FUJITSU Software
Compiler Package V1.0L21

MPI User's Guide

J2UL-2587-03ENZ0(02)
October 2023

Preface

Purpose of This Manual

This manual describes how to use the MPI library intended for the PRIMEHPC FX700 system (referred to below as "this computing system").

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 (referred to below as "this software system"). Note that this software system conforms to the MPI-3.1 Standard and a subset of the MPI-4.0 Standard regulated by the MPI Forum.

Intended Readers

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

Structure of This Manual

The structure of this manual is as follows:

Chapter 1 Overview

An overview of this software 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 software system

Chapter 7 Error Messages

Error messages detected by this software system

Appendix A Error Class List

Error classes detected by this software system

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
- C User's Guide
- C++ User's Guide

- C/C++ Compiler Optimization Messages
- Fortran/C/C++ Runtime Messages
- Profiler User's Guide
- Programmer's Guide for Usage of Mathematical Libraries
- FUJITSU 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
- MPI User's Guide Additional Volume Java Interface

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

MPI: A Message-Passing Interface Standard

Version 3.1

Message Passing Interface Forum

June 4, 2015

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

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

Notations

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}

Syntax Description Symbols

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

Symbol name	Symbol	Explanation
Selection symbols	{}	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 "{}".
Repeat symbol	...	The item immediately preceding the ellipsis can be specified repeatedly in the syntax.

Representation of Routine Names

This software system provides the language bindings for Fortran, C, C++, and Java.

As defined in the MPI standard and "MPI User's Guide Additional Volume Java Interface" which is an additional volume of this manual, the specifications of C functions, C++ member functions, Fortran subroutines and functions, and Java methods are almost same, although they are called differently.

Therefore, in this manual, they are called "routines" as a common name which does not depend on languages.

In addition, C function names written in capital letters is used to represent routine names as well as the MPI standard.

As for routines defined only in the Fortran bindings, they are represented by corresponding Fortran subroutine/function names written in capital letters.

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

- Java is registered trademarks of Oracle and/or its affiliates.
- Linux(R) is the registered trademark of Linus Torvalds in the U.S. and other countries.
- All other trademarks are the property of their respective owners.
- The trademark notice symbol (TM, (R)) is not necessarily added in the system name and the product name, etc. published in this material.

Date of Publication and Version

Version	Manual code
October 2023, Version 3.2	J2UL-2587-03ENZ0(02)
October 2022, Version 3.1	J2UL-2587-03ENZ0(01)
January 2022, 3rd Version	J2UL-2587-03ENZ0(00)
August 2021, Version 2.5	J2UL-2587-02ENZ0(05)
July 2021, Version 2.4	J2UL-2587-02ENZ0(04)
March 2021, Version 2.3	J2UL-2587-02ENZ0(03)
November 2020, Version 2.2	J2UL-2587-02ENZ0(02)
September 2020, Version 2.1	J2UL-2587-02ENZ0(01)
July 2020, 2nd Version	J2UL-2587-02ENZ0(00)
February 2020, 1st Version	J2UL-2587-01ENZ0(00)

Copyright

Update History

Changes	Location	Version
The explanation is improved.	Chapter 4	Version 3.2
The section "Memory Usage of MPI Library" is added.	6.7	
The explanation is added.	Chapter 4	Version 3.1
The explanations are improved.	-	

Changes	Location	Version
The note on the assumed-rank of the Fortran 2018 language specification is removed.	3.2	3rd Version
The following MCA parameters are added. - btl_openib_autoresize_memory_threshold - coll_basic_gatherv_sync	4.2	
The explanation to the following MCA parameter is corrected. - btl_openib_eager_limit	4.2	
The default of "value" for the following Info key is changed. - romio_ds_write	6.2.12	
The section "Avoiding Concentration of Communications in MPI_GATHERV Routine" is added.	6.8	
Improved the explanation.	-	
The default value of the MCA parameter btl_openib_receive_queues_default_num_bufs is changed.	4.2	Version 2.5
The article about the maximum value of the file input-output size is corrected.	6.2.11	
The article is added to "Compilation/linkage notes".	3.2	Version 2.4
Changed the description of the upper limit value for the MCA parameter btl_openib_eager_limit.	4.2	
The following MCA parameters are added. - btl_openib_max_send_size - btl_openib_receive_queues_default_num_bufs	4.2	
"MPI Program Execution Using Slurm" is added.	6.9	
The explanation to the following MCA parameter is corrected. - btl_openib_eager_limit	4.2	Version 2.3
The explanation to the following MCA parameter is added. - mpi_java_eager	4.2	
The explanations are corrected.	4.2 6.7	Version 2.2
Note is added.	6.7	
The explanations are corrected.	4.1	Version 2.1
The explanations are added.	6.2.11	
The descriptions of { -c | -np | --np | -n | --n } related options are corrected.	4.1	2nd Version
The local_options -am is changed to -tune.	4.1 4.2	
The explanations are corrected.	4.1 5.3.1.1	
The following MCA parameters are added. - opal_abort_delay - opal_progress_timeout	4.2	
The explanations are added.	4.2 6.4	

Changes	Location	Version
	6.8 6.8.1	
fjprof_spawn_dir_name key is deleted.	6.2.9.1	
The error messages are added.	7.3	
Improved the explanation.	-	
Changed the look according to product upgrades.	-	

Contents

Chapter 1 Overview....1
- 1.1 Features of This Software System....1
- 1.2 Outline of How to Use This Software System....1
  - 1.2.1 Flow from Compilation to Execution of an MPI Program....1

Chapter 2 Environment and Advance Settings....3
- 2.1 MPI Program Compilation/Linkage Environment....3
- 2.2 MPI Program Execution Environment....3
- 2.3 Online Manual....4

Chapter 3 MPI Program Compilation/Linkage....5
- 3.1 Overview of Compilation/Linkage Commands....5
- 3.2 Compilation/Linkage Command Format....5

Chapter 4 MPI Program Execution....8
- 4.1 Execution Command Formats....8
- 4.2 MCA Parameters....12
- 4.3 Environment Variables....17
- 4.4 mpiexec Command Return Values....17
- 4.5 Notes....18

Chapter 5 Extended Interfaces....19
- 5.1 Rank Query Interface....19
  - 5.1.1 Querying the Number of Dimensions and Shape....19
    - 5.1.1.1 FJMPI_TOPOLOGY_GET_DIMENSION....19
    - 5.1.1.2 FJMPI_TOPOLOGY_GET_SHAPE....20
  - 5.1.2 Querying the Coordinates....21
    - 5.1.2.1 FJMPI_TOPOLOGY_GET_COORDS....21
  - 5.1.3 Querying the Rank....23
    - 5.1.3.1 FJMPI_TOPOLOGY_GET_RANKS....23
  - 5.1.4 Querying the Ranking of a Communicator that Has a Cartesian Structure....24
    - 5.1.4.1 FJMPI_TOPOLOGY_CART_REORDER....24
- 5.2 MPI Statistical Information Section Specifying Interface....25
  - 5.2.1 The MPI Statistical Information Section Specifying Routine....26
    - 5.2.1.1 FJMPI_COLLECTION_START....26
    - 5.2.1.2 FJMPI_COLLECTION_STOP....26
    - 5.2.1.3 FJMPI_COLLECTION_PRINT....26
    - 5.2.1.4 FJMPI_COLLECTION_CLEAR....27
- 5.3 Extended Persistent Communication Requests Interface....27
  - 5.3.1 Extended Persistent Communication Requests Interface Specifications....28
    - 5.3.1.1 FJMPI_PREQUEST_SEND_INIT....28
    - 5.3.1.2 FJMPI_PREQUEST_RECV_INIT....28
    - 5.3.1.3 FJMPI_PREQUEST_START....29
    - 5.3.1.4 FJMPI_PREQUEST_STARTALL....30
- 5.4 MPI Asynchronous Communication Promotion Section Specifying Interface....30
  - 5.4.1 The MPI Asynchronous Communication Promotion Section Specifying Routine....31
    - 5.4.1.1 FJMPI_PROGRESS_START....31
    - 5.4.1.2 FJMPI_PROGRESS_STOP....31
- 5.5 Persistent Collective Communication Request Interface....31
  - 5.5.1 Overview....31
  - 5.5.2 Persistent Collective Communication Request Interface Specification....32
- 5.6 Additional Predefined Datatype....39
  - 5.6.1 Overview....39
  - 5.6.2 Predefined Datatype for the Half-Precision Floating-Point Type....40

Chapter 6 Supplementary Items....41

6.1 Parallelizing Memory Copy Processing in MPI Library with Threads........41
6.2 Notes Concerning MPI Standard Specifications........41
6.2.1 Supported Level of MPI Standard........41
6.2.2 Predefined Datatypes that can be Used in This Software System........43
6.2.3 Reserved Communicators........48
6.2.4 Values of Constants Set in This Software System........48
6.2.5 Operations in a Multi-Threaded Environment........48
6.2.6 Signal Operation Changes........48
6.2.7 One-sided Communications........48
6.2.7.1 Assertions for Optimization........48
6.2.7.2 Info Argument........49
6.2.8 Establishing Communication between Groups not Sharing a Communicator........49
6.2.8.1 MPI_COMM_JOIN Return Value........49
6.2.9 Dynamic Process Creation........49
6.2.9.1 info Argument Value........50
6.2.9.1.1 Designation of Current Directory by wdir Key........50
6.2.9.1.2 Designation of Environment Variables by env Key........50
6.2.9.2 Search for Executable Files........50
6.2.9.3 Dynamic Process Creation for Java program........50
6.2.10 Notes on Send Buffer and Receive Buffer........51
6.2.11 MPI Input-Output........51
6.2.12 Use of the Profiling Interface........53
6.2.13 MPI Tool Information Interface........53
6.2.14 Routines implemented by macros........53
6.2.15 Arguments of User-defined Error Handlers........53
6.2.16 Collective Communications in Inter-communicator........53
6.3 Eager Protocol and Rendezvous Protocol........55
6.4 Reduction Operation Sequence Guarantee in Collective Communication........55
6.5 Process Creation from Inside an MPI Program........56
6.6 Suppressing Memory Usage........56
6.6.1 Influence of Dynamic Connection on Performance........56
6.7 Memory Usage of MPI Library........56
6.8 MPI_BCAST/MPI_IBCAST routines When the Same Count is Used among the Processes........57
6.9 Avoiding Concentration of Communications in MPI_GATHERV Routine........58
6.10 Dynamic Debug during MPI Program Execution........58
6.10.1 Communication Timeout Setting........59
6.10.2 Monitoring Incorrect Writing to MPI Communication Buffer........59
6.11 MPI Program Execution Using Slurm........60
6.11.1 Environment Check........60
6.11.2 Slurm Commands........60
6.11.3 MPI Program Execution Using Slurm........61
6.11.4 Binding CPU Resources to Processes........62
6.11.5 Other Notes........63
6.11.5.1 MCA Parameter Settings........63
6.11.5.2 MPMD Model Execution........63
6.11.6 Restrictions........63
6.11.6.1 Restricted MPI Routines To Use........63

Chapter 7 Error Messages........65
7.1 Output Format for Information Related to Parallel Processes........65
7.2 mpiexec Command Error Messages........65
7.3 Communication Library Error Messages........70
7.4 Compilation/linkage command Error Messages........74

Appendix A Error Class List........75

Glossary........78

Chapter 1 Overview

This software system is based on an open source MPI library (Open MPI).

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

1.1 Features of This Software System

This MPI library is intended for use with this computing system. MPI (Message Passing Interface) is the set of standards defined by the MPI Forum. The library interface for the Fortran and C language is regulated in MPI to enable parallel programming based on message passing in parallel computing systems with distributed memory.
It is assumed that the C language interface is also used for C++ programs.
In order to enable parallel MPI programming in Java, this software system specially provides the Java binding of MPI. However, OpenJDK must be installed in the compute nodes. If there is a login node in your environment, OpenJDK must also be installed in the login node. Refer to "Table 6.2 Routines not provided by Java binding of this software system", for routines of not support.

1.2 Outline of How to Use This Software System

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

This software 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 this computing system.

1.2.1 Flow from Compilation to Execution of an MPI Program

The operations required for the user to execute an MPI program in this system is described as follows.

Compiling and linking an MPI program

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

The compilation/linkage commands of Fortran, C, and C++ internally invoke the corresponding Fujitsu compiler commands. The compilation command of Java internally invokes the corresponding Java compiler command.

Some of these compilation/linkage commands are used at the login node (cross compiler) and the others are used at compute nodes (native compiler). If there is no login node in your environment, only the native compiler is available.

These compilation/linkage commands are shown below.

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

Table 1.1 Compilation/linkage commands

Type	Command name	Command of the corresponding Fujitsu/Java compiler	Programming language to write MPI program
Cross compiler	mpifrtpx	frtpx	Fortran
	mpifccpx	fccpx	C
	mpiFCCpx	FCCpx	C++
	mpijavac	javac	Java
Native compiler	mpifrt	frt	Fortran
	mpifcc	fcc	C
	mpiFCC	FCC	C++
		javac	Java

Type	Command name	Command of the corresponding Fujitsu/Java compiler	Programming language to write MPI program
	mpijavac		

Invoke these cross compiler compilation/linkage commands from at the login node. They can be used to convert MPI programs to a format that can be executed on this computing system. 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 native compiler compilation/linkage commands at the compute nodes. If there is a job scheduler in your environment, the user can also request the job scheduler to launch the job that executes the commands at the compute node. Refer to your job scheduler manual for information of the job scheduler.

The cross compiler and native compiler for MPI programs have no functional differences other than the type of the node where the command is executed. Refer to "Chapter 3 MPI Program Compilation/Linkage" for information on how to use native compiler compilation/linkage commands.

Executing an MPI program

Use the mpiexec command to execute an MPI program that has been converted to an executable file format using a compilation/linkage command. The mpiexec command must be executed from the compute node within this computing system. If there is a job scheduler in your environment, the user can also request the job scheduler to launch the job that executes the MPI program. Refer to your job scheduler manual for information of the job scheduler.

This software system has two communication ways between two parallel processes of an MPI program internally, InfiniBand communication and shared memory communication. The InfiniBand 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 software system.

"*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

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

```
/installation_path/bin
```

When the javac command placed other than /usr/bin is used, append the javac command path name to user environment variable PATH.

The following setting is required to compile/link an MPI program using the native compiler. If there is a job scheduler in your environment and a job to execute the compilation/linkage command of MPI program is launched, the following setting is required in the job script.

- Append the following path name to user environment variable PATH

```
/installation_path/bin
```

In order to compile a Java program with the javac command placed other than /usr/bin, append the javac command path name to user environment variable PATH when the compilation/linkage command execution job is launched.

Refer to your job scheduler manual for information on job launching and job scripts.

The MPI library resources are shown below.

