FUJITSU

Fujitsu Software
Technical Computing Suite V4.0L20

Development Studio
MPI使用手引書

J2UL-2565-01Z0(15)
2025年9月

まえがき

本書の目的

本書は、富士通製CPU A64FXを搭載したシステム(以降、本システムと呼びます)向けのMPIライブラリの使用方法を説明しています。

MPI(Message Passing Interface)は、MPI Forumで規定されているMPIライブラリのインターフェースです。このMPIライブラリは、MPI規格で規定するインターフェースを実装したものです。以降、このMPIライブラリを本処理系と呼びます。

本処理系は、MPI Forumで規定されているMPI-3.1規格およびMPI-4.0規格の一部に準拠しています。

本書の読者

本書は、本処理系を使用して、Fortranプログラム、Cプログラム、C++プログラム、およびJavaプログラムを開発する人を対象に記述しています。本書を読むにあたっては、以下の基本的な知識が必要です。

- MPI
- Fortranプログラム、Cプログラム、C++プログラム、およびJavaプログラム
- Linuxのコマンド、ファイル操作、およびシェルプログラミング
- ジョブ運用ソフトウェア

本書の構成

本書は、以下の構成になっています。

第1章 概要

本処理系の概要を説明します。

第2章 環境と事前設定

事前に必要な環境設定について説明します。

第3章 MPIプログラムの翻訳/結合

MPIプログラムの翻訳/結合の方法について説明します。

第4章 MPIプログラムの実行

MPIプログラムの実行方法について説明します。

第5章 拡張インターフェース

拡張インターフェースについて説明します。

第6章 補足事項

本処理系の補足事項について説明します。

第7章 エラーメッセージ

本処理系で検出するエラーメッセージについて説明します。

第8章 ブロッキング集団通信の高速化

ブロッキング集団通信の高度なチューニング方法について説明します。

付録A エラークラス一覧

本処理系で検出するエラークラスについて説明します。

用語集

用語について説明します。

本書の位置付け

本書は、以下のマニュアルと関係があります。必要に応じて参照してください。

- Fortran文法書

- Fortran使用手引書
- Fortran使用手引書 別冊 COARRAY
- Fortran翻訳時メッセージ
- C言語使用手引書
- C++言語使用手引書
- C/C++最適化メッセージ説明書
- Fortran/C/C++実行時メッセージ
- 統合開発環境使用手引書
- プロファイラ使用手引書
- 並列実行デバッガ使用手引書
- 数学ライブラリの利用手引
- 富士通 SSL II 使用手引書
- FUJITSU SSL II 拡張機能使用手引書
- FUJITSU SSL II 拡張機能使用手引書II
- FUJITSU SSL II スレッド並列機能 使用手引書
- FUJITSU C-SSL II 使用手引書
- FUJITSU C-SSL II スレッド並列機能 使用手引書
- FUJITSU SSL II/MPI 使用手引書
- BLAS LAPACK ScaLAPACK使用手引書
- 高速4倍精度基本演算ライブラリ 使用手引書
- uTofu使用手引書
- MPI User's Guide Additional Volume Java Interface

上記以外に、以下の関連ソフトウェアのマニュアルも必要に応じて参照してください。

- ジョブ運用ソフトウェア
- FEFS/LLIO

また、MPIの仕様の詳細を知りたい場合は、以下の規格書を参照してください。

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

MPIに関する情報は、https://www.mpi-forum.org/ から入手できます。

ただし、上記から入手した情報と本処理系の仕様が異なる場合がありますので、ご注意ください。

本書の表記について

単位の表現

本書では、単位を表現する際の接頭語を以下のとおり使い分けています。

接頭語	値	接頭語	値
k (kilo)	10^3	Ki (kibi)	2^{10}

接頭語	値	接頭語	値
M (mega)	10^6	Mi (mebi)	2^{20}
G (giga)	10^9	Gi (gibi)	2^{30}

構文表記記号

構文表記記号とは、構文を記述するうえで特別な意味で定められた記号であり、以下のものがあります。

記号名	記号	説明
選択記号	{ }	この記号で囲まれた項目の中から、どれか1つを選択することを表します。
	|	この記号を区切りとして、複数の項目を列挙することを表します。
省略可能記号	[ ]	この記号で囲まれた項目を省略してよいことを表します。また、この記号は選択記号“{ }”の意味を含みます。
反復記号	...	この記号の直前の項目を繰り返して指定できることを表します。

ルーチンの表現

本処理系は、Fortran、C言語、C++、およびJavaに対する言語bindingを提供します。

MPI規格や本書の別冊(MPI User's Guide Additional Volume Java Interface)で定義されているように、C言語の関数、C++のメンバ関数、Fortranのサブルーチン/関数、およびJavaのメソッドは、呼び出し形式は異なりますが、ほぼ同等の仕様です。このため、言語に依存しない共通の呼び方として、本書ではこれらを“ルーチン”と表しています。また、各ルーチン名はMPI規格と同様に、C言語の関数名(Fortran bindingにしか存在しないものはFortranのサブルーチン名/関数名)をすべて英大文字で表記しています。

輸出管理規制について

本ドキュメントを輸出または第三者へ提供する場合は、お客様が居住する国および米国輸出管理関連法規等の規制をご確認のうえ、必要な手続きをおとりください。

商標

- Javaは、Oracle Corporation 及びその子会社、関連会社の米国及びその他の国における登録商標です。
- Linux(R)は米国及びその他の国におけるLinus Torvaldsの登録商標です。
- そのほか、本マニュアルに記載されている会社名および製品名は、それぞれ各社の商標または登録商標です。
- 本マニュアルに掲載されているシステム名、製品名などには、必ずしも商標表示(TM、(R))を付記しておりません。

出版年月および版数

版数	マニュアルコード
2025年 9月 第1.15版	J2UL-2565-01Z0(15)
2025年 3月 第1.14版	J2UL-2565-01Z0(14)
2024年 9月 第1.13版	J2UL-2565-01Z0(13)
2024年 3月 第1.12版	J2UL-2565-01Z0(12)
2023年 9月 第1.11版	J2UL-2565-01Z0(11)
2023年 3月 第1.10版	J2UL-2565-01Z0(10)
2022年 9月 第1.9版	J2UL-2565-01Z0(09)
2022年 3月 第1.8版	J2UL-2565-01Z0(08)
2021年11月 第1.7版	J2UL-2565-01Z0(07)
2021年 8月 第1.6版	J2UL-2565-01Z0(06)
2021年 7月 第1.5版	J2UL-2565-01Z0(05)
2021年 3月 第1.4版	J2UL-2565-01Z0(04)

版数	マニュアルコード
2020年11月 第1.3版	J2UL-2565-01Z0(03)
2020年 9月 第1.2版	J2UL-2565-01Z0(02)
2020年 6月 第1.1版	J2UL-2565-01Z0(01)
2020年 2月 初版	J2UL-2565-01Z0(00)

変更履歴

変更内容	変更箇所	版数
UFSに関する記述を削除しました。	6.4.12	第1.15版
以下の集団通信アルゴリズムを追加しました。 ・ knomial ・ rabenseifner	8章	
以下のMCAパラメーターの最大値を変更しました。 ・ common_tofu_large_recv_buf_size ・ common_tofu_medium_recv_buf_size	4.2	第1.14版
説明文を修正しました。	4.2	第1.13版
	6.5	
	6.10.1	
以下のMCAパラメーターを追加しました。 ・ common_tofu_conv_dim ・ common_tofu_conv_dim_log	4.2	第1.12版
メモリ使用量の説明を見直しました。	6.11	
“ジョブ次元変換機能”を追加しました。	6.15	
説明文を修正しました。	4章	第1.11版
説明文を追加しました。	6.4.12	第1.10版
図のデザインを改善しました。	―	
説明文を追加しました。	4章	第1.9版
ファイルシステムの説明を修正しました。	6.4.12	
isend/irecvの注意を追加しました。	6.10.2	
説明文を見直しました。	―	
以下のMCAパラメーターの説明を修正しました。 ・ mpi_print_stats	4.2	第1.8版
“Stride RDMA通信に関する注意事項”を追加しました。	6.6.1	
説明文を追加しました。	6.12.1 6.12.2 6.12.3	
説明文を見直しました。	―	

変更内容	変更箇所	版数
Fortran 2018言語仕様の次元引継ぎに関する注意事項を削除しました。	3.2	第1.7版
環境変数UTOFU_SWAP_PROTECTに関する説明を追加しました。	4.3	
説明文を追加しました。	4.4	
ファイル入出力データサイズの最大値に関する記事を修正しました。	6.4.12	第1.6版
“翻訳/結合に関する注意事項”に記事を追加しました。	3.2	第1.5版
以下のMCAパラメーターを追加しました。 ・ coll	4.2 6.12.1 6.12.2 6.12.3 6.12.4	
以下の環境変数を追加しました。 ・ UTOFU_SWAP_PROTECT	4.3	
以下のInfoキーにおいてvalueの省略値を変更しました。 ・ romio_ds_write ・ romio_llio_ds_in_coll	6.4.12	
説明文を修正しました。	6.17.1	
説明文を修正しました。	第4章	第1.4版
以下のMCAパラメーターに説明を追加しました。 ・ common_tofu_use_memory_pool ・ mpi_java_eager	4.2	
割り当てイメージを修正しました。	4.7.2	
説明文を追加しました。	6.9 6.11.1	第1.3版
以下のMCAパラメーターに説明を追加しました。 ・ common_tofu_large_recv_buf_size ・ coll_tuned_prealloc_size	4.2	
以下のMCAパラメーターを追加しました。 ・ common_tofu_use_memory_pool	4.2	
説明文を修正しました。	4.2 6.13 6.15	
注意を追加しました。	6.13	
MPI統計情報の出力例を修正しました。	6.15	
説明文を追加しました。	第4章 4.1	第1.2版
大規模MPIジョブを実行する場合の注意を追加しました。	4.1	
説明文を修正しました。	4.1	
エラーメッセージを追加しました。	7.3	
説明文を修正しました。	7.3	
例を修正しました。	8.3.2.2	
選択するアルゴリズムの適用条件を修正しました。	8.3.4.4	

変更内容	変更箇所	版数
説明文を追加しました。	4.2 6.8 6.18.5 8.3.1.4.4 8.3.1.4.6 8.3.1.5.5 8.3.2.2	第1.1版
LLIOの場合に使用可能なInfoオブジェクトのkeyからstriping関連を削除しました。	6.4.12	
メモリ使用量の見積もり式を変更しました。	6.11.1 6.11.2 6.11.3.2	
説明文を修正しました。	8.3.1.1	
アルゴリズムの適用に必要な条件を修正しました。	8.3.4.1 8.3.4.2 8.3.4.4	
表を追加しました。	8.3.4.12 8.4.1.12	

目　次

第1章 概要

本処理系は、オープンソースのMPIライブラリ(Open MPI)をベースに作成しています。この章では、本処理系の概要および利用の概観を説明します。

1.1 本処理系の特徴

MPI(Message Passing Interface)は、MPI Forumで定められた規格であり、分散メモリ型の並列計算機システムにおいて、FortranまたはC言語によるMPI並列プログラミングを可能とするライブラリインターフェースを規定しています。C++プログラムからはC言語のインターフェースを通して使用することが想定されています。本処理系では、JavaによるMPI並列プログラミングも可能とするため、MPIのJava bindingを独自に規定し使用できるようにしています。ただし、あらかじめログインノードおよび計算ノードにOpenJDKをインストールしておく必要があります。サポートしないルーチンは、“表6.3 本処理系のJava bindingで提供しないルーチン”をお読みください。

本システムでは、Tofuと呼ばれる6次元メッシュ／トーラスで構成されたインターコネクトが採用されています。Tofuインターコネクトでは、物理的な6次元メッシュ／トーラスから仮想的なトーラス形状を構成することができ、ユーザは1次元から3次元のトーラス形状のネットワーク構成を指定してプログラムを実行できます。本処理系は、このTofuインターコネクトに対応し、本処理系を使用するアプリケーションプログラムの性能を最大限に引き出せるようになっています。また、拡張インターフェースを用いることで、6次元メッシュ／トーラスを生かしたプログラムをユーザが記述できます。Tofuインターコネクトの詳細については“6.1 Tofuインターコネクト”をお読みください。

1.2 本処理系利用の概観

本処理系は、Fortran、C言語、C++、またはJavaで記述されるアプリケーションプログラムから利用できます。MPIライブラリを使用するアプリケーションプログラムを、本書では、MPIプログラムと呼びます。

本処理系には、MPIプログラムの翻訳/結合コマンドおよび実行コマンドが用意されています。

ここでは、本処理系を使用するにあたって、本システム向けのMPIプログラムの翻訳から実行までの手順について、簡単な流れを説明します。

1.2.1 MPIプログラムの翻訳から実行までの流れ

本システムでMPIプログラムを実行するには、利用者はログインノード上で必要な操作を行います。

MPIプログラムの翻訳/結合

本処理系では、Fortran、C言語、C++、またはJavaで記述されたMPIプログラムを翻訳/結合して本システム向けの実行可能ファイル形式に変換する翻訳/結合コマンドを用意しています。

Fortran、C言語、およびC++の翻訳/結合コマンドは、対応する富士通コンパイラのコマンドを内部的に呼び出します。Javaの翻訳コマンドは、対応するJavaコンパイラのコマンドを内部的に呼び出します。
翻訳/結合コマンドには、ログインノード上で使用するコマンド(クロスコンパイラ)と計算ノード上で使用するコマンド(ネイティブコンパイラ)があります。

翻訳/結合コマンドを下表に示します。MPIプログラムを記述するプログラム言語の種類に応じて、これらの翻訳/結合コマンドをお使いください。

表1.1 翻訳/結合コマンド

種別	コマンド名	対応する富士通/Javaコンパイラのコマンド	MPIプログラムを記述するプログラム言語
クロスコンパイラ	mpifrtpx	frtpx	Fortran
	mpifccpx	fccpx	C言語
	mpiFCCpx	FCCpx	C++
	mpijavac	javac	Java
ネイティブコンパイラ	mpifrt	frt	Fortran
	mpifcc	fcc	C言語
	mpiFCC	FCC	C++

種別	コマンド名	対応する富士通/Javaコンパイラの コマンド	MPIプログラムを記述する プログラム言語
	mpijavac	javac	Java

クロスコンパイラの翻訳/結合コマンドは、ログインノード上で使用します。クロスコンパイラの翻訳/結合コマンドを使用することで、MPIプログラムを本システムで実行可能な形式に変換できます。MPIプログラムの翻訳/結合コマンドの詳細な使用方法については、“第3章 MPIプログラムの翻訳/結合”をお読みください。

また、富士通コンパイラの詳細については、各コンパイラのマニュアルをお読みください。

ネイティブコンパイラの翻訳/結合コマンドは、計算ノード上で使用します。計算ノード上でのコマンド実行は、ジョブ運用ソフトウェアにジョブ投入を依頼することで行います。ジョブ運用ソフトウェアの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

MPIプログラム用のクロスコンパイラおよびネイティブコンパイラは、上述のコマンド投入方法を除き、機能上の差異はありません。ネイティブコンパイラの翻訳/結合コマンドの詳細な使用方法についても、“第3章 MPIプログラムの翻訳/結合”をお読みください。

MPIプログラムの実行

翻訳/結合コマンドを使って作成した実行可能ファイル形式のMPIプログラムは、mpiexecコマンドの実行によって行います。mpiexecコマンドの実行は、本システム内の計算ノード上で行う必要がありますが、計算ノード上での実際のmpiexecコマンドの実行は、利用者が直接行うのではなく、MPIプログラムを実行するジョブの投入をジョブ運用ソフトウェアを使って行います。ジョブ投入方法については、ジョブ運用ソフトウェアのマニュアルをお読みください。

また、MPIプログラムにおける並列プロセス間の通信は、並列プロセスが異なる複数の計算ノードに配置される場合、Tofuインターコネクトによる通信となり、並列プロセスが同一計算ノード内に配置される場合、共有メモリによる通信となります。

第2章 環境と事前設定

この章では、本処理系のご利用にあたって必要な環境設定について記述しています。

“*製品インストールパス*”は、システム管理者にお問い合わせください。

2.1 MPIプログラムの翻訳/結合環境

クロスコンパイラによるMPIプログラムの翻訳/結合を可能とするためには、ログインノードにおいて、以下の設定が必要です。

・ 利用者の環境変数PATHに以下を追加

```
/製品インストールパス/bin
```

Javaプログラムの翻訳において、/usr/bin以外に配置されているjavacコマンドを使用する場合は、利用者の環境変数PATHにjavacコマンドのパスの設定が必要です。

