NEC

NEC MPI ユーザーズガイド (G2AM01)

SX-Aurora TSUBASA

輸出する場合の注意事

本製品（ソフトウェアを含む）は，外国為替および外国貿易法で規定される規制貨物（または役務）に該当することがあります。

その場合，日本国外へ輸出する場合には日本国政府の輸出許可が必要です。

なお，輸出許可申請手続にあたり資料などが必要な場合には，お買い上げの販売店またはお近くの当社営業拠点にご相談ください。

は　し　が　き

本書は，SX-Aurora TSUBASA 上で分散メモリ型並列処理プログラミングを行うためのメッセージ通信ライブラリ NEC MPI について説明したものです。

NEC MPI は，2015 年に改定された MPI-3.1 仕様に準拠した機能を提供しており，さらに SX-Aurora TSUBASA アーキテクチャの特長の 1 つである共有メモリを活かし，また，InfiniBand を MPI 処理系から直接操作することで高速な通信を実現しています。

本書における MPI の基本概念，機能 および MPI 手続の引用仕様の説明は，MPI フォーラムから公開されている次の文書を参考にしています。

MPI: A Message-Passing Interface Standard Version 3.1
Message PassingInterface Forum
June 4, 2015

MPI に関してさらに詳しい仕様をお知りになりたい方は，MPI フォーラムより公開されている文書を，以下より入手していただくようお願いします。

http://www.mpi-forum.org/

本書の構成 および 関連説明書については，「本書の読み方」を参照してください。

2018 年　2 月　初　版
2020 年　7 月　第 11 版
2020 年　8 月　第 12 版
2020 年　10 月　第 13 版
2020 年　12 月　第 14 版
2020 年　12 月　第 15 版
2021 年　3 月　第 16 版
2021 年　5 月　第 17 版
2021 年　5 月　第 18 版
2021 年　7 月　第 19 版
2021 年　12 月　第 20 版
2022 年　3 月　第 21 版

備考

本書で説明している全ての機能は，プログラムプロダクトであり，以下のプロダクトに対応しています。

- NEC MPI
- NEC MPI/Scalar-Vector Hybrid

商標

- UNIX は The Open Group の登録商標です。
- InfiniBand は，InfiniBand(R) Trade Association の商標です。
- Intel は，Intel Corporation またはその子会社の商標です。
- NVIDIA,CUDA は，NVIDIA Corporation の商標または登録商標です。
- PBS Professional は，Altair Engineering Inc の商標です。
- CentOS は、Ret Hat, Inc.の商標です。
- その他に記載の社名及び製品名は、各社の商標です。

OSS(オープンソースソフトウェア)について

NEC MPI および NEC MPI/Scalar-Vector Hybrid には下記の OSS が含まれています。

- MPICH 1.1.1
- MVAPICH2 2.2b
- ptmalloc2 Jun 5th 2006
- ROMIO 1.0.1
- Notre Dame C++ bindings for MPI 1.0.3
- MPI Message Queue Dumping 1.06
- MVICH

また、必要に応じて NEC が提供する compat-hwloc1 パッケージが使用されます。本パッケージは CentOS の当該パッケージを再ビルドしたものであり、以下の OSS が含まれています。

- hwloc 1.11.9

これら各 OSS の著作権、および適用されるライセンスについては付録 B　著作権・ライセンスをご参照ください。

本書の読み方

対象読者

本書は主にシステム管理者，一般利用者ならびにプログラマを対象として書かれたものです。特にメッセージ通信機能を利用して初めて分散並列処理プログラムを作成しようと考えているプログラマには本書は有効な説明書となります。本書の読者は，Fortran 言語 または C 言語を理解していることを前提としています。

読み進め方

本書は，次の章から構成されています。表の右側の対象読者を参考にして読み進めてください。

章	タイトル	内容	対象読者
1	概説	MPI の概要	一般利用者 プログラマ
2	機能概要	MPI の主要な機能の説明	一般利用者 プログラマ
3	操作方法	NEC MPI の使用方法の説明	一般利用者 プログラマ システム管理者
4	MPI 手続引用仕様	MPI 手続の引用仕様についての説明	一般利用者 プログラマ

付録

付	タイトル	内容	対象読者
A	MPI 予約名一覧	MPI の予約名の説明	一般利用者 プログラマ
B	著作権・ライセンス	NEC MPI が使用する OSS の著作権・ライセンス	一般利用者 プログラマ

関連説明書

本書を使用するにあたり，関連する説明書として次のものがあります。これらの説明書は NEC Aurora Forum 内の「NEC SX-Aurora TSUBASA Documentation」に掲載しています。

- Fortran コンパイラ(nfort)の使用法
  Fortran コンパイラユーザーズガイド (G2AF02)
- C コンパイラ(ncc)と C++コンパイラ(nc++)の使用法
  C/C++コンパイラユーザーズガイド (G2AF01)
- FTRACE 機能
  PROGINF/FTRACE ユーザーズガイド (G2AT03)
- NQSV の使用法
  NEC Network Queuing System V(NQSV) 利用の手引 [操作編] (G2AD03)
- デバッガの使用法
  NEC Parallel Debugger ユーザーズガイド (G2AT02)
- ScaTeFS VE ダイレクト IB ライブラリの使用法
  NEC Scalable Technology File System (ScaTeFS) 運用の手引 (G2AH01)
- 高速 I/O の使用法
  VE プログラムの実行方法
- VEO の使用法
  The Tutorial and API Reference of Alternative VE Offloading

表記上の約束／用語の説明

本書では次の表記規則を使用しています。

縦棒	|	オプション または 必須の選択項目を分割します。

用語

用語	説　明
Vector Engine (VE)	ベクトルエンジン。SX-Aurora TSUBASAの中核であり，ベクトル演算を行う部分です。PCI Expressカードであり，x86サーバーに搭載して使用します。
Vector Host (VH)	ベクトルホスト。ベクトルエンジンが接続されているサーバー，つまり ホストコンピュータを指します。
スカラホスト	スカラクラスタ内のサーバー。
ホスト	VH、VEまたはスカラホスト。
ベクトルプロセス	VE上で動作するMPIプロセス。
スカラプロセス	VHまたはスカラホスト上で動作するMPIプロセス。
NQSV	SX-Aurora TSUBASAのジョブスケジューラ
NQSVリクエスト実行	NQSVを使用したプログラムの実行。
PBS	Altair Engineering Incのジョブスケジューラ。商用版のPBS ProfessionalとオープンソースのOpenPBSがあります。
PBSリクエスト実行	PBS ProfessionalまたはOpenPBSを使用したプログラム実行
VE番号	1つのVHに接続されているVEの識別番号。0から始まる連続した整数値です。
VH名	VHであるホストコンピュータのホスト名。
論理VE番号	NQSVリクエスト実行におけるVEの識別番号。0から始まる連続した整数値です。
論理VH番号	NQSVリクエスト実行におけるVHの識別番号。0から始まる連続した整数値です。
MPIデーモン	各VH上で，MPIプロセスを生成・管理するプロセス。
ハイブリッド実行	VEとVHやスカラホストにMPIプロセスを生成し、MPI実行を行うこと

目　　次

第 1 章　概説 11
1.1　MPI の概要 11
1.2　NEC MPI について 12
1.2.1　NEC MPI のコンポーネント 12
1.2.2　NEC MPI/Scalar-Vector Hybrid 12
1.2.3　VEO (VE Offloading)および CUDA 対応 13

第 2 章　機能概要 15
2.1　はじめに 15
2.2　実行管理 および 環境問合せ 15
2.2.1　MPI の初期化 および 終了処理 15
2.2.2　不透明オブジェクト および ハンドル 16
2.2.3　エラー処理 16
2.3　プロセス管理 17
2.3.1　MPI プロセス 17
2.3.2　コミュニケーター 17
2.3.3　MPI プロセス数の取得 17
2.3.4　MPI プロセスのランク 17
2.4　動的プロセス管理 18
2.4.1　動的プロセス生成 18
2.4.2　動的プロセス結合 18
2.5　プロセス間通信 19
2.5.1　1 対 1 通信 19
2.5.2　集団操作 29
2.5.3　片側通信 35
2.6　並列 I/O 40
2.6.1　概要 40
2.6.2　ビュー 40
2.6.3　主要手続 41
2.6.4　ファイル名の接頭辞 43
2.6.5　プログラム例 43
2.6.6　並列 I/O の高速化 46
2.7　スレッド実行環境 46
2.8　エラーハンドラー 47
2.9　属性キャッシュ 47

2.10 Fortran サポート 48
2.11 利用者指定情報(MPI_INFO) 48
2.12 言語間相互利用可能性(Language Interoperability) 49
2.13 非ブロッキング MPI 手続利用時の注意事項 50

第 3 章 操作方法 53
3.1 コンパイル・リンク 53
3.2 MPI プログラムの実行 57
3.2.1 プログラムの実行指定 59
3.2.2 実行時オプション 60
3.2.3 ホストの指定方法 68
3.2.4 環境変数 69
3.2.5 MPI プロセス識別用環境変数 83
3.2.6 他の処理系用の環境変数の指定 86
3.2.7 ランクの割当て 86
3.2.8 MPI プログラムの実行ディレクトリ 86
3.2.9 singularity コンテナを使用した MPI プログラムの実行 87
3.2.10 MPI プログラム実行例 87
3.3 MPI プロセスの標準出力 および 標準エラー出力 91
3.4 実行性能情報 91
3.5 MPI 通信情報 99
3.6 FTRACE 機能 105
3.7 MPI トレース機能 108
3.8 トレースバック機能 108
3.9 MPI 集団手続デバッグ支援機能 109
3.10 MPI プログラム終了状態 111
3.11 その他の注意事項 112

第 4 章 MPI 手続引用仕様 117
4.1 1 対 1 通信 118
4.2 データ型 156
4.3 集団操作 183
4.4 グループ，コンテキスト，コミュニケーター 218
4.5 プロセストポロジー 249
4.6 実行管理 274
4.7 利用者指定情報 290
4.8 プロセスの生成 および 管理 296
4.9 片側通信 304
4.10 外部インタフェース 330
4.11 入出力 336

4.12　言語バインディング 376
4.13　プロファイリング 380
4.14　廃止予定関数 382

付録 A　MPI 予約名一覧 387
A.1 MPI データ型 388
A.2 集計演算用 MPI データ型 391
A.3 エラーコード 392
A.4 予約コミュニケーター名 395
A.5 集計演算子 395

付録 B　著作権・ライセンス 397
B.1 MPICH 1.1.1 397
B.2 MVAPICH2 2.2b 398
B.3 ptmalloc2 Jun 5th 2006 401
B.4 ROMIO 1.0.1 401
B.5 Notre Dame C++ bindings for MPI 1.0.3 402
B.6 MPI Message Queue Dumping 1.06 404
B.7 MVICH 406
B.8 hwloc 1.11.9 407

索　　引 409

第 1 章　概説

1.1　MPI の概要

Message Passing Interface (MPI)によって，プロセス間のメッセージ通信（1 対 1 通信 ならびに 片側通信） および 集団通信操作を使用した，分散メモリ型の並列プログラム開発が可能となります。

MPI 規格ができる以前にも，さまざまなメッセージ通信ライブラリが，大学や政府研究機関，および コンピュータベンダによって作られていましたが，それらは，動作するプラットフォームマシン固有の機能に依存していたり，特定の OS 上でしか実現できない機能を提供しており，機能の呼出しも独自の形式を採っていたので，それらライブラリを使って並列化したアプリケーションは，必ずしも移植性や汎用性の高いものではありませんでした。

そこで，既存のメッセージ通信ライブラリの有効な機能を取り込み，かつ ほとんど手を加えなくとも多くのプラットフォームマシン上で動作するポータビリティのある並列アプリケーションが記述できるようにメッセージ通信仕様の統一を目指して，MPI という名称でインタフェースの標準化が行われました。

標準化を行うにあたって掲げられた目標には，以下のものがあります。

- アプリケーションプログラムから用いる呼出しインタフェースを設計する。
- 効率的な通信を実現する。たとえば通信コプロセッサが使えるのであればその利用を可能とし，かつ利用を妨げとならない仕様とする。
- 異機種接続環境でも使用できる実装を可能とする。
- C 言語と Fortran 言語から呼び出せる形式を定める。信頼できる通信インタフェースを提供する。すなわち，低レベルの通信エラーの対応は下位の通信システムで実現されているものとし，利用者は意識しなくてよいものとする。
- 既存の通信ライブラリのインタフェースと大きく異ならないものとし，フレキシビリティの高い拡張機能のインタフェースを定義する。
- 多くのプラットフォーム上で実現でき，かつ基礎となる通信ソフトウェアやシステムソフトウェアに大きな変更を加える必要のないインタフェースを定義する。
- インタフェース上の機能定義を言語に依存しないようにする。
- インタフェースをリエントラント対応に設計し，スレッドセーフティを可能にする。

標準化作業は，当初，米国テネシー大学の Dongarra 博士らを中心とするグループによって始まり，米国やヨーロッパの大学，政府研究機関，企業の研究者 および コンピュータベンダからの参加によって構成される私的な団体である MPI フォーラムによって進められ，1994 年 6 月に MPI 第 1.0 版が公開されました。その後，仕様の明確化や修正が重ねられ，1997 年 7 月には片側通信や並列 I/O など新しい機能が追加された MPI 第 2.0 版が公開されました。2012 年 9 月には非ブロッキング集団通信の追加や片側通信の機能強化が行われた MPI 第 3.0 版が公開され，さらに 2015 年 6 月には，MPI 第 3.0 版仕様の修正 および 明確化を行った MPI 第 3.1 版が公開されています。

1.2 NEC MPI について

NEC MPI は MPI 第 3.1 版の仕様に準拠した SX-Aurora TSUBASA システム向けメッセージ通信ライブラリであり，VE 内であれば共有メモリを利用し，InfiniBand で接続されたクラスタシステムであれば，InfiniBand を MPI 処理系から直接操作することで高速通信を実現しています。

1.2.1 NEC MPI のコンポーネント

NEC MPI は，次のコンポーネントから構成されています。

- MPI インクルードファイル
  Fortran 言語用 および C 言語用の 2 つのインクルードファイルがあります。
  MPI プログラミングでは,利用者プログラム内でこれらのファイルをインクルード行に指定します。
  インクルード行の記述方法は，付録 A を参照して下さい。
- MPI モジュール
  Fortran 言語用の MPI モジュールです。Fortran プログラム中で use して利用します。
- MPI コンパイルコマンド
  MPI ソースプログラムのコンパイル および リンクのために使用します。使い方は，第 3 章を参照して下さい。
- MPI ライブラリ
  MPI 実行プログラムのリンク時に必要としますが，MPI コンパイルコマンドを利用することで，利用者はその名前や格納場所を意識する必要はありません。
- MPI 実行コマンド（mpirun，mpiexec）
  MPI の実行プログラムを起動するのに使用します。
  コマンドの使い方は，第 3 章において説明しています。
- MPI デーモン
  利用者が，直接参照あるいは操作する必要はありません。
  MPI 実行プログラムの起動と同時に自動的に立ち上がり，MPI のプロセスの制御を行ないます。
  MPI 実行プログラムが終了すれば，MPI デーモンの実行も終了します。

1.2.2 NEC MPI/Scalar-Vector Hybrid

NEC MPI は VE 上での MPI プロセス実行をサポートしています。さらに NEC MPI/Scalar-Vector Hybrid を使用することで、VH やスカラホスト上での MPI プロセス実行や、VE 上で動作する MPI プロセスと VH やスカラホスト上で動作する MPI プロセスを連携実行させるハイブリッド実行が可能となります。ハイブリッド実行時における VH あるいはスカラホスト上のプロセスと VE 上で動作するプロセス間の通信手段は、システム構成の特性などを考慮し自動的に選択されます。なお、MPI から直接操作できる InfiniBand が利用可能な場合が最も高速となります。

ハイブリッド実行を行うには、VH やスカラホスト向けに MPI プログラムをコンパイル・リンクし、ハイブリッド実行に対応した mpirun コマンドの指定が必要です。詳細については第 3 章をご参照ください。

1.2.3 VEO (VE Offloading)および CUDA 対応

NEC MPI を利用した MPI プログラム実行では、VEO（VE Offload）機能を併用する場合、VH で動作する MPI プロセスの MPI 手続きの引数に VE メモリのアドレスを直接指定できます。CUDA 機能を併用する場合も同様に、スカラホストで動作する MPI プロセスの MPI 手続きの引数に GPU メモリのアドレスを直接指定できます。VEO 機能や CUDA 機能を併用する場合のコンパイル・リンク、実行方法、および、注意事項については第 3 章をご参照ください。なお、VEO 機能を併用する場合は VEOS Version 2.7.6 以降、CUDA 機能を併用する場合は NVIDIA CUDA Toolkit 11.1, V11.1.105 が必要となります。また、VEO 機能についての詳細は"The Tutorial and API Reference of Alternative VE Offloading"の情報を合わせてご参照ください。
(*) 同一ホストでの VE および GPU の同時実装、および、それらデバイス間の通信は今後サポート予定です。

1.2.3.1 AVEO UserDMA 機能

VEO 機能を併用する場合において、VH ノード内の VE メモリ間の直接転送を可能とする機能を提供します。本機能を使用するためには VEOS version 2.11.0 以降が必要です。本機能は自動的に有効になりますが、アプリケーションや実行環境によっては無効化した方が高速となる場合があります。無効化する方法については「3.2.4 環境変数」をご参照ください。本機能を無効化した場合や、NEC MPI version 2.20.0 以前を使用している場合、通信データは VH メモリを経由します。本機能の注意事項については「3.11 その他の注意事項」をご参照ください。

第 2 章　機能概要

2.1　はじめに

本章では，次のような MPI の主要な機能について説明します。

- 実行管理 および 環境問合せ
- プロセス管理，動的プロセス管理
- プロセス間通信
  - 1 対 1 通信
  - 集団操作
  - 片側通信
- 並列 I/O
- スレッド実行環境
- エラーハンドラー
- 属性キャッシュ
- Fortran サポート
- 利用者指定情報(MPI_INFO)
- 言語間相互利用可能性(Language Interoperability)
- 非ブロッキング MPI 手続利用時の注意事項

MPI 規格で規定されている機能について，更にお知りになりたい場合，MPI フォーラムから公開されている次のドキュメントを参照してください。

- MPI：A Message-Passing Interface Standard Version 3.1

2.2　実行管理 および 環境問合せ

2.2.1　MPI の初期化 および 終了処理

MPI を使用する場合，まず，手続 MPI_INIT または MPI_INIT_THREAD を呼び出すことにより，初期化を行います。また，手続 MPI_FINALIZE を呼び出すことで MPI の利用を終了します。利用者は，この 2 つの呼出しの間で 1 対 1 通信や集団通信といったさまざまな MPI 手続を呼び出すことができます。ただし，手続 MPI_INITIALIZED，MPI_FINALIZED，および MPI_GET_VERSION は，プログラムの任意の時点で呼び出すことができます。手続 MPI_INITIALIZED は，手続 MPI_INIT または MPI_INIT_THREAD が

既に呼び出されたかどうかを判定するのに使用します。手続 MPI_FINALIZED は，手続 MPI_FINALIZE が既に呼び出されたかどうかを判定するのに使用します。

以下の例は，Fortran または C プログラム中で，MPI の初期化と終了処理を行う部分を示しています。

```
PROGRAM MAIN
use mpi
INTEGER IERROR
! Establish the MPI environment
CALL MPI_INIT(IERROR)

!  Terminate the MPI environment
CALL MPI_FINALIZE(IERROR)

STOP
END
```

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

int main(int argc, char **argv){
/* Establish the MPI environment */
  MPI_Init( &argc, &argv );
/* Terminate the MPI environment */
  MPI_Finalize();
}
```

2.2.2 不透明オブジェクト および ハンドル

コミュニケーターや MPI データ型のような MPI 処理系が管理している MPI のオブジェクトを不透明オブジェクトと呼びます。MPI は，各不透明オブジェクトに対応して，ハンドルと呼ばれるデータを定義しています。利用者は，不透明オブジェクトを直接参照するのではなく，対応するハンドルを MPI 手続に渡すことにより，間接的に操作できます。

2.2.3 エラー処理

MPI 手続中で検出された異常(不正な引数が渡された場合など)に対する操作のために，異常検出を条件に呼び出す利用者手続(エラーハンドラー)の定義を行う機能が提供されています。詳しくは 2.8 節を参照してください。

2.3 プロセス管理

2.3.1 MPI プロセス

MPI プロセスとは，CPU 資源やメモリ空間が個々に割り当てられ独立に実行されるプログラム処理単位です。全ての MPI プロセスは，プログラムの最初から実行を行いますが，MPI の機能は，手続 MPI_INIT または MPI_INIT_THREAD を呼び出した後でなければ使用することができません。

動的プロセス生成機能を利用すると，プログラム実行中，動的に MPI プロセスを新たに生成することができます。動的プロセス生成機能については 2.4 節を参照してください。

2.3.2 コミュニケーター

MPI プロセスの順序付き集合をグループと呼びます。グループ中のプロセスは，0 から始まる一意な整数値の識別番号であるランクをもちます。コミュニケーターは，MPI プロセスのグループから構成される通信範囲を意味しています。コミュニケーターによってグループを特定し，そのグループ中の MPI プロセスは，ランクによって識別できます。この識別は通信相手のプロセスを特定する場合に必要となります。

利用者は，新たなグループに対して，新たなコミュニケーターを定義することもできます。MPI 規格は，コミュニケーターMPI_COMM_WORLD を事前定義しています。プログラム開始時のコミュニケーターMPI_COMM_WORLD は，プログラム開始時の全ての MPI プロセスから構成されます。

コミュニケーターは，単一の MPI プロセスグループから構成されるイントラコミュニケーター (intra-communicator) と，2 つの互いに素な MPI プロセスグループから構成されるインターコミュニケーター (inter-communicator)とに分類できます。インターコミュニケーターに属する各プロセスに対して，そのプロセスが所属するグループをローカルグループ，他方のグループをリモートグループといいます。

イントラコミュニケーター上の通信では，通信相手となる MPI プロセスは常に同一のグループ内に存在しています。一方，インターコミュニケーター上の通信では，通信相手の MPI プロセスは常にリモートグループ中のプロセスとなります。

2.3.3 MPI プロセス数の取得

プログラム開始時の MPI プロセスの総数は，コミュニケーターMPI_COMM_WORLD を引数にして手続 MPI_COMM_SIZE を呼び出すことで得ることができます。

2.3.4 MPI プロセスのランク

MPI プロセスは，コミュニケーター および ランクと呼ばれる識別番号によって識別できます。ランクは，0 以上，コミュニケーター内の全 MPI プロセス数-1 以下の整数値です。当該プロセスのランクは，コミュニケーターを引数として手続 MPI_COMM_RANK を呼び出すことで得ることができます。

利用者は，任意の MPI プロセスを集めてグループを作り，このグループに対してコミュニケーター MPI_COMM_WORLD 以外のコミュニケーターを定義することができます。新たにコミュニケーターを定義すると，そのコミュニケーターの中のプロセスは，新たにランクが与えられます。したがって，ある MPI プロセスが，2 つのコミュニケーターに属している場合，それぞれのコミュニケーターにおいて，一般には異なるランクが与えられることになります。

2.4 動的プロセス管理

2.4.1 動的プロセス生成

動的プロセス生成機能により，プログラム実行中，動的に MPI プロセスを新たに生成することができます。MPI プロセスの動的生成を行うための手続として MPI_COMM_SPAWN および MPI_COMM_SPAWN_MULTIPLE があります。

手続 MPI_COMM_SPAWN は，同一の実行可能プログラムを用いて複数の MPI プロセスを生成する場合に用います。一方，手続 MPI_COMM_SPAWN_MULTIPLE では，異なる実行可能プログラムを指定することができます。

同一の実行可能プログラムを用いて複数の MPI プロセスを生成する場合であっても，各 MPI プロセスに与える引数が異なる場合は手続 MPI_COMM_SPAWN_MULTIPLE を利用します。

MPI プロセスの動的生成を行った場合，生成された全ての MPI プロセスを含むコミュニケーター MPI_COMM_WORLD が，(既存のコミュニケーターMPI_COMM_WORLD とは別に)生成されます。また，既存のコミュニケーター MPI_COMM_WORLD と新たに生成されたコミュニケーター MPI_COMM_WORLD を包含するインターコミュニケーターが生成されます。

手続 MPI_COMM_SPAWN または MPI_COMM_SPAWN_MULTIPLE によって生成された MPI プロセスは，MPI 実行コマンドによって起動された場合と同様に動作します。手続 MPI_COMM_GET_PARENT を利用して，親プロセスグループとの通信に必要なインターコミュニケーターを取得することが可能です。

2.4.2 動的プロセス結合

動的プロセス結合機能により，別々に起動された MPI プログラムを互いに結合し，MPI 手続を利用して通信を行うことができます。本機能はクライアント/サーバ型の接続形態をとり，サーバ側の MPI プログラムは手続 MPI_OPEN_PORT により接続要求を受けつけるポートの準備を行い，手続 MPI_COMM_ACCEPT により接続要求を受理します。クライアント側の MPI プログラムは，手続 MPI_COMM_CONNECT を利用し，サーバ側への接続を要求します。

本機能を利用し MPI プログラムの結合を行った場合，サーバ側 MPI プログラムとクライアント側 MPI プログラムを包含するインターコミュニケーターが生成され，MPI プログラム間の通信はそのインターコミュニケーターを利用して行います。また，動的に結合された MPI プログラム間の通信は，TCP/IP プロトコルを利用して行われます。

本機能を利用して結合 および 通信することのできる MPI プログラムは，同一の数値記憶単位を利用したプログラムに限られます。

2.5 プロセス間通信

プロセス間通信機能は，以下のように大きく三つに分類することができます。

1 対 1 通信	一対の MPI プロセス間でメッセージのやり取りを行う通信（2.5.1 項)。
集団操作	ある MPI プロセスの集合に属す全ての MPI プロセスが同時に行う操作/通信(2.5.2 項)。
片側通信	ある MPI プロセスが一方的に他の MPI プロセスの特定のメモリ空間に対してデータの書込み，あるいは読込みを行う通信形態（2.5.3 項)。

以下ではこれらの通信機能について説明しています。

2.5.1 1 対 1 通信

1 対 1 通信は，2 つの MPI プロセスの間で，一方で送信手続を呼出し，もう一方で受信手続を呼び出すことで，通信が成立します。送信手続や受信手続の呼出しの引数には，通信対象のデータ，データ型，データの大きさ，ならびに 通信相手を識別するためのコミュニケーター および ランクがあり，さらに通信メッセージ自体を識別するためのタグがあります。このタグの値は，送信手続では通信メッセージに付加され，受信手続では受け取る通信メッセージを選択するために使用されます。通信メッセージには，利用者のデータだけでなく，そのデータの型や大きさ，そしてランクやタグの情報なども含まれます。

受信手続の呼出しで，タグに MPI 予約名 MPI_ANY_TAG を指定することによって，ランクで指定した通信相手から送られる通信メッセージを，その通信メッセージのタグの値に関係なく受け取ることができます。また，MPI 予約名 MPI_ANY_SOURCE をランクに指定することによって通信相手を特定することなく指定したタグの値をもつ通信メッセージを受け取ることもできます。MPI_ANY_SOURCE と MPI_ANY_TAG を同時に指定した場合には，通信相手もタグも特定することなく，送られてきた通信メッセージを受け取ります。送信手続の呼出しでは，MPI_ANY_SOURCE と MPI_ANY_TAG のいずれも指定することはできません。

2.5.1.1 通信の種類

1 対 1 通信には，同期モード，バッファモード，レディモード，および 標準モードの 4 つの通信モードがあります。この 4 つのモードの選択は，送信手続名の記述によって行います。受信手続名は，全てのモードに共通で MPI_RECV です。

- 同期モード

  同期モード通信は，対応する送信手続と受信手続のどちらかを先に呼出した場合に，両方の呼出しが揃うまで，先行した側の MPI プロセスは待ち合わせます。

  同期モードの送信手続の名前は MPI_SSEND です。

  以下に同期モードを使った Fortran と C プログラムの例を示します。

```
SUBROUTINE SUB(DATA, N)
use mpi
INTEGER MYRANK,IERROR,IRANK,ITAG
INTEGER DATA(N)
! "STATUS" is an output argument that provides information
! about the received message
!  Define the type of a handle, status
INTEGER STATUS(MPI_STATUS_SIZE)
! Each process in MPI_COMM_WORLD finds out its rank
CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)

ITAG=10
IF(MYRANK.EQ.0) THEN
  IRANK=1
! synchronous mode operation of data to the destination process with rank = 1
  CALL MPI_SSEND(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IERROR)
ENDIF
IF(MYRANK.EQ.1) THEN
  IRANK=0
! receive operation of data from the source process with rank = 0
  CALL MPI_RECV(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,STATUS,IERROR)
ENDIF

RETURN
END
```

```
int sub(int *data, int n){
 int myrank, irank, itag;
/* "status" is an output argument that provides information about the received message */
/* Define the type of a handle, status */
 MPI_Status status;
/* Each process in MPI_COMM_WORLD finds out its rank */
```

```
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

 itag=10;
 if(myrank==0){
  irank=1;
/* synchronous mode operation of data to the destination process with rank = 1 */
  MPI_Ssend(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD);
 }
 if(myrank==1){
  irank=0;
/* receive operation of data from the source process with rank = 0 */
  MPI_Recv(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD, &status);
 }
 return MPI_SUCCESS;
}
```

- バッファモード

  バッファモード通信は，利用者があらかじめ指定したバッファを使用する通信です。送信手続は，指定された利用者バッファに通信メッセージをいったん格納するため，受信手続の呼出しよりも送信手続の呼出しが先行しても同期モードのように待ち合わせることはありません。

  バッファモードの送信手続名は MPI_BSEND です。

  この機能を使用するためには，必要な大きさのバッファを事前に確保し，手続 MPI_BUFFER_ATTACH によって登録しておかなければなりません。利用者バッファには，NEC MPI の管理用データも格納されるので，送信するメッセージの大きさに加えて，MPI の定数 MPI_BSEND_OVERHEAD バイト分の大きさが，各メッセージに対して余分に必要となることに注意してください。バッファの利用が終了したら，手続 MPI_BUFFER_DETACH によって，バッファの登録を解除します。

  以下にバッファモードの通信を使った Fortran と C プログラムの例を示します。

```
SUBROUTINE SUB(DATA,N,BUFF,M)
use mpi
INTEGER MYRANK, IERROR,IRANK,ITAG
INTEGER DATA(N),BUFF(M),SIZE
INTEGER STATUS(MPI_STATUS_SIZE)

CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)
! allocation of a buffer with the size m*4 including an extra space
! for MPI_BSEND_OVERHEAD
CALL MPI_BUFFER_ATTACH(BUFF, M*4, IERROR)
```

```
ITAG=10
IF(MYRANK.EQ.0) THEN
  IRANK=1
! buffered mode send operation of data to the destination process
!  with rank = 1
  CALL MPI_BSEND(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IERROR)
ENDIF
IF(MYRANK.EQ.1) THEN
  IRANK=0
  CALL MPI_RECV(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,STATUS,IERROR)
ENDIF
! deallocation of a buffer
CALL MPI_BUFFER_DETACH(BUFF,SIZE,IERROR)

RETURN
END
```

```
int sub(int *data, int n, int m, int *buf){
int myrank, irank, itag, size;
 MPI_Status status;
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
/* allocation of a buffer with the size m*4 including an extra
space for MPI_BSEND_OVERHEAD */
 MPI_Buffer_attach(buf,m*4);
 itag=10;

 if(myrank==0){
  irank=1;
/* buffered mode send operation of data to the destination process with rank = 1 */
  MPI_Bsend(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD);
 }

 if(myrank==1){
  irank=0;
  MPI_Recv(data, n, MPI_INT, irank, itag, MPI_COMM_WORLD, &status);
 }
/* deallocation of a buffer */
```

```
 MPI_Buffer_detach(buf,&size);

 return MPI_SUCCESS;
}
```

- レディモード

  レディモード通信では，受信手続の呼出しが送信手続の呼出しよりも先行しなければなりません。送信手続の呼出しが先行した場合には，異常が検出される場合があります。

  レディモードの送信手続名は MPI_RSEND です。

  以下にレディモードの通信を使った Fortran プログラムの例を示します。このプログラムで，ランク 0 の MPI プロセスの送信手続呼出しが，ランク 1 の MPI プロセスの受信手続呼出しよりも先行すると送信手続では異常が検出される場合があります。

  (注) NEC MPI では，レディモードは標準モードと同等であり，送信手続が受信手続に先行した場合でも異常は検出されません。

```
SUBROUTINE SUB(DATA, N)
use mpi
INTEGER MYRANK,IERROR,IRANK,ITAG
INTEGER DATA(N)
INTEGER STATUS(MPI_STATUS_SIZE)

CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)

ITAG=10
IF(MYRANK.EQ.1) THEN
  IRANK=0
  CALL MPI_RECV(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,STATUS,IERROR)
ENDIF

IF(MYRANK.EQ.0) THEN
  IRANK=1
! ready mode operation of data to the destination process with rank = 1
  CALL MPI_RSEND(DATA,N,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IERROR)
ENDIF

IF(IERROR.NE.0) THEN
   WRITE(6,*)'ABNORMAL COMPLETED ON MPI_RSEND'
ENDIF
```

```
RETURN
END
```

```
int sub(int *data, int n){
 int myrank, irank, itag, size;
 MPI_Status status;

 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 itag=10;

 if(myrank==1){
  irank=0;
  MPI_Recv(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD, &status);
 }

 if(myrank==0){
  irank=1;
/* ready mode send operation of data to the destination process with rank = 1 */
  MPI_Rsend(data, n, MPI_INT, irank, itag, MPI_COMM_WORLD);
 }

 return MPI_SUCCESS;
}
```

- 標準モード

  標準モード通信では，バッファモード または 同期モードのいずれかの通信モードを適用します。ただし，バッファモードで使用するバッファは利用者が確保するのではなく，システムが準備します。いずれのモードを選択するかは，通信データの大きさや，通信の経路によって処理系が実行時に決定します。

  標準モードの送信手続名は MPI_SEND です。

  標準モードの通信手続を使ってプログラミングする場合，バッファモードが選択されると問題なく実行できても同期モードの適用によってデッドロックが発生する場合があることに注意が必要です。

  たとえば，以下のプログラムのように，ランク 0 の MPI プロセスでは，タグとして ITAG1，ITAG2 の順番で送信し，ランク 1 の MPI プロセスでは，タグとして ITAG2，ITAG1 の順番で受信するようプログラミングしていると，バッファモードの通信であれば問題なく実行できますが，処理系によっ

て同期モードが選択されると送信側，受信側それぞれが最初の通信手続の呼出しで相手側の 2 番目の通信手続の呼出しを待ち合わせるため，デッドロックが発生します。

```
SUBROUTINE SUB(DATA1, N1, DATA2, N2)
use mpi
INTEGER MYRANK,IERROR,IRANK,ITAG1, ITAG2
INTEGER DATA1(N1), DATA2(N2)
INTEGER STATUS(MPI_STATUS_SIZE)

CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)

ITAG1=10
ITAG2=20

IF(MYRANK.EQ.0) THEN
  IRANK=1
! standard mode send operations of DATA1 and DATA2 to the destination
! process with rank = 1
  CALL MPI_SEND(DATA1,N1,MPI_INTEGER,IRANK,ITAG1,MPI_COMM_WORLD,IERROR)
  CALL MPI_SEND(DATA2,N2,MPI_INTEGER,IRANK,ITAG2,MPI_COMM_WORLD,IERROR)
ENDIF

IF(MYRANK.EQ.1) THEN
  IRANK=0
! receive operations of DATA2 and DATA1 from the source process
! with rank = 0
  CALL MPI_RECV(DATA2,N2,MPI_INTEGER,IRANK,ITAG2,MPI_COMM_WORLD,STATUS,IERROR)
  CALL MPI_RECV(DATA1,N1,MPI_INTEGER,IRANK,ITAG1,MPI_COMM_WORLD,STATUS,IERROR)
ENDIF

RETURN
END
```

```
int sub(int *data1, int n1, int *data2, int n2){
 int myrank, irank, itag1, itag2;
 MPI_Status status;

 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
```

```
 itag1 = 10;
 itag2 = 20;
 if(myrank==0){
  irank=1;
/* standard mode send operations of data1 and
  data2 to the destination process with rank = 1 */
  MPI_Send(data1,n1,MPI_INT,irank, itag1, MPI_COMM_WORLD);
  MPI_Send(data2,n2,MPI_INT,irank, itag2, MPI_COMM_WORLD);
 }

 if(myrank==1){
  irank=0;
/* receive operations of data2 and data1 from the source process with rank = 0 */
  MPI_Recv(data2, n2, MPI_INT, irank, itag2, MPI_COMM_WORLD, &status);
  MPI_Recv(data1, n1, MPI_INT, irank, itag1, MPI_COMM_WORLD, &status );
 }

 return MPI_SUCCESS;
}
```

2.5.1.2 非ブロッキング通信

前節までに説明した 4 つのモードの通信をブロッキング通信ともいいます。1 対 1 通信には，この 4 つの通信モード全てについて非ブロッキング通信があります。

ブロッキング通信は，送信手続の中で通信メッセージの作成や送り出しなどを行い，受信手続の中で通信メッセージの選択や受け取りまでを行いますが，非ブロッキング通信の場合は，通信開始手続と通信完了確認手続があり，通信開始手続では，引数の取り込みと，通信識別子の生成後，制御は利用者プログラムに戻ります。したがって，通信メッセージの作成や送り出し， または 通信メッセージの選択や受け取りなどといった本来の通信処理は，利用者プログラムの処理と並行して MPI の処理系で行われます。

通信識別子は，通信開始手続と通信完了確認手続の対応を取るために用いられます。通信完了の確認は，この通信識別子を引数とする通信完了確認手続の呼出しによって行います。

非ブロッキング通信を利用する場合に注意する点は，通信開始手続の呼出しから通信完了確認手続を呼び出すまでの間，通信対象である利用者データについて送信側プロセスであれば更新を，受信側プロセスでは更新 または 参照を行ってはならないことです。

非ブロッキング通信の手続の名前は，ブロッキング通信の手続の名前に対して接頭辞 MPI_の直後に I を付加した形式をとります。すなわち，受信開始手続の名前は，MPI_IRECV となり，送信開始手続の名前は，同期モードであれば MPI_ISSEND，バッファモードであれば MPI_IBSEND，レディモードであれば MPI_IRSEND，標準モードであれば MPI_ISEND となります。

通信完了確認手続の名前は MPI_WAIT で，全てのモードの送信手続と受信手続に共通です。送信と受信の対応は，ブロッキング通信と非ブロッキング通信を一致させる必要はありません。非ブロッキングの送信に対して，ブロッキングの受信を行うことができ，その逆も可能です。

以下に送信と受信ともに標準モードの非ブロッキング通信を使った Fortran プログラムの例を示します。

```
SUBROUTINE SUB(DATA1, N1, DATA2, N2)
use mpi
INTEGER MYRANK, IERROR, IRANK, ITAG
INTEGER ISTAT(MPI_STATUS_SIZE)
! Define the type of two handles, IREQ0 and IREQ1
INTEGER IREQ0, IREQ1
DOUBLE PRECISION DATA1(N1), DATA2(N2)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYRANK,IERROR)

ITAG1=10
IF(MYRANK.EQ.0) THEN
  IRANK=1
! nonblocking standard mode send operation of DATA1 to the process with rank=1
  CALL MPI_ISEND(DATA1,N1,MPI_INTEGER, IRANK,ITAG,MPI_COMM_WORLD,IREQ0,IERROR)
ENDIF

IF(MYRANK.EQ.1) THEN
  IRANK=0
! nonblocking standard mode receive operation of DATA1 from the process with rank=0
  CALL MPI_IRECV(DATA1,N1,MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IREQ1,IERROR)
ENDIF

DO I=1,N2
  DATA2(I)=SQRT(REAL(I))
ENDDO

IF(MYRANK.EQ.0) THEN
!  wait until the send operation completes
  CALL MPI_WAIT(IREQ0,ISTAT,IERROR)
ENDIF

IF(MYRANK.EQ.1) THEN
! wait until the receive operation completes
  CALL MPI_WAIT(IREQ1,ISTAT,IERROR)
```

```
ENDIF
RETURN
END
```

```
int sub(double *data1, int n1, double *data2, int n2){
 int myrank, irank, itag;
/* Define the type of two handles, ireq0 and ireq1 */
 MPI_Request ireq0, ireq1;
 MPI_Status status;

 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

 itag=10;
 if(myrank==0){
  irank=1;
/* nonblocking standard mode send operation of data1 to the process with rank=1 */
  MPI_Isend(data1,n1,MPI_INT,irank, itag, MPI_COMM_WORLD, &ireq0);
  }
 if(myrank==1){
  irank=0;
/* nonblocking standard mode receive operation of data1 from the process with rank=0 */
  MPI_Irecv(data1,n1,MPI_INT,irank, itag, MPI_COMM_WORLD, &ireq1);
 }
 for(int i=0;i<n2;i++)data2[i]=sqrt((double) (i+1));
/* wait until the send operation completes */
 if(myrank == 0)MPI_Wait(&ireq0,&status);
/* wait until the receive operation completes */
 if(myrank == 1)MPI_Wait(&ireq1,&status);

 return MPI_SUCCESS;
}
```

2.5.1.3 通信メッセージの事前確認

1 対 1 通信では，受信手続を呼び出す前に，目的の通信メッセージが送られてきており受け取ることができるかどうか事前に確認することができます。この通信メッセージ問合せ手続として MPI_PROBE および MPI_IPROBE があります。目的の通信メッセージは，送信側 MPI プロセスのコミュニケーター および ランクの値と，通信メッセージのタグの値を引数に指定することによって確認します。手続 MPI_PROBE は，

目的の通信メッセージが受け取れる状態になるまで待ち合わせます。手続 MPI_IPROBE は，待合せは行わず，受け取れるか否かの状態だけを返却します。

2.5.2 集団操作

集団操作とは，特定のコミュニケーターに対応するグループに含まれる全ての MPI プロセスが，集団操作の手続を呼出して，通信や実行制御を行う機能です。集団操作手続の呼出しでは，引数に必ずコミュニケーターを指定します。

集団操作に利用できるコミュニケーターには，イントラコミュニケーター(intra-communicator)とインターコミュニケーター(inter-communicator)の 2 種類が用意されています。

本節では，集団操作のうち主要な機能である同期制御，同報通信，収集通信，および 拡散通信について説明します。

2.5.2.1 同期制御

同期制御の手続名は，MPI_BARRIER です。

引数で指定されたコミュニケーターに含まれる全ての MPI プロセスがこの同期制御手続を呼び出すまで待ち合わせます。

コミュニケーターとしてインターコミュニケーターが指定された場合，ローカルグループ および リモートグループの全ての MPI プロセスが手続 MPI_BARRIER を呼び出すまで待合せを行います。

以下に同期制御(バリア)を使った Fortran プログラムの例を示します。この例では，イントラコミュニケーターMPI_COMM_WORLD を利用しています。

```
SUBROUTINE SUB(DATA, N)
use mpi
INTEGER MYRANK,IERROR,IRANK,ITAG
INTEGER DATA(N)
INTEGER IREQ
INTEGER STATUS(MPI_STATUS_SIZE)

CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)

ITAG=10

IF(MYRANK.EQ.1) THEN
  IRANK=0
  CALL MPI_IRECV(DATA,N, MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IREQ,IERROR)
ENDIF
```

```
! Ensure that all processes in the communicator have completed
! the MPI_Irecv operation

CALL MPI_BARRIER(MPI_COMM_WORLD,IERROR)

IF(MYRANK.EQ.0) THEN
  IRANK=1
  CALL MPI_RSEND(DATA,N, MPI_INTEGER,IRANK,ITAG,MPI_COMM_WORLD,IERROR)
ENDIF

IF(MYRANK.EQ.1) THEN
 CALL MPI_WAIT(IREQ,STATUS,IERROR)
ENDIF

RETURN
END
```

```
int sub(int *data, int n){
 int myrank, irank, itag;
 MPI_Request ireq;
 MPI_Status status;

 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

 itag=10;

 if(myrank==1){
  irank=0;
  MPI_Irecv(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD, &ireq);
 }
/* Ensure that all processes in the communicator have completed the MPI_Irecv operation */
 MPI_Barrier(MPI_COMM_WORLD);

 if(myrank==0){
  irank=1;
  MPI_Isend(data,n,MPI_INT,irank, itag, MPI_COMM_WORLD, &ireq);
 }
```

```
 if(myrank == 1)MPI_Wait(&ireq,&status);

 return MPI_SUCCESS;
}
```

2.5.2.2 同報通信

同報通信の手続名は，MPI_BCAST です。

この手続の引数には，コミュニケーター以外にデータ領域，データの型，データの大きさ，および ランクがあります。このランクには，手続を呼び出す全ての MPI プロセスにおいて同じ値を指定しなければなりません。同報通信では，このランクの値をもつ MPI プロセスがデータの送信側となり，それ以外の MPI プロセスが受信側となります。このランクの値をもつ MPI プロセスをルートプロセスといいます。ルートプロセスは同報通信だけでなく後節で説明する収集通信や拡散通信でも引数として指定し，集団操作のプロセスグループの中で送信者 または 受信者となる MPI プロセスを特定します。

コミュニケーターとしてインターコミュニケーターが指定された場合，そのインターコミュニケーターを構成する 2 つのプロセスグループのうち，片方のプロセスグループにだけルートプロセスが存在します。ルートプロセスのもつデータは，ルートプロセスの所属していないプロセスグループ中の全 MPI プロセスに対して転送されます。

以下に同報通信を使った Fortran プログラムの例を示します。この例では，ランク値の引数に指定したランク 0 の MPI プロセスがルートプロセスとして DATA を送信し，ランク 0 以外の MPI プロセスが DATA を受信します。

```
SUBROUTINE SUB(DATA, N)
use mpi
INTEGER IERROR,IROOT,MYRANK
INTEGER DATA(N)

CALL MPI_COMM_RANK(MPI_COMM_WORLD,MYRANK,IERROR)

IF(MYRANK.EQ.0) THEN
  DO I = 1,N
    DATA(I) = I
  ENDDO
ENDIF
! broadcast send operation of data from the process with rank=0 to
! all other processes in MPI_COMM_WORLD
IROOT = 0
CALL MPI_BCAST(DATA,N,MPI_INTEGER,IROOT,MPI_COMM_WORLD,IERROR)
```

```
RETURN
END
```

```
int sub(int *data, int n){
 int myrank, iroot;
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);

 if(myrank == 0){
  for(int i=0; i<n; i++)data[i]=i;
 }
 iroot = 0;
/* broadcast send operation of data from the process
 with rank=0 to all other processes in MPI_COMM_WORLD */
 MPI_Bcast(data, n, MPI_INT, iroot, MPI_COMM_WORLD);

 return MPI_SUCCESS;

}
```

2.5.2.3 収集通信 および 拡散通信

収集通信を行う手続 MPI_GATHER は，複数の MPI プロセスから 1 つの MPI プロセスへデータを収集する場合に使用します。

拡散通信を行う手続 MPI_SCATTER は，1 つの MPI プロセスから複数の MPI プロセスへデータを散布する場合に使用します。拡散通信と同報通信の違いは，同報通信の場合，同じデータを複数の MPI プロセスに送信するのに対して，拡散通信の場合，送信データを分割して別々の部分をそれぞれ別々の MPI プロセスに送信することにあります。

インターコミュニケーターを指定した手続 MPI_GATHER によるデータの収集操作は，インターコミュニケーターを構成する片方のプロセスグループ内に閉じて行われます。その結果が他方のグループのルートプロセスへ転送されます。手続 MPI_SCATTER の場合は，ルートプロセス上のデータがルートプロセスの所属しないプロセスグループ中の各プロセスに対して分配されます。

以下に手続 MPI_GATHER を使った Fortran プログラムの例を示します。ここで，N は配列 ALLDATA の要素の個数，M は配列 PARTDATA の要素の個数ですが，N の値は，コミュニケーター MPI_COMM_WORLD に含まれるプロセス数×M 以上でなければなりません。

```
SUBROUTINE SUB(ALLDATA, N, PARTDATA, M)
```

```
use mpi
INTEGER IERROR,IROOT
INTEGER ALLDATA(N),PARTDATA(M)

! gather operation of the buffer PARTDATA from all processes in MPI_COMM_WORKLD
! to the buffer ALLDATA on the root process with rank=0
IROOT=0
CALL MPI_GATHER(PARTDATA,M,MPI_INTEGER,ALLDATA,M,MPI_INTEGER, IROOT,  &
       MPI_COMM_WORLD,IERROR)

RETURN
END
```

```
int sub(int *alldata, int n, int *partdata, int m){
 int iroot;
 iroot = 0;
/* gather operation of the buffer partdata from all processes in  MPI_COMM_WORKLD
 to the buffer alldata on the root process with rank=0 */
 MPI_Gather(partdata, m, MPI_INT,alldata, m, MPI_INTEGER,iroot, MPI_COMM_WORLD);

 return MPI_SUCCESS;
}
```

次に手続 MPI_SCATTER を使った Fortran プログラムの例を示します。ここで，N は配列 ALLDATA の要素の個数，M は配列 PARTDATA の要素の個数ですが，N の値はコミュニケーターMPI_COMM_WORLD に含まれるプロセス数×M 以上でなければなりません。

```
SUBROUTINE SUB(ALLDATA, N, PARTDATA, M)
use mpi
INTEGER IERROR,IROOT
INTEGER ALLDATA(N),PARTDATA(M)

! scatter operation of the buffer ALLDATA from the root process with rank=0
! to the buffers PARTDATA on all processes in MPI_COMM_WORKLD
IROOT=0
CALL MPI_SCATTER(ALLDATA,M,MPI_INTEGER,PARTDATA,M,MPI_INTEGER,   &
IROOT, MPI_COMM_WORLD,IERROR)
```

```
RETURN
END
```

```
int sub(int *alldata, int n, int *partdata, int m){
 int iroot;

 iroot = 0;
/* scatter operation of of the buffer alldata from the root process with rank=0 to buffers
 partdata on all processes in MPI_COMM_WORKLD */
 MPI_Scatter(partdata, m, MPI_INT,alldata, m, MPI_INTEGER,iroot, MPI_COMM_WORLD);

 return MPI_SUCCESS;
}
```

2.5.2.4 非ブロッキング通信

前節までに説明した集団操作はブロッキング手続であり，集団操作手続の呼出しによりデータ転送や演算が行われ，それらの処理が完了した後に利用者プログラムへ制御が戻ります。一方，1 対 1 通信の場合と同様に，集団操作にも非ブロッキング手続があります。非ブロッキング集団操作手続名は，ブロッキング手続名の接頭辞 MPI_の直後に I を付加した形式をとり，たとえば，ブロッキング手続 MPI_BCAST に対し，非ブロッキング手続は，MPI_IBCAST となります。

非ブロッキング集団操作手続の呼出しにより集団操作の処理が開始され，その完了前に手続から制御が戻ります。この場合，通信識別子が生成 および 返却され，その通信識別子を引数として通信完了確認手続を実行することにより，非ブロッキング集団操作の完了を待ち合わせます。このように，非ブロッキング集団操作手続は非ブロッキング 1 対 1 通信手続と同様に利用することができますが，以下の点が異なります。

- 非ブロッキング集団操作はブロッキング集団操作とは互いに対応しません。1 対 1 通信の場合は，送信側が非ブロッキングで受信側がブロッキング，あるいは，その逆であっても通信は可能ですが，集団操作の場合，たとえば，手続 MPI_BCAST と手続 MPI_IBCAST とは対応しません。
- 同一コミュニケーターを引数とする非ブロッキング集団通信は全プロセスで同じ順序で実行しなければなりません。
- 完了確認手続の呼出しにより設定される MPI_STATUS オブジェクトのうち，エラー状態（MPI_ERROR）だけが設定され，送信元ランク（MPI_SOURCE） および タグ値（MPI_TAG）は不定となります。
- 手続 MPI_REQUEST_FREE および MPI_CANCEL を利用することできません。

2.5.2.5 HPC-X (SHARP) サポート

NEC MPI は Mellanox 社が提供する HPC-X の SHARP 機能を利用し、集団通信を InfiniBand ネットワークにオフロードする機能をサポートしています。本機能により、MPI 集団通信（MPI_BARRIER および MPI_ALLREDUCE）の通信性能およびスケーラビリティを向上させることができます。SHARP 機能の利用については、3.2.4 節で説明される環境変数で制御することが可能です。

なお、本機能は、InfiniBand および SHARP 機能が利用可能な環境が構築されており、かつ、すべての MPI プロセスが VE 上で動作している場合にのみ利用可能です（ベクトルプロセスとスカラプロセスが混在するハイブリッド実行の場合は利用できません）。

2.5.3 片側通信

片側通信(one-sided communication)は，データ転送を行うために，他の MPI プロセスとの協調を必要としない通信形態です。つまり，ある MPI プロセスが単独で他の MPI プロセスに対してデータ転送(読込み または 書込み操作)を行うことができます。

2.5.3.1 ウィンドウ

片側通信を利用するためには，他プロセスからのアクセスを許可するメモリ領域を「ウィンドウ」としてあらかじめ登録する必要があります。ウィンドウの登録を行うには，手続 MPI_WIN_CREATE，MPI_WIN_ALLOCATE，MPI_WIN_ALLOCATE_SHARED，または MPI_WIN_CREATE_DYNAMIC を使用します。手続 MPI_WIN_CREATE_DYNAMIC を使用してウィンドウを登録する場合は，手続 MPI_WIN_ATTACH により，ウィンドウとして登録する領域を関連付ける操作が別途必要となります。

ウィンドウの登録を抹消するためには，手続 MPI_WIN_FREE を利用します。これらの手続の呼出しは全て集団操作です。

2.5.3.2 データ転送手続

片側通信を行うためには，次の手続を利用します。

- MPI_GET
- MPI_PUT
- MPI_ACCUMULATE
- MPI_GET_ACCUMULATE
- MPI_FETCH_AND_OP
- MPI_COMPARE_AND_SWAP
- MPI_RGET
- MPI_RPUT
- MPI_RACCUMULATE
- MPI_RGET_ACCUMULATE

手続 MPI_GET は，転送対象プロセスのウィンドウ領域からデータの読込みを行います。逆に，手続 MPI_PUT は，転送対象プロセスのウィンドウ領域に対してデータの書込みを行います。手続 MPI_ACCUMULATE は，手続 MPI_PUT と同様に転送対象プロセスのウィンドウ領域にデータを書込みますが，書込み対象領域の内容を単に更新(上書き)するのではなく，指定された演算を施した後，その演算結果によって対象領域の更新を行います。手続 MPI_GET_ACCUMULATE は，手続 MPI_ACCUMULATE と同様に転送対象プロセスのウィンドウ領域に指定された演算を行い更新する操作に加え，演算前のデータの読込みを行います。

手続 MPI_FETCH_AND_OP は，手続 MPI_GET_ACCUMULATE と同様に，演算前の値の取得と演算結果による更新を行いますが，転送長が一要素となります。手続 MPI_COMPARE_AND_SWAP は，転送元で指定したデータと転送対象プロセス側のデータが等しい場合に限り，転送対象プロセス側のデータを置換します。転送対象プロセス側のデータは常に読み込みます。

手続 MPI_RGET，MPI_RPUT，MPI_RACCUMULATE，および MPI_RGET_ACCUMULATE は，それぞれ手続 MPI_GET，MPI_PUT，MPI_ACCUMULAT，および MPI_GET_ACCUMULATE と同様なデータ転送処理を開始し，通信識別子を生成 および 返却します。これらデータ転送の完了待合せには，手続 MPI_WAIT または MPI_TEST などを使用します。なお，片側通信の同期手続（後述）によってもデータ転送処理は完了しますが，その場合も手続 MPI_WAIT または MPI_TEST などの呼出しが必要です。

2.5.3.3 同期手続

片側通信の同期を取るために，ポスト/ウェイト型，フェンス型，および ロック型の 3 種類の同期機構が提供されています。

- ポスト/ウェイト型同期

  ウィンドウ領域のアクセス許可とアクセス要求の区間を明示する同期機構です。アクセス許可 またはアクセス要求の区間を明示する場合，同時にアクセスを許可するプロセス集合(グループ) または アクセスを要求するプロセス集合も明示する必要があります。

  ポスト/ウェイト型同期機構では，きめ細かい同期制御が可能であり，同期を最小限にとどめるために利用されます。

  ポスト/ウェイト型同期機構は，手続 MPI_WIN_POST(アクセス許可区間開始)，MPI_WIN_WAIT(アクセス許可区間終了)，MPI_WIN_TEST(アクセス許可区間終了，非ブロッキング検査)，MPI_WIN_START(アクセス要求区間開始)，および MPI_WIN_COMPLETE(アクセス要求区間終了)により提供されます。

  手続 MPI_WIN_POST，MPI_WIN_WAIT，および MPI_WIN_TEST によるアクセス許可区間の制御は，ウィンドウをもつプロセスが実行します。

  また，手続 MPI_WIN_START および MPI_WIN_COMPLETE によるアクセス要求区間の制御は，ウィンドウへアクセスするプロセスが実行します。

- フェンス型同期

ウィンドウをアクセスする可能性のある(ウィンドウの登録を行った時に指定したコミュニケーターに含まれる)全てのプロセスが呼び出さなければなりません。フェンス型同期を行うためには，手続 MPI_WIN_FENCE を利用します。フェンス型同期機構は，ポスト/ウェイト型同期機構と比較して大きな処理区間を対象とした同期を行うために利用します。

- ロック型同期
  共有資源(この場合はウィンドウ)の排他制御機構を提供します。ロック型同期を行うために，手続 MPI_WIN_LOCK(アクセス権獲得)， MPI_WIN_UNLOCK(アクセス権解放)，MPI_WIN_LOCK_ALL（全プロセス一括アクセス権獲得)，および MPI_WIN_UNLOCK（全プロセス一括アクセス権解放）を利用します。これらの手続はウィンドウへのアクセスを行うプロセスで実行します。

2.5.3.4 最適化情報(assertions)

手続 MPI_WIN_POST， MPI_WIN_START， MPI_WIN_FENCE， MPI_WIN_LOCK，および MPI_WIN_LOCK_ALL に対して最適化情報を引数として渡すことができます。最適化情報には，

- MPI_MODE_NOCHECK
- MPI_MODE_NOSTORE
- MPI_MODE_NOPUT
- MPI_MODE_NOPRECEDE
- MPI_MODE_NOSUCCEED

の五つがあります。これらの最適化情報によって，プログラムの意味(実行結果)が変化することはありません。与えるべき最適化情報がない場合は 0 を渡します。

最適化情報の意味，および NEC MPI が実際に最適化に利用する情報の関係を**表 2-1** に示します。

表 2-1 片側通信における最適化情報

手続名:MPI_WIN_POST		
情報名	意味	NEC MPI での利用
MPI_MODE_NOCHECK	手続 MPI_WIN_START の対応する呼出しが，未発行であることを示します。MPI_MODE_NOCHECK は手続 MPI_WIN_START および MPI_WIN_POST の両方で指定しなければなりません。	利用します
MPI_MODE_NOSTORE	前回の同期操作以降，自プロセスのウィンドウが更新されていないことを示します。	無視されます

MPI_MODE_NOPUT	手続 MPI_WIN_POST の呼出し以降，自プロセスのウィンドウが更新されないことを示します。	無視されます
手続名:MPI_WIN_START		
情報名	意味	NEC MPI での利用
MPI_MODE_NOCHECK	手続 MPI_WIN_POST の対応する呼出しが，既に行われていることを示します。MPI_MODE_NOCHECK は，手続 MPI_WIN_START および MPI_WIN_POST の両方で指定しなければなりません。	利用します
手続名:MPI_WIN_FENCE		
情報名	意味	NEC MPI での利用
MPI_MODE_NOSTORE	前回の同期以降，自プロセスのウィンドウが更新されていないことを示します。	無視されます
MPI_MODE_NOPUT	手続 MPI_WIN_FENCE の次の呼出しまでの間，自プロセスのウィンドウが更新されないことをしめします。	利用します
MPI_MODE_NOPRECEDE	完了待ちを行う片側通信操作が無いことを示します。この情報は全てのプロセスが指定しなければなりません。	利用します
MPI_MODE_NOSUCCEED	今後，片側通信操作を行わないことを示します。この情報は全てのプロセスが指定しなければなりません。	利用します
手続名:MPI_WIN_LOCK，MPI_WIN_LOCK_ALL		
情報名	意味	NEC MPI での利用
MPI_MODE_NOCHECK	他プロセスがロックを取得していない，あるいは，自プロセスがロックを取得している間，他プロセスからのロック取得要求が発生しないことを示しています。	利用します

2.5.3.5 プログラム例

片側通信機能を利用した例として，引数 **root** で指定されたルートプロセスのもつデータを，コミュニケーター**comm** の全てのプロセスに転送する C プログラムを示します。この例では，ルート以外のプロセスで手続 MPI_GET を利用し，同期の手段としては手続 MPI_WIN_FENCE を利用しています。

```
SUBROUTINE PROPAGATE(BUFF,N)
use mpi
INTEGER ROOT, IERROR, RANK, DISP
! Define the type of a window object, WIN
INTEGER WIN
INTEGER BUFF(N)

INTEGER(KIND=MPI_ADDRESS_KIND) OFFSET
OFFSET=8
ROOT=0

CALL MPI_COMM_RANK(MPI_COMM_WORLD, RANK, IERROR)
! Create memory window for process root and all other processes in the communicator
IF(RANK==ROOT) THEN
  CALL MPI_WIN_CREATE(BUFF, OFFSET*N, DISP, MPI_INFO_NULL, MPI_COMM_WORLD,  &
  WIN, IERROR)
ELSE
  CALL MPI_WIN_CREATE(MPI_BOTTOM, OFFSET*0, DISP, MPI_INFO_NULL, MPI_COMM_WORLD, &
WIN, IERROR)
ENDIF

! All other processes get data from process root
CALL MPI_WIN_FENCE(MPI_MODE_NOPRECEDE, WIN, IERROR)
IF(RANK/=ROOT) THEN
  CALL MPI_GET(BUFF,N, MPI_INT, ROOT, OFFSET*0, N, MPI_INT, WIN, IERROR)
ENDIF
! Complete the one-sided communication operation
CALL MPI_WIN_FENCE(MPI_MODE_NOSUCCEED, WIN, IERROR)
! Free the window
CALL MPI_WIN_FREE(WIN, IERROR)

