified as the TO argument is a locked lock variable.

It is necessary that the lock variable should be unlocked by the UNLOCK statement before execution of the MOVE_ALLOC intrinsic subroutine.

The following example shows that the MOVE_ALLOC intrinsic subroutine is called with coarrays having allocated data objects specified as the FROM and TO arguments. The data object of the FROM argument is moved to the TO argument.

```
PROGRAM MAIN
  USE ISO_FORTRAN_ENV
  TYPE TDATA
    INTEGER::CMP
  END TYPE
  TYPE(TDATA),ALLOCATABLE::F_OBJ[:]
  TYPE(TDATA),ALLOCATABLE::T_OBJ[:]
  INTEGER::MY,NUM
  MY=THIS_IMAGE()
```

```
   NUM=NUM_IMAGES()

   ALLOCATE(F_OBJ[*])
   ALLOCATE(T_OBJ[*])

   T_OBJ%CMP=99
   F_OBJ%CMP=MY

   CALL MOVE_ALLOC(FROM=F_OBJ,TO=T_OBJ)

   PRINT *,T_OBJ%CMP
```
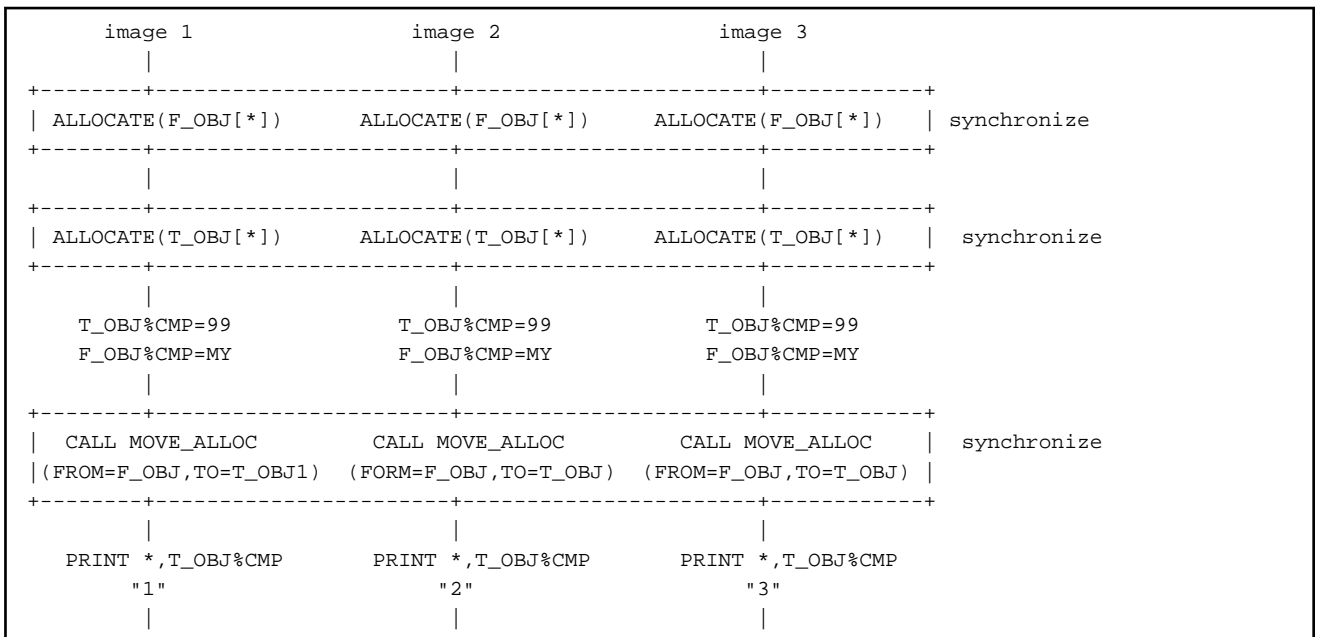
The following figure is the behavior of each image.

```
       image 1                  image 2                  image 3
          |                        |                        |
+--------+---------------------+---------------------+------------+
| ALLOCATE(F_OBJ[*])      ALLOCATE(F_OBJ[*])     ALLOCATE(F_OBJ[*]) |  synchronize
+--------+---------------------+---------------------+------------+
          |                        |                        |
+--------+---------------------+---------------------+------------+
| ALLOCATE(T_OBJ[*])      ALLOCATE(T_OBJ[*])     ALLOCATE(T_OBJ[*]) |  synchronize
+--------+---------------------+---------------------+------------+
          |                        |                        |
    T_OBJ%CMP=99             T_OBJ%CMP=99             T_OBJ%CMP=99
    F_OBJ%CMP=MY             F_OBJ%CMP=MY             F_OBJ%CMP=MY
          |                        |                        |
+--------+---------------------+---------------------+------------+
|  CALL MOVE_ALLOC         CALL MOVE_ALLOC          CALL MOVE_ALLOC  |  synchronize
|(FROM=F_OBJ,TO=T_OBJ1)  (FORM=F_OBJ,TO=T_OBJ)  (FROM=F_OBJ,TO=T_OBJ) |
+--------+---------------------+---------------------+------------+
          |                        |                        |
    PRINT *,T_OBJ%CMP        PRINT *,T_OBJ%CMP         PRINT *,T_OBJ%CMP
        "1"                      "2"                       "3"
          |                        |                        |
```

