FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Development Studio
MPI User's Guide

J2UL-2480-02ENZ0(12)
March 2024

Preface

Purpose of This Manual

This manual describes how to use the MPI library intended for the system that has the Fujitsu CPU A64FX (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, and a knowledge of Job Operation Software.

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

Chapter 8 Speeding Up Blocking Collective Communication

Description about advanced performance tuning method for blocking collective communication

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
- IDE User's Guide
- Profiler User's Guide
- Debugger for Parallel Applications 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
- uTofu User's Guide
- MPI User's Guide Additional Volume Java Interface

Also, refer to the manuals of provided with the following related software:

- Job Operation Software
- FEFS

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
March 2024, Version 2.12	J2UL-2480-02ENZ0(12)
September 2023, Version 2.11	J2UL-2480-02ENZ0(11)
March 2023, Version 2.10	J2UL-2480-02ENZ0(10)
September 2022, Version 2.9	J2UL-2480-02ENZ0(09)
March 2022, Version 2.8	J2UL-2480-02ENZ0(08)
November 2021, Version 2.7	J2UL-2480-02ENZ0(07)
August 2021, Version 2.6	J2UL-2480-02ENZ0(06)
July 2021, Version 2.5	J2UL-2480-02ENZ0(05)
March 2021, Version 2.4	J2UL-2480-02ENZ0(04)
November 2020, Version 2.3	J2UL-2480-02ENZ0(03)

Version	Manual code
September 2020, Version 2.2	J2UL-2480-02ENZ0(02)
June 2020, Version 2.1	J2UL-2480-02ENZ0(01)
March 2020, 2nd Version	J2UL-2480-02ENZ0(00)
January 2020, 1st Version	J2UL-2480-01ENZ0(00)

Copyright

Copyright FUJITSU LIMITED 2020-2024

Update History

Changes	Location	Version
The following MCA parameters are added. - common_tofu_conv_dim - common_tofu_conv_dim_log	4.2	Version 2.12
The explanation about the memory usage is improved.	6.11	
The section "Job Dimension Conversion Function" is added.	6.15	
The explanation is improved.	Chapter 4	Version 2.11
Improved figures design.	-	Version 2.10
The explanation is added.	Chapter 4	Version 2.9
The Notes on isend/irecv are added.	6.10.2	
The explanations are improved.	-	
The explanation to the following MCA parameter is corrected. - mpi_print_stats	4.2	Version 2.8
The section "Notes on Stride RDMA Communication" is added.	6.6.1	
The explanations are added.	6.12.1 6.12.2 6.12.3	
Improved the explanation.	-	
The note on the assumed-rank of the Fortran 2018 language specification is removed.	3.2	Version 2.7
An explanation about the environment variable UTOFU_SWAP_PROTECT is added.	4.3	
The explanations are added.	4.4	
The article about the maximum value of the file input-output size is corrected.	6.4.12	Version 2.6
The article is added to "Compilation/linkage notes".	3.2	Version 2.5
The following environment variable is added. - coll	4.2 6.12.1 6.12.2 6.12.3 6.12.4	
The following MCA parameter is added. - UTOFU_SWAP_PROTECT	4.3	
The default of "value" for the following Info key is changed. - romio_ds_write	6.4.12	

Changes	Location	Version
The explanations are corrected.	6.17.1	
The explanations are corrected.	Chapter 4	Version 2.4
The explanations to the following MCA parameters are added. - common_tofu_use_memory_pool - mpi_java_eager	4.2	
The allocation images are corrected.	4.7.2	
The explanations are added.	6.9 6.11.1	Version 2.3
The explanations to the following MCA parameters are added. - common_tofu_large_recv_buf_size - coll_tuned_prealloc_size	4.2	
The following MCA parameter is added. - common_tofu_use_memory_pool	4.2	
The explanations are corrected.	4.2 6.13 6.15	
Note is added	6.13	
The examples of MPI statistical information output are corrected.	6.15	
The explanations are added.	Chapter 4 4.1	Version 2.2
The section "Notes on running large-scale MPI jobs" is added.	4.1	
The explanations are corrected.	4.1	
The error messages are added.	7.3	
The explanations are corrected.	7.3	
The example are corrected.	8.3.2.2	
The application conditions of algorithms selected is corrected.	8.3.4.4	
The descriptions of { -c | -np | --np | -n | --n } related options are corrected.	4.1	Version 2.1
The explanations are added.	4.2 6.8 6.18.5 8.3.1.4.4 8.3.1.4.6 8.3.1.5.5 8.3.2.2	
The memory usage estimation formula is changed.	6.11.1 6.11.2 6.11.3.2	
The explanations are corrected.	8.3.1.1	
The conditions required for application the algorithm is corrected.	8.3.4.1 8.3.4.2 8.3.4.4	
The table are added.	8.3.4.12 8.4.1.12	

Changes	Location	Version
The descriptions of -of/-std 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 4.3 8.2.1.1 8.3.1.1 8.3.1.4.1 8.3.1.4.2 8.3.1.4.3 8.3.1.4.5 8.3.4.7 8.3.4.8 8.4.2.2	
The estimated size of the work area allocated statically by the following MCA parameters is corrected. - coll_tuned_prealloc_size	4.2	
The following MCA parameters are added. - opal_abort_delay - opal_progress_timeout	4.2	
The default value of the following MCA parameter is corrected. - pml_ob1_use_stride_rdma	4.2	
The info key is corrected.	6.4.9.1	
The explanations are added.	6.16 8.2.1.2 8.3.1.1 8.3.1.4	
The section "Notes on Job Execution Continuation Function at Link-down" is added.	6.18	
The error messages are added.	7.3	
The algorithms are added.	8.2.1.1 8.3.4.1 8.3.4.2 8.3.4.3 8.3.4.4 8.3.4.7 8.3.4.10 8.4.1.1 8.4.1.2 8.4.1.3 8.4.1.4 8.4.1.7 8.4.1.10 8.4.3.1 8.4.3.2 8.4.3.3	
The title of section is corrected.	8.3.1.4.2	
The examples are corrected.	8.3.1.4.2 8.3.1.4.3 8.3.1.4.4	

Changes	Location	Version
The title of table is corrected.	8.3.1.4.6	
The explanation of the Algorithm statement of the table "Definition of External Input File" is corrected.	8.3.1.5.5	
The caller_id is removed from the table "Variable Names that can be Specified in External Input File".	8.3.1.5.5	
The notes are added.	8.3.4.3 8.3.4.4 8.3.4.8 8.3.4.10 8.3.4.11 8.3.4.12	
Changed the look according to product upgrades.	-	

All rights reserved.
The information in this manual is subject to change without notice.

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....17
4.3 Environment Variables....32
4.4 mpiexec Command Return Values....33
4.5 VCOORD file format....34
4.6 Execution of Multiple MPI Programs on the Same Node....36
4.7 Settings in NUMA system....37
4.7.1 Setting value of NUMA memory allocation policy....37
4.7.2 Setting value of CPU (core) allocation policy....39

Chapter 5 Extended Interfaces....42
5.1 Rank Query Interface....42
5.1.1 Querying the Number of Dimensions and Shape....42
5.1.1.1 FJMPI_TOPOLOGY_GET_DIMENSION....42
5.1.1.2 FJMPI_TOPOLOGY_GET_SHAPE....43
5.1.2 Querying the Coordinates....44
5.1.2.1 FJMPI_TOPOLOGY_GET_COORDS....44
5.1.3 Querying the Rank....46
5.1.3.1 FJMPI_TOPOLOGY_GET_RANKS....46
5.1.4 Querying the Ranking of a Communicator that Has a Cartesian Structure....47
5.1.4.1 FJMPI_TOPOLOGY_CART_REORDER....47
5.1.5 Sample Program....48
5.2 MPI Statistical Information Section Specifying Interface....50
5.2.1 The MPI Statistical Information Section Specifying Routine....51
5.2.1.1 FJMPI_COLLECTION_START....51
5.2.1.2 FJMPI_COLLECTION_STOP....51
5.2.1.3 FJMPI_COLLECTION_PRINT....52
5.2.1.4 FJMPI_COLLECTION_CLEAR....53
5.2.2 Sample Program....53
5.3 Extended Persistent Communication Requests Interface....55
5.3.1 Overview....56
5.3.2 Extended Persistent Communication Requests Interface Specifications....56
5.3.2.1 FJMPI_PREQUEST_SEND_INIT....56
5.3.2.2 FJMPI_PREQUEST_RECV_INIT....57
5.3.2.3 FJMPI_PREQUEST_START....58
5.3.2.4 FJMPI_PREQUEST_STARTALL....59
5.3.3 Sample Program....59
5.4 MPI Asynchronous Communication Promotion Section Specifying Interface....60
5.4.1 The MPI Asynchronous Communication Promotion Section Specifying Routine....61
5.4.1.1 FJMPI_PROGRESS_START....61
5.4.1.2 FJMPI_PROGRESS_STOP....61

5.4.2 Sample Program....62
5.5 Persistent Collective Communication Request Interface....63
5.5.1 Overview....63
5.5.2 Persistent Collective Communication Request Interface Specification....63
5.5.3 Overlap of Computation and Communication....71
5.5.3.1 Conditions for Applying Overlap of Computation and Communication....71
5.5.3.2 Notes....71
5.6 Additional Predefined Datatype....72
5.6.1 Overview....72
5.6.2 Predefined Datatype for the Half-Precision Floating-Point Type....72

Chapter 6 Supplementary Items....73
6.1 Tofu Interconnect....73
6.1.1 Tofu Interconnect Configuration....73
6.1.2 Routing....74
6.1.3 Configuration within a Node....75
6.2 Promoting Asynchronous Communication Using an Assistant Core....76
6.3 Parallelizing Memory Copy Processing in MPI Library with Threads....77
6.4 Notes Concerning MPI Standard Specifications....78
6.4.1 Supported Level of MPI Standard....78
6.4.2 Predefined Datatypes that can be Used in This Software System....79
6.4.3 Reserved Communicators....84
6.4.4 Values of Constants Set in This Software System....84
6.4.5 Operations in a Multi-Threaded Environment....84
6.4.6 Signal Operation Changes....84
6.4.7 One-sided Communications....84
6.4.7.1 Assertions for Optimization....85
6.4.7.2 Info Argument....85
6.4.8 Establishing Communication between Groups not Sharing a Communicator....85
6.4.8.1 info Argument Value....86
6.4.8.2 MPI_COMM_JOIN Return Value....86
6.4.8.3 Service Names in the MPI_PUBLISH_NAME....86
6.4.9 Dynamic Process Creation....86
6.4.9.1 info Argument Value....86
6.4.9.1.1 Designation of Current Directory by wdir Key....86
6.4.9.1.2 Designation of Node and the Number of CPUs (cores) by vcoordfile Key....86
6.4.9.1.3 Designation of the Number of Nodes by num_nodes Key....87
6.4.9.1.4 Designation of the Rule of Rank Placement of Processes by rank_map Key....87
6.4.9.1.5 Designation of Environment Variables by env Key....88
6.4.9.1.6 Designation of Output of Profiling Data by fjprof_spawn_dir_name Key....88
6.4.9.1.7 Designation of Output of Result of Deadlock Detection Function by fjdbg_spawn_dir_name Key....89
6.4.9.2 Search for Executable Files....89
6.4.9.3 MPI_UNIVERSE_SIZE....89
6.4.9.4 Designation of max-proc-per-node....89
6.4.9.5 Identification of the Cause of Failure of Dynamic Process Creation by the Error Code....89
6.4.9.5.1 Error Code for Identifying the Cause of Failure of Dynamic Process Creation....89
6.4.9.5.2 Usage Example....90
6.4.9.6 Notes....90
6.4.9.7 Dynamic Process Creation for Java program....90
6.4.10 Rank Changes in Accordance with Cartesian Topology....91
6.4.10.1 Conditions Enabling Rank Changes....91
6.4.10.2 Rules for Rank Changes....92
6.4.10.3 Checking Rank Changes....92
6.4.10.4 Sample Program....92
6.4.11 Notes on Send Buffer and Receive Buffer....93
6.4.12 MPI Input-Output....93
6.4.13 Use of the Profiling Interface....95

6.4.14 MPI Tool Information Interface ... 95
6.4.15 Routines implemented by macros ... 95
6.4.16 Arguments of User-defined Error Handlers ... 95
6.4.17 Collective Communications in Inter-communicator ... 95
6.5 Eager Protocol and Rendezvous Protocol ... 97
6.6 Stride RDMA Communication ... 98
6.6.1 Notes on Stride RDMA Communication ... 98
6.7 Using Multiple TNIs ... 99
6.8 Reduction Operation Sequence Guarantee in Collective Communication ... 99
6.9 Process Creation from Inside an MPI Program ... 99
6.10 Suppressing Memory Usage ... 100
6.10.1 Switching between Fast Communication Mode and Memory-Saving Communication Mode ... 101
6.10.2 Influence of Dynamic Connection on Performance ... 101
6.11 Memory Usage Estimation Formulae and Tuning Guidelines ... 102
6.11.1 Memory Usage Estimation Formulae ... 102
6.11.2 Memory Usage Tuning Guidelines ... 105
6.11.3 Specifying Memory Allocation Restriction Values ... 106
6.11.3.1 Specification Memory Allocation Restriction Values ... 106
6.11.3.2 MCA Parameters Targeted by Automatic Tuning ... 107
6.11.3.3 Notes on Execution When Memory Allocation Restriction Values are Specified ... 107
6.12 Use of Tofu Barrier Communication for Better Performance ... 108
6.12.1 MPI_BARRIER ... 108
6.12.2 MPI_BCAST ... 108
6.12.3 MPI_REDUCE and MPI_ALLREDUCE ... 109
6.12.4 Notes on Tofu Barrier Communication ... 111
6.13 MPI_BCAST/MPI_IBCAST routines When the Same Count is Used among the Processes ... 112
6.14 Algorithms of Collective Communication and Shape of Compute Nodes Allocated to the Communicator ... 113
6.14.1 Shape of Compute Nodes Allocated to the MPI_COMM_WORLD ... 113
6.14.2 Compute Nodes Allocated to the Intra-communicator ... 114
6.14.3 Shape of Compute Nodes for Algorithms Tuned for Tofu Interconnect ... 114
6.14.4 Length of Axis with Cuboid Shape ... 116
6.15 Job Dimension Conversion Function ... 117
6.15.1 Overview of Job Dimension Conversion Function ... 117
6.15.2 Log Output for Job Dimension Conversion Function ... 117
6.16 MPI Statistical Information ... 118
6.17 Dynamic Debug during MPI Program Execution ... 129
6.17.1 Communication Timeout Setting ... 129
6.17.2 Monitoring Incorrect Writing to MPI Communication Buffer ... 130
6.17.3 Argument Check Function ... 131
6.18 Note on using 3rd Party Tools ... 131
6.18.1 Notes on Using Valgrind ... 131
6.19 Notes on Job Execution Continuation Function at Link-down ... 132
6.19.1 Communication Performance ... 132
6.19.2 Conditions in MPI_ALLTOALL Routine ... 132
6.19.3 Algorithms which are not Applicable ... 132
6.19.4 Note on Specification of MCA Parameters and Options ... 133
6.19.5 Dynamic Process Creation ... 133

Chapter 7 Error Messages ... 134
7.1 Output Format for Information Related to Parallel Processes ... 134
7.2 mpiexec Command Error Messages ... 134
7.3 Communication Library Error Messages ... 140
7.4 Compilation/linkage command Error Messages ... 155

Chapter 8 Speeding Up Blocking Collective Communication ... 156
8.1 Outline ... 156
8.2 MCA Parameter Tuning of Algorithms ... 156
8.2.1 Changing Segment Size ... 156

8.2.1.1 Change of Algorithm Performance by Changing Segment Size ... 156
8.2.1.2 Notes by Changing Segment Size ... 157
8.3 Tuning by Algorithm Selection ... 157
8.3.1 How to Select Algorithms ... 158
8.3.1.1 Flow of Selecting Algorithms ... 158
8.3.1.2 Flow of Selecting Algorithms in Special Case ... 159
8.3.1.3 Selecting Algorithms by MCA Parameter ... 159
8.3.1.4 Selecting Algorithms by Info Object ... 159
8.3.1.4.1 Parameter to Specify with key of Info Object ... 160
8.3.1.4.2 Specification for Each Collective Communication Routine Call ... 160
8.3.1.4.3 Specification for Each Communicator ... 161
8.3.1.4.4 Specifying Rules for Selecting Algorithms ... 161
8.3.1.4.5 Notes for Selecting Algorithms by Info Object ... 163
8.3.1.4.6 List of Values that can be Specified to Key of Info Object ... 163
8.3.1.5 Selecting Algorithms by External Input File ... 164
8.3.1.5.1 How to Use ... 164
8.3.1.5.2 Example of External Input File ... 164
8.3.1.5.3 Multiple Entries of Algorithms and Parameter ... 165
8.3.1.5.4 Conditional Statement ... 165
8.3.1.5.5 Entries of External Input File ... 166
8.3.1.5.6 Notes for Specifying External Input File ... 172
8.3.1.6 Conditions Required for Application of Algorithms ... 172
8.3.2 How to Confirm Selection Results ... 173
8.3.2.1 Displaying Selection Process of Algorithms ... 173
8.3.2.2 Obtaining Selection Results of Algorithms by Using Info Object ... 173
8.3.2.3 Obtaining Results Only with MCA Parameter ... 174
8.3.3 Notes for Selecting Algorithms ... 174
8.3.4 List of Algorithms and Conditions Required for Application ... 174
8.3.4.1 Algorithms Selected by MPI_ALLGATHER Routine ... 175
8.3.4.2 Algorithms Selected by MPI_ALLGATHERV Routine ... 175
8.3.4.3 Algorithms Selected by MPI_ALLREDUCE Routine ... 176
8.3.4.4 Algorithms Selected by MPI_ALLTOALL Routine ... 178
8.3.4.5 Algorithms Selected by MPI_ALLTOALLV Routine ... 179
8.3.4.6 Algorithms Selected by MPI_BARRIER Routine ... 179
8.3.4.7 Algorithms Selected by MPI_BCAST Routine ... 180
8.3.4.8 Algorithms Selected by MPI_GATHER Routine ... 181
8.3.4.9 Algorithms Selected by MPI_GATHERV Routine ... 181
8.3.4.10 Algorithms Selected by MPI_REDUCE Routine ... 181
8.3.4.11 Algorithms Selected by MPI_REDUCE_SCATTER Routine ... 183
8.3.4.12 Algorithms Selected by MPI_SCAN Routine ... 184
8.3.4.13 Algorithms Selected by MPI_SCATTER Routine ... 184
8.3.4.14 Algorithms Selected by MPI_SCATTERV Routine ... 184
8.4 MCA Parameter related to Algorithm Selection ... 185
8.4.1 MCA Parameter to Specify Algorithm Selection ... 185
8.4.1.1 coll_select_allgather_algorithm (Specifying the Algorithm of the MPI_ALLGATHER Routine) ... 185
8.4.1.2 coll_select_allgatherv_algorithm (Specifying the Algorithm of the MPI_ALLGATHERV Routine) ... 185
8.4.1.3 coll_select_allreduce_algorithm (Specifying the Algorithm of the MPI_ALLREDUCE Routine) ... 186
8.4.1.4 coll_select_alltoall_algorithm (Specifying the Algorithm of the MPI_ALLTOALL Routine) ... 186
8.4.1.5 coll_select_alltoallv_algorithm (Specifying the Algorithm of the MPI_ALLTOALLV Routine) ... 187
8.4.1.6 coll_select_barrier_algorithm (Specifying the Algorithm of the MPI_BARRIER Routine) ... 187
8.4.1.7 coll_select_bcast_algorithm (Specifying the Algorithm of the MPI_BCAST Routine) ... 187
8.4.1.8 coll_select_gather_algorithm (Specifying the Algorithm of the MPI_GATHER Routine) ... 188
8.4.1.9 coll_select_gatherv_algorithm (Specifying the Algorithm of the MPI_GATHERV Routine) ... 188
8.4.1.10 coll_select_reduce_algorithm (Specifying the Algorithm of the MPI_REDUCE Routine) ... 188
8.4.1.11 coll_select_reduce_scatter_algorithm (Specifying the Algorithm of the MPI_REDUCE_SCATTER Routine) ... 188
8.4.1.12 coll_select_scan_algorithm (Specifying the Algorithm of the MPI_SCAN Routine) ... 189
8.4.1.13 coll_select_scatter_algorithm (Specifying the Algorithm of the MPI_SCATTER Routine) ... 189

8.4.1.14 coll_select_scatterv_algorithm (Specifying the Algorithm of the MPI_SCATTERV Routine) 189
8.4.2 MCA Parameter related to Algorithm Selection Itself 189
8.4.2.1 coll_select_dectree_file (Specification of External Input User Definition File of Algorithm Selection) 189
8.4.2.2 coll_select_decision_process (Output of Algorithm Selection Process) 190
8.4.3 MCA Parameter to Tune Algorithm Itself 190
8.4.3.1 coll_select_allreduce_algorithm_segmentsize (Specifying the Segment Size of the MPI_ALLREDUCE Routine) 190
8.4.3.2 coll_select_bcast_algorithm_segmentsize (Specifying the Segment Size of the MPI_BCAST Routine) 191
8.4.3.3 coll_select_reduce_algorithm_segmentsize (Specifying the Segment Size of the MPI_REDUCE Routine) 191
8.4.4 MCA Parameter to Obtain Algorithm Selection Result 192
8.4.4.1 coll_select_get_tuning_info (Obtaining the Algorithm Information of the Collective Communication Executed Immediately Before) 192
8.5 Output Message 192
8.5.1 Output Message Related to Algorithm Selection (Warning) 192
8.5.2 Output Message by Display Function of Algorithm Selection Process (Warning) 192
8.5.3 Output Message by Display Function of Algorithm Selection Process (Information) 195

Appendix A Error Class List 198

Glossary 201

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 login node and compute nodes. Refer to "Table 6.3 Routines not provided by Java binding of this software system", for routines of not support.

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

1.2 Outline of How to Use This 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

To execute an MPI program at this computing system, the user performs the required operations at the login node.

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).

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

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

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. To execute a command at a compute node, submit a job which includes the command execution to the Job Operation Software. Refer to the Job Operation Software manual for details.

The cross compiler and native compiler for MPI programs have no functional differences other than the command launching methods described above. Refer to "Chapter 3 MPI Program Compilation/Linkage" for information on how to use 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 but the user does not execute it directly in the compute node. Instead, the user requests Job Operation Software to launch the job that executes the MPI program. Refer to the Job Operation Software manual for information on how to launch jobs.

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

Chapter 2 Environment and Advance Settings

This chapter describes the environment settings that must be set when using this 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 in the job script when the compilation/linkage command execution job is launched. The setting enables MPI program compilation and linkage with the native compiler.

- 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 command execution job is launched.

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

- Append the following path name to user environment variable PATH

```
/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

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

- Append the following path name to user environment variable MANPATH

```
/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. mpiexec command passes control to Job Operation Software. Process creation and execution of the MPI program takes place on the compute node. The Job Operation Software sets the environment variables PMIX_RANK and PLE_RANK_ON_NODE of the Job Operation Software to MPI process.

Refer to the Job Operation Software manual for information on Job Operation Software and the environment variables PMIX_RANK and PLE_RANK_ON_NODE.

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

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

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 ]
[ { -debuglib | --debuglib  } ]
[ { -h | --help  } ]
[ { -of | --of | -std | --std }  FILE  ]
[ { -oferr | --oferr | -stderr | --stderr }  ERR_FILE  ]
[ { -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc }  ERR_PROC_FILE  ]
[ { -ofout | --ofout | -stdout | --stdout }  OUT_FILE  ]
[ { -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc }  OUT_PROC_FILE  ]
[ { -of-proc | --of-proc | -std-proc | --std-proc }  PROC_FILE  ]
[ { -ofprefix | --ofprefix | -stdprefix | --stdprefix }  PREFIX  ]
[ { -stdin | --stdin }  STDIN_FILE  ]
[ { -vcoordfile | --vcoordfile }  VCOORD  ]
[ { -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.

{ -debuglib | --debuglib }

Links the debug MPI library.

If you are using the argument check function, which is one of the dynamic debug functions at the time of execution, specify this option.

Use of this option may cause execution of the MPI program to become very slow as the debug MPI library is linked. Take care when using this option.

Refer to "6.17.3 Argument Check Function" for information on the argument check function.

In addition, you need to use this option when you use Valgrind that is an open source software for memory checking and other functions. Read "6.18.1 Notes on Using Valgrind" for details.

{ -h | --help }

Displays help messages for this command and ends mpiexec command.

{ -of | --of | -std | --std } *FILE*

The parallel process standard output and standard error output are saved in the file. If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

The output destination varies depending on the restriction set in the job ACL (Access Control List) function of Job Operation Software.
If this option is permitted, the output is saved in the file with the name specified at FILE. In addition, metacharacters can be used for FILE.
If this option is not permitted, the output destination depends the job ACL function setting in the Job Operation Software.

Refer to the Job Operation Software manual for metacharacters specification and the job ACL function settings in the Job Operation Software.

{ -oferr | --oferr | -stderr | --stderr } *ERR_FILE*

The parallel process standard error output is saved in the file. If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

The output destination varies depending on the restriction set in the job ACL function of Job Operation Software.
If this option is permitted, the output is saved in the file with the name specified at *ERR_FILE*. In addition, metacharacters can be used for *ERR_FILE*.
If this option is not permitted, the output destination depends the job ACL function setting in the Job Operation Software.

Refer to the Job Operation Software manual for metacharacters specification and the job ACL function settings in the Job Operation Software.

{ -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc } *ERR_PROC_FILE*

Saves the parallel process standard error output to the file with the filename "*ERR_PROC_FILE.mpiexec.rank*" for each process.

In addition, metacharacters can be used for *ERR_PROC_FILE*. Refer to the Job Operation Software manual for metacharacters specification.

If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

{ -ofout | --ofout | -stdout | --stdout } *OUT_FILE*

The parallel process standard output is saved in the file. If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

The output destination varies depending on the restriction set in the job ACL function of Job Operation Software.
If this option is permitted, the output is saved in the file with the name specified at *OUT_FILE*. In addition, metacharacters can be used for *OUT_FILE*.
If this option is not permitted, the output destination depends the job ACL function setting in the Job Operation Software.

Refer to the Job Operation Software manual for metacharacters specification and the job ACL function settings in the Job Operation Software.

{ -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc } *OUT_PROC_FILE*

Saves the parallel process standard output to the file with the filename "*OUT_PROC_FILE.mpiexec.rank*" for each process.

In addition, metacharacters can be used for *OUT_PROC_FILE*. Refer to the Job Operation Software manual for metacharacters specification.

If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

{ -of-proc | --of-proc | -std-proc | --std-proc } *PROC_FILE*

Saves the parallel process standard output and standard error output to the file with the filename "*PROC_FILE.mpiexec.rank*" for each process.

In addition, metacharacters can be used for *PROC_FILE*. Refer to the Job Operation Software manual for metacharacters specification.

If just the filename or the relative path is specified as the output destination, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

{ -ofprefix | --ofprefix | -stdprefix | --stdprefix } *PREFIX*

Outputs the character string corresponding to the keyword specified at *PREFIX* at the start of the parallel process standard output and standard error output lines.

The output format of the character string depends on which keyword shown below is specified. Multiple keywords can be specified by separating them by commas "," (for example, date, rank, nid). When multiple keywords are specified, the character string corresponding to the keywords is output in the order of date, rank, nid.

date

The output time is attached at the start of the output character string.

rank

The rank under MPI_COMM_WORLD is attached at the start of the output character string.

nid

The node ID is attached at the start of the output character string.

If this option is specified more than once, the parameter specified last takes priority.

If a parallel process is created dynamically, the character string "*@spawn*" is added after the rank. The character string "*spawn*" indicates the number assigned by the Job Operation Software for each dynamically created process.

The node ID is the number that identifies the compute node allocated to the parallel process. Refer to the Job Operation Software manual for information concerning node IDs.

{ -stdin | --stdin } *STDIN_FILE*

Loads from the file with the filename specified at *STDIN_FILE*, the standard input for all parallel processes that were created by executing the MPI program. If just the filename or the relative path is specified, the relative path from the job execution current directory is used.

If this option is specified more than once, the parameter specified last takes priority.

{ -vcoordfile | --vcoordfile } *VCOORD*

Specifies that parallel processes are allocated on the basis of the process assignment information specified in the *VCOORD* file when an MPI program is executed. If just the filename or the relative path is specified, the relative path from the current directory of the mpiexec command process is used.

Ensure that this option is specified if background execution is used to execute more than one mpiexec command simultaneously. The maximum number of mpiexec command that can be executed simultaneously in background execution is 128.

Refer to "4.5 VCOORD file format" for details on the *VCOORD* file.

If this option is specified more than once, the parameter specified last takes priority.

{ -V | --version }

Displays version information for this command and ends mpiexec command.

Character strings in filenames for per-process standard output and standard error output are explained below.

The character string "*mpiexec*" indicates how many times the mpiexec command was executed in the job script. The character string "*rank*" indicates the actual rank under MPI_COMM_WORLD.

If a parallel process is created dynamically, the character string "*@spawn*" is added after the rank. The character string "*spawn*" indicates the number assigned by the Job Operation Software for each dynamically created process.

Note

Notes concerning parallel process standard input, standard output, and standard error output

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

Destinations of each process' standard output and standard error output according to the various -of/-std options specified for the mpiexec command are shown in the table below:

Table 4.4 Parallel process standard output and standard error output resulting from -of/-std option specifications

mpiexec option specification	Standard output	Standard error output
No type of -of/-std option specified	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job

mpiexec option specification	Standard output	Standard error output
	Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout" - For an interactive job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout(interact)"	Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr" - For an interactive job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr(interact)"
-of/-std option specified (permitted by the job ACL function of Job Operation Software)	*Specified file*	*Specified file*
-of/-std option specified (not permitted by the job ACL function of Job Operation Software)	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout" - For an interactive job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout(interact)"	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr" - For an interactive job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr(interact)"
-ofout/-stdout option specified (permitted by the job ACL function of Job Operation Software)	*Specified file*	-
-ofout/-stdout option specified (not permitted by the job ACL function of Job Operation Software)	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout" - For an interactive job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stdout(interact)"	-
-oferr/-stderr option specified (permitted by the job ACL function of Job Operation Software)	-	*Specified file*
-oferr/-stderr option specified (not permitted by the job ACL function of Job Operation Software)	-	Depends the job ACL function setting (following combination of the output item of pjacl commands) in the Job Operation Software. - For a batch job Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr" - For an interactive job

mpiexec option specification	Standard output	Standard error output
		Combination of "mpiexec-stdouterr-unit" and "mpiexec-stderr(interact)"
-of-proc/-std-proc option specified	*Specified file name.mpiexec.rank*	*Specified file name.mpiexec.rank*
-ofout-proc/-stdout-proc option specified	*Specified file name.mpiexec.rank*	-
-oferr-proc/-stderr-proc option specified	-	*Specified file name.mpiexec.rank*

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

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

Note

Notes on running large-scale MPI jobs

For MPI jobs that are highly parallel (Generates approximately 10000 or more MPI processes) and output standard output or standard error output to a file for each rank, it is recommend to run them as follows, considering the system load of writing to a file:

- Output the standard output/standard error output of the mpiexec command to a different file for each rank (process).
- Output the standard output/standard error output files for each rank to a different directory for every several files, rather than to the same directory.
- Do not create an empty file if there is no standard output/standard error output for ranks.

Example

Change the output directory for standard output and standard error output every 1000 rank numbers. Also, an empty file is not created if there is no standard output or standard error output.

```
export PLE_MPI_STD_EMPTYFILE="off"
mpiexec -stdout-proc ./%/1000R/%j.stdout -stderr-proc ./%/1000R/%j.stderr ./a.out
```

Note

Notes concerning DT_RPATH

If the DT_RPATH dynamic section attribute exists in a MPI program and /*installation_path*/lib64 is included in DT_RPATH, --debuglib option is disabled. Deal with either as follows for such a MPI program. "*installation_path*" is the product/software installation path. For "*installation_path*", contact the system administrator.

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

```
$ mpiexec -n 2 /lib64/ld-linux-aarch64.so.1 --inhibit-rpath :./a.out ./a.out
```

  - Set the environment variable PATH and LD_LIBRARY_PATH corresponding to the executed package.
  - Specify the argument of the --inhibit-rpath option of /lib64/ld-linux-aarch64.so.1 in the form of ":MPI program".