RETURN
END SUBROUTINE PROPAGATE
```

```
int propagate(int *buf, int n, int root, MPI_Comm comm){
/* Define the type of a window object, win */
 MPI_Win win;
 int unit = sizeof(int); /* displacement unit of the window */
```

```
 int rank;

 MPI_Comm_rank(comm, &rank);
/* Create memory window for process root and all other processes in the communicator */
 if (rank == root)
  MPI_Win_create(buf, unit * n, unit, MPI_INFO_NULL, comm, &win);
 else
  MPI_Win_create(NULL, 0, unit, MPI_INFO_NULL, comm, &win);

/* All other processes get data from process root */
 MPI_Win_fence(MPI_MODE_NOPRECEDE, win);
 if (rank != root)
 MPI_Get(buf, n, MPI_INT, root, 0, n, MPI_INT, win);
/* Complete the one-sided communication operation */
 MPI_Win_fence(MPI_MODE_NOSUCCEED, win);
/* Free the window */
 MPI_Win_free(&win);
 return MPI_SUCCESS;
}
```

2.6 並列 I/O

2.6.1 概要

MPI は，複数の MPI プロセスが同一のファイルに同時にアクセスするための仕様が規定されています。MPI データ型を利用したプロセスごとのファイルアクセスパターンの指定や，集団 I/O 操作の提供などにより MPI ライブラリ中での高度な最適化が可能であり，高速に I/O を行うことができます。

また，ファイル上のデータ格納パターンが不連続であったとしても，入出力操作を行う場合，その状態を意識せず，MPI 通信の場合と同様にプログラム中では連続したデータ列として扱うことができます。

2.6.2 ビュー

MPI の規定する並列 I/O(以下，MPI-IO)機能を利用して入出力を行う場合，アクセスの対象となるファイル中で自プロセスがアクセスする部分をあらかじめ指定する必要があります。ここで指定するアクセス対象部分をビューと呼びます。ビューは任意の基本データ型 または 利用者定義データ型によって表現され，多様なアクセスパターンを容易に表現することが可能です。

2.6.3 主要手続

MPI-IO 機能を利用する場合，

(1) オープン
(2) ビューの設定
(3) 入出力操作
(4) クローズ

の操作を行う必要があります。(2)ビューの設定は，一般にはファイルをオープンした直後に行いますが，オープンしているファイルに対しては，プログラムの実行中にビューを再設定して入出力を行うことも可能です。

ファイルのオープンは，手続 MPI_FILE_OPEN，クローズは，手続 MPI_FILE_CLOSE を利用します。ビューの設定には，手続 MPI_FILE_SET_VIEW を利用します。これらの手続はいずれも集団操作です。ビューの設定に派生データ型を使用する場合は，派生データ型生成手続を使用してデータ型を生成・登録しておく必要があります。

MPI-IO では多様なファイルアクセス手続が提供されており，次の全ての組合せに対して，対応する手続が存在します。

(1) 入出力種類
- 入力
- 出力

(2) ファイル位置指定
- 明示的オフセット指定
  ファイルアクセスごとにオフセット値を指定する方式

- 個別ファイルポインタ
  MPI プロセスごとに独立してもっているファイルポインタを利用する方式

- 共有ファイルポインタ
  グループ内全ての MPI プロセスで共有しているファイルポインタを利用する方式

(3) 同期状態
- ブロッキング操作
  入出力操作が終了するまで手続が復帰しない操作

- 非ブロッキング/分離集団操作
  入出力操作が終了を待たずに手続からリターンする操作。
  入出力操作の完了を確認する操作が別に必要(非集団操作の場合は手続 MPI_TEST，MPI_WAIT な

ど。集団操作の場合，手続 MPI_FILE_*XXX*_BEGIN で開始した操作は手続 MPI_FILE_*XXX*_END。*XXX*は位置指定方式により異なります)。

(4) プロセス間協調状態

- 集団操作
  グループ内全てのプロセスが協調して行う操作
- 非集団操作
  他プロセスとの協調を必要とせず，単独で行う操作

表 2-2 は各組合せに対応する手続名を示しています。

表 2-2 ファイルアクセス手続

位置指定	同期状態	プロセス間協調	
		非集団	集団
明示的 オフセット	ブロッキング	MPI_FILE_READ_AT MPI_FILE_WRITE_AT	MPI_FILE_READ_AT_ALL MPI_FILE_WRITE_AT_ALL
	非ブロッキング	MPI_FILE_IREAD_AT MPI_FILE_IWRITE_AT	MPI_FILE_IREAD_AT_ALL MPI_FILE_IWRITE_AT_ALL
	分離集団	N/A	MPI_FILE_READ_AT_ALL_BEGIN MPI_FILE_READ_AT_ALL_END MPI_FILE_WRITE_AT_ALL_BEGIN MPI_FILE_WRITE_AT_ALL_END
個別 ファイル ポインタ	ブロッキング	MPI_FILE_READ MPI_FILE_WRITE	MPI_FILE_READ_ALL MPI_FILE_WRITE_ALL
	非ブロッキング	MPI_FILE_IREAD MPI_FILE_IWRITE	MPI_FILE_IREAD_ALL MPI_FILE_IWRITE_ALL
	分離集団	N/A	MPI_FILE_READ_ALL_BEGIN MPI_FILE_READ_ALL_END MPI_FILE_WRITE_ALL_BEGIN MPI_FILE_WRITE_ALL_END
共有 ファイル ポインタ	ブロッキング	MPI_FILE_READ_SHARED MPI_FILE_WRITE_SHARED	MPI_FILE_READ_ORDERED MPI_FILE_WRITE_ORDERED
	非ブロッキング	MPI_FILE_IREAD_SHARED MPI_FILE_IWRITE_SHARED	N/A
	分離集団	N/A	MPI_FILE_READ_ORDERED_BEGIN MPI_FILE_READ_ORDERED_END

			MPI_FILE_WRITE_ORDERED_BEGIN MPI_FILE_WRITE_ORDERED_END

2.6.4 ファイル名の接頭辞

手続 MPI_FILE_OPEN に引数として渡すファイル名の接頭辞として，MPI-IO で利用するファイルシステムタイプを指定することが可能です。指定可能な接頭辞は"nfs:" および "scatefs:"です。"nfs:"は NFS ファイルシステム，"scatefs:"は ScaTeFS に対して利用します。

2.6.5 プログラム例

以下に MPI-IO 機能を利用した C プログラム例を示します。この例では，共有ファイルポインタを利用する手続 MPI_FILE_SEEK_SHARED および MPI_FILE_WRITE_SHARE を使用しており，*filename* で指定されたファイルへデータが追加されます。

```
PROGRAM MAIN
use mpi
IMPLICIT NONE
INTEGER :: IERR
INTEGER :: SBUF (4)
INTEGER  :: RBUF(16)
INTEGER :: N
INTEGER :: MYRANK
INTEGER :: NPROCS
! Define the type of a file handle, FH
INTEGER :: FH
INTEGER(8) :: DISP = 0
INTEGER(8) :: OFFSET = 0

CALL MPI_INIT(IERR)
CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYRANK, IERR)
CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NPROCS, IERR)
SBUF = MYRANK
RBUF = -1
N = 4

print *, MYRANK, ": SBUF =>", SBUF
! Open a common file with a name "output.dat"
```

```
CALL MPI_FILE_OPEN(MPI_COMM_WORLD, "output.dat", MPI_MODE_RDWR+MPI_MODE_CREATE, &
MPI_INFO_NULL, FH, IERR)
! Set a process's file view
CALL MPI_FILE_SET_VIEW(FH, DISP, MPI_INT, MPI_INT,"native", MPI_INFO_NULL, IERR)

! Write data starting from the current location of the shared file pointer
CALL MPI_FILE_WRITE_SHARED(FH, SBUF, N, MPI_INT, MPI_STATUS_IGNORE, IERR)
! Cause all previous writes to be transferred to the storage device
CALL MPI_FILE_SYNC(FH, IERR)
IF(MYRANK == 0) THEN
! Move an individual file pointer to the location in the file from which
! a process needs to read data
  CALL MPI_FILE_SEEK(FH, OFFSET, MPI_SEEK_SET, IERR)
! Read data from the current location of the individual file pointer in the file
  CALL MPI_FILE_READ(FH, RBUF, N*min(NPROCS,4), MPI_INT, MPI_STATUS_IGNORE, IERR)
  print *, MYRANK, ": RBUF =>", RBUF
ENDIF
! Close the common file
CALL MPI_FILE_CLOSE(FH, IERR)
CALL MPI_FINALIZE(IERR)

END PROGRAM
```

```
#include "mpi.h"
#include <stdio.h>

int main(int argc, char *argv[])
{
 int sbuf[4];
 int rbuf[16];
 int n;
 int myrank;
 int nprocess;
 int disp=0;
 int offset=0;
 int tmp;
/* Define the type of a file handle, fh */
 MPI_File fh;
```

```
 MPI_Init(&argc, &argv);
 MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
 MPI_Comm_size(MPI_COMM_WORLD, &nprocess);

 for(int i=0; i<4; i++)sbuf[i]=myrank;
 for(int i=0; i<4; i++)rbuf[i]=-1;
 n=4;
 printf("%d:sbuf=>",myrank);
 for(int i=0; i<4; ++i)printf("%d", sbuf[i]);
 printf("¥n");

/* Open a common file with a name "output.dat" */
 MPI_File_open(MPI_COMM_WORLD,"output.dat",MPI_MODE_RDWR|MPI_MODE_CREATE, MPI_INFO_NULL,&fh);
/* Set a process's file view */
 MPI_File_set_view(fh,disp,MPI_INT,MPI_INT,"native",MPI_INFO_NULL);

/* Write data starting from the current location of the shared file pointer */
 MPI_File_write_shared(fh,sbuf,n,MPI_INT,MPI_STATUS_IGNORE);
/* Cause all previous writes to be transferred to the storage device  */
 MPI_File_sync(fh);

 if(myrank==0){
  printf("%d READ ALL FROM FILE¥n", myrank);
/* Move an individual file pointer to the location in the file from which a process needs to
read data */
  MPI_File_seek(fh, offset, MPI_SEEK_SET);
  if(nprocess<4){tmp=4;
  }else{tmp = nprocess;}
/* Read data from the current location of the individual file pointer in the file */
  MPI_File_read(fh,rbuf,n*nprocess, MPI_INT, MPI_STATUS_IGNORE);
  printf("%d: rbuf=>",myrank);
  for(int i=0; i<16; ++i)printf("%d", rbuf[i]);
  printf("¥n");
}
/* Close the common file */
 MPI_File_close(&fh);
 MPI_Finalize();
}
```

2.6.6 並列 I/O の高速化

VE 上で MPI プログラムを実行する際、ScaTeFS VE ダイレクト IB ライブラリ、あるいは高速 I/O 機能を利用することにより並列 I/O の性能を向上させることができます。これらの利用条件と利用例は以下の通りです。

- ScaTeFS VE ダイレクト IB ライブラリ
  ScaTeFS がインストールされている環境で使用できます。詳細は「NEC Scalable Technology File System (ScaTeFS) 運用の手引」をご参照ください。
  (利用例)

```
$ export VE_LD_PRELOAD=libscatefsib.so.1
$ mpirun -np 1 ./a.out
```

- 高速 I/O
  高速 I/O のシステム設定が行われている環境で使用できます。詳細は「VE プログラムの実行方法」をご参照ください。
  (利用例)

```
$ export VE_ACC_IO=1
$ mpirun -np 1 ./a.out
```

2.7 スレッド実行環境

MPI 仕様では，MPI のスレッドサポートのレベルを以下の四つに分類しています。

- MPI_THREAD_SINGLE
  単一スレッドだけ実行可能

- MPI_THREAD_FUNNELED
  MPI プロセス中に複数のスレッドが存在してもよい。ただし，初期スレッドだけが MPI 手続を実行できる。

- MPI_THREAD_SERIALIZED
  MPI プロセス中に複数のスレッドが存在してもよく，どのスレッドも MPI 手続を呼び出すことができる。ただし，複数のスレッドが同時に MPI 手続を実行することはできない。

- MPI_THREAD_MULTIPLE
  MPI プロセス中に複数のスレッドが存在してもよく，全てのスレッドが MPI 手続を同時に実行できる。

MPI プロセス中で複数のスレッドを生成する場合，MPI の初期化は手続 MPI_INIT の代わりに手続 MPI_INIT_THREAD を利用します。手続 MPI_INIT_THREAD には，スレッドサポートレベルを指定します。

NEC MPI は，MPI_THREAD_SERIALIZED の機能を提供します。したがって，全てのスレッドが MPI 手続を呼び出すことが可能です。ただし，複数のスレッドが同時に MPI 手続を呼び出さないように利用者がスレッドの実行を適切に制御しなければなりません。

また，OpenMP などの共有並列処理や POSIX スレッド標準の同期機構を利用せずに，たとえばロックファイルやスレッド間共有変数により同期を実現している場合は，スレッド間でキャッシュの一貫性を保証できないことがあります。

2.8 エラーハンドラー

MPI 手続中で引数不正などのエラーを検出した場合，指定された手続(エラーハンドラー)を実行する機能が提供されています。

エラーハンドラーは，コミュニケーター，ファイルハンドラー，および ウィンドウに対して設定することができ，MPI 通信処理中にエラーが検出された場合，設定されているエラーハンドラーが実行されます。

利用者定義のエラーハンドラーを利用する場合，対応するエラー処理手続をエラーハンドラーとして登録する操作が必要です。この操作は，手続 MPI_*XXX*_CREATE_ERRHANDLER によって行います。ここで *XXX* は，コミュニケーター用のエラーハンドラーの場合は，COMM，ウィンドウ用のエラーハンドラーの場合は，WIN，ファイル用のエラーハンドラーの場合は，FILE となります。同様に，エラーハンドラーの設定には，手続 MPI_*XXX*_SET_ERRHANDLER，取得には，手続 MPI_*XXX*_GET_ERRHANDLER を利用します。

2.9 属性キャッシュ

コミュニケーター，ウィンドウ，および MPI データ型に対して，利用者が任意の属性を対応付け，利用できる機構が提供されています。

属性を使用する前に，属性キー(key)を生成して，登録しておく必要があります。この操作を行う手続は，MPI_*XXX*_CREATE_KEYVAL です。ここで *XXX* は，コミュニケーター用の属性キーの場合は COMM，ウィンドウ用の属性キーの場合は WIN，MPI データ型用の属性キーの場合は TYPE となります。

実際に属性を付加するためには，手続 MPI_*XXX*_SET_ATTR，属性値を取得するためには，手続 MPI_*XXX*_GET_ATTR，属性値を削除するためには，手続 MPI_*XXX*_DELETE_ATTR をそれぞれ使用します。

2.10 Fortran サポート

NEC MPI は，次の Fortran サポート方式を用意しています。

- モジュール mpi_f08
- モジュール mpi
- インクルードファイル mpif.h

なお、NEC Fortran Compiler、GNU Fortran Compiler でモジュール mpi_f08 を利用するには以下のバージョンが必要です。

NEC Fortran Compiler　Version 2.4.1 以降
GNU Fortran Compiler　Version 9.1.0 以降

2.11 利用者指定情報(MPI_INFO)

利用者から MPI に対して最適化情報などを与える場合，MPI_INFO オブジェクトを利用します。MPI_INFO オブジェクトは, キー(key)となる文字列とそのキーに対応する値である文字列を 1 つの対とし，その対を 0 個以上含むオブジェクトです。MPI_INFO は不透明オブジェクトであり，生成や参照の操作は全てハンドルを介して行います。

MPI_INFO オブジェクトは，手続 MPI_INFO_CREATE によりその枠組みが生成され，そのオブジェクトに対応したハンドル値が返却されます。この時点ではオブジェクトの内容は空の状態です。

MPI_INFO オブジェクトに(キー，値)の対を追加するためには，手続 MPI_INFO_SET を利用します。また，指定したキーに対応する値を取得するためには，手続 MPI_INFO_GET を利用します。

MPI_INFO オブジェクト操作手続としては他に，手続 MPI_INFO_DELETE((キー，値)の対の削除)，MPI_INFO_GET_VALUE_LEN(指定したキーに対応する値の文字列長)，MPI_INFO_GET_NKEYS(MPI_INFO オブジェクトに含まれるキーの個数)，MPI_INFO_GET_NTHKEY(N 番目の(キー，値)対のキーを取得)，MPI_INFO_DUP(MPI_INFO オブジェクトの複製)，および MPI_INFO_FREE(MPI_INFO オブジェクトの解放)があります。

MPI_INFO オブジェクトの利用例の 1 つとして，MPI-IO に対する情報提供があります。MPI-IO 機能を利用してファイル入出力を行う場合，ファイルアクセスのパターンや利用するファイルシステムの情報を与えることにより最適化を促進することができます。

表 2-3 に NEC MPI が解釈するキーとその意味を示します。キーの値として不正な値が指定された場合，その指定は無視され，既定値が採用されます。

表 2-3 NEC MPI の解釈するキーとその意味

キー	意味

cb_buffer_size	MPI-IO 手続のうち，集団操作手続中で利用するバッファの大きさをバイト単位で指定します。本指定がない場合の既定値は 4MB です。この値はコミュニケーターに含まれる全てのプロセスで同一でなければなりません。ここで指定した大きさ，あるいは既定値の大きさのバッファがプロセスごとに確保されます。
cb_nodes	集団操作手続中で実際に入出力を行うプロセス数を指定します。本指定がない場合の既定値はコミュニケーターに含まれるプロセス数です。入出力を行うプロセスはコミュニケーター中のランク順に選択されます。
host	手続 MPI_COMM_SPAWN または MPI_COMM_SPAWN_MULTIPLE によって MPI プロセスを生成するホストを指定します。この指定がない場合，ルートプロセスが存在するホスト上に MPI プロセスを生成します。
ve	手続 MPI_COMM_SPAWN または MPI_COMM_SPAWN_MULTIPLE によって MPI プロセスを生成する VE 番号を指定します。
vh sh	手続 MPI_COMM_SPAWN または MPI_COMM_SPAWN_MULTIPLE によって MPI プロセスを VH またはスカラホスト上に生成することを指示します。値は 1 を指定します。
ind_rd_buffer_size	不連続データの入力処理において利用する入力バッファの大きさをバイト単位で指定します。プロセスごとに異なる値を指定し，異なる大きさのバッファを確保することができます。本指定がない場合の既定値は 4MB です。
ind_wr_buffer_size	不連続データの出力処理において利用する出力バッファの大きさをバイト単位で指定します。プロセスごとに異なる値を指定し，異なる大きさのバッファを確保することができます。本指定がない場合の既定値は 4MB です。
ip_port	手続 MPI_OPEN_PORT によりポートの準備を行う場合に利用するポート番号を指定します。この指定がない場合，利用するポートは自動的に決定されます。

2.12 言語間相互利用可能性(Language Interoperability)

Fortran および C もしくは C++を利用してプログラミングを行う言語ミックスのケースにおいて，MPI オブジェクト(**表 2-4** 参照)を扱う場合，ある 1 つの MPI オブジェクトを言語間にまたがって利用するには，そのオブジェクトのハンドル値を変換しなければなりません。その場合には，ハンドル値の変換は C バインディングの関数を用います(Fortran バインディングの変換手続はありません)。

たとえば，C 言語で扱うコミュニケーターのハンドル値を Fortran 言語用に変換する場合には，次の関数

```
MPI_Fint MPI_Comm_c2f(MPI_Comm c_comm)
```

を使用します。ここで c_comm は C 言語で扱うコミュニケーターのハンドル値であり，関数 MPI_Comm_c2f の返却値が Fortran 言語で利用可能なハンドル値となります(MPI_Fint は Fortran 言語の INTEGER 型に対応するデータ型です)。同様に，Fortran 言語用のハンドル値(f_comm)を C 言語用に変換する場合には，次の手続

```
MPI_Comm MPI_Comm_f2c(MPI_Fint f_comm)
```

を利用します。なお，通信状態(MPI_Status)を変換する場合には，変換対象 および 変換結果を格納する領域へのポインタを引数で指定します。

```
int MPI_Status_c2f(MPI_Status *c_status, MPI_Fint   *f_status)
int MPI_Status_f2c(MPI_Fint   *f_status, MPI_Status *c_status)
```

表 2-4 は個々の MPI オブジェクトと，それらに対応する変換用の関数名の一覧です。

表 2-4 MPI オブジェクトとハンドル変換手続

MPI オブジェクト	MPI オブジェクトの型	ハンドル値変換関数	
		C -> Fortran	Fortran -> C
コミュニケーター	MPI_Comm	MPI_COMM_C2F	MPI_COMM_F2C
データ型	MPI_Datatype	MPI_TYPE_C2F	MPI_TYPE_F2C
グループ	MPI_Group	MPI_GROUP_C2F	MPI_GROUP_F2C
通信リクエスト	MPI_Request	MPI_REQUEST_C2F	MPI_REQUEST_F2C
ファイル	MPI_File	MPI_FILE_C2F	MPI_FILE_F2C
ウィンドウ	MPI_Win	MPI_WIN_C2F	MPI_WIN_F2C
集計演算	MPI_Op	MPI_OP_C2F	MPI_OP_F2C
利用者指定情報	MPI_Info	MPI_INFO_C2F	MPI_INFO_F2C
通信状態	MPI_Status	MPI_STATUS_C2F	MPI_STATUS_F2C
メッセージ	MPI_Message	MPI_MESSAGE_C2F	MPI_MESSAGE_F2C

2.13 非ブロッキング MPI 手続利用時の注意事項

Fortran プログラム中で，MPI 手続の通信バッファ または 入出力バッファの実引数として，

- 部分配列
- 配列式
- 配列ポインタ
- 形状引継ぎ配列

を指定した場合，バッファ中の要素並びの連続性を保証するため，Fortran コンパイラは内部的に作業配列を準備し，MPI 手続呼出しの前後において，実引数と作業配列との間でデータのコピーを行い，実際の MPI 手続の実引数としてはこの内部作業配列を渡す場合があります。しかし，非ブロッキング MPI 手続によるデータ移動(通信 または 入出力)は，当該 MPI 手続だけではなく，手続 MPI_WAIT などにより完了待合せを行うまでの区間で行われるため，内部作業配列の解放後にデータ移動が発生すると，正しい結果が得られないことや，プログラムの実行に異常をきたすことがあります。

非ブロッキング MPI 手続の通信 または 入出力バッファの実引数として，これらの配列を指定している場合には，MPI 手続での前後で明示的に作業配列を確保してコピーする必要があります。

利用者による作業配列を利用したコーディング例(送信の場合)

```
real :: A(1000)
integer :: request

call MPI_ISEND(A+10.0, ..., request, ...)  ! 配列式 (指定不可)
call MPI_WAIT(request, ...)
```

↓

```
real :: A(1000)
real,allocatable :: tmp(:)
integer :: request

allocate(tmp(1000))                        ! 利用者作業配列確保
tmp = A + 10.0
call MPI_ISEND(tmp, ..., request, ...)     ! 利用者作業配列を実引数として MPI 手続呼出し
call MPI_WAIT(request, ...)                ! 通信完了待合わせ
deallocate(tmp)                            ! 利用者作業配列解放
```

利用者による作業配列を利用したコーディング例(受信の場合)

```
real :: A(1000)
integer :: request

call MPI_IRECV(A(1:1000:2), ..., request, ...)  ! 部分配列 (指定不可)
call MPI_WAIT(request, ...)
```

↓

```
real :: A(1000)
```

```
    real,allocatable :: tmp(:)
    integer :: request

    allocate(tmp(500))                          ! 利用者作業配列確保
    call MPI_IRECV(tmp, ..., request, ...)  ! 利用者作業配列を実引数とした MPI 手続呼出し
    call MPI_WAIT(request, ...)                 ! 通信完了待合わせ
    A(1:1000:2) = tmp                  ! 利用者作業配列から目的のユーザバッファへのデータコピ
ー
    deallocate(tmp)                     ! 利用者作業配列解放
```

第 3 章　操作方法

本章では，NEC MPI における MPI プログラムのコンパイル・リンク，および 実行方法を含む，NEC MPI の使用方法を説明します。

3.1　　コンパイル・リンク

MPI プログラムをコンパイル・リンクする前に、次のコマンドを実行しセットアップスクリプトを読み込んでください。*{version}* はご使用になる NEC MPI のバージョンに対応するディレクトリ名です。これにより NEC MPI を利用するために必要な環境変数が設定されます。この設定は VH からログアウトするまで有効です。ログアウトすると無効となりますので、VH にログインするたびに再実行してください。

```
(bash の場合)
$ source /opt/nec/ve/mpi/{version}/bin/necmpivars.sh
(csh の場合)
% source /opt/nec/ve/mpi/{version}/bin/necmpivars.csh
```

MPI プログラムのコンパイル・リンクは，各プログラミング言語に対応した MPI コンパイルコマンドを使用して行います。

- Fortran で記述された MPI プログラムのコンパイル・リンクは，次のように mpinfort コマンドを実行します。また、mpinfort の代わりに別名の mpifort を利用することもできます。

```
mpinfort [options] {sourcefiles}
```

- C で記述された MPI プログラムのコンパイル・リンクは，次のように mpincc コマンドを実行します。また、mpincc の代わりに別名の mpicc を利用することもできます。

```
mpincc [options] {sourcefiles}
```

- C++で記述された MPI プログラムのコンパイル・リンクは，次のように mpinc++コマンドを実行します。また、mpinc++の代わりに別名の mpic++を利用することもできます。

```
mpinc++ [options] {sourcefiles}
```

上記のコマンド行中で，*{sourcefiles}* は，MPI のソースプログラムです。 コンパイラオプション [options]は，省略可能であり，Fortran コンパイラ(nfort)，C コンパイラ(ncc)，または C++コンパイラ(nc++)が用意するコンパイラオプションに加えて，表 3-1 の NEC MPI のコンパイラオプションが指定できます。

コンパイルコマンド mpincc(mpicc), mpinc++(mpic++), mpinfort(mpifort) はそれぞれ標準で ncc, nc++, nfort のコンパイラを起動します。標準のコンパイラ以外を使用する場合、コンパイルコマンドのオプション-compiler または表 3-2 の環境変数を使用して起動するコンパイラを変更できます。

例) コンパイラバージョン 2.x.x を使用して C プログラム program.c をコンパイル・リンクする場合

```
$ mpincc -compiler /opt/nec/ve/bin/ncc-2.x.x program.c
```

表 3-1 NEC MPI コンパイルコマンドのオプション

オプション	意味
-mpimsgq | -msgq	デバッガ向けの MPI メッセージキュー機能を有効にします。
-mpiprof	MPI 通信情報機能を有効にします。 また、MPI プロファイリングインタフェース(名前が PMPI_で始まる MPI 手続き)を有効にします。 MPI 通信情報機能については 3.5 を参照ください。
-mpitrace	MPI トレース機能を有効にします。 同時に MPI 通信情報機能および MPI プロファイリングインタフェースも有効になります。 MPI トレース機能については 3.7 を参照ください。
-mpiverify	MPI 集団手続デバッグ支援機能を有効にします。 同時に MPI 通信情報機能および MPI プロファイリングインタフェースも有効になります。 MPI 集団手続デバッグ支援機能については 3.9 を参照ください。
-ftrace	MPI プログラムにおいて FTRACE 機能を有効にします。 同時に MPI 通信情報機能および MPI プロファイリングインタフェースも有効になります。 FTRACE 機能については 3.6 を参照ください。
-show	MPI コンパイルコマンドによるコンパイラの起動イメージを表示します。 この起動イメージは実際には実行されません。
-ve	VE で動作する MPI プログラムをコンパイル・リンクします。(既定値)
-vh -sh	VH またはスカラホストで動作する MPI プログラムをコンパイル・リンクします。
-static-mpi	MPI ライブラリを静的リンクします。ただし MPI のメモリ管理ライブラ

<table>
<tr><td></td><td>リは動的リンクされます。(既定値)</td></tr>
<tr><td>-shared-mpi</td><td>すべての MPI ライブラリを動的リンクします。</td></tr>
<tr><td>-compiler <compiler></td><td>スペースに続けて MPI コンパイルコマンドが起動するコンパイラを指定します。本オプションが未指定の場合、各コンパイルコマンドは以下のコンパイラを起動します。

VH またはスカラホストで動作する MPI プログラム向けには、以下のコンパイラをサポートしています。

GNU Compiler Collection
• 4.8.5
• 8.3.0 および 8.3.1
• 8.4.0 および 8.4.1
• 8.5.0
• 9.1.0 および互換のあるバージョン
Intel C++ Compiler および Intel Fortran Compiler
• 19.0.4.243 (Intel Parallel Studio XE 2019 Update 4) および互換のあるバージョン
• 19.1.2.254 (Intel Parallel Studio XE 2020 Update 2)
NVIDIA Cuda compiler
• 11.1

Fortran の mpi_f08 モジュールのご利用については、2.10 もご参照ください。

<table>
<tr><td>コンパイルコマンド</td><td>起動するコンパイラ</td></tr>
<tr><td>mpincc (mpicc)</td><td>ncc</td></tr>
<tr><td>mpinc++ (mpic++)</td><td>nc++</td></tr>
<tr><td>mpinfort (mpifort)</td><td>nfort</td></tr>
<tr><td>コンパイルコマンド
(-vh か-sh が指定された場合)</td><td>起動するコンパイラ</td></tr>
<tr><td>mpincc (mpicc)</td><td>gcc</td></tr>
<tr><td>mpinc++ (mpic++)</td><td>g++</td></tr>
<tr><td>mpinfort (mpifort)</td><td>gfortran</td></tr>
</table>
</td></tr>
<tr><td>-compiler_host <compiler></td><td>VH またはスカラホストで動作する MPI プログラム向けに-compile オプションで GNU コンパイラあるいは Intel コンパイラ以外のコンパイラを指定する場合（例えば、CUDA 向けに nvcc コマンドを利用する場合など）、指定されたコンパイラと互換性のある GNU コンパイラあるいは Intel コンパイラコマンドを本オプションで指定してください。ただし、その GNU</td></tr>
</table>

	コンパイラあるいは Intel コンパイラと NEC MPI の既定値（-compiler オプション参照）が一致する場合、本オプションの指定は省略可能です。

表 3-2 NEC MPI コンパイルコマンドの環境変数

(注意) 以下の環境変数の指定よりも-compiler オプションによる指定が優先されます。

環境変数	意味
NMPI_CC	mpincc コマンドにて VE で動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。
NMPI_CXX	mpinc++コマンドにて VE で動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。
NMPI_FC	mpinfort コマンドにて VE で動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。
NMPI_CC_H	mpincc コマンドにて VH またはスカラホストで動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。
NMPI_CXX_H	mpinc++コマンドにて VH またはスカラホストで動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。
NMPI_FC_H	mpinfort コマンドにて VH またはスカラホストで動作する MPI プログラムをコンパイル・リンクする際に使用するコンパイラを指定します。

以下にコンパイラごとの利用例を示します。

例 1) NEC コンパイラ

```
$ source /opt/nec/ve/mpi/2.x.x/bin/necmpivars.sh
$ mpincc a.c
$ mpinc++ a.cpp
$ mpinfort a.f90
```

例 2) GNU コンパイラ

```
(GNU コンパイラの利用に必要な設定(PATH、LD_LIBRARY_PATH 等)を行う)
$ source /opt/nec/ve/mpi/2.x.x/bin/necmpivars.sh
$ mpincc -vh a.c
$ mpinc++ -vh a.cpp
$ mpinfort -vh a.f90
```

例 3) Intel コンパイラ

```
(Intel コンパイラの利用に必要な設定(PATH、LD_LIBRARY_PATH 等)を行う)
$ source /opt/nec/ve/mpi/2.x.x/bin/necmpivars.sh
$ export NMPI_CC_H=icc
$ export NMPI_CXX_H=icpc
$ export NMPI_FC_H=ifort
$ mpincc -vh a.c
$ mpinc++ -vh a.cpp
$ mpinfort -vh a.f90
```

VH あるいはスカラホスト上で動作するプログラムが VEO 機能または CUDA 機能を併用する場合、コンパイル・リンク方法は以下の通りです。

- VEO 機能を利用する場合
  VEO 機能を利用するためのヘッダファイルおよびライブラリを指定してください。

```
$ mpincc -vh mpi-veo.c -o mpi-veo -I/opt/nec/ve/veos/include -L/opt/nec/ve/veos/lib64 ¥
  -Wl,-rpath=/opt/nec/ve/veos/lib64 -lveo
```

  VEO 機能の利用方法など、詳細は"The Tutorial and API Reference of Alternative VE Offloading"の情報を合わせてご参照ください。

- CUDA 機能を利用する場合
  コンパイル・リンク時に CUDA 向けのコンパイラ（nvcc など）を指定してください。

```
$ mpincc -sh -compiler nvcc mpi-cuda.c -o mpi-cuda
```

3.2 MPI プログラムの実行

MPI プログラムを実行する前に、3.1 を参考にご利用のコンパイラの設定をした後で、次のコマンドを実行しセットアップスクリプトを読み込んでください。*{version}* はご使用になる NEC MPI のバージョンに対応するディレクトリ名です。これにより NEC MPI を利用するために必要な環境変数が設定されます。この設定は VH からログアウトするまで有効です。ログアウトすると無効となりますので、VH にログインするたびに再実行してください。

```
(bash の場合)
$ source /opt/nec/ve/mpi/{version}/bin/necmpivars.sh
(csh の場合)
% source /opt/nec/ve/mpi/{version}/bin/necmpivars.csh
```

MPI プログラムには、既定値でコンパイル・リンクに使用したバージョンの MPI ライブラリが検索され、必要に応じて動的リンクされます。セットアップスクリプトを読み込むことで、VE で動作する MPI プログラムについて、上記*{version}*に対応する MPI からライブラリが検索されるようになります。これにより、MPI プログラムの作成時に-shared-mpi を指定して、すべての MPI ライブラリを動的リンクしている場合、実行時に使用される MPI ライブラリを変更することができます。

MPI プログラムの作成時に-shared-mpi を指定していない場合、MPI のメモリ管理ライブラリは動的リンクされ、それ以外の MPI ライブラリは静的リンクされます。静的リンクされた MPI ライブラリは実行時に変更することはできません。

また、ベクトルプロセスとスカラプロセスが混在するハイブリッド実行などの場合、上記に代わって下記のコマンドを実行します。これにより、VE に加えて、VH またはスカラホストで動作する MPI プログラムについても下記*{version}*に対応する MPI ライブラリが動的リンクされます。

(bash の場合)
$ **source /opt/nec/ve/mpi/***{version}***/bin/necmpivars.sh** *[gnu | intel] [compiler-version]*
(csh の場合)
% **source /opt/nec/ve/mpi/***{version}***/bin/necmpivars.csh** *[gnu | intel] [compiler-version]*

上記の *{version}* は，動的リンクする MPI ライブラリを含む NEC MPI のバージョンに対応するディレクトリ名です。第一引数*[gnu | intel]*は gnu か intel を指定します。第二引数*[compiler-version]*はコンパイル・リンクに使用したコンパイラのバージョンを指定します。各引数に指定する値は MPI プログラムの RUNPATH で確認してください。第一引数は波線部分を指定します。第二引数は点線部分を指定します。

(例)
$ **/usr/bin/readelf -W -d vh.out | grep RUNPATH**
0x000000000000001d (RUNPATH) Library runpath: [/opt/nec/ve/mpi/2.3.0/lib64/vh/gnu/9.1.0]

NEC MPI は，MPI プログラムを実行するために，MPI 実行コマンド mpirun および mpiexec を用意しています。次のいずれかの指定が可能です。

mpirun [global-options] [local-options] *{MPIexec}* [args] [: [local-options] *{MPIexec}* [args]]...

mpiexec [global-options] [local-options] *{MPIexec}* [args] [: [local-options] *{MPIexec}* [args]]...

- global-options は，全ての MPI 実行ファイルに対する実行時オプション(グローバルオプション)です。
- local-options は，直後の MPI 実行ファイルに対する実行時オプション(ローカルオプション)です。

実行時オプションの詳細は，3.2.2 項を参照してください。

- *{MPIexec}* は，プログラムの実行指定(MPI 実行指定)です。詳細は，3.2.1 項を参照してください。
- args は，直前の MPI 実行指定*{MPIexec}* に対するコマンド行引数です。
- []の中の指定は，省略可能です。
- []... の角括弧の中は，ゼロ回以上繰返して指定可能です。

MPI 実行コマンドは、コマンドのバージョンと同じかそれよりも古い MPI ライブラリがリンクされた MPI プログラムの実行をサポートしています。

システムの標準パス/opt/nec/ve/bin に配置された MPI 実行コマンドを使用する場合は、MPI プログラムの実行前に necmpivars.sh または necmpivars.csh を読み込んでください。

システムの標準パスに配置されていない特定バージョンの MPI 実行コマンドを使用する場合は、necmpivars.sh または necmpivars.csh の代わりに、/opt/nec/ve/mpi/*{version}*/bin ディレクトリに配置された necmpivars-runtime.sh または necmpivars-runtime.csh を読み込んでください。*{version}* は使用する MPI 実行コマンドを含む NEC MPI のバージョンに対応するディレクトリ名です。これらのセットアップスクリプトは necmpivars.sh や necmpivars.csh と同様の方法で使用でき、necmpivar.sh や necmpivars.csh の設定に加えて、指定されたバージョンの MPI 実行コマンドを使用するように設定を行います。

なお、システムの標準パスに配置されていない特定バージョンの MPI 実行コマンドは、NEC MPI プロセスマネージャに MPD が選択されたバッチキューに投入した NQSV リクエストでは使用できません。このため、necmpivars-runtime.sh または necmpivars-runtime.csh を読み込んだ場合、以下の警告メッセージを出力し実行に必要な設定を行いません。

```
necmpivars-runtime.sh: Warning: This script cannot be used in NQSV Request submitted to a batch queue that MPD is selected as NEC MPI Process Manager.
```

3.2.1 プログラムの実行指定

MPI 実行コマンド中の MPI 実行指定*{MPIexec}* には，次のいずれかの指定が可能です。

- MPI 実行ファイル*{execfile}*
  次のように，MPI 実行ファイル*{execfile}* を指定します。

```
# mpirun -np 2 {execfile}
```

- MPI 実行ファイル*{execfile}* を実行するシェルスクリプト
  次のように，MPI 実行ファイル*{execfile}* を実行するシェルスクリプトを指定します。

```
# cat shell.sh
#!/bin/sh
```

```
{execfile}
# mpirun -np 2 ./shell.sh
```

なお，上記の説明は，SX Aurora TSUBASA の既定値の環境である Linux の binfmt_misc 機能が設定されていることを前提としています。binfmt_misc 機能の設定には，システムの管理者権限が必要です。詳細は，「SX-Aurora TSUBASA インストレーションガイド」を参照するか，システム管理者にお問い合わせください。

binfmt_misc 機能が設定されていない環境では，MPI 実行指定*{MPIexec}* として，次のような指定をしてください。

- ve_exec コマンド"/opt/nec/ve/bin/ve_exec" および MPI 実行ファイル*{execfile}*
  次のように，ve_exec コマンド"/opt/nec/ve/bin/ve_exec" および MPI 実行ファイル*{execfile}* を指定します。

```
# mpirun -np 2 /opt/nec/ve/bin/ve_exec {execfile}
```

- ve_exec コマンド"/opt/nec/ve/bin/ve_exec" および MPI 実行ファイル*{execfile}*を指定したシェルスクリプト
  次のように，ve_exec コマンド"/opt/nec/ve/bin/ve_exec" および MPI 実行ファイル*{execfile}* を指定したシェルスクリプトを指定します。

```
# cat shell.sh
#!/bin/sh
/opt/nec/ve/bin/ve_exec {execfile}
# mpirun -np 2 ./shell.sh
```

3.2.2 実行時オプション

MPI 実行コマンド中に指定可能なグローバルオプションは，表 3-3 のとおりです。MPI 実行コマンド中に指定可能なローカルオプションは，表 3-4 のとおりです。

実行時オプション中のホストは，VH または VE を意味しています。指定方法の詳細は， 3.2.3 項を参照してください。

表 3-3 グローバルオプションの一覧

グローバルオプション	意味
-machinefile | -machine *<filename>*	ホストとホスト上に起動するプロセス数を記載したファイル。 記載形式は、*hostname[:value]* です。1 行毎に左記

	形式で指定します。プロセス数(:value)を省略した場合、既定値は 1 です。
-configfile *<filename>*	実行時オプションを記載したファイル。 ファイル*<filename>*中では、1 つ以上のオプション行を指定します。1 つの行では、実行時オプションおよび MPI 実行ファイル等の MPI 実行指定 *{MPIexec}* を指定します。行の最初が``#"の場合、その行はコメントとして扱われます。
-hosts *<host-list>*	MPI プロセスが起動される，カンマで区切ったホストのリスト。 オプション-hosts および -hostfile を複数回指定した場合，指定した順に全部 1 つにつないだリストとして扱われます。 本オプションは，オプション-host，-node，または -nn と併用することはできません。
-hostfile *<filename>*	MPI プロセスが起動されるホストを指定するファイル名。 オプション-hosts および -hostfile を複数回指定した場合，指定した順に全部 1 つにつないだリストとして扱われます。 本オプションは，オプション-host，-node，または -nn と併用することはできません。
-gvenode	オプション中で指定されたホストは，VE を意味します。
-perhost | -ppn | -N | -npernode | -nnp *<value>*	MPI プロセスは，指定された個数ずつ，各ホストに順に割り当てられます。 MPI プロセスの各ホストへの割り当ては，全ての MPI プロセスがホストに割り当てられるまで，循環的に行われます。 本オプション省略時は，MPI プロセス総数をホストの個数で割った値(端数は切り上げ)を既定値とします。
-max_np *<max_np>*	*<max_np>* には、実行時に動的に生成される MPI プロセスも含め、MPI プロセス数の最大値を指定します。既定値は-np オプションで指定された値です(-np オプションが複数指定されている場合はその総和)。
-multi	MPI プログラムを複数ホスト上で実行することを指定します。プログラム実行開始時点ではすべての MPI プロセスが単一ホスト内にのみ生成され、その

	後、動的プロセス生成機能により他ホストに MPI プロセスを生成し、結果的に複数ホスト実行となる場合、本オプションを指定してください。
-genv *<varname> <value>*	値*<value>*をもつ環境変数*<varname>*を，全ての MPI プロセスに渡します。
-genvall	（既定値）全ての環境変数を，全ての MPI プロセスに渡します。 ただし，NQSV リクエスト実行または PBS リクエスト実行の場合，NQSV または PBS が既定値で設定する環境変数を除きます。
-genvlist *<varname-list>*	MPI プロセスに渡す，カンマで区切った環境変数のリスト
-genvnone	環境変数を MPI プロセスに渡しません。
-gpath *<dirname>*	MPI プロセスに渡す PATH 環境変数を*<dirname>*に設定します。
-gumask *<mode>*	MPI プロセス向けに"*umask <mode>*"を実行します。
-gwdir *<dirname>*	MPI プロセスの実行ディレクトリを*<dirname>*に設定します。
-gdb | -debug	起動される MPI プロセス 1 つにつき，1 つのデバッグ用スクリーンを開き，gdb デバッガ配下で MPI プログラムを実行します。
-display | -disp *<X-server>*	デバッグ用スクリーンのための X サーバを，"*host:display*" または "*host:display:screen*"の書式で指定します。
-v | -V | -version	NEC MPI のバージョン および 環境変数のような実行時情報を表示します。
-h | -help	MPI 実行コマンドのヘルプ情報を表示します。

表 3-4 ローカルオプションの一覧

ローカルオプション	意味
-ve *<first>*[-*<last>*]	MPI プロセスが実行される VE を指定します。本オプションが指定された場合，実行時オプション中で指定されたホストは，VH を意味します。 インタラクティブ実行の場合，VE 番号の範囲を指定します。 NQSV リクエスト実行の場合，論理 VE 番号の範囲を指定します。 <first>は，最初の VE 番号，<last>は，最後の VE 番

	号です。<last>は<first>以上でなければなりません。-<last>を省略した場合，-<first>が指定されたものとみなされます。 指定された VE は，ローカルオプション中で直前に指定された VH または グローバルオプション中で指定された VH 上の VE となります。 本オプションを省略し，VE 番号の指定がない場合，VE#0 が指定されたものとみなされます。ただし、NQSV リクエスト実行において、本オプションを省略し、かつ他オプションによるホストやホストの個数の指定も省略された場合、NQSV により割り当てられた全ての VE が指定されたものとみなされます。
-nve *<value>*	MPI プロセスが実行される VE 数を指定します。 指定された VE 数は，ローカルオプション中で直前に指定された VH または グローバルオプション中で指定された VH 上の VE 数となります。 本オプションは -ve 0-*<value-1>* と同じ意味です。
-venode	オプション中で指定されたホストは，VE を意味します。
-vh | -sh	VH またはスカラホスト上に MPI プロセスを生成する場合、本オプションを指定します。
-host *<host>*	MPI プロセスを起動するホストを 1 つ指定します。
-node *<hostrange>*	MPI プロセスを起動するホストの範囲を指定します。 インタラクティブ実行の場合，-venode オプションも指定する必要があります。 オプション-hosts，-hostfile，-host，または -nn が指定された場合，本オプションは無視されます。
-nn *<value>*	MPI プロセスを起動するホストの個数を指定します。 NQSV リクエスト実行の場合だけ指定できます。 本オプションは，各 MPI 実行ファイルに対して，それぞれ 1 度だけ指定できます。 本オプションを省略し、かつ他オプションによるホストやホストの個数の指定も省略された場合、NQSV により割り当てられたホストの個数が指定されたものとみなされます。 オプション-hosts，-hostfile，または -host が指定された場合，本オプションは無視されます。
-numa *<first>*[-*<last>*][,<...>]	MPI プロセスが実行される VE 上の NUMA ノードの範囲を指定します。

	<first>は，最初の NUMA ノード番号，<last>は，最後の NUMA ノード番号です。<last>は<first>以上でなければなりません。 -<last>を省略した場合，-<first>が指定されたものとみなされます。
-nnuma *<value>*	MPI プロセスが実行される VE 上の NUMA ノードの数を指定します。本オプションは -numa 0-*<value-1>* と同じ意味です。
-c | -n | -np *<value>*	対応するホスト上で起動される MPI プロセスの総数。指定された MPI プロセスは，ローカルオプション中で本オプションの直前に指定されたホスト または グローバルオプション中で指定されたホストに対応します。 本オプション省略時は，1 を既定値とします。
-ve_nnp *<value>*	VE 当たりに起動される MPI プロセスの個数。-np オプションや-nnp オプションなど MPI プロセスの個数を指定する他のオプションが指定された場所では、本オプションは無視されます。本オプションは-gvenode オプションや-venode オプションが指定された場所では使用できません。 本オプション省略時は、1 を既定値とします。
-env *<varname>* *<value>*	MPI プロセスに渡す，値*<value>*をもつ環境変数 *<varname>*
-envall	全ての環境変数を， MPI プロセスに渡します。 ただし，NQSV リクエスト実行または PBS リクエスト実行の場合，NQSV または PBS が既定値で設定する環境変数を除きます。
-envlist	MPI プロセスに渡す，カンマで区切った環境変数のリスト
-envnone	環境変数を MPI プロセスに渡しません。
-path *<dirname>*	MPI プロセスに渡す PATH 環境変数を*<dirname>*に設定します。
-umask *<mode>*	MPI プロセス向けに"*umask <mode>*"を実行します。
-wdir *<dirname>*	MPI プロセスの実行ディレクトリを*<dirname>*に設定します。
-ib_vh_memcpy_send < auto | on | off >	InfiniBand 通信の送信において、VH メモリコピー処理を利用します。本オプションは、環境変数 NMPI_IB_VH_MEMCPY_SEND より優先されます。

	auto: InfiniBand 通信の送信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。(Intel マシンを使用する場合の既定値) on: InfiniBand 通信の送信において、VH メモリコピー処理を利用します。(Intel 以外のマシンを使用する場合の既定値) off: InfiniBand 通信の送信において、VH メモリコピー処理を利用しません。
-ib_vh_memcpy_recv　< auto | on | off >	InfiniBand 通信の受信において、VH メモリコピー処理を利用します。本オプションは、環境変数 NMPI_IB_VH_MEMCPY_RECV より優先されます。 auto: InfiniBand 通信の受信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。 on: InfiniBand 通信の受信において、 VH メモリコピー処理を利用します。(Intel 以外のマシンを使用する場合の既定値) off: InfiniBand 通信の受信において、VH メモリコピー処理を利用しません。(Intel マシンを使用する場合の既定値)
-dma_vh_memcpy　< auto | on | off >	VH 内の VE 間通信において、VH メモリコピー処理を利用します。本オプションは、環境変数 NMPI_DMA_VH_MEMCPY より優先されます。 auto: VH 内の VE 間通信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。(既定値)

	on: VH 内の VE 間通信において、VH メモリコピー処理を利用します。 off: VH メモリコピー処理を利用しません。
-vh_memcpy < auto | on | off >	Infiniband 通信と VH 内の VE 間通信において、VH メモリコピー処理を利用します。本オプションは、環境変数 NMPI_VH_MEMCPY より優先されます。 auto: Infiniband 通信と VH 内の VE 間通信において、これらの通信が RootComplex を経由する場合に VH メモリコピー処理を利用します。 on: Infiniband 通信と VH 内の VE 間通信において、VH メモリコピー処理を利用します。 off: VH メモリコピー処理を利用しません。 備考： -ib_vh_memcpy_send, -ib_vh_memcpy_recv, -dma_vh_memcpy が明示的に指定された場合はこれらの設定値が優先されます。 上記オプションが指定されていない場合、本オプションの設定が有効になります
-vpin | -vpinning	VH またはスカラホスト上で実行される MPI プロセスに割り当てられた CPU 番号を表示します。 本オプションは -pin_mode, -cpu_list, -numa, -nnuma オプションに対して有効です。
-pin_mode { consec | spread | consec_rev | spread_rev | scatter | none | off | no}	VH またはスカラホスト上で実行される MPI プロセスの CPU アフィニティを設定します。 consec | spread: CPU 番号#0 から昇順に割り当てます。

	consec_rev | spread_rev: CPU コア数-1 の CPU 番号から降順に割り当てます。 scatter: 割当済みの CPU コアとの距離が最大になるように空き CPU コアを割り当てます。 none | off | no: (既定値) CPU アフィニティを設定しません 本オプションを指定した場合、先行して指定した-cpu_list オプションは無視されます。 未割当の CPU コア数が割り当てようとする CPU コア数よりも少ない場合、CPU アフィニティは設定されません。
-pin_reserve <num-reserved-ids>[h|H]	VH またはスカラホスト上で実行される MPI プロセス毎に割り当てる CPU コア数を指定します。 本オプションに h または H が指定された場合、ハイパースレッド数に分割して割当を行います。 1 以上の数を指定します。既定値は 1 です。
-cpu_list |-pin_cpu *<first-id>*[-*<last-id* [-*<increment>*[-*<num-reserved-ids>* [h|H]][, ...]]]]	VH またはスカラホスト上で実行される MPI プロセスに割り当てる CPU 番号を指定します。 最若番ランクのプロセスに CPU 番号*<first-id>*が割り当てられます。以降、若いランクのプロセスから順に+*<increment>* した CPU 番号が割り当てられます。*<num-reserved-ids>*にはプロセス毎に割り当てる CPU コア数を指定します。*<last-id>*には*<num-reserved-ids>*を含めない最後に割り当てる CPU 番号を指定します。 本オプションに h または H を指定した場合、CPU コア数はハイパースレッド数に分割して割り当てられます。 既定値は以下の通りです。 *<last-id>*: *<first-id>* *<increment>*:1 *<num-reserved-ids>*:1 本オプション指定した場合、先行して指定した-

	pin_mode オプションは無視されます。 未割当の CPU コア数が割り当てようとする CPU コア数よりも少ない場合、CPU アフィニティは設定されません。
-veo	VEO 機能を利用することを指定します。
-cuda	CUDA 機能を利用することを指定します。

- NQSV リクエスト実行において，オプション-hosts，-hostfile，host，-node，および -nn を省略すると，NQSV が確保した全てのホストが使用されます。

- PBS リクエスト実行において，オプション-machinefile，-machine，-hosts，-hostfile，-gvenode，-perhost，-ppn，-N，-npernode，-nnp，-ve，-nve，-venode，-ve_nnp, -host，-node，-nn は指定できません。

- オプション-hosts，-hostfile，-host，-node，および -nn の優先順位は，
  -hosts，-hostfile，-host > -nn > -node の順です。

- ローカルオプション -evn, -envall, -envlist, -envnone, -path, -umask, -wdir は、グローバルオプション -genv, -genvall, -genvlist, -genvnone, -gpath, -gumask, -gwdir より優先されます。

3.2.3 ホストの指定方法

MPI 実行ファイルに対応するホストは，指定した実行時オプションにより，次のように決定されます。

- -venode オプションを指定しない場合(既定値)
  この場合，指定されたホストは，VH を意味します。VH は，表 3-5 のように指定します。

表 3-5 VH の指定方法

実行方式	指定	説明
インタラクティブ実行	VH 名	VH であるホストコンピュータのホスト名を指定します。
NQSV リクエスト実行	*<first>*[-*<last>*]	• *<first>*は，最初の論理 VH 番号，*<last>*は，最後の論理 VH 番号です。 • 1 つの VH を指定する場合，*<first>* だけを指定します。特に，オプション-hosts，-hostfile，または -host 中では，*<fisrt>*だけを指定してください。

		• *<last>*は，*<first>*以上でなければなりません。

- -venode オプションを指定した場合
  この場合，指定されたホストは，VE を意味します。VE は，表 3-6 のように指定します。
  -venode オプションを指定した MPI 実行ファイルに対しては，-ve オプションは指定できないことに注意してください。

表 3-6 VE の指定方法

実行方式	指定	説明
インタラクティブ実行	*<first>*[-*<last>*][@*<VH>*]	• *<first>*は，最初の VE 番号，*<last>*は，最後の VE 番号です。 • *<VH>*は，VH 名です。省略した場合，MPI 実行コマンドを実行した VH が選択されます。 • 1 つの VE を指定する場合，*<first>* だけを指定します。特に，オプション-hosts，-hostfile，または -host 中では，*<fisrt>*だけを指定してください。 • *<last>*は，*<first>*以上でなければなりません。
NQSV リクエスト実行	*<first>*[-*<last>*][@*<VH>*]	• *<first>*は，最初の論理 VE 番号，*<last>*は，最後の論理 VE 番号です。 • *<VH>*は，論理 VH 番号です。省略した場合，ホスト(VE)は，NQSV が割り当てた全ての VE の中から選択されます。 • 1 つの VE を指定する場合，*<first>* だけを指定します。特に，オプション-hosts，-hostfile，または -host 中では，*<fisrt>*だけを指定してください。 • *<last>*は，*<first>*以上でなければなりません。

3.2.4 環境変数

MPI プログラムの実行時に，利用者が値を設定できる環境変数を表 3-7 に示します。

表 3-7 利用者が値を設定できる環境変数

環境変数	設定できる値	意味
NMPI_COMMINF	MPI 通信情報の出力を制御します。MPI 通信情報を利用するには、オプション-mpiprof, -mpitrace, -mpiverify または-ftrace オプションを指定して MPI プログラムを作成する必要があります。MPI 通信情報については 3.5 を参照ください。	
	NO	(既定値) MPI 通信情報を出力しません。
	YES	MPI 通信情報を集約形式で出力します。
	ALL	MPI 通信情報を拡張形式で出力します。
MPICOMMINF	環境変数 NMPI_COMMINF と同様です。	環境変数 NMPI_COMMINF と同様です。 両方の環境変数が指定された場合, 環境変数 NMPI_COMMINF の指定が優先されます。
NMPI_COMMINF_VIEW	MPI 通信情報の集約部分の表示の形式を指定します。	
	VERTICAL	(既定値) ベクトルプロセスとスカラプロセスを分けて集計し、縦に並べて表示します。
	HORIZONTAL	ベクトルプロセスとスカラプロセスを分けて集計し、横に並べて表示します。
	MERGED	ベクトルプロセスとスカラプロセスをまとめて集計し表示します。
NMPI_PROGINF	MPI 実行性能情報の出力を制御します。MPI 実行性能情報については 3.4 を参照ください。	
	NO	(既定値) 性能情報を出力しません。
	YES	性能情報を集約形式で出力します。
	ALL	性能情報を拡張形式で出力します。
	DETAIL	性能情報を詳細集約形式で出力

		します。
	ALL_DETAIL	性能情報を詳細拡張形式で出力します。
MPIPROGINF	環境変数 **NMPI_PROGINF** と同様です。	環境変数 **NMPI_PROGINF** と同様です。 両方の環境変数が指定された場合，環境変数 **NMPI_PROGINF** の指定が優先されます。
NMPI_PROGINF_COMPAT	**0**	MPI 実行性能情報を最新の形式で出力します。(既定値)
	1	MPI 実行性能情報を旧形式で出力します。本形式では性能項目 Non Swappable Memory Size Used、VE Card Data 部、および、VE プロセスが実行された VE カードの位置情報は出力されません。
VE_PROGINF_USE_SIGNAL	**YES**	(既定値) 性能情報の採取にシグナルを使用します。
	NO	性能情報の採取にシグナルを使用しません。 (ご利用の前に 3.11 (11)をご参照ください。)
VE_PERF_MODE	性能カウンタの種類を選択します。MPI 実行性能情報は選択された性能カウンタに対応する項目を出力します。	
	VECTOR-OP	(既定値)主にベクトル演算に関する性能カウンタを選択します。
	VECTOR-MEM	主にベクトルとメモリアクセスに関する性能カウンタを選択します。
NMPI_EXPORT	"*<環境変数名の列>*"	MPI プロセスに渡す環境変数名の列を，空白で区切って指定します。
MPIEXPORT	環境変数 **NMPI_EXPORT** と同様です。	環境変数 **NMPI_EXPORT** と同様です。 両方の環境変数が指定された場合,環境変数 **NMPI_EXPORT** の

		指定が優先されます。
NMPI_SEPSELECT	本環境変数を有効にするためには，シェルスクリプト mpisep.sh を使用する必要があります。詳細は，3.3 節を参照してください。	
	1	MPI プロセスの標準出力をプロセス別のファイルに保存します
	2	(既定値) MPI プロセスの標準エラー出力をプロセス別のファイルに保存します
	3	MPI プロセスの標準出力 および 標準エラー出力を，それぞれプロセス別のファイルに保存します。
	4	MPI プロセスの標準出力 および 標準エラー出力を，プロセス別の 1 つのファイルに保存します。
MPISEPSELECT	環境変数 **NMPI_SEPSELECT** と同様です。	環境変数 **NMPI_SEPSELECT** と同様です。 両方の環境変数が指定された場合，環境変数 **NMPI_SEPSELECT** の指定が優先されます。
NMPI_VE_TRACEBACK	VE の MPI プログラムが MPI_Abort 時に出力するトレースバックの形式を制御します。	
	ON	環境変数 **VE_TRACEBACK** に **VERBOSE** を指定した時に NEC コンパイラが出力するトレースバックと同じ形式でトレースバックを出力します。ファイル名や行番号情報が出力されます。
	OFF	backtrace_symbols が出力するトレースバックと同じ形式でトレースバックを出力します。(既定値)
NMPI_TRACEBACK_DEPTH	*<整数値>*	MPI_Abort のトレースバックの出力数を制御します。既定値は 50 です。この環境変数に 0 を 設

		定した場合、VE プログラムの場合は上限なしとなり、VH プログラムの場合は少なくとも 50 のトレースバックが表示されます。
NMPI_OUTPUT_COLLECT	NQSV バッチジョブ実行時にキュー設定の NEC MPI プロセスマネージャーが hydra の場合における MPI プログラムの出力を制御します。	
	ON	MPI プログラムの出力を MPI 実行コマンドの標準出力・標準エラー出力にします。
	OFF	（既定値）MPI プログラムの出力を mpd の場合と同様に論理ノードごとに出力します。
NMPI_VERIFY	MPI 集団手続きデバッグ支援機能の誤り検出機能を制御します。MPI 集団手続きデバッグ支援機能を利用するには、オプション-mpiverify を指定して MPI プログラムを作成する必要があります。MPI 集団手続きデバッグ支援機能については 3.9 を参照ください。	
	0	MPI 集団手続引用の誤りを検出しません。
	3	（既定値）手続 MPI_WIN_FENCE の引数 assert 以外の誤りを検出します。
	4	既定値の誤りに加えて，手続 MPI_WIN_FENCE の引数 assert の誤りを検出します。
NMPI_BLOCKLEN0	**OFF**	(既定値) 引数に blocklength をもつ MPI 派生データ型生成手続により生成されるデータ型において，blocklength が 0 のブロックは，上限値・下限値の計算に含まれません。
	ON	引数に blocklength をもつ MPI 派生データ型生成手続により生成されるデータ型において，blocklength が 0 のブロックも，上限値・下限値の計算に含まれ

		ます。
MPIBLOCKLEN0	環境変数 **NMPI_BLOCKLEN0** と同様です。	環境変数 **NMPI_BLOCKLEN0** と同様です。 両方の環境変数が指定された場合，環境変数 **NMPI_BLOCKLEN0** の指定が優先されます。
NMPI_COLLORDER	**OFF**	(既定値) 演算を伴う集団通信の実行において、共有メモリを利用したリダクション演算最適化を行います。この場合、同一のリダクション演算であっても、マルチノード実行かつプロセス配置が異なる場合に、MPI プロセスの演算順序の違いから異なるリダクション演算結果を生じることがあります。シングルノード実行の場合やマルチノード実行であっても同じプロセス配置の場合は、演算結果が異なることはありません。
	ON	共有メモリを利用したリダクション演算最適化を行いません。最適化を行った場合と比較して、リダクション演算性能が低下することがありますが、MPI プロセスの配置によって演算結果が異なることはありません。
MPICOLLORDER	環境変数 **NMPI_COLLORDER** と同じです。	環境変数 **NMPI_COLLORDER** と同様です。 両方の環境変数が指定された場合，環境変数 **NMPI_COLLORDER** の指定が優先されます。
NMPI_PORT_RANGE	*<整数値>:<整数値>*	NEC MPI が TCP/IP 接続を待ち受けるのに使用するポート番号の範囲を指定します。既定値は，25257:25266 です。
NMPI_INTERVAL_CONNECT	*<整数値>*	MPI プログラム実行開始時の，

		MPI デーモン間の接続確立の試行間隔を秒単位で指定します。既定値は，1(秒)です。
NMPI_RETRY_CONNECT	*<整数値>*	MPI プログラム実行開始時の，MPI デーモン間の接続確立の試行回数を指定します。既定値は，2 です。
NMPI_LAUNCHER_EXEC	*<フルパス名>*	MPI デーモンを起動するリモートシェルのフルパス名を指定します。既定値は，/usr/bin/ssh です。
NMPI_IB_ADAPTER_NAME	"*<文字列>*"	MPI が使用する InfiniBand アダプター名のリストを，カンマまたは 空白で区切って指定します。本環境変数は，インタラクティブ実行の場合だけ有効です。 省略した場合，処理系が自動的に最適な InfiniBand アダプターを選択します。
NMPI_IB_DEFAULT_PKEY	*<整数値>*	InfiniBand 通信の Partition Key を指定します。既定値は 0 です。
NMPI_IB_FAST_PATH	**ON**	InfiniBand RDMA fast path 機能を利用します。 InfiniBand HCA Relaxed Ordering または Adaptive Routing が有効な場合、本値は設定しないで下さい。 (Intel マシンを使用する場合の既定値)
	MTU	OFED の MTU(Maximum Transmission Unit)より小さいサイズの通信において、InfiniBand RDMA fast path 機能を利用します。 InfiniBand HCA Relaxed Ordering が有効な場合、本値は設定しないで下さい。

	OFF	InfiniBand RDMA fast path 機能を利用しません。 (Intel 以外のマシンを使用する場合の既定値)
NMPI_IB_VBUF_TOTAL_SIZE	*<整数値>*	InfiniBand 通信バッファのサイズをバイト単位で指定します。既定値は 12248 です。
NMPI_IB_VH_MEMCPY_SEND	**AUTO**	InfiniBand 通信の送信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。(Intel マシンを使用する場合の既定値)
	ON	InfiniBand 通信の送信において、VH メモリコピー処理を利用します。(Intel 以外のマシンを使用する場合の既定値)
	OFF	InfiniBand 通信の送信において、VH メモリコピー処理を利用しません。
NMPI_IB_VH_MEMCPY_RECV	**AUTO**	InfiniBand 通信の受信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。
	ON	InfiniBand 通信の受信において、 VH メモリコピー処理を利用します。(Intel 以外のマシンを使用する場合の既定値)
	OFF	InfiniBand 通信の受信において、VH メモリコピー処理を利用しません。(Intel マシンを使用する場合の既定値)
NMPI_DMA_VH_MEMCPY	**AUTO**	VH 内の VE 間通信において、RootComplex を経由する場合に、VH メモリコピー処理を利用します。(既定値)
	ON	VH 内の VE 間通信において、VH メモリコピー処理を利用します。
	OFF	VH メモリコピー処理を利用し

		ません。
NMPI_VH_MEMCPY	**AUTO**	InfiniBand 通信と VH 内の VE 間通信において、これらの通信が RootComplex を経由する場合に、VH メモリコピー処理を利用します。
	ON	InfiniBand 通信と VH 内の VE 間通信において、常に VH メモリコピー処理を利用します。
	OFF	VH メモリコピー処理を利用しません。
	備考： 環境変数 **NMPI_IB_VH_MEMCPY_SEND**, **NMPI_IB_VH_MEMCPY_RECV**, **NMPI_DMA_VH_MEMCPY** が明示的に指定された場合はこれらの設定値が優先されます。 上記環境変数が指定されていない場合、本環境変数の設定が有効になります。	
NMPI_DMA_RNDV_OVERLAP	**ON**	DMA 通信において通信バッファの転送長が 200KB 以上で連続領域であり、かつ 1 対 1 通信の非ブロッキング通信の場合はその通信と演算のオーバーラップが可能になります。
	OFF	(既定値) DMA 通信において通信バッファの転送長が 200KB 以上である 1 対 1 通信の非ブロッキング通信の場合は、その通信と演算のオーバーラップは行われません。
	備考： 本オプションが ON の場合は DMA 通信において 1 対 1 通信の非ブロッキング通信では VH メモリコピー処理を行いません。 本通信パターンにおいては 環境変数 **NMPI_DMA_VH_MEMCPY** の設定値は無視されます。	
NMPI_IB_VH_MEMCPY_THRESHOLD	*<整数値>*	Infiniband 通信において VH メモリコピー処理を開始する転送

		長をバイト単位で指定します。 既定値は 1048576 です。 本値は実行時オプション-v 指定時に出力される以下の項目に表示されます。 「IB Parameters for message transfer via VH memory」の「Threshold」
NMPI_IB_VH_MEMCPY_BUFFER_SIZE	*<整数値>*	Infiniband 通信の VH メモリコピー処理に割り当てる通信バッファのサイズをバイト単位で指定します。 既定値は 1048576 です。 本値は実行時オプション-v 指定時に出力される以下の項目に表示されます。 「IB Parameters for message transfer via VH memory」の「Buffer size」
NMPI_IB_VH_MEMCPY_SPLIT_THRESHOLD	*<整数値>*	Infninband 通信において VH メモリコピー処理を利用する場合に、通信バッファの分割転送を開始する転送長をバイト単位で指定します。 既定値は 1048576 です。 本値は実行時オプション-v 指定時に出力される以下の項目に表示されます。 「IB Parameters for message transfer via VH memory」の「Split threshold」
NMPI_IB_VH_MEMCPY_SPLIT_NUM	*<整数値>*	Infiniband 通信の VH メモリコピー処理に割り当てられた通信バッファの最大分割数を指定します。指定できる値は 1 から 8 までです。既定値は 2 です。

		本値は実行時オプション-v 指定時に出力される以下の項目に表示されます。 「IB Parameters for message transfer via VH memory」の「Split number」
NMPI_IP_USAGE	InfiniBand が搭載されているシステムにおいて InfiniBand 通信が利用できないとき(*)に TCP/IP を利用するかどうかを制御する。 (*) 例 1: InfiniBand のポートがダウンしている 例 2: HCA がジョブに割り当てられていない	
	ON	InfiniBand 通信が利用できなかった場合に TCP/IP を利用する。
	FALLBACK	
	OFF	(既定値) InfiniBand が搭載されているシステムにおいて InfiniBand 通信が利用できなかった場合にアプリケーションを終了させる。
NMPI_EXEC_MODE	**NECMPI**	NECMPI の実行時オプションで動作します。（既定値）
	INTELMPI	IntelMPI の基本的な実行時オプション（下記参照）で動作します。
	OPENMPI	OpenMPI の基本的な実行時オプション（下記参照）で動作します。
	MPICH	MPICH の基本的な実行時オプション（下記参照）で動作します。
	MPISX	MPISX の実行時オプションで動作します。
NMPI_SHARP_ENABLE	**ON**	SHARP 機能を利用します
	OFF	SHARP 機能を利用しません(既定値)
NMPI_SHARP_NODES	*<整数値>*	SHARP 機能を利用する最小 VE 数を指定します。既定値は 4 で

		す。
NMPI_SHARP_ALLREDUCE_MAX	*<整数値>* または **UNLIMITED**	MPI_ALLREDUCE において SHARP 機能を利用する場合の最大データサイズを指定します。単位はバイトです。なお、**UNLIMITED** を指定した場合は常に SHARP 機能を利用した動作となります。既定値は６４（バイト）です。
NMPI_SHARP_REPORT	SHARP 機能を利用するコミュニケータに関する情報の出力を制御します。	
	OFF	情報を出力しません。（既定値）
	ON	情報を出力します。
NMPI_DCT_ENABLE	Infiniband 通信において DCT (Dynamically Connected Transport Service) を使用するかを制御します。DCT を使用することで Inifniband 通信に必要なメモリを削減できます。(注) DCT は通信性能に影響する場合があります。	
	AUTOMATIC	MPI プロセス数が環境変数 **NMPI_DCT_SELECT_NP** に指定された値以上の場合に DCT 機能を利用します。(既定値)
	ON	DCT 機能を常に利用します。
	OFF	DCT 機能を利用しません。
NMPI_DCT_SELECT_NP	*<整数値>*	**NMPI_DCT_ENABLE** 環境変数が **AUTOMATIC** の場合に、DCT 機能を利用する最小プロセス数を指定します。既定値は 2049 です。
NMPI_DCT_NUM_CONNS	*<整数値>*	DCT のコネクション数を指定します。既定値は 4 です。
NMPI_COMM_PNODE	NQSV 実行における論理ノード間の通信種別の自動選択を制御します。	
	OFF	論理ノードを基準として通信種別を自動選択します（既定値）
	ON	物理ノードを基準として通信種別を自動選択します。
NMPI_EXEC_LNODE	インタラクティブ実行時において論理ノード実行を制御します。論理ノード実行時は論理ノードを基	

	準として通信種別を自動選択します。論理ノードは"ホスト名/文字列"の形式で指定します。 以下は1物理ノード上で3論理ノード実行する場合の実行例です。 % mpirun -host HOST1 -ve 0 -host HOST1/A -ve 1 -host HOST1/B -ve 2 ve.out	
	OFF	インタラクティブ実行時に論理ノード実行を利用しません（既定値）
	ON	インタラクティブ実行時に論理ノード実行を利用します。
NMPI_LNODEON **MPILNODEON**	環境変数 **NMPI_EXEC_LNODE** と同様です。両方の環境変数が指定された場合は環境変数 **NMPI_EXEC_LNODE** の設定が優先されます。	
	1	インタラクティブ実行時に論理ノード実行を利用します。
NMPI_VH_MEMORY_USAGE	MPI アプリケーションを実行する際に VH メモリが利用可能な状態であることを必須とするかどうかを制御する。	
	ON	VH メモリが利用可能な状態であることを必須とする。もし VH メモリを利用しようとしたときに VH メモリが利用できない場合はMPIアプリケーションをアボートさせる。(既定値)
	OFF **FALLBACK**	もし VH メモリを利用しようとしたときに VH メモリが利用できないときは遅い通信経路が利用される可能性がある。
NMPI_CUDA_ENABLE	CUDA によるメモリ転送の利用を制御します。	
	AUTOMATIC	可能であればCUDAによるメモリ転送を利用します。(既定値)
	ON	CUDA によるメモリ転送を利用します。 CUDA によるメモリ転送が利用できない場合はMPIアプリケーションをアボートさせます。
	OFF	CUDA によるメモリ転送を利用しません。

NMPI_CUDA_IPC_THRESHOLD	*<整数値>*	イントラノードの通信に IPC を利用する最大転送サイズを指定します。既定値は 32768(バイト)です。
NMPI_VE_AIO_METHOD	VE の MPI プログラムの非ブロッキング MPI-IO 手続きが使用する非同期 I/O の方式を指定します。	
	VEAIO	VE AIO を使用します。(既定値)
	POSIX	POSIX AIO を使用します。
NMPI_SWAP_ON_HOLD	NQSV で Partial Process Swapping 機能を使用して通常リクエストをサスペンドする場合、MPI プロセスが使用する Non Swappable Memory の解放を制御します。 モデル A412-8 または B401-8 上でのみ有効です。 既定値はシステム設定に依存し、実行時オプション -v を指定したときに表示される「Swappable on hold」の項目で確認できます。	
	ON	MPI プロセスが使用する Non Swappable Memory の一部を解放します
	OFF	MPI プロセスが使用する Non Swappable Memory を解放しません
NMPI_AVEO_UDMA_ENABLE	AVEO UserDMA 機能を制御します。	
	ON	AVEO UserDMA 機能を有効にします。(既定値)
	OFF	AVEO UserDMA 機能を無効にします。
NMPI_USE_COMMAND_SEARCH_PATH	MPI 実行コマンドに指定した実行可能ファイルの検索に PATH 環境変数を使用するかを制御します。 （注意）ファイル名の代わりにパス区切り文字を含むファイルパスを指定した場合は本環境変数の影響を受けません。	
	ON	PATH 環境変数を使用します。ファイルは PATH 環境変数に指定されたディレクトリを先頭から順番に検索されます。
	OFF	PATH 環境変数を使用しません。

		ファイルは現在の作業ディレクトリのみ検索されます。（既定値）
NMPI_OUTPUT_RUNTIMEINFO	NEC MPI プロセスマネージャに MPD が選択されたキューを使用して NQSV バッチジョブを実行する場合に、当該ジョブ内のオプション-v が指定された MPI 実行コマンドが出力するランタイム情報の頻度を制御します。	
	ON	オプション-v が指定された MPI 実行コマンドが実行されるたびにランタイム情報を出力します。（既定値）
	OFF	オプション-v が指定された最初の MPI 実行コマンドのみランタイム情報を出力します。それより後の MPI 実行コマンドはオプション-v が指定されていてもランタイム情報を出力しません。

- NMPI_EXEC_MODE=INTELMPI 設定時のサポートオプション
  -hosts, -f, -hostfile, -machinefile, -machine, -configfile, -perhost, -ppn, -genv, -genvall, -genvnone, -genvlist, -gpath, -gwdir, -gumask, -host, -n, -np, -env, -envall, -envnone, -envlist, -path, -wdir, -umask, および下記 Aurora 向け共通オプション

- NMPI_EXEC_MODE=OPENMPI 設定時のサポートオプション
  -N, -npernode, --npernode, -path, --path, -H, -host, --host, -n, --n, -c, -np, --np, -wdir, --wdir, -wd, --wd, -x, および下記 Aurora 向け共通オプション

- NMPI_EXEC_MODE=MPICH 設定時のサポートオプション
  -hosts, -f, -configfile, -ppn, -genv, -genvall, -genvnone, -genvlist, -wdir, -host, -n, -np, -env, -envall, -envnone, -envlist, および下記 Aurora 向け共通オプション

- Aurora 向け共通オプション
  -launcher-exec, -max_np, -multi, -debug, -display, -disp, -v, -V, -version, -h, -help, -gvenode, -ve, -venode, -ve_nnp

3.2.5 MPI プロセス識別用環境変数

NEC MPI は，MPI プロセス識別用に次の環境変数を用意し，自動的に値を設定します。

環境変数	値
MPIUNIVERSE	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD に対応する通信範囲の識別番号。
MPIRANK	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD における当該プロセスのランク。
MPISIZE	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD に属するプロセスの総数。
MPINODEID	MPI プロセスが動作するノードを示す論理ノード番号。
MPIVEID	MPI プロセスが動作する VE を示す VE 番号。 NQSV リクエスト実行の場合、論理 VE 番号を示します。また MPI プロセスが VE 上で実行されない場合、本環境変数は設定されません。
NMPI_LOCAL_RANK	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD に属するプロセスのノード内での相対ランク
NMPI_LOCAL_RANK_VHVE	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD に属するプロセスのノード内のホスト CPU 側または VE カード側での相対ランク
NMPI_LOCAL_RANK_DEVICE	MPI プログラム実行開始時のコミュニケーター MPI_COMM_WORLD に属するプロセスのノード内のホスト CPU 側または 1VE カード内での相対ランク

これらの環境変数は，MPI プログラムの実行中は，たとえ手続 MPI_INIT または MPI_INIT_THREAD の呼出し前であっても，いつでも引用可能です。

MPI プログラムの実行開始時点で，コミュニケーターMPI_COMM_WORLD に対応し，かつ 全ての MPI プロセスを含む事前定義された通信範囲が存在しています。 この通信範囲の識別番号は，0 です。

1 つの通信範囲において，各プロセスは，ランクと呼ばれる一意な整数値をもちます。ランクは，0 以上，プロセスの個数未満です。

動的プロセス生成機能を使用して，実行時に一群のプロセスを生成すると，新たなコミュニケーター MPI_COMM_WORLD に対応する，新たな通信範囲が生成されます。このように実行時に生成される通信範囲に含まれるプロセスには，1 から始まる連続した整数値の識別番号が割り当てられます。このような場合，1 つの MPI アプリケーションの実行中に，コミュニケーターMPI_COMM_WORLD が，同時に複数存在する可能性があります。そのため，MPI プロセスの識別には，環境変数 **MPIUNIVERSE** および **MPIRANK** の対の値を使用します。

SX-Aurora TSUBASA システムでは、ノードを構成するホスト CPU または VE カード上で MPI プロセスが動作します。環境変数 MPINODEID, MPIVEID, NMPI_LOCAL_RANK, NMPI_LOCAL_RANK_VHVE

および NMPI_LOCAL_RANK_DEVICE により、各 MPI プロセスについて、動作している場所、およびノード・CPU 側・VE カード側・1VE カード内の MPI プロセスグループ内における一意な番号が得られます。環境変数 MPIRANK は MPI_COMM_WORLD の MPI プロセスグループ内における一意な番号ですが、環境変数 NMPI_LOCAL_RANK, NMPI_LOCAL_RANK_VHVE および NMPI_LOCAL_RANK_DEVICE は、MPI_COMM_WORLD を以下の様にさらに細分化したグループ内における一意な番号になります。

- MPIRANK
  ↓MPI_COMM_WORLD のプロセスグループをノードごとに細分化
- NMPI_LOCAL_RANK
  ↓グループをホスト CPU 側と VE カード側に細分化
- NMPI_LOCAL_RANK_VHVE
  ↓グループを 1VE カード毎に細分化（ホスト CPU 側は細分化されない）
- NMPI_LOCAL_RANK_DEVICE

VH-VE ハイブリッド実行時の各環境変数の値の例

