FUJITSU Software
Technical Computing Suite V2.0

MPI User's Guide (PRIMEHPC FX100)

J2UL-1885-02ENZ0(00)
November 2015

Preface

Purpose of this Manual

This manual describes how to use the MPI library intended for supercomputer PRIMEHPC FX100 systems (hereafter referred to as FX100 systems). In particular, it details the MPI library included in the following product:

- Technical Computing Suite V2.0

MPI (Message Passing Interface) is the MPI library interface regulated by the MPI Forum, and is the interface installed in the MPI library documented here. This MPI library is referred to as "this system" below. Note that this product conforms to the MPI-3.0 Standard regulated by the MPI Forum.

Audience

The intended readers of this manual are those using the system to develop programs in Fortran, C, or C++. In addition to knowledge of MPI and of programming in Fortran, C, and C++, readers need to have a basic knowledge of Linux commands, file manipulation, and shell programming.

How This Manual is Organized

This manual consists of the following sections.

Chapter 1 Overview

An overview of this system

Chapter 2 Environment and Advance Settings

The environment settings that must be set in advance

Chapter 3 MPI Program Compilation/Linkage

How to compile and link MPI programs

Chapter 4 MPI Program Execution

How to execute MPI programs

Chapter 5 Extended Interfaces

Details the extended interface

Chapter 6 Supplementary Items

Supplementary information for this system

Chapter 7 Error Messages

Error messages detected by this system

Appendix A Error Class List

Error classes detected by this system

Appendix B Notes on Migration from FX10 System to FX100 System

This chapter provides notes on migrating from FX10 system to FX100 system on this system.

Appendix C Compatibility Information (FX10 system)

Provides compatibility information (FX10 system) with earlier releases of this product

Appendix D Compatibility Information (FX100 system)

Provides compatibility information (FX100 system) with earlier releases of this product

Glossary

Terminology

Related Manuals

The following manuals are related to this manual. Refer to these manuals in conjunction with this manual.

- *Fortran Language Reference*
- *Fortran User's Guide*
- *Fortran User's Guide Additional Volume COARRAY*
- *Fortran Compiler Messages*
- *Debugger User's Guide*
- *Runtime Information Output Function User's Guide*
- *C User's Guide*
- *C++ User's Guide*
- *C/C++ Compiler Optimization Messages*
- *XPFortran User's Guide*
- *Fortran/C/C++ Runtime Messages*
- *Programming Workbench User's Guide*
- *Profiler User's Guide*
- *Rank Map Automatic Tuning Tool User's Guide*
- *Programmer's Guide for Usage of Mathematical Libraries*
- *SSL II User's Guide*
- *FUJITSU SSL II Extended Capabilities User's Guide*
- *FUJITSU SSL II Extended Capabilities User's Guide II*
- *FUJITSU SSL II Thread-Parallel Capabilities User's Guide*
- *FUJITSU C-SSL II User's Guide*
- *FUJITSU C-SSL II Thread-Parallel Capabilities User's Guide*
- *FUJITSU SSL II/MPI User's Guide*
- *BLAS LAPACK ScaLAPACK User's Guide*
- *Fast Basic Operations Library for Quadruple Precision User's Guide*
- *Job Operation Software First Step Guide*
- *Job Operation Software End-user's Guide*
- *Job Operation Software System Log Messages*
- *FEFS User's Guide*

To learn details of the MPI specifications, refer to the following standard:

MPI: A Message-Passing Interface Standard

Version 3.0

Message Passing Interface Forum

September 21, 2012

Information concerning MPI is available from http://www.mpi-forum.org/.

However, note that the information obtained from the above website might vary slightly from installations of this system.

Expression of Units

In this manual, the following prefixes are used to express units:

Prefix	Value	Prefix	Value
k (kilo)	10^3	Ki (kibi)	2^{10}
M (mega)	10^6	Mi (mebi)	2^{20}
G (giga)	10^9	Gi (gibi)	2^{30}

Conventions Used in this Manual

The typographic conventions below are symbols used with pre-determined special meanings that express syntax.

Symbol name	Symbol	Explanation
Selection symbols	{}	Indicates to select any one of the enclosed items
	|	Used as a delimiter in a list of items
Omission permitted symbol	[]	Indicates that the enclosed item can be omitted. This symbol includes the meaning of the selection symbol "{}".

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

- Linux is a registered trademark or trademark of Linus Torvalds in the United States 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, 2nd Version	J2UL-1885-02ENZ0(00)
February 2015, Version 1.1	J2UL-1885-01ENZ0(01)
October 2014, 1st Version	J2UL-1885-01ENZ0(00)

Copyright

Update History

Changes	Location	Version
The article is corrected for the MPI-3.0 support.	Preface 6.3.1 6.3.2 6.3.7 6.3.14 7.2 7.3 A.3	2nd Version
The article on the argument check is corrected.	3.1	

Changes	Location	Version
Note concerning the -Knointentopt option is added.	3.2	
Examples of execution command are corrected.	4.1	
The article on the profiling interface is added.	6.3.13	
The explanation is added.	6.5	
The explanation is corrected.	6.10.1	
The error message is added / corrected.	7.2	
The value of MPI_ERR_LASTCODE is changed.	A.1	
The article is added.	B.1 D.1	
Fixed the error in writing.	-	
The article on the sequential program execution is added.	4.1	Version 1.1
The article on the execution definition file specification is added.	4.1	
The following MCA parameter is added. - coll_tuned_scatterv_use_linear_sync	4.2	
Notes concerning the non-contiguous mode are added.	4.2 5.1.2.5 5.1.3.5	
The explanation is added / corrected.	3.1 5.2 6.1.3 6.5 6.17	
"Value" in "Table 6.11" is changed.	6.11.1	
The error message is added.	7.2 7.3	

Contents

Chapter 1 Overview

This chapter gives an overview of this system and an outline of how to use it.

1.1 System Features

This MPI library is intended for use with FX100 system. MPI (Message Passing Interface) is the set of standards determined by the MPI Forum for regulating the library interface, which enables the Fortran, C, and C++ languages to be used for parallel MPI programming in parallel computing systems with distributed memory.

FX100 systems use an interconnect, known as Tofu, comprised of a 6-dimensional mesh/torus. A virtual torus shape can be configured from the physical 6-dimensional mesh/torus in the Tofu interconnect, and users can specify a network configuration having a torus shape of from one to three dimensions when executing programs. This system supports this Tofu interconnect, thus achieving maximum performance for the application programs that use the system. In addition, users can use the extended interface to describe programs that make use of the 6-dimensional mesh/torus. Read "6.1 Tofu Interconnect" for details of the Tofu interconnect.

1.2 Outline of How to Use This System

The MPI library provided by this system can be used from application programs written in Fortran, C, or C++. In this manual, application programs that use the MPI library are called MPI programs.

This system provides commands for compiling and linking MPI programs and MPI program execution commands.

This section gives a simple description of the flow of procedures, from compilation to execution of an MPI program intended for FX100 system.

1.2.1 Flow from Compilation to Execution of an MPI Program

To execute an MPI program, the user performs the required operations from the login node for FX100 system. For example, MPI program compilation and linkage (in other words, execution of the compilation/linkage command) is normally performed from the login node. Execution of an MPI program that has been converted to an executable file format by compilation and linkage is performed by requesting Job Operation Software to launch the job. Refer to the Job Operation Software manual for details.

Compiling and linking an MPI program

This system provides compilation/linkage commands that compile and link MPI programs written in Fortran, C, and C++ in order to convert them to the executable file format intended for FX100 system.

There are two types of compilation/linkage commands: commands used as the login node (cross compiler), and commands executed at compute nodes (own compiler).

These compilation/linkage commands are shown below.

Use these compilation/linkage commands in accordance with the program language you are using to write an MPI program.

Table 1.1 Compilation/linkage commands

Command name		Programming language of MPI program
`<Cross compiler>`	`mpifrtpx`	`Fortran`
	`mpifccpx`	`C`
	`mpiFCCpx`	`C++`
`<Own compiler>`	`mpifrt`	`Fortran`
	`mpifcc`	`C`
	`mpiFCC`	`C++`

Invoke these cross compiler compilation/linkage commands from the login node. They can be used to convert MPI programs to a format that can be executed by FX100 system. These compilation/linkage commands internally invoke the corresponding Fujitsu cross compilers (frtpx(1), fccpx(1), or FCCpx(1)). Refer to "Chapter 3 MPI Program Compilation/Linkage" for information on how to use the MPI program compilation/linkage commands. Refer to the compiler manuals for information on the Fujitsu compilers.

Use the own compiler compilation/linkage commands from the compute nodes. To execute a command on a compute node, ask the Job Operation Software to launch a job. Refer to the Job Operation Software manual for details. The own compiler compilation/linkage commands internally invoke the corresponding Fujitsu own compiler (frt(1), fcc(1), or FCC(1)).

The cross compiler and own compiler for MPI programs have no functional differences other than the command launching methods described above. Refer to "Chapter 3 MPI Program Compilation/Linkage" for information on how to use own compiler compilation/linkage commands.

Executing an MPI program

Use the mpiexec(1) command to execute an MPI program that has been converted to an executable file format using a compilation/linkage command. The mpiexec(1) command must be executed from the compute node within FX100 system but the user does not execute it directly in the compute node. Instead, the user requests Job Operation Software to launch the job that executes the MPI program. Refer to the Job Operation Software manual for information on how to launch jobs.

This system has two communication ways between two parallel processes of a MPI program internally, Tofu interconnect communication and shared memory communication. Tofu interconnect communication is used in MPI communication between nodes, and shared memory communication is used in MPI communication inside each node.

Chapter 2 Environment and Advance Settings

This chapter describes the environment settings that must be set when using this system.

2.1 MPI Program Compilation/Linkage Environment

The following setting is required at the login node in order to enable MPI program compilation and linkage with the cross compiler :

- Append the following path name to user environment variable PATH

```
/opt/FJSVmxlang/bin
```

The following setting is required in the job script when the compilation/linkage command execution job is launched. The setting enables MPI program compilation and linkage with the own compiler. Refer to the Job Operation Software manual for information on job launching and job scripts.

- Append the following path name to user environment variable PATH

```
/opt/FJSVmxlang/bin
```

The MPI library resources are shown below.

Table 2.1 MPI library resource list

Name		Usage Path within () indicates the install location
C C++	Fortran	
mpi.h mpi-ext.h	mpif.h mpi.mod	When compiling: MPI library header file (for Fortran, the module information file is also included) `(/opt/FJSVmxlang/include/mpi/fujitsu)`
mpifccpx mpiFCCpx mpifcc mpiFCC	mpifrtpx mpifrt	When compiling and linking: Fortran, C, and C++ compilation/linkage commands `(/opt/FJSVmxlang/bin)`

2.2 MPI Program Execution Environment

The settings below are required in the job script used to launch an MPI program execution job. Refer to the Job Operation Software manual for information on launching jobs and job scripts.

- Append the following path name to user environment variable PATH

```
/opt/FJSVmxlang/bin
```

- Append the following path name to user environment variable LD_LIBRARY_PATH

```
/opt/FJSVmxlang/lib64
```

2.3 Online Manual

The following setting is required at the login node in order to use the related online manual:

- Append the following path name to user environment variable MANPATH

```
/opt/FJSVmxlang/man
```

Chapter 3 MPI Program Compilation/Linkage

This chapter describes how to compile and link an MPI program.

3.1 Overview of Compilation/Linkage Commands

As described in "1.2 Outline of How to Use This System", an MPI program is a Fortran, C, or C++ program that includes invocation of the MPI library.

Fujitsu compilers frtpx(1), fccpx(1), or FCCpx(1) (or own compilers frt(1), fcc(1), or FCC(1)) are used to compile and link ordinary Fortran, C, or C++ programs; however the compile/edit commands for MPI programs, mpifrtpx(1), mpifccpx(1), and mpiFCCpx(1) (or own compilers mpifrt(1), mpifcc(1), or mpiFCC(1)), are used to compile and link MPI programs.

mpifrtpx(1), mpifccpx(1), mpiFCCpx(1), mpifrt(1), mpifcc(1), and mpiFCC(1) are wrapper commands for frtpx(1), fccpx(1), FCCpx(1), frt(1), fcc(1), or FCC(1) respectively, and internally invoke the corresponding Fujitsu compiler. Therefore, the corresponding Fujitsu compiler options can be specified as is in the compile/edit commands for MPI programs.

The conformance of types of arguments of MPI function calls can be checked when an MPI program is compiled.
In the case of a Fortran program, if the program uses the mpi module, the compiler checks argument types based on the contents of the module, and an error or a warning message is output. Note that the mpi_f08 module is not supported in this system.
In the case of a C program or a C++ program, the compiler checks argument types based on the contents of mpi.h, and an error or a warning message is output.

Refer to the compiler manuals for details on Fujitsu compilers.

3.2 Compilation/Linkage Command Format

Table 3.1 Compilation/linkage command format

Command name		Options
`Cross compiler`	`mpifrtpx`	`[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [`*`compiler_arguments`*`] file ...`
`Own compiler`	`mpifrt`	
`Cross compiler`	`mpifccpx`	`[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [`*`compiler_arguments`*`] file ...`
`Own compiler`	`mpifcc`	
`Cross compiler`	`mpiFCCpx`	`[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [`*`compiler_arguments`*`] file ...`
`Own compiler`	`mpiFCC`	

With this system, MPI programs can be a mix of Fortran, C, and C++. Refer to the compiler manuals for notes and details concerning mixing program languages.

The compilation/linkage command options are explained below:

Table 3.2 Compilation/linkage command options

Option	Explanation
`-showme`	Displays the call line used when the MPI program compilation/linkage command invokes the Fujitsu compiler command. Actual compilation/linkage processing is not performed.
`-showme:compile`	Displays the option list that is passed to the Fujitsu compiler command. Actual compilation/linkage processing is not performed.
`-showme:link`	Displays the option list that is passed to the linker. Actual compilation/linkage processing is not performed.
`-showme:version`	Displays the version information. Actual compilation/linkage processing is not performed.
`-SCALAPACK`	Links the ScaLAPACK library. With this option, specify the Fujitsu compiler option -SSL2 or -SSL2BLAMP.

Option	Explanation
`-SSL2MPI`	Links the SSL II/MPI library. With this option, specify the Fujitsu compiler option -SSL2 or -SSL2BLAMP.
`compiler_arguments`	Specifies the options passed to the Fujitsu compiler. Refer to the Fujitsu compiler manuals for practical details of the options that can be specified.

Note

Compilation/linkage notes

- The MPI library is provided in only the dynamic link library format.
- mpifrtpx(1) and mpifrt(1) automatically specify the following frtpx(1) and frt(1) option:
  - -f2004
  - -Knointentopt (-Kintentopt option is disable even if it is specified)
- When the following option of frtpx(1) and frt(1) is specified with mpifrtpx(1) and mpifrt(1), the language entities must be lowercase letter:
  - -AU
- When the following option of fccpx(1), fcc(1), FCCpx(1) and FCC(1) is specified with mpifccpx(1), mpifcc(1), mpiFCCpx(1) and mpiFCC(1), a warning message about language standard is output. There is no influence in the execution of the MPI program:
  - -Xc

Example

Examples of using mpifrtpx(1) to compile and link an MPI program

1. Compile the user program "test.f" and create the object program "test.o".

```
$ mpifrtpx -c test.f
```

2. Link edit the object program "test.o" and create the executable program "test".

```
$ mpifrtpx -o test test.o
```

Chapter 4 MPI Program Execution

This chapter describes how to execute MPI programs.

In this system, mpiexec(1) is used to execute MPI programs. mpiexec(1) passes control to Job Operation Software. Process generation and execution of the MPI program takes place on the compute node. Refer to the Job Operation Software manual for information on Job Operation Software.

4.1 Execution Command Formats

The format of the execution command varies depending on whether the SPMD model, the MPMD model, or the execution definition file specification is used for execution.

1. SPMD model

Table 4.1 SPMD model execution command format

Command name	Options
mpiexec	*global_options local_options execfile execfile_arguments*

2. MPMD model

Table 4.2 MPMD model execution command format

Command name	Options
mpiexec	*global_options local_options execfile1 execfile1_arguments* *: local_options execfile2 execfile2_arguments* [*: local_options execfile3 execfile3_arguments*] *...*

<Notes>

- If there are three or more different programs, the items enclosed in square brackets [ ] are specified repeatedly to give the required number of specifications.

3. Execution definition file

Table 4.3 Execution definition file specifying execution command format

Command name	Options
mpiexec	*global_options* { -app | --app } *execution_definition_file local_options*

The execution definition file is described in the form of the following:

```
local_options execfile1 execfile1_arguments
[ local_options execfile2 execfile2_arguments ] ...
```

<Notes>

- If there are two or more different programs, the items enclosed in square brackets [ ] are specified repeatedly to give the required number of specifications.
- When *local_options* are specified for mpiexec(1), the specified content is effective for all lines of the execution definition file.
- When *local_options* are specified for both mpiexec(1) and the execution definition file more than once, the specification of the execution definition file takes priority.
- When "#" or "//" is included in the execution definition file, the following content in the line is ignored.

Note

Note that, if a single colon (:) is specified in the command line of mpiexec(1), the colon is regarded as a delimiter. For example, a single colon cannot be specified as an *execfile* name or an argument name to be passed to the corresponding *execfile*.

The format for *global_options* is shown under "Table 4.4 global_options format". The format for *local_options* is shown under "Table 4.5 local_options format".

Explanations of all options are provided in "Table 4.6 Execution command options". Explanations of options that can be specified as *global_options* are listed below in "Table 4.7 Options that can be specified in global_options", and explanations of options that can be specified at *local_options* are given in "Table 4.9 Options that can be specified in local_options".

In this system, some variables, known as MCA parameters, are held internally by the MPI library. The operating conditions that apply when this system is executed can be changed by temporarily changing the values of these MCA parameters. Refer to "4.2 MCA Parameters" for information on MCA parameters. MCA parameters can also be set by environment variables. Refer to "4.3 Environment Variables" for information on using environment variables to set MCA parameters.

Example

Specification example for using mpiexec(1) to execute an MPI program

1. Example of execution command specification in SPMD model

```
$ mpiexec -of-proc procfile -mca mpi_print_stats 2 ./a.out
```

Execute the MPI program executable file a.out. The program output results and statistical information for each process is output to the file with the name generated by the specified method.

2. Example of execution command specification in MPMD model

```
$ mpiexec -n 2 ./a.out : -n 4 ./b.out : -n 6 ./c.out
```

Execute the MPI program files a.out, b.out, and c.out using 2, 4, and 6 parallel processes respectively.

3. Example of execution command specification in the format of the execution definition file

```
$ cat abc.exec
-n 2 ./a.out
-n 4 ./b.out
-n 6 ./c.out
$ mpiexec --app abc.exec
```

Execute the MPI program files a.out, b.out, and c.out using 2, 4, and 6 parallel processes respectively.

Table 4.4 global_options format

```
[ { -app | --app } APP_FILE ]

[ { -debuglib | --debuglib  } ]

[ { -h | --help  } ]

[ { -nompi | --nompi } ]

[ { -of | --of | -std | --std }  FILE  ]

[ { -oferr | --oferr | -stderr | --stderr }  ERR_FILE  ]

[ { -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc }  ERR_PROC_FILE  ]

[ { -ofout | --ofout | -stdout | --stdout }  OUT_FILE  ]
[ { -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc }  OUT_PROC_FILE  ]

[ { -of-proc | --of-proc | -std-proc | --std-proc }  PROC_FILE  ]

[ { -ofprefix | --ofprefix | -stdprefix | --stdprefix }  PREFIX  ]

[ { -stdin | --stdin }  STDIN_FILE  ]

[ { -vcoordfile | --vcoordfile }  VCOORD_FILE  ]

[ { -V | --version } ]
```

Table 4.5 local_options format

```
[ -am  AM_FILE ]

[ -x  NAME=VALUE ]

[ { -mca | --mca }  MCA_PARAM_NAME  MCA_PARAM_VALUE  ]

[ { -c | -np | --np | -n | --n }  N  ]
```

Table 4.6 Execution command options

Option	Explanation
execfile	Specifies a shell script, or an executable file. If specifying a shell script which do not execute an MPI program executable file or an executable file other than an MPI program, specify the -nompi option for the mpiexec(1). If the -nompi option is not specified, an executable file or a shell script will end abnormally when it is executed. A shell script cannot be coded to execute more than one MPI program. The shell script behavior is not guaranteed if it contains code to execute more than one MPI program. Recursive execution of mpiexec(1) cannot be specified. If it is executed recursively, an error message is output and then mpiexec(1) ends abnormally. For example, the mpiexec(1) start filename cannot be specified for *execfile*. Point If a shell script is specified, execution permission is required for the shell script.
execfile_arguments	Specifies the arguments to be passed to *execfile.*
execfile1 *execfile2* *execfile3*	Specifies a shell script, or an executable file. When the MPMD model is used for execution, the corresponding executable file is delimited by the colon and specified according to a different number of programs. If specifying a shell script which do not execute an MPI program executable file or an executable file other than an MPI program, specify the -nompi option for the mpiexec(1). If the -nompi option is not specified, an executable file or a shell script will end abnormally when it is executed. A shell script cannot be coded to execute more than one MPI program. The shell script behavior is not guaranteed if it contains code to execute more than one MPI program. Recursive execution of mpiexec(1) cannot be specified. If it is executed recursively, an error message is output and then mpiexec(1) ends abnormally. For example, the mpiexec(1) start filename cannot be specified for *execfile1, execfile2, execfile3.*

Option	Explanation
	Point If a shell script is specified, execution permission is required for the shell script.
execfile1_arguments *execfile2_arguments* *execfile3_arguments*	Specifies the arguments to be passed to *execfile1*. Specifies the arguments to be passed to *execfile2*. Specifies the arguments to be passed to *execfile3*.

Table 4.7 Options that can be specified in global_options

Option	Explanation
{ -app | --app } *APP_FILE*	Specifies when the execution definition file of *APP_FILE* is used. Following the app option, specify the path of execution definition file. Read permission to the user who executes the job is required for the specified file. Specify the number of parallel processes for each of the MPI programs. If this option is specified more than once, the parameter specified last takes priority.
{ -debuglib | --debuglib }	Links the debug MPI library. If you are using the argument check function, which is one of the dynamic debug functions at the time of execution, specify this option. Use of this option may cause execution of the MPI program to become very slow as the debug MPI library is linked. Take care when using this option. Refer to "6.16.3 Argument Check Function" for information on the argument check function.
{ -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.
{ -nompi | --nompi }	Specifies that the user executes the shell script which does not execute an MPI program executable file or the executable file other than an MPI program. If the shell script which does not execute an MPI program or the executable file other than an MPI program is executed without specifying this option, the mpiexec command will end abnormally. MPI program is the program that executes MPI_init function or MPI_Init_thread function.
{ -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. Note that the filename for a dynamically generated parallel process is "*ERR_PROC_FILE.rank number@spawn number*". The "*spawn number*" character string is the number allocated by Job Operation Software for each dynamically generated process, expressed as a numeric character string. 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.

Option	Explanation
{ -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. Note that the filename for a dynamically generated parallel process is "*OUT_PROC_FILE.rank number@spawn number*". The "*spawn number*" character string is the number allocated by Job Operation Software for each dynamically generated process, expressed as a numeric character string. 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.
{ -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. Note that the filename for a dynamically generated parallel process is "*PROC_FILE.rank number@spawn number*". The "*spawn number*" character string is the number allocated by Job Operation Software for each dynamically generated process, expressed as a numeric character string. 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.
{ -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 | nid | rank,nid | nid,rank } The output format of the character string depends on which keyword is specified, as shown below. 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. If a parallel process is generated dynamically, the character string "@*spawn number*" is added after the rank number. This string expresses, as an unchanged numeric character string, the number assigned by the Job Operation Software to each dynamically generated process. The node ID is the number that identifies the compute node allocated to the parallel process. Refer to the Job Operation Software manual for information concerning node IDs.
{ -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 MPI program. If just the filename or the relative path is specified, the relative path from the job execution current directory is used.

Option	Explanation
	If this option is specified more than once, the parameter specified last takes priority.
{ -vcoordfile | --vcoordfile } *VCOORD_FILE*	Specifies that parallel processes are allocated on the basis of the process assignment information specified in the *VCOORD_FILE* file when an MPI program is executed. If just the filename or the relative path is specified, the relative path from the current directory of the mpiexec(1) process is used. Ensure that this option is specified if background execution is used to execute more than one mpiexec(1) simultaneously. In this case, the same coordinates cannot be specified in the *VCOORD_FILE* files specified in the simultaneously executed mpiexec(1)s. If the same coordinates are specified in the *VCOORD_FILE* files, a Job Operation Software error occurs. The maximum number of mpiexec(1) that can be executed simultaneously in background execution is 128. Refer to "4.5 VCOORD_FILE file format" for details on the *VCOORD_FILE* file. Note that if an MPI program is executed with this option specified, the coding that performs dynamic process generated cannot be included in the MPI program. Refer to "6.3.9 Dynamic Process Generation". 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.

Note

Parallel process standard input, standard output, and standard error output

The standard output and standard error output of each parallel process and mpiexec(1) are normally connected to the job execution results file that is generated by Job Operation Software when the job is executed.

A list of the parallel process standard outputs and standard error outputs that result from whether or not the various -of/-std options are specified in the execution command mpiexec(1) is shown in table below:

Table 4.8 Parallel process standard output and standard error output resulting from -of/-std option specifications

mpiexec(1) option specification	Standard output	Standard error output
No type of -of/-std option specified	Job execution results file	Job execution results file
-of/-std option specified	Specified file	Specified file
-ofout/-stdout option specified	Specified file	Job execution results file
-oferr/-stderr option specified	Job execution results file	Specified file
-of-proc/-std-proc option specified	Specified file	Specified file
-ofout-proc/-stdout-proc option specified	Specified file	Job execution results file
-oferr-proc/-stderr-proc option specified	Job execution results file	Specified file

The redirected standard input of mpiexec(1) cannot be used as the standard input for each of the parallel processes. A function for specifying the standard input for each parallel process, and a function for changing the connection destination of the standard output and standard error output of each parallel process, are provided in the mpiexec(1) options. Note the dynamically generated parallel process standard input, standard output, and standard error output also conform to these option specifications for mpiexec(1).

Refer to the Job Operation Software manual for information concerning the job execution results file.

DT_RPATH

If the DT_RPATH dynamic section attribute exists in a MPI program and /opt/FJSVmxlang/lib64 is included in DT_RPATH, --debuglib option is disabled. Deal with either as follows for such a MPI program.

- Do not let /opt/FJSVmxlang/lib64 be included in the DT_RPATH. (Relinking program and so on)
- Specify the --inhibit-rpath option of a dynamic linker when a MPI program is executed. Specifying the option is shown below.

Example

```
$ mpiexec -n 2 /lib64/ld-linux.so.2 --inhibit-rpath :./a.out ./a.out
```

- Set the environment variable PATH and LD_LIBRARY_PATH corresponding to the executed FJSVmxlang package.
- Specify the argument of the --inhibit-rpath option of /lib64/ld-linux.so.2 in the form of ":MPI program".

Table 4.9 Options that can be specified in local_options

Option	Explanation

-am *AM_FILE*	Specifies the path name of the AMCA parameter file (MCA parameter settings file) corresponding to the relevant MPI program.
	
	Specify the same MCA parameter values for all programs. If different values are specified, the operation results are not guaranteed.
	
	The specification method within the file is as follows:
	
	- In each line, use following format for the specification:
	`MCA-parameter-name=value`
	
	- If multiple values are to be specified in the same MCA parameter, use commas as separators as shown below.
	`MCA-parameter-name=value1,value2`
	
	If this option is specified more than once, the parameter specified last takes priority.

-x *NAME=VALUE*	Specifies the environment variable when executing an MPI 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.
	
	However, if the environment variable name is specified more than once, the value specified last takes priority.
	`Specification example:`
	`-x OMP_NUM_THREADS=8 -x THREAD_STACK_SIZE=4096`

{ -mca | --mca } *MCA_PARAM_NAME MCA_PARAM_VALUE*	Specifies MCA parameters for the relevant MPI program. Specify the same value in the MCA parameter for all programs. If different values are specified, the operation is not guaranteed.
{ -c | -np | --np | -n | --n } *N*	Specifies the number (an integer) of parallel processes for the relevant MPI program. If this option is omitted when the SPMD model is used for execution, the maximum number of parallel processes that can be generated is assumed. This option must be specified when the MPMD model is used for execution.

Option	Explanation
	If this option is specified more than once for the same MPI program, the parameter specified last takes priority. For execution of an MPI program in an FX100 system, the parallel processes can be allocated to an appropriate torus format. Refer to the Job Operation Software manual for information concerning how to specify torus format and how to deploy parallel processes.

4.2 MCA Parameters

When an MPI program is executed in this system, the system operating conditions can be changed by temporarily changing the values of the MPI library internal variables. These variables are called MCA parameters. This section describes the MCA parameter types and how to use them. Environment variables can be used to set MCA parameters. Refer to "4.3 Environment Variables" for details.

In this system, MCA parameters can be specified in the following ways:

- Use the -am option of mpiexec(1) to specify the parameters in the MCA parameter settings file (AMCA parameter file).
- Use the -mca option of mpiexec(1) to specify the MCA parameters directly.
- Use the environment variables to set the MCA parameters.

If different methods are used to specify values for the same MCA parameter, the specification with the highest priority takes effect. The priority levels for the different MCA parameter specification methods are shown in table below:

Table 4.10 MCA parameter specification methods and priorities

Rank	MCA parameter setting method	Example of use
1	-mca option of mpiexec	`-mca  btl_tofu_eager_limit  4096`
2	Environment variable	`export  OMPI_MCA_btl_tofu_eager_limit=4096`
3	AMCA parameter file (MCA parameter settings file) specified in the -am option	`-am  mca_file`

Note: A smaller priority value indicates a higher priority.

The "MCA parameters" that can be used in this system are shown below. The text in parentheses after each MCA parameter name describes the function of that MCA parameter.

MCA parameters

Table 4.11 btl_tofu_eager_limit (changes the threshold value for switching the communication method)

MCA parameter value	Content
Integer value of 1 or more	Specifies the message size (number of bytes) used as the "threshold value" for switching between the Eager protocol and the Rendezvous protocol in the fast communication mode. Messages that are smaller than the specified message size (number of bytes) are sent under Eager protocol. More precisely, the value used is obtained by adding the size of the actual message (number of bytes) and the size of the internally added header part (several tens of bytes). Refer to "6.10.1 Switching between Fast Communication Mode and Memory-Saving Communication Mode" for information on the fast communication mode. If a value smaller than 256 is specified, 256 is set. Note that, if a large value is specified, it may be reduced internally to a value that is smaller than the specified value. Generally, values up to about 128,000 can be specified effectively but, depending on the memory usage status, the value may need to be smaller than this to take effect. In practice, this depends on the Large receive buffer size specified in the common_tofu_large_recv_buf_size MCA parameter and the send buffer size. More specifically, an approximation of the value that can be specified effectively can be obtained from the expression below, and has an upper limit value of around 512000, derived from the send buffer size. For a more precise value, if the control information and

MCA parameter value	Content
	other information used internally is considered, the value is reduced by several hundreds of bytes, however that can be ignored here. `Large receive buffer size / 2` Usually, this system dynamically determines an appropriate value internally as the "threshold value" for fast communication modes. This is because the distance of the compute node that is performing the communication is taken into account, not just the message size (number of bytes). For message communication between nearby compute nodes, for example, 45,352 is set internally. As the distance between the compute nodes performing the message communication increases, the "threshold value" also increases. With this MCA parameter, the specified value is used as the "threshold value" regardless of the compute node distance. Refer to "6.4 Eager Protocol and Rendezvous Protocol" for details.

"l" in the character string btl used in the MCA parameter name is a lowercase "L".

Table 4.12 coll_base_reduce_commute_safe (guarantees the reduction operation sequence)

MCA parameter value	Content
1	Guarantees the reduction operation sequence for MPI_Reduce, MPI_Ireduce, MPI_Allreduce, MPI_Iallreduce, MPI_Reduce_scatter, MPI_Ireduce_scatter, MPI_Reduce_scatter_block and MPI_Ireduce_scatter_block functions in collective communication that performs reduction operations. In collective communication that performs these reduction operations, the operation sequence may be changed in accordance with communication conditions to optimize the communication time. Changing the reduction operation sequence may affect the accuracy of the computing results. The operation sequence can be fixed by specifying a value in this parameter. Note that fixing the operation sequence lengthens the communication time. Refer to "6.8 Reduction Operation Sequence Guarantee in Collective Communication" for details.
0	Does not guarantee the reduction operation sequence. The reduction operation sequence may be changed internally to make the communication time as short as possible. Note that, if the communication conditions are the same, the computing results are the same regardless of the number of times the same program is executed. The default value for this parameter is 0.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.13 coll_tbi_intra_node_reduction (specifies the algorithms used by Tofu barrier communication for floating point type and complex type data reduction operations within a node if multiple processes are assigned within one node)

MCA parameter value	Content
1	If the conditions for applying Tofu barrier communication, as explained in "6.12.3 MPI_Reduce Function and MPI_Allreduce Function", are met, specifies that Tofu barrier communication uses ordinary algorithms for reduction operations of floating point type or complex type data within a node. The default value for this parameter is 1.
2	Specifies to use faster reduction operation algorithms compared with when 1 is specified as the value of this parameter. Read "6.12.5 Fast Reduction Operations for Floating Point Type and Complex Type Data within a Node" for details.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.14 coll_tbi_use_on_bcast (uses Tofu barrier communication in MPI_Bcast function)

MCA parameter value	Content
1	Specifies that Tofu barrier communication (hardware function) is used on execution of MPI_Bcast function. Refer to "6.12.2 MPI_Bcast Function" for details. The default value for this parameter is 1
0	Specifies that Tofu barrier communication is not used on execution of MPI_Bcast function.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.15 coll_tbi_use_on_comm_dup (uses Tofu barrier communication for a communicator created by MPI_Comm_dup function)

MCA parameter value	Content
1	If Tofu barrier communication (hardware function), explained in "6.12 Use of Tofu Barrier Communication to Increase Speeds", is applied, specifies that Tofu barrier communication is used for a communicator created by the MPI_Comm_dup function. The default value for this parameter is 1.
0	Specifies that Tofu barrier communication is not used for a communicator created by the MPI_Comm_dup function. Read "6.12.4 Notes on Tofu Barrier Communication" for details.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.16 coll_tbi_use_on_max_min (uses Tofu barrier communication for floating point datatypes MPI_MAX and MPI_MIN)

MCA parameter value	Content
1	Under the conditions for applying Tofu barrier communication, explained in "6.12.3 MPI_Reduce Function and MPI_Allreduce Function", the following operation combinations are added to the default operation combinations shown in "Table 6.16 Operation combinations that allow the MPI_Reduce and MPI_Allreduce functions to apply Tofu barrier communication". Communication speeds of the following operation combinations can be increased by using Tofu barrier communication. - Operation combinations to which Tofu barrier communication is newly applied. - MPI predefined operation - C/Fortran MPI_MAX MPI_MIN - C++ MPI::MAX MPI::MIN - Datatypes Floating point datatypes - Size 8 bytes or less A calculation result to which Tofu barrier communication is newly applied might be different in either of special following conditions according to whether Tofu barrier communication is applied. This fact must be noted if the value specified for the MCA parameter is being changed. - At least one input value is NaN.

MCA parameter value	Content
	- Tofu barrier communication is applied. A calculation result is compared from input values except NaN. If all input values are NaN, a calculation result is one of them. - Tofu barrier communication is not applied. A calculation result is either a result that input values except NaN are compared or one of NaN. - At least one input value is +0.0. And, at least one input value is -0.0. And, a calculation result is either +0.0 or -0.0. - Tofu barrier communication is applied. The functions compare sign of zero. (+0.0 > -0.0) - Tofu barrier communication is not applied. A calculation result is either +0.0 or -0.0.
0	Specifies that Tofu barrier communication is not used on the operation combinations explained by the content that the value of this parameter is 1. The default value for this parameter is 0.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.17 coll_tuned_bcast_same_count (achieves faster communication when MPI_Bcast/MPI_Ibcast function is used with the same count among the processes)

MCA parameter value	Content
1	Specifies to achieve faster communication when MPI_Bcast function or MPI_Ibcast function is used with the same count among the processes. Refer to "6.13 MPI_Bcast Function When the Same Count is Used among the Processes" for details.
0	Specifies not to use a faster communication mechanism. The default value for this parameter is 0.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.18 coll_tuned_prealloc_size (specifies the size of the static work area used internally by the collective communication function)

MCA parameter value	Content
Integer value of 1 or more	Specifies the size (MiB) of the work area allocated statically for the following collective communication functions: - MPI_Allreduce function - MPI_Reduce function - MPI_Reduce_scatter_block function - MPI_Reduce_scatter function - MPI_Allgather function - MPI_Gather function - MPI_Scatter function - MPI_Alltoall function For MPI programs that call these functions multiple times, the MPI program execution time can be reduced by using this parameter effectively for statically allocating memory.

MCA parameter value	Content

	The following size gives an estimate to be specified:
	For MPI_Allreduce function, MPI_Reduce function and MPI_Reduce_scatter_block function.
	`Size of messages sent in the program + 2MiB`
	For MPI_Reduce_scatter function.
	`(Size of message sent in the program * 2) + 2MiB`
	For MPI_Allgather function.
	`Size of message received in the program + 2MiB`
	For MPI_Gather function.
	`Size of message received on the root process in the program + 2MiB`
	For MPI_Scatter function.
	`Size of message sent on the root process in the program + 2MiB`
	For MPI_Alltoall function.
	`(Size of messages sent in the program * Number of ranks in communicators) + 2MiB`
	If the size of the work area allocated statically corresponding to this parameter specification is smaller than the work area size required to process the collective communication function, the statically allocated work area is not used.
	The default value for this parameter is 6(MiB).

0	Specifies that the static work area is not allocated.

Both "l" characters in the character string coll used in the MCA parameter name are the lowercase "L".

Table 4.19 coll_tuned_scatterv_use_linear_sync (uses the MPI_Scatterv algorithm which performs communication control)

MCA parameter value	Content
1	Uses the algorithm of MPI_Scatterv function that improves performance. This parameter is effective only for MPI_Scatterv function. If this algorithm is called continuously, an error message starting with [mpi::common-tofu::tofu-signal-mrq] might be output.
0	Does not use the algorithm of MPI_Scatterv function that improves performance. The default value for this parameter is 0.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.20 coll_tuned_use_6d_algorithm (use of algorithms tuned with recognition of Tofu coordinates)

MCA parameter value	Content
1	Enables algorithms tuned with recognition of Tofu coordinates to be called. This parameter is effective only for MPI_Alltoall. If MPI_Alltoall is called with this parameter specified, all processes that reference MPI_Alltoall temporarily use a memory area proportionate to the size of the messages received. Therefore, a memory shortage might occur. If this parameter is specified, execute the MPI program with plenty of memory available.

MCA parameter value	Content
	Read "6.14 Algorithms Tuned with Recognition of Tofu Coordinates" for details of algorithms tuned with recognition of Tofu coordinates. MPI statistical information can be used to check whether or not algorithms tuned with recognition of Tofu coordinates are selected. Read "6.15 MPI Statistical Information" for details. This parameter is ignored when the job type is mesh mode or non-contiguous mode. Refer to the Job Operation Software manual for information on mesh mode and non-contiguous mode.
0	Specifies that algorithms tuned with recognition of Tofu coordinates are not called. The default value for this parameter is 0.

"l" in the character string coll used in the MCA parameter name is a lowercase "L".

Table 4.21 common_tofu_fastmode_threshold (changes the conditions for switching to fast communication mode)

MCA parameter value	Content
Integer value of 0 or more	Specifies the communication count used as the condition for switching from memory-saving communication mode to fast communication mode. If 0 is specified, communication is performed in fast communication mode from the start, provided that the upper limit is not reached for the number of communication partner processes that communicate in fast communication mode. If -1 or a lower numeric is specified, a 0 specification is assumed. Refer to "6.10 Suppressing Memory Usage" for details. The default value for this parameter is 16.

Table 4.22 common_tofu_large_recv_buf_size (changes the size of the Large receive buffer)

MCA parameter value	Content
Integer value from 1024 to 16777216	Specifies the size (number of bytes) of the Large receive buffer. If a value lower than 1024 is specified, a specification of 1024 is assumed. If a value greater than 16777216 is specified, a specification of 16777216 is assumed. Refer to "6.10 Suppressing Memory Usage" for information on the large receive buffer. The default value for this parameter is 1048576.

Table 4.23 common_tofu_max_fastmode_procs (changes the upper limit for the number of processes that can communicate in fast communication mode)

MCA parameter value	Content
Integer value of -1 or 0 or more	Specifies the upper limit for the number of communication partner processes that each parallel process can communicate with in fast communication mode. If -1 is specified, communication with all processes is performed in fast communication mode. If 0 is specified, fast communication mode is not used and communication with all processes is in memory-saving communication mode. If a value of -2 or less is specified, a specification of -1 is assumed. Refer to "6.10 Suppressing Memory Usage" for details. The default value for this parameter is 1024.

Table 4.24 common_tofu_max_tnis (changes the upper limit for the number of TNIs to be used)

MCA parameter value	Content
Integer value of 1 or more	Specifies the upper limit for the number of network interface devices (TNIs) to be used. If a number greater than the maximum number (4) is specified, the parameter value becomes the number that can actually be used. Refer to "6.7 Using Multiple TNIs" for details.
-1	Specifies to use the maximum number of network interface devices (TNIs) that can be used. If a value of 0 or -2 or less is specified, a specification of -1 is assumed. The default value for this parameter is -1.

Table 4.25 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)

MCA parameter value	Content
Integer value from 256 to 16777216	Specifies the size (number of bytes) of the Medium receive buffer. If a value lower than 256 is specified, a specification of 256 is assumed. If a value greater than 16777216 is specified, a specification of 16777216 is assumed. Refer to "6.10 Suppressing Memory Usage" for information on the medium receive buffer. The default value for this parameter is 2048.

Table 4.26 common_tofu_memory_limit (specifies the memory allocation limit value)

MCA parameter value	Content
Integer value of 0 or more	Specifies the limit (MiB) for the memory allocation that this system's MPI library itself can use. A memory allocation limit cannot be specified if using dynamic process generation or if establishing communication between MPI process groups that do not share a communicator. Specify 0 to disable memory allocation restriction. If a value of -1 or less is specified, 0 is assumed. Refer to "6.11.3 Specifying Memory Allocation Restriction Values" for details. The default value for this parameter is 0MiB

Table 4.27 common_tofu_memory_limit_peers (specifies the assumed number of communication partner processes when the memory allocation is limited)

MCA parameter value	Content
Integer value of 0 or more	Specifies the assumed number of communication partner processes when the usable memory allocation of this system's MPI library is limited. The number of processes belonging to the communicator MPI_COMM_WORLD is set as the default value for this parameter. However, to perform automatic tuning more accurately, the number of connections for Tofu communication, obtainable from the MPI statistical information, must be specified. Refer to "6.11.3 Specifying Memory Allocation Restriction Values" for information on how to specify memory allocation limit values. The default value for this parameter is the number of processes belonging to the communicator MPI_COMM_WORLD.

Table 4.28 common_tofu_num_mrq_entries (change the number of entries in a completion queue)

MCA parameter value	Content
One of the following integer values: 2048, 8192, 32768, 131072, 524288	Specify the number of entries in a completion queue of Tofu interconnect. If you encountered an error message starting with [mpi::common-tofu::tofu-signal-mrq], the error may be avoided by changing the value of this parameter larger. Or memory usage of a MPI process can be decreased by changing this value smaller.

MCA parameter value	Content
	Specifiable values for this parameter are either 2048, 8192, 32768, 131072, or 524288. If a value that is not in the list is specified, the closest value in the list is assumed. If there are two closest values, the larger value is assumed. The default value for this parameter is 131072. Refer to "7.3 Communication Library Error Messages" for details of the error message starting with [mpi::common-tofu::tofu-signal-mrq].

Table 4.29 common_tofu_packet_gap (changes the packet transfer interval time)

MCA parameter value	Content
Integer value from 0 to 255	An integer value, called a gap value, specifies the packet transfer interval time. One unit of the gap value corresponds to 1/8 of the time taken to transfer data of the packet maximum transmission unit, described below. An integer value from 0 to 255 can be specified for this parameter. If a value of -1 or less is specified, 0 is assumed. If a value of 255 or more is specified, 255 is assumed. The default value for this parameter is 0. Message transfer is performed within the MPI library in units called packets. One packet has an upper limit value, known as the maximum transmission unit. If a message larger than the maximum transmission unit is being transferred, the message is split into multiple packets so that the size of each packet is the maximum transmission unit or less. This maximum transmission unit can be changed by the MCA parameter common_tofu_packet_mtu. Refer to "Table 4.30 common_tofu_packet_mtu (changes the maximum packet transfer size)" for information on this parameter. Bandwidth can be controlled by adjusting the packet transfer interval time. This may improve communication throughput time when other message communication is being attempted at the same time. For example, if gap value 8 is specified in this parameter, the time interval between packets is exactly the amount of time it takes to transfer one packet and the targeted message transfer bandwidth is adjusted to 1/2. However, depending on the circumstances, performance may deteriorate, so care is essential when using this parameter.

Table 4.30 common_tofu_packet_mtu (changes the maximum packet transfer size)

MCA parameter value	Content
One of the following integer values: 256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536, 1664, or 1792	Message transfer is performed within the MPI library in units called packets. This parameter specifies the packet maximum transmission unit in bytes. Specifiable values for this parameter are either 256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536, 1664, or 1792. If a value that is not in the list is specified, the closest value in the list is assumed. If there are two closest values, the larger value is assumed. The default value for this parameter is 1792. If communication of multiple large messages is attempted, communication throughput times can be improved by changing the value of this parameter in conjunction with specifying the MCA parameter common_tofu_packet_gap. However, depending on the circumstances, performance may deteriorate, so care is essential when using this parameter. Refer to "Table 4.29 common_tofu_packet_gap (changes the packet transfer interval time)" for details.

Table 4.31 common_tofu_use_multi_path (performs point-to-point communication using multiple communication paths)

MCA parameter value	Content
1	Specifies that multiple communication paths are used for point-to-point communication, that is, that trunking is implemented.

MCA parameter value	Content
	This parameter is mutually exclusive to Hasty Rendezvous communication. If 1 is specified for this parameter and 1 or 2 is specified for the MCA parameter pml_ob1_use_hasty_rendezvous, the MCA parameter pml_ob1_use_hasty_rendezvous is ignored. Since multiple TNIs are used to reserve multiple communication paths, the result of this parameter is also affected by the number of usable TNIs. In addition, depending on communication conditions, communication performance may deteriorate, so care is essential when using this parameter. Read "6.7 Using Multiple TNIs" for details of using multiple TNIs.
0	Specifies that multiple communication paths are not used for point-to-point communication. The default value for this parameter is 0.

Table 4.32 dpm_ple_no_establish_connection (specifies that the MPI program does not establish communication using background execution)

MCA parameter value	Content
1	Specifies that the MPI program does not establish communication between two groups of MPI processes that do not share a communicator using background execution of mpiexec command. The number of communication resources allocated to a process is determined by the number of CPUs (cores) allocated to a process. When the MPI program is run with description of the number of CPUs in the VCOORD_FILE file and the numbers of CPUs allocated to processes are not identical, the number of communication resources allocated to each process is aligned with the one for the process that has least CPUs in the entire job. This rule may degrade the performance of Tofu communication on processes that have more CPUs. By specifying this parameter, it is aligned with the one for the process that has least CPUs in the processes spawned by a same mpiexec process, not in the processes in the entire job. This may be able to prevent the communication performance degradation. However, the MPI program ends abnormally if the program establishes communication between two groups of MPI processes that do not share a communicator even this MCA parameter is specified. If background execution of mpiexec command is not used or the numbers of CPUs allocated to processes are identical, there is no benefit to specifying this parameter. Refer to "4.5 VCOORD_FILE file format" for details of the VCOORD_FILE.
0	Specifies that the MPI program may establish communication between two groups of MPI processes that do not share a communicator using background execution of mpiexec command. The default value for this parameter is 0.

Table 4.33 dpm_ple_socket_timeout (specifies the socket communication wait time when establishing communication)

MCA parameter value	Content
A positive integer value	Specifies the socket communication wait time (in seconds) used when establishing communication between MPI process groups that do not share a communicator. If the specified wait time is exceeded, an error message is output and MPI program execution ends. Refer to "6.3.9.1 Socket Communication Wait Time" for details.
0	Sets unlimited as the socket communication wait time used when establishing communication between MPI process groups that do not share a communicator. The default value for this parameter is 0.