ネイティブコンパイラによるMPIプログラムの翻訳/結合を可能とするためには、翻訳/結合コマンド実行のジョブ投入を行うとき、ジョブスクリプト内に以下の設定が必要です。

・ 利用者の環境変数PATHに以下を追加

```
/製品インストールパス/bin
```

Javaプログラムの翻訳において、/usr/bin以外に配置されているjavacコマンドを使用する場合は、翻訳コマンド実行のジョブ投入を行うとき、ジョブスクリプト内で利用者の環境変数PATHにjavacコマンドのパスの設定が必要です。

ジョブ投入およびジョブスクリプトの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

MPIライブラリの資源を下表に示します。

表2.1 MPIライブラリ資源一覧

名前			用途(()内はインストール場所を示します)
C言語/C++	Fortran	Java	
mpi.h mpi-ext.h	mpif.h mpif-ext.h mpi.mod mpi_ext.mod mpi_f08.mod mpi_f08_ext.mod	－	翻訳時:MPIライブラリのヘッダーファイル(Fortranの場合はモジュール情報ファイルも含む) (*/製品インストールパス*/include/mpi/fujitsu)
－	－	mpi.jar	翻訳時および実行時:MPIライブラリのjarファイル (*/製品インストールパス*/include/mpi/fujitsu)
mpifccpx mpiFCCpx mpifcc mpiFCC	mpifrtpx mpifrt	－	翻訳/結合時:Fortran、C言語、およびC++の翻訳/結合コマンド (*/製品インストールパス*/bin)
－	－	mpijavac	翻訳:Javaの翻訳コマンド (*/製品インストールパス*/bin)

2.2 MPIプログラムの実行環境

MPIプログラム実行のジョブ投入を行うとき、ジョブスクリプト内に以下の設定が必要です。ジョブ投入およびジョブスクリプトの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

・ 利用者の環境変数PATHに以下を追加

```
/製品インストールパス/bin
```

Javaプログラムの実行において、/usr/bin以外に配置されているjavaコマンドを使用する場合は、利用者の環境変数PATHにjavaコマンドのパスを追加します。

・ 利用者の環境変数LD_LIBRARY_PATHに以下を追加

```
/製品インストールパス/lib64
```

2.3 オンラインマニュアル

関連するオンラインマニュアルを利用するには、ログインノードにおいて、以下の設定が必要です。

・ 利用者の環境変数MANPATHに以下を追加

```
/製品インストールパス/man
```

第3章 MPIプログラムの翻訳/結合

この章では、MPIプログラムの翻訳/結合の方法について記述しています。

3.1 翻訳/結合コマンドの概要

“1.2 本処理系利用の概観”で述べたとおり、MPIプログラムは、MPIライブラリの呼出しを含むFortran、C言語、C++、またはJavaのプログラムです。

通常、Fortran、C言語、またはC++のプログラムの翻訳/結合には、それぞれ、frtpxコマンド、fccpxコマンド、またはFCCpxコマンド(ネイティブコンパイラの場合は、frtコマンド、fccコマンド、またはFCCコマンド)の富士通コンパイラを利用しますが、MPIプログラムの翻訳/結合には、MPIプログラム用の翻訳/結合コマンドとして、mpifrtpxコマンド、mpifccpxコマンド、またはmpiFCCpxコマンド(ネイティブコンパイラの場合は、mpifrtコマンド、mpifccコマンド、またはmpiFCCコマンド)を利用します。
また、通常Javaプログラムの翻訳にはjavacコマンドを利用しますが、MPIプログラムの翻訳には、MPIプログラム用の翻訳コマンドとして、mpijavacコマンドを利用します。

mpifrtpxコマンド、mpifccpxコマンド、mpiFCCpxコマンド、mpifrtコマンド、mpifccコマンド、およびmpiFCCコマンドは、それぞれ、frtpxコマンド、fccpxコマンド、FCCpxコマンド、frtコマンド、fccコマンド、およびFCCコマンドのラッパーコマンドになっており、内部的に対応する富士通コンパイラのコマンドを呼び出します。このため、MPIプログラム用の翻訳/結合コマンドには、対応する富士通コンパイラのオプションをそのまま指定できます。
また、mpijavacコマンドは、javacコマンドのラッパーコマンドになっており、内部的にJDKのjavacコマンドを呼び出します。このため、mpijavacコマンドにはjavacコマンドのオプションをそのまま指定できます。

なお、MPIプログラムの翻訳時に、MPIプログラムがMPIルーチンに渡している引数の型が整合しているかどうか、チェックできます。
Fortranプログラムは、そのプログラムがmpiモジュールまたはmpi_f08モジュールを使用している場合だけ、そのモジュールの内容を元にコンパイラがチェックし、エラーおよび警告メッセージを出力します。
C言語またはC++のプログラムは、mpi.hの内容を元にコンパイラがチェックし、エラーおよび警告メッセージを出力します。

富士通コンパイラの詳細については、各コンパイラのマニュアルをお読みください。

3.2 翻訳/結合コマンドの形式

表3.1 Fortran、C言語、C++の翻訳/結合コマンドの形式

コマンド名		オプション
クロスコンパイラ	mpifrtpx	[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [*compiler_arguments*] file ...
ネイティブコンパイラ	mpifrt	
クロスコンパイラ	mpifccpx	[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [*compiler_arguments*] file ...
ネイティブコンパイラ	mpifcc	
クロスコンパイラ	mpiFCCpx	[-showme|-showme:compile|-showme:link|-showme:version] [-SCALAPACK] [-SSL2MPI] [*compiler_arguments*] file ...
ネイティブコンパイラ	mpiFCC	

表3.2 Javaの翻訳コマンドの形式

コマンド名	オプション
mpijavac	[--showme|--verbose|--help|-help|-h] [*javac_arguments*] file ...

本処理系において、MPIプログラムは、Fortran、C言語、またはC++の混在が可能です。混在させる場合の注意事項および詳細は、各コンパイラのマニュアルをお読みください。

翻訳/結合コマンドの各オプションの意味を下表に示します。

表3.3 Fortran、C言語、C++の翻訳/結合コマンドの各オプション

オプション	意味
-showme	MPIプログラムの翻訳/結合コマンドが富士通コンパイラのコマンドを呼び出すときの呼出し行を表示します。実際には翻訳/結合処理は行いません。

オプション	意味
-showme:compile	富士通コンパイラのコマンドに渡すオプション一覧を表示します。実際には翻訳/結合処理は行いません。
-showme:help	ヘルプメッセージを表示します。実際には翻訳/結合処理は行いません。
-showme:link	リンカに渡すオプション一覧を表示します。実際には翻訳/結合処理は行いません。
-showme:version	バージョン情報を表示します。実際には翻訳/結合処理は行いません。
-SCALAPACK	ScaLAPACKライブラリを結合します。また、本オプションと合わせて、富士通コンパイラオプションの-SSL2または-SSL2BLAMPを指定してください。
-SSL2MPI	SSL II/MPIライブラリを結合します。また、本オプションと合わせて、富士通コンパイラオプションの-SSL2または-SSL2BLAMPを指定してください。
compiler_arguments	富士通コンパイラに渡すオプションを指定します。 具体的に指定できるオプションについては、各富士通コンパイラのマニュアルをお読みください。

表3.4 Javaの翻訳コマンドの各オプション

オプション	意味
--showme	MPIプログラムの翻訳コマンドがjavacコマンドを呼び出すときの呼出し行を表示します。 実際には翻訳処理は行いません。
--verbose	MPIプログラムの翻訳コマンドがjavacコマンドを呼び出すときの呼出し行を表示します。 翻訳処理も行います。
--help -help -h	ヘルプメッセージを表示します。実際には翻訳処理は行いません。
javac_arguments	javacコマンドに渡すオプションを指定します。

注意

翻訳/結合に関する注意事項

- MPIライブラリは、動的リンクライブラリの形式で提供されます。
- mpifrtpxコマンドおよびmpifrtコマンドでは、以下の翻訳時オプションを自動的に指定します。
  - -f2004
  - -Knointentopt (-Kintentoptの指定は無効になります)
- mpifrtpxコマンドおよびmpifrtコマンドでは、以下の翻訳時オプションを指定した場合、言語要素は小文字しか受け付けられません。
  - -AU
- Fortranプログラムの同じ有効域内で、INCLUDE行によってmpif.hファイルを2回以上展開することはできません。
- mpijavacコマンドは、MPIライブラリのクラスパスを設定せずにJavaプログラムを翻訳できます。
  利用者プログラムのjarファイルを使用した翻訳をするには、環境変数CLASSPATHまたはjavacコマンドの-classpathオプションで、利用者プログラムのjarファイルのパスを設定します。
  環境変数CLASSPATHと-classpathオプションの両方を設定した場合、-classpathオプションが優先されます。

例

mpifrtpxコマンドによるMPIプログラムの翻訳/結合例

1. 利用者プログラム test.f を翻訳し、オブジェクトプログラム test.o を作成します。

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

2. オブジェクトプログラム test.o を結合編集して、実行可能ファイル test を作成します。

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

第4章 MPIプログラムの実行

この章では、MPIプログラムの実行方法について記述しています。

本処理系において、MPIプログラムの実行は、mpiexecコマンドによって行います。mpiexecコマンドは、ジョブ運用ソフトウェアに制御を渡し、計算ノードにMPIプログラムのプロセスを生成して実行します。なお、MPIプロセス内では、ジョブ運用ソフトウェアによって、ジョブ運用ソフトウェアの環境変数PMIX_RANKおよびPLE_RANK_ON_NODEが設定されています。

ジョブ運用ソフトウェアおよび環境変数PMIX_RANKとPLE_RANK_ON_NODEについては、ジョブ運用ソフトウェアのマニュアルをお読みください。

本処理系がベースとしているOpen MPIのオプション等については、ほぼすべて指定可能です。ただし、本章内に掲載していないオプション等を指定した場合、動作が保証されないだけでなく、本システムに深刻な影響を与えることがあります。このため、特に多数のノードやプロセスを使ってMPIプログラムを実行する場合は、本章内に掲載していないオプション等は使用しないでください。

4.1 実行コマンドの形式

実行コマンドの形式は、SPMDモデルで実行する場合、MPMDモデルで実行する場合、実行定義ファイル指定で実行する場合、それぞれで異なります。

1. SPMDモデルで実行する場合

表4.1 SPMDモデルの実行コマンドの形式

コマンド名	オプション
mpiexec	*global_options local_options execfile execfile_arguments*

2. MPMDモデルで実行する場合

表4.2 MPMDモデルの実行コマンドの形式

コマンド名	オプション
mpiexec	*global_options local_options execfile1 execfile1_arguments* : *local_options execfile2 execfile2_arguments* [: *local_options execfile3 execfile3_arguments*] ...

<備考>

- [ ]で囲まれた項目は、異なるプログラムが3つ以上の場合、必要に応じて必要数分の指定を繰り返すことになります。

3. 実行定義ファイル指定で実行する場合

表4.3 実行定義ファイル指定方式による実行コマンドの形式

コマンド名	オプション
mpiexec	*global_options* { -app | --app } 実行定義ファイル *local_options*

実行定義ファイルは、以下の形式で記載します。

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

<備考>

- [ ]で囲まれた項目は、異なるプログラムが2つ以上の場合、必要に応じて必要数分の指定を繰り返すことになります。
- mpiexecコマンドに指定可能な*local_options*は、-tuneオプションと{ -mca | --mca }オプションだけです。
- 実行定義ファイルには、*local_options*の{ -mca | --mca }オプションが指定できません。
- mpiexecコマンドと実行定義ファイルの両方に-tuneオプションを指定した場合は、実行定義ファイルによる指定だけが有効になります。

- mpiexecコマンドの{ -mca | --mca }オプションと、実行定義ファイルにおける-tuneオプションのAMCAパラメーターファイルで、MCAパラメーターの指定が重複した場合は、mpiexecコマンドの指定を優先します。
- 実行定義ファイルの中に“#”または“//”を記載した場合、対象の行については、以降の内容は無視されます。

注意

mpiexecコマンドのコマンド文字列に、1個のコロン(:)だけからなる文字列を指定した場合、区切り文字としてみなされますので、ご注意ください。例えば、*execfile*または*execfile_argument*として、1個のコロンだけからなる文字列は指定できません。

Javaプログラムの場合、MPMDモデルでの実行は保証されません。

*global_options*の形式とオプション説明を“global_optionsの形式とオプション説明”に示します。

*local_options*の形式とオプションの説明を“local_optionsの形式とオプション説明”に示します。

すべてのオプションの説明を“実行コマンドの各オプション”に示します。

本処理系には、MPIライブラリ内部で持ついくつかの変数があり、MCAパラメーターと呼ばれます。MCAパラメーターの値を一時的に変更することで、本処理系の実行時の動作条件を変更できます。MCAパラメーターの詳細は、“4.2 MCAパラメーター”をお読みください。

MCAパラメーターは、環境変数によっても設定できます。環境変数による設定については、“4.3 環境変数”をお読みください。

例

mpiexecコマンドによるMPIプログラム実行の指定例

1. SPMDモデルにおける実行コマンドの指定例

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

MPIプログラムの実行可能ファイルa.outを実行します。プロセスごとに、プログラムの出力結果と統計情報が、指定の方法で生成される名前のファイルに出力されます。

2. MPMDモデルにおける実行コマンドの指定例

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

MPIプログラムの実行可能ファイルa.out、b.out、およびc.outを、それぞれ2、4、および6の並列プロセス数で実行します。

3. 実行定義ファイル形式の指定例

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

MPIプログラムの実行可能ファイルa.out、b.out、およびc.outを、それぞれ2、4、および6の並列プロセス数で実行します。

4. Javaプログラムの場合のSPMDモデルにおける実行コマンドの指定例

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

MPIプログラムのJavaクラスファイルとしてJavaTest.classを8並列プロセス数で実行します。
クラスパスはjavaコマンドの-classpathオプションで指定しています。

*global_options*の形式とオプション説明

```
[ { -app | --app } APP_FILE ]
[ { -debuglib | --debuglib } ]
[ { -h | --help } ]
[ { -of | --of | -std | --std }  FILE  ]
```

```
[ { -oferr | --oferr | -stderr | --stderr }  ERR_FILE  ]
[ { -oferr-proc | --oferr-proc | -stderr-proc | --stderr-proc }  ERR_PROC_FILE  ]
[ { -ofout | --ofout | -stdout | --stdout }  OUT_FILE  ]
[ { -ofout-proc | --ofout-proc | -stdout-proc | --stdout-proc }  OUT_PROC_FILE  ]
[ { -of-proc | --of-proc | -std-proc | --std-proc }  PROC_FILE  ]
[ { -ofprefix | --ofprefix | -stdprefix | --stdprefix }  PREFIX  ]
[ { -stdin | --stdin }  STDIN_FILE  ]
[ { -vcoordfile | --vcoordfile }  VCOORD  ]
[ { -V | --version } ]
```

{ -app | --app } *APP_FILE*

*APP_FILE*の実行定義ファイルを使用して実行する場合に指定します。

appオプションに続けて実行定義ファイルのパス名を指定します。指定するファイルは、ジョブを投入するユーザーに対する読込み権が必要です。

各MPIプログラムに対して並列プロセス数を指定してください。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -debuglib | --debuglib }

デバッグ用のMPIライブラリが結合されます。

実行時の動的デバッグ機能の1つである引数チェック機能を使用する場合、本オプションを指定してください。

本オプションを指定した場合、デバッグ用のMPIライブラリが結合されるため、MPIプログラムの実行時間が非常に遅くなることがあります。ご利用にあたっては、十分にご注意ください。

引数チェック機能の詳細については、“6.17.3 引数チェック機能”をお読みください。

また、メモリチェックなどを行うことができるオープンソースソフトウェアであるValgrindを利用する際には、このオプションの指定が必要です。詳細は“6.18.1 Valgrind 利用上の注意”をお読みください。

{ -h | --help }

本コマンドのヘルプメッセージを表示してmpiexecコマンドを終了します。

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

並列プロセスの標準出力と標準エラー出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL(Access Control List)機能で設定されている制限によって異なります。
本オプションが許可されている場合、*FILE*で指定したファイル名のファイルに出力します。また、*FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