```
mpirun ¥
-host hostA -vh     -np 2 ./vh.out : ¥
-host hostA -ve 0-1 -np 4 ./ve.out : ¥
-host hostB -ve 0-1 -np 6 ./ve.out : ¥
-host hostB -vh     -np 2 ./vh.out

MPIRANK                0  1  2  3  4  5  6  7  8  9 10 11 12 13
NMPI_LOCAL_RANK        0  1  2  3  4  5  0  1  2  3  4  5  6  7
NMPI_LOCAL_RANK_VHVE   0  1  0  1  2  3  0  1  2  3  4  5  0  1
NMPI_LOCAL_RANK_DEVICE 0  1  0  1  0  1  0  1  2  0  1  2  0  1
MPINODEID              0  0  0  0  0  0  1  1  1  1  1  1  1  1
MPIVEID                -  -  0  0  1  1  0  0  0  1  1  1  -  -
```

MPI プログラムが，シェルスクリプト経由で間接的に起動される場合，これらの環境変数は，シェルスクリプト中でも引用することができ，たとえば，各プロセスが互いに異なるファイルを処理するために利用できます。図 **3-1** のシェルスクリプトは，各プロセスが，それぞれ異なるファイルからデータを読み出し，異なるファイルへデータを保存するように記述されており，図 **3-2** のコマンド行では，そのシェルスクリプト経由で，MPI プログラムを間接的に起動しています。

図 3-1 MPI プログラム起動用シェルスクリプト mpi.shell の例

```
#!/bin/sh
INFILE=infile.$MPIUNIVERSE.$MPIRANK
OUTFILE=outfile.$MPIUNIVERSE.$MPIRANK
```

```
{MPIexec}  < $INFILE > $OUTFILE # MPI 実行指定{MPIexec}については，3.2.1 項を参照してください
exit $?
```

図 3-2 シェルスクリプトを利用した間接的な MPI プログラムの起動例

```
$ mpirun -np 8 /execdir/mpi.shell
```

3.2.6 他の処理系用の環境変数の指定

Fortran コンパイラ，C コンパイラ，または C++コンパイラなどの他の処理系の環境変数は、MPI プロセスに渡されます。これは実行時オプション-genvall が既定値で有効なためです。以下の例では，MPI プロセスに環境変数 **OMP_NUM_THREADS** および **VE_LD_LIBRARY_PATH** が渡されます。

```
#!/bin/sh
#PBS -T necmpi
#PBS -b 2

OMP_NUM_THREADS=8 ; export OMP_NUM_THREADS
VE_LD_LIBRARY_PATH={your shared library path} ; export VE_LD_LIBRARY_PATH

mpirun -node 0-1 -np 2 a.out
```

3.2.7 ランクの割当て

ランクは，各 MPI プロセスに対して，NEC MPI がホストを割り当てた順に，昇順に割り当てられます。

3.2.8 MPI プログラムの実行ディレクトリ

MPI プログラムの実行ディレクトリは，次のように決定されます。

- MPI 実行コマンドを実行した現在の作業ディレクトリ
- 上記が利用不可の場合は、利用者のホームディレクトリ

3.2.9 singularity コンテナを使用した MPI プログラムの実行

singularity コンテナ内で MPI プログラムを実行することができます。以下のように、mpirun の引数に singularity コマンドを指定します。

```
$ mpirun -ve 0 -np 8 /usr/bin/singularity exec --bind /var/opt/nec/ve/veos ./nmpi.sif ./ve.out
```

本実行において、singularity コマンドの名前空間関連のオプションはご利用になれません。
singularity イメージファイルの作成方法に関しては、以下をご参照ください。

https://github.com/veos-sxarr-NEC/singularity

3.2.10 MPI プログラム実行例

本項では，SX-Aurora TSUBASA 上での MPI プログラムの実行例を示します。

- インタラクティブ実行
  - 1 つの VE 上での実行

    ローカル VH の VE#3 上で，4 プロセスを使用する例

    ```
    $ mpirun -ve 3 -np 4 ./ve.out
    ```

  - 1 つの VH 上の，複数の VE 上での実行

    ローカル VH の VE#0 から VE#7 までの上で，16 プロセス(VE あたり 2 プロセス)を使用する例

    ```
    $ mpirun -ve 0-7 -np 16 ./ve.out
    ```

  - 複数の VH 上の，複数の VE 上での実行

    2 つの VH host1 および host2 それぞれの VE#0 および VE#1 の上で，計 32 プロセス（VE あたり 8 プロセス)を使用する例

    ```
    $ mpirun -hosts host1,host2 -ve 0-1 -np 32 ./ve.out
    ```

    VH host1 の VE#0 および VE#1、VH host2 の VE#2 および VE#3 の上で, 計 32 プロセス (VE あたり 8 プロセス)を使用する例

    ```
    $ mpirun -host host1 -ve 0-1 -np 16 -host host2 -ve 2-3 -np 16 ./ve.out
    ```

  - 1 つの VH 上と複数の VE 上でのハイブリッド実行

    VH host1 の上で vh.out を 8 プロセス、VH host1 の VE#0 および VE#1 の上で ve.out を計 16 プロセス（VE あたり 8 プロセス）使用する例

```
$ mpirun -vh -host host1 -np 8 vh.out : -host host1 -ve 0-1 -np 16 ve.out
```

- NQSV リクエスト実行

  NQSV リクエスト実行には，バッチジョブ および 会話ジョブがあります。バッチジョブにおいては，プログラム実行のための指示を記述したジョブスクリプトファイルを，qsub コマンドにより実行します。会話ジョブにおいては，qlogin コマンドを実行後に，会話的にコマンドを実行します。
  NQSV リクエスト実行の場合, MPI プロセスの VH および VE への割当ては, NQSV が自動的に行い，利用者は，それらの論理番号を指定できます。

  以下の例では，バッチジョブにおけるジョブスクリプトファイルの内容を示していますが，スクリプト中のコマンド行は，会話ジョブでも使用可能です。

  - 特定の VH 上の，VE 上での実行

    論理 VH#0 上の論理 VE#0 から論理 VE#3 までの上で, 32 プロセス(VE あたり 8 プロセス)を使用する例

```
#PBS -T necmpi
#PBS -b 2 # Number of VHs
#PBS --venum-lhost=4 # Number of VEs

source /opt/nec/ve/mpi/2.3.0/bin/necmpivars.sh
mpirun -host 0 -ve 0-3 -np 32 ./ve.out
```

  - 割り当てられた全ての VE 上での実行

    割り当てられた 4 つの VH それぞれの論理 VE#0 から論理 VE#7 までの上で, 計 32 プロセス(VE あたり 1 プロセス)を使用する例

```
#PBS -T necmpi
#PBS -b 4 # Number of VHs
#PBS --venum-lhost=8 # Number of VEs
#PBS --use-hca=2 # Number of HCAs

source /opt/nec/ve/mpi/2.3.0/bin/necmpivars.sh
mpirun -np 32 ./ve.out
```

  - 特定の VH 上と、割り当てられた全ての VE 上での VH-VE ハイブリッド MPI 実行

    論理 VH#0 上に vh.out を 1 プロセス、割り当てられた 4 つの VH それぞれの論理 VE#0 から論理 VE#7 までの上に ve.out を計 32 プロセス(VE あたり 1 プロセス)使用する例

```
#PBS -T necmpi
```

```
#PBS -b 4 # Number of VHs
#PBS --venum-lhost=8 # Number of VEs
#PBS --use-hca=2 # Number of available HCAs

source /opt/nec/ve/mpi/2.3.0/bin/necmpivars.sh
mpirun -vh -host 0 -np 1 vh.out : -np 32 ./ve.out
```

- PBS リクエスト実行

  PBS のジョブスクリプトでは、接頭辞"#PBS"で始まる PBS 指示行に、必要な計算資源などを指定します。例えば、使用する VE の台数は資源 nves で指定し、実行する MPI プロセスの個数は資源 mpiprocs で指定します。以下の例では、4 台の VE 上で 8 個の MPI プロセスを実行することを指定しています。PBS 指示行の"-l select="で始まる指定を selection 指示行と呼びます。

```
#PBS -l select=nves=4:mpiprocs=8
```

  "資源=値"を":"で連結した"nves=4:mpiprocs=8"のような要求資源の単位を chunk と呼びます。1 つの chunk で要求した資源は、必ず 1 つの VH から割り当てられます。したがって、例えば 4 つの VE が接続されている VH に対しては、nves の値が 4 以下になるように指定してください。

  同一の chunk を複数個要求する場合、次のように chunk の前に"個数:"を指定します。この場合、1 台の VE 上で 2 個の MPI プロセスを実行する chunk を 4 組要求していることになります。このような同一の chunk の組を chunk set と呼びます。

```
#PBS -l select=4:nves=1:mpiprocs=2
```

  - VE 上で MPI プログラムを実行する場合

    次のジョブスクリプトは、各 VE 上で 8 個の MPI プロセスを実行し、4 台の VE で合計 32 個の MPI プロセスを実行する場合の例です。1 台の VE 上で実行する MPI プロセス数を指定した chunk "nves=1:mpiprocs=8"を記述し、その chunk を 4 つ要求しています。
    mpirun コマンドにはオプション-np を指定し、MPI プロセスの総数を指定してください。

```
#!/bin/bash
#PBS -l select=4:nves=1:mpiprocs=8

source /opt/nec/ve/mpi/<version>/bin/necmpivars.sh
mpirun -np 32 ./a.out
```

  - OpenMP を併用したハイブリッド並列プログラムを実行する場合

次のジョブスクリプトは、4 個のスレッドからなる MPI プロセスを、各 VE 上で 2 個実行し、8 台の VE を使用して、合計 16 個の MPI プロセスで実行する例です。

```
#!/bin/bash
#PBS -l select=8:nves=1:mpiprocs=2:ompthreads=4

source /opt/nec/ve/mpi/<version>/bin/necmpivars.sh
mpirun -np 16 ./a.out
```

- VH-VE ハイブリッド MPI プログラムを実行する場合

  MPI プロセスを VH 上でも実行する場合、環境変数 NEC_PROCESS_DIST を指定して、各 chunk で実行する MPI プロセスの配置を指定する必要があります。

  VH 上で 2 個の MPI プロセスを実行すると同時に、各 VE 上で 4 個の MPI プロセスを実行し、8 台の VE を使用して、合計 34 個の MPI プロセスで VH-VE ハイブリッド MPI プログラムを実行する場合、次のように指定できます。

```
#!/bin/bash
#PBS -l select=ncpus=2:mpiprocs=2+8:nves=1:mpiprocs=4
#PBS -v NEC_PROCESS_DIST=s2+4

source /opt/nec/ve/mpi/<version>/bin/necmpivars.sh
mpirun -vh -np 2 vh.out : -np 32 ./ve.out
```

  VH 上で実行する MPI プロセスの場合、PBS 指示行には、上記の"ncpus=2:mpiprocs=2"のように、MPI プロセスを実行する VH のコアの個数を資源 ncpus で指定してください。この例の"ncpus=2:mpiprocs=2"および"8:nves=1:mpiprocs=4"のように、異なる種類の chunk set を指定する場合、それらを"+"で連結します。

  環境変数 NEC_PROCESS_DIST には、selection 指示行の各 chunk set で実行する MPI プロセスの配置を"+"で連結して指定します。上の例では、最初の chunk set は、VH 上で実行する MPI プロセスの個数を文字"s"に続けて指定しています。2 番目の chunk set は、各 VE 上で実行する MPI プロセスの個数を指定しています。

  selection 指示行および環境変数 NEC_PROCESS_DIST における VH プロセスおよび VE プロセスの順序が、コマンド mpirun における順序と一致するように指定してください。これは、MPI プロセスのランクが selection 指示行および環境変数 NEC_PROCESS_DIST で指定した chunk の順番で決定されるためです。

- VEO 機能を併用する場合

  MPI 実行ファイルに対して-veo オプションを指定します。

```
$ mpirun -veo -np 8 ./mpi-veo
```

- CUDA 機能を併用する場合

  MPI 実行ファイルに対して-cuda オプションを指定します。

```
$ mpirun -cuda -np 8 ./mpi-cuda
```

3.3 MPI プロセスの標準出力 および 標準エラー出力

各 MPI プロセスからの出力を分離するために，NEC MPI は，シェルスクリプト mpisep.sh をディレクトリ/opt/nec/ve/bin/に用意しています。

次のように,MPI 実行指定*{MPIexec}*の前に上記のスクリプトを指定すると,MPI プロセスからの出力を，プロセスごとに異なるファイルに保存することができます。(MPI 実行指定*{MPIexec}*については，3.2.1 項を参照してください。)

```
$ mpirun -np 2 /opt/nec/ve/bin/mpisep.sh {MPIexec}
```

出力の保存先は，環境変数 **NMPI_SEPSELECT** を使用して，次のように指定できます。ここで，uuu は，コミュニケーターMPI_COMM_WORLD に対応する事前定義された通信範囲の識別番号，rrr は，その通信範囲における当該プロセスのランクです。

NMPI_SEPSELECT	動作
1	各 MPI プロセスの標準出力だけを stdout.*uuu*:*rrr* へ保存します。
2	(既定値) 各 MPI プロセスの標準エラー出力だけを stderr.*uuu*:*rrr* へ保存します。
3	各 MPI プロセスの標準出力 および 標準エラー出力を，それぞれ stdout.uuu:rrr および stderr.uuu:rrr に保存します。
4	各 MPI プロセスの標準出力 および 標準エラー出力を，同一のファイル std.*uuu*:*rrr* に保存します。

3.4 実行性能情報

MPI プログラムの性能情報は，環境変数 **NMPI_PROGINF** を使用して取得できます。NEC MPI では，次の 4 種類の表示形式を選択できます。

集約形式	3 つの部分から構成されます。まず，Global Data 部で，全プロセスの性能値の最大，最小 および 平均が表示されます。続いて，Overall Data 部で，MPI プロセス全体の性能が表示されます。最後に，VE Card Data 部で, VE カード毎に集計された性能値の最大，最小 および 平均が表示されます。ベクトルプロセスとスカラプロセスの結果は分かれて集計出力されます。

拡張形式	集約形式の内容の後に，各 MPI プロセスの性能情報が，コミュニケーターMPI_COMM_WORLD におけるランクの昇順で表示されます。
詳細集約形式	3 つの部分から構成されます。まず，Global Data 部で，全プロセスの詳細な性能値の最大，最小 および 平均が表示されます。続いて，Overall Data 部で，MPI プロセス全体の性能が表示されます。最後に，VE Card Data 部で, VE カード毎に集計された性能値の最大，最小 および 平均が表示されます。ベクトルプロセスとスカラプロセスの結果は分かれて集計出力されます。
詳細拡張形式	詳細集約形式の内容の後に，各 MPI プロセスの詳細な性能情報が，コミュニケーターMPI_COMM_WORLD におけるランクの昇順で表示されます。

実行性能情報の表示形式は，環境変数 **NMPI_PROGINF** を，表 3-8 のように実行時に指定することにより選択できます。

表 3-8 環境変数 NMPI_PROGINF の値

NMPI_PROGINF	**表示される情報**
NO	性能情報を出力しません(既定値)。
YES	性能情報を集約形式で出力します。
ALL	性能情報を拡張形式で出力します。
DETAIL	性能情報を詳細集約形式で出力します。
ALL_DETAIL	性能情報を詳細拡張形式で出力します。

次の図は，詳細拡張形式の性能情報の出力例です。

図 3-3 詳細拡張形式の MPI プログラム性能情報（NMPI_PROGINF=ALL_DETAIL）

```
MPI Program Information:
========================
Note: It is measured from MPI_Init till MPI_Finalize.
      [U,R] specifies the Universe and the Process Rank in the Universe.
      Times are given in seconds.


Global Data of 4 Vector processes      :          Min [U,R]          Max [U,R]      Average
==================================

Real Time (sec)                        :       25.203 [0,3]       25.490 [0,2]       25.325
User Time (sec)                        :      199.534 [0,0]      201.477 [0,2]      200.473
Vector Time (sec)                      :       42.028 [0,2]       42.221 [0,1]       42.104
```

Inst. Count	:	94658554061 [0,1]	96557454164 [0,2]	95606075636
V. Inst. Count	:	11589795409 [0,3]	11593360015 [0,0]	11591613166
V. Element Count	:	920130095790 [0,3]	920199971948 [0,0]	920161556564
V. Load Element Count	:	306457838070 [0,1]	306470712295 [0,3]	306463228635
FLOP Count	:	611061870735 [0,3]	611078144683 [0,0]	611070006844
MOPS	:	6116.599 [0,2]	6167.214 [0,0]	6142.469
MOPS (Real)	:	48346.004 [0,2]	48891.767 [0,3]	48624.070
MFLOPS	:	3032.988 [0,2]	3062.528 [0,0]	3048.186
MFLOPS (Real)	:	23972.934 [0,2]	24246.003 [0,3]	24129.581
A. V. Length	:	79.372 [0,1]	79.391 [0,3]	79.382
V. Op. Ratio (%)	:	93.105 [0,2]	93.249 [0,1]	93.177
L1 Cache Miss (sec)	:	3.901 [0,0]	4.044 [0,2]	3.983
CPU Port Conf. (sec)	:	3.486 [0,1]	3.486 [0,2]	3.486
V. Arith. Exec. (sec)	:	15.628 [0,3]	15.646 [0,1]	15.637
V. Load Exec. (sec)	:	23.156 [0,2]	23.294 [0,1]	23.225
VLD LLC Hit Element Ratio (%)	:	90.954 [0,2]	90.965 [0,1]	90.959
Power Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
Thermal Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
Max Active Threads	:	8 [0,0]	8 [0,0]	8
Available CPU Cores	:	8 [0,0]	8 [0,0]	8
Average CPU Cores Used	:	7.904 [0,2]	7.930 [0,3]	7.916
Memory Size Used (MB)	:	1616.000 [0,0]	1616.000 [0,0]	1616.000
Non Swappable Memory Size Used (MB)	:	115.000 [0,1]	179.000 [0,0]	131.000
Global Data of 8 Scalar processes	:	Min [U,R]	Max [U,R]	Average
================================				
Real Time (sec)	:	25.001 [0,7]	25.010 [0,8]	25.005
User Time (sec)	:	199.916 [0,7]	199.920 [0,8]	199.918
Memory Size Used (MB)	:	392.000 [0,7]	392.000 [0,8]	392.000
Overall Data of 4 Vector processes				
================================				
Real Time (sec)	:	25.490		
User Time (sec)	:	801.893		
Vector Time (sec)	:	168.418		
GOPS	:	5.009		

```
GOPS (Real)                              :       157.578
GFLOPS                                   :         3.048
GFLOPS (Real)                            :        95.890
Memory Size Used (GB)                    :         6.313
Non Swappable Memory Size Used (GB)      :         0.512

Overall Data of 8 Scalar processes
==================================
Real Time (sec)                          :        25.010
User Time (sec)                          :      1599.344
Memory Size Used (GB)                    :         3.063



VE Card Data of 2 VEs
=====================

Memory Size Used (MB) Min                :      3232.000 [node=0,ve=0]
Memory Size Used (MB) Max                :      3232.000 [node=0,ve=0]
Memory Size Used (MB) Avg                :      3232.000
Non Swappable Memory Size Used (MB) Min  :       230.000 [node=0,ve=1]
Non Swappable Memory Size Used (MB) Max  :       294.000 [node=0,ve=0]
Non Swappable Memory Size Used (MB) Avg  :       262.000



Data of Vector Process [0,0] [node=0,ve=0]:
-------------------------------------------

  Real Time (sec)                        :            25.216335
  User Time (sec)                        :           199.533916
  Vector Time (sec)                      :            42.127823
  Inst. Count                            :         94780214417
  V. Inst. Count                         :         11593360015
  V. Element Count                       :        920199971948
  V. Load Element Count                  :        306461345333
  FLOP Count                             :        611078144683
  MOPS                                   :          6167.214211
  MOPS (Real)                            :         48800.446081
  MFLOPS                                 :          3062.527699
  MFLOPS (Real)                          :         24233.424158
```

```
A. V. Length                          :          79.373018
V. Op. Ratio (%)                      :          93.239965
L1 Cache Miss (sec)                   :           3.901453
CPU Port Conf. (sec)                  :           3.485787
V. Arith. Exec. (sec)                 :          15.642353
V. Load Exec. (sec)                   :          23.274564
VLD LLC Hit Element Ratio (%)         :          90.957228
Power Throttling (sec)                :           0.000000
Thermal Throttling (sec)              :           0.000000
Max Active Threads                    :                  8
Available CPU Cores                   :                  8
Average CPU Cores Used                :           7.912883
Memory Size Used (MB)                 :        1616.000000
Non Swappable Memory Size Used (MB)   :         179.000000
 ...
```

表 3-9 は，Global Data 部分および各プロセス部分の表示項目です。スカラプロセスは(*1)の項目のみ出力されます。ベクトルプロセスの場合、各プロセス部分のヘッダにユニバース番号と MPI_COMM_WORLD のランク番号に追加で、そのプロセスが実行された VE カード情報としてホスト名または論理ノード番号と論理 VE カード番号が表示されます。

表 3-9 Global Data 部分および各プロセス部分の表示項目

(*1) スカラプロセスがサポートする項目

(*2) 詳細集約形式 または 詳細拡張形式の場合のみ出力

(*3) マルチスレッド実行，かつ 詳細集約形式 または 詳細拡張形式の場合のみ出力

項目	単位	説明
Real Time (sec)	秒	経過時間 (*1)
User Time (sec)	秒	ユーザーCPU 時間 (*1)
Vector Time (sec)	秒	ベクトル命令実行時間
Inst. Count	-	実行命令回数
V.Inst. Count	-	ベクトル命令実行回数
V.Element Count	-	ベクトル命令で処理された要素の個数
V.Load Element Count	-	ベクトルロードされた要素の個数
FLOP Count	-	浮動小数点演算命令で処理された要素の個数
MOPS	-	ユーザーCPU 時間 1 秒あたりの 100 万演算実行回数
MOPS (Real)	-	経過時間 1 秒あたりの 100 万演算実行回数
FLOPS	-	ユーザーCPU 時間 1 秒あたりの 100 万浮動小数

		点演算実行回数
FLOPS (Real)	-	経過時間 1 秒あたりの 100 万浮動小数点演算実行回数
A.V.Length	-	平均ベクトル長
V.OP.RATIO	%	ベクトル演算率
L1 Cache Miss (sec)	秒	L1 キャッシュミス時間
CPU Port Conf.	秒	CPU ポート競合時間（*2)
V. Arith Exec.	秒	ベクトル演算実行時間（*2)
V. Load Exec.	秒	ベクトルロード命令実行時間（*2)
VLD LLC Hit Element Ratio	%	ベクトルロード命令によりロードされた要素のうち，LLC からロードされた要素の割合（*2)
FMA Element Count	-	FMA 命令実行要素数（*1)
Power Throttling	秒	電力要因によりハードウェアが減速した時間（*2)
Thermal Throttling	秒	温度要因によりハードウェアが減速した時間（*2)
Max Active Threads		同時に有効となったスレッドの最大個数（*3)
Available CPU Cores		プロセスが使用可能な CPU コアの個数（*3)
Average CPU Cores Used		平均使用 CPU コア個数（*3)
Memory Size Used (MB)	メガバイト	メモリの最大使用量（*1)
Non Swappable Memory Size Used (MB)	メガバイド	Partial Process Swapping 機能でスワップアウトできないメモリの最大使用量

表 3-10 は，Overall Data 部の表示項目です。スカラプロセスは(*1)の項目のみ出力されます。

表 3-10 Overall Data 部の表示項目

(*1) スカラプロセスがサポートする項目

項目	単位	説明
Real Time (sec)	秒	全 MPI プロセスの経過時間の最大値（*1)
User Time (sec)	秒	全 MPI プロセスのユーザーCPU 時間の総和（*1)
Vector Time (sec)	秒	全 MPI プロセスのベクトル命令実行時間の総和
GOPS	-	全 MPI プロセスの 10 億演算実行回数の総和を, 全 MPI プロセスのユーザーCPU 時間の総和で割算した値
GOPS (Real)	-	全 MPI プロセスの 10 億演算実行回数の総和を, 全 MPI プロセスの経過時間の最大値で割算した値
GFLOPS	-	全 MPI プロセスの 10 億浮動小数点演算実行回数の総和を，全 MPI プロセスのユーザーCPU 時間の総和で割算した値
GFLOPS (Real)	-	全 MPI プロセスの 10 億浮動小数点演算実行回数の総和を，全 MPI プロセスの経過時間の最大値で割算した値

Memory Size Used (GB)	ギガバイト	メモリの最大使用量の全 MPI プロセスの総和 (*1)
Non Swappable Memory Size Used (GB)	ギガバイド	Partial Process Swapping 機能でスワップアウトできないメモリの最大使用量の全 MPI プロセスの総和

表 3-11 は，VE Card Data 部の表示項目です。最大値および最小値の場合、その値となった VE カードの位置情報としてノード名または論理ノード番号と論理 VE カード番号が表示されます。

表 3-11 VE Card Data 部の表示項目

項目	単位	説明
Memory Size Used (MB) Min	メガバイト	メモリの最大使用量を VE カード毎に集計した値の最小値
Memory Size Used (MB) Max	メガバイト	メモリの最大使用量を VE カード毎に集計した値の最大値
Memory Size Used (MB) Avg	メガバイト	メモリの最大使用量を VE カード毎に集計した値の平均値
Non Swappable Memory Size Used (MB) Min	メガバイド	Partial Process Swapping 機能でスワップアウトできないメモリの最大使用量を VE カード毎に集計した値の最小値
Non Swappable Memory Size Used (MB) Max	メガバイド	Partial Process Swapping 機能でスワップアウトできないメモリの最大使用量を VE カード毎に集計した値の最大値
Non Swappable Memory Size Used (MB) Avg	メガバイド	Partial Process Swapping 機能でスワップアウトできないメモリの最大使用量を VE カード毎に集計した値の平均値

MPI の実行性能情報は Aurora ハードウェアの性能カウンタを採取して出力しています。採取する性能カウンタは VE_PERF_MODE 環境変数で変更することができ、これにより Global Data 部と各プロセス部の出力項目が切り替わります。図 3-3 は VE_PERF_MODE が未指定、または VE_PERF_MODE に VECTOR-OP を指定した場合の出力です。この場合、主にベクトル演算に関する情報が得られます。図 3-4 は VE_PERF_MODE に VECTOR-MEM を指定した場合の出力です。この場合、主にベクトルとメモリアクセス関連の情報が得られます。

図 3-4 詳細拡張形式の MPI プログラム性能情報
(VE_PERF_MODE=VECTOR-MEM 指定時の Global Data 部分)

```
Global Data of 16 Vector processes      :          Min [U,R]          Max [U,R]       Average
================================
```

Real Time (sec)	:	123.871 [0,12]	123.875 [0,10]	123.873
User Time (sec)	:	123.695 [0,0]	123.770 [0,4]	123.753
Vector Time (sec)	:	33.675 [0,8]	40.252 [0,14]	36.871
Inst. Count	:	94783046343 [0,8]	120981685418 [0,5]	109351879970
V. Inst. Count	:	2341570533 [0,8]	3423410840 [0,0]	2479317774
V. Element Count	:	487920413405 [0,15]	762755268183 [0,0]	507278230562
V. Load Element Count	:	47201569500 [0,8]	69707680610 [0,0]	49406464759
FLOP Count	:	277294180692 [0,15]	434459800790 [0,0]	287678800758
MOPS	:	5558.515 [0,8]	8301.366 [0,0]	5863.352
MOPS (Real)	:	5546.927 [0,8]	8276.103 [0,0]	5850.278
MFLOPS	:	2243.220 [0,15]	3518.072 [0,0]	2327.606
MFLOPS (Real)	:	2238.588 [0,13]	3507.366 [0,0]	2322.405
A. V. Length	:	197.901 [0,5]	222.806 [0,0]	204.169
V. Op. Ratio (%)	:	83.423 [0,5]	90.639 [0,0]	85.109
L1 I-Cache Miss (sec)	:	4.009 [0,5]	8.310 [0,0]	5.322
L1 O-Cache Miss (sec)	:	11.951 [0,5]	17.844 [0,9]	14.826
L2 Cache Miss (sec)	:	7.396 [0,5]	15.794 [0,0]	9.872
FMA Element Count	:	106583464050 [0,8]	166445323660 [0,0]	110529497704
Required B/F	:	2.258 [0,0]	3.150 [0,5]	2.948
Required Store B/F	:	0.914 [0,0]	1.292 [0,5]	1.202
Required Load B/F	:	1.344 [0,0]	1.866 [0,6]	1.746
Actual V. Load B/F	:	0.223 [0,0]	0.349 [0,14]	0.322
Power Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
Thermal Throttling (sec)	:	0.000 [0,0]	0.000 [0,0]	0.000
Memory Size Used (MB)	:	598.000 [0,0]	598.000 [0,0]	598.000
Non Swappable Memory Size Used (MB)	:	115.000 [0,1]	179.000 [0,0]	131.000

VE_PERF_MODE に VECTOR-MEM を指定した場合は VE_PERF_MODE 未指定時、または VE_PERF_MODE に VECTOR-OP を指定した場合に出力されていた項目 L1 Cache Miss、CPU Port Conf.、V. Arith Exec.、V. Load Exec.、VLD LLC Hit Element Ratio の代わりに表 3-12 の項目が追加で出力されます。

表 3-12 VE_PERF_MODE=VECTOR-MEM 指定時の追加表示項目

(*1) 詳細集約形式 または 詳細拡張形式の場合のみ出力

(*2) 100 以上の値は切り捨てられます

項目	単位	説明
L1 I-Cache Miss (sec)	秒	L1 命令キャッシュミス時間
L1 O-Cache Miss (sec)	秒	L1 オペランドキャッシュミス時間

L2 Cache Miss (sec)	秒	L2 キャッシュミス時間
Required B/F	-	ロード命令とストア命令に指定されたバイト数から算出した B/F (*1)(*2)
Required Store B/F	-	ストア命令に指定されたバイト数から算出した B/F (*1)(*2)
Required Load B/F	-	ロード命令に指定されたバイト数から算出した B/F (*1)(*2)
Actual V. Load B/F	-	ベクトルロード命令により実際に発生したメモリアクセスのバイト数から算出した B/F (*1)(*2)

3.5 MPI 通信情報

NEC MPI は MPI の通信情報を出力する機能を提供します。本機能は、オプション-mpiprof、-mpitrace、-mpiverify または-ftrace を指定して MPI プログラムを作成することで利用可能になります。

本機能は、MPI 通信情報として、全 MPI 手続き実行所要時間、MPI 通信待ち合わせ時間、送受信データ総量、および主要 MPI 手続き呼出し回数を表示します。表示形式は次の 2 種類から選択できます。

集約形式	全 MPI プロセスの MPI 通信情報の最大値，最小値，および 平均値が表示されます。
拡張形式	集約形式の内容の後に，各 MPI プロセスの MPI 通信情報が，コミュニケーター MPI_COMM_WORLD におけるランクの昇順で表示されます。

MPI 通信情報の有無および表示形式は，環境変数 **NMPI_COMMINF** を表 3-13 のように実行時に指定することにより選択できます。

表 3-13 環境変数 NMPI_COMMINF の値

NMPI_COMMINF	**表示される情報**
NO	MPI 通信情報を出力しません(既定値)。
YES	MPI 通信情報を集約形式で出力します。
ALL	MPI 通信情報を拡張形式で出力します。

また、環境変数 **NMPI_COMMINF_VIEW** を追加で指定することで集約形式の部分の集計方法と表示を以下のように変更できます。

表 3-14 環境変数 NMPI_COMMINF_VIEW の値

NMPI_COMMINF_VIEW	**表示形式**
VERTICAL	ベクトルプロセスとスカラプロセスを分けて集計し、縦に並べて表示しま

	す。ベクトルプロセスにのみ対応した項目について、スカラプロセス部分には出力されません。(既定値)
HORIZONTAL	ベクトルプロセスとスカラプロセスを分けて集計し、横に並べてして表示します。ベクトルプロセスにのみ対応した項目について、スカラプロセス部分の当該項目には N/A が出力されます。
MERGED	ベクトルプロセスとスカラプロセスをまとめて集計し表示します。ベクトルプロセスにのみ対応した項目は行末に(V)が付きます。(V)がついた項目についてはベクトルプロセスのみを集計し、最大・最少・平均を表示します。

図 3-5 は，拡張形式の MPI 通信情報の出力例です。

図 3-5 拡張形式の MPI 通信情報（NMPI_COMMINF=ALL）

```
MPI Communication Information of 4 Vector processes
---------------------------------------------------
                                                Min [U,R]          Max [U,R]       Average

Real MPI Idle Time (sec)                :       9.732 [0,1]       10.178 [0,3]         9.936
User MPI Idle Time (sec)                :       9.699 [0,1]       10.153 [0,3]         9.904
Total real MPI Time (sec)               :      13.301 [0,0]       13.405 [0,3]        13.374
Send       count                        :        1535 [0,2]         2547 [0,1]          2037
   Memory Transfer                      :         506 [0,3]         2024 [0,0]          1269
   DMA Transfer                         :           0 [0,0]         1012 [0,1]           388
Recv       count                        :        1518 [0,2]         2717 [0,0]          2071
   Memory Transfer                      :         506 [0,2]         2024 [0,1]          1269
   DMA Transfer                         :           0 [0,3]         1012 [0,2]           388
Barrier       count                     :        8361 [0,2]         8653 [0,0]          8507
Bcast         count                     :         818 [0,2]          866 [0,0]           842
Reduce        count                     :         443 [0,0]          443 [0,0]           443
Allreduce     count                     :        1815 [0,2]         1959 [0,0]          1887
Scan          count                     :           0 [0,0]            0 [0,0]             0
Exscan        count                     :           0 [0,0]            0 [0,0]             0
Redscat       count                     :         464 [0,0]          464 [0,0]           464
Redscat_block count                     :           0 [0,0]            0 [0,0]             0
Gather        count                     :         864 [0,0]          864 [0,0]           864
Gatherv       count                     :         506 [0,0]          506 [0,0]           506
Allgather     count                     :         485 [0,0]          485 [0,0]           485
```

```
Allgatherv    count                    :          506 [0,0]          506 [0,0]          506
Scatter       count                    :          485 [0,0]          485 [0,0]          485
Scatterv      count                    :          506 [0,0]          506 [0,0]          506
Alltoall      count                    :          506 [0,0]          506 [0,0]          506
Alltoallv     count                    :          506 [0,0]          506 [0,0]          506
Alltoallw     count                    :            0 [0,0]            0 [0,0]            0
Neighbor Allgather  count              :            0 [0,0]            0 [0,0]            0
Neighbor Allgatherv count              :            0 [0,0]            0 [0,0]            0
Neighbor Alltoall   count              :            0 [0,0]            0 [0,0]            0
Neighbor Alltoallv  count              :            0 [0,0]            0 [0,0]            0
Neighbor Alltoallw  count              :            0 [0,0]            0 [0,0]            0
Number of bytes sent                   :    528482333 [0,2]    880803843 [0,1]    704643071
   Memory Transfer                     :    176160755 [0,3]    704643020 [0,0]    440401904
   DMA Transfer                        :            0 [0,0]    352321510 [0,1]    132120600
Number of bytes recvd                  :    528482265 [0,2]    880804523 [0,0]    704643207
   Memory Transfer                     :    176160755 [0,2]    704643020 [0,1]    440401904
   DMA Transfer                        :            0 [0,3]    352321510 [0,2]    132120600
Put           count                    :            0 [0,0]            0 [0,0]            0
Get           count                    :            0 [0,0]            0 [0,0]            0
Accumulate count                       :            0 [0,0]            0 [0,0]            0
Number of bytes put                    :            0 [0,0]            0 [0,0]            0
Number of bytes got                    :            0 [0,0]            0 [0,0]            0
Number of bytes accum                  :            0 [0,0]            0 [0,0]            0

MPI Communication Information of 8 Scalar processes
------------------------------------------------
                                         Min [U,R]          Max [U,R]       Average

Real MPI Idle Time (sec)               :      4.837 [0,6]        5.367 [0,11]       5.002
User MPI Idle Time (sec)               :      4.825 [0,6]        5.363 [0,11]       4.992
Total real MPI Time (sec)              :     12.336 [0,11]      12.344 [0,5]       12.340
Send          count                    :       1535 [0,4]         1535 [0,4]         1535
   Memory Transfer                     :        506 [0,11]        1518 [0,5]         1328
Recv          count                    :       1518 [0,4]         1518 [0,4]         1518
   Memory Transfer                     :        506 [0,4]         1518 [0,5]         1328
...
Number of bytes accum                  :          0 [0,0]            0 [0,0]            0
```

```
Data of Vector Process [0,0] [node=0,ve=0]:
----------------------------------------

  Real MPI Idle Time (sec)                :            10.071094
  User MPI Idle Time (sec)                :            10.032894
  Total real MPI Time (sec)               :            13.301340
...
```

図 3-6 は環境変数 NMPI_COMMINF_VIEW に MERGED を指定した場合の集約形式の出力例です。

図 3-6 集約形式の MPI 通信情報 (NMPI_COMMINF_VIEW=MERGED)

```
MPI Communication Information of 4 Vector and 8 Scalar processes
------------------------------------------------------------------
                                          Min [U,R]          Max [U,R]       Average

Real MPI Idle Time (sec)            :     4.860 [0,10]      10.193 [0,3]       6.651
User MPI Idle Time (sec)            :     4.853 [0,10]      10.167 [0,3]       6.635
Total real MPI Time (sec)           :    12.327 [0,4]       13.396 [0,3]      12.679
Send       count                    :     1535 [0,2]         2547 [0,1]        1702
   Memory Transfer                  :      506 [0,3]         2024 [0,0]        1309
   DMA Transfer                     :        0 [0,0]         1012 [0,1]         388 (V)
Recv       count                    :     1518 [0,2]         2717 [0,0]        1702
   Memory Transfer                  :      506 [0,2]         2024 [0,1]        1309
   DMA Transfer                     :        0 [0,3]         1012 [0,2]         388 (V)
...
Number of bytes accum               :        0 [0,0]            0 [0,0]           0
```

表 3-15 は，MPI 通信情報の表示項目です。項目「DMA Transfer」はベクトルプロセスのみに対応した項目です。

表 3-15 MPI 通信情報の表示項目

項目	単位	説明
Real MPI Idle Time	秒	メッセージ待ちに費やした経過時間
User MPI Idle Time	秒	メーセージ待ちに費やしたユーザーCPU 時間
Total real MPI Time	秒	MPI 手続の実行に費やした経過時間
Send count	-	1 対 1 送信手続の呼出し回数
Memory Transfer	-	メモリコピーを使用する 1 対 1 送信手続の呼出し回数
DMA Transfer	-	DMA 転送を使用する 1 対 1 送信手続の呼出し回数
Recv count	-	1 対 1 受信手続の呼出し回数

Memory Transfer	-	メモリコピーを使用する 1 対 1 受信手続の呼出し回数
DMA Transfer	-	DMA 転送を使用する 1 対 1 受信手続の呼出し回数
Barrier count	-	手続 MPI_BARRIER および MPI_IBARRIER の呼出し回数
Bcast count	-	手続 MPI_BCAST および MPI_IBCAST の呼出し回数
Reduce count	-	手続 MPI_REDUCE および MPI_IREDUCE の呼出し回数
Allreduce count	-	手続 MPI_ALLREDUCE および MPI_IALLREDUCE の呼出し回数
Scan count	-	手続 MPI_SCAN および MPI_ISCAN の呼出し回数
Exscan count	-	手続 MPI_EXSCAN および MPI_IEXSCAN の呼出し回数
Redscat count	-	手続 MPI_REDUCE_SCATTER および MPI_IREDUCE_SCATTER の呼出し回数
Redscat_block count	-	手続 MPI_REDUCE_SCATTER_BLOCK および MPI_IREDUCE_SCATTER_BLOCK の呼出し回数
Gather count	-	手続 MPI_GATHER および MPI_IGATHER の呼出し回数
Gatherv count	-	手続 MPI_GATHERV および MPI_IGATHERV の呼出し回数
Allgather count	-	手続 MPI_ALLGATHER および MPI_IALLGATHER の呼出し回数
Allgatherv count	-	手続 MPI_ALLGATHERV および MPI_IALLGATHERV の呼出し回数
Scatter count	-	手続 MPI_SCATTER および MPI_ISCATTER の呼出し回数
Scatterv count	-	手続 MPI_SCATTERV および MPI_ISCATTERV の呼出し回数
Alltoall count	-	手続 MPI_ALLTOALL および MPI_IALLTOALL の呼出し回数
Alltoallv count	-	手続 MPI_ALLTOALLV および MPI_IALLTOALLV の呼出し回数
Alltoallw count	-	手続 MPI_ALLTOALLW および MPI_IALLTOALLW の呼出し回数
Neighbor Allgather count	-	手続 MPI_NEIGHBOR_ALLGATHER および MPI_INEIGHBOR_ALLGATHER の呼出し回数
Neighbor Allgatherv count	-	手続 MPI_NEIGHBOR_ALLGATHERV および MPI_INEIGHBOR_ALLGATHERV の呼出し回数
Neighbor Alltoall count	-	手続 MPI_NEIGHBOR_ALLTOALL および MPI_INEIGHBOR_ALLTOALL の呼出し回数
Neighbor Alltoallv count	-	手続 MPI_NEIGHBOR_ALLTOALLV および MPI_INEIGHBOR_ALLTOALLV の呼出し回数
Neighbor Alltoallw count	-	手続 MPI_NEIGHBOR_ALLTOALLW および MPI_INEIGHBOR_ALLTOALLW の呼出し回数
Number of bytes sent	バイト	1 対 1 送信手続により送信したバイト数

Memory Transfer	バイト	メモリコピーを使用する 1 対 1 送信手続により送信したバイト数
DMA Transfer	バイト	DMA 転送を使用する 1 対 1 送信手続により送信したバイト数
Number of bytes recvd	バイト	1 対 1 受信手続により受信したバイト数
Memory Transfer	バイト	メモリコピーを使用する 1 対 1 受信手続により受信したバイト数
DMA Transfer	バイト	DMA 転送を使用する 1 対 1 受信手続により受信したバイト数
Put count	-	手続 MPI_PUT および MPI_RPUT の呼出し回数
Memory Transfer	-	メモリコピーを使用する手続 MPI_PUT および MPI_RPUT の呼出し回数
DMA Transfer	-	DMA 転送を使用する手続 MPI_PUT および MPI_RPUT の呼出し回数
Get count	-	手続 MPI_GET および MPI_RGET の呼出し回数
Memory Transfer	-	メモリコピーを使用する手続 MPI_GET および MPI_RGET の呼出し回数
DMA Transfer	-	DMA 転送を使用する手続 MPI_GET および MPI_RGET の呼出し回数
Accumulate count	-	手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP の呼出し回数
Memory Transfer	-	メモリコピーを使用する手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP の呼出し回数
DMA Transfer	-	DMA 転送を使用する手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP の呼出し回数
Number of bytes put	バイト	手続 MPI_PUT および MPI_RPUT により送信したバイト数
Memory Transfer	バイト	メモリコピーを使用する手続 MPI_PUT および MPI_RPUT により送信したバイト数
DMA Transfer	バイト	DMA 転送を使用する手続 MPI_PUT および MPI_RPUT により送信したバイト数
Number of bytes got	バイト	手続 MPI_GET および MPI_RGET により受信したバイト数
Memory Transfer	バイト	メモリコピーを使用する手続 MPI_GET および MPI_RGET により受信したバイト数

DMA Transfer	バイト	DMA 転送を使用する手続 MPI_GET および MPI_RGET により受信したバイト数
Number of bytes accum	バイト	手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP により積算したバイト数
Memory Transfer	バイト	メモリコピーを使用する手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP により積算したバイト数
DMA Transfer	バイト	DMA 転送を使用する手続 MPI_ACCUMULATE、MPI_RACCUMULATE、MPI_GET_ACCUMULATE、MPI_RGET_ACCUMULATE、MPI_FETCH_AND_OP および MPI_COMPARE_AND_SWAP により積算したバイト数

3.6 FTRACE 機能

FTRACE 機能を使用すると，プログラムの各手続 および 指定した実行範囲における，プロセスごとの詳細な実行性能情報を取得できます。実行性能情報には MPI 通信情報も含まれます。詳細は，「PROGINF/FTRACE ユーザーズガイド」を参照してください。なお、FTRACE 機能は VE 上で動作するプログラムのみをサポートしています。

FTRACE 機能によって表示できる MPI 通信情報は，表 3-16 の通りです。

表 3-16 MPI 通信情報の表示項目

項目	単位	説明
ELAPSE	秒	経過時間
COMM.TIME	秒	MPI 手続の実行に費やした経過時間
COMM.TIME / ELAPSE		各プロセスにおいて，MPI 手続の実行に費やした経過時間が，経過時間に占める割合
IDLE TIME	秒	メッセージ待ちに費やした経過時間
IDLE TIME / ELAPSE		各プロセスにおいて，メッセージ待ちに費やした経過時間が，経過時間に占める割合
AVER.LEN	バイト	MPI 手続あたりの平均通信量
COUNT		MPI 手続による転送回数
TOTAL LEN	バイト	MPI 手続による総通信量

FTRACE 機能の利用手順は，次の通りです。

1. オプション -ftrace を指定してプログラムをコンパイル・リンクします。

```
(例)  $ mpinc++ -ftrace mpi.c
(例)  $ mpinfort -ftrace mpifort.f90
```

2. MPI プログラムを実行すると，解析情報ファイルが実行ディレクトリに生成されます。解析情報ファイルのファイル名は，ftrace.out.*uuu.rrr*(*uuu* および *rrr* は，それぞれ環境変数 MPIUNIVERSE および MPIRANK の値)です。

3. ftrace コマンドを使用して，解析情報ファイルを読み込み，実行性能情報を標準出力に表示 します。

```
(例)  $ ftrace -all -f ftrace.out.0.0  ftrace.out.0.1
(例)  $ ftrace -f ftrace.out.*
```

次の図は，FTRACE 機能による実行性能情報の表示例です。

図 3-7 FTRACE 機能による MPI プログラムの実行性能情報

```
*----------------------*
  FTRACE ANALYSIS LIST
*----------------------*

Execution Date : Sat Feb 17 12:44:49 2018 JST
Total CPU Time : 0:03'24"569 (204.569 sec.)