Table 2.1 MPI library resource list

Name			Usage
C/C++	Fortran	Java	Path within () indicates the install location
mpi.h mpi-ext.h	mpif.h mpif-ext.h mpi.mod mpi_ext.mod mpi_f08.mod mpi_f08_ext.mod	-	When compiling: MPI library header file (for Fortran, the module information file is also included) (`/installation_path/include/mpi/fujitsu`)
-	-	mpi.jar	When compiling and executing: MPI library jar file (`/installation_path/lib64`)
mpifccpx mpiFCCpx mpifcc mpiFCC	mpifrtpx mpifrt	-	When compiling and linking: Fortran, C, and C++ compilation/linkage commands (`/installation_path/bin`)
-	-	mpijavac	When compiling: Java compilation command (`/installation_path/bin`)

2.2 MPI Program Execution Environment

The settings below are required if there is a job scheduler in your environment and it is used to launch an MPI program execution job. Refer to your job scheduler manual for information on job launching and job scripts.

- Append the following path name to user environment variable PATH

```
/installation_path/bin
```

  In order to execute a Java program with the java command placed other than /usr/bin, append the java command path name to user environment variable PATH.

- Append the following path name to user environment variable LD_LIBRARY_PATH

```
/installation_path/lib64
```

2.3 Online Manual

Related online manual is available on a compute node. Related online manual is also available on a login node if there is a login node in your environment.

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

- Append the following path name to user environment variable MANPATH

```
/installation_path/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 Software System", an MPI program is a Fortran, C, C++, or Java program that includes invocation of the MPI library.

Fujitsu compiler frtpx, fccpx, or FCCpx command (native compiler: frt, fcc, or FCC command) is used to compile and link ordinary Fortran, C, or C++ programs. However the compilation/linkage command mpifrtpx, mpifccpx, or mpiFCCpx (native compiler: mpifrt, mpifcc, or mpiFCC command) is used to compile and link MPI programs.

javac command is used to compile Java programs; however the compile command for MPI programs, mpijavac command is used to compile MPI programs.

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

mpijavac command is a wrapper command for javac command and internally invokes javac command included in JDK. Therefore, javac command options can be specified as is in the compilation command for MPI programs.

The conformance of types of arguments of MPI routine 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 software 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 for Fortran, C, or C++

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

Table 3.2 Compilation command format for Java

Command name	Options
`mpijavac`	`[--showme|--verbose|--help|-help|-h] [`*`javac_arguments`*`] file ...`

With this software 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 3.3 Compilation/linkage command options for Fortran, C, or C++

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:help`	Displays the help information. 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.
`-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.

Table 3.4 Compilation command options for Java

Option	Explanation
`--showme`	Displays the call line where the MPI program compilation command invokes the javac command. Actual compilation processing is not performed.
`--verbose`	Displays the call line where the MPI program compilation command invokes the javac command. Actual compilation processing is performed.

`--help`
`-help`
`-h`
	Displays the help information. Actual compilation processing is not performed.

`javac_arguments`	Specifies the options passed to the javac command.

Note

Compilation/linkage notes

- The MPI library is provided in only the dynamic link library format.
- mpifrtpx command and mpifrt command automatically specify the following frtpx command and frt command option:
  - -f2004
  - -Knointentopt (-Kintentopt option is disable even if it is specified)
- When the following option of frtpx command and frt command is specified with mpifrtpx command and mpifrt command, the language entities must be lowercase letter:
  - -AU
- The mpif.h file cannot be included by the INCLUDE line more than once in one scoping unit of a Fortran program.
- Java programs can be compiled without setting the classpath of the MPI library.
  In order to compile an MPI program with jar files, set the path of the MPI program in the environment variable CLASSPATH or the -classpath option.
  If both the environment variable CLASSPATH and the -classpath option are set, the value specified for -classpath option takes priority.

Example

Examples of using mpifrtpx command 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 software system, mpiexec command is used to execute MPI programs. If there is a job scheduler in your environment and it is used to launch an MPI program execution job, mpiexec command passes control to the job scheduler. Process creation and execution of the MPI program takes place on the compute node. Refer to your job scheduler manual for information on the job scheduler.

Almost all options or the like of Open MPI on which this software system is based can be specified. However, the behavior of an option or the like not described in this chapter is not only not guaranteed but may also have a serious impact on this system. Therefore, do not use options or the like which are not described in this chapter especially when executing an MPI program using many nodes or many processes.

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.
- The options that can be specified for the *local_options* of the mpiexec command are only -tune option and { -mca | --mca } option.
- The { -mca | --mca } option in *local_options* cannot be specified in the execution definition file.
- When -tune option is specified for both the mpiexec command and the execution definition file, only that specified in the execution definition file is enabled.

- When the same MCA parameter is specified for both the { -mca | --mca } option of the mpiexec command and the AMCA parameter file (MCA parameter settings file) of -tune option in the execution definition file, the value specified by the mpiexec command 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 command, 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 execution of a Java program in the MPMD model is not guaranteed.

The format for *global_options* and explanations of options that can be specified as *global_options* are shown under "global_options format and explanation".

The format for *local_options* and explanations of options that can be specified at *local_options* are shown under "local_options format and explanation".

Explanations of all options are provided in "Execution command options".

In this software system, some variables, known as MCA parameters, are held internally by the MPI library. The operating conditions that apply when this software 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 command to execute an MPI program

1. Example of execution command specification in the SPMD model

```
$ mpiexec ./a.out
```

   Execute the MPI program executable file a.out.

2. Example of execution command specification in the 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.

4. Example of specifying execution command in the SPMD model for a Java program

```
$ mpiexec -n 8 java -classpath ./dir JavaTest
```

   Executes the JavaTest.class as the Java class file of the MPI program with 8 processes.

   The classpath is specified by -classpath option of java command.

global_options format and explanation

```
[ { -app | --app } APP_FILE ]
[ { -h | --help } ]
[ { -V | --version } ]
```

{ -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.

{ -h | --help }

Displays help messages for this command and ends mpiexec command.

{ -V | --version }

Displays version information for this command and ends mpiexec command.

local_options format and explanation

```
[ -tune  AM_FILE ]
[ -x  NAME=VALUE ]
[ { -mca | --mca } MCA_PARAM_NAME  MCA_PARAM_VALUE ]
[ { -c | -np | --np | -n | --n }  N ]
[ { -hostfile | --hostfile }  HOSTFILE ]
```

-tune *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 for the same MPI program, 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 for the same MPI program, 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 the SPMD model is used for execution and this option is omitted, it is assumed that the maximum number of parallel processes that can be created (If there is a job scheduler in your environment, it is decided by the job scheduler. If there is no job scheduler in your environment, it is 48.) is specified.

This option must be specified when the MPMD model is used for execution.

If this option is specified more than once for the same MPI program, the parameter specified last takes priority.

{ -hostfile | --hostfile } *HOSTFILE*

Specifies the path name of a host file where host names or IP addresses of the compute nodes where parallel MPI processes are launched are written. Specifying this option is necessary to execute an MPI program using multiple nodes.

The format of each line in a host file is either of the following Format 1 or 2.

- Format 1: Specifies the host name of a compute node and the maximum number of processes which can be launched on the node.

```
hostname slots=max_num_processes
```

- Format 2: Specifies the IP address of a compute node and the maximum number of processes which can be launched on the node.

```
IP_address slots=max_num_processes
```

```
Example:
hostname1 slots=4
aaa.bbb.ccc.ddd slots=2
hostname2 slots=8
```

- The total of values which are specified for the slots in the host file must be greater than or equal to the number of processes to be launched.
- The compute node where processes are launched, the number of processes which are launched on the node, and the ranks of the processes are decided in accordance with the order of items written in the specified host file. For example, when 12 processes are launched in the above example, the 4 processes whose rank are 0 to 3 are launched on the node whose host name is *hostname1*. The 2 processes whose rank s are 4 and 5 are launched on the node whose IP address is *aaa.bbb.ccc.ddd*, and the 6 processes whose ranks are 6 to 11 are launched on the node whose host name is *hostname2*.

Execution command options

execfile

Specifies an executable file of an MPI program, an executable file other than an MPI program, or a shell script.

Specifies java command when an MPI program is written in Java.

If a specified executable file or a shell script is not exist in path which is specified for the environment variable PATH, the absolute path or the relative path from the current directory where the mpiexec command is executed must be set.

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 command cannot be specified. If it is executed recursively, an error message is output and then mpiexec command ends abnormally. For example, the mpiexec command 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*.

For Java programs, specify the options to be passed to java command such as class name or -jar option.

execfile1
execfile2
execfile3

Specifies an executable file of an MPI program, an executable file other than an MPI program, or a shell script.

For execution in the MPMD model, specify multiple executable files of MPI programs or shell scripts by delimiting them by colons.

If a specified executable file or a shell script is not exist in path which is specified for the environment variable PATH, the absolute path or the relative path from the current directory where the mpiexec command is executed must be set.

A shell script cannot be coded to execute more than one MPI program. The shell script behaviour is not guaranteed if it contains code to execute more than one MPI program.

Recursive execution of mpiexec command cannot be specified. If it is executed recursively, an error message is output and then mpiexec command ends abnormally. For example, the mpiexec command start filename cannot be specified for *execfile1, execfile2, execfile3.*

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.*

4.2 MCA Parameters

When an MPI program is executed in this software 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 software system, MCA parameters can be specified in the following ways:

- Use the -tune option of mpiexec command to specify the parameters in the MCA parameter settings file (AMCA parameter file).
- Use the -mca option of mpiexec command 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 4.4 MCA parameter specification methods and priorities

Rank	MCA parameter setting method	Example of use
1	-mca option of mpiexec command	-mca opal_abort_print_stack 0
2	Environment variable	export OMPI_MCA_opal_abort_print_stack=0
3	AMCA parameter file (MCA parameter settings file) specified in the -tune option	-tune mca_file

Note: A smaller priority value indicates a higher priority.

MCA parameters

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

If only an integer value is specified, it is assumed to be in bytes. If 'k' is concatenated after an integer, the value is assumed to be in units of KiB. If 'm' is concatenated, the value is assumed to be in units of MiB.

Table 4.5 btl_openib_autoresize_memory_threshold (Specifies the upper limit of the amount of memory that the MPI library can use for InfiniBand communication resources)

MCA parameter value	Content
Integer value from 1073741824 to 21474836480	Specifies the upper limit of the amount of memory per compute node that the MPI library can use for the InfiniBand communication resources. When the MPI library automatically sets the number of SRQ buffers (*1) according to the number of processes, the setting is decided so that the memory usage is less than or equal to the value specified for this parameter if possible. Refer to "Table 4.8 btl_openib_receive_queues_default_num_bufs (changes the number of SRQ buffers)" for details about the setting of the number of SRQ buffers. The default value for this parameter is 10737418240.
Values other than the above	The specified value is assumed to be 10737418240.

*1: SRQ buffers is buffers to receive MPI messages using Shared Receive Queue of InfiniBand.

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

Table 4.6 btl_openib_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. 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). The default value of this parameter and the upper limit of values that can be specified are different whether the MCA parameter btl_openib_max_send_size is specified when executing an MPI program. - If the MCA parameter btl_openib_max_send_size is not specified, the default value of the MCA parameter btl_openib_eager_limit and the upper limit of values that can be specified are 65536. - If the MCA parameter btl_openib_max_send_size is specified, the value specified for the parameter is also the default value of the MCA parameter btl_openib_eager_limit and the upper limit of values that can be specified. Refer to "Table 4.7 btl_openib_max_send_size (changes the maximum segment size for pipeline transfer method in point-to-point communication)" for details of the MCA parameter btl_openib_max_send_size. Note that specifying a larger value for this parameter may cause memory shortage. Refer to "6.3 Eager Protocol and Rendezvous Protocol" for details.

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

Table 4.7 btl_openib_max_send_size (changes the maximum segment size for pipeline transfer method in point-to-point communication)

MCA parameter value	Content
Integer value of 1 or more	Specifies the maximum segment size for sending a message using pipeline transfer method in point-to-point communication .

MCA parameter value	Content
	The value of this parameter must be greater than or equal to that of the MCA parameter btl_openib_eager_limit. Refer to "Table 4.6 btl_openib_eager_limit (changes the threshold value for switching the communication method)" for details. The default value for this parameter is 65536.

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

Table 4.8 btl_openib_receive_queues_default_num_bufs (changes the number of SRQ buffers)

MCA parameter value	Content
(This parameter is not specified.)	Specifies that the MPI library automatically sets the number of SRQ buffers (*1) according to the number of processes. Since the MPI library tries to improve communication performance while keeping memory usage below a certain value, a memory shortage is less likely to occur even when the number of processes is large. Communication performance may improve when the value 1024 or more is specified for this parameter. However, note a memory shortage.
-1	Specifies that the MPI library automatically sets the number of SRQ buffers (*1) regardless of the number of processes. A memory shortage may occur.
1024 to 16384	Specifies the number of SRQ buffers (*1). When receiving messages from many processes, communication performance may improve by specifying a value greater than 1024 for this parameter. However, memory usage also increases.
Values other than the above	Sets the number of SRQ buffers assuming that 1024 is specified for this parameter.

*1: SRQ buffers is buffers to receive MPI messages using Shared Receive Queue of InfiniBand.

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

Table 4.9 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, MPI_IREDUCE_SCATTER_BLOCK, and MPI_SCAN routines 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 "6.4 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 4.10 coll_tuned_bcast_same_count (achieves faster communication when MPI_BCAST/MPI_IBCAST routines are used with the same count among the processes)

MCA parameter value	Content
1	Specifies to achieve faster communication when MPI_BCAST routine or MPI_IBCAST routine is used with the same count among the processes. This MCA parameter also affects the MPI_ALLGATHER and MPI_ALLGATHERV routines. Refer to "6.8 MPI_BCAST/MPI_IBCAST routines 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 4.11 coll_basic_gatherv_sync (synchronizes periodically in MPI_GATHERV)

MCA parameter value	Content
1	Enables the function to avoid the concentration of communications by synchronizing periodically in the MPI GATHERV routine. Communication performance of the MPI_GATHERV routine may improve by avoiding the concentration of communications. On the other hand, note that this function may also decrease communication performance of the routine due to the overhead of the synchronization processing. Refer to "6.9 Avoiding Concentration of Communications in MPI_GATHERV Routine" for details.
0	Specifies that periodic synchronization processing is disabled inside the MPI_GATHERV routine. The default value of this parameter is 0.

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

Table 4.12 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 4.13 mpi_check_buffer_write (monitors incorrect writing to communication buffers)

MCA parameter value	Content
1	Specifies to monitor whether there is incorrect writing, which is data writing to a send buffer which is in use for a nonblocking communication. If data is written in the send buffer before completing the send operation, a message and stack trace information are output to the standard error output, and the MPI program execution ends. Refer to "6.10.2 Monitoring Incorrect Writing to MPI Communication Buffer" for details.
0	Specifies not to monitor whether there is incorrect writing to a communication buffer. The default value for this parameter is 0.

Table 4.14 mpi_java_eager (Specifies the size of temporary buffer for Java program)

MCA parameter value	Content
Integer value of 1 or more	Specifies the size (in bytes) of the temporary buffer to be allocated for Java programs. The default value for this parameter is 65536 (64 KiB).

MCA parameter value	Content
	Using the temporary buffer secured in advance, the time to copy data between the Java heap area and the C heap area can be reduced, and the execution time of blocking communication routines can also be reduced. The roughly estimated value to be set for this parameter is the size of data which are sent or received in a routine call. In other words, it is the product of the datatype size and the number of elements specified for the routine. If the value of this parameter is less than the roughly estimated value, the execution time of blocking communication routines may increase because the number of times to secure buffers increase. If the value of this parameter is greater than the roughly estimated value, the execution time of blocking communication routines is rarely improved. Note that this parameter is enabled only when a primitive datatype is specified for a blocking communication routine. Also note that specifying a larger value for this parameter may cause memory shortage.

Table 4.15 opal_abort_delay (delays program termination when an error is detected)

MCA parameter value	Content
A positive integer value	If the MPI_ABORT routine is called or the MPI library detects an error in an MPI program execution, termination of the program is delayed by the specified time (seconds).
0	If the MPI_ABORT routine is called or the MPI library detects an error in an MPI program execution, the program ends immediately. The default value for this parameter is 0.

Table 4.16 opal_abort_print_stack (outputs stack trace information)

MCA parameter value	Content
1	If MPI_ABORT routine 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 4.17 opal_mt_memcpy (parallelizes some memory copy processings performed in the MPI library using multiple threads)

MCA parameter value	Content
1	Specifies that some memory copy processings in the MPI library are parallelized with multiple threads depending on the conditions. If the specified value is less than 0 or greater than 1, the specified value is assumed to be 1. Whether actual thread parallelization is performed or not is decided by the MPI library depending on the conditions. The number of threads used for thread parallelization is specified by the user. Processings of the following MPI routines can be parallelized with threads when this function is enabled. - MPI_PACK and MPI_UNPACK routines - MPI routines for point-to-point communication - MPI routines for collective communication and one-sided communication that point-to-point communication is performed as a part of processings in the MPI library

MCA parameter value	Content
	Refer to "6.1 Parallelizing Memory Copy Processing in MPI Library with Threads" for details.
0	Specifies that all processings in the MPI library are performed with only a thread where an MPI routine is called. The default value for this parameter is 0.

Table 4.18 opal_progress_timeout (specifies the timeout time in communication wait)

MCA parameter value	Content
A positive integer value	Specifies the timeout time (in seconds) for the communication timeout setting function. If the wait time of an MPI communication exceeds the time (in seconds) specified for this parameter, a message and stack trace information are output to the standard error, and the MPI program execution ends. Refer to "6.10.1 Communication Timeout Setting" for details.
0	Specifies to disable the communication timeout setting function. The default value for this parameter is 0.

4.3 Environment Variables

In this software system, environment variables can be used to control the behavior of the MPI program.

Dynamically created parallel processes inherit environment variables of the root process of the original parallel processes that created them. Do not set environment variables which have names starting with the string "OMPI_MCA" in the program of the original parallel processes.

Environment variables with name starting with the "OMPI_"

These environment variables can control the behavior when the MPI library is executed.

The environment variables provided by this software 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 Command Return Values

In principle, the mpiexec command return values are the values that the user has specified in the MPI program or the values set by the language processing systems.

- If there are multiple MPI program processes, the return value of the first process identified internally becomes the mpiexec command 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 created parallel process ended abnormally, the return value becomes the return value of the abnormally ended dynamic process.
- If this software system ends abnormally, the return values specified by this software system become the return values.

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

Table 4.19 mpiexec command return values reserved in this software system

mpiexec command return value	Explanation
1	Indicates occurrence of an mpiexec command option settings error, an internal inconsistency within this software system, or a fatal error within an MPI routine.
2 to 92	If there is an error class regulated by the MPI standard within an MPI routine 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 standard.
Logical sum of signal number and 0x80	Indicates the return value if mpiexec command 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.

4.5 Notes

- It is necessary that the MPI program executable file must be copied to a network file system such as NFS or must be copied to a local file system of all compute nodes which are used at the time of the execution.
- It is necessary that each compute node can connect to all other compute nodes using SSH or the like without input of passwords or passphrases in order to execute an MPI program using multiple nodes. In addition, it is also necessary to specify the compute nodes to be used using the -hostfile option or the --hostfile option.
- When an MPI program is executed using multiple compute nodes, the environment variables which are set on the compute nodes where the mpiexec command is executed are not automatically passed to MPI parallel processes on other compute nodes. Specify the -x option in order to set the same environment variables for all MPI parallel processes. Refer to "local_options format and explanation" for details of the -x option.

Chapter 5 Extended Interfaces

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

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

Information

All extended interfaces support the C language and Fortran.
The C++ program can use the C language interface.
In the Fortran, either "USE mpi_f08_ext" or "USE mpi_ext", which respectively correspond to "USE mpi_f08" and "USE mpi" defined in the MPI standard, can be used. In addition, "INCLUDE 'mpif-ext.h'" can be used instead of "USE mpi_ext".

5.1 Rank Query Interface

The software system in the PRIMEHPC FX1000 can execute MPI programs in logical node space that has a torus structure of from one to three dimensions. Job Operation Software in the PRIMEHPC FX1000 allocates logical coordinates in this logical node space.

It may be useful to know from within the MPI program the position (coordinates) where each parallel process rank of the MPI program is deployed. For example, it is possible for a user to write a communication performance aware program such that communications are performed among processes on compute nodes where the physical distance is close to each other as much as possible.

The software system in the PRIMEHPC FX1000 has the rank query interface to get information such as coordinates where each parallel process rank is deployed and the shape formed by the coordinates of the processes. This software system also has the rank query interface for API compatibility with the PRIMEHPC FX1000. However, note that useful information for programming cannot be gotten in this software system even if the interface is used.

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

Table 5.1 Rank query interface routine list

Routine name	Routine 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_GET_COORDS	Gets the coordinate values from the rank
FJMPI_TOPOLOGY_GET_RANKS	Gets the ranks from the coordinates
FJMPI_TOPOLOGY_CART_REORDER	Gets the value that determines the rank of a communicator with a Cartesian structure

5.1.1 Querying the Number of Dimensions and Shape

5.1.1.1 FJMPI_TOPOLOGY_GET_DIMENSION

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Topology_get_dimension(size, ierror)
INTEGER, INTENT(OUT) :: size
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_TOPOLOGY_GET_DIMENSION(SIZE, IERROR)
INTEGER SIZE, IERROR
```