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

並列プロセスの標準エラー出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL機能で設定されている制限によって異なります。
本オプションが許可されている場合、*ERR_FILE*で指定したファイル名のファイルに出力します。また、*ERR_FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

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

並列プロセスの標準エラー出力をプロセスごとに“*ERR_PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*ERR_PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

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

並列プロセスの標準出力をファイルに出力します。出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

出力先は、ジョブ運用ソフトウェアのジョブACL機能で設定されている制限によって異なります。
本オプションが許可されている場合、*OUT_FILE*で指定したファイル名のファイルに出力します。また、*OUT_FILE*の指定にはメタ文字を指定することもできます。
本オプションが許可されていない場合、出力先はジョブ運用ソフトウェアのジョブACL機能の設定に従います。

メタ文字の指定およびジョブACL機能の設定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

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

並列プロセスの標準出力をプロセスごとに“*OUT_PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*OUT_PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

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

並列プロセスの標準出力と標準エラー出力をプロセスごとに“*PROC_FILE.mpiexec.rank*”というファイル名のファイルに出力します。また、*PROC_FILE*の指定にはメタ文字を指定することもできます。メタ文字の指定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

出力先としてファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

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

並列プロセスの標準出力と標準エラー出力の行の先頭に、*PREFIX*に指定したキーワードに対応する内容の文字列を出力します。

*PREFIX*に指定できるキーワードは、date、rank、およびnidです。キーワードはコンマ(,)で区切って複数指定できます(例: date, rank, nid)。キーワードを複数指定した場合は、date、rank、nid の順に対応する内容を出力します。

それぞれのキーワードの指定によって、出力する文字列の形式は次のとおりです。

date

出力する文字列の先頭に出力時刻を付加します。

rank

出力する文字列の先頭にMPI_COMM_WORLDでのランクを付加します。

nid

出力する文字列の先頭にノードIDを付加します。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

ランクは、動的に生成された並列プロセスの場合、ランクの後ろに“*@spawn*”の文字列が付加されます。“*spawn*”の文字列は、動的にプロセスが生成されるごとに、ジョブ運用ソフトウェアによって割り振られる番号を表したものです。

ノードIDは、並列プロセスが割り当てられた計算ノードを識別する番号です。ノードIDについては、ジョブ運用ソフトウェアのマニュアルをお読みください。

{ -stdin | --stdin } *STDIN_FILE*

MPIプログラムの実行によって生成されるすべての並列プロセスの標準入力を、*STDIN_FILE*に指定したファイル名のファイルから読み込みます。ファイル名だけまたは相対パスを指定した場合、ジョブ実行のカレントディレクトリからの相対パスとなります。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -vcoordfile | --vcoordfile } *VCOORD*

MPIプログラムを実行する際、*VCOORD*ファイル内に指定されたプロセス割り当て情報に基づいて、並列プロセスを割り当てることを指定します。ファイル名だけまたは相対パスを指定した場合、mpiexecコマンド実行時のカレントディレクトリからの相対パスとなります。

バックグラウンド実行を利用して、複数のmpiexecコマンドを同時に実行させる場合、本オプションを必ず指定してください。なお、バックグラウンド実行で同時に実行可能な個数は128個までです。

*VCOORD*ファイルの記述形式の詳細は、“4.5 VCOORDファイルの記述形式”をお読みください。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

{ -V | --version }

本コマンドのバージョン情報を表示してmpiexecコマンドを終了します。

プロセスごとに出力される標準出力ファイル名と標準エラー出力ファイル名の文字列について説明します。

“*mpiexec*”の文字列は、ジョブスクリプト内で何回目のmpiexecコマンドの実行であるかを表したものです。“*rank*”の文字列は、実際のMPI_COMM_WORLDでのランクを表したものです。

なお、動的に生成された並列プロセスの場合、ランクの後ろに“*@spawn*”の文字列が付加されます。“*spawn*”の文字列は、動的にプロセスが生成されるごとに、ジョブ運用ソフトウェアによって割り振られる番号を表したものです。

並列プロセスの標準入力/標準出力/標準エラー出力に関する注意事項

各並列プロセスとmpiexecコマンドの標準出力/エラー出力は、通常、ジョブ実行時にジョブ運用ソフトウェアによって生成されるジョブ実行の結果ファイルに接続されます。

mpiexecコマンドの-of/-std系オプション指定の有無による、並列プロセスの標準出力と標準エラー出力の出力先の一覧を下表に示します。

表4.4 -of/-std系オプション指定による並列プロセスの標準出力と標準エラー出力

mpiexecオプション指定	標準出力	標準エラー出力
-of/-std系オプション指定なし	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stdout”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stdout(interact)”の組み合わせ	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stderr”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stderr(interact)”の組み合わせ
-of/-std指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可あり)	*指定したファイル*	*指定したファイル*
-of/-std指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可なし)	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合

mpiexecオプション指定	標準出力	標準エラー出力
	“mpiexec-stdouterr-unit”と“mpiexec-stdout”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stdout(interact)”の組み合わせ	“mpiexec-stdouterr-unit”と“mpiexec-stderr”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stderr(interact)”の組み合わせ
-ofout/-stdout指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可あり)	*指定したファイル*	－
-ofout/-stdout指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可なし)	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stdout”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stdout(interact)”の組み合わせ	－
-oferr/-stderr指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可あり)	－	*指定したファイル*
-oferr/-stderr指定あり (ジョブ運用ソフトウェアのジョブACL機能で許可なし)	－	ジョブ運用ソフトウェアのジョブACL機能の設定(以下に示すpjaclコマンドの出力項目の組み合わせ)に従う。 ・ バッチジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stderr”の組み合わせ ・ 会話型ジョブの場合 “mpiexec-stdouterr-unit”と“mpiexec-stderr(interact)”の組み合わせ
-of-proc/-std-proc指定あり	*指定したファイル名.mpiexec.rank*	*指定したファイル名.mpiexec.rank*
-ofout-proc/-stdout-proc指定あり	*指定したファイル名.mpiexec.rank*	－
-oferr-proc/-stderr-proc指定あり	－	*指定したファイル名.mpiexec.rank*

mpiexecコマンドのリダイレクション指定による標準入力は、各並列プロセスの標準入力として使用できません。mpiexecコマンドのオプションには、すべての並列プロセスに同じ標準入力ファイルを指定する機能および各並列プロセスの標準出力/標準エラー出力の接続先を変更する機能が用意されています。なお、動的に生成された並列プロセスの標準入力、標準出力、および標準エラー出力も、これらmpiexecコマンドへのオプション指定に準じます。

ジョブの実行によって出力される結果ファイルについては、ジョブ運用ソフトウェアのマニュアルをお読みください。

注意

大規模MPIジョブを実行する場合の注意

並列度が高く(おおむね10000個以上のMPIプロセスを生成)、ランクごとに標準出力や標準エラー出力をファイルへ出力するMPIジョブは、ファイルに書き出す処理のシステム負荷を考慮して、以下のように実行することを推奨します。

－ mpiexecコマンドの標準出力/標準エラー出力は、ランク(プロセス)ごとに異なるファイルに出力する。

ー 各ランクの標準出力/標準エラー出力ファイルは同じディレクトリに出力せず、複数のファイルごとに異なるディレクトリに出力する。

ー ランクの標準出力/標準エラー出力がない場合に空ファイルを作成しない。

例

ランク番号が1000ごとに標準出力および標準エラー出力の出力先ディレクトリを変える。また、標準出力や標準エラー出力がない場合に空ファイルは作成しない。

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

注意

DT_RPATHに関する注意事項

MPIプログラムに動的セクション属性DT_RPATHが存在し、かつ、DT_RPATHに"/*製品インストールパス*/lib64"が含まれている場合は、--debuglibオプションの指定が無効となります。このようなMPIプログラムの場合、以下のどちらかの対処をしてください。"*製品インストールパス*"は、システム管理者にお問い合わせください。

ー DT_RPATHに"/*製品インストールパス*/lib64"が含まれないように(再リンクを行うなど)したMPIプログラムを使用してください。

ー 実行時に、動的リンカの--inhibit-rpathオプションを指定してください。オプションの指定方法を以下に示します。

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

- 実行するパッケージに対応した環境変数PATHとLD_LIBRARY_PATHを設定してください。
- /lib64/ld-linux-aarch64.so.1の--inhibit-rpathオプションの引数(LIST)は、":MPIプログラム"の形式で指定してください。

*local_options*の形式とオプション説明

```
[ -tune  AM_FILE ]
[ -x  NAME=VALUE ]
[ { -mca | --mca } MCA_PARAM_NAME  MCA_PARAM_VALUE ]
[ { -c | -np | --np | -n | --n }  N ]
[ { -fjdbg-dlock | --fjdbg-dlock } ]
[ { -fjdbg-sig | --fjdbg-sig } SIGNAL ]
[ { -fjdbg-out-dir | --fjdbg-out-dir } OUTPUT-DIR ]
[ { -gdbx | --gdbx } "[ RANK: ] COMMAND-FILE [ :... ]" ]
```

-tune *AM_FILE*

対応するMPIプログラムに対して、AMCAパラメーターファイル(MCAパラメーターの設定ファイル)のパス名を指定します。

すべてのプログラムに対するMCAパラメーターの指定内容(値)が同じになるように指定してください。結果的に異なる値が指定された場合には動作を保証できません。

ファイル内での指定方法は、次のとおりです。

ー 各行には、以下の形式で指定してください。

```
MCAパラメーター名=値
```

ー 同一のMCAパラメーターに複数の値を指定する場合には、以下のようにコンマ(,)で区切って指定してください。

```
MCAパラメーター名=値1,値2
```

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

-x *NAME=VALUE*

MPIプログラムに対して環境変数に値を設定することを指定します。

*NAME*は環境変数名を表します。*VALUE*はその環境変数に設定する値を表します。指定に空白を含めたい場合、次のように引用符で括ることで可能となります。

```
"NAME=VALUE"
```

本オプションは、複数回の指定が可能です。複数の環境変数を指定したい場合、本オプションをその回数指定することで可能となります。ただし環境変数名を重複して指定した場合、最後に指定した内容を優先します。

```
指定例:
-x OMP_NUM_THREADS=8  -x THREAD_STACK_SIZE=4096
```

{ -mca | --mca } *MCA_PARAM_NAME MCA_PARAM_VALUE*

対応するMPIプログラムに対してMCAパラメーターを指定します。

すべてのプログラムに対するMCAパラメーターの指定内容が同じ値になるように指定してください。異なる値が指定された場合には動作を保証できません。

{ -c | -np | --np | -n | --n } *N*

対応するMPIプログラムの並列プロセス数(整数)を指定します。

SPMDモデルでの実行では、本オプションの指定が省略された場合、生成できる最大値が指定されたものとみなします。

MPMDモデルでの実行では、本オプションを必ず指定してください。

本オプションを重複して指定した場合、最後に指定したパラメーターを優先します。

本システムでは、MPIプログラムの実行において、各並列プロセスを1次元から3次元のトーラス形状に割り当てることができます。トーラス形状の指定方法と並列プロセスの配置方法については、ジョブ運用ソフトウェアのマニュアルをお読みください。

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

デバッグ支援機能のデッドロック調査機能を有効にします。デッドロック調査機能の詳細は並列実行デバッガ使用手引書をお読みください。

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

デバッグ支援機能の異常終了調査機能を有効にします。異常終了調査機能の詳細は並列実行デバッガ使用手引書をお読みください。

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

デバッグ支援機能のデッドロック調査機能および異常終了調査機能における調査結果ファイルを格納するディレクトリを指定します。デッドロック調査機能および異常終了調査機能の詳細は並列実行デバッガ使用手引書をお読みください。

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

デバッグ支援機能のコマンドファイルによるデバッガ制御機能を有効にします。コマンドファイルによるデバッガ制御機能の詳細は並列実行デバッガ使用手引書をお読みください。

実行コマンドの各オプション

execfile

MPIプログラムの実行可能ファイル、MPIプログラム以外の実行可能ファイル、またはシェルスクリプトを指定します。

JavaのMPIプログラムの場合はjavaコマンドを指定します。

指定する実行可能ファイルやシェルスクリプトが環境変数PATHで指定したパスに存在しない場合は、絶対パスまたはmpiexecコマンド実行時のカレントディレクトリからの相対パスで指定する必要があります。

シェルスクリプト内に、複数のMPIプログラムを実行する記述はできません。シェルスクリプト内に複数のMPIプログラムを実行する記述をした場合、動作は保証されません。

mpiexecコマンドを再帰的に実行するような指定はできません。再帰的に実行された場合、エラーメッセージを出力した後で、mpiexecコマンドは異常終了します。例えば、*execfile*にmpiexecコマンドの起動ファイル名を指定してはいけません。

ポイント

シェルスクリプトを指定する場合、シェルスクリプトには実行権が必要です。

execfile_arguments

*execfile*に渡す引数を指定します。

Javaプログラムの場合は、クラス名や -jar オプションなどのjavaコマンドに対するオプションを指定します。

execfile1
execfile2
execfile3

MPIプログラムの実行可能ファイル、MPIプログラム以外の実行可能ファイル、またはシェルスクリプトを指定します。

MPMDモデルでの実行では、並列に実行するプログラムの実行可能ファイルやシェルスクリプトを、コロンで区切って指定します。

指定する実行可能ファイルやシェルスクリプトが環境変数PATHで指定したパスに存在しない場合は、絶対パスまたはmpiexecコマンド実行時のカレントディレクトリからの相対パスで指定する必要があります。

シェルスクリプト内に、複数のMPIプログラムを実行する記述はできません。シェルスクリプト内に複数のMPIプログラムを実行する記述をした場合、動作は保証されません。

mpiexecコマンドを再帰的に実行するような指定はできません。再帰的に実行された場合、エラーメッセージを出力した後で、mpiexecコマンドは異常終了します。例えば、*execfile1*、*execfile2*、または*execfile3*にmpiexecコマンドの起動ファイル名を指定してはいけません。

ポイント

シェルスクリプトを指定する場合、シェルスクリプトには実行権が必要です。

execfile1_arguments
execfile2_arguments
execfile3_arguments

*execfile1*に渡す引数を指定します。
*execfile2*に渡す引数を指定します。
*execfile3*に渡す引数を指定します。

4.2 MCAパラメーター

本処理系では、MPIプログラムの実行の際、MPIライブラリ内部の変数の値を一時的に変更することによって、本処理系の動作条件を変更できます。このような変数をMCAパラメーターと呼びます。ここでは、MCAパラメーターの種類、使用方法について説明します。これらMCAパラメーターは、環境変数によって設定できます。環境変数による設定方法については、“4.3 環境変数”をお読みください。

本処理系におけるMCAパラメーターの指定方法には、次の方法があります。

- mpiexecコマンドの-tuneオプションを使用して、MCAパラメーターの設定ファイル(AMCAパラメーターファイル)内に指定
- mpiexecコマンドの-mcaオプションを使用して、MCAパラメーターを直接指定
- MCAパラメーターを環境変数によって指定

同じMCAパラメーターに対して異なる方法で指定した場合、優先順位の高い指定が有効となります。MCAパラメーターの指定方法の違いによる優先順位を下表に示します。

表4.5 MCAパラメーターの指定方法と優先順位

優先順位	MCAパラメーター設定方法	使用例
1	mpiexecコマンドの-mcaオプション	-mca btl_tofu_eager_limit 4096
2	環境変数	export OMPI_MCA_btl_tofu_eager_limit=4096

優先順位	MCAパラメーター設定方法	使用例
3	-tuneオプションに指定したAMCAパラメーターファイル(MCAパラメーターの設定ファイル)	-tune mca_file

備考:「優先順位」は、値が小さいほど優先順位が高いことを表します。

本処理系で使用できるMCAパラメーター

本処理系で使用できるMCAパラメーターを以下に示します。各MCAパラメーターの機能概要は、それぞれ、表タイトルの括弧内で説明しています。
値が整数だけの場合はバイト単位、整数に続けて“k”を指定した場合はKiB単位、整数に続けて“m”を指定した場合はMiB単位での指定となります。

表4.6 btl_tofu_eager_limit (通信方式を切り替えるしきい値を変更)