FREQUENCY  EXCLUSIVE       AVER.TIME     MOPS   MFLOPS  V.OP  AVER.    VECTOR L1CACHE .... PROC.NAME
           TIME[sec](  %  )    [msec]                   RATIO V.LEN      TIME    MISS

     1012    49.093( 24.0)    48.511  23317.2  14001.4  96.97  83.2    42.132   5.511      funcA
   160640    37.475( 18.3)     0.233  17874.6   9985.9  95.22  52.2    34.223   1.973      funcB
   160640    30.515( 14.9)     0.190  22141.8  12263.7  95.50  52.8    29.272   0.191      funcC
   160640    23.434( 11.5)     0.146  44919.9  22923.2  97.75  98.5    21.869   0.741      funcD
   160640    22.462( 11.0)     0.140  42924.5  21989.6  97.73  99.4    20.951   1.212      funcE
```

```
 53562928    15.371(  7.5)     0.000   1819.0    742.2   0.00   0.0     0.000   1.253       funcG
        8    14.266(  7.0)  1783.201   1077.3     55.7   0.00   0.0     0.000   4.480       funcH
   642560     5.641(  2.8)     0.009    487.7      0.2  46.45  35.1     1.833   1.609       funcF
     2032     2.477(  1.2)     1.219    667.1      0.0  89.97  28.5     2.218   0.041       funcI
        8     1.971(  1.0)   246.398  21586.7   7823.4  96.21  79.6     1.650   0.271       funcJ
------------------------------------------------------------------------------------ .... ----------
 54851346   204.569(100.0)     0.004  22508.5  12210.7  95.64  76.5   154.524  17.740       total


   ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
   TIME[sec]        [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]                [byte]

      12.444        0.000                   0.000                 0.0          0        0.0  funcA
       9.420        0.000                   0.000                 0.0          0        0.0  funcB
       7.946        0.000                   0.000                 0.0          0        0.0  funcG
       7.688        0.000                   0.000                 0.0          0        0.0  funcC
       7.372        0.000                   0.000                 0.0          0        0.0  funcH
       5.897        0.000                   0.000                 0.0          0        0.0  funcD
       5.653        0.000                   0.000                 0.0          0        0.0  funcE
       1.699        1.475                   0.756                 3.1K    642560        1.9G funcF
       1.073        1.054                   0.987                 1.0M      4064        4.0G funcI
       0.704        0.045                   0.045                80.0         4      320.0  funcK
-------------------------------------------------------------------------------------------------------


FREQUENCY  EXCLUSIVE        AVER.TIME     MOPS   MFLOPS  V.OP  AVER.     VECTOR L1CACHE .... PROC.NAME
           TIME[sec](  % )     [msec]                    RATIO V.LEN       TIME     MISS

     1012    49.093( 24.0)     48.511  23317.2  14001.4 96.97  83.2     42.132    5.511       funcA
      253    12.089            47.784  23666.9  14215.9 97.00  83.2     10.431    1.352        0.0
      253    12.442            49.177  23009.2  13811.8 96.93  83.2     10.617    1.406        0.1
      253    12.118            47.899  23607.4  14180.5 97.00  83.2     10.463    1.349        0.2
      253    12.444            49.185  23002.8  13808.2 96.93  83.2     10.622    1.404        0.3
...
------------------------------------------------------------------------------------ .... ----------
 54851346   204.569(100.0)     0.004  22508.5  12210.7  95.64  76.5   154.524  17.740       total


   ELAPSED      COMM.TIME  COMM.TIME   IDLE TIME  IDLE TIME  AVER.LEN      COUNT  TOTAL LEN PROC.NAME
   TIME[sec]        [sec]  / ELAPSED        [sec]  / ELAPSED    [byte]                [byte]
```

12.444	0.000		0.000		0.0	0	0.0	funcA
12.090	0.000	0.000	0.000	0.000	0.0	0	0.0	0.0
12.442	0.000	0.000	0.000	0.000	0.0	0	0.0	0.1
12.119	0.000	0.000	0.000	0.000	0.0	0	0.0	0.2
12.444	0.000	0.000	0.000	0.000	0.0	0	0.0	0.3

3.7 MPI トレース機能

NEC MPI は，MPI プログラムの実行中，MPI 手続の呼出しを検出し，標準出力に出力する機能を用意しています。この機能を利用することにより，

- 呼び出された MPI 手続の種類
- MPI 手続を呼び出した MPI プロセスのランク
- MPI 手続の呼出し および 復帰

を得ることができ，プログラムのデバッグや動作の確認を容易に行うことができます。ただし，MPI 手続の呼出し回数の増加とともに，トレース出力量も大きくなるので注意が必要です。

この機能を利用するためには，MPI コンパイルコマンドに-mpitrace オプションを指定して、MPI プログラムを作成してください。

3.8 トレースバック機能

MPI_Abort を呼び出した際に、トレースバック情報を出力します。以下は出力例です。

例

```
[0,0] MPI Abort by user Aborting program !
[0,0] Obtained 5 stack frames.
[0,0] aborttest() [0x60000003eb18]
[0,0] aborttest() [0x600000006ad0]
[0,0] aborttest() [0x600000005b48]
[0,0] aborttest() [0x600000005cf8]
[0,0] /opt/nec/ve/lib/libc.so.6(__libc_start_main+0x340) [0x600c01c407b0]
[0,0] aborttest() [0x600000005a08]
[0,0] Aborting program!
```

既定値では libc の backtrace_symbols の形式でトレースバック情報を出力しますが、ファイル名や行番号情報も出力したい場合には、コンパイル・リンク時に-traceback=verbose を指定し、プログラム実行時に環

境変数 NMPI_VE_TRACEBACK に"ON"を指定します。この環境変数による形式変更は VE の MPI プログラムにのみ有効です。以下は出力例です。

例

```
[0,0] MPI Abort by user Aborting program !
[0,0] [ 0] 0x600000001718 abort_test       abort.c:33
[0,0] [ 1] 0x600000001600 out              out.c:9
[0,0] [ 2] 0x600000001460 hey              hey.c:9
[0,0] [ 3] 0x600000001530 main             main.c:13
[0,0] [ 4] 0x600c01c407a8 ?                ?:?
[0,0] [ 5] 0x600000000b00 ?                ?:?
[0,0] Aborting program!
```

また NMPI_TRACEBACK_DEPTH によってトレースバック情報 の上限の値を設定します。値を指定しない場合は、50 が設定されます。

3.9 MPI 集団手続デバッグ支援機能

MPI 集団手続デバッグ支援機能は，MPI 集団手続引用のプロセス間にまたがる誤りを検出し，誤りの詳細情報を標準エラー出力に表示することで，MPI 集団手続引用に関するデバッグ作業を支援します。

MPI 集団手続引用の誤りには，同一コミュニケーターに属するプロセスが，異なる MPI 集団手続を同時に呼び出した場合や，プロセス間で同一の値を指定しなければならない引数に対して，異なる値を指定した場合などがあります。

本機能を利用する場合，次のように，MPI プログラムの作成時に -mpiverify オプションを指定します。

```
%> mpinfort -mpiverify f.f90
```

誤りが検出された場合，次のような情報を標準エラー出力に出力します。

- MPI 集団手続名
- その手続を呼び出したプロセスのランク
- 誤りの内容

次の例は，手続 MPI_BCAST において，ランク 3 のプロセスが引数 root に 2 を指定しているのに対して，ランク 0 のプロセスは引数 root に 1 を指定している場合の表示です。

```
VERIFY MPI_Bcast(3): root 2 inconsistent with root 1 of 0
```

検出される内容は, 環境変数 **NMPI_VERIFY** を表 3-17 のように実行時に指定することにより選択できます。

表 3-17 環境変数 NMPI_VERIFY の値

NMPI_VERIFY	検出内容
0	MPI 集団手続引用の誤りを検出しません。
3	(既定値) 手続 MPI_WIN_FENCE の引数 assert 以外の誤りを検出します。
4	既定値の内容に加えて，手続 MPI_WIN_FENCE の引数 assert の誤りを検出します。

本機能で検出可能な項目は，表 3-18 の通りです。

表 3-18 MPI 集団手続デバッグ支援機能の検出項目一覧

手続	項目	条件
全集団手続	集団手続呼出し順序	同一コミュニケーターに属するか，または 同一のウィンドウ もしくは ファイルハンドルに対応するプロセスが, 同時に異なる MPI 集団手続を呼び出した場合。
引数 root をもつ集団手続	引数 root	引数 root の値がプロセス間で一致しない場合。
集団通信手続	メッセージ長 (各要素の寸法 * 転送要素の個数)	送信メッセージ長と，対応する受信メッセージ長が一致しない場合。
集計演算を行う集団通信手続	引数 op	引数 op (集計演算子) がプロセス間で一致しない場合。
トポロジー集団手続	グラフ情報 および 次元情報	引数で指定されたグラフ情報 または 次元情報がプロセス間で整合しない場合。
MPI_COMM_CREATE	引数 group	引数 group で指定されたグループがプロセス間で一致しない場合。
MPI_INTERCOMM_CREATE	引数 local_leader および tag	引数 local_leader の値が，ローカルコミュニケーター内のプロセス間で一致しない場合 または 引数 tag の値が，引数 local_leader もしくは

		remote_leader に対応するプロセス間で一致しない場合。
MPI_INTERCOMM_MERGE	引数 high	引数 high の値が，プロセス間で一致しない場合。
MPI_FILE_SET_VIEW	引数 etype および datarep	引数 etype で指定されたデータ型 または 引数 datarep で指定されたデータ表現が，プロセス間で一致しない場合。
MPI_WIN_FENCE	引数 assert	引数 assert の値がプロセス間で整合しない場合。

- 本機能は，検出オーバーヘッドによる性能低下を伴う可能性があるので，集団手続引用の正当性を確認した後は，-mpiverify オプションなしで MPI プログラムを再作成してください。

3.10 MPI プログラム終了状態

NEC MPI は，MPI プログラムの終了状態(正常終了 または 異常終了)を判定するため，各 MPI プロセスを監視しています。全ての MPI プロセスが終了状態として 0 を返す場合，その MPI プログラムは正常終了したものとみなされ，0 以外の終了状態を返す MPI プロセスが 1 つでも存在する場合，その MPI プログラムは異常終了したものとみなされます。

したがって，MPI プログラムの終了状態が正しく検出されるためには，プログラムの終了状態を次のように指定する必要があります。

- 正常終了時，MPI プロセスの終了状態として 0 を指定する。たとえば C プログラムの場合，main 関数の返却値として 0 を指定する。

- 異常終了時，MPI プロセスの終了状態として 0 以外の値を指定する。たとえば C プログラムの場合は，main 関数の返却値，exit システムコールに指定する終了状態，または 手続 MPI_ABORT のエラーコードとして 0 以外の値を指定する。

- シェルスクリプトなどを利用して間接的に MPI プログラムを起動する場合，その終了状態として，MPI プログラムの終了状態を継承する。たとえば，次のようなシェルスクリプトを利用する。

```
#!/bin/sh
{MPIexec}              # MPI 実行指定(MPI プロセスの起動：3.2.1 項参照)
RC=$?                  # MPI プロセス終了状態の保存
command                # MPI プログラム以外のプログラム または コマンド実行
exit $RC               # MPI プロセス終了状態の指定
```

3.11　その他の注意事項

本節では，その他の NEC MPI 使用上の注意事項について説明します。

(1) MPI の実行に使用する MPI プログラムや共有ライブラリには，同一バージョンの MPI ライブラリをリンクする必要があります。MPI ライブラリのバージョンは以下(a)(b)の方法で確認できます。

(a) nreadelf コマンドにより、実行可能ファイルに動的リンクされる MPI ライブラリの既定のディレクトリパス(RUNPATH)が取得できます。このパスのうち下線部分が MPI ライブラリのバージョンとなります。なお、オプション-shared-mpi を指定して MPI プログラムを作成した場合は、実行前に読み込んだセットアップスクリプトに対応するバージョンの MPI ライブラリが優先して動的リンクされます。

(例)
```
$ /opt/nec/ve/bin/nreadelf -W -d a.out | grep RUNPATH
0x000000000000001d (RUNPATH) Library runpath: [/opt/nec/ve/mpi/2.2.0/lib64/ve:...]
```

(b) オプション-shared-mpi を指定せずに MPI プログラムを作成した場合、strings コマンドを利用して静的リンクされた MPI ライブラリのバージョンを取得できます。

(例)
```
$ /usr/bin/strings a.out | /bin/grep "library Version"
NEC MPI: library Version 2.2.0 (17. April 2019): Copyright (c) NEC Corporation 2018-2019
```

(2) Fortran で記述された MPI プログラムのコンパイル時に，Fortran コンパイラの精度拡張コンパイラオプションを指定する場合，コンパイラオプション-fdefault-integer=8 および -fdefault-real=8 をともに指定する必要があり，かつ それ以外の精度拡張コンパイラオプションを指定してはなりません。

(3) C の関数 および Fortran コンパイラの精度拡張コンパイラオプションを指定してコンパイルした Fortran の手続を 1 つのプログラム中で混在させる場合，NEC MPI を使用することはできません。

(4) NEC MPI は，異常終了した MPI プログラムを適切に制御するため，シグナル SIGINT，SIGTERM，および SIGXCPU のそれぞれにシグナルハンドラーを登録しています。利用者のプログラム自身が，これらのシグナルに利用者定義のハンドラーを登録する場合，そのハンドラー中で，NEC MPI のシグナルハンドラーを呼び出してください。さもなければ,プログラムが正常に終了しない可能性があります。

(5) NEC MPI では，MPI-3 で廃止された C++構文による関数インタフェースの仕様(C++バインディング)を使用することはできません。使用している場合，C 構文による関数インタフェースの仕様(C バインデ

ィング)に修正するか，または ソースプログラム中でインクルードファイル mpi.h をインクルードする前に，マクロ **NMPI_ENABLE_CXX** を定義してください。

(6) MPI コンパイルコマンド中に，コンパイラオプション-x を指定して，ソースプログラムの記述言語を指定することはできません。

(7) 環境変数 **NMPI_PROGINF** を指定して MPI 実行性能情報を取得する場合、MPI プログラムと pthread ライブラリとのリンクにはオプション-pthread を使用する必要があります。オプション-pthread でなく -lpthread を使用した場合、MPI 実行性能情報が正しく表示されない場合があります。

- マルチスレッド実行時に有効となる項目が表示されない
- 各項目にスレッドの性能情報が反映されない
  (User Time、Real Time、Memory Size Used および Non Swappable Memory Size Used を除く)

(8) MPI ライブラリは既定値で MPI のメモリ管理ライブラリ以外の MPI ライブラリが静的リンクされますが、MPI コンパイルコマンドにコンパイラオプション-shared を指定し共有ライブラリを作成する場合、すべての MPI ライブラリが動的リンクされます。また、すべての MPI ライブラリを動的リンクした共有ライブラリを実行可能ファイルにリンクする場合、オプション-shared-mpi を指定し、すべての MPI ライブラリを動的リンクしてください。

(9) MPI プログラムは実行にシステムの共有ライブラリと MPI のメモリ管理の共有ライブラリを必要とします。このため、MPI コンパイルコマンドにコンパイラオプション-static を指定した場合、以下のように動作します。

- VE で動作する MPI プログラムを作成する場合
  ユーザーが明示的に指定したライブラリは静的リンクされますが、コンパイラが追加するライブラリの一部と MPI のメモリ管理ライブラリは動的リンクされます。

- VH またはスカラホストで動作する MPI プログラムを作成する場合
  コンパイラオプション-static は使用できません。使用した場合、リンクエラーまたは実行時エラーが発生する可能性があります。

ライブラリを静的リンクしたい場合、コンパイラオプション-static の代わりに、リンカオプション-Bstatic およびコンパイラが提供するコンパイラライブラリを静的リンクするオプションが利用できます。リンカオプション-Bstatic を使用する場合、静的リンクしたいライブラリを-Wl,-Bstatic と-Wl,-Bdynamic で囲ってください。囲われた範囲のライブラリが静的リンクされます。以下は libww と libxx を静的リンクする例です。

```
mpincc a.c -lvv -Wl,-Bstatic -lww -lxx -Wl,-Bdynamic -lyy
```

コンパイラライブラリを静的リンクするコンパイラオプションについては各コンパイラのマニュアルをご参照ください。

(10) MPI プログラムの実行ディレクトリには書き込み権限が必要です。権限が足りない場合、下記警告メッセージが出力され、MPI 通信性能が低下する場合があります。

```
mkstemp: Permision denied
```

(11) MPI 実行性能情報機能の利用時、性能情報採取のために MPI_Init と MPI_Finalize の実行時に VE のスレッドにシグナル SIGUSR1 が発行されます。このため、デバッガ利用時、デバッガが SIGUSR1 を捕捉し MPI 実行を停止する場合があります。また、VE の MPI プログラムが非ブロッキング MPI-IO 手続きを使用し、かつ、その手続きが使用する非同期 I/O の方式として POSIX AIO を選択している場合、非同期 I/O のために生成される POSIX AIO のワーカースレッドが SIGUSR1 に応答せず実行が停止する場合があります。これらの場合、環境変数 VE_PROGINF_USE_SIGNAL を NO に設定することでシグナルの発行が抑止できます。ただし、シグナルの発行を抑止した場合は、代わりにコンパイラの自動並列および OpenMP によるワーカースレッドのみを対象にスレッドを停止して性能情報を採取するため、OpenMP やコンパイラ自動並列以外のスレッドの性能値が性能項目に反映されません。(User Time、Real Time、Memory Size Used および Non Swappable Memory Size Used を除く)

(12) MPI は通信の最適化のため、ホストの HugePage を使用します。ホストの HugePage が確保できない場合、下記警告メッセージが出力され、MPI 通信性能が低下する場合があります。ホストの HugePage の設定にはシステムの管理者権限が必要です。本メッセージが出力された場合、「SX-Aurora TSUBASA インストレーションガイド」を参照するか、システム管理者にお問い合わせください。

```
mpid(0): Allocate_system_v_shared_memory: key = 0x420bf67e, len = 16777216 shmget
allocation: Cannot allocate memory
```

(13) MPI は IB 通信や HugePage を利用するために memlock リソースリミットが unlimited に設定されている必要があります。本設定は自動的に行われますので、ulimit コマンドなどで unlimited から変更しないようにしてください。memlock リソースリミットが unlimited でない場合は、以下のようなメッセージが出力され、実行に失敗したり、MPI 通信性能が低下したりする場合があります。

```
libibverbs: Warning: RLIMIT_MEMLOCK is 0 bytes.
    This will severely limit memory registrations.
```

```
[0] MPID_OFED_Open_hca: open device failed ib_dev 0x60100002ead0 name mlx5_0
[0] Error in Infiniband/OFED initialization. Execution aborts
```

```
mpid(0): Allocate_system_v_shared_memory: key = 0xd34d79c0, len = 16777216
shmget allocation: Operation not permitted
```

また、memlock リソースが unlimited に設定されている場合であっても、システムログに以下のメッセージが出力される場合がありますが、MPI 実行には問題ありません。

```
kernel: mpid (20934): Using mlock ulimits for SHM_HUGETLB is deprecated
```

(14) アプリケーション実行においてプロセスが異常終了した場合、ユニバース番号およびランク番号と共に、異常終了の原因に関連する情報（エラー内容や終了状態など）が出力されます。しかしながら、異常終了のタイミングによっては以下の様なメッセージが多数出力され、異常終了の原因に関連する情報が参照しにくい場合があります。

```
[3] mpisx_sendx: left (abnormally) (rc=-1), sock = -1 len 0 (12)
Error in send () called by mpisx_sendx: Bad file descriptor
```

この場合、上記メッセージを除外することで、本情報を参照しやすくなる場合があります。以下にコマンド例を示します。

```
$ grep -v mpisx_sendx outputfile
```

(15) 複数の論理ノードを要求する NQSV リクエストにて、モデル A412-8 または B401-8 上で MPI プログラムを実行する場合、NEC MPI が適切な HCA を選択するために NQSV のオプション--use-hca に利用可能な HCA 数を指定する必要があります。そうしない場合、MPI 実行の終了時に以下のエラーが発生する場合があります。

```
mpid(0): accept_process_answer: Application 0: No valid IB device
found which is requested by environment variable
NMPI_IP_USAGE=OFF. Specify NMPI_IP_USAGE=FALLBACK if TCP/IP should be used in this
case !
```

(16) VEO 機能を併用し、VE メモリを MPI 手続きの引数として直接指定する場合、VE メモリは veo_alloc_hmem 手続きにより確保した領域に限られます。

(17) VE 上で動作する MPI プロセスから以下のシステムコールまたはライブラリ関数を実行することはできません。

fork, system, popen, posix_spawn

これらのシステムコールまたはライブラリ関数を実行した場合、プログラムの実行ストールあるいは結果不正などの問題が発生することがあります。

(18) MPI プログラムでは malloc_info 関数を実行することはできません。malloc_info 関数を実行した場合、不正な値が返却されることがあります。また、mallopt 関数に指定した M_PERFTURB、M_ARENA_MAX、M_ARENA_TEST 引数、および、MPI プログラムに指定した MALLOC_PERFTURB_、MALLOC_ARENA_MAX、MALLOC_ARENA_TEST 環境変数(VE プログラムの場合は、環境変数の前に VE_が付きます)は無視されます。

(19) シェルスクリプト内で、セットアップスクリプト necmpivars.sh、necmpivars.csh、necmpivars-runtime.sh および necmpivars-runtime.csh を明示的な引数無しで読み込んだ場合、シェルスクリプトに指定された引数がセットアップスクリプトに渡されることがあります。また、セットアップスクリプトに不正な引数が渡された場合、以下のメッセージが出力され、LD_LIBRARY_PATH は更新されません。

```
necmpivars.sh: Warning: invalid argument. LD_LIBRARY_PATH is not updated.
```

(20) AVEO UserDMA 機能を使用する場合、veo_free_hmem 手続きにより VE メモリを解放した場合や、veo_proc_destroy 手続きにより VEO プロセスを終了した場合でも、使用可能な VE メモリが増加しない場合があります。

(21) AVEO UserDMA 機能を使用する場合、veo_proc_destroy 手続きの呼び出し後に veo_proc_create などの手続きにより VEO プロセスを作成することはできません。異常終了、結果不正などの問題が発生する場合があります。

第 4 章　MPI 手続引用仕様

本章では，全ての MPI 手続について，C 言語 および Fortran 言語それぞれの構文による手続インタフェースの仕様(引用仕様)を説明します。

C 言語からの引用仕様は，関数原型により，各関数の型，関数名，および 引数について記述します。C 言語における MPI 関数の名前は，接頭辞 MPI_とその直後に続く 1 文字は大文字とし，それ以降の文字は，全て小文字にしなければなりません。MPI 関数の戻り符号は int 型の関数値で返却されます。

Fortran 言語からの引用仕様は，手続名，引数，および 引数の型について記述します。MPI 手続の戻り符号は，整数型の最後の引数 IERROR に返却されます。

MPI 手続の戻り符号として返却される値を，MPI 予約名として定義されているエラーコードと比較することで，MPI 手続が正常に実行されたかどうかを判定できます。エラーコードの一覧は A.3 エラーコードを参照ください。

引数の引用 または 更新の区別は，文字列 IN，OUT，　または INOUT を用いて示します。それぞれは次の意味をもちます。

記号	意味
IN	引数は MPI 手続中で引用されるが，更新は行われません。
OUT	引数は MPI 手続中で更新される場合があります
INOUT	引数は MPI 手続中で引用 および 更新されます。 同じ手続の呼出しでも，あるプロセスでは引用だけが行われ，別のプロセスでは更新だけが行われる場合もあります。

4.1　1 対 1 通信

MPI_BSEND	MPI_BSEND_INIT	MPI_BUFFER_ATTACH
MPI_BUFFER_DETACH	MPI_CANCEL	MPI_GET_COUNT
MPI_IBSEND	MPI_IMPROBE	MPI_IMRECV
MPI_IPROBE	MPI_IRECV	MPI_IRSEND
MPI_ISEND	MPI_ISSEND	MPI_MPROBE
MPI_MRECV	MPI_PROBE	MPI_RECV
MPI_RECV_INIT	MPI_REQUEST_FREE	MPI_REQUEST_GET_STATUS
MPI_RSEND	MPI_RSEND_INIT	MPI_SEND
MPI_SEND_INIT	MPI_SENDRECV	MPI_SENDRECV_REPLACE
MPI_SSEND	MPI_SSEND_INIT	MPI_START
MPI_STARTALL	MPI_TEST	MPI_TEST_CANCELLED
MPI_TESTALL	MPI_TESTANY	MPI_TESTSOME
MPI_WAIT	MPI_WAITALL	MPI_WAITANY
MPI_WAITSOME		

バッファモードによるブロッキング送信

基本構文

MPI_BSEND (data, count, datatype, dest, tag, comm)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Bsend (void* data, int count, MPI_Datatype, int dest, int tag, MPI_Comm comm)

Fortran バインディング

call MPI_BSEND (DATA, DATATYPE, DEST, TAG, COMM, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, IERROR

バッファモード送信の通信識別子の生成

基本構文

MPI_BSEND_INIT (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Bsend (void* data, int count, MPI_Datatype, int dest, int tag, MPI_Comm comm,
MPI_Request *request)

Fortran バインディング

call MPI_BSEND (DATA, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

補足説明

実際の送信処理は，生成した通信識別子を MPI_START または MPI_STARTALL 手続の引数に指定して呼び出すことで開始します。
送信処理は非ブロッキング通信と同様に行なわれ， MPI_WAIT または MPI_TEST などの手続によって送信の完了を確認します。
通信の完了後，通信識別子の解放は MPI_REQUEST_FREE 手続で行います。

MPI_Buffer_attach(C) MPI_BUFFER_ATTACH(Fortran)

バッファモードの送信で使用する通信用バッファの設定

基本構文

MPI_BUFFER_ATTACH (buffer, size)

引数	値	説明	IN/OUT
buffer	任意	通信用バッファの先頭アドレス	IN
size	整数	通信用バッファの大きさ(バイト単位)	IN

C バインディング

int MPI_Buffer_attach (void* buffer, int size)

Fortran バインディング

call MPI_BUFFER_ATTACH (BUFFER, SIZE, IERROR)

任意の型	BUFFER(*)
整数型	SIZE, IERROR

MPI_Buffer_detach(C) MPI_BUFFER_DETACH(Fortran)

バッファモードの送信で使用する通信用バッファの解除

基本構文

MPI_BUFFER_DETACH (buffer_addr, size)

引数	値	説明	IN/OUT
buffer_addr	任意	通信用バッファの先頭アドレス	OUT
size	整数	通信用バッファの大きさ(バイト単位)	OUT

C バインディング

int MPI_Buffer_detach (void* buffer_addr, int* size)

Fortran バインディング

call MPI_BUFFER_DETACH (BUFFER_ADDR, SIZE, IERROR)

任意の型	BUFFER_ADDR(*)
整数型	SIZE, IERROR

保留状態の非ブロッキング通信のキャンセル

基本構文

MPI_CANCEL(request)

引数	値	説明	IN/OUT
request	handle	通信識別子	IN

C バインディング

int MPI_Cancel (MPI_Request* request)

Fortran バインディング

call MPI_CANCEL (REQUEST, IERROR)

整数型 REQUEST, IERROR

補足説明

既に発行した非ブロッキング通信に対応する受信 または 送信手続が まだ発行されないために保留状態となっている非ブロッキング通信をキャンセルします。

受信した通信状態からデータ型の要素の個数の取得

基本構文

MPI_GET_COUNT(status, datatype, count)

引数	値	説明	IN/OUT
status	status	受信した通信状態	IN
datatype	handle	受信で使用したデータ型	IN
count	整数	受信したデータ型の要素の個数	OUT

C バインディング

int MPI_Get_count (MPI_Status *status, MPI_Datatype datatype, int *count)

Fortran バインディング

call MPI_GET_COUNT (STATUS, DATATYPE, COUNT, IERROR)

整数型 STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR

補足説明

MPI_GET_COUNT と MPI_GET_ELEMENTS との違いは， MPI_GET_ELEMENTS は， 引数 datatype に指定されたデータ型に依存することなく INTEGER や float といった基本データ型を単位に 受信メッセージのデータの要素の個数をカウントするのに対して， MPI_GET_COUNT は，基本データ型だけでなく 利用者定義のデータ型をも含め 引数 datatype に指定されたデータ型を単位に 要素の個数をカウントすることです。
MPI_GET_COUNT では，受信したメッセージデータが 引数 datatype で指定されたデータ型によって 正確にカウントできない場合， すなわち，受信データの大きさが datatype に指定されたデータ型の大きさの倍数でなければ， 引数 count には予約名 MPI_UNDEFINED で定義された値が返却されます。

バッファモードによる非ブロッキング送信

基本構文

MPI_IBSEND (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Ibsend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
                comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_IBSEND (DATA, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

通信メッセージのチェックとマーク付け

基本構文

MPI_IMPROBE (source, tag, comm, flag, status)

引数	値	説明	IN/OUT
source	整数	通信相手のランク または MPI_ANY_SOURCE	IN
tag	整数	タグの値 または MPI_ANY_TAG	IN
comm	handle	コミュニケーター	IN
flag	論理	チェック結果フラグ	OUT
message	handle	メッセージ識別子	OUT
status	status	通信状態	OUT

C バインディング

```
int MPI_Improbe (int source, int tag, MPI_Comm comm, int *flag, MPI_Message *message,
                 MPI_Status *status)
```

Fortran バインディング

```
call MPI_IMPROBE (SOURCE, TAG, COMM, FLAG, MESSAGE, STATUS, IERROR)
```

論理型	FLAG
整数型	SOURCE, TAG, COMM, MESSAGE, STATUS(MPI_STATUS_SIZE), IERROR

マーク付き通信メッセージの非ブロッキング受信

基本構文

MPI_IMRECV (data, count, datatype, message, request)

引数	値	説明	IN/OUT
data	任意	受信バッファの先頭アドレス	OUT
count	整数	受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	受信バッファの要素の型	IN
message	handle	メッセージ識別子	INOUT
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Imrecv (void* data, int count, MPI_Datatype datatype, MPI_Message *message,
                MPI_Request *request)
```

Fortran バインディング

```
call MPI_IMRECV (DATA, COUNT, DATATYPE, MESSAGE, REQUEST, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, MESSAGE, REQUEST, IERROR

送られてきた通信メッセージのチェック

基本構文

MPI_IPROBE (source, tag, comm, flag, status)

引数	値	説明	IN/OUT
source	整数	通信相手のランク または MPI_ANY_SOURCE	IN
tag	整数	タグの値 または MPI_ANY_TAG	IN
comm	handle	コミュニケーター	IN
flag	論理	チェック結果フラグ	OUT
status	status	通信状態	OUT

C バインディング

int MPI_Iprobe (int source, int tag, MPI_Comm comm, int *flag, MPI_Status *status)

Fortran バインディング

call MPI_IPROBE (SOURCE, TAG, COMM, STATUS, FLAG, IERROR)

論理型	FLAG
整数型	SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

補足説明

source,tag および comm に相当する通信メッセージが届いていれば flag に TRUE を返し，届いていなければ FLASE を返します。
通信メッセージの到着状態に関係なくこの手続の呼出しは ブロッキングされません。
通信メッセージが届いていてもデータの受け取り操作は行いません。

非ブロッキング受信

基本構文

MPI_IRECV (data, count, datatype, source, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	受信バッファの先頭アドレス	OUT
count	整数	受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	受信バッファの要素の型	IN
source	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Irecv (void* data, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IRECV (DATA, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

レディモードによる非ブロッキング送信

基本構文

MPI_IRSEND (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Irsend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
                comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_IRSEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

標準モードによる非ブロッキング送信

基本構文

MPI_ISEND (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Isend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
        comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_ISEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

同期モードによる非ブロッキング送信

基本構文

MPI_ISSEND (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Issend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
                comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_ISSEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

通信メッセージのチェックとマーク付け

基本構文

MPI_MPROBE (source, tag, comm, message, status)

引数	値	説明	IN/OUT
source	整数	通信相手のランク または MPI_ANY_SOURCE	IN
tag	整数	タグの値 または MPI_ANY_TAG	IN
comm	handle	コミュニケーター	IN
message	handle	メッセージ識別子	OUT
status	status	通信状態	OUT

C バインディング

```
int MPI_Mprobe (int source, int tag, MPI_Comm comm, MPI_Message *message, MPI_Status
                *status)
```

Fortran バインディング

```
call MPI_MPROBE (SOURCE, TAG, COMM, MESSAGE, STATUS, IERROR)

    整数型 SOURCE, TAG, COMM, MESSAGE, STATUS(MPI_STATUS_SIZE), IERROR
```

マーク付き通信メッセージの受信

基本構文

MPI_MRECV (data, count, datatype, message, status)

引数	値	説明	IN/OUT
data	任意	受信バッファの先頭アドレス	OUT
count	整数	受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	受信バッファの要素の型	IN
message	handle	メッセージ識別子	INOUT
status	status	通信状態	OUT

C バインディング

```
int MPI_Mrecv (void* data, int count, MPI_Datatype datatype, MPI_Message *message,
               MPI_Status *status)
```

Fortran バインディング

```
call MPI_MRECV (DATA, COUNT, DATATYPE, MESSAGE, STATUS, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, MESSAGE, STATUS(MPI_STATUS_SIZE), IERROR

送られてきたメッセージのチェック

基本構文

MPI_PROBE (source, tag, comm, status)

引数	値	説明	IN/OUT
source	整数	通信相手のランク または MPI_ANY_SOURCE	IN
tag	整数	タグの値 または MPI_ANY_TAG	IN
comm	handle	コミュニケーター	IN
status	status	通信状態	OUT

C バインディング

int MPI_Probe (int source, int tag, MPI_Comm comm, MPI_Status *status)

Fortran バインディング

call MPI_PROBE (SOURCE, TAG, COMM, STATUS, IERROR)

整数型 SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

補足説明

source,tag および comm に相当する通信メッセージが届くまで この手続呼出しはブロッキングされます。 メッセージが届いていても受け取り操作は行いません。

ブロッキング受信

基本構文

MPI_RECV (data, count, datatype, source, tag, comm, status)

引数	値	説明	IN/OUT
data	任意	受信バッファの先頭アドレス	OUT
count	整数	受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	受信バッファの要素の型	IN
source	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
status	status	通信状態	OUT

C バインディング

int MPI_Recv (void* data, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Fortran バインディング

call MPI_RECV (DATA, COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, SOURCE, TAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

メッセージ受信の通信識別子の生成

基本構文

MPI_RECV_INIT (data, count, datatype, source, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	受信バッファの先頭アドレス	OUT
count	整数	受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	受信バッファの要素の型	IN
source	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Recv_init (void* data, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_RECV_INIT (DATA, COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, SOURCE, TAG, COMM, REQUEST, IERROR

補足説明

実際の受信処理は生成した通信識別子を MPI_START または MPI_STARTALL 手続の引数に指定して呼び出すことで 開始します。
受信処理は非ブロッキング通信と同様に行なわれ MPI_WAIT または MPI_TEST などの手続によって受信の完了を確認します。
通信の完了後，通信識別子の解放は MPI_REQUEST_FREE 手続で行います。

MPI_Request_free (C) / MPI_REQUEST_FREE (Fortran)

通信識別子の解放

基本構文

MPI_REQUEST_FREE (request)

引数	値	説明	IN/OUT
request	handle	通信識別子	INOUT

C バインディング

int MPI_Request_free (MPI_Request *request)

Fortran バインディング

call MPI_REQUEST_FREE (REQEUST, IERROR)

整数型 REQUEST, IERROR

MPI_Request_get_status (C) / MPI_REQUEST_GET_STATUS (Fortran)

通信識別子の解放を行わない非ブロッキング通信の完了テスト

基本構文

MPI_REQUEST_GET_STATUS (request, flag, status)

引数	値	説明	IN/OUT
request	handle	通信識別子	IN
flag	論理	通信が完了していれば true	OUT
status	status	通信状態	OUT

C バインディング

int MPI_Request_get_status (MPI_Request request, int *flag, MPI_Status *status)

Fortran バインディング

call MPI_REQUEST_GET_STATUS (REQUEST, FLAG, STATUS, IERROR)

INTEGER REQUEST, STATUS(MPI_STATUS_SIZE), IERROR
LOGICAL FLAG

レディモードによるブロッキング送信

基本構文

MPI_RSEND (data, count, datatype, dest, tag, comm)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Rsend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
    comm)
```

Fortran バインディング

```
call MPI_RSEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, IERROR

レディモード送信の通信識別子の生成

基本構文

MPI_RSEND_INIT (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Rsend_init (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_RSEND_INIT (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST IERROR

補足説明

実際の送信処理は生成した通信識別子を MPI_START または MPI_STARTALL 手続の引数に指定して呼び出すことで 開始します。
送信処理は非ブロッキング通信と同様に行なわれ MPI_WAIT または MPI_TEST などの手続によって送信の完了を確認します。
通信の完了後，通信識別子の解放は MPI_REQUEST_FREE 手続で行います。

標準モードによるブロッキング送信

基本構文

MPI_SEND (data, count, datatype, dest, tag, comm)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Send (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
        comm)
```

Fortran バインディング

```
call MPI_SEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, IERROR

標準モード送信の通信識別子の生成

基本構文

MPI_SEND_INIT (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Send_init (void* data, int count, MPI_Datatype datatype, int dest, int tag,
MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_SEND_INIT (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST,
IERROR)

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST IERROR

補足説明

実際の送信処理は生成した通信識別子を MPI_START または MPI_STARTALL 手続の引数に指定して呼び出すことで 開始します。
送信処理は非ブロッキング通信と同様に行なわれ MPI_WAIT または MPI_TEST などの手続によって送信の完了を確認します。
通信の完了後，通信識別子の解放は MPI_REQUEST_FREE 手続で行います。

ブロッキング送受信

基本構文

MPI_SENDRECV (senddata, sendcount, sendtype, dest, sendtag, recvdata, recvcount, recvtype, source, recvtag, comm, status)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数(0 以上の値)	IN
sendtype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク(送信用)	IN
sendtag	整数	メッセージタグ(送信用)	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	受信バッファの要素の個数(0 以上の値)	IN
recvtype	handle	受信バッファの要素の型	IN
source	整数	通信相手のランク(受信用)	IN
recvtag	整数	メッセージタグ(受信用)	IN
comm	handle	コミュニケーター	IN
status	status	通信状態	OUT

C バインディング

```
int MPI_Sendrecv (void* senddata, int sendcount, MPI_Datatype sendtype, int dest, int
                  sendtag, void* recvdata, int recvcount, MPI_Datatype recvtype, int source,
                  int recvtag, MPI_Comm comm, MPI_Status *status)
```

Fortran バインディング

```
call MPI_SENDRECV (SENDDATA, SENDCOUNT, SENDTYPE, DEST, SENDTAG,
                   RECVDATA, RECVCOUNT, RECVTYPE, SOURCE, RECVTAG,
                   COMM, STATUS, IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	SENDCOUNT, SENDTYPE, DEST, SENDTAG, RECVCOUNT, RECVTYPE, SOURCE, RECVTAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

同一送受信バッファによるブロッキング送受信

基本構文

MPI_SENDRECV_REPLACE (data, count, datatype, dest, sendtag, source, recvtag, comm, status)

引数	値	説明	IN/OUT
data	任意	送受信バッファの先頭アドレス	INOUT
count	整数	送受信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送受信バッファの要素の型	IN
dest	整数	通信相手のランク(送信用)	IN
sendtag	整数	メッセージタグ(送信用)	IN
source	整数	通信相手のランク(受信用)	IN
recvtag	整数	メッセージタグ(受信用)	IN
comm	handle	コミュニケーター	IN
status	status	通信状態	OUT

C バインディング

```
int MPI_Sendrecv_replace (void* data, int count, MPI_Datatype datatype, int dest, int sendtag,
                          int source, int recvtag, MPI_Comm comm, MPI_Status *status)
```

Fortran バインディング

```
call MPI_SENDRECV_REPLACE (DATA, COUNT, DATATYPE, DEST, SENDTAG, SOURCE,
                           RECVTAG, COMM, STATUS, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, SENDTAG, SOURCE, RECVTAG, COMM, STATUS(MPI_STATUS_SIZE), IERROR

同期モードによるブロッキング送信

基本構文

MPI_SSEND (data, count, datatype, dest, tag, comm)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Ssend (void* data, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
    comm)
```

Fortran バインディング

```
call MPI_SSEND (DATA, COUNT, DATATYPE, DEST, TAG, COMM, IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, IERROR

同期モード送信の通信識別子の生成

基本構文

MPI_SSEND_INIT (data, count, datatype, dest, tag, comm, request)

引数	値	説明	IN/OUT
data	任意	送信バッファの先頭アドレス	IN
count	整数	送信バッファの要素の個数(0 以上の値)	IN
datatype	handle	送信バッファの要素の型	IN
dest	整数	通信相手のランク	IN
tag	整数	メッセージタグ	IN
comm	handle	コミュニケーター	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Ssend_init (void* data, int count, MPI_Datatype datatype, int dest, int tag,
                    MPI_Comm comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_SSEND_INIT (DATA, COUNT, DATATYPE, DEST, TAG, COMM, REQUEST,
                     IERROR)
```

任意の型	DATA(*)
整数型	COUNT, DATATYPE, DEST, TAG, COMM, REQUEST, IERROR

補足説明

実際の送信処理は生成した通信識別子を MPI_START または MPI_STARTALL 手続の引数に指定して呼び出すことで 開始します。
送信処理は非ブロッキング通信と同様に行なわれ MPI_WAIT または MPI_TEST などの手続によって送信の完了を確認します。
通信の完了後，通信識別子の解放は MPI_REQUEST_FREE 手続で行います。

通信識別子による通信の開始要求

基本構文

MPI_START (request)

引数	値	説明	IN/OUT
request	handle	通信識別子	INOUT

C バインディング

int MPI_Start (MPI_Request *request)

Fortran バインディング

call MPI_START (REQUEST, IERROR)

整数型 REQUEST, IERROR

補足説明

MPI_SEND_INIT,MPI_SSEND_INIT,MPI_BSEND_INIT,MPI_RSEND_INIT および MPI_RECV_INIT 手続によって生成した通信識別子によって 非ブロッキングの通信を開始します。
通信の完了確認は，MPI_WAIT または MPI_TEST などの手続によって行います。

複数の通信識別子による通信の開始要求

基本構文

MPI_STARTALL (count, array_of_requests)

引数	値	説明	IN/OUT
count	整数	通信識別子の数	IN
array_of_requests	handle	通信識別子の配列	INOUT

C バインディング

int MPI_Startall (int count, MPI_Request *array_of_requests)

Fortran バインディング

call MPI_STARTALL (COUNT, ARRAY_OF_REQUESTS, IERROR)

整数型 COUNT, ARRAY_OF_REQUESTS(*), IERROR

補足説明

MPI_SEND_INIT,MPI_SSEND_INIT,MPI_BSEND_INIT,MPI_RSEND_INIT および MPI_RECV_INIT 手続によって生成した通信識別子によって 非ブロッキングの通信を開始します。
通信の完了確認は，MPI_WAIT または MPI_TEST などの手続によって行います。

MPI_Test (C)　　MPI_TEST (Fortran)

非ブロッキング通信の完了テスト

基本構文

MPI_TEST (request, flag, status)

引数	値	説明	IN/OUT
request	handle	通信識別子	INOUT
flag	論理	通信完了フラグ	OUT
status	status	通信状態	OUT

C バインディング

int MPI_Test (MPI_Request *request, int *flag, MPI_Status *status)

Fortran バインディング

call MPI_TEST (REQUEST, FLAG, STATUS, IERROR)

論理型　　FLAG
整数型　　REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

MPI_Test_cancelled (C)　　MPI_TEST_CANCELLED (Fortran)

通信のキャンセルの確認

基本構文

MPI_TEST_CANCELLED (status, flag)

引数	値	説明	IN/OUT
status	status	通信状態	IN
flag	論理	通信完了フラグ	OUT

C バインディング

int MPI_Test_cancelled (MPI_Status *status, int *flag)

Fortran バインディング

call MPI_TEST_CANCELLED (STATUS, FLAG, IERROR)

論理型	FLAG
整数型	STATUS(MPI_STATUS_SIZE), IERROR

MPI_Testall (C) MPI_TESTALL (Fortran)

1 つ以上の非ブロッキング通信全ての完了テスト

基本構文

MPI_TESTALL (count, array_of_requests, flag, array_of_statuses)

引数	値	説明	IN/OUT
count	整数	テストする通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
flag	論理	通信完了フラグ	OUT
array_of_statuses	status	通信状態の配列	OUT

C バインディング

```
int MPI_Testall (int count, MPI_Request *array_of_requests, int *flag, MPI_Status
                 *array_of_statuses)
```

Fortran バインディング

```
call MPI_TESTALL (COUNT, ARRAY_OF_REQUESTS, FLAG, ARRAY_OF_STATUSES,
                  IERROR)
```

論理型	FLAG
整数型	COUNT, ARRAY_OF_REQUESTS(*), ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

1 つ以上の非ブロッキング通信の完了テスト

基本構文

MPI_TESTANY (count, array_of_requests, index, flag, status)

引数	値	説明	IN/OUT
count	整数	テストする通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
index	整数	完了した通信識別子の添字値	OUT
flag	論理	通信完了フラグ	OUT
status	status	通信状態	OUT

C バインディング

```
int MPI_Testany (int count, MPI_Request *array_of_requests, int *index, int *flag,
                 MPI_Status *status)
```

Fortran バインディング

```
call MPI_TESTANY (COUNT, ARRAY_OF_REQUESTS, INDEX, FLAG, STATUS, IERROR)
```

論理型	FLAG
整数型	COUNT, ARRAY_OF_REQUESTS(*), INDEX, STATUS(MPI_STATUS_SIZE), IERROR

MPI_Testsome (C) MPI_TESTSOME (Fortran)

1 つ以上の非ブロッキング通信の少なくとも 1 つの通信完了テスト

基本構文

MPI_TESTSOME (incount, array_of_requests, outcount, array_of_indices, array_of_statuses)

引数	値	説明	IN/OUT
incount	整数	テストする通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
outcount	整数	完了した通信の数	OUT
array_of_indices	整数	完了した通信識別子の添字値の配列	OUT
array_of_statuses	status	通信状態の配列	OUT

C バインディング

```
int MPI_Testsome (int incount, MPI_Request *array_of_requests, int *outcount, int
                  *array_of_indices, MPI_Status *array_of_statuses)
```

Fortran バインディング

```
call MPI_TESTSOME (INCOUNT, ARRAY_OF_REQUESTS, OUTCOUNT,
                   ARRAY_OF_INDICES, ARRAY_OF_STATUSES, IERROR)

                   INCOUNT, ARRAY_OF_REQUESTS(*), OUTCOUNT,
整数型             ARRAY_OF_INDICES(*),
                   ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR
```

MPI_Wait (C) MPI_WAIT (Fortran)

非ブロッキング通信の完了待合せ

基本構文

MPI_WAIT (request, status)

引数	値	説明	IN/OUT
request	handle	通信識別子	INOUT
status	status	通信状態	OUT

C バインディング

int MPI_Wait (MPI_Request *request, MPI_Status *status)

Fortran バインディング

call MPI_WAIT (REQUEST, STATUS, IERROR)

整数型 REQUEST, STATUS(MPI_STATUS_SIZE), IERROR

MPI_Waitall (C) MPI_WAITALL (Fortran)

一つ以上の非ブロッキング通信の全ての完了待合せ

基本構文

MPI_WAITALL (count, array_of_requests, array_of_statuses)

引数	値	説明	IN/OUT
count	整数	待ち合わせる通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
array_of_statuses	status	通信状態の配列	OUT

C バインディング

int MPI_Waitall (int count, MPI_Request *array_of_requests, MPI_Status *array_of_statuses)

Fortran バインディング

call MPI_WAITALL (COUNT, ARRAY_OF_REQUESTS, ARRAY_OF_STATUSES, IERROR)

整数型　　COUNT, ARRAY_OF_REQUESTS(*),
ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR

MPI_Waitany (C)　　MPI_WAITANY (Fortran)

---

1 つ以上の非ブロッキング通信のうち，いずれか 1 つの完了待合せ

基本構文

MPI_WAITANY (count, array_of_requests, index, status)

引数	値	説明	IN/OUT
count	整数	待ち合わせる通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
index	整数	完了した通信識別子の添字値	OUT
status	status	通信状態	OUT

C バインディング

int MPI_Waitany (int count, MPI_Request *array_of_requests, int *index, MPI_Status *status)

Fortran バインディング

call MPI_WAITANY (COUNT, ARRAY_OF_REQUESTS, INDEX, STATUS, IERROR)

整数型　　COUNT, ARRAY_OF_REQUESTS(*), INDEX,
STATUS(MPI_STATUS_SIZE), IERROR

MPI_Waitsome (C) MPI_WAITSOME (Fortran)

1 つ以上の非ブロッキング通信のうち，少なくとも 1 つの完了待合せ

基本構文

MPI_WAITSOME (incount, array_of_requests, outcount, array_of_indices, array_of_statuses)

引数	値	説明	IN/OUT
incount	整数	待ち合わせる通信の数	IN
array_of_requests	handle	通信識別子の配列	INOUT
outcount	整数	完了した通信の数	OUT
array_of_indices	整数	完了した通信識別子の添字値の配列	OUT
array_of_statuses	status	通信状態の配列	OUT

C バインディング

```
int MPI_Waitsome (int incount, MPI_Request *array_of_requests, int *outcount, int
                  *array_of_indices, MPI_Status *array_of_statuses)
```

Fortran バインディング

```
call MPI_WAITSOME (INCOUNT, ARRAY_OF_REQUESTS, OUTCOUNT,
                   ARRAY_OF_INDICES, ARRAY_OF_STATUSES, IERROR)

    整数型          INCOUNT, ARRAY_OF_REQUESTS(*), OUTCOUNT,
                    ARRAY_OF_INDICES(*),
                    ARRAY_OF_STATUSES(MPI_STATUS_SIZE,*), IERROR
```

4.2 データ型

MPI_GET_ADDRESS	MPI_GET_ELEMENTS	MPI_GET_ELEMENTS_X
MPI_PACK	MPI_PACK_EXTERNAL	MPI_PACK_EXTERNAL_SIZE
MPI_PACK_SIZE	MPI_TYPE_COMMIT	MPI_TYPE_CONTIGUOUS
MPI_TYPE_CREATE_DARRAY	MPI_TYPE_CREATE_HINDEXED	MPI_TYPE_CREATE_HINDEXED_BLOCK
MPI_TYPE_CREATE_HVECTOR	MPI_TYPE_CREATE_INDEXED_BLOCK	MPI_TYPE_CREATE_RESIZED
MPI_TYPE_CREATE_STRUCT	MPI_TYPE_CREATE_SUBARRAY	MPI_TYPE_DUP
MPI_TYPE_FREE	MPI_TYPE_GET_CONTENTS	MPI_TYPE_GET_ENVELOPE
MPI_TYPE_GET_EXTENT	MPI_TYPE_GET_EXTENT_X	MPI_TYPE_GET_TRUE_EXTENT
MPI_TYPE_GET_TRUE_EXTENT_X	MPI_TYPE_INDEXED	MPI_TYPE_SIZE
MPI_TYPE_SIZE_X	MPI_TYPE_VECTOR	MPI_UNPACK
MPI_UNPACK_EXTERNAL	MPI_AINT_ADD	MPI_AINT_DIFF

MPI_Get_address (C) MPI_GET_ADDRESS (Fortran)

アドレスを返す。

基本構文

MPI_GET_ADDRESS (location, address)

引数	値	説明	IN/OUT
location	任意	メモリ内の位置	IN
address	整数	location のアドレス(バイト単位)	OUT

C バインディング

```
int MPI_Get_address (void *location, MPI_Aint *address)
```

Fortran バインディング

```
call MPI_GET_ADDRESS (LOCATION, ADDRESS, IERROR)

    任意の型                                LOCATION(*)

    INTEGER                                 IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) ADDRESS
```

受信した通信状態から基本データ型を単位とした要素の個数の取得

基本構文

MPI_GET_ELEMENTS (status, datatype, count)

引数	値	説明	IN/OUT
status	status	受信した通信状態	IN
datatype	handle	受信で使用したデータ型	IN
count	整数	基本データ型の要素の個数	OUT

C バインディング

int MPI_Get_elements (MPI_Status *status, MPI_Datatype datatype, int *count)

Fortran バインディング

call MPI_GET_ELEMENTS (STATUS, DATATYPE, COUNT, IERROR)

整数型 STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR

補足説明

MPI_GET_COUNT の補足説明を参照して下さい。

受信した通信状態から基本データ型を単位とした要素の個数の取得

基本構文

MPI_GET_ELEMENTS_X (status, datatype, count)

引数	値	説明	IN/OUT
status	status	受信した通信状態	IN
datatype	handle	受信で使用したデータ型	IN
count	整数	基本データ型の要素の個数	OUT

C バインディング

int MPI_Get_elements (MPI_Status *status, MPI_Datatype datatype, MPI_Count *count)

Fortran バインディング

call MPI_GET_ELEMENTS (STATUS, DATATYPE, COUNT, IERROR)

INTEGER STATUS(MPI_STATUS_SIZE), DATATYPE, IERROR
INTEGER(KIND=MPI_COUNT_KIND) COUNT

補足説明

MPI_GET_COUNT の補足説明を参照して下さい。

データのパッキング

基本構文

MPI_PACK (indata, incount, datatype, outarea, outsize, position, comm)

引数	値	説明	IN/OUT
indata	任意	パッキングデータ	IN
incount	整数	パッキングデータの要素の個数	IN
datatype	handle	パッキングデータの型	IN
outarea	任意	パッキング出力バッファ	OUT
outsize	整数	パッキング出力バッファの大きさ(バイト単位)	IN
position	整数	パッキング出力バッファ内の位置(バイト単位)	INOUT
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Pack (void* indata, int incount, MPI_Datatype datatype, void* outarea, int outsize,
        int *position, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_PACK (INDATA, INCOUNT, DATATYPE, OUTAREA, OUTSIZE, POSITION,
        COMM, IERROR)
```

任意の型	INDATA(*), OUTAREA(*)
整数型	INCOUNT, DATATYPE, OUTSIZE, POSITION, COMM, IERROR

データ表現指定によるデータのパッキング

基本構文

MPI_PACK_EXTERNAL (datarep, inbuf, incount, datatype, outbuf, outsize, position)

引数	値	説明	IN/OUT
datarep	文字	データ表現(文字列)	IN
inbuf	任意	入力バッファの始点	IN
incount	整数	入力データの要素の個数	IN
datatype	handle	入力データのデータ型	IN
outbuf	任意	出力バッファの始点	OUT
outsize	整数	出力バッファの大きさ(バイト単位)	IN
position	整数	バッファ内の現在位置(バイト単位)	INOUT

C バインディング

```
int MPI_Pack_external (char *datarep, void *inbuf, int incount, MPI_Datatype datatype, void
*outbuf, MPI_Aint outsize, MPI_Aint *position)
```

Fortran バインディング

```
call MPI_PACK_EXTERNAL (DATAREP, INBUF, INCOUNT, DATATYPE, OUTBUF,
OUTSIZE, POSITION, IERROR)

INTEGER                                INCOUNT, DATATYPE, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) OUTSIZE, POSITION
CHARACTER*(*)                          DATAREP
任意                                   INBUF(*), OUTBUF(*)
```

データ表現指定によるパッキングサイズの取得

基本構文

MPI_PACK_EXTERNAL_SIZE (datarep, incount, datatype, size)

引数	値	説明	IN/OUT
datarep	文字	データ表現(文字列)	IN
incount	整数	入力データの要素の個数	IN
datatype	handle	入力データのデータ型	IN
size	整数	出力バッファの大きさ(バイト単位)	OUT

C バインディング

```
int MPI_Pack_external_size (char *datarep, int incount, MPI_Datatype datatype, MPI_Aint
                            *size)
```

Fortran バインディング

```
call MPI_PACK_EXTERNAL_SIZE (DATAREP, INCOUNT, DATATYPE, SIZE, IERROR)

  INTEGER                                INCOUNT, DATATYPE, IERROR
  INTEGER(KIND=MPI_ADDRESS_KIND) SIZE
  CHARACTER*(*)                          DETAREP
```

MPI_Pack_size (C) MPI_PACK_SIZE (Fortran)

パッキングサイズの取得

基本構文

MPI_PACK_SIZE (incount, datatype, comm, size)

引数	値	説明	IN/OUT
incount	整数	パッキングするデータの要素の個数	IN
datatype	handle	パッキングするデータ型	IN
comm	handle	コミュニケーター	IN
size	整数	パッキングサイズ(単位バイト数)	OUT

C バインディング

int MPI_Pack_size (int incount, MPI_Datatype datatype, MPI_Comm comm, int *size)

Fortran バインディング

call MPI_PACK_SIZE (INCOUNT, DATATYPE, COMM, SIZE IERROR)

整数型 INCOUNT, DATATYPE, COMM, SIZE, IERROR

MPI_Type_commit (C) MPI_TYPE_COMMIT (Fortran)

データ型の登録

基本構文

MPI_TYPE_COMMIT (datatype)

引数	値	説明	IN/OUT
datatype	handle	登録するデータ型	INOUT

C バインディング

int MPI_Type_commit (MPI_Datatype *datatype)

Fortran バインディング

call MPI_TYPE_COMMIT (DATATYPE, IERROR)

整数型 DATATYPE, IERROR

データ型の連続的な繰返しによる新たなデータ型の生成

基本構文

MPI_TYPE_CONTIGUOUS (count, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	繰返しの数(0 以上の値)	IN
oldtype	handle	元のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

int MPI_Type_contiguous (int count, MPI_Datatype oldtype, MPI_Datatype *newtype)

Fortran バインディング

call MPI_TYPE_CONTIGUOUS (COUNT, OLDTYPE, NEWTYPE, IERROR)

整数型　　COUNT, OLDTYPE, NEWTYPE, IERROR

分散配列データ型の生成

基本構文

MPI_TYPE_CREATE_DARRAY (size, rank, ndims, array_of_gsizes, array_of_distribs, array_of_dargs, array_of_psizes, order, oldtype, newtype)

引数	値	説明	IN/OUT
size	整数	プロセスグループの大きさ	IN
rank	整数	プロセスグループ中のランク	IN
ndims	整数	配列の次元数	IN
array_of_gsizes	整数	配列の各次元の要素の個数(整数型配列)	IN
array_of_distribs	状態	各次元における分配方法(状態の配列)	IN
array_of_dargs	整数	各次元の分配引数（整数型配列)	IN
array_of_psizes	整数	各次元のプロセスグリッドの大きさ(整数型配列)	IN
order	状態	配列記憶順序フラグ	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_darray (int size, int rank, int ndims, int *array_of_gsizes, int
                            *array_of_distribs,
                            int *array_of_dargs, int *array_of_psizes, int order,
                            MPI_Datatype oldtype, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_DARRAY (SIZE, RANK, NDIMS, ARRAY_OF_GSIZES,
                             ARRAY_OF_DISTRIBS, ARRAY_OF_DARGS,
                             ARRAY_OF_PSIZES, ORDER, OLDTYPE, NEWTYPE,
                             IERROR)

    INTEGER      SIZE, RANK, NDIMS, ARRAY_OF_GSIZES(*),
                 ARRAY_OF_DISTRIBS(*), ARRAY_OF_DARGS(*),
                 ARRAY_OF_PSIZES(*), ORDER, OLDTYPE, NEWTYPE, IERROR
```

個々のブロックの要素の個数 および 変位(バイト単位)の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_HINDEXED (count, array_of_blocklengths, array_of_displacements, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数	IN
array_of_blocklengths	整数	ブロックごとの要素の個数(整数型配列)	IN
array_of_displacements	整数	ブロックごとの変位(バイト単位，整数型配列)	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_hindexed (int count, int *array_of_blocklengths, MPI_Aint
                              *array_of_displacements, MPI_Datatype oldtype,
                              MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_HINDEXED (COUNT, ARRAY_OF_BLOCKLENGTHS,
                               ARRAY_OF_DISPLACEMENTS, OLDTYPE,
                               NEWTYPE, IERROR)

    INTEGER                                 COUNT, ARRAY_OF_BLOCKLENGTHS(*),
                                            OLDTYPE, NEWTYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) ARRAY_OF_DISPLACEMENTS(*)
```

ブロックの要素の個数 および 変位(バイト単位)の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_HINDEXED_BLOCK (count, blocklength, array_of_displacements, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数	IN
blocklength	整数	ブロックごとの要素の個数	IN
array_of_displacements	整数	ブロックごとの変位(バイト単位，整数型配列)	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_hindexed_block (int count, int blocklength, MPI_Aint
                                    *array_of_displacements, MPI_Datatype oldtype,
                                    MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_HINDEXED_BLOCK (COUNT, BLOCKLENGTH,
                                     ARRAY_OF_DISPLACEMENTS, OLDTYPE,
                                     NEWTYPE, IERROR)

    INTEGER                                 COUNT, BLOCKLENGTH, OLDTYPE,
                                            NEWTYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) ARRAY_OF_DISPLACEMENTS(*)
```

ブロックの要素の個数 および ブロック間隔(バイト単位)の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_HVECTOR (count, blocklength, stride, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数	IN
blocklength	整数	ブロックごとの要素の個数	IN
stride	整数	各ブロックの先頭位置の間隔のバイト数	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_hvector (int count, int blocklength, MPI_Aint stride, MPI_Datatype
    oldtype, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_HVECTOR (COUNT, BLOCKLENGTH, STIDE, OLDTYPE,
                              NEWTYPE, IERROR)

  INTEGER                                 COUNT, BLOCKLENGTH, OLDTYPE,
                                          NEWTYPE, IERROR
  INTEGER(KIND=MPI_ADDRESS_KIND) STRIDE
```