<Explanation>

This query returns the number of dimensions in the process shape assuming that MPI processes are deployed on some coordinates. The MPI processes which form the shape belong to the MPI_COMM_WORLD created when the MPI_INIT routine is called. When this routine returns to its caller successfully, the number of dimensions stored to the size argument is 1 because this software system assumes that MPI processes are deployed on one-dimensional coordinates.

Type	Variable	Explanation	IN/OUT
int*	size	Number of dimensions in the process shape of processes belonging to MPI_COMM_WORLD, which is 1 in this software system	OUT

<Return value>

Normal	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_INVALID_COMM	If this routine was called from a dynamically created MPI process
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	This value is never returned in this software system

<Notes>

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

- This routine is called before the MPI_INIT routine is executed
- This routine is called after the MPI_FINALIZE routine is executed

5.1.1.2 FJMPI_TOPOLOGY_GET_SHAPE

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Topology_get_shape(x, y, z, ierror)
INTEGER, INTENT(OUT) :: x, y, z
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_TOPOLOGY_GET_SHAPE(X, Y, Z, IERROR)
INTEGER X, Y, Z, IERROR
```

<Explanation>

This query returns the MPI parallel process shape XYZ assuming that MPI processes are deployed on some coordinates. The MPI processes which form the shape belong to the MPI_COMM_WORLD created when the MPI_INIT routine is called. When this routine returns to its caller successfully, a value is stored in the argument x only because this software system assumes that MPI processes are deployed on one-dimensional coordinates.

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	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_INVALID_COMM	If this routine was called from a dynamically created MPI process
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	This value is never returned in this software system

<Notes>

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

- This routine is called before the MPI_INIT routine is executed
- This routine is called after the MPI_FINALIZE routine is executed

5.1.2 Querying the Coordinates

5.1.2.1 FJMPI_TOPOLOGY_GET_COORDS

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Topology_get_coords(MPI_Comm comm, int rank, int view, int maxdims, int coords[])
```

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Topology_get_coords(comm, rank, view, maxdims, coords, ierror)
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, INTENT(IN) :: rank, view, maxdims
INTEGER, INTENT(OUT) :: coords(maxdims)
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_TOPOLOGY_GET_COORDS(COMM, RANK, VIEW, MAXDIMS, COORDS, IERROR)
INTEGER COMM, RANK, VIEW, MAXDIMS, COORDS(*), IERROR
```

<Explanation>

This routine gets the coordinates corresponding to the rank of a process in the specified communicator. Although this software system assumes that MPI processes are deployed on one-dimensional coordinates, there are the following three types of concept about the coordinates in the PRIMEHPC FX1000.

- The three-dimensional coordinates which are consisted of logical X, Y, and Z axes, which represent the logical positions of MPI processes
- The six-dimensional coordinates which are consisted of X, Y, Z, A, B, C axes based on the specification of the Tofu D interconnect, which represent the physical positions of MPI processes

- The six-dimensional coordinates which represent the relative positions of MPI processes on the basis of the coordinates of a certain process rank

Therefore, this routine has the view argument to specify the type of coordinates to get. In this software system, this routine gets the coordinates assuming that the MPI processes are deployed on six-dimensional coordinates only when the FJMPI_TOFU_SYS or FJMPI_TOFU_REL is specified for the view argument of this routine for API compatibility with the PRIMEHPC FX1000. However, note that the coordinates when the FJMPI_TOFU_REL is specified for the view argument is same as the coordinates when the FJMPI_TOFU_SYS is specified.

Type	Variable	Explanation	IN/OUT
MPI_Comm	comm	Specify the communicator.	IN
int	rank	Specify the rank of a process in the communicator comm.	IN
int	view	Specify the macro to choose the type of coordinates which is gotten using this routine. - FJMPI_LOGICAL The coordinates when it is assumed that processes are deployed on one-dimensional coordinates. - FJMPI_TOFU_SYS The coordinates when it is assumed that processes are deployed on six-dimensional coordinates. - FJMPI_TOFU_REL The coordinates when it is assumed that processes are deployed on six-dimensional coordinates, which is same as the coordinates when the FJMPI_TOFU_SYS is specified for this argument.	IN
int	maxdims	Specify the number of dimensions of the coordinates. If the view argument is FJMPI_LOGICAL: Specify a number which is greater than 0 and less than 4. If the view argument is not FJMPI_LOGICAL: Specify a number which is greater than 0 and less than 7.	IN
int[]	coords	Array of the coordinates corresponding to the communicator and rank.	OUT

- To get the coordinates when it is assumed that processes are deployed on one-dimensional coordinates
  Specify FJMPI_LOGICAL for the view argument, and a number which is greater than 0 and less than 4 for the maxdims argument. The coordinate X of the node which corresponds to the arguments comm and rank is stored in the coords[0] argument. In this software system, note that no values are stored in the coords[1] and coords[2] when FJMPI_LOGICAL is specified for the view argument.

- To get the coordinates when it is assumed that processes are deployed on six-dimensional coordinates
  Specify FJMPI_TOFU_SYS or FJMPI_TOFU_REL for the view argument, and 6 for the maxdims argument. The Tofu coordinates of the node which corresponds to the arguments comm and rank are stored in the coords argument. The coordinates X, Y, Z, A, B, C are stored in coords[0], coords[1], coords[2], coords[3], coords[4], coords[5], respectively.

<Return value>

Normal	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	This value is never returned in this software system

<Notes>

The number of elements in the array specified for the coords argument must be greater than or equal to the value specified for the maxdims argument.

The specified value for the argument rank must correspond to the rank of a process in the communicator specified for the argument comm.

If FJMPI_LOGICAL is specified for the argument view and the value 2 or 3 is specified for the argument maxdims, the behavior is same as when the value 1 is specified for the argument maxdims.

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

- This routine is called before the MPI_INIT routine is executed
- This routine is called after the MPI_FINALIZE routine is executed

In this software system, when an inter-communicator is specified for the comm argument, this routine does not get the coordinates but returns FJMPI_SUCCESS.

5.1.3 Querying the Rank

5.1.3.1 FJMPI_TOPOLOGY_GET_RANKS

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Topology_get_ranks(MPI_Comm comm, int view, int coords[], int maxppn, int *outppn, int
ranks[])
```

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Topology_get_ranks(comm, view, coords, maxppn, outppn, ranks, ierror)
TYPE(MPI_Comm), INTENT(IN) :: comm
INTEGER, INTENT(IN) :: view, coords(*), maxppn
INTEGER, INTENT(OUT) :: outppn, ranks(maxppn)
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_TOPOLOGY_GET_RANKS(COMM, VIEW, COORDS, MAXPPN, OUTPPN, RANKS, IERROR)
INTEGER COMM, VIEW, COORDS(*), MAXPPN, OUTPPN, RANKS(*), IERROR
```

<Explanation>

This routine gets the ranks of the processes which are in the specified communicator and are assumed to be deployed on the specified coordinates. The maximum number of ranks obtained by this routine is same as the value specified for the argument maxppn. The number of ranks actually gotten is stored in the outppn argument.

Examples of how to use this routine are shown below.

- When processes are assumed to be deployed on one-dimensional coordinates and one process is on the specified coordinates

  Specify FJMPI_LOGICAL for the view argument, the number greater than 0 for the maxppn argument, the coordinate X for the coords[0] arguments, respectively. The rank of the process which is assumed to be allocated to the specified arguments comm and coords is stored in ranks[0] of the array specified for the argument ranks. The value 1 is stored in the outppn argument.

- When processes are assumed to be deployed on six-dimensional coordinates and four processes are on the specified coordinates

  Specify FJMPI_TOFU_SYS or FJMPI_TOFU_REL for the view argument, the number greater than 4 for the maxppn argument, the six-dimensional coordinates X, Y, Z, A, B, C for the coords[0], coords[1], coords[2], coords[3], coords[4], coords[5] arguments, respectively. The ranks of the processes which are assumed to be allocated to the specified arguments comm and coords are stored in ranks[0], ranks[1], ranks[2], and ranks[3] of the array specified for the argument ranks. The value 4 is stored in the outppn argument.

Type	Variable	Explanation	IN/OUT
MPI_Comm	comm	Specify the communicator.	IN
int	view	Specify the macro to choose the type of coordinates which is gotten using this routine.	IN

Type	Variable	Explanation	IN/OUT
		- FJMPI_LOGICAL The coordinates when it is assumed that processes are deployed on one-dimensional coordinates. - FJMPI_TOFU_SYS The coordinates when it is assumed that processes are deployed on six-dimensional coordinates. - FJMPI_TOFU_REL The coordinates when it is assumed that processes are deployed on six-dimensional coordinates, which is same as the coordinates when the FJMPI_TOFU_SYS is specified for this argument.	
int[]	coords	Specify the coordinates where processes which have ranks to get are assumed to be allocated.	IN
int	maxppn	Specify the maximum number of processes which have ranks to get.	IN
int*	outppn	The number of ranks actually stored.	OUT
int[]	ranks	The array to store ranks.	OUT

<Return value>

Normal	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_NO_PROCESS	There is no process in the specified coordinates
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	This value is never returned in this software system

<Note>

If FJMPI_LOGICAL is specified for the view argument, the number of values set in the array specified for the coords argument must be greater than or equal to 1. If FJMPI_TOFU_SYS or FJMPI_TOFU_REL is specified for the view argument, six values must be set in the array specified for the coords argument. The operation is not guaranteed if the values set in the array specified for the coords argument are incorrect.

In order to get the actual number of processes which are allocated to the coordinates specified for the coords argument, refer to the value set in the outppn argument after calling this routine with specifying a large value for the maxppn argument.

The value stored in the outppn argument can be less than the value specified for the maxppn argument depending on the number of processes actually existing. In this case, the number of updated values in the array specified for the ranks argument is same as the value of the outppn.

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

- This routine is called before the MPI_INIT routine is executed
- This routine is called after the MPI_FINALIZE routine is executed

In this software system, when an inter-communicator is specified for the comm argument, this routine does not get the ranks but returns FJMPI_SUCCESS.

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

5.1.4.1 FJMPI_TOPOLOGY_CART_REORDER

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Topology_cart_reorder(comm, reorder, ierror)
TYPE(MPI_Comm), INTENT(IN) :: comm
LOGICAL, INTENT(OUT) :: reorder
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_TOPOLOGY_CART_REORDER(COMM, REORDER, IERROR)
INTEGER COMM, IERROR
LOGICAL REORDER
```

<Explanation>

In the PRIMEHPC FX1000, 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. However, note that the value of the reorder argument is always 0 because ranking is not performed in this software system.