Table 4.34 mca_base_param_file_prefix (specifies the AMCA parameter file)

MCA parameter value	Content
File path name of the AMCA parameter file	Interprets the specified file as being an AMCA parameter file (MCA parameter settings file). If an MCA parameter coded within this settings file has already been set as an environment variable, the relevant MCA parameter setting coded in this settings file has no effect. If an invalid file path name is specified, a warning message is output and the AMCA parameter file specification has no effect.

Table 4.35 mpi_check_buffer_write (monitors communication buffer write damage)

MCA parameter value	Content
1	Specifies to monitor for damages when writing to the send buffer during nonblocking communication. Before the send ends, if writing to the send buffer occurred, a message confirming the write and stack trace information are output to the standard error, and MPI program execution ends. Refer to "6.16.2 Monitoring of Write Damage in MPI Communication Buffer" for information on monitoring communication buffer write damages.
0	Specifies to not monitor for damages when writing to the communication buffer. The default value for this parameter is 0.

Table 4.36 mpi_deadlock_timeout (specifies the communication wait timeout time)

MCA parameter value	Content
A positive integer value	Specifies the communication wait timeout time (in seconds) in order to detect deadlocks. If the communication wait time during MPI communication exceeds the time (in seconds) specified in this parameter, a message confirming the waittime and stack trace information are output to the standard error, and MPI program execution ends. Refer to "6.16.1 Deadlock Detection" for information on deadlock detection.
0	Specifies to not implement communication wait timeouts. The default value for this parameter is 0.

Table 4.37 mpi_deadlock_timeout_delay (delays program termination caused by detection of a deadlock)

MCA parameter value	Content
A positive integer value	Specifies the wait time (in seconds) between message output and actual program termination if the deadlock detection function detects a deadlock. This parameter specification is enabled when a positive integer value is specified in the MCA parameter mpi_deadlock_timeout. Specifying this wait time relates to increasing the number of processes targeted for message output and stack trace information output when deadlock detection causes program termination. It may be useful for investigating the location where the deadlock occurred. Refer to "6.16.1 Deadlock Detection" for information on deadlock detection.
0	If the deadlock detection function detects a deadlock, the program ends as soon as the message is output. The default value for this parameter is 0.

Table 4.38 mpi_preconnect_mpi (specifies the timing for establishing connections)

MCA parameter value	Content
Integer value of 1 or more	If this parameter is not specified in this system, Tofu connection is established at the time of first communication with each process that is a communication partner.

MCA parameter value	Content
	If a positive integer value is specified in this parameter, connections are established within the MPI_Init function from all process to all processes that communicate internally. This increases the execution time of the MPI_Init function but makes execution times stable for MPI functions that communicate. Normally, specify 1 for this parameter. If a value of 2 or more is specified, this will make the execution time of the MPI_Init function longer than necessary. If there is no communication between compute nodes, there is no benefit to specifying this parameter.
0	Does not establish Tofu connections within the MPI_Init function. At the point when an MPI function that performs communication is invoked, a connection is established with the communication partner process. This reduces the execution time of the MPI_Init function but may increase the execution time of MPI functions that perform communication by the number of initial communications. The default value for this parameter is 0.

Table 4.39 mpi_print_stats (outputs MPI statistical information)

MCA parameter value	Content
1	Specifies to output MPI statistical information to the standard error output. However, with this specification, the MPI statistical information of all parallel processes is aggregated and output by parallel processes belonging to MPI_COMM_WORLD with a rank number of 0. The MPI statistical information is output in MPI_Finalize function.
2	Specifies to output MPI statistical information to the standard error output. However, with this specification, the MPI statistical information of each parallel process is output separately by each parallel process itself. The MPI statistical information is output when MPI_Finalize function or MPI_Abort function is called or when this system terminates the parallel process due to a detected abnormal state. Process Mapping information is output only at the normal termination. To output statistical information from a particular parallel process, specify the MCA parameter mpi_print_stats_ranks. Refer to "Table 4.40 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)" for details.
3	It is similar to parameter value 1. However, it is necessary to specify the FJMPI_Collection_print function to output it to the standard error output. In addition, the content of the output is output separately for the header department, body department including section line, and the footer department. Refer to "5.3.1.3 FJMPI_Collection_print" for details.
4	It is similar to parameter value 2. However, it is necessary to specify the FJMPI_Collection_print function to output it to the standard error output. In addition, the content of the output is output separately for the header department, body department including section line, and the footer department. Refer to "5.3.1.3 FJMPI_Collection_print" for details.
0	Specifies to not output MPI statistical information. Refer to "6.15 MPI Statistical Information" for information on MPI statistical information. If a value other than an integer from 0 to 2 is specified in this parameter, 0 is assumed. The default value for this parameter is 0.

Table 4.40 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)

MCA parameter value	Content
0 or higher integer value	Specifies the rank number of the parallel process that outputs MPI statistical information. This MCA parameter is enabled only if 2 or 4 is specified for the MCA parameter mpi_print_stats. Specify the rank number belonging to MPI_COMM_WORLD.

MCA parameter value	Content
	Multiple rank numbers can be specified, separated by commas ",". If a rank number that does not exist is specified, it is ignored. Refer to "Table 4.39 mpi_print_stats (outputs MPI statistical information)" for information on the MCA parameter mpi_print_stats.
-1	Specifies to output MPI statistical information from all parallel processes. This MCA parameter is enabled only if 2 or 4 is specified for the MCA parameter mpi_print_stats. Refer to "Table 4.39 mpi_print_stats (outputs MPI statistical information)" for details. If a value of -1 or less is specified in this parameter, a specification of -1 is assumed. The default value for this parameter is -1.

Table 4.41 opal_progress_thread_mode (specifies the operation mode of the MPI asynchronous processing progress thread)

MCA parameter value	Content
1	Specifies to use manual section (without MPI call) mode to promote asynchronous communication using an assistant core. Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details.
2	Specifies to use manual section (with MPI call) mode to promote asynchronous communication using an assistant core.
3	Specifies to use automatic section mode to promote asynchronous communication using an assistant core.
0	Specifies that function of promoting asynchronous communication using an assistant core is not used. The default value for this parameter is 0. If a value smaller than 0 or larger then 3 is specified in this parameter, 0 is assumed.

Table 4.42 orte_abort_print_stack (outputs stack trace informations)

MCA parameter value	Content
1	If MPI_Abort function is called, or if the MPI library ends the execution of the MPI program detecting abnormalities of the execution environment and the communication, stack trace information are output following the error message to the standard error. It might be useful for the specification of the cause of abnormal termination. The default value for this parameter is 1.
0	Stack trace information are not output.

Table 4.43 plm_ple_cpu_affinity (specifies CPU affinity for MPI processes)

MCA parameter value	Content
1	Specifies that the optimum number of CPUs (cores) is bound to each MPI process if neither compiler automatic parallelization function nor OpenMP function is used. If those functions are used, this MCA parameter specification is disabled because CPU (core) binding for parallel threads is performed by the compiler. Which CPU (core) bind to the process is decided based on the numanode_assign_policy in the VCOORD_FILE file or the MCA parameter plm_ple_numanode_assign_policy. The default value for this parameter is 1.
0	Specifies that MPI processes are not bound to CPUs (cores) and they are scheduled by the operating system, if neither compiler automatic parallelization function nor OpenMP function is used. If those functions are used, this MCA parameter specification is disabled because CPU (core) binding for parallel threads is performed by the compiler.

MCA parameter value	Content
	Refer to Fujitsu compiler manuals for details of CPU (core) binding for threads. In case of specifying this MCA parameter value, two or more processes may be bound to one CPU (core), but only rarely. Use of the sched_setaffinity functionin case of specifying this MCA parameter value is recommended. The operation is not guaranteed if a value other than 0 or 1 is specified.

"l" in both the character string pml and ple used in the MCA parameter name is a lowercase "L".

Table 4.44 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)

MCA parameter value	Content
Either of following value localalloc interleave_local interleave_nonlocal interleave_all bind_local bind_nonlocal bind_all prefer_local prefer_nonlocal	Specifies the NUMA memory policy of the MPI processes. The value below can be specified. Refer to "Table 4.53 NUMA memory allocation policy" for details. - localalloc: The memory is allocated from the NUMA node that CPU (core) where the process is allocated belongs. - interleave_local: The memory is alternately allocated from each NUMA node in "Local node set". - interleave_nonlocal: The memory is alternately allocated from each NUMA node in "Non-local node set". - interleave_all: The memory is alternately allocated from each NUMA node in "All node set". - bind_local: The memory allocations will come from the NUMA node that belongs to "Local node set" with the lowest numeric node ID first. - bind_nonlocal: The memory allocations will come from the NUMA node that belongs to "Non-local node set" with the lowest numeric node ID first. - bind_all: The memory is allocated in the NUMA node of "All node set". - prefer_local: The lowest numeric node ID in the NUMA node that belongs to "Local node set" will be selected as the preferred node, then the memory allocations will come from the preferred node. - prefer_nonlocal: The lowest numeric node ID in the NUMA node that belongs to "Non-local node set" will be selected as the preferred node, then the memory allocations will come from the preferred node. Refer to the Job Operation Software manual for information on NUMA node. The specification of the NUMA memory policy is decided by the following priority levels: 1. Specification with VCOORD_FILE file 2. Specification with this parameter 3. (If memory_allocation_policy is not specified in the VCOORD_FILE file, and this parameter is omitted) localalloc

"l" in both the character string pml and ple used in the MCA parameter name is a lowercase "L".

Table 4.45 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)

MCA parameter value	Content
Either of following value simplex share_cyclic share_band	Specifies the CPUs (cores) allocation policy that allocates the MPI process to the NUMA nodes. The value below can be specified. Refer to "Table 4.54 CPU (core) allocation policy" for details. - simplex: The processes are allocated to the NUMA node without sharing with other processes. - share_cyclic: The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in different NUMA nodes.

MCA parameter value	Content
	- share_band: The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in a same NUMA node. Refer to the Job Operation Software manual for information on NUMA node. The specification of the CPUs (cores) allocation policy is decided by the following priority levels: 1. Specification with VCOORD_FILE file 2. Specification with this parameter 3. (If numanode_assign_policy is not specified in the VCOORD_FILE file, and this parameter is omitted) share_cyclic

"l" in both the character string pml and ple used in the MCA parameter name is a lowercase "L".

Table 4.46 pml_ob1_use_hasty_rendezvous (use of Hasty Rendezvous communication)

MCA parameter value	Content
1	Specifies that Hasty Rendezvous communication is used. Read "6.5 Hasty Rendezvous Communication" for details. Usually, with Rendezvous communication, the send side performs write communication (Put) to the receive side. Hasty Rendezvous communication uses this communication (Put).
2	Specify that Hasty Rendezvous communication uses read communication (Get) by the receive side from the send side, in addition to write communication (Put) which is used when 1 is specified for this parameter. Therefore, Hasty Rendezvous communication uses both communications (Put/Get). However, with nonblocking communication when the receive side performs read communication (Get) from the send side and there are repeated attempts to communicate simultaneously with multiple processes, conversely this might cause performance to deteriorate. It is essential to take care with using this specification.
0	Specifies that Hasty Rendezvous communication is not used. The default value for this parameter is 0.

"l" in the character string pml used in the MCA parameter name is a lowercase "L", and the "1" in the ob1 character string is a numeric.

Table 4.47 pml_ob1_use_stride_rdma (use of Stride RDMA communication)

MCA parameter value	Content
1	Specifies that Stride RDMA communication is used. Refer to "6.6 Stride RDMA Communication" for details. The default value for this parameter is 1.
0	Specifies that Stride RDMA communication is not used.

"l" in the character string pml used in the MCA parameter name is a lowercase "L", and the "1" in the ob1 character string is a numeric.

4.3 Environment Variables

When an MPI program is executed by this system, environment variables can be used to control the behavior when the MPI library is executed. The environment variables provided by this system have names starting with the reserved character string "OMPI_".

A dynamically generated parallel process inherits the environment variables of the original parallel process that generated it. However, of all the environment variables set in the program during execution of the original parallel process, only those with names starting with the string "OMPI" are inherited.

The environment variables provided by this system are just items derived from the MCA parameters. By adding "OMPI_MCA_" to the start of an MCA parameter name, the MCA parameter can be used as an environment variable. This is possible for all of the MCA parameters. Refer to "4.2 MCA Parameters" for details.

An example of setting an MCA parameter as an environment variable is shown below.

Example

MCA parameter specification example:

```
-mca  mca_base_param_file_prefix  MCAFILE
```

The MCA parameter "mca_base_param_file_prefix" specifies the AMCA parameter file. Attaching "OMPI_MCA_" to the start of this parameter name allows it to be used as the environment variable name.

Example of the above MCA parameter used as an environment variable:

```
OMPI_MCA_mca_base_param_file_prefix=MCAFILE
```

4.4 mpiexec(1) Return Values

In principle, the mpiexec(1) return values are the values that the user has specified in the MPI program or the values set by the language processor. If there are multiple MPI program processes, the return value of the first process identified internally becomes the mpiexec(1) return value. If an MPI program ends abnormally, the return values of the MPI program that ended abnormally become the return values. If a dynamically generated parallel process ended abnormally, the return value becomes the return value of the abnormally ended dynamic process. If this system ends abnormally, the return values specified by this system become the return values.

However, in this system, the return values shown below are reserved. The return values reserved by this system take priority. Therefore, users must avoid using these return values when setting return values in MPI programs.

Table 4.48 mpiexec(1) return values reserved in this system

mpiexec return value	Explanation
1	Indicates occurrence of an mpiexec(1) option settings error, an internal inconsistency within this system, or a fatal error within an MPI function.
2 to 54	If there is an error class regulated by the MPI standards within an MPI function in the MPI program, the corresponding error code becomes the return value. Refer to "Appendix A Error Class List" for the error classes regulated by the MPI standards.
Logical sum of signal number and 0x80	Indicates the return value if mpiexec(1) or the MPI program ended abnormally. The value is the logical sum of the signal number received when the MPI program ended abnormally and 0x80.
255	Indicates the return value if the parallel execution environment side of Job Operation Software ended abnormally.

4.5 VCOORD_FILE file format

The VCOORD_FILE file specifies coordinates and the number of CPUs (cores) allocated to the processes in the form of the following.

Table 4.49 Form of the coordinates

Type of coordinates	Format
1-dimensional	(X)
2-dimensional	(X,Y)
3-dimensional	(X,Y,Z)

Table 4.50 Form of the number of CPUs (cores)

Type of options	Format
The number of CPUs (cores)	core=N

Table 4.51 Specification of NUMA memory allocation policy

Content	Format
Method of allocating NUMA memory	memory_allocation_policy=value

Either of value in "Table 4.44 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)" can be specified for a policy.

When MCA parameter plm_ple_memory_allocation_policy specification exists and this specification exists in the VCOORD_FILE file, the VCOORD_FILE file specification becomes effective.

When both specifications do not exist, it is equal to the result of specifying "localalloc" for MCA parameter plm_ple_memory_allocation_policy.

Refer to the Job Operation Software manual for information on NUMA node.

Table 4.52 Specification of CPU (core) allocation policy to the NUMA node

Content	Format
Method of allocating CPU (core)	numanode_assign_policy=value

Either of value in "Table 4.45 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)" can be specified for a policy.

When MCA parameter plm_ple_numanode_assign_policy specification exists and this specification exists in the VCOORD_FILE file, the VCOORD_FILE file specification becomes effective.

When both specifications do not exist, it is equal to the result of specifying "share_cyclic" for MCA parameter plm_ple_numanode_assign_policy.

Refer to the Job Operation Software manual for information on NUMA node.

The following examples show the format of the VCOORD_FILE file.

Example

Format 1. Specifying the logical coordinates and the numbers of CPUs (cores)

In this form, both the coordinates with which each process is generated and the numbers of CPUs (cores) allocated to the processes are specified.

```
(0) core=8
(0) core=8
(1) core=4
(1) core=4
(1) core=4
(1) core=4
(2) core=1
(3) core=1
```

Format 2. Specifying only the logical coordinates

In this form, only the coordinates with which each process is generated are specified. The numbers of CPUs (cores) allocated to the processes are decided based on the MCA parameter plm_ple_cpu_affinity by Job Operation Software.

```
(0)
(0)
(1)
(1)
(2)
(2)
```

```
(3)
(3)
```

Format 3. specifying only the number of CPUs (cores)

In this form, only the numbers of CPUs (cores) allocated to the processes are specified. The coordinates with which each process is generated are decided by Job Operation Software.

```
core=8
core=8
core=4
core=4
core=4
core=4
core=1
core=1
```

Format 4. Specifying the NUMA memory allocation policy

In this form, the allocation policy of the NUMA memory in addition to form 1 or 2 or 3 are specified.

```
(0) core=2 memory_allocation_policy=localalloc
```

```
(0) memory_allocation_policy=interleave_local
```

```
core=2 memory_allocation_policy=interleave_all
```

Format 5. Specifying the CPU (core) allocation policy to the NUMA node

In this form, the CPU (core) allocation policy to the NUMA node in addition to form 1 or 2 or 3 are specified.

```
(0) core=2 numanode_assign_policy=simplex
```

```
(0) numanode_assign_policy=share_cyclic
```

```
core=2 numanode_assign_policy=share_band
```

2-dimensional coordinates or 3-dimensional coordinates also can be specified. Moreover, multiple processes can be generated to the same coordinates by writing the same coordinates multiple times.

When two or more of core, memory_allocation_policy, and numanode_assign_policy are specified, there is no restriction in order.

Note

In the following cases, Job Operation Software error occurs

- The number of processes generated with the same coordinates exceeds the number of processes per node decided by "--mpi proc=" option of pjsub command.
- The total number of CPUs (cores) allocated to the process generated with the same coordinates exceeds the number of CPUs (cores) installed in the compute node.
- The lines with coordinates and the lines without coordinates exist together in one VCOORD_FILE file.
- The coordinate is written besides the head of the line.
- The number of processes specified with mpiexec(1) (the number of processes specified with pjsub(1) when the mpiexec(1)'s specification is omitted) exceeds the number of lines of the VCOORD_FILE file.
- There is a possibility that CPU (core) allocation to the process fails due to the lack of the number of CPUs (cores) when one or more processes where "simplex" was specified for CPU (core) allocation policy to the NUMA node exist.

4.6 Settings in NUMA system

The compute nodes in the FX100 system are NUMA system. The MCA parameter is prepared to decrease job execution performance deteriorate because of the memory access speed in the NUMA system.

4.6.1 Setting value of NUMA memory allocation policy

The memory policy can be set by the MCA parameter plm_ple_memory_allocation_policy. Values that can be specified are shown in the table below. Refer to "Table 4.44 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)" for information on the MCA parameter plm_ple_memory_allocation_policy.

In the explanation of this table, NUMA node sets are defined as follows.

- All node set (all)

  Set of all NUMA nodes in a compute node

- Local node set (local)

  Set union of NUMA node that each CPU core where process is allocated belongs

- Non-local node set (nonlocal)

  Set of elements in "All node set" but not in "Local node set"

Table 4.53 NUMA memory allocation policy

Value	Content	Note
localalloc	The memory is allocated from the NUMA node that CPU (core) where the process is allocated belongs. If that NUMA node contains no free memory, the system will attempt to allocate memory from a "nearby" node.	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_DEFAULT,NULL, ..)
interleave_local	The memory is alternately allocated from each NUMA node in "Local node set". The memory is allocated from the next NUMA node in "Local node set" when there is no remainder capacity in the memory of the NUMA node that tried to be allocated. It operates according to the specification of OS when there is no remainder capacity of the memory of all NUMA nodes that belongs to the "Local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_INTERLEAVE,local, ..)
interleave_nonlocal	The memory is alternately allocated from each NUMA node in "Non-local node set". The memory is allocated from the next NUMA node in "Non-local node set" when there is no remainder capacity in the memory of the NUMA node that tried to be allocated. It operates according to the specification of OS when there is no remainder capacity of the memory of all NUMA nodes that belongs to the "Non-local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_INTERLEAVE,nonlocal, ..) It fails in the call of set_mempolicy(2) when "Non-local node set" is empty. In that case, the warning message PLE 0601 is output to the standard error output of the job, and processing is continued. In this case, the NUMA memory allocation policy of parallel processes is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_DEFAULT,NULL, ..)

Value	Content	Note
interleave_all	The memory is alternately allocated from each NUMA node in "All node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_INTERLEAVE,all, ..)
bind_local	The memory allocations will come from the NUMA node that belongs to "Local node set" with the lowest numeric node ID first. It fails in the allocation when there is no remainder capacity of the memory of the NUMA node that belongs to "Local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,local, ..)
bind_nonlocal	The memory allocations will come from the NUMA node that belongs to "Non-local node set" with the lowest numeric node ID first. It fails in the allocation when there is no remainder capacity of the memory of the NUMA node that belongs to "Non-local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,nonlocal, ..) It fails in the call of set_mempolicy(2) when "Non-local node set" is empty. In that case, the warning message PLE 0601 is output to the standard error output of the job, and processing is continued. In this case, the NUMA memory allocation policy of parallel processes is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_DEFAULT,NULL, ..)
bind_all	The memory is allocated in the NUMA nodes of "All node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,all, ..)
prefer_local	The lowest numeric node ID in the NUMA node that belongs to "Local node set" will be selected as the preferred node. The memory allocation is done the preferred node by priority. If that NUMA node contains no free memory, the system will attempt to allocate memory from a "nearby" node.	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_PREFERRED,local, ..)
prefer_nonlocal	The lowest numeric node ID in the NUMA node that belongs to "Non-local node set" will be selected as the preferred node. The memory allocation is done the preferred node by priority. If that NUMA node contains no free memory, the system will attempt to allocate memory from a "nearby" node.	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_PREFERRED,nonlocal, ..) The specification of "nonlocal" is disregarded when "Non-local node set" does not exist, and it is equal to the result of specifying "localalloc".

4.6.2 Setting value of CPU (core) allocation policy

The CPU (core) allocation policy can be set by the MCA parameter plm_ple_numanode_assign_policy. Values that can be specified are shown in the table below. Refer to "Table 4.45 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)" for information on the MCA parameter plm_ple_numanode_assign_policy.

Table 4.54 CPU (core) allocation policy

Value	Content
simplex	The processes are allocated to the NUMA node without sharing with other processes.
share_cyclic	The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in a different NUMA node.
share_band	The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in a same NUMA node.

The allocation image is shown as follows.

Figure 4.1 Example of allocating process for simplex

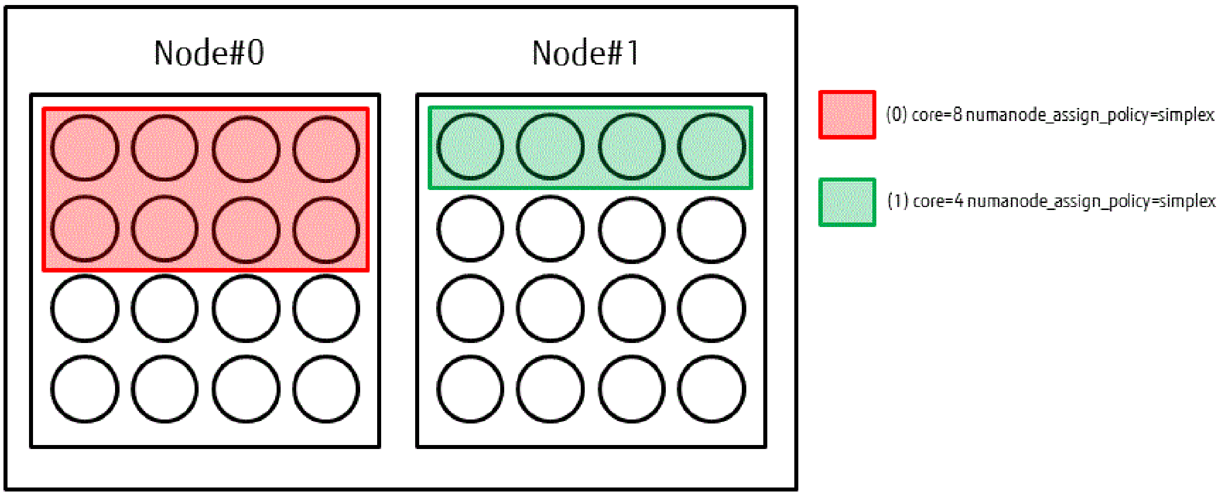

Node#0
Node#1
(0) core=8 numanode_assign_policy=simplex
(1) core=4 numanode_assign_policy=simplex

Figure 4.2 Example of allocating process for share_cyclic

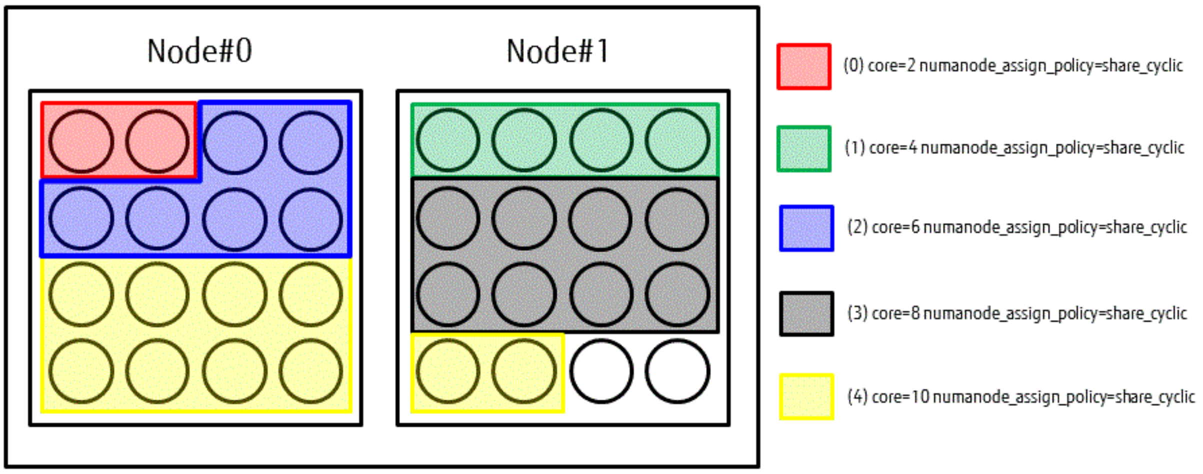
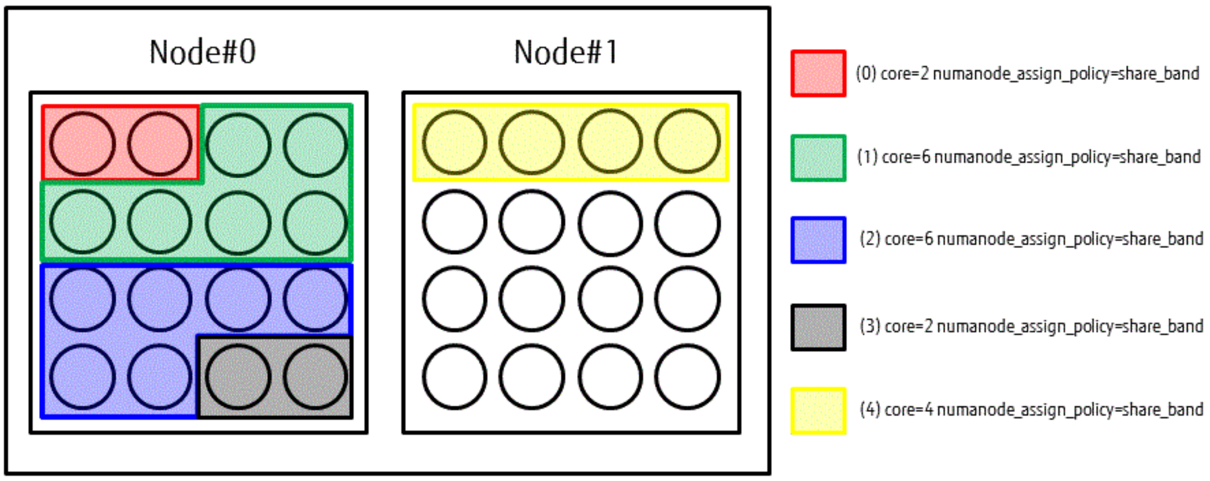

Node#0
Node#1
(0) core=2 numanode_assign_policy=share_cyclic
(1) core=4 numanode_assign_policy=share_cyclic
(2) core=6 numanode_assign_policy=share_cyclic
(3) core=8 numanode_assign_policy=share_cyclic
(4) core=10 numanode_assign_policy=share_cyclic

Figure 4.3 Example of allocating process for share_band


Node#0
Node#1
(0) core=2 numanode_assign_policy=share_band
(1) core=6 numanode_assign_policy=share_band
(2) core=6 numanode_assign_policy=share_band
(3) core=2 numanode_assign_policy=share_band
(4) core=4 numanode_assign_policy=share_band

Chapter 5 Extended Interfaces

This chapter describes the following MPI extended interfaces provided by this system:

- Rank query interface
- Extended RDMA interface
- Section specifying MPI statistical information interface
- Extended persistent communication requests interface
- MPI asynchronous communication promotion section specifying interface

Information

Supplementation 1) Rank query interface, Extended RDMA interface

Note that these extended interfaces support only the C language, and cannot be used in the following cases.

- When the MPI program uses dynamic process generation
- When the job type is node-sharing job

  Refer to the Job Operation Software manual for information on node-sharing job.

Supplementation 2) Section specifying MPI statistical information interface, Extended persistent communication requests interface, MPI asynchronous communication promotion section specifying interface

This extended interface supports C language and Fortran.

Use C interface in C++.

5.1 Rank Query Interface

This system can execute MPI programs in logical node space that has a torus structure of from one to three dimensions. Job Operation Software allocates logical coordinates in this logical node space. These logical coordinates may be referred to simply as coordinates. An MPI program with a torus structure process shape can be deployed at a suitable location within this logical node space.

It is useful to know from within the MPI program the position (coordinates) where each parallel process rank of the MPI program is deployed in the torus structure process shape. For example, this system normally deploys two parallel processes with neighboring torus structure shape coordinates such that they are physically at a distance of one hop. Knowing the rank numbers of two neighboring parallel processes enables communication performance to be considered when programming.

Table below shows a list of concrete functions for the rank query interface provided by this system.

Table 5.1 Rank query interface function list

Function name	Function overview
FJMPI_Topology_get_dimension	Gets the number of dimensions given to MPI_COMM_WORLD
FJMPI_Topology_get_shape	Gets the process shape given to MPI_COMM_WORLD
FJMPI_Topology_rank2x	Gets the X coordinate value from the rank number
FJMPI_Topology_rank2xy	Gets the XY coordinate value from the rank number
FJMPI_Topology_rank2xyz	Gets the XYZ coordinate value from the rank number
FJMPI_Topology_x2rank	Gets the rank number from the X coordinate value
FJMPI_Topology_xy2rank	Gets the rank number from the XY coordinate value
FJMPI_Topology_xyz2rank	Gets the rank number from the XYZ coordinate value

Function name	Function overview
FJMPI_Topology_cart_reorder	Gets the value that determines the rank of a communicator with a Cartesian structure
FJMPI_Topology_sys_rank2xyzabc	Gets the Tofu coordinates from the rank number
FJMPI_Topology_sys_xyzabc2rank	Gets the rank number from the Tofu coordinates
FJMPI_Topology_rel_rank2xyzabc	Gets the relative Tofu coordinates from the rank number
FJMPI_Topology_rel_xyzabc2rank	Gets the rank number from the relative Tofu coordinates

5.1.1 Querying the Number of Dimensions and Shape

5.1.1.1 FJMPI_Topology_get_dimension

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_get_dimension(int *size)
```

<Explanation>

This query returns the number of dimensions in the process shape where the MPI processes belonging to the MPI_COMM_WORLD generated internally when MPI_Init is executed are deployed.

Type	Variable	Explanation	IN/OUT
int*	size	Number of dimensions in the process shape of processes belonging to MPI_COMM_WORLD	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.1.2 FJMPI_Topology_get_shape

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_get_shape(int *x, int *y, int *z)
```

<Explanation>

This query returns the MPI parallel process shape XYZ given to the MPI_COMM_WORLD generated internally when the MPI_Init function is executed.

Type	Variable	Explanation	IN/OUT
int*	x	Size of the X axis of the process shape given to MPI_COMM_WORLD	OUT
int*	y	Size of the Y axis of the process shape given to MPI_COMM_WORLD	OUT
int*	z	Size of the Z axis of the process shape given to MPI_COMM_WORLD	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The Y axis and Z axis values are 0 if the process shape is one-dimensional. The Z axis value is 0 if the process shape is two-dimensional.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.2 Querying the Coordinates

5.1.2.1 FJMPI_Topology_rank2x

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_rank2x(int rank, int *x)
```

<Explanation>

This query returns the X coordinate value internally allocated to each parallel process when the MPI_Init function is executed.

Type	Variable	Explanation	IN/OUT
int	rank	Specify the rank number of the parallel process for which the coordinate is to be fetched.	IN
int*	x	Value of the X coordinate of the parallel process corresponding to the rank number	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though the process shape when the job was executed was two-dimensional or three-dimensional
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The rank number to be specified must be within the rank number range given to MPI_COMM_WORLD.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.2.2 FJMPI_Topology_rank2xy

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_rank2xy(int rank, int *x, int *y)
```

<Explanation>

This query returns the XY coordinate values internally allocated to each parallel process when the MPI_Init function is executed.

Type	Variable	Explanation	IN/OUT
int	rank	Specify the rank number of the parallel process for which the coordinates are to be fetched.	IN
int*	x	Value of the X coordinate of the parallel process corresponding to the rank number	OUT
int*	y	Value of the Y coordinate of the parallel process corresponding to the rank number	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though the process shape when the job was executed was one-dimensional or three-dimensional
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The rank number to be specified must be within the rank number range given to MPI_COMM_WORLD.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.2.3 FJMPI_Topology_rank2xyz

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_rank2xyz(int rank, int *x, int *y, int *z)
```

<Explanation>

This query returns the XYZ coordinate values internally allocated to each parallel process when the MPI_Init function is executed.

Type	Variable	Explanation	IN/OUT
int	rank	Specify the rank number of the parallel process for which the coordinates are to be fetched.	IN
int*	x	Value of the X coordinate of the parallel process corresponding to the rank number	OUT
int*	y	Value of the Y coordinate of the parallel process corresponding to the rank number	OUT
int*	z	Value of the Z coordinate of the parallel process corresponding to the rank number	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process

		- If this function was called even though the process shape when the job was executed was one-dimensional or two-dimensional
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The rank number to be specified must be within the rank number range given to MPI_COMM_WORLD.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.2.4 FJMPI_Topology_sys_rank2xyzabc

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_sys_rank2xyzabc(int rank, int *x, int *y, int *z, int *a, int *b, int *c)
```

<Explanation>

This query returns the Tofu coordinate values allocated to each parallel process when the MPI_Init function is executed.

The coordinates actually allocated on the system are returned.

Type	Variable	Explanation	IN/OUT
int	rank	Specify the rank number of the parallel process for which the coordinates are to be fetched.	IN
int*	x	Value of the Tofu coordinates X coordinate of the parallel process corresponding to the rank number	OUT
int*	y	Value of the Tofu coordinates Y coordinate of the parallel process corresponding to the rank number	OUT
int*	z	Value of the Tofu coordinates Z coordinate of the parallel process corresponding to the rank number	OUT
int*	a	Value of the Tofu coordinates A coordinate of the parallel process corresponding to the rank number	OUT
int*	b	Value of the Tofu coordinates B coordinate of the parallel process corresponding to the rank number	OUT
int*	c	Value of the Tofu coordinates C coordinate of the parallel process corresponding to the rank number	OUT

<Return value>

Normal	0	
Error	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The rank number to be specified must be within the rank number range given to MPI_COMM_WORLD.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.2.5 FJMPI_Topology_rel_rank2xyzabc

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_rel_rank2xyzabc(int rank, int *x, int *y, int *z, int *a, int *b, int *c)
```

<Explanation>

This query returns the relative Tofu coordinates, the base coordinates being a node within space allocated by the job that executes the MPI program, of the node with the corresponding rank.

The base coordinates are the rank 0 that applies when the rank-map-hostfile is not specified in the --mpi option of the pjsub command.

Type	Variable	Explanation	IN/OUT
int	rank	Specify the rank number of the parallel process for which the coordinates are to be fetched.	IN
int*	x	Value of the relative Tofu coordinates X coordinate of the parallel process corresponding to the rank number	OUT
int*	y	Value of the relative Tofu coordinates Y coordinate of the parallel process corresponding to the rank number	OUT
int*	z	Value of the relative Tofu coordinates Z coordinate of the parallel process corresponding to the rank number	OUT
int*	a	Value of the relative Tofu coordinates A coordinate of the parallel process corresponding to the rank number	OUT
int*	b	Value of the relative Tofu coordinates B coordinate of the parallel process corresponding to the rank number	OUT
int*	c	Value of the relative Tofu coordinates C coordinate of the parallel process corresponding to the rank number	OUT

<Return value>

Normal	0	
Error	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

The rank number to be specified must be within the rank number range given to MPI_COMM_WORLD.

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

This function cannot be used when the job type is mesh mode or non-contiguous mode.

Refer to the Job Operation Software manual for information on mesh mode and non-contiguous mode.

5.1.3 Querying the Rank

5.1.3.1 FJMPI_Topology_x2rank

< Format >

```
#include <mpi-ext.h>
int FJMPI_Topology_x2rank(int x, int *rank)
```

<Explanation>

From the specified X coordinate value, this query returns the parallel process rank number allocated to MPI_COMM_WORLD.

Type	Variable	Explanation	IN/OUT
int	x	Specify the X coordinate value of the parallel process rank number to be fetched.	IN
int*	rank	Parallel process rank number deployed to the coordinate	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though the process shape when the job was executed was two-dimensional or three-dimensional - If this function was called even though multiple parallel processes are allocated within the node
	-2	- If no parallel processes are deployed to the specified coordinate
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.3.2 FJMPI_Topology_xy2rank

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_xy2rank(int x, int y, int *rank)
```

<Explanation>

From the specified XY coordinate values, this query returns the parallel process rank number allocated to MPI_COMM_WORLD.

Type	Variable	Explanation	IN/OUT
int	x	Specify the X coordinate value of the parallel process rank number to be fetched.	IN
int	y	Specify the Y coordinate value of the parallel process rank number to be fetched.	IN
int*	rank	Parallel process rank number deployed to the coordinate	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though the process shape when the job was executed was one-dimensional or three-dimensional - If this function was called even though multiple parallel processes are allocated within the node
	-2	- If no parallel processes are deployed to the specified coordinate
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.3.3 FJMPI_Topology_xyz2rank

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_xyz2rank(int x, int y, int z, int *rank)
```

<Explanation>

From the specified XYZ coordinate values, this query returns the parallel process rank number allocated to MPI_COMM_WORLD.

Type	Variable	Explanation	IN/OUT
int	x	Specify the X coordinate value of the parallel process rank number to be fetched.	IN
int	y	Specify the Y coordinate value of the parallel process rank number to be fetched.	IN
int	z	Specify the Z coordinate value of the parallel process rank number to be fetched.	IN
int*	rank	Parallel process rank number deployed to the coordinate	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though the process shape when the job was executed was one-dimensional or two-dimensional - If this function was called even though multiple parallel processes are allocated within the node
	-2	- If no parallel processes are deployed to the specified coordinate
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.3.4 FJMPI_Topology_sys_xyzabc2rank

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_sys_xyzabc2rank(int x, int y, int z, int a, int b, int c, int *rank)
```

<Explanation>

From the specified Tofu coordinate values, this query returns the rank number of parallel process allocated to MPI_COMM_WORLD.

Type	Variable	Explanation	IN/OUT
int	x	Specify the X coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	y	Specify the Y coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	z	Specify the Z coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	a	Specify the A coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	b	Specify the B coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	c	Specify the C coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int*	rank	Parallel process rank number deployed to the coordinate	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though multiple parallel processes are allocated within the node
	-2	- If no parallel processes are deployed to the specified coordinate
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.3.5 FJMPI_Topology_rel_xyzabc2rank

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_rel_xyzabc2rank(int x, int y, int z, int a, int b, int c, int *rank)
```

<Explanation>

From the specified relative Tofu coordinates, this query returns the rank number of parallel process allocated to MPI_COMM_WORLD.

The base coordinates are the rank 0 that applies when the rank-map-hostfile is not specified in the --mpi option of the pjsub command.

Type	Variable	Explanation	IN/OUT
int	x	Specify the X coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	y	Specify the Y coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	z	Specify the Z coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	a	Specify the A coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN

Type	Variable	Explanation	IN/OUT
int	b	Specify the B coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int	c	Specify the C coordinate value of the Tofu coordinates of the parallel process rank number to be fetched.	IN
int*	rank	Parallel process rank number deployed to the coordinate	OUT

<Return value>

Normal	0	
Error	-1	- If this function was called from a dynamically generated MPI process - If this function was called even though multiple parallel processes are allocated within the node
	-2	- If no parallel processes are deployed to the specified coordinate
	-3	- If the job type is node-sharing job Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

This function cannot be used when the job type is mesh mode or non-contiguous mode.

Refer to the Job Operation Software manual for information on mesh mode and non-contiguous mode.

5.1.4 Querying the Ranking of a Communicator that Has a Cartesian Structure

5.1.4.1 FJMPI_Topology_cart_reorder

<Format>

```
#include <mpi-ext.h>
int FJMPI_Topology_cart_reorder(MPI_Comm comm, int *reorder)
```

<Explanation>

This query returns information used to determine whether or not rankings were executed from the topology information of a communicator that has a Cartesian structure. If the value of the reorder argument is 1, this indicates that rank reordering is implemented. If the value of the reorder argument is 0, this indicates that rank reordering is not implemented.

Type	Variable	Explanation	IN/OUT
MPI_Comm	comm	Communicator for which ranking is to be determined	IN
int*	reorder	Communicator ranking information	OUT

<Return value>

Normal	0	
Error	-1	- If an inter-group communicator was specified - If a communicator that does not have a Cartesian structure was specified

<Notes>

If any of the following conditions apply, the behavior is uncertain and not guaranteed:

- This function is called before the MPI_Init function is executed
- This function is called after the MPI_Finalize function is executed

5.1.5 Sample Program

A sample program of a rank query interface is shown below.

This program performs the following processing with assumption that the process shape is three-dimensional.

1. Queries the MPI process number of dimensions and process shape
2. Obtains the coordinates of this process and the neighboring process for each coordinate, and gets the rank information from those coordinates
3. Obtains the coordinates for each coordinate from the rank information queried in 2 above, and checks that these are the same as the original coordinates

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <mpi.h>
#include <mpi-ext.h>

/* Extended function return value */
#define ERR_USER_OP -1
#define ERR_NO_PROC -2

#define FAILURE 1

int main(int argc, char *argv[])
{
    int x, y, z, i, size, rank;
    int rc, dim;
    int shape_x, shape_y, shape_z;
    int tmp_x, tmp_y, tmp_z;
    int next_x, next_y, next_z;
    int left, right, high, low, far, near, ans;
    char host[255];

    gethostname(host, 255);

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    rc = FJMPI_Topology_get_dimension(&dim);
    if (MPI_SUCCESS != rc) {
        fprintf(stderr, "[%s] FJMPI_Topology_get_dimension ERROR\n", host);
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }

    /* Result check */
    if (3 != dim) {
        fprintf(stderr, "[%s] Dimension size ERROR\n", host);
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }

    rc = FJMPI_Topology_get_shape(&shape_x, &shape_y, &shape_z);
    if (MPI_SUCCESS != rc) {
        fprintf(stderr, "[%s] FJMPI_Topology_get_shape ERROR\n", host);
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }
```

```
/***************************************************************************
 * Get own coordinates                                                    *
 ***************************************************************************/
rc = FJMPI_Topology_rank2xyz(rank, &x, &y, &z);
if (MPI_SUCCESS != rc) {
    fprintf(stderr, "[%s] FJMPI_Topology_rank2xyz ERROR\n", host);
    MPI_Abort(MPI_COMM_WORLD, FAILURE);
}

/***************************************************************************
 * Get neighboring processes before and after each coordinate *
 ***************************************************************************/
for(i=0; i < 6; i++)
{
    switch(i)
    {
        case 0:        /* Neighboring X axis */
        case 1:
            ans = (0 == i) ? left : right;
            tmp_x = (0 == i) ?
                      (x - 1 >= 0) ? x - 1 : shape_x - 1
                    : (x + 1 < shape_x) ? x + 1 : 0;
            tmp_y = y;
            tmp_z = z;
            break;
        case 2:        /* Neighboring Y axis */
        case 3:
            ans = (2 == i) ? low : high;
            tmp_y = (2 == i) ?
                      (y - 1 >= 0) ? y - 1 : shape_y - 1
                    : (y + 1 < shape_y) ? y + 1 : 0;
            tmp_x = x;
            tmp_z = z;
            break;
        case 4:        /* Neighboring Z axis  */
        case 5:
            ans = (4 == i) ? near : far;
            tmp_z = (4 == i) ?
                      (z - 1 >= 0) ? z - 1 : shape_z - 1
                    : (z + 1 < shape_z) ? z + 1 : 0;
            tmp_x = x;
            tmp_y = y;
            break;
    }
    rc = FJMPI_Topology_xyz2rank(tmp_x, tmp_y, tmp_z, &ans);
    switch (rc) {
        case MPI_SUCCESS:
            break;
        case ERR_USER_OP:
            fprintf(stderr, "[%s] USER OP ERROR\n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        case ERR_NO_PROC:
            /* Searches until the closest MPI process is found */
            while (rc == ERR_NO_PROC) {
                tmp_x = tmp_x - 1;
                rc = FJMPI_Topology_xyz2rank(tmp_x, tmp_y, tmp_z, &ans);
            }
            break;
        default:
            fprintf(stderr, "[%s] FATAL ERROR\n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }
```

```
        /************************************************************************
         * Checks if used coordinates and fetched coordinates are the same *
         ************************************************************************/
        rc = FJMPI_Topology_rank2xyz(ans, &next_x, &next_y, &next_z);
        if (MPI_SUCCESS != rc) {
            fprintf(stderr, "[%s] FJMPI_Topology_rank2xyz ERROR\n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }

        if ((next_x != tmp_x) || (next_y != tmp_y) || (next_z != tmp_z)) {
            fprintf(stderr, "[%s] PARAM ERROR\n", host);
            fprintf(stderr, "[%s %d] [user:%u-%u-%u] [get:%u-%u-%u] [rank|next:%d|%d]\n",
                    host, i, tmp_x, tmp_y, tmp_z, next_x, next_y, next_z, rank, ans);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
    }

    MPI_Finalize();

    return 0;

}
```

5.2 Extended RDMA Interface

Note

The extended RDMA interface is an interface for the senior of the communication programming. When using this interface, well versed in the specification of the Tofu interconnect is required. Use this interface with extreme caution.

Use of this interface enables communication that maximizes the use of the Tofu interconnect physical configuration, such as communication using four TNIs and communication using detour routes.

See table below for a list of the practical functions for the extended RDMA interface supported by this system.

Table 5.2 List of functions supported by the extended RDMA interface

Function name	Overview of feature
FJMPI_Rdma_init	Initialization of the extended RDMA interface
FJMPI_Rdma_finalize	Extended RDMA interface end processing
FJMPI_Rdma_reg_mem	Memory registration
FJMPI_Rdma_dereg_mem	Release of memory registration
FJMPI_Rdma_get_remote_addr	Fetches remote DMA address
FJMPI_Rdma_put	RDMA WRITE communication
FJMPI_Rdma_get	RDMA READ communication
FJMPI_Rdma_armw	RDMA ARMW communication
FJMPI_Rdma_poll_cq	RDMA completion confirmation
FJMPI_Rdma_poll_cq_ret_data	RDMA completion confirmation and data fetch associated with the communication

This system supports operations in a multi-thread environment. The thread support level of this system is MPI_THREAD_SERIALIZED. See Section "6.3.5 Operations in a Multi-Threaded Environment" for details.

5.2.1 Extended RDMA Interface Assumed Knowledge

5.2.1.1 Terminology

Term	Meaning
TNI (Tofu Network Interface)	A physical NIC that is used for communication in the Tofu Interconnect. Each node has 4 TNIs. Each TNI has an instruction queue and a completion queue.
DMA address	A virtual address used to specify an area to exchange data in a buffer during RDMA communication. By adding an offset to a DMA address that can be obtained using FJMPI_Rdma_get_remote_addr function or FJMPI_Rdma_reg_mem function, data from any location in the buffer can be sent.
Memory ID	The ID for identifying a buffer. 0 to 510 can be used as the memory ID.
Message tag number	The number arbitrarily specified by the user for identifying send data when RDMA communication is used for sending. 0 to 14 can be used as the message tag number.
Virtual NIC	A virtual communication resource. A process has 4 virtual NICs and each NIC is associated with a physical NIC (TNIs). The correspondence between physical NICs (TNIs) and virtual NICs is 1:1 if there is one node for one process. If there are multiple processes at one node, multiple virtual NICs may correspond to one physical NIC (TNI).
Instruction queue	A queue that stores communication instructions. The instruction queue has a maximum capacity of 1,997 entries.
Completion queue (CQ)	A queue for notifying completion. The completion queue has a maximum capacity of 114,688 entries by default.
Local node	A node that sends a request of RDMA communication and receives response packets.
Remote node	A node that receives a request of RDMA communication and sends response packets.