MPI_Type_create_indexed_block (C)	MPI_TYPE_CREATE_INDEXED_BLOCK (Fortran)

ブロックの要素の個数 および 個々のブロックの変位(バイト単位)の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_INDEXED_BLOCK (count, blocklength, array_of_displacements, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数	IN
bocklength	整数	ブロックの要素の個数(整数型配列)	IN
array_of_displacements	整数	ブロックごとの変位(バイト単位，整数型配列)	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_indexed_block (int count, int blocklength, int *array_of_displacements,
                                   MPI_Datatype oldtype, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_INDEXED_BLOCK (COUNT, BLOCKLENGTH,
                                    ARRAY_OF_DISPLACEMENTS, OLDTYPE,
                                    NEWTYPE, IERROR)

    INTEGER        COUNT, BLOCKLENGTH, ARRAY_OF_ DISPLACEMENTS(*),
                   OLDTYPE, NEWTYPE, IERROR
```

下限 および 寸法の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_RESIZED (oldtype, lb, extent, newtype)

引数	値	説明	IN/OUT
oldtype	handle	旧データ型	IN
lb	整数	新たなデータ型の下限	IN
extent	整数	新たなデータ型の寸法	IN
newtype	handle	新データ型	OUT

C バインディング

```
int MPI_Type_create_resized (MPI_Datatype oldtype, MPI_Aint lb, MPI_Aint extent,
                             MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_RESIZED (OLDTYPE, LB, EXTENT, NEWTYPE, IERROR)

    INTEGER                                OLDTYPE, NEWTYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) LB, EXTENT
```

MPI_Type_create_struct (C) | MPI_TYPE_CREATE_STRUCT (Fortran)

個々のブロックのデータ型，要素の個数 および 変位(バイト単位)の指定による新たなデータ型の生成

基本構文

MPI_TYPE_CREATE_STRUCT (count, array_of_blocklengths, array_of_displacements, array_of_types,
newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数	IN
array_of_blocklengths	整数	個々のブロック内の要素の個数（整数型配列）	IN
array_of_displacements	整数	個々のブロックの変位（バイト単位，整数型配列）	IN
array_of_types	整数	個々のブロックの要素のデータ型(handle の配列)	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_struct (int count, int array_of_blocklengths[], MPI_Aint
                            array_of_displacements[], MPI_Datatype array_of_types[],
                            MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_STRUCT (COUNT, ARRAY_OF_BLOCKLENGTHS,
                             ARRAY_OF_DISPLACEMENTS, ARRAY_OF_TYPES,
                             NEWTYPE, IERROR)

    INTEGER                                 COUNT, ARRAY_OF_BLOCKLENGTHS(*),
                                            ARRAY_OF_TYPES(*), NEWTYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) ARRAY_OF_DISPLACEMENTS(*)
```

MPI_Type_create_subarray (C)

MPI_TYPE_CREATE_SUBARRAY (Fortran)

部分配列データ型の生成

基本構文

MPI_TYPE_CREATE_SUBARRAY (ndims, array_of_sizes, array_of_subsizes, array_of_starts, order, oldtype, newtype)

引数	値	説明	IN/OUT
ndims	整数	配列の次元数	IN
array_of_sizes	整数	配列の各次元の要素の個数(整数型配列)	IN
array_of_subsizes	整数	部分配列の各次元の要素の個数(整数型配列)	IN
array_of_starts	整数	部分配列の各次元の開始座標(整数型配列)	IN
order	状態	配列記憶順序フラグ	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

```
int MPI_Type_create_subarray (int ndims, int *array_of_sizes, int *array_of_subsizes, int
                              *array_of_starts,
                              int order, MPI_Datatype oldtype, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_SUBARRAY (NDIMS, ARRAY_OF_SIZES, ARRAY_OF_SUBSIZES,
                               ARRAY_OF_STARTS, ORDER, OLDTYPE,
                               NEWTYPE, IERROR)

    INTEGER        NDIMS, ARRAY_OF_SIZES(*), ARRAY_OF_SUBSIZES(*),
                   ARRAY_OF_STARTS(*), ORDER, OLDTYPE, NEWTYPE, IERROR
```

MPI_Type_dup (C) MPI_TYPE_DUP (Fortran)

データ型の複製

基本構文

MPI_TYPE_DUP (type, newtype)

引数	値	説明	IN/OUT
type	handle	データ型	IN
newtype	handle	type の複製	OUT

C バインディング

```
int MPI_Type_dup (MPI_Datatype type, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_DUP (TYPE, NEWTYPE, IERROR)

    INTEGER          TYPE, NEWTYPE, IERROR
```

MPI_Type_free (C) MPI_TYPE_FREE (Fortran)

データ型の解放

基本構文

MPI_TYPE_FREE (datatype)

引数	値	説明	IN/OUT
datatype	handle	解放するデータ型	INOUT

C バインディング

```
int MPI_Type_free (MPI_Datatype *datatype)
```

Fortran バインディング

call MPI_TYPE_FREE (DATATYPE, IERROR)

整数型　　DATATYPE, IERROR

MPI_Type_get_contents (C)　　MPI_TYPE_GET_CONTENTS (Fortran)

データ型生成時のレイアウト情報の取得

基本構文

MPI_TYPE_GET_CONTENTS (datatype, max_integers, max_addresses, max_datatypes, array_of_integers, array_of_addresses, array_of_datatypes)

引数	値	説明	IN/OUT
datatype	handle	アクセスのためのデータ型	IN
max_integers	整数	array_of_integers における要素の個数(非負整数)	IN
max_addresses	整数	array_of_addresses における要素の個数(非負整数)	IN
max_datatypes	整数	array_of_datatypes における要素の個数(非負整数)	IN
array_of_integers	整数	datatype 生成時に使用された整数引数(整数配列)	OUT
array_of_addresses	整数	datatype 生成時に使用されたアドレス引数(整数配列)	OUT
array_of_datatypes	handle	datatype 生成時に使用されたデータ型引数(handle の配列)	OUT

C バインディング

int MPI_Type_get_contents (MPI_Datatype datatype, int max_integers, int max_addresses, int max_datatypes, int *array_of_integers, MPI_Aint *array_of_addresses, MPI_Datatype *array_of_datatypes)

Fortran バインディング

call MPI_TYPE_GET_CONTENTS (DATATYPE, MAX_INTEGERS, MAX_ADDRESSES, MAX_DATATYPES, ARRAY_OF_INTEGERS, ARRAY_OF_ADDRESSES, ARRAY_OF_DATATYPES, IERROR)

INTEGER　　DATATYPE, MAX_INTEGERS, MAX_ADDRESSES, MAX_DATATYPES, ARRAY_OF_INTEGERS(*), ARRAY_OF_DATATYPES(*), IERROR

INTEGER(KIND=MPI_ADDRESS_KIND)　　ARRAY_OF_ADDRESSES(*)

データ型生成時のレイアウト情報の取得

基本構文

MPI_TYPE_GET_ENVELOPE (datatype, num_integers, num_addresses, num_datatypes, combiner)

引数	値	説明	IN/OUT
datatype	handle	アクセスのためのデータ型	IN
num_integers	整数	combiner 呼出し時に使用した入力整数の数(非負整数)	OUT
num_addresses	整数	combiner 呼出し時に使用した入力アドレスの数(非負整数)	OUT
num_datatypes	整数	combiner 呼出し時に使用した入力データ型の数(非負整数)	OUT
combiner	state	結合子	OUT

C バインディング

```
int MPI_Type_get_envelope (MPI_Datatype datatype, int *num_integers, int *num_addresses,
                           int *num_datatypes, int *combiner)
```

Fortran バインディング

```
call MPI_TYPE_GET_ENVELOPE (DATATYPE, NUM_INTEGERS, NUM_ADDRESSES,
                            NUM_DATATYPES, COMBINER, IERROR)

    INTEGER        DATATYPE, NUM_INTEGERS, NUM_ADDRESSES,
                   NUM_DATATYPES, COMBINER, IERROR
```

MPI_Type_get_extent (C) / MPI_TYPE_GET_EXTENT (Fortran)

データ型の下限 および 寸法の返却

基本構文

MPI_TYPE_GET_EXTENT (datatype, lb, extent)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN
lb	整数	下限	OUT
extent	整数	寸法	OUT

C バインディング

```
int MPI_Type_get_extent (MPI_Datatype datatype, MPI_Aint *lb, MPI_Aint *extent)
```

Fortran バインディング

```
call MPI_TYPE_GET_EXTENT (DATATYPE, LB, EXTENT, IERROR)

    INTEGER                                DATATYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) LB, EXTENT
```

MPI_Type_get_extent_x (C) / MPI_TYPE_GET_EXTENT_X (Fortran)

データ型の下限 および 寸法の返却

基本構文

MPI_TYPE_GET_EXTENT_X (datatype, lb, extent)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN
lb	整数	下限	OUT
extent	整数	寸法	OUT

C バインディング

```
int MPI_Type_get_extent_x (MPI_Datatype datatype, MPI_Count *lb, MPI_Count *extent)
```

Fortran バインディング

```
call MPI_TYPE_GET_EXTENT_X (DATATYPE, LB, EXTENT, IERROR)

    INTEGER                                DATATYPE, IERROR
    INTEGER(KIND=MPI_COUNT_KIND) LB, EXTENT
```

MPI_Type_get_true_extent (C)

MPI_TYPE_GET_TRUE_EXTENT (Fortran)

データ型の真の下限 および 寸法の取得

基本構文

MPI_TYPE_GET_TRUE_EXTENT (datatype, true_lb, true_extent)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN
true_lb	整数	データ型の真の下限	OUT
true_extent	整数	データ型の真の寸法	OUT

C バインディング

```
int MPI_Type_get_true_extent (MPI_Datatype datatype, MPI_Aint *true_lb, MPI_Aint
                              *true_extent)
```

Fortran バインディング

```
call MPI_TYPE_GET_TRUE_EXTENT (DATATYPE, TRUE_LB, TRUE_EXTENT, IERROR)

    INTEGER                                DATATYPE, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) TRUE_LB, TRUE_EXTENT
```

MPI_Type_get_true_extent_x (C)

MPI_TYPE_GET_TRUE_EXTENT_X (Fortran)

データ型の真の下限 および 寸法の取得

基本構文

MPI_TYPE_GET_TRUE_EXTENT_X (datatype, true_lb, true_extent)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN
true_lb	整数	データ型の真の下限	OUT
true_extent	整数	データ型の真の寸法	OUT

C バインディング

int MPI_Type_get_true_extent_x (MPI_Datatype datatype, MPI_Count *true_lb, MPI_Count *true_extent)

Fortran バインディング

call MPI_TYPE_GET_TRUE_EXTENT_X (DATATYPE, TRUE_LB, TRUE_EXTENT, IERROR)

INTEGER DATATYPE, IERROR
INTEGER(KIND=MPI_COUNT_KIND) TRUE_LB, TRUE_EXTENT

MPI_Type_indexed (C) MPI_TYPE_INDEXED (Fortran)

個々のブロックの要素の個数 および 先頭位置(バイト単位)の指定による 新たなデータ型の生成

基本構文

MPI_TYPE_INDEXED (count, array_of_blocklengths, array_of_displacements, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数(0 以上の値)	IN
array_of_blocklengths	整数	ブロックごとの要素の個数(0 以上の値)	IN
array_of_displacements	整数	ブロックごとの先頭位置(単位は要素の個数)	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

int MPI_Type_indexed (int count, int *array_of_blocklengths, int *array_of_displacements, MPI_Datatype oldtype, MPI_Datatype *newtype)

Fortran バインディング

call MPI_TYPE_INDEXED (COUNT, ARRAY_OF_BLOCKLENGTHS, ARRAY_OF_DISPLACEMENTS, OLDTYPE, NEWTYPE, IERROR)

整数型 COUNT, ARRAY_OF_BLOCKLENGTHS(*), ARRAY_OF_DISPLACEMENTS(*), OLDTYPE, NEWTYPE, IERROR

MPI_Type_size (C) MPI_TYPE_SIZE (Fortran)

データ型の大きさの返却

基本構文

MPI_TYPE_SIZE (datatype, size)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN

size	整数	データ型の大きさ	OUT

C バインディング

int MPI_Type_size (MPI_Datatype datatype, int *size)

Fortran バインディング

call MPI_TYPE_SIZE (DATATYPE, SIZE, IERROR)

INTEGER DATATYPE, SIZE, IERROR

MPI_Type_size_x (C) MPI_TYPE_SIZE_X (Fortran)

データ型の大きさの返却

基本構文

MPI_TYPE_SIZE_X (datatype, size)

引数	値	説明	IN/OUT
datatype	handle	データ型	IN
size	整数	データ型の大きさ	OUT

C バインディング

int MPI_Type_size_x (MPI_Datatype datatype, MPI_Count *size)

Fortran バインディング

call MPI_TYPE_SIZE_X (DATATYPE, SIZE, IERROR)

INTEGER DATATYPE, IERROR
INTEGER(KIND=MPI_COUNT_KIND) SIZE

MPI_Type_vector (C) MPI_TYPE_VECTOR (Fortran)

ブロックの要素の個数 および ブロック間隔(要素単位)の指定による 新たなデータ型の生成

基本構文

MPI_TYPE_VECTOR (count, blocklength, stride, oldtype, newtype)

引数	値	説明	IN/OUT
count	整数	ブロック数(0 以上の値)	IN
blocklength	整数	ブロックを構成する要素の個数(0 以上の値)	IN
stride	整数	各ブロックの先頭位置(単位は要素の個数)	IN
oldtype	handle	元の要素のデータ型	IN
newtype	handle	新たに生成されるデータ型	OUT

C バインディング

int MPI_Type_vector (int count, int blocklength, int stride, MPI_Datatype oldtype,
MPI_Datatype *newtype)

Fortran バインディング

call MPI_TYPE_VECTOR (COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE,
IERROR)

整数型 COUNT, BLOCKLENGTH, STRIDE, OLDTYPE, NEWTYPE, IERROR

MPI_Unpack (C) MPI_UNPACK (Fortran)

データのアンパッキング

基本構文

MPI_UNPACK (inarea, insize, position, outdata, outcount, comm)

引数	値	説明	IN/OUT
inarea	任意	アンパッキング入力バッファ	IN
insize	整数	アンパッキング入力バッファの大きさ(単位バイト数)	IN
position	整数	アンパッキング入力バッファ内の位置(単位バイト数)	INOUT
outdata	任意	アンパッキングデータ	OUT
outcount	整数	アンパッキングデータの要素の個数	IN
datatype	handle	アンパッキングデータの型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Unpack (void* inarea, int insize, int *position, void* outdata, int outcount,
MPI_Datatype datatype, MPI_Comm comm)

Fortran バインディング

call MPI_UNPACK (INAREA, INSIZE, POSITION, OUTDATA, OUTCOUNT, DATATYPE,
COMM, IERROR)

任意の型 INAREA(*), OUTDATA(*)

整数型 INSIZE, POSITION, OUTCOUNT, DATATYPE, COMM, IERROR

MPI_Unpack_external (C)

MPI_UNPACK_EXTERNAL (Fortran)

データ表現指定によるデータのアンパッキング

基本構文

MPI_UNPACK_EXTERNAL (datarep, inbuf, insize, datatype, outbuf, outcount, position)

引数	値	説明	IN/OUT
datarep	文字	データ表現(文字列)	IN
inbuf	任意	入力バッファの始点	IN
insize	整数	入力バッファの大きさ(バイト単位)	IN
position	整数	バッファ内の現在位置(バイト単位)	INOUT
outbuf	任意	出力バッファの始点	OUT
outcount	整数	出力データの要素の個数	IN
datatype	handle	出力データのデータ型	IN

C バインディング

```
int MPI_Unpack_external (char *datarep, void *inbuf, MPI_Aint insize, MPI_Aint *position,
                         void *outbuf,
                         int outcount, MPI_Datatype datatype)
```

Fortran バインディング

```
call MPI_UNPACK_EXTERNAL (DATAREP, INBUF, INSIZE, POSITION, OUTBUF,
                          OUTCOUNT, DATATYPE, IERROR)

  INTEGER                              OUTCOUNT, DATATYPE, IERROR
  INTEGER(KIND=MPI_ADDRESS_KIND) INSIZE, POSITION
  CHARACTER*(*)                        DATAREP
  任意                                 INBUF(*), OUTBUF(*)
```

MPI_Aint_add (C) MPI_AINT_ADD (Fortran)

MPI_Aint_add は，引数 base および disp の和を返します。

基本構文

MPI_AINT_ADD (base, disp)

引数	値	説明	IN/OUT
base	整数	手続 MPI_GET_ADDRESS の呼出しにより返される起点アドレス	IN
disp	整数	符号付き整数型の変位	IN

C バインディング

int MPI_Aint_add (MPI_Aint base, MPI_Aint disp)

Fortran バインディング

INTEGER(KIND=MPI_ADDRESS_KIND) MPI_AINT_ADD (BASE, DISP)

INTEGER(KIND=MPI_ADDRESS_KIND), INTENT(IN) BASE, DISP

MPI_Aint_diff (C) MPI_AINT_DIFF (Fortran)

MPI_Aint_diff は，引数 addr1 および addr2 の間の差を返します。

基本構文

MPI_AINT_DIFF (addr1, addr2)

引数	値	説明	IN/OUT
addr1	整数	手続 MPI_GET_ADDRESS の呼出しにより返されるアドレス	IN
addr2	整数	手続 MPI_GET_ADDRESS の呼出しにより返されるアドレス	IN

C バインディング

```
int MPI_Aint_diff (MPI_Aint addr1, MPI_Aint addr1)
```

Fortran バインディング

```
INTEGER(KIND=MPI_ADDRESS_KIND) MPI_AINT_DIFF (ADDR1, ADDR2)

    INTEGER(KIND=MPI_ADDRESS_KIND),
    INTENT(IN)                          ADDR1, ADDR2
```

4.3 集団操作

MPI_ALLGATHER	MPI_ALLGATHERV	MPI_ALLREDUCE
MPI_ALLTOALL	MPI_ALLTOALLV	MPI_ALLTOALLW
MPI_BARRIER	MPI_BCAST	MPI_EXSCAN
MPI_GATHER	MPI_GATHERV	MPI_IALLGATHER
MPI_IALLGATHERV	MPI_IALLREDUCE	MPI_IALLTOALL
MPI_IALLTOALLV	MPI_IALLTOALLW	MPI_IBARRIER
MPI_IBCAST	MPI_IEXSCAN	MPI_IGATHER
MPI_IGATHERV	MPI_IREDUCE	MPI_IREDUCE_SCATTER
MPI_IREDUCE_SCATTER_BLOCK	MPI_ISCAN	MPI_ISCATTER
MPI_ISCATTERV	MPI_OP_COMMUTATIVE	MPI_OP_CREATE
MPI_OP_FREE	MPI_REDUCE	MPI_REDUCE_LOCAL
MPI_REDUCE_SCATTER	MPI_REDUCE_SCATTER_BLOCK	MPI_SCAN
MPI_SCATTER	MPI_SCATTERV	

MPI_Allgather (C) MPI_ALLGATHER (Fortran)

全プロセスへのデータの収集

基本構文

MPI_ALLGATHER (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	個々のプロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Allgather (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_ALLGATHER (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

MPI_Allgatherv (C) MPI_ALLGATHERV (Fortran)

全てのプロセスへのデータの収集

基本構文

MPI_ALLGATHERV (sendarea, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信バッファの要素の個数	IN
displs	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Allgatherv (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int *recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_ALLGATHERV (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, DISPLS, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, COMM, IERROR

集計演算と全プロセスへの結果の同報

基本構文

MPI_ALLREDUCE (senddata, recvdata, count, datatype, op, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファのアドレス	IN
recvdata	任意	受信バッファのアドレス	OUT
count	整数	送信バッファの要素の個数	IN
datatype	handle	送信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Allreduce (void* senddata, void* recvdata, int count, MPI_Datatype datatype,
                   MPI_Op op, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_ALLREDUCE (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM,
                    IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, IERROR

データの収集と拡散

基本構文

MPI_ALLTOALL (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	各プロセスへ送信する要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Alltoall (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_ALLTOALL (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

データの収集と拡散

基本構文

MPI_ALLTOALLV (sendarea, sendcounts, sdispls, sendtype, recvarea, recvcounts, rdispls, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置	IN
sendtype	handle	送信バッファのデータ型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素のデータ型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Alltoallv (void* sendarea, int *sendcounts, int *sdispls, MPI_Datatype sendtype, void* recvarea, int *recvcounts, int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_ALLTOALLV (SENDAREA, SENDCOUNTS, SDISPLS, SENDTYPE, RECVAREA, RECVCOUNTS, RDISPLS, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPE, RECVCOUNTS(*), RDISPLS(*), RECVTYPE, COMM, IERROR

データの収集と拡散

基本構文

MPI_ALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounts, rdispls, recvtypes, comm)

引数	値	説明	IN/OUT
sendbuf	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置 (バイト単位)	IN
sendtypes	handle	送信バッファのデータ型(プロセスごと)	IN
recvbuf	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置 (バイト単位)	IN
recvtypes	handle	受信バッファの要素のデータ型(プロセスごと)	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Alltoallw (void *sendbuf, int *sendcounts, int *sdispls, MPI_Datatype *sendtypes,
                   void *recvbuf,
                   int *recvcounts, int *rdispls, MPI_Datatype *recvtypes, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_ALLTOALLV (SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES, RECVBUF,
                    RECVCOUNTS, RDISPLS, RECVTYPES, COMM, IERROR)
```

任意の型	SENDBUF(*), RECVBUF(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPES(*), RECVCOUNTS(*), RDISPLS(*), RECVTYPES(*), COMM, IERROR

MPI_Barrier (C)　　MPI_BARRIER (Fortran)

バリア同期

基本構文

MPI_BARRIER (comm)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Barrier (MPI_Comm comm)

Fortran バインディング

call MPI_BARRIER (COMM, IERROR)

整数型 COMM, IERROR

MPI_Bcast (C)　　MPI_BCAST (Fortran)

同報通信

基本構文

MPI_BCAST (data, count, datatype, root, comm)

引数	値	説明	IN/OUT
data	任意	データの先頭アドレス	INOUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
root	整数	同報送信プロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Bcast (void* data, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Fortran バインディング

call MPI_BCAST (DATA, COUNT, DATATYPE, ROOT, COMM, IERROR)

任意の型　　DATA(*)
整数型　　COUNT, DATATYPE, ROOT, COMM, IERROR

集計演算

基本構文

MPI_EXSCAN (senddata, recvdata, count, datatype, op, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Exscan (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Fortran バインディング

call MPI_EXSCAN (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, IERROR

1 つのプロセスへのデータの収集

基本構文

MPI_GATHER (senddata, sendcount, sendtype, recvarea, recvcount, recvtype, root, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス(ルートプロセスだけ意味をもつ)	OUT
recvcount	整数	個々のプロセスから受信する要素の個数(ルートプロセスだけ意味をもつ)	IN
recvtype	handle	受信バッファのデータ型(ルートプロセスだけ意味をもつ)	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Gather (void* senddata, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Fortran バインディング

call MPI_GATHER (SENDDATA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)

任意の型	SENDDATA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

1 つのプロセスへのデータの収集

基本構文

MPI_GATHERV (senddata, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, root, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス(ルートプロセスだけ意味をもつ)	OUT
recvcounts	整数	受信する要素の個数(ルートプロセスだけ意味をもつ)	IN
displs	整数	各プロセスからの受信データの先頭位置(ルートプロセスだけ意味をもつ)	IN
recvtype	handle	受信バッファのデータ型(ルートプロセスだけ意味をもつ)	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Gatherv (void* senddata, int sendcount, MPI_Datatype sendtype, void* recvarea, int
                 *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_GATHERV (SENDDATA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNTS,
                  DISPLS, RECVTYPE, ROOT, COMM, IERROR)
```

任意の型	SENDDATA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, ROOT, COMM, IERROR

手続 MPI_Allgather の非ブロッキング版

基本構文

MPI_IALLGATHER (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	個々のプロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Iallgather (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int
        recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request
        *request)
```

Fortran バインディング

```
call MPI_IALLGATHER (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA,
                     RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR

手続 MPI_Allgatherv の非ブロッキング版

基本構文

MPI_IALLGATHERV (sendarea, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信バッファの要素の個数	IN
displs	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Iallgatherv (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea,
                     int *recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm,
                     MPI_Request *request)
```

Fortran バインディング

```
call MPI_IALLGATHERV (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA,
                      RECVCOUNT, DISPLS, RECVTYPE, COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, COMM, REQUEST, IERROR

手続 MPI_Allreduce の非ブロッキング版

基本構文

MPI_IALLREDUCE (senddata, recvdata, count, datatype, op, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファのアドレス	IN
recvdata	任意	受信バッファのアドレス	OUT
count	整数	送信バッファの要素の個数	IN
datatype	handle	送信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Iallreduce (void* senddata, void* recvdata, int count, MPI_Datatype datatype,
                    MPI_Op op, MPI_Comm comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_IALLREDUCE (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM,
                     REQUEST, IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, REQUEST, IERROR

手続 MPI_Alltoall の非ブロッキング版

基本構文

MPI_IALLTOALL (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	各プロセスへ送信する要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ialltoall (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int
                   recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request
                   *request)
```

Fortran バインディング

```
call MPI_IALLTOALL (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA,
                    RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR

手続 MPI_Alltoallv の非ブロッキング版

基本構文

MPI_IALLTOALLV (sendarea, sendcounts, sdispls, sendtype, recvarea, recvcounts, rdispls, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置	IN
sendtype	handle	送信バッファのデータ型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素のデータ型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ialltoallv (void* sendarea, int *sendcounts, int *sdispls, MPI_Datatype sendtype, void* recvarea, int *recvcounts, int *rdispls, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IALLTOALLV (SENDAREA, SENDCOUNTS, SDISPLS, SENDTYPE, RECVAREA, RECVCOUNTS, RDISPLS, RECVTYPE, COMM, REQUEST, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPE, RECVCOUNTS(*), RDISPLS(*), RECVTYPE, COMM, REQUEST, IERROR

手続 MPI_Alltoallw の非ブロッキング版

基本構文

MPI_IALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounts, rdispls, recvtypes, comm, request)

引数	値	説明	IN/OUT
sendbuf	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置 (バイト単位)	IN
sendtypes	handle	送信バッファのデータ型(プロセスごと)	IN
recvbuf	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置 (バイト単位)	IN
recvtypes	handle	受信バッファの要素のデータ型(プロセスごと)	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ialltoallw (void *sendbuf, int *sendcounts, int *sdispls, MPI_Datatype *sendtypes,
                    void *recvbuf,
                    int *recvcounts, int *rdispls, MPI_Datatype *recvtypes, MPI_Comm comm,
                    MPI_Request *request)
```

Fortran バインディング

```
call MPI_IALLTOALLV (SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES, RECVBUF,
                     RECVCOUNTS, RDISPLS, RECVTYPES, COMM, REQUEST,
                     IERROR)
```

任意の型	SENDBUF(*), RECVBUF(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPES(*), RECVCOUNTS(*), RDISPLS(*), RECVTYPES(*), COMM, REQUEST, IERROR

MPI_Ibarrier (C) MPI_IBARRIER (Fortran)

手続 MPI_Barrier の非ブロッキング版

基本構文

MPI_IBARRIER (comm, request)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ibarrier (MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IBARRIER (COMM, REQUEST, IERROR)

整数型 COMM, REQUEST, IERROR

MPI_Ibcast (C) MPI_IBCAST (Fortran)

手続 MPI_Bcast の非ブロッキング版

基本構文

MPI_IBCAST (data, count, datatype, root, comm, request)

引数	値	説明	IN/OUT
data	任意	データの先頭アドレス	INOUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
root	整数	同報送信プロセスのランク	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ibcast (void* data, int count, MPI_Datatype datatype, int root, MPI_Comm comm,
MPI_Request *request)

Fortran バインディング

call MPI_IBCAST (DATA, COUNT, DATATYPE, ROOT, COMM, REQUEST, IERROR)

任意の型 DATA(*)
整数型 COUNT, DATATYPE, ROOT, COMM, REQUEST, IERROR

手続 MPI_Exscan の非ブロッキング版

基本構文

MPI_IEXSCAN (senddata, recvdata, count, datatype, op, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Iexscan (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IEXSCAN (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM, REQUEST, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, REQUEST, IERROR

手続 MPI_Gather の非ブロッキング版

基本構文

MPI_IGATHER (senddata, sendcount, sendtype, recvarea, recvcount, recvtype, root, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
sendcount	整数 handle	送信バッファの要素の個数	IN
sendtype	任意	送信バッファの要素の型	IN
recvarea	整数 handle	受信バッファの先頭アドレス(ルートプロセスだけ意味をもつ)	OUT
recvcount		個々のプロセスから受信する要素の個数(ルートプロセスだけ意味をもつ)	IN
recvtype		受信バッファのデータ型(ルートプロセスだけ意味をもつ)	IN
root	整数 handle	ルートプロセスのランク	IN
comm		コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Igather (void* senddata, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IGATHER (SENDDATA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, ROOT, COMM, REQUEST, IERROR)

任意の型	SENDDATA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, REQUEST, IERROR

手続 MPI_Gatherv の非ブロッキング版

基本構文

MPI_IGATHERV (senddata, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, root, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス(ルートプロセスだけ意味をもつ)	OUT
recvcounts	整数	受信する要素の個数(ルートプロセスだけ意味をもつ)	IN
displs	整数	各プロセスからの受信データの先頭位置(ルートプロセスだけ意味をもつ)	IN
recvtype	handle	受信バッファのデータ型(ルートプロセスだけ意味をもつ)	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Igatherv (void* senddata, int sendcount, MPI_Datatype sendtype, void* recvarea, int
                  *recvcounts, int *displs, MPI_Datatype recvtype, int root, MPI_Comm comm,
                  MPI_Request *request)
```

Fortran バインディング

```
call MPI_IGATHERV (SENDDATA, SENDCOUNT, SENDTYPE, RECVAREA,
                   RECVCOUNTS, DISPLS, RECVTYPE, ROOT, COMM, REQUEST,
                   IERROR)
```

任意の型	SENDDATA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, ROOT, COMM, REQUEST, IERROR

手続 MPI_Reduce の非ブロッキング版

基本構文

MPI_IREDUCE (senddata, recvdata, count, datatype, op, root, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファのアドレス	IN
recvdata	任意	受信バッファのアドレス(ルートプロセスだけ意味をもつ)	OUT
count	整数	送信バッファの要素の個数	IN
datatype	handle	送信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ireduce (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_IREDUCE (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, ROOT, COMM, REQUEST, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, ROOT, COMM, REQUEST, IERROR

手続 MPI_Reduce_scatter の非ブロッキング版

基本構文

MPI_IREDUCE_SCATTER (senddata, recvdata, recvcounts, datatype, op, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	各プロセスへ分散させる結果の要素の個数	IN
datatype	handle	受信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ireduce_scatter (void* senddata, void* recvdata, int *recvcounts, MPI_Datatype
                         datatype, MPI_Op op, MPI_Comm comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_IREDUCE_SCATTER (SENDDATA, RECVDATA, RECVCOUNTS, DATATYPE, OP,
                          COMM, REQUEST, IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	RECVCOUNTS(*), DATATYPE, OP, COMM, REQUEST, IERROR

手続 MPI_Reduce_scatter_block の非ブロッキング版

基本構文

MPI_IREDUCE_SCATTER_BLOCK (senddata, recvdata, recvcounts, datatype, op, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスへ分散させる結果の要素の個数	IN
datatype	handle	受信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ireduce_scatter_block (void* senddata, void* recvdata, int recvcount, MPI_Datatype
    datatype, MPI_Op op, MPI_Comm comm, MPI_Request
    *request)
```

Fortran バインディング

```
call MPI_IREDUCE_SCATTER_BLOCK (SENDDATA, RECVDATA, RECVCOUNT,
    DATATYPE, OP, COMM, REQUEST, IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	RECVCOUNT, DATATYPE, OP, COMM, REQUEST, IERROR

手続 MPI_Scan の非ブロッキング版

基本構文

MPI_ISCAN (senddata, recvdata, count, datatype, op, comm, request)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Iscan (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_ISCAN (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM, REQUEST, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, REQUEST, IERROR

手続 MPI_Scatter の非ブロッキング版

基本構文

MPI_ISCATTER (sendarea, sendcount, sendtype, recvdata, recvcount, recvtype, root, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファのアドレス(ルートプロセスだけ意味をもつ)	IN
sendcount	整数	各プロセスへ送信する要素の個数(ルートプロセスだけ意味をもつ)	IN
sendtype	handle	送信バッファの要素のデータ型(ルートプロセスだけ意味をもつ)	IN
recvdata	任意	受信バッファのアドレス	OUT
recvcount	整数	受信バッファの要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Iscatter (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvdata, int
                  recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request
                  *request)
```

Fortran バインディング

```
call MPI_ISCATTER (SENDDATA, SENDCOUNT, SENDTYPE, RECVDATA, RECVCOUNT,
                   RECVTYPE, ROOT, COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVDATA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, REQUEST, IERROR

手続 MPI_Scatterv の非ブロッキング版

基本構文

MPI_ISCATTERV (sendarea, sendcounts, displs, sendtype, recvdata, recvcount, recvtype, root, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファのアドレス(ルートプロセスだけ意味をもつ)	IN
sendcounts	整数	各プロセスへ送信する要素の個数(ルートプロセスだけ意味をもつ)	IN
displs	整数	各プロセスへの送信データの先頭位置(ルートプロセスだけ意味をもつ)	IN
sendtype	handle	送信バッファの要素の型(ルートプロセスだけ意味をもつ)	IN
recvdata	任意	受信バッファのアドレス	OUT
recvcount	整数	受信バッファの要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Iscatterv (void* sendarea, int *sendcounts, int *displs, MPI_Datatype sendtype, void* recvdata, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_ISCATTERV (SENDDATA, SENDCOUNTS, DISPLS, SENDTYPE, RECVDATA, RECVCOUNT, RECVTYPE, ROOT, COMM, REQUEST, IERROR)

任意の型	SENDAREA(*), RECVDATA(*)
整数型	SENDCOUNTS(*), DISPLS(*), SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, REQUEST, IERROR

集計演算子の可換性問合せ

基本構文

MPI_OP_COMMUTATIVE (function, commute, op)

引数	値	説明	IN/OUT
op commute	handle 論理	集計演算子 フラグ	IN OUT

C バインディング

int MPI_Op_commutative (MPI_Op op, int *commute)

Fortran バインディング

call MPI_OP_ COMMUTATIVE (OP, COMMUTE, IERROR)

論理型 COMMUTE
整数型 OP, IERROR

利用者定義集計演算子の生成

基本構文

MPI_OP_CREATE (function, commute, op)

引数	値	説明	IN/OUT
function	function	利用者定義関数	IN
commute	論理	可換の場合 真, さもなければ 偽	IN
op	handle	集計演算子	OUT

C バインディング

int MPI_Op_create (MPI_User_function *function, int commute, MPI_Op *op)

typedef void MPI_User_function (void *invec, void *inoutvec, int *len, MPI_Datatype *datatype)

Fortran バインディング

call MPI_OP_CREATE (USER_FUNCTION, COMMUTE, OP, IERROR)
EXTERNAL USER_FUNCTION
論理型 COMMUTE
整数型 OP, IERROR

FUNCTION USER_FUNCTION (INVEC(*), INOUTVEC(*), LEN, TYPE)

任意の型 INVEC(LEN), INOUTVEC(LEN)
整数型 LEN, TYPE

MPI_Op_free (C)　　MPI_OP_FREE (Fortran)

利用者定義集計演算子の解放

基本構文

MPI_OP_FREE (op)

引数	値	説明	IN/OUT
op	handle	集計演算子	INOUT

C バインディング

int MPI_Op_free (MPI_Op *op)

Fortran バインディング

call MPI_OP_FREE (OP, IERROR)

整数型 OP, IERROR

MPI_Reduce (C)　　MPI_REDUCE (Fortran)

集計演算

基本構文

MPI_REDUCE (senddata, recvdata, count, datatype, op, root, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファのアドレス	IN
recvdata	任意	受信バッファのアドレス(ルートプロセスだけ意味をもつ)	OUT
count	整数	送信バッファの要素の個数	IN
datatype	handle	送信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Reduce (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Fortran バインディング

call MPI_REDUCE (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, ROOT, COMM, IERROR)

任意の型　　SENDDATA(*), RECVDATA(*)

整数型　　COUNT, DATATYPE, OP, ROOT, COMM, IERROR

プロセス内集計演算

基本構文

MPI_REDUCE_LOCAL (inbuf, inoutbuf, count, datatype, op)

引数	値	説明	IN/OUT
inbuf	任意	入力データのアドレス	IN
inoutbuf	任意	入出力データのアドレス	INOUT
count	整数	入力データの要素の個数	IN
datatype	handle	入力データの型	IN
op	handle	集計演算の集計演算子	IN

C バインディング

```
int MPI_Reduce_local (void* inbuf, void* inoutbuf, int count, MPI_Datatype datatype, MPI_Op
    op)
```

Fortran バインディング

```
call MPI_REDUCE_LOCAL (INBUF, INOUTBUF, COUNT, DATATYPE, OP, IERROR)

    任意の型        INBUF(*), INOUTBUF(*)
    整数型          COUNT, DATATYPE, OP, IERROR
```

集計演算と各プロセスへの結果の拡散

基本構文

MPI_REDUCE_SCATTER (senddata, recvdata, recvcounts, datatype, op, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	各プロセスへ分散させる結果の要素の個数	IN
datatype	handle	受信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Reduce_scatter (void* senddata, void* recvdata, int *recvcounts, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Fortran バインディング

call MPI_REDUCE_SCATTER (SENDDATA, RECVDATA, RECVCOUNTS, DATATYPE, OP, COMM, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	RECVCOUNTS(*), DATATYPE, OP, COMM, IERROR

集計演算と各プロセスへの結果の拡散

基本構文

MPI_REDUCE_SCATTER_BLOCK (senddata, recvdata, recvcounts, datatype, op, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスへ分散させる結果の要素の個数	IN
datatype	handle	受信バッファの要素の型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Reduce_scatter_block (void* senddata, void* recvdata, int recvcount, MPI_Datatype
                              datatype, MPI_Op op, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_REDUCE_SCATTER_BLOCK (SENDDATA, RECVDATA, RECVCOUNT,
                               DATATYPE, OP, COMM, IERROR)
```

任意の型	SENDDATA(*), RECVDATA(*)
整数型	RECVCOUNT, DATATYPE, OP, COMM, IERROR

累計演算

基本構文

MPI_SCAN (senddata, recvdata, count, datatype, op, comm)

引数	値	説明	IN/OUT
senddata	任意	送信バッファの先頭アドレス	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
count	整数	データの要素の個数	IN
datatype	handle	データの型	IN
op	handle	集計演算の集計演算子	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Scan (void* senddata, void* recvdata, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)

Fortran バインディング

call MPI_SCAN (SENDDATA, RECVDATA, COUNT, DATATYPE, OP, COMM, IERROR)

任意の型	SENDDATA(*), RECVDATA(*)
整数型	COUNT, DATATYPE, OP, COMM, IERROR

データの拡散

基本構文

MPI_SCATTER (sendarea, sendcount, sendtype, recvdata, recvcount, recvtype, root, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファのアドレス(ルートプロセスだけ意味をもつ)	IN
sendcount	整数	各プロセスへ送信する要素の個数(ルートプロセスだけ意味をもつ)	IN
sendtype	handle	送信バッファの要素のデータ型(ルートプロセスだけ意味をもつ)	IN
recvdata	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	受信バッファの要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Scatter (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvdata, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)

Fortran バインディング

call MPI_SCATTER (SENDDATA, SENDCOUNT, SENDTYPE, RECVDATA, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)

任意の型	SENDAREA(*), RECVDATA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

データの拡散

基本構文

MPI_SCATTERV (sendarea, sendcounts, displs, sendtype, recvdata, recvcount, recvtype, root, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファのアドレス(ルートプロセスだけ意味をもつ)	IN
sendcounts	整数	各プロセスへ送信する要素の個数(ルートプロセスだけ意味をもつ)	IN
displs	整数	各プロセスへの送信データの先頭位置(ルートプロセスだけ意味をもつ)	IN
sendtype	handle	送信バッファの要素の型(ルートプロセスだけ意味をもつ)	IN
recvdata	任意	受信バッファのアドレス	OUT
recvcount	整数	受信バッファの要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
root	整数	ルートプロセスのランク	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Scatterv (void* sendarea, int *sendcounts, int *displs, MPI_Datatype sendtype, void*
                  recvdata, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_SCATTERV (SENDDATA, SENDCOUNTS, DISPLS, SENDTYPE, RECVDATA,
                   RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR)
```

任意の型	SENDAREA(*), RECVDATA(*)
整数型	SENDCOUNTS(*), DISPLS(*), SENDTYPE, RECVCOUNT, RECVTYPE, ROOT, COMM, IERROR

4.4 グループ，コンテキスト，コミュニケーター

MPI_COMM_COMPAR	MPI_COMM_CREATE	MPI_COMM_CREATE_GROUP
MPI_COMM_CREATE_KEYVAL	MPI_COMM_DELETE_ATTR	MPI_COMM_DUP
MPI_COMM_DUP_WITH_INFO	MPI_COMM_FREE	MPI_COMM_FREE_KEYVAL
MPI_COMM_GET_ATTR	MPI_COMM_GET_INFO	MPI_COMM_GET_NAME
MPI_COMM_GROUP	MPI_COMM_IDUP	MPI_COMM_RANK
MPI_COMM_REMOTE_GROUP	MPI_COMM_REMOTE_SIZE	MPI_COMM_SET_ATTR
MPI_COMM_SET_INFO	MPI_COMM_SET_NAME	MPI_COMM_SIZE
MPI_COMM_SPLIT	MPI_COMM_SPLIT_TYPE	MPI_COMM_TEST_INTER
MPI_GROUP_COMPARE	MPI_GROUP_DIFFERENCE	MPI_GROUP_EXCL
MPI_GROUP_FREE	MPI_GROUP_INCL	MPI_GROUP_INTERSECTION
MPI_GROUP_RANGE_EXCL	MPI_GROUP_RANGE_INCL	MPI_GROUP_RANK
MPI_GROUP_SIZE	MPI_GROUP_TRANSLATE_RANKS	MPI_GROUP_UNION
MPI_INTERCOMM_CREATE	MPI_INTERCOMM_MERGE	MPI_TYPE_CREATE_KEYVAL
MPI_TYPE_DELETE_ATTR	MPI_TYPE_FREE_KEYVAL	MPI_TYPE_GET_ATTR
MPI_TYPE_GET_NAME	MPI_TYPE_SET_ATTR	MPI_TYPE_SET_NAME
MPI_WIN_CREATE_KEYVAL	MPI_WIN_DELETE_ATTR	MPI_WIN_FREE_KEYVAL
MPI_WIN_GET_ATTR	MPI_WIN_GET_NAME	MPI_WIN_SET_ATTR
MPI_WIN_SET_NAME		

MPI_Comm_compare (C) MPI_COMM_COMPARE (Fortran)

コミュニケーターの比較

基本構文

MPI_COMM_COMPARE (comm1, comm2, result)

引数	値	説明	IN/OUT
comm1	handle	コミュニケーター1	IN
comm2	handle	コミュニケーター2	IN
result	整数	比較結果	OUT

C バインディング

```
int MPI_Comm_compare (MPI_Comm comm1, MPI_Comm comm2, int *result)
```

Fortran バインディング

```
call MPI_COMM_COMPARE (COMM1, COMM2, RESULT, IERROR)
    整数型          COMM1, COMM2, RESULT, IERROR
```

MPI_Comm_create (C)　　MPI_COMM_CREATE (Fortran)

新たなコミュニケーターの生成(コミュニケータの部分集合)

基本構文

MPI_COMM_CREATE (comm, group, newcomm)

引数	値	説明	IN/OUT
comm	handle	生成元コミュニケーター	IN
group	handle	comm の部分集合のグループ	IN
newcomm	handle	新たに生成されたコミュニケーター	OUT

C バインディング

int MPI_Comm_create (MPI_Comm comm, MPI_Group group, MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_CREATE (COMM, GROUP, NEWCOMM, IERROR)
整数型　　COMM, GROUP, NEWCOMM, IERROR

MPI_Comm_create_group (C)　　MPI_COMM_CREATE_GROUP (Fortran)

新たなコミュニケーターの生成(コミュニケータの部分集合)

基本構文

MPI_COMM_CREATE_GROUP (comm, group, tag, newcomm)

引数	値	説明	IN/OUT
comm	handle	生成元コミュニケーター	IN
group	handle	comm の部分集合のグループ	IN
tag	整数	タグ	IN
newcomm	handle	新たに生成されたコミュニケーター	OUT

C バインディング

int MPI_Comm_create_group (MPI_Comm comm, MPI_Group group, int tag, MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_CREATE_GROUP (COMM, GROUP, TAG, NEWCOMM, IERROR)

整数型　　COMM, GROUP, TAG, NEWCOMM, IERROR

MPI_Comm_create_keyval (C)　　MPI_COMM_CREATE_KEYVAL (Fortran)

コミュニケーターに付加する属性の生成

基本構文

MPI_COMM_CREATE_KEYVAL (comm_copy_attr_fn, comm_delete_attr_fn, comm_keyval, extra_state)

引数	値	説明	IN/OUT
comm_copy_attr_fn	関数	comm_keyval に関する複製コールバック関数	IN
comm_delete_attr_fn	関数	comm_keyval に関する削除コールバック関数	IN
comm_keyval	整数	今後のアクセスのためのキー値	OUT
extra_state	状態	コールバック関数の拡張状態	IN

C バインディング

```
int MPI_Comm_create_keyval (MPI_Comm_copy_attr_function *comm_copy_attr_fn,
                            MPI_Comm_delete_attr_function *comm_delete_attr_fn, int
                            *comm_keyval, void *extra_state)
```

Fortran バインディング

```
call MPI_COMM_CREATE_KEYVAL (COMM_COPY_ATTR_FN,
                             COMM_DELETE_ATTR_FN, COMM_KEYVAL,
                             EXTRA_STATE, IERROR)

EXTERNAL                                COMM_COPY_ATTR_FN,
                                        COMM_DELETE_ATTR_FN
INTEGER                                 COMM_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND)          EXTRA_STATE
```

MPI_Comm_delete_attr (C)　　MPI_COMM_DELETE_ATTR (Fortran)

コミュニケーターに付加する属性の削除

基本構文

MPI_COMM_DELETE_ATTR (comm, comm_keyval)

引数	値	説明	IN/OUT
comm comm_keyval	handle 整数	属性が削除されるコミュニケーター キー値	INOUT IN

C バインディング

int MPI_Comm_delete_attr (MPI_Comm comm, int comm_keyval)

Fortran バインディング

call MPI_COMM_DELETE_ATTR (COMM, COMM_KEYVAL, IERROR)

INTEGER　　COMM, COMM_KEYVAL, IERROR

MPI_Comm_dup (C)　　MPI_COMM_DUP (Fortran)

コミュニケーターの複製

基本構文

MPI_COMM_DUP (comm, newcomm)

引数	値	説明	IN/OUT
comm newcomm	handle handle	複製元コミュニケーター 複製されたコミュニケーター	IN OUT

C バインディング

int MPI_Comm_dup (MPI_Comm comm, MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_DUP (COMM, NEWCOMM, IERROR)

整数型　　COMM, NEWCOMM, IERROR

コミュニケーターの複製

基本構文

MPI_COMM_DUP_WITH_INFO (comm, info, newcomm)

引数	値	説明	IN/OUT
comm	handle	複製元コミュニケーター	IN
info	handle	info オブジェクト	IN
newcomm	handle	複製されたコミュニケーター	OUT

C バインディング

int MPI_Comm_dup_with_info (MPI_Comm comm, MPI_Info info, MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_DUP_WITH_INFO (COMM, INFO, NEWCOMM, IERROR)
整数型 COMM, INFO, NEWCOMM, IERROR

コミュニケーターの解放

基本構文

MPI_COMM_FREE (comm)

引数	値	説明	IN/OUT
comm	handle	解放対象コミュニケーター	INOUT

C バインディング

int MPI_Comm_free (MPI_Comm *comm)

Fortran バインディング

call MPI_COMM_FREE (COMM, IERROR)
整数型 COMM, IERROR

コミュニケーターに付加する属性の解放

基本構文

MPI_COMM_FREE_KEYVAL (comm_keyval)

引数	値	説明	IN/OUT
comm_keyval	整数	キー値	INOUT

C バインディング

```
int MPI_Comm_free_keyval (int *comm_keyval)
```

Fortran バインディング

```
call MPI_COMM_FREE_KEYVAL (COMM_KEYVAL, IERROR)

    INTEGER          COMM_KEYVAL, IERROR
```

MPI_Comm_get_attr (C) MPI_COMM_GET_ATTR (Fortran)

コミュニケーターに付加された属性値の取得

基本構文

MPI_COMM_GET_ATTR (comm, comm_keyval, attribute_val, flag)

引数	値	説明	IN/OUT
comm	handle	属性が結び付けられたコミュニケーター	IN
comm_keyval	整数	キー値	IN
attribute_val	属性	属性値	OUT
flag	論理	フラグ	OUT

C バインディング

```
int MPI_Comm_get_attr (MPI_Comm comm, int comm_keyval, void *attribute_val, int *flag)
```

Fortran バインディング

```
call MPI_COMM_GET_ATTR (COMM, COMM_KEYVAL, ATTRIBUTE_VAL, FLAG,
                        IERROR)

    INTEGER                                COMM, COMM_KEYVAL, IERROR
    INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL
```

LOGICAL FLAG

MPI_Comm_get_info (C) MPI_COMM_GET_INFO (Fortran)

コミュニケーターに付加された info オブジェクトの取得

基本構文

MPI_COMM_GET_INFO (comm, info_used)

引数	値	説明	IN/OUT
comm	handle	属性が結び付けられたコミュニケーター	IN
info_used	handle	info オブジェクト	OUT

C バインディング

int MPI_Comm_get_info (MPI_Comm comm, MPI_Info *info_used)

Fortran バインディング

call MPI_COMM_GET_INFO (COMM, INFO_USED, IERROR)

INTEGER COMM, INFO_USED, IERROR

MPI_Comm_get_name (C) MPI_COMM_GET_NAME (Fortran)

コミュニケーターに付加された名前の取得

基本構文

MPI_COMM_GET_NAME (comm, comm_name, resultlen)

引数	値	説明	IN/OUT
comm	handle	名前が返却されるコミュニケーター	IN
comm_name	文字	コミュニケーターに設定された名前(文字列)	OUT
resultlen	整数	返却された名前の長さ	OUT

C バインディング

int MPI_Comm_get_name (MPI_Comm comm, char *comm_name, int *resultlen)

Fortran バインディング

call MPI_COMM_GET_NAME (COMM, COMM_NAME, RESULTLEN, IERROR)

INTEGER COMM, RESULTLEN, IERROR
CHARACTER*(*) COMM_NAME

MPI_Comm_group (C) MPI_COMM_GROUP (Fortran)

コミュニケーターからグループへの変換

基本構文

MPI_COMM_GROUP (comm, group)

引数	値	説明	IN/OUT
comm	handle	変換対象コミュニケーター	IN
group	handle	変換結果グループ	OUT

C バインディング

int MPI_Comm_group (MPI_Comm comm, MPI_Group *group)

Fortran バインディング

call MPI_COMM_GROUP (COMM, GROUP, IERROR)
整数型 COMM, GROUP, IERROR

MPI_Comm_idup (C) MPI_COMM_IDUP (Fortran)

コミュニケーターの複製 (非ブロッキング版)

基本構文

MPI_COMM_IDUP (comm, newcomm, request)

引数	値	説明	IN/OUT
comm	handle	複製元コミュニケーター	IN
newcomm	handle	複製されたコミュニケーター	OUT
request	handle	複製要求	OUT

C バインディング

int MPI_Comm_dup (MPI_Comm comm, MPI_Comm *newcomm, MPI_Request *request)

Fortran バインディング

call MPI_COMM_DUP (COMM, NEWCOMM, REQUEST, IERROR)
整数型 COMM, NEWCOMM, REQUEST, IERROR

MPI_Comm_rank (C)　　MPI_COMM_RANK (Fortran)

コミュニケーター中のランク

基本構文

MPI_COMM_RANK (comm, rank)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
rank	整数	コミュニケーター中のランク	OUT

C バインディング

```
int MPI_Comm_rank (MPI_Comm comm, int *rank)
```

Fortran バインディング

```
call MPI_COMM_RANK (COMM, RANK, IERROR)
    整数型          COMM, RANK, IERROR
```

MPI_Comm_remote_group (C)　　MPI_COMM_REMOTE_GROUP (Fortran)

インターコミュニケーターのリモートグループ

基本構文

MPI_COMM_REMOTE_GROUP (comm, group)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
group	handle	リモートグループ	OUT

C バインディング

```
int MPI_Comm_remote_group (MPI_Comm comm, MPI_Group *group)
```

Fortran バインディング

```
call MPI_COMM_REMOTE_GROUP (COMM, GROUP, IERROR)
    整数型          COMM, GROUP, IERROR
```

MPI_Comm_remote_size (C) MPI_COMM_REMOTE_SIZE (Fortran)

インターコミュニケーターのリモートグループの大きさ

基本構文

MPI_COMM_REMOTE_SIZE (comm, size)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
size	整数	リモートグループの大きさ	OUT

C バインディング

```
int MPI_Comm_remote_size (MPI_Comm comm, int *size)
```

Fortran バインディング

```
call MPI_COMM_REMOTE_SIZE (COMM, SIZE, IERROR)
    整数型          COMM, SIZE, IERROR
```

MPI_Comm_set_attr (C) MPI_COMM_SET_ATTR (Fortran)

コミュニケーターに対する属性の付加

基本構文

MPI_COMM_SET_ATTR (comm, comm_keyval, attribute_val)

引数	値	説明	IN/OUT
comm	handle	属性を結びつけるコミュニケーター	INOUT
comm_keyval	整数	キー値	IN
attribute_val	属性	属性値	IN

C バインディング

```
int MPI_Comm_set_attr (MPI_Comm comm, int comm_keyval, void *attribute_val)
```

Fortran バインディング

```
call MPI_COMM_SET_ATTR (COMM, COMM_KEYVAL, ATTRIBUTE_VAL, IERROR)
```

INTEGER COMM, COMM_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL

MPI_Comm_set_info (C) MPI_COMM_SET_INFO (Fortran)

コミュニケーターに対する info オブジェクトの付加

基本構文

MPI_COMM_SET_INFO (comm, info)

引数	値	説明	IN/OUT
comm	handle	属性が結び付けられたコミュニケーター	INOUT
info	handle	info オブジェクト	IN

C バインディング

int MPI_Comm_set_info (MPI_Comm comm, MPI_Info info)

Fortran バインディング

call MPI_COMM_SET_INFO (COMM, INFO, IERROR)

INTEGER COMM, INFO, IERROR

MPI_Comm_set_name (C) MPI_COMM_SET_NAME (Fortran)

コミュニケーターに対する名前の付加

基本構文

MPI_COMM_SET_NAME (comm, comm_name)

引数	値	説明	IN/OUT
comm	handle	識別子を設定するコミュニケーター	INOUT
comm_name	文字	名前として記憶される文字列	IN

C バインディング

int MPI_Comm_set_name (MPI_Comm comm, char *comm_name)

Fortran バインディング

call MPI_COMM_SET_NAME (COMM, COMM_NAME, IERROR)

INTEGER COMM, IERROR
CHARACTER*(*) COMM_NAME

MPI_Comm_size (C) MPI_COMM_SIZE (Fortran)

コミュニケーターの大きさ

基本構文

MPI_COMM_SIZE (comm, size)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
size	整数	コミュニケーターの大きさ	OUT

C バインディング

int MPI_Comm_size (MPI_Comm comm, int *size)

Fortran バインディング

call MPI_COMM_SIZE (COMM, SIZE, IERROR)
整数型 COMM, SIZE, IERROR

MPI_Comm_split (C) MPI_COMM_SPLIT (Fortran)

コミュニケーターの分割

基本構文

MPI_COMM_SPLIT (comm, color, key, newcomm)

引数	値	説明	IN/OUT
comm	handle	分割対象コミュニケーター	IN
color	整数	分割基準値(color の値が同じ MPI プロセスの部分集合が 1 つのコミュニケーターを形成する)	IN
key	整数	分割されたコミュニケーター内でのランクを決定するための値	IN
newcomm	handle	新たに生成されたコミュニケーター	OUT

C バインディング

int MPI_Comm_split (MPI_Comm comm, int color, int key, MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_SPLIT (COMM, COLOR, KEY, NEWCOMM, IERROR)
整数型　　COMM, COLOR, KEY, NEWCOMM, IERROR

MPI_Comm_split_type (C)　　MPI_COMM_SPLIT_TYPE (Fortran)

コミュニケーターの分割

基本構文

MPI_COMM_SPLIT_TYPE (comm, split_type, key, info, newcomm)

引数	値	説明	IN/OUT
comm	handle	分割対象コミュニケーター	IN
split_type	整数	分割種別	IN
key	整数	分割されたコミュニケーター内でのランクを決定するための値	IN
info	handle	info オブジェクト	IN
newcomm	handle	新たに生成されたコミュニケーター	OUT

C バインディング

int MPI_Comm_split_type (MPI_Comm comm, int split_type, int key, MPI_Info info,
MPI_Comm *newcomm)

Fortran バインディング

call MPI_COMM_SPLIT_TYPE (COMM, SPLIT_TYPE, KEY, INFO, NEWCOMM, IERROR)
整数型　　COMM, SPLIT_TYPE, KEY, INFO, NEWCOMM, IERROR

MPI_Comm_test_inter (C)　　MPI_COMM_TEST_INTER (Fortran)

インターコミュニケーター または イントラコミュニケーターの判定（インターコミュニケーターなら真，さもなければ，偽を返却する。）

基本構文

MPI_COMM_TEST_INTER (comm, flag)

引数	値	説明	IN/OUT
comm	handle	判定対象コミュニケーター	IN
flag	整数	判定結果	OUT

C バインディング

int MPI_Comm_test_inter (MPI_Comm comm, int *flag)

Fortran バインディング

call MPI_COMM_TEST_INTER (COMM, FLAG, IERROR)

整数型　　COMM, IERROR

論理型　　FLAG

MPI_Group_compare (C)　　MPI_GROUP_COMPARE (Fortran)

グループメンバの比較

基本構文

MPI_GROUP_COMPARE (group1, group2, result)

引数	値	説明	IN/OUT
group1	handle	グループ 1	IN
group2	handle	グループ 2	IN
result	整数	比較結果	OUT

C バインディング

int MPI_Group_compare (MPI_Group group1, MPI_Group group2, int *result)

Fortran バインディング

call MPI_GROUP_COMPARE (GROUP1, GROUP2, RESULT, IERROR)

整数型　　GROUP1, GROUP2, RESULT, IERROR

MPI_Group_difference (C)　　MPI_GROUP_DIFFERENCE (Fortran)

グループメンバの相違

基本構文

MPI_GROUP_DIFFERENCE (group1, group2, newgroup)

引数	値	説明	IN/OUT
group1	handle	グループ 1	IN
group2	handle	グループ 2	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_difference (MPI_Group group1, MPI_Group group2, int *newgroup)

Fortran バインディング

call MPI_GROUP_DIFFERENCE (GROUP1, GROUP2, NEWGROUP, IERROR)
整数型 GROUP1, GROUP2, NEWGROUP, IERROR

MPI_Group_excl (C) MPI_GROUP_EXCL (Fortran)

新たなグループの生成(グループの部分集合)

基本構文

MPI_GROUP_EXCL (group, n, ranks, newgroup)

引数	値	説明	IN/OUT
group	handle	グループ	IN
n	整数	ranks の要素の個数	IN
ranks	整数	部分集合を成すランク	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_excl (MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

Fortran バインディング

call MPI_GROUP_EXCL (GROUP, N, RANKS, NEWGROUP, IERROR)
整数型 GROUP, N, RANKS(*), NEWGROUP, IERROR

MPI_Group_free (C) MPI_GROUP_FREE (Fortran)

グループの解放

基本構文

MPI_GROUP_FREE (group)

引数	値	説明	IN/OUT
group	handle	解放対象のグループ	INOUT

C バインディング

int MPI_Group_free (MPI_Group *group)

Fortran バインディング

call MPI_GROUP_FREE (GROUP, IERROR)

整数型　　GROUP, IERROR

」

MPI_Group_incl (C)　　MPI_GROUP_INCL (Fortran)

新たなグループの生成(グループの部分集合)

基本構文

MPI_GROUP_INCL (group, n, ranks, newgroup)

引数	値	説明	IN/OUT
group	handle	グループ	IN
n	整数	ranks の要素の個数	IN
ranks	整数	部分集合を成すランク	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_incl (MPI_Group group, int n, int *ranks, MPI_Group *newgroup)

Fortran バインディング

call MPI_GROUP_INCL (GROUP, N, RANKS, NEWGROUP, IERROR)
整数型　　GROUP, N, RANKS(*), NEWGROUP, IERROR

MPI_Group_intersection (C)　　MPI_GROUP_INTERSECTION (Fortran)

グループメンバの積集合

基本構文

MPI_GROUP_INTERSECTION (group1, group2, newgroup)

引数	値	説明	IN/OUT
group1	handle	グループ 1	IN
group2	handle	グループ 2	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_intersection (MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)

Fortran バインディング

call MPI_GROUP_INTERSECTION (GROUP1, GROUP2, NEWGROUP, IERROR)
整数型　　GROUP1, GROUP2, NEWGROUP, IERROR

MPI_Group_range_excl (C)　　MPI_GROUP_RANGE_EXCL (Fortran)

新たなグループの生成(グループの部分集合)

基本構文

MPI_GROUP_RANGE_EXCL (group, n, ranges, newgroup)

引数	値	説明	IN/OUT
group	handle	グループ	IN
n	整数	ranges の三組みの個数	IN
ranges	整数	ランクの範囲を表す三組み	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_range_excl (MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)