Each image moves the data object of the coarray F_OBJ to T_OBJ by calling the MOVE_ALLOC intrinsic subroutine. Therefore, the PRINT statement on each image outputs the image index of each image.

📖 Note

The following errors may occur in the MOVE_ALLOC intrinsic subroutine with coarrays.

Return value	Meaning
1306	In MOVE_ALLOC intrinsic subroutine, The character length of the FROM and TO must be the same.
1713	The image control statement cannot be executed in the CRITICAL construct.
1723	A deadlock was detected on this image.
1731	The MOVE_ALLOC intrinsic subroutine was terminated because the termination process was started on other images. (The return value is STAT_STOPPED_IMAGE in the intrinsic module ISO_FORTRAN_ENV.)
1743	An error was detected in the deallocation process for the coarray area.
1744	An error was detected in the deallocation process for the coarray area.
1745	The deallocation process for the lock variable cannot be executed without unlocking it.
1751	An error was detected in the synchronization process for the MOVE_ALLOC intrinsic subroutine.

Return value	Meaning
1752	An error was detected in the communication process for the MOVE_ALLOC intrinsic subroutine.

2.4 Environment Variables at Runtime

This section describes optional environment variables that control execution of COARRAY programs.

Environment variables which are specified in submitted job scripts are passed to each process. It is also possible to set the environment variables by the -x option of the mpiexec(1) command.

FLIB_COARRAY_DEADLOCK_TIMEOUT *time*

> The FLIB_COARRAY_DEADLOCK_TIMEOUT environment variable changes the time limit to detect deadlocks.
>
> The value of *time* is in second and satisfies 0<=*time*<=86400 (86400 seconds is 24 hours). The deadlock is not detected if the value of *time* is 0.
>
> If this is not specified, it is assumed that the value 0, the default value of the Fortran system, is set to *time*.

> ### 📋 Example
>
> The following example shows how to set the FLIB_COARRAY_DEADLOCK_TIMEOUT environment variable.
>
> ```
> $ export FLIB_COARRAY_DEADLOCK_TIMEOUT=60
> $ mpiexec ./a.out
> ```
>
> A deadlock is detected if there is no response for 60 seconds at waiting for synchronization among images in this example.

FLIB_COARRAY_LOCKNO *no*

> The FLIB_COARRAY_LOCKNO environment variable changes the number of the lock variables, which are available locking resources, in the program.
>
> The value of *no* satisfies 100<=*no*<=32767.
>
> If this is no specified, it is assumed that the value 1024, the default value in the Fortran system, is set to *no*.
>
> The following example shows how to use the FLIB_COARRAY_LOCKNO environment variable.