5.2.1.2 RDMA Communication Execution Model

5.2.1.2.1 Preparation

Before communication starts, the user program must prepare by fetching the DMA addresses of the local node and the remote node buffers. Accordingly, the user program at local and remote nodes allocates a memory ID to each of its own buffers using FJMPI_Rdma_reg_mem function ("Figure 5.1 Registering buffers"). Then, the local node uses the FJMPI_Rdma_get_remote_addr function to request the buffer DMA addresses and allocated memory IDs to be fetched from the remote node ("Figure 5.2 Fetching remote node DMA addresses").

Figure 5.1 Registering buffers

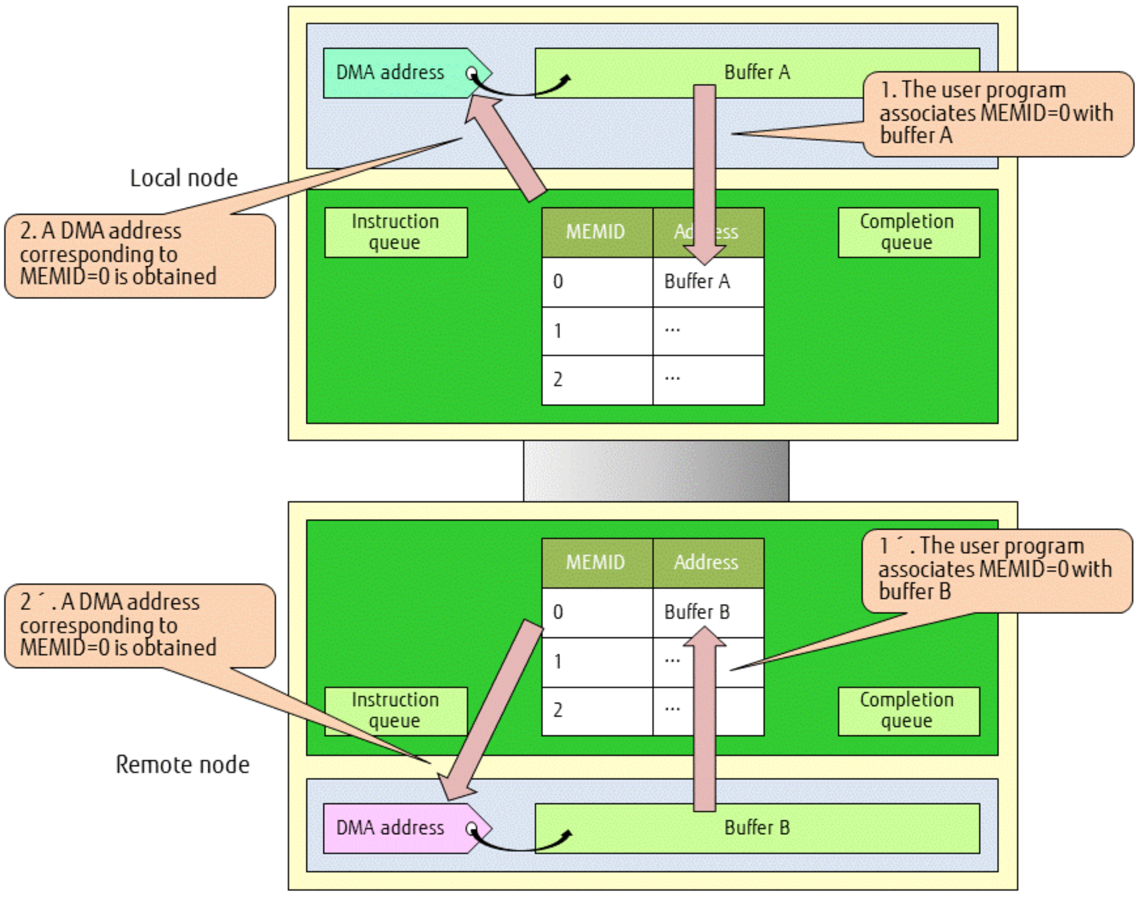
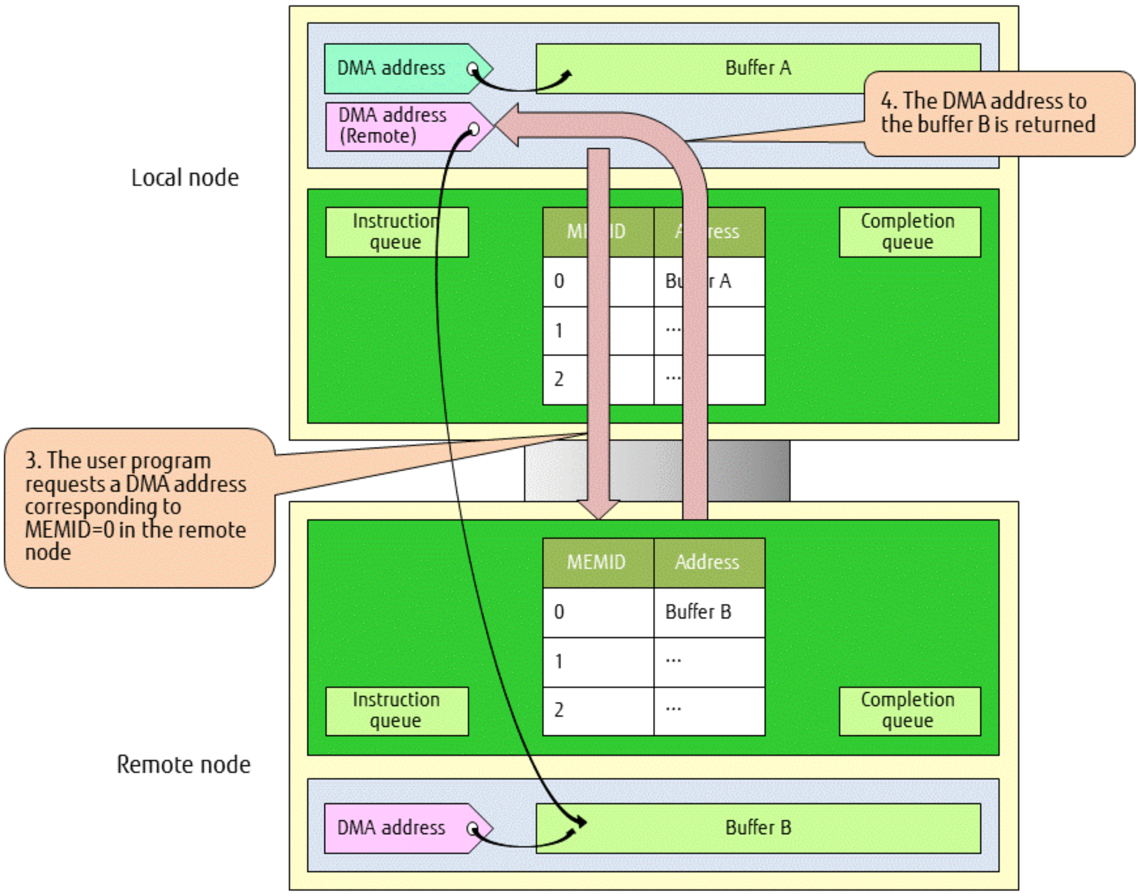
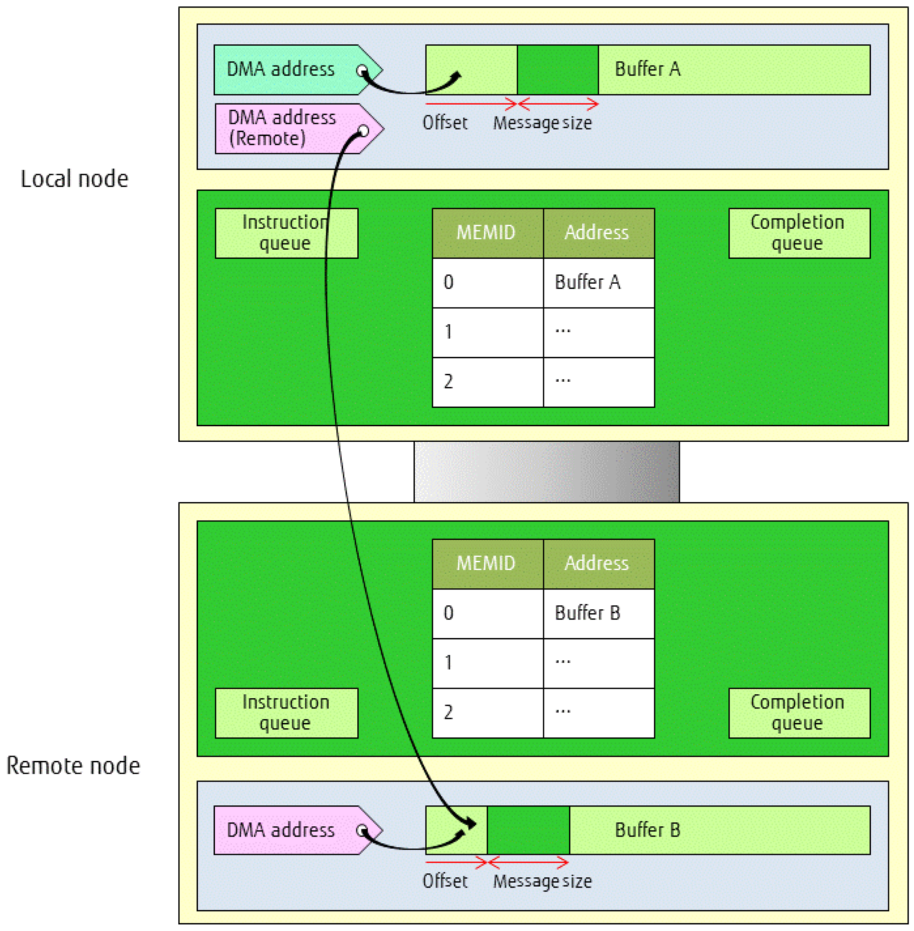

DMA address
Buffer A
1. The user program associates MEMID=0 with buffer A
Local node
2. A DMA address corresponding to MEMID=0 is obtained
Instruction queue
MEMID
Address
0
Buffer A
1
…
2
…
Completion queue
MEMID
Address
0
Buffer B
1
…
2
…
1´. The user program associates MEMID=0 with buffer B
2´. A DMA address corresponding to MEMID=0 is obtained
Instruction queue
Completion queue
Remote node
DMA address
Buffer B

Figure 5.2 Fetching remote node DMA addresses



The local node is then able to use the DMA addresses to read data from any area in the buffer. The area can be specified by adding an offset to a DMA address ("Figure 5.3 Specifying an area in a buffer").

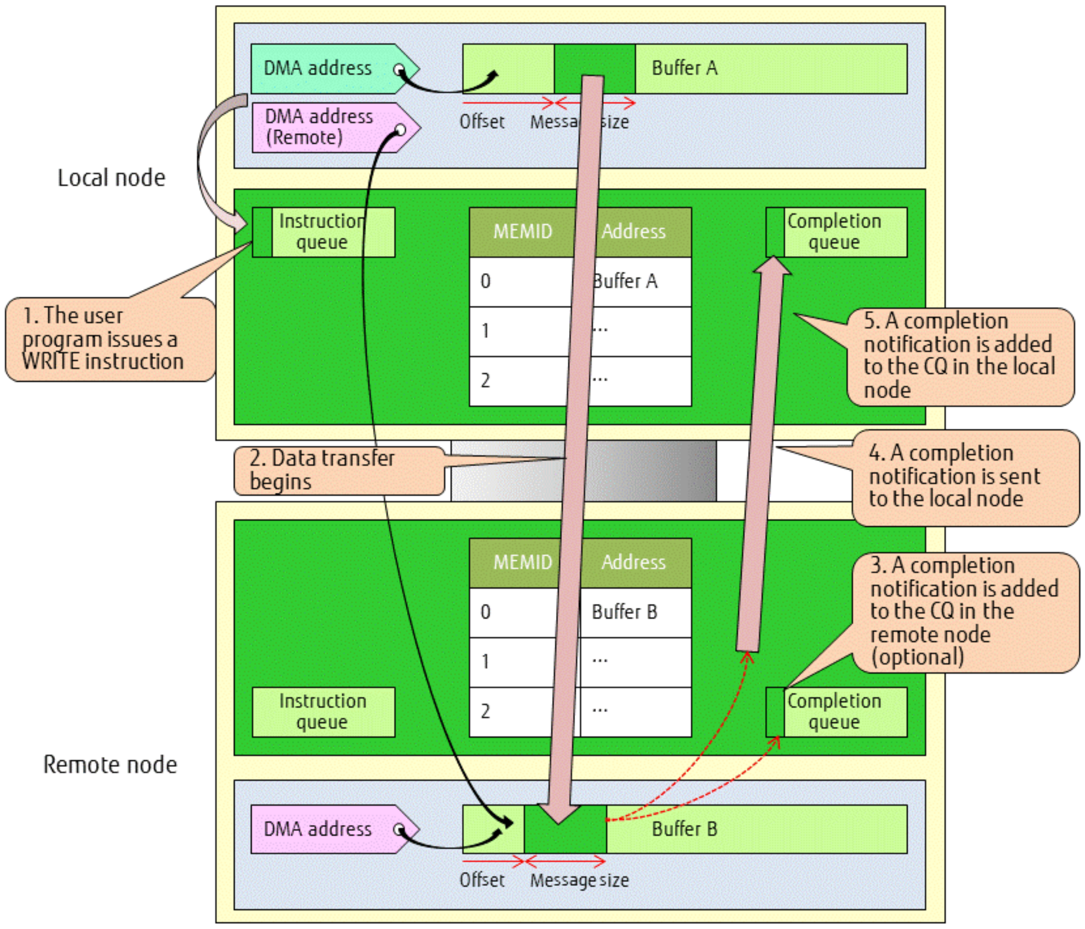
Figure 5.3 Specifying an area in a buffer



5.2.1.2.2 RDMA WRITE

When RDMA WRITE communication is performed, a send instruction is first stored in the instruction queue. When the send instruction is executed by a TNI, the send data is split into packets and sent to the remote node. When all the sent data has been written to the remote node, a notification is stored in the completion queue at the remote node when the RDMA WRITE instruction is issued with a flag to notify at the remote node. The remote node sends a RDMA WRITE communication completion notification to the local node. When the local node receives this notification, the notification is stored in the completion queue at the local node.

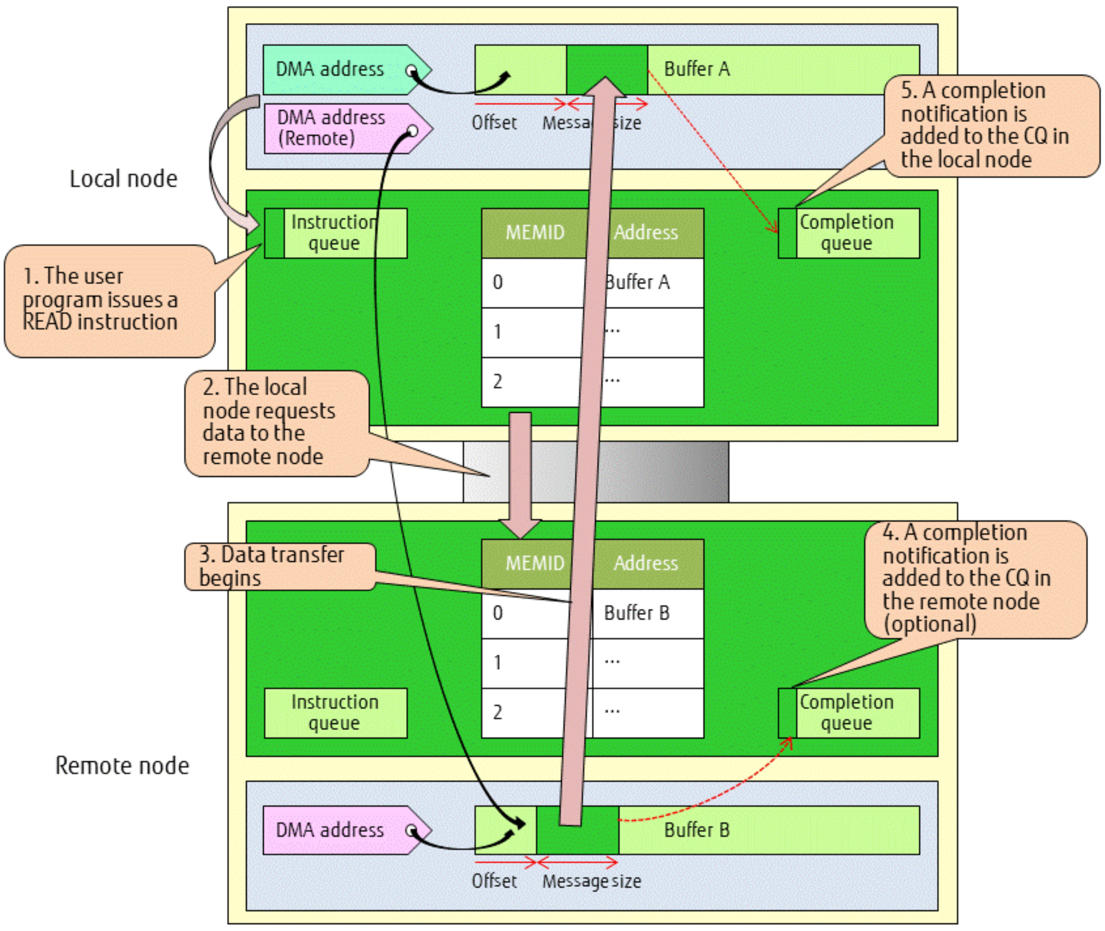
Figure 5.4 RDMA WRITE flow


DMA address
Buffer A
DMA address (Remote)
Offset
Message size
Local node
Instruction queue
MEMID
Address
0
Buffer A
1
…
2
…
Completion queue
1. The user program issues a WRITE instruction
5. A completion notification is added to the CQ in the local node
2. Data transfer begins
4. A completion notification is sent to the local node
3. A completion notification is added to the CQ in the remote node (optional)
Buffer B
Remote node
DMA address
Buffer B

5.2.1.2.3 RDMA READ

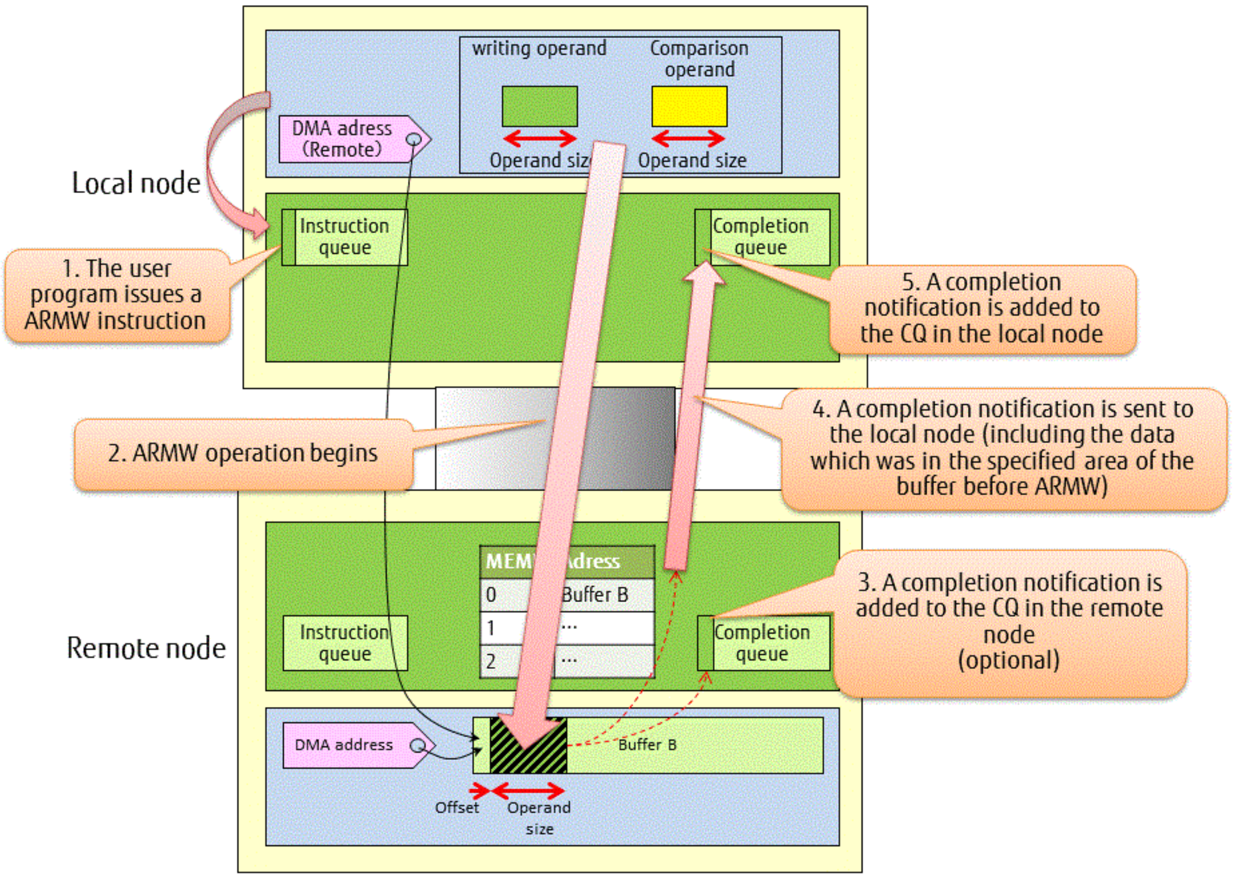
When RDMA READ communication is performed, a send request instruction is first stored in the instruction queue. When the send request instruction is executed by a TNI, the send request is sent to the remote node. When a send request is received, the remote node splits the send data into packets and sends it to the local node. As soon as the data send processing is completed at the remote node, a notification is stored in the completion queue at the remote node when the RDMA READ instruction is issued with a flag to notify at the remote node. When all the data has been written in the buffer at the local node, a notification is stored in the completion queue at the local node.

Figure 5.5 RDMA READ flow



5.2.1.2.4 RDMA ARMW

Figure 5.6 RDMA ARMW flow


writing operand
Comparison operand
DMA adress (Remote)
Operand size
Operand size
Local node
Instruction queue
Completion queue
1. The user program issues a ARMW instruction
5. A completion notification is added to the CQ in the local node
4. A completion notification is sent to the local node (including the data which was in the specified area of the buffer before ARMW)
2. ARMW operation begins
MEM address
0
Buffer B
1
2
Remote node
Instruction queue
Completion queue
3. A completion notification is added to the CQ in the remote node (optional)
DMA address
Buffer B
Offset
Operand size

RDMA ARMW communication is the communication that the local node performs Atomic Read Modify Write (ARMW) to a remote node buffer. It is guaranteed that ARMW is performed atomically against other TNIs and CPUs. Therefore, while ARMW is performed to a remote node buffer, the memory accesses for the same buffer from other TNIs and CPUs is kept waiting. ARMW can be performed only when the address of the specified area is 4-byte alignment and its size is 4-byte or its address is 8-byte alignment and its size is 8-byte.

"Table 5.3 Operations that can be specified for RDMA ARMW communication" shows the six ARMW operations that can be specified for RDMA ARMW communication.

Table 5.3 Operations that can be specified for RDMA ARMW communication

ARMW operation name	Operation details
Compare and Swap	If the data in the specified area of the remote node buffer is equal to the comparison operand, rewrite that data to the writing operand. Otherwise, do nothing.
Swap	Rewrite the data in the specified area of the remote node buffer to the writing operand.
Fetch and Add	Rewrite the data in the specified area of the remote node buffer to the result of add operation with that data and the writing operand.
XOR	Rewrite the data in the specified area of the remote node buffer to the result of XOR operation with that data and the writing operand.
AND	Rewrite the data in the specified area of the remote node buffer to the result of AND operation with that data and the writing operand.
OR	Rewrite the data in the specified area of the remote node buffer to the result of OR operation with that data and the writing operand.

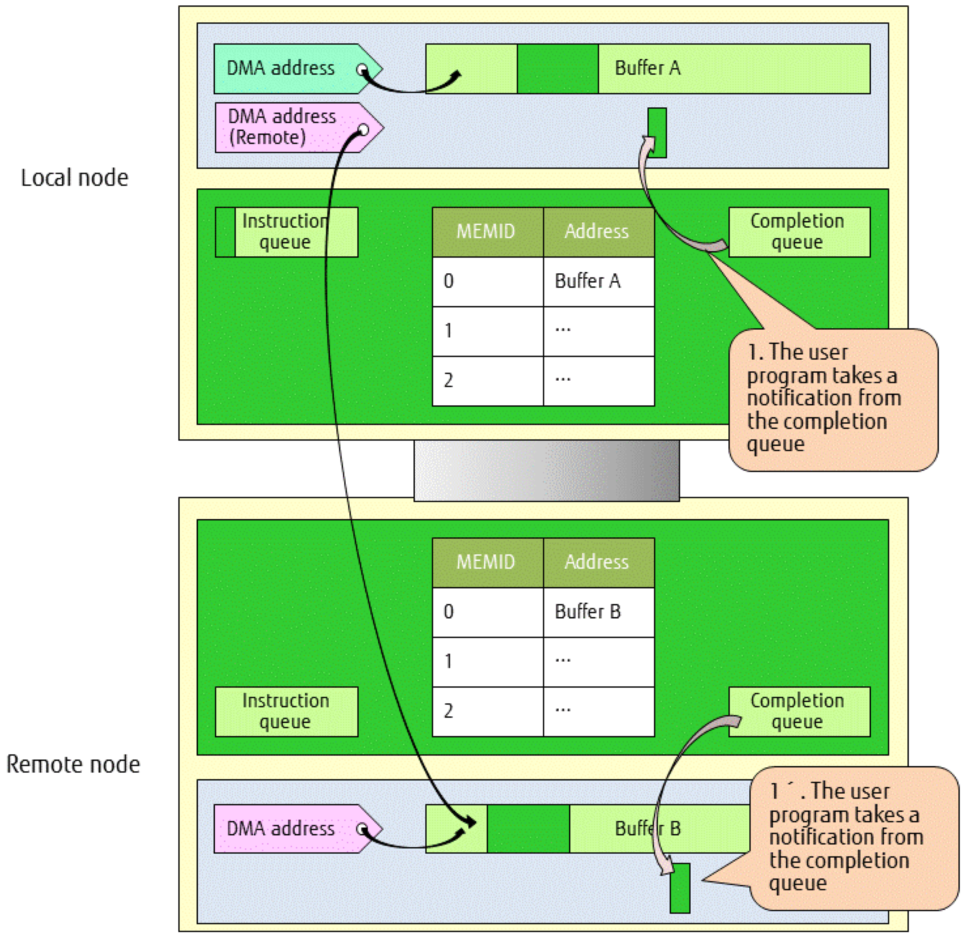
When RDMA ARMW communication is performed, a RDMA ARMW instruction is first stored in the instruction queue. When the RDMA ARMW instruction is executed by a TNI, the specified ARMW operation is performed.

When the ARMW operation to the specified area has been performed, a notification is stored in the completion queue at the remote node when the RDMA ARMW instruction is issued with a flag to notify at the remote node. The remote node sends a RDMA ARMW communication completion notification to the local node. When the local node receives this notification, the notification is stored in the completion queue at the local node. The data which was in the specified area before ARMW was performed is stored in the RDMA ARMW communication completion notification sent from the remote node to the local node. Therefore, the data can be fetched when the RDMA ARMW communication completion is checked.

5.2.1.2.5 Confirmation of Communication Notification

As noted in "5.2.1.2.2 RDMA WRITE", "5.2.1.2.3 RDMA READ" and "5.2.1.2.4 RDMA ARMW", notifications related to communication are progressively added to the completion queue. Therefore, at suitable times, the user program must confirm these completion notifications and take them from the queues ("Figure 5.7 Confirmation of communication notification"). If completion notification is not checked, there is a danger of overflow of the completion queue, so checking is essential. The completion queue has 114,688 entries by default, though overflow might occur at a lower number if the queue is used for both MPI point-to-point communication functions and collective communication functions.

Figure 5.7 Confirmation of communication notification



5.2.1.3 Process Identification

For process identification, the extended RDMA interface uses the rank numbers used by MPI. However, only MPI_COMM_WORLD rank numbers are supported. Operations by dynamically generated processes are not supported.

5.2.1.4 Communication Resource Allocation

The Tofu interconnect has four communication resources (TNIs) and each communication resource has an instruction queue and a completion queue. Through extended RDMA interface, the user can use these resources to communicate in parallel. These communication resources are virtualized by the extended RDMA interface, and users communicate by using macros that specify the virtual NICs used for communication.

The correspondence between communication resources and virtual NICs is 1:1 if there is one node for one process. If there are multiple processes at one node, multiple virtual NICs correspond to one communication resource. Therefore, if multiple processes operate at one node, communication might be performed using the same communication resource even if a different macro is specified.

5.2.1.5 Communication Route Selection

With the exception of some nodes, Tofu interconnect communication enables 12 routes to be selected as the communication route between each node. This can be used for detours when faults occur, for multiple communications, and so on. The extended RDMA interface provides an interface enabling uses to use macros to select these communication routes. Macros can be used to select a maximum of four routes, FJMPI_RDMA_PATH 0 to 3. However, the system determines which of the Tofu interconnect routes are allocated as the routes specified by the macros in a way that the route has lower number of hops in order of FJMPI_RDMA_PATH0, 1, 2, 3. Figure below shows an image of communication route allocation.

Example of RDMA WRITE communication flag specification

```
Put#0: FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_REMOTE_NIC0 | FJMPI_RDMA_PATH0
Put#1: FJMPI_RDMA_LOCAL_NIC1 | FJMPI_RDMA_REMOTE_NIC2 | FJMPI_RDMA_PATH1
Put#2: FJMPI_RDMA_LOCAL_NIC2 | FJMPI_RDMA_REMOTE_NIC3 | FJMPI_RDMA_PATH2
Put#3: FJMPI_RDMA_LOCAL_NIC3 | FJMPI_RDMA_REMOTE_NIC1 | FJMPI_RDMA_PATH3
```

Figure 5.8 Image showing how routes are allocated

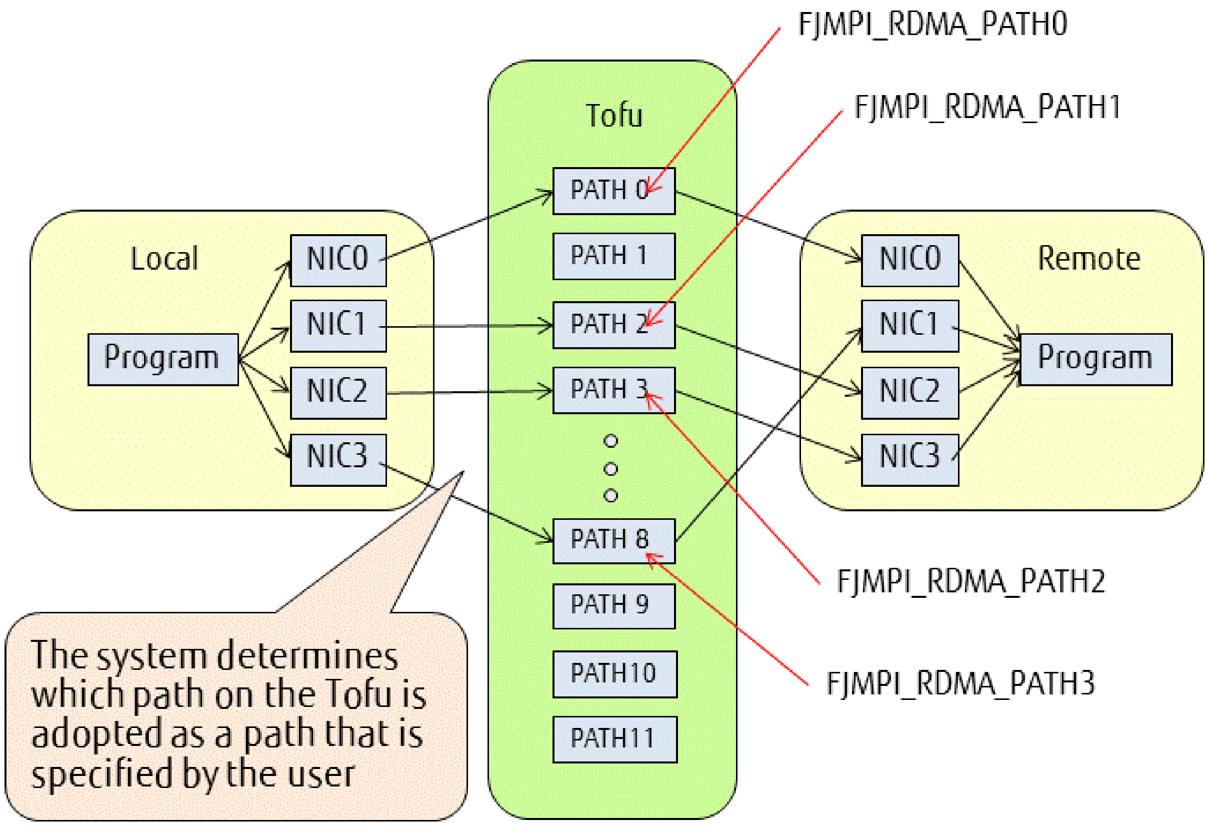
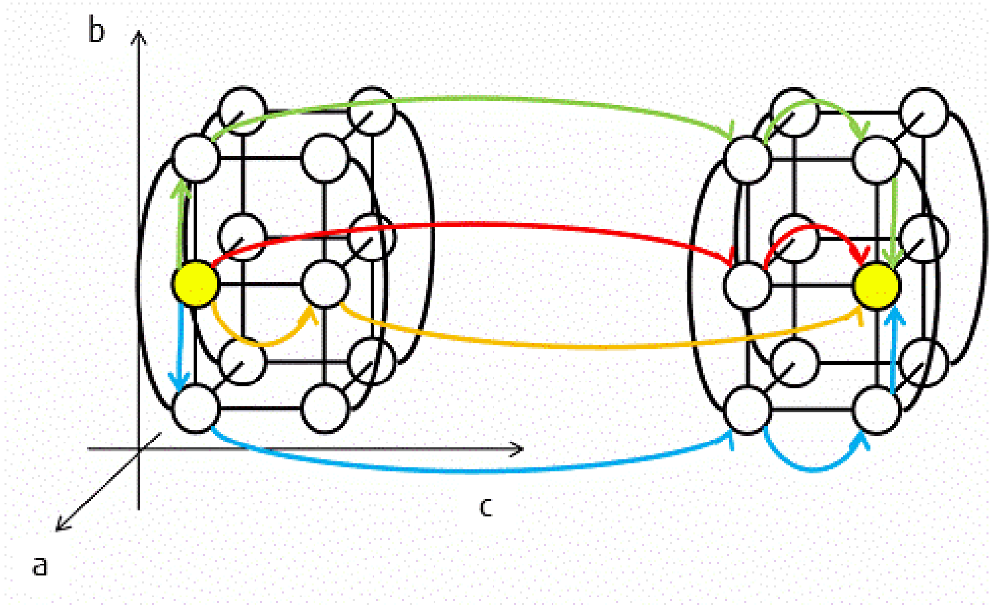


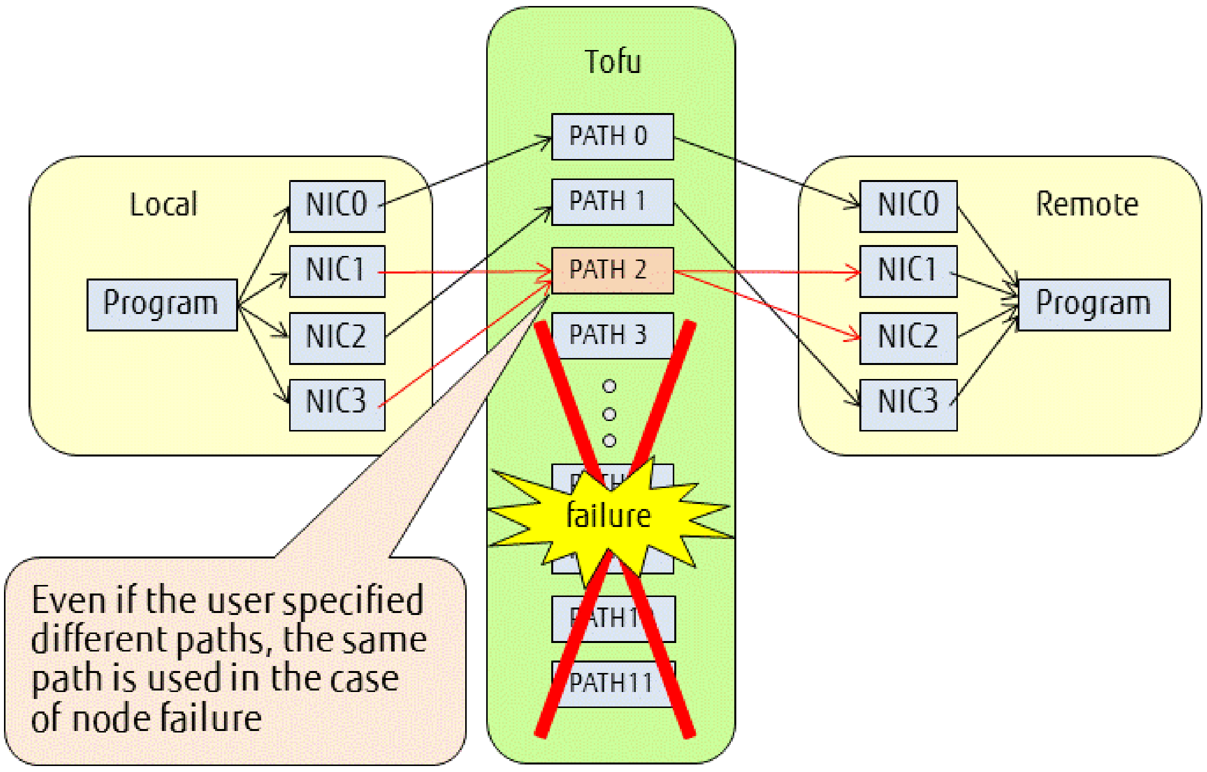
Figure below shows an image of Tofu interconnect physical routes. Since there are four non-overlapping physical routes from the start node to the target node, FJMPI_RDMA_PATH 0 to 3 are allocated to any of these.

Figure 5.9 Tofu allocation image



Sometimes less than four of the four Tofu communication routes can be used due to a fault or due to the relative position of the communication destination. In this case, the same route might be used even though a different macro is specified. Figure below shows an image or overlapping routes.

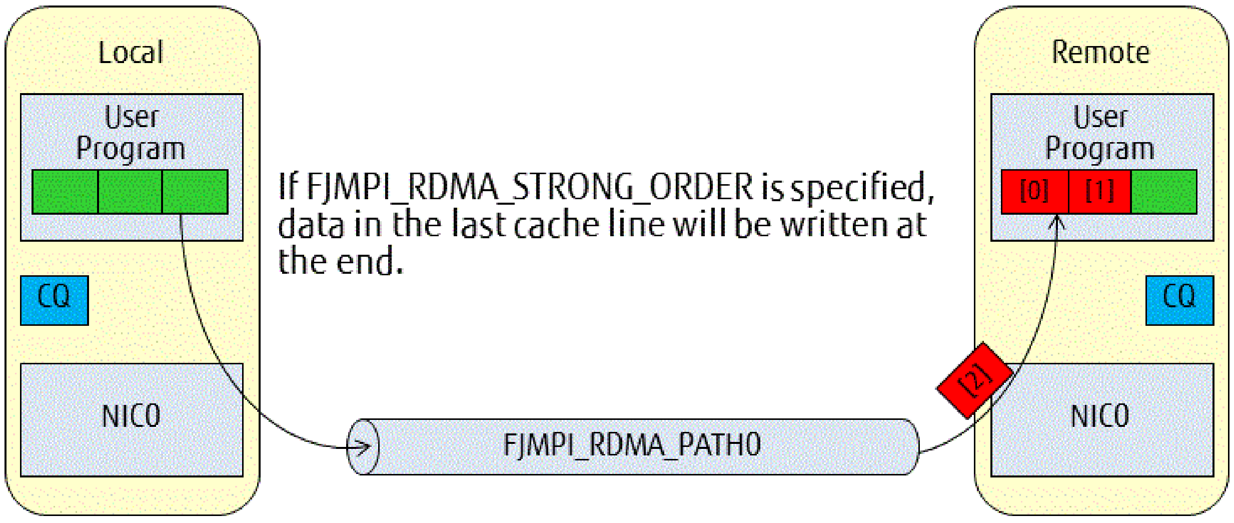
Figure 5.10 Even though a different route is specified, the same route may be used when less than four routes can be used



5.2.1.6 Sequence Guarantee within a Single RDMA

The Tofu interconnect splits into packets the data that is read and written in memory, then sends the data, and the receive side writes it to memory in cache line (128 bytes) units. As a result, sending of the size that fits into a single cache line at the receive side does not break the memory access sequence. On the other hand, the memory access sequence is not guaranteed for sends that span multiple cache lines. However, if the user specified the FJMPI_RDMA_STRONG_ORDER macro when executing the RDMA communication, the interface splits and sends the final cache line of data as a separate RDMA communication. Therefore, it is guaranteed that all data is written to memory at the point when the final cache line of data is written to memory because of sequence guarantee between multiple RDMAs ("5.2.1.7 Sequence Guarantee between Multiple RDMAs"), as shown in "Figure 5.11 Sequence Guarantee within a Single RDMA". As mentioned above, the FJMPI_RDMA_STRONG_ORDER macro is used to guarantee memory write sequence within a single RDMA.

Figure 5.11 Sequence Guarantee within a Single RDMA

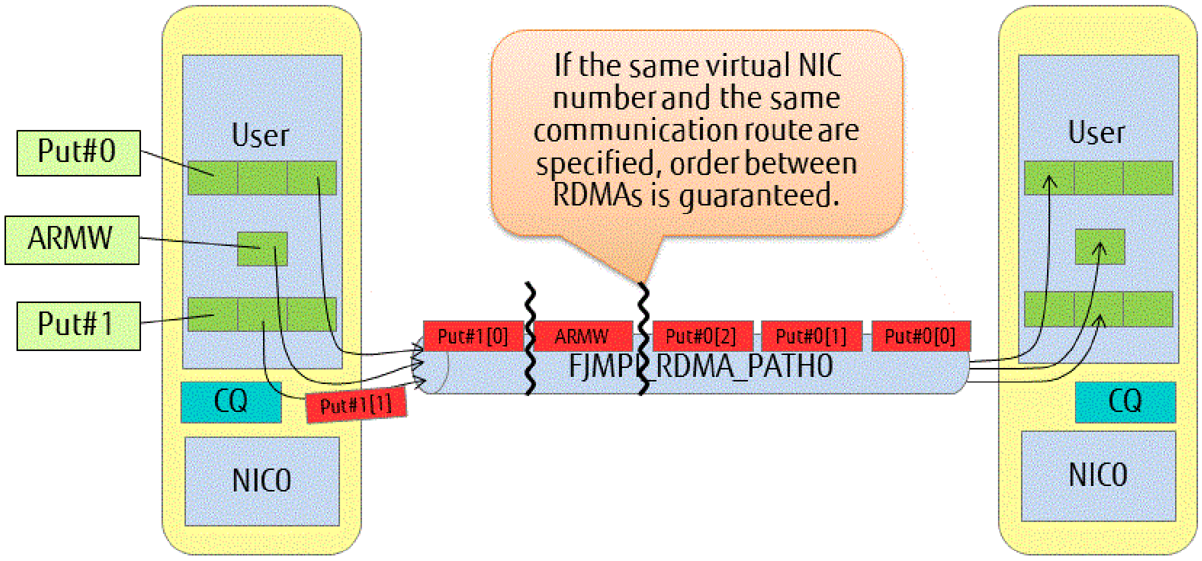

Local
User Program
CQ
NIC0
If FJMPI_RDMA_STRONG_ORDER is specified, data in the last cache line will be written at the end.
FJMPI_RDMA_PATH0
Remote
User Program
[0]
[1]
[2]
CQ
NIC0

5.2.1.7 Sequence Guarantee between Multiple RDMAs

If the same kinds of multiple RDMA communications are issued, the packet sequence between multiple RDMAs is guaranteed, as shown in "Figure 5.12 Sequence Guarantee between Multiple RDMAs", provided that the same virtual NIC number and the same communication route are specified. In other cases, packets from RDMA communications issued later might reach remote node's NIC first. In order to guarantee the sequence for communication between RDMAs when the virtual NIC number and the communication route are not the same, set the program so that the next RDMA communication starts after confirming completion of the previous RDMAs. If the user specified the same virtual NIC number and the same communication route, memory write sequence between multiple RDMAs is guaranteed, since memory write of the following RDMA begins after that of the previous RDMA has completed.

If different kinds of multiple RDMA communications that have the same local node's memory region are issued, in the case of that the first instruction is READ and the following is WRITE, the local node's TNI issues the latter WRITE instruction immediately after issuing READ send request to the remote node without waiting data from the remote node. This may cause invalid data to be written to the buffer in the remote node. Therefore, completion confirmation is required after issuing READ instruction in order to use the data that is written by the READ instruction in the following WRITE instruction.

Figure 5.12 Sequence Guarantee between Multiple RDMAs

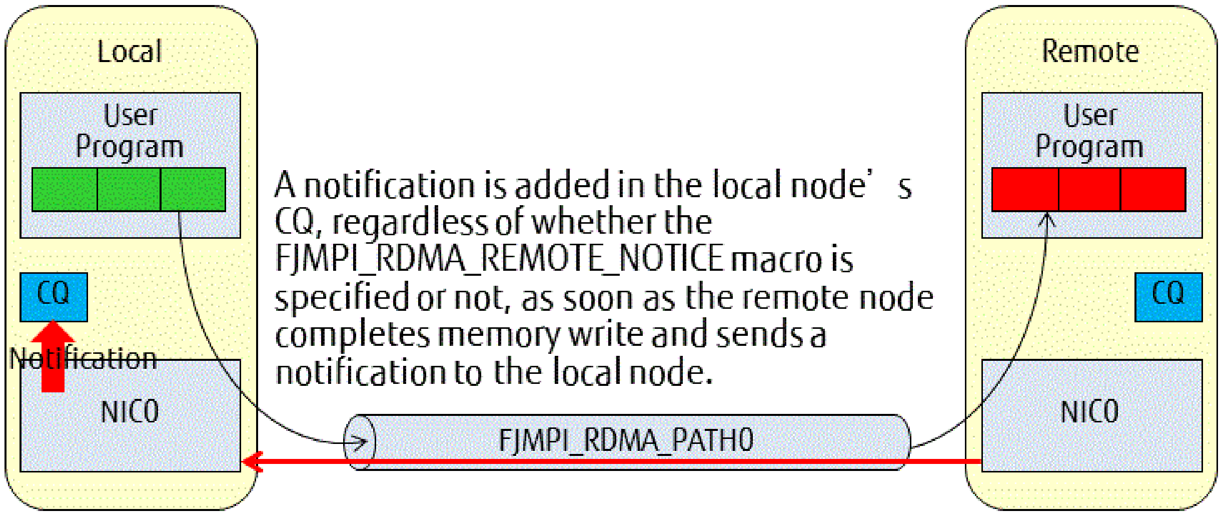

Put#0
ARMW
Put#1
User
CQ
NIC0
If the same virtual NIC number and the same communication route are specified, order between RDMAs is guaranteed.
Put#1[0]
ARMW
Put#0[2]
Put#0[1]
Put#0[0]
Put#1[1]
FJMPI_RDMA_PATH0
User
CQ
NIC0

5.2.1.8 Method for Checking RDMA Communication Completion for RDMA WRITE

To check RDMA WRITE completion at the local node (send side), use the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function to perform polling. At this time the send destination rank number and message tag number can also be fetched. In addition, when the FJMPI_Rdma_poll_cq_ret_data function is used to perform polling, the RDMA communication type can also be fetched.

After writing on the buffer has completed at the remote node, it automatically sends a completion notification to the local node through the Tofu interconnect. When the notification has reached the local node, a completion notification is added to the completion queue at the local node. Therefore, completion of write to the buffer at the remote node is guaranteed when completion confirmation is performed at the local node ("Figure 5.13 RDMA WRITE notification at the local node"). If completion notification is not checked, there is a danger of overflow of the completion queue, so checking is essential.