MCAパラメーターの値	内容
1以上の整数値	高速型通信モードにおけるEager通信方式とRendezvous通信方式を切り替える“しきい値”となるメッセージのサイズ(バイト数)を指定します。指定したメッセージのサイズ(バイト数)よりも小さいメッセージについてはEager通信方式によって送信します。厳密には、実際のメッセージのサイズ(バイト数)に内部的に付加されるヘッダー分のサイズ(数十バイト)を加えた値となります。高速型通信モードについての詳細は、“6.10.1 高速型通信モードと省メモリ型通信モードの切り替え”をお読みください。 256よりも小さい値を指定した場合、256に設定します。 大きい値を指定した場合、内部的に指定した値よりも小さい値に切り下げられることがありますので、ご注意ください。一般に128000程度の値まで有効に指定できますが、メモリの使用状況によっては、もっと小さい値までしか有効とならないことがあります。実際には、MCAパラメーターcommon_tofu_large_recv_buf_sizeに指定したLarge受信バッファサイズおよび送信バッファサイズに依存します。具体的に、有効に指定できるのは、送信バッファサイズに起因する512000程度の値を上限として、おおよそ次の式で求められる値までとなります。厳密には、内部的に使用する制御情報なども考慮すると、さらに数百バイト程度引いた値となりますが、ここでは無視できるものとしています。 Large受信バッファサイズ ÷ 2 本処理系では、通常、高速型通信モードにおけるこの“しきい値”をシステム内部で動的に適切な値に決めています。メッセージのサイズ(バイト数)だけでなく、通信を行う計算ノードの距離についても考慮しているためです。例えば、隣接する計算ノード間でのメッセージ通信の場合、内部的に38896に設定されます。メッセージ通信を行う計算ノード間の距離が長くなるに従って、この“しきい値”も大きくしています。このMCAパラメーターでは、計算ノードの距離に関係なく、指定した値を“しきい値”として使用します。 詳細については、“6.5 Eager通信方式とRendezvous通信方式”をお読みください。

備考:MCAパラメーター名に使われている文字列btlの“l”(エル)は英小文字です。

表4.7 coll_base_reduce_commute_safe (リダクション演算の順序を保証)

MCAパラメーターの値	内容
1	リダクション演算を行う集団通信MPI_REDUCEルーチン、MPI_IREDUCEルーチン、MPI_ALLREDUCEルーチン、MPI_IALLREDUCEルーチン、MPI_REDUCE_SCATTERルーチン、MPI_IREDUCE_SCATTERルーチン、MPI_REDUCE_SCATTER_BLOCKルーチン、MPI_IREDUCE_SCATTER_BLOCKルーチン、およびMPI_SCANルーチンにおいて、リダクション演算の順序を保証します。 これらリダクション演算を行う集団通信では、通信条件に応じて、通信時間が最適になるように演算順序を変えることがあります。リダクション演算の順序が変わることで、結果的に計算結果の精度にも影響を与えることがあります。 本パラメーターの値を指定して、演算順序を固定化できます。

MCAパラメーターの値	内容
	なお、リダクション演算の順序が固定化される場合、通信時間が長くなることにご注意ください。 詳細については、“6.8 集団通信におけるリダクション演算の順序保証”をお読みください。
0	リダクション演算の順序を保証しません。通信時間がなるべく短くなるように、内部的にリダクション演算の順序を変えることがあります。なお、通信条件が同じである場合、同じプログラムを何度実行しても計算結果は同じになります。 本パラメーターの省略値は0です。

備考：MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.8 coll_tbi_intra_node_reduction (1ノード内に複数プロセスが割り当てられる場合に、バリア通信がノード内の浮動小数点型または複素数型データのリダクション演算で使用するアルゴリズムを指定)

MCAパラメーターの値	内容
2	“6.12.3 MPI_REDUCEおよびMPI_ALLREDUCE”において説明するバリア通信の適用条件に該当する場合に、ノード内の浮動小数点型または複素数型データのリダクション演算で、バリア通信を使用しません。 代わりに、ソフトウェアによるrecursive_doublingアルゴリズムを使用します。
3	ノード内の浮動小数点型または複素数型データのリダクション演算で、バリア通信を使用します。 必要なバリアゲートの数を節約できるbinary_treeアルゴリズムを使用します。 本パラメーターの省略値は3です。
4	ノード内の浮動小数点型または複素数型データのリダクション演算で、バリア通信を使用します。 本パラメーターに3を指定した場合より多くのバリアゲート数を必要としますが、高速化が期待されるrecursive_doublingアルゴリズムを使用します。

備考：MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.9 coll (すべての集団通信に共通して適用される設定を変更)

MCAパラメーターの値	内容
^tbi	MPI_BARRIERルーチン、MPI_BCASTルーチン、MPI_REDUCEルーチン、MPI_ALLREDUCEルーチンのいずれにおいても、バリア通信機能(Tofuインターコネクトのハードウェア機構)を適用しないことを指定します。詳細については、“6.12 Tofuバリア通信による高速化”をお読みください。 バリア通信機能を適用する場合は、本パラメーターを指定しないでください。

備考：MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.10 coll_tbi_repeat_max (バリア通信により通信を行うメッセージの長さの範囲を制御)

MCAパラメーターの値	内容
1以上の整数値	MPI_BCASTルーチン、MPI_REDUCEルーチン、およびMPI_ALLREDUCEルーチンにおいて、この値の回数を上限としてバリア通信を行います。 1回のバリア通信につき演算できるメッセージの要素数は限られています。そのため、MPIルーチンの引数で、バリア通信1回あたりの上限を超える複数の要素が指定された場合、内部でバリア通信を複数回実施することで、バリア通信が適用される範囲を拡張できます。 本パラメーターの省略値は1です。 バリア通信1回あたりの上限については、“表6.19 MPI_BCASTルーチンでバリア通信が適用可能な組み合わせ”および“表6.20 MPI_REDUCEルーチンおよびMPI_ALLREDUCEルーチンでバリア通信が適用可能な演算の組み合わせ”を参照してください。

例

- MPI_BCASTルーチン、基本データ型(MPI_UINT64_T)の場合
  - メッセージの要素数に6を指定した場合、バリア通信機能が適用されます。
  - メッセージの要素数に7を指定した場合、本MCAパラメーターを指定しないとバリア通信機能が適用されませんが、本MCAパラメーターに2を指定すればバリア通信機能が適用されます。
- MPI_BCASTルーチンで、MPI_TYPE_CONTIGUOUSルーチンを使い、基本データ型(MPI_UINT64_T)の要素を7個連結して作成したデータ型を指定した場合
  - メッセージの要素数に1を指定した場合、本MCAパラメーターを指定しないとバリア通信機能が適用されませんが、本MCAパラメーターに2を指定すればバリア通信機能が適用されます。
  - メッセージの要素数に2を指定した場合、本MCAパラメーターに3を指定することで、バリア通信機能が適用されます。
- MPI_REDUCEルーチン、基本データ型(MPI_DOUBLE)、MPI_SUM演算の場合
  - メッセージの要素数に6を指定した場合、本MCAパラメーターを指定しないとバリア通信機能が適用されませんが、本MCAパラメーターに2を指定すればバリア通信機能が適用されます。
  - メッセージの要素数に9を指定した場合、本MCAパラメーターに3を指定することでバリア通信機能が適用されます。

表4.11 coll_tbi_use_on_bcast (MPI_BCASTルーチンにおいてバリア通信機能を適用)

MCAパラメーターの値	内容
1	MPI_BCASTルーチンにおいてバリア通信機能(Tofuインターコネクトのハードウェア機構)を適用することを指定します。 詳細については、“6.12.2 MPI_BCAST”をお読みください。 本パラメーターの省略値は1です。
0	MPI_BCASTルーチンにおいてバリア通信機能を適用しないことを指定します。

備考:MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.12 coll_tbi_use_on_comm_dup (MPI_COMM_DUPルーチン、MPI_COMM_IDUPルーチン、MPI_COMM_DUP_WITH_INFOルーチンで作成されたコミュニケータにおいてバリア通信機能を適用)

MCAパラメーターの値	内容
1	“6.12 Tofuバリア通信による高速化”において説明するバリア通信機能(Tofuインターコネクトのハードウェア機構)が適用される条件で、MPI_COMM_DUPルーチン、MPI_COMM_IDUPルーチン、MPI_COMM_DUP_WITH_INFOルーチンによって作成されたコミュニケータにおいてバリア通信機能を適用することを指定します。 本パラメーターの省略値は1です。
0	MPI_COMM_DUPルーチン、MPI_COMM_IDUPルーチン、MPI_COMM_DUP_WITH_INFOルーチンによって作成されたコミュニケータにおいてバリア通信機能を適用しないことを指定します。 詳細については、“6.12.4 バリア通信の注意事項”をお読みください。

備考:MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.13 coll_tbi_use_on_max_min (浮動小数点型MPI_MAX/MPI_MIN演算にバリア通信機能を適用)

MCAパラメーターの値	内容
1	MPI_REDUCEルーチン、MPI_ALLREDUCEルーチンの浮動小数点型MPI_MAX/MPI_MIN演算において、バリア通信機能(Tofuインターコネクトのハードウェア機構)を適用することを指定します。

MCAパラメーターの値	内容
	以下に示す特別な条件のどちらかでバリア通信を適用するか否かによって計算結果が異なることがあります。このため、MCAパラメーターに指定する値の変更にあたっては注意が必要です。 ・ 演算に使われる値にNaNが含まれている場合は、以下に注意してください。 — バリア通信を適用する場合は、NaN以外の値を比較した結果となります。ただし、すべての値がNaNの場合は、NaNのうちのどれか1つの値となります。 — バリア通信を適用しない場合は、実行時の状況によって、NaN以外の値を比較した結果またはNaNのうちのどれか1つの値となります。 ・ 演算に使われる値に+0.0と-0.0の両方が含まれている場合は、以下に注意してください。 — バリア通信を適用する場合は、0の符号を考慮して比較します。(+0.0 > -0.0) — バリア通信を適用しない場合は、0の符号は考慮されません。+0.0と-0.0の大小比較において、どちらが選ばれるかは実行時の状況に依存します。 本パラメーターの省略値は1です。
0	MPI_REDUCEルーチンおよびMPI_ALLREDUCEルーチンの浮動小数点型MPI_MAX/MPI_MIN演算においてバリア通信機能を適用しないことを指定します。

備考:MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.14 coll_tuned_bcast_same_count (ランク間で同じ要素数を用いたMPI_BCASTルーチン/MPI_IBCASTルーチンの通信を高速化)

MCAパラメーターの値	内容
1	MPI_BCASTルーチンまたはMPI_IBCASTルーチンにおいて、ランク間で同じ要素数を用いた通信を行う場合に指定します。 本MCAパラメーターはMPI_ALLGATHERルーチンとMPI_ALLGATHERVルーチンにも影響があります。 詳細については、“6.13 ランク間で要素数が同一の場合のMPI_BCASTルーチン/MPI_IBCASTルーチン”をお読みください。
0	MPI_BCASTルーチンまたはMPI_IBCASTルーチンにおいて、ランク間で異なる要素数を用いた通信を行う場合に指定します。 本パラメーターの省略値は0です。

備考:MCAパラメーター名に使われている文字列collの“l”(エル)はどちらも英小文字です。

表4.15 coll_tuned_prealloc_size (集団通信ルーチン内部で使用する静的作業領域の大きさを指定)

MCAパラメーターの値	内容
1以上の整数値	集団通信の以下のルーチンにおいて、静的に確保する作業領域の大きさ(MiB)を指定します。 ・ MPI_ALLREDUCEルーチン ・ MPI_REDUCEルーチン ・ MPI_REDUCE_SCATTER_BLOCKルーチン ・ MPI_REDUCE_SCATTERルーチン ・ MPI_ALLGATHERルーチン ・ MPI_GATHERルーチン ・ MPI_SCATTERルーチン ・ MPI_ALLTOALLルーチン

MCAパラメーターの値	内容
	これらのルーチンを複数回呼び出すようなMPIプログラムでは、本パラメーターの指定によって静的に確保した領域を有効利用することで、MPIプログラムの実行時間を短縮できることがあります。 以下の大きさが指定する目安となります。 MPI_ALLREDUCEルーチンの場合: (プログラム中の送信メッセージのサイズ × 2) + 2MiB MPI_REDUCEルーチン/MPI_REDUCE_SCATTER_BLOCKルーチンの場合: プログラム中の送信メッセージのサイズ × 3) + 2MiB MPI_REDUCE_SCATTERルーチンの場合: (プログラム中の送信メッセージのサイズ × 2) + 2MiB MPI_ALLGATHERルーチンの場合: プログラム中の受信メッセージのサイズ + 2MiB MPI_GATHERルーチンの場合: プログラム中のルートプロセスの受信メッセージのサイズ + 2MiB MPI_SCATTERルーチンの場合: プログラム中のルートプロセスの送信メッセージのサイズ + 2MiB MPI_ALLTOALLルーチンの場合: (プログラム中の送信メッセージのサイズ × コミュニケータ内のランク数) + 2MiB 本パラメーターの指定によって静的に確保される作業領域の大きさが、集団通信ルーチンの処理で必要とされる作業領域の大きさよりも小さい場合、静的に確保した作業用領域は使用されません。 本パラメーターの省略値は6(MiB)です。 ただし、MCAパラメーターcommon_tofu_use_memory_poolの値が1の場合は、確保される静的な作業領域が本パラメーターに指定した値よりも大きくなる場合があります。ご注意ください。MCAパラメーターcommon_tofu_use_memory_poolについては、"表4.31 common_tofu_use_memory_pool (MPIライブラリ用のメモリプールを使用)"を参照してください。
0	静的な作業領域を確保しないことを指定します。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

表4.16 common_tofu_conv_dim (各プロセスの座標の次元をより高次元に変換してMPIプログラムを実行)

MCAパラメーターの値	内容
2	可能であれば2次元のジョブであるとみなしてMPIプログラムを実行することを指定します。 ジョブが1次元で実行された場合のみ有効です。 詳細については、"6.15 ジョブ次元変換機能"をお読みください。
3	可能であれば3次元のジョブであるとみなしてMPIプログラムを実行することを指定します。 ジョブが1次元または2次元で実行された場合のみ有効です。 詳細については、"6.15 ジョブ次元変換機能"をお読みください。
0	実行開始時に指定された次元のジョブとして、MPIプログラムを実行することを指定します。 本パラメーターに0、2、または3以外の値を指定した場合、0が指定されたものとみなされます。

MCAパラメーターの値	内容
	本パラメーターの省略値は0です。

表4.17 common_tofu_conv_dim_log (各プロセスの座標の次元をより高次元に変換してMPIプログラムを実行した際のログを出力)

MCAパラメーターの値	内容
1	ランク0の並列プロセスが、MPIライブラリ内での次元の拡張に関するログを標準エラーに出力することを指定します。次元変換機能の適用が行われなかった場合、ログにはその理由が含まれます。出力内容の詳細については“6.15.2 ジョブ次元変換機能のログ出力”をお読みください。
0	MPIライブラリ内での次元の拡張に関するログを出力しないことを指定します。 本パラメーターに0または1以外の値を指定した場合、0が指定されたものとみなされます。 本パラメーターの省略値は0です。

表4.18 common_tofu_fastmode_threshold (高速型通信モードに切り替わる条件を変更)

MCAパラメーターの値	内容
0以上の整数値	省メモリ型通信モードから高速型通信モードに切り替えるときの条件となる通信回数を指定します。 0を指定した場合、高速型通信モードで通信を行う通信相手プロセス数の上限に達していない限り、最初から高速型通信モードで通信を行います。-1以下の数値を指定した場合、0が指定されたものとみなされます。 詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は16です。

表4.19 common_tofu_large_recv_buf_size (Large受信バッファのサイズを変更)

MCAパラメーターの値	内容
1024から16700000までの整数値	Large受信バッファのサイズ(バイト数)を指定します。 1024以下の値を指定した場合、1024が指定されたものとみなされます。16700000以上の値を指定した場合、16700000が指定されたものとみなされます。 Large受信バッファの詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は1048576です。 ただし、MCAパラメーターcommon_tofu_use_memory_poolの値が1の場合は、本パラメーターの省略値が1048064に変更されます。MCAパラメーターcommon_tofu_use_memory_poolについては、“表4.31 common_tofu_use_memory_pool (MPIライブラリ用のメモリプールを使用)”を参照してください。

表4.20 common_tofu_max_fastmode_procs (高速型通信モードで通信できるプロセス数の上限を変更)