> ### 📋 Example
>
> ```
> $ export FLIB_COARRAY_LOCKNO=3000
> $ mpiexec ./a.out
> ```

FLIB_COARRAY_ERRMSG *out*

> The FLIB_COARRAY_ERRMSG environment variable controls the output of the additional header information for COARRAY programs.
>
> The valid value for *out* is 0 or 1.
>
> If the value of *out* is 0, the COARRAY header is output. If it is 1, the COARRAY header is not output.
>
> If this is not specified, it is assumed that the value 0, the default value in the Fortran system, is set to *out*.

> ### 📋 Example
>
> The following example shows how to use the FLIB_COARRAY_ERRMSG environment variable.
>
> ```
> $ export FLIB_COARRAY_ERRMSG=1
> $ mpiexec ./a.out
> ```

The COARRAY header is not output in this example.

・・・

2.4.1 Environment Variable for Specifying Argument of Execution Command

The FORT90L environment variable in job scripts is passed to the COARRAY program as its execution argument.

Chapter 3 Using MPI from COARRAY Programs

MPI subroutines and functions are available in COARRAY programs in this system.

This chapter gives usage of MPI in COARRAY programs.

3.1 Linking and Execution

The following describes how to compile, link, and execute COARRAY programs using MPI.

3.1.1 How to Link

Use mpifrt (or mpifrtpx) with the -Ncoarray option to compile and link COARRAY programs using MPI.

It is necessary to use mpifrt (or mpifrtpx) with the -Ncoarray option to link the following object programs: object programs which are compiled by frt (or frtpx) with the -Ncoarray option and object programs which are compiled by mpifrt (or mpifrtpx) or mpifcc (or mpifccpx).

Use mpiFCC (or mpiFCCpx) with --linkcoarray to link C++ object programs.

See "MPI User's Guide" for details of mpifrt, mpifrtpx, mpiFCC, and mpiFCCpx.

3.1.2 How to Execute

Use mpiexec(1) to execute. See "MPI User's Guide" for details.

3.2 Notes on COARRAY Programs Using MPI

The following describes notes on using MPI from COARRAY programs.

- In COARRAY programs in this system, the following subroutines, which are defined in the MPI standards, cannot be used: establishing communication between groups not sharing a communicator and dynamic process creation.

- According to the MPI standards, it is possible to set delete callback subroutines to predefined communicators, datatypes and windows by the following functions: MPI_COMM_SET_ATTR, MPI_TYPE_SET_ATTR, MPI_WIN_SET_ATTR, and MPI_ATTR_PUT. In COARRAY programs in this system, it is impossible to set delete callback subroutines to predefined communicators, datatypes and windows. If they are set, they may be called untimely.

- According to the MPI standards, the MPI_IS_THREAD_MAIN subroutine returns a flag indicating whether the calling thread called the MPI_INIT or MPI_INIT_THREAD subroutine.
  In COARRAY programs in this system, it could return an incorrect flag.

- The MPI profiling interfaces covers MPI subroutines and functions called by this system itself.

- MPI statistical information of COARRAY programs in this system is output when the programs terminate though the information of programs not using coarrays is supposed to be output when the MPI_FINALIZE subroutine is called.

- In MPI statistical information, information about MPI subroutines and functions called by this system itself is accumulated without distinguishing from programs not using coarrays.

Chapter 4 Debugging of COARRAY Programs

Debugging functions provided for Fortran programs are also available in COARRAY programs.

Refer the chapter "Debugging" in "Fortran User's Guide" for details of the debugging functions.

4.1 Runtime Error Messages of MPI

Error messages of MPI could be output at runtime since COARRAY uses the MPI library.

See "6.3.1 Runtime Error Message" for details.

4.2 Output of Debugging Functions

Outputs of the debugging functions are connected to the standard error by default.

The connection can be changed to each image's file by the option of mpiexec(1).

4.2.1 Runtime Output Information

Diagnostic messages, trace back maps and error correction information with a leading COARRAY header are output if a COARRAY program causes runtime errors.