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	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_INVALID_COMM	This value is never returned in this software system

<Notes>

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

- This routine is called before the MPI_INIT routine is executed
- This routine is called after the MPI_FINALIZE routine is executed

5.2 MPI Statistical Information Section Specifying Interface

The software system in the PRIMEHPC FX1000 has the MPI statistical information section specifying interface to specify the range to measure the communication data of MPI. This software system also has the MPI statistical information section specifying interface for API compatibility with the PRIMEHPC FX1000. However, note that useful information for programming cannot be gotten in this software system even if the interface is used.

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

Table 5.2 Overview of MPI statistical information section specifying routines list

Routine name	Routine 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

5.2.1 The MPI Statistical Information Section Specifying Routine

5.2.1.1 FJMPI_COLLECTION_START

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Collection_start()
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_COLLECTION_START()
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.2.1.2 FJMPI_COLLECTION_STOP

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Collection_stop()
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_COLLECTION_STOP()
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.2.1.3 FJMPI_COLLECTION_PRINT

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Collection_print(str)
CHARACTER(LEN=*), INTENT(IN) :: str
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_COLLECTION_PRINT(STR)
CHARACTER*(*) STR
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.2.1.4 FJMPI_COLLECTION_CLEAR

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Collection_clear()
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_COLLECTION_CLEAR()
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.3 Extended Persistent Communication Requests Interface

In the PRIMEHPC FX1000, 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. This software system also has this interface for API compatibility with the PRIMEHPC FX1000. However, in this software system, note that the behavior of each routine of this interface is same as the corresponding MPI routine.

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

Table 5.3 Extended persistent communication requests interface routine list

Routine name	Routine 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

5.3.1 Extended Persistent Communication Requests Interface Specifications

5.3.1.1 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 (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Prequest_send_init(buf, count, datatype, dest, tag, comm, request, ierror)
TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: buf
INTEGER, INTENT(IN) :: count, dest, tag
TYPE(MPI_Datatype), INTENT(IN) :: datatype
TYPE(MPI_Comm), INTENT(IN) :: comm
TYPE(MPI_Request), INTENT(OUT) :: request
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
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 and behavior of this routine are same as those of the MPI_SEND_INIT routine.

Like a send request created with the MPI_SEND_INIT routine, a send request created with this routine can be used by other MPI routines except for the conditions described in the notes.

<Return value>

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

<Notes>

- The request created with this routine can be initiated only by the FJMPI_PREQUEST_START, the FJMPI_PREQUEST_STARTALL, the MPI_START, or the MPI_STARTALL routine.
- The communication using the request created with this routine must be received using the request created with the FJMPI_PREQUEST_RECV_INIT or the FJMPI_RECV_INIT routine.

5.3.1.2 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 (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Prequest_recv_init(buf, count, datatype, source, tag, comm, request, ierror)
TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buf
INTEGER, INTENT(IN) :: count, source, tag
TYPE(MPI_Datatype), INTENT(IN) :: datatype
TYPE(MPI_Comm), INTENT(IN) :: comm
TYPE(MPI_Request), INTENT(OUT) :: request
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
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 and behavior of this routine are same as those of the MPI_RECV_INIT routine.

Like a receive request created with the MPI_RECV_INIT routine, a receive request created with this routine can be used by other MPI routines except for the conditions described in the notes.

<Return value>

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

<Notes>

- The communication using the request created with this routine must be received using the request created with the FJMPI_PREQUEST_SEND_INIT or the MPI_SEND_INIT routine.
- The request created with this routine can be initiated only by the FJMPI_PREQUEST_START, the FJMPI_PREQUEST_STARTALL, the MPI_START, or the MPI_STARTALL routine.

5.3.1.3 FJMPI_PREQUEST_START

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Prequest_start(request, ierror)
TYPE(MPI_Request), INTENT(INOUT) :: request
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

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

<Explanation>

The arguments and behavior of this routine are same as those of the MPI_START routine.

<Return value>

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

<Notes>

- Only communication requests that were created by the FJMPI_PREQUEST_SEND_INIT, the FJMPI_PREQUEST_RECV_INIT, the MPI_SEND_INIT, or the MPI_RECV_INIT routine can be passed to this routine.

5.3.1.4 FJMPI_PREQUEST_STARTALL

<Format>

C language format

```
#include <mpi-ext.h>
int FJMPI_Prequest_startall(int count, MPI_Request array_of_requests[])
```

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Prequest_startall(count, array_of_requests, ierror)
INTEGER, INTENT(IN) :: count
TYPE(MPI_Request), INTENT(INOUT) :: array_of_requests(count)
INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

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

<Explanation>

The arguments and behavior of this routine are same as those of the MPI_STARTALL routine.

<Return value>

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

<Notes>

- Only communication requests that were created by the FJMPI_PREQUEST_SEND_INIT, the FJMPI_PREQUEST_RECV_INIT, the MPI_SEND_INIT, or the MPI_RECV_INIT routine can be passed to this routine.

5.4 MPI Asynchronous Communication Promotion Section Specifying Interface

In the PRIMEHPC FX1000, in addition to calculation cores which are dedicated to execution of user programs, there are cores called assistant core which are dedicated to the OS and IO processing. The MPI asynchronous communication promotion section specifying interface is a routine to specify the range to promote asynchronous communication using an assistant core. This software system also has this interface for API compatibility with the PRIMEHPC FX1000. However, note that asynchronous communication in an MPI program is not promoted in this software system even if this interface is used.

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

Table 5.4 Overview of asynchronous communication promotion section specifying routines list

Routine name	Routine overview
FJMPI_PROGRESS_START	Starts the promotion of asynchronous communication
FJMPI_PROGRESS_STOP	Stops the promotion of asynchronous communication

5.4.1 The MPI Asynchronous Communication Promotion Section Specifying Routine

5.4.1.1 FJMPI_PROGRESS_START

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Progress_start()
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_PROGRESS_START()
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.4.1.2 FJMPI_PROGRESS_STOP

<Format>

C language format

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

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
FJMPI_Progress_stop()
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
FJMPI_PROGRESS_STOP()
```

<Explanation>

This routine does nothing in this software system.

<Return value>

None.

5.5 Persistent Collective Communication Request Interface

5.5.1 Overview

In this software system, the interfaces of persistent collective communication request included in the draft specification (https://www.mpi-forum.org/docs/drafts/mpi-2019-draft-report.pdf) for the MPI-4.0 Standard are implemented with routine names prefixed with MPIX_ instead of MPI_.

5.5.2 Persistent Collective Communication Request Interface Specification

The list of persistent collective communication request routines implemented in this software system follows below.

See the MPI standard draft for the behavior and the meaning of arguments of these routines.

<Format>

C language format

```
#include <mpi-ext.h>
int MPIX_Allgather_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Allgatherv_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Allreduce_init(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype,
MPI_Op op,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Alltoall_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Alltoallv_init(const void *sendbuf, const int sendcounts[], const int sdispls[],
MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int rdispls[], MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Alltoallw_init(const void *sendbuf, const int sendcounts[], const int sdispls[], const
MPI_Datatype sendtypes[],
    void *recvbuf, const int recvcounts[], const int rdispls[], const MPI_Datatype recvtypes[],
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Barrier_init(MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Bcast_init(void *buffer, int count, MPI_Datatype datatype, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Exscan_init(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op
op,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Gather_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Gatherv_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Reduce_init(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op
op, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Reduce_scatter_init(const void *sendbuf, void *recvbuf, const int recvcounts[],
MPI_Datatype datatype, MPI_Op op,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Reduce_scatter_block_init(const void *sendbuf, void *recvbuf, int recvcount, MPI_Datatype
datatype, MPI_Op op,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Scan_init(const void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Scatter_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype, void *recvbuf,
int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Scatterv_init(const void *sendbuf, const int sendcounts[], const int displs[],
MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype, int root,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Neighbor_allgather_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Neighbor_allgatherv_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Neighbor_alltoall_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, int recvcount, MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Neighbor_alltoallv_init(const void *sendbuf, const int sendcounts[], const int sdispls[],
MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int rdispls[], MPI_Datatype recvtype,
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

int MPIX_Neighbor_alltoallw_init(const void *sendbuf, const int sendcounts[], const MPI_Aint
sdispls[], const MPI_Datatype sendtypes[],
    void *recvbuf, const int recvcounts[], const MPI_Aint rdispls[], const MPI_Datatype recvtypes[],
    MPI_Comm comm, MPI_Info info, MPI_Request *request)
```

Fortran (USE mpi_f08_ext) format

```
  MPIX_Allgather_init(sendbuf, sendcount, sendtype, recvbuf, recvcount,
  recvtype, comm, info, request, ierror)
      TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
      TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
      INTEGER, INTENT(IN) :: sendcount, recvcount
      TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
      TYPE(MPI_Comm), INTENT(IN) :: comm
      TYPE(MPI_Info), INTENT(IN) :: info
      TYPE(MPI_Request), INTENT(OUT) :: request
      INTEGER, OPTIONAL, INTENT(OUT) :: ierror

  MPIX_Allgatherv_init(sendbuf, sendcount, sendtype, recvbuf, recvcounts,
  displs, recvtype, comm, info, request, ierror)
      TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
      TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
      INTEGER, INTENT(IN) :: sendcount
      INTEGER, INTENT(IN), ASYNCHRONOUS :: recvcounts(*), displs(*)
      TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
      TYPE(MPI_Comm), INTENT(IN) :: comm
      TYPE(MPI_Info), INTENT(IN) :: info
      TYPE(MPI_Request), INTENT(OUT) :: request
      INTEGER, OPTIONAL, INTENT(OUT) :: ierror

  MPIX_Allreduce_init(sendbuf, recvbuf, count, datatype, op, comm, info,
  request, ierror)
```

```
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: count
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Alltoall_init(sendbuf, sendcount, sendtype, recvbuf, recvcount,
recvtype, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, recvcount
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Alltoallv_init(sendbuf, sendcounts, sdispls, sendtype, recvbuf,
recvcounts, rdispls, recvtype, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: sendcounts(*), sdispls(*), recvcounts(*), rdispls(*)
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Alltoallw_init(sendbuf, sendcounts, sdispls, sendtypes, recvbuf,
recvcounts, rdispls, recvtypes, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: sendcounts(*), sdispls(*), recvcounts(*), rdispls(*)
    TYPE(MPI_Datatype), INTENT(IN), ASYNCHRONOUS :: sendtypes(*), recvtypes(*)
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Barrier_init(comm, info, request, ierror)
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Bcast_init(buffer, count, datatype, root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: buffer
    INTEGER, INTENT(IN) :: count, root
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Exscan_init(sendbuf, recvbuf, count, datatype, op, comm, info, request,
ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: count
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Gather_init(sendbuf, sendcount, sendtype, recvbuf, recvcount, recvtype,
root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, recvcount, root
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Gatherv_init(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs,
recvtype, root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, root
    INTEGER, INTENT(IN), ASYNCHRONOUS :: recvcounts(*), displs(*)
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Reduce_init(sendbuf, recvbuf, count, datatype, op, root, comm, info,
request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: count, root
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Reduce_scatter_init(sendbuf, recvbuf, recvcounts, datatype, op, comm,
info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: recvcounts(*)
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Reduce_scatter_block_init(sendbuf, recvbuf, recvcount, datatype, op,
comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: recvcount
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Scan_init(sendbuf, recvbuf, count, datatype, op, comm, info, request,
ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: count
    TYPE(MPI_Datatype), INTENT(IN) :: datatype
    TYPE(MPI_Op), INTENT(IN) :: op
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Scatter_init(sendbuf, sendcount, sendtype, recvbuf, recvcount,
recvtype, root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, recvcount, root
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Scatterv_init(sendbuf, sendcounts, displs, sendtype, recvbuf,
recvcount, recvtype, root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: sendcounts(*), displs(*)
    INTEGER, INTENT(IN) :: recvcount, root
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Neighbor_allgather_init(sendbuf, sendcount, sendtype, recvbuf,
recvcount, recvtype, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, recvcount
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Neighbor_allgatherv_init(sendbuf, sendcount, sendtype, recvbuf,
recvcounts, displs, recvtype, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount
    INTEGER, INTENT(IN), ASYNCHRONOUS :: recvcounts(*), displs(*)
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Neighbor_alltoall_init(sendbuf, sendcount, sendtype, recvbuf,
recvcount, recvtype, comm, info, request, ierror)
```

```
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN) :: sendcount, recvcount
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Request), INTENT(OUT) :: request
    TYPE(MPI_Info), INTENT(IN) :: info
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Neighbor_alltoallv_init(sendbuf, sendcounts, sdispls, sendtype,
recvbuf, recvcounts, rdispls, recvtype, comm, info, request,
ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: sendcounts(*), sdispls(*), recvcounts(*), rdispls(*)
    TYPE(MPI_Datatype), INTENT(IN) :: sendtype, recvtype
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

MPIX_Neighbor_alltoallw_init(sendbuf, sendcounts, sdispls, sendtypes,
recvbuf, recvcounts, rdispls, recvtypes, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
    INTEGER, INTENT(IN), ASYNCHRONOUS :: sendcounts(*), recvcounts(*)
    INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN), ASYNCHRONOUS :: sdispls(*), rdispls(*)
    TYPE(MPI_Datatype), INTENT(IN), ASYNCHRONOUS :: sendtypes(*), recvtypes(*)
    TYPE(MPI_Comm), INTENT(IN) :: comm
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror
```

Fortran (USE mpi_ext) format

```
MPIX_ALLGATHER_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
    RECVTYPE, COMM, INFO, REQUEST, IERROR)
    <type>    SENDBUF(*), RECVBUF (*)
    INTEGER    SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM
    INTEGER    INFO, REQUEST, IERROR