Figure 5.13 RDMA WRITE notification at the local node

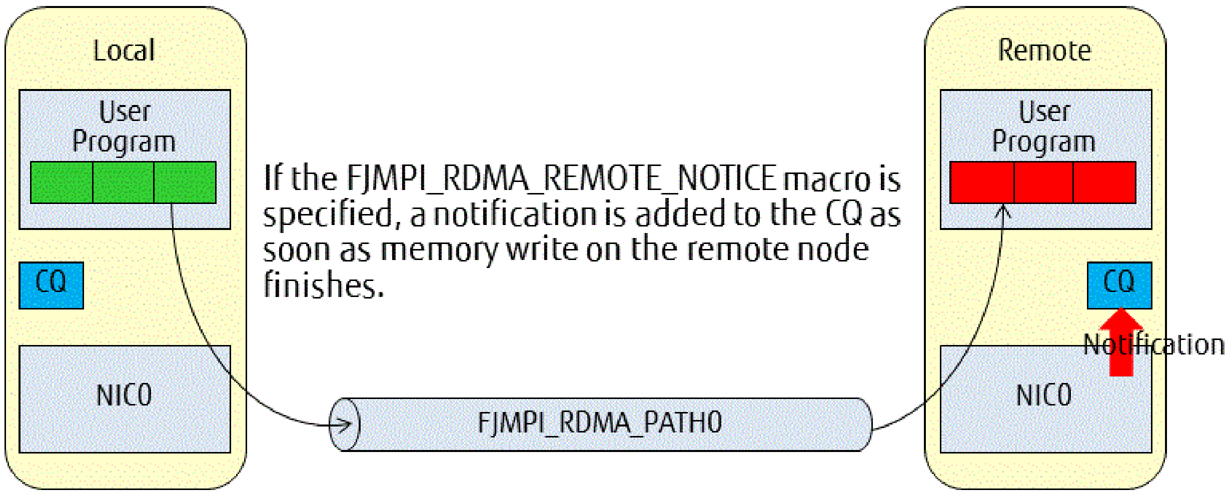

Local
User Program
CQ
Notification
NIC0
A notification is added in the local node' s CQ, regardless of whether the FJMPI_RDMA_REMOTE_NOTICE macro is specified or not, as soon as the remote node completes memory write and sends a notification to the local node.
FJMPI_RDMA_PATH0
Remote
User Program
CQ
NIC0

Either of two methods can be selected as the method for checking RDMA WRITE completion at the remote node (receive side): arrival confirmation by means of data polling, or arrival confirmation by means of completion queue polling. In general, the data polling method has lower overheads.

To perform data polling, the FJMPI_RDMA_STRONG_ORDER macro should be set when RDMA communication is executed. Whether all data has been written to the buffer can be checked by performing data polling for the last four bytes. When data polling is performed, be careful that the user must make the program not to have the same value at the end of the buffer between that of before the RDMA communication and that of after the RDMA communication.

To perform completion queue polling, the FJMPI_RDMA_REMOTE_NOTICE macro should be set when RDMA communication is executed. It can be confirmed that all data has been written to buffer at the remote node by performing polling using the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function. At this time the send source rank number and message tag number can also be fetched ("Figure 5.14 RDMA WRITE notification at the remote node"). In addition, when the FJMPI_Rdma_poll_cq_ret_data function is used to perform polling, the RDMA communication type can also be fetched.

Figure 5.14 RDMA WRITE notification at the remote node


Local
User Program
CQ
NIC0
If the FJMPI_RDMA_REMOTE_NOTICE macro is specified, a notification is added to the CQ as soon as memory write on the remote node finishes.
FJMPI_RDMA_PATH0
Remote
User Program
CQ
Notification
NIC0

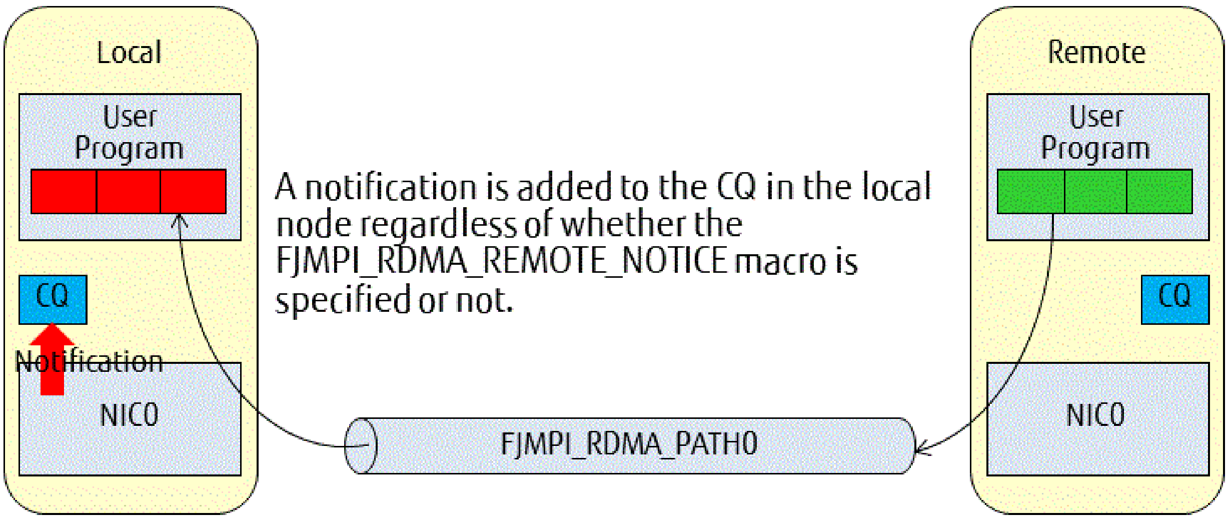
5.2.1.9 Method for Checking RDMA Communication Completion for RDMA READ

Either of two methods can be selected as the method for checking RDMA READ completion at the local node (receive side): arrival confirmation by means of data polling, or arrival confirmation by means of completion queue polling. In general, the data polling method has lower overheads.

As with RDMA WRITE, to perform data polling, the FJMPI_RDMA_STRONG_ORDER macro should be set when RDMA communication is executed. It can be confirmed that all data has been written to the buffer by performing data polling for the last four bytes. However, completion is notified to the local node ("Figure 5.15 RDMA READ notification at the local node") regardless of whether or not the FJMPI_RDMA_REMOTE_NOTICE macro is specified at the time of RDMA READ. Therefore, at any timing after data polling, the completion notification must be removed by calling the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function. When data polling is performed, the program must be such that the data entered at the end of the buffer before RDMA communication does not match the data written at the end of the buffer after RDMA communication.

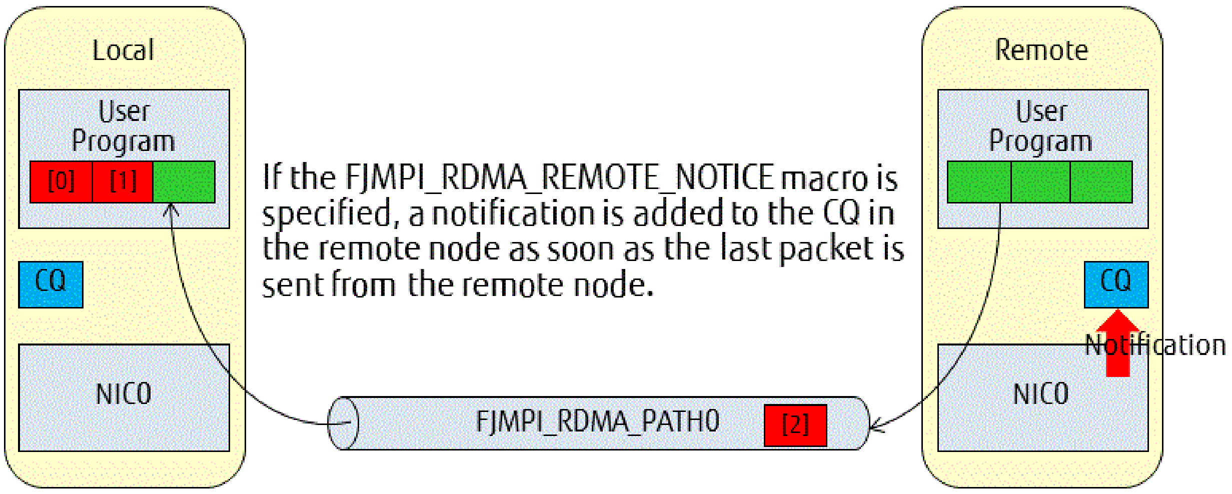
If completion queue polling is performed, it can be confirmed that all data has been written to the buffer by performing polling using the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function. At this time the send source rank number and message tag number can also be fetched. In addition, when the FJMPI_Rdma_poll_cq_ret_data function is used to perform polling, the RDMA communication type can also be fetched.

Figure 5.15 RDMA READ notification at the local node


Local
User Program
CQ
Notification
NIC0
A notification is added to the CQ in the local node regardless of whether the FJMPI_RDMA_REMOTE_NOTICE macro is specified or not.
FJMPI_RDMA_PATH0
Remote
User Program
CQ
NIC0

To check RDMA READ completion at the remote node, perform polling using the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function after executing RDMA communication with the FJMPI_RDMA_REMOTE_NOTICE macro set. At this time the send destination rank number and message tag number can also be fetched. In addition, when the FJMPI_Rdma_poll_cq_ret_data function is used to perform polling, the RDMA communication type can also be fetched. The RDMA READ completion notification at the remote node indicates that the send by the remote node has been completed. Note that this does not guarantee that writing to the buffer at the local node has been completed ("Figure 5.16 RDMA READ notification at the remote node").

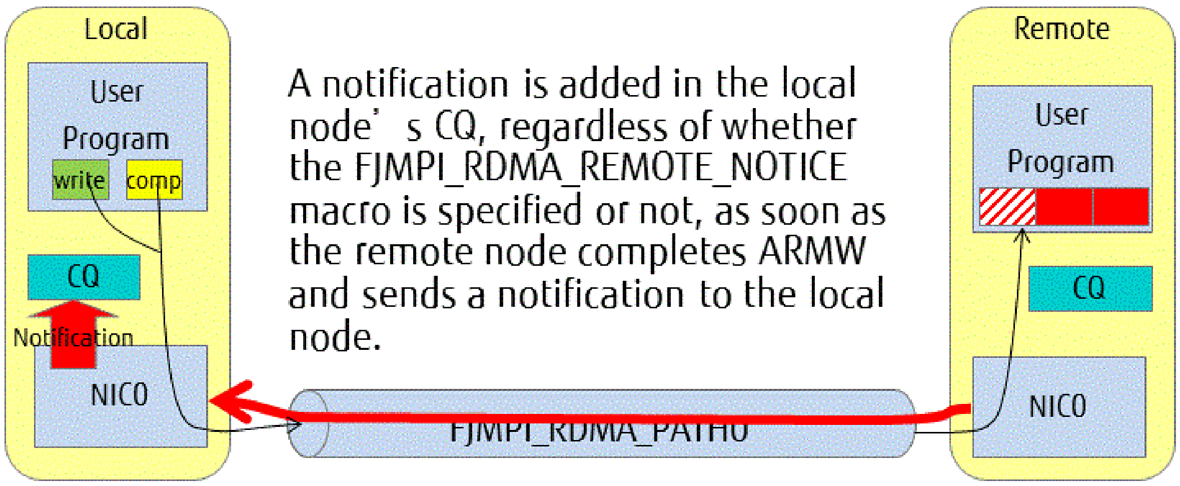
Figure 5.16 RDMA READ notification at the remote node


Local
User Program
[0]
[1]
CQ
NIC0
If the FJMPI_RDMA_REMOTE_NOTICE macro is specified, a notification is added to the CQ in the remote node as soon as the last packet is sent from the remote node.
FJMPI_RDMA_PATH0
[2]
Remote
User Program
CQ
Notification
NIC0

5.2.1.10 Method for Checking RDMA Communication Completion for RDMA ARMW

To check RDMA ARMW completion at the local node (the side that instructs to perform ARMW), use the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data to perform polling. At this time the ARMW target rank number and message tag number can also be fetched. In addition, when FJMPI_Rdma_poll_cq_ret_data is used to perform polling, the RDMA communication type and the data which was in the specified area of the remote node before ARMW can also be fetched. After specified ARMW operation has completed at the remote node, it automatically sends a completion notification to the local node through the Tofu interconnect. When the notification has reached the local node, a completion notification is added to the completion queue at the local node. Therefore, completion of ARMW to the buffer at the remote node is guaranteed when completion confirmation is performed at the local node ("Figure 5.17 RDMA ARMW notification at the local node"). If completion notification is not checked, there is a danger of overflow of the completion queue, so checking is essential.

Figure 5.17 RDMA ARMW notification at the local node

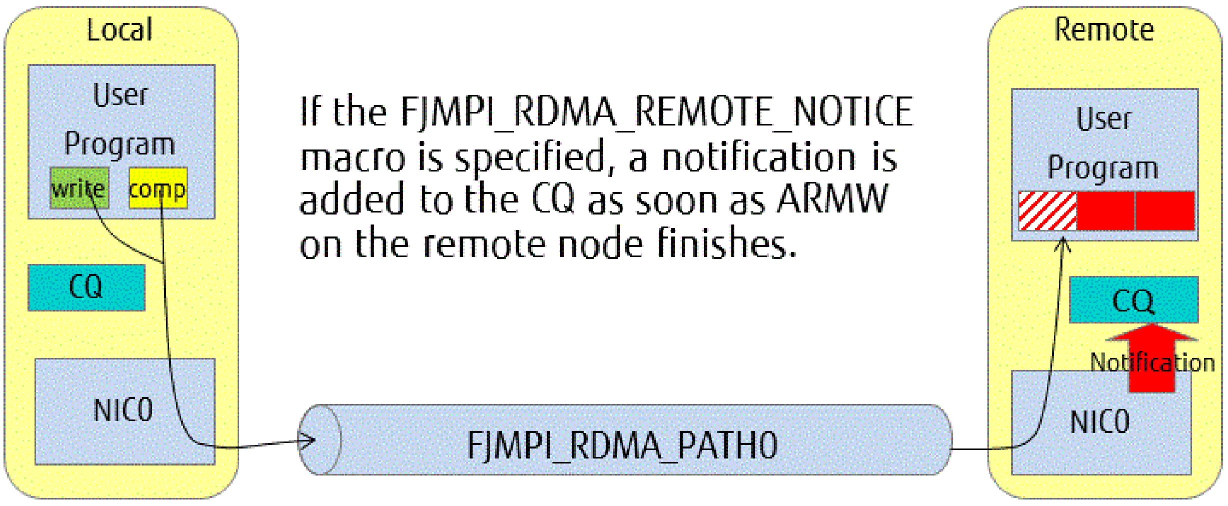

Local
User
Program
write
comp
CQ
Notification
NIC0
A notification is added in the local node's CQ, regardless of whether the FJMPI_RDMA_REMOTE_NOTICE macro is specified or not, as soon as the remote node completes ARMW and sends a notification to the local node.
FJMPI_RDMA_PATH0
Remote
User
Program
CQ
NIC0

Either of two methods can be selected as the method for checking RDMA ARMW completion at the remote node (the side that is instructed to perform ARMW): arrival confirmation by means of data polling, or arrival confirmation by means of completion queue polling. In general, the data polling method has lower overheads.

To perform data polling, whether specified ARMW operation to the buffer has been completed can be checked by performing data polling for the specified data size (4-bytes or 8-bytes). When data polling is performed, be careful that the user must make the program not to have the same value at the specified area of the buffer between that of before the RDMA communication and that of after the RDMA communication. For example, when "Compare and Swap" is specified as an ARMW operation, nothing is done to the specified area of the buffer if its data is equals to the comparison operand. Therefore, in this case, checking RDMA ARMW completion by means of data polling can't be performed.

To perform completion queue polling, the FJMPI_RDMA_REMOTE_NOTICE macro should be set when RDMA communication is executed. It can be confirmed that an ARMW to the specified area in the remote node buffer has been completed by performing polling using the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function. At this time the send source rank number and message tag number can also be fetched ("Figure 5.18 RDMA ARMW notification at the remote node"). In addition, when the FJMPI_Rdma_poll_cq_ret_data function is used to perform polling, the RDMA communication type can also be fetched. But unlike checking RDMA ARMW completion at the local node, the data which was in the specified area of the remote node buffer before ARMW can't be fetched.

Figure 5.18 RDMA ARMW notification at the remote node


Local
User
Program
write
comp
CQ
NIC0
If the FJMPI_RDMA_REMOTE_NOTICE macro is specified, a notification is added to the CQ as soon as ARMW on the remote node finishes.
FJMPI_RDMA_PATH0
Remote
User
Program
CQ
Notification
NIC0

5.2.1.11 RDMA WRITE/RDMA READ/RDMA ARMW Immediate Return

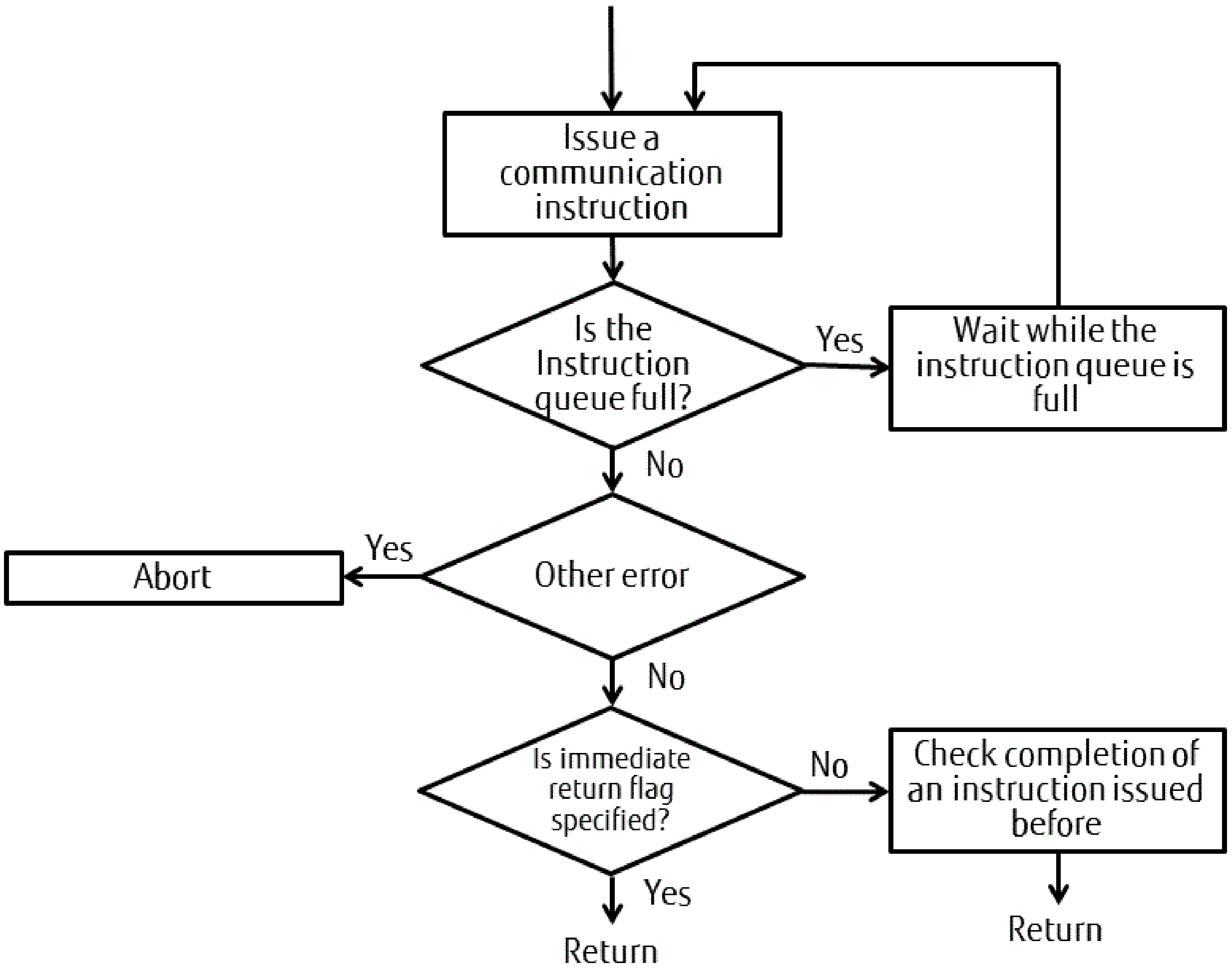
After communication, the FJMPI_Rdma_put function, the FJMPI_Rdma_get function and the FJMPI_Rdma_armw function perform instruction completion confirmation of previously instructed request processing. In some cases, however, it is possible to improve the execution time of the MPI program as a whole by skipping instruction completion confirmation.

Specify FJMPI_RDMA_IMMEDIATE_RETURN in the flags argument in order to skip the instruction completion confirmation in the FJMPI_Rdma_put function, the FJMPI_Rdma_get function and the FJMPI_Rdma_armw function.

The instruction completion confirmation that is omitted by specifying FJMPI_RDMA_IMMEDIATE_RETURN is processed the next time the FJMPI_Rdma_poll_cq function or the FJMPI_Rdma_poll_cq_ret_data function is executed or the next time the FJMPI_Rdma_put function or the FJMPI_Rdma_get function or the FJMPI_Rdma_armw function is executed without FJMPI_RDMA_IMMEDIATE_RETURN being specified. Therefore, the execution time for these functions might be slower.

Waits might occur due to instruction queue shortages even if FJMPI_RDMA_IMMEDIATE_RETURN is specified. Figure below shows waits during RDMA communication and a flow chart of returns. Especially, the potential of exhausting the instruction queue increases on the following conditions because two communication instructions are internally produced per single RDMA communication.

Figure 5.19 Flow when an RDMA communication is issued


Issue a communication instruction
Is the Instruction queue full?
Yes
Wait while the instruction queue is full
No
Yes
Abort
Other error
No
Is immediate return flag specified?
No
Check completion of an instruction issued before
Yes
Return
Return

<FJMPI_Rdma_put>

- The FJMPI_RDMA_STRONG_ORDER flag is specified, and the local node sends to an address that spans multiple cache lines at the remote node.
- 12 to 32 byte communication

<FJMPI_Rdma_get>

- The FJMPI_RDMA_STRONG_ORDER flag is specified, and the remote node sends to an address that spans multiple cache lines at the local node.

<FJMPI_Rdma_armw>

- The FJMPI_RDMA_ARMW_CAS is specified in the op argument, and 8-bytes is specified in the length argument.

5.2.2 Extended RDMA Interface Specifications

5.2.2.1 FJMPI_Rdma_init

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_init()
```

<Explanation>

This function initializes the extended RDMA interface. The program ends abnormally if the following problems are detected within this function:

- Library initialization failed
- Memory allocation failed
- Communication resource acquisition failed

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

<Notes>

Since the program ends abnormally if a problem is detected in the current implementation, no values other than 0 are returned by this function.

5.2.2.2 FJMPI_Rdma_finalize

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_finalize()
```

<Explanation>

This function performs the extended RDMA interface end processing.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

<Notes>

Since problems are not detected in the current implementation, no values other than 0 are returned by this function.

Any extended RDMA interface function that includes FJMPI_Rdma_init cannot be used after FJMPI_Rdma_finalize is called.

5.2.2.3 FJMPI_Rdma_reg_mem

<Format>

```
#include <mpi-ext.h>
uint64_t FJMPI_Rdma_reg_mem(int memid, void *buf, size_t length)
```

<Explanation>

This function registers the memory to be used to perform RDMA communication transfers.

Type	Variable	Explanation	IN/OUT
int	memid	Specify the ID of the memory being registered.	IN
void *	buf	Specify the start address of the memory being registered.	IN
size_t	length	Specify the length of the memory being registered.	IN

The program ends abnormally if the following problems are detected within this function:

- Memory ID is outside range
- Memory ID is already allocated
- Communication resource acquisition failed

<Restrictions>

- The memory ID is 0 to 510.

<Return value>

Normal	DMA address is returned.
Error	FJMPI_RDMA_ERROR is returned.

<Notes>

Since problems are not detected in the current implementation, FJMPI_RDMA_ERROR is not returned by this function.

The memory specified at buf must be an area in user space where physical memory is allocated.

On this system, a DMA address that was registered once can be used unless the user releases it by the FJMPI_Rdma_dereg_mem function.

The FJMPI_Rdma_reg_mem function and the FJMPI_Rdma_get_remote_addr function might return different DMA address values. Therefore, always use the FJMPI_Rdma_get_remote_addr function to fetch remote DMA addresses.

5.2.2.4 FJMPI_Rdma_dereg_mem

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_dereg_mem(int memid)
```

<Explanation>

This function releases the registration of memory that was registered using the FJMPI_Rdma_reg_mem function.

Type	Variable	Explanation	IN/OUT
int	memid	Specify the ID of the memory for which registration is being released.	IN

The program ends abnormally if the following problems are detected by this function internally:

- Memory ID is outside range
- Memory ID is not allocated

<Restrictions>

- The memory ID is 0 to 510.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

<Notes>

Since problems are not detected in the current implementation, no values other than 0 are returned by this function.

5.2.2.5 FJMPI_Rdma_get_remote_addr

<Format>

```
#include <mpi-ext.h>
uint64_t FJMPI_Rdma_get_remote_addr(int pid, int memid)
```

<Explanation>

This function gets the remote DMA address of the memory ID specified from the remote rank pid.

Type	Variable	Explanation	IN/OUT
int	pid	Specify the rank number.	IN
int	memid	Specify the memory ID.	IN

The program ends abnormally if the following problems are detected within this function:

- Memory ID is outside range
- Rank number is outside range

<Restrictions>

- The memory ID is 0 to 510.

<Return value>

Normal	DMA address is returned.
If not registered	FJMPI_RDMA_ERROR is returned.

5.2.2.6 FJMPI_Rdma_put

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_put(int pid, int tag, uint64_t raddr, uint64_t laddr, size_t length, int flags)
```

<Explanation>

This function performs RDMA WRITE from the local node laddr to the raddr of the remote rank pid.

Type	Variable	Explanation	IN/OUT
int	pid	Specify the rank number.	IN
int	tag	Specify the message tag number.	IN
uint64_t	raddr	Specify the remote side node DMA address.	IN
uint64_t	laddr	Specify the local side node DMA address.	IN
size_t	length	Specify the message size.	IN
int	flags	Specify the options.	IN

The options below can be specified using flags. Specify logical inclusive OR.

Macro	Meaning
FJMPI_RDMA_LOCAL_NIC0	Send using send side virtual NIC number 0 (default).
FJMPI_RDMA_LOCAL_NIC1	Send using send side virtual NIC number 1.
FJMPI_RDMA_LOCAL_NIC2	Send using send side virtual NIC number 2.
FJMPI_RDMA_LOCAL_NIC3	Send using send side virtual NIC number 3.
FJMPI_RDMA_REMOTE_NIC0	Send using receive side virtual NIC number 0 (default).
FJMPI_RDMA_REMOTE_NIC1	Send using receive side virtual NIC number 1.
FJMPI_RDMA_REMOTE_NIC2	Send using receive side virtual NIC number 2.
FJMPI_RDMA_REMOTE_NIC3	Send using receive side virtual NIC number 3.
FJMPI_RDMA_PATH0	Use path 0 (default).
FJMPI_RDMA_PATH1	Use path 1.
FJMPI_RDMA_PATH2	Use path 2.
FJMPI_RDMA_PATH3	Use path 3.
FJMPI_RDMA_REMOTE_NOTICE	Set remote notification to ON.
FJMPI_RDMA_STRONG_ORDER	Guarantees that the last four bytes are written last during data write.
FJMPI_RDMA_IMMEDIATE_RETURN	Return from the function immediately. Refer to "5.2.1.11 RDMA WRITE/RDMA READ/RDMA ARMW Immediate Return" for details.

Note that, if an incorrect parameter is specified in this function, the program where the error is detected may end abnormally when the FJMPI_Rdma_poll_cq function, FJMPI_Rdma_poll_cq_ret_data function or an MPI communication function is called.

<Restrictions>

The maximum transmission unit is 16,777,215 (2^{24} - 1) bytes.

The message tag number is 0 to 14.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

5.2.2.7 FJMPI_Rdma_get

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_get(int pid, int tag, uint64_t raddr, uint64_t laddr, size_t length, int flags)
```

<Explanation>

This function performs READ to the local node laddr from the raddr of the remote rank pid.

Type	Variable	Explanation	IN/OUT
int	pid	Specify the rank number.	IN
int	tag	Specify the message tag number.	IN
uint64_t	raddr	Specify the remote side node DMA address.	IN
uint64_t	laddr	Specify the local side node DMA address.	IN

Type	Variable	Explanation	IN/OUT
size_t	length	Specify the message size.	IN
int	flags	Specify the options.	IN

The options below can be specified using flags. Specify logical inclusive OR.

Macro	Meaning
FJMPI_RDMA_LOCAL_NIC0	Receive using the receive side virtual NIC number 0 (default).
FJMPI_RDMA_LOCAL_NIC1	Receive using the receive side virtual NIC number 1.
FJMPI_RDMA_LOCAL_NIC2	Receive using the receive side virtual NIC number 2.
FJMPI_RDMA_LOCAL_NIC3	Receive using the receive side virtual NIC number 3.
FJMPI_RDMA_REMOTE_NIC0	Receive using the send side virtual NIC number 0 (default).
FJMPI_RDMA_REMOTE_NIC1	Receive using the send side virtual NIC number 1.
FJMPI_RDMA_REMOTE_NIC2	Receive using the send side virtual NIC number 2.
FJMPI_RDMA_REMOTE_NIC3	Receive using the send side virtual NIC number 3.
FJMPI_RDMA_PATH0	Use path 0 (default).
FJMPI_RDMA_PATH1	Use path 1.
FJMPI_RDMA_PATH2	Use path 2.
FJMPI_RDMA_PATH3	Use path 3.
FJMPI_RDMA_REMOTE_NOTICE	Set remote notification to ON.
FJMPI_RDMA_STRONG_ORDER	Guarantees that the last four bytes are written last during data write.
FJMPI_RDMA_IMMEDIATE_RETURN	Return from the function immediately. Refer to "5.2.1.11 RDMA WRITE/RDMA READ/RDMA ARMW Immediate Return" for details.

Note that, if an incorrect parameter is specified in this function, the program where the error is detected may end abnormally when the FJMPI_Rdma_poll_cq function, FJMPI_Rdma_poll_cq_ret_data function or an MPI communication function is called.

<Restrictions>

The maximum transmission unit is 16,777,215 (2^{24} - 1) bytes.

The message tag number is 0 to 14.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

5.2.2.8 FJMPI_Rdma_armw

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_armw(int pid, int tag, uint64_t raddr, size_t length, uint64_t write_op,
                    uint64_t comp_op, int operation, int flags)
```

<Explanation>

Execute atomic communication. (Atomic Read Modify Write)

Type	Variable	Explanation	IN/OUT
int	pid	Specify the rank number.	IN
int	tag	Specify the message tag number.	IN
uint64_t	raddr	Specify the remote side node DMA address.	IN
size_t	length	Specify the operand size. Only 4 bytes or 8 bytes can be specified.	IN
uint64_t	write_val	Specify the writing operand. Only 4 bytes or 8 bytes can be specified.	IN
uint64_t	comp_val	Specify the comparison operand. Only 4 bytes or 8 bytes can be specified.	IN
int	op	Specify the operation.	IN
int	flags	Specify the options.	IN

The kind of the operation is selected by specifying the value for the argument op. The value below can be specified. The data of the object area is rewritten according to the content of the operation.

Macro	Kind of operation	Content of operation
FJMPI_RDMA_ARMW_CAS	Compare and Swap	If the data in the specified area of the remote node buffer is equal to the comparison operand, rewrite that data to the writing operand. Otherwise, do nothing.
FJMPI_RDMA_ARMW_SWAP	Swap	Rewrite the data in the specified area of the remote node buffer to the writing operand.
FJMPI_RDMA_ARMW_ADD	Fetch and Add	Rewrite the data in the specified area of the remote node buffer to the result of add operation with that data and the writing operand.
FJMPI_RDMA_ARMW_XOR	XOR	Rewrite the data in the specified area of the remote node buffer to the result of XOR operation with that data and the writing operand.
FJMPI_RDMA_ARMW_AND	AND	Rewrite the data in the specified area of the remote node buffer to the result of AND operation with that data and the writing operand.
FJMPI_RDMA_ARMW_OR	OR	Rewrite the data in the specified area of the remote node buffer to the result of OR operation with that data and the writing operand.

Only when FJMPI_RDMA_ARMW_CAS is specified for the argument op, argument comp_val specification becomes effective.

The options below can be specified using flags. Specify logical inclusive OR.

Macro	Meaning
FJMPI_RDMA_LOCAL_NIC0	Receive using the receive side virtual NIC number 0 (default).
FJMPI_RDMA_LOCAL_NIC1	Receive using the receive side virtual NIC number 1.
FJMPI_RDMA_LOCAL_NIC2	Receive using the receive side virtual NIC number 2.
FJMPI_RDMA_LOCAL_NIC3	Receive using the receive side virtual NIC number 3.
FJMPI_RDMA_REMOTE_NIC0	Receive using the send side virtual NIC number 0 (default).
FJMPI_RDMA_REMOTE_NIC1	Receive using the send side virtual NIC number 1.
FJMPI_RDMA_REMOTE_NIC2	Receive using the send side virtual NIC number 2.
FJMPI_RDMA_REMOTE_NIC3	Receive using the send side virtual NIC number 3.
FJMPI_RDMA_PATH0	Use path 0 (default).
FJMPI_RDMA_PATH1	Use path 1.
FJMPI_RDMA_PATH2	Use path 2.
FJMPI_RDMA_PATH3	Use path 3.
FJMPI_RDMA_REMOTE_NOTICE	Set remote notification to ON.

Macro	Meaning
FJMPI_RDMA_STRONG_ORDER	Guarantee that the memory of the following packet is accessed after the early packet writing in the memory is completed.
FJMPI_RDMA_IMMEDIATE_RETURN	Return from the function immediately. Refer to "5.2.1.11 RDMA WRITE/RDMA READ/RDMA ARMW Immediate Return" for details.

Note that, if an incorrect parameter is specified in this function, the program where the error is detected may end abnormally when the FJMPI_Rdma_poll_cq function, FJMPI_Rdma_poll_cq_ret_data function or an MPI communication function is called.

<Restrictions>

The message tag number is 0 to 14.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.

5.2.2.9 FJMPI_Rdma_poll_cq

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_poll_cq(int nic, struct FJMPI_Rdma_cq *cq)
```

<Explanation>

This function checks the completion of the RDMA communication of the specified NIC number. The send (receive) rank number and message tag information can be obtained by specifying the FJMPI_Rdma_cq structure address at cq. Specify NULL if detailed information is not required.

Type	Variable	Explanation	IN/OUT
int	nic	Specify the virtual NIC number. The following macros can be used: FJMPI_RDMA_NIC0...virtual NIC0 FJMPI_RDMA_NIC1...virtual NIC1 FJMPI_RDMA_NIC2...virtual NIC2 FJMPI_RDMA_NIC3...virtual NIC3	IN

struct FJMPI_Rdma_cq *	cq	Specify the FJMPI_Rdma_cq structure pointer. The format of the FJMPI_Rdma_cq structure is as follows:
		`struct FJMPI_Rdma_cq {`
		`int pid;  /* remote side node rank number */`
		`int tag;  /* message tag number */`
		`};`
			OUT

The program ends abnormally if the following problem is detected within this function:

- Error completion notification received

<Return value>

If there is an RDMA completion notification originating from the local side node	FJMPI_RDMA_NOTICE is returned.
If there is an RDMA completion notification originating from the remote side node	FJMPI_RDMA_HALFWAY_NOTICE is returned.

If Put/Get/ARMW not completed	0 is returned.

<Note>

This completion notification is posted when send and receive are both completed. Therefore, the notification sequence may not be the same as the FJMPI_Rdma_put/FJMPI_Rdma_get/FJMPI_Rdma_armw issue sequence. Remote side node RDMA completion notifications are enabled only if FJMPI_RDMA_REMOTE_NOTICE is specified as FJMPI_Rdma_put/FJMPI_Rdma_get/ FJMPI_Rdma_armw flags.

5.2.2.10 FJMPI_Rdma_poll_cq_ret_data

<Format>

```
#include <mpi-ext.h>
int FJMPI_Rdma_poll_cq_ret_data(int nic, struct FJMPI_Rdma_cq_ret_data *cq_rd)
```

<Explanation>

This function checks the completion of the RDMA communication of the specified NIC number. The send (receive) rank number, message tag information, kind of RDMA communication and specific data that accompanies communication can be obtained by specifying the FJMPI_Rdma_cq_ret_data structure address at cq_rd.

When you do not acquire specific data that accompanies the communication, the completion confirmation can be done even if the FJMPI_Rdma_poll_cq function is used. When you acquire specific data that accompanies the communication, the completion confirmation is executed by using this function.

In that case, the user should manage the communication by either of the following methods.

- The FJMPI_Rdma_poll_cq_ret_data function is called when there is only RDMA completion notification of a communication for data fetch.
- Virtual NIC number different because of the communication and other RDMA communications of the data acquisition object or different tag numbers are specified.

Type	Variable	Explanation	IN/OUT
int	nic	Specify the virtual NIC number. The following macros can be used: FJMPI_RDMA_NIC0...virtual NIC0 FJMPI_RDMA_NIC1...virtual NIC1 FJMPI_RDMA_NIC2...virtual NIC2 FJMPI_RDMA_NIC3...virtual NIC3	IN

struct FJMPI_Rdma_cq_ret_data *	cq_rd	Specify the FJMPI_Rdma_cq_ret_data structure pointer.
		The format of the FJMPI_Rdma_cq_ret_data structure is as follows:
		`struct FJMPI_Rdma_cq {`
		` int pid; /* remote side node rank number */`
		` int tag; /* message tag number */`
		` int rdma_type; /* kind of RDMA communication */`
		` uint64_t ret_data; /* specific datas associated with the communication */`
		`}`
			OUT

Either of the following values is stored in rdma_type.

Kind of RDMA communication	value of rdma_type
Put	FJMPI_RDMA_TYPE_PUT
Get	FJMPI_RDMA_TYPE_GET
ARMW	FJMPI_RDMA_TYPE_ARMW

Either of the following values is stored in ret_data.

Kind of RDMA communication	Value of ret_data
ARMW originating from the local side node	Remote side node area data before ARMW is executed
Put/Get/ARMW originating from the local side node	No change from an initialized value

<Return value>

If there is an RDMA completion notification originating from the local side node	FJMPI_RDMA_NOTICE is returned.
If there is an RDMA completion notification originating from the remote side node	FJMPI_RDMA_HALFWAY_NOTICE is returned.
If Put/Get/ARMW not completed	0 is returned.

5.2.3 Sample Program

Extended RDMA interface sample programs are shown below. The first is a simple program in which two nodes both perform RDMA WRITE to each other. Data polling or completion queue polling can be selected by switching the DATA_POLLING compile option. In the second program, an even numbered node and an odd numbered node pair perform pingpong communication and, if there are an even number of nodes, operation is either one process at one node or multiple processes at one node using any number of parallels. In the third program, an even numbered node and an odd numbered node pair perform RDMA ARMW to each other in pingpong form. The AND operation is specified as a ARMW operation. Checking RDMA ARMW completion at the local node is performed by using the FJMPI_Rdma_poll_ret_data function, and checking RDMA ARMW completion at the remote node is performed by data polling. Like the second program, if there are an even number of nodes, operation is either one process at one node or multiple processes at one node using any number of parallels.

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <mpi-ext.h>

#define BUFSIZE 8192

enum {
    MEMID_S = 0,
    MEMID_R
};

#define INVALID -1
#define DATA_POLLING 0

void communicate(){
    int i;
    int lrank, rrank;
    uint64_t laddr;
    uint64_t raddr;
    /* volatile is required */
    volatile int *sbuf, *rbuf;
    int flag_nic, send_nic, recv_nic;
    struct FJMPI_Rdma_cq cq;

    sbuf = (int*)malloc(sizeof(int)*BUFSIZE);
    rbuf = (int*)malloc(sizeof(int)*BUFSIZE);

    MPI_Comm_rank(MPI_COMM_WORLD, &lrank);

    rrank = 1 - lrank;
    for(i=0; i<BUFSIZE; ++i){
        sbuf[i] = (int)rrank;
        rbuf[i] = INVALID;
    }
    laddr = FJMPI_Rdma_reg_mem(MEMID_S, (void*)sbuf, BUFSIZE*sizeof(int));
    FJMPI_Rdma_reg_mem(MEMID_R, (void*)rbuf, BUFSIZE*sizeof(int));

    /* Waits until the RDMA address of the remote reception buffer can be fetched*/
    while((raddr = FJMPI_Rdma_get_remote_addr(rrank, MEMID_R)) == FJMPI_RDMA_ERROR);

    if(rrank == 0){
        flag_nic = FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_REMOTE_NIC0;
        send_nic = FJMPI_RDMA_NIC0;
        recv_nic = FJMPI_RDMA_NIC1;
    }else{
        flag_nic = FJMPI_RDMA_LOCAL_NIC1 | FJMPI_RDMA_REMOTE_NIC1;
        send_nic = FJMPI_RDMA_NIC1;
        recv_nic = FJMPI_RDMA_NIC0;
    }

#if DATA_POLLING /* Data polling */
    FJMPI_Rdma_put(rrank, 0, raddr, laddr, BUFSIZE*sizeof(int), flag_nic | FJMPI_RDMA_STRONG_ORDER);
#else /* Completion queue polling*/
    FJMPI_Rdma_put(rrank, 0, raddr, laddr, BUFSIZE*sizeof(int), flag_nic | FJMPI_RDMA_REMOTE_NOTICE);
#endif

    while(FJMPI_Rdma_poll_cq(send_nic, &cq) != FJMPI_RDMA_NOTICE);
#if DATA_POLLING /* Data polling */
    while(rbuf[BUFSIZE-1] != lrank);
#else /* Completion queue polling*/
    while(FJMPI_Rdma_poll_cq(recv_nic, &cq) != FJMPI_RDMA_HALFWAY_NOTICE);
#endif
    FJMPI_Rdma_dereg_mem(MEMID_S);
    FJMPI_Rdma_dereg_mem(MEMID_R);
    free((void*)sbuf);
    free((void*)rbuf);
}

int main(int argc, char **argv){
    int size;

    MPI_Init(&argc, &argv);
    FJMPI_Rdma_init();

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    if(size != 2) return 1;

    communicate();

    FJMPI_Rdma_finalize();
    MPI_Finalize();

    return 0;
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <mpi-ext.h>

#define MEMID1 10
#define MEMID2 11
#define BUFSIZE 1024

#define FAILURE 1
```

```
int main(int argc, char *argv[])
{
    int lrank, rrank;
    int i;
    uint64_t laddr1;
    uint64_t raddr2;
    struct FJMPI_Rdma_cq cq;
    volatile int *sbuf = malloc(BUFSIZE*sizeof(int));
    volatile int *rbuf = malloc(BUFSIZE*sizeof(int));

    MPI_Init(&argc, &argv);
    FJMPI_Rdma_init();

    MPI_Comm_rank(MPI_COMM_WORLD,&lrank);
    if((lrank%2)==0){
        rrank=lrank+1;
    }else{
        rrank=lrank-1;
    }

    for(i=0;i<BUFSIZE;++i){
        sbuf[i]=i;
        rbuf[i]=-1;
    }
    laddr1 = FJMPI_Rdma_reg_mem(MEMID1, (void *)sbuf, BUFSIZE*sizeof(int));
    FJMPI_Rdma_reg_mem(MEMID2, (void *)rbuf, BUFSIZE*sizeof(int));
    while((raddr2 = FJMPI_Rdma_get_remote_addr(rrank, MEMID2)) == FJMPI_RDMA_ERROR);
    if((lrank%2)==0){
        for(i=0;i<BUFSIZE;++i){
            if(FJMPI_Rdma_put(rrank, i%14, raddr2+i*sizeof(int), laddr1+i*sizeof(int), sizeof(int),
                          FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_REMOTE_NIC0 | FJMPI_RDMA_STRONG_ORDER)){
                fprintf(stderr, "FJMPI_Rdma_put ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
            while(FJMPI_Rdma_poll_cq(FJMPI_RDMA_NIC0, &cq)!=FJMPI_RDMA_NOTICE);
            if((cq.pid != rrank) || (cq.tag != (i%14))){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
            while(rbuf[i]!=i);
        }
    }else{
        for(i=0;i<BUFSIZE;++i){
            while(rbuf[i]!=i);
            if(FJMPI_Rdma_put(rrank, i%14, raddr2+i*sizeof(int), laddr1+i*sizeof(int), sizeof(int),
                          FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_REMOTE_NIC0 | FJMPI_RDMA_STRONG_ORDER)){
                fprintf(stderr, "FJMPI_Rdma_put ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
            while(FJMPI_Rdma_poll_cq(FJMPI_RDMA_NIC0, &cq)!=FJMPI_RDMA_NOTICE);
            if((cq.pid != rrank) || (cq.tag != (i%14))){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
        }
    }
    FJMPI_Rdma_dereg_mem(MEMID1);
    FJMPI_Rdma_dereg_mem(MEMID2);
    FJMPI_Rdma_finalize();
    MPI_Finalize();
    free((void*)sbuf);
    free((void*)rbuf);
    return 0;
}
```

```
#include <stdlib.h>
#include <stdio.h>
#include <mpi.h>
#include <mpi-ext.h>

#define BUFSIZE 16
#define INVALID 0xaaaaaaaa

#define MEMID_R 0
#define FAILURE 1

void test(size_t length) {
    int i;
    int lrank, rrank;
    int tag = 5;
    uint64_t raddr;
    uint64_t write_val = 0x77777777dddddddd;
    uint64_t comp_val = 0;
    int op = FJMPI_RDMA_ARMW_AND;
    volatile uint32_t *rbuf;
    uint64_t *rbuf64;
    struct FJMPI_Rdma_cq_ret_data cq_rd;

    rbuf = (uint32_t *)malloc(BUFSIZE);
    rbuf64 = (uint64_t *)rbuf;

    MPI_Comm_rank(MPI_COMM_WORLD, &lrank);

    if(lrank % 2 == 0){
        rrank = lrank + 1;
    }else{
        rrank = lrank - 1;
    }

    for(i=0; i<(BUFSIZE/4); ++i){
        rbuf[i] = INVALID;
    }

    FJMPI_Rdma_reg_mem(MEMID_R, (void*)rbuf, BUFSIZE);
    while((raddr = FJMPI_Rdma_get_remote_addr(rrank, MEMID_R)) == FJMPI_RDMA_ERROR);

    MPI_Barrier(MPI_COMM_WORLD);

    if(lrank % 2 == 0){
        FJMPI_Rdma_armw(rrank, tag, raddr, length, write_val, comp_val, op,
                        FJMPI_RDMA_LOCAL_NIC0 | FJMPI_RDMA_STRONG_ORDER);
        while(FJMPI_Rdma_poll_cq_ret_data(FJMPI_RDMA_NIC0, &cq_rd) != FJMPI_RDMA_NOTICE);
        if((cq_rd.pid != rrank) || (cq_rd.tag != tag) || (cq_rd.rdma_type != FJMPI_RDMA_TYPE_ARMW)){
            fprintf(stderr, "CQ ERROR\n");
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
        if(length == 4){
            /* Check the data which was in the specified area of the remote node before ARMW */
            if(cq_rd.ret_data != (uint32_t)INVALID){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
            /* Data polling */
            while(rbuf[0] != ((uint32_t)INVALID & (uint32_t)write_val));
        }else{/* length == 8 */
```

```
          /* Check the data which was in the specified area of the remote node before ARMW */
            if(cq_rd.ret_data != ((uint64_t)INVALID << 32) + (uint64_t)INVALID){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
            /* Data polling */
            while(rbuf64[0] != ((((uint64_t)INVALID << 32) + (uint64_t)INVALID) & write_val));
        }
    }else{
        if(length == 4){
            while(rbuf[0] != ((uint32_t)INVALID & (uint32_t)write_val));
        }else{
            while(rbuf64[0] != ((((uint64_t)INVALID << 32) + (uint64_t)INVALID) & write_val));
        }
        FJMPI_Rdma_armw(rrank, tag, raddr, length, write_val, comp_val, op,
                        FJMPI_RDMA_LOCAL_NIC1 | FJMPI_RDMA_REMOTE_NOTICE);
        while(FJMPI_Rdma_poll_cq_ret_data(FJMPI_RDMA_NIC1, &cq_rd) != FJMPI_RDMA_NOTICE);
        if((cq_rd.pid != rrank) || (cq_rd.tag != tag) || (cq_rd.rdma_type != FJMPI_RDMA_TYPE_ARMW)){
            fprintf(stderr, "CQ ERROR\n");
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
        if(length == 4){
            if(cq_rd.ret_data != (uint32_t)INVALID){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
        }else{
            if(cq_rd.ret_data != ((uint64_t)INVALID << 32) + (uint64_t)INVALID){
                fprintf(stderr, "CQ ERROR\n");
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
            }
        }
    }

    FJMPI_Rdma_dereg_mem(MEMID_R);
    free((void*)rbuf);
}

int main(int argc, char **argv){
    MPI_Init(&argc, &argv);
    FJMPI_Rdma_init();

    test(4);
    MPI_Barrier(MPI_COMM_WORLD);
    test(8);

    FJMPI_Rdma_finalize();
    MPI_Finalize();

    return 0;
}
```

5.3 MPI Statistical Information Section Specifying Interface

The MPI statistical information section specifying function is a function to specify the range to measure the communication data of MPI. To specify the analyzing range on the source code, please insert the functions in the measurement beginning and end position. These functions are meaningful only if a value of MCA parameter mpi_print_stats is equal to 3 or 4.