Fortran バインディング

call MPI_GROUP_RANGE_EXCL (GROUP, N, RANGES, NEWGROUP, IERROR)
整数型　　GROUP, N, RANGES(3,*), NEWGROUP, IERROR

MPI_Group_range_incl (C)　　MPI_GROUP_RANGE_INCL (Fortran)

新たなグループの生成(グループの部分集合)

基本構文

MPI_GROUP_RANGE_INCL (group, n, ranges, newgroup)

引数	値	説明	IN/OUT
group	handle	グループ	IN
n	整数	ranges の三組みの個数	IN
ranges	整数	ランクの範囲を表す三組み	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

int MPI_Group_range_incl (MPI_Group group, int n, int ranges[][3], MPI_Group *newgroup)

Fortran バインディング

call MPI_GROUP_RANGE_INCL (GROUP, N, RANGES, NEWGROUP, IERROR)
整数型 GROUP, N, RANGES(3,*), NEWGROUP, IERROR

MPI_Group_rank (C) MPI_GROUP_RANK (Fortran)

グループ内のランク

基本構文

MPI_GROUP_RANK (group, rank)

引数	値	説明	IN/OUT
group	handle	グループのハンドル	IN
rank	整数	グループ内のランク	OUT

C バインディング

int MPI_Group_rank (MPI_Group group, int *rank)

Fortran バインディング

call MPI_GROUP_RANK (GROUP, RANK, IERROR)
整数型 GROUP, RANK, IERROR

MPI_Group_size (C) MPI_GROUP_SIZE (Fortran)

グループの大きさ

基本構文

MPI_GROUP_SIZE (group, size)

引数	値	説明	IN/OUT
group	handle	グループのハンドル	IN
size	整数	グループ内の大きさ	OUT

C バインディング

int MPI_Group_size (MPI_Group group, int *size)

Fortran バインディング

call MPI_GROUP_SIZE (GROUP, SIZE, IERROR)

整数型　　GROUP, SIZE, IERROR

MPI_Group_translate_ranks (C)　　MPI_GROUP_TRANSLATE_RANKS (Fortran)

異なるグループ中のランク問合せ

基本構文

MPI_GROUP_TRANSLATE_RANKS (group1, n, ranks1, group2, ranks2)

引数	値	説明	IN/OUT
group1	handle	グループ 1	IN
n	整数	ranks1 および ranks2 中のランク数	IN
ranks1	整数	group1 中でのランク	IN
group2	handle	グループ 2	IN
ranks2	整数	group2 中でのランク	OUT

C バインディング

```
int MPI_Group_translate_ranks (MPI_Group group1, int n, int *ranks1, MPI_Group group2,
                               int *ranks2)
```

Fortran バインディング

```
call MPI_GROUP_TRANSLATE_RANKS (GROUP1, N, RANKS1, GROUP2, RANKS2,
                                IERROR)
```

整数型　　GROUP1, N, RANKS1(*), GROUP2, RANKS2(*), IERROR

MPI_Group_union (C)　　MPI_GROUP_UNION (Fortran)

グループメンバの和集合

基本構文

MPI_GROUP_UNION (group1, group2, newgroup)

引数	値	説明	IN/OUT
group1	handle	グループ 1	IN
group2	handle	グループ 2	IN
newgroup	handle	新たに生成されたグループ	OUT

C バインディング

```
int MPI_Group_union (MPI_Group group1, MPI_Group group2, MPI_Group *newgroup)
```

Fortran バインディング

call MPI_GROUP_UNION (GROUP1, GROUP2, NEWGROUP, IERROR)

整数型　　GROUP1, GROUP2, NEWGROUP, IERROR

MPI_Intercomm_create (C)　　MPI_INTERCOMM_CREATE (Fortran)

インターコミュニケーター生成

基本構文

MPI_INTERCOMM_CREATE (local_comm, local_leader, peer_comm, remote_leader, tag, newintercomm)

引数	値	説明	IN/OUT
local_comm	handle	ローカルコミュニケータ	IN
local_leader	整数	ローカルコミュニケータの leader	IN
peer_comm	handle	local_leader と remote_leader が属すコミュニケーター (local_leader でだけ意味がある)	IN
remote_leader	整数	peer_comm 内での remote_leader のランク(local_leader でだけ意味がある)	IN
tag	整数	タグ(inter-communicator 生成のために内部で使用する)	IN
newintercomm	handle	新たに生成された inter-communicator	OUT

C バインディング

int MPI_Intercomm_create (MPI_Comm local_comm, int local_leader, MPI_Comm peer_comm, int remote_leader, int tag, MPI_Comm *newintercomm)

Fortran バインディング

call MPI_INTERCOMM_CREATE (LOCAL_COMM, LOCAL_LEADER, PEER_COMM, REMOTE_LEADER, TAG, NEWINTERCOMM, IERROR)

整数型　　LOCAL_COMM, LOCAL_LEADER, PEER_COMM, REMOTE_LEADER, TAG, NEWINTERCOMM, IERROR

インターコミュニケーターのローカルグループとリモートグループをマージし， イントラコミュニケーターを生成する

基本構文

MPI_INTERCOMM_MERGE (intercomm, high, newintracomm)

引数	値	説明	IN/OUT
intercomm	handle	マージ対象 inter-communicator	IN
high	整数	マージ後のランク付け順序の指示	IN
newintracomm	Handle	ローカルグループとリモートグループをマージして 新たに生成されたコミュニケーター	OUT

C バインディング

int MPI_Intercomm_merge (MPI_Comm intercomm, int high, MPI_Comm *newintracomm)

Fortran バインディング

call MPI_INTERCOMM_MERGE (INTERCOMM, HIGH, NEWINTRACOMM, IERROR)
整数型　　INTERCOMM, HIGH, NEWINTRACOMM, IERROR

MPI_Type_create_keyval (C)

MPI_TYPE_CREATE_KEYVAL (Fortran)

データ型に付加する属性の生成

基本構文

MPI_TYPE_CREATE_KEYVAL (type_copy_attr_fn, type_delete_attr_fn, type_keyval, extra_state)

引数	値	説明	IN/OUT
type_copy_attr_fn	関数	type_keyval に関する複製コールバック関数	IN
type_delete_attr_fn	関数	type_keyval に関する削除コールバック関数	IN
type_keyval	整数	今後のアクセスのためのキー値	OUT
extra_state	状態	コールバック関数の拡張状態	IN

C バインディング

```
int MPI_Type_create_keyval (MPI_Type_copy_attr_function *type_copy_attr_fn,
                            MPI_Type_delete_attr_function *type_delete_attr_fn, int
                            *type_keyval, void *extra_state)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_KEYVAL (TYPE_COPY_ATTR_FN, TYPE_DELETE_ATTR_FN,
                             TYPE_KEYVAL, EXTRA_STATE, IERROR)

EXTERNAL                                TYPE_COPY_ATTR_FN,
                                        TYPE_DELETE_ATTR_FN
INTEGER                                 TYPE_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) EXTRA_STATE
```

MPI_Type_delete_attr (C)　　MPI_TYPE_DELETE_ATTR (Fortran)

データ型に付加する属性の削除

基本構文

MPI_TYPE_DELETE_ATTR (type, type_keyval)

引数	値	説明	IN/OUT
type type_keyval	handle 整数	属性が削除されるデータ型 キー値	INOUT IN

C バインディング

```
int MPI_Type_delete_attr (MPI_Datatype type, int type_keyval)
```

Fortran バインディング

```
call MPI_TYPE_DELETE_ATTR (TYPE, TYPE_KEYVAL, IERROR)

    INTEGER          TYPE, TYPE_KEYVAL, IERROR
```

MPI_Type_free_keyval (C)　　MPI_TYPE_FREE_KEYVAL (Fortran)

データ型に付加する属性の解放

基本構文

MPI_TYPE_FREE_KEYVAL (type_keyval)

引数	値	説明	IN/OUT
type_keyval	整数	キー値	INOUT

C バインディング

```
int MPI_Type_free_keyval (int *type_keyval)
```

Fortran バインディング

```
call MPI_TYPE_FREE_KEYVAL (TYPE_KEYVAL, IERROR)

    INTEGER          TYPE_KEYVAL, IERROR
```

MPI_Type_get_attr (C) MPI_TYPE_GET_ATTR (Fortran)

データ型に付加された属性値の取得

基本構文

MPI_TYPE_GET_ATTR (type, type_keyval, attribute_val, flag)

引数	値	説明	IN/OUT
type	handle	属性が結び付けられたデータ型	IN
type_keyval	整数	キー値	IN
attribute_val	属性	属性値	OUT
flag	論理	フラグ	OUT

C バインディング

int MPI_Type_get_attr (MPI_Datatype type, int type_keyval, void *attribute_val, int *flag)

Fortran バインディング

call MPI_TYPE_GET_ATTR (TYPE, TYPE_KEYVAL, ATTRIBUTE_VAL, FLAG, IERROR)

```
INTEGER                                 TYPE, TYPE_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL
LOGICAL                                 FLAG
```

MPI_Type_get_name (C) MPI_TYPE_GET_NAME (Fortran)

データ型に付加された名前の取得

基本構文

MPI_TYPE_GET_NAME (type, type_name, resultlen)

引数	値	説明	IN/OUT
type	handle	名前が返却されるデータ型	IN
type_name	文字	データ型に設定された名前(文字列)	OUT
resultlen	整数	返却された名前の長さ	OUT

C バインディング

int MPI_Type_get_name (MPI_Datatype type, char *type_name, int *resultlen)

Fortran バインディング

call MPI_TYPE_GET_NAME (TYPE, TYPE_NAME, RESULTLEN, IERROR)

```
INTEGER          TYPE, RESULTLEN, IERROR
CHARACTER*(*)    TYPE_NAME
```

MPI_Type_set_attr (C)　　MPI_TYPE_SET_ATTR (Fortran)

データ型に対する属性の付加

基本構文

MPI_TYPE_SET_ATTR (type, type_keyval, attribute_val)

引数	値	説明	IN/OUT
type	handle	属性を結びつけるデータ型	INOUT
type_keyval	整数	キー値	IN
attribute_val	属性	属性値	IN

C バインディング

int MPI_Type_set_attr (MPI_Datatype type, int type_keyval, void *attribute_val)

Fortran バインディング

call MPI_TYPE_SET_ATTR (TYPE, TYPE_KEYVAL, ATTRIBUTE_VAL, IERROR)

```
INTEGER                                 TYPE, TYPE_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL
```

MPI_Type_set_name (C)　　MPI_TYPE_SET_NAME (Fortran)

データ型に対する名前の付加

基本構文

MPI_TYPE_SET_NAME (type, type_name)

引数	値	説明	IN/OUT
type	handle	識別子を設定するデータ型	INOUT
type_name	文字	名前として記憶される文字列	IN

C バインディング

int MPI_Type_set_name (MPI_Datatype type, char *type_name)

Fortran バインディング

call MPI_TYPE_SET_NAME (TYPE, TYPE_NAME, IERROR)

```
INTEGER             TYPE, IERROR
CHARACTER*(*)       TYPE_NAME
```

MPI_Win_create_keyval (C) | MPI_WIN_CREATE_KEYVAL (Fortran)

ウィンドウに付加する属性の生成

基本構文

MPI_WIN_CREATE_KEYVAL (win_copy_attr_fn, win_delete_attr_fn, win_keyval, extra_state)

引数	値	説明	IN/OUT
win_copy_attr_fn	関数	win_keyval に関する複製コールバック関数	IN
win_delete_attr_fn	関数	win_keyval に関する削除コールバック関数	IN
win_keyval	整数	今後のアクセスのためのキー値	OUT
extra_state	状態	コールバック関数の拡張状態	IN

C バインディング

```
int MPI_Win_create_keyval (MPI_Win_copy_attr_function *win_copy_attr_fn,
                           MPI_Win_delete_attr_function *win_delete_attr_fn, int
                           *win_keyval, void *extra_state)
```

Fortran バインディング

```
call MPI_WIN_CREATE_KEYVAL (WIN_COPY_ATTR_FN, WIN_DELETE_ATTR_FN,
                            WIN_KEYVAL, EXTRA_STATE, IERROR)

EXTERNAL                                WIN_COPY_ATTR_FN,
                                        WIN_DELETE_ATTR_FN
INTEGER                                 WIN_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND)          EXTRA_STATE
```

MPI_Win_delete_attr (C)　　MPI_WIN_DELETE_ATTR (Fortran)

ウィンドウに付加する属性の削除

基本構文

MPI_WIN_DELETE_ATTR (win, win_keyval)

引数	値	説明	IN/OUT
win	handle	属性が削除されるウィンドウ	INOUT
win_keyval	整数	キー値	IN

C バインディング

int MPI_Win_delete_attr (MPI_Win win, int win_keyval)

Fortran バインディング

call MPI_WIN_DELETE_ATTR (WIN, WIN_KEYVAL, IERROR)

INTEGER　　WIN, WIN_KEYVAL, IERROR

MPI_Win_free_keyval (C)　　MPI_WIN_FREE_KEYVAL (Fortran)

ウィンドウに付加する属性の解放

基本構文

MPI_WIN_FREE_KEYVAL (win_keyval)

引数	値	説明	IN/OUT
win_keyval	整数	キー値	INOUT

C バインディング

int MPI_Win_free_keyval (int *win_keyval)

Fortran バインディング

call MPI_WIN_FREE_KEYVAL (WIN_KEYVAL, IERROR)

INTEGER　　WIN_KEYVAL, IERROR

MPI_Win_get_attr (C) MPI_WIN_GET_ATTR (Fortran)

ウィンドウに付加された属性値の取得

基本構文

MPI_WIN_GET_ATTR (win, win_keyval, attribute_val, flag)

引数	値	説明	IN/OUT
win	handle	属性が結び付けられたウィンドウ	IN
win_keyval	整数	キー値	IN
attribute_val	属性	属性値	OUT
flag	論理	フラグ	OUT

C バインディング

int MPI_Win_get_attr (MPI_Win win, int win_keyval, void *attribute_val, int *flag)

Fortran バインディング

call MPI_WIN_GET_ATTR (WIN, WIN_KEYVAL, ATTRIBUTE_VAL, FLAG, IERROR)

INTEGER WIN, WIN_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL
LOGICAL FLAG

MPI_Win_get_name (C) MPI_WIN_GET_NAME (Fortran)

ウィンドウに付加された名前の取得

基本構文

MPI_WIN_GET_NAME (win, win_name, resultlen)

引数	値	説明	IN/OUT
win	handle	名前が返却されるウィンドウ	IN
win_name	文字	ウィンドウに設定された名前(文字列)	OUT
resultlen	整数	返却された名前の長さ	OUT

C バインディング

int MPI_Win_get_name (MPI_Win win, char *win_name, int *resultlen)

Fortran バインディング

call MPI_WIN_GET_NAME (WIN, WIN_NAME, RESULTLEN, IERROR)

```
INTEGER          WIN, RESULTLEN, IERROR
CHARACTER*(*)    WIN_NAME
```

MPI_Win_set_attr (C) MPI_WIN_SET_ATTR (Fortran)

ウィンドウに対する属性の付加

基本構文

MPI_WIN_SET_ATTR (win, win_keyval, attribute_val)

引数	値	説明	IN/OUT
win	handle	属性を結びつけるウィンドウ	INOUT
win_keyval	整数	キー値	IN
attribute_val	属性	属性値	IN

C バインディング

int MPI_Win_set_attr (MPI_Win win, int win_keyval, void *attribute_val)

Fortran バインディング

call MPI_WIN_SET_ATTR (WIN, WIN_KEYVAL, ATTRIBUTE_VAL, IERROR)

```
INTEGER                                 WIN, WIN_KEYVAL, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) ATTRIBUTE_VAL
```

MPI_Win_set_name (C) MPI_WIN_SET_NAME (Fortran)

ウィンドウに対する名前の付加

基本構文

MPI_WIN_SET_NAME (win, win_name)

引数	値	説明	IN/OUT
win	handle	識別子を設定するウィンドウ	INOUT
win_name	文字	名前として記憶される文字列	IN

C バインディング

int MPI_Win_set_name (MPI_Win win, char *win_name)

Fortran バインディング

```
call MPI_WIN_SET_NAME (WIN, WIN_NAME, IERROR)

    INTEGER           WIN, IERROR
    CHARACTER*(*)     WIN_NAME
```

4.5 プロセストポロジー

MPI_CART_COORDS	MPI_CART_CREATE	MPI_CART_GET
MPI_CART_MAP	MPI_CART_RANK	MPI_CART_SHIFT
MPI_CART_SUB	MPI_CARTDIM_GET	MPI_DIMS_CREAT
MPI_DIST_GRAPH_CREATE	MPI_DIST_GRAPH_CREATE_ADJACENT	MPI_DIST_GRAPH_NEIGHBORS
MPI_DIST_GRAPH_NEIGHBORS_COUNT	MPI_GRAPH_CREATE	MPI_GRAPH_GET
MPI_GRAPH_MAP	MPI_GRAPH_NEIGHBORS	MPI_GRAPH_NEIGHBORS_COUNT
MPI_GRAPHDIMS_GET	MPI_INEIGHBOR_ALLGATHER	MPI_INEIGHBOR_ALLGATHERV
MPI_INEIGHBOR_ALLTOALL	MPI_INEIGHBOR_ALLTOALLV	MPI_INEIGHBOR_ALLTOALLW
MPI_NEIGHBOR_ALLGATHER	MPI_NEIGHBOR_ALLGATHERV	MPI_NEIGHBOR_ALLTOALL
MPI_NEIGHBOR_ALLTOALLV	MPI_NEIGHBOR_ALLTOALLW	MPI_TOPO_TEST

MPI_Cart_coords (C) MPI_CART_COORDS (Fortran)

ランクからデカルト座標への変換

基本構文

MPI_CART_COORDS (comm, rank, maxdims, coords)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
rank	整数	ランク	IN
maxdims	整数	配列 coords の大きさ	IN
coords	整数	変換結果の座標	OUT

C バインディング

```
int MPI_Cart_coords (MPI_Comm comm, int rank, int maxdims, int *coords)
```

Fortran バインディング

```
call MPI_CART_COORDS (COMM, RANK, MAXDIMS, COORDS, IERROR)
    整数型          COMM, RANK, MAXDIMS, COORDS(*), IERROR
```

デカルトトポロジーをもつコミュニケーターの生成

基本構文

MPI_CART_CREATE (comm_old, ndims, dims, periods, reorder, comm_cart)

引数	値	説明	IN/OUT
comm_old	handle	元となるコミュニケーター	IN
ndims	整数	次元数	IN
dims	整数	各次元の大きさ	IN
periods	論理	各次元が periodic か否かを示すフラグ	IN
reorder	論理	新規グループ中でプロセスのランク付けを変更するか否かを示すフラグ	IN
comm_cart	handle	新たに生成されたコミュニケーター	OUT

C バインディング

```
int MPI_Cart_create (MPI_Comm comm_old, int ndims, int *dims, int *periods, int reorder,
                     MPI_Comm *comm_cart)
```

Fortran バインディング

```
call MPI_CART_CREATE (COMM_OLD, NDIMS, DIMS, PERIODS, REORDER,
                      COMM_CART, IERROR)
    整数型          COMM_OLD, NDIMS, DIMS(*), COMM_CART, IERROR
    論理型          PERIODS(*), REORDER
```

MPI_Cart_get (C) MPI_CART_GET (Fortran)

デカルトトポロジー情報の取得

基本構文

MPI_CART_GET (comm, maxdims, dims, periods, coords)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
maxdims	整数	配列 dims, periods, coords の大きさ	IN
dims	整数	各次元のプロセス数	OUT
periods	論理	各次元が periodic か否か	OUT
coords	整数	呼出しプロセスの存在する座標	OUT

C バインディング

int MPI_Cart_get (MPI_Comm comm, int maxdims, int *dims, int *periods, int *coords)

Fortran バインディング

call MPI_CART_GET (COMM, MAXDIMS, DIMS, PERIODS, COORDS, IERROR)
整数型 COMM, MAXDIMS, DIMS(*), COORDS(*), IERROR
論理型 PERIODS(*)

MPI_Cart_map (C) MPI_CART_MAP (Fortran)

デカルトトポロジーにおける自身のランクの計算(低レベルトポロジー関数)

基本構文

MPI_CART_MAP (comm, ndims, dims, periods, newrank)

引数	値	説明	IN/OUT
comm	handle	元となるコミュニケーター	IN
ndims	整数	次元数	IN
dims	整数	各次元の大きさ	IN
periods	論理	各次元が periodic か否かを示すフラグ	IN
newrank	整数	自分自身のランク	OUT

C バインディング

int MPI_Cart_map (MPI_Comm comm, int ndims, int *dims, int *periods, int *newrank)

Fortran バインディング

call MPI_CART_MAP (COMM, NDIMS, DIMS, PERIODS, NEWRANK, IERROR)
整数型 COMM, NDIMS, DIMS(*), NEWRANK, IERROR
論理型 PERIODS(*)

MPI_Cart_rank (C)　　MPI_CART_RANK (Fortran)

座標位置からランクへの変換

基本構文

MPI_CART_RANK (comm, coords, rank)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
coords	整数	座標	IN
rank	整数	ランク	OUT

C バインディング

```
int MPI_Cart_rank (MPI_Comm comm, int *coords, int *rank)
```

Fortran バインディング

```
call MPI_CART_RANK (COMM, COORDS, RANK, IERROR)
    整数型          COMM, COORDS, RANK, IERROR
```

MPI_Cart_shift (C)　　MPI_CART_SHIFT (Fortran)

デカルトトポロジ座標における，指定された次元 および 方向の隣接プロセスのランクを返す。

基本構文

MPI_CART_SHIFT (comm, direction, disp, rank_source, rank_dest)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
direction	整数	シフトする座標軸(0 から次元数-1 で指定)	IN
disp	整数	シフト方向(>0:正方向,<0:負方向)	IN
rank_source	整数	データを受信するプロセスのランク	OUT
rank_dest	整数	データを送信するプロセスのランク	OUT

C バインディング

```
int MPI_Cart_shift (MPI_Comm comm, int direction, int disp, int *rank_source, int
                    *rank_dest)
```

Fortran バインディング

```
call MPI_CART_SHIFT (COMM, DIRECTION, DISP, RANK_SOURCE, RANK_DEST,
                     IERROR)
    整数型          COMM, DIRECTION, DISP, RANK_SOURCE, RANK_DEST, IERROR
```

MPI_Cart_sub (C)　　MPI_CART_SUB (Fortran)

デカルト構造の分割

基本構文

MPI_CART_SUB (comm, remain_dims, newcomm)

引数	値	説明	IN/OUT
comm	handle	分割対象コミュニケーター	IN
remain_dims	論理	分割する座標の指定	IN
newcomm	handle	分割されたデカルトトポロジーをもつコミュニケーター	OUT

C バインディング

int MPI_Cart_sub (MPI_Comm comm, int *remain_dims, MPI_Comm *newcomm)

Fortran バインディング

call MPI_CART_SUB (COMM, REMAIN_DIMS, NEWCOMM, IERROR)
整数型　　COMM, NEWCOMM, IERROR
論理型　　REMAIN_DIMS(*)

MPI_Cartdim_get (C)　　MPI_CARTDIM_GET (Fortran)

デカルトトポロジー情報の取得

基本構文

MPI_CARTDIM_GET (comm, ndims)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
ndims	整数	次元数	OUT

C バインディング

int MPI_Cartdim_get (MPI_Comm comm, int *ndims)

Fortran バインディング

call MPI_CARTDIM_GET (COMM, NDIMS, IERROR)
整数型　　COMM, NDIMS, IERROR

各次元の大きさのバランスを考慮したデカルトトポロジー選択

基本構文

MPI_DIMS_CREATE (nnodes, ndims, dims)

引数	値	説明	IN/OUT
nnodes	整数	ノード数	IN
ndims	整数	次元数	IN
dims	整数	各次元の大きさ	INOUT

C バインディング

```
int MPI_Dims_create (int nnodes, int ndims, int *dims)
```

Fortran バインディング

```
call MPI_DIMS_CREATE (NNODES, NDIMS, DIMS, IERROR)
    整数型          NNODES, NDIMS, DIMS(*), IERROR
```

分散グラフトポロジーをもつコミュニケーターの生成

基本構文

MPI_DIST_GRAPH_CREATE (comm_old, n, sources, degrees, destinations, weights, info, reorder, comm_dist_graph)

引数	値	説明	IN/OUT
comm_old	handle	元となるコミュニケーター	IN
n	整数	グラフのノード数	IN
sources	整数	ソースノードリスト	IN
degrees	整数	ソースノードごとのデスティネーションノード数リスト	IN
destinations	整数	デスティネーションノードリスト	IN
weights	整数	グラフの重みづけリスト	IN
info	handle	グラフ最適化のための info オブジェクト	IN
reorder	論理	新規グループ中でプロセスのランク付けを変更するか否かを示すフラグ	IN
com_dist_graph	handle	新たに生成されたコミュニケーター	OUT

C バインディング

```
int MPI_Dist_graph_create (MPI_Comm comm_old, int n, int sources[], int degrees[], int
                           destinations[],
                           int weights[], MPI_Info info, int reorder, MPI_Comm
                           *comm_dist_graph)
```

Fortran バインディング

```
call MPI_DIST_GRAPH_CREATE (COMM_OLD, N, SOURCES, DEGREES, DESTINATIONS,
                            WEIGHTS, INFO, REORDER, COMM_DIST_GRAPH,
                            IERROR)
```

整数型	COMM_OLD, N, SOURCES(*), DEGREES(*), DESTINATIONS(*), WEIGHTS(*), INFO, COMM_DIST_GRAPH, IERROR
論理型	REORDER

分散グラフトポロジーをもつコミュニケーターの生成

基本構文

MPI_DIST_GRAPH_CREATE_ADJACENT (comm_old, indegree, sources, sourceweights, outdegree, destinations, destweights, info, reorder, comm_dist_graph)

引数	値	説明	IN/OUT
comm_old	handle	元となるコミュニケーター	IN
indegree	整数	ソースノード数	IN
sources	整数	ソースノードリスト	IN
sourceweights,	整数	ソースノードごとのグラフの重みづけリスト	IN
outdegree	整数	デスティネーションノード数	IN
destinations	整数	デスティネーションノードリスト	IN
destweights	整数	デスティネーションノードごとのグラフの重みづけリスト	IN
info	handle	グラフ最適化のための info オブジェクト	IN
reorder	論理	新規グループ中でプロセスのランク付けを変更するか否かを示すフラグ	IN
com_dist_graph	handle	新たに生成されたコミュニケーター	OUT

C バインディング

```
int MPI_Dist_graph_create_adjacent (MPI_Comm comm_old, int indegree, int *sources, int
                                    *sourceweights,
                                    int outdegree, int *destinations, int *destweights,
                                    MPI_Info info,
                                    int reorder, MPI_Comm *comm_dist_graph)
```

Fortran バインディング

```
call MPI_DIST_GRAPH_CREATE_ADJACENT (COMM_OLD, INDEGREE, SOURCES,
                                     SOURCEWEIGHTS, OUTDEGREE,
                                     DESTINATIONS, DESTWEIGHTS, INFO,
                                     REORDER, COMM_DIST_GRAPH, IERROR)
```

整数型	COMM_OLD, INDEGREE, SOURCES(*), SOURCEWEIGHTS(*), OUTDEGREE, DESTINATIONS(*), DESTWEIGHTS(*), INFO, COMM_DIST_GRAPH, IERROR
論理型	REORDER

分散グラフトポロジーをもつコミュニケーターの隣接プロセス情報問合せ

基本構文

MPI_DIST_GRAPH_NEIGHBORS (comm, maxindegree, sources, sourceweights, maxoutdegree, destinations, destweights)

引数	値	説明	IN/OUT
comm	handle	分散グラフ トポロジーをもつコミュニケーター	IN
maxindegree	整数	ソースノード数	IN
sources	整数	ソースノードリスト	OUT
sourceweights,	整数	ソースノードごとのグラフの重みづけリスト	OUT
maxoutdegree	整数	デスティネーションノード数	IN
destinations	整数	デスティネーションノードリスト	OUT
destweights	整数	デスティネーションノードごとのグラフの重みづけリスト	OUT

C バインディング

```
int MPI_Dist_graph_neighbors (MPI_Comm comm, int maxindegree, int *sources, int
                              *sourceweights,
                              int maxoutdegree, int *destinations, int *destweights)
```

Fortran バインディング

```
call MPI_DIST_GRAPH_NEIGHBORS (COMM, MAXINDEGREE, SOURCES,
                               SOURCEWEIGHTS, MAXOUTDEGREE,
                               DESTINATIONS, DESTWEIGHTS)
```

整数型 COMM, MAXINDEGREE, SOURCES(*), SOURCEWEIGHTS(*), MAXOUTDEGREE, DESTINATIONS(*), DESTWEIGHTS(*), IERROR

分散グラフトポロジーをもつコミュニケーターの隣接プロセス問合せ

基本構文

MPI_DIST_GRAPH_NEIGHBORS_COUNT (comm, indegree, outdegree, weighted)

引数	値	説明	IN/OUT
comm	handle	分散グラフ トポロジーをもつコミュニケーター	IN
indegree	整数	ソースノード数	OUT
outdegree	整数	デスティネーションノード数	OUT
weighted	論理	グラフに重みづけが設定されているか否か	OUT

C バインディング

```
int MPI_Dist_graph_neighbors_count (MPI_Comm comm, int *indegree, int *outdegree, int
*weighted)
```

Fortran バインディング

```
call MPI_DIST_GRAPH_NEIGHBORS_COUNT (COMM, INDEGREE, OUTDEGREE,
                                     WEIGHTED)
```

整数型　　COMM, INDEGREE, OUTDEGREE, IERROR

論理型　　WEIGHTED

グラフトポロジーをもつコミュニケーターの生成

基本構文

MPI_GRAPH_CREATE (comm_old, nnodes, index, edges, reorder, comm_graph)

引数	値	説明	IN/OUT
comm_old	handle	元となるコミュニケーター	IN
nnodes	整数	グラフのノード数	IN
index	整数	グラフ情報(累積エッジ数,edges のインデックス)	IN
edges	整数	グラフ情報(エッジリストの１次元表現)	IN
reorder	論理	新規グループ中でプロセスのランク付けを変更するか否かを示すフラグ	IN
comm_graph	handle	新たに生成されたコミュニケーター	OUT

C バインディング

```
int MPI_Graph_create (MPI_Comm comm_old, int nnodes, int *index, int *edges, int reorder,
                      MPI_Comm *comm_graph)
```

Fortran バインディング

```
call MPI_GRAPH_CREATE (COMM_OLD, NNODES, INDEX, EDGES, REORDER,
                       COMM_GRAPH, IERROR)
```

整数型	COMM_OLD, NNODES, INDEX(*), EDGES(*), COMM_GRAPH, IERROR
論理型	REORDER

MPI_Graph_get (C)　　MPI_GRAPH_GET (Fortran)

グラフトポロジー情報の取得

基本構文

MPI_GRAPH_GET (comm, maxindex, maxedges, index, edges)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
maxindex	整数	配列 index の大きさ	IN
maxedges	整数	配列 edges の大きさ	IN
index	整数	グラフ情報(累積エッジ数,edges のインデックス)	OUT
edges	整数	グラフ情報(エッジリストの 1 次元表現)	OUT

C バインディング

int MPI_Graph_get (MPI_Comm comm, int maxindex, int maxedges, int *index, int *edges)

Fortran バインディング

call MPI_GRAPH_GET (COMM, MAXINDEX, MAXEDGES, INDEX, EDGES, IERROR)
整数型　　COMM, MAXINDEX, MAXEDGES, INDEX(*), EDGES(*), IERROR

MPI_Graph_map (C)　　MPI_GRAPH_MAP (Fortran)

グラフトポロジーにおける自身のランクの計算(低レベルトポロジー関数)

基本構文

MPI_GRAPH_MAP (comm, nnodes, index, edges, newrank)

引数	値	説明	IN/OUT
comm	handle	元となるコミュニケーター	IN
nnodes	整数	グラフのノード数	IN
index	整数	グラフ情報(累積エッジ数,edges のインデックス)	IN
edges	整数	グラフ情報(エッジリストの 1 次元表現)	IN
newrank	整数	自分自身のランク	OUT

C バインディング

int MPI_Graph_map (MPI_Comm comm, int nnodes, int *index, int *edges, int *newrank)

Fortran バインディング

call MPI_GRAPH_MAP (COMM, NNODES, INDEX, EDGES, NEWRANK, IERROR)
整数型　　COMM, NNODES, INDEX(*), EDGES(*), NEWRANK, IERROR

MPI_Graph_neighbors (C)

MPI_GRAPH_NEIGHBORS (Fortran)

グラフ上での近隣プロセス情報取得

基本構文

MPI_GRAPH_NEIGHBORS (comm, rank, maxneighbors, neighbors)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
rank	整数	指定プロセスのランク	IN
maxneighbors	整数	配列 neighbors の大きさ	IN
neighbors	整数	指定されたプロセスの近隣プロセスのランク	OUT

C バインディング

int MPI_Graph_neighbors (MPI_Comm comm, int rank, int maxneighbors, int *neighbors)

Fortran バインディング

call MPI_GRAPH_NEIGHBORS (COMM, RANK, MAXNEIGHBORS, NEIGHBORS, IERROR)
整数型 COMM, RANK, MAXNEIGHBORS, NEIGHBORS, IERROR

MPI_Graph_neighbors_count (C)

MPI_GRAPH_NEIGHBORS_COUNT (Fortran)

グラフ上での近隣プロセス数

基本構文

MPI_GRAPH_NEIGHBORS_COUNT (comm, rank, nneighbors)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
rank	整数	指定プロセスのランク	IN
nneighbors	整数	指定されたプロセスの近隣プロセス数	OUT

C バインディング

int MPI_Graph_neighbors_count (MPI_Comm comm, int rank, int *nneighbors)

Fortran バインディング

call MPI_GRAPH_NEIGHBORS_COUNT (COMM, RANK, NNEIGHBORS, IERROR)
整数型 COMM, RANK, NNEIGHBORS, IERROR

グラフトポロジー情報の取得

基本構文

MPI_GRAPHDIMS_GET (comm, nnodes, nedges)

引数	値	説明	IN/OUT
comm	handle	対象コミュニケーター	IN
nnodes	整数	グラフ中のノード数(グループ中のプロセス数と等しい)	OUT
nedges	整数	グラフ中のエッジ数	OUT

C バインディング

int MPI_Graphdims_get (MPI_Comm comm, int *nnodes, int *nedges)

Fortran バインディング

call MPI_GRAPHDIMS_GET (COMM, NNODES, NEDGES, IERROR)
整数型 COMM, NNODES, NEDGES, IERROR

隣接プロセス間でのデータの収集 (非ブロッキング版)

基本構文

MPI_INEIGHBOR_ALLGATHER (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	個々のプロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ineighbor_allgather (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_INEIGHBOR_ALLGATHER (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR

隣接プロセス間でのデータの収集 (非ブロッキング版)

基本構文

MPI_INEIGHBOR_ALLGATHERV (sendarea, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信バッファの要素の個数	IN
displs	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ineighbor_allgatherv (void* sendarea, int sendcount, MPI_Datatype sendtype, void*
                              recvarea, int *recvcounts, int *displs, MPI_Datatype recvtype,
                              MPI_Comm comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_INEIGHBOR_ALLGATHERV (SENDAREA, SENDCOUNT, SENDTYPE,
                               RECVAREA, RECVCOUNT, DISPLS, RECVTYPE,
                               COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, COMM, REQUEST, IERROR

隣接プロセス間での隣接プロセス間でのデータの収集と拡散(非ブロッキング版)

基本構文

MPI_INEIGHBOR_ALLTOALL (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	各プロセスへ送信する要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

int MPI_Ineighbor_alltoall (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm comm, MPI_Request *request)

Fortran バインディング

call MPI_INEIGHBOR_ALLTOALL (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, REQUEST, IERROR

隣接プロセス間でのデータの収集と拡散（非ブロッキング版）

基本構文

MPI_INEIGHBOR_ALLTOALLV (sendarea, sendcounts, sdispls, sendtype, recvarea, recvcounts, rdispls, recvtype, comm, request)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置	IN
sendtype	handle	送信バッファのデータ型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素のデータ型	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ineighbor_alltoallv (void* sendarea, int *sendcounts, int *sdispls, MPI_Datatype
                             sendtype, void* recvarea, int *recvcounts, int *rdispls,
                             MPI_Datatype recvtype, MPI_Comm comm, MPI_Request
                             *request)
```

Fortran バインディング

```
call MPI_INEIGHBOR_ALLTOALLV (SENDAREA, SENDCOUNTS, SDISPLS, SENDTYPE,
                              RECVAREA, RECVCOUNTS, RDISPLS, RECVTYPE,
                              COMM, REQUEST, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPE, RECVCOUNTS(*), RDISPLS(*), RECVTYPE, COMM, REQUEST, IERROR

隣接プロセス間でのデータの収集と拡散（非ブロッキング版）

基本構文

MPI_INEIGHBOR_ALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounts, rdispls, recvtypes, comm, request)

引数	値	説明	IN/OUT
sendbuf	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置 (バイト単位)	IN
sendtypes	handle	送信バッファのデータ型(プロセスごと)	IN
recvbuf	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置 (バイト単位)	IN
recvtypes	handle	受信バッファの要素のデータ型(プロセスごと)	IN
comm	handle	コミュニケーター	IN
request	handle	通信要求	OUT

C バインディング

```
int MPI_Ineighbor_alltoallw (void *sendbuf, int *sendcounts, MPI_Aint *sdispls,
                             MPI_Datatype *sendtypes, void *recvbuf,
                             int *recvcounts, MPI_Aint *rdispls, MPI_Datatype *recvtypes,
                             MPI_Comm comm, MPI_Request *request)
```

Fortran バインディング

```
call MPI_INEIGHBOR_ALLTOALLV (SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES,
                              RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPES,
                              COMM, REQUEST, IERROR)
```

任意の型	SENDBUF(*), RECVBUF(*)
INTEGER	SENDCOUNTS(*),SENDTYPES(*), RECVCOUNTS(*),RECVTYPES(*), COMM, REQUEST, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND)	SDIPSLS(*), RDISPLS(*)

隣接プロセス間でのデータの収集

基本構文

MPI_NEIGHBOR_ALLGATHER (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	個々のプロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Neighbor_allgather (void* sendarea, int sendcount, MPI_Datatype sendtype, void*
                            recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm
                            comm)
```

Fortran バインディング

```
call MPI_NEIGHBOR_ALLGATHER (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA,
                             RECVCOUNT, RECVTYPE, COMM, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

隣接プロセス間でのデータの収集

基本構文

MPI_NEIGHBOR_ALLGATHERV (sendarea, sendcount, sendtype, recvarea, recvcounts, displs, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	送信バッファの要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信バッファの要素の個数	IN
displs	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Neighbor_allgatherv (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int *recvcounts, int *displs, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_NEIGHBOR_ALLGATHERV (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, DISPLS, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNTS(*), DISPLS(*), RECVTYPE, COMM, IERROR

隣接プロセス間での隣接プロセス間でのデータの収集と拡散

基本構文

MPI_NEIGHBOR_ALLTOALL (sendarea, sendcount, sendtype, recvarea, recvcount, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcount	整数	各プロセスへ送信する要素の個数	IN
sendtype	handle	送信バッファの要素の型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcount	整数	各プロセスから受信する要素の個数	IN
recvtype	handle	受信バッファの要素の型	IN
comm	handle	コミュニケーター	IN

C バインディング

int MPI_Neighbor_alltoall (void* sendarea, int sendcount, MPI_Datatype sendtype, void* recvarea, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)

Fortran バインディング

call MPI_NEIGHBOR_ALLTOALL (SENDAREA, SENDCOUNT, SENDTYPE, RECVAREA, RECVCOUNT, RECVTYPE, COMM, IERROR)

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNT, SENDTYPE, RECVCOUNT, RECVTYPE, COMM, IERROR

隣接プロセス間での隣接プロセス間でのデータの収集と拡散

基本構文

MPI_NEIGHBOR_ALLTOALLV (sendarea, sendcounts, sdispls, sendtype, recvarea, recvcounts, rdispls, recvtype, comm)

引数	値	説明	IN/OUT
sendarea	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置	IN
sendtype	handle	送信バッファのデータ型	IN
recvarea	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置	IN
recvtype	handle	受信バッファの要素のデータ型	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Neighbor_alltoallv (void* sendarea, int *sendcounts, int *sdispls, MPI_Datatype
                            sendtype, void* recvarea, int *recvcounts, int *rdispls,
                            MPI_Datatype recvtype, MPI_Comm comm)
```

Fortran バインディング

```
call MPI_NEIGHBOR_ALLTOALLV (SENDAREA, SENDCOUNTS, SDISPLS, SENDTYPE,
                             RECVAREA, RECVCOUNTS, RDISPLS, RECVTYPE,
                             COMM, IERROR)
```

任意の型	SENDAREA(*), RECVAREA(*)
整数型	SENDCOUNTS(*), SDIPSLS(*), SENDTYPE, RECVCOUNTS(*), RDISPLS(*), RECVTYPE, COMM, IERROR

隣接プロセス間でのデータの収集と拡散

基本構文

MPI_NEIGHBOR_ALLTOALLW (sendbuf, sendcounts, sdispls, sendtypes, recvbuf, recvcounts, rdispls, recvtypes, comm)

引数	値	説明	IN/OUT
sendbuf	任意	送信バッファの先頭アドレス	IN
sendcounts	整数	送信する要素の個数(プロセスごと)	IN
sdispls	整数	各プロセスへの送信データの先頭位置(バイト単位)	IN
sendtypes	handle	送信バッファのデータ型(プロセスごと)	IN
recvbuf	任意	受信バッファの先頭アドレス	OUT
recvcounts	整数	受信する要素の個数(プロセスごと)	IN
rdispls	整数	各プロセスからの受信データの先頭位置(バイト単位)	IN
recvtypes	handle	受信バッファの要素のデータ型(プロセスごと)	IN
comm	handle	コミュニケーター	IN

C バインディング

```
int MPI_Neighbor_alltoallw (void *sendbuf, int *sendcounts, MPI_Aint *sdispls, MPI_Datatype
                            *sendtypes, void *recvbuf,
                            int *recvcounts, MPI_Aint *rdispls, MPI_Datatype *recvtypes,
                            MPI_Comm comm)
```

Fortran バインディング

```
call MPI_NEIGHBOR_ALLTOALLV (SENDBUF, SENDCOUNTS, SDISPLS, SENDTYPES,
                             RECVBUF, RECVCOUNTS, RDISPLS, RECVTYPES,
                             COMM, IERROR)
```

任意の型	SENDBUF(*), RECVBUF(*)
INTEGER	SENDCOUNTS(*),SENDTYPES(*), RECVCOUNTS(*),RECVTYPES(*), COMM, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND)	SDIPSLS(*), RDISPLS(*)

トポロジーの問合せ

基本構文

MPI_TOPO_TEST (comm, status)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
status	choice	問合せ結果(トポロジータイプ)	OUT

C バインディング

```
int MPI_Topo_test (MPI_Comm comm, int *status)
```

Fortran バインディング

```
call MPI_TOPO_TEST (COMM, STATUS, IERROR)
    整数型          COMM, STATUS, IERROR
```

4.6 実行管理

MPI_WTICK	MPI_WTIME	MPI_ABORT
MPI_ADD_ERROR_CLASS	MPI_ADD_ERROR_CODE	MPI_ADD_ERROR_STRING
MPI_ALLOC_MEM	MPI_COMM_CALL_ERRHANDLER	MPI_COMM_CREATE_ERRHANDLER
MPI_COMM_GET_ERRHANDLER	MPI_COMM_SET_ERRHANDLER	MPI_ERRHANDLER_FREE
MPI_ERROR_CLASS	MPI_ERROR_STRING	MPI_FILE_CALL_ERRHANDLER
MPI_FILE_CREATE_ERRHANDLER	MPI_FILE_GET_ERRHANDLER	MPI_FILE_SET_ERRHANDLER
MPI_FINALIZE	MPI_FINALIZED	MPI_FREE_MEM
MPI_GET_LIBRARY_VERSION	MPI_GET_PROCESSOR_NAME	MPI_GET_VERSION
MPI_INIT	MPI_INITIALIZED	MPI_WIN_CALL_ERRHANDLER
MPI_WIN_CREATE_ERRHANDLER	MPI_WIN_GET_ERRHANDLER	MPI_WIN_SET_ERRHANDLER

MPI_Wtick (C) MPI_WTICK (Fortran)

関数 MPI_WTIME の分解能(秒単位)を返す

基本構文

MPI_WTICK ()

C バインディング

double MPI_Wtick (void)

Fortran バインディング

DOUBLE MPI_WTICK ( )

MPI_Wtime (C) MPI_WTIME (Fortran)

過去のある時点からの経過時間(秒単位)を返す

基本構文

MPI_WTIME ()

C バインディング

double MPI_Wtime (void)

Fortran バインディング

DOUBLE PRECESION MPI_WTIME ( )

MPI_Abort (C) MPI_ABORT (Fortran)

コミュニケーター中のプロセスのアボート

基本構文

MPI_ABORT (comm, errorcode)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
errorcode	整数	エラーコード	IN

C バインディング

int MPI_Abort (MPI_Comm comm, int errorcode)

Fortran バインディング

call MPI_ABORT (COMM, ERRORCODE, IERROR)
整数型 COMM, STATUS, IERROR

MPI_Add_error_class (C) MPI_ADD_ERROR_CLASS (Fortran)

新しいエラークラスの生成

基本構文

MPI_ADD_ERROR_CLASS (int *errorclass)

引数	値	説明	IN/OUT
errorclass	整数	新しいエラークラスのための値	OUT

C バインディング

int MPI_Add_error_class (int *errorclass)

Fortran バインディング

call MPI_ADD_ERROR_CLASS (ERRORCLASS, IERROR)

INTEGER ERRORCLASS, IERROR

MPI_Add_error_code (C) MPI_ADD_ERROR_CODE (Fortran)

新しいエラーコードの生成

基本構文

MPI_ADD_ERROR_CODE (errorclass, errorcode)

引数	値	説明	IN/OUT
errorclass errorcod	整数 整数	エラークラス errorclass に関連付けられた新しいエラーコード	IN OUT

C バインディング

int MPI_Add_error_code (int errorclass, int *errorcode)

Fortran バインディング

call MPI_ADD_ERROR_CODE (ERRORCLASS, ERRORCODE, IERROR)

INTEGER ERRORCLASS, ERRORCODE, IERROR

エラーコード または エラークラスに対するエラー文字列の関連付け

基本構文

MPI_ADD_ERROR_STRING (errorcode, string)

引数	値	説明	IN/OUT
errorcode	整数	エラーコード または エラークラス	IN
string	文字	errorcode に対応するテキスト(文字列)	IN

C バインディング

int MPI_Add_error_string (int errorcode, char *string)

Fortran バインディング

call MPI_ADD_ERROR_STRING (ERRORCODE, STRING, IERROR)

INTEGER ERRORCODE, IERROR
CHARACTER*(*) STRING

メモリの割当て

基本構文

MPI_ALLOC_MEM (size, info, baseptr)

引数	値	説明	IN/OUT
size	整数	メモリセグメントの大きさ(バイト単位)	IN
info	handle	info 引数	IN
baseptr	整数	割り当てられたメモリセグメントの先頭へのポインタ	OUT

C バインディング

int MPI_Alloc_mem (MPI_Aint size, MPI_Info info, void *baseptr)

Fortran バインディング

call MPI_ALLOC_MEM (SIZE, INFO, BASEPTR, IERROR)

INTEGER INFO, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR

MPI_Comm_call_errhandler (C) | MPI_COMM_CALL_ERRHANDLER (Fortran)

エラーコードによるエラーハンドラーの起動

基本構文

MPI_COMM_CALL_ERRHANDLER (comm, errorcode)

引数	値	説明	IN/OUT
comm	handle	エラーハンドラーをもつコミュニケーター	IN
errorcode	整数	エラーコード	IN

C バインディング

```
int MPI_Comm_call_errhandler (MPI_Comm comm, int errorcode)
```

Fortran バインディング

```
call MPI_COMM_CALL_ERRHANDLER (COMM, ERRORCODE, IERROR)

    INTEGER         COMM, ERRORCODE, IERROR
```

MPI_Comm_create_errhandler (C) | MPI_COMM_CREATE_ERRHANDLER (Fortran)

コミュニケーターに付加するエラーハンドラーの生成

基本構文

MPI_COMM_CREATE_ERRHANDLER (function, errhandler)

引数	値	説明	IN/OUT
function	関数	利用者定義のエラー処理手続	IN
errhandler	handle	MPI エラーハンドラー	OUT

C バインディング

```
int MPI_Comm_create_errhandler (MPI_Comm_errhandler_fn *function, MPI_Errhandler
                                *errhandler)
```

Fortran バインディング

```
call MPI_COMM_CREATE_ERRHANDLER (FUNCTION, ERRHANDLER, IERROR)

    EXTERNAL        FUNCTION
    INTEGER         ERRHANDLER, IERROR
```

MPI_Comm_get_errhandler (C) / MPI_COMM_GET_ERRHANDLER (Fortran)

コミュニケーターに現在付加されているエラーハンドラーの取得

基本構文

MPI_COMM_GET_ERRHANDLER (comm, errhandler)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
errhandler	handle	comm に現在付加されているエラーハンドラー	OUT

C バインディング

int MPI_Comm_get_errhandler (MPI_Comm comm, MPI_Errhandler *errhandler)

Fortran バインディング

call MPI_COMM_GET_ERRHANDLER (COMM, ERRHANDLER, IERROR)

INTEGER COMM, ERRHANDLER, IERROR

MPI_Comm_set_errhandler (C) / MPI_COMM_SET_ERRHANDLER (Fortran)

コミュニケーターに対するエラーハンドラーの付加

基本構文

MPI_COMM_SET_ERRHANDLER (comm, errhandler)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	INOUT
errhandler	handle	新しいエラーハンドラー	IN

C バインディング

int MPI_Comm_set_errhandler (MPI_Comm comm, MPI_Errhandler errhandler)

Fortran バインディング

call MPI_COMM_SET_ERRHANDLER (COMM, ERRHANDLER, IERROR)

INTEGER COMM, ERRHANDLER, IERROR

MPI_Errhandler_free (C) | MPI_ERRHANDLER_FREE (Fortran)

エラーハンドラーの解放

基本構文

MPI_ERRHANDLER_FREE (errhandler)

引数	値	説明	IN/OUT
errhandler	handle	エラーハンドラー	INOUT

C バインディング

int MPI_Errhandler_free (MPI_Errhandler *errhandler)

Fortran バインディング

call MPI_ERRHANDLER_FREE (ERRHANDLER, IERROR)
整数型 ERRHANDLER, IERROR

MPI_Error_class (C) | MPI_ERROR_CLASS (Fortran)

エラーコードからエラークラスへの変換

基本構文

MPI_ERROR_CLASS (errorcode, errorclass)

引数	値	説明	IN/OUT
errorcode errorclass	整数 整数	エラーコード エラークラス	IN OUT

C バインディング

int MPI_Error_class (int errorcode, int *errorclass)

Fortran バインディング

call MPI_ERROR_CLASS (ERRORCODE, ERRORCLASS, IERROR)
整数型 ERRORCODE, ERRORCLASS, IERROR

MPI_Error_string (C)　　MPI_ERROR_STRING (Fortran)

エラーコードからエラー文字列への変換

基本構文

MPI_ERROR_STRING (errorcode, string, resultlen)

引数	値	説明	IN/OUT
errorcode	整数	エラーコード	IN
string	文字	エラー文字列	OUT
resultlen	整数	エラー文字列の長さ	OUT

C バインディング

int MPI_Error_string (int errorcode, char *string, int *resultlen)

Fortran バインディング

call MPI_ERROR_STRING (ERRORCODE, STRING, RESULTLEN, IERROR)

整数型　　ERRORCODE, RESULTLEN, IERROR

文字型　　STRING

MPI_File_call_errhandler (C)　　MPI_FILE_CALL_ERRHANDLER (Fortran)

エラーコードによるエラーハンドラーの起動

基本構文

MPI_FILE_CALL_ERRHANDLER (fh, errorcode)

引数	値	説明	IN/OUT
fh	handle	エラーハンドルをもつファイル	IN
errorcode	整数	エラーコード	IN

C バインディング

int MPI_File_call_errhandler (MPI_File fh, int errorcode)

Fortran バインディング

call MPI_FILE_CALL_ERRHANDLER (FH, ERRORCODE, IERROR)

INTEGER　　FH, ERRORCODE, IERROR

MPI_File_create_errhandler (C) | MPI_FILE_CREATE_ERRHANDLER (Fortran)

ファイルに付加するエラーハンドラーの生成

基本構文

MPI_FILE_CREATE_ERRHANDLER (function, errhandler)

引数	値	説明	IN/OUT
function	関数	利用者定義のエラー処理手続	IN
errhandler	handle	MPI エラーハンドラー	OUT

C バインディング

```
int MPI_File_create_errhandler (MPI_File_errhandler_fn *function, MPI_Errhandler
                                *errhandler)
```

Fortran バインディング

```
call MPI_FILE_CREATE_ERRHANDLER (FUNCTION, ERRHANDLER, IERROR)

    EXTERNAL        FUNCTION
    INTEGER         ERRHANDLER, IERROR
```

MPI_File_get_errhandler (C) | MPI_FILE_GET_ERRHANDLER (Fortran)

ファイルに現在付加されているエラーハンドラーの取得

基本構文

MPI_FILE_GET_ERRHANDLER (file, errhandler)

引数	値	説明	IN/OUT
file	handle	ファイル	IN
errhandler	handle	file に現在付加されているエラーハンドラー	OUT

C バインディング

```
int MPI_File_get_errhandler (MPI_File file, MPI_Errhandler *errhandler)
```

Fortran バインディング

```
call MPI_FILE_GET_ERRHANDLER (FILE, ERRHANDLER, IERROR)

    INTEGER         FILE, ERRHANDLER, IERROR
```

ファイルに対するエラーハンドラーの付加

基本構文

MPI_FILE_SET_ERRHANDLER (file, errhandler)

引数	値	説明	IN/OUT
file	handle	ファイル	INOUT
errhandler	handle	新しいエラーハンドラー	IN

C バインディング

int MPI_File_set_errhandler (MPI_File file, MPI_Errhandler errhandler)

Fortran バインディング

call MPI_FILE_SET_ERRHANDLER (FILE, ERRHANDLER, IERROR)

INTEGER FILE, ERRHANDLER, IERROR

MPI_Finalize (C) MPI_FINALIZE (Fortran)

MPI 機能の使用終了指示

基本構文

MPI_FINALIZE ()

C バインディング

int MPI_Finalize (void)

Fortran バインディング

call MPI_FINALIZE (IERROR)

整数型 IERROR

MPI_Finalized (C) MPI_FINALIZED (Fortran)

手続 MPI_Finalize の完了確認

基本構文

MPI_FINALIZED (flag)

引数	値	説明	IN/OUT
flag	論理	MPI_Finalize が完了していれば true	OUT

C バインディング

```
int MPI_Finalized (int *flag)
```

Fortran バインディング

```
call MPI_FINALIZED (FLAG, IERROR)

    LOGICAL         FLAG
    INTEGER         IERROR
```

MPI_Free_mem (C) MPI_FREE_MEM (Fortran)

メモリの解放

基本構文

MPI_FREE_MEM (base)

引数	値	説明	IN/OUT
base	任意	MPI_ALLOC_MEM によって割り当てられたメモリ領域の先頭アドレス	IN

C バインディング

```
int MPI_Free_mem (void *base)
```

Fortran バインディング

```
call MPI_FREE_MEM (BASE, IERROR)

    任意の型        BASE(*)
    INTEGER         IERROR
```

MPI_Get_library_version (C)　　MPI_GET_LIBRARY_VERSION (Fortran)

MPI バージョン情報の取得

基本構文

MPI_GET_LIBRARY_VERSION (version, resultlen)

引数	値	説明	IN/OUT
version	文字	バージョン名の文字列	OUT
resultlen	整数	バージョン名の長さ	OUT

C バインディング

int MPI_Get_library_version ((char *version, int *resultlen)

Fortran バインディング

call MPI_GET_LIBRARY_VERSION (VERSION, RESULTLEN, IERROR)

整数型　　RESULTLEN, IERROR

文字型　　VERSION

MPI_Get_processor_name (C)　　MPI_GET_PROCESSOR_NAME (Fortran)

プロセッサ名の返却

基本構文

MPI_GET_PROCESSOR_NAME (name, resultlen)

引数	値	説明	IN/OUT
name	文字	プロセッサ名の文字列	OUT
resultlen	整数	プロセッサ名の長さ	OUT

C バインディング

int MPI_Get_processor_name (char *name, int *resultlen)

Fortran バインディング

call MPI_GET_PROCESSOR_NAME (NAME, RESULTLEN, IERROR)

整数型　　RESULTLEN, IERROR

文字型　　NAME

MPI_Get_version (C)　　MPI_GET_VERSION (Fortran)

MPI バージョン情報の取得

基本構文

MPI_GET_VERSION (version, subversion)

引数	値	説明	IN/OUT
version	整数	バージョン数	OUT
subversion	整数	サブバージョン数	OUT

C バインディング

int MPI_Get_version ((int *version, int *subversion)

Fortran バインディング

call MPI_GET_VERSION (VERSION, SUBVERSION, IERROR)
整数型　　VERSION, SUBVERSION, IERROR

MPI_Init (C)　　MPI_INIT (Fortran)

MPI 環境の初期化

基本構文

MPI_INIT ()

C バインディング

int MPI_Init (int *argc, char ***argv)

Fortran バインディング

call MPI_INIT (IERROR)
整数型　　IERROR

MPI_Initialized (C)　　MPI_INITIALIZED (Fortran)

MPI 環境の初期化状態の問合せ

基本構文

MPI_INITIALIZED (flag)

引数	値	説明	IN/OUT
flag	論理	初期化状態のフラグ	OUT

C バインディング

int MPI_Initialized (int *flag)

Fortran バインディング

call MPI_INITIALIZED (FLAG, IERROR)

整数型　　IERROR
論理型　　FLAG

MPI_Win_call_errhandler (C)　　MPI_WIN_CALL_ERRHANDLER (Fortran)

errorcode によるエラーハンドラーの起動

基本構文

MPI_WIN_CALL_ERRHANDLER (win, errorcode)

引数	値	説明	IN/OUT
win	handle	エラーハンドラーをもつウィンドウ	IN
errorcode	整数	エラーコード	IN

C バインディング

int MPI_Win_call_errhandler (MPI_Win win, int errorcode)

Fortran バインディング

call MPI_WIN_CALL_ERRHANDLER (WIN, ERRORCODE, IERROR)

INTEGER　　WIN, ERRORCODE, IERROR

MPI_Win_create_errhandler (C)

MPI_WIN_CREATE_ERRHANDLER (Fortran)

ウィンドウに付加するエラーハンドラーの生成

基本構文

MPI_WIN_CREATE_ERRHANDLER (function, errhandler)

引数	値	説明	IN/OUT
function	関数	利用者定義のエラー処理手続	IN
errhandler	handle	MPI エラーハンドラー	OUT

C バインディング

```
int MPI_Win_create_errhandler (MPI_Win_errhandler_fn *function, MPI_Errhandler
                               *errhandler)
```

Fortran バインディング

```
call MPI_WIN_CREATE_ERRHANDLER (FUNCTION, ERRHANDLER, IERROR)

    EXTERNAL        FUNCTION
    INTEGER         ERRHANDLER, IERROR
```

MPI_Win_get_errhandler (C)

MPI_WIN_GET_ERRHANDLER (Fortran)

ウィンドウに現在付加されているエラーハンドラーの取得

基本構文

MPI_WIN_GET_ERRHANDLER (win, errhandler)

引数	値	説明	IN/OUT
win	handle	ウィンドウ	IN
errhandler	handle	win に現在付加されているエラーハンドラー	OUT

C バインディング

```
int MPI_Win_get_errhandler (MPI_Win win, MPI_Errhandler *errhandler)
```

Fortran バインディング

```
call MPI_WIN_GET_ERRHANDLER (WIN, ERRHANDLER, IERROR)

    INTEGER         WIN, ERRHANDLER, IERROR
```

ウィンドウに対するエラーハンドラーの付加

基本構文

MPI_WIN_SET_ERRHANDLER (win, errhandler)

引数	値	説明	IN/OUT
win	handle	ウィンドウ	INOUT
errhandler	handle	新しいエラーハンドラー	OUT

C バインディング

```
int MPI_Win_set_errhandler (MPI_Win win, MPI_Errhandler errhandler)
```

Fortran バインディング

```
call MPI_WIN_SET_ERRHANDLER (WIN, ERRHANDLER, IERROR)

    INTEGER          WIN, ERRHANDLER, IERROR
```

4.7 利用者指定情報

MPI_INFO_CREATE	MPI_INFO_DELETE	MPI_INFO_DUP
MPI_INFO_FREE	MPI_INFO_GET	MPI_INFO_GET_NKEYS
MPI_INFO_GET_NTHKEY	MPI_INFO_GET_VALUELEN	MPI_INFO_SET

MPI_Info_create (C) MPI_INFO_CREATE (Fortran)

---

info オブジェクトの生成

基本構文

MPI_INFO_CREATE (info)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	OUT

C バインディング

```
int MPI_Info_create (MPI_Info *info)
```

Fortran バインディング

```
call MPI_INFO_CREATE (INFO, IERROR)

    INTEGER          INFO, IERROR
```

MPI_Info_delete (C) MPI_INFO_DELETE (Fortran)

info における(key, value)ペアの削除

基本構文

MPI_INFO_DELETE (info, key)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	INOUT
key	文字	キー(文字列)	IN

C バインディング

```
int MPI_Info_delete (MPI_Info info, char *key)
```

Fortran バインディング

```
call MPI_INFO_DELETE (INFO, KEY, IERROR)

    INTEGER           INFO, IERROR
    CHARACTER*(*)     KEY
```

MPI_Info_dup (C) MPI_INFO_DUP (Fortran)

同じ(key, value)ペア, および 同じキーの順番をもつ info オブジェクトの複製

基本構文

MPI_INFO_DUP (info, newinfo)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
newinfo	handle	info オブジェクト	OUT

C バインディング

```
int MPI_Info_dup (MPI_Info info, MPI_Info *newinfo)
```

Fortran バインディング

```
call MPI_INFO_DUP (INFO, NEWINFO, IERROR)