The output of the COARRAY header is controlled by the environment variable FLIB_COARRAY_ERRMSG. See "2.4 Environment Variables at Runtime" for details.

The format of the COARRAY header is as follows:

```
timestamp[image-index/the-number-of-images:process-id]
```

timestamp	The current time obtained by gettimeofday(2) is displayed as the number of seconds that have elapsed since 00:00:00(UTC), January 1st, 1970. It is represented as a 16-digit number, of which 10 digits are for seconds and 6 digits are for microseconds. It is the current time of the compute node on which the diagnostic message is output.
image-index	The image index which outputs the diagnostic message is displayed.
the-number-of-images	The total number of images in the COARRAY program which outputs the diagnostic message is displayed.
process-id	The process id assigned to the image which outputs the diagnostic message is displayed.

📒 Note
. . . .

When multiple images output diagnostic messages, the order of the messages depends on the mpiexec(1).

Note that timestamps on compute nodes could be different from each other since it represents the current time on each compute node.
. . . .

📘 Example
. . . .

The following example is an extraction about image 1 from a diagnostic message where the total number of the image index is 2.

```
1429142357.004211[1/2:21294] jwe1744i-s line 25 An error was detected in the deallocation process for
the COARRAY area.
1429142357.319930[1/2:21294]  error occurs at MAIN__   line 25 loc 0000000000400df5 offset
0000000000000125
1429142357.319948[1/2:21294]  MAIN__      at loc 0000000000400cd0 called from o.s.
1429142357.319964[1/2:21294] jwe0903i-u Error number 1744 was detected. Maximum error count exceeded.
1429142357.320249[1/2:21294] error summary (Fortran)
```

```
1429142357.320259[1/2:21294] error number  error level  error count
1429142357.320268[1/2:21294]   jwe1744i         s           1
1429142357.320272[1/2:21294] total error count = 1
```

4.2.2 Error Control Table Standard Values for COARRAY

The standard values of error items for each error number of COARRAY in this system are shown in the table below.

Table 4.1 Error control table standard values for COARRAY

Error item number	Error count limit	Message count limit	Error item editable	Buffer	Trace back map	Standard correction	Error level
1701 - 1707	1	1	Editable	Not printed	Printed	Yes	s
1711 - 1713	1	1	Editable	Not printed	Printed	Yes	s
1719 - 1722	1	1	Editable	Not printed	Printed	Yes	s
1723	1	1	Not Editable	Not printed	Printed	None	u
1724	1	1	Editable	Not printed	Printed	Yes	s
1731 - 1739	1	1	Editable	Not printed	Printed	Yes	s
1741 - 1745	1	1	Editable	Not printed	Printed	Yes	s
1751 - 1752	1	1	Editable	Not printed	Printed	Yes	s
1790	1	1	Editable	Not printed	Printed	Yes	s

4.2.3 COARRAY Error Processing

The table below lists the standard system corrections and user-defined corrections for runtime errors of COARRAY.

The following arguments are passed to the user-defined error subroutine:

- Return code

- Error number

Table 4.2 Error processing for COARRAY

Error number	Standard correction processing	User-defined correction	Data passed to user-defined subroutine
1701 - 1707	Terminates the program	Not correctable	None
1711 - 1713	Terminates the program	Not correctable	None
1719 - 1722	Terminates the program	Not correctable	None
1724	Terminates the program	Not correctable	None
1731 - 1739	Terminates the program	Not correctable	None
1741 - 1745	Terminates the program	Not correctable	None
1751 - 1752	Terminates the program	Not correctable	None
1790	Terminates the program	Not correctable	None

4.3 Notes on Debugging

The debugging functions of the option -H do not covers coindexed variables.

Chapter 5 Tuning of COARRAY Programs

This chapter describes tuning of COARRAY programs.

5.1 Effective Usage of COARRAY

The following describes the usage of COARRAY with taking the execution performance into account.

It is important to consider transfer time and synchronization in COARRAY programs. Otherwise, the execution performance of the program may become significantly worse than the program not using COARRAY.