MCAパラメーターの値	内容
-1 または 0以上の整数値	各並列プロセスが通信相手となるプロセスと高速型通信モードで通信を行うことができるプロセス数の上限値を指定します。 -1を指定した場合、すべてのプロセスとの通信を高速型通信モードで行います。0を指定した場合、高速型通信モードを使用せず、すべてのプロセスとの通信を省メモリ型通信モードで行います。-2以下の数値を指定した場合、-1が指定されたものとみなされます。 詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は256です。

表4.21 common_tofu_max_tnis (TNIの使用数の上限値を変更)

MCAパラメーターの値	内容
1以上の整数値	使用するネットワークインターフェース装置(TNI)の上限を指定します。最大数(6)を超えて指定した場合、実際に使用可能な数になります。詳細については、“6.7 複数TNIの利用”をお読みください。
-1	使用可能な最大のネットワークインターフェース装置(TNI)を使用します。 0または-2以下の数値を指定した場合、-1が指定されたものとみなされます。 本パラメーターの省略値は-1です。

表4.22 common_tofu_medium_recv_buf_size (Medium受信バッファのサイズを変更)

MCAパラメーターの値	内容
256から16700000までの整数値	Medium受信バッファのサイズ(バイト数)を指定します。 256以下の値を指定した場合、256が指定されたものとみなされます。16700000以上の値を指定した場合、16700000が指定されたものとみなされます。 Medium受信バッファの詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は2048です。

表4.23 common_tofu_memory_limit (メモリ使用量の制限値を指定)

MCAパラメーターの値	内容
0以上の整数値	本処理系自身が使用できるメモリ使用量の制限値(MiB)を指定します。 動的プロセス生成を使用する場合またはコミュニケータを共有しないMPIプロセスグループ間での通信の確立を使用する場合、メモリ使用量の制限値は指定できません。 メモリ使用量の制限を無効にする場合、0を指定してください。-1以下の数値が指定された場合、0を指定したものとみなされます。 詳細については、“6.11.3 メモリ使用量の制限値の指定”をお読みください。 本パラメーターの省略値は0MiBです。

表4.24 common_tofu_memory_limit_peers (メモリ使用量を制限する際に想定する通信相手プロセス数を指定)

MCAパラメーターの値	内容
0以上の整数値	本処理系自身が使用できるメモリ使用量を制限する場合に想定する、通信相手のプロセス数を指定します。 本パラメーターの省略値として、同じコミュニケータMPI_COMM_WORLDに属するプロセスの数が設定されますが、より正確な自動チューニングを行うためには、MPI統計情報によって得られるTofu通信のためのコネクション数を指定する必要があります。 メモリ使用量の制限値を指定する方法の詳細については、“6.11.3 メモリ使用量の制限値の指定”をお読みください。 本パラメーターの省略値は、同じコミュニケータMPI_COMM_WORLDに属するプロセスの数です。

表4.25 common_tofu_memory_saving_method (省メモリ型通信モードで使用する方式を変更)

MCAパラメーターの値	内容
1	省メモリ型通信モードでの通信時に、Medium受信バッファを使用する方式を使います。 1未満の値または3以上の値を指定した場合は、1が指定されたものとみなされます。 詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は1です。

MCAパラメーターの値	内容
2	省メモリ型通信モードでの通信時に、Shared受信バッファを使用する方式を使います。 詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。

表4.26 common_tofu_num_mrq_entries (完了キューのエントリ数を指定)

MCAパラメーターの値	内容
2048、8192、32768、131072、524288、または2097152	Tofuインターコネクトの完了キューのエントリ数を指定します。 [mpi::common-tofu::tofu-mrq-overflow]から始まるエラーメッセージが出力された場合は、本パラメーターの値を大きくすることで、エラーを回避できる場合があります。また、本パラメーターの値を小さくすることで、MPIプロセスのメモリ使用量を削減できます。 本パラメーターに指定できる値は、2048、8192、32768、131072、524288、または2097152です。一覧にない値を指定した場合、一覧の中で一番近い値が指定されたものとみなされます。一番近い値が2つある場合、大きい方の値が指定されたものとみなされます。 本パラメーターの省略値は131072です。 [mpi::common-tofu::tofu-mrq-overflow]から始まるエラーメッセージの詳細については、“7.3 通信ライブラリのエラーメッセージ”をお読みください。

表4.27 common_tofu_packet_gap (パケットの転送間隔時間を変更)

MCAパラメーターの値	内容
0から255までの整数値	パケットの転送間隔時間を、ギャップ値と呼ばれる整数値で指定します。 ギャップ値の1単位は、以下に述べる、パケットの最大転送サイズのデータの転送にかかる時間の1/8の時間に相当します。 本パラメーターには、0から255までの整数値を指定できます。-1以下の値を指定した場合、0が指定されたものとみなされます。255より大きい値を指定した場合、255が指定されたものとみなされます。 本パラメーターの省略値は0です。 メッセージの転送は、MPIライブラリ内でパケットと呼ばれる単位で送受信が行われます。1つのパケットには、最大転送サイズと呼ばれる上限値があります。最大転送サイズよりも大きいメッセージを転送する場合、各パケットのサイズが最大転送サイズ以下となるように、複数のパケットに分割されて転送されます。この最大転送サイズは、MCAパラメーターcommon_tofu_packet_mtuによって変更できます。MCAパラメーターcommon_tofu_packet_mtuの詳細は、“表4.28 common_tofu_packet_mtu (パケットの最大転送サイズを変更)”をお読みください。 パケットの転送間隔時間を調整することで、帯域の制御ができるため、他のメッセージ通信が同時に行われるような場合、通信のスループット時間が改善できることがあります。例えば、本パラメーターにギャップ値8を指定した場合、パケット間にちょうど1つのパケットの転送にかかる時間だけの時間間隔をあけたことになり、対象となるメッセージ転送の帯域を1/2に調整したことになります。しかし、場合によっては、逆に性能劣化を招くこともありますので、ご利用にあたっては十分な注意が必要です。

表4.28 common_tofu_packet_mtu (パケットの最大転送サイズを変更)

MCAパラメーターの値	内容
256、512、768、1024、1280、1536、または1792	メッセージの転送は、MPIライブラリ内でパケットと呼ばれる単位で送受信が行われます。本パラメーターは、パケットの最大転送サイズをバイト数で指定します。 本パラメーターに指定できる値は、256、512、768、1024、1280、1536、または1792です。一覧にない値を指定した場合、256単位で切り捨てた値、または切り上げた値がパケットの最大転送サイズとなります。 本パラメーターの省略値は1792です。 サイズの大きい複数のメッセージ通信が同時に行われるような場合、MCAパラメーターcommon_tofu_packet_gapの指定と組み合わせ、本パラメーターの値を変更することにより、通信

MCAパラメーターの値	内容
	のスループット時間が改善できることがあります。しかし、場合によっては、逆に性能劣化を招くこともありますので、ご利用にあたっては十分な注意が必要です。MCAパラメーターcommon_tofu_packet_gapの詳細については、“表4.27 common_tofu_packet_gap (パケットの転送間隔時間を変更)”をお読みください。

表4.29 common_tofu_shared_recv_buf_size (Shared受信バッファのサイズを変更)

MCAパラメーターの値	内容
65536以上の整数値	Shared受信バッファのサイズ(バイト数)を指定します。 指定された値が2のべき乗でない場合は、2のべき乗の値へと切り上げられます。65536未満の値が指定された場合は、65536が指定されたものとみなされます。 Shared受信バッファの詳細については、“6.10 メモリ使用量を抑えるための考慮”をお読みください。 本パラメーターの省略値は16777216です。

表4.30 common_tofu_use_multi_path (複数の通信経路を利用した1対1通信を行う)

MCAパラメーターの値	内容
1	1対1通信において、複数の通信経路を利用した通信、すなわちトランキングを行うことを指定します。 複数のTNIを使用して複数の通信経路を確保するため、本MCAパラメーターの効果は、使用できるTNI数によっても影響を受けます。また、通信条件によっては、逆に通信性能が悪くなることもあります。ご利用にあたっては十分な注意が必要です。 複数のTNI利用についての詳細は、“6.7 複数TNIの利用”をお読みください。
0	1対1通信において、複数の通信経路を利用しないことを指定します。 本パラメーターの省略値は0です。

表4.31 common_tofu_use_memory_pool (MPIライブラリ用のメモリプールを使用)

MCAパラメーターの値	内容
1	MPIライブラリ用のメモリプールを使用します。 ページ単位で獲得したメモリプールを使用することで、MPIライブラリ内で保持する通信バッファのメモリ領域とユーザープログラム側のメモリ領域が異なるページに割り当たるようになります。これにより、未完了の通信がない状態であれば、MPIプログラム内からのプロセス生成時の制約を回避できるようになります。 ただし、MCAパラメーターcommon_tofu_memroy_saving_methodに2を指定している場合は、本機能を有効にしても、MPIプログラム内からのプロセス生成時の制約を回避できないことがあります。ご注意ください。MPIプログラム内からのプロセス生成時の制約については、“6.9 MPIプログラム内からのプロセス生成について”を参照してください。 本機能を有効にすると、下記の影響があります。ご注意ください。 ・ Large受信バッファサイズの省略値が、1048576 Bではなく1048064 Bになります。Large受信バッファについては、“6.10 メモリ使用量を抑えるための考慮”を参照してください。 ・ メモリ使用量が大きくなる場合があります。 ・ 環境変数XOS_MMM_L_HPAGE_TYPE=noneを指定した場合でも、MPIライブラリではラージページを使用します。環境変数XOS_MMM_L_HPAGE_TYPEおよびラージページについては、“ジョブ運用ソフトウェア エンドユーザ向けガイド HPC拡張機能編”をお読みください。 本パラメーターに0、1以外の値を指定した場合は、1が指定されたものと見なされます。
0	MPIライブラリ用のメモリプールを使用しません。

MCAパラメーターの値	内容
	MPIライブラリ内で保持する通信バッファのメモリ領域は、ユーザープログラム側のメモリ領域と同じページ上に割り当てられる可能性があります。このため、未完了の通信がない状態であっても、MPIプログラム内からのプロセス生成時の制約を回避できないことがあります。プログラム内からのプロセス生成時の制約については、“6.9 MPIプログラム内からのプロセス生成について”を参照してください。 本パラメーターの省略値は0です。

表4.32 mca_base_param_file_prefix (AMCAパラメーターファイルの指定)

MCAパラメーターの値	内容
AMCAパラメーターファイルのファイルパス名	指定したファイルをAMCAパラメーターファイル(MCAパラメーターの設定ファイル)として解釈します。この設定ファイル内に記載されるMCAパラメーターがすでに環境変数として設定されている場合、この設定ファイルに記載されている該当のMCAパラメーターの設定が無効になります。 無効なファイルパス名を指定した場合、警告メッセージが出力され、AMCAパラメーターファイルの指定は無効になります。

表4.33 mpi_check_buffer_write (通信バッファの書込み破壊を監視)

MCAパラメーターの値	内容
1	ノンブロッキング通信中の送信バッファへの書込み破壊の監視を行うことを指定します。 送信が完了する前に、その送信バッファへの書込みが発生した場合、標準エラー出力にメッセージおよびスタックのトレース情報を出力してMPIプログラムの実行を終了します。 通信バッファの書込み破壊の監視の詳細については、“6.17.2 通信バッファの書込み破壊の監視”をお読みください。
0	通信バッファの書込み破壊の監視を行わないことを指定します。 本パラメーターの省略値は0です。

表4.34 mpi_java_eager (Javaプログラムのテンポラリバッファサイズを変更)

MCAパラメーターの値	内容
1以上の整数値	Javaプログラムの場合に確保されるテンポラリバッファのサイズを指定します。省略値は65536(64KiB)です。 あらかじめ領域確保したテンポラリバッファを使いまわすことで、Javaヒープ領域とCヒープ領域間のデータコピーの時間が削減され、ブロッキング通信ルーチンの処理時間が短縮できます。 本パラメーターの設定値の目安は、1回のルーチンで送信または受信するデータのサイズ(データ型サイズ × 要素数)です。 目安より小さい値を設定した場合、領域確保の回数が増えるので、ブロッキング通信ルーチンの処理時間が長くなる可能性があります。 目安より大きい値を設定した場合、ブロッキング通信ルーチンの処理時間が改善される可能性は少ないです。 なお、本パラメーターは、プリミティブ型が使えるブロッキング通信ルーチンで、プリミティブ型を使ったときだけ有効です。 また、本パラメーターに大きな値を指定することによって、メモリ枯渇を招くことがあります。ご注意ください。

表4.35 mpi_no_establish_communication (バックグラウンド実行において通信の確立を行わないことを指定)

MCAパラメーターの値	内容
1	mpiexecコマンドのバックグラウンド実行において、コミュニケータを共有しないMPIプロセスグループ間での通信の確立を行わない場合に指定します。

MCAパラメーターの値	内容
	MPIプロセスに割り当てられる通信資源(TNI)の数は、そのプロセスに割り当てられたCPU(コア)数によって決まります。VCOORDファイルのCPU数の指定を使用して、CPU数をプロセスによって変えた場合には、各プロセスに割り当てられるTNIの数は、ジョブ全体で最小のCPU数のプロセスのTNIの数に合わせられます。これにより、CPU数の多いプロセスでは本来よりTofu通信の性能が低下することがあります。 本パラメーターを指定することにより、ジョブ全体ではなく、同じmpiexecコマンドから生成されたMPIプロセスの中で、最小のCPU数のプロセスのTNIの数に合わせるようになります。これにより、この通信性能の低下を抑えることができる場合があります。 ただし、本パラメーターを指定したにもかかわらず通信の確立を行った場合は、MPIプロセスがエラーメッセージとともに異常終了します。 mpiexecコマンドのバックグラウンド実行を行わない場合またはVCOORDファイルでCPU数をプロセスによって変えない場合は、本パラメーターを指定するメリットはありません。 VCOORDファイルの詳細については、“4.5 VCOORDファイルの記述形式”をお読みください。
0	mpiexecコマンドのバックグラウンド実行において、コミュニケータを共有しないMPIプロセスグループ間での通信の確立を行う可能性がある場合に指定します。 本パラメーターの省略値は0です。

表4.36 mpi_preconnect_mpi (コネクションを確立するタイミングを指定)

MCAパラメーターの値	内容
1以上の整数値	本処理系では、本パラメーターを指定しない場合、通信相手となる各プロセスと初めて通信を行う時点で、Tofuのコネクションを確立します。 本パラメーターに正の整数値を指定した場合は、MPI_INITルーチンの中で、内部的に通信を行い全プロセス対全プロセスのコネクションを確立します。これにより、MPI_INITルーチンの実行時間は長くなりますが、通信を行うMPIルーチンの実行時間は安定します。 通常は本パラメーターに1を指定してください。2以上の値を指定した場合、MPI_INITルーチンの実行時間が必要以上に長くなります。 計算ノード間での通信が行われない場合は、本パラメーターを指定するメリットはありません。
0	MPI_INITルーチンの中ではTofuのコネクションを確立しません。通信を行うMPIルーチンが呼ばれた時点で、その通信相手となるプロセスとコネクションを確立します。 これにより、MPI_INITルーチンの実行時間は短くなりますが、通信を行うMPIルーチンの実行時間が初回の数回だけ長くなることがあります。 本パラメーターの省略値は0です。

表4.37 mpi_print_stats (MPI統計情報を出力)

MCAパラメーターの値	内容
1	MPI統計情報を標準エラー出力に出力することを指定します。この場合、すべての並列プロセスのMPI統計情報が総括され、MPI_COMM_WORLDに属するランクが0の並列プロセスによって出力されます。 ただし、集団通信のアルゴリズムに関する情報については、MPI_COMM_WORLDに属するランクが0のプロセスが参加した通信に関するものだけが表示されます。 MPI統計情報は、MPI_FINALIZEルーチンが呼ばれたときに出力されます。
2	MPI統計情報を標準エラー出力に出力することを指定します。この場合、並列プロセスごとのMPI統計情報が、それぞれの並列プロセス自身によって出力されます。 MPI統計情報は、MPI_FINALIZEルーチンまたはMPI_ABORTルーチンが呼ばれたときに出力されます。また、本処理系がエラーを検出して並列プロセスを終了させるときにも出力されます。 Process Mapping情報は正常終了時だけ出力されます。