    INTEGER           INFO, NEWINFO, IERROR
```

MPI_Info_free (C)　　MPI_INFO_FREE (Fortran)

info オブジェクトの解放

基本構文

MPI_INFO_FREE (info)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	INOUT

C バインディング

```
int MPI_Info_free (MPI_Info *info)
```

Fortran バインディング

```
call MPI_INFO_FREE (INFO, IERROR)

    INTEGER         INFO, IERROR
```

MPI_Info_get (C)　　MPI_INFO_GET (Fortran)

info に設定された key に対応する値の取得

基本構文

MPI_INFO_GET (info, key, valuelen, value, flag)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
key	文字	キー(文字列)	IN
valuelen	整数	value 引数の長さ	IN
value	文字	値(文字列)	OUT
flag	論理	key が定義されていれば true	OUT

C バインディング

```
int MPI_Info_get (MPI_Info info, char *key, int valuelen, char *value, int *flag)
```

Fortran バインディング

```
call MPI_INFO_GET (INFO, KEY, VALUELEN, VALUE, FLAG, IERROR)

    INTEGER         INFO, VALUELEN, IERROR
    CHARACTER*(*)   KEY, VALUE
    LOGICAL         FLAG
```

MPI_Info_get_nkeys (C) MPI_INFO_GET_NKEYS (Fortran)

info において定義されたキーの数の取得

基本構文

MPI_INFO_GET_NKEYS (info, nkeys)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
nkeys	整数	定義されたキーの数	OUT

C バインディング

```
int MPI_Info_get_nkeys (MPI_Info info, int *nkeys)
```

Fortran バインディング

```
call MPI_INFO_GET_NKEYS (INFO, NKEYS, IERROR)

    INTEGER         INFO, NKEYS, IERROR
```

MPI_Info_get_nthkey (C) MPI_INFO_GET_NTHKEY (Fortran)

info において n 番目に定義されたキーの取得

基本構文

MPI_INFO_GET_NTHKEY (info, n, key)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
n	整数	キーの数	IN
key	文字	キー(文字列)	OUT

C バインディング

```
int MPI_Info_get_nthkey (MPI_Info info, int n, char *key)
```

Fortran バインディング

```
call MPI_INFO_GET_NTHKEY (INFO, N, KEY, IERROR)

    INTEGER         INFO, N, IERROR
    CHARACTER*(*)   KEY
```

key に対応する value の長さの取得

基本構文

MPI_INFO_GET_VALUELEN (info, key, valuelen, flag)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
key	文字	キー(文字列)	IN
valuelen	整数	value 引数の長さ	OUT
flag	論理	key が定義されていれば true，そうでなければ false	OUT

C バインディング

```
int MPI_Info_get_valuelen (MPI_Info info, char *key, int *valuelen, int *flag)
```

Fortran バインディング

```
call MPI_INFO_GET_VALUELEN (INFO, KEY, VALUELEN, FLAG, IERROR)

    INTEGER          INFO, VALUELEN, IERROR
    LOGICAL          FLAG
    CHARACTER*(*)    KEY
```


info に対する(key, value)ペアの設定

基本構文

MPI_INFO_SET (info, key, value)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	INOUT
key	文字	キー(文字列)	IN
value	文字	値(文字列)	IN

C バインディング

```
int MPI_Info_set (MPI_Info info, char*key, char *value)
```

Fortran バインディング

```
call MPI_INFO_SET (INFO, KEY, VALUE, IERROR)

    INTEGER          INFO, IERROR
    CHARACTER*(*)    KEY, VALUE
```

4.8 プロセスの生成 および 管理

MPI_CLOSE_PORT	MPI_COMM_ACCEPT	MPI_COMM_CONNECT
MPI_COMM_DISCONNECT	MPI_COMM_GET_PARENT	MPI_COMM_JOIN
MPI_COMM_SPAWN	MPI_COMM_SPAWN_MULTIPLE	MPI_LOOKUP_NAME
MPI_OPEN_PORT	MPI_PUBLISH_NAME	MPI_UNPUBLISH_NAME

MPI_Close_port (C) MPI_CLOSE_PORT (Fortran)

サーバクライアント接続で使用したポートの解放

基本構文

MPI_CLOSE_PORT (port_name)

引数	値	説明	IN/OUT
port_name	文字	ポート(文字列)	IN

C バインディング

```
int MPI_Close_port (char *port_name)
```

Fortran バインディング

```
call MPI_CLOSE_PORT (PORT_NAME, IERROR)

    CHARACTER*(*)   PORT_NAME
    INTEGER         IERROR
```

クライアントからの接続要求の受理，通信経路の確立

基本構文

MPI_COMM_ACCEPT (port_name, info, root, comm, newcomm)

引数	値	説明	IN/OUT
port_name	文字	ポート名(文字列，root だけ使用される)	IN
info	handle	実装依存の情報(root だけ使用される)	IN
root	整数	ルートノードの comm 上のランク	IN
comm	handle	呼出しを集団的に行うグループ内コミュニケーター	IN
newcomm	handle	リモートグループであるクライアントとのグループ間コミュニケーター	OUT

C バインディング

```
int MPI_Comm_accept (char *port_name, MPI_Info info, int root, MPI_Comm comm,
MPI_Comm *newcomm)
```

Fortran バインディング

```
call MPI_COMM_ACCEPT (PORT_NAME, INFO, ROOT, COMM, NEWCOMM, IERROR)

    CHARACTER*(*)   PORT_NAME
    INTEGER         INFO, ROOT, COMM, NEWCOMM, IERROR
```

サーバとの通信経路の確立

基本構文

MPI_COMM_CONNECT (port_name, info, root, comm, newcomm)

引数	値	説明	IN/OUT
port_name	文字	ポート名(文字列，root だけ使用される)	IN
info	handle	実装依存の情報(root だけ使用される)	IN
root	整数	ルートノードの comm 上のランク	IN
comm	handle	呼出しを集団的に行うグループ内コミュニケーター	IN
newcomm	handle	リモートグループであるサーバとのグループ間コミュニケーター	OUT

C バインディング

```
int MPI_Comm_connect (char *port_name, MPI_Info info, int root, MPI_Comm comm,
MPI_Comm *newcomm)
```

Fortran バインディング

```
call MPI_COMM_CONNECT (PORT_NAME, INFO, ROOT, COMM, NEWCOMM, IERROR)

    CHARACTER*(*)   PORT_NAME
    INTEGER         INFO, ROOT, COMM, NEWCOMM, IERROR
```

MPI_Comm_disconnect (C) MPI_COMM_DISCONNECT (Fortran)

保留中である通信の完了待合せ， および コミュニケーターの解放

基本構文

MPI_COMM_DISCONNECT (comm)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	INOUT

C バインディング

```
int MPI_Comm_disconnect (MPI_Comm *comm)
```

Fortran バインディング

```
call MPI_COMM_DISCONNECT (COMM, IERROR)

    INTEGER         COMM, IERROR
```

MPI_Comm_get_parent (C)　　MPI_COMM_GET_PARENT (Fortran)

親グループ間コミュニケーターの取得

基本構文

MPI_COMM_GET_PARENT (parent)

引数	値	説明	IN/OUT
parent	handle	親グループ間コミュニケーター	OUT

C バインディング

```
int MPI_Comm_get_parent (MPI_Comm *parent)
```

Fortran バインディング

```
call MPI_COMM_GET_PARENT (PARENT, IERROR)

    INTEGER          PARENT, IERROR
```

MPI_Comm_join (C)　　MPI_COMM_JOIN (Fortran)

通信経路の確立

基本構文

MPI_COMM_JOIN (fd, intercomm)

引数	値	説明	IN/OUT
fd	整数	ソケットファイル記述子	IN
intercomm	handle	新しいグループ間コミュニケーター	OUT

C バインディング

```
int MPI_Comm_join (int fd, MPI_Comm *intercomm)
```

Fortran バインディング

```
call MPI_COMM_JOIN (FD, INTERCOMM, IERROR)

    INTEGER          FD, INTERCOMM, IERROR
```

プロセスの起動 および 通信経路の確立

基本構文

MPI_COMM_SPAWN (command, argv, maxprocs, info, root, comm, intercomm, array_of_errcodes)

引数	値	説明	IN/OUT
command	文字	生成するプログラム名（文字列，root においてだけ意味をもつ)	IN
argv	文字	command への引数（文字列配列，root においてだけ意味をもつ)	IN
maxprocs	整数	起動するプロセスの最大数(root においてだけ意味をもつ)	IN
info	handle	起動するプロセスの実行時環境を指示する info オブジェクト(root においてだけ意味をもつ)	IN
root	整数	上記引数を評価するルートプロセスのランク	IN
comm	handle	オリジナルグループを含むグループ内コミュニケーター	OUT
intercomm	handle	オリジナルグループと生成されたグループ間のグループ間コミュニケーター	OUT
array_of_errcodes	整数	起動するプロセスごとのエラーコード(整数型配列)	IN

C バインディング

```
int MPI_Comm_spawn (char *command, char *argv[], int maxprocs, MPI_Info info, int root,
                    MPI_Comm comm, MPI_Comm *intercomm, int array_of_errcodes[])
```

Fortran バインディング

```
call MPI_COMM_SPAWN (COMMAND, ARGV, MAXPROCS, INFO, ROOT, COMM,
                     INTERCOMM, ARRAY_OF_ERRCODES, IERROR)

CHARACTER*(*)  COMMAND, ARGV(*)
INTEGER        INFO, MAXPROCS, ROOT, COMM, INTERCOMM,
               ARRAY_OF_ERRCODES(*), IERROR
```

複数のプログラムからのプロセスの起動 および 通信経路の確立

基本構文

MPI_COMM_SPAWN_MULTIPLE (count, array_of_commands, array_of_argv, array_of_maxprocs, array_of_info, root, comm, intercomm, array_of_errcodes)

引数	値	説明	IN/OUT
count	整数	生成するプログラム数（ルートプロセスにおいてだけ意味をもつ)	IN
array_of_commands	文字	生成するプログラム名	IN
array_of_argv	文字	(文字列配列，ルートプロセスにおいてだけ意味をもつ)	IN
array_of_maxprocs	整数	command への引数（文字列配列，root においてだけ意味をもつ)	IN
array_of_info	handle	起動するプロセスの最大数(root においてだけ意味をもつ)	IN
root	整数	起動するプロセスの実行時環境を指示する info オブジェクト(handle の配列，ルートプロセスにおいてだけ意味をもつ)	IN
comm	handle	上記引数を評価するルートプロセスのランク	IN
intercomm	handle	オリジナルグループを含むグループ内コミュニケーター	OUT
array_of_errcodes	整数	オリジナルグループと生成されたグループ間のグループ間コミュニケーター	OUT
		起動するプロセスごとのエラーコード(整数型配列)	

C バインディング

```
int MPI_Comm_spawn_multiple (int count, char *array_of_commands[], char
                             **array_of_argv[], int array_of_maxprocs[], MPI_Info
                             array_of_info[], int root, MPI_Comm comm, MPI_Comm
                             *intercomm, int array_of_errcodes[])
```

Fortran バインディング

```
call MPI_COMM_SPAWN_MULTIPLE (COUNT, ARRAY_OF_COMMANDS,
                              ARRAY_OF_ARGV, ARRAY_OF_MAXPROCS,
                              ARRAY_OF_INFO, ROOT, COMM, INTERCOMM,
                              ARRAY_OF_ERRCODES, IERROR)

    CHARACTER*(*)  ARRAY_OF_COMMANDS(*), ARRAY_OF_ARGV(COUNT, *)
    INTEGER        COUNT, ARRAY_OF_INFO(*), ARRAY_OF_MAXPROCS(*), ROOT,
                   COMM, INTERCOMM, ARRAY_OF_ERRCODES(*), IERROR
```

MPI_Lookup_name (C)　　MPI_LOOKUP_NAME (Fortran)

ポートの検索

基本構文

MPI_LOOKUP_NAME (service_name, info, port_name)

引数	値	説明	IN/OUT
service_name	文字	サービス名(文字列)	IN
info	handle	実装固有の情報	IN
port_name	文字	ポート名(文字列)	OUT

C バインディング

```
int MPI_Lookup_name (char *service_name, MPI_Info info, char *port_name)
```

Fortran バインディング

```
call MPI_LOOKUP_NAME (SERVICE_NAME, INFO, PORT_NAME, IERROR)

    CHARACTER*(*)   SERVICE_NAME, PORT_NAME
    INTEGER         INFO, IERROR
```

MPI_Open_port (C)　　MPI_OPEN_PORT (Fortran)

サーバクライアント接続で使用するポートの開設

基本構文

MPI_OPEN_PORT(info, port_name)

引数	値	説明	IN/OUT
info	handle	アドレスを開設する方法についての実装固有の情報	IN
port_name	文字	新たに開設したポート(文字列)	OUT

C バインディング

```
int MPI_Open_port (MPI_Info info, char *port_name)
```

Fortran バインディング

```
call MPI_OPEN_PORT (INFO, PORT_NAME, IERROR)

    CHARACTER*(*)   PORT_NAME
    INTEGER         INFO, IERROR
```

MPI_Publish_name (C)　　MPI_PUBLISH_NAME (Fortran)

ポートの公開

基本構文

MPI_PUBLISH_NAME (service_name, info, port_name)

引数	値	説明	IN/OUT
service_name	文字列	サービス名(文字列)	IN
info	handle	実装固有の情報	IN
port_name	文字列	ポート名(文字列)	IN

C バインディング

int MPI_Publish_name (char *service_name, MPI_Info info, char *port_name)

Fortran バインディング

call MPI_PUBLISH_NAME (SERVICE_NAME, INFO, PORT_NAME, IERROR)

CHARACTER*(*)　SERVICE_NAME, PORT_NAME
INTEGER　　INFO, IERROR

MPI_Unpublish_name (C)　　MPI_UNPUBLISH_NAME (Fortran)

ポートの非公開

基本構文

MPI_UNPUBLISH_NAME (service_name, info, port_name)

引数	値	説明	IN/OUT
service_name	文字列	サービス名(文字列)	IN
info	handle	実装固有の情報	IN
port_name	文字列	ポート名(文字列)	IN

C バインディング

int MPI_Unpublish_name (char *service_name, MPI_Info info, char *port_name)

Fortran バインディング

call MPI_UNPUBLISH_NAME (SERVICE_NAME, INFO, PORT_NAME, IERROR)

CHARACTER*(*)　SERVICE_NAME, PORT_NAME
INTEGER　　INFO, IERROR

4.9 片側通信

MPI_ACCUMULATE	MPI_COMPARE_AND_SWAP	MPI_FETCH_AND_OP
MPI_GET	MPI_GET_ACCUMULATE	MPI_PUT
MPI_RACCUMULATE	MPI_RGET	MPI_RGET_ACCUMULATE
MPI_RPUT	MPI_WIN_ALLOCATE	MPI_WIN_ALLOCATE_SHARED
MPI_WIN_ATTACH	MPI_WIN_COMPLETE	MPI_WIN_CREATE
MPI_WIN_CREATE_DYNAMIC	MPI_WIN_DETACH	MPI_WIN_FENCE
MPI_WIN_FLUSH	MPI_WIN_FLUSH_ALL	MPI_WIN_FLUSH_LOCAL
MPI_WIN_FLUSH_LOCAL_ALL	MPI_WIN_FREE	MPI_WIN_GET_GROUP
MPI_WIN_GET_INFO	MPI_WIN_LOCK	MPI_WIN_LOCK_ALL
MPI_WIN_POST	MPI_WIN_SET_INFO	MPI_WIN_SHARED_QUERY
MPI_WIN_START	MPI_WIN_SYNC	MPI_WIN_TEST
MPI_WIN_UNLOCK	MPI_WIN_UNLOCK_ALL	MPI_WIN_WAIT

起点メモリ および ターゲットメモリに対する集計演算，ターゲットメモリへの演算結果の格納

基本構文

MPI_ACCUMULATE (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count,
target_datatype, op, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
op	handle	集計演算子	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Accumulate (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int
                    target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype
                    target_datatype, MPI_Op op, MPI_Win win)
```

Fortran バインディング

```
call MPI_ACCUMULATE (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE,
                     TARGET_RANK, TARGET_DISP, TARGET_COUNT,
                     TARGET_DATATYPE, OP, WIN, IERROR)
```

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, IERROR

一要素に対するコンペア・アンド・スワップ

基本構文

MPI_COMPARE_AND_SWAP (origin_addr, compare_addr, result_addr, datatype, target_rank, target_disp, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
compare_addr	任意	比較用バッファの先頭アドレス	IN
result_addr	任意	結果用バッファの先頭アドレス	OUT
datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Compare_and_swap (void *origin_addr, void *compare_addr, void *result_addr,
                          MPI_Datatype datatype, int target_rank, MPI_Aint target_disp,
                          MPI_Win win)
```

Fortran バインディング

```
call MPI_COMPARE_AND_SWAP (ORIGIN_ADDR, COMPARE_ADDR, RESULT_ADDR,
                           DATATYPE, TARGET_RANK, TARGET_DISP, WIN,
                           IERROR)

    任意の型                                ORIGIN_ADDR(*) COMPARE_ADDR(*),
                                            RESULT_ADDR(*)
    INTEGER(KIND=MPI_ADDRESS_KIND)          TARGET_DISP
    INTEGER                                 DATATYPE, TARGET_RANK, WIN, IERROR
```

一要素に対するターゲットメモリから結果メモリへのデータ転送と起点メモリ および ターゲットメモリに対する集計演算，ターゲットメモリへの演算結果の格納

基本構文

MPI_FETCH_AND_OP (origin_addr, result_addr, datatype, target_rank, target_disp, op, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
result_addr	任意	結果用バッファの先頭アドレス	OUT
datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
op	handle	集計演算子	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Fetch_and_op (void *origin_addr, void *result_addr, MPI_Datatype datatype, int
                      target_rank, MPI_Aint target_disp, MPI_Op op, MPI_Win win)
```

Fortran バインディング

```
call MPI_FETCH_AND_OP (ORIGIN_ADDR, RESULT_ADDR, DATATYPE, TARGET_RANK,
                       TARGET_DISP, OP, WIN, IERROR)
```

任意の型	ORIGIN_ADDR(*), RESULT_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	DATATYPE, TARGET_RANK, OP, WIN, IERROR

ターゲットメモリから起点メモリへのデータ転送

基本構文

MPI_GET (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	OUT
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Get (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int
        target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype
        target_datatype, MPI_Win win)
```

Fortran バインディング

```
call MPI_GET (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK,
        TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, IERROR)
```

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, WIN, IERROR

ターゲットメモリから結果メモリへのデータ転送と起点メモリ および ターゲットメモリに対する集計演算，ターゲットメモリへの演算結果の格納

基本構文

MPI_GET_ACCUMULATE (origin_addr, origin_count, origin_datatype, result_addr, result_count, result_datatype, target_rank, target_disp, target_count, target_datatype, op, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
result_addr	任意	結果用バッファの先頭アドレス	OUT
result_count	整数	結果用バッファ内のエントリ数(非負整数)	IN
result_datatype	handle	結果用バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
op	handle	reduce 演算	IN
win	handle	ウィンドウオブジェクト	

C バインディング

```
int MPI_Get_accumulate (void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
                        void *result_addr, int result_count, MPI_Datatype result_datatype,
                        int target_rank, MPI_Aint target_disp, int target_count,
                        MPI_Datatype target_datatype, MPI_Op op, MPI_Win win)
```

Fortran バインディング

```
call MPI_GET_ACCUMULATE (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE,
                         RESULT_ADDR, RESULT_COUNT, RESULT_DATATYPE,
                         TARGET_RANK, TARGET_DISP, TARGET_COUNT,
                         TARGET_DATATYPE, OP, WIN, IERROR)
```

任意の型	ORIGIN_ADDR(*) RESULT_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE, RESULT_COUNT, RESULT_DATATYPE, TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, IERROR

起点メモリからターゲットメモリへのデータ転送

基本構文

MPI_PUT (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Put (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win)

Fortran バインディング

call MPI_PUT (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, IERROR)

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, WIN, IERROR

通信識別子と紐づけされた起点メモリ および ターゲットメモリに対する集計演算，ターゲットメモリへの演算結果の格納

基本構文

MPI_RACCUMULATE (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count,
target_datatype, op, win, request)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
op	handle	集計演算子	IN
win	handle	ウィンドウオブジェクト	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Raccumulate (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int
                     target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype
                     target_datatype, MPI_Op op, MPI_Win win, MPI_request request)
```

Fortran バインディング

```
call MPI_RACCUMULATE (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE,
                      TARGET_RANK, TARGET_DISP, TARGET_COUNT,
                      TARGET_DATATYPE, OP, WIN, REQUEST, IERROR)
```

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, REQUEST, IERROR

通信識別子と紐づけされたターゲットメモリから起点メモリへのデータ転送

基本構文

MPI_RGET (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win, requeset)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	OUT
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
win	handle	ウィンドウオブジェクト	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Rget (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win, MPI_Request *request)

Fortran バインディング

call MPI_RGET (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST, IERROR)

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST, IERROR

通信識別子と紐づけされたターゲットメモリから結果メモリへのデータ転送と起点メモリ および ターゲットメモリに対する集計演算，ターゲットメモリへの演算結果の格納

基本構文

MPI_RGET_ACCUMULATE (origin_addr, origin_count, origin_datatype, result_addr, result_count, result_datatype, target_rank, target_disp, target_count, target_datatype, op, win, request)

引数	値	説明	IN/OUT
origin_addr	任意整数	バッファの先頭アドレス	
origin_count	handle	バッファ内のエントリ数(非負整数)	OUT
origin_datatype	任意整数	バッファのデータ型	IN
result_addr	handle	結果用バッファの先頭アドレス	IN
result_count	整数	結果用バッファ内のエントリ数(非負整数)	OUT
result_datatype	整数	結果用バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	handle	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	handle	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
op	handle	集計演算子	IN
win	handle	ウィンドウオブジェクト	IN
request	handle	通信識別子	OUT

C バインディング

```
int MPI_Rget_accumulate (void *origin_addr, int origin_count, MPI_Datatype origin_datatype,
                         void *result_addr, int result_count, MPI_Datatype result_datatype,
                         int target_rank, MPI_Aint target_disp, int target_count,
                         MPI_Datatype target_datatype, MPI_Op op, MPI_Win win,
                         MPI_Request *request)
```

Fortran バインディング

```
call MPI_RGET_ACCUMULATE (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE,
                          RESULT_ADDR, RESULT_COUNT, RESULT_DATATYPE,
                          TARGET_RANK, TARGET_DISP, TARGET_COUNT,
                          TARGET_DATATYPE, OP, WIN, REQUEST, IERROR)
```

任意の型	ORIGIN_ADDR(*) RESULT_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE, RESULT_COUNT, RESULT_DATATYPE, TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, OP, WIN, REQUEST, IERROR

通信識別子と紐づけされた起点メモリからターゲットメモリへのデータ転送

基本構文

MPI_RPUT (origin_addr, origin_count, origin_datatype, target_rank, target_disp, target_count, target_datatype, win, request)

引数	値	説明	IN/OUT
origin_addr	任意	バッファの先頭アドレス	IN
origin_count	整数	バッファ内のエントリ数(非負整数)	IN
origin_datatype	handle	バッファのデータ型	IN
target_rank	整数	ターゲットのランク(非負整数)	IN
target_disp	整数	ウィンドウの始点からターゲットバッファの始点までの変位(非負整数)	IN
target_count	整数	ターゲットバッファ内のエントリ数(非負整数)	IN
target_datatype	handle	ターゲットバッファ内の各エントリのデータ型	IN
win	handle	ウィンドウオブジェクト	IN
request	handle	通信識別子	OUT

C バインディング

int MPI_Rput (void *origin_addr, int origin_count, MPI_Datatype origin_datatype, int target_rank, MPI_Aint target_disp, int target_count, MPI_Datatype target_datatype, MPI_Win win, MPI_Request *request)

Fortran バインディング

call MPI_RPUT (ORIGIN_ADDR, ORIGIN_COUNT, ORIGIN_DATATYPE, TARGET_RANK, TARGET_DISP, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST, IERROR)

任意の型	ORIGIN_ADDR(*)
INTEGER(KIND=MPI_ADDRESS_KIND)	TARGET_DISP
INTEGER	ORIGIN_COUNT, ORIGIN_DATATYPE,TARGET_RANK, TARGET_COUNT, TARGET_DATATYPE, WIN, REQUEST, IERROR

メモリの割当てとウィンドウオブジェクトの生成

基本構文

MPI_WIN_ALLOCATE (size, disp_unit, info, comm, baseptr, win)

引数	値	説明	IN/OUT
size	整数	ウィンドウの大きさ(バイト単位，非負整数)	IN
disp_unit	整数	変位(バイト単位，正整数)	IN
info	handle	info オブジェクト	IN
comm	handle	コミュニケーター	IN
baseptr	整数	ウィンドウの先頭アドレス	OUT
win	handle	ウィンドウオブジェクト	OUT

C バインディング

```
int MPI_Win_allocate (void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm
                      comm, void *baseptr, MPI_Win *win)
```

Fortran バインディング

```
call MPI_WIN_ALLOCATE (BASE, SIZE, DISP_UNIT, INFO, COMM, WIN, IERROR)

    INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
    INTEGER                        DISP_UNIT, INFO, COMM, WIN, IERROR
```

共有メモリの割当てとウィンドウオブジェクトの生成

基本構文

MPI_WIN_ALLOCATE_SHARED (size, disp_unit, info, comm, baseptr, win)

引数	値	説明	IN/OUT
size	整数	ウィンドウの大きさ(バイト単位，非負整数)	IN
disp_unit	整数	変位(バイト単位，正整数)	IN
info	handle	info オブジェクト	IN
comm	handle	コミュニケーター	IN
baseptr	整数	ウィンドウの先頭アドレス	OUT
win	handle	ウィンドウオブジェクト	OUT

C バインディング

```
int MPI_Win_allocate_shared (void *base, MPI_Aint size, int disp_unit, MPI_Info info,
                             MPI_Comm comm, void *baseptr, MPI_Win *win)
```

Fortran バインディング

```
call MPI_WIN_ALLOCATE_SHARED (BASE, SIZE, DISP_UNIT, INFO, COMM, WIN,
                              IERROR)

    INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
    INTEGER                        DISP_UNIT, INFO, COMM, WIN, IERROR
```

MPI_Win_attach (C) MPI_WIN_ATTACH (Fortran)

ウィンドウオブジェクトへのメモリのアタッチ

基本構文

MPI_WIN_ATTACH (win, base, size)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN
base	任意	アタッチするメモリの先頭アドレス	IN
size	整数	ウィンドウの大きさ(バイト単位)	IN

C バインディング

int MPI_Win_attach (MPI_Win win, void *base, MPI_Aint size)

Fortran バインディング

call MPI_WIN_ATTACH (WIN, BASE, SIZE, IERROR)

任意の型 BASE(*)
INTEGER(KIND=MPI_ADDRESS_KIND) SIZE
INTEGER WIN, IERROR

MPI_Win_complete (C) MPI_WIN_COMPLETE (Fortran)

ウィンドウに対するアクセス要求区間の終了

基本構文

MPI_WIN_COMPLETE (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_complete (MPI_Win win)

Fortran バインディング

call MPI_WIN_COMPLETE (WIN, IERROR)

INTEGER WIN, IERROR

MPI_Win_create (C) MPI_WIN_CREATE (Fortran)

ウィンドウオブジェクトの生成

基本構文

MPI_WIN_CREATE (base, size, disp_unit, info, comm, win)

引数	値	説明	IN/OUT
base	任意	ウィンドウの先頭アドレス	IN
size	整数	ウィンドウの大きさ(バイト単位，非負整数)	IN
disp_unit	整数	変位(バイト単位，正整数)	IN
info	handle	info オブジェクト	IN
comm	handle	コミュニケーター	IN
win	handle	ウィンドウオブジェクト	OUT

C バインディング

```
int MPI_Win_create (void *base, MPI_Aint size, int disp_unit, MPI_Info info, MPI_Comm
                    comm, MPI_Win *win)
```

Fortran バインディング

```
call MPI_WIN_CREATE (BASE, SIZE, DISP_UNIT, INFO, COMM, WIN, IERROR)

    任意の型                                BASE(*)
    INTEGER(KIND=MPI_ADDRESS_KIND) SIZE
    INTEGER                                 DISP_UNIT, INFO, COMM, WIN, IERROR
```

MPI_Win_create_dynamic (C) MPI_WIN_CREATE_DYNAMIC (Fortran)

ウィンドウオブジェクトの生成（メモリの割当ては行わない）

基本構文

MPI_WIN_CREATE_DYNAMIC (info, comm, win)

引数	値	説明	IN/OUT
info	handle	info オブジェクト	IN
comm	handle	コミュニケーター	IN
win	handle	ウィンドウオブジェクト	OUT

C バインディング

```
int MPI_Win_create_dynamic (MPI_Info info, MPI_Comm comm, MPI_Win *win)
```

Fortran バインディング

```
call MPI_WIN_CREATE_DYNAMIC (INFO, COMM, WIN, IERROR)

    INTEGER                 INFO, COMM, WIN, IERROR
```

MPI_Win_detach (C)　　MPI_WIN_DETACH (Fortran)

ウィンドウオブジェクトへのメモリのデタッチ

基本構文

MPI_WIN_DETACH (win, base, size)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN
base	任意	デタッチするメモリの先頭アドレス	IN

C バインディング

```
int MPI_Win_detach (MPI_Win win, void *base)
```

Fortran バインディング

```
call MPI_WIN_DETACH (WIN, BASE, IERROR)

    任意の型                BASE(*)
    INTEGER                 WIN, IERROR
```

MPI_Win_fence (C)　　MPI_WIN_FENCE (Fortran)

ウィンドウに対するアクセス要求区間 および アクセス許可区間の開始と終了

基本構文

MPI_WIN_FENCE (assert, win)

引数	値	説明	IN/OUT
assert	整数	最適化情報	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Win_fence (int assert, MPI_Win win)
```

Fortran バインディング

```
call MPI_WIN_FENCE (ASSERT, WIN, IERROR)

    INTEGER         ASSERT, WIN, IERROR
```

MPI_Win_flush (C) MPI_WIN_FLUSH (Fortran)

ウィンドウ上のターゲットプロセスに対する片側通信の完了待合せ

基本構文

MPI_WIN_FLUSH (rank, win)

引数	値	説明	IN/OUT
rank	整数	ターゲットプロセスのランク	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Win_flush (int rank, MPI_Win win)
```

Fortran バインディング

```
call MPI_WIN_FLUSH (RANK, WIN, IERROR)

    INTEGER          RANK, WIN, IERROR
```

MPI_Win_flush_all (C) MPI_WIN_FLUSH_ALL (Fortran)

ウィンドウ上の全ターゲットプロセスに対する片側通信の完了待合せ

基本構文

MPI_WIN_FLUSH_ALL (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Win_flush_all (MPI_Win win)
```

Fortran バインディング

```
call MPI_WIN_FLUSH_ALL (WIN, IERROR)

    INTEGER          WIN, IERROR
```

MPI_Win_flush_local (C) MPI_WIN_FLUSH_LOCAL (Fortran)

ウィンドウのターゲットプロセスに対する片側通信のオリジンプロセス側の完了待合せ

基本構文

MPI_WIN_FLUSH_LOCAL (rank, win)

引数	値	説明	IN/OUT
rank	整数	ターゲットプロセスのランク	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_flush_local (int rank, MPI_Win win)

Fortran バインディング

call MPI_WIN_FLUSH_LOCAL (RANK, WIN, IERROR)

INTEGER RANK, WIN, IERROR

MPI_Win_flush_local_all (C) MPI_WIN_FLUSH_LOCAL_ALL (Fortran)

ウィンドウの全ターゲットプロセスに対する片側通信のオリジンプロセス側の完了待合せ

基本構文

MPI_WIN_FLUSH_LOCAL_ALL (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_flush_local_all (MPI_Win win)

Fortran バインディング

call MPI_WIN_FLUSH_LOCAL_ALL (WIN, IERROR)

INTEGER WIN, IERROR

MPI_Win_free (C)　　MPI_WIN_FREE (Fortran)

ウィンドウオブジェクトの解放

基本構文

MPI_WIN_FREE (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	INOUT

C バインディング

```
int MPI_Win_free (MPI_Win *win)
```

Fortran バインディング

```
call MPI_WIN_FREE (WIN, IERROR)

    INTEGER          WIN, IERROR
```

MPI_Win_get_group (C)　　MPI_WIN_GET_GROUP (Fortran)

ウィンドウへのアクセスを共有するプロセスグループの複製

基本構文

MPI_WIN_GET_GROUP (win, group)

引数	値	説明	IN/OUT
win group	handle handle	ウィンドウオブジェクト ウィンドウへのアクセスを共有するプロセスグループ	IN OUT

C バインディング

```
int MPI_Win_get_group (MPI_Win win, MPI_Group *group)
```

Fortran バインディング

```
call MPI_WIN_GET_GROUP (WIN, GROUP, IERROR)

    INTEGER          WIN, GROUP, IERROR
```

MPI_Win_get_info (C) MPI_WIN_GET_INFO (Fortran)

ウィンドウに付加された info オブジェクトの取得

基本構文

MPI_WIN_GET_INFO (win, info)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN
info	handle	info オブジェクト	OUT

C バインディング

int MPI_Win_get_info (MPI_Win win, MPI_Info *info)

Fortran バインディング

call MPI_WIN_GET_INFO (WIN, INFO, IERROR)

INTEGER WIN, INFO, IERROR

MPI_Win_lock (C) MPI_WIN_LOCK (Fortran)

ウィンドウに対するアクセス権の確保, および アクセス要求区間の開始

基本構文

MPI_WIN_LOCK (lock_type, rank, assert, win)

引数	値	説明	IN/OUT
lock_type	状態	MPI_LOCK_EXCLUSIVE または , MPI_LOCK_SHARED のいずれか	IN
rank	整数	ロック対象ウィンドウをもつプロセスのランク(非負整数)	IN
assert	整数	最適化情報	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_lock (int lock_type, int rank, int assert, MPI_Win win)

Fortran バインディング

call MPI_WIN_LOCK (LOCK_TYPE, RANK, ASSERT, WIN, IERROR)

INTEGER LOCK_TYPE, RANK, ASSERT, WIN, IERROR

MPI_Win_lock_all (C)　　MPI_WIN_LOCK_ALL (Fortran)

ウィンドウに対するアクセス権の確保,　および アクセス要求区間の開始 (全プロセス対象)

基本構文

MPI_WIN_LOCK_ALL (assert, win)

引数	値	説明	IN/OUT
assert	整数	最適化情報	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_lock_all (int assert, MPI_Win win)

Fortran バインディング

call MPI_WIN_LOCK_ALL (ASSERT, WIN, IERROR)

INTEGER　　ASSERT, WIN, IERROR

MPI_Win_post (C)　　MPI_WIN_POST (Fortran)

ウィンドウに対するアクセス許可区間の開始

基本構文

MPI_WIN_POST (group, assert, win)

引数	値	説明	IN/OUT
group	整数	起点プロセスを含むグループ	IN
assert	整数	最適化情報	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_post (MPI_Group group, int assert, MPI_Win win)

Fortran バインディング

call MPI_WIN_POST (GROUP, ASSERT, WIN, IERROR)

INTEGER　　INTEGER GROUP, ASSERT, WIN, IERROR

MPI_Win_set_info (C) MPI_WIN_SET_INFO (Fortran)

ウィンドウに対する info オブジェクトの付加

基本構文

MPI_WIN_SET_INFO (win, info)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	INOUT
info	handle	info オブジェクト	OUT

C バインディング

```
int MPI_Win_set_info (MPI_Win win, MPI_Info info)
```

Fortran バインディング

```
call MPI_WIN_SET_INFO (WIN, INFO, IERROR)

    INTEGER          WIN, INFO, IERROR
```

MPI_Win_shared_query (C) MPI_WIN_SHARED_QUERY (Fortran)

他プロセスの共有メモリのアドレス問合せ

基本構文

MPI_WIN_SHARED_QUERY (win, rank, size, disp_unit, baseptr)

引数	値	説明	IN/OUT
win	handle	共有メモリのウィンドウオブジェクト	IN
rank	整数	ランクもしくは MPI_PROC_NULL	IN
size	整数	ウィンドウの大きさ(バイト単位，非負整数)	OUT
disp_unit	整数	変位(バイト単位，正整数)	OUT
baseptr	整数	他プロセスの先頭アドレス	OUT

C バインディング

```
int MPI_Win_shared_query (MPI_Win win, int rank, MPI_Aint *size, int *disp_unit, void
                          *baseptr)
```

Fortran バインディング

```
call MPI_WIN_SHARED_QUERY (WIN, RANK, SIZE, DISP_UNIT, BASEPTR, IERROR)

    INTEGER(KIND=MPI_ADDRESS_KIND) SIZE, BASEPTR
    INTEGER                        WIN, RANK, DISP_UNIT, IERROR
```

MPI_Win_start (C) MPI_WIN_START (Fortran)

ウィンドウに対するアクセス要求区間の開始

基本構文

MPI_WIN_START (group, assert, win)

引数	値	説明	IN/OUT
group	handle	対象プロセスを含むグループ	IN
assert	整数	最適化情報	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_start (MPI_Group group, int assert, MPI_Win win)

Fortran バインディング

call MPI_WIN_START (GROUP, ASSERT, WIN, IERROR)

INTEGER GROUP, ASSERT, WIN, IERROR

MPI_Win_sync (C) MPI_WIN_SYNC (Fortran)

ウィンドウ データの同期

基本構文

MPI_WIN_SYNC (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_sync (MPI_Win win)

Fortran バインディング

call MPI_WIN_SYNC (WIN, IERROR)

INTEGER WIN, IERROR

MPI_Win_test (C) MPI_WIN_TEST (Fortran)

ウィンドウに対するアクセス許可区間の終了テスト

基本構文

MPI_WIN_TEST (win, flag)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN
flag	論理	フラグ	OUT

C バインディング

```
int MPI_Win_test (MPI_Win win, int *flag)
```

Fortran バインディング

```
call MPI_WIN_TEST (WIN, FLAG, IERROR)

    INTEGER         WIN, IERROR
    LOGICAL         FLAG
```

MPI_Win_unlock (C) MPI_WIN_UNLOCK (Fortran)

ウィンドウに対するアクセス要求区間の終了， および アクセス権の解放

基本構文

MPI_WIN_UNLOCK (rank, win)

引数	値	説明	IN/OUT
rank	整数	ロック対象ウィンドウをもつプロセスのランク(非負整数)	IN
win	handle	ウィンドウオブジェクト	IN

C バインディング

```
int MPI_Win_unlock (int rank, MPI_Win win)
```

Fortran バインディング

```
call MPI_WIN_UNLOCK (RANK, WIN, IERROR)

    INTEGER         RANK, WIN, IERROR
```

MPI_Win_unlock_all (C) MPI_WIN_UNLOCK_ALL (Fortran)

ウィンドウに対するアクセス要求区間の終了， および アクセス権の解放（全プロセス対象）

基本構文

MPI_WIN_UNLOCK_ALL (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_unlock_all (MPI_Win win)

Fortran バインディング

call MPI_WIN_UNLOCK_ALL (WIN, IERROR)

INTEGER WIN, IERROR

MPI_Win_wait (C) MPI_WIN_WAIT (Fortran)

ウィンドウに対するアクセス許可区間の終了

基本構文

MPI_WIN_WAIT (win)

引数	値	説明	IN/OUT
win	handle	ウィンドウオブジェクト	IN

C バインディング

int MPI_Win_wait (MPI_Win win)

Fortran バインディング

call MPI_WIN_WAIT (WIN, IERROR)

INTEGER WIN, IERROR

4.10 外部インタフェース

MPI_GREQUEST_COMPLETE	MPI_GREQUEST_START	MPI_INIT_THREAD
MPI_IS_THREAD_MAIN	MPI_QUERY_THREAD	MPI_STATUS_SET_CANCELLED
MPI_STATUS_SET_ELEMENTS	MPI_STATUS_SET_ELEMENTS_X	

MPI_Grequest_complete (C) | MPI_GREQUEST_COMPLETE (Fortran)

一般化要求の完了

基本構文

MPI_GREQUEST_COMPLETE (request)

引数	値	説明	IN/OUT
request	handle	一般化要求	INOUT

C バインディング

```
int MPI_Grequest_complete (MPI_Request request)
```

Fortran バインディング

```
call MPI_GREQUEST_COMPLETE (REQUEST, IERROR)

    INTEGER          REQUEST, IERROR
```

MPI_Grequest_start (C) MPI_GREQUEST_START (Fortran)

新しい一般化要求の開始

基本構文

MPI_GREQUEST_START (query_fn, free_fn, cancel_fn, extra_state, request)

引数	値	説明	IN/OUT
query_fn	関数	要求状態が問い合わせられた時に呼ばれるコールバック関数	IN
free_fn	関数	要求が解放された時に呼ばれるコールバック関数	IN
cancel_fn	関数	要求がキャンセルされた時に呼ばれるコールバック関数	IN
extra_state	状態	拡張状態	IN
request	handle	一般化要求	OUT

C バインディング

```
int MPI_Grequest_start (MPI_Grequest_query_function *query_fn,
                        MPI_Grequest_free_function *free_fn, MPI_Grequest_cancel_function
                        *cancel_fn, void *extra_state, MPI_Request *request)
```

Fortran バインディング

```
call MPI_GREQUEST_START (QUERY_FN, FREE_FN, CANCEL_FN, EXTRA_STATE,
                         REQUEST, IERROR)

    INTEGER                                 REQUEST, IERROR
    EXTERNAL                                QUERY_FN, FREE_FN, CANCEL_FN
    INTEGER(KIND=MPI_ADDRESS_KIND) EXTRA_STATE
```

MPI_Init_thread (C) MPI_INIT_THREAD (Fortran)

MPI および MPI スレッド環境の初期化

基本構文

MPI_INIT_THREAD (required, provided)

引数	値	説明	IN/OUT
required	整数	要求されるスレッドサポートのレベル	IN
rpovided	整数	提供されるスレッドサポートのレベル	OUT

C バインディング

```
int MPI_Init_thread (int *argc, char *((*argv)[]), int required, int *provided)
```

Fortran バインディング

```
call MPI_INIT_THREAD (REQUIRED, PROVIDED, IERROR)

    INTEGER          REQUIRED, PROVIDED, IERROR
```

MPI_Is_thread_main (C) MPI_IS_THREAD_MAIN (Fortran)

メインスレッド(MPI_INIT または MPI_INIT_THREAD を呼出したスレッド)か否かの検査

基本構文

MPI_IS_THREAD_MAIN (flag)

引数	値	説明	IN/OUT
flag	論理	手続を呼び出したスレッドが主スレッドならば真, そうでなければ偽	OUT

C バインディング

```
int MPI_Is_thread_main (int *flag)
```

Fortran バインディング

```
call MPI_IS_THREAD_MAIN (FLAG, IERROR)

    LOGICAL         FLAG
    INTEGER         IERROR
```

MPI_Query_thread (C) MPI_QUERY_THREAD (Fortran)

スレッドサポートレベルの問合せ

基本構文

MPI_QUERY_THREAD (provided)

引数	値	説明	IN/OUT
provided	整数	現在のスレッドサポートレベル	OUT

C バインディング

```
int MPI_Query_thread (int *provided)
```

Fortran バインディング

```
call MPI_QUERY_THREAD (PROVIDED, IERROR)

    INTEGER         PROVIDED, IERROR
```

MPI_Status_set_cancelled (C)　MPI_STATUS_SET_CANCELLED (Fortran)

通信状態の不透明部分 cancelled の設定

基本構文

MPI_STATUS_SET_CANCELLED (status, flag)

引数	値	説明	IN/OUT
status	status	flag に関連する通信状態	INOUT
flag	整数	真ならば，要求がキャンセルされるを意味する	IN

C バインディング

```
int MPI_Status_set_cancelled (MPI_Status *status, int flag)
```

Fortran バインディング

```
call MPI_STATUS_SET_CANCELLED (STATUS, FLAG, IERROR)

    INTEGER         STATUS(MPI_STATUS_SIZE), IERROR
    LOGICAL         FLAG
```

MPI_Status_set_elements (C)　MPI_STATUS_SET_ELEMENTS (Fortran)

通信状態中の要素の個数の値の設定

基本構文

MPI_STATUS_SET_ELEMENTS (status, datatype, count)

引数	値	説明	IN/OUT
status	status	通信状態	INOUT
datatype	handle	count に関連するデータ型	IN
count	整数	status に関連する要素の個数	IN

C バインディング

```
int MPI_Status_set_elements (MPI_Status *status, MPI_Datatype datatype, int count)
```

Fortran バインディング

```
call MPI_STATUS_SET_ELEMENTS (STATUS, DATATYPE, COUNT, IERROR)

    INTEGER         STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR
```

通信状態中の要素の個数の値の設定

基本構文

MPI_STATUS_SET_ELEMENTS_X (status, datatype, count)

引数	値	説明	IN/OUT
status	status	通信状態	INOUT
datatype	handle	count に関連するデータ型	IN
count	整数	status に関連する要素の個数	IN

C バインディング

```
int MPI_Status_set_elements_x (MPI_Status *status, MPI_Datatype datatype, MPI_Count
                               *count)
```

Fortran バインディング

```
call MPI_STATUS_SET_ELEMENTS_X (STATUS, DATATYPE, COUNT, IERROR)

    INTEGER          STATUS(MPI_STATUS_SIZE), DATATYPE, COUNT, IERROR
```

4.11 入出力

MPI_FILE_CLOSE	MPI_FILE_DELETE	MPI_FILE_GET_AMODE
MPI_FILE_GET_ATOMICITY	MPI_FILE_GET_BYTE_OFFSET	MPI_FILE_GET_GROUP
MPI_FILE_GET_INFO	MPI_FILE_GET_POSITION	MPI_FILE_GET_POSITION_SHARED
MPI_FILE_GET_SIZE	MPI_FILE_GET_TYPE_EXTENT	MPI_FILE_GET_VIEW
MPI_FILE_IREAD	MPI_FILE_IREAD_AT	MPI_FILE_IREAD_SHARED
MPI_FILE_IWRITE	MPI_FILE_IWRITE_AT	MPI_FILE_IWRITE_SHARED
MPI_FILE_OPEN	MPI_FILE_PREALLOCATE	MPI_FILE_READ
MPI_FILE_READ_ALL	MPI_FILE_READ_ALL_BEGIN	MPI_FILE_READ_ALL_END
MPI_FILE_READ_AT	MPI_FILE_READ_AT_ALL	MPI_FILE_READ_AT_ALL_BEGIN
MPI_FILE_READ_AT_ALL_END	MPI_FILE_READ_ORDERED	MPI_FILE_READ_ORDERED_BEGIN
MPI_FILE_READ_ORDERED_END	MPI_FILE_READ_SHARED	MPI_FILE_SEEK
MPI_FILE_SEEK_SHARED	MPI_FILE_SET_ATOMICITY	MPI_FILE_SET_INFO
MPI_FILE_SET_SIZE	MPI_FILE_SET_VIEW	MPI_FILE_SYNC
MPI_FILE_WRITE	MPI_FILE_WRITE_ALL	MPI_FILE_WRITE_ALL_BEGIN
MPI_FILE_WRITE_ALL_END	MPI_FILE_WRITE_AT	MPI_FILE_WRITE_AT_ALL
MPI_FILE_WRITE_AT_ALL_BEGIN	MPI_FILE_WRITE_AT_ALL_END	MPI_FILE_WRITE_ORDERED
MPI_FILE_WRITE_ORDERED_BEGIN	MPI_FILE_WRITE_ORDERED_END	MPI_FILE_WRITE_SHARED
MPI_REGISTER_DATAREP	MPI_FILE_IREAD_AT_ALL	MPI_FILE_IWRITE_AT_ALL
MPI_FILE_IREAD_ALL	MPI_FILE_IWRITE_ALL	

MPI_File_close (C) MPI_FILE_CLOSE (Fortran)

ファイルのクローズ

基本構文

MPI_FILE_CLOSE (fh)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT

C バインディング

```
int MPI_File_close (MPI_File *fh)
```

Fortran バインディング

```
call MPI_FILE_CLOSE (FH, IERROR)

    INTEGER         FH, IERROR
```

MPI_File_delete (C) MPI_FILE_DELETE (Fortran)

ファイルの削除

基本構文

MPI_FILE_DELETE (filename, info)

引数	値	説明	IN/OUT
filename	文字	ファイル名(文字列)	IN
info	handle	info オブジェクト	IN

C バインディング

```
int MPI_File_delete (char *filename, MPI_Info info)
```

Fortran バインディング

```
call MPI_FILE_DELETE (FILENAME, INFO, IERROR)

    CHARACTER*(*)   FILENAME
    INTEGER         INFO, IERROR
```

MPI_File_get_amode (C)　　MPI_FILE_GET_AMODE (Fortran)

ファイルアクセスモードの取得

基本構文

MPI_FILE_GET_AMODE (fh, amode)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
amode	整数	ファイルオープン時に使用したファイルアクセスモード	OUT

C バインディング

int MPI_File_get_amode (MPI_File fh, int *amode)

Fortran バインディング

call MPI_FILE_GET_AMODE (FH, AMODE, IERROR)

INTEGER　　FH, AMODE, IERROR

MPI_File_get_atomicity (C)　　MPI_FILE_GET_ATOMICITY (Fortran)

アトミックモードの取得

基本構文

MPI_FILE_GET_ATOMICITY (fh, flag)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
flag	論理	アトミックモードの時 真, 非アトミックモードの時 偽	OUT

C バインディング

int MPI_File_get_atomicity (MPI_File fh, int *flag)

Fortran バインディング

call MPI_FILE_GET_ATOMICITY (FH, FLAG, IERROR)

INTEGER　　FH, IERROR
LOGICAL　　FLAG

MPI_File_get_byte_offset (C) | MPI_FILE_GET_BYTE_OFFSET (Fortran)

ビュー相対オフセットから絶対バイト位置への変換

基本構文

MPI_FILE_GET_BYTE_OFFSET (fh, offset, disp)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
disp	整数	ファイルオフセットの絶対バイト位置	OUT

C バインディング

```
int MPI_File_get_byte_offset (MPI_File fh, MPI_Offset offset, MPI_Offset *disp)
```

Fortran バインディング

```
call MPI_FILE_GET_BYTE_OFFSET (FH, OFFSET, DISP, IERROR)

    INTEGER                                FH, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET, DISP
```

MPI_File_get_group (C) | MPI_FILE_GET_GROUP (Fortran)

ファイルオープン時に使用したコミュニケーターのグループの複製の取得

基本構文

MPI_FILE_GET_GROUP (fh, group)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
group	handle	ファイルをオープンしたグループ	OUT

C バインディング

```
int MPI_File_get_group (MPI_File fh, MPI_Group *group)
```

Fortran バインディング

```
call MPI_FILE_GET_GROUP (FH, GROUP, IERROR)

    INTEGER          FH, GROUP, IERROR
```

MPI_File_get_info (C) MPI_FILE_GET_INFO (Fortran)

ファイルハンドルに対応するファイルのヒントの取得

基本構文

MPI_FILE_GET_INFO (fh, info_used)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
info_used	handle	新しい info オブジェクト	OUT

C バインディング

```
int MPI_File_get_info (MPI_File fh, MPI_Info *info_used)
```

Fortran バインディング

```
call MPI_FILE_GET_INFO (FH, INFO_USED, IERROR)

    INTEGER             FH, INFO_USED, IERROR
```

MPI_File_get_position (C) MPI_FILE_GET_POSITION (Fortran)

個別ファイルポインタの現在位置のビュー相対オフセットの取得

基本構文

MPI_FILE_GET_POSITION (fh, offset)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	個別ファイルポインタのオフセット	OUT

C バインディング

```
int MPI_File_get_position (MPI_File fh, MPI_Offset *offset)
```

Fortran バインディング

```
call MPI_FILE_GET_POSITION (FH, OFFSET, IERROR)

    INTEGER                              FH, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

MPI_File_get_position_shared (C) | MPI_FILE_GET_POSITION_SHARED (Fortran)

共有ファイルポインタの現在位置のビュー相対オフセットの取得

基本構文

MPI_FILE_GET_POSITION_SHARED (fh, offset)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	共有ファイルポインタのオフセット	OUT

C バインディング

int MPI_File_get_position_shared (MPI_File fh, MPI_Offset *offset)

Fortran バインディング

call MPI_FILE_GET_POSITION_SHARED (FH, OFFSET, IERROR)

INTEGER FH, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET

MPI_File_get_size (C) | MPI_FILE_GET_SIZE (Fortran)

fh に対応するファイルの大きさの取得

基本構文

MPI_FILE_GET_SIZE (fh, size)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
size	整数	ファイルの大きさ(バイト単位)	OUT

C バインディング

int MPI_File_get_size (MPI_File fh, MPI_Offset *size)

Fortran バインディング

call MPI_FILE_GET_SIZE (FH, SIZE, IERROR)

INTEGER FH, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) SIZE

MPI_File_get_type_extent (C) MPI_FILE_GET_TYPE_EXTENT (Fortran)

ファイルハンドルに対応するファイルのデータ型の寸法の取得

基本構文

MPI_FILE_GET_TYPE_EXTENT (fh, datatype, extent)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
datatype	handle	データ型	IN
extent	整数	データ型の寸法	OUT

C バインディング

int MPI_File_get_type_extent (MPI_File fh, MPI_Datatype datatype, MPI_Aint *extent)

Fortran バインディング

call MPI_FILE_GET_TYPE_EXTENT (FH, DATATYPE, EXTENT, IERROR)

INTEGER FH, DATATYPE, IERROR
INTEGER(KIND=MPI_ADDRESS_KIND) EXTENT

MPI_File_get_view (C) MPI_FILE_GET_VIEW (Fortran)

ファイル中でプロセスのもつデータのビューの取得

基本構文

MPI_FILE_GET_VIEW (fh, disp, etype, filetype, datarep)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
disp	整数	変位	OUT
etype	handle	基本データ型	OUT
filetype	handle	ファイル型	OUT
datarep	文字	データ表現(文字列)	OUT

C バインディング

int MPI_File_get_view (MPI_File fh, MPI_Offset *disp, MPI_Datatype *etype, MPI_Datatype *filetype, char *datarep)

Fortran バインディング

call MPI_FILE_GET_VIEW (FH, DISP, ETYPE, FILETYPE, DATAREP, IERROR)

INTEGER FH, ETYPE, FILETYPE, IERROR
CHARACTER*(*) DATAREP
INTEGER(KIND=MPI_OFFSET_KIND) DISP

個別ファイルポインタ利用による非ブロッキングファイル入力

基本構文

MPI_FILE_IREAD (fh, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iread (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Request
                    *request)
```

Fortran バインディング

```
call MPI_FILE_IREAD (FH, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

明示的オフセット指定による非ブロッキングファイル入力

基本構文

MPI_FILE_IREAD_AT (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iread_at (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
                       datatype, MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IREAD_AT (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型                              BUF(*)
    INTEGER                               FH, COUNT, DATATYPE, REQUEST, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

共有ファイルポインタ利用による非ブロッキングファイル入力

基本構文

MPI_FILE_IREAD_SHARED (fh, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iread_shared (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                           MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IREAD_SHARED (FH, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

個別ファイルポインタ利用による非ブロッキングファイル出力

基本構文

MPI_FILE_IWRITE (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iwrite (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
                     datatype, MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IWRITE (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

明示的オフセット指定による非ブロッキングファイル出力

基本構文

MPI_FILE_IWRITE_AT (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iwrite_at (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
                        datatype, MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IWRITE_AT (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型                              BUF(*)
    INTEGER                               FH, COUNT, DATATYPE, REQUEST, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

共有ファイルポインタ利用による非ブロッキングファイル出力

基本構文

MPI_FILE_IWRITE_SHARED (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	request オブジェクト	OUT

C バインディング

```
int MPI_File_iwrite_shared (MPI_File fh, MPI_Offset offset, void *buf, int count,
                            MPI_Datatype datatype, MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IWRITE_SHARED (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST,
                             IERROR)
```

```
任意の型        BUF(*)
INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

MPI_File_open (C) MPI_FILE_OPEN (Fortran)

ファイルのオープン

基本構文

MPI_FILE_OPEN (comm, filename, amode, info, fh)

引数	値	説明	IN/OUT
comm	handle	コミュニケーター	IN
filename	文字	ファイル名(文字列)	IN
amode	整数	ファイルアクセスモード	IN
info	handle	info オブジェクト	IN
fh	handle	新しいファイルハンドル	OUT

C バインディング

```
int MPI_File_open (MPI_Comm comm, char *filename, int amode, MPI_Info info, MPI_File
                   *fh)
```

Fortran バインディング

```
call MPI_FILE_OPEN (COMM, FILENAME, AMODE, INFO, FH, IERROR)

    CHARACTER*(*)   FILENAME
    INTEGER         COMM, AMODE, INFO, FH, IERROR
```

MPI_File_preallocate (C) MPI_FILE_PREALLOCATE (Fortran)

ファイルハンドルに対応するファイルの記憶領域の確保

基本構文

MPI_FILE_PREALLOCATE (fh, size)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
size	整数	確保するファイルの大きさ(バイト単位)	IN

C バインディング

```
int MPI_File_preallocate (MPI_File fh, MPI_Offset size)
```

Fortran バインディング

```
call MPI_FILE_PREALLOCATE (FH, SIZE, IERROR)

    INTEGER                         FH, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND)   SIZE
```

個別ファイルポインタ利用によるブロッキングファイル入力

基本構文

MPI_FILE_READ (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_read (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status
                   *status)
```

Fortran バインディング

```
call MPI_FILE_READ (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
```

個別ファイルポインタ利用によるブロッキング集団ファイル入力

基本構文

MPI_FILE_READ_ALL (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_read_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status
                       *status)
```

Fortran バインディング

```
call MPI_FILE_READ_ALL (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
```

MPI_File_read_all_begin (C)　　MPI_FILE_READ_ALL_BEGIN (Fortran)

個別ファイルポインタ利用による非ブロッキング集団ファイル入力の開始

基本構文

MPI_FILE_READ_ALL_BEGIN (fh, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

int MPI_File_read_all_begin (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_READ_ALL_BEGIN (FH, BUF, COUNT, DATATYPE, IERROR)

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, IERROR

MPI_File_read_all_end (C)　　MPI_FILE_READ_ALL_END (Fortran)

個別ファイルポインタ利用による非ブロッキング集団ファイル入力の終了

基本構文

MPI_FILE_READ_ALL_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_read_all_end (MPI_File fh, void *buf, MPI_Status *status)

Fortran バインディング

call MPI_FILE_READ_ALL_END (FH, BUF, STATUS, IERROR)

任意の型	BUF(*)
INTEGER	FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_read_at (C) MPI_FILE_READ_AT (Fortran)

明示的オフセット指定によるブロッキングファイル入力

基本構文

MPI_FILE_READ_AT (fh, offset, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_read_at (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
                      datatype, MPI_Status *status)
```

Fortran バインディング

```
call MPI_FILE_READ_AT (FH, OFFSET, BUF, COUNT, DATATYPE, STATUS, IERROR)
```

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
INTEGER(KIND=MPI_OFFSET_KIND)	OFFSET

明示的オフセット指定によるブロッキング集団ファイル入力

基本構文

MPI_FILE_READ_AT_ALL (fh, offset, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_read_at_all (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
datatype, MPI_Status *status)
```

Fortran バインディング

```
call MPI_FILE_READ_AT_ALL (FH, OFFSET, BUF, COUNT, DATATYPE, STATUS,
IERROR)
```

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
INTEGER(KIND=MPI_OFFSET_KIND)	OFFSET

MPI_File_read_at_all_begin (C)　　MPI_FILE_READ_AT_ALL_BEGIN (Fortran)

明示的オフセット指定による非ブロッキング集団ファイル入力の開始

基本構文

MPI_FILE_READ_AT_ALL_BEGIN (fh, offset, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

```
int MPI_File_read_at_all_begin (MPI_File fh, MPI_Offset offset, void *buf, int count,
                                MPI_Datatype datatype)
```

Fortran バインディング

```
call MPI_FILE_READ_AT_ALL_BEGIN (FH, OFFSET, BUF, COUNT, DATATYPE, IERROR)

    任意の型                              BUF(*)
    INTEGER                               FH, COUNT, DATATYPE, IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

MPI_File_read_at_all_end (C)　　MPI_FILE_READ_AT_ALL_END (Fortran)

明示的オフセット指定による非ブロッキング集団ファイル入力の終了

基本構文

MPI_FILE_READ_AT_ALL_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
buf	任意	バッファの先頭アドレス	OUT
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_read_at_all_end (MPI_File fh, void *buf, MPI_Status *status)
```

Fortran バインディング

call MPI_FILE_READ_AT_ALL_END (FH, BUF, STATUS, IERROR)

任意の型	BUF(*)
INTEGER	FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_read_ordered (C)

MPI_FILE_READ_ORDERED (Fortran)

共有ファイルポインタ利用によるブロッキング集団ファイル入力

基本構文

MPI_FILE_READ_ORDERED (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_read_ordered (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
MPI_Status *status)

Fortran バインディング

call MPI_FILE_READ_ORDERED (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_read_ordered_begin (C) | MPI_FILE_READ_ORDERED_BEGIN (Fortran)

共有ファイルポインタ利用による非ブロッキング集団ファイル入力の開始

基本構文

MPI_FILE_READ_ORDERED_BEGIN (fh, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

int MPI_File_read_ordered_begin (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_READ_ORDERED_BEGIN (FH, BUF, COUNT, DATATYPE, IERROR)

任意の型 BUF(*)
INTEGER FH, COUNT, DATATYPE, IERROR

MPI_File_read_ordered_end (C) | MPI_FILE_READ_ORDERED_END (Fortran)

共有ファイルポインタ利用による非ブロッキング集団ファイル入力の終了

基本構文

MPI_FILE_READ_ORDERED_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_read_ordered_end (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_READ_ORDERED_END (FH, BUF, STATUS, IERROR)

任意の型	BUF(*)
INTEGER	FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_read_shared (C) | MPI_FILE_READ_SHARED (Fortran)

共有ファイルポインタ利用によるブロッキングファイル入力

基本構文

MPI_FILE_READ_SHARED (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_read_shared (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Fortran バインディング

call MPI_FILE_READ_SHARED (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_seek (C) | MPI_FILE_SEEK (Fortran)

個別ファイルポインタの更新

基本構文

MPI_FILE_SEEK (fh, offset, whence)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	個別ファイルポインタのオフセット	IN
whence	state	更新モード	IN

C バインディング

int MPI_File_seek (MPI_File fh, MPI_Offset offset, int whence)

Fortran バインディング

call MPI_FILE_SEEK (FH, OFFSET, WHENCE, IERROR)

INTEGER FH, WHENCE, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET

MPI_File_seek_shared (C) MPI_FILE_SEEK_SHARED (Fortran)

共有ファイルポインタの更新

基本構文

MPI_FILE_SEEK_SHARED (fh, offset, whence)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	共有ファイルポインタのオフセット	IN
whence	state	更新モード	IN

C バインディング

int MPI_File_seek_shared (MPI_File fh, MPI_Offset offset, int whence)

Fortran バインディング

call MPI_FILE_SEEK_SHARED (FH, OFFSET, WHENCE, IERROR)

INTEGER FH, WHENCE, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET

MPI_File_set_atomicity (C) MPI_FILE_SET_ATOMICITY (Fortran)

アトミックモードの設定

基本構文

MPI_FILE_SET_ATOMICITY (fh, flag)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
flag	論理	真の時はアトミックモード，偽の時は非アトミックモード	IN

C バインディング

int MPI_File_set_atomicity (MPI_File fh, int flag)

Fortran バインディング