5.1.1 Access to Other Images by Coarrays

Coarrays have local objects on each image, which correspond to coindexed objects.

Instead of using coarrays with the image selector ("["), using local objects enables the same performance as the conventional. In contrast, access to data on other images with the image selector is accompanied with significant cost since the access is made through the interconnect.

5.1.1.1 Array Expression and Access to Other Images

Use array expressions to access to other images rather than refer by element.

- Undesirable example

```
REAL,SAVE::AA(1000)[*]
...
DO I=1,1000
  AA(I)[N]=0
END DO
 ...
```

- Desirable example

Array expression are expected to reduce the cost of transfer through the interconnect. However, the reduction could not be expected when transferring multiple small data with gaps. Some measures are necessary to improve the performance. For example, it is effective to make data contiguous by packing data on the source image and extracting it on the destination image.

```
REAL,SAVE::AA(1000)[*]
...
AA(:)[N]=0
...
```

In general, when accessing to different images, an array is divided into some units and transferred one by one through the interconnect rather than the entire array is transferred at once. You do not need to think about the size of the unit because the runtime system determines it appropriately.

5.1.1.2 Access Concentration of Data

Avoid the communication such that all images write in the same image simultaneously.

Give an example in "5.2.5 All-to-All Communication".

5.1.1.3 CO_SUM, CO_MAX, and CO_MIN Intrinsic Subroutines

Use the CO_SUM intrinsic subroutine to get the sum of the data over all images. Use the CO_MAX intrinsic subroutine to get the maximum and the CO_MIN intrinsic subroutine to get the minimum.

The following example describes how to get the sum of the data over all images.

Example

- Undesirable example

```
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    VAL=VAL+VAL[I]
  END DO
END IF
SYNC ALL
```

- Desirable example (Using CO_SUM intrinsic subroutine)

```
SYNC ALL
CALL CO_SUM(VAL,RESULT_IMAGE=1)
```

Note

- The order of operations is not guaranteed.

- If you use collective subroutines, you must use them on all images.

- You must refer the intrinsic module ISO_FORTRAN_ENV.

5.1.2 Usage of Image Control Statement

It is recommended to use the SYNC IMAGES statement rather than the SYNC ALL statement.

See "2.3.2 SYNC IMAGES Statement" for details.

5.2 Example of Collective Communication

The following describes the usage of COARRAY which corresponds to the collective communication in the MPI.

Note that collective subroutines tuned for this system and MPI collective communications are expected to give better execution performance than the following example.

5.2.1 Reduction Operation

Use the collective subroutines CO_SUM, CO_MAX and CO_MIN for the summation, maximum and minimum over all images.

The following describes an example to use the reduction operation LOGICAL AND.

Example

```
LOGICAL,SAVE::VAL[*]
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    VAL=VAL.AND.VAL[I]
  END DO
END IF
```

```
SYNC ALL
...
```
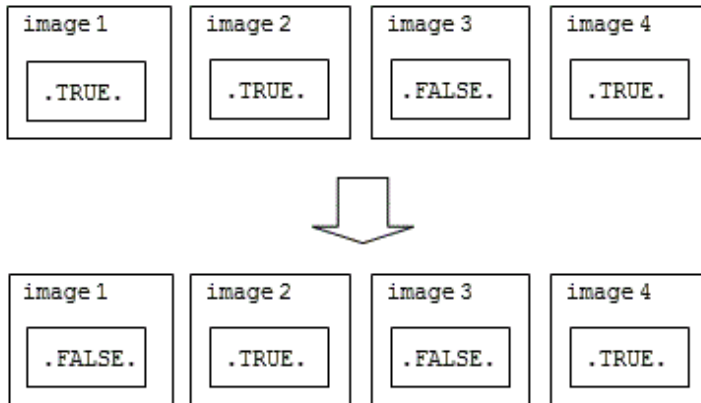

The following describes an example of LOGICAL AND used for an array.

In this example, coarrays are transferred to local arrays on each image in order to transfer at once.