MCAパラメーターの値	内容
	特定の並列プロセスを出力対象としたい場合、MCAパラメーターmpi_print_stats_ranksによって指定できます。MCAパラメーターmpi_print_stats_ranksの詳細については、“表4.38 mpi_print_stats_ranks (MPI統計情報を出力する並列プロセスを特定)”をお読みください。
3	パラメーター値1と同様です。ただし、標準エラー出力に出力させるためにはFJMPI_COLLECTION_PRINTルーチンの指定が必要です。さらに出力内容は、ヘッダー部、section行を含めたボディー部、フッター部に分けて出力されます。 FJMPI_COLLECTION_PRINTルーチンの詳細については、“5.2.1.3 FJMPI_COLLECTION_PRINT”をお読みください。
4	パラメーター値2と同様です。ただし、標準エラー出力に出力させるためにはFJMPI_COLLECTION_PRINTルーチンの指定が必要です。さらに出力内容は、ヘッダー部、section行を含めたボディー部、フッター部に分けて出力されます。 FJMPI_COLLECTION_PRINTルーチンの詳細については、“5.2.1.3 FJMPI_COLLECTION_PRINT”をお読みください。
0	MPI統計情報を出力しないことを指定します。 MPI統計情報の詳細については、“6.16 MPI統計情報”をお読みください。 本パラメーターに0から4の整数値以外の値を指定した場合、0が指定されたものとみなされます。 本パラメーターの省略値は0です。

表4.38 mpi_print_stats_ranks (MPI統計情報を出力する並列プロセスを特定)

MCAパラメーターの値	内容
0以上の整数値	MPI統計情報を出力する並列プロセスのランクを特定します。本MCAパラメーターは、MCAパラメーターmpi_print_statsに2または4を指定した場合にだけ有効となります。 MPI_COMM_WORLDに属するランクを指定してください。 コンマ(,)でランクを区切ることで、複数のランクを指定できます。 本パラメーターに存在しないランクを指定した場合、そのランクの指定は無視されます。 MCAパラメーターmpi_print_statsについては、“表4.37 mpi_print_stats (MPI統計情報を出力)”をお読みください。
-1	すべての並列プロセスのMPI統計情報を出力することを指定します。本MCAパラメーターは、MCAパラメーターmpi_print_statsに2または4を指定した場合にだけ有効となります。 MCAパラメーターmpi_print_statsについては、“表4.37 mpi_print_stats (MPI統計情報を出力)”をお読みください。 本パラメーターに-1より小さい値を指定した場合、-1が指定されたものとみなされます。 本パラメーターの省略値は-1です。

表4.39 opal_abort_delay (異常を検出したときのプログラム終了を遅延)

MCAパラメーターの値	内容
正の整数値	MPIプログラムがMPI_Abortルーチンを呼んだとき、またはMPIライブラリが異常を検知したとき、指定した時間(秒)だけプログラムの終了が遅延されます。
0	MPIプログラムがMPI_Abortルーチンを呼んだとき、またはMPIライブラリが異常を検知したとき、直ちにプログラムを終了します。 本パラメーターの省略値は0です。

表4.40 opal_abort_print_stack (スタックのトレース情報を出力)

MCAパラメーターの値	内容
1	MPIプログラムがMPI_ABORTルーチンを呼んだとき、またはMPIライブラリが実行環境や通信などの異常を検知してMPIプログラムの実行を終了させるときに、エラーメッセージに続けてスタックのトレース情報を出力します。 異常終了の原因を特定するのに役に立つ場合があります。 本パラメーターの省略値は1です。
0	スタックのトレース情報を出力しません。

表4.41 opal_mt_memcpy (MPIライブラリ内で行われる特定のメモリコピー処理をスレッド並列化)

MCAパラメーターの値	内容
1	MPIライブラリ内部で行われる特定のメモリコピー処理を、条件次第でスレッド並列化します。 0未満または2以上の値が指定された場合は、1が指定されたものとみなされます。 実際にスレッド並列化するか否かは、処理を行うときの条件に応じてMPIライブラリが決定します。 スレッド並列化する場合は、ユーザーによって指定されたスレッド数で処理が並列化されます。 本機能が有効の場合にスレッド並列化の対象となるMPIルーチンは以下の通りです。 ・ MPI_PACK、MPI_UNPACKルーチン ・ 1対1通信ルーチン ・ 集団通信と片側通信のルーチンのうち、MPIライブラリ内で1対1通信が行われるもの 詳細については、“6.3 MPIライブラリ内メモリコピー処理のスレッド並列化”をお読みください。
0	MPIルーチンを呼んだスレッドだけでMPIライブラリ内部の処理を行います(アシスタントコアが使われる場合を除く)。アシスタントコアの使用については、“6.2 アシスタントコアを用いた非同期通信の促進”をお読みください。 本パラメーターの省略値は0です。

表4.42 opal_progress_thread_mode (MPI非同期処理進行スレッドの動作モードを指定)

MCAパラメーターの値	内容
1	アシスタントコアを用いた非同期通信の促進において、指定区間(MPI呼び出しなし)モードを使用することを指定します。 詳細については、“6.2 アシスタントコアを用いた非同期通信の促進”をお読みください。
2	アシスタントコアを用いた非同期通信の促進において、指定区間(MPI呼び出しあり)モードを使用することを指定します。
3	アシスタントコアを用いた非同期通信の促進において、自動区間モードを使用することを指定します。
0	アシスタントコアを用いた非同期通信の促進の機能を使用しないことを指定します。 本パラメーターの省略値は0です。0から3以外の値を指定した場合、[mpi::mca-var::invalid-value-enum]および[mpi::opal-runtime::opal_init:startup:internal-failure]から始まるエラーメッセージが出力され、mpiexecコマンドが異常終了します。

表4.43 opal_progress_timeout (通信待ちの打切り時間を指定)

MCAパラメーターの値	内容
正の整数値	通信タイムアウト設定機能における、通信待ちの打切り時間(秒)を指定します。 MPI通信において、通信待ち時間が本パラメーターに指定した時間(秒)を超えた場合、標準エラー出力にメッセージおよびスタックのトレース情報を出力してMPIプログラムの実行を終了します。 詳細については、“6.17.1 通信タイムアウト設定”をお読みください。

MCAパラメーターの値	内容
0	通信待ちの打切りを行わないことを指定します。 本パラメーターの省略値は0です。

表4.44 plm_ple_cpu_affinity (MPIプロセスのCPU affinityを指定)

MCAパラメーターの値	内容
1	コンパイラの自動並列もOpenMP機能も使用していない場合に、1MPIプロセスあたりにバインドされるCPU(コア)数を最適化することを指定します。 バインドされるCPU(コア)は、VCOORDファイルでの numanode_assign_policy の指定、またはMCAパラメーター plm_ple_numanode_assign_policy の指定によって決定されます。 本パラメーターの省略値は1です。
0	コンパイラの自動並列もOpenMP機能も使用していない場合、OSのスケジューリングに従ってMPIプロセスをCPU(コア)に配置することを指定します。 本パラメーター値を指定した場合、まれにMPIプロセスが同じCPUに配置されることがあります。本パラメーター値は、利用者がsched_setaffinity関数などを利用して使用するCPUを決定する場合に指定してください。 コンパイラの自動並列またはOpenMP機能を使用している場合、コンパイラが並列スレッドのCPUバインドを行うため、本パラメーターの指定は無効となります。コンパイラによるCPU(コア)バインドの詳細は、各コンパイラのマニュアルをお読みください。 本パラメーターに0、1以外の値を指定した場合、動作は保証されません。

備考:MCAパラメーター名に使われている文字列plmおよびpleの“l”(エル)はどちらも英小文字です。

表4.45 plm_ple_memory_allocation_policy (NUMAメモリポリシーを指定する)

MCAパラメーターの値	内容
localalloc、 interleave_local、 interleave_nonlocal、 interleave_all、 bind_local、 bind_nonlocal、 bind_all、 prefer_local、または prefer_nonlocal	MPIプロセスのNUMAメモリポリシーを指定します。指定可能な値は以下の通りです。詳細については、“表4.53 NUMAメモリ割り当てポリシーの設定値”をお読みください。 ・ localalloc プロセスが動作中のCPU(コア)の属するNUMAノードからメモリを割り当てます。 ・ interleave_local プロセスの「ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てを行います。 ・ interleave_nonlocal プロセスの「非ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てを行います。 ・ interleave_all プロセスの「全ノード集合」内の各NUMAノードから交互にメモリを取得します。 ・ bind_local プロセスの「ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行います。 ・ bind_nonlocal プロセスの「非ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行います。 ・ bind_all プロセスの「全ノード集合」のNUMAノードにバインドします。 ・ prefer_local プロセスの「ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行います。

MCAパラメーターの値	内容
	・ prefer_nonlocal プロセスの「非ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行います。 NUMAノードの概要に関しては、ジョブ運用ソフトウェアのマニュアルをお読みください。 NUMAメモリポリシーの指定は、以下の優先順位で決定します。 1. VCOORDファイルでの指定 2. 本パラメーターでの指定 3. (VCOORDファイルでmemory_allocation_policyの指定がなく、かつ本パラメーターが省略された場合)localalloc

備考:MCAパラメーター名に使われている文字列plmおよびpleの“l”(エル)はどちらも英小文字です。

表4.46 plm_ple_numanode_assign_policy (NUMAノードへのCPU(コア)割り当てポリシーを指定する)

MCAパラメーターの値	内容
simplex、 share_cyclic、または share_band	MPIプロセスのNUMAノードへのCPU(コア)割り当てポリシーを指定します。指定可能な値は以下の通りです。詳細については、“表4.54 CPU(コア)割り当てポリシーの設定値”をお読みください。 ・ simplex NUMAノードを占有するように割り当てます。 ・ share_cyclic NUMAノードを他のプロセスと共有するように割り当てます。異なるNUMAノードに順番にプロセスを割り当てます。 ・ share_band NUMAノードを他のプロセスと共有するように割り当てます。同一NUMAノードに連続してプロセスを割り当てます。 NUMAノードの概要に関しては、ジョブ運用ソフトウェアのマニュアルをお読みください。 NUMAノードへのCPU(コア)割り当てポリシーの指定は、以下の優先順位で決定します。 1. VCOORDファイルでの指定 2. 本パラメーターでの指定 3. (VCOORDファイルでnumanode_assign_policyの指定がなく、かつ本パラメーターが省略された場合)share_cyclic

備考:MCAパラメーター名に使われている文字列plmおよびpleの“l”(エル)はどちらも英小文字です。

表4.47 pml_ob1_use_stride_rdma (Stride RDMA通信を使用)

MCAパラメーターの値	内容
1	Stride RDMA通信を使用することを指定します。詳細については、“6.6 Stride RDMA通信”をお読みください。 本パラメーターの省略値は1です。
0	Stride RDMA通信を使用しないことを指定します。

備考:MCAパラメーター名に使われている文字列pmlの“l”(エル)は英小文字であり、文字列ob1の“1”は数字です。

4.3 環境変数

本処理系では、環境変数を使用することによって、MPIプログラムの動作を制御できます。

動的に生成された並列プロセスは、その生成を行った元の並列プロセスの生成時点でのルートプロセスの環境変数を引き継ぎます。また、元の並列プロセスのプログラム内では“OMPI_MCA”で始まる環境変数を設定しないでください。

PLE_MPI_STD_EMPTYFILE

並列プロセスの標準出力/標準エラー出力への出力がない場合に、空ファイルを作成するかどうかを指定します。

on : 作成する (デフォルト)

off : 作成しない

これは、--stdオプションなどで指定するmpiexec単位の出力ファイルおよび--std-procオプションなどで指定するプロセス単位の出力ファイルのどちらに対しても適用されます。

on、off以外の値が指定された場合は、ジョブ運用ソフトウェアのジョブACL機能(pjaclコマンドの“mpiexec-std-emptyfile”)の設定に従います。ジョブACL機能の設定については、ジョブ運用ソフトウェアのマニュアルをお読みください。

UTOFU_SWAP_PROTECT

計算ノード上では、獲得済みのメモリ領域がメモリ不足等の理由でスワップアウトされることがあります。Tofuインターコネクトでの通信に使用するメモリ領域がスワップアウトされると、そのメモリ領域を用いた通信でエラーが発生することがあります。この場合、MPIプロセスは以下のいずれかのエラーメッセージを出力して異常終了します。

- “[mpi::common-tofu::tcq-error] Communication error is reported by Tofu TCQ.”を含むメッセージ
- “[mpi::common-tofu::mrq-error] Communication error is reported by Tofu MRQ.”を含むメッセージ
- “[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ.”を含むメッセージ

環境変数UTOFU_SWAP_PROTECTの指定により、通信用のメモリ領域をスワップアウトの対象から除外するかどうかを選択できます。

整数値の0または1を指定でき、省略値は0です。0と1以外の値が指定された場合は、1が指定されたものと見なします。

本環境変数に1を指定した場合、MPIライブラリ内部でmlockシステムコールを呼ぶことで、Tofuインターコネクトでの通信に使用するメモリ領域がスワップアウトの対象にならないことを保証します。スワップアウトの対象から除外できるメモリ領域のサイズは、ソフト資源制限RLIMIT_MEMLOCKに従います。ジョブ実行時のRLIMIT_MEMLOCKの確認方法や変更方法については、ジョブ運用ソフトウェアのマニュアルをお読みください。

本環境変数に0を指定した場合、Tofuインターコネクトでの通信に使用するメモリ領域が、スワップアウトの対象になる可能性があります。

注意

- 通信用メモリ領域をスワップアウトの対象から除外すると、通信性能が低下することがあります。
- RLIMIT_MEMLOCKの値が、ジョブで使用される通信用メモリ領域の合計サイズよりも小さい場合、本環境変数に1を指定してもスワップアウトによる通信エラーを回避できない場合があります。

“OMPI_”で始まる環境変数

MPIライブラリの実行時の動きを制御します。

本処理系で提供する環境変数は、MCAパラメーターが派生したものです。MCAパラメーター名の先頭に“OMPI_MCA_”を付加することで環境変数として使用できます。これは、すべてのMCAパラメーターに対して可能です。MCAパラメーターの詳細については、“4.2 MCAパラメーター”をお読みください。

MCAパラメーターを環境変数で設定する場合の例を、以下に示します。

例

MCAパラメーターの指定例

```
-mca  mca_base_param_file_prefix  MCAFILE
```

AMCAパラメーターファイルを指定するMCAパラメーター名“mca_base_param_file_prefix”の先頭に“OMPI_MCA_”を付加した名前が環境変数名として使用できます。

上記MCAパラメーターを環境変数として使用した例

```
OMPI_MCA_mca_base_param_file_prefix=MCAFILE
```

4.4 mpiexecコマンドの復帰値

mpiexecコマンドの復帰値は、基本的に利用者がMPIプログラムに指定した値または言語処理系が設定した値となります。言語処理系の設定する値に関しては、言語処理系の使用手引書(Fortran使用手引書、C言語使用手引書、C++言語使用手引書)をお読みください。

- MPIプログラムのプロセスが複数ある場合、内部的に最初に認識できたプロセスの復帰値がmpiexecコマンドの復帰値となります。
- MPIプログラムが異常終了した場合、異常終了したMPIプログラムの復帰値となります。ただし、MPIプログラムが何らかのシグナルを受信して異常終了した場合は、受信したシグナル番号と0x80の論理和の結果値になります。
- 動的に生成された並列プロセスが異常終了した場合、異常終了した動的プロセスの復帰値となります。
- 本処理系が異常終了した場合、本処理系によって指定される復帰値となります。

ただし、本処理系では、下表に示す復帰値を予約しています。本処理系で予約している復帰値を優先しますので、MPIプログラムに復帰値を設定する場合、利用者は、これらの予約値を避けるようにしてください。

表4.48 本処理系で予約済のmpiexecコマンドの復帰値

mpiexecコマンドの復帰値	意味
1	mpiexecコマンドのオプションの設定誤り、本処理系内の内部矛盾、またはMPIルーチン内で致命的なエラーが発生したことを表します。
2～92	MPIプログラムのMPIルーチン内で、MPI規格で規定されるエラークラスと対応する場合のエラーコードが復帰値になります。 MPI規格で規定されるエラークラスについては、“付録A エラークラス一覧”をご覧ください。
シグナル番号と0x80の論理和	mpiexecコマンドまたはMPIプログラムがシグナルを受信して異常終了した場合の復帰値を表します。 MPIプログラムが異常終了時に受信したシグナル番号と0x80の論理和の結果値になります。 この挙動は、一般的なUNIXシェルスクリプトにおける、コマンドがシグナルを受信して終了した場合の終了ステータスの取り扱いに準じたものです。
255	ジョブ運用ソフトウェアの並列実行環境側が異常終了した場合の復帰値を表します。