Table below shows the overview of section specifying function of MPI statistical information.

Table 5.4 Overview of MPI statistical information section specifying functions list

Function name	Function overview
FJMPI_Collection_start	Starts MPI statistical information measurement
FJMPI_Collection_stop	Stops MPI statistical information measurement
FJMPI_Collection_print	Prints MPI statistical information measurement
FJMPI_Collection_clear	Initializes MPI statistical information

5.3.1 The MPI Statistical Information Section Specifying Function

5.3.1.1 FJMPI_Collection_start

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Collection_start()
```

Fortran format

```
CALL FJMPI_COLLECTION_START()
```

<Explanation>

This function starts measuring MPI statistics. This function is meaningful only if a value of MCA parameter mpi_print_stats is equal to 3 or 4. This function is ignored if any of the conditions are met below.

- When this subroutine is called before MPI_Init.
- When this subroutine is called after MPI_Finalize.

<Return value>

None.

<Notes>

When this function is continuously called, the first start instruction becomes effective.

5.3.1.2 FJMPI_Collection_stop

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Collection_stop()
```

Fortran format

```
CALL FJMPI_COLLECTION_STOP()
```

<Explanation>

This function stops measuring MPI statistics. This function is meaningful only if a value of MCA parameter mpi_print_stats is equal to 3 or 4. This function is ignored if any of the conditions below are met.

- When this subroutine is called before MPI_Init.
- When this subroutine is called after MPI_Finalize.

<Return value>

None.

<Notes>

It accumulates the collection result when the data collection is repeated.

When this function is continuously called, the first stop instruction becomes effective.

5.3.1.3 FJMPI_Collection_print

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Collection_print (char *str)
```

Fortran format

```
CALL FJMPI_COLLECTION_PRINT(STR)
CHARACTER*(*) STR
```

<Explanation>

Type	Variable	Explanation	IN/OUT
char*	str	Character string to be output each section to standard error output.	IN

str is a string of up to 30 alphanumeric characters for distinguishing statistical information. The characters over the 30th character of str are dropped.

This function prints measuring MPI statistical information to standard error output. This function is meaningful only if a value of MCA parameter mpi_print_stats is equal to 3 or 4. This function is ignored if any of the conditions are met below.

- When this subroutine is called before MPI_Init.
- When this subroutine is called after MPI_Finalize.

This function changes depending on the value of MCA parameter mpi_print_stats.

mpi_print_stats	Operation	Explanation
3	The collective operation	This function must be executed by all processes in MPI_COMM_WORLD.
4	The independent operation	This function can be executed independently of other processes.

<Return value>

None.

<Notes>

Please specify a string of printable single-byte characters.

MPI statistical information is printed only if the program executed this function.

5.3.1.4 FJMPI_Collection_clear

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Collection_clear ()
```

Fortran format

```
CALL FJMPI_COLLECTION_CLEAR()
```

<Explanation>

This function clears all MPI statistical data. This function is meaningful only a value of MCA parameter mpi_print_stats is equal to 3 or 4. This function is ignored if any of the conditions are met below.

- When this subroutine is called before MPI_Init.
- When this subroutine is called after MPI_Finalize.

<Return value>

None.

<Notes>

This function clears all statistical data including collected data and elapsed time.

5.3.2 Sample Program

The sample program of the MPI statistical information section specifying interface is shown below. This program measures the MPI statistical information that is section specifying of the MPI_Alltoall and section specifying of the user function including MPI_Allgather.

```
#include <mpi.h>
#include <stdio.h>
#include <stdlib.h>
#include <mpi-ext.h>

int test_aa = 200;
int test_ag = 300;

void func_aa(int,int,MPI_Comm,int);
void func_ag(int,int,MPI_Comm,int);

int main(int argc, char *argv[])
{
    int size,rank,loop;
    MPI_Comm comm=MPI_COMM_WORLD;

    MPI_Init( &argc, &argv);
    MPI_Comm_size(comm, &size);
    MPI_Comm_rank(comm, &rank);

    loop=20;
    func_aa(rank,size,comm,loop);
    FJMPI_Collection_print("alltoall");

    FJMPI_Collection_clear();

    FJMPI_Collection_start();
    loop=40;
    func_ag(rank,size,comm,loop);
    FJMPI_Collection_print("func_ag");

    MPI_Finalize();

    return 0;
}
void func_aa(int rank,int size,MPI_Comm comm, int comm_count)
{
    int i,*sendbuf,*recvbuf;

    sendbuf = (int*)malloc(sizeof(int) * test_aa * size);
    recvbuf = (int*)malloc(sizeof(int) * test_aa * size);

    for(i = 0; i < test_aa * size; i++) {
        sendbuf[i] = rank;
        recvbuf[i] = -1;
    }

    FJMPI_Collection_start();
    for(i = 0; i < comm_count ; i++) {
        MPI_Alltoall( sendbuf, test_aa, MPI_INT,
                      recvbuf, test_aa, MPI_INT, comm);
    }
    FJMPI_Collection_stop();

    free(sendbuf);
    free(recvbuf);
    MPI_Barrier(comm);
}
void func_ag(int rank,int size,MPI_Comm comm, int comm_count)
{
    int *sendbuf,*recvbuf;
    int i;

    sendbuf = (int*)malloc(sizeof(int) * test_ag * size);
    recvbuf = (int*)malloc(sizeof(int) * test_ag * size);

    for(i = 0; i < test_ag * size; i++) {
        sendbuf[i] = rank;
        recvbuf[i] = -1;
    }

    for(i = 0; i < comm_count ; i++) {
        MPI_Allgather(sendbuf, test_ag, MPI_INT,
                      recvbuf, test_ag, MPI_INT, comm);
    }

    free(sendbuf);
    free(recvbuf);
    MPI_Barrier(comm);
}
```

The sample program of the section specifying MPI statistical information interface is shown. This program is section specifying of MPI_Bcast, a program that is section specifying of MPI_Allreduce, and either one node one process or a process two or more one node operates by a parallel number of arbitrary.

```
program main
use mpi
implicit none
integer i,ierr,myrank,size
real(8) buf1(100),buf2(100)

call MPI_INIT(ierr)
call MPI_COMM_RANK(MPI_COMM_WORLD,myrank,ierr)
call MPI_COMM_SIZE(MPI_COMM_WORLD,size,ierr)
buf2 = -1
do i=1,100
  buf1(i) = (size - myrank) * 0.001
end do

call FJMPI_COLLECTION_START()
do i=1,50
  call MPI_BCAST(buf1,100,MPI_REAL8,0,MPI_COMM_WORLD,ierr)
end do
call FJMPI_COLLECTION_STOP()
call FJMPI_COLLECTION_PRINT('Bcast')
call MPI_BARRIER(MPI_COMM_WORLD,ierr)

call FJMPI_COLLECTION_CLEAR()
call FJMPI_COLLECTION_START()
do i=1,100
  call MPI_ALLREDUCE(buf1,buf2,100,MPI_REAL8,MPI_SUM,MPI_COMM_WORLD,ierr)
end do
call FJMPI_COLLECTION_PRINT('Allreduce')

call MPI_BARRIER(MPI_COMM_WORLD,ierr)
call MPI_FINALIZE(ierr)
end program main
```

5.4 Extended Persistent Communication Requests Interface

Using this interface, overlap of computation and communication, which could not be achieved completely by using only the persistent communication request interface in the MPI standard, can be achieved by starting communication asynchronously.

See table below for a list of the practical functions for the extended persistent communication requests interface supported by this system.

Table 5.5 Extended persistent communication requests interface function list

Function name	Function overview
FJMPI_Prequest_send_init	Initialization of send using an extended persistent communication requests interface
FJMPI_Prequest_recv_init	Initialization of receive using an extended persistent communication requests interface
FJMPI_Prequest_start	Starts communication using an extended persistent communication requests interface
FJMPI_Prequest_startall	Starts all communication using persistent an extended persistent communication requests interface

5.4.1 Overview

The extended persistent communication requests interface overlaps computation and communication by the following mechanism.

Table 5.6 The extended persistent communication requests interface mechanism

`FJMPI_Prequest_send_init...`
`  1. Notify the partner process of information`
 	
`FJMPI_Prequest_start`
`  2. Begin the send of message body with start`
 	
`(computation)`
 	
`MPI_Wait`
	`FJMPI_Prequest_recv_init...`
	`  1. Notify the partner process of information`
	
	`FJMPI_Preqeust_start`
	`  2. Begin the recieve of message body with start`
	
	`(computation)`
	
	`MPI_Wait`

Note that the communication performance may degrade compared to the MPI function if a derived datatype which represents non-contiguous message on memory is specified.

5.4.2 Extended Persistent Communication Requests Interface Specifications

5.4.2.1 FJMPI_Prequest_send_init

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Prequest_send_init(void *buf, int count, MPI_Datatype datatype,
                             int dest, int tag, MPI_Comm comm,
                             MPI_Request *request)
```

Fortran format

```
FJMPI_PREQUEST_SEND_INIT(BUF, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
<type>     BUF(*)
INTEGER    COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR
```

<Explanation>

The arguments are same as those of the MPI_Send_init function.

The request generated with this function can be used with the request operation function which is defined in the MPI standard.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.
If the job type is node-sharing job	A value other than 0 is returned. Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

- The request generated with this function can be initiated only by the FJMPI_Prequest_start function or the FJMPI_Prequest_startall function.
- The communication using the request generated with this function must be received using the request generated with FJMPI_Prequest_recv_init function.
- It is not possible to cancel using the MPI_Cancel function. In that case, the MPI_Cancel function returns an error.
- Communication requests that have same source/destination rank, message tag, and communicator cannot exist at the same time. The following message is output if another communication request that was created with the same arguments already exists on calling this function.

```
[mpi::fjmpi-prequest::same-request-args] The arguments of source/destination rank, message tag,
and communicator for the request are identical to those of another request.
```

5.4.2.2 FJMPI_Prequest_recv_init

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Prequest_recv_init(void *buf, int count, MPI_Datatype datatype,
                             int source, int tag, MPI_Comm comm,
                             MPI_Request *request)
```

Fortran format

```
FJMPI_PREQUEST_RECV_INIT(BUF, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)
<type>     BUF(*)
INTEGER    COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR
```

<Explanation>

The arguments are same as those of the MPI_Recv_init function.

The request generated with this function can be used with the request operation function which is defined in the MPI standard.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.
If the job type is node-sharing job	A value other than 0 is returned. Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

- MPI_ANY_SOURCE cannot be specified for the argument source.
- The communication using the request generated with this function must be received using the request generated with FJMPI_Prequest_send_init function.
- The request generated with this function can be initiated only by the FJMPI_Prequest_start function or the FJMPI_Prequest_startall function.
- It is not possible to cancel using the MPI_Cancel function. In that case, the MPI_Cancel function returns an error.
- The probe functions such as MPI_Probe cannot check for incoming messages which are bound to requests created by this function.
- Communication requests that have same source/destination rank, message tag, and communicator cannot exist at the same time. The following message is output if another communication request that was created with the same arguments already exists on calling this function.

```
[mpi::fjmpi-prequest::same-request-args] The arguments of source/destination rank, message tag,
and communicator for the request are identical to those of another request.
```

5.4.2.3 FJMPI_Prequest_start

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Prequest_start(MPI_Request *request)
```

Fortran format

```
FJMPI_PREQUEST_START(REQUEST, IERROR)
INTEGER    REQUEST, IERROR
```

<Explanation>

The arguments are same as those of the MPI_Start function.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.
If the job type is node-sharing job	A value other than 0 is returned. Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

- Only communication requests that were created by FJMPI_Prequest_send_init function or FJMPI_Prequest_recv_init function can be passed to this function.
- When the request generated with a FJMPI_Prequest_send_init function is used, the procedure is non-local and waits calling FJMPI_Prequest_start or FJMPI_Prequest_startall function by the process which executes receive function.

5.4.2.4 FJMPI_Prequest_startall

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Prequest_startall(int count, MPI_Request *requests)
```

Fortran format

```
FJMPI_PREQUEST_STARTALL(COUNT, ARRAY_OF_REQUESTS, IERROR)
INTEGER    COUNT, ARRAY_OF_REQUESTS(*), IERROR
```

<Explanation>

The arguments are same as those of the MPI_Startall function.

<Return value>

Normal	0 is returned.
Error	A value other than 0 is returned.
If the job type is node-sharing job	A value other than 0 is returned. Refer to the Job Operation Software manual for information on node-sharing job.

<Notes>

- Only communication requests that were created by FJMPI_Prequest_send_init function or FJMPI_Prequest_recv_init function can be passed to this function.
- When the request generated with a FJMPI_Prequest_send_init function is included, the procedure of the request is non-local and wait calling FJMPI_Prequest_start or FJMPI_Prequest_startall function by the process which executes receive function.

5.4.3 Sample Program

The sample program of the extended persistent communication requests interface is shown below.

```
#include <stdio.h>
#include <mpi.h>
#include <mpi-ext.h>

#define VEC_LEN  (4*1024*1024)
#define BSIZE    (1024*1024)

static double x[VEC_LEN], y[VEC_LEN], a = 0.1;

int main(int argc, char *argv[])
{
    int j, rank, size, prev, next;
    double sb[BSIZE], rb[BSIZE];
    MPI_Request reqs[2];
    double stime, etime;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&size);
    MPI_Comm_rank(MPI_COMM_WORLD,&rank);

    prev = (rank - 1 + size) % size;
    next = (rank + 1) % size;

    MPI_Barrier(MPI_COMM_WORLD);

    FJMPI_Prequest_recv_init(rb,BSIZE,MPI_DOUBLE,prev,1000,MPI_COMM_WORLD,&reqs[0]);
    FJMPI_Prequest_send_init(sb,BSIZE,MPI_DOUBLE,next,1000,MPI_COMM_WORLD,&reqs[1]);

    MPI_Barrier(MPI_COMM_WORLD);
    stime = MPI_Wtime();

    FJMPI_Prequest_startall(2,&reqs[0]);

    for (j = 0; j < VEC_LEN; j++) {
        y[j] = a * x[j] + y[j];
    }

    MPI_Waitall(2,&reqs[0],MPI_STATUSES_IGNORE);

    etime = MPI_Wtime();

    MPI_Barrier(MPI_COMM_WORLD);

    if (0 == rank) {
        printf("%.6f sec\n", etime - stime);
    }

    MPI_Finalize();

    return (0);
}
```

5.5 MPI Asynchronous Communication Promotion Section Specifying Interface

The MPI asynchronous communication promotion section specifying function is a function to specify the range to promote asynchronous communication using an assistant core. Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details of the asynchronous communication promotion.

These functions are meaningful only if a value of MCA parameter opal_progress_thread_mode is equal to 1 or 2.

Table below shows the overview of asynchronous communication promotion section specifying function.

Table 5.7 Overview of asynchronous communication promotion section specifying functions list

Function name	Function overview
FJMPI_Progress_start	Starts the promotion of asynchronous communication
FJMPI_Progress_stop	Stops the promotion of asynchronous communication

5.5.1 The MPI Asynchronous Communication Promotion Section Specifying Function

5.5.1.1 FJMPI_Progress_start

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Progress_start(void);
```

Fortran format

```
CALL FJMPI_PROGRESS_START()
```

<Explanation>

This function starts the promotion of asynchronous communication on the manual section (without MPI call) mode or the manual section (with MPI call) mode of the MPI asynchronous processing progress thread.

On the manual section (without MPI call) mode, any MPI functions or extended interfaces cannot be called before FJMPI_Progress_stop function is called. Function calls in the section are not checked and the behavior on the calls is uncertain. On the manual section (with MPI call) mode, those functions can be called but they involves a performance overhead.

This function also can be called when one or more active requests exist. Also, this function can be called when no active request exists. The latter case causes almost no performance effect.

Call of this function is ignored in any of the following cases.

- The operation mode is not the manual section (without MPI call) mode nor the manual section (with MPI call) mode.
- The promotion of asynchronous communication was started already. That is, FJMPI_Progress_start function was already called and FJMPI_Progress_stop function is not called yet.

<Return value>

None.

5.5.1.2 FJMPI_Progress_stop

<Format>

C language format

```
#include <mpi-ext.h>
void FJMPI_Progress_stop(void);
```

Fortran format

```
CALL FJMPI_PROGRESS_STOP()
```

<Explanation>

This function stops the promotion of asynchronous communication on the manual section (without MPI call) mode or the manual section (with MPI call) mode of the MPI asynchronous processing progress thread.

Call of this function is ignored in any of the following cases.

- The operation mode is not the manual section (without MPI call) mode nor the manual section (with MPI call) mode
- The promotion of asynchronous communication is not started yet. That is, FJMPI_Progress_start function has never called, or FJMPI_Progress_start function was called and corresponding FJMPI_Progress_stop function was also already called

<Return value>

None.

5.5.2 Sample Program

A sample program of a MPI asynchronous communication promotion section specifying interface is shown below. This program performs computation and four communications concurrently.

You can see an execution time difference between the case of value 0 and 1 that is specified for the MCA parameter opal_progress_thread_mode. But if the difference of the execution time of computation only and the execution time of communication only is large, the effect of the overlap of computation and communication becomes small. So the degree of the effect varies greatly depend on the compiler options on the compilation, the process layout on the execution, and so on.

```
#include <stdio.h>
#include <mpi.h>
#include <mpi-ext.h>

#define VEC_LEN  (4*1024*1024)
#define MSG_LEN (16*1024*1024)

static double x[VEC_LEN], y[VEC_LEN], a = 0.1;
static double sbuf[2][MSG_LEN], rbuf[2][MSG_LEN];

int main(int argc, char *argv[])
{
    int i, j, size, rank, prev, next;
    MPI_Request reqs[4];
    double stime, etime;

    MPI_Init(&argc, &argv);
    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    prev = (rank - 1 + size) % size;
    next = (rank + 1) % size;

    for (i = 0; i < 2; i++) {

        stime = MPI_Wtime();

        MPI_Irecv(rbuf[0], MSG_LEN, MPI_DOUBLE, prev, 0, MPI_COMM_WORLD, &reqs[0]);
        MPI_Irecv(rbuf[1], MSG_LEN, MPI_DOUBLE, next, 0, MPI_COMM_WORLD, &reqs[1]);
        MPI_Isend(sbuf[0], MSG_LEN, MPI_DOUBLE, prev, 0, MPI_COMM_WORLD, &reqs[2]);
        MPI_Isend(sbuf[1], MSG_LEN, MPI_DOUBLE, next, 0, MPI_COMM_WORLD, &reqs[3]);

        FJMPI_Progress_start();

        for (j = 0; j < VEC_LEN; j++) {
            y[j] = a * x[j] + y[j];
        }

        FJMPI_Progress_stop();

        MPI_Waitall(4, reqs, MPI_STATUSES_IGNORE);

        etime = MPI_Wtime();

    }

    MPI_Finalize();

    if (rank == 0) {
        printf("%.6f sec\n", etime - stime);
    }

    return 0;
}
```

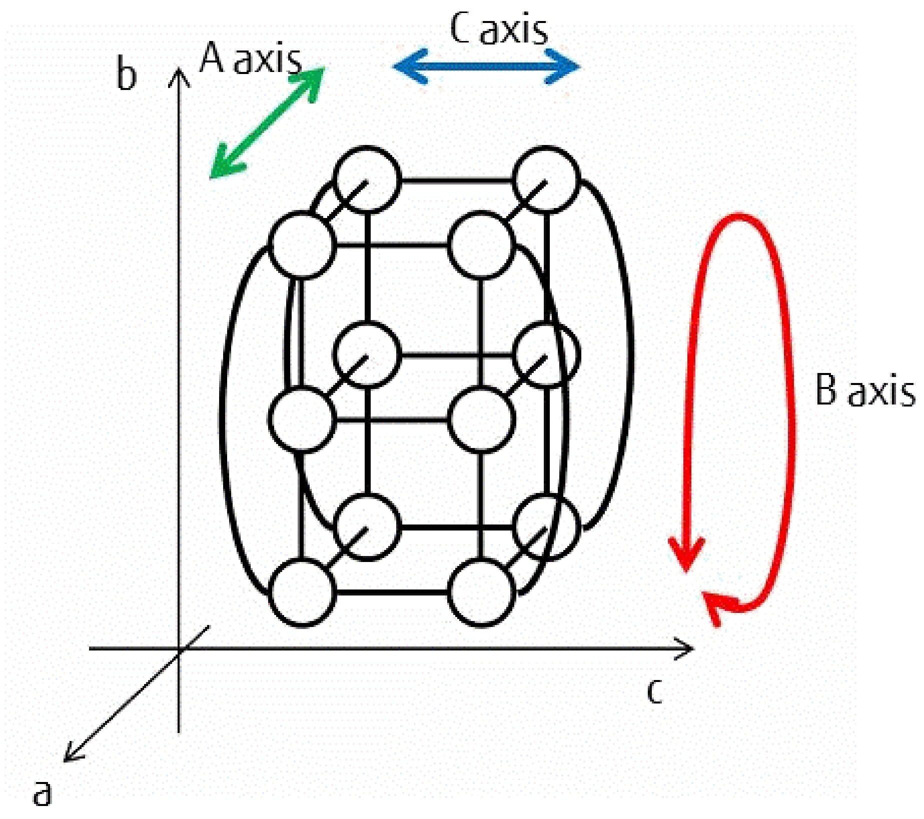
Chapter 6 Supplementary Items

6.1 Tofu Interconnect

6.1.1 Tofu Interconnect Configuration

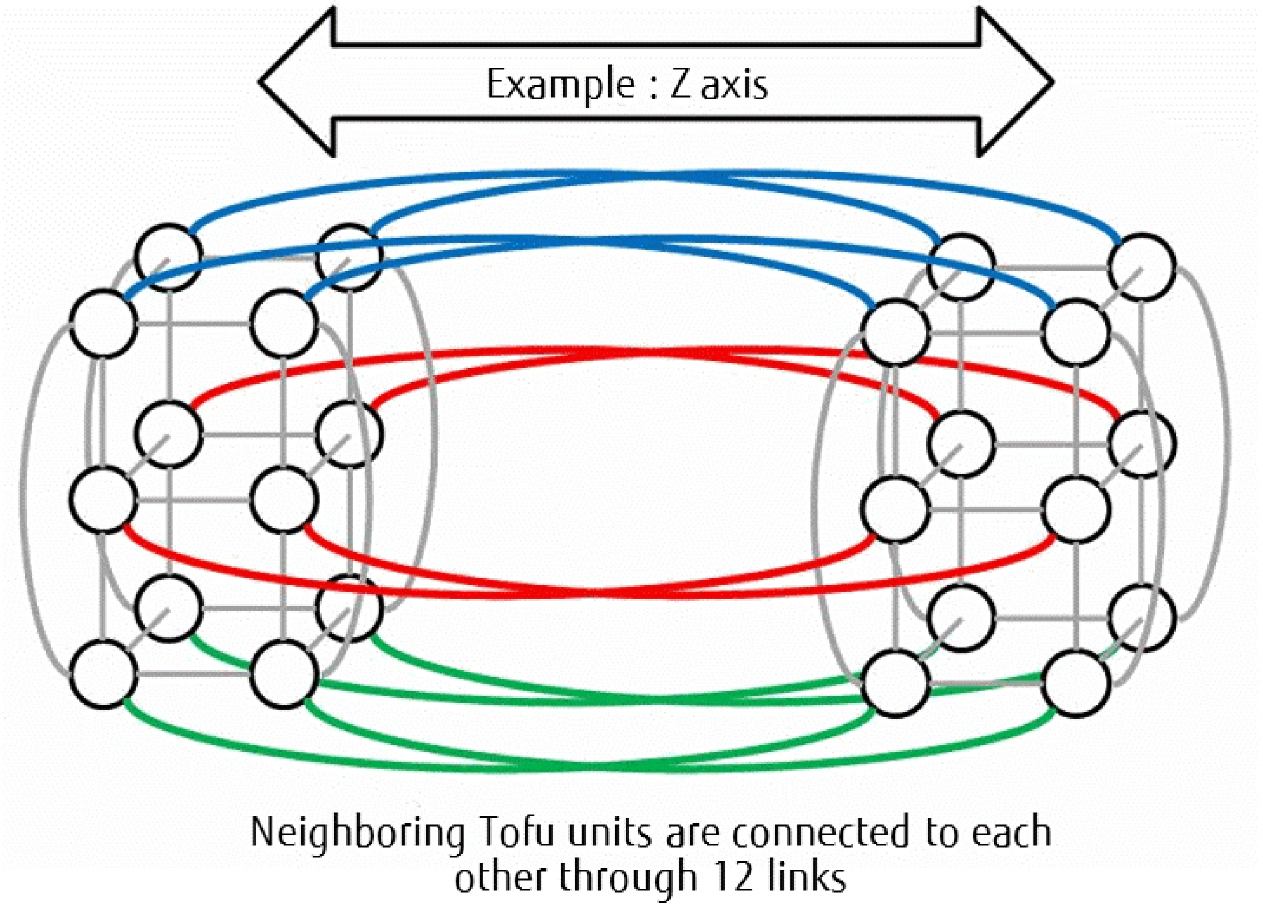
A Tofu interconnect is physically comprised of a 6-dimensional mesh/torus network. The coordinates of the 6-dimensional mesh/torus network are given by the six dimensions X, Y, Z, A, B, and C. The unit comprised of the A, B, and C axes of size 2 x 3 x 2 is known as a Tofu unit.

Figure 6.1 A Tofu unit



Neighboring Tofu units are connected by the X, Y, and Z axes, and constructed such that nodes having the same A, B, and C axis coordinates are tied together. Accordingly, a Tofu unit has 12 connections in each X, Y, and Z direction.

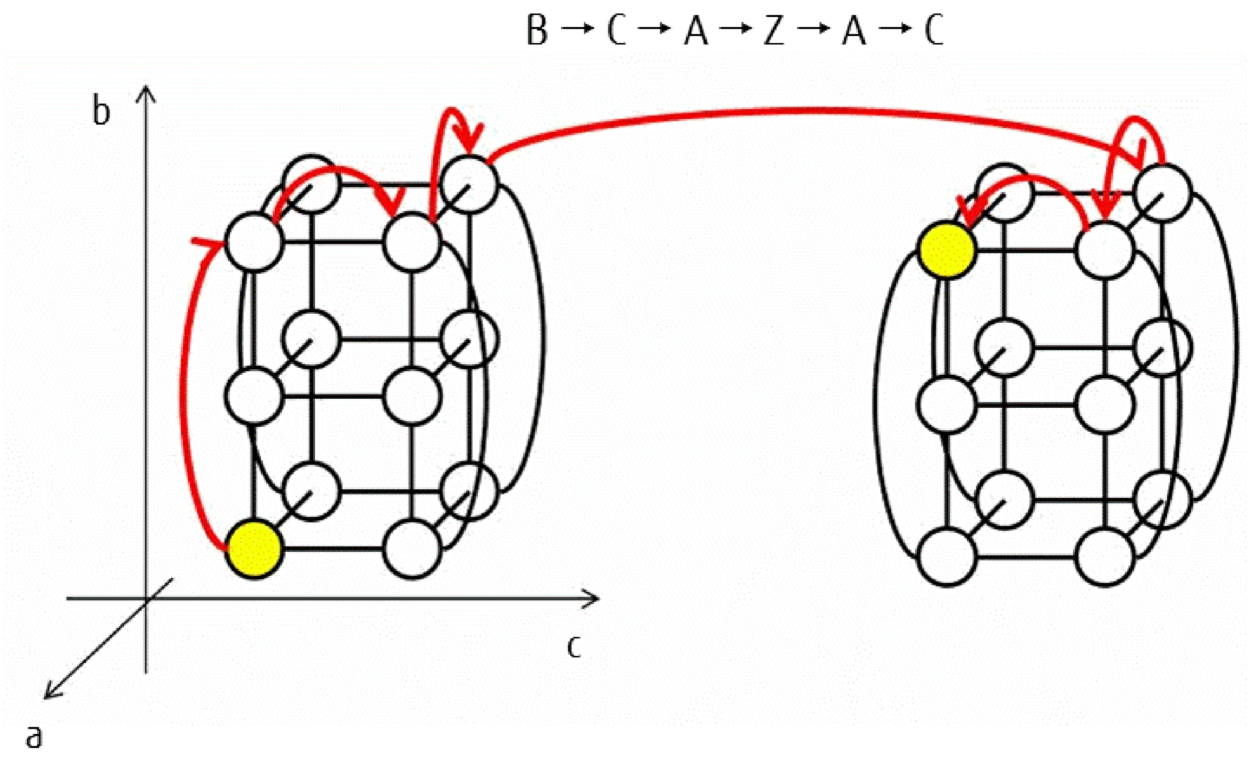
Figure 6.2 Connections to other Tofu units


Example : Z axis
Neighboring Tofu units are connected to each other through 12 links

6.1.2 Routing

Packets move in the Tofu interconnect in the coordinate axis sequence of B, C, A, X, Y, Z, A, C, B. The initial ABC axis routes are aimed at avoiding faulty nodes and distributing routes. The rest routes are aimed at reaching the destination node. The MPI library specifies which route to be used among 12 initial ABC routes. The Tofu interconnect's routing have more routes than ordinary dimension order routing. Therefore, this is known as extended dimension order routing.

Figure 6.3 Example of initial ABC axis routing moving all ABC axes

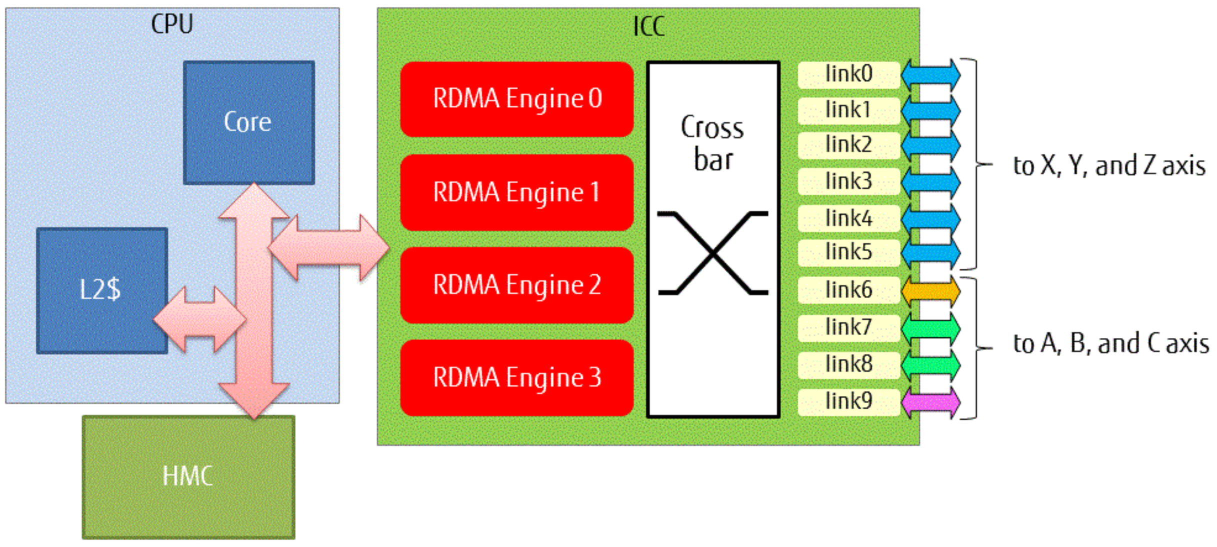


6.1.3 Configuration within a Node

Each node contains a module known as an Interconnect Controller (ICC) for controlling communication with other nodes. Internally there are four RDMA engines, known as Tofu Network Interfaces (TNI). Each TNI is capable of one send and one receive simultaneously. By using four TNIs, an ICC is capable of four sends and four receives simultaneously. There are a total of 10 ports, six in the XYZ direction and four in the ABC direction.

The MPI library uses these four RDMA engines, and executes RDMA communication.

Figure 6.4 ICC configuration



6.2 Promoting Asynchronous Communication Using an Assistant Core

On this system, even if you use nonblocking communication with an expectation of overlap of computation and communication, the communication may not progress completely asynchronously behind the computation, and the transfer of the message body may begin when a completion function such as MPI_Wait is called. In this case, the expected overlap cannot be achieved. SPARC64 XIfx, the CPU of the FX100 system, has 2 assistant cores that are dedicated to the OS and IO processing, in addition to 32 calculation cores that are dedicated to execution of user programs submitted as jobs. So this system has a function to promote asynchronous communication in an MPI programs that use nonblocking communication, by using the assistant core. This function creates a thread called "MPI asynchronous processing progress thread" on the assistant core in order to progress nonblocking communication when an MPI program is executed. This enables asynchronous communication on the assistant core behind computation on the calculation cores. In other words, it encourages overlap of computation and communication, aiming for a reduced MPI program execution time. This effect tends to be larger when the message size to be sent by nonblocking communication is large, especially when the message size is equal to or more than the threshold value.

However, as the assistant core is shared and used by multiple user processes, daemon processes, and OS in the compute node, there is a possibility that the side effects are generated that the execution times vary on multiple runs or the execution time is increased. These side effects may become larger especially if there are more than four processes at one node as a guide.

Moreover, when an MPI program calls an MPI function, an exclusive lock operation needed between a thread that processes the MPI function on the calculation core and the MPI asynchronous processing progress thread on the assistant core becomes a high cost. Therefore this system provides three operation modes shown in the "Table 6.1 Operation modes of the MPI asynchronous processing progress thread" below. The "value" column means values to specify for the MCA parameter opal_progress_thread_mode.

Table 6.1 Operation modes of the MPI asynchronous processing progress thread

Operation mode name	Value	Explanation of operation mode
Manual section (without MPI call) mode	1	Promote asynchronous communication in the sections specified by section specifying functions. MPI functions and extended interfaces cannot be called in the section. Modification of the MPI program is required to use this mode. In this mode, the exclusive lock operation is processed only when the section specifying functions are called. Therefore the performance overhead is smallest among three modes.
Manual section (with MPI call) mode	2	Promote asynchronous communication in the sections specified by section specifying functions. MPI functions and extended interfaces can be called in the section. Modification of the MPI program is required to use this mode. In this mode, the exclusive lock operation is processed only when the section specifying functions are called and when any MPI functions are called in the section. Therefore the performance overhead is relatively small if there are few MPI function calls in the section. But each MPI function call involves a small overhead even on the outside the section.
Automatic section mode	3	Promote asynchronous communication in the sections where at least one active request of nonblocking communication exists. MPI functions and extended interfaces can be called in the section. Modification of the MPI program is not required to use this mode. The section is automatically detected by this system. In this mode, the exclusive lock operation is processed when MPI functions are called in the section. Therefore the performance overhead is large if the MPI program calls many MPI functions between the start of the nonblocking communication by MPI_Isend function etc. and the completion the nonblocking communication by MPI_Wait function etc.

Refer to "5.5 MPI Asynchronous Communication Promotion Section Specifying Interface" for details of the section specifying functions described above.

To use this function, the MCA parameter opal_progress_thread_mode must be specified. Refer to "Table 4.41 opal_progress_thread_mode (specifies the operation mode of the MPI asynchronous processing progress thread)" for details.

Though this function can be used in combination with the extended interface described in "5.4 Extended Persistent Communication Requests Interface", there will be almost no performance gain.

Example

Extract from a program where asynchronous communications are promoted