Example

```
LOGICAL,DIMENSION(:),CODIMENSION[:],ALLOCATABLE::VAL
LOGICAL,DIMENSION(:),ALLOCATABLE::TMPVAL
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
ALLOCATE(VAL(MSGSIZE)[*])
ALLOCATE(TMPVAL(MSGSIZE))
...
SYNC ALL
IF(ID==1)THEN
  DO I=2,NIMG
    TMPVAL(1:MSGSIZE)=VAL(1:MSGSIZE)[I]
    VAL(1:MSGSIZE)=VAL(1:MSGSIZE).AND.TMPVAL(1:MSGSIZE)
  END DO
END IF
SYNC ALL
...
```

5.2.2 Broadcast

The following describes an example of broadcast.

Example

```
INTEGER :: NIMG,ID,I,MSGSIZE
REAL(8),DIMENSION(:),CODIMENSION[:],ALLOCATABLE :: ARRAY
...
NIMG= NUM_IMAGES()
ID= THIS_IMAGE()
...
ALLOCATE(ARRAY(MSGSIZE)[*])
...
```

```
SYNC ALL
IF(ID /= 1) THEN
ARRAY(1:MSGSIZE) = ARRAY(1:MSGSIZE)[1]
END IF
SYNC ALL
...
```
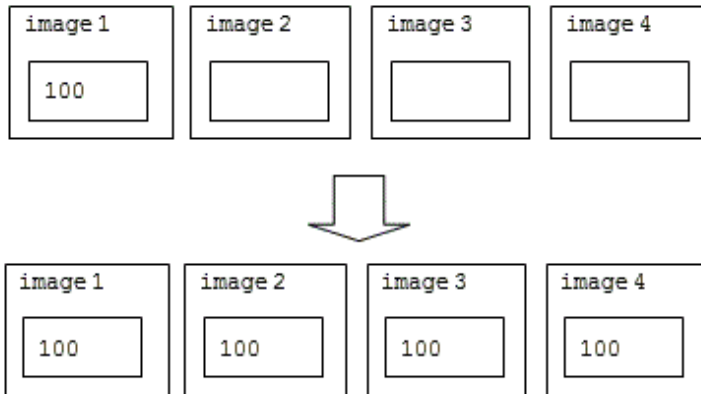

5.2.3 Gather

The following describes an example of gather.

Example

```
INTEGER::NIMG,ID,MSGSIZE
REAL(8),DIMENSION(:),ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::SOURCE
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
ALLOCATE(DEST(MSGSIZE))
ALLOCATE(SOURCE(MSGSIZE,NIMG)[*])
...
SYNC ALL
DEST(1:MSGSIZE,ID)[1] = SOURCE(1:MSGSIZE)
SYNC ALL
...
```

5.2.4 Scatter

The following describes an example of scatter.

Example

```
INTEGER::NIMG,ID,MSGSIZE
REAL(8),DIMENSION(:),ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::SOURCE
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
ALLOCATE(DEST(MSGSIZE))
ALLOCATE(SOURCE(MSGSIZE,NIMG)[*])
...
SYNC ALL
DEST(1:MSGSIZE)=SOURCE(1:MSGSIZE,ID)[1]
SYNC ALL
...
```

5.2.5 All-to-All Communication

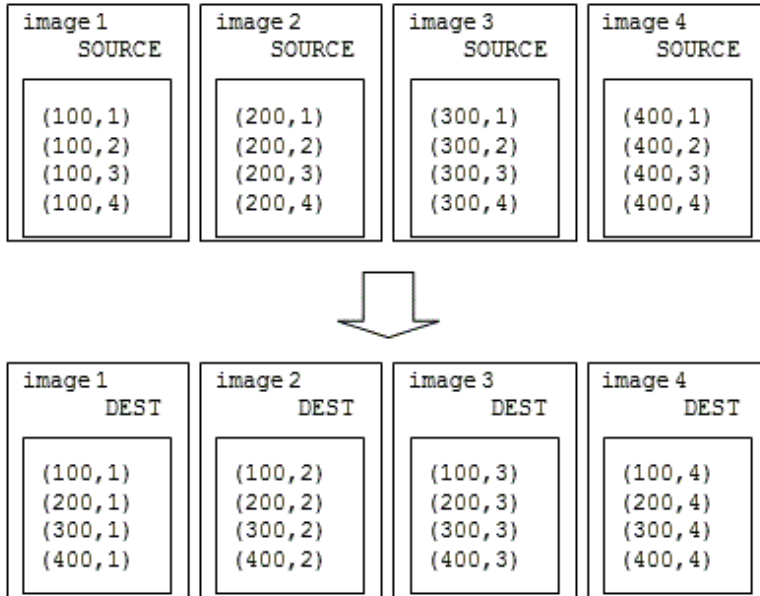The following describes an example of all-to-all communication.