4.5 VCOORDファイルの記述形式

VCOORDファイルでは、以下の形式でプロセスに割り当てる座標とCPU(コア)数を指定します。

表4.49 座標の指定方法

座標の種別	記述形式
1次元座標	(X)
2次元座標	(X,Y)
3次元座標	(X,Y,Z)

表4.50 CPU(コア)数の指定

指定する内容	記述内容
CPU(コア)数	core=N

表4.51 NUMAメモリ割り当てポリシーの指定

指定する内容	記述内容
NUMAメモリ割り当て方法	memory_allocation_policy=設定値

設定値には、“表4.45 plm_ple_memory_allocation_policy (NUMAメモリポリシーを指定する)”の内のどれかの値が指定可能です。

MCAパラメーター plm_ple_memory_allocation_policy の指定が存在し、かつ、VCOORDファイルにて本指定が存在した場合、本指定が優先されます。そのどちらの指定もない場合、MCAパラメーター plm_ple_memory_allocation_policy に“localalloc”を指定した場合と同等の動作となります。

NUMAノードの概要に関しては、ジョブ運用ソフトウェアのマニュアルをお読みください。

表4.52 NUMAノードへのCPUコア割り当てポリシーを指定

指定する内容	記述内容
CPUコア割り当て方法	numanode_assign_policy=設定値

設定値には、“表4.46 plm_ple_numanode_assign_policy (NUMAノードへのCPU(コア)割り当てポリシーを指定する)”の内のどれかの値が指定可能です。

MCAパラメーター plm_ple_numanode_assign_policy の指定が存在し、かつ、VCOORDファイルにて本指定が存在した場合、本指定が優先されます。そのどちらの指定もない場合、MCAパラメーター plm_ple_numanode_assign_policy に“share_cyclic”を指定した場合と同等の動作となります。

NUMAノードの概要に関しては、ジョブ運用ソフトウェアのマニュアルをお読みください。

以下に、VCOORDファイルの記述例を示します。

例

形式1. 論理座標とCPU(コア)数を指定する

本形式では、各プロセスが生成される座標と、プロセスへ割り当てられるCPU(コア)数の両方を指定します。

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

形式2. 論理座標だけ指定する

本指定では、各プロセスを割り当てる座標だけを指定します。各プロセスに割り当てられるCPU(コア)数は、MCA パラメーター plm_ple_cpu_affinityに基づいて運用ソフトウェアが決定します。

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

形式3. CPU(コア)数だけ指定する

本指定では、各プロセスに割り当てるCPU(コア)数だけを指定します。各プロセスに割り当てる座標は、運用ソフトウェアが決定します。

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

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

形式4. NUMAメモリ割り当てポリシーを指定する

本指定では、形式1～3のどれかの指定に加え、NUMAメモリの割り当てポリシーを指定します。

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

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

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

形式5. NUMAノードへのCPUコア割り当てポリシーを指定する

本指定では、形式1～3のどれかの指定に加え、NUMAノードへの CPU コア割り当てポリシーを指定します。

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

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

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

論理座標が2次元、3次元の場合も、1次元の場合と同様の形式で指定可能です。また、同一の座標を複数行に記述することで、同一座標に複数のプロセスを生成することが可能です。

core、memory_allocation_policy、numanode_assign_policyの2つ以上を指定する場合、順番に制約はありません。

以下の場合は、ジョブ運用ソフトウェアでエラーとなります。

- 同一座標で生成するプロセス数が、pjsubコマンドの --mpi "proc=" オプションにより決定されるノードあたりのプロセス数を超過した場合
- 同一座標で生成するプロセスに割り当てるCPU(コア)数の合計値が、計算ノードに搭載されているCPU(コア)数を超過した場合
- 座標指定のある行と、座標指定が省略された行が、1つのVCOORDファイル内に混在している場合
- CPU(コア)数指定のある行と、CPU(コア)数指定が省略された行が、1つのVCOORDファイル内に混在している場合
- 行の先頭以外に座標が記述されている場合
- mpiexecコマンドで生成するよう指定したプロセス数(指定省略時は、pjsubコマンドで指定したプロセス数)が、VCOORDファイルの行数よりも大きい場合
- NUMAノードへの CPU コア割り当てポリシーにsimplexが指定されたプロセスが1つ以上存在する場合、CPUコア数が不足してプロセスへのCPUコア割り当てに失敗する可能性があります。

4.6 同一ノード上で複数のMPIプログラムを実行

本処理系では、MPMDモデルを使用せずに、同一ノード上で複数のMPIプログラムを実行できます。詳細な実行方法については、ジョブ運用ソフトウェアのマニュアルをお読みください。

ここでは、同一ノード上で複数のMPIプログラムを実行する際の注意事項を説明します。

本節では、mpiexecコマンド、MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンをまとめてプロセス生成手続きと呼びます。

- 後から実行するプロセス生成手続きについては、VCOORDファイルによる座標の指定を行ってください。VCOORDファイルでCPU(コア)数だけが指定されていた場合、各プロセスに割り当てる座標は、ジョブ運用ソフトウェアが決定します。
- VCOORDファイルにCPU(コア)数の指定が無い場合、1プロセスに割り当てるCPU(コア)数は、

  “1ノードあたりのCPU(コア)数” ÷ “pjsubコマンドの --mpi max-proc-per-node オプションで指定した値”

  で求められる値となります。
- pjsubコマンドの --mpi max-proc-per-node オプションで指定した値を超えて、1ノードに対してプロセスを作成しようとした場合、対象のプロセス生成手続きはエラーとなります。その場合、mpiexecコマンドは復帰値1で復帰し、MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンは、エラークラスMPI_ERR_SPAWNをエラーコードとして返します。
- pjsubコマンドの --mpi max-proc-per-node オプションの指定をせずに、同じノードに複数のプロセス生成手続きを実行しようとした場合、1ノードあたりの作成可能プロセス数の上限は、pjsubコマンドの --mpi proc オプションにより決定される値となります。
- MCAパラメーターmpi_no_establish_communicationが指定された場合、同一ノード上で複数のMPIプログラムを実行することはできません。MCAパラメーターmpi_no_establish_communicationの指定とpjsubコマンドの --mpi max-proc-per-node オプションの指定は排他関係となります。両方指定された場合、MCAパラメーターmpi_no_establish_communicationの指定が有効となります。

4.7 NUMA構成における設定

本システムにおける計算ノードは、NUMA構成となっております。NUMA構成におけるメモリアクセス速度に起因したジョブの実行性能のブレ/劣化を低減するためのMCAパラメーターが用意されています。

4.7.1 NUMAメモリ割り当てポリシーの設定値

MCAパラメーターplm_ple_memory_allocation_policyを指定することで、メモリポリシーを設定できます。指定可能な値を下表に示します。MCAパラメーターplm_ple_memory_allocation_policyの詳細は、“表4.45 plm_ple_memory_allocation_policy (NUMAメモリポリシーを指定する)”をお読みください。

本表の説明において、NUMAノード集合は以下のように定義しています。

- 全ノード集合 (all)
  計算ノードに属するすべてのNUMAノードを含む集合
- ローカルノード集合 (local)
  プロセスが割り当たった各CPU(コア)の属する NUMAノードの和集合
- 非ローカルノード集合 (nonlocal)
  「全ノード集合」から、「ローカルノード集合」の要素を取り除いたもの

表4.53 NUMAメモリ割り当てポリシーの設定値

設定値	意味	備考
localalloc	プロセスが動作中のCPU(コア)の属するNUMAノードからメモリを割り当てます。CPU(コア)の属するNUMAノードのメモリに空きが無い場合、CPU(コア)からのアクセスコストが小さい順にメモリ割り当てを行います。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_DEFAULT, NULL, ..)
interleave_local	プロセスの「ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てを行います。割り当てを行おうとしたNUMAノードのメモリに空きが無い場合、「ローカルノード集合」内の次のNUMAノードからメモリ割り当てを行います。ローカルノード集合に属する全NUMAノードのメモリに空きが無い場合の動作はOSの仕様に依存します。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_INTERLEAVE, local, ..)
interleave_nonlocal	プロセスの「非ローカルノード集合」内の各NUMAノードから交互にメモリ割り当てを行います。割り当てを行おうとしたNUMAノードのメモリに空きが無い場合、「非ローカルノード集合」内の次のNUMAノードからメモリ	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_INTERLEAVE, nonlocal, ..)

設定値	意味	備考
	割り当てを行います。非ローカルノード集合に属する全NUMAノードのメモリに空きが無い場合の動作はOSの仕様に依存します。	非ローカルノード集合が空の場合、set_mempolicy(2) の呼び出しに失敗します。その場合、ジョブの標準エラー出力に警告メッセージ PLE 0601 が出力され、処理は継続されます。このときの並列プロセスのNUMAメモリ割り当てポリシーは、以下のシステムコールを呼び出した場合と等価になります。 set_mempolicy (MPOL_DEFAULT, NULL, ..)
interleave_all	プロセスの「全ノード集合」内の各NUMAノードから交互にメモリを取得します。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_INTERLEAVE, all, ..)
bind_local	プロセスの「ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行います。「ローカルノード集合」に属するNUMAノードのメモリに空きが無い場合、割り当てに失敗します。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_BIND, local, ..)
bind_nonlocal	プロセスの「非ローカルノード集合」に属する各NUMAノードで、ノードIDの若い順にメモリ割り当てを行います。「非ローカルノード集合」に属するNUMAノードのメモリに空きが無い場合、割り当てに失敗します。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_BIND, nonlocal, ..) 非ローカルノード集合が空の場合、set_mempolicy(2) の呼び出しに失敗します。その場合、ジョブの標準エラー出力に警告メッセージ PLE 0601 が出力され、処理は継続されます。このときの並列プロセスのNUMAメモリ割り当てポリシーは、以下のシステムコールを呼び出した場合と等価になります。 set_mempolicy (MPOL_DEFAULT, NULL, ..)
bind_all	プロセスの「全ノード集合」のNUMAノードにバインドします。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_BIND, all, ..)
prefer_local	プロセスの「ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行います。「優先ノード」のメモリに空きが無い場合、動作中のCPU(コア)からのアクセスコストが小さい順に割り当てを行います。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_PREFERRED, local, ..)
prefer_nonlocal	プロセスの「非ローカルノード集合」のうち、NUMAノードIDが最も若いものを「優先ノード」とし、「優先ノード」からメモリ割り当てを行います。「優先ノード」のメモリに空きが無い場合、動作中のCPU(コア)からのアクセスコストが小さい順に割り当てを行います。	並列プロセスから以下のシステムコールを呼び出した場合と等価。 set_mempolicy (MPOL_PREFERRED, nonlocal, ..) 非ローカルノード集合が空の場合、“nonlocal”の指定は無視され、“localalloc”を指定した場合と等価となります。

4.7.2 CPU(コア)割り当てポリシーの設定値

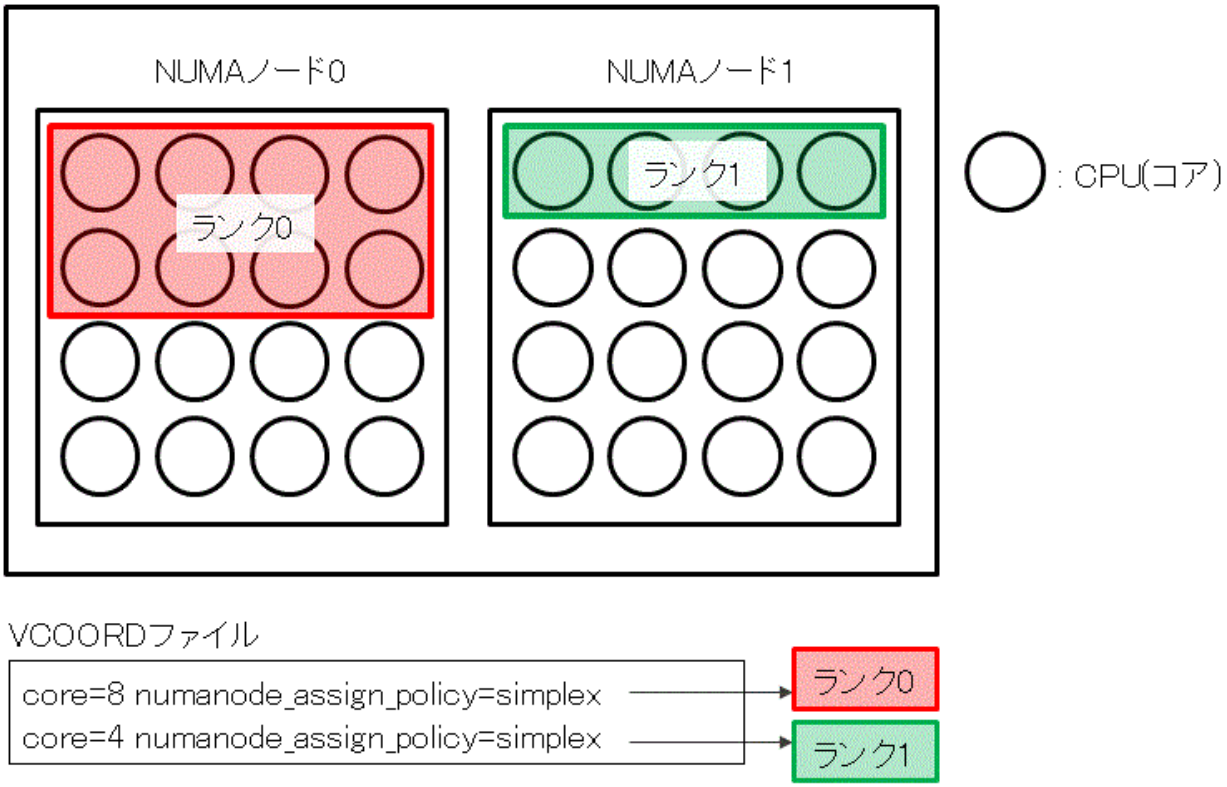
MCAパラメーターplm_ple_numanode_assign_policyを指定することで、NUMA ノードを意識したプロセスへのCPU(コア)割り当てを行います。指定可能な値を下表に示します。MCAパラメーターplm_ple_numanode_assign_policyの詳細は、
“表4.46 plm_ple_numanode_assign_policy (NUMAノードへのCPU(コア)割り当てポリシーを指定する)”をお読みください。

表4.54 CPU(コア)割り当てポリシーの設定値

設定値	意味
simplex	NUMAノードを占有するように割り当てます。
share_cyclic	NUMAノードを他のプロセスと共有するように割り当てます。 異なるNUMAノードに順番にプロセスを割り当てます。
share_band	NUMAノードを他のプロセスと共有するように割り当てます。 同一NUMAノードに連続してプロセスを割り当てます。