MPIX_ALLGATHERV_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF,
    RECVCOUNT, DISPLS, RECVTYPE, COMM, INFO, REQUEST, IERROR)
    <type>    SENDBUF(*), RECVBUF(*)
    INTEGER    SENDCOUNT, SENDTYPE, RECVCOUNT(*)
    INTEGER    DISPLS(*), RECVTYPE, COMM, INFO
    INTEGER    REQUEST, IERROR

MPIX_ALLREDUCE_INIT(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM,
    INFO, REQUEST, IERROR)
    <type>    SENDBUF(*), RECVBUF(*)
    INTEGER    COUNT, DATATYPE, OP, COMM, INFO, REQUEST, IERROR

MPIX_ALLTOALL_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
    RECVTYPE, COMM, INFO, REQUEST, IERROR)
    <type>    SENDBUF(*), RECVBUF(*)
    INTEGER    SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE
    INTEGER    COMM, INFO, REQUEST, IERROR

MPIX_ALLTOALLV_INIT(SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPE,
     RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPE, REQUEST, COMM,
     INFO, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNTS(*), SDISPLS(*), SENDTYPE
     INTEGER   RECVCOUNTS(*), RDISPLS(*), RECVTYPE
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_ALLTOALLW_INIT(SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES,
     RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPES, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNTS(*), SDISPLS(*), SENDTYPES(*)
     INTEGER   RECVCOUNTS(*), RDISPLS(*), RECVTYPES(*)
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_BARRIER_INIT(COMM, INFO, REQUEST, IERROR)
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_BCAST_INIT(BUFFER, COUNT, DATATYPE, ROOT, COMM, INFO, REQUEST, IERROR)
     <type>    BUFFER(*)
     INTEGER   COUNT, DATATYPE, ROOT, COMM, INFO, REQUEST, IERROR

MPIX_EXSCAN_INIT(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, INFO, REQUEST,
     IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   COUNT, INFO, DATATYPE, OP, COMM, REQUEST, IERROR

MPIX_GATHER_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
     RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_GATHERV_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNTS,
     DISPLS, RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*)
     INTEGER   RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR

MPIX_REDUCE_INIT(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, ROOT, COMM,
     INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   COUNT, DATATYPE, OP, ROOT, COMM, INFO, REQUEST, IERROR

MPIX_REDUCE_SCATTER_INIT(SENDBUF, RECVBUF, RECVCOUNTS, DATATYPE, OP,
     COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   RECVCOUNTS(*), DATATYPE, OP, COMM, INFO, REQUEST, IERROR

MPIX_REDUCE_SCATTER_BLOCK_INIT(SENDBUF, RECVBUF, RECVCOUNT, DATATYPE, OP,
     COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   RECVCOUNT, DATATYPE, OP, COMM, INFO, REQUEST, IERROR

MPIX_SCAN_INIT(SENDBUF, RECVBUF, COUNT, DATATYPE, OP, COMM, INFO, REQUEST,
     IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   COUNT, DATATYPE, OP, COMM, INFO, REQUEST, IERROR

MPIX_SCATTER_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
     RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_SCATTERV_INIT(SENDBUF, SENDCOUNTS, DISPLS, SENDTYPE, RECVBUF,
     RECVCOUNT, RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNTS(*), DISPLS(*), SENDTYPE
     INTEGER   RECVCOUNT, RECVTYPE, ROOT, COMM, INFO, REQUEST, IERROR

MPIX_NEIGHBOR_ALLGATHER_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
     RECVTYPE, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF (*), RECVBUF (*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, INFO
     INTEGER   REQUEST, IERROR

MPIX_NEIGHBOR_ALLGATHERV_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF,
     RECVCOUNT, DISPLS, RECVTYPE, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT(*)
     INTEGER   DISPLS(*), RECVTYPE, COMM, INFO, REQUEST, IERROR

MPIX_NEIGHBOR_ALLTOALL_INIT(SENDBUF, SENDCOUNT, SENDTYPE, RECVBUF, RECVCOUNT,
     RECVTYPE, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_NEIGHBOR_ALLTOALLV_INIT(SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPE,
     RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPE, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNTS(*), SDISPLS(*), SENDTYPE
     INTEGER   RECVCOUNTS(*), RDISPLS(*), RECVTYPE
     INTEGER   COMM, INFO, REQUEST, IERROR

MPIX_NEIGHBOR_ALLTOALLW_INIT(SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES,
     RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPES, COMM, INFO, REQUEST, IERROR)
     <type>    SENDBUF(*), RECVBUF(*)
     INTEGER   SENDCOUNTS(*), SDISPLS(*), SENDTYPES(*)
     INTEGER   RECVCOUNTS(*), RDISPLS(*), RECVTYPES(*)
     INTEGER   COMM, INFO, REQUEST, IERROR
```

<Explanation>

The requests created with these routines can be used with the request operation routine which is defined in the MPI standard.

These routines ignore hints specified in the info argument.

<Return Value>

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

<Note>

These routines are based on the draft version of the MPI standard. Therefore, the behavior or the arguments of these routines may be changed in the future. Both this interface prefixed with MPIX_ and MPI-standardized interface prefixed with MPI_ will be available in the future releases, and after a certain period of time, this interface will be deleted from this software system.

5.6 Additional Predefined Datatype

5.6.1 Overview

Clang Mode of the Fujitsu C compiler and the Fujitsu C++ compiler supports the following two types for the half-precision (16 bit) floating-point type.

- _Float16
- __fp16

Since the MPI-3.1 Standard does not define a predefined datatype for these types, this software system provides an own predefined datatype.

Refer to C User's Guide and C++ User's Guide for information on the half-precision floating-point type supported by the Fujitsu compilers.

5.6.2 Predefined Datatype for the Half-Precision Floating-Point Type

<Format>

C language format

```
#include <mpi-ext.h>
MPI_Datatype MPIX_C_FLOAT16
```

Fortran (USE mpi_f08_ext) format

```
USE mpi_f08_ext
TYPE(MPI_Datatype) :: mpix_c_float16
```

Fortran (USE mpi_ext) format

```
USE MPI_EXT
INTEGER mpix_c_float16
```

<Explanation>

This is a named predefined datatype for the types _Float16 and __fp16 in C and C++.

This datatype can be used as a basic datatype in MPI routines in the same way as other named predefined datatypes of complex types like MPI_FLOAT. However, the datatype name returned by the MPI_TYPE_GET_NAME routine is different from "MPIX_C_FLOAT16".

Information

- For the half-precision floating-point type REAL(2) in Fortran, the named predefined datatype MPI_REAL2, which is defined in the MPI-3.1 Standard, can be used.
- Fortran formats are required to use these C/C++ types in a Fortran program using interlanguage linkage. Refer to the compiler manuals for information on interlanguage linkage.

Chapter 6 Supplementary Items

6.1 Parallelizing Memory Copy Processing in MPI Library with Threads

On this software system, although processings in the MPI library are basically performed on a thread where an MPI routine is called, a function that some memory copy processings in the MPI library can be parallelized with multiple threads is provided. In order to use this function, specifying the compiler option, the environment variable, and the MCA parameter shown in "Table 6.1 Usage of the function of parallelizing memory copy processing in MPI library with threads" are necessary.

Main processings which can be parallelized with threads are packing and unpacking using derived datatype data. Packing is a processing which positions noncontiguous data into a contiguous buffer. Unpacking is a processing which unpacks the packed data according to the original noncontiguous layout. This function may be able to improve performance of MPI routines such as the MPI_PACK, MPI_UNPACK, and communication routines using derived datatype data. The number of threads used for thread parallelization is decided by the environment variable specified at the time of the MPI program execution. If the omp_set_num_threads routine of the OpenMP is called in the MPI program, the number of threads set by the routine is used.

However, if any of the following conditions is met, an MPI routine is not parallelized even if the routine is a target of thread parallelization.

- The MPI routine is called in a parallel region of the OpenMP in the MPI program.
- According to some conditions such as the number of threads and the size of data to be handled, the performance of the MPI routine with single thread is considered to be better than with multiple threads.

Table 6.1 Usage of the function of parallelizing memory copy processing in MPI library with threads

Timing to specify a value	Content
MPI program compilation/linkage	Specify -Nlibomp and at least one of the -Kparallel and -Kopenmp as options of a compilation/linkage command. Refer to the compiler manuals for details of -Kparallel and -Kopenmp options.
MPI program execution	- Specify the number of threads for the environment variable OMP_NUM_THREADS. If the environment variable is not specified, the number of threads is same as that of available CPUs in the job. - Specify the value 1 for the MCA parameter opal_mt_memcpy as an option of the mpiexec command. Refer to "Table 4.17 opal_mt_memcpy (parallelizes some memory copy processings performed in the MPI library using multiple threads)" for details of the MCA parameter opal_mt_memcpy.

Note

If different numbers of threads are specified with multiple ways, the number of threads actually used when thread parallelization is performed is decided according to the following priority.

1. The value set by the omp_set_num_threads routine in the MPI program
2. The value specified for the environment variable OMP_NUM_THREADS

6.2 Notes Concerning MPI Standard Specifications

6.2.1 Supported Level of MPI Standard

The MPI library provided by this software system conforms to the MPI-3.1 Standard and a subset of the MPI-4.0 Standard.

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

Refer to the "MPI User's Guide Additional Volume Java Interface" for information on Java interface.

"Table 6.2 Routines not provided by Java binding of this software system" shows the routines which are defined in the range of the MPI-3.1 Standard but are not supported for use in Java programs.

The interfaces removed from the MPI standard may also be removed from a future product version of this software system. Do not use them. Refer to the MPI standard for the removed interfaces and their replacements.

Table 6.2 Routines not provided by Java binding of this software system

Routines
MPI_AINT_ADD
MPI_AINT_DIFF
MPI_ALLOC_MEM
MPI_COMM_CREATE_ERRHANDLER
MPI_COMM_JOIN
MPI_ERRHANDLER_FREE
MPI_FILE_CREATE_ERRHANDLER
MPI_FREE_MEM
MPI_GET_ADDRESS
MPI_GREQUEST_COMPLETE
MPI_GREQUEST_START
MPI_INEIGHBOR_ALLTOALLW
MPI_INFO_GET_VALUELEN
MPI_NEIGHBOR_ALLTOALLW
MPI_PACK_EXTERNAL
MPI_PACK_EXTERNAL_SIZE
MPI_PCONTROL
MPI_REGISTER_DATAREP
MPI_TYPE_CREATE_DARRAY
MPI_TYPE_CREATE_HINDEXED_BLOCK
MPI_TYPE_CREATE_INDEXED_BLOCK
MPI_TYPE_CREATE_SUBARRAY
MPI_TYPE_GET_CONTENTS
MPI_TYPE_GET_ENVELOPE
MPI_T_CATEGORY_CHANGED
MPI_T_CATEGORY_GET_CATEGORIES
MPI_T_CATEGORY_GET_CVARS
MPI_T_CATEGORY_GET_INDEX
MPI_T_CATEGORY_GET_INFO
MPI_T_CATEGORY_GET_NUM
MPI_T_CATEGORY_GET_PVARS
MPI_T_CVAR_GET_INDEX
MPI_T_CVAR_GET_INFO
MPI_T_CVAR_GET_NUM
MPI_T_CVAR_HANDLE_ALLOC
MPI_T_CVAR_HANDLE_FREE
MPI_T_CVAR_READ
MPI_T_CVAR_WRITE
MPI_T_ENUM_GET_INFO
MPI_T_ENUM_GET_ITEM
MPI_T_FINALIZE
MPI_T_INIT_THREAD
MPI_T_PVAR_GET_INDEX
MPI_T_PVAR_GET_INFO
MPI_T_PVAR_GET_NUM
MPI_T_PVAR_HANDLE_ALLOC
MPI_T_PVAR_HANDLE_FREE
MPI_T_PVAR_READ
MPI_T_PVAR_READRESET

Routines
MPI_T_PVAR_RESET
MPI_T_PVAR_SESSION_CREATE
MPI_T_PVAR_SESSION_FREE
MPI_T_PVAR_START
MPI_T_PVAR_STOP
MPI_T_PVAR_WRITE
MPI_UNPACK_EXTERNAL
MPI_WIN_CREATE_ERRHANDLER
MPI_WIN_SHARED_QUERY

6.2.2 Predefined Datatypes that can be Used in This Software System

The predefined MPI datatypes that can be used in this software system vary depending on the language bindings used for an MPI program.

The predefined MPI datatypes and the corresponding datatypes of each program language are shown in "Table 6.3 Predefined MPI datatypes usable by Fortran binding", "Table 6.4 Predefined MPI datatypes usable by C binding", "Table 6.5 Predefined MPI datatypes usable by C++ binding", and "Table 6.6 Predefined MPI datatypes usable by Java binding".

The valid datatypes in reduction operations for Java binding are shown in "Table 6.6 Predefined MPI datatypes usable by Java binding". Refer to the MPI standard for the datatypes that can be specified in reduction operations.

Table 6.3 Predefined MPI datatypes usable by Fortran binding

Predefined MPI datatypes	Fortran datatypes
[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_REAL2	REAL(2)
MPI_REAL4	REAL(4)
MPI_REAL8	REAL(8)
MPI_REAL16	REAL(16)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_COMPLEX4	COMPLEX(2)
MPI_COMPLEX8	COMPLEX(4)
MPI_COMPLEX16	COMPLEX(8)
MPI_COMPLEX32	COMPLEX(16)
MPI_DOUBLE_COMPLEX	COMPLEX(8)
MPI_CHAR	char (C)
MPI_SHORT	signed short int (C)
MPI_INT	signed int (C)
MPI_LONG	signed long int (C)
MPI_LONG_LONG_INT	signed long long int (C)
MPI_LONG_LONG	signed long long int (C)
MPI_SIGNED_CHAR	signed char (C)
MPI_UNSIGNED_CHAR	unsigned char (C)
MPI_UNSIGNED_SHORT	unsigned short int (C)

Predefined MPI datatypes	Fortran datatypes
MPI_UNSIGNED	unsigned int (C)
MPI_UNSIGNED_LONG	unsigned long int (C)
MPI_UNSIGNED_LONG_LONG	unsigned long long int (C)
MPI_FLOAT	float (C)
MPI_DOUBLE	double (C)
MPI_LONG_DOUBLE	long double (C)
MPI_WCHAR	wchar_t (C)
MPI_BYTE	
MPI_PACKED	
MPI_C_BOOL	_Bool (C)
MPI_C_COMPLEX	float _Complex (C)
MPI_C_FLOAT_COMPLEX	float _Complex (C)
MPI_C_DOUBLE_COMPLEX	double _Complex (C)
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex (C)
MPI_INT8_T	int8_t (C)
MPI_INT16_T	int16_t (C)
MPI_INT32_T	int32_t (C)
MPI_INT64_T	int64_t (C)
MPI_UINT8_T	uint8_t (C)
MPI_UINT16_T	uint16_t (C)
MPI_UINT32_T	uint32_t (C)
MPI_UINT64_T	uint64_t (C)
MPIX_C_FLOAT16	_Float16, __fp16 (C)
MPI_AINT	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_OFFSET	INTEGER (KIND=MPI_OFFSET_KIND)
MPI_COUNT	INTEGER (KIND=MPI_COUNT_KIND)
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++)
[Other than the Basic datatypes]	
MPI_LB	
MPI_UB	
MPI_FLOAT_INT	float (C) and signed int (C) pair
MPI_DOUBLE_INT	double (C) and signed int (C) pair
MPI_LONG_INT	signed long int (C) and signed int (C) pair
MPI_2INT	signed int (C) pair
MPI_SHORT_INT	signed short int (C) and signed int (C) pair
MPI_LONG_DOUBLE_INT	long double (C) and signed int (C) pair
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 6.4 Predefined MPI datatypes usable by C binding

Predefined MPI datatypes	C datatypes
[Basic datatypes]	
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_REAL2	REAL(2) (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_COMPLEX4	COMPLEX(2) (Fortran)
MPI_COMPLEX8	COMPLEX(4) (Fortran)
MPI_COMPLEX16	COMPLEX(8) (Fortran)
MPI_COMPLEX32	COMPLEX(16) (Fortran)
MPI_DOUBLE_COMPLEX	COMPLEX(8) (Fortran)
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_C_COMPLEX	float _Complex
MPI_C_FLOAT_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t

Predefined MPI datatypes	C datatypes
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPIX_C_FLOAT16	_Float16, __fp16
MPI_AINT	MPI_Aint
MPI_OFFSET	MPI_Offset
MPI_COUNT	MPI_Count
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++)
[Other than the Basic datatypes]	
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_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

Table 6.5 Predefined MPI datatypes usable by C++ binding

Predefined MPI datatypes	C++ datatypes
[Basic datatypes]	
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::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

Predefined MPI datatypes	C++ datatypes
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::BOOL	bool
MPI::COMPLEX	Complex<float>
MPI::DOUBLE_COMPLEX	Complex<double>
MPI::LONG_DOUBLE_COMPLEX	Complex<long double>
[Other than the Basic datatypes]	
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::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::TWOREAL	REAL (Fortran) pair
MPI::TWODOUBLE_PRECISION	DOUBLE PRECISION (Fortran) pair
MPI::TWOINTEGER	INTEGER (Fortran) pair

Table 6.6 Predefined MPI datatypes usable by Java binding

Predefined MPI datatypes	Java datatypes	Datatypes kind
[Basic datatypes]		
MPI.CHAR	char	C integer
MPI.SHORT	short	C integer
MPI.INT	int	C integer
MPI.LONG	long	C integer
MPI.FLOAT	float	Floating point
MPI.DOUBLE	double	Floating point
MPI.BYTE	byte	Byte
MPI.PACKED		
MPI.BOOLEAN	boolean	Logical
[Other than the Basic datatypes]		
MPI.FLOAT_COMPLEX	java.nio.FloatBuffer (The data of java.nio.FloatBuffer can be acquired to float [] using the mpi.FloatComplex class.)	
MPI.DOUBLE_COMPLEX	java.nio.DoubleBuffer (The data of java.nio.DoubleBuffer can be acquired to double [] using the mpi.DoubleComplex class.)	
MPI.FLOAT_INT	float and int pair	
MPI.DOUBLE_INT	double and int pair	
MPI.LONG_INT	long and int pair	
	int pair	
	short and int pair	

Predefined MPI datatypes	Java datatypes	Datatypes kind
MPI.INT2 MPI.SHORT_INT		

6.2.3 Reserved Communicators

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

- MPI_COMM_WORLD
- MPI_COMM_SELF

In addition, MPI_COMM_NULL is reserved as a predefined constant.

6.2.4 Values of Constants Set in This Software System

In this software system, the values of the following named constants of Fortran defined in the MPI standard are ".FALSE.".

- MPI_SUBARRAYS_SUPPORTED
- MPI_ASYNC_PROTECTS_NONBLOCKING

6.2.5 Operations in a Multi-Threaded Environment

This software system supports operations in a multi-thread environment. The thread support level of this software 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 standard for details.

6.2.6 Signal Operation Changes

When the MPI_INIT, MPI_INIT_THREAD, or MPI_T_INIT_THREAD routine is called, this software system sets a handler for each signal shown below if a handler other than the default one is not set yet.

Names of signals with changed system standard operation

- `SIGABRT`
- `SIGBUS`
- `SIGFPE`
- `SIGSEGV`
- `SIGXCPU`

6.2.7 One-sided Communications

This section provides notes on one-sided communications.

6.2.7.1 Assertions for Optimization

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

Table 6.7 Assertions for optimizations in one-sided communications

MPI routine name	Assertion	Operation in this software system
MPI_WIN_POST	MPI_MODE_NOCHECK	Ignored

MPI routine name	Assertion	Operation in this software system
	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

6.2.7.2 Info Argument

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

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

Table 6.8 Info key for MPI_WIN_CREATE, MPI_WIN_ALLOCATE, and MPI_WIN_CREATE_DYNAMIC

Info key	Operation in this software system
no_locks	When true is specified as a key value of this info key, it is guaranteed that the window is not locked. The default value for this info key is false.

Table 6.9 Info key for MPI_WIN_ALLOCATE_SHARED

Info key	Operation in this software 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.

6.2.8 Establishing Communication between Groups not Sharing a Communicator

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

6.2.8.1 MPI_COMM_JOIN Return Value

The output of the MPI_COMM_JOIN routine is MPI_COMM_NULL.

6.2.9 Dynamic Process Creation

This section provides notes on dynamic process creation.

6.2.9.1 info Argument Value

The keys that can be specified for the info argument of the MPI_COMM_SPAWN routine or the array_of_info argument of the MPI_COMM_SPAWN_MULTIPLE routine provided by this software system are shown below.

Table 6.10 Info keys for dynamic process creation

Info key	Value	Explanation
wdir	Directory path name	Specify the current directory when the dynamic process creation is executed.
env	String (NAME=VALUE)	Specify the environment variables that are set to the dynamic processes.

6.2.9.1.1 Designation of Current Directory by wdir Key

The current directory path name of the dynamic processes can be passed by specifying the wdir key as the info argument. If the relative path is specified, the value will be the relative path from the current directory when the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is called.

6.2.9.1.2 Designation of Environment Variables by env Key

The env key is used to set the new environment variables to processes which are created by dynamic process creation function. The designation of value to env key is described in the form of "NAME=VALUE". And, to set multiple environment variables, use the "\n" as a delimiter.

6.2.9.2 Search for Executable Files

The executable files specified for the command argument of the MPI_COMM_SPAWN routine or the array_of_commands argument of the MPI_COMM_SPAWN_MULTIPLE routine are searched according to the following rules.

- When the wdir info key is specified:

  The files are searched in the specified directory.

- When the wdir info key is not specified:

  First, the files are searched in the current directory when the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is called.

  Next, the files are searched in the directories which are specified with the user environment variable PATH.

6.2.9.3 Dynamic Process Creation for Java program

If the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is called in a Java program, the classpath in which the necessary Java class files exist must be set with not the -classpath option but the environment variable CLASSPATH.

An example of specifying the execution command and environment variables to execute a java program where a routine for dynamic process creation is called is shown as follows.

An example of such a program is also shown. "*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

Example

Example of specifying the execution command and environment variables