📋 Example

```
INTEGER::NIMG,ID,I,MSGSIZE
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),ALLOCATABLE::SOURCE
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
ALLOCATE(DEST(MSGSIZE,NIMG)[*])
ALLOCATE(SOURCE(MSGSIZE,NIMG))
...
SYNC ALL
DO I=1,NIMG
  DEST(1:MSGSIZE,ID)[I]=SOURCE(1:MSGSIZE,I)
END DO
SYNC ALL
...
```

. . . .

Better execution performance is expected by avoiding the collision of communications with shifting each image's destination as shown below.

📝 Example

. . . .

```
INTEGER::NIMG,ID,I,ISHIFT,MSGSIZE
REAL(8),DIMENSION(:,:),CODIMENSION[:],ALLOCATABLE::DEST
REAL(8),DIMENSION(:,:),ALLOCATABLE::SOURCE
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
ALLOCATE(DEST(MSGSIZE,NIMG)[*])
ALLOCATE(SOURCE(MSGSIZE,NIMG))
...
SYNC ALL
ISHIFT=ID
DO I=1,NIMG
  DEST(1:MSGSIZE,ID)[ISHIFT]=SOURCE(1:MSGSIZE,ISHIFT)
  ISHIFT=ISHIFT+1
  IF(ISHIFT>NIMG)THEN
     ISHIFT=1
  END IF
END DO
SYNC ALL
...
```

. . . .

5.3 Parallel Input and Output

The following describes input and output of the file.

5.3.1 Parallel Input and Output

An example where multiple images create each local file and access to it in parallel

Including the image index in the filename enables you to create respective files.

```
INTEGER,PARAMETER::NMAX=1000
INTEGER::ID,NIMG
CHARACTER(100)::FNAME_W,FNAME_R
REAL(4),SAVE::A(NMAX)[*],B(NMAX)[*]
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
WRITE(FNAME_W,'("A", I4.4, ".DAT")')ID
OPEN(10,FILE=FNAME_W,FORM='UNFORMATTED')
WRITE(10)A
CLOSE(10)
...
WRITE(FNAME_R,'("A", I4.4, ".DAT")')ID
OPEN(20,FILE=FNAME_R,FORM='UNFORMATTED')
READ(20)B
CLOSE(20)
...
```

Note that it is also possible to use "the rank number directory" in the same way as the MPI program.

See "1.1.2 Image and Rank" for correspondence of the rank number and the image index.

An example where multiple images access one shared file in parallel

Each image writes in the corresponding record number.

The execution of the CLOSE(30,STATUS='FSYNC') statement makes the file on memory synchronized with the file on the external storage unit. From this, each image can access the shared file after execution of the SYNC ALL statement.

```
INTEGER,PARAMETER::NMAX=1000
INTEGER::ID,NIMG
REAL(4),SAVE::A(NMAX)[*]
REAL(4),DIMENSION(:,:),ALLOCATABLE::D
...
NIMG=NUM_IMAGES()
ID=THIS_IMAGE()
...
SYNC ALL
OPEN(30,FILE='A.DAT',ACCESS='DIRECT',&
     FORM='UNFORMATTED',RECL=NMAX*4)
     ...
  WRITE(30,REC=ID)A
  ...
! FLUSH AND SYNC
CLOSE(30,STATUS='FSYNC')
...
SYNC ALL
IF(ID==1)THEN
  ALLOCATE(D(NMAX,NIMG))
  OPEN(40,FILE='A.DAT',FORM='BINARY')
  READ(40)D
  CLOSE(40)
  ...
END IF
...
```

Chapter 6 Notes

This chapter gives notes on usage of COARRAY in this system.