```
call MPI_FILE_SET_ATOMICITY (FH, FLAG, IERROR)

    INTEGER         FH, IERROR
    LOGICAL         FLAG
```

MPI_File_set_info (C) MPI_FILE_SET_INFO (Fortran)

ファイルハンドルに対応するファイルのヒントの設定

基本構文

MPI_FILE_SET_INFO (fh, info)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
info	handle	info オブジェクト	IN

C バインディング

```
int MPI_File_set_info (MPI_File fh, MPI_Info info)
```

Fortran バインディング

```
call MPI_FILE_SET_INFO (FH, INFO, IERROR)

    INTEGER         FH, INFO, IERROR
```

MPI_File_set_size (C) MPI_FILE_SET_SIZE (Fortran)

ファイルハンドルに対応するファイルの大きさ変更

基本構文

MPI_FILE_SET_SIZE (fh, size)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
size	整数	ファイル大きさ	IN

C バインディング

```
int MPI_File_set_size (MPI_File fh, MPI_Offset size)
```

Fortran バインディング

call MPI_FILE_SET_SIZE (FH, SIZE, IERROR)

```
INTEGER                                 FH, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) SIZE
```

MPI_File_set_view (C)　　MPI_FILE_SET_VIEW (Fortran)

ファイル中でプロセスのもつデータのビューの変更

基本構文

MPI_FILE_SET_VIEW (fh, disp, etype, filetype, datarep, info)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
disp	整数	変位	IN
etype	handle	基本データ型	IN
filetype	handle	ファイル型	IN
datarep	文字	データ表現(文字列)	IN
info	handle	info オブジェクト	IN

C バインディング

```
int MPI_File_set_view (MPI_File fh, MPI_Offset disp, MPI_Datatype etype, MPI_Datatype
                       filetype, char *datarep, MPI_Info info)
```

Fortran バインディング

call MPI_FILE_SET_VIEW (FH, DISP, ETYPE, FILETYPE, DATAREP, INFO, IERROR)

```
INTEGER                                 FH, ETYPE, FILETYPE, INFO, IERROR
CHARACTER*(*)                           DATAREP
INTEGER(KIND=MPI_OFFSET_KIND) DISP
```

MPI_File_sync (C)　　MPI_FILE_SYNC (Fortran)

ファイルハンドルに対応するファイルにおける更新データの記憶装置への転送

基本構文

MPI_FILE_SYNC (fh)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT

C バインディング

int MPI_File_sync (MPI_File fh)

Fortran バインディング

call MPI_FILE_SYNC (FH, IERROR)

INTEGER FH, IERROR

MPI_File_write (C) MPI_FILE_WRITE (Fortran)

個別ファイルポインタ利用によるブロッキングファイル出力

基本構文

MPI_FILE_WRITE (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_write (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status *status)

Fortran バインディング

call MPI_FILE_WRITE (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

任意の型 BUF(*)
INTEGER FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_write_all (C)　　MPI_FILE_WRITE_ALL (Fortran)

個別ファイルポインタ利用によるブロッキング集団ファイル出力

基本構文

MPI_FILE_WRITE_ALL (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_write_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype, MPI_Status
    *status)
```

Fortran バインディング

```
call MPI_FILE_WRITE_ALL (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
```

MPI_File_write_all_begin (C)　　MPI_FILE_WRITE_ALL_BEGIN (Fortran)

個別ファイルポインタ利用による非ブロッキング集団ファイル出力の開始

基本構文

MPI_FILE_WRITE_ALL_BEGIN (fh, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

int MPI_File_write_all_begin (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_WRITE_ALL_BEGIN (FH, BUF, COUNT, DATATYPE, IERROR)

任意の型　BUF(*)
INTEGER　FH, COUNT, DATATYPE, IERROR

MPI_File_write_all_end (C)　　MPI_FILE_WRITE_ALL_END (Fortran)

個別ファイルポインタ利用による非ブロッキング集団ファイル出力の終了

基本構文

MPI_FILE_WRITE_ALL_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_write_all_end (MPI_File fh, void *buf, MPI_Status *status)

Fortran バインディング

call MPI_FILE_WRITE_ALL_END (FH, BUF, STATUS, IERROR)

任意の型　BUF(*)
INTEGER　FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_write_at (C) MPI_FILE_WRITE_AT (Fortran)

明示的オフセット指定によるブロッキングファイル出力

基本構文

MPI_FILE_WRITE_AT (fh, offset, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_write_at (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
                       datatype, MPI_Status *status)
```

Fortran バインディング

```
call MPI_FILE_WRITE_AT (FH, OFFSET, BUF, COUNT, DATATYPE, STATUS, IERROR)

    任意の型                              BUF(*)
    INTEGER                               FH, COUNT, DATATYPE,
                                          STATUS(MPI_STATUS_SIZE), IERROR
    INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

MPI_File_write_at_all (C)

MPI_FILE_WRITE_AT_ALL (Fortran)

明示的オフセット指定によるブロッキング集団ファイル出力

基本構文

MPI_FILE_WRITE_AT_ALL (fh, offset, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_write_at_all (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
datatype, MPI_Status *status)
```

Fortran バインディング

```
call MPI_FILE_WRITE_AT_ALL (FH, OFFSET, BUF, COUNT, DATATYPE, STATUS,
IERROR)
```

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
INTEGER(KIND=MPI_OFFSET_KIND)	OFFSET

MPI_File_write_at_all_begin (C) MPI_FILE_WRITE_AT_ALL_BEGIN (Fortran)

明示的オフセット指定による非ブロッキング集団ファイル出力の開始

基本構文

MPI_FILE_WRITE_AT_ALL_BEGIN (fh, offset, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

int MPI_File_write_at_all_begin (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_WRITE_AT_ALL_BEGIN (FH, OFFSET, BUF, COUNT, DATATYPE, IERROR)

任意の型 BUF(*)
INTEGER FH, COUNT, DATATYPE, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET

MPI_File_write_at_all_end (C) MPI_FILE_WRITE_AT_ALL_END (Fortran)

明示的オフセット指定による非ブロッキング集団ファイル出力の終了

基本構文

MPI_FILE_WRITE_AT_ALL_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_write_at_all_end (MPI_File fh, void *buf, MPI_Status *status)

Fortran バインディング

call MPI_FILE_WRITE_AT_ALL_END (FH, BUF, STATUS, IERROR)

任意の型 BUF(*)
INTEGER FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_write_ordered (C) MPI_FILE_WRITE_ORDERED (Fortran)

共有ファイルポインタ利用によるブロッキング集団ファイル出力

基本構文

MPI_FILE_WRITE_ORDERED (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_write_ordered (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
MPI_Status *status)

Fortran バインディング

call MPI_FILE_WRITE_ORDERED (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

任意の型 BUF(*)
INTEGER FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_write_ordered_begin (C) MPI_FILE_WRITE_ORDERED_BEGIN (Fortran)

共有ファイルポインタ利用による非ブロッキング集団ファイル出力の開始

基本構文

MPI_FILE_WRITE_ORDERED_BEGIN (fh, buf, count, datatype)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN

C バインディング

int MPI_File_write_ordered_begin (MPI_File fh, void *buf, int count, MPI_Datatype datatype)

Fortran バインディング

call MPI_FILE_WRITE_ORDERED_BEGIN (FH, BUF, COUNT, DATATYPE, IERROR)

任意の型　BUF(*)
INTEGER　FH, COUNT, DATATYPE, IERROR

MPI_File_write_ordered_end (C)　MPI_FILE_WRITE_ORDERED_END (Fortran)

共有ファイルポインタ利用による非ブロッキング集団ファイル出力の終了

基本構文

MPI_FILE_WRITE_ORDERED_END (fh, buf, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
status	status	状態オブジェクト	OUT

C バインディング

int MPI_File_write_ordered_end (MPI_File fh, void *buf, MPI_Status *status)

Fortran バインディング

call MPI_FILE_WRITE_ORDERED_END (FH, BUF, STATUS, IERROR)

任意の型　BUF(*)
INTEGER　FH, STATUS(MPI_STATUS_SIZE), IERROR

MPI_File_write_shared (C)　MPI_FILE_WRITE_SHARED (Fortran)

共有ファイルポインタ利用によるブロッキングファイル出力

基本構文

MPI_FILE_WRITE_SHARED (fh, buf, count, datatype, status)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
status	status	状態オブジェクト	OUT

C バインディング

```
int MPI_File_write_shared (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                           MPI_Status *status)
```

Fortran バインディング

```
call MPI_FILE_WRITE_SHARED (FH, BUF, COUNT, DATATYPE, STATUS, IERROR)

    任意の型          BUF(*)
    INTEGER           FH, COUNT, DATATYPE, STATUS(MPI_STATUS_SIZE), IERROR
```

MPI_Register_datarep (C)　　MPI_REGISTER_DATAREP (Fortran)

データ表現の定義

基本構文

```
MPI_REGISTER_DATAREP (datarep, read_conversion_fn, write_conversion_fn,
                      dtype_file_extent_fn, extra_state)
```

引数	値	説明	IN/OUT
datarep	文字	データ表現識別子	IN
read_conversion_fn	関数	ファイル表現からネイティブ表現への変換時に呼ばれる関数	IN
write_conversion_fn	関数	ネイティブ表現からファイル表現への変換時に呼ばれる関数	IN
type_file_extent_fn	関数	ファイルのデータ型の寸法取得時に呼ばれる関数	IN
extra_state	状態	拡張状態	IN

C バインディング

```
int MPI_Register_datarep (char *datarep, MPI_Datarep_conversion_function
                          *read_conversion_fn, MPI_Datarep_conversion_function
                          *write_conversion_fn, MPI_Datarep_extent_function
                          *dtype_file_extent_fn, void *extra_state)
```

Fortran バインディング

```
call MPI_REGISTER_DATAREP (DATAREP, READ_CONVERSION_FN,
                           WRITE_CONVERSION_FN, DTYPE_FILE_EXTENT_FN,
                           EXTRA_STATE, IERROR)

    CHARACTER*(*)                        DATAREP
    EXTERNAL                             READ_CONVERSION_FN,
                                         WRITE_CONVERSION_FN,
                                         DTYPE_FILE_EXTENT_FN
    INTEGER(KIND=MPI_ADDRESS_KIND)       EXTRA_STATE
    INTEGER                              IERROR
```

MPI_File_iread_at_all (C)

MPI_FILE_IREAD_AT_ALL (Fortran)

手続 MPI_File_read_at_all の非ブロッキング版

基本構文

MPI_FILE_IREAD_AT_ALL (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	IN
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	リクエスト実体	OUT

C バインディング

```
int MPI_File_iread_at_all (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype
datatype, MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IREAD_AT_ALL (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST,
IERROR)

任意の型                              BUF(*)
INTEGER                               FH, COUNT, DATATYPE, REQUEST, IERROR
INTEGER(KIND=MPI_OFFSET_KIND) OFFSET
```

MPI_File_iwrite_at_all (C)

MPI_FILE_IWRITE_AT_ALL (Fortran)

手続 MPI_File_write_at_all の非ブロッキング版

基本構文

MPI_FILE_IWRITE_AT_ALL (fh, offset, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
offset	整数	ファイルオフセット	IN
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	リクエスト実体	OUT

C バインディング

int MPI_File_iwrite_at_all (MPI_File fh, MPI_Offset offset, void *buf, int count, MPI_Datatype datatype, MPI_Request *request)

Fortran バインディング

call MPI_FILE_IWRITE_AT_ALL (FH, OFFSET, BUF, COUNT, DATATYPE, REQUEST, IERROR)

任意の型	BUF(*)
INTEGER	FH, COUNT, DATATYPE, REQUEST, IERROR
INTEGER(KIND=MPI_OFFSET_KIND)	OFFSET

MPI_File_iread_all (C) MPI_FILE_IREAD_ALL (Fortran)

手続 MPI_File_read_all の非ブロッキング版

基本構文

MPI_FILE_IREAD_ALL (fh, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	OUT
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	リクエスト実体	OUT

C バインディング

```
int MPI_File_iread_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                        MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IREAD_ALL (FH, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

手続 MPI_File_write_all の非ブロッキング版

基本構文

MPI_FILE_IWRITE_ALL (fh, buf, count, datatype, request)

引数	値	説明	IN/OUT
fh	handle	ファイルハンドル	INOUT
buf	任意	バッファの先頭アドレス	IN
count	整数	バッファの要素の個数	IN
datatype	handle	バッファのデータ型	IN
request	handle	リクエスト実体	OUT

C バインディング

```
int MPI_File_iwrite_all (MPI_File fh, void *buf, int count, MPI_Datatype datatype,
                         MPI_Request *request)
```

Fortran バインディング

```
call MPI_FILE_IWRITE_ALL (FH, BUF, COUNT, DATATYPE, REQUEST, IERROR)

    任意の型        BUF(*)
    INTEGER         FH, COUNT, DATATYPE, REQUEST, IERROR
```

4.12 言語バインディング

MPI_SIZEOF	MPI_TYPE_CREATE_F90_COMPLEX	MPI_TYPE_CREATE_F90_INTEGER
MPI_TYPE_CREATE_F90_REAL	MPI_TYPE_MATCH_SIZE	

MPI_Sizeof (C) MPI_SIZEOF (Fortran)

変数のデータ型の大きさ(バイト単位)の返却

基本構文

MPI_SIZEOF (x, size)

引数	値	説明	IN/OUT
x	任意	Fortran 言語の組込み数値型の変数	IN
size	整数	データ型の大きさ(バイト単位)	OUT

Fortran バインディング

call MPI_SIZEOF (X, SIZE, IERROR)

任意 X
INTEGER SIZE, IERROR

MPI_Type_create_f90_complex (C) | MPI_TYPE_CREATE_F90_COMPLEX (Fortran)

種別型パラメタ selected_real_kind(p,r)となる複素数型に一致する定義済み MPI データ型の返却

基本構文

MPI_TYPE_CREATE_F90_COMPLEX (p, r, newtype)

引数	値	説明	IN/OUT
p	整数	精度，十進数	IN
r	整数	十進数の指数レンジ	IN
newtype	handle	要求された MPI データ型	OUT

C バインディング

```
int MPI_Type_create_f90_complex (int p, int r, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_F90_COMPLEX (P, R, NEWTYPE, IERROR)

    INTEGER          P, R, NEWTYPE, IERROR
```

MPI_Type_create_f90_integer (C) | MPI_TYPE_CREATE_F90_INTEGER (Fortran)

種別型パラメタ selected_int_kind(p,r)となる整数型に一致する定義済み MPI データ型の返却

基本構文

MPI_TYPE_CREATE_F90_INTEGER (r, newtype)

引数	値	説明	IN/OUT
r	整数	十進数の指数レンジ(たとえば十進数の桁数)	IN
newtype	handle	要求された MPI データ型	OUT

C バインディング

```
int MPI_Type_create_f90_integer (int r, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_F90_INTEGER (R, NEWTYPE, IERROR)

    INTEGER          R, NEWTYPE, IERROR
```

MPI_Type_create_f90_real (C) | MPI_TYPE_CREATE_F90_REAL (Fortran)

種別型パラメタ selected_real_kind(p,r)となる実数型に一致する定義済み MPI データ型の返却

基本構文

MPI_TYPE_CREATE_F90_REAL (p, r, newtype)

引数	値	説明	IN/OUT
p	整数	精度，十進数	IN
r	整数	十進数の指数レンジ	IN
newtype	handle	要求された MPI データ型	OUT

C バインディング

```
int MPI_Type_create_f90_real (int p, int r, MPI_Datatype *newtype)
```

Fortran バインディング

```
call MPI_TYPE_CREATE_F90_REAL (P, R, NEWTYPE, IERROR)

    INTEGER          P, R, NEWTYPE, IERROR
```

MPI_Type_match_size (C) | MPI_TYPE_MATCH_SIZE (Fortran)

typeclass, size に一致する定義済み MPI データ型の返却

基本構文

MPI_TYPE_MATCH_SIZE (typeclass, size, type)

引数	値	説明	IN/OUT
typeclass	整数	一般型識別子	IN
size	整数	データ型の大きさ(バイト単位)	IN
type	handle	データ型	OUT

C バインディング

```
int MPI_Type_match_size (int typeclass, int size, MPI_Datatype *type)
```

Fortran バインディング

```
call MPI_TYPE_MATCH_SIZE (TYPECLASS, SIZE, TYPE, IERROR)

    INTEGER          TYPECLASS, SIZE, TYPE, IERROR
```

4.13　プロファイリング

MPI_PCONTROL

MPI_Pcontrol (C)　　MPI_Pcontrol (Fortran)

プロファイリング制御

基本構文

MPI_PCONTROL (level, ...)

引数	値	説明	IN/OUT
level	整数	プロファイリングレベル	IN

C バインディング

int MPI_Pcontrol (int level, ...)

Fortran バインディング

call MPI_PCONTROL (LEVEL)
整数型　　LEVEL

4.14　廃止予定関数

MPI_ATTR_DELETE	MPI_ATTR_GET	MPI_ATTR_PUT
MPI_KEYVAL_CREATE	MPI_KEYVAL_FREE	

MPI_Attr_delete (C)　　MPI_ATTR_DELETE (Fortran)

コミュニケーターから属性を消去

基本構文

MPI_ATTR_DELETE (comm, keyval)

引数	値	説明	IN/OUT
comm	handle	属性消去の対象となるコミュニケーター	IN
keyval	整数	キー値	IN

C バインディング

int MPI_Attr_delete (MPI_Comm comm, int keyval)

Fortran バインディング

call MPI_ATTR_DELETE (COMM, KEYVAL, IERROR)
整数型　　COMM, KEYVAL, IERROR

MPI_Attr_get (C) MPI_ATTR_GET (Fortran)

コミュニケーターの属性を取得

基本構文

MPI_ATTR_GET (comm, keyval, attribute_val, flag)

引数	値	説明	IN/OUT
comm	handle	属性取得の対象となるコミュニケーター	IN
keyval	整数	キー値	IN
attribute_val	属性	コミュニケーターに付加された属性値	OUT
flag	論理	属性値が取得できたか否かのフラグ	OUT

C バインディング

int MPI_Attr_get (MPI_Comm comm, int keyval, void* attribute_val, int *flag)

Fortran バインディング

call MPI_ATTR_GET (COMM, KEYVAL, ATTRIBUTE_VAL, FLAG, IERROR)

整数型 COMM, KEYVAL, ATTRIBUTE_VAL, IERROR

論理型 FLAG

MPI_Attr_put (C) MPI_ATTR_PUT (Fortran)

コミュニケーターの属性の付加

基本構文

MPI_ATTR_PUT (comm, keyval, attribute_val)

引数	値	説明	IN/OUT
comm	handle	属性取得の対象となるコミュニケーター	IN
keyval	整数	属性キー	IN
attribute_val	属性	付加される属性値	IN

C バインディング

int MPI_Attr_put (MPI_Comm comm, int keyval, void* attribute_val)

Fortran バインディング

call MPI_ATTR_PUT (COMM, KEYVAL, ATTRIBUTE_VAL, IERROR)

属性キーの生成

基本構文

MPI_KEYVAL_CREATE (copy_fn, delete_fn, keyval, extra_state)

引数	値	説明	IN/OUT
copy_fn	関数	コミュニケーターの複製を生成する場合に呼ばれるコピー関数	IN
delete_fn	関数	コミュニケーターを破棄する場合に呼ばれる関数	IN
keyval	整数	属性キー	OUT
extra_state	整数	copy_fn, delete_fn に対する補助引数	IN

C バインディング

int MPI_Keyval_create (MPI_Copy_function *copy_fn, MPI_Copy_function *delete_fn, int *keyval, void *extra_state)

typedef int MPI_Copy_function (MPI_Comm oldcomm, int keyval, void *extra_sate, void *attribute_val_in, void *attribute_out, int *flag)

Fortran バインディング

call MPI_KEYVAL_CREATE (COPY_FN, DELETE_FN, KEYVAL, EXTRA_STATE, IERROR)

EXTERNAL COPY_FN, DELETE_FN IERROR

整数型 KEYVAL, EXTRA_STATE, IERROR

SUBROUTINE COPY_FUNCTION (OLDCOMM, KEYVAL, EXTRA_STATE, ATTRIBUTE_VAL_IN, ATTRIBUTE_VAL_OUT, FLAG, IERR)

整数型 OLDCOMM, KEYVAL, EXTRA_STATE, ATTIBUTE_VAL_IN, ATTRIBUTE_VAL_OUT
IERR

論理型 FLAG

属性キーの解放

基本構文

MPI_KEYVAL_FREE (keyval)

引数	値	説明	IN/OUT
keyval	整数	解放対象属性キー	INOUT

C バインディング

int MPI_Keyval_free (int *keyval)

Fortran バインディング

call MPI_KEYVAL_FREE (KEYVAL, IERROR)
整数型 KEYVAL, IERROR

付録 A　MPI 予約名一覧

NEC MPI は，データ型やエラーコードといった MPI 予約名を定義したインクルードファイルを用意しています。 MPI 予約名は，全て接頭辞 MPI_で始まります。インクルードファイル名は，Fortran 言語向けの mpif.h，C 言語向けの mpi.h，および C++言語向けの mpi++.h です。

以下は，インクルードファイルの記述例です。

C 言語の場合

```
#include "mpi.h"
main( argc, argv )
int argc ;
char **argv ;
{
        :
```

Fortran 言語の場合

```
      SUBROUTINE SUB(I,J)
      INCLUDE "mpif.h"
      INTEGER I,J
          :
```

A.1 MPI データ型

NEC MPI が，各プログラミング言語に対して事前定義しているデータ型は，次のとおりです。利用者は，これらのデータ型から，新たに利用者定義データ型を生成することもできます。

NEC MPI は，Fortran コンパイラが用意している型に対応して，次の MPI データ型を用意しています。

MPI データ型	Fortran の型
MPI_INTEGER	INTEGER
MPI_INTEGER2	INTEGER*2
MPI_INTEGER4	INTEGER*4
MPI_INTEGER8	INTEGER*8
MPI_LOGICAL	LOGICAL
MPI_LOGICAL1 (注 1)	LOGICAL*1
MPI_LOGICAL4 (注 1)	LOGICAL*4
MPI_LOGICAL8 (注 1)	LOGICAL*8
MPI_REAL	REAL
MPI_REAL4	REAL*4
MPI_REAL8	REAL*8
MPI_REAL16	REAL*16
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_QUADRUPLE_PRECISION (注 1)	QUADRUPLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_COMPLEX8	COMPLEX*8
MPI_COMPLEX16	COMPLEX*16
MPI_COMPLEX32	COMPLEX*32
MPI_DOUBLE_COMPLEX	DOUBLE COMPLEX
MPI_QUADRUPLE_COMPLEX (注 1)	COMPLEX QUADRUPLE
MPI_CHARACTER	CHARACTER(1)
MPI_AINT	INTEGER(KIND=MPI_ADDRESS_KIND)
MPI_COUNT	INTEGER(KIND=MPI_COUNT_KIND)
MPI_OFFSET	INTEGER(KIND=MPI_OFFSET_KIND)

注 1　NEC MPI 独自のデータ型です。

C 言語の型に対しては，次の MPI データ型を用意しています。

MPI データ型	対応する C の型
MPI_CHAR	char
MPI_SHORT	signed short
MPI_INT	signed int
MPI_LONG	signed long
MPI_LONG_LONG_INT	signed long long
MPI_LONG_LONG	signed long long
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsgined short
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long
MPI_UNSIGNED_LONG_LONG	unsigned long long
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_C_BOOL	_Bool
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPI_AINT	MPI_Aint
MPI_COUNT	MPI_Count
MPI_OFFSET	MPI_Offset
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

C++言語の型に対しては，次の MPI データ型を用意しています。

MPI データ型	C++の型
MPI_CXX_BOOL	bool
MPI_CXX_FLOAT_COMPLEX	std::complex<float>
MPI_CXX_DOUBLE_COMPLEX	std::complex<double>
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double>

また，Fortran 言語 および C 言語共通の MPI データ型として次のものがあります。

MPI データ型	対応する型
MPI_BYTE	任意の型
MPI_PACKED	任意の型

A.2 集計演算用 MPI データ型

Fortran において，集計演算子 MPI_MAXLOC および MPI_MINLOC で使用する MPI データ型として，以下を用意しています。

- MPI_2REAL
- MPI_2DOUBLE_PRECISION
- MPI_2INTEGER
- MPI_2COMPLEX (注 1)
- MPI_2DOUBLE_COMPLEX (注 1)

(注 1) NEC MPI 独自の MPI データ型です。

C において，集計演算子 MPI_MAXLOC および MPI_MINLOC で使用する MPI データ型として，以下を用意しています。

- MPI_FLOAT_INT
- MPI_DOUBLE_INT
- MPI_LONG_INT
- MPI_2INT
- MPI_SHORT_INT
- MPI_LONG_DOUBLE_INT

A.3 エラーコード

MPI 手続の実行中にエラーが発生した場合，NEC MPI が返却するエラーコードは，次の表のとおりです。エラーコードは，Fortran 言語であれば手続呼出しの最後の引数に，C 言語であれば関数値として返却されます。 基本的なエラーコードは，エラークラスと呼ばれ，各エラーコードは，エラークラスの 1 つに属します。

エラーコード	意味
MPI_SUCCESS	正常終了した。
MPI_ERR_BUFFER	不正なバッファアドレスが引数に指定された。
MPI_ERR_COUNT	不正な要素 または ブロックの個数が引数に指定された。
MPI_ERR_TYPE	不正なデータ型が引数に指定された。
MPI_ERR_TAG	不正なタグが引数に指定された。
MPI_ERR_COMM	不正なコミュニケーターが引数に指定された。
MPI_ERR_RANK	不正なランクが引数に指定された。
MPI_ERR_REQUEST	不正な通信識別子が引数に指定された。
MPI_ERR_ROOT	不正なルートが引数に指定された。
MPI_ERR_GROUP	不正なグループの値が引数に指定された。
MPI_ERR_OP	不正な操作要求が引数に指定された。
MPI_ERR_TOPOLOGY	不正なトポロジーが引数に指定された。
MPI_ERR_DIMS	不正な次元が引数に指定された。
MPI_ERR_ARG	不正な引数が指定された。
MPI_ERR_UNKNOWN	NEC MPI では認識できないエラーが発生した。
MPI_ERR_TRUNCATE	受信メッセージが不正に分割された。
MPI_ERR_OTHER	エラーコードで定義していないエラーが発生した。詳細情報は，手続 MPI_ERROR_STRING によって取得できる。
MPI_ERR_INTERN	NEC MPI 処理系で内部的なエラーが発生した。
MPI_ERR_IN_STATUS	引数 STATUS(通信状態)にエラーコードを返却した。
MPI_ERR_PENDING	通信識別子 request による通信要求は，保留状態となっている。
MPI_ERR_KEYVAL	不正な keyval が引数に渡された。
MPI_ERR_NO_MEM	メモリが枯渇したため手続 MPI_ALLOC_MEM が失敗した。
MPI_ERR_BASE	不正な base が手続 MPI_FREE_MEM の引数に渡された。
MPI_ERR_INFO_KEY	Info オブジェクトのキーが MPI_MAX_INFO_KEY よりも長い。
MPI_ERR_INFO_VALUE	Info オブジェクトの値が MPI_MAX_INFO_VALUE よりも長い。
MPI_ERR_INFO_NOKEY	不正なキーが手続 MPI_INFO_DELETE に渡された。
MPI_ERR_SPAWN	動的生成においてプロセス生成時にエラーが発生した。
MPI_ERR_PORT	不正なポート名が手続 MPI_COMM_CONNECT に渡された。

MPI_ERR_SERVICE	不正なサービス名が手続 MPI_UNPUBLISH_NAME に渡された。
MPI_ERR_NAME	不正なサービス名が手続 MPI_LOOKUP_NAME に渡された。
MPI_ERR_WIN	不正なウィンドウ(win)が引数に渡された。
MPI_ERR_SIZE	不正な size が引数に渡された。
MPI_ERR_DISP	不正な disp が引数に渡された。
MPI_ERR_INFO	不正な Info オブジェクト(info)が引数に渡された。
MPI_ERR_LOCKTYPE	不正な locktype が引数に渡された。
MPI_ERR_ASSERT	不正な assert が引数に渡された。
MPI_ERR_RMA_CONFLICT	ウィンドウへのアクセスが衝突している。
MPI_ERR_RMA_SYNC	RMA 通信の同期呼出しが誤っている。
MPI_ERR_RMA_RANGE	ターゲットのメモリがウィンドウ部分でない。
MPI_ERR_RMA_ATTACH	メモリがアタッチできない。
MPI_ERR_RMA_SHARED	メモリが共有できない。
MPI_ERR_RMA_FLAVOR	誤った flavor をもつウィンドウが MPI 手続に渡された。
MPI_ERR_FILE	ファイルハンドルが不正である。
MPI_ERR_NOT_SAME	集団通信の引数が全プロセスで一致しない。 または 集団通信の呼び出し順がプロセスによって異なる。
MPI_ERR_AMODE	手続 MPI_FILE_OPEN に渡された amode に関するエラーが発生した。
MPI_ERR_UNSUPPORTED_DATAREP	未サポートの datarep が手続 MPI_FILE_SET_VIEW に渡された。
MPI_ERR_UNSUPPORTED_OPERATION	未サポートの処理を行った。
MPI_ERR_NO_SUCH_FILE	ファイルが見つからない。
MPI_ERR_FILE_EXISTS	ファイルが既に存在する。
MPI_ERR_BAD_FILE	不正なファイル名が指定された。
MPI_ERR_ACCESS	アクセスが拒否された。
MPI_ERR_NO_SPACE	空き容量が不足している。
MPI_ERR_QUOTA	クォータが超過した。
MPI_ERR_READ_ONLY	ファイル， または ファイルシステムが読み込み専用である。
MPI_ERR_FILE_IN_USE	ファイルが open されているため，処理が完了できなかった。
MPI_ERR_DUP_DATAREP	手続 MPI_REGISTER_DATAREP に，定義済みの data representation identifier が渡されたため，変換関数が登録できなかった。
MPI_ERR_CONVERSION	利用者定義の変換関数でエラーが発生した。
MPI_ERR_IO	その他の I/O エラーが発生した。
MPI_ERR_LASTCODE	エラーコードの上限値(エラーの意味をもたない)。

備考　エラーコードの値は，MPI_SUCCESS(=0)<内部値≦MPI_ERR_LASTCODE の範囲をとります。

A.4 予約コミュニケーター名

NEC MPI は，次の予約コミュニケーター名を用意しています。

コミュニケーター名	意味
MPI_COMM_WORLD	全ての MPI プロセスが属するコミュニケーター
MPI_COMM_SELF	当該 MPI プロセスだけが属するコミュニケーター

A.5 集計演算子

集計演算は，各プロセス上のデータ要素を，1 つの演算子を使用して結合することにより，1 つの値を算出します。集計演算を行う MPI 手続に指定できる事前定義された集計演算子は，次の通りです。

集計演算子	機能
MPI_MAX	最大値
MPI_MIN	最小値
MPI_SUM	総和
MPI_PROD	総積
MPI_MAXLOC	最大値 および その場所
MPI_MINLOC	最小値 および その場所
MPI_BAND	ビット論理積
MPI_BOR	ビット論理和
MPI_BXOR	ビット排他的論理和
MPI_LAND	論理積
MPI_LOR	論理和
MPI_LXOR	排他的論理和

付録 B　著作権・ライセンス

B.1 MPICH 1.1.1

COPYRIGHT

The following is a notice of limited availability of the code, and disclaimer which must be included in the prologue of the code and in all source listings of the code.

Copyright Notice
+ 1993 University of Chicago
+ 1993 Mississippi State University

Permission is hereby granted to use, reproduce, prepare derivative works, and to redistribute to others. This software was authored by:

Argonne National Laboratory Group
W. Gropp: (708) 252-4318; FAX: (708) 252-7852; e-mail: gropp@mcs.anl.gov
E. Lusk: (708) 252-7852; FAX: (708) 252-7852; e-mail: lusk@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory, Argonne IL 60439

Mississippi State Group
N. Doss: (601) 325-2565; FAX: (601) 325-7692; e-mail: doss@erc.msstate.edu
A. Skjellum:(601) 325-8435; FAX: (601) 325-8997; e-mail: tony@erc.msstate.edu
Mississippi State University, Computer Science Department &
NSF Engineering Research Center for Computational Field Simulation
P.O. Box 6176, Mississippi State MS 39762

GOVERNMENT LICENSE

Portions of this material resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

DISCLAIMER

This computer code material was prepared, in part, as an account of work sponsored by an agency of the United States Government. Neither the United States, nor the University of Chicago, nor Mississippi State University, nor any of their employees, makes any warranty express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

B.2 MVAPICH2 2.2b

COPYRIGHT

Copyright (c) 2001-2015, The Ohio State University. All rights reserved.

The MVAPICH2 software package is developed by the team members of The Ohio State University's Network-Based Computing Laboratory (NBCL), headed by Professor Dhabaleswar K. (DK) Panda.

Contact:
Prof. Dhabaleswar K. (DK) Panda
Dept. of Computer Science and Engineering
The Ohio State University
2015 Neil Avenue
Columbus, OH - 43210-1277

Tel: (614)-292-5199; Fax: (614)-292-2911
E-mail:panda@cse.ohio-state.edu

This program is available under BSD licensing.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

(1) Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

(3) Neither the name of The Ohio State University nor the names of their contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

COPYRIGHT

The following is a notice of limited availability of the code, and disclaimer

which must be included in the prologue of the code and in all source listings of the code.

Copyright Notice
+ 2002 University of Chicago

Permission is hereby granted to use, reproduce, prepare derivative works, and to redistribute to others. This software was authored by:

Mathematics and Computer Science Division
Argonne National Laboratory, Argonne IL 60439

(and)

Department of Computer Science
University of Illinois at Urbana-Champaign

GOVERNMENT LICENSE

Portions of this material resulted from work developed under a U.S. Government Contract and are subject to the following license: the Government is granted for itself and others acting on its behalf a paid-up, nonexclusive, irrevocable worldwide license in this computer software to reproduce, prepare derivative works, and perform publicly and display publicly.

DISCLAIMER

This computer code material was prepared, in part, as an account of work sponsored by an agency of the United States Government. Neither the United States, nor the University of Chicago, nor any of their employees, makes any warranty express or implied, or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights.

B.3 ptmalloc2 Jun 5th 2006

```
Copyright (c) 2001-2006 Wolfram Gloger

Permission to use, copy, modify, distribute, and sell this software
and its documentation for any purpose is hereby granted without fee,
provided that (i) the above copyright notices and this permission
notice appear in all copies of the software and related documentation,
and (ii) the name of Wolfram Gloger may not be used in any advertising
or publicity relating to the software.

THE SOFTWARE IS PROVIDED "AS-IS" AND WITHOUT WARRANTY OF ANY KIND,
EXPRESS, IMPLIED OR OTHERWISE, INCLUDING WITHOUT LIMITATION, ANY
WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

IN NO EVENT SHALL WOLFRAM GLOGER BE LIABLE FOR ANY SPECIAL,
INCIDENTAL, INDIRECT OR CONSEQUENTIAL DAMAGES OF ANY KIND, OR ANY
DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS,
WHETHER OR NOT ADVISED OF THE POSSIBILITY OF DAMAGE, AND ON ANY THEORY
OF LIABILITY, ARISING OUT OF OR IN CONNECTION WITH THE USE OR
PERFORMANCE OF THIS SOFTWARE.
```

B.4 ROMIO 1.0.1

```
                                COPYRIGHT

The following is a notice of limited availability of the code and
disclaimer, which must be included in the prologue of the code and in
all source listings of the code.

Copyright (C) 1997 University of Chicago

Permission is hereby granted to use, reproduce, prepare derivative
works, and to redistribute to others.

The University of Chicago makes no representations as to the suitability,
```

operability, accuracy, or correctness of this software for any purpose.
It is provided "as is" without express or implied warranty.

This software was authored by:
Rajeev Thakur: (630) 252-1682; thakur@mcs.anl.gov
Mathematics and Computer Science Division
Argonne National Laboratory, Argonne IL 60439, USA

GOVERNMENT LICENSE

Portions of this material resulted from work developed under a U.S.
Government Contract and are subject to the following license: the
Government is granted for itself and others acting on its behalf a
paid-up, nonexclusive, irrevocable worldwide license in this computer
software to reproduce, prepare derivative works, and perform publicly
and display publicly.

DISCLAIMER

This computer code material was prepared, in part, as an account of
work sponsored by an agency of the United States Government. Neither
the United States Government, nor the University of Chicago, nor any
of their employees, makes any warranty express or implied, or assumes
any legal liability or responsibility for the accuracy, completeness,
or usefulness of any information, apparatus, product, or process
disclosed, or represents that its use would not infringe privately
owned rights.

B.5 Notre Dame C++ bindings for MPI 1.0.3

COPYRIGHT NOTICE:

Copyright 1997, University of Notre Dame.
Authors: Andrew Lumsdaine, Michael P. McNally, Jeremy G. Siek,

Jeffery M. Squyres.

LICENSE AGREEMENT:

In consideration of being allowed to copy and/or use this software, user agrees to be bound by the terms and conditions of this License Agreement as "Licensee." This Agreement gives you, the LICENSEE, certain rights and obligations. By using the software, you indicate that you have read, understood, and will comply with the following terms and conditions.

Permission is hereby granted to use or copy this program for any purpose, provided the text of this NOTICE (to include COPYRIGHT NOTICE, LICENSE AGREEMENT, and DISCLAIMER) is retained with all copies. Permission to modify the code and to distribute modified code is granted, provided the text of this NOTICE is retained, a notice that the code was modified is included with the above COPYRIGHT NOTICE and with the COPYRIGHT NOTICE in any modified files, and that this file ("LICENSE") is distributed with the modified code.

Title to copyright to this software and its derivatives and to any associated documentation shall at all times remain with Licensor and LICENSEE agrees to preserve the same. Nothing in this Agreement shall be construed as conferring rights to use in advertising, publicity or otherwise any trademark or the name of the University of Notre Dame du Lac.

DISCLAIMER:

LICENSOR MAKES NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. By way of example, but not limitation, Licensor MAKES NO REPRESENTATIONS OR WARRANTIES OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF THE LICENSED SOFTWARE COMPONENTS OR DOCUMENTATION WILL NOT INFRINGE ANY PATENTS, COPYRIGHTS, TRADEMARKS OR OTHER RIGHTS.

The Authors and the University of Notre Dame du Lac shall not be held liable for any liability nor for any direct, indirect or consequential

```
damages with respect to any claim by LICENSEE or any third party on
account of or arising from this Agreement or use of this software.

Any disputes arising out of this Agreement or LICENSEE'S use of the
software at any time shall be resolved by the courts of the state of
Indiana.  LICENSEE hereby consents to the jurisdiction of the Indiana
courts and waives the right to challenge the jurisdiction thereof in
any dispute arising out of this Agreement or Licensee's use of the
software.
```

B.6 MPI Message Queue Dumping 1.06

mpi_interface.h

```
/**************************************************************************
 * Copyright (C) 2000 Etnus, LLC
 *
 * Permission is hereby granted to use, reproduce, prepare derivative
 * works, and to redistribute to others.
 *
 *                                DISCLAIMER
 *
 * Neither Etnus, nor any of their employees, makes any warranty
 * express or implied, or assumes any legal liability or
 * responsibility for the accuracy, completeness, or usefulness of any
 * information, apparatus, product, or process disclosed, or
 * represents that its use would not infringe privately owned rights.
 *
 * This code was written by
 * James Cownie: Etnus, LLC. <jcownie@etnus.com>
 **************************************************************************/

/**************************************************************************
 * Copyright (C) 1997-1998 Dolphin Interconnect Solutions Inc.
 *
 * Permission is hereby granted to use, reproduce, prepare derivative
 * works, and to redistribute to others.
```

```
 *
 *                                DISCLAIMER
 *
 * Neither Dolphin Interconnect Solutions, nor any of their employees,
 * makes any warranty express or implied, or assumes any legal
 * liability or responsibility for the accuracy, completeness, or
 * usefulness of any information, apparatus, product, or process
 * disclosed, or represents that its use would not infringe privately
 * owned rights.
 *
 * This code was written by
 * James Cownie: Dolphin Interconnect Solutions. <jcownie@dolphinics.com>
 ************************************************************************/
```

mpich_dll_defs.h

```
/************************************************************************
 * Copyright (C) 1997-1998 Dolphin Interconnect Solutions Inc.
 *
 * Permission is hereby granted to use, reproduce, prepare derivative
 * works, and to redistribute to others.
 *
 *                                DISCLAIMER
 *
 * Neither Dolphin Interconnect Solutions, nor any of their employees,
 * makes any warranty express or implied, or assumes any legal
 * liability or responsibility for the accuracy, completeness, or
 * usefulness of any information, apparatus, product, or process
 * disclosed, or represents that its use would not infringe privately
 * owned rights.
 *
 * This code was written by
 * James Cownie: Dolphin Interconnect Solutions. <jcownie@dolphinics.com>
 ************************************************************************/
```

dll_mpich.h

```
/************************************************************************
 * Copyright (C) 1997-1998 Dolphin Interconnect Solutions Inc.
```

```
 * Copyright (C) 1999-2001 Etnus LLC.
 *
 * Permission is hereby granted to use, reproduce, prepare derivative
 * works, and to redistribute to others.
 *
 *                                DISCLAIMER
 *
 * Neither Dolphin Interconnect Solutions, Etnus LLC, nor any of their
 * employees, makes any warranty express or implied, or assumes any
 * legal liability or responsibility for the accuracy, completeness,
 * or usefulness of any information, apparatus, product, or process
 * disclosed, or represents that its use would not infringe privately
 * owned rights.
 *
 * This code was written by
 * James Cownie: Dolphin Interconnect Solutions. <jcownie@dolphinics.com>
 *               Etnus LLC <jcownie@etnus.com>
 ************************************************************************/
```

B.7 MVICH

```
MVICH License Agreement
August 2004

MVICH Copyright (c) 1998-2000, The Regents of the University of
California, through Lawrence Berkeley National Laboratory (subject to
receipt of any required approvals from U.S. Dept. of Energy).  All
rights reserved.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions are
met:

(1) Redistributions of source code must retain the above copyright
    notice, this list of conditions and the following disclaimer.
```

(2) Redistributions in binary form must reproduce the above copyright
notice, this list of conditions and the following disclaimer in
the documentation and/or other materials provided with the
distribution.

(3) Neither the name of the University of California, Lawrence
Berkeley National Laboratory, U.S. Dept. of Energy nor the names
of its contributors may be used to endorse or promote products derived
from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
"AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

B.8 hwloc 1.11.9

Copyright © 2004-2006 The Trustees of Indiana University and Indiana University Research and Technology Corporation. All rights reserved.
Copyright © 2004-2005 The University of Tennessee and The University of Tennessee Research Foundation. All rights reserved.
Copyright © 2004-2005 High Performance Computing Center Stuttgart, University of Stuttgart. All rights reserved.
Copyright © 2004-2005 The Regents of the University of California. All rights reserved.
Copyright © 2009 CNRS
Copyright © 2009-2016 Inria. All rights reserved.
Copyright © 2009-2015 Université Bordeaux
Copyright © 2009-2015 Cisco Systems, Inc. All rights reserved.
Copyright © 2009-2012 Oracle and/or its affiliates. All rights reserved.

Copyright © 2010 IBM
Copyright © 2010 Jirka Hladky
Copyright © 2012 Aleksej Saushev, The NetBSD Foundation
Copyright © 2012 Blue Brain Project, EPFL. All rights reserved.
Copyright © 2015 Research Organization for Information Science and Technology (RIST). All rights reserved.
Copyright © 2015-2016 Intel, Inc. All rights reserved.
See COPYING in top-level directory.

Redistribution and use in source and binary forms, with or without
modification, are permitted provided that the following conditions
are met:
1. Redistributions of source code must retain the above copyright
   notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright
   notice, this list of conditions and the following disclaimer in the
   documentation and/or other materials provided with the distribution.
3. The name of the author may not be used to endorse or promote products
   derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE AUTHOR ``AS IS'' AND ANY EXPRESS OR
IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED.
IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT,
INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY
THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
(INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF
THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

索　　引

F

FTRACE 機能 105

M

MPI_ABORT 275
MPI_ACCUMULATE 305
MPI_ADD_ERROR_CLASS 276
MPI_ADD_ERROR_CODE 276
MPI_ADD_ERROR_STRING 277
MPI_AINT_ADD 181
MPI_AINT_DIFF 182
MPI_ALLGATHER 183
MPI_ALLGATHERV 184
MPI_ALLOC_MEM 277
MPI_ALLREDUCE 185
MPI_ALLTOALL 186
MPI_ALLTOALLV 187
MPI_ALLTOALLW 188
MPI_ATTR_DELETE 382
MPI_ATTR_GET 383
MPI_ATTR_PUT 383
MPI_BARRIER 189
MPI_BCAST 189
MPI_BSEND 119
MPI_BSEND_INIT 120
MPI_BUFFER_ATTAC 121
MPI_BUFFER_DETACH 121
MPI_CANCEL 122
MPI_CART_COORDS 249
MPI_CART_CREATE 250
MPI_CART_GET 251
MPI_CART_MAP 251
MPI_CART_RANK 252
MPI_CART_SHIFT 252
MPI_CART_SUB 253
MPI_CARTDIM_GET 253
MPI_CLOSE_PORT 296
MPI_COMM_ACCEPT 297
MPI_COMM_CALL_ERRHANDLER 278
MPI_COMM_COMPARE 218
MPI_COMM_CONNECT 298
MPI_COMM_CREATE 219
MPI_COMM_CREATE_ERRHANDLER 278
MPI_COMM_CREATE_GROUP 219
MPI_COMM_CREATE_KEYVAL 220
MPI_COMM_DELETE_ATTR 221
MPI_COMM_DISCONNECT 298
MPI_COMM_DUP 221
MPI_COMM_DUP_WITH_INFO 223
MPI_COMM_FREE_KEYVAL 224
MPI_COMM_GET_ATTR 224
MPI_COMM_GET_ERRHANDLER 279
MPI_COMM_GET_INFO 225
MPI_COMM_GET_NAME 225
MPI_COMM_GET_PARENT 299
MPI_COMM_GROUP 226
MPI_COMM_IDUP 226
MPI_COMM_JOIN 299
MPI_COMM_RANK 227
MPI_COMM_REMOTE_GROUP 227
MPI_COMM_REMOTE_SIZE 228
MPI_COMM_SET_ATTR 228
MPI_COMM_SET_ERRHANDLER 279
MPI_COMM_SET_INFO 229
MPI_COMM_SET_NAME 229
MPI_COMM_SIZE 230
MPI_COMM_SPAWN 300

MPI_COMM_SPAWN_MULTIPLE 301
MPI_COMM_SPLIT 230
MPI_COMM_SPLIT_TYPE 231
MPI_COMM_TEST_INTER 231
MPI_COMPARE_AND_SWAP 306
MPI_DIMS_CREATE 254
MPI_DIST_GRAPH_CREATE 255
MPI_DIST_GRAPH_CREATE_ADJACENT 256
MPI_DIST_GRAPH_NEIGHBORS 257
MPI_DIST_GRAPH_NEIGHBORS_COUNT 258
MPI_ERRHANDLER_FREE 280
MPI_ERROR_CLASS 280
MPI_ERROR_STRING 281
MPI_EXSCAN 190
MPI_FETCH_AND_OP 307
MPI_FILE_CALL_ERRHANDLER 281
MPI_FILE_CLOSE 337
MPI_FILE_CREATE_ERRHANDLER 282
MPI_FILE_DELETE 337
MPI_FILE_GET_AMODE 338
MPI_FILE_GET_ATOMICITY 338
MPI_FILE_GET_BYTE_OFFSET 339
MPI_FILE_GET_ERRHANDLER 282
MPI_FILE_GET_GROUP 339
MPI_FILE_GET_INFO 340
MPI_FILE_GET_POSITION 340
MPI_FILE_GET_POSITION_SHARED 341
MPI_FILE_GET_SIZE 341
MPI_FILE_GET_TYPE_EXTENT 342
MPI_FILE_GET_VIEW 342
MPI_FILE_IREAD 343
MPI_FILE_IREAD_AT 344
MPI_FILE_IREAD_SHARED 345
MPI_FILE_IWRITE 346
MPI_FILE_IWRITE_AT 347
MPI_FILE_IWRITE_SHARED 348
MPI_FILE_OPEN 349
MPI_FILE_PREALLOCATE 349
MPI_FILE_READ 350
MPI_FILE_READ_ALL 351, 373
MPI_FILE_READ_ALL_BEGIN 352
MPI_FILE_READ_ALL_END 352
MPI_FILE_READ_AT 353
MPI_FILE_READ_AT_ALL 354, 371
MPI_FILE_READ_AT_ALL_BEGIN 355
MPI_FILE_READ_AT_ALL_END 355
MPI_FILE_READ_ORDERED 356
MPI_FILE_READ_ORDERED_BEGIN 357
MPI_FILE_READ_ORDERED_END 357
MPI_FILE_READ_SHARED 358
MPI_FILE_SEEK 358
MPI_FILE_SEEK_SHARED 359
MPI_FILE_SET_ATOMICITY 359
MPI_FILE_SET_ERRHANDLER 283
MPI_FILE_SET_INFO 360
MPI_FILE_SET_SIZE 360
MPI_FILE_SET_VIEW 361
MPI_FILE_SYNC 361
MPI_FILE_WRITE 362
MPI_FILE_WRITE_ALL 363, 374
MPI_FILE_WRITE_ALL_BEGIN 363
MPI_FILE_WRITE_ALL_END 364
MPI_FILE_WRITE_AT 365
MPI_FILE_WRITE_AT_ALL 366, 372
MPI_FILE_WRITE_AT_ALL_BEGIN 367
MPI_FILE_WRITE_AT_ALL_END 367
MPI_FILE_WRITE_ORDERED 368
MPI_FILE_WRITE_ORDERED_BEGIN 368
MPI_FILE_WRITE_ORDERED_END 369
MPI_FILE_WRITE_SHARED 369
MPI_FINALIZE 283
MPI_FINALIZED 284
MPI_FREE_MEM 284
MPI_GATHER 191
MPI_GATHERV 192
MPI_GET 308
MPI_GET_ACCUMULATE 309
MPI_GET_ADDRESS 156
MPI_GET_COUNT 123
MPI_GET_ELEMENTS 157

MPI_GET_ELEMENTS_X 158
MPI_GET_LIBRARY_VERSION 285
MPI_GET_PROCESSOR_NAME 285
MPI_GET_VERSION 286
MPI_GRAPH_CREATE 259
MPI_GRAPH_GET 260
MPI_GRAPH_MAP 260
MPI_GRAPH_NEIGHBORS 261
MPI_GRAPH_NEIGHBORS_COUNT 261
MPI_GRAPHDIMS_GET 262
MPI_GREQUEST_COMPLETE 330
MPI_GREQUEST_START 331
MPI_GROUP_COMPARE 232
MPI_GROUP_DIFFERENCE 232
MPI_GROUP_EXCL 233
MPI_GROUP_FREE 233
MPI_GROUP_INTERSECTION 234
MPI_GROUP_RANGE_EXCL 235
MPI_GROUP_RANGE_INCL 235
MPI_GROUP_RANK 236
MPI_GROUP_SIZE 236
MPI_GROUP_TRANSLATE_RANKS 237
MPI_GROUP_UNION 237
MPI_IALLGATHER 193
MPI_IALLGATHERV 194
MPI_IALLREDUCE 195
MPI_IALLTOALL 196
MPI_IALLTOALLV 197
MPI_IALLTOALLW 198
MPI_IBARRIER 199
MPI_IBCAST 199
MPI_IBSEND 124
MPI_IEXSCAN 200
MPI_IGATHER 201
MPI_IGATHERV 202
MPI_IMPROBE 125
MPI_IMRECV 126
MPI_INEIGHBOR_ALLGATHER 263
MPI_INEIGHBOR_ALLGATHERV 264
MPI_INEIGHBOR_ALLTOALL 265
MPI_INEIGHBOR_ALLTOALLV 266
MPI_INEIGHBOR_ALLTOALLW 267
MPI_INFO_CREATE 290
MPI_INFO_DELETE 291
MPI_INFO_DUP 291
MPI_INFO_FREE 292
MPI_INFO_GET 292
MPI_INFO_GET_NKEYS 293
MPI_INFO_GET_NTHKEY 293
MPI_INFO_GET_VALUELEN 294
MPI_INFO_SET 294
MPI_INIT 286
MPI_INIT_THREAD 331
MPI_INITIALIZED 287
MPI_INTERCOMM_CREATE 238
MPI_INTERCOMM_MERGE 239
MPI_IPROBE 127
MPI_IRECV 128
MPI_IREDUCE 203
MPI_IREDUCE_SCATTER 204
MPI_IREDUCE_SCATTER_BLOCK 205
MPI_IRSEND 129
MPI_IS_THREAD_MAIN 332
MPI_ISCAN 206
MPI_ISCATTER 207
MPI_ISCATTERV 208
MPI_ISEND 130
MPI_ISSEND 131
MPI_KEYVAL_CREATE 384
MPI_KEYVAL_FREE 385
MPI_LOOKUP_NAME 302
MPI_MPROBE 132
MPI_MRECV 133
MPI_NEIGHBOR_ALLGATHER 268
MPI_NEIGHBOR_ALLGATHERV 269
MPI_NEIGHBOR_ALLTOALL 270
MPI_NEIGHBOR_ALLTOALLV 271
MPI_NEIGHBOR_ALLTOALLW 272
MPI_OP_COMMUTATIVE 209
MPI_OP_CREATE 210

MPI_OP_FREE 211
MPI_OPEN_PORT 302
MPI_PACK 159
MPI_PACK_EXTERNAL 160
MPI_PACK_EXTERNAL_SIZE 161
MPI_PACK_SIZE 162
MPI_PCONTROL 380
MPI_PROBE 134
MPI_PUBLISH_NAME 303
MPI_PUT 310
MPI_QUERY_THREAD 332
MPI_RACCUMULATE 311
MPI_RECV 135
MPI_RECV_INIT 136
MPI_REDUCE 211
MPI_REDUCE_LOCAL 212
MPI_REDUCE_SCATTER 213
MPI_REDUCE_SCATTER_BLOCK 214
MPI_REGISTER_DATAREP 370
MPI_REQUEST_FREE 137
MPI_REQUEST_GET_STATUS 137
MPI_RGET 312
MPI_RGET_ACCUMULATE 313
MPI_RPUT 314
MPI_RSEND 138
MPI_RSEND_INIT 139
MPI_SCAN 215
MPI_SCATTER 216
MPI_SCATTERV 217
MPI_SEND 140
MPI_SEND_INIT 141
MPI_SENDRECV 142
MPI_SENDRECV_REPLACE 143
MPI_SIZEOF 376
MPI_SSEND 144
MPI_SSEND_INIT 145
MPI_START 146
MPI_STARTALL 147
MPI_STATUS_SET_CANCELLED 333
MPI_STATUS_SET_ELEMENTS 333
MPI_STATUS_SET_ELEMENTS_X 334
MPI_TEST 148
MPI_TEST_CANCELLED 148
MPI_TESTALL 149
MPI_TESTANY 150
MPI_TESTSOME 151
MPI_TOPO_TEST 273
MPI_TYPE_COMMIT 162
MPI_TYPE_CONTIGUOUS 163
MPI_TYPE_CREATE_DARRAY 164
MPI_TYPE_CREATE_F90_COMPLEX 377
MPI_TYPE_CREATE_F90_INTEGER 377
MPI_TYPE_CREATE_F90_REAL 378
MPI_TYPE_CREATE_HINDEXED 165
MPI_TYPE_CREATE_HINDEXED_BLOCK 166
MPI_TYPE_CREATE_HVECTOR 167
MPI_TYPE_CREATE_INDEXED_BLOCK 168
MPI_TYPE_CREATE_KEYVAL 240
MPI_TYPE_CREATE_RESIZED 169
MPI_TYPE_CREATE_STRUCT 170
MPI_TYPE_CREATE_SUBARRAY 171
MPI_TYPE_DELETE_ATTR 241
MPI_TYPE_DUP 172
MPI_TYPE_FREE 172
MPI_TYPE_FREE_KEYVAL 241
MPI_TYPE_GET_ATTR 242
MPI_TYPE_GET_CONTENTS 173
MPI_TYPE_GET_ENVELOPE 174
MPI_TYPE_GET_EXTENT 175
MPI_TYPE_GET_EXTENT_X 175
MPI_TYPE_GET_NAME 242
MPI_TYPE_GET_TRUE_EXTENT 176
MPI_TYPE_GET_TRUE_EXTENT_X 176
MPI_TYPE_INDEXED 177
MPI_TYPE_MATCH_SIZE 378
MPI_TYPE_SET_ATTR 243
MPI_TYPE_SET_NAME 243
MPI_TYPE_SIZE 177
MPI_TYPE_SIZE_X 178
MPI_TYPE_VECTOR 178

MPI_UNPACK ... 179
MPI_UNPACK_EXTERNAL ... 180
MPI_UNPUBLISH_NAME ... 303
MPI_WAIT ... 152
MPI_WAITALL ... 152
MPI_WAITANY ... 153
MPI_WAITSOME ... 154
MPI_WIN_ALLOCATE ... 315
MPI_WIN_ALLOCATE_SHARED ... 316
MPI_WIN_ATTACH ... 317
MPI_WIN_CALL_ERRHANDLER ... 287
MPI_WIN_COMPLETE ... 317
MPI_WIN_CREATE ... 318
MPI_WIN_CREATE_DYNAMIC ... 318
MPI_WIN_CREATE_ERRHANDLER ... 288
MPI_WIN_CREATE_KEYVAL ... 244
MPI_WIN_DELETE_ATTR ... 245
MPI_WIN_DETACH ... 319
MPI_WIN_FENCE ... 319
MPI_WIN_FLUSH ... 320
MPI_WIN_FLUSH_ALL ... 320
MPI_WIN_FLUSH_LOCAL ... 321
MPI_WIN_FLUSH_LOCAL_ALL ... 321
MPI_WIN_FREE ... 322
MPI_WIN_FREE_KEYVAL ... 245
MPI_WIN_GET_ERRHANDLER ... 288
MPI_WIN_GET_GROUP ... 322
MPI_WIN_GET_INFO ... 323
MPI_WIN_GET_NAME ... 246
MPI_WIN_LOCK ... 323
MPI_WIN_LOCK_ALL ... 324
MPI_WIN_POST ... 324
MPI_WIN_SET_ATTR ... 247
MPI_WIN_SET_ERRHANDLER ... 289
MPI_WIN_SET_INFO ... 325
MPI_WIN_SET_NAME ... 247
MPI_WIN_SHARED_QUERY ... 325
MPI_WIN_START ... 326
MPI_WIN_SYNC ... 326
MPI_WIN_TEST ... 327
MPI_WIN_UNLOCK ... 327
MPI_WIN_UNLOCK_ALL ... 328
MPI_WIN_WAIT ... 328
MPI_WTICK ... 274
MPI_WTIME ... 275
MPI インクルードファイル ... 12
MPI コンパイルコマンド ... 12, 53
MPI デーモン ... 12
MPI トレース機能 ... 108
MPI プロセス ... 17
MPI モジュール ... 12
MPI ライブラリ ... 12
MPI 実行コマンド ... 12, 58
MPI 実行指定 ... 59, 61
MPI 集団手続デバッグ支援機能 ... 109
MPI 通信情報 ... 99

N

NEC MPI コンパイラオプション ... 54
NQSV ... 87

い

異常終了 ... 111
1 対 1 通信 ... 19
インターコミュニケーター ... 17
イントラコミュニケーター ... 17
引用仕様 ... 117

う

ウィンドウ ... 35

え

エラークラス ... 392
エラーコード ... 392
エラー処理 ... 16
エラーハンドラー ... 47

か

片側通信 ... 35
環境変数 ... 68, 69

き

キー 48

く

グループ 17
グローバルオプション 58, 60

し

シグナル 112
実行性能情報 91
集計演算 395
集計演算子 395
集団操作 29

す

スレッド実行環境 46

せ

正常終了 111

そ

属性 47
属性キー 47

た

タグ 19

つ

通信識別子 26
通信メッセージ 19
通信モード 19

て

データ型 388
デッドロック 24

と

同期制御 29
同期モード 20
動的プロセス管理 18
動的プロセス結合 18
動的プロセス生成 18, 62
同報通信 31

は

バッファモード 21
バリア 29
ハンドル 16

ひ

非ブロッキング 26
ビュー 40
標準モード 24

ふ

不透明オブジェクト 16
プログラム終了状態 111
ブロッキング 26

へ

並列 I/O 40

ら

ランク 17

り

リモートグループ 17
利用者定義データ型 388
利用者バッファ 21

る

ルートプロセス 31

れ

レディモード 23

ろ

ローカルオプション 58, 60
ローカルグループ 17

SX-Aurora TSUBASA システムソフトウェア
NEC MPI ユーザーズガイド

2018 年 2 月　初　　版
2022 年 3 月　第２１版

日 本 電 気 株 式 会 社

東 京 都 港 区 芝 五 丁 目 7 番 1 号
ＴＥＬ (03) 3454-1111 (大代表)

© NEC Corporation 2018-2022
日本電気株式会社の許可なく複製・改変などを行うことはできません。
本書の内容に関しては将来予告なしに変更することがあります。