```
MPI_Isend(sendbuf, 1048576, MPI_BYTE, sendpeer, tag, MPI_COMM_WORLD, &request[0]);
MPI_Irecv(recvbuf, 1048576, MPI_BYTE, recvpeer, tag, MPI_COMM_WORLD, &request[1]);
(computations)
MPI_Waitall(2, request, stat);
```

6.3 Notes Concerning MPI Standards Specifications

6.3.1 Supported Level of MPI Standards

The MPI library provided by this system conforms to the MPI-3.0 Standard specifications.

C++ is supported within the range of the MPI-2.2 Standard.

6.3.2 Predefined Datatypes that can be Used in This System

The predefined MPI datatypes and the corresponding data types of each program language that can be used in this system are shown below. The predefined MPI datatypes for Fortran are listed in "Table 6.2 Predefined MPI datatypes usable by Fortran", the predefined MPI datatypes for C are listed in "Table 6.3 Predefined MPI datatypes usable by C", and the predefined MPI datatypes for C++ are listed in "Table 6.4 Predefined MPI datatypes usable by C++".

Table 6.2 Predefined MPI datatypes usable by Fortran

Predefined MPI datatypes	Fortran data types
[Basic datatypes]	
MPI_CHARACTER	CHARACTER
MPI_LOGICAL	LOGICAL
MPI_LOGICAL1	LOGICAL(1)
MPI_LOGICAL2	LOGICAL(2)
MPI_LOGICAL4	LOGICAL(4)
MPI_LOGICAL8	LOGICAL(8)
MPI_INTEGER	INTEGER
MPI_INTEGER1	INTEGER(1)
MPI_INTEGER2	INTEGER(2)
MPI_INTEGER4	INTEGER(4)
MPI_INTEGER8	INTEGER(8)
MPI_REAL	REAL
MPI_REAL4	REAL(4)
MPI_REAL8	REAL(8)
MPI_REAL16	REAL(16)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_COMPLEX8	COMPLEX(4)
MPI_COMPLEX16	COMPLEX(8)
MPI_COMPLEX32	COMPLEX(16)
MPI_DOUBLE_COMPLEX	COMPLEX(8)
MPI_BYTE	
MPI_PACKED	
MPI_AINT	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_OFFSET	INTEGER (KIND=MPI_OFFSET_KIND)
MPI_COUNT	INTEGER (KIND=MPI_COUNT_KIND)

Predefined MPI datatypes	Fortran data types
MPI_LB	
MPI_UB	
MPI_2REAL	REAL pair
MPI_2DOUBLE_PRECISION	DOUBLE PRECISION pair
MPI_2INTEGER	INTEGER pair
MPI_2COMPLEX	COMPLEX pair
MPI_2DOUBLE_COMPLEX	DOUBLE COMPLEX pair

Table 6.3 Predefined MPI datatypes usable by C

Predefined MPI datatypes	C data types
[Basic datatypes]	
MPI_CHAR	char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	
MPI_PACKED	
MPI_C_BOOL	_Bool
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPI_AINT	MPI_Aint
MPI_OFFSET	MPI_Offset
MPI_COUNT	MPI_Count
MPI_LB	
MPI_UB	
MPI_FLOAT_INT	float and signed int pair
MPI_DOUBLE_INT	double and signed int pair
MPI_LONG_INT	signed long int and signed int pair
MPI_2INT	signed int pair
MPI_SHORT_INT	signed short int and signed int pair
MPI_LONG_DOUBLE_INT	long double and signed int pair
MPI_C_COMPLEX	float _Complex
MPI_C_FLOAT_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex

Predefined MPI datatypes	C data types
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_CHARACTER	CHARACTER (Fortran)
MPI_LOGICAL	LOGICAL (Fortran)
MPI_LOGICAL1	LOGICAL(1) (Fortran)
MPI_LOGICAL2	LOGICAL(2) (Fortran)
MPI_LOGICAL4	LOGICAL(4) (Fortran)
MPI_LOGICAL8	LOGICAL(8) (Fortran)
MPI_INTEGER	INTEGER (Fortran)
MPI_INTEGER1	INTEGER(1) (Fortran)
MPI_INTEGER2	INTEGER(2) (Fortran)
MPI_INTEGER4	INTEGER(4) (Fortran)
MPI_INTEGER8	INTEGER(8) (Fortran)
MPI_REAL	REAL (Fortran)
MPI_REAL4	REAL(4) (Fortran)
MPI_REAL8	REAL(8) (Fortran)
MPI_REAL16	REAL(16) (Fortran)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION (Fortran)
MPI_COMPLEX	COMPLEX (Fortran)
MPI_COMPLEX8	COMPLEX(4) (Fortran)
MPI_COMPLEX16	COMPLEX(8) (Fortran)
MPI_COMPLEX32	COMPLEX(16) (Fortran)
MPI_DOUBLE_COMPLEX	COMPLEX(8) (Fortran)
MPI_2REAL	REAL (Fortran) pair
MPI_2DOUBLE_PRECISION	DOUBLE PRECISION (Fortran) pair
MPI_2INTEGER	INTEGER (Fortran) pair
MPI_2COMPLEX	COMPLEX (Fortran) pair
MPI_2DOUBLE_COMPLEX	COMPLEX(8) (Fortran) pair
MPI_CXX_BOOL	bool (C++)
MPI_CXX_FLOAT_COMPLEX	std::complex<float> (C++)
MPI_CXX_DOUBLE_COMPLEX	std::complex<double> (C++)
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double> (C++)

Table 6.4 Predefined MPI datatypes usable by C++

Predefined MPI datatypes	C++ data types
[Basic datatypes]	
MPI::CHAR	char
MPI::SHORT	signed short int
MPI::INT	signed int
MPI::LONG	signed long int
MPI::LONG_LONG	signed long long int
MPI::SIGNED_CHAR	signed char
MPI::UNSIGNED_CHAR	unsigned char
MPI::UNSIGNED_SHORT	unsigned short int
MPI::UNSIGNED	unsigned int
MPI::UNSIGNED_LONG	unsigned long int
MPI::UNSIGNED_LONG_LONG	unsigned long long int
MPI::FLOAT	float
MPI::DOUBLE	double
MPI::LONG_DOUBLE	long double
MPI::WCHAR	wchar_t
MPI::BYTE	
MPI::PACKED	
MPI::LB	
MPI::UB	

Predefined MPI datatypes	C++ data types
MPI::BOOL	bool
MPI::COMPLEX	Complex<float>
MPI::DOUBLE_COMPLEX	Complex<double>
MPI::LONG_DOUBLE_COMPLEX	Complex<long double>
MPI::FLOAT_INT	float and signed int pair
MPI::DOUBLE_INT	double and signed int pair
MPI::LONG_INT	signed long int and signed int pair
MPI::TWOINT	signed int pair
MPI::SHORT_INT	signed short int and signed int pair
MPI::LONG_DOUBLE_INT	long double and signed int pair
MPI::INTEGER	INTEGER (Fortran)
MPI::INTEGER1	INTEGER(1) (Fortran)
MPI::INTEGER2	INTEGER(2) (Fortran)
MPI::INTEGER4	INTEGER(4) (Fortran)
MPI::REAL	REAL (Fortran)
MPI::REAL4	REAL(4) (Fortran)
MPI::REAL8	REAL(8) (Fortran)
MPI::DOUBLE_PRECISION	DOUBLE PRECISION (Fortran)
MPI::F_COMPLEX	COMPLEX (Fortran)
MPI::LOGICAL	LOGICAL (Fortran)
MPI::CHARACTER	CHARACTER(1) (Fortran)
MPI::TWOREAL	REAL (Fortran) pair
MPI::TWODOUBLE_PRECISION	DOUBLE PRECISION (Fortran) pair
MPI::TWOINTEGER	INTEGER (Fortran) pair

6.3.3 Allowed Datatypes in Collective Communication (Reduction Operation)

Table below shows the datatypes that can be specified in collective communication reduction operations.

Table 6.5 Datatypes that can be specified in collective communication reduction operations

Operation	Datatypes	C	C/C++	Fortran
[C/ Fortran] MPI_MAX MPI_MIN [C++] MPI::MAX MPI::MIN	C integer datatypes Fortran integer datatypes	MPI_SIGNED_CHAR MPI_SHORT MPI_INT MPI_LONG MPI_LONG_LONG_INT MPI_LONG_LONG MPI_UNSIGNED_CHAR MPI_UNSIGNED_SHORT MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_UNSIGNED_LONG_LONG MPI_INT8_T MPI_INT16_T MPI_INT32_T MPI_INT64_T MPI_UINT8_T MPI_UINT16_T MPI_UINT32_T MPI_UINT64_T	MPI::SIGNED_CHAR MPI::SHORT MPI::INT MPI::LONG MPI::LONG_LONG MPI::UNSIGNED_CHAR MPI::UNSIGNED_SHORT MPI::UNSIGNED MPI::UNSIGNED_LONG MPI::UNSIGNED_LONG_LONG MPI::INTEGER MPI::INTEGER1 MPI::INTEGER2 MPI::INTEGER4	MPI_INTEGER MPI_INTEGER1 MPI_INTEGER2 MPI_INTEGER4 MPI_INTEGER8 MPI_AINT MPI_OFFSET Return value of MPI_TYPE_CREATE_F90_INTEGER(handle)
	Floating point datatypes	MPI_FLOAT MPI_DOUBLE MPI_LONG_DOUBLE	MPI::FLOAT MPI::DOUBLE MPI::LONG_DOUBLE	MPI_REAL MPI_REAL4 MPI_REAL8

Operation	Datatypes	C	C/C++	Fortran
			MPI::REAL MPI::REAL4 MPI::REAL8 MPI::DOUBLE_PRECISION	MPI_REAL16 MPI_DOUBLE_PRECIS ION Return value of MPI_TYPE_CREATE_F 90_REAL(handle)
[C/ Fortran] MPI_SUM MPI_PROD [C++] MPI::SUM MPI::PROD	C integer datatypes Fortran integer datatypes	MPI_SIGNED_CHAR MPI_SHORT MPI_INT MPI_LONG MPI_LONG_LONG_INT MPI_LONG_LONG MPI_UNSIGNED_CHAR MPI_UNSIGNED_SHORT MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_UNSIGNED_LONG_LONG MPI_INT8_T MPI_INT16_T MPI_INT32_T MPI_INT64_T MPI_UINT8_T MPI_UINT16_T MPI_UINT32_T MPI_UINT64_T	MPI::SIGNED_CHAR MPI::SHORT MPI::INT MPI::LONG MPI::LONG_LONG MPI::UNSIGNED_CHAR MPI::UNSIGNED_SHORT MPI::UNSIGNED MPI::UNSIGNED_LONG MPI::UNSIGNED_LONG_LO NG MPI::INTEGER MPI::INTEGER1 MPI::INTEGER2 MPI::INTEGER4	MPI_INTEGER MPI_INTEGER1 MPI_INTEGER2 MPI_INTEGER4 MPI_INTEGER8 MPI_AINT MPI_OFFSET Return value of MPI_TYPE_CREATE_F 90_INTEGER(handle)
	Floating point datatypes	MPI_FLOAT MPI_DOUBLE MPI_LONG_DOUBLE	MPI::FLOAT MPI::DOUBLE MPI::LONG_DOUBLE MPI::REAL MPI::REAL4 MPI::REAL8 MPI::DOUBLE_PRECISION	MPI_REAL MPI_REAL4 MPI_REAL8 MPI_REAL16 MPI_DOUBLE_PRECIS ION Return value of MPI_TYPE_CREATE_F 90_REAL(handle)
	Complex number datatypes	MPI_C_COMPLEX MPI_C_FLOAT_COMPLEX MPI_C_DOUBLE_COMPLEX MPI_C_LONG_DOUBLE_COMP LEX MPI_CXX_FLOAT_COMPLEX MPI_CXX_DOUBLE_COMPLEX MPI_CXX_LONG_DOUBLE_CO MPLEX	MPI::F_COMPLEX MPI::COMPLEX MPI::F_DOUBLE_COMPLEX MPI::DOUBLE_COMPLEX MPI::LONG_DOUBLE_COMP LEX	MPI_COMPLEX MPI_COMPLEX4 MPI_COMPLEX8 MPI_COMPLEX16 MPI_COMPLEX32 MPI_DOUBLE_COMPLE X Return value of MPI_TYPE_CREATE_F 90_COMPLEX(handle)
[C/ Fortran] MPI_LAND MPI_LOR MPI_LXOR [C++]	C integer datatypes Logical datatypes	MPI_SIGNED_CHAR MPI_SHORT MPI_INT MPI_LONG MPI_LONG_LONG_INT MPI_LONG_LONG MPI_UNSIGNED_CHAR	MPI::SIGNED_CHAR MPI::SHORT MPI::INT MPI::LONG MPI::LONG_LONG MPI::UNSIGNED_CHAR MPI::UNSIGNED_SHORT	MPI_LOGICAL MPI_LOGICAL1 MPI_LOGICAL2 MPI_LOGICAL4 MPI_LOGICAL8 MPI_AINT MPI_OFFSET

Operation	Datatypes	C	C/C++	Fortran
MPI::LAND MPI::LOR MPI::LXOR		MPI_UNSIGNED_SHORT MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_UNSIGNED_LONG_LONG MPI_INT8_T MPI_INT16_T MPI_INT32_T MPI_INT64_T MPI_UINT8_T MPI_UINT16_T MPI_UINT32_T MPI_UINT64_T MPI_C_BOOL	MPI::UNSIGNED MPI::UNSIGNED_LONG MPI::UNSIGNED_LONG_LONG MPI::LOGICAL MPI::BOOL	Return value of MPI_TYPE_CREATE_F90_INTEGER(handle)
[C/ Fortran] MPI_BAND MPI_BOR MPI_BXOR [C++] MPI::BAND MPI::BOR MPI::BXOR	C integer datatypes Fortran integer datatypes	MPI_SIGNED_CHAR MPI_SHORT MPI_INT MPI_LONG MPI_LONG_LONG_INT MPI_LONG_LONG MPI_UNSIGNED_CHAR MPI_UNSIGNED_SHORT MPI_UNSIGNED MPI_UNSIGNED_LONG MPI_UNSIGNED_LONG_LONG MPI_INT8_T MPI_INT16_T MPI_INT32_T MPI_INT64_T MPI_UINT8_T MPI_UINT16_T MPI_UINT32_T MPI_UINT64_T	MPI::SIGNED_CHAR MPI::SHORT MPI::INT MPI::LONG MPI::LONG_LONG MPI::UNSIGNED_CHAR MPI::UNSIGNED_SHORT MPI::UNSIGNED MPI::UNSIGNED_LONG MPI::UNSIGNED_LONG_LONG MPI::INTEGER MPI::INTEGER1 MPI::INTEGER2 MPI::INTEGER4	MPI_INTEGER MPI_INTEGER1 MPI_INTEGER2 MPI_INTEGER4 MPI_INTEGER8 MPI_AINT MPI_OFFSET Return value of MPI_TYPE_CREATE_F90_INTEGER(handle)
	Byte datatypes	MPI_BYTE	MPI::BYTE	MPI_BYTE
[C/ Fortran] MPI_MAXLOC MPI_MINLOC [C++] MPI::MAXLOC MPI::MINLOC	-	MPI_FLOAT_INT MPI_DOUBLE_INT MPI_LONG_DOUBLE_INT MPI_LONG_INT MPI_2INT MPI_SHORT_INT	MPI::FLOAT_INT MPI::DOUBLE_INT MPI::LONG_DOUBLE_INT MPI::LONG_INT MPI::TWOINT MPI::SHORT_INT MPI::TWOREAL MPI::TWODOUBLE_PRECISION MPI::TWOINTEGER	MPI_2REAL MPI_2DOUBLE_PRECISION MPI_2INTEGER

6.3.4 Reserved Communicators

As specified in the MPI standards, the following communicators are reserved in this system:

- MPI_COMM_WORLD
- MPI_COMM_SELF

In addition, MPI_COMM_NULL is reserved as a predefined constant.

6.3.5 Operations in a Multi-Threaded Environment

This system supports operations in a multi-thread environment. The thread support level of this system is MPI_THREAD_SERIALIZED.

MPI_THREAD_SERIALIZED means that MPI can be called from multiple threads but that they cannot be called simultaneously. In other words, the invocation of MPI from all of the threads must be serialized. A user application program must internally support this serialization of MPI calls. Note that the behavior of an MPI program is not guaranteed if MPI invocation is not serialized. Refer to the MPI standards for details.

6.3.6 Signal Operation Changes

In this system, the system standard operation of the signals shown in table below is changed.

Table 6.6 Names of signals with changed system standard operation

Signal name
SIGABRT SIGBUS SIGFPE SIGSEGV SIGCHLD

In addition, this system uses one realtime signal. Refer to the commercial Linux-related publications for information on realtime signals.

6.3.7 One-sided Communications

This section provides notes on one-sided communications.

6.3.7.1 Assertions for Optimization

In this system, the argument assert of MPI_WIN_POST, MPI_WIN_START, MPI_WIN_FENCE, MPI_WIN_LOCK, and MPI_WIN_LOCK_ALL is used for optimization. Assertions supported in this system are shown in table below.

Table 6.7 Assertions for optimizations in one-sided communications

MPI function name	Assertion	Operation in this system
MPI_WIN_POST	MPI_MODE_NOCHECK	Ignored
	MPI_MODE_NOSTORE	Ignored
	MPI_MODE_NOPUT	Ignored
MPI_WIN_START	MPI_MODE_NOCHECK	Ignored
MPI_WIN_FENCE	MPI_MODE_NOSTORE	Ignored
	MPI_MODE_NOPUT	Ignored
	MPI_MODE_NOPRECEDE	The fence does not complete any sequence of locally issued RMA calls. If this assertion is given by any group, then by any process in the window group, then it must be given by all processes in the group.
	MPI_MODE_NOSUCCEED	The fence does not start sequence of locally issued RMA calls. If this assertion is given by any group, then by any process in the window group, then it must be given by all processes in the group.
MPI_WIN_LOCK, MPI_WIN_LOCK_ALL	MPI_MODE_NOCHECK	Ignored

6.3.7.2 Info Argument

In this system, the info argument of MPI_WIN_CREATE, MPI_WIN_ALLOCATE, MPI_WIN_CREATE_DYNAMIC, and MPI_WIN_ALLOCATE_SHARED functions is used to decide the behavior of the window created by these functions.

The info keys that can be specified for an info argument in this system are shown below.

Table 6.8 Info key for MPI_WIN_CREATE, MPI_WIN_ALLOCATE, and MPI_WIN_CREATE_DYNAMIC functions

Info key	Operation in this system
accumulate_ops	The behavior when the same_op is specified as a key value is the same as when the same_op_no_op is specified. The default value for this info key is the same_op_no_op.

Table 6.9 Info key for MPI_WIN_ALLOCATE_SHARED function

Info key	Operation in this system
alloc_shared_noncontig	When true is specified for the value of this info key, the memory is allocated in a location that is close to each process. When false is specified for the value of this info key, the memory is allocated continuously across process ranks. The default value for this info key is false.

6.3.8 Establishing Communication between Groups not Sharing a Communicator

This section provides notes on establishing communication between two MPI program groups that do not share a communicator.

This communication can be established using background execution within one job, in particular, by using background execution and specifying execution of the corresponding multiple mpiexec(1) in the job script. The -vcoordfile option or the --vcoordfile option must be specified for mpiexec(1). Refer to "4.1 Execution Command Formats" for details.

6.3.8.1 info Argument Value

Specify MPI_INFO_NULL as the info input argument of the MPI_Open_port function, the MPI_Comm_accept function, the MPI_Comm_connect function, the MPI_Publish_name function, the MPI_Unpublish_name function, and the MPI_Lookup_name function.

6.3.8.2 MPI_Open_port Function Behavior

One MPI_Open_port function call consumes one TCP port. The number of available TCP ports is limited and depends on the system you use. MPI_Open_port function call may fail by the shortage of TCP ports if you call MPI_Open_port function repeatedly without closing TCP port by MPI_Close_port function. In this case, an error message is output and an error of the error class MPI_ERR_INTERN may occur.

6.3.8.3 MPI_Comm_join Function Return Value

The output of the MPI_Comm_join function is MPI_COMM_NULL.

6.3.8.4 Service Names in the MPI_Publish_name Function

The maximum length of a service name in this system is 63 characters (63 bytes) (NULL characters are not included in C or C++). In this system, the job unit is the effective range for a published service name. If there is an attempt to publish the same service name again within one job, an error of the error class MPI_ERR_SERVICE may occur.

6.3.8.5 MPI_Unpublish_name Function Behavior

In this system, if a process other than the MPI process that published a service name attempts to unpublish that service name using the MPI_Unpublish_name function, an error of the error class MPI_ERR_ACCESS may occur.

6.3.8.6 Socket Communication Wait Time

Socket communication is used to establish communication. In this system, the socket communication wait time is normally set as unlimited. This wait time can be changed using the MCA parameter dpm_ple_socket_timeout. However, if this wait time is exceeded during socket communication, an error message is output and an error of the error class MPI_ERR_PORT may occur. Refer to "Table 4.33 dpm_ple_socket_timeout (specifies the socket communication wait time when establishing communication)" for information on the MCA parameter dpm_ple_socket_timeout.

After communication is established, the socket communication path between the MPI process groups is disconnected. Socket communication is not used for communication between MPI processes.

6.3.9 Dynamic Process Generation

This section provides notes on dynamic process generation,

6.3.9.1 Socket Communication Wait Time

When the MPI_Comm_spawn and MPI_Comm_spawn_multiple functions are called to launch processes dynamically and establish communication, socket communication is used to establish communication with the dynamically generated processes as described in "6.3.8.6 Socket Communication Wait Time".

6.3.9.2 info Argument Value

For the MPI_Comm_spawn and MPI_Comm_spawn_multiple functions provided by this system, wdir is the only key that can be specified in the info argument. If a key other than wdir is specified, it is ignored.

The current directory path name can be passed by specifying the wdir key as the info argument of the MPI_Comm_spawn and the MPI_Comm_spawn_multiple functions of this system. Alternatively, Job Operation Software also provides an option (rankdir specification) related to specifying the current directory. The current directory interpretation varies, depending on the combination of wdir key and rankdir specifications. Table below shows the current directory interpretation for each wdir key and rankdir specification. Refer to the Job Operation Software manual for details.

Table 6.10 Current directory interpretation for each wdir key and rankdir specification

Info argument wdir	rankdir specification	Current directory of dynamically generated process
Not specified	Specified	Job number directory (/path/jobid)
	Not specified	Current directory when the MPI_Comm_spawn function is called
Specified Relative path specified	Specified	Specified relative path from the job number directory (/path/jobid)
	Not specified	Specified relative path from the current directory when the MPI_Comm_spawn function is called
Specified Absolute path specified	Specified	Specified absolute path
	Not specified	Specified absolute path

6.3.9.3 Notes

Do not use the dynamic process generation function of this system when the following conditions apply:

- The --vcoordfile option for mpiexec(1) execution is specified for background execution.
- The total invocation count for the MPI_Comm_spawn function or the MPI_Comm_spawn_multiple function exceeds 4294967295 for one execution of mpiexec(1) when dynamic process generation is used
- The number of MPI_COMM_WORLD for the dynamic processes that exists at the same time exceeds 65535.
- Dynamic process generation executes something other than an MPI program

If dynamic process generation is used under the first three conditions, Job Operation Software outputs an error message and then the MPI program ends abnormally.

Note that the behavior is not guaranteed if something other than an MPI program is executed under the fourth condition.

Refer to the Job Operation Software manual for details on the Job Operation Software.

If the same process repeatedly executes dynamic process generation, the generated process generation information and communicator information accumulates in memory and may cause memory shortages. Use with care,

6.3.10 Rank Changes in Accordance with Cartesian Topology

When the MPI_Cart_create function is used to generate a new communicator with attached Cartesian topology, new rank numbers can be allocated in the order of the Cartesian coordinates, to the parallel processes belonging to the generated communicator by specifying true in the reorder argument. In other words, ranks can be changed on the basis of the Cartesian coordinates.

The conditions for changing ranks on the basis of Cartesian coordinates and the rules for rank changes are explained below.

6.3.10.1 Conditions Enabling Rank Changes

This section explains the conditions that must be met in order to change ranks using the MPI_Cart_create function.

Note that rank changes are performed on the basis of the coordinates of the node space that Job Operation Software allocates to each parallel process. To distinguish the coordinates of parallel processes from Cartesian coordinates, this manual refers to them as "logical coordinates" or simply "coordinates".

The MPI_Cart_create function changes ranks if all the following conditions are met:

- true is specified for the reorder argument
- The process shape of the input communicator comm_old (that is, the number of dimensions and the number of processes in each dimension) is the same as the shape of the Cartesian topology being newly attached
- The process shape of the input communicator comm_old has no logical coordinate duplications or gaps

In this system, shape matching when changing ranks compares the X, Y, and Z axes of the logical coordinates in sequence from the start of the array in the dims argument of the MPI_Cart_create function.

6.3.10.2 Rules for Rank Changes

When a Cartesian topology is generated by the MPI_Cart_create function, the Cartesian coordinates allocate rank numbers in ascending order starting from 0, based on the logical coordinates of the process shape in accordance with the sequence below, to the parallel processes belonging to input communicator comm_old.

- If the shape is one-dimensional (X axis), the process with coordinates closest to the logical coordinates starting point (0) is set as rank 0, and ranks are set, with that as a starting point, in sequence as processes become further away on the X axis.
- If the shape is two-dimensional (X axis and Y axis), the process with coordinates closest to the logical coordinates starting point (0,0) is set as rank 0, and ranks are set, with that as a starting point, in sequence as processes become further away on the Y axis, and then the X axis.
- If the shape is three-dimensional (X axis, Y axis, and Z axis), the process with coordinates closest to the logical coordinates starting point (0,0,0) is set as rank 0, and ranks are set, with that as a starting point, in sequence as processes become further away on the Z axis, then the Y axis, and then the X axis.

6.3.10.3 Checking Rank Changes

This system provides the extended interface FJMPI_Topology_cart_reorder for checking whether or not ranks have been changed. Refer to "5.1 Rank Query Interface" for information on the extended interface FJMPI_Topology_cart_reorder.

6.3.10.4 Sample Program

The MPI program example below executes the MPI_Cart_create function for a communicator with the following conditions and checks whether or not the ranks have actually been changed:

1. Dimensions: 3 dimensions
2. Node shape (X:2,Y:3,Z:4)

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <mpi.h>
#include <mpi-ext.h>
```

```
#define FAILURE 1

int main(int argc, char *argv[])
{
    MPI_Comm cart_comm_on;
    MPI_Comm cart_comm_off;
    int size, rank;
    int dims_on[3] = {2, 3, 4};    /* 3-dimensions 2x3x4 is reordered. */
    int dims_off[3] = {3, 4, 2};   /* 3-dimensions 3x4x2 is not reordered. */
    int periods[3] = {1, 1, 1};    /* cyclic fixed */
    int cart_result;
    char host[255];

    gethostname(host, 255);

    MPI_Init(&argc, &argv);

    MPI_Comm_size(MPI_COMM_WORLD, &size);
    MPI_Comm_rank(MPI_COMM_WORLD, &rank);

    fprintf(stderr, "[%s] MPI_COMM_WORLD's MPI_Comm_size : %d\n", host, size);
    fprintf(stderr, "[%s] rank of MPI_COMM_WORLD = %d\n", host, rank);

    MPI_Cart_create(MPI_COMM_WORLD, 3, dims_on, periods, 1, &cart_comm_on);

    MPI_Comm_rank(cart_comm_on, &rank);
    fprintf(stderr, "rank after MPI_Cart_create() = %d\n", rank);

    if(MPI_SUCCESS != FJMPI_Topology_cart_reorder(cart_comm_on, &cart_result)) {
        fprintf(stderr, "FJMPI_Topology_cart_reorder ERROR\n");
        MPI_Abort(cart_comm_on, FAILURE);
    }

    fprintf(stderr, "[%s] Cartesian Reorder cart_comm_on --> %s \n",
            host, cart_result ? "ON" : "OFF");

    MPI_Cart_create(MPI_COMM_WORLD, 3, dims_off, periods, 1, &cart_comm_off);

    if(MPI_SUCCESS != FJMPI_Topology_cart_reorder(cart_comm_off, &cart_result)) {
        fprintf(stderr, "FJMPI_Topology_cart_reorder ERROR\n");
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }

    fprintf(stderr, "[%s] Cartesian Reorder cart_comm_off --> %s \n",
            host, cart_result ? "ON" : "OFF");

    MPI_Finalize();

    return 0;

}
```

Ranks are changed by the first MPI_Cart_create function invocation, but not by the second invocation. This is because the Cartesian topology shape specified in the argument in the second MPI_Cart_create function invocation is not the same as the process shape specified when the MPI program was executed.

After these MPI_Cart_create functions are called, the program uses the FJMPI_Topology_cart_reorder function to evaluate whether or not the ranks have in fact been changed.

The evaluation result is returned to the second argument (cart_result variable in the program) of the FJMPI_Topology_cart_reorder function. If the evaluation result value is 0, this indicates that the ranks were not changed.

6.3.11 Notes on Send Buffer and Receive Buffer

A memory area to which an MPI program parallel process cannot write should not be specified in the send buffer or receive buffer. Behavior is not guaranteed if the incorrect type of memory area is specified in the send buffer.

Memory areas to which an MPI program parallel process cannot write include, for example, MPI program instruction areas (.text sections), and similar.

6.3.12 MPI Input-Output

The behavior of the I/O of MPI in this system conforms to the implementation of ROMIO.

Information concerning ROMIO is available from http://www.mcs.anl.gov/research/projects/romio/ .

The following restrictions exist.

- In FEFS, when the MPI_File_open function is executed without specifying MPI_MODE_CREATE and the file does not exist, the file is created, and it succeeds in executing the function. In other filesystems, it fails in the execution of the function.
- The file input-output data size that can be handled once is less than 2GiB. The data size is "the datatypes size of one element * number of elements".

The file systems handled as MPI input-output are FEFS, UFS, and NFS. Other file systems are not supported. In addition, for NFS, you must specify noac for mount option to prevent the unmatch of the content because of the data update delay. Refer to the FEFS manual for information on FEFS.

If external32 is specified for data expression, do not use the following datatypes:

- MPI_LONG
- MPI_UNSIGNED_LONG
- MPI_WCHAR

The MPI_File_open function provided by this system does not change the size of the existing file.

The input-output function of this system may create a temporary file in the same directory as the user's input-output file. These are the files generated when MPI_File_open is used to open a file. The size of one of these files is about eight bytes. The MPI_File_close function usually deletes this file but it may remain if the program is forcibly ended, or if an error occurs in the system.

If a file with a name like the one below remains after MPI program execution has ended, manually delete the file.

```
.[MPI I/O filename].shfp.numeric
```

The information set in the status of collective input-output functions (MPI_File_read_all and similar) is always based on the argument information at the time of invocation.

If the rank number directory (rankdir) is used in Job Operation Software, create the shared file to be used by MPI input-output in the directory ("../") that is one level higher than the current directory. Refer to the Job Operation Software manual for information on the rank number directory.

Table below shows the Info object keys and values that can be used by MPI input-output in this system.

Table 6.11 Info object keys and values that can be used by MPI input-output

Key	Default value	Description
cb_buffer_size	"16777216"	Size of the temporary buffer area used for collective access
cb_nodes	Number of hosts assigned to communicator that performs the input-output	Number of processes that actually perform input-output using collective access
ind_rd_buffer_size	"4194304"	Size of buffer area at time of individual process read
ind_wr_buffer_size	"524288"	Size of buffer area at time of individual process write

In addition to the above, the key shown in table below can be specified if the file system is FEFS.

Table 6.12 Info object keys and values that can be used with FEFS

Key	Default value	Meaning
direct_read	"false"	Specify whether or not to perform direct I-O (read). Specify either "true" or "TRUE" to use direct I-O (read).
direct_write	"false"	Specify whether or not to perform direct I-O (write). Specify either "true" or "TRUE" to use direct I-O (write).
striping_unit	New directory settings value	Width of one stripe
striping_factor	New directory settings value	Number of input-output devices that perform file striping

Note

1. The "striping_unit" value must be a whole number multiple of "65536"(64KiB). The minimum "striping_unit" value is "65536"(64KiB) and the maximum is "2147483648"(2GiB). If "0" is specified, 1048576 is used.
2. The "striping_factor" value must be within the range "-1" to "20000". If the value exceeds the number of OSTs in the FEFS file system, the entire number of OSTs in the file system is used as the specified value. If "-1" is specified, striping is performed using all OSTs. If "0" is specified, 1 is used.

6.3.13 Use of the Profiling Interface

- For C programs, MPI calls can be intercepted using the C binding of the profiling interface.
- For Fortran programs, MPI calls can be intercepted using the Fortran binding and the C binding of the profiling interface as described below.
  - MPI calls can be intercepted using Fortran binding.
  - MPI calls can also be intercepted using C binding except for some Fortran-specific subroutines/functions as follows:
    - MPI_ATTR_GET
    - MPI_ATTR_PUT
    - MPI_COMM_CREATE_ERRHANDLER
    - MPI_COMM_CREATE_KEYVAL
    - MPI_COMM_GET_ATTR
    - MPI_COMM_SET_ATTR
    - MPI_ERRHANDLER_CREATE
    - MPI_FILE_CREATE_ERRHANDLER
    - MPI_KEYVAL_CREATE
    - MPI_TYPE_CREATE_HVECTOR
    - MPI_TYPE_CREATE_KEYVAL
    - MPI_TYPE_GET_ATTR
    - MPI_TYPE_MATCH_SIZE
    - MPI_TYPE_SET_ATTR
    - MPI_WIN_CREATE_ERRHANDLER
    - MPI_WIN_CREATE_KEYVAL

        - MPI_WIN_GET_ATTR
        - MPI_WIN_SET_ATTR
    - MPI calls are intercepted in two levels if the subroutines/functions are configured to be intercepted using both the C binding and the Fortran binding.
- For C++ programs, MPI calls can be intercepted using the C binding of the profiling interface.

6.3.14 MPI Tool Information Interface

In this system, the MPI tool information interface is supported. But no control variables and no performance variables are exposed.

6.4 Eager Protocol and Rendezvous Protocol

This system implements two communication protocols, the Eager protocol and the Rendezvous protocol, for message communication.

Under the Eager protocol, the send side sends messages regardless of the receive side status. This is referred to as an asynchronous type of communication. In contrast, under the Rendezvous protocol, the send side and receive side perform linkage processing, and the send side does not send the message until the receive side message storage destination is established. This is referred to as a synchronous type of communication. Under the Eager protocol communication, the send side process sends messages without having information about the receive destination memory area of the user program. Therefore, it prepares in advance a buffer area, the size of the message or greater, in the internal MPI library area. The Eager protocol communication does not have the overhead of performing linkage processing between the send side and the receive side, but it does have the overhead of copying between the internal buffer area and the user program memory space. In contrast, under the Rendezvous protocol, the send side and receive side perform linkage processing in advance, and the send side process sends the message after the receive destination memory area of the user program is established. Thus, even if a large message is sent, a large internal buffer area is not required. In particular, if a message is continuous data, the message can be copied directly from the send side memory space to the receive side memory space of the user program without using the internal buffer area.

This system internally switches between these protocols according to the size of the message being sent. The Eager protocol is selected for short messages, and the Rendezvous protocol is selected for communication of large messages. More precisely, the distance (number of hops) of the message communication is also taken into account, not just the message size. The compute nodes in an FX100 system are physically connected by a mesh or torus format, and message communication between any two compute nodes is performed via a number of other compute nodes, as required. In this system, the "threshold value" (number of bytes) for switching between Eager communication and Rendezvous communication in the fast communication mode is obtained using the following formula:

```
Threshold value = 45,056 + number-of-hops x 296
```

In the memory-saving communication mode, this system automatically sets appropriate "threshold values" that reduce memory usage. Refer to "6.10 Suppressing Memory Usage" for information on the fast communication mode and the memory-saving communication mode.

In the following cases, the Rendezvous protocol communication may be faster regardless of the message size:

- MPI programs that use nonblocking communication to perform multiple communications simultaneously
- MPI programs (when Hasty Rendezvous communication is enabled) that execute receive functions (the MPI_Recv function, etc.) more quickly than send functions (the MPI_Send function, etc.)

In these cases, improved MPI program performance can be expected due to changing this "threshold value". The MCA parameter btl_tofu_eager_limit can be used to change the "threshold value". Refer to "Table 4.11 btl_tofu_eager_limit (changes the threshold value for switching the communication method)" for information on the MCA parameter btl_tofu_eager_limit.

6.5 Hasty Rendezvous Communication

Under Rendezvous protocol linkage processing is performed internally in advance, in order to synchronize the send side and receive side before messages are actually sent. This advance linkage processing performs control communications, that is, a request to communicate and a response to the request, under the Rendezvous protocol.

This system provides two mechanisms with different processing operations for Rendezvous control communications. The first mechanism synchronizes the responses. The second mechanism performs control communications asynchronously without synchronizing control communication responses. In this system, this second mechanism is known as Hasty Rendezvous communication. Normally, this system is set not to use Hasty Rendezvous communication. The Hasty Rendezvous communication setting can be changed by switching the MCA parameter pml_ob1_use_hasty_rendezvous value. Refer to "Table 4.46 pml_ob1_use_hasty_rendezvous (use of Hasty Rendezvous communication)" for information on the MCA parameter pml_ob1_use_hasty_rendezvous.

When the setting is changed to use the Hasty Rendezvous communication, there is the following restriction. Be careful when specifying message size to be sent and received.

- An error occurrence related to the threshold value for switching between Eager communication and Rendezvous communication
  When a communication that message size less than the threshold value is specified in the sender side and message size equal to or more than the threshold value is specified in the corresponding receiver side is performed, an error of the error class MPI_ERR_TRUNCATE may occur after that.

Hasty Rendezvous communication increases the promotion of asynchronous communication in MPI programs that use nonblocking communication. In other words, it encourages overlap of calculations and communication, aiming for a reduced MPI program execution time.
Specifically, in case all the following conditions are satisfied, Hasty Rendezvous communication is used in the system.

- Message size to be sent equal to or more than the threshold value
- Message data are continuous on both buffers of the send side and the receive side
- Neither MPI_ANY_SOURCE nor MPI_ANY_TAG is used
- Receive function is invoked before the corresponding send function (necessary only if the value of the MCA parameter pml_ob1_use_hasty_rendezvous is 1)
- Message data with size of more than the threshold value are sent continuously

However, memory usage may increase and, in the specific case where the send side and the receive side both perform nonblocking communication, performance is more likely to drop.

Example

Extract from a program where Hasty Rendezvous communication is applied

```
    if(myrank == 0){
        MPI_Send(sendbuf0, 1048576, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
        MPI_Send(sendbuf1, 1048576, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
        MPI_Send(sendbuf2, 1048576, MPI_BYTE, 1, tag, MPI_COMM_WORLD);
    }
    if(myrank == 1){
        MPI_Recv(recvbuf0, 1048576, MPI_BYTE, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
        MPI_Irecv(recvbuf1, 1048576, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &request[0]);
        MPI_Irecv(recvbuf2, 1048576, MPI_BYTE, 0, tag, MPI_COMM_WORLD, &request[1]);
        (computations)
        MPI_Waitall(2,request, stat);
    }
```

6.6 Stride RDMA Communication

With general derived datatype message communication, messages are split into multiple messages internally, and using Eager communication, messages are sent via a message pipeline. An internal buffer area is secured and the split message fragments are temporarily copied to that buffer area. As a result, the time taken for communication increases significantly for large messages.

This system uses the Tofu interconnect RDMA communication function instead of the pipeline-type processing used under Eager protocol. This improves communication because messages are copied directly from the user program data area and the internal buffer area is not used. In this system, this mechanism is called Stride RDMA communication. Stride RDMA communication enables better communication performance if the derived datatype configuration is relatively simple and the overall message size is large.
In practical terms, Stride RDMA communication is effective if all the following conditions are met:

- Same datatype on the receive side and the send side
- Datatype created using the MPI_Type_vector function or the MPI_Type_hvector function

- 16 or less contiguous blocks in the datatype
- Two or more entities in each block
- Address of each block adjusted to a 4-byte boundary
- Total size (number of bytes) of messages being sent is the same or greater than the "threshold value" for switching the protocol

Refer to "6.4 Eager Protocol and Rendezvous Protocol" for information on the "threshold value" for switching the protocol.

Stride RDMA communication can be stopped by setting 0 as the value of the MCA parameter pml_ob1_use_stride_rdma. Refer to "Table 4.47 pml_ob1_use_stride_rdma (use of Stride RDMA communication)" for information on the MCA parameter pml_ob1_use_stride_rdma.

Example

Extract from a program where Stride RDMA communication is applied

```
count = 4;
blength = 16384;
stride = blength * 2;

MPI_Type_vector(count, blength, stride, MPI_BYTE, &newtype);
MPI_Type_commit(&newtype);

if (myrank ==0) {
    MPI_Send(sendbuf, 1, newtype, 1, tag, MPI_COMM_WORLD);
    MPI_Recv(recvbuf, 1, newtype, 1, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
}
if (myrank == 1) {
    MPI_Recv(recvbuf, 1, newtype, 0, tag, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
    MPI_Send(sendbuf, 1, newtype, 0, tag, MPI_COMM_WORLD);
}
```

6.7 Using Multiple TNIs

FX100 systems are connected by means of Tofu interconnects. A network interface device, known as a TNI (Tofu Network Interface), is deployed at each compute node. FX100 systems have four TNIs at each compute node. The MPI programs allocated to each node use these TNIs, thus enabling highly efficient message send-receive. For MPI programs that have a small number of simultaneous communications, the communication time for small-sized messages can be shortened by reducing the number of TNIs. The upper limit for the number of TNIs to be used can be changed using the MCA parameter common_tofu_max_tnis. Refer to "Table 4.24 common_tofu_max_tnis (changes the upper limit for the number of TNIs to be used)" for information on the MCA parameter common_tofu_max_tnis.

Using multiple TNIs can improve the throughput performance of point-to-point communication by providing more opportunities to use multiple communication paths, even for large messages. The use of multiple communication paths makes it possible to split the message, thus increasing the potential for efficient communication. This is referred to as trunking. Trunking can be enabled by specifying a value of 1 in the MCA parameter common_tofu_use_multi_path. Refer to "Table 4.31 common_tofu_use_multi_path (performs point-to-point communication using multiple communication paths)" for information on using the MCA parameter common_tofu_use_multi_path.

Communication performance throughput can be expected to improve using trunking as described above. However, conversely, the multiple TNIs may be used exclusively for one communication partner even if there are multiple communication partners. This may affect the overall throughput performance when message communication is implemented with the other communication partners. In addition, if the communication path that is to be used is already in use by another communication process, contention occurs and the overall communication performance may decrease instead of improving. Thus, depending on the application program and other communication environment factors, the benefits of trunking may not be realized. Trunking should therefore be used with care.

The number of TNIs that can be used for trunking depends on the number of processes executed in the same node or the MCA parameter common_tofu_max_tnis specification. For example, if 2 is specified as the value for common_tofu_max_tnis, a maximum of 2 TNIs can be used by MPI processes. In this case, 2 is also the maximum number of TNIs that can be used for trunking.

6.8 Reduction Operation Sequence Guarantee in Collective Communication

This system may change the sequence of reduction operations performed by the collective communication functions MPI_Reduce, MPI_Ireduce, MPI_Allreduce, MPI_Iallreduce, MPI_Reduce_scatter, MPI_Ireduce_scatter, MPI_Reduce_scatter_block and MPI_Ireduce_scatter_block. The sequence is changed internally depending on communicator size, message size, rank placement, and other communication conditions in order to optimize the execution time for collective communication. For floating point data, changing the sequence of reduction operations may affect the accuracy of the computed results.

This system provides a function that fixes and guarantees the reduction operation sequence in order to prevent these types of reduction operations from affecting the computed results. To configure this, specify 1 for the MCA parameter coll_base_reduce_commute_safe to guarantee that the reduction operation sequence is always fixed. However, note that when this function is used, the execution time for collective communication will increase because the reduction operation sequence is fixed. Refer to "Table 4.12 coll_base_reduce_commute_safe (guarantees the reduction operation sequence)" for information on the MCA parameter coll_base_reduce_commute_safe.

Normally, the default value of 0 is set for the MCA parameter coll_base_reduce_commute_safe. Use of the default settings as much as possible is recommended, except when it is necessary to take precautions with the reduction operation sequence changing.

Note that the MPI_Op_create function can be used in an MPI program to achieve a result that is equivalent to using this sequence guarantee function. For example, if false is specified as the commute argument of the MPI_Op_create function, a new non-commute operation is defined and this new operation is used to perform reduction operations, giving an equivalent result to that of the sequence guarantee function.

Moreover, when you want to specify the reduction operation order expressly, collect the data used for the operation in a specific rank, use the MPI_Reduce_local function, and execute while specifying the order.

6.9 Process Creation in MPI Program

In this system, if you use system call or system library functions to create processes without using MPI functions in your MPI program, there are the following restrictions.

- Data allocated into the data section in the shared object created by user can't be transferred with MPI functions

Data allocated into the data section (".data" section) include data object in common block (Fortran) and initialized data object.

6.10 Suppressing Memory Usage

When an MPI program is executed, this system internally secures the memory, such as receive buffers in each parallel process, required by the MPI library itself. For each parallel process, memory must be secured for each of the communication partner processes. In order to avoid unnecessary allocation of memory, this system only secures memory for a communication partner process at the time of the first communication with that process. In this manual, this method is referred to as dynamic connection. Use of uses dynamic connection enables memory usage to be suppressed to a certain extent. However, even when the dynamic connection mode is used and the memory usage is low immediately after execution of the MPI_Init function, actual memory usage increases if the number of communication partner processes increases during execution of the MPI program.

This section describes methods for minimizing the amount of memory used by this system.

6.10.1 Switching between Fast Communication Mode and Memory-Saving Communication Mode

In order to suppress memory usage as much as possible without greatly hindering overall communication performance, this system provides two communication modes, the fast communication mode and the memory-saving communication mode. This system distinguishes internally between these two communication modes for each of the communication partner processes.

The fast communication mode uses a comparatively large receive buffer, called the Large receive buffer, and an additional receive buffer, called the Small receive buffer, and performs communication as fast as possible. In contrast, the memory-saving communication mode uses only a comparatively small receive buffer, called the Medium receive buffer, and communicates in a way that uses the smallest possible amount of memory. In other words, the memory-saving communication mode can suppress the communication memory usage even though it sacrifices a small degree of communication performance. It is important to achieve a good balance in the way these two communication modes are used by, for example, using the fast communication mode for communications that are performed frequently and using the memory-saving communication mode for communications that are infrequent.

At the time of execution, the MPI program communication patterns determine the communication partner processes with which fast communication mode is used. Normally, initial communication with any communication partner process is performed in memory-saving communication mode. When the communication count with that process is reached the standard value, subsequent communication with that process is switched from the memory-saving to the fast communication mode. However, an upper limit can be set for the number of processes that use fast communication mode. When the number of processes that have been switched to use the fast communication mode reaches this limit, no further switching is performed.

Normally, 1024 is set as the upper limit for the number of processes that use the fast communication mode. This upper limit can be changed using the MCA parameter common_tofu_max_fastmode_procs. This MCA parameter setting method can also be used to perform all communication in the fast communication mode. Alternatively, it can be used to set that all communication be performed in the memory-saving communication mode. Refer to "Table 4.23 common_tofu_max_fastmode_procs (changes the upper limit for the number of processes that can communicate in fast communication mode)" for information on the MCA parameter common_tofu_max_fastmode_procs.

Normally, 16 is set as the communication count standard value that is the condition for switching from the memory-saving communication mode to the fast communication mode. This standard value can be changed using the MCA parameter common_tofu_fastmode_threshold. However, this value includes the count of control communications executed in the MPI library. Therefore this switching may occur before calling MPI functions fewer times than the specified value. Refer to "Table 4.21 common_tofu_fastmode_threshold (changes the conditions for switching to fast communication mode)" for information on the MCA parameter common_tofu_fastmode_threshold.

The sizes of the Large receive buffer and the Medium receive buffer can be changed using the MCA parameters common_tofu_large_recv_buf_size and common_tofu_medium_recv_buf_size. Refer to "Table 4.22 common_tofu_large_recv_buf_size (changes the size of the Large receive buffer)" and "Table 4.25 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)" for information on the MCA parameters common_tofu_large_recv_buf_size and common_tofu_medium_recv_buf_size respectively.

If there are a large number of parallel processes, it is important to use these MCA parameters to tune the system according to the amount of memory required for the MPI program itself and the desired communication performance. Refer to "6.11 Memory Usage Estimation Formulae and Tuning Guidelines" for details.

6.10.2 Influence of Dynamic Connection on Performance

The dynamic connection built in this system executes some processes including control communications at the time of the first communication with that process. Therefore, the first communication time is longer than subsequent usual communication time.

For example, if an MPI program has a loop including a code that performs a broadcast changing root rank with each cycle, without using MPI_Bcast/MPI_Ibcast function, the root rank in each cycle of the loop becomes bottleneck and communication performance may come down significantly. In that case, all the code that performs the first communication with each of the communication partner processes should be moved out of (and before) the loop. Such significant communication performance degradation may be prevented.

6.11 Memory Usage Estimation Formulae and Tuning Guidelines

Each job has a limit in size of usable memory on each compute node, and the size of memory which an MPI program including MPI library uses on each node cannot exceed the limit. In case an MPI program is not likely to be executed under the memory limitation, program tuning is required from the efficiency viewpoint on memory usage.

This section describes memory usage estimation formulae, tuning guidelines, and the specification of restriction values.

6.11.1 Memory Usage Estimation Formulae

The MPI library memory usage for a specific MPI process can be estimated using the formula shown in "Figure 6.5 Estimation formula". However, note the following:

- Because the memory management of operating system is done per page, more memory than the estimated result may be required depending on page size.
- Use the estimation formula shown in the figure below if the number of parallel processes in the MPI program is about 200 or more. If the number of parallel processes is less than this, note that there will be large errors in the calculated results, so they do not make good references. Also note that the values obtained are not necessarily accurate even if the number of parallel processes is over 200.

- The estimation formula shown in the figure below cannot be used if the MPI program uses one-sided communication, if dynamic process generation is being used, or if communication is established between MPI process groups that do not share a communicator.

"Table 6.13 Variables in memory usage estimation formulae" shows the meanings of the variables in this estimation formula.

Figure 6.5 Estimation formula

$$
\begin{aligned}
& C_{\mathrm{Proc}} \times N_{\mathrm{Proc}} + C_{\mathrm{Base}} \\
& + (B_{\mathrm{Large}} + B_{\mathrm{Small}}) \times N_{\mathrm{FastPeer}} \times K_{\mathrm{Buffer}} \\
& + B_{\mathrm{Medium}} \times (N_{\mathrm{Peer}} - N_{\mathrm{FastPeer}}) \times K_{\mathrm{Buffer}} \\
& + B_{\mathrm{Loopback}} \times N_{\mathrm{Loopback}} \\
& + \sum_{\mathrm{communicator}} (C_{\mathrm{Member}} \times N_{\mathrm{Member}} + C_{\mathrm{PeerMember}} \times N_{\mathrm{PeerMember}}) \\
& + B_{\mathrm{UnexpectedMessage}}
\end{aligned}
$$

The first line in the estimation formula shown in the Figure above indicates the minimum amount of memory consumed by any MPI program. This is the amount of memory secured when execution of the MPI program is started and the MPI_Init function is called. This value depends on the total number of processes.

The value in the second line through the fourth line in the estimation formula shown in the Figure above changes depending on the total number of communication partner processes, the number of processes communicating in the fast communication mode, and whether the process has MPI communications with itself or not. This value increases at the point when communication with a new communication partner process is first performed.

The fifth line in the estimation formula shown in the figure above indicates the sum of the memory amounts consumed by each communicator. These communicators include MPI_COMM_WORLD. The value increases at the point when each communicator is created and when a communication is issued by that communicator.

The sixth line in the estimation formula shown in the Figure above becomes larger when an unexpected message is issued. An unexpected message is a message for which invocation of a receive function, such as the MPI_Recv function, in response to a send function, such as the MPI_Send function, is delayed. The receive side process uses memory to temporarily save the received message.

Table 6.13 Variables in memory usage estimation formulae

Variable	Meaning of variable	Explanation of variable	Value
N_{Proc}	Total number of processes	The number of processes belonging to the communicator MPI_COMM_WORLD including that process.	
N_{Peer}	Number of communication partner processes	The number of communication partner processes of that MPI process. The value is 0 immediately after the MPI_Init function is called, but the value increases at the point of the first communication with a new communication partner process. This value includes processes performed by communication internally in the MPI library during collective communication or similar, not just the number of processes performed by point-to-point communication coded in the MPI program.	

Variable	Meaning of variable	Explanation of variable	Value
N_{FastPeer}	Number of communication partner processes using fast communication mode	Out of the number of communication partner processes, the number of processes that use the fast communication mode. If the memory-saving communication mode is not used, this value is the same as the number of communication partner processes. The value specified in the MCA parameter common_tofu_max_fastmode_procs becomes the upper limit value.	
N_{Loopback}	Whether the process has MPI communications with itself or not	The value is 1 if the process has MPI communication with itself. The value is 0 immediately after the MPI_Init function is called, but the value becomes 1 at the point of the first communication with itself.	
N_{Member}	Number of processes belonging to that communicator	The size of that communicator. The value varies for different communicators.	
$N_{\mathrm{PeerMember}}$	Number of communication partner processes in that communicator	Number of communication partner processes that communicate using that communicator. The value varies for different communicators. The value is 0 immediately after the communicator is created, but the value increases at the point of the first communication with a new communication partner process. This value includes processes performed by communication internally in the MPI library during collective communication or similar, not just the number of processes performed by point-to-point communication coded in the MPI program. If multiple communicators are used for communication with a particular MPI process, the values for those communicators are added.	
B_{Large}	Size of Large receive buffer	The value specified in the MCA parameter common_tofu_large_recv_buf_size	Default value: 1MiB
B_{Small}	Size of Small receive buffer	A constant	32KiB
B_{Medium}	Size of Medium receive buffer	The value specified in the MCA parameter common_tofu_medium_recv_buf_size	Default value: 2KiB
B_{Loopback}	Size of buffer for communication with the process itself	A constant	4MiB
$B_{\mathrm{UnexpectedMessage}}$	Quantity of unexpected messages	Increases with each unexpected message that is stored	
K_{Buffer}	Efficiency of memory allocation	A constant	1.0
C_{Base}	Coefficient	A constant determined in accordance with the number of processes per node (referred to as "ppn" in the next column)	1ppn: 53MiB 2ppn: 52MiB 3-4ppn: 52MiB 5-8ppn: 30MiB 9-16ppn: 19MiB 17-32ppn: 15MiB
C_{Proc}	Coefficient	A constant determined in accordance with the number of processes per node (referred to as "ppn" in the next column)	1ppn: 1640B 2ppn: 1848B 3-4ppn: 1704B

Variable	Meaning of variable	Explanation of variable	Value
			5-8ppn: 1224B 9-16ppn: 944B 17-32ppn: 904B
C_{Member}	Coefficient	A constant	336B
$C_{PeerMember}$	Coefficient	A constant. This is 0 if Hasty Rendezvous communication is disabled. This is 256 if Hasty Rendezvous communication is enabled.	0 or 256

6.11.2 Memory Usage Tuning Guidelines

Tuning considerations vary for different MPI program patterns. Table below shows the tuning issues to be considered for the various patterns.

Table 6.14 Tuning guidelines

Pattern	Tuning considerations
The number of processes which require communication performance is fewer compared with the number of all communication partner processes	Use the MCA parameter common_tofu_max_fastmode_procs to set the upper limit for the number of processes that use the fast communication mode for communications. However, consider the performance of communication with the communication partners (processes) that are unable to switch to fast communication mode after the upper limit for the number of processes that can use the fast communication mode for communication has been reached. If the processes to which the fast communication mode and the memory-saving communication mode are assigned are not as anticipated, use the MCA parameter common_tofu_fastmode_threshold to adjust the number at which communication switches from the memory-saving communication mode to the fast communication mode.
Almost all communication partner processes require equal communication performance	Use the MCA parameter common_tofu_large_recv_buf_size to adjust the size of the Large receive buffer for the fast communication mode. However, note that communication performance may deteriorate uniformly if the data size ranges from several KiB to several 10s of KiBs.
If the above measures are insufficient	In addition to tuning using the MCA parameters common_tofu_max_fastmode_procs and common_tofu_large_recv_buf_size, use the MCA parameter common_tofu_medium_recv_buf_size to adjust the size of the Medium receive buffer for the memory-saving communication mode.

6.11.3 Specifying Memory Allocation Restriction Values

As described in "6.11.2 Memory Usage Tuning Guidelines", when using the memory-saving communication mode, the upper limit for the number of communication partner processes using the fast communication mode must be specified in MCA parameter common_tofu_max_fastmode_procs. However, just specifying this upper limit may not be sufficient to achieve both performance and memory allocation. If not, the reception buffer size must also be specified.

Using a different approach, if the user knows the amount of memory used by the MPI program itself, and if the MPI library can operate in the remaining memory range, the user need not calculate a value for the above MCA parameter. In practice, the memory allocation that the MPI library is allowed to use can be specified. If the user specifies this limit, the system automatically tunes the MCA parameters internally and, as much as possible, operates within the specified range of restriction values for memory allocations. The actual memory allocation when an MPI program is executed will vary depending on the MPI functions used within the MPI program, the number of parallels, the execution method, and so on. Operation is not necessarily possible within the specified range of restriction values for memory allocations. Refer to the notes in "6.11.3.3 Notes on Execution When Memory Allocation Restriction Values are Specified" that apply to your usage circumstances.

Note that memory allocation restriction values cannot be specified if dynamic process generation is used or if communication is being established between MPI process groups that do not share a communicator. If specified, the behavior is unpredictable.

6.11.3.1 Specification Memory Allocation Restriction Values

In practice, memory allocation restrictions are enabled if at least one value is specified in MCA parameter common_tofu_memory_limit. The value specified in this MCA parameter is interpreted as being the restriction value (MiB) for the memory allocation that can be used by the MPI library, and other MCA parameters are tuned automatically. At this time, the value specified in MCA parameter common_tofu_memory_limit_peers is used as the number of communication partner processes. If the MCA parameter common_tofu_memory_limit_peers is not specified, the number of processes belonging to the same communicator MPI_COMM_WORLD is used as the number of communication partner processes. Refer to "Table 4.26 common_tofu_memory_limit (specifies the memory allocation limit value)" and "Table 4.27 common_tofu_memory_limit_peers (specifies the assumed number of communication partner processes when the memory allocation is limited)" for information on the MCA parameters common_tofu_memory_limit and common_tofu_memory_limit_peers respectively.

Note that if the MPI library for debug is being used, operations do not necessarily conform to the specified values and, for example, more memory than the specified restriction values for memory allocations may be used.

6.11.3.2 MCA Parameters Targeted by Automatic Tuning

Table below shows the MCA parameters targeted for automatic tuning.

Table 6.15 MCA parameters tuned automatically in accordance with memory allocation restriction values

MCA parameter	Description	Priority
common_tofu_max_fastmode_procs	Upper limit for the number of communication partner processes that use the fast communication mode	1
common_tofu_large_recv_buf_size	Large receive buffer size	2
common_tofu_medium_recv_buf_size	Medium receive buffer size	3

Note: Smaller priority values indicate a higher priority.

If automatic tuning of specified memory allocations can be achieved just by using the highest priority MCA parameter, tuning is not required for the lower priority MCA parameters and, therefore, the default values remain unchanged.

Regardless of whether a value of 1 or higher is specified in MCA parameter common_tofu_memory_limit, if two or fewer of the MCA parameters shown in the table above are specified simultaneously, the specified values are enabled as is for the specified MCA parameters. Then, the remaining unspecified MCA parameters in the table above are the only parameters tuned automatically, in order of highest priority (smallest numerical value).

If the three MCA parameters shown in the table above and the MCA parameter common_tofu_memory_limit are specified simultaneously, automatic tuning is not performed.

Assume, for example, that the maximum number of partner processes that one process communicates with is known from MPI statistical information, and that the MCA parameters common_tofu_memory_limit, common_tofu_memory_limit_peers, and common_tofu_max_fastmode_procs are specified. In this case, the MCA parameter common_tofu_large_recv_buf_size value is tuned automatically first. If this is insufficient, the MCA parameter common_tofu_medium_recv_buf_size is then also tuned automatically.

6.11.3.3 Notes on Execution When Memory Allocation Restriction Values are Specified

Automatic tuning is performed by calculating back from the estimation expressions in "Figure 6.5 Estimation formula". Therefore, note the following when executing MPI programs in which memory allocation restriction values are specified.

The minimum memory allocation required by the MPI library of this system varies in accordance with the number of parallel processes and other conditions. Therefore, the memory allocation restriction values specified by the user may be exceeded.

For example, if unexpected messages are issued, these are saved within the MPI library, and may therefore affect the amount of memory used. MPI library automatic tuning does not consider the likelihood of unexpected messages because the number of unexpected messages issued during execution of an MPI program cannot be known. This means that the memory allocation restriction values specified by the user may be exceeded for MPI programs that issue a large number of unexpected messages.

Additional memory is also used if automatic process generation is performed, if communicators are created, if communications are issued using Hasty Rendezvous, and so on. These allocations are also excluded from the calculations performed internally during automatic tuning because the system cannot know these memory allocations in advance. Note that in these cases too, the memory allocation restriction values specified by the user may be exceeded.

6.12 Use of Tofu Barrier Communication to Increase Speeds

When the functions MPI_Barrier, MPI_Bcast, MPI_Reduce, and MPI_Allreduce are executed, faster communication speeds can be achieved by using Tofu barrier communication provided by this system as a Tofu interconnect hardware function.