```
CLASSPATH=/home/user1/bin:$CLASSPATH
export CLASSPATH
PATH=/installation_path/bin:$PATH
export PATH
LD_LIBRARY_PATH=/installation_path/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

```
mpiexec -n 2 java Spawn
```

Java program example

```
import mpi.*;

public class Spawn {
   private final static String CMD_ARGV1 = "THIS IS ARGV 1"; /* Argument1 to pass to the child process
*/
   private final static String CMD_ARGV2 = "THIS IS ARGV 2"; /* Argument2 to pass to the child process
*/
    private final static String CMD = "Spawn";  /* Name of the class file to be spawned */

    public static void main(String args[]) throws MPIException {
        MPI.Init(args);
        System.out.println("Start");

        String spawn_argv[] = {
            CMD,
            CMD_ARGV1,
            CMD_ARGV2
        };

        int count = 1;
        int errcode[] = new int[1];

        Intercomm parent;
        parent = Intercomm.getParent();

        if (!parent.isNull()) {
           /* child process */
           System.out.println("I am a child");
        }else{
           /* parent process : Spawn oneself */
           MPI.COMM_WORLD.spawn("java", spawn_argv, count, MPI.INFO_NULL, 0, errcode);
           System.out.println("I am a parent");
        }

        System.out.println("End");
        MPI.Finalize();
    }
}
```

6.2.10 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.

6.2.11 MPI Input-Output

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

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

The following restrictions exist.

- The file input-output data size that can be handled once is less than or equal to "2GiB - 64KiB". The data size is "the datatypes size of one element * number of elements".

The file systems handled as MPI input-output are Lustre, FEFS, UFS, and NFS. Other file systems are not supported.

Note the following when using NFS.

- You must specify noac for mount option to prevent the unmatch of the content because of the data update delay.
- According to the NFS specifications, when a file is opened newly for write, the file area may not be allocated.

In this software system, native is the only supported data expression.

The MPI_FILE_OPEN routine provided by this software system does not change the size of the existing file.

The input-output routine of this software system may create a temporary file in the same directory as the user's input-output file. These are the files created when MPI_FILE_OPEN routine is used to open a file. The size of one of these files is about eight bytes. The MPI_FILE_CLOSE routine 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 routines (MPI_FILE_READ_ALL routine and similar) is always based on the argument information at the time of invocation.

Table below shows the Info object keys and values that can be used by MPI input-output in this software 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
romio_ds_write	"disable"	Specify whether or not to apply data sieving on write. Specify either "enable" or "ENABLE" to enable data sieving on write. Specify either "disable" or "DISABLE" not to enable data sieving on write. Specify either "automatic" or "AUTOMATIC" to let the MPI library judge whether to enable data sieving on write.
romio_ds_read	"automatic"	Specify whether or not to apply data sieving on read. Specify either "enable" or "ENABLE" to enable data sieving on read. Specify either "disable" or "DISABLE" not to enable data sieving on read. Specify either "automatic" or "AUTOMATIC" to let the MPI library judge whether to enable data sieving on read.

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 Lustre/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

Key	Default value	Meaning
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.

6.2.12 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 of the profiling interface.
- For C++ programs, MPI calls can be intercepted using the C binding of the profiling interface.

6.2.13 MPI Tool Information Interface

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

6.2.14 Routines implemented by macros

The following routines are defined as macros in the mpi.h file for the C language.

- MPI_AINT_ADD
- MPI_AINT_DIFF
- PMPI_AINT_ADD
- PMPI_AINT_DIFF

6.2.15 Arguments of User-defined Error Handlers

Usually, when an error is detected during execution of an MPI program, an error handler defined in the MPI library is called. This error handler can be changed to a user-defined routine. In the C bindings of the MPI, the prototypes of user-defined error handlers are defined as the following routines with a variable number of arguments.

```
typedef void MPI_Comm_errhandler_function(MPI_Comm *, int *, ...);
typedef void MPI_Win_errhandler_function(MPI_Win *, int *, ...);
typedef void MPI_File_errhandler_function(MPI_File *, int *, ...);
```

In the MPI standard, the arguments except for the first and second ones of these error handlers are assumed to be implementation-dependent. In this software system, a const char * type of string which represents the name of the MPI routine where an error is detected is passed to the third argument of these error handlers. No value is passed to the fourth and following arguments.

6.2.16 Collective Communications in Inter-communicator

In this section, notes on collective communications executed in an inter-communicator of this library is described.

A process whose argument root is MPI_ROOT is called the root process. A group that have the root process is called the first group and the otherwise is called the second group.

Number of elements

This library does not guarantee any operations when the conditions showed in the following table are met.

Routine name	Condition of not guaranteeing
MPI_Allgather	When the product of the size of the second group and the second argument sendcount specified in rank 0 or the product of the size of the first group and the fifth argument recvcount in rank 0 is over 2147483647.
MPI_Allgatherv	When the sum of the second argument sendcount specified in ranks in a certain group is over 2147483647.
MPI_Gather	When the product of the size of the second group and the second argument sendcount specified in rank 0 or the product of the size of the first group and the fifth argument recvcount in rank 0 is over 2147483647.
MPI_Gatherv	When the sum of the second argument sendcount specified in ranks in the second group is over 2147483647.
MPI_Reduce_scatter	When the sum of the elements of the third argument recvcounts specified in ranks in a certain group is over 2147483647.
MPI_Reduce_scatter_block	When the product of the third argument recvcount in a certain rank and the size of the group to which the rank belongs is over 2147483647.
MPI_Scatter	When the product of the size of the second group and the second argument sendcount specified in the root process in the first group or the product of the size of the first group and the fifth argument recvcount in rank 0 is over 2147483647.
MPI_Scatterv	When the sum of the elements of the second argument sendcounts specified in a certain rank in the second group is over 2147483647.
MPI_Iallgather MPIX_Allgather_init	When the product of the fifth argument recvcount specified in a certain rank and the value obtained by subtracting 1 from the size of the opposite group of the rank is over 2147483647.
MPI_Ialltoall MPIX_Alltoall_init	When the product of the second argument sendcount specified in a certain rank and the value obtained by subtracting 1 from the size of the opposite group of the rank or the product of the fifth argument recvcount specified in a certain rank and the value obtained by subtracting 1 from the size of the opposite group of the rank is over 2147483647.
MPI_Igather MPIX_Gather_init	When the product of the fifth argument recvcount specified in the root process in the first group and the value obtained by subtracting 1 from the size of the second group is over 2147483647.
MPI_Ineighbor_allgather MPIX_Neighbor_allgather_init	When the product of the value obtained by subtracting 1 from in-degree of a certain rank and the fifth argument recvcount specified in that rank is over 2147483647.
MPI_Ineighbor_alltoall MPIX_Neighbor_alltoall_init	When the product of the value obtained by subtracting 1 from out-degree of a certain rank and the fifth argument recvcount specified in that rank is over 2147483647.
MPI_Ireduce_scatter MPIX_Reduce_scatter_init	When the sum of the elements of the fifth argument recvcounts specified in a certain rank is over 2147483647.
MPI_Ireduce_scatter_block MPIX_Reduce_scatter_block_init	When the product of the fifth argument recvcount in a certain rank and the size of its local communicator is over 2147483647.
MPI_Iscatter MPIX_Scatter_init	When the product of the second argument sendcount specified in the root process of the first group and the value obtained by subtracting 1 from the size of the second group is over 2147483647.

Datatypes

This library does not guarantee any operations when the conditions showed in the following table are met.

Routine name	Condition of not guaranteeing
MPI_Allgatherv	When the size of datatype is not same between the ranks in the same group.
MPI_Gatherv	When the size of datatype is not same between the ranks in the same group.
MPI_Scatterv	When the size of datatype is not same between the ranks in the same group.

6.3 Eager Protocol and Rendezvous Protocol

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

Eager protocol

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.

Under the Eager protocol, 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 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.

Rendezvous protocol

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 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 software 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.