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  ]
[ { -fjdbg-dlock | --fjdbg-dlock } ]
[ { -fjdbg-sig | --fjdbg-sig } SIGNAL ]
[ { -fjdbg-out-dir | --fjdbg-out-dir } OUTPUT-DIR ]
[ { -gdbx | --gdbx } "[ RANK: ] COMMAND-FILE [ ;... ]" ]
```

-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, the maximum number of parallel processes that can be created is assumed.

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.

For execution of an MPI program in this computing system, the parallel processes can be allocated to an appropriate torus format. Refer to the Job Operation Software manual for information concerning how to specify torus format and how to deploy parallel processes.

{ -fjdbg-dlock | --fjdbg-dlock }

Enables the deadlock investigative function in the debugging support functions. For more information about the deadlock investigative function, refer to Debugger for Parallel Applications User's Guide.

{ -fjdbg-sig | --fjdbg-sig } *SIGNAL*

Enables the abnormal termination investigative function in the debugging support functions. For more information about the abnormal termination investigative function, refer to Debugger for Parallel Applications User's Guide.

{ -fjdbg-out-dir | --fjdbg-out-dir } *OUTPUT-DIR*

Specifies the directory to store the result file for the deadlock investigative function or abnormal termination investigative function in the debugging support functions. For more information about the deadlock investigative function and the abnormal termination investigative function, refer to Debugger for Parallel Applications User's Guide.

{ -gdbx | --gdbx } "[*RANK*:] *COMMAND-FILE* [;...]"

Enables the debugging control function with command files in the debugging support functions. For more information about the debugging control function with command files, refer to Debugger for Parallel Applications User's Guide.

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*.

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.*

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 4.5 MCA parameter specification methods and priorities

Rank	MCA parameter setting method	Example of use
1	-mca option of mpiexec command	-mca btl_tofu_eager_limit 4096
2	Environment variable	export OMPI_MCA_btl_tofu_eager_limit=4096
3	AMCA parameter file (MCA parameter settings file) specified in the -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 4.6 btl_tofu_eager_limit (changes the threshold value for switching the communication method)

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

MCA parameter value	Content
	information used internally is considered, the value is reduced by several hundreds of bytes, however that can be ignored here. `Large receive buffer size / 2` Usually, this software system dynamically determines an appropriate value internally as the "threshold value" for fast communication modes. This is because the distance of the compute node that is performing the communication is taken into account, not just the message size (number of bytes). For message communication between nearby compute nodes, for example, 45,352 is set internally. As the distance between the compute nodes performing the message communication increases, the "threshold value" also increases. With this MCA parameter, the specified value is used as the "threshold value" regardless of the compute node distance. Refer to "6.5 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 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 "6.8 Reduction Operation Sequence Guarantee in Collective Communication" for details.
0	Does not guarantee the reduction operation sequence. The reduction operation sequence may be changed internally to make the communication time as short as possible. Note that, if the communication conditions are the same, the computing results are the same regardless of the number of times the same program is executed. The default value for this parameter is 0.

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

Table 4.8 coll_tbi_intra_node_reduction (specifies the algorithms used in intra-node Tofu barrier communication for reduction operations of floating point type or complex type data if multiple processes are assigned within a node)

MCA parameter value	Content
2	If the conditions for applying Tofu barrier communication, as explained in "6.12.3 MPI_REDUCE and MPI_ALLREDUCE", are met, Tofu barrier communication is not used for reduction operations of floating point type or complex type data within a node. Instead, recursive_doubling algorithm implemented at software is used.
3	Tofu Barrier communication is used for reduction operations of floating point type or complex type data within a node. Binary_tree algorithm which can save the required number of gates is used. The default value for this parameter is 3.
4	Tofu Barrier communication is used for reduction operations of floating point type or complex type data within a node.

MCA parameter value	Content
	Although the required number of barrier gates is greater than when 3 is specified for this MCA parameter, recursive_doubling algorithm that better performance can be expected is used.

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

Table 4.9 coll (changes settings applied to all collective communications in common)

MCA parameter value	Content
^tbi	Specifies that Tofu barrier communication (hardware function) is not used in execution of the following MPI routines: MPI_BARRIER, MPI_BCAST, MPI_REDUCE, and MPI_ALLREDUCE. Refer to "6.12 Use of Tofu Barrier Communication for Better Performance" for details. Do not specify this parameter when you want to use Tofu barrier communication.

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

Table 4.10 coll_tbi_repeat_max (Controls the range of message length to communicate by Tofu barrier communication)

MCA parameter value	Content
Integer value of 1 or more	For the MPI_BCAST routine, MPI_REDUCE routine, and MPI_ALLREDUCE routine, Tofu barrier communication uses the count of this value as an upper bound. The number of elements in a message that can be operated per the Tofu barrier communication is limited. Therefore, when count exceeding the maximum number of elements in a message per the Tofu barrier communication is specified in an argument of the MPI routine, the Tofu barrier communication range can be expanded by implementing Tofu barrier communication multiple times. The default value for this parameter is 1. Refer to "Table 6.19 Combinations that allow the MPI_BCAST routine to apply Tofu barrier communication" and "Table 6.20 Operation combinations that allow the MPI_REDUCE and MPI_ALLREDUCE routines to apply Tofu barrier communication" for the maximum number of elements in a message per the Tofu barrier communication.

Example

- For the MPI_BCAST routine, the basic datatype (MPI_UINT64_T)
  - When 6 is specified for the number of elements in a message, the Tofu barrier communication function is applied.
  - When 7 is specified for the number of elements in a message, the Tofu barrier communication function is not applied by default, but if 2 is specified for this MCA parameter, the Tofu barrier communication function is applied.
- Using the MPI_TYPE_CONTIGUOUS routine for the MPI_BCAST routine, concatenate seven elements of the basic datatype (MPI_UINT64_T) to create a derived datatype.
  - When 1 is specified for the number of elements in a message, the Tofu barrier communication function is not applied by default, but if 2 is specified for this MCA parameter, the Tofu barrier communication function is applied.
  - When 2 is specified for the number of elements in a message, the Tofu barrier communication function is not applied by default, but if 3 is specified for this MCA parameter, the Tofu barrier communication function is applied.
- For the MPI_REDUCE routine, the basic datatype (MPI_DOUBLE), the reduction operation (MPI_SUM)
  - When 6 is specified for the number of elements in a message, the Tofu barrier communication function is applied, but if 2 is specified for this MCA parameter, the Tofu barrier communication function is applied.
  - When 9 is specified for the number of elements in a message, the Tofu barrier communication function is not applied, but if 3 is specified for this MCA parameter, the Tofu barrier communication function is applied.

Table 4.11 coll_tbi_use_on_bcast (uses Tofu barrier communication in MPI_BCAST routine)

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

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

Table 4.12 coll_tbi_use_on_comm_dup (uses Tofu barrier communication for a communicator created by MPI_COMM_DUP routine, MPI_COMM_IDUP routine, and MPI_COMM_DUP_WITH_INFO routine)

MCA parameter value	Content
1	Specifies that Tofu barrier communication is used for a communicator created by the MPI_COMM_DUP routine, MPI_COMM_IDUP routine, and MPI_COMM_DUP_WITH_INFO routine if Tofu barrier communication (hardware function), explained in "6.12 Use of Tofu Barrier Communication for Better Performance", is applied. The default value for this parameter is 1.
0	Specifies that Tofu barrier communication is not used for a communicator created by the MPI_COMM_DUP routine, MPI_COMM_IDUP routine, and MPI_COMM_DUP_WITH_INFO routine. Read "6.12.4 Notes on Tofu Barrier Communication" for details.

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

Table 4.13 coll_tbi_use_on_max_min (uses Tofu barrier communication for floating point datatypes MPI_MAX and MPI_MIN)

MCA parameter value	Content

1	Specifies that Tofu barrier communication is used on execution of floating point datatypes MPI_MAX and MPI_MIN operations in the MPI_REDUCE routine and MPI_ALLREDUCE routine.
	A calculation result to which Tofu barrier communication is applied might be different in either of special following conditions according to whether Tofu barrier communication is applied. This fact must be noted if the value specified for the MCA parameter is being changed.
	- When there is a NaN in the values used for the operation, note the following.
	  - When Tofu Barrier Communication is applied, the values except for NaN are used for comparison. However, when all values are NaN, the calculation result is one of them.
	  - When Tofu Barrier Communication is not applied, the values except for NaN are used for comparison or the calculation result is one of NaNs depending on the conditions at the time of execution.
	- When there are both of +0.0 and -0.0 in the values used for the operation, note the following.
	  - When Tofu Barrier Communication is applied, +0.0 is assumed to be greater than -0.0.
	  - When Tofu Barrier Communication is not applied, sign of 0.0 is not concerned. Therefore, which one of +0.0 and -0.0 is chosen is depending on the conditions at the time of execution.
	The default value for this parameter is 1.

0	Specifies that Tofu barrier communication is not used on the operation combinations explained by the content that the value of this parameter is 1. The default value for this parameter is 0.

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

Table 4.14 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 "6.13 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 4.15 coll_tuned_prealloc_size (specifies the size of the static work area used internally by the collective communication routine)

MCA parameter value	Content

Integer value of 1 or more	Specifies the size (MiB) of the work area allocated statically for the following collective communication routines:
	- MPI_ALLREDUCE routine
	- MPI_REDUCE routine
	- MPI_REDUCE_SCATTER_BLOCK routine
	- MPI_REDUCE_SCATTER routine
	- MPI_ALLGATHER routine
	- MPI_GATHER routine
	- MPI_SCATTER routine
	- MPI_ALLTOALL routine
	For MPI programs that call these routines multiple times, the MPI program execution time can be reduced by using this parameter effectively for statically allocating memory.
	The following size gives an estimate to be specified:
	For MPI_ALLREDUCE routine
	`(Size of messages sent in the program * 2) + 2MiB`
	For MPI_REDUCE routine and MPI_REDUCE_SCATTER_BLOCK routine
	`(Size of messages sent in the program * 3) + 2MiB`
	For MPI_REDUCE_SCATTER routine
	`(Size of messages sent in the program * 2) + 2MiB`
	For MPI_ALLGATHER routine
	`Size of messages received in the program + 2MiB`
	For MPI_GATHER routine
	`Size of messages received on the root process in the program + 2MiB`
	For MPI_SCATTER routine

MCA parameter value	Content

	`Size of messages sent on the root process in the program + 2MiB`
	For MPI_ALLTOALL routine
	`(Size of messages sent in the program * Number of ranks in communicators) + 2MiB`
	If the size of the work area allocated statically corresponding to this parameter specification is smaller than the work area size required to process the collective communication routine, the statically allocated work area is not used.
	The default value for this parameter is 6(MiB).
	However, if the value of the MCA parameter common_toufu_use_memory_pool is 1, the size of the work area allocated statically may be greater than the value specified for this parameter. Refer to "Table 4.31 common_tofu_use_memory_pool (uses memory pool for the MPI library)" for information on the MCA parameter common_tofu_use_memory_pool.

0	Specifies that the static work area is not allocated.

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

Table 4.16 common_tofu_conv_dim (executes an MPI program after converting the dimensions of the coordinate of each process to higher dimensions)

MCA parameter value	Content
2	Specifies that the MPI program runs as a two-dimensional job if possible. Valid only when the job is run in one dimension. Refer to "6.15 Job Dimension Conversion Function" for details.
3	Specifies that the MPI program runs as a three-dimensional job if possible. Valid only when the job is run in one or two dimensions. Refer to "6.15 Job Dimension Conversion Function" for details.
0	Specifies that the MPI program runs as a job in the specified dimension at the start of execution. If a value other than 0, 2, or 3 is specified, the specified value is assumed to be 0. The default value for this parameter is 0.

Table 4.17 common_tofu_conv_dim_log (outputs a log when executing an MPI program after converting the dimensions of the coordinate of each process to higher dimensions)

MCA parameter value	Content
1	Specifies that a parallel process with a rank of 0 outputs log about dimension conversion in the MPI library to standard error. If the dimension conversion function was not applied, the log contains the reason. Refer to "6.15.2 Log Output for Job Dimension Conversion Function" for details about the output contents.
0	Specifies that no logs are output for dimension conversion in the MPI library. If a value other than 0 or 1 is specified, the specified value is assumed to be 0. The default value for this parameter is 0.

Table 4.18 common_tofu_fastmode_threshold (changes the conditions for switching to fast communication mode)

MCA parameter value	Content
Integer value of 0 or more	Specifies the communication count used as the condition for switching from memory-saving communication mode to fast communication mode.

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

Table 4.19 common_tofu_large_recv_buf_size (changes the size of the Large receive buffer)

MCA parameter value	Content
Integer value from 1024 to 16777216	Specifies the size (number of bytes) of the Large receive buffer. If a value lower than 1024 is specified, a specification of 1024 is assumed. If a value greater than 16777216 is specified, a specification of 16777216 is assumed. Refer to "6.10 Suppressing Memory Usage" for information on the large receive buffer. The default value for this parameter is 1048576. However, if the value of the MCA parameter common_toufu_use_memory_pool is 1, the default value of this parameter is changed to 1048064. Refer to "Table 4.31 common_tofu_use_memory_pool (uses memory pool for the MPI library)" for information on the MCA parameter common_tofu_use_memory_pool.

Table 4.20 common_tofu_max_fastmode_procs (changes the upper limit for the number of processes that can communicate in fast communication mode)

MCA parameter value	Content
Integer value of -1 or 0 or more	Specifies the upper limit for the number of communication partner processes that each parallel process can communicate with in fast communication mode. If -1 is specified, communication with all processes is performed in fast communication mode. If 0 is specified, fast communication mode is not used and communication with all processes is in memory-saving communication mode. If a value of -2 or less is specified, a specification of -1 is assumed. Refer to "6.10 Suppressing Memory Usage" for details. The default value for this parameter is 256.

Table 4.21 common_tofu_max_tnis (changes the upper limit for the number of TNIs to be used)

MCA parameter value	Content
Integer value of 1 or more	Specifies the upper limit for the number of network interface devices (TNIs) to be used. If a number greater than the maximum number (6) is specified, the parameter value becomes the number that can actually be used. Refer to "6.7 Using Multiple TNIs" for details.
-1	Specifies to use the maximum number of network interface devices (TNIs) that can be used. If a value of 0 or -2 or less is specified, a specification of -1 is assumed. The default value for this parameter is -1.

Table 4.22 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)

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

Table 4.23 common_tofu_memory_limit (specifies the memory allocation limit value)

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

Table 4.24 common_tofu_memory_limit_peers (specifies the assumed number of communication partner processes when the memory allocation is limited)

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

Table 4.25 common_tofu_memory_saving_method (changes the method used for the memory-saving communication mode)

MCA parameter value	Content
1	Uses the method which uses the Medium receive buffer when a communication using the memory-saving communication mode is performed. If a value less than 1 or greater than 2 is specified, the specified value is assumed to be 1. Refer to "6.10 Suppressing Memory Usage" for details. The default value for this parameter is 1.
2	Uses the method which uses the Shared receive buffer when a communication using the memory-saving communication mode is performed. Refer to "6.10 Suppressing Memory Usage" for details.

Table 4.26 common_tofu_num_mrq_entries (change the number of entries in a completion queue)

MCA parameter value	Content
One of the following integer values: 2048, 8192, 32768, 131072, 524288, or 2097152	Specify the number of entries in a completion queue of Tofu interconnect. If you encountered an error message starting with [mpi::common-tofu::tofu-mrq-overflow], the error may be avoided by changing the value of this parameter larger. Or memory usage of a MPI process can be decreased by changing this value smaller. Specifiable values for this parameter are either 2048, 8192, 32768, 131072, 524288, or 2097152. If a value that is not in the list is specified, the closest value in the list is assumed. If there are two closest values, the larger value is assumed. The default value for this parameter is 131072.

MCA parameter value	Content
	Refer to "7.3 Communication Library Error Messages" for details of the error message starting with [mpi::common-tofu::tofu-mrq-overflow].

Table 4.27 common_tofu_packet_gap (changes the packet transfer interval time)

MCA parameter value	Content
Integer value from 0 to 255	An integer value, called a gap value, specifies the packet transfer interval time. One unit of the gap value corresponds to 1/8 of the time taken to transfer data of the packet maximum transmission unit, described below. An integer value from 0 to 255 can be specified for this parameter. If a value of -1 or less is specified, 0 is assumed. If a value of 255 or more is specified, 255 is assumed. The default value for this parameter is 0. Message transfer is performed within the MPI library in units called packets. One packet has an upper limit value, known as the maximum transmission unit. If a message larger than the maximum transmission unit is being transferred, the message is split into multiple packets so that the size of each packet is the maximum transmission unit or less. This maximum transmission unit can be changed by the MCA parameter common_tofu_packet_mtu. Refer to "Table 4.28 common_tofu_packet_mtu (changes the maximum packet transfer size)" for information on this parameter. Bandwidth can be controlled by adjusting the packet transfer interval time. This may improve communication throughput time when other message communication is being attempted at the same time. For example, if gap value 8 is specified in this parameter, the time interval between packets is exactly the amount of time it takes to transfer one packet and the targeted message transfer bandwidth is adjusted to 1/2. However, depending on the circumstances, performance may deteriorate, so care is essential when using this parameter.

Table 4.28 common_tofu_packet_mtu (changes the maximum packet transfer size)

MCA parameter value	Content
One of the following integer values: 256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536, 1664, 1792, or 1920	Message transfer is performed within the MPI library in units called packets. This parameter specifies the packet maximum transmission unit in bytes. Specifiable values for this parameter are either 256, 384, 512, 640, 768, 896, 1024, 1152, 1280, 1408, 1536, 1664, 1792, or 1920. If a value that is not in the list is specified, the closest value in the list is assumed. If there are two closest values, the larger value is assumed. The default value for this parameter is 1920. If communication of multiple large messages is attempted, communication throughput times can be improved by changing the value of this parameter in conjunction with specifying the MCA parameter common_tofu_packet_gap. However, depending on the circumstances, performance may deteriorate, so care is essential when using this parameter. Refer to "Table 4.27 common_tofu_packet_gap (changes the packet transfer interval time)" for details.

Table 4.29 common_tofu_shared_recv_buf_size (changes the size of the Shared receive buffer)

MCA parameter value	Content
Integer value greater than or equal to 65536	Specifies the size (in bytes) of the Shared receive buffer. If the specified value is not a power of 2, the value is rounded up to the next power of 2. If the specified value is less than 65536, the specified value is assumed to be 65536. Refer to "6.10 Suppressing Memory Usage" for details of the Shared receive buffer. The default value for this parameter is 16777216.

Table 4.30 common_tofu_use_multi_path (performs point-to-point communication using multiple communication paths)

MCA parameter value	Content
1	Specifies that multiple communication paths are used for point-to-point communication, that is, that trunking is implemented. Since multiple TNIs are used to reserve multiple communication paths, the result of this parameter is also affected by the number of usable TNIs. In addition, depending on communication conditions, communication performance may deteriorate, so care is essential when using this parameter. Read "6.7 Using Multiple TNIs" for details of using multiple TNIs.
0	Specifies that multiple communication paths are not used for point-to-point communication. The default value for this parameter is 0.

Table 4.31 common_tofu_use_memory_pool (uses memory pool for the MPI library)

MCA parameter value	Content
1	Uses memory pool for the MPI library. By using the memory pool which is a multiple of page size in size and aligned by page boundary, the memory region of the communication buffers held within the MPI library and the memory region of the user program are allocated on separate pages. This allows you to avoid restrictions on process creation from inside an MPI program if there is no uncompleted communication. However, note that you may not be able to avoid the restrictions when the value 2 is specified for the MCA parameter common_tofu_memory_saving method even if this function is enabled. See "6.9 Process Creation from Inside an MPI Program" for the restrictions. Also note that enabling this function has the following effects. - The default value of the Large receive buffer size is set to not 1048576 bytes but 1048064 bytes. See "6.10 Suppressing Memory Usage" for information on the Large receive buffer. - Memory usage may increase. - The MPI library uses large pages even if the environment variable XOS_MMM_L_HPAGE_TYPE=none is specified. See "Job Operation Software End-user's Guide for HPC Extensions" for information on the environment variable XOS_MMM_L_HPAGE_TYPE and large page. If a value other than 0 or 1 specified for this parameter, the specified value is assumed to be 1.
0	Does not use memory pool for the MPI library. The memory region of the communication buffers held within the MPI library may be allocated on the same page as the memory region of the user program. For this reason, you may not be able to avoid restrictions on process creation from inside an MPI program even if there is no uncompleted communication. See "6.9 Process Creation from Inside an MPI Program" for the restrictions. The default value for this parameter is 0.

Table 4.32 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 4.33 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 "6.17.2 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 4.34 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). 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 4.35 mpi_no_establish_communication (specifies that the MPI program does not establish communication using background execution)

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

MCA parameter value	Content
0	Specifies that the MPI program may establish communication between two groups of MPI processes that do not share a communicator using background execution of mpiexec command. The default value for this parameter is 0.

Table 4.36 mpi_preconnect_mpi (specifies the timing for establishing connections)

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

Table 4.37 mpi_print_stats (outputs MPI statistical information)

MCA parameter value	Content
1	Specifies to output MPI statistical information to the standard error output. With this specification, the MPI statistical information of all parallel processes is aggregated and output by parallel processes belonging to MPI_COMM_WORLD with a rank of 0. However, for the information about collective communication algorithms, only the information about communications that the process belonging to MPI_COMM_WORLD with a rank of 0 involved is displayed. The MPI statistical information is output when the MPI_FINALIZE routine is called.
2	Specifies to output MPI statistical information to the standard error output. With this specification, the MPI statistical information of each parallel process is output separately by each parallel process itself. The MPI statistical information is output when the MPI_FINALIZE routine or MPI_ABORT routine is called. It is also output when this software system terminates parallel processes due to a detected error. Process Mapping information is output only at the normal termination. To output statistical information from a particular parallel process, specify the MCA parameter mpi_print_stats_ranks. Refer to "Table 4.38 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)" for details.
3	It is similar to parameter value 1. However, it is necessary to specify the FJMPI_COLLECTION_PRINT routine to output it to the standard error output. In addition, the content of the output is output separately for the header department, body department including section line, and the footer department. Refer to "5.2.1.3 FJMPI_COLLECTION_PRINT" for details.
4	It is similar to parameter value 2. However, it is necessary to specify the FJMPI_COLLECTION_PRINT routine to output it to the standard error output. In addition, the

MCA parameter value	Content
	content of the output is output separately for the header department, body department including section line, and the footer department. Refer to "5.2.1.3 FJMPI_COLLECTION_PRINT" for details.
0	Specifies to not output MPI statistical information. Refer to "6.16 MPI Statistical Information" for information on MPI statistical information. If a value other than an integer from 0 to 2 is specified in this parameter, 0 is assumed. The default value for this parameter is 0.

Table 4.38 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)

MCA parameter value	Content
0 or higher integer value	Specifies the rank of the parallel process that outputs MPI statistical information. This MCA parameter is enabled only if 2 or 4 is specified for the MCA parameter mpi_print_stats. Specify the rank belonging to MPI_COMM_WORLD. Multiple ranks can be specified, separated by commas ",". If a rank that does not exist is specified, it is ignored. Refer to "Table 4.37 mpi_print_stats (outputs MPI statistical information)" for information on the MCA parameter mpi_print_stats.
-1	Specifies to output MPI statistical information from all parallel processes. This MCA parameter is enabled only if 2 or 4 is specified for the MCA parameter mpi_print_stats. Refer to "Table 4.37 mpi_print_stats (outputs MPI statistical information)" for details. If a value of -1 or less is specified in this parameter, a specification of -1 is assumed. The default value for this parameter is -1.

Table 4.39 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 4.40 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 4.41 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.

MCA parameter value	Content
	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 Refer to "6.3 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 (except for the case that an assistant core is used). Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details of the use of an assistant core. The default value for this parameter is 0.

Table 4.42 opal_progress_thread_mode (specifies the operation mode of the MPI asynchronous processing progress thread)

MCA parameter value	Content
1	Specifies to use manual section (without MPI call) mode to promote asynchronous communication using an assistant core. Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details.
2	Specifies to use manual section (with MPI call) mode to promote asynchronous communication using an assistant core.
3	Specifies to use automatic section mode to promote asynchronous communication using an assistant core.
0	Specifies that function of promoting asynchronous communication using an assistant core is not used. The default value for this parameter is 0. If a value smaller than 0 or larger than 3 is specified in this parameter, the error messages starting with [mpi::mca-var::invalid-value-enum] and [mpi::opal-runtime::opal_init:startup:internal-failure] might be output, and execution of the mpiexec command ends.

Table 4.43 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 "6.17.1 Communication Timeout Setting" for details.
0	Specifies to disable the communication timeout setting function. The default value for this parameter is 0.

Table 4.44 plm_ple_cpu_affinity (specifies CPU affinity for MPI processes)

MCA parameter value	Content
1	Specifies that the optimum number of CPUs (cores) is bound to each MPI process if neither compiler automatic parallelization function nor OpenMP function is used.

MCA parameter value	Content
	If those functions are used, this MCA parameter specification is disabled because CPU (core) binding for parallel threads is performed by the compiler. Which CPU (core) bind to the process is decided based on the numanode_assign_policy in the VCOORD file or the MCA parameter plm_ple_numanode_assign_policy. The default value for this parameter is 1.
0	Specifies that MPI processes are not bound to CPUs (cores) and they are scheduled by the operating system, if neither compiler automatic parallelization function nor OpenMP function is used. In case of specifying this MCA parameter value, two or more processes may be bound to one CPU (core), but only rarely. Use of the sched_setaffinity function in case of specifying this MCA parameter value is recommended. If those functions are used, this MCA parameter specification is disabled because CPU (core) binding for parallel threads is performed by the compiler. Refer to Fujitsu compiler manuals for details of CPU (core) binding for threads. The operation is not guaranteed if a value other than 0 or 1 is specified.

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

Table 4.45 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)

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

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

Table 4.46 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)

MCA parameter value	Content
Either of following value simplex share_cyclic share_band	Specifies the CPUs (cores) allocation policy that allocates the MPI process to the NUMA nodes. The value below can be specified. Refer to "Table 4.54 CPU (core) allocation policy" for details. - simplex: The processes are allocated to the NUMA node without sharing with other processes. - share_cyclic: The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in different NUMA nodes. - share_band: The processes are allocated to the NUMA node with sharing with other processes. The processes are sequentially allocated in a same NUMA node. Refer to the Job Operation Software manual for information on NUMA node. The specification of the CPUs (cores) allocation policy is decided by the following priority levels: 1. Specification with VCOORD file 2. Specification with this parameter 3. (If numanode_assign_policy is not specified in the VCOORD file, and this parameter is omitted) share_cyclic

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

Table 4.47 pml_ob1_use_stride_rdma (use of Stride RDMA communication)

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

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

4.3 Environment Variables

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.

PLE_MPI_STD_EMPTYFILE

If there is no output to the parallel process standard output and standard error output, specify whether to create an empty file.

on: The empty file is created. (default)

off: The empty file is not created.

This is applied to both the output file for each execution of mpiexec specified with --std option etc. and the output file for each process specified with --std-proc option etc.

If a value other than on or off is specified in this parameter, the behavior depends on the job ACL function ("mpiexec-std-emptyfile" by the pjacl command) setting in the Job Operation Software. Refer to the Job Operation Software manual for the job ACL function settings.

UTOFU_SWAP_PROTECT

On a compute node, an allocated memory region may be swapped out due to memory shortage or other reasons. When a memory region used for communication on the Tofu interconnect is swapped out, an error may occur in communication using the memory region. In this case, the MPI process outputs any of the following error messages and terminates abnormally.

- An error message containing the string "[mpi::common-tofu::tcq-error] Communication error is reported by Tofu TCQ.".

- An error message containing the string "[mpi::common-tofu::mrq-error] Communication error is reported by Tofu MRQ.".
- An error message containing the string "[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ.".

By specifying the environment variable UTOFU_SWAP_PROTECT, you can select whether to exclude the memory region for communication from swap out.

An integer value 0 or 1 can be specified, and the default value is 0. If a value other than 0 or 1 is specified, it is assumed to be 1.

When the value 1 is specified for this environment variable, the mlock system call is called inside the MPI library and the memory region used for communication on the Tofu interconnect is guaranteed not to be subject to swap out. The size of the memory region that can be excluded from the swap out target is subject to the software resource limit RLIMIT_MEMLOCK. See the Job Operation Software manual for details on how to check or change the RLIMIT_MEMLOCK at job execution.

When the value 0 is specified for this environment variable, the memory region used for communication on the Tofu interconnect may be subject to swap out.

- Communication performance may decrease if a memory region for communication is excluded from swap out.
- If the value of RLIMIT_MEMLOCK is less than the total size of the memory region for communication used by the job, the communication error due to swap out may not be avoided even if the value 1 is specified for this environment variable.

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.

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. Refer to the manuals of the language processing systems (Fortran User's Guide, C User's Guide, and C++ User's Guide) for 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. However, if an MPI program ends abnormally after receiving a signal, the return value is a logical sum of the received signal number and 0x80.

- 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 4.48 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 after receiving a signal. The value is the logical sum of the signal number received when the MPI program ended abnormally and 0x80. This behavior is based on how an exit status is handled in general UNIX shells when a command exits after receiving a signal.
255	Indicates the return value if the parallel execution environment side of Job Operation Software ended abnormally.

4.5 VCOORD file format

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

Table 4.49 Form of the coordinates

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

Table 4.50 Form of the number of CPUs (cores)

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

Table 4.51 Specification of NUMA memory allocation policy

Content	Format
Method of allocating NUMA memory	memory_allocation_policy=value

Either of value in "Table 4.45 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)" can be specified for a policy.

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

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

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

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

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

Either of value in "Table 4.46 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)" can be specified for a policy.

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

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

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

The following examples show the format of the VCOORD file.

Example

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

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

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

Format 2. Specifying only the logical coordinates

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

```
(0)
(0)
(1)
(1)
(2)
(2)
(3)
(3)
```

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

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

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

Format 4. Specifying the NUMA memory allocation policy

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

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

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

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

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

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

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

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

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

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

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

In the following cases, Job Operation Software error occurs

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

4.6 Execution of Multiple MPI Programs on the Same Node

In this software system, multiple MPI programs can be executed on the same node without using the MPMD model. Refer to the Job Operation Software manual for details on the execution method.

This section provides notes on execution of multiple MPI programs on the same node.

In this section, the mpiexec command, the MPI_COMM_SPAWN routine, and MPI_COMM_SPAWN_MULTIPLE routine are collectively referred as the process creation procedure.

- Specify the coordinates with the VCOORD file as for the process creation procedure executed later than the first execution of the process creation procedure. If only the number of CPUs (cores) is specified with the VCOORD file, the coordinates allocated to the processes are decided by Job Operation Software.
- If the number of CPUs (cores) is not specified with the VCOORD file, the number of CPUs (cores) per node is the value calculated from the expression :

```
"The number of CPUs (cores) per node"
    / "The value specified with --mpi max-proc-per-node option of pjsub command"
```

- If the number of processes per node exceeds the value specified with "--mpi max-proc-per-node" option of pjsub command, an error occurs in the process creation procedure. In that case, mpiexec command is returned with the value 1, and the MPI_COMM_SPAWN routine or MPI_COMM_SPAWN_MULTIPLE routine returns the error class MPI_ERR_SPAWN as the error code.
- If multiple process creation procedures are executed without the "--mpi max-proc-per-node" option of the pjsub command, the limit of the number of processes per node is the value decided with the "--mpi proc" option of the pjsub command.
- If the MCA parameter mpi_no_establish_communication is specified, multiple MPI programs cannot be executed on the same node. The specification of MCA parameter mpi_no_establish_communication is mutually exclusive to the specification of "--mpi max-proc-per-node" option of pjsub command. If both of them are specified, the specification of MCA parameter mpi_no_establish_communication is effective.

4.7 Settings in NUMA system

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

4.7.1 Setting value of NUMA memory allocation policy

The memory policy can be set by the MCA parameter plm_ple_memory_allocation_policy. Values that can be specified are shown in the table below. Refer to "Table 4.45 plm_ple_memory_allocation_policy (specifies the NUMA memory policy)" for information on the MCA parameter plm_ple_memory_allocation_policy.

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

- All node set (all)

  Set of all NUMA nodes in a compute node

- Local node set (local)

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

- Non-local node set (nonlocal)

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

Table 4.53 NUMA memory allocation policy

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

Value	Content	Note
		that case, the warning message PLE 0601 is output to the standard error output of the job, and processing is continued. In this case, the NUMA memory allocation policy of parallel processes is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_DEFAULT,NULL, ..)
interleave_all	The memory is alternately allocated from each NUMA node in "All node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_INTERLEAVE,all, ..)
bind_local	The memory allocations will come from the NUMA node that belongs to "Local node set" with the lowest numeric node ID first. It fails in the allocation when there is no remainder capacity of the memory of the NUMA node that belongs to "Local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,local, ..)
bind_nonlocal	The memory allocations will come from the NUMA node that belongs to "Non-local node set" with the lowest numeric node ID first. It fails in the allocation when there is no remainder capacity of the memory of the NUMA node that belongs to "Non-local node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,nonlocal, ..) It fails in the call of set_mempolicy(2) when "Non-local node set" is empty. In that case, the warning message PLE 0601 is output to the standard error output of the job, and processing is continued. In this case, the NUMA memory allocation policy of parallel processes is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_DEFAULT,NULL, ..)
bind_all	The memory is allocated in the NUMA nodes of "All node set".	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_BIND,all, ..)
prefer_local	The lowest numeric node ID in the NUMA node that belongs to "Local node set" will be selected as the preferred node. The memory allocation is done the preferred node by priority. If that NUMA node contains no free memory, the system will attempt to allocate memory from a "nearby" node.	It is equal to the result of calling the following system calls from the parallel processes. set_mempolicy (MPOL_PREFERRED,local, ..)
prefer_nonlocal	The lowest numeric node ID in the NUMA node that belongs to "Non-local node set" will be selected as the preferred node. The memory allocation is done the preferred node by priority. If that NUMA node contains	It is equal to the result of calling the following system calls from the parallel processes.

Value	Content	Note
	no free memory, the system will attempt to allocate memory from a "nearby" node.	set_mempolicy (MPOL_PREFERRED,nonlocal, ..) The specification of "nonlocal" is disregarded when "Non-local node set" does not exist, and it is equal to the result of specifying "localalloc".

4.7.2 Setting value of CPU (core) allocation policy

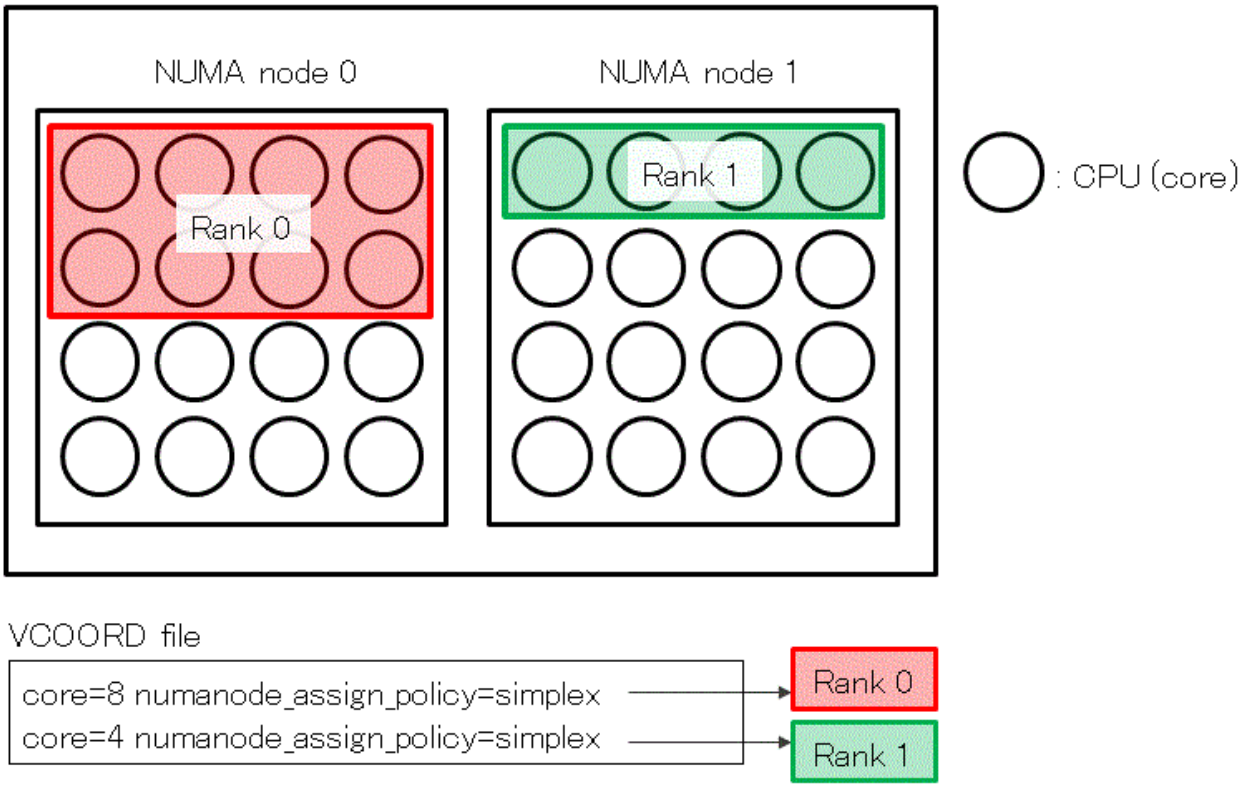
The CPU (core) allocation policy can be set by the MCA parameter plm_ple_numanode_assign_policy. Values that can be specified are shown in the table below. Refer to "Table 4.46 plm_ple_numanode_assign_policy (specifies the CPUs (cores) allocation policy to the NUMA nodes)" for information on the MCA parameter plm_ple_numanode_assign_policy.

Table 4.54 CPU (core) allocation policy

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

The allocation images are shown as follows.

Figure 4.1 Example of allocating process for simplex

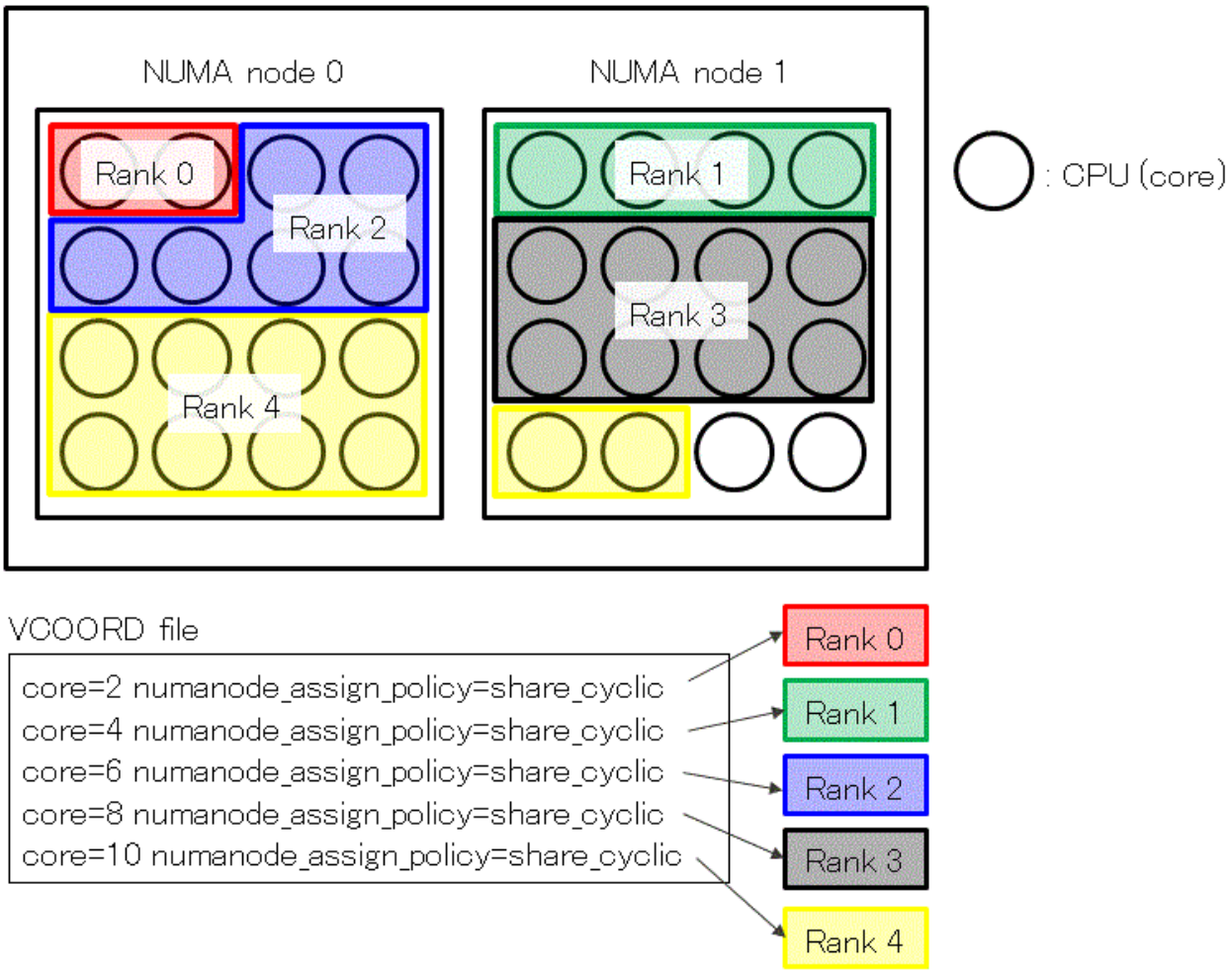


In the "Figure 4.1 Example of allocating process for simplex", rank 0 (red) and rank 1 (green) are assigned to different NUMA nodes in order. A NUMA node is never shared by multiple ranks.

In the "Figure 4.1 Example of allocating process for simplex", the number of CPUs (cores) allocated to each process is specified with the VCOORD file as follows.

- Rank 0 uses 8 CPUs (cores).
- Rank 1 uses 4 CPUs (cores).

Figure 4.2 Example of allocating process for share_cyclic

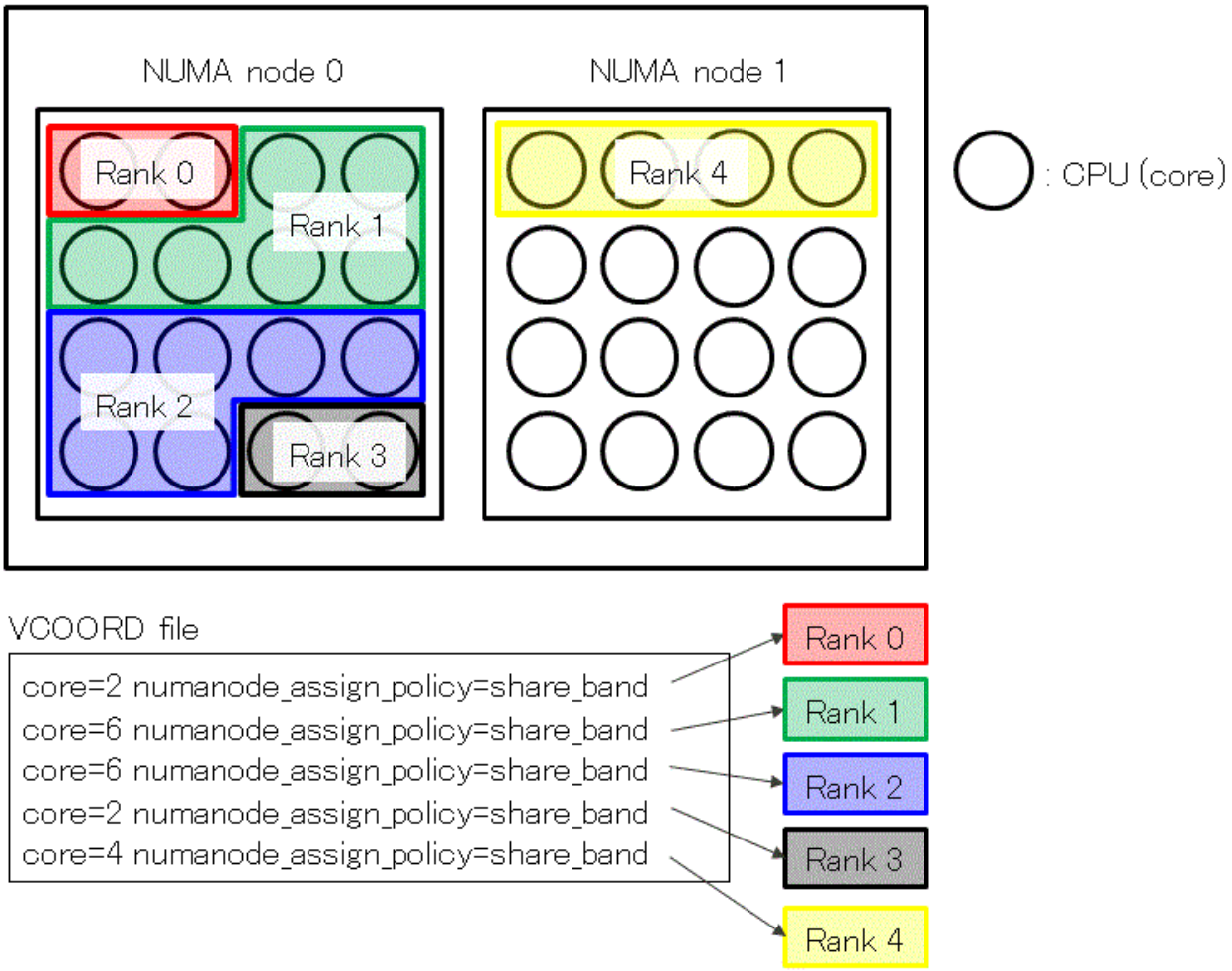


In the "Figure 4.2 Example of allocating process for share_cyclic", rank 0 (red), rank 1 (green), rank 2 (blue), rank 3 (gray), and rank 4 (yellow) are assigned to different NUMA nodes in a round-robin fashion. A NUMA node can be shared by multiple ranks because ranks are assigned to free CPUs (cores) within NUMA nodes.

In the "Figure 4.2 Example of allocating process for share_cyclic", the number of CPUs (cores) allocated to each process is specified with the VCOORD file as follows.

- Rank 0 uses 2 CPUs (cores).
- Rank 1 uses 4 CPUs (cores).
- Rank 2 uses 6 CPUs (cores).
- Rank 3 uses 8 CPUs (cores).
- Rank 4 uses 10 CPUs (cores).

Figure 4.3 Example of allocating process for share_band



In the "Figure 4.3 Example of allocating process for share_band", rank 0 (red), rank 1 (green), rank 2 (blue), rank 3 (gray), and rank 4 (yellow) are assigned to pack into the same NUMA node in that order. A NUMA node can be shared by multiple ranks because ranks are assigned to free CPUs (cores) within NUMA nodes.

In the "Figure 4.3 Example of allocating process for share_band", the number of CPUs (cores) allocated to each process is specified with the VCOORD file as follows.

- Rank 0 uses 2 CPUs (cores).
- Rank 1 uses 6 CPUs (cores).
- Rank 2 uses 6 CPUs (cores).
- Rank 3 uses 2 CPUs (cores).
- Rank 4 uses 4 CPUs (cores).

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".
- When the job type is node-sharing job, extended interfaces, Rank query interface and Extended persistent communication requests interface, cannot be used.
  Refer to the Job Operation Software manual for information on node-sharing job.

5.1 Rank Query Interface

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

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

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 where the MPI processes belonging to the MPI_COMM_WORLD created internally when MPI_INIT is executed are deployed.

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

<Return value>

Normal	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	If the job type is node-sharing job (Refer to the Job Operation Software manual for information on node-sharing job)

<Notes>

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

- This 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 given to the MPI_COMM_WORLD created internally when the MPI_INIT routine is executed.

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

<Return value>

Normal	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	If the job type is node-sharing job (Refer to the Job Operation Software manual for information on node-sharing job)

<Notes>

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

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

- This 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 logical coordinates or the Tofu coordinates corresponding to the rank of a process in the specified communicator.

- To get the logical 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 logical coordinates of the node which corresponds to the arguments comm and rank are stored in the coords argument. The coordinates X, Y, Z are stored in coords[0], coords[1], coords[2], respectively.

- To get the Tofu coordinates (the coordinates actually allocated on the system)

  Specify FJMPI_TOFU_SYS 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.

- To get the Tofu coordinates (the relative coordinates based on rank 0 in the argument comm)

  Specify FJMPI_TOFU_REL for the view argument, and 6 for the maxdims argument. The Tofu coordinates (the relative coordinates based on rank 0 in the argument comm) 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.

Type	Variable	Explanation	IN/OUT
MPI_Comm	comm	Specify the communicator.	IN
int	rank	Specify the rank of a process in the communicator comm. If the communicator comm is an inter-communicator, specify the rank in the remote group.	IN
int	view	Specify the macro defined to choose the logical coordinates or the Tofu coordinates. - FJMPI_LOGICAL The logical coordinates. - FJMPI_TOFU_SYS The Tofu coordinates (The coordinates actually allocated on the system). - FJMPI_TOFU_REL The Tofu coordinates (The relative coordinates based on rank 0 in the argument comm. If the communicator comm is an inter-communicator, the relative coordinates based on rank 0 in the remote group of the argument comm).	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

<Return value>

Normal	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	If the job type is node-sharing job (Refer to the Job Operation Software manual for information on node-sharing job)

<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, the specified value for the argument maxdims can be different from the job shape. In that case, coordinates whose number of dimensions is same as lesser value of the two are gotten with this routine.

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

5.1.3 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 on the specified logical coordinates or Tofu 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 one process is allocated on the logical coordinates

  Specify FJMPI_LOGICAL for the view argument, the number greater than 0 for the maxppn argument, the logical coordinates X, Y, Z for the coords[0], coords[1], coords[2] arguments, respectively. The rank of the process which is 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 four processes are allocated on the Tofu coordinates

  Specify FJMPI_TOFU_SYS or FJMPI_TOFU_REL for the view argument, the number greater than 4 for the maxppn argument, the Tofu 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 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 defined to choose the logical coordinates or the Tofu coordinates. - FJMPI_LOGICAL The logical coordinates.	IN

Type	Variable	Explanation	IN/OUT
		- FJMPI_TOFU_SYS The Tofu coordinates (the coordinates actually allocated on the system). - FJMPI_TOFU_REL The Tofu coordinates (The relative coordinates based on rank 0 in the argument comm. If the communicator comm is an inter-communicator, the relative coordinates based on rank 0 in the remote group of the argument comm).	
int[]	coords	Specify the coordinates where processes which have ranks to get are 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. If the inter-communicator is specified for the comm argument, the ranks of processes in the remote group are stored.	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	If the job type is node-sharing job (Refer to the Job Operation Software manual for information on node-sharing job)

<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 same as the number of dimensions in the job shape. 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

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>

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

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

<Return value>

Normal	FJMPI_SUCCESS	-
Error	FJMPI_ERR_TOPOLOGY_INVALID_COMM	- If an inter-communicator was specified - If a communicator that does not have a Cartesian structure was specified

<Notes>

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

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

5.1.5 Sample Program

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

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

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

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

#define FAILURE 1

int main(int argc, char *argv[])
{
    int coords[3], i, size, rank;
```

```
int rc, dim;
int shape_x, shape_y, shape_z;
int tmp_coords[3];
int next_coords[3];
int ans;
int outppn;
char host[255];

gethostname(host, 255);

MPI_Init(&argc, &argv);

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

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

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

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

/*****************************************************************************
 * Get own coordinates                                                       *
 *****************************************************************************/
rc = FJMPI_Topology_get_coords(MPI_COMM_WORLD, rank, FJMPI_LOGICAL, 3, coords);
if (FJMPI_SUCCESS != rc) {
    fprintf(stderr, "[%s] FJMPI_Topology_get_coords ERROR\n", host);
    MPI_Abort(MPI_COMM_WORLD, FAILURE);
}

/*****************************************************************************
 * Get neighboring processes before and after each coordinate                *
 *****************************************************************************/
for(i = 0; i < 6; i++)
{
    switch(i)
    {
        case 0: /* Neighboring X axis */
        case 1:
            tmp_coords[0] = (0 == i) ?
            (coords[0] - 1 >= 0) ? coords[0] - 1 : shape_x - 1
            : (coords[0] + 1 < shape_x) ? coords[0] + 1 : 0;
            tmp_coords[1] = coords[1];
            tmp_coords[2] = coords[2];
            break;
        case 2: /* Neighboring Y axis */
        case 3:
            tmp_coords[1] = (2 == i) ?
            (coords[1] - 1 >= 0) ? coords[1] - 1 : shape_y - 1
            : (coords[1] + 1 < shape_y) ? coords[1] + 1 : 0;
            tmp_coords[0] = coords[0];
```

```
                tmp_coords[2] = coords[2];
                break;
            case 4: /* Neighboring Z axis */
            case 5:
                tmp_coords[2] = (4 == i) ?
                (coords[2] - 1 >= 0) ? coords[2] - 1 : shape_z - 1
                : (coords[2] + 1 < shape_z) ? coords[2] + 1 : 0;
                tmp_coords[0] = coords[0];
                tmp_coords[1] = coords[1];
                break;
        }

        rc = FJMPI_Topology_get_ranks(MPI_COMM_WORLD, FJMPI_LOGICAL, tmp_coords,
                                      1, &outppn, &ans);
        switch (rc)
        {
            case FJMPI_SUCCESS:
                break;
            case FJMPI_ERR_TOPOLOGY_NO_PROCESS:
                /* Searches until the closest MPI process is found */
                while (rc == FJMPI_ERR_TOPOLOGY_NO_PROCESS) {
                    coords[0] = coords[0] - 1;
                    rc = FJMPI_Topology_get_ranks(MPI_COMM_WORLD, FJMPI_LOGICAL, coords,
                                                  1, &outppn, &ans);
                }
                break;
            default:
                fprintf(stderr, "[%s] FATAL ERROR\n", host);
                MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }

        /************************************************************************
         * Checks if used coordinates and fetched coordinates are the same      *
         ************************************************************************/
        rc = FJMPI_Topology_get_coords(MPI_COMM_WORLD, ans, FJMPI_LOGICAL, 3, next_coords);
        if (MPI_SUCCESS != rc) {
            fprintf(stderr, "[%s] FJMPI_Topology_rank2xyz ERROR\n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
        if ((next_coords[0] != tmp_coords[0]) ||
            (next_coords[1] != tmp_coords[1]) ||
            (next_coords[2] != tmp_coords[2])) {
            fprintf(stderr, "[%s] PARAM ERROR\n", host);
            fprintf(stderr, "[%s %d] [user:%u-%u-%u] [get:%u-%u-%u] [rank|next:%d|%d]\n",
            host, i, tmp_coords[0], tmp_coords[1], tmp_coords[2],
            next_coords[0], next_coords[1], next_coords[2], rank, ans);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
    }

    MPI_Finalize();

    return 0;
}
```