6.1 Notes on COARRAY Programs

The following describes notes on the specification of COARRAY programs.

6.1.1 ACQUIRED_LOCK Specifier of LOCK Statement

The ACQUIRED_LOCK specifier of LOCK statement is unavailable. If it is specified, the diagnostic message (jwe1739i-s) is displayed and error termination is started.

6.1.2 Assignment Statement between Coarrays

It is impossible to transfer data between images where neither of the destination or the source is not self-image. If such statements are executed, the diagnostic message (jwe1790i-s) is displayed and error termination is started.

6.1.3 Reservation of Procedure Name and Common Block Name

Procedures and blocks whose name starts with either of the following words are reserved:

- fjmpi_

- mpi_

- ompi_

- pmpi_

Note

If the compiler option -AU is not specified, source programs are not case-sensitive.

If the compiler option -AU is specified, procedure names and common block names become as they are spelled in the source program.

See "Fortran User's Guide" for details of processing external procedure names.

Binding labels whose name starts with either of the following words are reserved when using C programs with BIND statements or using procedure language binding specifiers.

- ARMCII_

- ARMCIX_

- ARMCI_

- FJMPI_

- MPI_

- OMPI_

- PARMCI_

- PFJMPI_

- PMPI_

- armci_

- fjmpi_

- mca_

- mpi_

- ompi_

- opal_

- orte_

- parmci_

- pmpi_

6.1.4 Notes on Using OpenMP

It is possible to specify the -Kopenmp option and the -Ncoarray option at the same time in this system.

Use the COARRAY feature in the master thread. Otherwise, the following diagnostic message is displayed and error termination starts regardless of whether the STAT= specifier and its arguments are specified.

```
jwe1704i-s The statement which uses the COARRAY feature cannot be executed on threads other than the
master thread.
```

6.1.4.1 Coarrays

- It is impossible to specify coarrays to THREADPRIVATE directing statements.

- It is impossible to specify coindexed variables to OpenMP clauses.

- A coarray specified in the following clauses is not able to be coindexed over its scope.

    - PRIVATE

    - FIRSTPRIVATE

    - LASTPRIVATE

    - COPYPRIVATE

    - REDUCTION

- A coarray with an ALLOCATABLE attribute is not to be specified in the following clauses.

    - PRIVATE

    - FIRSTPRIVATE

    - LASTPRIVATE

    - COPYPRIVATE

    - REDUCTION

6.1.5 Notes on Creating Process

The following restriction exists when COARRAY programs create processes by using system call or the Fortran intrinsic function library.

- Data allocated in data section in the shared object created by users cannot be the source of assignment statements defining data on other images nor the destination of assignment statements referring data on other images.

6.1.6 Note on Transferring Constant Area to Other Images

When a constant area is transferred to other images, the following interconnect error occurs if the size of the area is more than 32 bytes.

```
[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag. [Information for detail]
```

This error can be avoided by one of the followings:

- Specifying -Nnouse_rodata option at compilation time.

- Transferring the constant area via a writable area.

6.2 Restrictions on Inline Expansion

Inline expansion is not applied to procedures including coarrays. In addition, it is not applied to procedures whose host includes coarrays.

6.3 Notes on Runtime

The following describes notes on executing COARRAY programs.

6.3.1 Runtime Error Message

Error messages of MPI could be output at runtime since COARRAY uses the MPI library.

MPI error messages start from "[mpi::". See "MPI User's Guide" for MPI error messages. In addition, read a rank to image in message. Replace the word "rank" with the word "image" appropriately.

The image index is the rank number plus one (See "1.1.2 Image and Rank").