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

- MPI programs that use nonblocking communication to perform multiple communications simultaneously
- MPI programs that execute receive routines (the MPI_RECV routine, etc.) more quickly than send routines (the MPI_SEND routine, etc.)

In these cases, improved MPI program performance can be expected due to changing this "threshold value". The MCA parameter btl_openib_eager_limit can be used to change the "threshold value". Refer to "Table 4.6 btl_openib_eager_limit (changes the threshold value for switching the communication method)" for information on the MCA parameter btl_openib_eager_limit.

6.4 Reduction Operation Sequence Guarantee in Collective Communication

This software system may change the sequence of reduction operations performed by the collective communication routines MPI_REDUCE, MPI_IREDUCE, MPI_ALLREDUCE, MPI_IALLREDUCE, MPI_REDUCE_SCATTER, MPI_IREDUCE_SCATTER, MPI_REDUCE_SCATTER_BLOCK, MPI_IREDUCE_SCATTER_BLOCK, and MPI_SCAN. 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 software 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 4.9 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 routine 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 routine, 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 routine, and execute while specifying the order.

6.5 Process Creation from Inside an MPI Program

In this software system, system calls or system library functions to create processes in an MPI program cannot be used.

The InfiniBand enables the high-speed transfer performance by processing at physical memory level, using the direct transfer which does not use memory copy via CPU. Process creation by system calls or system library functions involves a duplication of the memory space, and it violates memory management in the InfiniBand.

6.6 Suppressing Memory Usage

When an MPI program is executed, this software 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 software 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 routine, 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 software system.

6.6.1 Influence of Dynamic Connection on Performance

The dynamic connection built in this software 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 routine, 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.

6.7 Memory Usage of MPI Library

The MPI library memory usage in a compute node can be estimated using the formula shown in "Memory usage estimation formula". "Table 6.13 Variables in memory usage estimation formula" shows the meanings of the variables in "Memory usage estimation formula".

Memory usage estimation formula

$$C_{qp} * N_{node} * N^2_{ppn} + (C_{qp} + C_{ib\_other}) * N_{ppn} \qquad \text{[GiB/node]}$$

Table 6.13 Variables in memory usage estimation formula

Variable	Meaning of variable	Explanation of variable	Value
C_{qp}	Coefficient	A constant in accordance with the number of processes per node (referred to as "ppn" in this section)	1 [ppn]: 0.000126 2-4 [ppn]: 0.000127 5-24 [ppn]: 0.000128 24-32 [ppn]: 0.000129 33-48 [ppn]: 0.000131
N_{node}	Number of compute nodes	The number of compute nodes	

Variable	Meaning of variable	Explanation of variable	Value
N_{ppn}	Number of processes per node	The number of processes per node (referred to as "ppn" in this section)	
C_{ib_other}	Coefficient	A constant in accordance with the number of processes per node (referred to as "ppn" in this section)	1 [ppn]: 0.035194 2-4 [ppn]: 0.021279 5-8 [ppn]: 0.018683 9-16 [ppn]: 0.016128 17-24 [ppn]: 0.015824 25-32 [ppn]: 0.013867 33-48 [ppn]: 0.010758

Example

Example of memory usage estimation for the MPI library using "Memory usage estimation formula" (in the case of 36 compute nodes and 48 ppn)

$$C_{qp} * N_{node} * N^2_{ppn} + (C_{qp} + C_{ib\_other}) * N_{ppn}$$

```
= 0.000131 * 36 * 48² + (0.000131 + 0.010758) * 48
= 11.388   [GiB/node]
```

Note

Note the following when using "Memory usage estimation formula".

- Use "Memory usage estimation formula" when the number of processes per compute node N_{ppn} is 8 or more and the number of MPI processes N_{proc} meets the following formula.

  $$N_{proc} >= 76016 / N_{ppn}$$

- The "Memory usage estimation formula" assumes the conditions that result in maximum memory usage of the MPI library. Specifically, it assumes the conditions that each process communicates with all processes directly. Therefore, the actual memory usage in an application where each process communicates with few processes may significantly differ from the estimated memory usage using the estimation formula.

- There are algorithms which allocate temporary buffers in MPI collective communication routines. However, memory usage of these temporary buffers is not included in "Memory usage estimation formula".

- If there are multiple MPI processes in a compute node, the larger percentage of memory on the compute node is used by the MPI processes. Therefore, pay particular attention to memory usage if the number of MPI processes per compute node is large.

- Because the operating system manages the memory per page, more memory than the estimated result may be required depending on page size.

6.8 MPI_BCAST/MPI_IBCAST routines When the Same Count is Used among the Processes

In this software system, a faster communication mechanism is available when the MPI_BCAST routine or MPI_IBCAST routine 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 the MPI_BCAST routine or MPI_IBCAST routine with different counts among the processes, a deadlock may be caused in the MPI library. According to the MPI standard, MPI_BCAST routine 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 the MPI_BCAST routine or MPI_IBCAST routine 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 4.10 coll_tuned_bcast_same_count (achieves faster communication when MPI_BCAST/MPI_IBCAST routines are used with the same count among the processes)" for information on the MCA parameter coll_tuned_bcast_same_count.

Note

The MPI_ALLGATHER and MPI_ALLGATHERV routines have algorithms that calls the processing of the MPI_BCAST routine internally.

Therefore, if the MCA parameter coll_tuned_bcast_same_count is specified to 1 in the following MPI program, an inconsistency may occur in the MPI library.

- In the case of the MPI_ALLGATHER routine
  The MPI program with different send counts or receive counts among the processes.

- In the case of the MPI_ALLGATHERV routine
  The MPI program that has a different number of counts on the receiver (The number of counts received for each process of the sender is specified individually in the array) for each process.

The following measures are required to prevent inconsistency.

- In the case of the MPI_ALLGATHER routine
  The sender and receiver must have the same counts.

- In the case of the MPI_ALLGATHERV routine
  The receive counts represents as the array. In the array, receive count for each rank of sender is specified. Therefore, same counts for all ranks is required. The send counts with the different counts for the processes are fine.

6.9 Avoiding Concentration of Communications in MPI_GATHERV Routine

In the MPI_GATHERV routine in this software system, non-root processes send a message to the root process given as an argument by the application program. The communication time of the MPI_GATHERV routine may increase because the communications concentrate on the root process. This software system has a function to avoid concentration of communications to the root in the MPI_GATHERV routine by periodically calling the synchronization processing equivalent to the MPI_BARRIER routine. To use this function, set the MCA parameter coll_basic_gatherv_sync to 1. For more information about the MCA parameter coll_basic_gatherv_sync, see "Table 4.11 coll_basic_gatherv_sync (synchronizes periodically in MPI_GATHERV)".

As described above, the communication performance of the MPI_GATHERV routine may improve by enabling this function to avoid concentration of communications. On the other hand, the overhead of the synchronization processing may reduce communication performance. Therefore, note that the effect of this function may not be obtained depending on the application program or other communication environment.

6.10 Dynamic Debug during MPI Program Execution

This software system provides the following functions for performing debugging during MPI program execution:

- Communication timeout setting (abort of communication wait)

- Monitoring incorrect writing to MPI communication buffer

Note that performance of an MPI program may become worse if these debug functions are used. Use these functions with caution.

6.10.1 Communication Timeout Setting

If the state of communication wait continues during an MPI program execution, it is possible that the program is hanging due to communication deadlock or the like. In order to use system resources efficiently, a hanging program should not exist for a long time. This software system provides a function to avoid an MPI program to continue communication wait more than a user expected. If an upper limit value for communication waits is set and time of a communication wait exceeds this upper limit during an MPI program execution, the program can be terminated after the corresponding message output. Hereinafter, the state that time of a communication wait exceeds its upper limit is called "communication timeout".

The MCA parameter opal_progress_timeout is used to set an upper limit value (in seconds) of communication wait time. Stack trace information is included in the message output when time of a communication wait exceeds the upper limit. In order to output symbol names (routine names) in the trace information, specify "-Wl,-export-dynamic" as an option of the MPI program compilation/linkage command. Refer to "Table 4.18 opal_progress_timeout (specifies the timeout time in communication wait)" for details of the MCA parameter.

However, if time of communication wait exceeds the upper limit, this communication timeout setting function also terminates an MPI program where it takes long time for communication wait but there is no error. Use this function with caution. In addition, the location in the process or the MPI program which was being executed when communication timeout was detected is not always the location which caused the hang. Therefore, in order to identify the cause of hang, tracing and reviewing the program may be necessary.

This function cannot detect all hang of programs. One effective method to identify the cause of hang is inserting the MPI_BARRIER routines before and after communication routine calls in a program.

The MCA parameter opal_abort_delay can delay actual timing to terminate an MPI program after a communication timeout detection. If the timing to terminate the program is delayed, communication timeout in other processes may also be detected. In this way, getting information of multiple processes when communication timeout was detected may also be useful to identify the cause of hang. Refer to "Table 4.15 opal_abort_delay (delays program termination when an error is detected)" for details of the MCA parameter.

6.10.2 Monitoring Incorrect Writing to MPI Communication Buffer

Incorrect behaviors such as result error, memory corruption and so on may occur without reproducibility if another writing to a send buffer which is in use for a not completed nonblocking communication occurs. In order to prevent such incorrect behaviors, this software system provides a function to detect incorrect writing to a send buffer for a nonblocking communication.

This buffer monitoring function can be enabled using the MCA parameter mpi_check_buffer_write. When this monitoring function is enabled and incorrect writing to a send buffer which is in use for a nonblocking communication occurs, a corresponding message is output and the MPI program execution terminates. Stack trace information is included in the output message. In order to output symbol names (routine names) in the trace information, specify "-Wl,-export-dynamic" as an option of the MPI program compilation/linkage command. Refer to "Table 4.13 mpi_check_buffer_write (monitors incorrect writing to communication buffers)" for details of the MCA parameter.

Note that nonblocking send operation in buffered mode using the MPI_IBSEND routine is not monitored even if this monitoring function is enabled.

Example

Example

It is assumed that a send buffer in the following program is monitored.

```
main( )
{
    ...
    int buf = 0;
    MPI_Isend(&buf, 1, MPI_INT, 1, 0, MPI_COMM_WORLD, &req);
    buf = 1;
    MPI_Wait(&req, &status);
    ...
 }
```

First, specify "-Wl,-export-dynamic" as an option of MPI program compilation/linkage command mpifccpx to compile the above program. Next, when the above program is executed, specify the value 1 for the MCA parameter mpi_check_buffer_write as an option of the mpiexec command. Then, an error message like the one below is output to the standard error output. "*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

```
[mpi::opal-util::check-buffer-write] The buffer was destroyed in this process.
/installation_path/lib64/libmpi.so.0(PMPI_Wait+0x50) [0xffffffff20296ad0] (1)
./a.out(main+0x4a4) [0x1012e4](2)
/.../... (  ...             ) [0xffffffffa07f381c]
./a.out(  ...              )[0x100cec]
```

Lines from 2 to 5 in the above error message are stack trace information. This stack trace information shows where the send operation of nonblocking communication which used a send buffer written in incorrect timing was completed. In this example, from (1) and (2) in the above message, it can be found that such the send operation of nonblocking communication was completed with the MPI_Wait function called in the main function. Then, by checking the program refering the address information included in (1), the location where the send operation (the MPI_Isend function) of such the nonblocking communication is performed can be found. Finally, by checking by checking between the MPI_Isend and the MPI_Wait function calls in the program, the location where writing to the send buffer buf occurred can be found.

6.11 MPI Program Execution Using Slurm

There is the Slurm, which is an open source job scheduler. If the Slurm is available in your environment, you can submit a job to run an MPI program from the login node using the Slurm.

This section describes how to execute an MPI program using the Slurm, comparing with when using the job operation software of the PRIMEHPC FX1000.

Refer to https://slurm.schedmd.com/ for details of the Slurm.

6.11.1 Environment Check

To use the Slurm with this software system, the PMIx Version 3 must be enabled in the Slurm. If the PMIx Version 3 is not enabled in your Slurm, apply the PMIx Version 3 and rebuild the Slurm.

Whether the PMIx Version 3 is enabled in your Slurm can be checked by the following way.

How to check

```
$ srun --mpi=list
srun: MPI types are...

srun: pmix_v3      <-  If "srun: pmix_v3" is output, PMIx Version 3 is enabled in your Slurm.
```

6.11.2 Slurm Commands

The table below shows a comparison of the major commands of the Slurm and those of the job operation software for the PRIMEHPC FX1000.

Table 6.14 Comparison of the major commands of the Slurm and those of the job operation software for the PRIMEHPC FX1000

Command purpose	Slurm	Job operation software for PRIMEHPC FX1000
Submitting a job	sbatch	pjsub
Deleting a job	scancel	pjdel
Acquiring job information	squeue	pjstat
Acquiring the resource status	sinfo	pjshowrsc

examples of command execution

The display may differ depending on the version of the Slurm. Refer to https://slurm.schedmd.com/ for details.

- Submitting a job

  When submitting a job using the Slurm, prepare a script file called a job script, as in the case of using the job operation software for the PRIMEHPC FX1000.

```
$ sbatch slurm-bat.sh
Submitted batch job 6025
```

- Deleting a job

```
$ scancel 6025
$ squeue
JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
```

- Acquiring job information

```
$ squeue
JOBID PARTITION     NAME     USER  ST       TIME  NODES NODELIST(REASON)
6025    normal      slurm-ba    app1   PD       0:00   2 (Priority)
```

- Acquiring the resource status

```
$ sinfo
PARTITION AVAIL TIMELIMIT NODES STATE  NODELIST
normal     up    infinite         4  idle     cn[01-04]
```

6.11.3 MPI Program Execution Using Slurm

The following is an example of a job script for running an MPI program using the Slurm. In this example, the number of nodes is 4 and the number of processes per compute node is 4, for a total of 16 processes to execute the MPI program. We have confirmed that it works using the Slurm version 20.11.

Refer to https://slurm.schedmd.com/ for details on job script for the Slurm.

Example of a job script to execute an MPI program using the Slurm

```
#!/bin/sh
#SBATCH --job-name slurm-job
#SBATCH --partition=normal
#SBATCH --nodes=4
#SBATCH --ntasks=16
#SBATCH --time=10:00
#SBATCH --output a.%J.stdout
#SBATCH --error a.%J.stderr

MPI_HOME=/opt/FJSVstclanga/cp-1.0.20.05
export PATH=${MPI_HOME}/bin:${PATH}
export LD_LIBRARY_PATH=${MPI_HOME}/lib64:${LD_LIBRARY_PATH}

srun --mpi=pmix_v3 ./a.out
```

In the job script for the Slurm, the options that can be specified for the job submission command sbatch can be specified on lines beginning with #SBATCH. On the other hand, in the job script for the job operation software of the PRIMEHPC FX1000, the options that can be specified for the pjsub command can be specified on lines beginning with #PJM.

Table 1.1 provides an overview of the sbatch command options of Slurm that are used in the job script shown in "Example of a job script to execute an MPI program using the Slurm". The pjsub command options of the job operation software for the PRIMEHPC FX1000 corresponding to these options are also described in Table 1.1. However, even if there are multiple option names with the same meaning, only one of them is described here.