5.2 MPI Statistical Information Section Specifying Interface

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

Table below shows the overview of section specifying 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 starts measuring MPI statistics.

This routine is enabled only if the value of MCA parameter mpi_print_stats is equal to 3 or 4. However, this routine is ignored if any of the following conditions is met.

- When this routine is called before MPI_INIT routine.
- When this routine is called after MPI_FINALIZE routine.

<Return value>

None.

<Notes>

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

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 stops measuring MPI statistics.

This routine is enabled only if the value of MCA parameter mpi_print_stats is equal to 3 or 4. However, this routine is ignored if any of the following conditions is met.

- When this routine is called before MPI_INIT routine.
- When this routine is called after MPI_FINALIZE routine.

<Return value>

None.

<Notes>

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

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

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>

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

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

This routine prints measuring MPI statistical information to standard error output.

This routine is enabled only if the value of MCA parameter mpi_print_stats is equal to 3 or 4. However, this routine is ignored if any of the following conditions is met.

- When this routine is called before MPI_INIT routine.
- When this routine is called after MPI_FINALIZE routine.

The behavior of this routine changes depending on the value of MCA parameter mpi_print_stats.

mpi_print_stats	Operation	Explanation
3	The collective operation	This routine must be executed by all processes in MPI_COMM_WORLD.

mpi_print_stats	Operation	Explanation
4	The independent operation	This routine can be executed independently of other processes.

<Return value>

None.

<Notes>

Please specify a string of printable single-byte characters.

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

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 clears all MPI statistical data.

This routine is enabled only the value of MCA parameter mpi_print_stats is equal to 3 or 4. However, this routine is ignored if any of the following conditions is met.

- When this routine is called before MPI_INIT routine.
- When this routine is called after MPI_FINALIZE routine.

<Return value>

None.

<Notes>

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

5.2.2 Sample Program

The sample program of the MPI statistical information section specifying interface is shown below. This program measures the MPI statistical information that is section specifying of the MPI_ALLTOALL and section specifying of the user routine including MPI_ALLGATHER routine.

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

int test_aa = 200;
int test_ag = 300;

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

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

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

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

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

    FJMPI_Collection_clear();

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

    MPI_Finalize();

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

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

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

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

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

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

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

    for(i = 0; i < comm_count ; i++) {
```

```
        MPI_Allgather(sendbuf, test_ag, MPI_INT,
                      recvbuf, test_ag, MPI_INT, comm);
    }

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

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

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

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

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

call MPI_BARRIER(MPI_COMM_WORLD,ierr)

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

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

5.3 Extended Persistent Communication Requests Interface

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

See table below for a list of the practical 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

Routine name	Routine overview
FJMPI_PREQUEST_STARTALL	Starts all communication using persistent an extended persistent communication requests interface

5.3.1 Overview

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

Table 5.4 The extended persistent communication requests interface mechanism

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

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

5.3.2 Extended Persistent Communication Requests Interface Specifications

5.3.2.1 FJMPI_PREQUEST_SEND_INIT

<Format>

C language format

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

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

<Notes>

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

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

5.3.2.2 FJMPI_PREQUEST_RECV_INIT

<Format>

C language format

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

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

<Notes>

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

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

5.3.2.3 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 are same as those of the MPI_START routine.

<Return value>

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

<Notes>

- Only communication requests that were created by FJMPI_PREQUEST_SEND_INIT routine or FJMPI_PREQUEST_RECV_INIT routine can be passed to this routine.
- When the request created with a FJMPI_PREQUEST_SEND_INIT routine is used, the procedure is non-local and waits calling FJMPI_PREQUEST_START or FJMPI_PREQUEST_STARTALL routine by the process which executes receive routine.

5.3.2.4 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 are same as those of the MPI_STARTALL routine.

<Return value>

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

<Notes>

- Only communication requests that were created by FJMPI_PREQUEST_SEND_INIT routine or FJMPI_PREQUEST_RECV_INIT routine can be passed to this routine.
- When the request created with a FJMPI_PREQUEST_SEND_INIT routine is included, the procedure of the request is non-local and wait calling FJMPI_PREQUEST_START or FJMPI_PREQUEST_STARTALL routine by the process which executes receive routine.

5.3.3 Sample Program

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

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

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

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

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

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

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

    MPI_Barrier(MPI_COMM_WORLD);

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

    MPI_Barrier(MPI_COMM_WORLD);
    stime = MPI_Wtime();

    FJMPI_Prequest_startall(2,&reqs[0]);

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

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

    etime = MPI_Wtime();

    MPI_Barrier(MPI_COMM_WORLD);

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

    MPI_Request_free(&reqs[0]);
    MPI_Request_free(&reqs[1]);
    MPI_Finalize();

    return (0);
}
```

5.4 MPI Asynchronous Communication Promotion Section Specifying Interface

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

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

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

Table 5.5 Overview of asynchronous communication promotion section specifying routines list

Routine name	Routine overview
FJMPI_PROGRESS_START	Starts the promotion of asynchronous communication

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

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

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

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

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

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

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

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

<Return value>

None.

5.4.2 Sample Program

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

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

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

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

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

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

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

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

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

        stime = MPI_Wtime();

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

        FJMPI_Progress_start();

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

```
        }

        FJMPI_Progress_stop();

        MPI_Waitall(4, reqs, MPI_STATUSES_IGNORE);

        etime = MPI_Wtime();

    }

    MPI_Finalize();

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

    return 0;
}
```

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-2018-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.

5.5.3 Overlap of Computation and Communication

Some communications invoked from persistent collective communication requests can be overlapped with computation using the feature of the Tofu interconnect. Otherwise if promotion of asynchronous communication using assistant cores is applied, communication invoked by the request can be overlapped with computation.

When overlap of computation and communication using the feature of the Tofu interconnect is applied, the communication starts immediately after MPI_START or MPI_STARTALL routines are called and can be overlapped with computation code until MPI_WAIT or MPI_WAITALL routines are called.

Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for the behavior when asynchronous communication promotion is applied.

5.5.3.1 Conditions for Applying Overlap of Computation and Communication

Communications invoked from a persistent collective communication request which meet all the following conditions can be overlapped with computation using the feature of the Tofu interconnect.

- The request is created from MPIX_ALLGATHER_INIT routine
- The start operation for the request is the second time or later
- The send/receive datatypes are basic datatypes
- The communicator is intra-communicator
- The number of processes per node (proc_per_node), the communicator size (comm_size) and the send message size (msg_size) satisfy the following inequality

```
comm_size-2+ceil(ceil(msg_size/(floor((15-proc_per_node)/15)+2))/16777215)*(comm_size-1) <= 2048
```

- The number of processes per node of the MPI job is not more than 30
- When the request is created, other requests which meet the conditions above do not exist in the process

These conditions may be changed in the future version.

5.5.3.2 Notes

When overlap of computation and communication using the feature of the Tofu interconnect is applied on persistent collective communication requests, monitoring communication buffer write damage using the MCA parameter mpi_check_buffer_write is not applied on the requests.

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.

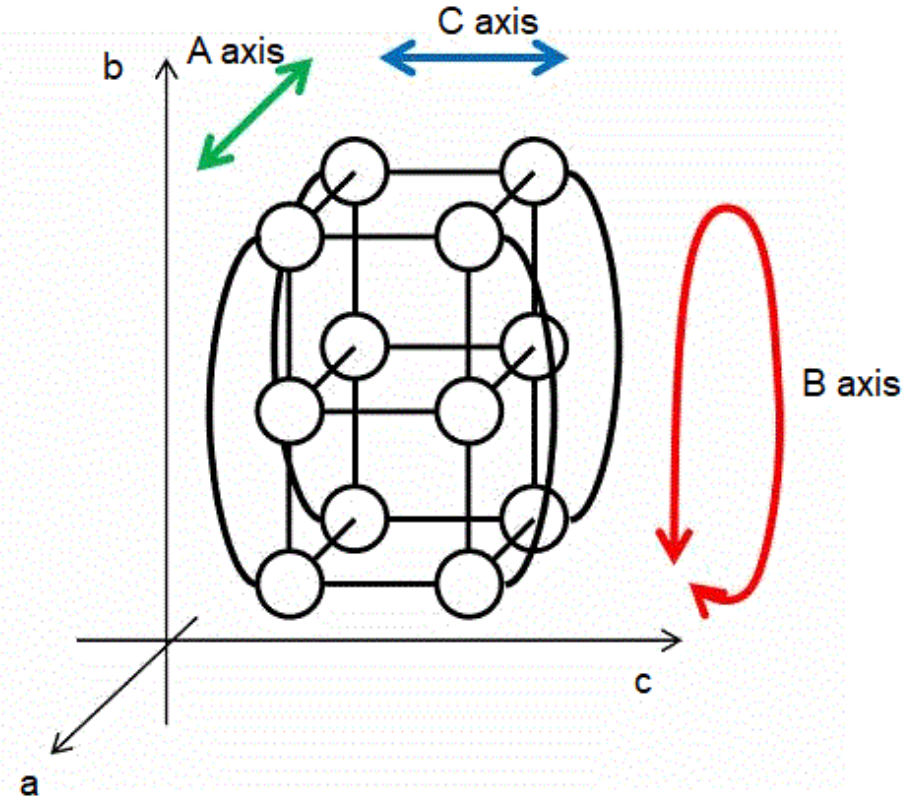
Chapter 6 Supplementary Items

6.1 Tofu Interconnect

6.1.1 Tofu Interconnect Configuration

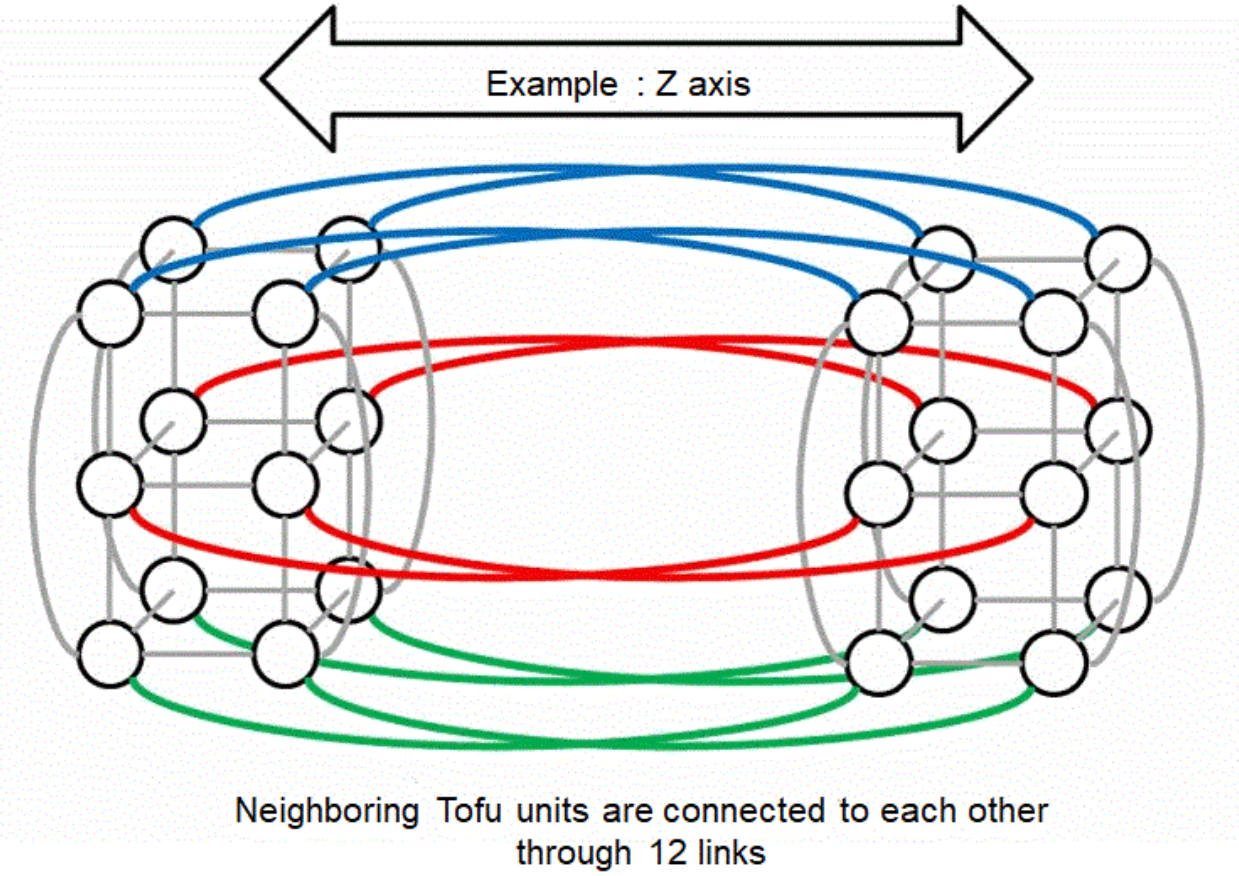
A Tofu interconnect is physically comprised of a 6-dimensional mesh/torus network. The coordinates of the 6-dimensional mesh/torus network are given by the six dimensions X, Y, Z, A, B, and C. The unit comprised of the A, B, and C axes of size 2*3*2 is known as a Tofu unit.

Figure 6.1 A Tofu unit



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

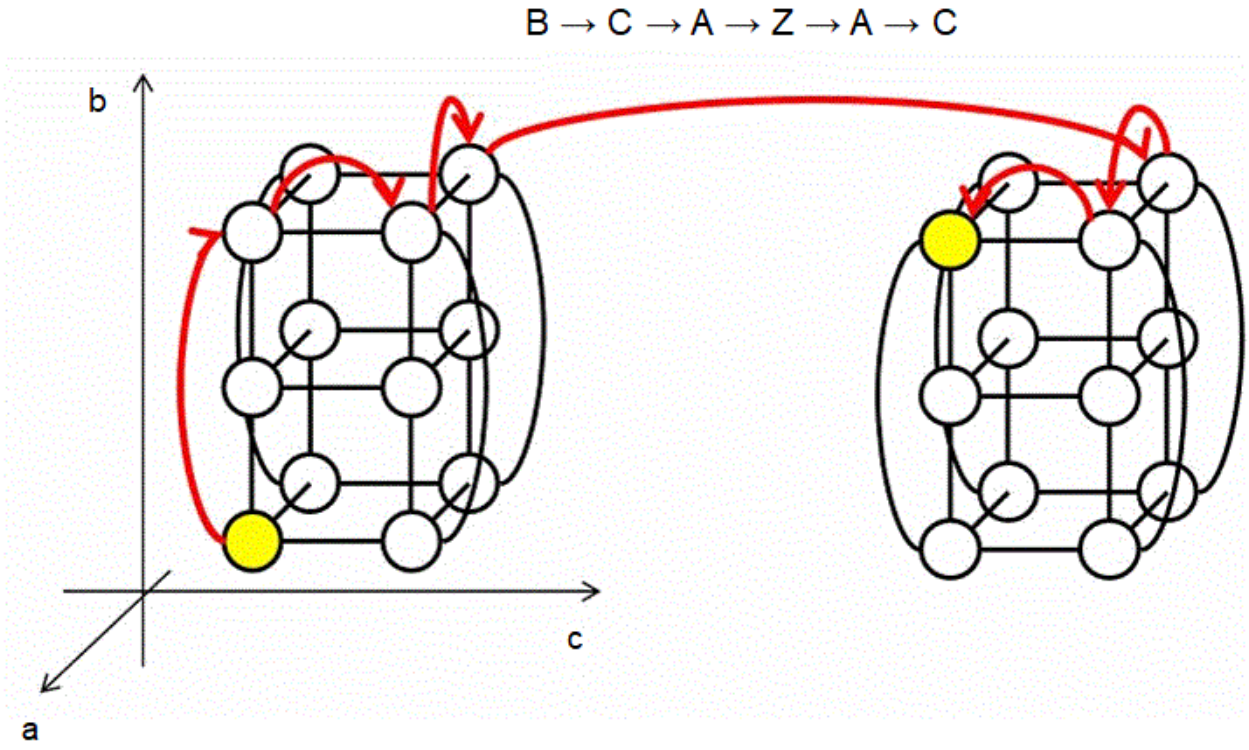
Figure 6.2 Connections to other Tofu units



6.1.2 Routing

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

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

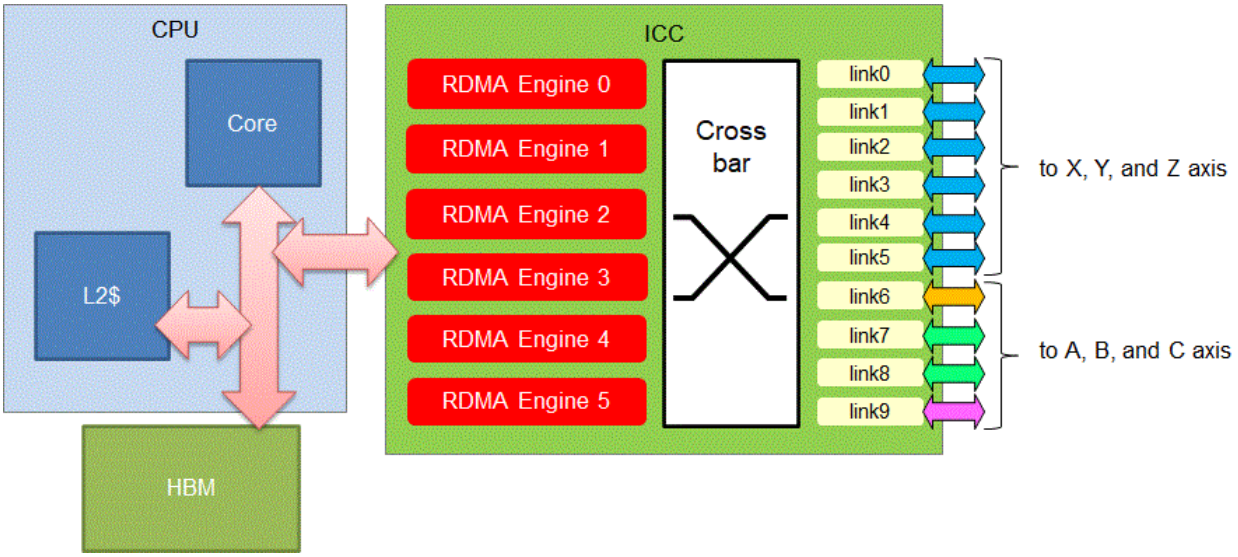


6.1.3 Configuration within a Node

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

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

Figure 6.4 ICC configuration



6.2 Promoting Asynchronous Communication Using an Assistant Core

On this software system, even if you use nonblocking communication with an expectation of overlap of computation and communication, the communication may not progress completely asynchronously behind the computation, and the transfer of the message body may begin when a completion routine such as MPI_WAIT routine is called. In this case, the expected overlap cannot be achieved. A64FX, the CPU of the this computing system, has 2 or 4 assistant cores that are dedicated to the OS and IO processing, in addition to 48 calculation cores that are dedicated to execution of user programs submitted as jobs. So this software system has a function to promote asynchronous communication in an MPI programs that use nonblocking communication, by using the assistant core. This function creates a thread called "MPI asynchronous processing progress thread" on the assistant core in order to progress nonblocking communication when an MPI program is executed. This enables asynchronous communication on the assistant core behind computation on the calculation cores. In other words, it encourages overlap of computation and communication, aiming for a reduced MPI program execution time. This effect tends to be larger when the message size to be sent by nonblocking communication is large, especially when the message size is equal to or more than the threshold value.

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

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

Table 6.1 Operation modes of the MPI asynchronous processing progress thread

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

Refer to "5.4 MPI Asynchronous Communication Promotion Section Specifying Interface" for details of the section specifying functions described above.

To use this function, the MCA parameter opal_progress_thread_mode must be specified. Refer to "Table 4.42 opal_progress_thread_mode (specifies the operation mode of the MPI asynchronous processing progress thread)" for details.

Though this function can be used in combination with the extended interface described in "5.3 Extended Persistent Communication Requests Interface", there will be almost no performance gain.

Example

Extract from a program where asynchronous communications are promoted

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

6.3 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 6.2 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 6.2 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 4.41 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

6.4 Notes Concerning MPI Standard Specifications

6.4.1 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 6.3 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 6.3 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

Routines
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
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

6.4.2 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 6.4 Predefined MPI datatypes usable by Fortran binding", "Table 6.5 Predefined MPI datatypes usable by C binding", "Table 6.6 Predefined MPI datatypes usable by C++ binding", and "Table 6.7 Predefined MPI datatypes usable by Java binding".

The valid datatypes in reduction operations for Java binding are shown in "Table 6.7 Predefined MPI datatypes usable by Java binding". Refer to the MPI standard for the datatypes that can be specified in reduction operations.

Table 6.4 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)

Predefined MPI datatypes	Fortran datatypes
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)
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

Predefined MPI datatypes	Fortran datatypes
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 6.5 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	

Predefined MPI datatypes	C datatypes
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
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 6.6 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)

Predefined MPI datatypes	C++ datatypes
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
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 6.7 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	

Predefined MPI datatypes	Java datatypes	Datatypes kind
	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	
MPI.INT2	int pair	
MPI.SHORT_INT	short and int pair	

6.4.3 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.

6.4.4 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

6.4.5 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.

6.4.6 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

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

6.4.7 One-sided Communications

This section provides notes on one-sided communications.

6.4.7.1 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 6.8 Assertions for optimizations in one-sided communications

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

6.4.7.2 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 6.9 Info key for MPI_WIN_CREATE, MPI_WIN_ALLOCATE, and MPI_WIN_CREATE_DYNAMIC

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

Table 6.10 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.

6.4.8 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.

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

6.4.8.1 info Argument Value

Specify MPI_INFO_NULL as the info input argument of the MPI_OPEN_PORT routine, the MPI_COMM_ACCEPT routine, the MPI_COMM_CONNECT routine, the MPI_PUBLISH_NAME routine, the MPI_UNPUBLISH_NAME routine, and the MPI_LOOKUP_NAME routine.

6.4.8.2 MPI_COMM_JOIN Return Value

The output of the MPI_COMM_JOIN routine is MPI_COMM_NULL.

6.4.8.3 Service Names in the MPI_PUBLISH_NAME

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

6.4.9 Dynamic Process Creation

This section provides notes on dynamic process creation.

6.4.9.1 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. When the vcoordfile info key or the num_nodes info key or the rank_map info key is specified to the array_of_info argument of MPI_COMM_SPAWN_MULTIPLE routine, set the key and value to the first element of the array_of_info argument.

Table 6.11 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.
vcoordfile	File path name	Specify the path of the VCOORD file which describes the logical coordinates of dynamic processes and number of CPUs (cores) allocated to each process etc.
num_nodes	String (Integer value of 1 or more)	Specify the number of nodes that are used for the dynamic process creation.
rank_map	String (bychip or bynode)	Specify the rule for the rank placement of the dynamic processes.
env	String (NAME=VALUE)	Specify the environment variables that are set to the dynamic processes.
fjprof_spawn_dir_name	Directory path name	Specify the output directory of the profiling data for the dynamic processes.
fjdbg_spawn_dir_name	Any string	Specify the identification name that is used for the output directory name of deadlock detection of profiler.

6.4.9.1.1 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.

6.4.9.1.2 Designation of Node and the Number of CPUs (cores) by vcoordfile Key

The vcoordfile key is used to specify the nodes of the dynamic processes and the number of CPUs (cores) allocated to each process. The absolute path or the relative path can be specified. If the specified value is the file name or the relative path, 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. Since the VCOORD file that is specified with vcoordfile key can describe the nodes and number of CPUs (cores) of each process, the user

can specify the node, the node shape, the number of processes per node, the number of CPUs (cores), and rank placement of dynamic processes using the VCOORD file that is appropriately described.

Refer to "4.5 VCOORD file format" for details on the VCOORD file.

Note

- In the following cases, the error class MPI_ERR_SPAWN is returned as the error code.
  - An invalid coordinate is specified in the VCOORD file.
  - The specified VCOORD file does not exist.
  - The number of processes which is specified with the argument maxprocs of the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is greater than the number of lines of the VCOORD file.
  - The number of processes to be created on a node exceeds the limit on the number of processes that can be created per node.
  - When only the number of CPUs (cores) that are assigned to each process is specified, the number of free nodes is too short.
  - The number of CPUs (cores) exceeds the number of CPUs (cores) on the node.
  - The format of the VCOORD file is invalid.
  - Process allocation to the CPU (core) cannot be implemented under the value of numanode_assign_policy.
- In the following case, FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES is returned as the error code. Refer to "6.4.9.5.1 Error Code for Identifying the Cause of Failure of Dynamic Process Creation" for details on the error code.
  - The specified coordinates in the VCOORD file are already used, and the free CPUs (cores) do not exist.

6.4.9.1.3 Designation of the Number of Nodes by num_nodes Key

The num_nodes key is used to specify the number of nodes for the dynamic process creation function. When this key is specified, the number of processes per node is calculated from the expression "maxprocs / num_nodes". The "maxprocs" means the number of the dynamic processes specified as the argument of the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine.

Note

Designation of the number of nodes with the num_nodes key should be used with care.

- If the vcoordfile key is specified simultaneously, the designation of the number of nodes with num_nodes key is disabled.
- The nodes for the dynamic processes are decided by Job Operation Software automatically. If it is necessary to allocate the node effectively in the node shape of job, use the vcoordfile key to specify the node in detail.
- If the number of nodes is invalid (for example, the number of nodes exceed the number of nodes which are allocated to the job), the error class MPI_ERR_SPAWN is returned as the error code.
- If the specified number of nodes is not allocated since the part of nodes which are allocated to the job are in use, the error code FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES is returned. Refer to "6.4.9.5.1 Error Code for Identifying the Cause of Failure of Dynamic Process Creation" for details on error code.

6.4.9.1.4 Designation of the Rule of Rank Placement of Processes by rank_map Key

The rank_map key is used to specify the rule of rank placement of the dynamic processes.

The values below can be specified for the rank_map key. The default value is bychip.

Table 6.12 Values that can be specified with rank_map key

Value	Explanation
bychip	The rank is set to CPU (core) on the same node first.

Value	Explanation
	When 8 is specified for the maxprocs argument and 2 is specified for the num_nodes argument, the rank is set as the figure below.
bynode	The rank is set to different nodes in order of node ID in round-robin fashion. When 8 is specified for the maxprocs argument and 2 is specified for the num_nodes argument, the rank is set as the figure below.

Figure 6.5 Rank placement when bychip is specified

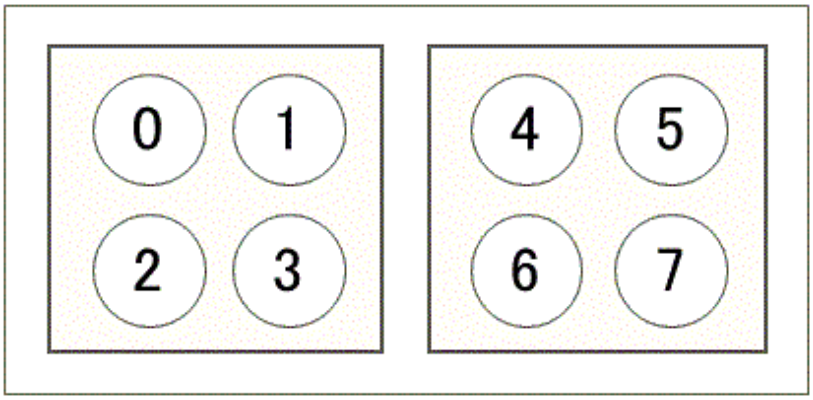
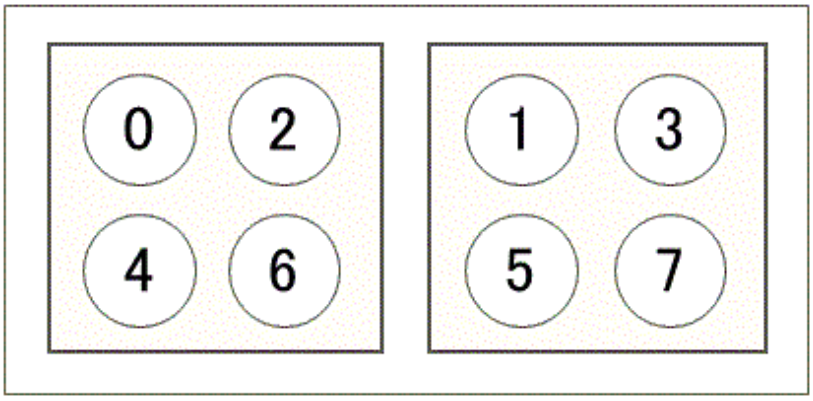


Figure 6.6 Rank placement when bynode is specified



Note

Designation of the rank placement with the rank_map key should be used with care.

- If the vcoordfile key is specified simultaneously, the designation of the rank placement with rank_map key is disable.
- If the num_nodes key is specified simultaneously, both designations are effective.

6.4.9.1.5 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.

6.4.9.1.6 Designation of Output of Profiling Data by fjprof_spawn_dir_name Key

The fjprof_spawn_dir_name key is set to specify the storage location of the result files for the profiler. The key-value is used as path name of the output directory. For more information about the profiler, refer to Profiler User's Guide.

6.4.9.1.7 Designation of Output of Result of Deadlock Detection Function by fjdbg_spawn_dir_name Key

The fjdbg_spawn_dir_name key is set to specify the storage location of the result files for the deadlock investigative function or abnormal termination investigative function in the debugging support functions. The key-value is used as a part of the output directory name. For more information about the deadlock investigative function and the abnormal termination investigative function, refer to Debugger for Parallel Applications User's Guide.

6.4.9.2 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.

6.4.9.3 MPI_UNIVERSE_SIZE

The value of MPI_UNIVERSE_SIZE which is defined as the predefined attribute of MPI_COMM_WORLD is calculated by the following formula.