This section describes the conditions that apply to Tofu barrier communication for each of the MPI functions, and gives notes about this communication.

6.12.1 MPI_Barrier Function

The MPI_Barrier function can apply the Tofu barrier communication function if all the following conditions are met:

- The communicator is a communicator in a group (intra-communicator).
- If multiple processes are allocated within one node and the communicator is a sub-communicator, the communicator meets all the following conditions:
  - The communicator is a sub-communicator created directly from MPI_COMM_WORLD.
  - The communicator is a sub-communicator created by the MPI_Comm_split function.
  - The communicator is a sub-communicator created with color=0.
- A total of four or more compute nodes are allocated to the communicator.
- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.

6.12.2 MPI_Bcast Function

The MPI_Bcast function can apply the Tofu barrier communication function if all the following conditions are met:

- MCA parameter coll_tbi_use_on_bcast is not set to 0.
- The communicator is a communicator in a group (intra-communicator).
- If multiple processes are allocated within one node and the communicator is a sub-communicator, the communicator meets all the following conditions:
  - The communicator is a sub-communicator created directly from MPI_COMM_WORLD.
  - The communicator is a sub-communicator created by the MPI_Comm_split function.
  - The communicator is a sub-communicator created with color=0.
- A total of four or more compute nodes are allocated to the communicator.
- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.
- The number of message requests is 1.
- The message size is 8 bytes or less.
- The message datatype is one of the following:
  - Basic datatypes
  - A derived datatypes that can be derived from one component element of a basic datatype

If the type signature of the send side of MPI_Bcast function is different from that of the receive side, that is incorrect according to the MPI standard. The MPI program may not work correctly, such as abnormal end, because Tofu barrier cannot be used correctly. This is a problem that causes inconsistency in the environment of Tofu barrier communication because any of conditions above is not met on either side. Originally, such a program is incorrect according to the MPI standard, but you can avoid using Tofu barrier when executing the MPI program including MPI_Bcast by setting 0 in the MCA parameter coll_tbi_use_on_bcast value.

Refer to "Table 4.14 coll_tbi_use_on_bcast (uses Tofu barrier communication in MPI_Bcast function)" for information on the MCA parameter coll_tbi_use_on_bcast.

6.12.3 MPI_Reduce Function and MPI_Allreduce Function

The MPI_Reduce function and the MPI_Allreduce function can apply the Tofu barrier communication function if all the following conditions are met:

- The communicator is a communicator in a group (intra-communicator).
- If multiple processes are allocated within one node and the communicator is a sub-communicator, the communicator meets all the following conditions:
    - The communicator is a sub-communicator created directly from MPI_COMM_WORLD.
    - The communicator is a sub-communicator created by the MPI_Comm_split function.
    - The communicator is a sub-communicator created with color=0.
- A total of four or more compute nodes are allocated to the communicator.
- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.
- The MCA parameter coll_base_reduce_commute_safe is not set to guarantee the sequence of reduction operations.
- The number of message requests is 1.
- The message datatype is one of the following:
    - Basic datatypes
    - A derived datatypes that can be derived from one component element of a basic datatype
- The reduction operation, datatype (if a derived datatype, the component element datatype), and the size combine in one of the ways shown in "Table 6.16 Operation combinations that allow the MPI_Reduce and MPI_Allreduce functions to apply Tofu barrier communication".

Table 6.16 Operation combinations that allow the MPI_Reduce and MPI_Allreduce functions to apply Tofu barrier communication

MPI predefined operation	Datatypes	Size
[C/Fortran] MPI_MAX MPI_MIN [C++] MPI::MAX MPI::MIN	Integer datatypes (Floating point datatypes (*))	8 bytes or less
[C/Fortran] MPI_SUM [C++] MPI::SUM	Integer datatypes Floating point datatypes	8 bytes or less
	Complex datatypes	8 bytes (4 bytesx2) or 16 bytes (8 bytesx2)
[C/Fortran] MPI_LAND MPI_LOR MPI_LXOR [C++] MPI::LAND MPI::LOR MPI::LXOR	Integer datatypes Logical datatypes	8 bytes or less

MPI predefined operation	Datatypes	Size
[C/Fortran] MPI_BAND MPI_BOR MPI_BXOR [C++] MPI::BAND MPI::BOR MPI::BXOR	Integer datatypes Byte datatypes	8 bytes or less

*1:Tofu barrier communication is applied only when 1 is specified for the value of MCA parameter coll_tbi_use_on_max_min. Refer to "Table 4.16 coll_tbi_use_on_max_min (uses Tofu barrier communication for floating point datatypes MPI_MAX and MPI_MIN)" for details of MCA parameter coll_tbi_use_on_max_min.

6.12.4 Notes on Tofu Barrier Communication

Tofu barrier communication is performed by securing the required number of barrier gates from the multiple Tofu interconnect barrier gates provided in each compute node, and then configuring a barrier network within the corresponding communicator. To configure a barrier network for one occurrence of Tofu barrier communication, one barrier gate that fulfils the role of the start point and end point, and multiple barrier gates that fulfil the role of relay points, must be secured in each node. A maximum total number of 8 barrier gates in each node can be used to perform the start point and end point roles. A maximum total number of 56 barrier gates in each node can be used to perform the relay point role. Note that these maximum values are provided as guidelines, and the total number of barrier gates may be changed by system conditions or future product version upgrades. In addition, if Tofu barrier communication is applied by the MPI_Barrier, MPI_Bcast, MPI_Reduce, or MPI_Allreduce function with N processes in one occurrence of Tofu barrier communication, the current maximum for the number of barrier gates used to fulfil the relay role is about $2\log_2 N$ barrier gates at each node.

If the maximum number of barrier gates in each node is exceeded, the required number of barrier gates will not be able to be secured and Tofu barrier communication will not be able to be used. As a result, the MPI_Barrier, MPI_Bcast, MPI_Reduce, and MPI_Allreduce functions will all be executed by software. Execution by software results in longer communication times compared with when Tofu barrier communication is applied, and so care is required concerning the execution performance aspect.

A barrier network is configured when a communicator or a window is created. The execution time might become longer by barrier networks frequently configured in the MPI program that repeats duplication of communicators by the MPI_Comm_dup function or repeats creation of windows. In such a case, the execution time becomes faster by setting 0 in the MCA parameter coll_tbi_use_on_comm_dup value.

Read "Table 4.15 coll_tbi_use_on_comm_dup (uses Tofu barrier communication for a communicator created by MPI_Comm_dup function)" for more information on the MCA parameter coll_tbi_use_on_comm_dup.

6.12.5 Fast Reduction Operations for Floating Point Type and Complex Type Data within a Node

If the conditions for Tofu barrier communication are met, the Tofu interconnect hardware functions are used for communication between nodes. In this case, hardware functions cannot be used for communication within the same node so this is executed by means of software.

If there are many processes within one node, speeds can be increased by specifying 2 as the value of the MCA parameter coll_tbi_intra_node_reduction. This changes the algorithm used by the floating point type and complex type data reduction operations.

Changing the value specified for this MCA parameter changes the reduction operation sequence, and so the calculation results might be different due to precision error. This fact must be noted if the value specified for the MCA parameter is being changed.

6.13 MPI_Bcast Function When the Same Count is Used among the Processes

In this system, a faster communication mechanism is available when MPI_Bcast function is used with the same count among the processes. This mechanism can be used by specifying 1 for the MCA parameter coll_tuned_bcast_same_count.

But, when this mechanism is used in MPI programs that use MPI_Bcast function with different counts among the processes, a deadlock may be caused in the MPI library. According to the MPI standard, MPI_Bcast function allows to use different datatypes and counts among the processes as long as type signature of datatype and count on any process is equal to that on the root process.

For example, the following condition is valid:

- rank 0: datatype = MPI_INT, count = 2
- rank 1: datatype = derived datatype of two MPI_INTs, count = 1

The default value for the MCA parameter coll_tuned_bcast_same_count is usually set to 0 in order to execute such MPI programs correctly.

If it is guaranteed that MPI_Bcast function is used with the same count among the processes, it is recommended to specify 1 for this parameter for faster communication speeds. Refer to "Table 4.17 coll_tuned_bcast_same_count (achieves faster communication when MPI_Bcast/MPI_Ibcast function is used with the same count among the processes)" for information on the MCA parameter coll_tuned_bcast_same_count.

6.14 Algorithms Tuned with Recognition of Tofu Coordinates

As in the assumed knowledge concerning "6.1 Tofu Interconnect", Tofu coordinates are allocated to compute nodes. If the MCA parameter coll_tuned_use_6d_algorithm is enabled, usable algorithms perform communication based on Tofu coordinate information.

In order to enable use of these algorithms, it is necessary that "the communicator that invokes collective communication is a 6-dimensional rectangle". A 6-dimensional rectangle is defined as being one in which both the following are the same:

- Product of the axis lengths
- Number of nodes belonging to the communicator

For Tofu coordinates within a particular communicator, the length of each axis is defined as being the difference between the minimum value and maximum value of the coordinate plus one. This is explained with reference to the 6-dimensional rectangle examples in "Figure 6.6 Example of 6-dimenional rectangle communicator" and "Figure 6.7 Example of communicator that is not a 6-dimensional rectangle". In each figure, the circles represent compute nodes, and the lines represent links. The colored circles represent compute nodes belonging to the communicator. "Figure 6.6 Example of 6-dimenional rectangle communicator" is an example of a communicator that is a 6-dimensional rectangle. "Figure 6.7 Example of communicator that is not a 6-dimensional rectangle" is an example of a communicator that is not a 6-dimensional rectangle. In both figures, the axis lengths of the X, Y, Z, A, B, and C axes are the same, being 1, 1, 1, 2, 3, and 2. The product of the axis lengths is 12. In "Figure 6.6 Example of 6-dimenional rectangle communicator", the number of nodes belonging to the communicator is 12. However, in "Figure 6.7 Example of communicator that is not a 6-dimensional rectangle" the number of nodes belonging to the communicator is 10. Therefore, "Figure 6.6 Example of 6-dimenional rectangle communicator" is a 6-dimensional rectangle, but "Figure 6.7 Example of communicator that is not a 6-dimensional rectangle" is not.

Refer to the Tofu coordinates to check whether a communicator is a 6-dimensional rectangle or not. Check "5.1 Rank Query Interface" for information on how to reference Tofu coordinates. Read "Table 4.20 coll_tuned_use_6d_algorithm (use of algorithms tuned with recognition of Tofu coordinates)" for details of the MCA parameter coll_tuned_use_6d_algorithm.

Figure 6.6 Example of 6-dimenional rectangle communicator

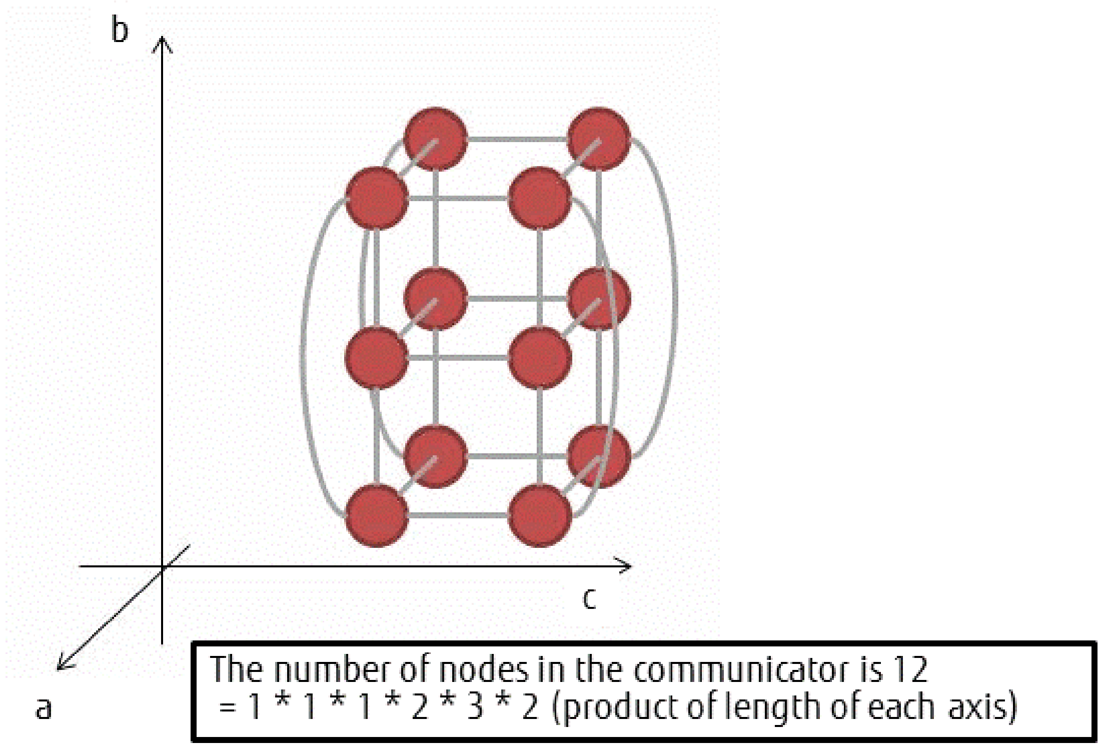
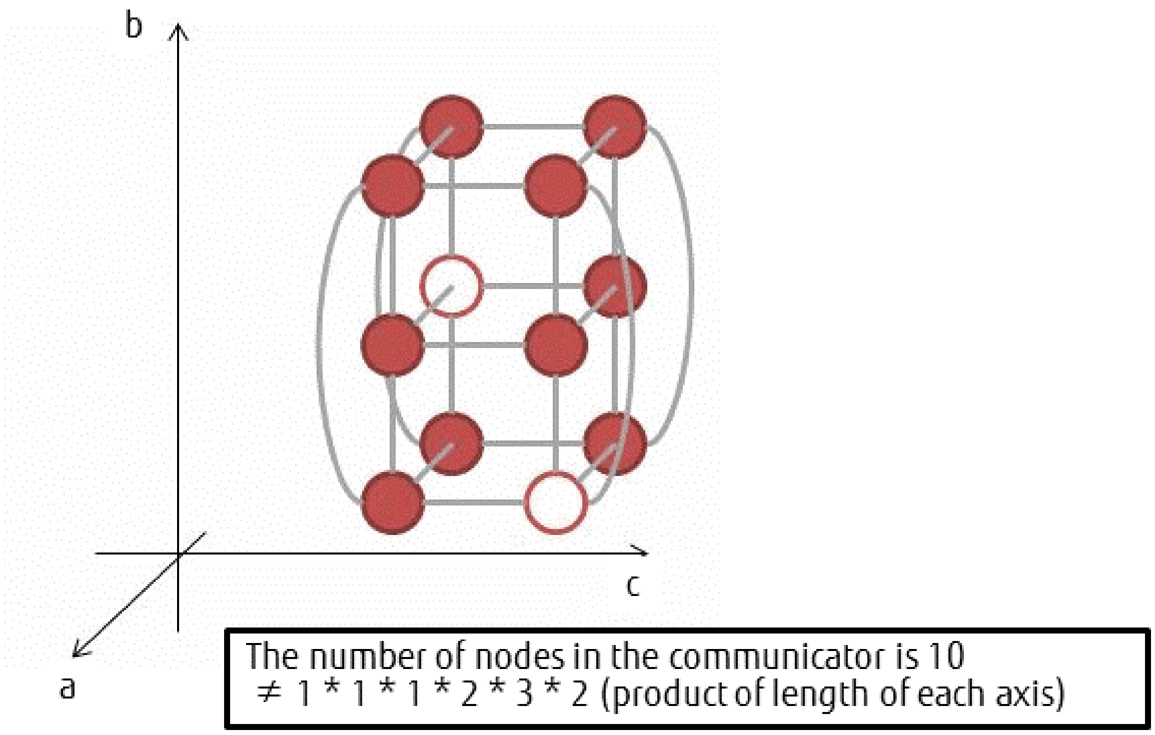

b
c
a
The number of nodes in the communicator is 12
= 1 * 1 * 1 * 2 * 3 * 2 (product of length of each axis)

Figure 6.7 Example of communicator that is not a 6-dimensional rectangle


b
c
a
The number of nodes in the communicator is 10
≠ 1 * 1 * 1 * 2 * 3 * 2 (product of length of each axis)

6.15 MPI Statistical Information

This system can display statistical information concerning MPI communications. In this manual, this information is referred to as MPI statistical information.

There are 2 varieties of a whole output mode and the section specifying output mode in the method of outputting the MPI statistical information.

The output of MPI statistical information can be controlled by the MCA parameter mpi_print_stats. Refer to "Table 4.39 mpi_print_stats (outputs MPI statistical information)" for details.

"Table 6.17 Contents of MPI statistical information output for whole output mode" shows the MPI statistical information that can be output in this system.

If 1 is specified for the MCA parameter mpi_print_stats, maximum values, minimum values, and average values are output for all parallel processes for all output items other than Process Mapping shown in whole output mode "Table 6.17 Contents of MPI statistical information output for whole output mode". For maximum values and minimum values, the corresponding parallel process rank numbers are also output.

If 2 is specified for the MCA parameter mpi_print_stats, information is output for the corresponding parallel processes for all output items other than Process Mapping shown in whole output mode "Table 6.17 Contents of MPI statistical information output for whole output mode". In this case, the parallel processes targeted for MPI statistical information output can be indicated using the MCA parameter mpi_print_stats_ranks. Refer to "Table 4.40 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)" for details on mpi_print_stats_ranks. Note that these statistics are output to the standard error output. To avoid output overlaps when statistics are output for multiple parallel processes, specifications that suit the mpiexec(1) options -of-proc/-std-proc, --of-proc/--std-proc, -oferr-proc/-stderr-proc, or --oferr-proc/--stderr-proc are recommended.

"Table 6.18 Contents of MPI statistical information output for section specifying output mode" shows the contents of section specifying function of MPI statistical information. If a value of MCA parameter mpi_print_stats is equal to 3, all items except "Process Mapping" is output when a value of MCA parameter mpi_print_stats is equal to 1. However, the header part, the body part and the footer part are output separately.

If a value of MCA parameter mpi_print_stats is equal to 4, all items except "Process Mapping" is output when a value of MCA parameter mpi_print_stats is equal to 2. However, the header part, the body part and the footer part are output separately.

Refer to "5.3 MPI Statistical Information Section Specifying Interface" for details.

The Process Mapping output item of MPI statistical information is output only for parallel processes with a rank number of 0, regardless of the value in the above MCA parameter mpi_print_stats_ranks.

When you use section specifying output mode, the specified section is ignored for Connection, Max_Hop and Average_Hop output items of MPI statistical information. The information at the point of an FJMPI_Collection_print function call or an MPI_Finalize function call is always output.

Output item Per-protocol Nonblocking/Persistent Communication Count and Per-protocol Nonblocking/Persistent Communication Count Started in Wait of the MPI statistical information will be a reference when MCA parameter opal_progress_thread_mode is used. Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details.

Table 6.17 Contents of MPI statistical information output for whole output mode

Output title and output item name	Output content
MPI Information	
Dimension	Number of dimensions in the torus structure where the parallel processes belonging to MPI_COMM_WORLD are deployed
Shape	Process shape of the torus structure where the parallel processes belonging to MPI_COMM_WORLD are deployed
MPI Memory Usage	
Estimated_Memory_Size	Estimated MPI library memory allocation Estimated memory allocation value described in "6.11.1 Memory Usage Estimation Formulae"
Per-peer Communication Count	
In_Node	Communication count within node for point-to-point communication
Neighbor	Communication count between neighboring nodes for point-to-point communication
Not_Neighbor	Communication count between non-neighboring nodes for point-to-point communication

Output title and output item name	Output content
Total_Count	Total communication count for point-to-point communication
Connection	Number of connections for Tofu communication
Max_Hop	Maximum number of hops between processes for Tofu communication
Average_Hop	Average number of hops between processes for Tofu communication
Per-peer Transmission size	
In_Node	Transfer data size within node for point-to-point communication
Neighbor	Transfer data size between neighboring nodes for point-to-point communication
Not_Neighbor	Transfer data size between non-neighboring nodes for point-to-point communication
Total_Size	Transfer data size for point-to-point communication
Per-protocol Communication Count	
Eager	Eager communication mode use count at send side for point-to-point communication
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication
Hasty_Rendezvous	Hasty Rendezvous communication mode use count at send side for point-to-point communication
Prequest_Extended_IF	Extended persistent communication requests interface use count at send side for point-to-point communication
Unexpected_Message	Maximum number of messages saved temporarily in the internal buffer during point-to-point communication
Barrier Communication Count	
Tofu	Count for barriers using the Tofu barrier communication function
Soft	Count for software-style barriers executed
Tofu Barrier Collective Communication Count	
Bcast	Invocation count for collective communication MPI_Bcast using the Tofu barrier communication function
Reduce	Invocation count for collective communication MPI_Reduce using the Tofu barrier communication function
Allreduce	Invocation count for collective communication MPI_Allreduce using the Tofu barrier communication function
6D-Tofu-specific Collective Communication Count	
Alltoall	Invocation count for collective communication MPI_Alltoall using the algorithms tuned with recognition of Tofu interconnect 6-dimensional coordinates. This is the count for algorithms tuned with recognition of Tofu coordinates and invoked by enabling the MCA parameter coll_tuned_use_6d_algorithm. Read "Table 4.20 coll_tuned_use_6d_algorithm (use of algorithms tuned with recognition of Tofu coordinates)" for details of the MCA parameter coll_tuned_use_6d_algorithm.
Tofu-specific Collective Communication Count	
Bcast	Count for collective communication MPI_Bcast invocations that specifically invoke algorithms tuned for Tofu interconnect
Reduce	Count for collective communication MPI_Reduce invocations that specifically invoke algorithms tuned for Tofu interconnect
Gather	Count for collective communication MPI_Gather invocations that specifically invoke algorithms tuned for Tofu interconnect

Output title and output item name	Output content
Gatherv	Count for collective communication MPI_Gatherv invocations that specifically invoke algorithms tuned for Tofu interconnect
Allreduce	Count for collective communication MPI_Allreduce invocations that specifically invoke algorithms tuned for Tofu interconnect
Alltoall	Count for collective communication MPI_Alltoall invocations that specifically invoke algorithms tuned for Tofu interconnect
Alltoallv	Count for collective communication MPI_Alltoallv invocations that specifically invoke algorithms tuned for Tofu interconnect
Allgather	Count for collective communication MPI_Allgather invocations that specifically invoke algorithms tuned for Tofu interconnect
Allgatherv	Count for collective communication MPI_Allgatherv invocations that specifically invoke algorithms tuned for Tofu interconnect
Non-Tofu-specific Collective Communication Count	
Bcast	Count for collective communication MPI_Bcast invocations that could not specifically use algorithms tuned for Tofu interconnect
Reduce	Count for collective communication MPI_Reduce invocations that could not specifically use algorithms tuned for Tofu interconnect
Gather	Count for collective communication MPI_Gather invocations that could not specifically use algorithms tuned for Tofu interconnect
Gatherv	Count for collective communication MPI_Gatherv invocations that could not specifically use algorithms tuned for Tofu interconnect
Allreduce	Count for collective communication MPI_Allreduce invocations that could not specifically use algorithms tuned for Tofu interconnect
Alltoall	Count for collective communication MPI_Alltoall invocations that could not specifically use algorithms tuned for Tofu interconnect
Alltoallv	Count for collective communication MPI_ Alltoallv invocations that could not specifically use algorithms tuned for Tofu interconnect
Allgather	Count for collective communication MPI_Allgather invocations that could not specifically use algorithms tuned for Tofu interconnect
Allgatherv	Count for collective communication MPI_Allgatherv invocations that could not specifically use algorithms tuned for Tofu interconnect
Per-protocol Nonblocking/Persistent Communication Count	
Eager	Eager communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request
Hasty_Rendezvous	Hasty Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request
Collective	Nonblocking collective operations use count
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	
Eager	Eager communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request and the transfer of the message body started when any of MPI_Wait, MPI_Waitany, MPI_Waitall, or MPI_Waitsome function is called
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request and the transfer of the

Output title and output item name	Output content
	message body started when any of MPI_Wait, MPI_Waitany, MPI_Waitall, or MPI_Waitsome function is called
Hasty_Rendezvous	Hasty Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request and the transfer of the message body started when any of MPI_Wait, MPI_Waitany, MPI_Waitall, or MPI_Waitsome function is called
Process Mapping	
	List of rank numbers and coordinate correspondences of all parallel processes belonging to MPI_COMM_WORLD However, this information is output for only parallel processes having rank number 0.

Example

Example of MPI statistical information output when 1 is specified as the value of MCA parameter mpi_print_stats

```
===================================================================
/***************** MPI Statistical Information *****************/
===================================================================

------------------------- MPI Information -------------------------
Dimension                             3
Shape                             2x3x4

---------------------- MPI Memory Usage (MiB) ---------------------
                              MAX              MIN              AVE
Estimated_Memory_Size       93.90 [    0]     44.39 [    1]     47.29

------------------- Per-peer Communication Count ------------------
                              MAX              MIN              AVE
In_Node                      1024 [    0]         0 [    1]     512.0
Neighbor                     3072 [    1]         0 [    8]    1621.3
Not_Neighbor                 3072 [   11]         0 [    0]     938.7
Total_Count                  3072 [    0]      3072 [    0]    3072.0
Connection                     46 [    0]         9 [    4]      11.8
Max_Hop                         4 [    0]         2 [    4]       3.1
Average_Hop                  2.27 [   35]      1.60 [    6]      1.84

----------------- Per-peer Transmission Size (MiB) ----------------
                              MAX              MIN              AVE
In_Node                    256.00 [    0]      0.00 [    1]    128.00
Neighbor                   768.00 [    1]      0.00 [    8]    405.33
Not_Neighbor               768.00 [   11]      0.00 [    0]    234.67
Total_Size                 768.00 [    0]    768.00 [    0]    768.00

----------------- Per-protocol Communication Count ----------------
                              MAX              MIN              AVE
Eager                           0 [    0]         0 [    0]       0.0
Rendezvous                   3072 [    0]      3072 [    0]    3072.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0
Persistent_Extended_IF          0 [    0]         0 [    0]       0.0
Unexpected_Message              1 [    0]         1 [    0]       1.0

------------------- Barrier Communication Count -------------------
                              MAX              MIN              AVE
Tofu                         8217 [    0]      8217 [    0]    8217.0
Soft                            1 [    0]         1 [    0]       1.0
```

```
----------- Tofu Barrier Collective Communication Count -----------
                              MAX              MIN              AVE
Bcast                           0 [    0]        0 [    0]       0.0
Reduce                          0 [    0]        0 [    0]       0.0
Allreduce                      32 [    0]       32 [    0]      32.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                              MAX              MIN              AVE
Alltoall                        0 [    0]        0 [    0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX              MIN              AVE
Bcast                         512 [    0]      512 [    0]     512.0
Reduce                          0 [    0]        0 [    0]       0.0
Gather                          8 [    0]        8 [    0]       8.0
Gatherv                         0 [    0]        0 [    0]       0.0
Allreduce                       0 [    0]        0 [    0]       0.0
Alltoall                     1024 [    0]     1024 [    0]    1024.0
Alltoallv                     512 [    0]      512 [    0]     512.0
Allgather                       0 [    0]        0 [    0]       0.0
Allgatherv                      0 [    0]        0 [    0]       0.0

--------- Non-Tofu-specific Collective Communication Count --------
                              MAX              MIN              AVE
Bcast                           0 [    0]        0 [    0]       0.0
Reduce                        128 [    0]      128 [    0]     128.0
Gather                          0 [    0]        0 [    0]       0.0
Gatherv                         4 [    0]        4 [    0]       4.0
Allreduce                       0 [    0]        0 [    0]       0.0
Alltoall                        0 [    0]        0 [    0]       0.0
Alltoallv                       0 [    0]        0 [    0]       0.0
Allgather                     128 [    0]      128 [    0]     128.0
Allgatherv                    256 [    0]      256 [    0]     256.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
                              MAX              MIN              AVE
Eager                           0 [    0]        0 [    0]       0.0
Rendezvous                      0 [    0]        0 [    0]       0.0
Hasty_Rendezvous                0 [    0]        0 [    0]       0.0
Collective                      0 [    0]        0 [    0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX              MIN              AVE
Eager                           0 [    0]        0 [    0]       0.0
Rendezvous                      0 [    0]        0 [    0]       0.0
Hasty_Rendezvous                0 [    0]        0 [    0]       0.0

------------------------- Process Mapping -------------------------
(0,0,0)                    0,1
(1,0,0)                    2,3
(0,1,0)                    4,5
(1,1,0)                    6,7
(0,2,0)                    8,9
(1,2,0)                    10,11
(0,0,1)                    12,13
(1,0,1)                    14,15
(0,1,1)                    16,17
(1,1,1)                    18,19
(0,2,1)                    20,21
(1,2,1)                    22,23
(0,0,2)                    24,25
(1,0,2)                    26,27
```

```
(0,1,2)                 28,29
(1,1,2)                 30,31
(0,2,2)                 32,33
(1,2,2)                 34,35
(0,0,3)                 36,37
(1,0,3)                 38,39
(0,1,3)                 40,41
(1,1,3)                 42,43
(0,2,3)                 44,45
(1,2,3)                 46,47
```

Table 6.18 Contents of MPI statistical information output for section specifying output mode

Output title and output item name	Output content
Header part **Content output by FJMPI_Collection_print execution time of the first time point**	
MPI Information	
Dimension	*Refer to a whole output mode
Shape	
Body part Content output with FJMPI_Collection_print run unit	
Section	
Time(Sec)	Execution time at each section specifying (second)
Per-peer Communication Count	
In_Node	*Refer to a whole output mode
Neighbor	
Not_Neighbor	
Total_Count	
Connection	
Max_Hop	
Average_Hop	
Per-peer Transmission size	
In_Node	*Refer to a whole output mode
Neighbor	
Not_Neighbor	
Total_Size	
Per-protocol Communication Count	
Eager	*Refer to a whole output mode
Rendezvous	
Hasty_Rendezvous	
Persistent_Extended_IF	
Unexpected_Message	
Barrier Communication Count	
Tofu	*Refer to a whole output mode
Soft	

Output title and output item name	Output content
Tofu Barrier Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Allreduce	
6D-Tofu-specific Collective Communication Count	
Alltoall	*Refer to a whole output mode
Tofu-specific Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	
Allgatherv	
Non-Tofu-specific Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	
Allgatherv	
Per-protocol Nonblocking/Persistent Communication Count	
Eager	*Refer to a whole output mode
Rendezvous	
Hasty_Rendezvous	
Collective	
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	
Eager	*Refer to a whole output mode
Rendezvous	
Hasty_Rendezvous	
Footer part Content output when MPI_Finalize is executed	
MPI Memory Usage	
Estimated_Memory_Size	*Refer to a whole output mode

Output title and output item name	Output content
Process Mapping	
	List of rank numbers and coordinate correspondences of all parallel processes belonging to MPI_COMM_WORLD However, this information is output for only parallel processes having rank number 0.

Example

Example of MPI statistical information output when 3 is specified as the value of MCA parameter mpi_print_stats

```
===================================================================
/***************** MPI Statistical Information *****************/
===================================================================

------------------------- MPI Information -------------------------
Dimension                        3
Shape                        2x3x4

----------- Section    1 My message (Execute section1)  -----------
                              MAX              MIN              AVE
Time(Sec)                  191.82 [  25]    191.65 [   0]    191.78

------------------- Per-peer Communication Count ------------------
                              MAX              MIN              AVE
In_Node                       512 [   0]         0 [   1]     256.0
Neighbor                     1536 [   1]         0 [   8]     810.7
Not_Neighbor                 1536 [  11]         0 [   0]     469.3
Total_Count                  1536 [   0]      1536 [   0]    1536.0
Connection                     46 [   0]         9 [   4]      11.8
Max_Hop                         4 [   0]         2 [   4]       3.1
Average_Hop                  2.27 [  35]      1.60 [   6]      1.84

----------------- Per-peer Transmission Size (MiB) ----------------
                              MAX              MIN              AVE
In_Node                    128.00 [   0]      0.00 [   1]     64.00
Neighbor                   384.00 [   1]      0.00 [   8]    202.67
Not_Neighbor               384.00 [  11]      0.00 [   0]    117.33
Total_Size                 384.00 [   0]    384.00 [   0]    384.00

----------------- Per-protocol Communication Count ----------------
                              MAX              MIN              AVE
Eager                           0 [   0]         0 [   0]       0.0
Rendezvous                   1536 [   0]      1536 [   0]    1536.0
Hasty_Rendezvous                0 [   0]         0 [   0]       0.0
Persistent_Extended_IF          0 [   0]         0 [   0]       0.0
Unexpected_Message              1 [   0]         1 [   0]       1.0

------------------- Barrier Communication Count -------------------
                              MAX              MIN              AVE
Tofu                         4108 [   0]      4108 [   0]    4108.0
Soft                            0 [   0]         0 [   0]       0.0

----------- Tofu Barrier Collective Communication Count -----------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Allreduce                      16 [   0]        16 [   0]      16.0

--------- 6D-Tofu-specific Collective Communication Count ---------
```

```
                              MAX              MIN              AVE
Alltoall                        0 [    0]        0 [    0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX              MIN              AVE
Bcast                         256 [    0]      256 [    0]     256.0
Reduce                          0 [    0]        0 [    0]       0.0
Gather                          4 [    0]        4 [    0]       4.0
Gatherv                         0 [    0]        0 [    0]       0.0
Allreduce                       0 [    0]        0 [    0]       0.0
Alltoall                      512 [    0]      512 [    0]     512.0
Alltoallv                     256 [    0]      256 [    0]     256.0
Allgather                       0 [    0]        0 [    0]       0.0
Allgatherv                      0 [    0]        0 [    0]       0.0
--------- Non-Tofu-specific Collective Communication Count --------
                              MAX              MIN              AVE
Bcast                           0 [    0]        0 [    0]       0.0
Reduce                         64 [    0]       64 [    0]      64.0
Gather                          0 [    0]        0 [    0]       0.0
Gatherv                         8 [    0]        8 [    0]       8.0
Allreduce                       0 [    0]        0 [    0]       0.0
Alltoall                        0 [    0]        0 [    0]       0.0
Alltoallv                       0 [    0]        0 [    0]       0.0
Allgather                      64 [    0]       64 [    0]      64.0
Allgatherv                    128 [    0]      128 [    0]     128.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
                              MAX              MIN              AVE
Eager                           0 [    0]        0 [    0]       0.0
Rendezvous                      0 [    0]        0 [    0]       0.0
Hasty_Rendezvous                0 [    0]        0 [    0]       0.0
Collective                      0 [    0]        0 [    0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX              MIN              AVE
Eager                           0 [    0]        0 [    0]       0.0
Rendezvous                      0 [    0]        0 [    0]       0.0
Hasty_Rendezvous                0 [    0]        0 [    0]       0.0

----------- Section    2 My message (Execute section2)  -----------
                              MAX              MIN              AVE
Time(Sec)                   14.35 [   34]    14.35 [    0]     14.35

------------------ Per-peer Communication Count ------------------
                              MAX              MIN              AVE
In_Node                       512 [    0]        0 [    1]     256.0
Neighbor                     1536 [    1]        0 [    8]     810.7
Not_Neighbor                 1536 [   11]        0 [    0]     469.3
Total_Count                  1536 [    0]     1536 [    0]    1536.0
Connection                     46 [    0]        9 [    4]      11.8
Max_Hop                         4 [    0]        2 [    4]       3.1
Average_Hop                  2.27 [   35]     1.60 [    6]      1.84

---------------- Per-peer Transmission Size (MiB) ----------------
                              MAX              MIN              AVE
In_Node                    128.00 [    0]     0.00 [    1]     64.00
Neighbor                   384.00 [    1]     0.00 [    8]    202.67
Not_Neighbor               384.00 [   11]     0.00 [    0]    117.33
Total_Size                 384.00 [    0]   384.00 [    0]    384.00

---------------- Per-protocol Communication Count ----------------
                              MAX              MIN              AVE
Eager                           0 [    0]        0 [    0]       0.0
```

```
Rendezvous                   1536 [    0]      1536 [    0]    1536.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0
Persistent_Extended_IF          0 [    0]         0 [    0]       0.0
Unexpected_Message              1 [    0]         1 [    0]       1.0

------------------- Barrier Communication Count -------------------
                              MAX               MIN               AVE
Tofu                         4096 [    0]      4096 [    0]    4096.0
Soft                            0 [    0]         0 [    0]       0.0

----------- Tofu Barrier Collective Communication Count -----------
                              MAX               MIN               AVE
Bcast                           0 [    0]         0 [    0]       0.0
Reduce                          0 [    0]         0 [    0]       0.0
Allreduce                      16 [    0]        16 [    0]      16.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                              MAX               MIN               AVE
Alltoall                        0 [    0]         0 [    0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX               MIN               AVE
Bcast                         256 [    0]       256 [    0]     256.0
Reduce                          0 [    0]         0 [    0]       0.0
Gather                          0 [    0]         0 [    0]       0.0
Gatherv                         0 [    0]         0 [    0]       0.0
Allreduce                       0 [    0]         0 [    0]       0.0
Alltoall                      512 [    0]       512 [    0]     512.0
Alltoallv                     256 [    0]       256 [    0]     256.0
Allgather                       0 [    0]         0 [    0]       0.0
Allgatherv                      0 [    0]         0 [    0]       0.0

--------- Non-Tofu-specific Collective Communication Count --------
                              MAX               MIN               AVE
Bcast                           0 [    0]         0 [    0]       0.0
Reduce                          0 [    0]         0 [    0]       0.0
Gather                          0 [    0]         0 [    0]       0.0
Gatherv                         8 [    0]         8 [    0]       8.0
Allreduce                       0 [    0]         0 [    0]       0.0
Alltoall                        0 [    0]         0 [    0]       0.0
Alltoallv                       0 [    0]         0 [    0]       0.0
Allgather                      64 [    0]        64 [    0]      64.0
Allgatherv                    128 [    0]       128 [    0]     128.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
                              MAX               MIN               AVE
Eager                           0 [    0]         0 [    0]       0.0
Rendezvous                      0 [    0]         0 [    0]       0.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0
Collective                      0 [    0]         0 [    0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX               MIN               AVE
Eager                           0 [    0]         0 [    0]       0.0
Rendezvous                      0 [    0]         0 [    0]       0.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0

----------- Section    3 My message (Execute section3)  -----------
                              MAX               MIN               AVE
Time(Sec)                  177.47 [   25]    177.29 [    0]    177.43

------------------- Per-peer Communication Count ------------------
                              MAX               MIN               AVE
```

```
In_Node                        0 [    0]         0 [    0]        0.0
Neighbor                       0 [    0]         0 [    0]        0.0
Not_Neighbor                   0 [    0]         0 [    0]        0.0
Total_Count                    0 [    0]         0 [    0]        0.0
Connection                    46 [    0]         9 [    4]       11.8
Max_Hop                        4 [    0]         2 [    4]        3.1
Average_Hop                 2.27 [   35]      1.60 [    6]       1.84

----------------- Per-peer Transmission Size (MiB) ----------------
                              MAX               MIN               AVE
In_Node                     0.00 [    0]      0.00 [    0]       0.00
Neighbor                    0.00 [    0]      0.00 [    0]       0.00
Not_Neighbor                0.00 [    0]      0.00 [    0]       0.00
Total_Size                  0.00 [    0]      0.00 [    0]       0.00

----------------- Per-protocol Communication Count ----------------
                              MAX               MIN               AVE
Eager                          0 [    0]         0 [    0]        0.0
Rendezvous                     0 [    0]         0 [    0]        0.0
Hasty_Rendezvous               0 [    0]         0 [    0]        0.0
Persistent_Extended_IF         0 [    0]         0 [    0]        0.0
Unexpected_Message             1 [    0]         0 [    6]        0.8

------------------- Barrier Communication Count -------------------
                              MAX               MIN               AVE
Tofu                          12 [    0]        12 [    0]       12.0
Soft                           0 [    0]         0 [    0]        0.0

----------- Tofu Barrier Collective Communication Count -----------
                              MAX               MIN               AVE
Bcast                          0 [    0]         0 [    0]        0.0
Reduce                         0 [    0]         0 [    0]        0.0
Allreduce                      0 [    0]         0 [    0]        0.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                              MAX               MIN               AVE
Alltoall                       0 [    0]         0 [    0]        0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX               MIN               AVE
Bcast                        512 [    0]       512 [    0]      512.0
Reduce                         0 [    0]         0 [    0]        0.0
Gather                         4 [    0]         4 [    0]        4.0
Gatherv                        0 [    0]         0 [    0]        0.0
Allreduce                      0 [    0]         0 [    0]        0.0
Alltoall                       0 [    0]         0 [    0]        0.0
Alltoallv                      0 [    0]         0 [    0]        0.0
Allgather                      0 [    0]         0 [    0]        0.0
Allgatherv                     0 [    0]         0 [    0]        0.0

--------- Non-Tofu-specific Collective Communication Count --------
                              MAX               MIN               AVE
Bcast                          0 [    0]         0 [    0]        0.0
Reduce                        64 [    0]        64 [    0]       64.0
Gather                         0 [    0]         0 [    0]        0.0
Gatherv                        0 [    0]         0 [    0]        0.0
Allreduce                      0 [    0]         0 [    0]        0.0
Alltoall                       0 [    0]         0 [    0]        0.0
Alltoallv                      0 [    0]         0 [    0]        0.0
Allgather                      0 [    0]         0 [    0]        0.0
Allgatherv                     0 [    0]         0 [    0]        0.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
```

```
                              MAX              MIN              AVE
Eager                           0 [    0]         0 [    0]       0.0
Rendezvous                      0 [    0]         0 [    0]       0.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0
Collective                      0 [    0]         0 [    0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX              MIN              AVE
Eager                           0 [    0]         0 [    0]       0.0
Rendezvous                      0 [    0]         0 [    0]       0.0
Hasty_Rendezvous                0 [    0]         0 [    0]       0.0

---------------------- MPI Memory Usage (MiB) ---------------------
                              MAX              MIN              AVE
Estimated_Memory_Size       93.90 [    0]     44.39 [    4]     48.13

------------------------- Process Mapping ------------------------
(0,0,0)                    0,1
(1,0,0)                    2,3
(0,1,0)                    4,5
(1,1,0)                    6,7
(0,2,0)                    8,9
(1,2,0)                    10,11
(0,0,1)                    12,13
(1,0,1)                    14,15
(0,1,1)                    16,17
(1,1,1)                    18,19
(0,2,1)                    20,21
(1,2,1)                    22,23
(0,0,2)                    24,25
(1,0,2)                    26,27
(0,1,2)                    28,29
(1,1,2)                    30,31
(0,2,2)                    32,33
(1,2,2)                    34,35
(0,0,3)                    36,37
(1,0,3)                    38,39
(0,1,3)                    40,41
(1,1,3)                    42,43
(0,2,3)                    44,45
(1,2,3)                    46,47
```

6.16 Dynamic Debug during MPI Program Execution

This system provides the following functions for performing debugging during MPI program execution:

- Deadlock detection (communication wait truncation)
- Monitoring of write damage in MPI communication buffer
- Argument check function

Note that MPI program execution speeds may become slower if these debug functions are used. Use these functions with care.

6.16.1 Deadlock Detection

If a continuing message communication wait status occurs in MPI communication, a deadlock may have occurred in the execution of the MPI program. This system provides a procedure for breaking out of a state where a communication wait continues for a long time in case an unexpected deadlock occurs. An upper limit value for communication waits during MPI program execution can be set, and if the communication wait time exceeds this upper limit during MPI program execution, an appropriate message can be output and execution of the MPI program ended.

The MCA parameter mpi_deadlock_timeout can be used to set an upper limit value (seconds) for communication waits. Stack trace information is also output in the message. However, to display symbol names (function names) in the trace information, specify the linker option -export-dynamic in advance at the time of MPI program compilation. In particular, specify "-Wl,-export-dynamic" as an option passed to the compiler by the MPI program compile/edit command. Refer to "Table 4.36 mpi_deadlock_timeout (specifies the communication wait timeout time)" for information on the MCA parameter mpi_deadlock_timeout.

However, this deadlock detection function makes judgments solely on communication wait times. Therefore, great care is needed in using this function because, if there are no errors in the MPI program coding and communication wait times are truly long enough to exceed the specified wait time upper limit values, MPI program execution is ended. In addition, the process or the location within the MPI program at the time of deadlock detection is not necessarily the location where the deadlock actually occurred. Therefore, tracking back through the processes and reviewing the program may be needed to identify the location that caused the deadlock.

This function cannot necessarily detect all deadlocks. One method of detecting deadlocks effectively is to insert the MPI_Barrier function before and after communication locations. Another effective method is to use the MCA parameter mpi_deadlock_timeout_delay function to allow plenty of wait time after deadlock detection until the program ends to check the status of other processes.

The actual timing for ending a program when a deadlock is detected can be delayed by using the MCA parameter mpi_deadlock_timeout_delay. Delaying the time when the program is ended can increase the chances for detecting deadlocks in other processes. Refer to "Table 4.37 mpi_deadlock_timeout_delay (delays program termination caused by detection of a deadlock)" for information on the MCA parameter mpi_deadlock_timeout_delay.

6.16.2 Monitoring of Write Damage in MPI Communication Buffer

If another write is issued for a send buffer before send is completed for nonblocking communication, this may cause result errors, area corruption, or other incorrect operation that reduces reproducibility. In order to detect this sort of incorrect operation, this system provides a procedure for monitoring area corruption caused by writing to a nonblocking communication send buffer.

The MCA parameter mpi_check_buffer_write can be used to monitor a nonblocking communication send buffer. With this monitoring, if write to a nonblocking communication send buffer occurs, an appropriate message is output, and execution of the MPI program ends. Stack trace information is also output in the message. However, to display symbol names (function names) in the trace information, specify the linker option -export-dynamic in advance at the time of MPI program compilation. In particular, specify "-Wl,-export-dynamic" as an option passed to the compiler by the MPI program compile/edit command. Refer to "Table 4.35 mpi_check_buffer_write (monitors communication buffer write damage)" for information on the MCA parameter mpi_check_buffer_write.

Note that nonblocking communication is not monitored if send is in buffer mode (MPI_Ibsend function).

Example

Example

Set monitoring of the send buffer for a program like the one below.

```
main( )
{
    ...
    int buf = 0;
    MPI_Isend(&buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &req);
    buf = 1;
    MPI_Wait(&req, &status);
    ...
 }
```

Specify the "-Wl,-export-dynamic" option and use the MPI program compile/link command mpifccpx(1) to compile the above program. Next, specify 1 as the value for the MCA parameter mpi_check_buffer_write. When the above program is executed, an error message like the one below is output to the standard error output.

```
[mpi::opal-util::check-buffer-write] The buffer was destroyed in this process.
/.../FJSVmxlang/lib64/libmpi.so.0(PMPI_Wait+0x50) [0xffffffff20296ad0] (1)
./a.out(main+0x4a4) [0x1012e4](2)
/.../... (  ...              ) [0xfffffffa07f381c]
./a.out(  ...              )[0x100cec]
```

Lines 2 to 5 in the above error message are stack trace information. The location displayed in this stack trace information shows the completion location of the send processing of the nonblocking communication that used the send buffer when the write to the send buffer occurred. From the upper part of the stack (locations (1) and (2) in the output example in the above message), it is known that a write occurred to the send buffer used by the send (MPI_Isend function) of the nonblocking communication corresponding to the request checked by the MPI_Wait function called from the main function.

6.16.3 Argument Check Function

By using the debug MPI library of this system, it is easy to investigate whether or not MPI function call arguments are correct when an MPI program is executed.

To use this argument check function, specify either the -debuglib or --debuglib option when executing mpiexec(1). Refer to "4.1 Execution Command Formats" for the specification method.

If the argument check function detects an error, a message indicating this is output and it returns with the error class below. See "Appendix A Error Class List" for a list of the error classes output by this system.

```
MPI_ERR_COMM      Invalid communicator
MPI_ERR_COUNT     Invalid count
MPI_ERR_TAG       Invalid tag
MPI_ERR_RANK      Invalid rank
MPI_ERR_TYPE      Invalid data type
MPI_ERR_BUFFER    Invalid buffer pointer
MPI_ERR_REQUEST   Invalid request
MPI_ERR_TOPOLOGY  Invalid topology
MPI_ERR_DIMS      Invalid dimension
MPI_ERR_ROOT      Invalid root
MPI_ERR_GROUP     Invalid group
MPI_ERR_OP        Invalid operation
MPI_ERR_ARG       Other invalid argument
```

Example

Example

```
#include <mpi.h>

int main(int argc, char *argv[]){
    int  sbuf = 1, rbuf;

    MPI_Init(&argc, &argv);
    MPI_Reduce(&sbuf, &rbuf, 1, MPI_INT, MPI_SUM, -1, MPI_COMM_WORLD); // root(-1) is invalid
    MPI_Finalize();
}
```

In this program example, an incorrect root process is specified in the argument of the MPI_Reduce function. If this MPI program is used and an argument check is performed, an error message like the one below is output. The [em99-cn071:18342] output at the start of the second and subsequent lines is the host name and pid information.

```
[mpi::mpi-errors::mpi_errors_are_fatal]
[em99-cn071:18342] *** An error occurred in MPI_Reduce
[em99-cn071:18342] *** reported by process [11111,0]
[em99-cn071:18342] *** on communicator MPI_COMM_WORLD
[em99-cn071:18342] *** MPI_ERR_ROOT: invalid root
[em99-cn071:18342] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[em99-cn071:18342] ***    and potentially your MPI job)
```

6.17 Behavior on Forced Termination of MPI Programs

When the -nompi option is not specfied for mpiexec(1), a message that indicates the MPI program terminated abnormally by a communication error or a received signal may be output, if the job executing an MPI program is force-quitted, or the MPI program terminates before calling MPI_Finalize function.