Table 6.15 Overview of the sbatch command option used in "Example of a job script to execute an MPI program using the Slurm" and the corresponding pjsub command option

sbatch command option	Meaning	Corresponding pjsub command option
--job-name=*jobname*	Specifies the job name.	--name *name*
--partition=*partition_names*	Specifies the partition for the job execution.	--rsc-list rscgrp=*rscgname*

sbatch command option	Meaning	Corresponding pjsub command option
--nodes=*minnodes*[-*maxnodes*]	Specifies the number of compute nodes to be assigned for the job.	--rsc-list node=*n*
--ntasks=*number*	Specifies the number of processes to be launched in the job.	--mpi proc=*procnum*
--time=*time*	Specifies the upper limit of time for job execution.	--rsc-list elapse=*elapsedtime*
--output *filename_pattern*	Specifies the destination file name for standard output.	--out *pathname*
--error *filename_pattern*	Specifies the destination file name for standard error output.	--err *pathname*

In the job script shown in "Example of a job script to execute an MPI program using the Slurm", the --mpi option of the srun command is specified. Table below shows an overview of the --mpi option of the srun command. In the PRIMEHPC FX1000, the command that corresponds to the srun command of the Slurm is the mpiexec command, but there is no option that corresponds to the --mpi option.

Table 6.16 Overview of the srun command option used in "Example of a job script to execute an MPI program using the Slurm"

srun command option	Meaning
--mpi=*mpi_type*	Specifies the type of MPI to be used. In this software system, specifying "--mpi=pmix_v3" is necessary.

6.11.4 Binding CPU Resources to Processes

The Slurm binds CPU resources to each process according to the default settings without specifying any special option at the time of job execution. It is also possible to specify the binding of CPU resources using the options of the sbatch command. Refer to https://slurm.schedmd.com/ for details.

Overview of the main sbatch command options for CPU resource binding is listed in table below. The pjsub command options of the job operation software for the PRIMEHPC FX1000 corresponding to these options are also listed in table below. However, even if there are multiple option names with the same meaning, only one of them is listed here.

Table 6.17 Overview of the main sbatch command options for CPU resource binding

sbatch command option	Meaning	Corresponding pjsub command option
--ntasks-per-node=*ntasks*	Specifies the number of processes to be allocated to each compute node.	None.
--cpus-per-task=*ncpus*	Specifies the number of CPU cores to be allocated to each process.	None.
--distribution=*distribution*	Specifies the rules on allocating processes to each node. In this section, only the following two of the possible values for this option are described. - block If the number of processes to be allocated per node is n, n processes are allocated to one node and then the next node is selected. - cyclic Allocates processes to the compute nodes in the round-robin method.	The behavior when "block" and "cyclic" are specified corresponds to the behavior when "--mpi rank-map-bychip" and "--mpi rank-map-bynode" are specified in the pjsub command options, respectively.

Note

The specified values of --ntasks-per-node option and --cpus-per-task option must meet the following conditions. The default value for the number of CPU cores in this system is 48. If other value is set in your environment, that setting is applied.

```
"value of --ntasks-per-node" * "value of --cpus-per-task" <= "number of CPU cores"
```

6.11.5 Other Notes

6.11.5.1 MCA Parameter Settings

To specify the MCA parameter when executing an MPI program using the Slurm, prefix the MCA parameter name with "OMPI_MCA_" and set it as an environment variable.

Refer to "4.2 MCA Parameters" for information on how to specify MCA parameters when executing an MPI program without the Slurm.

The following is an excerpt of the description of a job script when the MCA parameter btl_openib_eager_limit is specified.

```
export OMPI_MCA_btl_openib_eager_limit=1024    # Specify 1024 for the MCA parameter
btl_openib_eager_limit using the corresponding environment variable
srun --mpi=pmix_v3 ./a.out
```

6.11.5.2 MPMD Model Execution

Use a configuration file to execute an MPI program in the MPMD model. Refer to the --multi-prog option of the srun command for details of the configuration file.

Refer to "4.1 Execution Command Formats" for information on how to execute an MPI program in the MPMD model without the Slurm.

How to execute

When executing the srun command in a job script, specify the configuration file for --multi-prog option.

```
srun  --multi-prog configuration_file
```

6.11.6 Restrictions

6.11.6.1 Restricted MPI Routines To Use

The following MPI routines are not supported by the combination of the Slurm and PMIx Version 3. Therefore, in this system, the use of the following MPI routines is restricted when executing an MPI program using the Slurm.

- MPI_COMM_SPAWN
- MPI_COMM_SPAWN_MULTIPLE
- MPI_LOOKUP_NAME
- MPI_COMM_ACCEPT
- MPI_COMM_CONNECT

When the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is called, the process outputs the following message and terminates abnormally.

Example of the message output when the MPI_COMM_SPAWN routine is called

```
[mpi::mpi-errors::mpi_errors_are_fatal]
[cn01:202496] *** An error occurred in MPI_Comm_spawn
[cn01:202496] *** reported by process [2732541117,0]
[cn01:202496] *** on communicator MPI_COMM_WORLD
[cn01:202496] *** MPI_ERR_SPAWN: could not spawn processes
```

```
[cn01:202496] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[cn01:202496] ***    and potentially your MPI job)
```

When the MPI_LOOKUP_NAME, the MPI_COMM_ACCEPT, or the MPI_COMM_CONNECT routine is called, the process outputs the following message and terminates abnormally.

Example of the message output when the MPI_LOOKUP_NAME routine is called

```
[mpi::mpi-api::MPI function not supported]
Your application has invoked an MPI function that is not supported in this environment.
```

Chapter 7 Error Messages

This chapter explains the error messages output for this software 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::invalid-value-enum]
An invalid value was supplied for an enum variable.

Variable   : *name*
Value   : *val*
Valid values : *values*
--

- Description

  The value specified for the MCA parameter is invalid. Execution of the mpiexec command ends.

- Parameters

  name : MCA parameter name

  val : MCA parameter value

  values : MCA parameter values that can be specified

- Action method

  Check the value specified for the MCA parameter.

--
[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-runtime::opal_init:startup:internal-failure]
It looks like opal_init failed for some reason; your parallel process is
likely to abort. There are many reasons that a parallel process can
fail during opal_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 OPAL_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 routine 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::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] *mpiexec* 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

  mpiexec : mpiexec command

  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] *mpiexec* Error: unknown option "*opt*"

- Description

  An unsupported option was specified in the mpiexec command. Execution of the mpiexec command ends.

- Parameters

  mpiexec : mpiexec command

  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.

- 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 routine 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::schizo-ompi::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::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::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 in MPI_COMM_WORLD

  error : Error details

  file : Error file path

  lineno : Line number

- 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 routine was called. Execution of the MPI program ends.

- Parameters

  rank : Rank in MPI_COMM_WORLD

  comm : Detailed information of the communicator in the MPI_ABORT routine first argument

  rc : MPI_ABORT routine second argument

- Action method

  Check whether there is an error in the MPI program content.

--
[mpi::mpi-runtime::mpi_init: already finalized]
Open MPI has detected that this process has attempted to initialize MPI (via MPI_INIT or
MPI_INIT_THREAD) after MPI_FINALIZE has been called.
This is erroneous.
--

- Description

  After MPI_FINALIZE routine, an MPI routine that cannot be called after MPI_FINALIZE routine due to MPI standard was called.

- Action method

  Check if an MPI routine was called after MPI_FINALIZE routine in the program.

  If called, this contravenes MPI standard, so correct the program.

However, the following routines can be called after MPI_FINALIZE routine:

- MPI_INITIALIZED routine
- MPI_FINALIZED routine
- MPI_GET_VERSION routine

```
--------------------------------------------------------------------------
[mpi::mpi-runtime::mpi_init: invoked multiple times]
Open MPI has detected that this process has attempted to initialize MPI (via MPI_INIT or
MPI_INIT_THREAD) more than once.
This is erroneous.
--------------------------------------------------------------------------
```

- Description

  Either the MPI_INIT routine or the MPI_INIT_THREAD routine was called twice.

- Action method

  Check if either the MPI_INIT routine or the MPI_INIT_THREAD routine was called more than once in the program. Either the MPI_INIT routine or the MPI_INIT_THREAD routine can be called only once due to MPI standard. If called more than once, correct the program.

```
--------------------------------------------------------------------------
[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 in MPI_COMM_WORLD

  type : Information on either the communicator, file, or window (depending on the cause of the problem)

  error class : See "Appendix A Error Class List"

- 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::opal-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.

[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. Stack trace information is output and the MPI program execution ends.

- Action method

  Refer to the stack trace information and revise the MPI program so that no processes overwrite the nonblocking communication send buffer.

[*info*] Possible hang-up (no progress) is detected on [[*jobid*,*snum*],*rank*] *<Stack trace information>* [*data*]

- Description

  The communication wait time exceeded the upper limit (seconds) specified by the user. A deadlock may have occurred. Stack trace information and data for System Engineer for analysis purposes are output and execution of the MPI program ends.

- Parameters

  info : Host name and pid information

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

  data : Data for System Engineer for analysis purposes

- 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.

[*info*] Delaying for *time* seconds before aborting

- Description

  Termination of the program is delayed by the specified time (*time* seconds).

- Parameters

  info : Host name and pid information

  time : The value specified for the MCA parameter opal_abort_delay

7.4 Compilation/linkage command Error Messages

comm unrecognized option: -showme:*param*

- Description

  An unsupported parameter was specified in the -showme option.

- Parameters

  comm : Compilation/linkage command

  param : Relevant parameter

- Action method

  Specify the correct parameter for -showme option.

Appendix A Error Class List

This appendix lists the error classes output by this software system. These are the error classes regulated by the MPI standard. Refer to the MPI standard for details.

Table A.1 Error class list

Error class	Description	Value
MPI_SUCCESS	No error	0
MPI_ERR_BUFFER	Invalid buffer pointer	1
MPI_ERR_COUNT	Invalid count argument	2
MPI_ERR_TYPE	Invalid datatype 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 of some other kind	13
MPI_ERR_UNKNOWN	Unknown error	14
MPI_ERR_TRUNCATE	Message truncated on receive	15
MPI_ERR_OTHER	Known error not in this list	16
MPI_ERR_INTERN	Internal MPI (implementation) error	17
MPI_ERR_IN_STATUS	Error code is in status	18
MPI_ERR_PENDING	Pending request	19
MPI_ERR_ACCESS	Permission denied	20
MPI_ERR_AMODE	Error related to the amode passed to MPI_FILE_OPEN	21
MPI_ERR_ASSERT	Invalid assert argument	22
MPI_ERR_BAD_FILE	Invalid file name (e.g., path name too long)	23
MPI_ERR_BASE	Invalid base passed to MPI_FREE_MEM	24
MPI_ERR_CONVERSION	An error occurred in a user supplied data conversion function	25
MPI_ERR_DISP	Invalid disp argument	26
MPI_ERR_DUP_DATAREP	Conversion functions could not be registered because a data representation identifier that was already defined was passed to MPI_REGISTER_DATAREP	27
MPI_ERR_FILE_EXISTS	File exists	28
MPI_ERR_FILE_IN_USE	File operation could not be completed, as the file is currently open by some process	29
MPI_ERR_FILE	Invalid file handle	30
MPI_ERR_INFO_KEY	Key longer than MPI_MAX_INFO_KEY	31
MPI_ERR_INFO_NOKEY	Invalid key passed to MPI_INFO_DELETE	32

Error class	Description	Value
MPI_ERR_INFO_VALUE	Value longer than MPI_MAX_INFO_VAL	33
MPI_ERR_INFO	Invalid Info argument	34
MPI_ERR_IO	Other I/O error	35
MPI_ERR_KEYVAL	Invalid keyval has been passed	36
MPI_ERR_LOCKTYPE	Invalid locktype argument	37
MPI_ERR_NAME	Invalid service name passed to MPI_LOOKUP_NAME	38
MPI_ERR_NO_MEM	MPI_ALLOC_MEM failed because memory is exhausted	39
MPI_ERR_NOT_SAME	Collective argument not identical on all processes, or collective routines called in a different order by different processes	40
MPI_ERR_NO_SPACE	Not enough space	41
MPI_ERR_NO_SUCH_FILE	File does not exist	42
MPI_ERR_PORT	Invalid port name passed to MPI_COMM_CONNECT	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	Wrong synchronization of RMA calls	47
MPI_ERR_SERVICE	Invalid service name passed to MPI_UNPUBLISH_NAME	48
MPI_ERR_SIZE	Invalid size argument	49
MPI_ERR_SPAWN	Error in spawning processes	50
MPI_ERR_UNSUPPORTED_DATAREP	Unsupported datarep passed to MPI_FILE_SET_VIEW	51
MPI_ERR_UNSUPPORTED_OPERATION	Unsupported operation, such as seeking on a file which supports sequential access only	52
MPI_ERR_WIN	Invalid win argument	53
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	57
MPI_T_ERR_INVALID_ITEM	The item index queried is out of range	58
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 (in the case of a window created with MPI_WIN_CREATE_DYNAMIC, target memory is not attached)	68

Error class	Description	Value
MPI_ERR_RMA_ATTACH	Memory cannot be attached (e.g., because of resource exhaustion)	69
MPI_ERR_RMA_FLAVOR	Passed window has the wrong flavor for the called function	70
MPI_ERR_RMA_SHARED	Memory cannot be shared (e.g., some process in the group of the specified communicator cannot expose shared memory)	71
MPI_T_ERR_INVALID	Invalid use of the interface or bad parameter values(s)	72
MPI_T_ERR_INVALID_NAME	The variable name or category name is invalid	73
MPI_ERR_LASTCODE	Last error code	92

Glossary

Blocking communication

Indicates message send-receive for which the user buffer specified at MPI routine invocation can be re-used if there is a return from the MPI routine.

Inter-communicator

This refers to the communicator connecting the two different communicators. The term is defined as inter-communicator in the MPI standard. For example, it is used for the communication between the communicator generated by the MPI_COMM_SPAWN routine and the communicator that called the MPI_COMM_SPAWN routine.

Intra-communicator

This refers to the communicator generated from the MPI_COMM_WORLD or the communicator generated by the communicator generating routine other than the MPI_INTERCOMM_CREATE. The term is defined as intra-communicator in the MPI standard.

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 standard.

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 routine before the actual procedures of the MPI routine are completed. The user buffer specified at MPI routine invocation cannot be re-used until completion of operations is confirmed.

Parallel process

A process started on the compute node by mpiexec command is called a parallel process. A number, starting from 0, is assigned to each parallel process. In this software system, these numbers correspond to the rank numbers of the MPI program communicator MPI_COMM_WORLD.

Pipeline transfer

It means to transfer the message by dividing it into segments of a certain size instead of transferring all messages at one time. In the collective communication, there are algorithms that implement the pipeline transfer.

Segment size

Transfer amount of data for each time of pipeline transfer. In the MCA parameter, you can specify the segment size for each collective communication routine. By changing this value, the number of pipeline transfers and the transfer time for each transfer will change.

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.

Type signature

It is essential that the messages transmitted by MPI routines 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 standard.

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 routine (such as the MPI_RECV routine) in response to a send-type routine (such as the MPI_SEND routine).