```
"The number of nodes allocated to the job"
   * "The number of processes per node in the processes which are created by mpiexec command"
  or
"The number of nodes allocated to the job"
   * "The value specified with the -mpi max-proc-per-node option"
```

6.4.9.4 Designation of max-proc-per-node

If the number of processes per node of the dynamic processes is greater than the number of processes per node of the processes created by mpiexec command, specify the "--mpi max-proc-per-node" option of Job Operation Software necessarily. Specify the number of processes per node of dynamic process for the value of this option. Refer to "4.6 Execution of Multiple MPI Programs on the Same Node" for details.

6.4.9.5 Identification of the Cause of Failure of Dynamic Process Creation by the Error Code

In this software system, when the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine is failed, whether or not the cause of failure is shortage of free node can be identified by the error code. Using this error code, MPI_COMM_GET_ERRHANDLER routine, and MPI_COMM_SET_ERRHANDLER routine, the user can select whether to continue the processing of the program when the dynamic process creation is failed.

6.4.9.5.1 Error Code for Identifying the Cause of Failure of Dynamic Process Creation

The original error code with this software system is shown below.

Table 6.13 Original error code for dynamic process creation

Error code	Error class	Explanation
FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES	MPI_ERR_SPAWN	Dynamic process creation failed because of the shortage of free nodes.

6.4.9.5.2 Usage Example

The overview of the program which executes the dynamic process creation as soon as the node is freed using the error code FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES is shown below.

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

    MPI_Comm_get_errhandler(...); /* Get the original error handler */
    MPI_Comm_set_errhandler(...); /* Switch the error handler */
    while (1) {
        while (1) {
            ret = MPI_Comm_spawn(..., &inter_comm, ...);
            if (MPI_SUCCESS == ret) {
                break;
            } else if (FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES != ret) {
                /* End abnormally when the program for a reason other than
                   the shortage of free node */
                MPI_Abort(...);
            }
        }
        MPI_Comm_disconnect(&inter_comm);
        if (...) {
            break;
        }
    }
    MPI_Comm_set_errhandler(...); /* Set the original error handler */
```

6.4.9.6 Notes

- Do not use the dynamic process creation function of this software system when the following conditions apply:
  - The total invocation count for the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine exceeds 4294967295 for one execution of mpiexec command when dynamic process creation is used.

    Job Operation Software outputs an error message and then the MPI program ends abnormally.
  - The number of MPI_COMM_WORLD for the dynamic processes that exists at the same time exceeds 65535.

    Job Operation Software outputs an error message and then the MPI program ends abnormally.
  - Dynamic process creation executes something other than an MPI program.

    The behavior is not guaranteed if something other than an MPI program is executed for the dynamic process creation.

  Refer to the Job Operation Software manual for details on the Job Operation Software.
- If the same process repeatedly executes dynamic process creation, the created process information and communicator information accumulates in memory and may cause memory shortages. Use with care.

6.4.9.7 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 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();
    }
}
```

6.4.10 Rank Changes in Accordance with Cartesian Topology

When the MPI_CART_CREATE routine is used to create a new communicator with attached Cartesian topology, new ranks can be allocated in the order of the Cartesian coordinates, to the parallel processes belonging to the created communicator by specifying true in the reorder argument. In other words, ranks can be changed on the basis of the Cartesian coordinates.

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

6.4.10.1 Conditions Enabling Rank Changes

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

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

The MPI_CART_CREATE routine changes ranks if all the following conditions are met:

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

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

6.4.10.2 Rules for Rank Changes

When a Cartesian topology is created by the MPI_CART_CREATE routine, the Cartesian coordinates allocate ranks in ascending order starting from 0, based on the logical coordinates of the process shape in accordance with the sequence below, to the parallel processes belonging to input communicator comm_old.

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

6.4.10.3 Checking Rank Changes

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

6.4.10.4 Sample Program

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

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

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

#define FAILURE 1

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

```
    char host[255];

    gethostname(host, 255);

    MPI_Init(&argc, &argv);

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

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

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

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

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

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

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

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

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

    MPI_Finalize();

    return 0;

}
```

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

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

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

6.4.11 Notes on Send Buffer and Receive Buffer

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

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

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

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 6.14 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 6.15 Info object keys and values that can be used with FEFS

Key	Default value	Meaning
direct_read	"false"	Specify whether or not to perform direct I-O (read). Specify either "true" or "TRUE" to use direct I-O (read).
direct_write	"false"	Specify whether or not to perform direct I-O (write).

Key	Default value	Meaning
		Specify either "true" or "TRUE" to use direct I-O (write).
striping_unit	New directory settings value	Width of one stripe
striping_factor	New directory settings value	Number of input-output devices that perform file striping

Note

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

6.4.13 Use of the Profiling Interface

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

6.4.14 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.

6.4.15 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

6.4.16 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.

6.4.17 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.

6.5 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. More precisely, the distance (number of hops) of the message communication is also taken into account, not just the message size. The compute nodes in this computing system are physically connected by a mesh or torus format, and message communication between any two compute nodes is performed via a number of other compute nodes, as required. In this software system, the "threshold value" (number of bytes) for switching between Eager communication and Rendezvous communication in the fast communication mode is obtained using the following formula:

```
"Threshold value" = 45,056 + "number of hops" * 296
```

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

In the following cases, the Rendezvous protocol 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_tofu_eager_limit can be used to change the "threshold value". Refer to "Table 4.6 btl_tofu_eager_limit (changes the threshold value for switching the communication method)" for information on the MCA parameter btl_tofu_eager_limit.

6.6 Stride RDMA Communication

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

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

- The datatypes on the sender and the receiver are equivalent
- The message is non-contiguous in memory
- The communication is performed between different compute nodes
- Total size (in bytes) of messages being sent is the same or greater than the "threshold value" for switching the protocol

Refer to "6.5 Eager Protocol and Rendezvous Protocol" for information on the "threshold value" for switching the protocol.

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

Example

Extract from a program where Stride RDMA communication is applied

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

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

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

6.6.1 Notes on Stride RDMA Communication

A program may terminate abnormally after outputting the following error message in Stride RDMA communication if memory regions that are not guaranteed to be within the same sequential storage are used for communication using a derived datatype.

- An error message containing the string "[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag.".

In this case, disable Stride RDMA communication by specifying the value 0 for the MCA parameter pml_ob1_use_stride_rdma. However, note that the communication performance may decrease.

Refer to "Table 4.47 pml_ob1_use_stride_rdma (use of Stride RDMA communication)" for information on the MCA parameter pml_ob1_use_stride_rdma.

6.7 Using Multiple TNIs

This computing system is connected by means of Tofu interconnects. A network interface device, known as a TNI (Tofu Network Interface), is deployed at each compute node. This computing system has six TNIs at each compute node. The MPI programs allocated to each node use these TNIs, thus enabling highly efficient message send-receive. For MPI programs that have a small number of simultaneous communications, the communication time for small-sized messages can be shortened by reducing the number of TNIs. The upper limit for the number of TNIs to be used can be changed using the MCA parameter common_tofu_max_tnis. Refer to "Table 4.21 common_tofu_max_tnis (changes the upper limit for the number of TNIs to be used)" for information on the MCA parameter common_tofu_max_tnis.

Using multiple TNIs can improve the throughput performance of point-to-point communication by providing more opportunities to use multiple communication paths, even for large messages. The use of multiple communication paths makes it possible to split the message, thus increasing the potential for efficient communication. This is referred to as trunking. Trunking can be enabled by specifying a value of 1 in the MCA parameter common_tofu_use_multi_path. Refer to "Table 4.30 common_tofu_use_multi_path (performs point-to-point communication using multiple communication paths)" for information on using the MCA parameter common_tofu_use_multi_path.

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

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

6.8 Reduction Operation Sequence Guarantee in Collective Communication

This 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 4.7 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.

6.9 Process Creation from Inside an MPI Program

In this software system, the following restriction applies to MPI programs if they create a child process using a system call

(for example, fork), library function (for example, system), service routine provided by a Fortran system (for example,

FORK), or the like.

- If there is a memory region which is in use for an MPI communication buffer at the time of process creation, pages containing the memory region may not be inherited to the child process and the child process may not be able to access the pages.

"In use" periods are defined by the following rules.

- For a send/receive buffer of a blocking operation of point-to-point communication, collective communication, or neighborhood collective communication, a period starts when a communication routine is called and ends when the routine returns.
- For a send/receive buffer of a nonblocking operation of point-to-point communication, collective communication, or neighborhood collective communication, a period starts when a communication is initiated by the MPI_ISEND routine or the like and ends when the communication is completed by the MPI_WAIT routine or the like.
- For a send/receive buffer of a persistent request operation of point-to-point communication, collective communication, or neighborhood collective communication, a period starts when a communication request is created by the MPI_SEND_INIT routine or the like and ends when the request is freed by the MPI_REQUEST_FREE routine. Before freeing the request, the corresponding communication must have been completed by the MPI_WAIT routine or the like.
- For a window of one-sided communication created by a routine other than the MPI_WIN_CREATE_DYNAMIC routine, a period starts when the window is created by the MPI_WIN_CREATE routine or the like and ends when the window is freed by the MPI_WIN_FREE routine.
- For a memory region attached by the MPI_WIN_ATTACH routine, a period starts when the memory region is attached by the MPI_WIN_ATTACH routine and ends when the memory region is detached by the MPI_WIN_DETACH routine or the corresponding window is freed by the MPI_WIN_FREE routine.
- For an origin buffer of RMA communication of one-sided communication, a period starts when an RMA communication is started and ends when the communication is completed locally by the MPI_WIN_FENCE, MPI_WIN_FLUSH, or MPI_WAIT routines, or the like.
- For a memory allocated by the MPI_ALLOC_MEM routine, a periods starts when the MPI_ALLOC_MEM routine is called and ends when the corresponding MPI_FREE_MEM routine is called.

For example, suppose that a child process created by the fork system call accesses such a memory region before the process calls the execve system call or _exit system call. Then, due to this restriction, a segmentation fault error may occur in the process.

Whether accessible or not is decided for each page managed by the OS. Therefore, if a memory region is in use, the neighboring memory regions may also not be able to be accessed.

The memory region of a created child process may be allocated on the same page as the memory region used for a communication buffer within the MPI library. In this case, a segmentation fault error may occur in the child process even if it is not judged to be "in use" by the above criteria. To make this less likely to occur, specify 1 for the MCA parameter common_tofu_use_memory_pool. Note that this option may increase memory usage. See "Table 4.31 common_tofu_use_memory_pool (uses memory pool for the MPI library)" for information on the MCA parameter common_tofu_use_memory_pool.

6.10 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.

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

In order to save memory usage as much as possible with suppressing degradation of overall communication performance, this software system provides two communication modes: the fast communication mode and the memory-saving communication mode. In this software system, these two communication modes are internally distinguished for each peer process.

The fast communication mode uses a comparatively large receive buffer called the Large receive buffer and an additional receive buffer called the Small receive buffer for each peer process. Using these receive buffers, communications are performed as fast as possible.

In contrast, a communication with the memory-saving communication mode is performed so that memory usage is saved as much as possible. There are two methods for the memory-saving communication mode. The first one is the method which uses only a comparatively small receive buffer called the Medium receive buffer. A process has only one Medium receive buffer for each peer process. This method can save memory usage, although communication performance is somewhat worse. The second one is the method which uses only a receive buffer called the Shared receive buffer. A process has only one Shared receive buffer and the buffer is used in common by all peer processes which communicate with the process using the memory-saving communication mode. Thus, the total memory usage for receive buffers is constant when this method is used. Therefore, this method can save memory usage more than the first method of the memory-saving communication mode when there are especially many peer processes which a process actually communicate with. However, note that communication performance is much worse. The method which is used for the memory-saving communication mode can be chosen using the MCA parameter common_tofu_memory_saving_method. Refer to "Table 4.25 common_tofu_memory_saving_method (changes the method used for the memory-saving communication mode)" for details of this MCA parameter. In order to balance communication performance and memory usage, it is important to use these communication modes appropriately. For example, there is a usage that the fast communication mode is used for a peer process which a process frequently communicates with and the memory-saving communication mode is used for a peer process which a process less frequently communicates with.

Peer processes which a process communicate with using the fast communication mode are decided at the time of execution depending on the communication pattern in an MPI program. Normally, the memory-saving communication mode is used for the first time communication even if a process communicate with any peer process. If the number of communications with a peer process reaches a threshold value, the fast communication mode is used for communications with the peer process after that. The upper limit of the number of peer processes which a process can communicate with using the fast communication mode can be set by a user. If the number of processes which a process communicates with reaches this upper limit, communication mode on this process will not be switched any more.

Normally, 256 is set as the upper limit for the number of processes that use the fast communication mode. This upper limit can be changed using the MCA parameter common_tofu_max_fastmode_procs. Depending on this MCA parameter setting, all communications can be performed in the fast communication mode. Alternatively, all communications can also be performed in the memory-saving communication mode. Refer to "Table 4.20 common_tofu_max_fastmode_procs (changes the upper limit for the number of processes that can communicate in fast communication mode)" for information on the MCA parameter common_tofu_max_fastmode_procs.

Normally, 16 is set as the communication count standard value that is the condition for switching from the memory-saving communication mode to the fast communication mode. This standard value can be changed using the MCA parameter common_tofu_fastmode_threshold. However, this value includes the count of control communications performed in the MPI library. Therefore this switching may occur before calling the MPI routines fewer times than the specified value. Refer to "Table 4.18 common_tofu_fastmode_threshold (changes the conditions for switching to fast communication mode)" for information on the MCA parameter common_tofu_fastmode_threshold.

The sizes of the Large receive buffer, the Medium receive buffer, and the Shared receive buffer can be changed using the MCA parameters common_tofu_large_recv_buf_size, common_tofu_medium_recv_buf_size, and common_tofu_shared_recv_buf_size, respectively. Refer to "Table 4.19 common_tofu_large_recv_buf_size (changes the size of the Large receive buffer)", "Table 4.22 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)", and "Table 4.29 common_tofu_shared_recv_buf_size (changes the size of the Shared receive buffer)" for details of these three MCA parameters.

If there are a large number of parallel processes, it is important to use these MCA parameters to tune the system according to the amount of memory required for the MPI program itself and the desired communication performance. Refer to "6.11 Memory Usage Estimation Formulae and Tuning Guidelines" for details.

6.10.2 Influence of Dynamic Connection on Performance

The dynamic connection built in this 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.

If the number of peer processes under connection establishment exceeds 130, communication performance may decrease due to frequent retries of the communication processing. In that case, divide calls of the MPI_WAITALL routines, etc. so that the number of the first time communication requests to each process issued simultaneously is less than or equal to 130.

6.11 Memory Usage Estimation Formulae and Tuning Guidelines

Each job has a limit in size of usable memory on each compute node, and the size of memory which an MPI program including MPI library uses on each node cannot exceed the limit. In case an MPI program is not likely to be executed under the memory limitation, program tuning is required from the efficiency viewpoint on memory usage.

This section describes memory usage estimation formulae, tuning guidelines, and the specification of restriction values.

6.11.1 Memory Usage Estimation Formulae

The MPI library memory usage for a specific MPI process can be estimated using the formula shown in "Figure 6.7 Estimation formula". However, note the following:

- The estimation formula shown in "Figure 6.7 Estimation formula" can be used only for when using large pages. Refer to "Job Operation Software End-user's Guide for HPC Extensions" for information on large pages.
- Because the memory management of operating system is done per page, more memory than the estimated result may be required depending on page size.
- Due to fluctuations in memory usage by factors other than the MPI library, the difference between the result estimated by the formula shown in "Figure 6.7 Estimation formula" and the actual result measured in an MPI job execution may be large.
- Use the estimation formula shown in the figure below if the number of parallel processes in the MPI program is about 200 or more. If the number of parallel processes is less than this, note that there will be large errors in the calculated results, so they do not make good references. Also note that the values obtained are not necessarily accurate even if the number of parallel processes is over 200.
- The estimation formula shown in the figure below cannot be used if the MPI program uses one-sided communication, if dynamic process creation is being used, or if communication is established between MPI process groups that do not share a communicator.
- The estimation formula shown in the figure below also cannot be used if the MCA parameter common_tofu_use_memory_pool is set to 1. Refer to "Table 4.31 common_tofu_use_memory_pool (uses memory pool for the MPI library)" for information on the MCA parameter common_tofu_use_memory_pool.
- The difference between the result estimated by the formula shown in "Figure 6.7 Estimation formula" and the actual memory usage may be large when the value 2 or 3 is specified for the MCA parameter common_tofu_conv_dim. Refer to "Table 4.16 common_tofu_conv_dim (executes an MPI program after converting the dimensions of the coordinate of each process to higher dimensions)" for information on the MCA parameter common_tofu_conv_dim.
- 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. For example, if the number of compute nodes which are used in a job execution is 25600 and the number of MPI processes per compute node is 48, the memory usage immediately after the MPI_INIT routine is called will reach 90% of the memory on the compute node and it is likely that the application will not work.

"Table 6.16 Variables in memory usage estimation formulae" shows the meanings of the variables in this estimation formula.

Figure 6.7 Estimation formula

$$
\begin{aligned}
& C_{\mathrm{Proc}} \times N_{\mathrm{Proc}} + C_{\mathrm{Base}} + B_{\mathrm{Shared}} \times K_{\mathrm{Buffer}} \\
& + (B_{\mathrm{Large}} + B_{\mathrm{Small}}) \times N_{\mathrm{FastPeer}} \times K_{\mathrm{Buffer}} \\
& + B_{\mathrm{Medium}} \times (N_{\mathrm{Peer}} - N_{\mathrm{FastPeer}}) \times K_{\mathrm{Buffer}} \\
& + C_{\mathrm{Peer}} \times N_{\mathrm{Peer}} \\
& + B_{\mathrm{Loopback}} \times N_{\mathrm{Loopback}} \\
& + \sum_{\mathrm{communicator}} (C_{\mathrm{Member}} \times N_{\mathrm{Member}} + C_{\mathrm{PeerMember}} \times N_{\mathrm{PeerMember}}) \\
& + B_{\mathrm{UnexpectedMessage}}
\end{aligned}
$$

The first line in the estimation formula shown in the Figure above indicates the minimum amount of memory consumed by any MPI program. This is the amount of memory secured when execution of the MPI program is started and the MPI_INIT routine is called. This value depends on the total number of processes.

The value in the second line through the fifth line in the estimation formula shown in the Figure above changes depending on the total number of communication partner processes, the number of processes communicating in the fast communication mode, and whether the process has MPI communications with itself or not. This value increases at the point when communication with a new communication partner process is first performed.

The sixth line in the estimation formula shown in the figure above indicates the sum of the memory amounts consumed by each communicator. These communicators include MPI_COMM_WORLD. The value increases at the point when each communicator is created and when a communication is issued by that communicator.

The seventh line in the estimation formula shown in the Figure above becomes larger when an unexpected message is issued. An unexpected message is a message for which invocation of a receive routine, such as the MPI_RECV routine, in response to a send routine, such as the MPI_SEND routine, is delayed. The receive side process uses memory to temporarily save the received message.

Table 6.16 Variables in memory usage estimation formulae

Variable	Meaning of variable	Explanation of variable	Value
N_{Proc}	Total number of processes	The number of processes belonging to the communicator MPI_COMM_WORLD including that process.	-
N_{Peer}	Number of communication partner processes	The number of communication partner processes of that MPI process. The value is 0 immediately after the MPI_INIT routine is called, but the value increases at the point of the first communication with a new communication partner process. This value includes processes performed by communication internally in the MPI library during collective communication or similar, not just the number of processes performed	-

Variable	Meaning of variable	Explanation of variable	Value
		by point-to-point communication coded in the MPI program.	
$N_{FastPeer}$	Number of communication partner processes using fast communication mode	Out of the number of communication partner processes, the number of processes that use the fast communication mode. If the memory-saving communication mode is not used, this value is the same as the number of communication partner processes. The value specified in the MCA parameter common_tofu_max_fastmode_procs becomes the upper limit value.	-
$N_{Loopback}$	Whether the process has MPI communications with itself or not	The value is 1 if the process has MPI communication with itself. The value is 0 immediately after the MPI_INIT routine is called, but the value becomes 1 at the point of the first communication with itself.	-
N_{Member}	Number of processes belonging to that communicator	The number of processes varies for different communicators.	-
$N_{PeerMember}$	Number of communication partner processes in that communicator	Number of communication partner processes that communicate using that communicator. The value varies for different communicators. The value is 0 immediately after the communicator is created, but the value increases at the point of the first communication with a new communication partner process. This value includes processes performed by communication internally in the MPI library during collective communication or similar, not just the number of processes performed by point-to-point communication coded in the MPI program. If multiple communicators are used for communication with a particular MPI process, the values for those communicators are added.	-
B_{Large}	Size of Large receive buffer	The value specified in the MCA parameter common_tofu_large_recv_buf_size	Default value: 1MiB
B_{Small}	Size of Small receive buffer	A constant	64KiB
B_{Medium}	Size of Medium receive buffer	The value specified in the MCA parameter common_tofu_medium_recv_buf_size	Default value: 2KiB Note that if the value 2 is specified for the MCA parameter common_tofu_memory_saving_method, the value of B_{Medium} is 0B regardless of the value specified for the MCA parameter common_tofu_medium_recv_buf_size. Refer to "Table 4.25 common_tofu_memory_saving_method (changes the method used for the memory-

Variable	Meaning of variable	Explanation of variable	Value
			saving communication mode)" for details of the MCA parameter common_tofu_memory_saving_method.
B_{Shared}	Size of Shared receive buffer	The value specified in the MCA parameter common_tofu_shared_recv_buf_size	Default value: 16MiB Note that if the value 1 is specified for the MCA parameter common_tofu_memory_saving_method, the value of B_{Shared} is 0B regardless of the value specified for the MCA parameter common_tofu_shared_recv_buf_size. Refer to "Table 4.25 common_tofu_memory_saving_method (changes the method used for the memory-saving communication mode)" for details of the MCA parameter common_tofu_memory_saving_method.
$B_{Loopback}$	Size of buffer for communication with the process itself	A constant	4MiB
$B_{UnexpectedMessage}$	Quantity of unexpected messages	Increases with each unexpected message that is stored	-
K_{Buffer}	Efficiency of memory allocation	A constant	1.0
C_{Base}	Coefficient	A constant determined in accordance with the number of processes per node (referred to as "ppn" in the next column)	1-4ppn: 89MiB 5ppn: 67MiB 6-9ppn: 45MiB 10-15ppn: 30MiB 16-48ppn: 20MiB
C_{Proc}	Coefficient	A constant determined in accordance with the number of processes per node (referred to as "ppn" in the next column)	1ppn: 1702B 2ppn: 941B 3ppn: 841B 4ppn: 740B 5ppn: 673B 6-15ppn: 606B 16-48ppn: 514B
C_{Peer}	Coefficient	A constant	1952B
C_{Member}	Coefficient	A constant determined in accordance with the number of processes per node (referred to as "ppn" in the next column)	1ppn: 408B 2ppn: 184B 3-4ppn: 144B 5-8ppn: 80B 9-16ppn: 56B 17-48ppn: 32B
$C_{PeerMember}$	Coefficient	A constant.	0

6.11.2 Memory Usage Tuning Guidelines

Tuning considerations vary for different MPI program patterns. Table below shows the tuning issues to be considered for the various patterns.

Table 6.17 Tuning guidelines

Pattern	Tuning considerations
The number of processes which require communication performance is fewer compared with the number of all communication partner processes	Use the MCA parameter common_tofu_max_fastmode_procs to set the upper limit for the number of processes that use the fast communication mode for communications. However, consider the performance of communication with the communication partners (processes) that are unable to switch to fast communication mode after the upper limit for the number of processes that can use the fast communication mode for communication has been reached. If the processes to which the fast communication mode and the memory-saving communication mode are assigned are not as anticipated, use the MCA parameter common_tofu_fastmode_threshold to adjust the number at which communication switches from the memory-saving communication mode to the fast communication mode.
Almost all communication partner processes require equal communication performance	Use the MCA parameter common_tofu_large_recv_buf_size to adjust the size of the Large receive buffer for the fast communication mode. However, note that communication performance may deteriorate uniformly if the data size ranges from several KiB to several 10s of KiBs.
If the above measures are insufficient	In addition to tuning using the MCA parameters common_tofu_max_fastmode_procs and common_tofu_large_recv_buf_size, use the MCA parameter common_tofu_medium_recv_buf_size to adjust the size of the Medium receive buffer for the memory-saving communication mode. If these measures are also insufficient, specify the value 2 for the MCA parameter common_tofu_memory_saving_method and adjust the size of the Shared receive buffer using the MCA parameter common_tofu_shared_recv_buf_size.

6.11.3 Specifying Memory Allocation Restriction Values

As described in "6.11.2 Memory Usage Tuning Guidelines", when using the memory-saving communication mode, the upper limit for the number of communication partner processes using the fast communication mode must be specified in MCA parameter common_tofu_max_fastmode_procs. However, just specifying this upper limit may not be sufficient to achieve both performance and memory allocation. If not, the reception buffer size must also be specified.

Using a different approach, if the user knows the amount of memory used by the MPI program itself, and if the MPI library can operate in the remaining memory range, the user need not calculate a value for the above MCA parameter. In practice, the memory allocation that the MPI library is allowed to use can be specified. If the user specifies this limit, this software system automatically tunes the MCA parameters internally and, as much as possible, operates within the specified range of restriction values for memory allocations. The actual memory allocation when an MPI program is executed will vary depending on the MPI routines used within the MPI program, the number of parallels, the execution method, and so on. Operation is not necessarily possible within the specified range of restriction values for memory allocations. Refer to the notes in "6.11.3.3 Notes on Execution When Memory Allocation Restriction Values are Specified" that apply to your usage circumstances.

Note that memory allocation restriction values cannot be specified if dynamic process creation is used or if communication is being established between MPI process groups that do not share a communicator. If specified, the behavior is unpredictable.

6.11.3.1 Specification Memory Allocation Restriction Values

In practice, memory allocation restrictions are enabled if at least one value is specified in MCA parameter common_tofu_memory_limit. The value specified in this MCA parameter is interpreted as being the restriction value (MiB) for the memory allocation that can be used by the MPI library, and other MCA parameters are tuned automatically. At this time, the value specified in MCA parameter common_tofu_memory_limit_peers is used as the number of communication partner processes. If the MCA parameter common_tofu_memory_limit_peers is not specified, the number of processes belonging to the same communicator

MPI_COMM_WORLD is used as the number of communication partner processes. Refer to "Table 4.23 common_tofu_memory_limit (specifies the memory allocation limit value)" and "Table 4.24 common_tofu_memory_limit_peers (specifies the assumed number of communication partner processes when the memory allocation is limited)" for information on the MCA parameters common_tofu_memory_limit and common_tofu_memory_limit_peers respectively.

Note that if the MPI library for debug is being used, operations do not necessarily conform to the specified values and, for example, more memory than the specified restriction values for memory allocations may be used.

6.11.3.2 MCA Parameters Targeted by Automatic Tuning

Table below shows the MCA parameters targeted for automatic tuning.

Table 6.18 MCA parameters tuned automatically in accordance with memory allocation restriction values

Priority	MCA parameter	Description
1	common_tofu_max_fastmode_procs	Upper limit for the number of communication partner processes that use the fast communication mode
2	common_tofu_large_recv_buf_size	Large receive buffer size
3	common_tofu_medium_recv_buf_size	Medium receive buffer size. This parameter is enabled if the value of the MCA parameter common_tofu_memory_saving_method is 1.
	common_tofu_shared_recv_buf_size	Shared receive buffer size. This parameter is enabled if the value of the MCA parameter common_tofu_memory_saving_method is 2.

Note: Smaller priority values indicate a higher priority.

If automatic tuning of specified memory allocations can be achieved just by using the highest priority MCA parameter, tuning is not required for the lower priority MCA parameters and, therefore, the default values remain unchanged.

Regardless of whether a value of 1 or higher is specified in MCA parameter common_tofu_memory_limit, if two or fewer of the MCA parameters shown in the table above are specified simultaneously, the specified values are enabled as is for the specified MCA parameters. Then, the remaining unspecified MCA parameters in the table above are the only parameters tuned automatically, in order of highest priority (smallest numerical value).

If the three MCA parameters shown in the table above and the MCA parameter common_tofu_memory_limit are specified simultaneously, automatic tuning is not performed.

Assume, for example, that the maximum number of partner processes that one process communicates with is known from MPI statistical information, and that the MCA parameters common_tofu_memory_limit, common_tofu_memory_limit_peers, and common_tofu_max_fastmode_procs are specified. In this case, the MCA parameter common_tofu_large_recv_buf_size value is tuned automatically first. If this is insufficient, the MCA parameter common_tofu_medium_recv_buf_size is then also tuned automatically. If the value of the MCA parameter common_tofu_memory_saving_method is 2, the MCA parameter common_tofu_shared_recv_buf_size is tuned automatically instead of the common_tofu_medium_recv_buf_size.

6.11.3.3 Notes on Execution When Memory Allocation Restriction Values are Specified

Automatic tuning is performed by calculating back from the estimation expressions in "Figure 6.7 Estimation formula". Therefore, note the following when executing MPI programs in which memory allocation restriction values are specified.

The minimum memory allocation required by this software system varies in accordance with the number of parallel processes and other conditions. Therefore, the memory allocation restriction values specified by the user may be exceeded.

For example, if unexpected messages are issued, these are saved within the MPI library, and may therefore affect the amount of memory used. MPI library automatic tuning does not consider the likelihood of unexpected messages because the number of unexpected messages issued during execution of an MPI program cannot be known. This means that the memory allocation restriction values specified by the user may be exceeded for MPI programs that issue a large number of unexpected messages.

Additional memory is also used if automatic process creation is performed, if communicators are created, and so on. These allocations are also excluded from the calculations performed internally during automatic tuning because this software system cannot know these memory allocations in advance. Note that in these cases too, the memory allocation restriction values specified by the user may be exceeded.

6.12 Use of Tofu Barrier Communication for Better Performance

When the routines MPI_BARRIER, MPI_BCAST, MPI_REDUCE, and MPI_ALLREDUCE are executed, communication performance can be improved by using Tofu barrier communication provided by this software system as a Tofu interconnect hardware function.

This section describes the conditions that Tofu barrier communication is applied for each of the MPI routines, and gives notes about this communication.

6.12.1 MPI_BARRIER

The MPI_BARRIER routine can apply the Tofu barrier communication function if all the following conditions are met:

- The value ^tbi is not specified for the MCA parameter coll.
- The communicator is a communicator in a group (intra-communicator) and is not created using the MPI_INTERCOMM_MERGE routine.
- Any of the following conditions a. or b. is met.
    a. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 2, and the number of compute nodes allocated to the communicator is greater than or equal to 4.
    b. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 3 or 4, and any of the following conditions 1. or 2. is met.
        1. The number of processes in the communicator is greater than or equal to 4. In addition, if the number of compute nodes allocated to the communicator is less than or equal to 3, the number of processes per compute node is greater than or equal to 2.
        2. Both the number of processes in the communicator and the number of compute nodes allocated to the communicator are greater than or equal to 4.
- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.

6.12.2 MPI_BCAST

The MPI_BCAST routine can apply the Tofu barrier communication function if all the following conditions are met:

- The value ^tbi is not specified for the MCA parameter coll.
- MCA parameter coll_tbi_use_on_bcast is not set to 0.
- The communicator is a communicator in a group (intra-communicator) and is not created using the MPI_INTERCOMM_MERGE routine.
- Any of the following conditions a. or b. is met.
    a. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 2, and the number of compute nodes allocated to the communicator is greater than or equal to 4.
    b. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 3 or 4, and any of the following conditions 1. or 2. is met.
        1. The number of processes in the communicator is greater than or equal to 4.In addition, if the number of compute nodes allocated to the communicator is less than or equal to 3, the number of processes per compute node is greater than or equal to 2.
        2. Both the number of processes in the communicator and the number of compute nodes allocated to the communicator are greater than or equal to 4.
- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.

- The combination of the datatype, the size of each element, and the maximum number of elements in a message is the one of the ways shown in "Table 6.19 Combinations that allow the MPI_BCAST routine to apply Tofu barrier communication".
  - When using a datatype other than the basic datatype, the number of elements in a message is within a range not exceeding the maximum number of elements in a message described in "Table 6.19 Combinations that allow the MPI_BCAST routine to apply Tofu barrier communication" for the number of the basic datatype as a constituent element.
  - When combined with the MCA parameter coll_tbi_repeat_max described in "Table 4.10 coll_tbi_repeat_max (Controls the range of message length to communicate by Tofu barrier communication)", the number of elements in a message is within a range not exceeding the value obtained by multiplying the maximum number of elements in a message by coll_tbi_repeat_max.

Table 6.19 Combinations that allow the MPI_BCAST routine to apply Tofu barrier communication

Datatypes	Size of each element	Maximum number of elements in a message
Integer datatypes Floating point datatypes Logical datatypes	8 bytes or less	6 (Each element is 8 bytes) 12 (Each element is 4 bytes) 24 (Each element is 2 bytes) 48 (Each element is 8 bytes)
Complex datatypes	4 bytes (2 bytes * 2) or 8 bytes (4 bytes * 2) or 16 bytes (8 bytes * 2)	12 (Each element is 4 bytes) or 6 (Each element is 8 bytes) or 3 (Each element is 16 bytes)
Byte datatypes	1 byte	48
Multi-language type	8 bytes	6

If the type signature of the send side of MPI_BCAST routine is different from that of the receive side, that is incorrect according to the MPI standard. The MPI program may not work correctly, such as abnormal end, because Tofu barrier cannot be used correctly. This is a problem that causes inconsistency in the environment of Tofu barrier communication because any of conditions above is not met on either side. Originally, such a program is incorrect according to the MPI standard, but you can avoid using Tofu barrier when executing the MPI program including MPI_BCAST routine by setting 0 in the MCA parameter coll_tbi_use_on_bcast value. It is also possible to avoid using Tofu barrier regardless of the MPI routine by setting the value ^tbi in the MCA parameter coll.

Refer to "Table 4.11 coll_tbi_use_on_bcast (uses Tofu barrier communication in MPI_BCAST routine)" for information on the MCA parameter coll_tbi_use_on_bcast.

Refer to "Table 4.9 coll (changes settings applied to all collective communications in common)" for information on the MCA parameter coll.

6.12.3 MPI_REDUCE and MPI_ALLREDUCE

The MPI_REDUCE routine and the MPI_ALLREDUCE routine can apply the Tofu barrier communication function if all the following conditions are met:

- The value ^tbi is not specified for the MCA parameter coll.
- The communicator is a communicator in a group (intra-communicator) and is not created using the MPI_INTERCOMM_MERGE routine.
- Any of the following conditions a. or b. is met.
  a. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 2, and the number of compute nodes allocated to the communicator is greater than or equal to 4.
  b. The specified value for the MCA parameter coll_tbi_intra_node_reduction is 3 or 4, and any of the following conditions 1. or 2. is met.
     1. The number of processes in the communicator is greater than or equal to 4.In addition, if the number of compute nodes allocated to the communicator is less than or equal to 3, the number of processes per compute node is greater than or equal to 2.

2. Both the number of processes in the communicator and the number of compute nodes allocated to the communicator are greater than or equal to 4.

- The required number of barrier gates, explained in "6.12.4 Notes on Tofu Barrier Communication", can be secured.
- The MCA parameter coll_base_reduce_commute_safe is not set to guarantee the sequence of reduction operations.
- The combination of the reduction operation, the datatype, the size of each element, and the maximum number of elements in a message is the one of the ways shown in "Table 6.20 Operation combinations that allow the MPI_REDUCE and MPI_ALLREDUCE routines to apply Tofu barrier communication".
  - When combined with the MCA parameter coll_tbi_repeat_max described in "Table 4.10 coll_tbi_repeat_max (Controls the range of message length to communicate by Tofu barrier communication)", the number of elements in a message is within a range not exceeding the value obtained by multiplying the maximum number of elements in a message by coll_tbi_repeat_max.

Table 6.20 Operation combinations that allow the MPI_REDUCE and MPI_ALLREDUCE routines to apply Tofu barrier communication

MPI predefined operation	Datatypes	Size	Maximum number of elements in a message
[C/Fortran] MPI_MAX MPI_MIN [C++] MPI::MAX MPI::MIN [Java] MPI.MAX MPI.MIN	Integer datatypes Floating point datatypes	8 bytes or less	6
	Multi-language type	8 bytes	6
[C/Fortran] MPI_SUM [C++] MPI::SUM [Java] MPI.SUM	Integer datatypes	8 bytes or less	6
	Floating point datatypes	8 bytes or less	3
	Complex datatypes	4 bytes (2 bytes * 2) or 8 bytes (4 bytes * 2) or 16 bytes (8 bytes * 2)	1
	Multi-language type	8 bytes	6
[C/Fortran] MPI_LAND MPI_LOR MPI_LXOR [C++] MPI::LAND MPI::LOR MPI::LXOR [Java] MPI.LAND MPI.LOR MPI.LXOR	Integer datatypes Logical datatypes	8 bytes or less	384
[C/Fortran] MPI_BAND MPI_BOR MPI_BXOR	Integer datatypes	8 bytes or less	6 (Each element is 8 bytes) 12 (Each element is 4 bytes) 24 (Each element is 2 bytes)

MPI predefined operation	Datatypes	Size	Maximum number of elements in a message
[C++] MPI::BAND MPI::BOR MPI::BXOR [Java] MPI.BAND MPI.BOR MPI.BXOR			48 (Each element is 8 bytes)
	Byte datatypes	1 byte	48
	Multi-language type	8 bytes	6
[C/Fortran] MPI_MAXLOC MPI_MINLOC [C++] MPI::MAXLOC MPI::MINLOC [Java] MPI.MAXLOC MPI.MINLOC	[C] MPI_2INT MPI_LONG_INT MPI_SHORT_INT [Fortran] MPI_2INTEGER [C++] MPI::TWOINT MPI::LONG_INT MPI::SHORT_INT MPI::TWOINTEGER [Java] MPI.INT2 MPI.LONG_INT MPI.SHORT_INT	8 bytes (4 bytes + 4 bytes) 12 bytes (8 bytes + 4 bytes) 6 bytes (2 bytes + 4 bytes)	3

6.12.4 Notes on Tofu Barrier Communication

- For Tofu barrier communication, the required barrier gates are secured from the multiple Tofu interconnect barrier gates provided in each compute node, and a barrier network is configured within the corresponding communicator. In order to configure a barrier network for a Tofu barrier communication, a barrier gate for start/end point and multiple barrier gates for relay points must be secured in each node.

  These barrier gates must be secured from the same TNI in each node.

  The maximum number of available barrier gates for start/end point is 16 per TNI. Therefore, it is 96 per compute node. The maximum number of available barrier gates for relay points is 32 per TNI. Therefore, it is 192 per compute node. Note that these maximum values are provided as guidelines, and the total number of barrier gates may be changed by system conditions or future product version upgrades.

- When Tofu barrier communication is applied in the MPI_BARRIER, MPI_BCAST, MPI_REDUCE, or MPI_ALLREDUCE routine, the number of barrier gates used for relay points is estimated as follows.

  Note that the number of compute node used on a Tofu barrier communication and the number of processes per compute node are assumed N and P, respectively.

  - When 2 is specified for the MCA parameter coll_tbi_intra_node_reduction, the maximum number of barrier gates which is used for each compute node is about $\log_2 N$.
  - When 3 is specified for the MCA parameter coll_tbi_intra_node_reduction, the maximum number of barrier gates which is used for each compute node node is about $\log_2 N+(3P-3)$.
  - When 4 is specified for the MCA parameter coll_tbi_intra_node_reduction, the maximum number of barrier gates which is used for each compute node is about $(\log_2 N+\log_2 P)*P$.

  The number of barrier gates which is actually available changes depending on barrier gates usage in each node.

Even if the same program is executed multiple times, the number of available barrier gates may be different every time if other jobs are using barrier gates at the same time.

There is also the case that the algorithm corresponding to the specified value for the MCA parameter coll_tbi_intra_node_reduction is not selected because of the number of available barrier gates.

- Even if 4 is specified for the MCA parameter coll_tbi_intra_node_reduction, there is the case that the algorithm which is used when 2 or 3 is specified for the MCA parameter is selected because of the number of barrier gates usage.

  There is also the case that a software algorithm which does not use Tofu barrier communication is selected.

  Therefore, note that execution time of a program can be longer than the algorithm which is expected to be selected is used.

  Also note that calculation result can be different due to precision error because the order of reduction operation is different for each algorithm.

- A barrier network is configured when a communicator or a window is created.

  In the following cases, the execution time of the program may be longer because a barrier network is frequently configured, but the execution time may be able to be improved by specifying ^tbi for the MCA parameter coll or by specifying 0 for the MCA parameter coll_tbi_use_on_comm_dup.

  - An MPI program which repeats communicator duplication by the MPI_COMM_DUP, MPI_COMM_IDUP, and MPI_COMM_DUP_WITH_INFO routine.
  - An MPI program which frequently creates and releases communicators.
  - An MPI program which repeats window creation.

  Read "Table 4.9 coll (changes settings applied to all collective communications in common)" for more information on the MCA parameter coll.

  Read "Table 4.12 coll_tbi_use_on_comm_dup (uses Tofu barrier communication for a communicator created by MPI_COMM_DUP routine, MPI_COMM_IDUP routine, and MPI_COMM_DUP_WITH_INFO routine)" for more information on the MCA parameter coll_tbi_use_on_comm_dup.

6.13 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 4.14 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.

6.14 Algorithms of Collective Communication and Shape of Compute Nodes Allocated to the Communicator

Collective communication has multiple algorithms that use the function of Tofu interconnect. Also, some of these algorithms become ready for calling when the shape of compute nodes allocated to the communicator has a specific shape. This section refers to this specific shape as "cuboid."

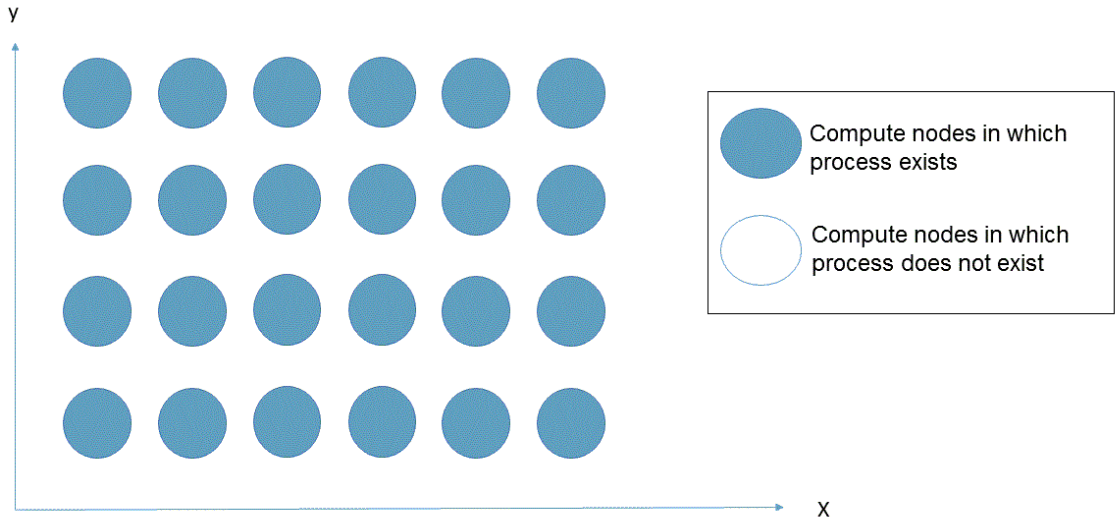
MPI statistical information has the function to display the shape of compute nodes allocated to the communicator. For the shape of the compute nodes allocated to the communicator, confirm with "6.16 MPI Statistical Information".

6.14.1 Shape of Compute Nodes Allocated to the MPI_COMM_WORLD

The user can specify the process shape when executing a job. For details of the "specification of the process shape", see the manual of the Job Operation Software. According to the shape of the process specified by the user, the MPI_COMM_WORLD is generated. When one or more processes are generated for all of the nodes specified for MPI_COMM_WORLD, the shape of this node is referred to as "cuboid." You can determine whether or not the shape is "cuboid" according to whether or not the process exists in the compute node, not by the number of processes within the compute node.

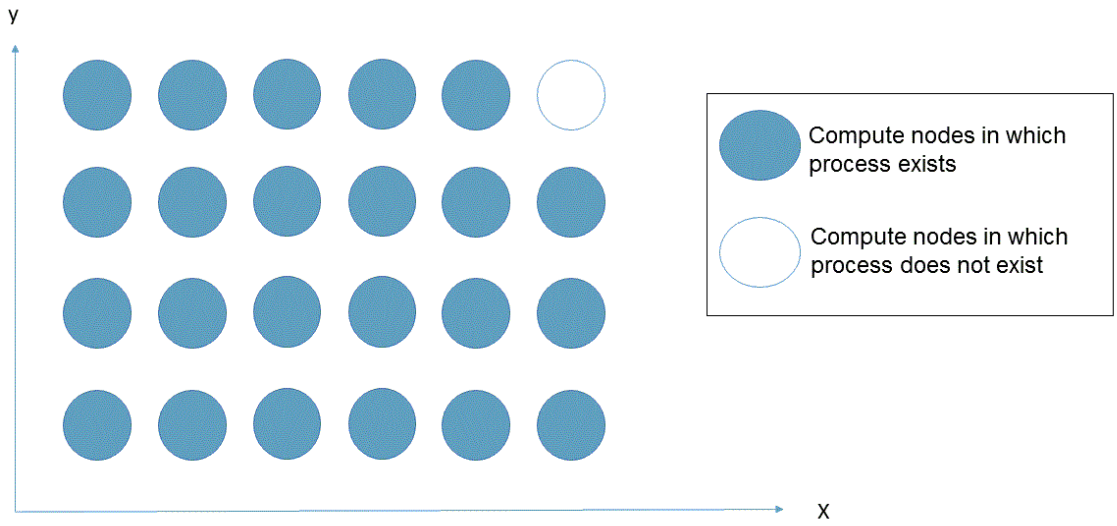
The following explanation is provided using an example when the user specifies the shape of process two-dimensionally. In the "Figure 6.8 Example of 24 Processes Assigned on 6*4 Plane". 24 processes exist on the plane of X*Y = 6*4. At that time one process each is assigned to all compute nodes. Therefore, MPI_COMM_WORLD is "cuboid".

Figure 6.8 Example of 24 Processes Assigned on 6*4 Plane



In the "Figure 6.9 Example of Indefinite Shape Where 23 Processes are Assigned on 6*4 Plane", 23 processes exist on the plane of X*Y = 6*4. At that time, the compute node to which no process is assigned exists. Therefore, MPI_COMM_WORLD is not "cuboid."

Figure 6.9 Example of Indefinite Shape Where 23 Processes are Assigned on 6*4 Plane



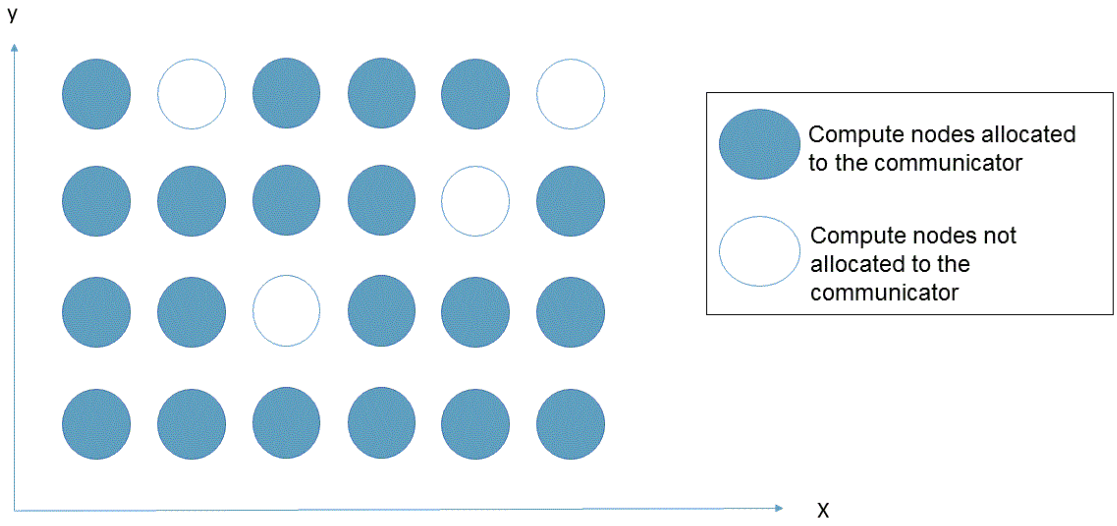
Later, the section provides explanation assuming that the compute nodes allocated to the MPI_COMM_WORLD are "cuboid."

6.14.2 Compute Nodes Allocated to the Intra-communicator

Intra-communicator is composed of the reproduction of MPI_COMM_WORLD or part of the processes of the MPI_COMM_WORLD. New intra-communicator generated from part of the MPI_COMM_WORLD may not become "cuboid." In the "Figure 6.8 Example of 24 Processes Assigned on 6*4 Plane", 24 processes exist on the plane of X*Y = 6*4 as MPI_COMM_WORLD.

The "Figure 6.10 Example of Generating the Communicator on 6*4 Plane at Random" is the example of generating the communicator at random from the MPI_COMM_WORLD of the "Figure 6.8 Example of 24 Processes Assigned on 6*4 Plane". This is not "cuboid." Such a shape cannot call the algorithms tuned for Tofu interconnect.

Figure 6.10 Example of Generating the Communicator on 6*4 Plane at Random



6.14.3 Shape of Compute Nodes for Algorithms Tuned for Tofu Interconnect

This section describes the "cuboid" in intra-communicator. The method of determining depends on the shape of the process specified by the user.

- when one dimension is specified

  It is always "cuboid" no matter what intra-communicator is generated.

- when two dimension is specified

  Suppose the user specifies the shape of the process as X*Y. Then, the value of 1..X-1 is I and the value of 1..Y-1 is J. When the process of the communicator is allocated so as to meet the following conditions, assume that the shape of the compute nodes allocated to that communicator is "cuboid".

  - X*Y itself
  - The shape that can be made by repeated processing to remove the rectangular of I*Y or X*J from X*Y

- when three dimensions are specified

  First, suppose the user specifies the shape of process as X*Y*Z. Then, the value of 1..X-1 is I, the value of 1..Y-1 is J, and the value of 1..Z-1 is K. When the process of the communicator is allocated so as to meet the following conditions, assume that the shape of the compute nodes allocated to that communicator is "cuboid".

  - X*Y*Z itself
  - The shape that can be made by repeated processing to remove the cuboid of I*Y*Z, X*J*Z, or X*Y*K from X*Y*Z

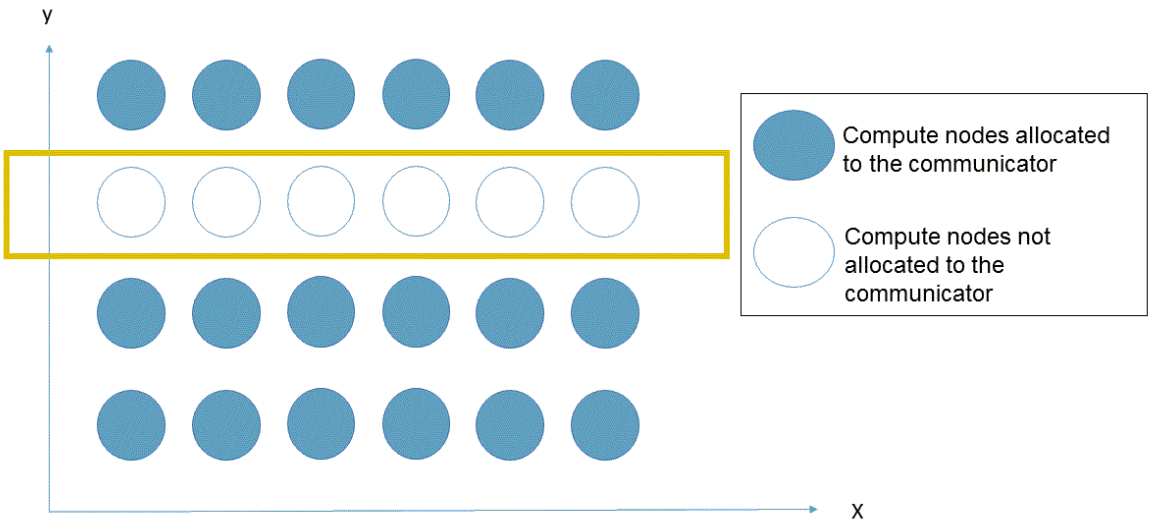
There exist algorithms that determine if the shape is "cuboid" based on the similar concept in six-dimensional Tofu coordinates.

As an example, the following explanation is provided for the case of specifying the shape of the process two-dimensionally.

In the "Figure 6.8 Example of 24 Processes Assigned on 6*4 Plane", processes are assigned on the plane of X*Y = 6*4 as MPI_COMM_WORLD. At that time, MPI_COMM_WORLD is "cuboid."

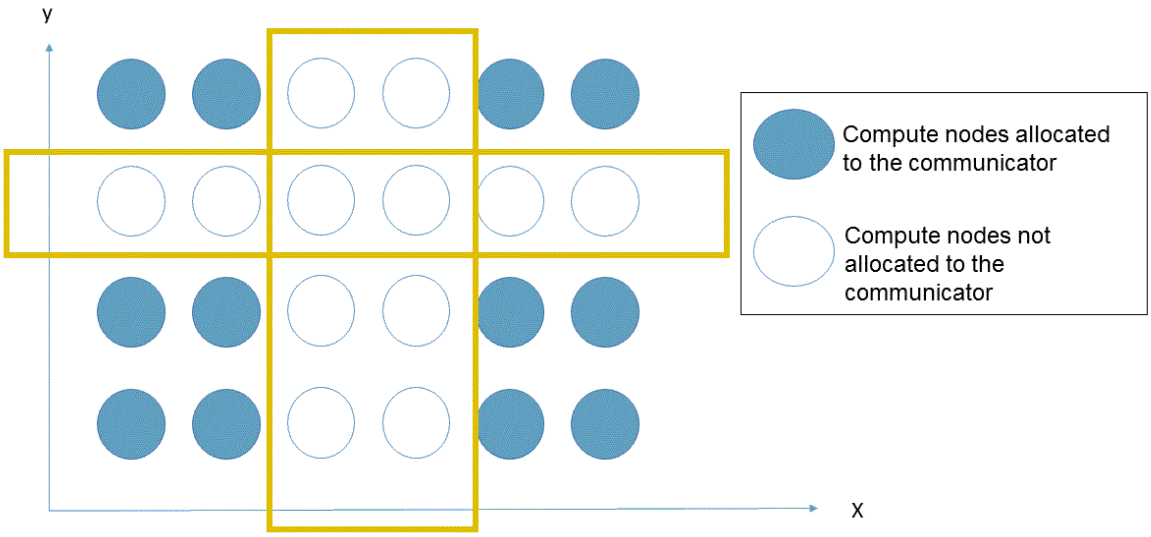
As is shown in the "Figure 6.11 Example of Specific One Row of 6*4 Removed", we will consider the case where the compute node that satisfies Y = 2 is not allocated to the new communicator. At that time, the communicator X is "cuboid."

Figure 6.11 Example of Specific One Row of 6*4 Removed



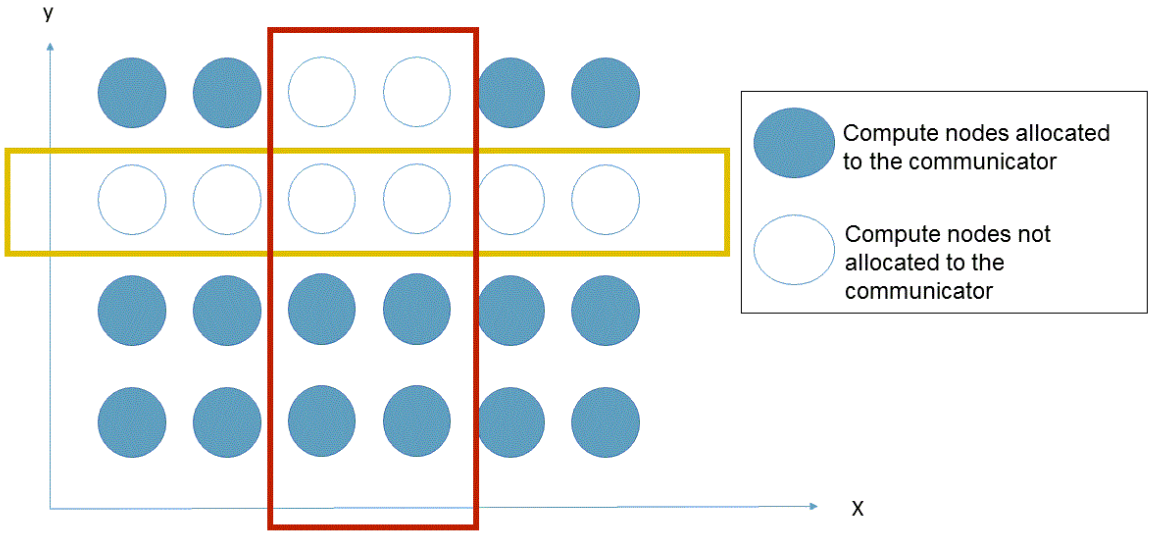
The "Figure 6.12 Example of Specific Two Rows Removed From Figure "Example of Specific One Row of 6*4 Removed"" shows the case where all compute nodes that satisfy X = 1 or X = 2 in the communicator X are not allocated to new communicator Y. At that time, the communicator Y is "cuboid."

Figure 6.12 Example of Specific Two Rows Removed From Figure "Example of Specific One Row of 6*4 Removed"



The "Figure 6.13 Example of Indefinite Shape" shows the case where part of the compute nodes that satisfy X = 1 or X = 2 in the communicator X are not allocated to new communicator Y. At that time, the communicator Y is not "cuboid."

Figure 6.13 Example of Indefinite Shape



6.14.4 Length of Axis with Cuboid Shape

The length of the axis is expressed by the number of the compute nodes allocated to the communicator that exist on each axis.

In the case of the "Figure 6.12 Example of Specific Two Rows Removed From Figure "Example of Specific One Row of 6*4 Removed"", the length of the X axis is 4 because the compute nodes allocated to the communicator exist on X = 0, 1, 4, 5. In a similar way, the length of the Y axis is 3 because the compute nodes allocated to the communicator exist on Y = 0, 2, 3. Therefore, this shape is indicated as 4*3 in MPI statistical information.

6.15 Job Dimension Conversion Function

6.15.1 Overview of Job Dimension Conversion Function

When a value greater than the job dimension is specified for the MCA parameter common_tofu_conv_dim at the time of an MPI program execution, the logical coordinates assigned to the MPI processes are replaced to the coordinates in the specified dimension, if possible. This makes it possible to reconcile the advantages of low-dimensional jobs, such as a short time to start job execution, with the advantages of high-dimensional jobs, such as high collective communication performance. However, there are some conditions that prevent the job dimension conversion function to be applied, and if these conditions are met, the execution continues using the original job dimension. Refer to "Table 6.21 Explanation of reason in the log of the job dimension conversion function" for the applicable conditions.

Notes on memory usage

When the job dimension conversion function is applied, the MPI library memory usage increases. The increment per MPI process is the number of nodes in the job multiplied by 24 bytes.

Notes on collective communication algorithms

Collective communication typically works faster for high-dimensional jobs than for low-dimensional jobs, but it is not guaranteed to be faster. In addition, in collective communication where reduction operations are performed, the sequence of reduction operations may change due to differences in coordinates and algorithms between high- and low-dimensional jobs. This can also affect the accuracy of the computed results. Refer to "6.8 Reduction Operation Sequence Guarantee in Collective Communication" for more information about the sequence of reduction operations.

6.15.2 Log Output for Job Dimension Conversion Function

The MCA parameter common_tofu_conv_dim_log can be used to output information about whether the job dimension conversion function was applied and why it was not applied.

If applied, the following message is output:

```
The job dimensions were converted from dim1 to dim2.
```

where *dim1* is the original job dimension and *dim2* is the job dimension to which the conversion was attempted.

If not applied, the following message is output :

```
Failed to convert the job dimensions to dim1. Continue as dim2-dimensional job. [reason]
```

where *dim1* is the job dimension for which the conversion was attempted, *dim2* is the original job dimension, and *reason* is the reason why it could not be applied. For a detailed explanation of *reason*, see "Table 6.21 Explanation of reason in the log of the job dimension conversion function".

Table 6.21 Explanation of reason in the log of the job dimension conversion function

reason	Description
1	The job dimension conversion function is available only when the job dimension is less than the value specified for the MCA parameter common_tofu_conv_dim. Check the MCA parameter specification or the specified node shape. Refer to "Job Operation Software End-user's Guide" for details on how to specify the node shape.
2	The job dimension conversion function is available only for jobs executed in torus mode. Check that the torus mode is specified as the method of placing nodes. Refer to "Job Operation Software End-user's Guide" for details of the method of placing nodes.

reason	Description
3	The job dimension conversion function is not available because the number of request nodes for the job is not a multiple of 12. Check the specified node shape. Refer to "Job Operation Software End-user's Guide" for details on how to specify the node shape.
4	The job dimension conversion function cannot be used because the number of nodes in the area allocated by the Job Operation Software does not match the number of requested nodes in the job. Check that the torus mode is specified as the method of placing nodes. However, even if the torus mode is specified, the Job Operation Software may allocate an area where the job dimension conversion function cannot be used, depending on the job shape or the status of free nodes in the system. For example, the following methods can be used to change the probability that an area that satisfies the conditions for applying this function is allocated. However, neither method guarantees that an area that satisfies the conditions for this function will be allocated. ・Change the dimensions of the job shape ・Change the number of nodes ・Change the length of each axis ・Change the resource group ・Check system usage status and avoid periods of long job runs that reduce space allocation flexibility Refer to "Job Operation Software End-user's Guide" for details of a node shape, the method of placing nodes, and the method of specifying a resource group.
5	If a process shape is specified in the shape parameter of the Job Operation Software and its size is smaller than the node shape specified in the node parameter, the job dimension conversion function cannot be used. Do not specify the shape or check the specified value for the shape parameter. Refer to "Job Operation Software End-user's Guide" for details about the shape and node parameters.
6	The job dimension conversion function cannot be used because a breakdown node is included in the area allocated by the Job Operation Software. To avoid breakdown nodes, specify the Job Operation Software option --mpi "assign-online-node". Refer to "Job Operation Software End-user's Guide" for details about the --mpi "assign-online-node" option.

6.16 MPI Statistical Information

This software system can display statistical information concerning MPI communications. In this manual, this information is referred to as MPI statistical information.

There are 2 varieties of a whole output mode and the section specifying output mode in the method of outputting the MPI statistical information.

The output of MPI statistical information can be controlled by the MCA parameter mpi_print_stats. Refer to "Table 4.37 mpi_print_stats (outputs MPI statistical information)" for details.

The "Table 6.22 Contents of MPI statistical information output for whole output mode" shows the MPI statistical information that can be output in this software system.

If 1 is specified for the MCA parameter mpi_print_stats, maximum values (MAX), minimum values (MIN), and average values (AVE) are output for all parallel processes for all output items other than "MPI Information", "Collective Communication Information", and "Process Mapping" shown in whole output mode "Table 6.22 Contents of MPI statistical information output for whole output mode". For maximum values and minimum values, the corresponding parallel process ranks are also output in parentheses.

If 2 is specified for the MCA parameter mpi_print_stats, information is output for the corresponding parallel processes for all output items other than "MPI Information" and "Process Mapping" shown in whole output mode "Table 6.22 Contents of MPI statistical information output for whole output mode". In this case, the parallel processes targeted for MPI statistical information output can be indicated using the MCA parameter mpi_print_stats_ranks. Refer to "Table 4.38 mpi_print_stats_ranks (specifies the parallel process that outputs MPI statistical information)" for details on mpi_print_stats_ranks. Note that these statistics are output to the standard error output. To avoid output overlaps when statistics are output for multiple parallel processes, specifications that suit the mpiexec command options -of-proc/-std-proc, --of-proc/--std-proc, -oferr-proc/-stderr-proc, or --oferr-proc/--stderr-proc are recommended.

If 3 is specified for the MCA parameter mpi_print_stats, maximum values (MAX), minimum values (MIN), and average values (AVE) are output for all parallel processes for all output items other than "MPI Information", "Collective Communication Information" and "Process

Mapping" shown in specifying output mode "Table 6.23 Contents of MPI statistical information output for section specifying output mode" similarly to when a value of MCA parameter mpi_print_stats is equal to 1. For maximum values and minimum values, the corresponding parallel process ranks are also output in parentheses. However, the header part, the body part and the footer part are output separately.

If 4 is specified for the MCA parameter mpi_print_stats, all items except "MPI Information" and "Process Mapping" shown in specifying output mode "Table 6.23 Contents of MPI statistical information output for section specifying output mode" similarly to when a value of MCA parameter mpi_print_stats is equal to 2. However, the header part, the body part and the footer part are output separately.

Refer to "5.2 MPI Statistical Information Section Specifying Interface" for details.

The Process Mapping output item of MPI statistical information is output only for parallel processes with a rank of 0, regardless of the value in the above MCA parameter mpi_print_stats_ranks.

When you use section specifying output mode, the specified section is ignored for Connection, Max_Hop and Average_Hop output items of MPI statistical information. The information at the point of an FJMPI_COLLECTION_PRINT routine call or an MPI_FINALIZE routine call is always output.

Output item Per-protocol Nonblocking/Persistent Communication Count and Per-protocol Nonblocking/Persistent Communication Count Started in Wait of the MPI statistical information will be a reference when MCA parameter opal_progress_thread_mode is used. Refer to "6.2 Promoting Asynchronous Communication Using an Assistant Core" for details.

Table 6.22 Contents of MPI statistical information output for whole output mode

Output title and output item name	Output content
MPI Information	
Dimension	Number of dimensions in the torus structure where the parallel processes belonging to MPI_COMM_WORLD are deployed
Shape	Process shape of the torus structure where the parallel processes belonging to MPI_COMM_WORLD are deployed
MPI Memory Usage	
Estimated_Memory_Size	Estimated MPI library memory allocation Estimated memory allocation value described in "6.11.1 Memory Usage Estimation Formulae"
Per-peer Communication Count	
In_Node	Communication count within node for point-to-point communication
Neighbor	Communication count between neighboring nodes for point-to-point communication
Not_Neighbor	Communication count between non-neighboring nodes for point-to-point communication
Total_Count	Total communication count for point-to-point communication
Connection	Number of connections for Tofu communication
Max_Hop	Maximum number of hops between processes for Tofu communication
Average_Hop	Average number of hops between processes for Tofu communication
Per-peer Transmission size	
In_Node	Transfer data size within node for point-to-point communication
Neighbor	Transfer data size between neighboring nodes for point-to-point communication
Not_Neighbor	Transfer data size between non-neighboring nodes for point-to-point communication
Total_Size	Transfer data size for point-to-point communication
Per-protocol Communication Count	
Eager	Eager communication mode use count at send side for point-to-point communication
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication

Output title and output item name	Output content
Prequest_Extended_IF	Extended persistent communication requests interface use count at send side for point-to-point communication
Unexpected_Message	Maximum number of messages saved temporarily in the internal buffer during point-to-point communication
Barrier Communication Count	
Tofu	Count for barriers using the Tofu barrier communication function
Soft	Count for software-style barriers executed
Tofu Barrier Collective Communication Count	
Bcast	Invocation count for collective communication MPI_BCAST routine using the Tofu barrier communication function
Reduce	Invocation count for collective communication MPI_REDUCE routine using the Tofu barrier communication function
Allreduce	Invocation count for collective communication MPI_ALLREDUCE routine using the Tofu barrier communication function
6D-Tofu-specific Collective Communication Count	
Alltoall	It is the number of executions of the blacc6d algorithm of the MPI_ALLTOALL routine, which is the algorithm tuned for Tofu interconnect.
Tofu-specific Collective Communication Count	
Bcast	Count for collective communication MPI_BCAST routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Reduce	Count for collective communication MPI_REDUCE routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Gather	Count for collective communication MPI_GATHER routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Gatherv	Count for collective communication MPI_GATHERV routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Allreduce	Count for collective communication MPI_ALLREDUCE routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Alltoall	Count for collective communication MPI_ALLTOALL routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Alltoallv	Count for collective communication MPI_ALLTOALLV routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Allgather	Count for collective communication MPI_ALLGATHER routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Allgatherv	Count for collective communication MPI_ALLGATHERV routine invocations that specifically invoke algorithms tuned for Tofu interconnect
Non-Tofu-specific Collective Communication Count	
Bcast	Count for collective communication MPI_BCAST routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Reduce	Count for collective communication MPI_REDUCE routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Gather	Count for collective communication MPI_GATHER routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Gatherv	Count for collective communication MPI_GATHERV routine invocations that could not specifically use algorithms tuned for Tofu interconnect

Output title and output item name	Output content
Allreduce	Count for collective communication MPI_ALLREDUCE routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Alltoall	Count for collective communication MPI_ALLTOALL routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Alltoallv	Count for collective communication MPI_ALLTOALLV routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Allgather	Count for collective communication MPI_ALLGATHER routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Allgatherv	Count for collective communication MPI_ALLGATHERV routine invocations that could not specifically use algorithms tuned for Tofu interconnect
Per-protocol Nonblocking/Persistent Communication Count	
Eager	Eager communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request
Collective	Nonblocking collective operations use count
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	
Eager	Eager communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request and the transfer of the message body started when any of MPI_WAIT, MPI_WAITANY, MPI_WAITALL, or MPI_WAITSOME routine is called
Rendezvous	Rendezvous communication mode use count at send side for point-to-point communication that used a nonblocking or persistent request and the transfer of the message body started when any of MPI_WAIT, MPI_WAITANY, MPI_WAITALL, or MPI_WAITSOME routine s called
Collective Communication Information	
COLLECTIVE	Name of the executed blocking collective communication routine. Each block collective communication routine name minus MPI_ is displayed.
ALG	Figure of the algorithm number of the executed blocking collective communication routine. For the algorithm that corresponds to the figure, see the table of the "8.3.4 List of Algorithms and Conditions Required for Application".
MSGMIN - MSGMAX	Range of the message size of the blocking collective communication routine. It shows that the message sizes which were specified for each blocking collective communication routine were within MSGMIN to MSGMAX. For the definition of message size of each blocking collective communication routine, see "Table 8.4 Definition of Message Size by Blocking Collective Communication Routine to Be Used for Info Object". However, the collective communication routines given below are expressed at a rough estimate based on the count of the output rank. - MPI_GATHERV routine - MPI_SCATTERV routine
NODE	Shape of the compute nodes allocated to the communicator that called the executed blocking collective communication. For the shape, see "6.14 Algorithms of Collective Communication and Shape of Compute Nodes Allocated to the Communicator".

Output title and output item name	Output content
	There are four types of expressions as given below. - Those with indefinite shape are expressed by the "number of compute nodes (I)". - Those with one-dimensional shape are expressed by the "number of compute nodes (R)". - Those with two-dimensional shape are expressed by the "X axis length x Y axis length". - Those with three-dimensional shape are expressed by the "X axis length x Y axis length x Z axis length".
COMM	Number of processes of the communicator that called the executed blocking collective communication.
COUNT	Number of executions.
AVE.TIME(MSEC)	Average execution time (millisecond unit)
Process Mapping	
	List of ranks and coordinate correspondences of all parallel processes belonging to MPI_COMM_WORLD However, this information is output for only parallel processes having rank 0.

Example

Example of MPI statistical information output when 1 is specified as the value of MCA parameter mpi_print_stats

```
================================================================
/****************** MPI Statistical Information ******************/
================================================================

------------------------- MPI Information -------------------------
Dimension                      3
Shape                      2x2x2

---------------------- MPI Memory Usage (MiB) ----------------------
                             MAX              MIN              AVE
Estimated_Memory_Size      89.05 [   3]     89.04 [   5]     89.05

------------------- Per-peer Communication Count ------------------
                             MAX              MIN              AVE
In_Node                        0 [   0]         0 [   0]       0.0
Neighbor                       0 [   0]         0 [   0]       0.0
Not_Neighbor                   0 [   0]         0 [   0]       0.0
Total_Count                    0 [   0]         0 [   0]       0.0
Connection                     8 [   3]         6 [   5]       7.1
Max_Hop                        4 [   0]         2 [   6]       3.4
Average_Hop                 2.62 [  15]      1.33 [   6]      2.03

----------------- Per-peer Transmission Size (MiB) ----------------
                             MAX              MIN              AVE
In_Node                     0.00 [   0]      0.00 [   0]      0.00
Neighbor                    0.00 [   0]      0.00 [   0]      0.00
Not_Neighbor                0.00 [   0]      0.00 [   0]      0.00
Total_Size                  0.00 [   0]      0.00 [   0]      0.00

----------------- Per-protocol Communication Count ----------------
                             MAX              MIN              AVE
Eager                          0 [   0]         0 [   0]       0.0
Rendezvous                     0 [   0]         0 [   0]       0.0
```

```
Persistent_Extended_IF          0 [   0]         0 [   0]       0.0
Unexpected_Message              2 [   0]         1 [   1]       1.5

------------------- Barrier Communication Count -------------------
                              MAX              MIN              AVE
Tofu                           10 [   0]        10 [   0]      10.0
Soft                            0 [   0]         0 [   0]       0.0

----------- Tofu Barrier Collective Communication Count -----------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Allreduce                       0 [   0]         0 [   0]       0.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                              MAX              MIN              AVE
Alltoall                        0 [   0]         0 [   0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Gather                          0 [   0]         0 [   0]       0.0
Gatherv                         1 [   0]         1 [   0]       1.0
Allreduce                       0 [   0]         0 [   0]       0.0
Alltoall                        1 [   0]         1 [   0]       1.0
Alltoallv                       0 [   0]         0 [   0]       0.0
Allgather                       1 [   0]         1 [   0]       1.0
Allgatherv                      1 [   0]         1 [   0]       1.0

--------- Non-Tofu-specific Collective Communication Count --------
                              MAX              MIN              AVE
Bcast                           1 [   0]         1 [   0]       1.0
Reduce                          1 [   0]         1 [   0]       1.0
Gather                          0 [   0]         0 [   0]       0.0
Gatherv                         0 [   0]         0 [   0]       0.0
Allreduce                       1 [   0]         1 [   0]       1.0
Alltoall                        0 [   0]         0 [   0]       0.0
Alltoallv                       0 [   0]         0 [   0]       0.0
Allgather                       0 [   0]         0 [   0]       0.0
Allgatherv                      0 [   0]         0 [   0]       0.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
                              MAX              MIN              AVE
Eager                           0 [   0]         0 [   0]       0.0
Rendezvous                      0 [   0]         0 [   0]       0.0
Collective                      0 [   0]         0 [   0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX              MIN              AVE
Eager                           0 [   0]         0 [   0]       0.0
Rendezvous                      0 [   0]         0 [   0]       0.0


-------------- Collective Communication Information --------------
COLLECTIVE      ALG    MSGMIN -    MSGMAX               NODE     COMM    COUNT AVE.TIME(MSEC)
Allgather       101  16777216 -  67108863     2x     2x     2       32        1 33.773
Allgatherv      101  67108864 -       inf     2x     2x     2       32        1 137.815
Alltoall        100   1048576 -   4194303     2x     2x     2       32        1 25.393
Barrier         200         0 -       inf     2x     2x     2       32        6 2.438
Gatherv         100   4194304 -  16777215     2x     2x     2       32        1 4.499
Scan              1         0 -      4095     2x     2x     2       32        1 0.022
Scatter         300    262144 -   1048575     2x     2x     2       32        1 16.054
```