Chapter 7 Error Messages

This chapter explains the error messages output for this system.

7.1 Output Format for Information Related to Parallel Processes

At the start of a message specifically related to parallel processes, the host name (host) and process ID (pid) corresponding to that parallel process may be output. If so, the output format is as follows:

```
[host:pid] message ID and message text character string
```

7.2 mpiexec Command Error Messages

```
--------------------------------------------------------------------------
[mpi::mca-base::duplicated-mca-params]
The following MCA parameter has been listed multiple times on
the command line:

  MCA param:   MCA parameter

MCA parameters can only be listed once on a command line to ensure there
is no ambiguity as to its value.  Please correct the situation and
try again.
--------------------------------------------------------------------------
```

- Description

  The same MCA parameter can only be listed once on a command line.

- Parameters

  MCA parameter: MCA parameter

- Action method

  Check the MCA parameters.

```
--------------------------------------------------------------------------
[mpi::mca-base::find-available:not-valid]
A requested component was not found, or was unable to be opened.  This
means that this component is either not installed or is unable to be
used on your system (e.g., sometimes this means that shared libraries
that the component requires are unable to be found/loaded).  Note that
Open MPI stopped checking at the first component that it did not find.

Host:          host
Framework:     frame
Component:     comp
--------------------------------------------------------------------------
```

- Description

  The specified MPI library function (framework component) could not be selected. An unsupported function may have been specified. Execution of the mpiexec command or the MPI program ends.

- Parameters

  host: Host name

  frame: Framework name

comp: Component name

- Action method

  Check the value specified for the MCA parameter.

[mpi::mca-base::getcwd-error] Error: Unable to get the current working directory

- Description

  Processing failed for the system call getcwd. The current path is set as the current directory. Execution of the mpiexec command continues.

- Action method

  The system may not be operating correctly. Contact the system administrator.

-- [mpi::mca-var::missing-param-file] Process *pid* Unable to locate the variable file "*file*" in the following search path: *wdir* --

- Description

  The AMCA parameter file (MCA parameter settings file) could not be found in the specified path. After message output, execution of the mpiexec command or the MPI program continues.

- Parameters

  pid: Process ID

  file: Specified file path

  wdir: Directory path executed by the mpiexec command

- Action method

  Check the AMCA parameter file (MCA parameter settings file) specification.

[mpi::opal-util::keyval-error] keyval parser: error *num* reading file *file* at line *lineno*: *code*

- Description

  The AMCA parameter file (MCA parameter settings file) contains characters that cannot be used. After message output, execution of the mpiexec command or the MPI program continues.

- Parameters

  num: Error number

  file: File path of the AMCA parameter file (MCA parameter settings file)

  lineno: Line number

  code: Character that cannot be used

- Action method

  Check whether there are any characters that cannot be used in the AMCA parameter file (MCA parameter settings file)

[mpi::opal-util::memory-error] Unable to allocate memory for the private addresses array

- Description

  Memory cannot be allocated for the private address array. Memory acquisition failed. After message output, execution of the mpiexec command or the MPI program continues.

- Action method

  Check the maximum memory size limit value of the program. If there is no problem, the system may not be operating correctly. Contact the system administrator.

[mpi::opal-util::param-option] Error: option "*opt*" did not have enough parameters (*num*)

- Description

  There are not enough arguments in the option specified in the mpiexec command. Execution of the mpiexec command ends.

- Parameters

  opt: Relevant option

  num: Number of required arguments

- Action method

  Specify the arguments required for the relevant mpiexec command option.

[mpi::opal-util::private-ipv4-error] FOUND BAD!

- Description

  An unsupported MCA parameter may have been specified. (OMPI_MCA_opal_net_private_ipv4) After message output, execution of the mpiexec command or the MPI program continues.

- Action method

  Check the value specified for the MCA parameter.

[mpi::opal-util::unknown-option] Error: unknown option "*opt*"

- Description

  An unsupported option was specified in the mpiexec command. Execution of the mpiexec command ends.

- Parameters

  opt: Relevant option

- Action method

  Specify the correct option in the mpiexec command.

--
[mpi::orterun::event-def-failed]
mpiexec was unable to define an event required for proper operation of
the system. The reason for this error was:

Error: *syserr*
--

- Description

  System call execution failed. Execution of the mpiexec command ends.

- Parameters

  syserr: System error details

- Action method

  The system may not be operating correctly. Contact the system administrator.

--
[mpi::orterun::multi-apps-and-zero-np]
mpiexec found multiple applications specified on the command line, with
at least one that failed to specify the number of processes to execute.

When specifying multiple applications, you must specify how many processes of each to launch via the -np argument.
--

- Description

  Processing cannot continue because the number of parallel processes for each MPI program was not specified when executing in MPMD model. Execution of the mpiexec command ends.

- Action method

  Specify the number of parallel processes for each of the MPI programs specified in the mpiexec command.

--
[mpi::orterun::nothing-to-do]
mpiexec could not find anything to do.

It is possible that you forgot to specify how many processes to run via the "-np" argument.
--

- Description

  An internal error may have occurred. Execution of the mpiexec command ends.

- Action method

  Consult System Engineer about the message that was output.

--
[mpi::orterun::orterun:appfile-not-found]
Unable to open the appfile:

file

Double check that this file exists and is readable.
--

- Description

  The file specified by the execution definition file specification method is not found. Execution of the mpiexec command ends.

- Parameters

  file: Specified file path

- Action method

  Check the file path.

--
[mpi::orterun::orterun:executable-not-specified]
No executable was specified on the mpiexec command line.

Aborting.
--

- Description

  No MPI programs were specified in the mpiexec command. Execution of the mpiexec command ends.

- Action method

  Specify an MPI program in the mpiexec command.

--
[mpi::orterun::precondition]

mpiexec was unable to precondition transports
Returned value *errno* instead of ORTE_SUCCESS.
--

- Description

  An internal error occurred. Execution of the mpiexec command ends.

- Parameters

  errno: Error number

- Action method

  Consult System Engineer about the message that was output.

--
[mpi::orte-runtime::orte_init:startup:internal-failure]
It looks like orte_init failed for some reason; your parallel process is
likely to abort. There are many reasons that a parallel process can
fail during orte_init; some of which are due to configuration or
environment problems. This failure appears to be an internal failure;
here's some additional information (which may only be relevant to an
Open MPI developer):

  ***fun* failed**
  --> Returned value *errinfo* (*errno*) instead of ORTE_SUCCESS
--

- Description

  Initialization processing failed for the mpiexec command or the MPI program. Execution of the mpiexec command or the MPI program ends.

- Parameters

  fun: Error function name

  errinfo: Error details

  errno: Error number

- Action method

  Check whether there are errors in the MCA parameter specification. If there are no errors in the specification, consult System Engineer about the message that was output.

[mpi::plm-ple::exec-plexec] Failed to invoke PLE. [errinfo:*errinfo*(*errno*) path:*com*]

- Description

  Execution of the plexec command failed. The parallel execution environment (PLE) of Job Operation Software may not be operating correctly. Execution of the mpiexec command ends.

- Parameters

  errinfo: Error details

  errno: Error number

  com: Executing command

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::plm-ple::parallel] Specified number of parallel processes is incorrect.

- Description

  The number of parallel processes specified in the mpiexec command is incorrect. Execution of the mpiexec command ends.

- Action method

  Check the number of parallel processes in the mpiexec command.

[mpi::plm-ple::recursive-mpiexec] mpiexec cannot be invoked recursively.

- Description

  Duplicate startup of the mpiexec command from the mpiexec command is not possible. Execution of the mpiexec command ends.

- Action method

  Specify an MPI program in the mpiexec command.

[mpi::plm-ple::signal-plexec] Received signal sent by PLE. [signo:*signo*]

- Description

  The plexec command received a signal and ended abnormally. The parallel execution environment (PLE) of Job Operation Software may not be operating correctly. Execution of the mpiexec command ends.

- Parameters

  signo: Signal number received by child process

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::plm-ple::wait-plexec] System error caused by waitpid. [errinfo:*errinfo*(*errno*)]

- Description

  Operation failed for the system call waitpid. Execution of the mpiexec command ends.

- Parameters

  errinfo: Error details

  errno: Error number

- Action method

  The system may not be operating correctly. Contact the system administrator.

[mpi::tools-orterun::param-env] Warning: could not find environment variable "*env*"

- Description

  The specified environment variable value has not been set. After message output, execution of the mpiexec command continues.

- Parameters

  env: Specified environment variable

- Action method

  Check the value of the environment variable specified in the option (-x) of the mpiexec command.

7.3 Communication Library Error Messages

[mpi::btl-tofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed when communicating through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::coll-mtofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check the amount of memory used and the maximum memory size limit value of the program. If the amount of memory used is too high, reduce it.

[mpi::coll-tbi::comm-query-failure] Internal error. [*reason*]

- Description

  Creation of a Tofu barrier network failed.

- Parameters

  reason: Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tbi::internal-file-error] Unable to *operate* file *file*.

- Description

  Access to the file used internally by the Tofu barrier failed.

- Parameters

  operate: Operation

  file: File path

- Action method

  The file system may not be operating correctly. Contact the system administrator.

[mpi::coll-tbi::memory-error] Unable to allocate memory. [*errno*]

- Description

  Memory acquisition failed for the tofu barrier.

- Parameters

  errno: Error number

- Action method

  Check the memory usage. If there is no problem, the system may not be operating correctly. Contact the system administrator.

[mpi::coll-tbi::operation-error] Operation error is reported by Tofu barrier communication. [*function arguments*] [*data*]

<Stack trace information>

- Description

  An operation error was detected in the Tofu barrier communication. Execution of the MPI program ends.

- Parameters

  function: MPI function

  arguments: Argument of MPI function

  data: Data for System Engineer for analysis purposes

- Action method

  Check whether there is either of the following descriptions at the shown MPI function in your MPI program.

  The reduction operation is showed in arguments as op.

  - Different reduction operations were specified among processes

  - Different collective communication functions were specified among processes

  If this checking indicates no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::coll-tuned::init-subcommunicator-failure] Internal error. [*reason*]

- Description

  Initialization processing failed for the collective communication subcommunicator processing.

- Parameters

  reason: Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tuned::memory-error] Unable to allocate memory.

- Description

  Memory acquisition failed.

- Action method

  Check the amount of memory used and the maximum memory size limit value of the program. If the amount of memory used is too high, reduce it.

[mpi::common-ple::jrm-open-failure] dlopen error caused by PLE. [dlerr:*dlerr*]

- Description

  dlopen failed. A problem may have occurred in the parallel execution environment (PLE) of Job Operation Software. Execution of the MPI program ends.

- Parameters

  dlerr: dlopen error details

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::common-ple::jrm-symbol-failure] dlsym error caused by PLE. [dlerr:*dlerr*]

- Description

  dlsym failed. A problem may have occurred in the parallel execution environment (PLE) of Job Operation Software. Execution of the MPI program ends.

- Parameters

  dlerr: dlsym error details

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::common-tofu::connection-error] Connection error. [*data*]

- Description

  An error was detected when establishing the connection of communication through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed when communicating through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::common-tofu::mrq-error] Communication error is reported by Tofu MRQ. [*data*]

- Description

  The Tofu interconnect detected a problem. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::mrq-memory-error] Communication memory error is reported by Tofu MRQ. [*data*]

- Description

  The Tofu interconnect detected a communication memory specification error. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check whether there is an error in the start address, datatype, or number of elements in the send buffer and receive buffer specified in the MPI communication function.

[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ. This error may be caused by abort of peer process. [*data*]

- Description

  The Tofu interconnect detected an error on the compute node where the communication partner process is being executed. The error may be caused by reason that the communication partner process was stopped or that the communication partner process released the send buffer or the receive buffer. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  In case the job was force-quitted, or the MPI program was terminated before MPI_Finalize is called, ignore this message. In case the send buffer or the receive buffer is released before the completion of communication, revise the processing of releasing buffer. Otherwise, consult System Engineer about the message that was output.

[mpi::common-tofu::set-signal] Failed to set realtime signal. [rc:*rc*]

- Description

  An internal error was detected. Realtime signal settings failed. Execution of the MPI program ends.

- Parameters

  rc: Error number

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tcq-error] Communication error is reported by Tofu TCQ. [*data*]

- Description

  The Tofu interconnect detected a communication error. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tcq-memory-error] Communication memory error is reported by Tofu TCQ. [*data*]

- Description

  The Tofu interconnect detected a communication memory specification error. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check whether there is an error in the start address, datatype, or number of elements in the send buffer and receive buffer specified in the MPI communication function.

[mpi::common-tofu::tofu-init-failure] Internal error. [*reason*]

- Description

  Initialization processing for the Tofu interconnect failed. Execution of the MPI program ends.

- Parameters

  reason: Cause of initialization processing failure

- Action method

  The system may not be operating correctly. Contact the system administrator.

[mpi::common-tofu::tofu-signal-corruption] Tofu driver detected corruption of system reserved area. [signo:*signo* cq:*num*]

- Description

  Tofu driver detected corruption of system reserved area. Execution of the MPI program ends.

- Parameters

  signo: Signal number

  num: CQ number

- Action method

  Check whether there is an error code that causes memory corruption in your MPI program. If there is no problem in the program, a system internal error or a hardware fault may have occurred. Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-signal-exception] Internal error of exception with signal. [signo:*signo* error:*error* cq:*num*]

- Description

  An internal error was detected. Execution of the MPI program ends.

- Parameters

  signo: Signal number

  error: Error number

  num: CQ number

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-signal-failure] Tofu interconnect detected an error. [signo:*signo* error:*error* cq:*num*]

- Description

  An internal error was detected. A signal was received because a Tofu interconnect error was detected. Execution of the MPI program ends.

- Parameters

  signo: Signal number

  error: Error number

  num: CQ number

- Action method

  If the error number is a value from 37 to 44 and your MPI program applies conditions of Tofu barrier communication, explained in "6.12 Use of Tofu Barrier Communication to Increase Speeds", check whether there is either of the following descriptions in your MPI program.

- Different reduction operations were specified among processes.
- Different collective communication functions were specified among processes.

If this checking indicates no errors or the error number is a value other than the above, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::common-tofu::tofu-signal-mrq] Tofu interconnect detected MRQ overflow. [signo:*signo* cq:*num*]

- Description

  An error was detected and a signal was received because the upper limit value for the nonblocking communication instruction queue of the Tofu interconnect was exceeded. There is a problem with the issue count for MPI program nonblocking communication processing. Execution of the MPI program ends.

- Parameters

  signo: Signal number

  num: CQ number

- Action method

  Revise the termination process for MPI program nonblocking communication processing or for sending using the Eager communication mode. Or increase the number of entries in a completion queue of Tofu interconnect as described in "Table 4.28 common_tofu_num_mrq_entries (change the number of entries in a completion queue)".

[mpi::common-tofu::tofu-stag-error] Failed query/register Tofu STag. [*data*]

- Description

  A buffer usage error or a shortage in a Tofu interconnect memory management resource was detected.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Check for errors in the start address of the send buffer and receive buffer, the data type, and the number of elements. Memory areas that MPI program parallel processes cannot write to cannot be specified in send or receive buffers. Alternatively, if the large page is not used, use the large page, or decrease patterns of the start address and number of elements in the send buffer and receive buffer specified in the MPI communication function. If the large page is used and there are no specification errors, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-stag-release-error] Failed to release Tofu STag. [data]

- Description

  Inconsistency was detected when releasing Tofu interconnect memory management resource.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::dpm-ple::init-failure] [[*jobid*,*snum*],*rank*] PLE is not yet initialized.

- Description

  Initialization of the Job Operation Software parallel execution environment (PLE) did not complete. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

rank: Rank number

- Action method

  Check that the MPI function was not executed before the MPI_Init function or after the MPI_Finalize function. If there are no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::dpm-ple::invalid-arg] [[*jobid*,*snum*],*rank*] Error by invalid argument.

- Description

  An invalid value was passed to the Job Operation Software parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Consult System Engineer about the message that was output.

[mpi::dpm-ple::jrm-spawn-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  Dynamic process generation failed in the Job Operation Software parallel execution environment (PLE). A problem may have occurred in the parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::dpm-ple::mpi-jobid-invalid] [[*jobid*,*snum*],*rank*] MPI jobid was invalid.

- Description

  The MPI program job ID is invalid. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  An internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::dpm-ple::port-name-error] [[*jobid*,*snum*],*rank*] The specified port name was invalid. [port_name:*name*]

- Description

  The port name specified in the argument is invalid. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Port number

- Action method

  Revise and specify the correct value for the port name. Alternatively, check whether or not the MPI process to which connection is being attempted using the port name is correctly in the reception wait state. If this checking indicates no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::dpm-ple::recv-wait-timeout] [[*jobid*,*snum*],*rank*] The wait time for the socket communication has passed.

- Description

  The value set as the reception wait time for socket communication was exceeded. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Revise the configured reception wait time for socket communication. If the set value is correct, the MPI program may not be operating correctly, so revise the MPI program.

[mpi::dpm-ple::spawn-limit-error] [[*jobid*,*snum*],*rank*] The occurrences of dynamic process creation exceeded the upper limit.

- Description

  The upper limit value for the dynamic process generation count was exceeded. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Set 4294967295 or less as the dynamic process generation count.

[mpi::dpm-ple::spawn-limit-error] [[jobid,snum],rank] The maximum number of dynamic process creation that was able to be generated at the same time exceeded the upper limit.

- Description

  The upper limit number of MPI_COMM_WORLD for the dynamic processes that can exist at the same time was exceeded. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Set 65535 or less as the number of MPI_COMM_WORLD for the dynamic processes that exists at the same time.

[mpi::dpm-ple::spawn-vcoordfile-error] [[*jobid*,*snum*],*rank*] Dynamic process creation is not allowed if case the option "vcoordfile" is specified.

- Description

  Dynamic process generation cannot be executed if -vcoordfile or --vcoordfile is specified in the mpiexec command. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  If --vcoordfile is specified in the mpiexec command, do not call dynamic process generation.

[mpi::dpm-ple::spawn-resource-error] [[*jobid*,*snum*],*rank*] There are not enough compute nodes to create the specified processes dynamically.

- Description

  There are no free nodes that can execute the specified dynamic process generation. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Check if a free node, required for dynamic process generation, is available.

[mpi::dpm-ple::tofu-update-failure] [[*jobid*,*snum*],*rank*] Internal error caused by Tofu. [errno:*errno*]

- Description

  An internal error was detected. Tofu interconnect update processing failed. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Consult System Engineer about the message that was output.

[mpi::dpm-ple::violated-establishing-communication] MPI_Comm_connect or MPI_Comm_accept is called despite the MCA parameter dpm_ple_no_establish_connection.

- Description

  The program tried to establish communication between two groups of MPI processes that do not share a communicator even the MCA parameter dpm_ple_no_establish_connection is specified. Execution of the MPI program ends.

- Action method

  Check the value specified for the MCA parameter or that the program does not establish communication between two groups of MPI processes that do not share a communicator.

[mpi::errmgr-base::orte-error] [[*jobid*,*snum*],*rank*] ORTE_ERROR_LOG:*error* in file *file* at line *lineno*

- Description

  An internal error occurred. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  error: Error details

  file: Error file path

  lineno: Line number

- Action method

  Consult System Engineer about the message that was output.

[mpi::ess-ple::init-failure] [[*jobid*,*snum*],*rank*] PLE is not yet initialized.

- Description

  Initialization of the Job Operation Software parallel execution environment (PLE) did not complete. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Check that the MPI function was not executed before the MPI_Init function or after the MPI_Finalize function. If there are no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::ess-ple::invalid-arg] [[*jobid*,*snum*],*rank*] Error by invalid argument.

- Description

  An invalid value was passed to the Job Operation Software parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Consult System Engineer about the message that was output.

[mpi::ess-ple::jrm-abort-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  Job harvesting processing failed for the parallel execution environment (PLE) of Job Operation Software. A problem may have occurred in the parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::ess-ple::jrm-fin-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  End processing failed for the parallel execution environment (PLE) of Job Operation Software. A problem may have occurred in the parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::ess-ple::jrm-init-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  Initialization processing failed for the parallel execution environment (PLE) of Job Operation Software. A problem may have occurred in the parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::ess-ple::jrm-rank-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  Acquisition of process mapping information failed. A problem may have occurred in the parallel execution environment (PLE) of Job Operation Software. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::ess-ple::process-illegal] [[*jobid*,*snum*],*rank*] Process rank or size illegal. [size:*num* rank:*rank*]

- Description

  The rank number and number of parallel processes for the MPI program are incorrect. There is a problem in the environment variable content set by the parallel execution environment (PLE) of Job Operation Software. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  num: Number of processes for the MPI program

  rank: Rank number

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::ess-ple::recursive-mpiexec] [[*jobid*,*snum*],*rank*] mpiexec cannot be invoked recursively.

- Description

  Duplicate startup of the mpiexec command from the mpiexec command is not possible. Execution of the mpiexec command ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Specify an MPI program in the mpiexec command.

[mpi::ess-ple::tofu-address-failure] [[*jobid*,*snum*],*rank*] Internal error caused by Tofu. [errno:*errno*]

- Description

  An internal error was detected. The address information fetch processing failed. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

snum: spawn number

rank: Rank number

errno: Error number

- Action method

  Consult System Engineer about the message that was output.

[mpi::fjmpi-prequest::same-request-args] The arguments of source/destination rank, message tag, and communicator for the request are identical to those of another request.

- Description

  The arguments of source/destination rank, message tag, and communicator for the request are identical to those of another request.

- Action method

  Specify other source/destination rank, message tag, or the communicator as the arguments of the FJMPI_Prequest_send_init function or the FJMPI_Prequest_send_init function. Or execute the FJMPI_Prequest_send_init function or FJMPI_Prequest_send_init function after freeing another request that already exists by the MPI_Request_free function.

[mpi::fjmpi-rdma::deregmem-not-allocated] Memory ID *id* not allocated.

- Description

  The memory ID specified as an argument is not associated to any buffer. It is impossible to unregister an unalloced memory ID. Execution of the MPI program ends.

- Parameters

  id: Memory ID

- Action method

  Revise the memory ID specification in the MPI program.

[mpi::fjmpi-rdma::get-timeout] Cannot get remote address in several seconds.

- Description

  The processing to get the remote node DMA address timed out. After message output, execution of the MPI program continues.

- Action method

  If this error continues, the system may not be operating correctly. Consult the system administrator.

[mpi::fjmpi-rdma::init-alloc] Out of memory.

- Description

  Memory cannot be allocated for Extended RDMA interface internal data. Memory acquisition failed. Execution of the MPI program ends.

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::fjmpi-rdma::init-mpi] MPI initialization error.

- Description

  An error was detected in MPI internal communication resource management during Extended RDMA interface initialization. Execution of the MPI program ends.

- Action method

  If the job type is node-sharing job, Extended RDMA interface cannot be used.

  In other cases, consult System Engineer about the message that was output.

Refer to the Job Operation Software manual for information on node-sharing job.

[mpi::fjmpi-rdma::init-stag] STag allocation error. [*data*]

- Description

  A Tofu interconnect memory management resource shortage was detected during Extended RDMA interface initialization. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::fjmpi-rdma::memid-alloc] Out of memory.

- Description

  Memory cannot be allocated for Extended RDMA interface internal data. Memory acquisition failed. Execution of the MPI program ends.

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::fjmpi-rdma::memid-error] Memory ID *id* out of range.

- Description

  An unsuitable memory ID was specified in an argument. Execution of the MPI program ends.

- Parameters

  id: Memory ID

- Action method

  Revise the memory ID specification in the MPI program. Refer to the manual for the allowable values.

[mpi::fjmpi-rdma::mrq-error] Communication error is reported by Tofu MRQ. [*data*]

- Description

  The Tofu interconnect detected a problem. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::fjmpi-rdma::mrq-overflow] Complete queue overflow.

- Description

  The Extended RDMA interface internal completion queue has overflowed. The CQ polling processing in the MPI program may be inadequate. Execution of the MPI program ends.

- Action method

  Revise the CQ polling processing in the MPI program.

[mpi::fjmpi-rdma::pid-error] Process ID *id* out of range.

- Description

  An unsuitable rank number was specified in an argument. Execution of the MPI program ends.

- Parameters

  id: Rank number

- Action method

  Revise the rank number specification in the MPI program.

[mpi::fjmpi-rdma::raddr-error] Remote DMA address *raddr* invalid.

- Description

  An unsuitable remote node DMA address was specified in an argument. Execution of the MPI program ends.

- Parameters

  raddr: Remote node DMA address

- Action method

  Revise the remote node DMA address specification in the MPI program.

[mpi::fjmpi-rdma::regmem-allocated] Memory ID *id* already allocated.

- Description

  The memory ID specified in the argument is already in use. Multiple addresses cannot be registered for the same memory ID. Execution of the MPI program ends.

- Parameters

  id: Memory ID

- Action method

  Revise the memory ID specification in the MPI program.

[mpi::fjmpi-rdma::regmem-stag] STag allocation error. [*data*]

- Description

  A Tofu interconnect memory management resource shortage was detected during memory ID registration. Execution of the MPI program ends.

- Parameters

  data: Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

--
[mpi::mpi-api::mpi-abort]
MPI_ABORT was invoked on rank *rank* in communicator *comm*
with errorcode *rc*.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.
--

- Description

  The MPI_Abort function was called. Execution of the MPI program ends.

- Parameters

  rank: Rank number

  comm: Detailed information of the communicator in the MPI_Abort function first argument

rc: MPI_Abort function second argument

- Action method

  Check whether there is an error in the MPI program content.

--
[mpi::mpi-api::mpi-function-after-finalize]
Calling any MPI-function after calling MPI_Finalize is erroneous.
The only exceptions are MPI_Initialized, MPI_Finalized and MPI_Get_version.
--

- Description

  After MPI_Finalize, an MPI function that cannot be called after MPI_Finalize due to MPI specifications was called.

- Action method

  Check if an MPI function was called after MPI_Finalize in the program.

  If called, this contravenes MPI specifications, so correct the program.

  However, the following functions can be called after MPI_Finalize:

  MPI_Initialized

  MPI_Finalized

  MPI_Get_version

--
[mpi::mpi-api::mpi-initialize-twice]
Calling MPI_Init or MPI_Init_thread twice is erroneous.
--

- Description

  Either the MPI_Init function or the MPI_Init_thread function was called twice.

- Action method

  Check if either the MPI_Init function or the MPI_Init_thread function was called more than once in the program. Either the MPI_Init function or the MPI_Init_thread function can be called only once due to MPI specifications. If called more than once, correct the program.

--
[mpi::mpi-runtime::mpi-param-check-enabled-but-compiled-out]
WARNING: The MCA parameter mpi_param_check has been set to true, but
parameter checking has been compiled out of Open MPI. The
mpi_param_check value has therefore been ignored.
--

- Description

  The MCA parameter "mpi_param_check" was set. MCA parameter check cannot be specified unless the library is the debug MPI library. After message output, execution of the MPI program continues.

- Action method

  Remove the MCA parameter "mpi_param_check" specification.

--

[mpi::mpi-errors::mpi_errors_are_fatal]
[*info*] * An error occurred [*msg*]**
[*info*] * reported by process [[*jobid*,*rank*]]**
[*info*] * on [*type*]**
[*info*] * [*error class*]**

[*info*] *** MPI_ERRORS_ARE_FATAL (processes in this [*type*] will now abort,
[*info*] ***   and potentially your MPI job)

- Description

  A fatal error occurred during execution of the MPI program. The program will abort.

- Parameters

  info: Host name and pid information

  msg: Explanation of the cause of the problem

  jobid: MPI job ID

  rank: Rank number under MPI_COMM_WORLD

  type: Information on either the communicator, file, or window (depending on the cause of the problem)

  error class: See Appendix A

- Action method

  Refer to the ***msg*** and ***error class*** and check for problems in the program. If there is no problem, note the message that is output and contact the system administrator.

[mpi::ompi-communicator::error] [[*jobid*,*snum*],*rank*] Internal error. [rc:*rc*]

- Description

  An internal error was detected. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  rc: Function return value

- Action method

  Consult System Engineer about the message that was output.

[mpi::ompi-communicator::recv-wait-timeout] [[*jobid*,*snum*],*rank*] The wait time for the socket communication has passed.

- Description

  The value set as the reception wait time for socket communication was exceeded. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Revise the configured reception wait time for socket communication. If the set value is correct, the MPI program may not be operating correctly, so revise the MPI program.

[mpi::ompi-free-list::memory-error] Out of memory.

- Description

  Memory acquisition failed. Execution of the MPI program ends.

- Action method
    - Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.
    - Because the MPI library saves messages to other regions if corresponding receive calls are delayed, these saved messages affect the amount of memory used. Therefore, if the number of all communication partner processes is more than the upper limit for the number of processes that can use the fast communication mode, and MPI program issues a large number of unexpected messages, then this problem might be avoided by speeding up the communication according to the following procedure.
        1. Execute the MPI program after specifying value 2 for the MCA parameter mpi_print_stats. Do not specify MCA parameter mpi_print_stats_ranks. When you use the MCA parameter, you must specify -1 for mpi_print_stats_ranks, and the output from all the parallel processes must enable. Confirm the number of unexpected messages by outputting the MPI statistical information. Refer to "6.15 MPI Statistical Information" for details.
        2. If the number of unexpected messages increased, enlarge the size of the Medium receive buffer. Refer to "Table 4.25 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)" for details.

[mpi::opal-runtime::assistant-core-bind-failed] Failed to bind the progress thread to an assistant core. Your program needs relinking.

- Description

  The MPI asynchronous processing progress thread could not be bound to an assistant core because the MPI program was linked by a compilation/linkage command in an old system. After message output, function of promoting asynchronous communication using an assistant core is disabled and execution of the MPI program continues.

- Action method

  Relink the MPI program by a compilation/linkage command in this system.

[mpi::opal-util::check-buffer-write] The buffer was destroyed in this process.

<Stack trace information>

- Description

  Another write occurred in the nonblocking communication send buffer. Execution of the MPI program ends.

- Action method

  Refer to the stack trace information and revise the MPI program so that no processes overwrite the nonblocking communication send buffer.

[mpi::opal-util::deadlock-timeout] This process detected a deadlock.

<Stack trace information>

- Description

  The communication wait time exceeded the upper limit (seconds) specified by the user. A deadlock may have occurred. Execution of the MPI program ends.

- Action method

  Refer to the stack trace information and revise the MPI program to ensure that there is no code that causes deadlocks.

[mpi::opal-util::dynamic-debug-failure] Internal error. [*reason*]

- Description

  Execution of the dynamic debug function failed.

- Parameters

  reason: Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::opal-util::dynamic-debug-memory-error] Unable to allocate memory. [*errno*]

- Description

  Memory allocation for use by the dynamic debug function failed.

- Parameters

  errno: Error number

- Action method

  Check the memory usage. If there is no problem, the system may not be operating correctly. Contact the system administrator.

[mpi::pubsub-ple::init-failure] [[*jobid*,*snum*],*rank*] PLE is not yet initialized.

- Description

  Initialization of the Job Operation Software parallel execution environment (PLE) did not complete. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Check that the MPI function was not executed before the MPI_Init function or after the MPI_Finalize function. If there are no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::pubsub-ple::invalid-arg] [[*jobid*,*snum*],*rank*] Error by invalid argument.

- Description

  An invalid value was passed to the Job Operation Software parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

- Action method

  Consult System Engineer about the message that was output.

[mpi::pubsub-ple::jrm-port-failure] [[*jobid*,*snum*],*rank*] Error caused by PLE. [errno:*errno*]

- Description

  Information related to socket communication failed to be set for the Job Operation Software parallel execution environment (PLE). Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Function return value

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::pubsub-ple::memory-error] [[*jobid*,*snum*],*rank*] Unable to allocate memory. [errno:*errno*]

- Description

  Allocation of the memory required for getting the port name failed. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  errno: Error number

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::pubsub-ple::port-name-error] [[*jobid*,*snum*],*rank*] The specified port name was invalid. [port_name:*name*]

- Description

  The port name specified in the argument is invalid. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Port number

- Action method

  Check if the value specified as the port name is correct. Alternatively, revise so that the MPI process connected to using the port name is executed.

[mpi::pubsub-ple::service-delete-cannot] [[*jobid*,*snum*],*rank*] The specified service name cannot be unpublished by any process other than the one that published it. [service_name:*name*]

- Description

  The specified service name cannot be unpublished because it was published by a different process. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Service name

- Action method

  Do not unpublish a service name from an MPI process other than the one that published it.

[mpi::pubsub-ple::service-delete-error] [[*jobid*,*snum*],*rank*] The specified service name was not open to the public. [service_name:*name*]

- Description

  The specified service name is not open to the public. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Service name

- Action method

  Revise the specified service name.

[mpi::pubsub-ple::service-get-error] [[*jobid*,*snum*],*rank*] The specified service name was not open to the public. [service_name:*name*]

- Description

  The specified service name is not open to the public. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Service name

- Action method

  Revise the specified service name value.

[mpi::pubsub-ple::service-set-error] [[*jobid*,*snum*],*rank*] The specified service name was already open to the public. [service_name:*name*]

- Description

  The specified service name is already open to the public. Execution of the MPI program ends.

- Parameters

  jobid: MPI job ID

  snum: spawn number

  rank: Rank number

  name: Service name

- Action method

  Revise the specified service name value.

Appendix A Error Class List

This appendix lists the error classes output by this system. These are the error classes regulated by the MPI standards. Refer to the MPI standards for details.

A.1 MPI1 Error Class List

Table A.1 MPI1 error class list

Error class	Description	Value
MPI_SUCCESS	No errors	0
MPI_ERR_BUFFER	Invalid buffer pointer	1
MPI_ERR_COUNT	Invalid count argument	2
MPI_ERR_TYPE	Invalid data type argument	3
MPI_ERR_TAG	Invalid tag argument	4
MPI_ERR_COMM	Invalid communicator	5
MPI_ERR_RANK	Invalid rank	6
MPI_ERR_REQUEST	Invalid request (handle)	7
MPI_ERR_ROOT	Invalid root	8
MPI_ERR_GROUP	Invalid group	9
MPI_ERR_OP	Invalid operation	10
MPI_ERR_TOPOLOGY	Invalid topology	11
MPI_ERR_DIMS	Invalid dimension argument	12
MPI_ERR_ARG	Invalid argument	13
MPI_ERR_UNKNOWN	Error with unknown cause	14
MPI_ERR_TRUNCATE	Message truncated at reception	15
MPI_ERR_OTHER	An error not in this list	16
MPI_ERR_INTERN	MPI internal error	17
MPI_ERR_IN_STATUS	Error code in status	18
MPI_ERR_PENDING	Reserve requested	19
MPI_ERR_LASTCODE	Last error code	92

A.2 MPI2 Error Class List

Table A.2 MPI2 error class list

Error class	Description	Value
MPI_ERR_ACCESS	Permission error	20
MPI_ERR_AMODE	Invalid amode	21
MPI_ERR_ASSERT	Invalid assert argument	22
MPI_ERR_BAD_FILE	Invalid filename	23
MPI_ERR_BASE	Invalid base argument	24
MPI_ERR_CONVERSION	Error during conversion of user-defined data	25

Error class	Description	Value
MPI_ERR_DISP	Invalid displacement argument	26
MPI_ERR_DUP_DATAREP	datarep already defined	27
MPI_ERR_FILE_EXISTS	File already exists	28
MPI_ERR_FILE_IN_USE	File operations by multiple processes not yet completed for the open file	29
MPI_ERR_FILE	Invalid file handle	30
MPI_ERR_INFO_KEY	Invalid Info key	31
MPI_ERR_INFO_NOKEY	Key is not defined	32
MPI_ERR_INFO_VALUE	Invalid Info value	33
MPI_ERR_INFO	Invalid Info argument	34
MPI_ERR_IO	Other I/O error	35
MPI_ERR_KEYVAL	Invalid key value	36
MPI_ERR_LOCKTYPE	Invalid locktype argument	37
MPI_ERR_NAME	Not an established name	38
MPI_ERR_NO_MEM	No valid memory	39
MPI_ERR_NOT_SAME	collective argument is different	40
MPI_ERR_NO_SPACE	Insufficient space	41
MPI_ERR_NO_SUCH_FILE	File does not exist	42
MPI_ERR_PORT	Port does not exist	43
MPI_ERR_QUOTA	quota exceeded	44
MPI_ERR_READ_ONLY	Read-only file or file system	45
MPI_ERR_RMA_CONFLICT	Window access conflict	46
MPI_ERR_RMA_SYNC	Incorrect synchronization for RMA invocation	47
MPI_ERR_SERVICE	Service already established	48
MPI_ERR_SIZE	Invalid size argument	49
MPI_ERR_SPAWN	Child process cannot be generated	50
MPI_ERR_UNSUPPORTED_DATAREP	Unsupported datarep	51
MPI_ERR_UNSUPPORTED_OPERATION	Unsupported operation	52
MPI_ERR_WIN	Invalid window argument	53
MPI_ERR_NOT_IMPLEMENTED	Unsupported function	None

A.3 MPI3 Error Class List

Table A.3 MPI3 error class list

Error class	Description	Value
MPI_T_ERR_MEMORY	Out of memory	54
MPI_T_ERR_NOT_INITIALIZED	Interface not initialized	55
MPI_T_ERR_CANNOT_INIT	Interface not in the state to be initialized	56
MPI_T_ERR_INVALID_INDEX	The enumeration index is invalid or has been deleted	57
MPI_T_ERR_INVALID_ITEM	The item index queried is out of range	58

Error class	Description	Value
MPI_T_ERR_INVALID_HANDLE	The handle is invalid	59
MPI_T_ERR_OUT_OF_HANDLES	No more handles available	60
MPI_T_ERR_OUT_OF_SESSIONS	No more sessions available	61
MPI_T_ERR_INVALID_SESSION	Session argument is not a valid session	62
MPI_T_ERR_CVAR_SET_NOT_NOW	Variable cannot be set at this moment	63
MPI_T_ERR_CVAR_SET_NEVER	Variable cannot be set until end of execution	64
MPI_T_ERR_PVAR_NO_STARTSTOP	Variable cannot be started or stopped	65
MPI_T_ERR_PVAR_NO_WRITE	Variable cannot be written or reset	66
MPI_T_ERR_PVAR_NO_ATOMIC	Variable cannot be read and written atomically	67
MPI_ERR_RMA_RANGE	Target memory is not part of the window	68
MPI_ERR_RMA_ATTACH	Memory cannot be attached	69
MPI_ERR_RMA_FLAVOR	Passed window has the wrong flavor for the called function	70
MPI_ERR_RMA_SHARED	Memory cannot be shared	71

Appendix B Notes on Migration from FX10 System to FX100 System

This appendix provides notes on migrating from FX10 system (Generation number:09 or later) to FX100 system.

For migrating from FX10 system (Generation number:08 or earlier), refer to "Appendix C Compatibility Information (FX10 system)" also.

B.1 Behavior of mpiexec(1) when a same MCA parameter is specified more than once is changed

This note corresponds to the migration to FX100 system (Generation Number:03 or later).

Refer to "D.1.1 Behavior of mpiexec(1) when a same MCA parameter is specified more than once is changed".

B.2 Value of MPI_ERR_LASTCODE is changed

This note corresponds to the migration to FX100 system (Generation Number:03 or later).

Refer to "D.1.2 Value of MPI_ERR_LASTCODE is changed".

B.3 Change of the "Threshold Value" for Switching between Eager Protocol and Rendezvous Protocol

a. Changes

The formula of the "threshold value" for switching between Eager protocol and Rendezvous protocol is changed.

[Previous version]

The "threshold value" was obtained by using the following formula.

```
Threshold value = 13,312 + number-of-hops * 296
```

[This version]

The "threshold value" is obtained by using the following formula.

```
Threshold value = 45,056 + number-of-hops * 296
```

b. Influence

The "threshold value" for switching between Eager protocol and Rendezvous protocol is changed if the MCA parameter btl_tofu_eager_limit is not specified. If your MPI program transfers a message which has a size between the threshold values of previous version and this version, the protocol to transfer the message changes. This may cause changes of runtime performance characteristics of the MPI program because these two protocols have different characteristics. These performance characteristics include whether a nonblocking communication overlaps with computation or other communications.

c. Coping

Set the MCA parameter btl_tofu_eager_limit to change the "threshold value".

B.4 Change of Extended RDMA Interface Error Message

This note corresponds to the migration from FX10 system (Generation Number:09 or earlier).

Refer to "C.1.1 Change of Extended RDMA Interface Error Message".

B.5 The default value of MCA parameter orte_abort_print_stack is changed from 0 to 1

This note corresponds to the migration from FX10 system (Generation Number:09 or earlier).

Refer to "C.1.2 The default value of MCA parameter orte_abort_print_stack is changed from 0 to 1".

Appendix C Compatibility Information (FX10 system)

This appendix provides compatibility information as notes on migrating.

C.1 Migrating to V2.0L10 (Generation Number:10)

C.1.1 Change of Extended RDMA Interface Error Message

a. Changes

When an unsuitable remote node DMA address *raddr* is specified in an argument of communication functions of extended RDMA interface, the error message is different from before.

[Previous version]

When an unsuitable remote node DMA address is specified in an argument of the FJMPI_Rdma_put function or the FJMPI_Rdma_get function, the following message was output.

```
[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ. This
error may be caused by abort of peer process. [data]
```

[This version]

When an unsuitable remote node DMA address is specified in an argument of the FJMPI_Rdma_put function or the FJMPI_Rdma_get function, the following message is output.

```
[mpi::fjmpi-rdma::raddr-error] Remote DMA address raddr invalid.
```

b. Influence

When an unsuitable remote node DMA address is specified in an argument of the FJMPI_Rdma_put function or the FJMPI_Rdma_get function in a user program, the error message in [This version] is output.

c. Coping

It is not necessary to action.

C.1.2 The default value of MCA parameter orte_abort_print_stack is changed from 0 to 1

a. Changes

The default value of MCA parameter orte_abort_print_stack is changed from 0 to 1.

[Previous version]

The default value for this parameter was 0.

[This version]

The default value for this parameter is 1.

b. Influence

If MPI_Abort function is called, or if the MPI library ends the execution of the MPI program detecting abnormalities of the execution environment and the communication, stack trace information are output following the error message to the standard error.

c. Coping

It is not necessary to action.

C.2 Migrating to V1.0L30 (Generation Number:09)

C.2.1 Change of Communication Library Error Message

a. Changes

Communication Library Error Message is changed.

[Previous version]

When some problems occur in the communicator, the file, or the window managed in MPI, the following message was output.

```
--------------------------------------------------------------------------
[mpi::mpi-errors::mpi_errors_are_fatal]
[info] *** An error occurred [msg]
[info] *** on [type]
[info] *** [error class]
[info] *** MPI_ERRORS_ARE_FATAL (your MPI job will now abort)
--------------------------------------------------------------------------
```

[This version]

When some problems occur in the communicator, the file, or the window managed in MPI, the following message is output.

```
--------------------------------------------------------------------------
[mpi::mpi-errors::mpi_errors_are_fatal]
[info] *** An error occurred [msg]
[info] *** on [type]
[info] *** [error class]
[info] *** MPI_ERRORS_ARE_FATAL: your MPI job will now abort
--------------------------------------------------------------------------
```

b. Influence

When some problems occur in the communicator, the file, or the window managed in MPI, the error message in [This version] is output.

c. Coping

It is not necessary to action.

C.3 Migrating to V1.0L20

C.3.1 Changes To Retrieved Values Of The Predefined Attributes

a. Changes

The values of the predefined attributes of the communicator duplicated from MPI_COMM_WORLD are changed if these values are retrieved by using the subroutine MPI_COMM_GET_ATTR or MPI_ATTR_GET in Fortran programs.

[Previous version]

- MPI_COMM_GET_ATTR

  The address of the predefined attribute was retrieved.

- MPI_ATTR_GET

  The address of the predefined attribute copied into default integer type was retrieved.

[This version]

- MPI_COMM_GET_ATTR

  The value of the predefined attribute is retrieved.

- MPI_ATTR_GET

  The value of the predefined attribute is retrieved.

b. Influence

   When the values of the predefined attributes of the communicator duplicated from MPI_COMM_WORLD are retrieved by using the subroutine MPI_COMM_GET_ATTR or MPI_ATTR_GET, and those values are used in Fortran programs, the behavior of the program may change.

c. Coping

   Modify the source program.

Appendix D Compatibility Information (FX100 system)

This appendix provides compatibility information as notes on migrating.

D.1 Migrating to V2.0L20 (Generation Number:03)

D.1.1 Behavior of mpiexec(1) when a same MCA parameter is specified more than once is changed

a. Changes

The behavior of mpiexec(1) when a same MCA parameter is specified more than once is changed.

[Previous version]

If there are multiple specifications for the same MCA parameter name for one program, the value specified first in the parameter specification contents was enabled.

[This version]

The following error message is output.

```
[mpi::mca-base::duplicated-mca-params]
The following MCA parameter has been listed multiple times on the command line:

    MCA param:   MCA parameter

MCA parameters can only be listed once on a command line to ensure there is no ambiguity as
to its value.
Please correct the situation and try again.
```

b. Influence

The mpiexec(1) outputs error message when a same MCA parameter is specified more than once.

c. Coping

Execute the MPI program with eliminating duplicated parameters.

D.1.2 Value of MPI_ERR_LASTCODE is changed

a. Changes

The value of MPI_ERR_LASTCODE is changed.

[Previous version]

The value of MPI_ERR_LASTCODE was 54.

[This version]

The value of MPI_ERR_LASTCODE is 92.

b. Influence

The behavior of a program is changed when the program where MPI_ERR_LASTCODE is referred is compiled using a compiler released before V2.0L20 (Generation Number:03) and is executed using the MPI library in V2.0L20 (Generation Number:03 or later).

c. Coping

Recompile the program where MPI_ERR_LASTCODE is referred using the compiler in V2.0L20 (Generation Number:03 or later).

Glossary

barrier gate

A hardware resource used for performing Tofu barrier communication. Barrier gates include two types of gates: input-output gates that fulfil the role of a start point and end point, and relay gates that fulfil the role of relay points. MPI can use a maximum of 8 input-output gates and 56 relay gates. These maximum numbers may be changed when the edition number of this system product is changed.

blocking communication

Indicates message send-receive for which the user buffer specified at MPI function invocation can be re-used if there is a return from the MPI function.

maximum transmission unit

Message transfer is performed by transmitting units, known as packets, within the MPI library. Each packet has an upper limit value, and the maximum transmission unit indicates this upper limit.

Messages that are larger than the maximum transmission unit are split into multiple packets such that the size of each packet is the maximum transmission unit or less, and then transferred.

message length

Indicates the number of elements in a message. This conforms to the message length definition in the MPI standards.

message size

The message size expressed as the number of bytes. In this manual, this term is used to distinguish the number of bytes from the "message length".

MPMD

Acronym meaning Multiple Program/Multiple Data. This is one parallel programming model. It uses two or more different MPI programs and operates by sharing processing.

nonblocking communication

Indicates message send-receive for which it is possible that there will be a return from the MPI function before the actual procedures of the MPI function are completed. The user buffer specified at MPI function invocation cannot be re-used until completion of operations is confirmed.

parallel process

A process started on the compute node by mpiexec(1) is called a parallel process. A number, starting from 0, is assigned to each parallel process. In this system, these numbers correspond to the rank numbers of the MPI program communicator MPI_COMM_WORLD.

SPMD

Acronym meaning Single Program/Multiple Data. This is one parallel programming model. It uses the same MPI program for each process and operates by sharing processing.

Tofu barrier communication

A hardware communication mechanism that provides a data reduction function for data that is eight bytes or less and that performs barrier synchronization between nodes under a Tofu interconnect.

type signature

It is essential that the messages transmitted by MPI functions can be split into basic data type data lists. The type signatures are these "basic data type lists". Type signature is a term defined in the MPI standards.

unexpected message

A message that needs to be left saved in the temporary buffer during the receive-side process due to a delay in calling a receive-type function (such as the MPI_Recv function) in response to a send-type function (such as the MPI_Send function.)