```
Allreduce         3     16384 -     65535                5(I)        5        1 0.809
Barrier         200         0 -       inf                5(I)        5        4 0.101
Bcast             5      4096 -     16383                5(I)        5        1 0.257
Exscan            1         0 -      4095                5(I)        5        1 0.050
Reduce            6     65536 -    262143                5(I)        5        1 3.170

------------------------ Process Mapping ------------------------
(0,0,0)                   0,1,2,3
(1,0,0)                   4,5,6,7
(0,1,0)                   8,9,10,11
(1,1,0)                   12,13,14,15
(0,0,1)                   16,17,18,19
(1,0,1)                   20,21,22,23
(0,1,1)                   24,25,26,27
(1,1,1)                   28,29,30,31
```

Table 6.23 Contents of MPI statistical information output for section specifying output mode

Output title and output item name	Output content
Header part **Content output by FJMPI_COLLECTION_PRINT execution time of the first time point**	
MPI Information	
Dimension	*Refer to a whole output mode
Shape	
Body part Content output with FJMPI_COLLECTION_PRINT run unit	
Section	
Time(Sec)	Execution time at each section specifying (second)
Per-peer Communication Count	
In_Node	*Refer to a whole output mode
Neighbor	
Not_Neighbor	
Total_Count	
Connection	
Max_Hop	
Average_Hop	
Per-peer Transmission size	
In_Node	*Refer to a whole output mode
Neighbor	
Not_Neighbor	
Total_Size	
Per-protocol Communication Count	
Eager	*Refer to a whole output mode
Rendezvous	
Persistent_Extended_IF	
Unexpected_Message	

Output title and output item name	Output content
Barrier Communication Count	
Tofu	*Refer to a whole output mode
Soft	
Tofu Barrier Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Allreduce	
6D-Tofu-specific Collective Communication Count	
Alltoall	*Refer to a whole output mode
Tofu-specific Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	
Allgatherv	
Non-Tofu-specific Collective Communication Count	
Bcast	*Refer to a whole output mode
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	
Allgatherv	
Per-protocol Nonblocking/Persistent Communication Count	
Eager	*Refer to a whole output mode
Rendezvous	
Collective	
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	
Eager	*Refer to a whole output mode
Rendezvous	
Collective Communication Information	
COLLECTIVE	*Refer to a whole output mode
ALG	

Output title and output item name	Output content
MSGMIN - MSGMAX	
NODE	
COMM	
COUNT	
AVE.TIME(MSEC)	
Footer part Content output when MPI_FINALIZE routine is executed	
MPI Memory Usage	
Estimated_Memory_Size	*Refer to a whole output mode
Process Mapping	
	List of ranks and coordinate correspondences of all parallel processes belonging to MPI_COMM_WORLD However, this information is output for only parallel processes having rank 0.

Example

Example of MPI statistical information output when 3 is specified as the value of MCA parameter mpi_print_stats

```
================================================================
/****************** MPI Statistical Information ******************/
================================================================

------------------------- MPI Information -------------------------
Dimension                       3
Shape                       2x2x2

----------- Section    1 MPI_COMM_WORLD                 -----------
                              MAX              MIN              AVE
Time(Sec)                  0.03 [  28]      0.03 [  15]      0.03

------------------- Per-peer Communication Count ------------------
                              MAX              MIN              AVE
In_Node                      0 [   0]         0 [   0]       0.0
Neighbor                     0 [   0]         0 [   0]       0.0
Not_Neighbor                 0 [   0]         0 [   0]       0.0
Total_Count                  0 [   0]         0 [   0]       0.0
Connection                   3 [   0]         3 [   0]       3.0
Max_Hop                      4 [   8]         2 [   0]       2.6
Average_Hop               2.67 [   8]      1.33 [   0]      1.92

----------------- Per-peer Transmission Size (MiB) ----------------
                              MAX              MIN              AVE
In_Node                   0.00 [   0]      0.00 [   0]      0.00
Neighbor                  0.00 [   0]      0.00 [   0]      0.00
Not_Neighbor              0.00 [   0]      0.00 [   0]      0.00
Total_Size                0.00 [   0]      0.00 [   0]      0.00

----------------- Per-protocol Communication Count ----------------
                              MAX              MIN              AVE
Eager                        0 [   0]         0 [   0]       0.0
Rendezvous                   0 [   0]         0 [   0]       0.0
Persistent_Extended_IF       0 [   0]         0 [   0]       0.0
Unexpected_Message           2 [   4]         1 [   0]       1.4
```

```
------------------- Barrier Communication Count -------------------
                              MAX              MIN              AVE
Tofu                            5 [   0]         5 [   0]       5.0
Soft                            0 [   0]         0 [   0]       0.0

----------- Tofu Barrier Collective Communication Count -----------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Allreduce                       0 [   0]         0 [   0]       0.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                              MAX              MIN              AVE
Alltoall                        0 [   0]         0 [   0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Gather                          0 [   0]         0 [   0]       0.0
Gatherv                         1 [   0]         1 [   0]       1.0
Allreduce                       0 [   0]         0 [   0]       0.0
Alltoall                        1 [   0]         1 [   0]       1.0
Alltoallv                       0 [   0]         0 [   0]       0.0
Allgather                       1 [   0]         1 [   0]       1.0
Allgatherv                      1 [   0]         1 [   0]       1.0

--------- Non-Tofu-specific Collective Communication Count --------
                              MAX              MIN              AVE
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Gather                          0 [   0]         0 [   0]       0.0
Gatherv                         0 [   0]         0 [   0]       0.0
Allreduce                       0 [   0]         0 [   0]       0.0
Alltoall                        0 [   0]         0 [   0]       0.0
Alltoallv                       0 [   0]         0 [   0]       0.0
Allgather                       0 [   0]         0 [   0]       0.0
Allgatherv                      0 [   0]         0 [   0]       0.0

----- Per-protocol Nonblocking/Persistent Communication Count -----
                              MAX              MIN              AVE
Eager                           0 [   0]         0 [   0]       0.0
Rendezvous                      0 [   0]         0 [   0]       0.0
Collective                      0 [   0]         0 [   0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                              MAX              MIN              AVE
Eager                           0 [   0]         0 [   0]       0.0
Rendezvous                      0 [   0]         0 [   0]       0.0

--------------- Collective Communication Information --------------
COLLECTIVE      ALG    MSGMIN -    MSGMAX               NODE     COMM    COUNT AVE.TIME(MSEC)
Allgather       101  16777216 -  67108863     2x     2x     2       32        1 33.632
Allgatherv      101  67108864 -       inf     2x     2x     2       32        1 144.723
Alltoall        100   1048576 -   4194303     2x     2x     2       32        1 25.479
Barrier         200         0 -       inf     2x     2x     2       32        5 0.063
Gatherv         100   4194304 -  16777215     2x     2x     2       32        1 4.371
Scatter         300    262144 -   1048575     2x     2x     2       32        1 17.390

----------- Section    2 Splited communicator           -----------
                              MAX              MIN              AVE
Time(Sec)                  0.00 [  12]      0.00 [  28]      0.00
```

```
------------------- Per-peer Communication Count ------------------
                             MAX              MIN              AVE
In_Node                        0 [   0]         0 [   0]       0.0
Neighbor                       0 [   0]         0 [   0]       0.0
Not_Neighbor                   0 [   0]         0 [   0]       0.0
Total_Count                    0 [   0]         0 [   0]       0.0
Connection                     7 [   0]         5 [  11]       6.4
Max_Hop                        4 [   0]         2 [   6]       3.3
Average_Hop                 2.60 [  11]      1.33 [   6]      2.01

----------------- Per-peer Transmission Size (MiB) ----------------
                             MAX              MIN              AVE
In_Node                     0.00 [   0]      0.00 [   0]      0.00
Neighbor                    0.00 [   0]      0.00 [   0]      0.00
Not_Neighbor                0.00 [   0]      0.00 [   0]      0.00
Total_Size                  0.00 [   0]      0.00 [   0]      0.00

----------------- Per-protocol Communication Count ----------------
                             MAX              MIN              AVE
Eager                          0 [   0]         0 [   0]       0.0
Rendezvous                     0 [   0]         0 [   0]       0.0
Persistent_Extended_IF         0 [   0]         0 [   0]       0.0
Unexpected_Message             1 [   1]         0 [   0]       0.8

------------------- Barrier Communication Count -------------------
                             MAX              MIN              AVE
Tofu                           3 [   0]         3 [   0]       3.0
Soft                           0 [   0]         0 [   0]       0.0

----------- Tofu Barrier Collective Communication Count -----------
                             MAX              MIN              AVE
Bcast                          0 [   0]         0 [   0]       0.0
Reduce                         0 [   0]         0 [   0]       0.0
Allreduce                      0 [   0]         0 [   0]       0.0

--------- 6D-Tofu-specific Collective Communication Count ---------
                             MAX              MIN              AVE
Alltoall                       0 [   0]         0 [   0]       0.0

----------- Tofu-specific Collective Communication Count ----------
                             MAX              MIN              AVE
Bcast                          0 [   0]         0 [   0]       0.0
Reduce                         0 [   0]         0 [   0]       0.0
Gather                         0 [   0]         0 [   0]       0.0
Gatherv                        0 [   0]         0 [   0]       0.0
Allreduce                      0 [   0]         0 [   0]       0.0
Alltoall                       0 [   0]         0 [   0]       0.0
Alltoallv                      0 [   0]         0 [   0]       0.0
Allgather                      0 [   0]         0 [   0]       0.0
Allgatherv                     0 [   0]         0 [   0]       0.0

--------- Non-Tofu-specific Collective Communication Count --------
                             MAX              MIN              AVE
Bcast                          1 [   0]         1 [   0]       1.0
Reduce                         1 [   0]         1 [   0]       1.0
Gather                         0 [   0]         0 [   0]       0.0
Gatherv                        0 [   0]         0 [   0]       0.0
Allreduce                      1 [   0]         1 [   0]       1.0
Alltoall                       0 [   0]         0 [   0]       0.0
Alltoallv                      0 [   0]         0 [   0]       0.0
Allgather                      0 [   0]         0 [   0]       0.0
Allgatherv                     0 [   0]         0 [   0]       0.0
```

```
----- Per-protocol Nonblocking/Persistent Communication Count -----
                            MAX              MIN              AVE
Eager                         0 [   0]         0 [   0]       0.0
Rendezvous                    0 [   0]         0 [   0]       0.0
Collective                    0 [   0]         0 [   0]       0.0

-- Per-protocol Nonblocking/Persistent Communication Count Started in Wait --
                            MAX              MIN              AVE
Eager                         0 [   0]         0 [   0]       0.0
Rendezvous                    0 [   0]         0 [   0]       0.0

--------------- Collective Communication Information --------------
COLLECTIVE      ALG    MSGMIN -    MSGMAX               NODE     COMM    COUNT AVE.TIME(MSEC)
Allreduce         3     16384 -     65535               5(I)        5        1 2.926
Barrier         200         0 -       inf               5(I)        5        3 0.881
Bcast             5      4096 -     16383               5(I)        5        1 0.262
Reduce            6     65536 -    262143               5(I)        5        1 4.427

---------------------- MPI Memory Usage (MiB) ----------------------
                            MAX              MIN              AVE
Estimated_Memory_Size       92.23 [   3]     92.23 [   5]     92.23


------------------------- Process Mapping -------------------------
(0,0,0)                   0,1,2,3
(1,0,0)                   4,5,6,7
(0,1,0)                   8,9,10,11
(1,1,0)                   12,13,14,15
(0,0,1)                   16,17,18,19
(1,0,1)                   20,21,22,23
(0,1,1)                   24,25,26,27
(1,1,1)                   28,29,30,31
```

6.17 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
- Argument check function

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

6.17.1 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 4.43 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 4.39 opal_abort_delay (delays program termination when an error is detected)" for details of the MCA parameter.

6.17.2 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 4.33 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)
/.../... (  ...              ) [0xfffffffffa07f381c]
./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.

6.17.3 Argument Check Function

In this software system, using the debug MPI library, simple check whether arguments specified for the MPI routine calls are correct can be done.

In order to use this argument check function, specify either the -debuglib or --debuglib option at the time of the mpiexec command execution. Refer to "4.1 Execution Command Formats" for how to specify these options.

If the argument check function detects an error, a corresponding message is output and the MPI routine returns to its caller with the error class below. See "Appendix A Error Class List" for a list of the error classes output by this software system.

```
MPI_ERR_COMM      Invalid communicator
MPI_ERR_COUNT     Invalid count
MPI_ERR_TAG       Invalid tag
MPI_ERR_RANK      Invalid rank
MPI_ERR_TYPE      Invalid data type
MPI_ERR_BUFFER    Invalid buffer pointer
MPI_ERR_REQUEST   Invalid request
MPI_ERR_TOPOLOGY  Invalid topology
MPI_ERR_DIMS      Invalid dimension
MPI_ERR_ROOT      Invalid root
MPI_ERR_GROUP     Invalid group
MPI_ERR_OP        Invalid operation
MPI_ERR_ARG       Other invalid argument
```

Example

Example

```
#include <mpi.h>

int main(int argc, char *argv[]){
    int  sbuf = 1, rbuf;

    MPI_Init(&argc, &argv);
    MPI_Reduce(&sbuf, &rbuf, 1, MPI_INT, MPI_SUM, -1, MPI_COMM_WORLD); // root(-1) is invalid
    MPI_Finalize();
}
```

In this program example, an incorrect root process is specified for the argument of the MPI_Reduce function. If this MPI program is executed and an argument check is performed, an error message like the one below is output. The [em99-cn071:18342] output in the second and following lines show the host name and pid information.

```
[mpi::mpi-errors::mpi_errors_are_fatal]
[em99-cn071:18342] *** An error occurred in MPI_Reduce
[em99-cn071:18342] *** reported by process [11111,0]
[em99-cn071:18342] *** on communicator MPI_COMM_WORLD
[em99-cn071:18342] *** MPI_ERR_ROOT: invalid root
[em99-cn071:18342] *** MPI_ERRORS_ARE_FATAL (processes in this communicator will now abort,
[em99-cn071:18342] ***    and potentially your MPI job)
```

6.18 Note on using 3rd Party Tools

6.18.1 Notes on Using Valgrind

The Valgrind is an open source software which can detects an illegal memory access or the like. In this software system, the behavior when the Valgrind is used is not guaranteed. However, the Valgrind may be available when executing an MPI program depending on the version of the Valgrind.

To use the Valgrind in this software system, you need to use the debug MPI library. Specify -debuglib or --debuglib for an mpiexec command option.

An example is shown as follows.

```
$ mpiexec -debuglib valgrind ./a.out
```

If the Valgrind is used for an MPI program execution, errors and warnings not only in the MPI program but also in the routines or the like called inside the MPI library may be detected by the Valgrind.

Therefore, note that whether the errors and warnings detected by the Valgrind are regarding the MPI program.

Information

Many functions called inside the MPI library have names begin with the following prefixes.

- opal
- ompi
- orte
- mca
- MPI

6.19 Notes on Job Execution Continuation Function at Link-down

This section provides notes on job execution continuation function at link-down. Refer to the Job Operation Software manual for information on job execution continuation function at link-down.

6.19.1 Communication Performance

When this function is enabled and any of the following routines is called in an MPI program, the communication performance may deteriorate compared to when this function is disabled.

- Following routines which are affected by not applying the Tofu barrier communication
  - The MPI_ALLREDUCE routine
  - The MPI_BARRIER routine
  - The MPI_BCAST routine
  - The MPI_REDUCE routine
- The routines of the one-sided communications
- The MPIX_ALLGATHER_INIT routine

6.19.2 Conditions in MPI_ALLTOALL Routine

When this function is enabled and the MPI_ALLTOALL routine is called in an MPI program, the following two conditions must be met. If they are not met, a deadlock may occur when executing the MPI program.

1. Datatype for the send buffer is equivalent in all ranks.
2. Datatype for the receive buffer is equivalent in all ranks.

6.19.3 Algorithms which are not Applicable

When this function is enabled, algorithms shown in the table below are not applicable in the MPI_ALLGATHER routine, the MPI_ALLGATHERV routine, and the MPI_ALLTOALL routine.

Table 6.24 List of algorithms which are not applicable

Collective Communication Routine Name	Algorithms which are not applicable
MPI_ALLGATHER	3dtorus_sm
MPI_ALLGATHERV	3dtorus_sm
MPI_ALLTOALL	blacc3d blacc6d

6.19.4 Note on Specification of MCA Parameters and Options

In this function, the -tune option in "local_options format and explanation" cannot be specified. If the -tune option is specified, this function is disabled.

When this function is enable, the specified values for the MCA parameters shown in the table below are invalid. In this case, the values for these MCA parameters are decided by this software system.

Table 6.25 The behavior when the specified value of the MCA parameter is invalid

Name of MCA Parameter	Invalid Value of MCA Parameter	Value which is decided by this system
	Meaning of Value	Meaning of Value
common_tofu_memory_saving_method	2	1 (default value)
	The method which uses the Shared receive buffer is used when a communication using the memory-saving communication mode is performed.	The method which uses the Medium receive buffer is used when a communication using the memory-saving communication mode is performed.
common_tofu_use_multi_path	1	0 (default value)
	Multiple communication paths are used for point-to-point communication. In other words, trunking is performed.	Multiple communication paths are not used for point-to-point communication.

6.19.5 Dynamic Process Creation

When this function is enabled, and the dynamic process creation function is used, do not terminate the original parallel process that created the dynamic process until the execution of the dynamic process's MPI_FINALIZE routine is complete.

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::plm-ple::exec-plexec] Failed to invoke PLE. [errinfo:*errinfo*(*errno*) path:*com*]

- Description

  Execution of the plexec command failed. The parallel execution environment (PLE) of Job Operation Software may not be operating correctly. Execution of the mpiexec command ends.

- Parameters

  errinfo : Error details

  errno : Error number

  com : Executing command

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::plm-ple::parallel] Specified number of parallel processes is incorrect.

- Description

  The number of parallel processes specified in the mpiexec command is incorrect. Execution of the mpiexec command ends.

- Action method

  Check the number of parallel processes in the mpiexec command.

[mpi::plm-ple::recursive-mpiexec] mpiexec cannot be invoked recursively.

- Description

  Duplicate startup of the mpiexec command from the mpiexec command is not possible. Execution of the mpiexec command ends.

- Action method

  Specify an MPI program in the mpiexec command.

[mpi::plm-ple::signal-plexec] Received signal sent by PLE. [signo:*signo*]

- Description

  The plexec command received a signal and ended abnormally. The parallel execution environment (PLE) of Job Operation Software may not be operating correctly. Execution of the mpiexec command ends.

- Parameters

  signo : Signal number received by child process

- Action method

  Ask the system administrator to check whether the parallel execution environment (PLE) of Job Operation Software is operating correctly. If it is operating correctly, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::plm-ple::wait-plexec] System error caused by waitpid. [errinfo:*errinfo*(*errno*)]

- Description

  Operation failed for the system call waitpid. Execution of the mpiexec command ends.

- Parameters

  errinfo : Error details

  errno : Error number

- Action method

  The system may not be operating correctly. Contact the system administrator.

[mpi::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::btl-tofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed when communicating through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::coll-mtofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check the amount of memory used and the maximum memory size limit value of the program. If the amount of memory used is too high, reduce it.

[mpi::coll-select::memory-error] Unable to allocate memory. [*data*]

- Description

  Failed to obtain memory.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Confirm the used amount of memory and the maximum memory size limit value of the program. When the used amount of memory is large, reduce it.

[mpi::coll-select::module-enable-failure] Internal error. [*data*]

- Description

  Problem occurred during the initialization of the algorithm selection. Exit the program.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-select::module-destruct-failure] Internal error. [*data*]

- Description

  Problem occurred during the process of ending the algorithm selection. Exit the program.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-select::system-file-error] Algorithm selection by the system file does not work. [*reason*]

- Description

  Problem occurred with the algorithm selection of the system. At that time, part of the function of the MPI statistical information cannot be displayed correctly.

- Parameters

  reason : Figure to indicate reason

reason	Content of Reason
0	There is an error in string or syntax described in the file to be used at the algorithm selection by the system.

reason	Content of Reason
1	The number of rules exceeds the one that can be described in the file to be used at the algorithm selection by the system.
2	The number of conditional statements exceeds the one that can be described in the file to be used at the algorithm selection by the system.
3	There is division by zero in calculation described in the file to be used at the algorithm selection by the system.
4	There is an error in a variable described in the file to be used at the algorithm selection by the system.
5	There is an error in the algorithm name.
6	No executable algorithms exist.
7	The rule for blocking collective communication does not encompass this.
8	Barrier communication is specified in the file to be used at the algorithm selection by the system.
9	An internal error of the MPI library occurred during analysis of the file to be used at the algorithm selection by the system.

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tbi::comm-query-failure] Internal error. [*reason*]

- Description

  Creation of a Tofu barrier network failed.

- Parameters

  reason : Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tbi::module-destruct-failure] Internal error. [*reason*]

- Description

  Release of a Tofu barrier network failed.

- Parameters

  reason : Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tbi::internal-file-error] Unable to *operate* file *file*.

- Description

  Access to the file used internally by the Tofu barrier failed.

- Parameters

  operate : Operation

  file : File path

- Action method

  The file system may not be operating correctly. Contact the system administrator.

[mpi::coll-tbi::memory-error] Unable to allocate memory. [*errno*]

- Description

  Memory acquisition failed for the tofu barrier.

- Parameters

  errno : Error number

- Action method

  Check the memory usage. If there is no problem, the system may not be operating correctly. Contact the system administrator.

[mpi::coll-tbi::operation-error] Operation error is reported by Tofu barrier communication. [*rank function arguments*] [*data*] *<Stack trace information>*

- Description

  An operation error was detected in the Tofu barrier communication. An error of the error class MPI_ERR_OP occurs.

- Parameters

  rank : Rank in the communicator

  function : MPI routine

  arguments : Argument of MPI routine

  data : Data for System Engineer for analysis purposes

- Action method

  Check whether there is either of the following descriptions at the shown MPI routine in your MPI program.

  The reduction operation is showed in arguments as op.

  - Different reduction operations were specified among processes
  - Different collective communication routines were specified among processes

  If this checking indicates no errors, an internal error may have occurred. Consult the System Engineer about the message that was output.

[mpi::coll-tuned::init-subcommunicator-failure] Internal error. [*reason*]

- Description

  Initialization processing failed for the collective communication subcommunicator processing.

- Parameters

  reason : Cause of failure

- Action method

  Consult System Engineer about the message that was output.

[mpi::coll-tuned::memory-error] Unable to allocate memory.

- Description

  Memory acquisition failed.

- Action method

  Check the amount of memory used and the maximum memory size limit value of the program. If the amount of memory used is too high, reduce it.

[mpi::common-tofu::connection-error] Connection error. [*data*]

- Description

  An error was detected when establishing the connection of communication through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::memory-error] Unable to allocate memory. [*data*]

- Description

  Memory acquisition failed when communicating through Tofu interconnect. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check the memory usage and memory size limit of the program. If there is a problem in the memory usage, reduce memory usage.

[mpi::common-tofu::mrq-error] Communication error is reported by Tofu MRQ. [*data*]

- Description

  The Tofu interconnect detected a problem. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::mrq-memory-error] Communication memory error is reported by Tofu MRQ. [*data*]

- Description

  The Tofu interconnect detected a communication memory specification error. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check whether there is an error in the start address, datatype, or number of elements in the send buffer and receive buffer specified in the MPI communication routine.

[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ. This error may be caused by abort of peer process. [*data*]

- Description

  The Tofu interconnect detected an error on the compute node where the communication partner process is being executed. The error may be caused by reason that the communication partner process was stopped or that the communication partner process released the send buffer or the receive buffer. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  In case the job was force-quitted, or the MPI program was terminated before MPI_FINALIZE routine is called, ignore this message. In case the send buffer or the receive buffer is released before the completion of communication, revise the processing of releasing buffer. Otherwise, consult System Engineer about the message that was output.

[mpi::common-tofu::tcq-error] Communication error is reported by Tofu TCQ. [*data*]

- Description

  The Tofu interconnect detected a communication error. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tcq-memory-error] Communication memory error is reported by Tofu TCQ. [*data*]

- Description

  The Tofu interconnect detected a communication memory specification error. Execution of the MPI program ends.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check whether there is an error in the start address, datatype, or number of elements in the send buffer and receive buffer specified in the MPI communication routine.

[mpi::common-tofu::tofu-async-error] Tofu interconnect detected an asynchronous error. [Event: *description* (*eventno*), TNI: *tniid*, BG: *bgid*]

- Description

  An internal error was detected in Tofu barrier communication. Execution of the MPI program ends.

- Parameters

  description : Error message provided by the communication library (Tofu library)

  eventno : Management number of the communication library (Tofu library) corresponding to the detected internal error

  tniid : ID of the network interface device (TNI) which detected the error

  bgid : ID of the barrier gate (BG) which detected the error

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-async-error] Tofu interconnect detected an asynchronous error. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- Description

  An internal error was detected in Tofu one-sided communication. Execution of the MPI program ends.

- Parameters

  description : Error message provided by the communication library (Tofu library)

  eventno : Management number of the communication library (Tofu library) corresponding to the detected internal error

  tniid : ID of the network interface device (TNI) which detected the error

  cqid : ID of the completion queue (CQ) which detected the error

- Action method

  Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-init-failure] Internal error. [*reason*]

- Description

  Initialization processing for the Tofu interconnect failed. Execution of the MPI program ends.

- Parameters

  reason : Cause of initialization processing failure

- Action method

  The system may not be operating correctly. Contact the system administrator.

[mpi::common-tofu::tofu-mrq-overflow] Tofu interconnect detected an MRQ overflow. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- Description

  The number of completion notices from the Tofu interconnect exceeded the number of entries in a completion queue. There is a problem with the issue count of nonblocking communication processing in the MPI program. Execution of the MPI program ends.

- Parameters

  description : Error message provided by the communication library (Tofu library)

  eventno : Management number of the communication library (Tofu library) corresponding to the detected internal error

  tniid : ID of the network interface device (TNI) which detected the error

  cqid : ID of the completion queue (CQ) which detected the error

- Action method

  Revise the processing of nonblocking communication or the completion processing of Eager communication in your MPI program, or increase the number of entries in a completion queue of the Tofu interconnect as described in "Table 4.26 common_tofu_num_mrq_entries (change the number of entries in a completion queue)".

[mpi::common-tofu::tofu-congestion] Tofu interconnect detected the congestion. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- Description

  Congestion has been detected on the Tofu interconnect. Execution of the MPI program ends.

- Parameters

  description : Error message provided by the communication library (Tofu library)

  eventno : Management number of the communication library (Tofu library) corresponding to the detected internal error

  tniid : ID of the network interface device (TNI) which detected the error

  cqid : ID of the completion queue (CQ) which detected the error

- Action method

  Revise the processing of the MPI program so that communication is not congested to a specific process.

[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag. [*data*]

- Description

  A buffer usage error or a shortage in a Tofu interconnect memory management resource was detected.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Check for errors in the start address of the send buffer and receive buffer, the data type, and the number of elements. Memory areas that MPI program parallel processes cannot write to cannot be specified in send or receive buffers. Alternatively, if the large page is not used, use the large page, or decrease patterns of the start address and number of elements in the send buffer and receive buffer specified in the MPI communication routine. If the large page is used and there are no specification errors, an internal error may have occurred. Consult System Engineer about the message that was output.

[mpi::common-tofu::tofu-stag-release-error] Failed to release Tofu STag. [*data*]

- Description

  Inconsistency was detected when releasing Tofu interconnect memory management resource.

- Parameters

  data : Data for System Engineer for analysis purposes

- Action method

  Consult System Engineer about the message that was output.

[mpi::dpm::num-nodes-invalid] [[*jobid*,*snum*],*rank*] The specified number of nodes is invalid.

- Description

  The number of nodes specified with the info key num_nodes may be greater than the number of nodes which are allocated to the job.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the number of nodes specified with the info key num_nodes.

[mpi::dpm::rankmap-invalid] [[*jobid*,*snum*],*rank*] The value specified with the info key rank_map is invalid.

- Description

  The value specified with the info key rank_map of the dynamic process creation routine is invalid. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Specify "bychip" or "bynode" with the info key rank_map.

[mpi::dpm::spawn-exist-error] [[*jobid*,*snum*],*rank*] The number of MPI_COMM_WORLD for the dynamic processes that can exist at the same time exceeds the upper limit.

- Description

  The upper limit number of MPI_COMM_WORLD for the dynamic processes that can exist at the same time was exceeded. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

snum : spawn number

rank : Rank in MPI_COMM_WORLD

- Action method

  Set 65535 or less as the number of MPI_COMM_WORLD for the dynamic processes that exists at the same time.

[mpi::dpm::spawn-limit-error] [[*jobid*,*snum*],*rank*] The total invocation count of the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine exceeds the upper limit.

- Description

  The upper limit value for the dynamic process creation count was exceeded. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Set 4294967295 or less as the dynamic process creation count.

[mpi::dpm::spawn-resource-error] [[*jobid*,*snum*],*rank*] There are not enough compute nodes to create processes dynamically according to the requirement.

- Description

  There are not enough free nodes that can execute the specified dynamic process creation. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Confirm the designations of the number of nodes or the number of processes per node for dynamic process creation. If only the number of CPUs (cores) allocated to each process is specified in the VCOORD file, the designation order may be the cause of this error.

[mpi::dpm::vcoord-core-num-invalid] [[*jobid*,*snum*],*rank*] The number of CPUs (cores) on a compute node is not sufficient.

- Description

  There are not enough CPUs (cores) to allocate for the dynamic process creation. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  In the VCOORD file specified at dynamic process creation, the total number of CPUs (cores) allocated to processes created on a node should be less than or equal to the number of CPUs (cores) mounted on the node.

[mpi::dpm::vcoord-format-error] [[*jobid*,*snum*],*rank*] The VCOORD file format is invalid.

- Description

  For dynamic process creation, the VCOORD file specified with the info key vcoordfile contains a format error. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the contents of the file specified in dynamic process creation.

[mpi::dpm::vcoord-invalid] [[*jobid*,*snum*],*rank*] The specified logical coordinates are invalid.

- Description

  In the VCOORD file specified with the info key vcoordfile of dynamic process creation, invalid logical corrdinates are specified. The specified logical coordinates may be out of the range of the nodes allocated to the job. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the contents of the file specified in dynamic process creation.

[mpi::dpm::vcoord-maxproc-invalid] [[*jobid*,*snum*],*rank*] The number of processes exceeds the number of logical coordinates specified in the VCOORD file.

- Description

  The VCOORD file is short of lines compared to the number of processes specified with maxprocs argument of dynamic process creation routine. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  The number of lines of the VCOORD file should be greater than or equal to the number of processes specified with maxprocs argument of dynamic process creation routine.

[mpi::dpm::vcoord-not-exist] [[*jobid*,*snum*],*rank*] The file specified with the vcoordfile info key does not exist.

- Description

  The VCOORD file specified with the info key vcoordfile of the dynamic process creation routine does not exist. Execution of the MPI program ends.

- Parameters

  jobid : MPI job ID

  snum : spawn number

rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the VCOORD file path specified with the info key vcoordfile.

[mpi::dpm::vcoord-numa-error] [[*jobid*,*snum*],*rank*] The process cannot be bound to CPUs (cores) under numanode_assign_policy.

- Description

  For dynamic process creation, a process cannot be bound to the CPU (core) under the value of numanode_assign_policy.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the value of numanode_assign_policy specified in the VCOORD file of dynamic process creation.

[mpi::dpm::vcoord-ppn-invalid] [[*jobid*,*snum*],*rank*] The number of processes exceeds the limit on a compute node.

- Description

  For dynamic process creation, the number of processes to be created on a node exceeds the limit on the number of processes that can be created per node.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Revise the VCOORD file not to exceed the limit on the number of processes that can be created per node.

[mpi::dpm::vcoord-use-error] [[*jobid*,*snum*],*rank*] The specified logical coordinates are already used.

- Description

  For dynamic process creation, the logical coordinates specified with the VCOORD file are already used.

- Parameters

  jobid : MPI job ID

  snum : spawn number

  rank : Rank in MPI_COMM_WORLD

- Action method

  Specify other logical coordinates or execute again after the executing processes end. FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES is returned for the error code.

[mpi::dpm::violated-establishing-communication] MPI_COMM_CONNECT or MPI_COMM_ACCEPT is called although the specified value for the MCA parameter mpi_no_establish_communication is 1.

- Description

  The program tried to establish communication between two groups of MPI processes that do not share a communicator although the MCA parameter mpi_no_establish_communication is specified. Execution of the MPI program ends.

- Action method

  Check the value specified for the MCA parameter or that the program does not establish communication between two groups of MPI processes that do not share a communicator.

[mpi::dpm::violated-spawn] MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE is called although the specified value for the MCA parameter mpi_no_establish_communication is 1.

- Description

  The program tried to create the dynamic processes although 1 is specified for the MCA parameter mpi_no_establish_communication. Execution of the MPI program ends.

- Action method

  Check the value specified for the MCA parameter or that the program does not create dynamic processes.

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

- Description

  The arguments of source/destination rank, message tag, and communicator for the request are identical to those of another request.

- Action method

  Specify other source/destination rank, message tag, or the communicator as the arguments of the FJMPI_PREQUEST_SEND_INIT routine or the FJMPI_PREQUEST_RECV_INIT routine. Or execute the FJMPI_PREQUEST_SEND_INIT routine or FJMPI_PREQUEST_RECV_INIT routine after freeing another request that already exists by the MPI_REQUEST_FREE routine.

--
[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-runtime::mpi-param-check-enabled-but-compiled-out]
WARNING: The MCA parameter mpi_param_check has been set to true, but parameter checking has been compiled out of Open MPI. The mpi_param_check value has therefore been ignored.

- Description

  The MCA parameter "mpi_param_check" was set. MCA parameter check cannot be specified unless the library is the debug MPI library. After message output, execution of the MPI program continues.

- Action method

  Remove the MCA parameter "mpi_param_check" specification.

--
[mpi::mpi-errors::mpi_errors_are_fatal]
[*info*] * An error occurred [*msg*]**
[*info*] * reported by process [[*jobid*,*rank*]]**
[*info*] * on [*type*]**
[*info*] * [*error class*]**
[*info*] * MPI_ERRORS_ARE_FATAL (processes in this [*type*] will now abort,**
[*info*] * and potentially your MPI job)**
--

- Description

  A fatal error occurred during execution of the MPI program. The program will abort.

- Parameters

  info : Host name and pid information

  msg : Explanation of the cause of the problem

  jobid : MPI job ID

  rank : Rank 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.
  - Because the MPI library saves messages to other regions if corresponding receive calls are delayed, these saved messages affect the amount of memory used. Therefore, if the number of all communication partner processes is more than the upper limit for the number of processes that can use the fast communication mode, and MPI program issues a large number of unexpected messages, then this problem might be avoided by speeding up the communication according to the following procedure.

    1. Execute the MPI program after specifying value 2 for the MCA parameter mpi_print_stats. Do not specify MCA parameter mpi_print_stats_ranks. When you use the MCA parameter, you must specify -1 for mpi_print_stats_ranks, and the output from all the parallel processes must enable. Confirm the number of unexpected messages by outputting the MPI statistical information. Refer to "6.16 MPI Statistical Information" for details.
    2. If the number of unexpected messages increased, enlarge the size of the Medium receive buffer. Refer to "Table 4.22 common_tofu_medium_recv_buf_size (changes the size of the Medium receive buffer)" for details.

[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.

Chapter 8 Speeding Up Blocking Collective Communication

This chapter describes two advanced performance tuning method for blocking collective communication. One is parameter tuning of algorithms and the other is algorithm selection. The intended readers of this chapter are those who have the following knowledge related to collective communication.

- Functions of available MCA parameters in this software system
- Features and performance characteristics of each algorithm
- Understanding of "6.14 Algorithms of Collective Communication and Shape of Compute Nodes Allocated to the Communicator"
- Performance analysis

Note

Refer to ompi/mca/coll/base directory in source code of Open MPI and the following references for available algorithms and each feature.

- Performance analysis of MPI collective operations

  "https://link.springer.com/article/10.1007/s10586-007-0012-0"

- The design of ultra scalable MPI collective communication on the K computer

  "https://link.springer.com/article/10.1007/s00450-012-0211-7"

8.1 Outline

In this system, one or more algorithm is implemented for one blocking collective communication routine. This system selects high-speed algorithms according to such information as the argument of the collective communication routine and the shape of the communicator, when the blocking collective communication routine is called. The performance of some algorithms may change by changing the parameter. In this way, the performance of the algorithm changes by changing the selection of algorithms and the parameter of the algorithm itself.

8.2 MCA Parameter Tuning of Algorithms

The performance of some algorithms may change by changing the MCA parameter.

8.2.1 Changing Segment Size

8.2.1.1 Change of Algorithm Performance by Changing Segment Size

Some algorithms may perform pipeline transfer. The pipeline transfer is the method to transfer message by dividing it into segments of a certain size. The performance of the pipeline transfer depends on the message size and segment size. This system adjusts the segment size by message size so as to enhance performance.

The following three MCA parameters can be specified: coll_select_allreduce_algorithm_segmentsize, coll_select_bcast_algorithm_segmentsize, and coll_select_reduce_algorithm_segmentsize. Algorithms corresponding to the MCA parameters are given in the following table.

Table 8.1 List of MCA Parameters to Specify Segment Size and Specifiable Algorithms

MCA Parameter	Collective Communication Routine Name	Specifiable Algorithm
coll_select_allreduce_algorithm_segmentsize	MPI_ALLREDUCE	segmented_ring trinaryx6 (trix6) trinaryx3 (trix3)

MCA Parameter	Collective Communication Routine Name	Specifiable Algorithm
coll_select_bcast_algorithm_segmentsize	MPI_BCAST	chain pipeline split_binary_tree binary_tree binomial trinaryx6 (trix6) bintree3d (bin3d) trinaryx3 (trix3) bintree6d (bin6d)
coll_select_reduce_algorithm_segmentsize	MPI_REDUCE	chain pipeline binary binomial in-order_binary trinaryx6 (trix6) trinaryx3 (trix3)

Use them with the MCA parameter to specify algorithms in each collective communication routine.

Example

Example of Specifying the Segment Size by MCA Parameter

In the MPI_BCAST routine, specify 1 to the MCA parameter coll_tuned_bcast_same_count in order to enable segment division. For the MCA parameter coll_tuned_bcast_same_count, see "Table 4.14 coll_tuned_bcast_same_count (achieves faster communication when MPI_BCAST/MPI_IBCAST routines are used with the same count among the processes)".

```
$ mpiexec --mca coll_tuned_bcast_same_count 1 \
--mca coll_select_bcast_algorithm binomial \
--mca coll_select_bcast_algorithm_segmentsize 1024  ./a.out
```

8.2.1.2 Notes by Changing Segment Size

By changing the size, the number of transfers of pipeline transfer changes. In particular, by decreasing the segment size, the number of transfers of pipeline transfer increases. As a result of the increased number of pipeline transfers, an abnormal end may occur due to the lack of communication resources. The abnormal end caused by the lack of communication resources outputs the error message beginning with the following character string.

```
[mpi::common-tofu::tofu-mrq-overflow]
```

When the above error message appears by changing the segment size, increase the segment size.

The upper limit of a segment size is 16776960. See "8.4.3 MCA Parameter to Tune Algorithm Itself" for changing the segment size.

8.3 Tuning by Algorithm Selection

This system provides the function to enable users to specify algorithms. This function is the function to allow changing of the performance of blocking collective communication. This section describes how to select algorithms, how to confirm, and the notes.

Selecting algorithms changes the performance of blocking collective communication. However, it does not necessarily enhance the performance. Use at the responsibility of users.

8.3.1 How to Select Algorithms

You can select algorithms by repeating the following two processes.

1. Select candidate algorithms

   Number 1 selects appropriate candidate algorithms from among multiple algorithms according to the conditions.

2. Determine whether or not the algorithms are call ready

   Number 2 determines whether or not there is any problem in executing the candidate algorithms selected in the above 1. Some algorithms may not have judgment conditions.

First of all, this section describes the flow of selecting algorithms in this system. Next, the section describes multiple methods of selecting algorithms appearing during the flow. Finally, the section describes how to determine whether or not the algorithms are call ready.

8.3.1.1 Flow of Selecting Algorithms

First of all, this section describes the flow of selecting algorithms in this system.

There are five methods of selecting algorithms. They perform algorithm selection in the order of high priority as shown in "Table 8.2 Flow of Selecting Algorithms".

If any of the following conditions is satisfied, the method of selecting algorithms specified by the user cannot be applied.

1. The size of the communicator is 1.
2. The communicator is created by using the MPI_INTERCOMM_MERGE routine.
3. The communicator is the inter-communicator.
4. A blocking collective communication whose algorithm is specified by the user is performed as internal processing in another blocking collective communication.

Also, there are three methods of selecting algorithms that the users can specify. However, the blocking collective communication routine given below has only one type of algorithm and therefore you cannot specify the algorithm.

- MPI_ALLTOALLW
- MPI_EXSCAN
- MPI_REDUCE_SCATTER_BLOCK

Table 8.2 Flow of Selecting Algorithms

Priority	How to Select	Outline	User can Specify	Reference of Details
1	Selection of special case	This system automatically selects if specific conditions are satisfied.	No	"8.3.1.2 Flow of Selecting Algorithms in Special Case"
2	Specification by MCA parameter	Specify algorithms by specifying the MCA parameter coll_select_(*type of collective communication*)_algorithm.	Yes	"8.3.1.3 Selecting Algorithms by MCA Parameter"
3	Specification by Info object	Specify algorithms by modifying the program and using the Info object. There are two types of methods for specifying the Info object.	Yes	"8.3.1.4 Selecting Algorithms by Info Object"
4	Specification by external input file	Create the rules of selecting algorithms in the external input file and specify the algorithms by specifying the MCA parameter at execution.	Yes	"8.3.1.5 Selecting Algorithms by External Input File"
5	Selection by this system	Automatically selected by this system.	No	None

If not specifying the method of selecting algorithms that the users can specify, automatic algorithm selection by this system is used. In this case, it does not necessarily select algorithms optimum. Therefore, using methods of selecting algorithms that the users can specify may improve performance.

The following table lists the advantages and disadvantages of the three types of algorithm selection methods that the users can specify. The following section describes the details.

Table 8.3 Advantages and Disadvantages of Algorithm Selection Method that Users can Specify

How to Select	Advantage	Disadvantage
Specification by MCA parameter	Impossible to be retranslated.	The specified algorithms apply to the entire application program.
Algorithm specification by Info object	Possible to specify algorithms for each collective communication routine.	Necessary to modify and retranslate the source file.
Algorithm specification by external input file	Not necessary to retranslate. Possible to specify algorithms for each condition.	Necessary to create the input file. Therefore, it may take time to create the input file.

8.3.1.2 Flow of Selecting Algorithms in Special Case

This selection method performs selection preferentially in the case of special conditions rather than other selection methods. Special cases refer to any of the following three.

1. Barrier communication is selectable
2. The reduction operations is non-commutative (if any of the following conditions is satisfied)
    a. User specifies the operation as non-commutative when creating the user definition operation
    b. User specifies 1 to the MCA parameter coll_base_reduce_commute_safe
3. MPI_IN_PLACE is specified to the send buffer in the MPI_ALLTOALL routine and the MPI_ALLTOALLV routine

If any of the above conditions is satisfied, this system performs the algorithm selection that has been implemented in advance in the system. In the case of condition 1, the system always selects the algorithm by barrier communication. For the conditions required for the application of barrier communication, see "6.12 Use of Tofu Barrier Communication for Better Performance". If any of the conditions of above 2 is satisfied, this system selects special algorithms. For the MCA parameter coll_base_reduce_commute_safe, see "Table 4.7 coll_base_reduce_commute_safe (guarantees the reduction operation sequence)". In the case of condition 3, the system selects the algorithm that corresponds to the MPI_IN_PLACE.

8.3.1.3 Selecting Algorithms by MCA Parameter

This selection method is effective when the blocking collective communication routine is always called by the same argument. When using the algorithm selection by the MCA parameter, all algorithms of certain collective communication become the algorithms that are specified by the MCA parameter. This selection method does not require retranslation of the application program.

Example

Example of Selecting Algorithms by MCA Parameter

```
$ mpiexec --mca coll_select_allgather_algorithm bruck ./a.out
```

This example always selects the bruck algorithm as the algorithm of the MPI_ALLGATHER routine. For the values that can be specified as the MCA parameter, see "8.4.1 MCA Parameter to Specify Algorithm Selection".

8.3.1.4 Selecting Algorithms by Info Object

This system supports the method of selecting algorithms by using the Info object. To enable the tuning of the blocking collective communication routine with this selection method, you need to modify the application program to use in the following procedure. The modified application program requires retranslation.

1. Create Info object.
2. Set key and value to the Info object.
3. Set the Info object to the communicator.
4. For the blocking collective communication routine to which you want tuning, specify as the argument the communicator you set in above 3.
5. Release the Info object.

Example

Example of Pseudo-code of Algorithm Selection by Info Object

This example shows the pseudo-code of algorithm selection by Info object. Rows from the second row correspond to the above 1 to 5.

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, key, value);
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm);
MPI_Info_free(&info);
```

You can specify key and value for the Info object. In the algorithm selection, key allows you to set the name of the blocking collective communication routine to which you want to specify the algorithm and the usage range. With value, you can set the name of the algorithm to specify and the range of the message size to use such algorithm. Also, it is possible to specify the segment size for value in specific algorithm.

For the character strings to specify with key, see "8.3.1.4.1 Parameter to Specify with key of Info Object". For the character strings to specify with value, see "8.3.1.4.4 Specifying Rules for Selecting Algorithms".

8.3.1.4.1 Parameter to Specify with key of Info Object

To set the usage range of the rules for the algorithm selection, you need to specify key to pass to the Info object. The types of usage range specified with key are two. They specify the algorithm for each collective communication routine call and for each communicator.

8.3.1.4.2 Specification for Each Collective Communication Routine Call

Specify (type of collective communication)_oneshot_rule when using a specific algorithm only once at any of collective communication routine that can be specified in the external input file. See "Table 8.9 Type of Collective Communication that can be Specified in External Input File" for specifying type of collective communication.

Example

Example of Specifying Algorithms for Each Collective Communication Routine

This example shows the pseudo-code of changing algorithm only once at collective communication routine.

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "allgather_oneshot_rule", "bruck");
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm); //(1)
MPI_Allgather(..., comm); //(2)
MPI_Info_free(&info);
```

(*type of collective communication*)_oneshot_rule is the selection method of an algorithm which is effective only for the collective communication routine specified immediately after the MPI_COMM_SET_INFO routine. This example uses the bruck algorithm in the MPI_ALLGATHER routine of (1) within the figure. However, after (1) is executed, the specification of (*type of collective communication*)_oneshot_rule becomes invalid. Therefore, the MPI_ALLGATHER routine of (2) performs the algorithm selection by different means.

For specifiable key, see "Table 8.5 Key When Changing Collective Communication Routine Algorithm Only Once".

8.3.1.4.3 Specification for Each Communicator

Specify (type of collective communication)_rule when changing algorithm in the unit of communicator. See "Table 8.9 Type of Collective Communication that can be Specified in External Input File" for specifying type of collective communication.

Example of Specifying Algorithms for Each Communicator by Using Info Object

This example shows the pseudo-code of the method to change the algorithm of the collective communication routine to the unit of communicator.

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "allgather_rules", "bruck");
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm); //(1)
MPI_Allgather(..., comm); //(2)
MPI_Info_free(&info);
```

This example executes the bruck algorithm in the MPI_ALLGATHER routine of (1) within the figure. Unlike (*type of collective communication*)_oneshot_rule, it executes the bruck algorithm even in the MPI_ALLGATHER routine of (2).

For specifiable key, see "Table 8.6 Key When Changing Collective Communication Routine Algorithm in Unit of Communicator".

8.3.1.4.4 Specifying Rules for Selecting Algorithms

The section describes how to specify the conditions to call algorithms with value. There are the following two types of specifying the conditions to call algorithms.

1. (*Algorithm name*)

   The specified algorithm is always called regardless of the message size.

2. (*Range of message size*): (*Algorithm name*)

   The specified algorithm is called to the extent of the specified message size.

For this specification, note the following.

- When (*type of collective communication*)_oneshot_rule is specified to key, even if you specify multiple algorithms to value, only the first algorithm becomes enabled. See "Table 8.9 Type of Collective Communication that can be Specified in External Input File" for specifying type of collective communication.

The algorithm name that can be specified is the same as the character strings that can be specified with the MCA parameter. For details, see "8.4.1 MCA Parameter to Specify Algorithm Selection".

Some algorithms may set the parameter. The parameter that can be specified currently is the segment size only. For the segment size and the specifiable algorithm, see "8.2 MCA Parameter Tuning of Algorithms". To specify the segment size, specify value as shown in below.

Example of Specifying Segment Size of Algorithms by Using Info Object

This is an example of a pseudo-code with a segment size to an algorithm that can specify the segment size.

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "bcast_rules", "chain(segsize=2048)");
MPI_Comm_set_info(comm, info);
```

```
MPI_Bcast(..., comm); //Chain algorithm is executed with the segment size 2048 bytes.
MPI_Info_free(&info);
```

Example

Example of Specifying Message Size of Algorithms by Using Info Object

This is an example of pseudo-code that specified the message size.

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "bcast_rules", "1024:chain;2048-4096:binomial");
MPI_Comm_set_info(comm, info);
MPI_Bcast(..., comm); //MPI_BCAST routine with the message size of 1024 bytes and the chain
algorithm is executed.
MPI_Bcast(..., comm); //MPI_BCAST routine with the message size of 3072 bytes and the binomial
algorithm is executed.
MPI_Info_free(&info);
```

This example uses chain algorithm when the message size (count * datatype size) is 1,024 bytes. Also, the example uses a binomial algorithm when the size is 2,048 bytes or more and 4,096 bytes or less. When the message size is other than those, the example selects the algorithm in a different way.

This section describes the definition of the message size that can be specified to value. The definition of the message size is in accordance with "Table 8.4 Definition of Message Size by Blocking Collective Communication Routine to Be Used for Info Object" shown below. However, as the message size is not used or the message size cannot be defined between ranks for the collective communication routine indicated as non-usable, you cannot specify the message size to value. Any message size specified in the applicable collective communication routine shall be disabled.

Table 8.4 Definition of Message Size by Blocking Collective Communication Routine to Be Used for Info Object

Collective Communication Routine Name	Definition of Message Size
MPI_ALLGATHER	Count * datatype size * communicator size
MPI_ALLGATHERV	Sum of count * datatype size
MPI_ALLREDUCE	Count * datatype size
MPI_ALLTOALL	Count * datatype size * communicator size
MPI_ALLTOALLV	Non-usable
MPI_BARRIER	Non-usable
MPI_BCAST	Count * datatype size
MPI_GATHER	Send count * datatype size * communicator size
MPI_GATHERV	Sum of receive count of root rank * datatype size
MPI_REDUCE	Count * datatype size
MPI_REDUCE_SCATTER	Count * datatype size * communicator size
MPI_SCAN	Count * datatype size
MPI_SCATTER	Receive count * datatype size * communicator size
MPI_SCATTERV	Sum of send count of root rank * datatype size

Next, the section describes how to change algorithm selection for each message size. There are two types of methods for specifying the message size.

1. Specific message size

   Select the specified algorithm only for specific message size.

2. msgmin - msgmax

   Select the specified algorithm if the message size is msgmin bytes or more and msgmax bytes or less.

To select multiple algorithms, enter value as in "Example of Algorithms Selection to Specify Message Size by Info Object". Enter into value the character string delimiting the conditions to call algorithms with ";".

Example of Algorithms Selection to Specify Message Size by Info Object

```
value = "(condition 1 to call algorithm); ... ;(condition N to call algorithm)"
```

8.3.1.4.5 Notes for Selecting Algorithms by Info Object

This section describes notes for selecting algorithms by Info object.

- Specify key and value that are equal at all processes of the communicator to the Info object.
- The character strings that can be entered as value shall be up to the length of character strings defined as MPI_MAX_INFO_VAL. This system allows the length under 255 one-byte alphabet characters and numbers.
- As for (*type of collective communication*)_oneshot_rule to change the algorithms only once, this system deletes key and value of (*type of collective communication*)_oneshot_rule after use. Therefore, if you want to use (*type of collective communication*)_oneshot_rule again, establish Info object to the communicator again. See "Table 8.9 Type of Collective Communication that can be Specified in External Input File" for specifying type of collective communication.
- In MPI_GATHERV routine and MPI_SCATTERV routine, this system performs the processing equivalent to MPI_BCAST routine inside in order to get the message size. Therefore, it takes time to execute.

8.3.1.4.6 List of Values that can be Specified to Key of Info Object

This section describes key that can be specified to Info object.

Table 8.5 Key When Changing Collective Communication Routine Algorithm Only Once

key	Meaning
allgather_oneshot_rule	Fix the algorithm of the MPI_ALLGATHER routine only once.
allgatherv_oneshot_rule	Fix the algorithm of the MPI_ALLGATHERV routine only once.
allreduce_oneshot_rule	Fix the algorithm of the MPI_ALLREDUCE routine only once.
alltoall_oneshot_rule	Fix the algorithm of the MPI_ALLTOALL routine only once.
alltoallv_oneshot_rule	Fix the algorithm of the MPI_ALLTOALLV routine only once.
barrier_oneshot_rule	Fix the algorithm of the MPI_BARRIER routine only once.
bcast_oneshot_rule	Fix the algorithm of the MPI_BCAST routine only once.
gather_oneshot_rule	Fix the algorithm of the MPI_GATHER routine only once.
gatherv_oneshot_rule	Fix the algorithm of the MPI_GATHERV routine only once.
reduce_oneshot_rule	Fix the algorithm of the MPI_REDUCE routine only once.
reduce_scatter_oneshot_rule	Fix the algorithm of the MPI_REDUCE_SCATTER routine only once.
scan_oneshot_rule	Fix the algorithm of the MPI_SCAN routine only once.
scatter_oneshot_rule	Fix the algorithm of the MPI_SCATTER routine only once.
scatterv_oneshot_rule	Fix the algorithm of the MPI_SCATTERV routine only once.

Table 8.6 Key When Changing Collective Communication Routine Algorithm in Unit of Communicator

key	Meaning
allgather_rules	Fix the algorithm of the MPI_ALLGATHER routine in the unit of communicator.
allgatherv_rules	Fix the algorithm of the MPI_ALLGATHER routine in the unit of communicator.
allreduce_rules	Fix the algorithm of the MPI_ALLREDUCE routine in the unit of communicator.

key	Meaning
alltoall_rules	Fix the algorithm of the MPI_ALLTOALL routine in the unit of communicator.
alltoallv_rules	Fix the algorithm of the MPI_ALLTOALLV routine in the unit of communicator.
barrier_rules	Fix the algorithm of the MPI_BARRIER routine in the unit of communicator.
bcast_rules	Fix the algorithm of the MPI_BCAST routine in the unit of communicator.
gather_rules	Fix the algorithm of the MPI_GATHER routine in the unit of communicator.
gatherv_rules	Fix the algorithm of the MPI_GATHERV routine in the unit of communicator.
reduce_rules	Fix the algorithm of the MPI_REDUCE routine in the unit of communicator.
reduce_scatter_rules	Fix the algorithm of the MPI_REDUCE_SCATTER routine in the unit of communicator.
scan_rules	Fix the algorithm of the MPI_SCAN routine in the unit of communicator.
scatter_rules	Fix the algorithm of the MPI_SCATTER routine in the unit of communicator.
scatterv_rules	Fix the algorithm of the MPI_SCATTERV routine in the unit of communicator.

8.3.1.5 Selecting Algorithms by External Input File

This system supports selecting algorithms by external input file. The user makes an entry in the external input file in accordance with "8.3.1.5.5 Entries of External Input File". This section describes how to use external input file, three examples of external input file, entries of external input file, and notes of external input file.

8.3.1.5.1 How to Use

You can perform the rules to select algorithms entered in the external input file by specifying in the MCA parameter coll_select_dectree_file. For details of the MCA parameter, see "8.4.2.1 coll_select_dectree_file (Specification of External Input User Definition File of Algorithm Selection)".

8.3.1.5.2 Example of External Input File

This section describes a basic example of making an entry in the external input file.

Basic External Input File

The external input file consists of the header and the collective communication rules. Be sure to enter the header at the top of the external input file.

Example of Simple External Input File with Algorithm for MPI_GATHER Routine

```
header:                    <--+
  version: 1.0                 | Header
  require: mtofu           <--+

gather: simple             <--- Collective communication
```

The header portion indicates that this file is the file for algorithm selection.

Be sure to enter these three rows at the top when creating the file for algorithm selection rules. The version number of the function to analyze the rules entered in the external input file is version 1.0. Specify the component information in require in order to declare selecting the algorithms that exist in which component. "Example of Simple External Input File with Algorithm for MPI_GATHER Routine" enters mtofu to require because it uses the algorithm tuned for Tofu interconnect. When using the algorithms of Open MPI, enter base. When using both, enter mtofu, base.

For the collective communication rules, enter the collective communication to which the algorithm is specified and the algorithm to specify. For the collective communication with no entry made, this system selects the algorithm. For example, to fix in one algorithm, enter as follows.

```
(Type of collective communication) : (Algorithm name)

Or
```

```
(Type of collective communication) :
    (Algorithm name)
```

To use multiple algorithms separately, see explanation of "8.3.1.5.3 Multiple Entries of Algorithms and Parameter".

In "Example of Simple External Input File with Algorithm for MPI_GATHER Routine", the MPI_GATHER routine always attempts to call the simple algorithm. If the condition to call the simple algorithm is not satisfied, it selects the algorithm by the selection by this system.

8.3.1.5.3 Multiple Entries of Algorithms and Parameter

This section describes multiple entries of algorithms and parameter. As is described in "8.3.1 How to Select Algorithms", there may be conditions to call some algorithms and therefore it does not always select the expected algorithms. As such, you can enter by changing the priority of algorithms to call even under the same condition.

Example of Multiple Entries of Algorithms

```
header:
  version: 1.0
  require: mtofu, base

gather: simple, binomial
```

This example specifies the algorithm of the MPI_GATHER routine. This example attempts to call the simple algorithm at first. If the simple algorithm cannot be called, the binomial algorithm will be selected. In this way, it judges sequentially from the algorithm entered in the left side. If any of the entered algorithms cannot be selected, it selects the algorithm by the selection by this system.

Also, some algorithms may be set with the parameter. The parameter that can be specified currently is the segment size only. The example is the selection method even to specify the segment size of the MPI_BCAST routine. For the segment size and the specifiable algorithm, see "8.2 MCA Parameter Tuning of Algorithms".

Example of Specifying Parameter of Algorithms

```
header:
  version: 1.0
  require:base

bcast: binomial(segsize=1024)
```

If the segment size is not specified, it is treated as if 0 is specified. If 0 is specified, it performs communication without dividing the messages. This example executes the binomial algorithm with the segment size as 1,024 bytes.

8.3.1.5.4 Conditional Statement

In some cases you may want to change the algorithm to select according to conditions in the collective communication rules of the external input file. This section describes how to write the usable conditional statement in such a case. The conditional statements are divided into the following two types.

1. When entering two conditions by using if and else
2. When entering multiple conditions by using case and match

Firstly, this section describes the conditional statements using if. Use the if statement in the following way. else: portion is always necessary.

Outline of if Statement

```
  if (conditional expression) :
    Statement
  else :
    Statement
```

Example of Using if Statement

```
header:
  version: 1.0
  require: base
```

```
allreduce:
  if msg_size <= 2097152:
    recursive_doubling
  else:
    ring
```

This example of the MPI_ALLREDUCE routine changes the algorithm selection depending on the message size as shown below.

- When the message size is 2,097,152 bytes or less: Execute the recursive_doubling algorithm
- When the message size is over 2,097,152 bytes: Execute the ring algorithm

Next, the section describes the example of using case and match.

Outline of case and match

```
    case (expression):
        match A .. B:
             Statement 1
        match C .. D:
             Statement 2
        match _:
             Statement 3
```

The above example performs Statement 1 if the expression is A or more and B or less, and performs Statement 2 if the expression is C or more and D or less. When the expression is not applicable to any range, then it performs Statement 3. It does not matter even if three rows or more of match (range) exist. However, you need to enter match _: at last.

Example of Using case and match

```
header:
  version: 1.0
  require: mtofu, base

bcast:
  case msg_size:
    match 1 .. 65536
      split_binary_tree
    match 65537 .. 1048576
      bintree3d(segsize = 2048)
    match _:
      bintree3d(segsize = 16384)
```

In this example of the MPI_BCAST routine, the selection of algorithm changes depending on the message size as shown below.

- When the message size is 1 byte or more and 65,536 bytes or less: Execute the split_binary_tree algorithm.
- When the message size is 65,537 bytes or more and 1,048,576 bytes or less: Execute the bintree3d algorithm with the segment size of 2,048 bytes.
- When the message size is other than above: Execute the bintree3d algorithm with the segment size of 16,384 bytes.

The case statement is evaluated from the front. Therefore, if there are multiple match statements that correspond to the expression, the preceding conditions take priority.

8.3.1.5.5 Entries of External Input File

This section describes the entered content in the external input file.

As is explained in "Basic External Input File", enter the "head portion" and the "collective communication rules" in the external input file.

Outline of External Input File

```
header:
    version: 1.0
    (Component information)
(Collective communication rules)
```

The header portion consists of the character string "header", ":", version information, and component information. The version information consists of "version", ":", and version number. The version number of the function to analyze the rules entered in the external input file is version 1.0.

- Use space between words and symbols.
- Describe comment with #. All characters following # will be recognized as without meaning.
- You can use indentation using one-byte spaces, too. However, a syntax error occurs when the indention level is wrong such as the example below.

```
header:
    version: 1.0
    require: base

allreduce:
    if msg_size <= 2097152:
        recursive_doubling
# A syntax error occurs because the indentation level of the following
# else statement line does not match that of the corresponding
# if statement line.
# In this case, modify the indentation level of the else statement so that
# it matches the indentation level of the corresponding if statement line.
  else:
        ring
```

For detailed example, see "8.3.1.5.2 Example of External Input File".

The table below shows the items that can be entered in the external input file. The symbols in the table below are given as follows:

Table 8.7 Symbols Written in Entry Method and Meanings

Symbol	Meaning
" "	For the items embedded in these symbols, the user creates selection conditions in accordance with the format written in the entry method.
[ ]	Perform the content embedded with [ ] once.
[ ]+	Repeat the content embedded with [ ] once or more.
|	Select one item from the items separated by this symbol.

The item names embedded with " " as shown in the entry method of the table below indicate that the entry method of the item name applies.

For example, in the case of the header item, the entry method is "header: 'Version information' 'Component information'". Therefore, enter in the external input file the content pursuant to the format of the entry method of the version information and component information after "header:" as shown in the example of "Outline of External Input File".

Table 8.8 Definition of External Input File

External Input File	Item	Entry Method	Remarks
Header portion	Header	header: "Version information" "Component information"	
	Version information	version: "Version number"	
	Version number	1.0	

External Input File	Item	Entry Method	Remarks
	Component information	require: (Component location)	
	Component location	mtofu | base | mtofu, base	
Collective communication rules	Collective communication rule	["Type of collective communication"]: "Text"	
	Type of collective communication	See "Table 8.9 Type of Collective Communication that can be Specified in External Input File".	
	Text	"if-else statement" | "case-match statement" | "Algorithm statement"	
	if-else statement	if " comparative sentence " : "text" else : "text"	
	case-match statement	case "expression" : [match "range" :"text"]+ match _: "text"	

	Range	"positive integer of lower limit".." positive integer of upper limit"
		| "positive decimal of lower limit.."positive decimal of upper limit"
			Specify the range required by the expression with case-match statement. The meaning of the range varies depending on the expression.
			When the expression indicates the message size, specify as follows:
			Ex: When specifying the rage from 1,024 to 2,048 bytes
			`case msg_size:`
			`  match 1024..2048:`
			When the expression is sharpness, specify the dimension ratio of the axis in Tofu coordinates as follows:
			Ex: When indicating the range from 1.0 to 2.0
			`case sharpness:`
			`  match 1.0..2.0:`

	Algorithm statement	"Algorithm name" | "Algorithm name" , "Algorithm statement" | "Algorithm name" (segsize = "positive integer") | "Algorithm name" (segsize = "positive integer"), "Algorithm statement" | "Algorithm name" (segsize = "positive integer" , "positive integer")	Specify one segsize in principle. However, trinaryx3 and trinaryx6 which are the algorithms of the MPI_ALLREDUCE routine call reduce and bcast inside. Therefore, you can enter two segsize. Ex:

External Input File	Item	Entry Method	Remarks
		| "Algorithm name" (segsize = "positive integer" , "positive integer"), "Algorithm statement"	`trinaryx3(segsize=4096, 2048)` The segment size in this example is 4,096 bytes for reduce and 2,048 bytes for bcast. When specifying only one segsize, the segment size of reduce and bcast are the same.
	Comparison statement	"bool-type variable" | "variable" is even_num | "variable" is odd_num | "expression" "comparative operator" "term" | "term" "comparative operator" "expression"	When specifying only variables in the comparison statement, specify only the bool-type variables. For "variable" to be specified before "is even_num" or "is odd_num", bool type and double type cannot be specified. For "variable" to be specified as term, bool type cannot be specified.
	Comparative operator	<= | < | >= | > | == | !=	
	Term	"positive integer" | "positive decimal" | "variable"	For "variable", bool type cannot be specified.
	Comparative operator	+ | - | * | /	
	Expression	"term" | "term" "operator" "term" | "term" "operator" "term" "operator" "term" |("term" "operator" "term") "operator" "term" | "term" "operator" ("term" "operator" "term")	The expression consists of terms, operators, and brackets. Three terms can be entered at maximum. Two operators can be entered at maximum. One pairs of brackets can be entered at maximum. Ex: `(proc_count + 1024) * msg_size`
	Algorithm name	See the algorithms described in the table of "8.4.1 MCA Parameter to Specify Algorithm Selection".	
	Variable	See "Table 8.10 Variable Names that can be Specified in External Input File".	
	bool type variable	log_cuboid | pow_two	See the meaning described in "Table 8.10 Variable Names that can be Specified in External Input File".

External Input File	Item	Entry Method	Remarks
		| 6d_cuboid | equal_proc	

The table below shows the type of collective communication that can be specified in the external input file.

Table 8.9 Type of Collective Communication that can be Specified in External Input File

Type of Collective Communication that can be Specified in External Input File	Corresponding Blocking Collective Communication Routine
allgather	MPI_ALLGATHER
allgatherv	MPI_ALLGATHERV
allreduce	MPI_ALLREDUCE
alltoall	MPI_ALLTOALL
alltoallv	MPI_ALLTOALLV
barrier	MPI_BARRIER
bcast	MPI_BCAST
gather	MPI_GATHER
gatherv	MPI_GATHERV
reduce	MPI_REDUCE
reduce_scatter	MPI_REDUCE_SCATTER
scan	MPI_SCAN
scatter	MPI_SCATTER
scatterv	MPI_SCATTERV

The table below shows the variable names that can be specified as variables in the external input file. The table below also shows the datatype corresponding to the variable names. Note that the variables that can be specified as the variables of bool type in the comparison statement are the following four types only (log_cuboid, pow_two, 6d_cuboid, qual_proc).

Regarding the message size of the collective communication of the MPI_GATHERV and MPI_SCATTERV (total_msg_size), this system performs the processing equivalent to MPI_BCAST routine inside in order to get the message size. Therefore, it takes time to execute.

Table 8.10 Variable Names that can be Specified in External Input File

Variable Name	Datatype	Meaning
node_count	int32_t	Number of nodes in the communicator
proc_count	int32_t	Number of processes in the communicator
max_proc	int32_t	Maximum number of processes in the nodes in the communicator
log_cuboid	bool	Whether the shape of the nodes in the communicator is cuboid or not (allow missing specific logical axis).
x_len, y_len, z_len	int32_t	Each length of logical axis of the nodes to which processes in the communicator belong.
comm_dim	int32_t	Value based on the dimension number of the node shape where processes in the communicator exist. Value is decided under the following conditions. comm_dim=3: When log_cuboid is true and x_len, y_len, and z_len are all 2 or more comm_dim=2: When log_cuboid is true and two of x_len, y_len, and z_len are 2 or more comm_dim=1: When log_cuboid is true and one of x_len, y_len, and z_len are 2 or more comm_dim=0: When log_cuboid is false, or x_len, y_len, and z_len are all 1 or less

Variable Name	Datatype	Meaning
msg_count	int32_t	Count of its own process. Can be used only for the MPI_ALLREDUCE routine.
msg_size	int64_t	Message size of the collective communication. For the definition of the message size, see "Table 8.11 Definition of msg_size for Each Blocking Collective Communication Routine".
total_msg_size	int64_t	Sum of the message size of the collective communication. Use when it is not possible to represent simply with msg_size * proc_count. Specifically, it shows the value of the following collective communication routine. - Receive buffer size of the MPI_ALLGATHERV routine - Send buffer of the MPI_REDUCE_SCATTER routine - Receive buffer size of root rank of the MPI_GATHERV routine - Send buffer size of root rank of the MPI_SCATTERV routine
pow_two	bool	Whether the number of processes in the communicator is as a power of 2 or not.
sharpness	double	Sharpness (ratio of the dimension of the physical maximum axis in the communicator and the remaining two axes). It represents the ratio of the dimensions of the maximum axis and the remaining axis for X axis, Y axis, and Z axis in the Tofu coordinates in the shape of containing the communicator.
6d_cuboid	bool	Whether the shape of the nodes in the communicator is cuboid or not in the Tofu coordinates (allow missing specific logical axis).
equal_proc	bool	Whether the number of processes in the node is even or not.
job_topo	int32_t	Type of method of placing nodes at Tofu coordinates. Method of placing nodes at the time of job execution can be specified using this variable. If this variable is not specified, the job is executed according to the method of placing nodes by Job Operation Software. 0: Torus mode 1: Mesh mode 2: Non-contiguous mode
tni_count	int32_t	Number of TNI that can be used at each process.
world_size	int32_t	Number of processes for MPI_COMM_WORLD.

For the node shape for inside the communicator, see "6.14 Algorithms of Collective Communication and Shape of Compute Nodes Allocated to the Communicator". In addition, see Job Operation Software manual for details of methods of placing nodes at Tofu coordinates.

Table 8.11 Definition of msg_size for Each Blocking Collective Communication Routine

Collective Communication Routine Name	Definition of Message Size
MPI_ALLGATHER	Count * datatype size
MPI_ALLGATHERV	Non-usable
MPI_ALLREDUCE	Count * datatype size
MPI_ALLTOALL	Count * datatype size
MPI_ALLTOALLV	Non-usable
MPI_BARRIER	Non-usable
MPI_BCAST	Count * datatype size
MPI_GATHER	Send count * datatype size
MPI_GATHERV	Non-usable
MPI_REDUCE	Count * datatype size
MPI_REDUCE_SCATTER	Non-usable

Collective Communication Routine Name	Definition of Message Size
MPI_SCAN	Count * datatype size
MPI_SCATTER	Receive count * datatype size
MPI_SCATTERV	Non-usable

The table below shows "is" that can be specified in comparative statement of the external input file.

Table 8.12 "is" that can be Specified in Comparison Statement of External Input File

Specification	Meaning
Expression is even_num	Return true when expression value is even number and return false if not.
Expression is odd_num	Return true when expression value is odd number and return false if not.

8.3.1.5.6 Notes for Specifying External Input File

This section describes notes for specifying the external input file.

Conditions to be Noted	Notes

number of entreies that can be made in the external input file	There is an upper limit for the number of entries that can be made in the external input file. The number of entries is decided according to the total number of entered algorithms and the number of entered conditional statements. The expression to obtain the number of entries is given below.
	`Number of entries = total number of entered algorithms + number of if statements + sum of case and match written with case-match statement`
	The upper limit on the number of entries that can be made in this system is approximately 3,500. When the number of entries exceeds the upper limit, the following warning is output if 1 or 2 is specified in the MCA parameter coll_select_show_decision_process.
	`[mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [2]`

number of conditional statements that can be made in the external input file	There is an upper limit for the number of conditional statements that can be made in the external input file, and the upper limit is 30. The number of conditional statements is sum of the number of if statements and matches of case-match statement. When the number of entries exceeds the upper limit, the following warning is output if 1 or 2 is specified in the MCA parameter coll_select_show_decision_process. `[mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [3]`

8.3.1.6 Conditions Required for Application of Algorithms

This section describes the characteristics of algorithms and conditions required for the application. When executing the algorithms, it may determine according to the requirements such as the number of processes and communicator shape. This system determines whether it can actually call to execute the algorithms to be called at the time of execution. The determination is divided into the following two types.

1. It is possible to determine only by its own process
2. It is required to determine all processes that comprise the communicator.

Number 1 is used when it is possible to determine uniquely without process. It is, for example, the item depending on the communicator such as the number of processes.

Number 2 is used when the argument other than the communicator is the condition. It is, for example, the datatype. In MPI standard, most of the blocking collective communication is allowed to have different datatypes between ranks with conditions. As such, it is not possible to determine if the condition of datatype is satisfied only by the argument of its own process. For the conditions of argument, see MPI standard. To perform determination, you must select the same algorithm in the entire communicator. Therefore, it is necessary to make determination results agree with in the communicator. As such, when performing the determination, perform processing equivalent to the

MPI_ALLREDUCE routine inside this system. For the condition required for applying each algorithm, see "8.3.4 List of Algorithms and Conditions Required for Application".

8.3.2 How to Confirm Selection Results

This section describes the function to confirm whether the selected algorithm is executed as expected.

8.3.2.1 Displaying Selection Process of Algorithms

When 1 or 2 is specified in the MCA parameter coll_select_show_decision_process, the selection process of the algorithm is displayed in the standard error output. For the MCA parameter, see "8.4.2.2 coll_select_decision_process (Output of Algorithm Selection Process)". For the output message, see "8.5.2 Output Message by Display Function of Algorithm Selection Process (Warning)".

8.3.2.2 Obtaining Selection Results of Algorithms by Using Info Object

Just like the algorithm selection, you can obtain the selection results of algorithms by using the Info object. When obtaining the algorithm selection results, specify 1 to the MCA parameter coll_select_get_tuning_info. For the MCA parameter coll_select_get_tuning_info, see "8.4.4.1 coll_select_get_tuning_info (Obtaining the Algorithm Information of the Collective Communication Executed Immediately Before)".

Example

Pseudo-code of Algorithms Obtained by Using Info Object

```
char val[MPI_MAX_INFO_VAL];
int flag = 0;
MPI_Info info;
MPI_Allgather(..., comm)//(1)
MPI_Comm_get_info(comm, &info);
MPI_Info_get(info, "last_allgather_algorithm", MPI_MAX_INFO_VAL, val, &flag);
if(flag){//flag is 1, Info is obtained
    printf("val is %s\n",val);//(2)
}
MPI_Info_free(&info);
```

Example of Results for Obtaining Algorithm Number

```
val is 5(segsize=0)
val is 1
```

In "Pseudo-code of Algorithms Obtained by Using Info Object", the value output in the line (2) is the algorithm number corresponding to the algorithm used in the line (1).

Each line in "Example of Results for Obtaining Algorithm Number" is an exapmle of value output in the line (2) in "Pseudo-code of Algorithms Obtained by Using Info Object", For the algorithms to which the segment size can be set, the segment size is indicated as segsize as shown in the example below. In other cases, only the algorithm number is output.

"Table 8.13 List of key for Obtaining Results of Algorithm Selection" is the list of key to obtain with the MPI_INFO_GET routine.

Table 8.13 List of key for Obtaining Results of Algorithm Selection

key	Meaning
last_allgather_algorithm	Obtain the last used algorithm in the MPI_ALLGATHER routine.
last_allgatherv_algorithm	Obtain the last used algorithm in the MPI_ALLGATHERV routine.
last_allreduce_algorithm	Obtain the last used algorithm in the MPI_ALLREDUCE routine.
last_alltoall_algorithm	Obtain the last used algorithm in the MPI_ALLTOALL routine.
last_alltoallv_algorithm	Obtain the last used algorithm in the MPI_ALLTOALLV routine.
last_barrier_algorithm	Obtain the last used algorithm in the MPI_BARRIER routine.

key	Meaning
last_bcast_algorithm	Obtain the last used algorithm in the MPI_BCAST routine.
last_gather_algorithm	Obtain the last used algorithm in the MPI_GATHER routine.
last_gatherv_algorithm	Obtain the last used algorithm in the MPI_GATHERV routine.
last_reduce_algorithm	Obtain the last used algorithm in the MPI_REDUCE routine.
last_reduce_scatter_algorithm	Obtain the last used algorithm in the MPI_REDUCE_SCATTER routine.
last_scan_algorithm	Obtain the last used algorithm in the MPI_SCAN routine.
last_scatter_algorithm	Obtain the last used algorithm in the MPI_SCATTER routine.
last_scatterv_algorithm	Obtain the last used algorithm in the MPI_SCATTERV routine.

8.3.2.3 Obtaining Results Only with MCA Parameter

As a function of the MPI statistical information, you can display the algorithms executed with the blocking collective communication routine. For the MPI statistical information, see "6.16 MPI Statistical Information".

8.3.3 Notes for Selecting Algorithms

When thousands or more communications concentrate on one compute node at the same time, the Tofu network near that compute node becomes busy, thus significantly delaying the network processing. In some algorithms, multiple communications may concentrate on one process at the same time. Do not use such algorithms for the scale of thousands or more processes.

Also, if you continuously execute the collective communication routine that allows multiple communications to concentrate on one process, communication resources may be depleted. The abnormal end caused by the lack of communication resources outputs the error message beginning with the following character string.

```
[mpi::common-tofu::tofu-mrq-overflow]
```

As a result of continuous execution, if the above message is output, specify 2 to the MCA parameter common_tofu_memory_saving_method. However, this may decrease the execution performance. For the MCA parameter common_tofu_memory_saving_method, see "Table 4.25 common_tofu_memory_saving_method (changes the method used for the memory-saving communication mode)".

Barrier communication does not allow the user to select whether or not to use. This system selects as necessary.

When the selection method of algorithms is specified in the MPI_REDUCE routine, the order of operations is different from the case when the selection method of algorithms is not specified. As such, even when you execute with the same argument as in the case when the selection method of algorithms is not specified, the operation results may differ. According to the MPI standard, it is highly recommended that the MPI_REDUCE routine has the same result when it is executed by the same argument. Therefore, use the order guarantee mode so as not to change the order of operations. For the order guarantee mode, see "6.10 Suppressing Memory Usage".

8.3.4 List of Algorithms and Conditions Required for Application

This section describes the following information.

- Numbers indicated by the output function of algorithms that correspond to the MCA parameter
- Values of the specifiable MCA parameter as algorithm selection
- Conditions required for application of algorithms

Numbers indicated by the output function of algorithms have the following rules to follow.

Number	Contents
1 to 6	The algorithms published by the Open MPI
100 to 103	The algorithms tuned for CPU of Tofu interconnect or this system
200	The barrier communication

Number	Contents
300	The algorithms developed by this system

8.3.4.1 Algorithms Selected by MPI_ALLGATHER Routine

Table 8.14 Number of Algorithm in MPI_ALLGATHER Routine, Corresponding MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_allgather_algorithm	Conditions Required for Application
1	linear	None
2	bruck	None
3	recursive_doubling	pow_two is true.
4	ring	None
5	neighbor	proc_count is even.
6	two_proc	proc_count is 2.
100	gtbc	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true.
101	3dtorus, 3dtorus_fm	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true.
102	3dtorus_sm	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true. Number of processes in the node in the MPI_COMM_WORLD is 24 or less. proc_count * max_proc is less than 2048. msg_size is less than or equal to 16776960B.

Even if the 3dtorus_sm algorithm is specified, it may not be used while the MPI_REQUEST_FREE routine is called after calling the MPIX_ALLGATHER_INIT routine. In this case, another algorithm is selected by this software system.

For pow_two, proc_count, log_cuboid, max_proc, and msg_size, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.2 Algorithms Selected by MPI_ALLGATHERV Routine

Table 8.15 Number of Algorithm in MPI_ALLGATHERV Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_allgatherv_algorithm	Conditions Required for Application
1	default	None
2	bruck	None
3	ring	None
4	neighbor	proc_count is even.
5	two_proc	proc_count is 2.
100	gtvbc	Datatype is not the MPI_PACKED, but predefined datatype without space.

Number of Algorithm	MCA Parameter coll_select_allgatherv_algorithm	Conditions Required for Application
		log_cuboid is true. There is no space in the alignment of data received by the receive buffer.
101	3dtorus, 3dtorus_fm	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true.
102	3dtorus_sm	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true. Number of processes in the node in the MPI_COMM_WORLD is 24 or less. proc_count * max_proc is less than 2048. The product of the maximum number of receive counts and the size of receive datatype is 16776960 bytes or less.

Even if the 3dtorus_sm algorithm is specified, it may not be used while the MPI_REQUEST_FREE routine is called after calling the MPIX_ALLGATHER_INIT routine. In this case, another algorithm is selected by this software system.

For proc_count, log_cuboid, and max_proc, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.3 Algorithms Selected by MPI_ALLREDUCE Routine

Table 8.16 Number of Algorithm in MPI_ALLREDUCE Routine, Corresponding MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_allreduce_algorithm	Conditions Required for Application
1	basic_linear	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
2	nonoverlapping	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
3	recursive_doubling	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
4	ring	msg_count is greater than or equal to proc_count.
5	segmented_ring	msg_count is greater than or equal to proc_count. The segment size is greater than or equal to 1.

Number of Algorithm	MCA Parameter coll_select_allreduce_algorithm	Conditions Required for Application
		msg_count * datatype size is greater than the segment size * proc_count.
100	rdbc	Datatype is not the MPI_PACKED, but predefined datatype without space. Operation is predefined. log_cuboid is true. **Note** In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
101	trinaryx6, trix6	Datatype is not the MPI_PACKED, but predefined datatype without space. Operation is predefined. log_cuboid is true. tni_count is greater than or equal to comm_dim * 2. All of the following conditions about x_len, y_len, and z_len must be met. - x_len is greater than 3, or equal to either 1 or 3. - y_len is greater than 3, or equal to any of 0, 1, and 3. - z_len is greater than 3, or equal to any of 0, 1, and 3. **Note** In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
102	trinaryx3, trix3	Datatype is not the MPI_PACKED, but predefined datatype without space. Operation is predefined. log_cuboid is true. tni_count is greater than or equal to comm_dim. **Note** In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
200	None	The MPI_ALLREDUCE routine of the barrier communication can be called. (For the conditions to call barrier communication, see "6.12 Use of Tofu Barrier Communication for Better Performance".)

For msg_count, proc_count, log_cuboid, tni_count, comm_dim, x_len, y_len, and z_len, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.4 Algorithms Selected by MPI_ALLTOALL Routine

Table 8.17 Number of Algorithm in MPI_ALLTOALL Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_alltoall_algorithm	Conditions Required for Application
1	linear	proc_count is less than or equal to 1536. **Note** This algorithm may cause multiple communications to concentrate on one process at the same time. As a result, it may significantly delay the networking processing. Therefore, do not use it for the scale of thousands or more processes.
2	pairwise	None
3	modified_bruck	**Note** In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
4	linear_sync	None
5	two_proc	proc_count is 2.
100	doublespread	Datatype is not the MPI_PACKED, but predefined datatype without space.
101	blacc3d	Datatype is not the MPI_PACKED, but predefined datatype without space. log_cuboid is true. comm_dim of the communicator is 3. x_len, y_len, and z_len of the communicator are all even. msg_size is less than or equal to 33553920B. equal_proc is true.
102	blacc6d	Datatype is not the MPI_PACKED, but predefined datatype without space. 6d_cuboid is true. Shape of the communicator has no space in abc axis in Tofu coordinates. job_topo is not 2. equal_proc is true. tni_count is 6.

Number of Algorithm	MCA Parameter coll_select_alltoall_algorithm	Conditions Required for Application
		Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.
103	crp	Datatype is not the MPI_PACKED, but predefined datatype without space. node_count is 1 node.

For proc_count, log_cuboid, tni_count, comm_dim, x_len, y_len, z_len, msg_size, 6d_cuboid, equal_proc, job_topo, and node_count, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.5 Algorithms Selected by MPI_ALLTOALLV Routine

Table 8.18 Number of Algorithm in MPI_ALLTOALLV Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_alltoallv_algorithm	Conditions Required for Application
1	basic_linear	proc_count is less than or equal to 1536. Note This algorithm may cause multiple communications to concentrate on one process at the same time. As a result, it may significantly delay the networking processing. Therefore, do not use it for the scale of thousands or more processes.
2	pairwise	None
100	doublespread	Datatype is not the MPI_PACKED, but predefined datatype without space.

For proc_count, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.6 Algorithms Selected by MPI_BARRIER Routine

Table 8.19 Number of Algorithm in MPI_BARRIER Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_barrier_algorithm	Conditions Required for Application
1	linear	None
3	recursive_doubling	None
4	bruck	None
5	two_proc	proc_count is 2.
6	tree	None
200	None	The MPI_BARRIER routine of the barrier communication can be called. (For the conditions to call barrier communication, see "6.12 Use of Tofu Barrier Communication for Better Performance".)

For proc_count, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.7 Algorithms Selected by MPI_BCAST Routine

Table 8.20 Number of Algorithm in MPI_BCAST Routine, Corresponding to MCA Parameter and the Application Conditions

<table>
<tr><th>Number of Algorithm</th><th>MCA Parameter coll_select_bcast_algorithm</th><th>Conditions Required for Application</th></tr>
<tr><td>1</td><td>basic_linear</td><td rowspan="3">None

Note
If you continuously use these algorithms for thousands or more processes, communication resources may be depleted. For details, see "8.3.3 Notes for Selecting Algorithms".</td></tr>
<tr><td>2</td><td>chain</td></tr>
<tr><td>3</td><td>pipeline</td></tr>
<tr><td>4</td><td>split_binary_tree</td><td>Value of MCA parameter coll_tuned_bcast_same_count is 1.
Count (number of entries in buffer) is greater than or equal to 2.
The segment size is less than or equal to count / 2 *datatype size (if count is odd, the numbers after the decimal point of "count / 2" is rounded down.).</td></tr>
<tr><td>5</td><td>binary_tree</td><td>None</td></tr>
<tr><td>6</td><td>binomial</td><td>None</td></tr>
<tr><td>100</td><td>trinaryx6, trix6</td><td>Datatype is not the MPI_PACKED, but predefined datatype without space.
log_cuboid is true.
tni_count is greater than or equal to comm_dim * 2.
All of the following conditions about x_len, y_len, and z_len must be met.
- x_len is greater than 3, or equal to either 1 or 3.
- y_len is greater than 3, or equal to any of 0, 1, and 3.
- z_len is greater than 3, or equal to any of 0, 1, and 3.</td></tr>
<tr><td>101</td><td>bintree3d, bin3d</td><td>Datatype is not the MPI_PACKED, but predefined datatype without space.
log_cuboid is true.</td></tr>
<tr><td>102</td><td>trinaryx3, trix3</td><td>Datatype is not the MPI_PACKED, but predefined datatype without space.
log_cuboid is true.</td></tr>
<tr><td>103</td><td>bintree6d, bin6d</td><td>Datatype is not the MPI_PACKED, but predefined datatype without space.
log_cuboid is true.
tni_count is greater than or equal to comm_dim * 2.</td></tr>
<tr><td>200</td><td>None</td><td>The MPI_BCAST routine of the barrier communication can be called.
(For the conditions to call barrier communication, see "6.12 Use of Tofu Barrier Communication for Better Performance".)</td></tr>
</table>

For log_cuboid, tni_count, comm_dim, x_len, y_len, and z_len, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.8 Algorithms Selected by MPI_GATHER Routine

Table 8.21 Number of Algorithm in MPI_GATHER Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_gather_algorithm	Conditions Required for Application
1	basic_linear	None
2	binomial	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the receive buffer in root process is large.
3	linear_sync	None Note Algorithm to execute at high speed in the case of the same count between ranks. If the count is different between ranks, the executed an abnormal end may occur. Make sure that the count between ranks correspond and call the algorithm.
100	simple	Datatype is not the MPI_PACKED, but predefined datatype without space.

8.3.4.9 Algorithms Selected by MPI_GATHERV Routine

Table 8.22 Number of Algorithm in MPI_GATHERV Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_gatherv_algorithm	Conditions Required for Application
1	default	None
100	simple	Datatype is not the MPI_PACKED, but predefined datatype without space.

8.3.4.10 Algorithms Selected by MPI_REDUCE Routine

Table 8.23 Number of Algorithm in MPI_REDUCE Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_reduce_algorithm	Conditions Required for Application
1	linear	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
2	chain	

Number of Algorithm	MCA Parameter coll_select_reduce_algorithm	Conditions Required for Application
		Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
3	pipeline	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
4	binary	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
5	binomial	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
6	in-order_binary	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
100	trinaryx6, trix6	Datatype is not the MPI_PACKED, but predefined datatype without space. Operation is predefined. log_cuboid is true. tni_count is greater than or equal to comm_dim * 2. All of the following conditions about x_len, y_len, and z_len must be met. - x_len is greater than 3, or equal to either 1 or 3. - y_len is greater than 3, or equal to any of 0, 1, and 3. - z_len is greater than 3, or equal to any of 0, 1, and 3.

Number of Algorithm	MCA Parameter coll_select_reduce_algorithm	Conditions Required for Application
		Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
101	trinaryx3, trix3	Datatype is not the MPI_PACKED, but predefined datatype without space. Operation is predefined. log_cuboid is true. tni_count is greater than or equal to comm_dim. Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer in root process are large.
200	None	The MPI_REDUCE routine of the barrier communication can be called. (For the conditions to call barrier communication, see "6.12 Use of Tofu Barrier Communication for Better Performance".)

For log_cuboid, tni_count, comm_dim, x_len, y_len, and z_len, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.11 Algorithms Selected by MPI_REDUCE_SCATTER Routine

Table 8.24 Number of Algorithm in MPI_REDUCE_SCATTER Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_reduce_scatter_algorithm	Conditions Required for Application
1	non-overlapping	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer is large.
2	recursive_halving	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer is large.
3	ring	

Number of Algorithm	MCA Parameter coll_select_reduce_scatter_algorithm	Conditions Required for Application
		Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer is large.

8.3.4.12 Algorithms Selected by MPI_SCAN Routine

Table 8.25 Number of Algorithm in MPI_SCAN Routine, Corresponding to MCA Parameter and the Application Condition

Number of Algorithm	MCA Parameter coll_select_scan_algorithm	Conditions Required for Application
1	linear	None
2	recursive_doubling	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer and the receive buffer are large.

8.3.4.13 Algorithms Selected by MPI_SCATTER Routine

Table 8.26 Number of Algorithm in MPI_SCATTER Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_scatter_algorithm	Conditions Required for Application
1	basic_linear	None
2	binomial	Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer in root process is large.
300	use_bcast	msg_size * proc_count is less than or equal to 2147483647. Note In this algorithm, the MPI library allocates much memory for temporary use. Therefore, note that a shortage of memory may occur when the size of the send buffer in root process is large.

For msg_size and proc_count, see "Table 8.10 Variable Names that can be Specified in External Input File".

8.3.4.14 Algorithms Selected by MPI_SCATTERV Routine

Table 8.27 Number of Algorithm in MPI_SCATTERV Routine, Corresponding to MCA Parameter and the Application Conditions

Number of Algorithm	MCA Parameter coll_select_scatterv_algorithm	Conditions Required for Application
1	basic_linear	None
300	linear_sync	None

8.4 MCA Parameter related to Algorithm Selection

This section describes types of the MCA parameter related to the algorithm selection. For the specification method of the MCA parameter and the priority order, see "Table 4.5 MCA parameter specification methods and priorities".

8.4.1 MCA Parameter to Specify Algorithm Selection

8.4.1.1 coll_select_allgather_algorithm (Specifying the Algorithm of the MPI_ALLGATHER Routine)

Fix the algorithm to be executed in the MPI_ALLGATHER routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
3dtorus_sm	Use the algorithm 3dtorus_sm tuned for Tofu interconnect.
3dtorus_fm, 3dtorus	Use the algorithm 3dtorus_fm tuned for Tofu interconnect. Both character strings of 3dtorus_fm and 3dtorus are effective.
gtbc	Use the algorithm gtbc tuned for Tofu interconnect.
two_proc	Use the algorithm two_proc implemented with the Open MPI.
neighbor	Use the algorithm neighbor implemented with the Open MPI.
ring	Use the algorithm ring implemented with the Open MPI.
recursive_doubling	Use the algorithm recursive_doubling implemented with the Open MPI.
bruck	Use the algorithm bruck implemented with the Open MPI.
linear	Use the algorithm linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.2 coll_select_allgatherv_algorithm (Specifying the Algorithm of the MPI_ALLGATHERV Routine)

Fix the algorithm to be executed in the MPI_ALLGATHERV routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
3dtorus_sm	Use the algorithm 3dtorus_sm tuned for Tofu interconnect.
3dtorus_fm, 3dtorus	Use the algorithm 3dtorus_fm tuned for Tofu interconnect. Both character strings 3dtorus_fm and 3dtorus is available.
gtvbc	Use the algorithm gtvbc tuned for Tofu interconnect.
two_proc	Use the algorithm two_proc implemented with the Open MPI.
neighbor	Use the algorithm neighbor implemented with the Open MPI.
ring	Use the algorithm ring implemented with the Open MPI.

Value of MCA Parameter	Contents
bruck	Use the algorithm bruck implemented with the Open MPI.
default	Use the algorithm default implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.3 coll_select_allreduce_algorithm (Specifying the Algorithm of the MPI_ALLREDUCE Routine)

Fix the algorithm to be executed in the MPI_ALLREDUCE routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
trinaryx3, trix3	Use the algorithm trinaryx3 tuned for Tofu interconnect. Both character strings of trinaryx3 and trix3 are effective.
trinaryx6, trix6	Use the algorithm trinaryx6 tuned for Tofu interconnect. Both character strings of trinaryx6 and trix6 are effective.
rdbc	Use the algorithm rdbc tuned for Tofu interconnect.
segmented_ring	Use the algorithm segmented_ring implemented with the Open MPI.
ring	Use the algorithm ring implemented with the Open MPI.
recursive_doubling	Use the algorithm recursive_doubling implemented with the Open MPI.
nonoverlapping	Use the algorithm nonoverlapping implemented with the Open MPI.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.4 coll_select_alltoall_algorithm (Specifying the Algorithm of the MPI_ALLTOALL Routine)

Fix the algorithm to be executed in the MPI_ALLTOALL routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
crp	Use the algorithm crp tuned for Tofu interconnect.
blacc6d	Use the algorithm blacc6d tuned for Tofu interconnect.
blacc3d	Use the algorithm blacc3d tuned for Tofu interconnect.
doublespread	Use the algorithm doublespread tuned for Tofu interconnect.
two_proc	Use the algorithm two_proc implemented with the Open MPI.
linear_sync	Use the algorithm linear_sync implemented with the Open MPI.
modified_bruck	Use the algorithm modified_bruck implemented with the Open MPI.
pairwise	Use the algorithm pairwise implemented with the Open MPI.
linear	Use the algorithm linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.5 coll_select_alltoallv_algorithm (Specifying the Algorithm of the MPI_ALLTOALLV Routine)

Fix the algorithm to be executed in the MPI_ALLTOALLV routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
doublespread	Use the algorithm doublespread tuned for Tofu interconnect.
pairwise	Use the algorithm pairwise implemented with the Open MPI.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI.

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

8.4.1.6 coll_select_barrier_algorithm (Specifying the Algorithm of the MPI_BARRIER Routine)

Fix the algorithm to be executed in the MPI_BARRIER routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
tree	Use the algorithm tree implemented with the Open MPI.
two_proc	Use the algorithm two_proc implemented with the Open MPI.
bruck	Use the algorithm bruck implemented with the Open MPI.
recursive_doubling	Use the algorithm recursive_doubling implemented with the Open MPI.
linear	Use the algorithm linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.7 coll_select_bcast_algorithm (Specifying the Algorithm of the MPI_BCAST Routine)

Fix the algorithm to be executed in the MPI_BCAST routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
bintree6d, bin6d	Use the algorithm bintree6d tuned for Tofu interconnect. Both bintree6d and bin6d are effective.
trinaryx3, trix3	Use the algorithm trinaryx3 tuned for Tofu interconnect. Both trinaryx3 and trix3 are effective.
bintree3d, bin3d	Use the algorithm bintree3d tuned for Tofu interconnect. Both bintree3d and bin3d are effective.
trinaryx6, trix6	Use the algorithm trinaryx6 tuned for Tofu interconnect. Both trinaryx6 and trix6 are effective.
binomial	Use the algorithm binomial implemented with the Open MPI.
binary_tree	Use the algorithm binary_tree implemented with the Open MPI.
split_binary_tree	Use the algorithm split_binary_tree implemented with the Open MPI.
pipeline	Use the algorithm pipeline implemented with the Open MPI.
chain	Use the algorithm chain implemented with the Open MPI.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.8 coll_select_gather_algorithm (Specifying the Algorithm of the MPI_GATHER Routine)

Fix the algorithm to be executed in the MPI_GATHER routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
simple	Use the algorithm simple tuned for Tofu interconnect.
linear_sync	Use the algorithm linear_sync implemented with the Open MPI.
binomial	Use the algorithm binomial implemented with the Open MPI.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.9 coll_select_gatherv_algorithm (Specifying the Algorithm of the MPI_GATHERV Routine)

Fix the algorithm to be executed in the MPI_GATHERV routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
simple	Use the algorithm simple tuned for Tofu interconnect.
default	Use the algorithm default implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.10 coll_select_reduce_algorithm (Specifying the Algorithm of the MPI_REDUCE Routine)

Fix the algorithm to be executed in the MPI_REDUCE routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
trinaryx3, trix3	Use the algorithm trinaryx3 tuned for Tofu interconnect. Both trinaryx3 and trix3 are effective.
trinaryx6, trix6	Use the algorithm trinaryx6 tuned for Tofu interconnect. Both trinaryx6 and trix6 are effective.
in-order_binary	Use the algorithm in-order_binary implemented with the Open MPI.
binomial	Use the algorithm binomial implemented with the Open MPI.
binary	Use the algorithm binary implemented with the Open MPI.
pipeline	Use the algorithm pipeline implemented with the Open MPI.
chain	Use the algorithm chain implemented with the Open MPI.
linear	Use the algorithm linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.11 coll_select_reduce_scatter_algorithm (Specifying the Algorithm of the MPI_REDUCE_SCATTER Routine)

Fix the algorithm to be executed in the MPI_REDUCE_SCATTER routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
ring	Use the algorithm ring implemented with the Open MPI.
recursive_halving	Use the algorithm recursive_halving implemented with the Open MPI.
non-overlapping	Use the algorithm non-overlapping implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.12 coll_select_scan_algorithm (Specifying the Algorithm of the MPI_SCAN Routine)

Fix the algorithm to be executed in the MPI_SCAN routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
recursive_doubling	Use the algorithm recursive_doubling implemented with the Open MPI.
linaer	Use the algorithm linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.13 coll_select_scatter_algorithm (Specifying the Algorithm of the MPI_SCATTER Routine)

Fix the algorithm to be executed in the MPI_SCATTER routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
use_bcast	Use the algorithm use_bcast created for this system.
binomial	Use the algorithm binomial implemented with the Open MPI.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.1.14 coll_select_scatterv_algorithm (Specifying the Algorithm of the MPI_SCATTERV Routine)

Fix the algorithm to be executed in the MPI_SCATTERV routine in the program to specific algorithm at all times.

Value of MCA Parameter	Contents
linear_sync	Use the algorithm linear_sync created for this system.
basic_linear	Use the algorithm basic_linear implemented with the Open MPI. For selecting algorithms by MCA parameter, see "8.3.1.3 Selecting Algorithms by MCA Parameter".

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

8.4.2 MCA Parameter related to Algorithm Selection Itself

8.4.2.1 coll_select_dectree_file (Specification of External Input User Definition File of Algorithm Selection)

Value of MCA Parameter	Contents
Path of Entered File	Specify the file in which the algorithm selection of the collective communication is entered. For details of the entry method for the file and each algorithm, see "8.3.1.5 Selecting Algorithms by External Input File". Perform algorithm selection based on the rules for the algorithm selection that the user entered. However, in case of an error in the rule or an unentered collective communication, perform the algorithm selected by this system. The default value of this parameter is NULL. For selecting algorithms by the external input file, see "8.3.1.5 Selecting Algorithms by External Input File".

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

8.4.2.2 coll_select_decision_process (Output of Algorithm Selection Process)

Value of MCA Parameter	Contents
2	Output the information of the algorithm selection of the collective communication. It is output in the standard error output. Rank 0 outputs information in each communicator.
1	Rank 0 outputs information in the MPI_COMM_WORLD.
0	Information is not output. The default value of this parameter is 0. For detailed information, see "8.3.2.1 Displaying Selection Process of Algorithms".

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

8.4.3 MCA Parameter to Tune Algorithm Itself

8.4.3.1 coll_select_allreduce_algorithm_segmentsize (Specifying the Segment Size of the MPI_ALLREDUCE Routine)

Value of MCA Parameter	Contents
Integer value from 1 to 16776960	Specifies the segment size in bytes of data which is transferred using some algorithm of the MPI_ALLREDUCE routine. Specific algorithm is specified in the MCA parameter "8.4.1.3 coll_select_allreduce_algorithm (Specifying the Algorithm of the MPI_ALLREDUCE Routine)". If a value greater than 16776960 is specified, the specified value is assumed to be 16776960. Specifiable algorithms are given below. `segmented_ring, trinaryx6 (trix6), trinaryx3 (trix3)` When the value specified for this parameter is not divisible by the datatype size specified for the argument of the MPI_ALLREDUCE routine, the value specified this parameter is changed to a divisible value. Adjusting this parameter changes the execution time of specific algorithms. It may decrease performance compared to when this parameter is not specified.
0	The default value of this parameter is 0. If the message size is less than or equal to 16776960 bytes, segment division is not performed. If the message size is greater than 16776960 bytes, the behavior is same as when 16776960 is specified for this parameter. For detailed information, see "8.2 MCA Parameter Tuning of Algorithms".

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

8.4.3.2 coll_select_bcast_algorithm_segmentsize (Specifying the Segment Size of the MPI_BCAST Routine)

Value of MCA Parameter	Contents
Integer value from 1 to 16776960	Specifies the segment size in bytes of data which is transferred using some algorithm of the MPI_BCAST routine. Specific algorithm is specified using the MCA parameter "8.4.1.7 coll_select_bcast_algorithm (Specifying the Algorithm of the MPI_BCAST Routine)". If a value greater than 16776960 is specified, the specified value is assumed to be 16776960. Specifiable algorithms are given below. `chain, pipeline, split_binary_tree, binary_tree, binomial, trinaryx6 (trix6), bintree3d (bin3d), trinaryx3 (trix3), bintree6d (bin6d)` When the value specified for this parameter is not divisible by the datatype size specified for the argument of the MPI_BCAST routine, the value specified for this parameter is changed to a divisible value. Adjusting this parameter changes the execution time of specific algorithms. It may decrease performance compared to when this parameter is not specified. It is assumed that the value specified for the count argument of the MPI_BCAST routine is the same among all ranks. It is necessary to specify 1 to the MCA parameter coll_tuned_bcast_same_count. For the MCA parameter coll_tuned_bcast_same_count, see "Table 4.14 coll_tuned_bcast_same_count (achieves faster communication when MPI_BCAST/MPI_IBCAST routines are used with the same count among the processes)".
0	The default value of this parameter is 0. If the message size is less than or equal to 16776960 bytes, segment division is not performed. If the message size is greater than 16776960 bytes, segment division is performed as below. - If specified algorithm is chain, pipeline, split_binary_tree, binary_tree, or binomial, segment division is not performed. - If specified algorithm is trinaryx6 (trix6), bintree3d (bin3d), trinaryx3 (trix3), or bintree6d (bin6d), the behavior is same as when 16776960 is specified for this parameter. For detailed information, see "8.2 MCA Parameter Tuning of Algorithms".

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

8.4.3.3 coll_select_reduce_algorithm_segmentsize (Specifying the Segment Size of the MPI_REDUCE Routine)

Value of MCA Parameter	Contents
Integer value from 1 to 16776960	Specifies the segment size in bytes of data which is transferred using some algorithm of the MPI_REDUCE routine. Specific algorithm is specified using the MCA parameter "8.4.1.10 coll_select_reduce_algorithm (Specifying the Algorithm of the MPI_REDUCE Routine)". If a value greater than 16776960 is specified, the specified value is assumed to be 16776960. Specifiable algorithms are given below. `chain, pipeline, binary, binomial, in-order_binary, trinaryx6 (trix6), trinaryx3 (trix3)` When the value specified for this parameter is not divisible by the datatype size specified for the argument of the MPI_REDUCE routine, the value specified for this parameter is changed to a divisible value. Adjusting this parameter changes the execution time of specific algorithms. It may decrease performance compared to when this parameter is not specified.

Value of MCA Parameter	Contents
0	The default value of this parameter is 0. If the message size is less than or equal to 16776960 bytes, segment division is not performed. If the message size is greater than 16776960 bytes, the behavior is same as when 16776960 is specified for this parameter. For detailed information, see "8.2 MCA Parameter Tuning of Algorithms".

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

8.4.4 MCA Parameter to Obtain Algorithm Selection Result

8.4.4.1 coll_select_get_tuning_info (Obtaining the Algorithm Information of the Collective Communication Executed Immediately Before)

Value of MCA Parameter	Contents
1	By using the MPI_Comm_get_info, it becomes possible to obtain the algorithm of the collective information executed immediately before. If this parameter is used, the execution time is longer than usual.
0	Even by using the MPI_Comm_get_info, it is not possible to obtain the algorithm of the collective information executed immediately before. The default value of this parameter is 0. For detailed information, see "8.3.2.2 Obtaining Selection Results of Algorithms by Using Info Object".

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

8.5 Output Message

This section describes the warning message related to the algorithm selection and the output message by the display function of the algorithm selection process.

8.5.1 Output Message Related to Algorithm Selection (Warning)

[mpi::coll-select::user-file-warn] Unable to open file. [*path*]

- Description

  Cannot reference the file specified by the MCA parameter coll_select_dectree_file. Ignore the file specification by the external input file and continue processing.

- Parameters

  path : Path name specified by MCA parameter coll_select_dectree_file.

- Action method

  Make sure that the character string specified by the MCA parameter is correct. If it is correct, consult the Service Engineer (SE) together with the output error message. If it is not correct, specify the file correctly by the MCA parameter.

8.5.2 Output Message by Display Function of Algorithm Selection Process (Warning)

[mpi-coll::show-decision-process::warn] Algorithm selection does not work. [*reason*]

- Description

  Cannot use the function of algorithm selection in this communicator. Also, cannot obtain part of the MPI statistical information.

- Parameters

  reason : Figure to indicate reason

reason	Content of Reason
0	The communicator is the inter-communicator.
1	The size of the communicator is 1.
2	The communicator is created by the MPI_INTERCOMM_MERGE routine.
3	Cannot use the function of algorithm selection for other reasons.

- Action method

  Make sure that the created communicator satisfies the above reasons. If it satisfies, need not to take measures. If it does not satisfy, consult the Service Engineer (SE) together with the error message.

[mpi-coll::show-decision-process::warn] The algorithm is judged as not applicable. [*collname*] [*algnum*] [*reason*] [*new_algnum*]

- Description

  The specified algorithm does not satisfy the application conditions. It selects a different algorithm.

- Parameters

  collname : Type of collective communication

  algnum : Number to indicate algorithm

  reason : Figure to indicate reason

  new_algnum : Alternative algorithm number which is selected when *algnum* is 1 to 6 or 300.

reason	Content of Reason
0	The size of communicator does not satisfy the conditions.
1	It does not satisfy the application conditions of algorithm excluding the datatype.
2	It does not satisfy the conditions of datatype.
3	Cannot use gtvbc algorithm as there is space in the data alignment.
4	Cannot divide data as the MCA parameter coll_tuned_bcast_same_count is disabled. Set the segment size to 0.
5	Cannot select for other reasons.

- Action method

  If the intended algorithms are not selected due to this message, see the conditions described in "8.3.4 List of Algorithms and Conditions Required for Application".

[mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [*reason*]

- Description

  Problem occurred for the algorithm selection by the external input file. Change to the algorithm selection of the system.

- Parameters

  reason : Figure to indicate reason

reason	Content of Reason
0	Cannot reference the external input file.
1	There is an error in string or syntax entered in the external input file.

reason	Content of Reason
2	The number of entries that can be made in the external input file is exceeded.
3	The number of conditional statements that can be entered in the external input file is exceeded.
4	There is division by zero in calculation described in the external input file.
5	There is an error in a variable described in the external input file.
6	Algorithm name is not correct.
7	No executable algorithms exist.
8	An internal error of the MPI library occurred during analysis of the external input file.

- Action method

  Confirm whether the various settings are correct according to *reason* if the *reason* is other than 8.

  If the *reason* is 8, consult the Service Engineer (SE) together with the external input file and the message that was output.

[mpi::coll-select::show-decision-process::warn] Algorithm selection by the Info object does not work. [*method*]

- Description

  Problem occurred for the algorithm selection by the Info object. Change to the algorithm selection of next priority.

- Parameters

  method : Method for algorithm selection

  - info-oneshot

    Method for selecting algorithm only once by the Info object. See "8.3.1.4.2 Specification for Each Collective Communication Routine Call".

  - info-rules

    Method for selecting algorithm in the unit of communicator by the Info object. See "8.3.1.4.3 Specification for Each Communicator".

- Action method

  Confirm if the value corresponding to the *method* specified by the Info object is correct. If the value specified by the Info object is correct, consult the Service Engineer (SE) together with processing of the applicable portion and the error message.

[mpi::coll-select::show-decision-process::warn] Tofu Barrier Communication can not be applicable. [*method*]

- Description

  Cannot specify barrier communication. Change to the algorithm selection of next priority.

- Parameters

  method : Method for algorithm selection

  - mca-param

    Algorithm selection by the MCA parameter. Confirm "8.3.1.3 Selecting Algorithms by MCA Parameter".

  - info-oneshot

    Method for selecting algorithm only once by the Info object. See "8.3.1.4.2 Specification for Each Collective Communication Routine Call".

  - info-rules

    Method for selecting algorithm in the unit of communicator by the Info object. See "8.3.1.4.3 Specification for Each Communicator".

- user-file

  Algorithm selection by the external input file. See "8.3.1.5 Selecting Algorithms by External Input File".

- Action method

  Do not specify the barrier communication to *method*. Barrier communication can be used only for the automatic selection by the MPI library.

[mpi::coll-select::show-decision-process::warn] An algorithm different from the selected one was used. [*collname*][*algnum*][*reason*][*new_algnum*]

- Description

  The 3dtorus_sm algorithm was specified, but could not be used due to contention for communication resource used in the algorithm. Therefore, another algorithm was used.

- Parameters

  collname : Type of collective communication

  algnum : Number to indicate algorithm

  reason : Figure to indicate reason

  new_algnum : Algorithm number for actually selected.

reason	Content of Reason
0	Cannot use the specified algorithm due to contention for communication resource.

- Action method

  See the conditions described in "8.3.4 List of Algorithms and Conditions Required for Application" if this message was output and the specified algorithm is not selected.

8.5.3 Output Message by Display Function of Algorithm Selection Process (Information)

[mpi::coll-select::show-decision-process::info] Any algorithm is not selected. [*method*] [*collname*]

- Description

  Algorithm selection by *method* was not performed.

- Parameters

  method : Method for algorithm selection

  - mca-param

    Method for algorithm selection. See "8.3.1.3 Selecting Algorithms by MCA Parameter".

  - info-oneshot

    Method for selecting algorithm only once by the Info object. See "8.3.1.4.2 Specification for Each Collective Communication Routine Call".

  - info-rules

    Method for selecting algorithm in the unit of communicator by the Info object. See "8.3.1.4.3 Specification for Each Communicator".

  - user-file

    Algorithm selection by the external input file. See "8.3.1.5 Selecting Algorithms by External Input File".

  collname : Type of collective communication

- Action method

  No need to take measures.

[mpi::coll-select::show-decision-process::info] An algorithm is selected. [*method*] [*collname*] [*algnum*] [*reason*]

- Description

  Algorithm selection by *method* was performed. Determine the conditions required for the application from now.

- Parameters

  method : Method for algorithm selection

  - special-route

    Algorithm selection by special cases. See "8.3.1.2 Flow of Selecting Algorithms in Special Case".

  - mca-param

    Algorithm selection by the MCA parameter. See "8.3.1.3 Selecting Algorithms by MCA Parameter".

  - info-oneshot

    Method for selecting algorithm only once by the Info object. See "8.3.1.4.2 Specification for Each Collective Communication Routine Call".

  - info-rules

    Method for selecting algorithm in the unit of communicator by the Info object. See "8.3.1.4.3 Specification for Each Communicator".

  - user-file

    Algorithm selection by the external input file. See "8.3.1.5 Selecting Algorithms by External Input File".

  - system-file

    Automatic algorithm selection by this system.

  collname : Type of collective communication

  algnum : Number to indicate algorithm

  reason : Reason for selecting the selection method in case of special-route

reason	Content of the reason for Special
0	Select barrier communication preferentially.
1	Select the algorithms that correspond to non-commutative operations and the order guarantee mode.
2	Select the algorithm for the MPI_IN_PLACE.

- Action method

  No need to take measures.

[mpi::coll-select::show-decision-process::info] The algorithm is judged as applicable. [*method*] [*collname*] [*algnum*]

- Description

  Algorithm selection by *method* was performed. As the conditions required for the application are satisfied, execute this algorithm.

- Parameters

  method : Method for algorithm selection

  - mca-param

    Algorithm selection by the MCA parameter. See "8.3.1.3 Selecting Algorithms by MCA Parameter".

- info-oneshot

  Method for selecting algorithm only once by the Info object. See "8.3.1.4.2 Specification for Each Collective Communication Routine Call".

- info-rules

  Method for selecting algorithm in the unit of communicator by the Info object. See "8.3.1.4.3 Specification for Each Communicator".

- user-file

  Algorithm selection by the external input file. See "8.3.1.5 Selecting Algorithms by External Input File".

- system-file

  Automatic algorithm selection by this system.

collname : Type of collective communication

algnum : Number to indicate algorithm

- Action method

  No need to take measures.

[mpi::coll-select::show-decision-process::info] Rules are not set for this collective communication. [*collname*]

- Description

  The rules for this collective communication are not entered in the external input file specified by the user.

- Parameters

  collname : Type of collective communication

- Action method

  Make sure that the type of displayed collective communication is not entered in the external input file. If it is entered in the external input file, consult the Service Engineer (SE) together with the external input file and the error message.

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

Barrier gate

A hardware resource used for Tofu barrier communication. There are two types of barrier gates. One is input-output gate for start and end point and the other is relay gate for relay point. The total number of input-output gates and relay gates per network interface device (TNI) is 48. An MPI library can use up to 96 input-output gates and 192 relay gates per compute node. These maximum numbers of barrier gates which are available for an MPI library may be changed when the edition number of a product on this software system is changed.

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.

Compute nodes allocated to the communicator

This refers to the compute nodes where at least one parallel process that belongs to the communicator exists in a certain communicator.

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.

Tofu barrier communication

A hardware communication function. It provides a data reduction function for data which is 48 bytes or less. It also provides barrier synchronization between processes on a Tofu interconnect.

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).