以下に割り当てイメージを示します。

図4.1 simplexの場合のプロセス割り当て例

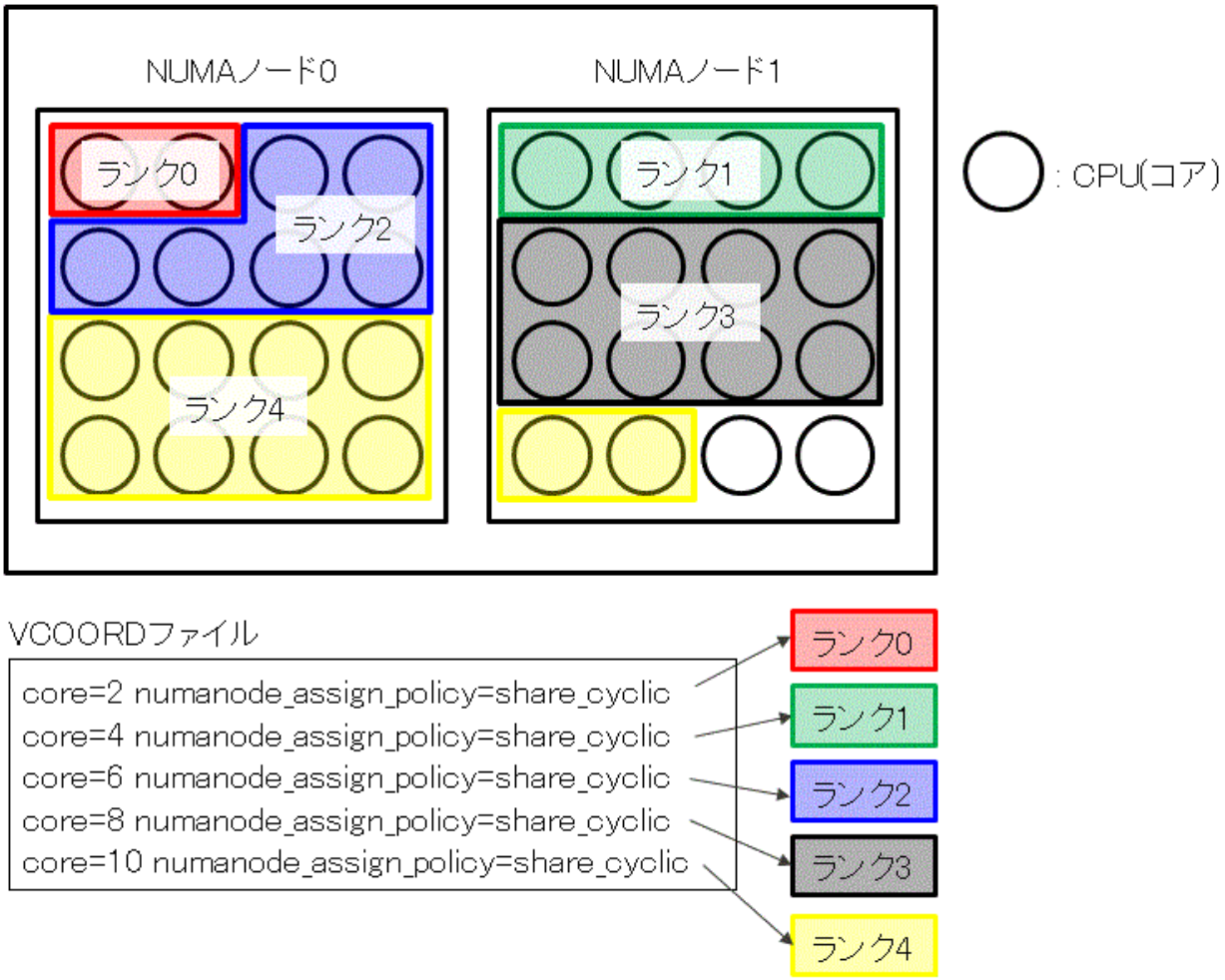


“図4.1 simplexの場合のプロセス割り当て例”では、ランク0(赤)およびランク1(緑)は、順番に異なるNUMAノードに割り当てられます。1つのNUMAノードが複数のランクで共有されることはありません。

“図4.1 simplexの場合のプロセス割り当て例”では、各プロセスに割り当てるCPU(コア)数をVCOORDファイルを用いて下記のように指定しています。

- ランク0がCPU(コア)8個使用
- ランク1がCPU(コア)4個使用

図4.2 share_cyclicの場合のプロセス割り当て例

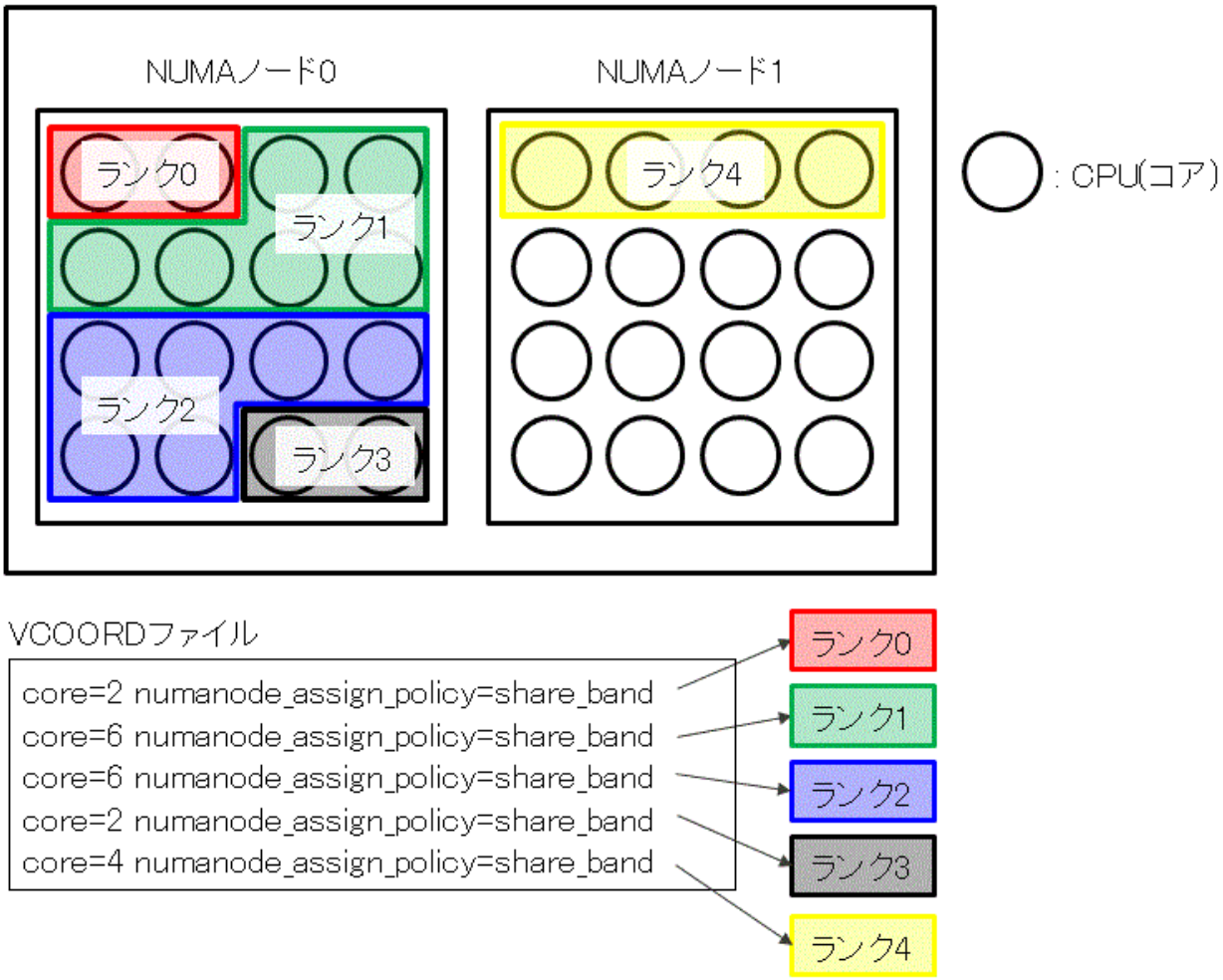


“図4.2 share_cyclicの場合のプロセス割り当て例”では、ランク0(赤)、ランク1(緑)、ランク2(青)、ランク3(灰色)、およびランク4(黄)は、順番に異なるNUMAノードにラウンドロビン方式で割り当てられます。ランクがNUMAノード内の空いているCPU(コア)に割り当てられるため、1つのNUMAノードが複数のランクで共有されることがあります。

“図4.2 share_cyclicの場合のプロセス割り当て例”では、各プロセスに割り当てるCPU(コア)数をVCOORDファイルを用いて下記のように指定しています。

- ランク0がCPU(コア)2個使用
- ランク1がCPU(コア)4個使用
- ランク2がCPU(コア)6個使用
- ランク3がCPU(コア)8個使用
- ランク4がCPU(コア)10個使用

図4.3 share_bandの場合のプロセス割り当て例



“図4.3 share_bandの場合のプロセス割り当て例”では、ランク0(赤)、ランク1(緑)、ランク2(青)、ランク3(灰色)、およびランク4(黄)は、順番になるべく同じNUMAノードに詰め込むように割り当てられます。ランクがNUMAノード内の空いているCPU(コア)に割り当てられるため、1つのNUMAノードが複数のランクで共有されることがあります。

“図4.3 share_bandの場合のプロセス割り当て例”では、各プロセスに割り当てるCPU(コア)数をVCOORDファイルを用いて下記のように指定しています。

- ランク0がCPU(コア)2個使用
- ランク1がCPU(コア)6個使用
- ランク2がCPU(コア)6個使用
- ランク3がCPU(コア)2個使用
- ランク4がCPU(コア)4個使用

第5章 拡張インターフェース

本章では、本処理系が提供する、MPI規格からの拡張インターフェースについて記述しています。

本処理系では、次のような拡張インターフェースを提供しています。

- ランク問合せインターフェース
- 区間指定MPI統計情報インターフェース
- 拡張持続的通信要求インターフェース
- 区間指定MPI非同期通信促進インターフェース
- 追加の定義済みデータ型

参考

- すべての拡張インターフェースは、C言語とFortranに対応しています。
  C++プログラムでは、C言語インターフェースを利用します。
  Fortranは、MPIの「USE mpi_f08」と「USE mpi」に対応した、「USE mpi_f08_ext」と「USE mpi_ext」を用意しています。どちらかを使用してください。「USE mpi_ext」の代わりに「INCLUDE 'mpif-ext.h'」を使用することもできます。
- ジョブ種別がノード共有ジョブの場合、ランク問合せインターフェースと拡張持続的通信要求インターフェースは使用できません。
  ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

5.1 ランク問合せインターフェース

本処理系では、1次元から3次元のトーラス構成の論理的なノード空間において、MPIプログラムを実行できます。この論理的なノード空間には、ジョブ運用ソフトウェアによって論理的な座標が割り振られています。本処理系では、この座標を論理座標または単に座標と呼びます。さらに、この論理的なノード空間内の適切な場所に、トーラス構成のプロセス形状をもつMPIプログラムが配置されます。

MPIプログラムの並列プロセスの各ランクがトーラス構成のプロセス形状の中でどの位置(座標)に配置されているかについて、MPIプログラムの中から知ることができれば、便利な場合があります。例えば、本処理系では、トーラス構成の形状の座標で隣接する2つの並列プロセスは、通常、物理的に1ホップの距離になるように配置されます。すなわち、隣接する2つの並列プロセスのランクを知ることができれば、通信性能を意識したプログラミングも可能になります。

本処理系において提供するランク問合せインターフェースについて、具体的なルーチンの一覧を下表に示します。

表5.1 ランク問合せインターフェースのルーチン一覧

ルーチン名	機能概要
FJMPI_TOPOLOGY_GET_DIMENSION	MPI_COMM_WORLDに与えられる次元数を取得する
FJMPI_TOPOLOGY_GET_SHAPE	MPI_COMM_WORLDに与えられるプロセス形状を取得する
FJMPI_TOPOLOGY_GET_COORDS	ランクから座標を取得する
FJMPI_TOPOLOGY_GET_RANKS	座標からランクを取得する
FJMPI_TOPOLOGY_CART_REORDER	カルテシアン構造をもつコミュニケータのランク付け判別値を取得する

5.1.1 次元数・形状の問合せ

5.1.1.1 FJMPI_TOPOLOGY_GET_DIMENSION

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

MPI_INITルーチンを実行したときに内部的に生成されるMPI_COMM_WORLDに属するMPIプロセスが配置されたプロセス形状の次元数を返します。

型	変数	説明	IN/OUT
int*	size	MPI_COMM_WORLDのもつプロセス形状の次元数	OUT

<復帰値>

正常時	FJMPI_SUCCESS	―
異常時	FJMPI_ERR_TOPOLOGY_INVALID_COMM	動的生成されたMPIプロセスから本ルーチンが呼び出された場合
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	ジョブ種別がノード共有ジョブの場合 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

以下のどちらかの場合、動作は不定となり保証されません。

― MPI_INITルーチンの実行前に本ルーチンが呼び出された

― MPI_FINALIZEルーチンの実行後に本ルーチンが呼び出された

5.1.1.2 FJMPI_TOPOLOGY_GET_SHAPE

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

MPI_INITルーチンを実行したときに内部的に生成されるMPI_COMM_WORLDに与えられるMPI並列プロセスの形状XYZを返します。

型	変数	説明	IN/OUT
int*	x	MPI_COMM_WORLDに与えられるプロセス形状のX軸のサイズ	OUT
int*	y	MPI_COMM_WORLDに与えられるプロセス形状のY軸のサイズ	OUT
int*	z	MPI_COMM_WORLDに与えられるプロセス形状のZ軸のサイズ	OUT

<復帰値>

正常時	FJMPI_SUCCESS	―
異常時	FJMPI_ERR_TOPOLOGY_INVALID_COMM	動的生成されたMPIプロセスから本ルーチンが呼び出された場合
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	ジョブ種別がノード共有ジョブの場合 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

プロセス形状が1次元の場合、Y軸およびZ軸の値は0になります。2次元の場合、Z軸の値は0になります。

以下のどちらかの場合、動作は不定となり保証されません。

― MPI_INITルーチンの実行前に本ルーチンが呼び出された

― MPI_FINALIZEルーチンの実行後に本ルーチンが呼び出された

5.1.2 座標の問合せ

5.1.2.1 FJMPI_TOPOLOGY_GET_COORDS

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

指定されたコミュニケータとそのコミュニケータにおけるプロセスのランクに対応する論理座標またはTofu座標を取得します。

― 論理座標を取得する場合

view = FJMPI_LOGICAL、maxdims = 1～3 を指定します。指定したcomm、rankに対応するノードの論理X座標、論理Y座標、論理Z座標の値がそれぞれcoords[0]、coords[1]、coords[2]に格納されます。

ー Tofu座標(実際に割り当てられた座標)を取得する場合

view = FJMPI_TOFU_SYS、maxdims = 6を指定します。指定したcomm、rankに対応するノードのTofu座標のX座標、Y座標、Z座標、A座標、B座標、C座標の値がそれぞれcoords[0]、coords[1]、coords[2]、coords[3]、coords[4]、coords[5]に格納されます。

ー Tofu座標(引数commのランク0を基準とした相対座標、commがグループ間コミュニケータの場合リモートグループのランク0を基準とした相対座標)を取得する場合

view = FJMPI_TOFU_REL、maxdims = 6を指定します。指定したcomm、rankに対応するノードのTofu座標(引数commのランク0を基準とした相対座標)のX座標、Y座標、Z座標、A座標、B座標、C座標の値がそれぞれcoords[0]、coords[1]、coords[2]、coords[3]、coords[4]、coords[5]に格納されます。

型	変数	説明	IN/OUT
MPI_Comm	comm	コミュニケータを指定	IN
int	rank	コミュニケータ内のランクを指定 引数commにグループ間コミュニケータを指定した場合、リモートグループのランクを指定	IN
int	view	論理座標かTofu座標かを表すマクロを指定 FJMPI_LOGICAL: 論理座標 FJMPI_TOFU_SYS: Tofu座標(実際に割り当てられた座標) FJMPI_TOFU_REL: Tofu座標(引数commのランク0を基準とした相対座標、commがグループ間コミュニケータの場合リモートグループのランク0を基準とした相対座標)	IN
int	maxdims	取得する座標の次元数を指定 viewがFJMPI_LOGICALの場合: 1～3を指定 viewがFJMPI_LOGICAL以外の場合: 1～6を指定	IN
int[]	coords	コミュニケータとランクに対応する座標の配列	OUT

<復帰値>

正常時	FJMPI_SUCCESS	ー
異常時	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	ジョブ種別がノード共有ジョブの場合 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

引数 coords に指定する配列の要素数は、引数 maxdims 以上でなければなりません。

引数 rank に指定するランクは、引数 comm に指定するコミュニケータ内のプロセスが持つランクの範囲内でなければなりません。

引数viewにFJMPI_LOGICALを指定した場合、引数maxdimsの値とジョブ形状が異なっていても構いません。その場合は、2つのうち小さい方の値と同じ次元数分の座標が取得されます。

以下のどちらかの場合、動作は不定となり保証されません。

ー MPI_INITルーチンの実行前に本ルーチンが呼び出された

ー MPI_FINALIZEルーチンの実行後に本ルーチンが呼び出された

5.1.3 ランクの問合せ

5.1.3.1 FJMPI_TOPOLOGY_GET_RANKS

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

指定された論理座標またはTofu座標に存在するプロセスのうち、指定されたコミュニケータに割り当てられているプロセスのランクを取得します。最大で、引数 maxppn に指定した個数分のプロセスのランクを取得します。実際に取得したランクの個数は、引数 outppn に格納されます。本ルーチンの利用例を以下に示します。

- 論理座標を対象とし、対象の座標に1プロセス存在する場合

  view = FJMPI_LOGICAL、maxppn = 1以上の値を指定します。また、論理X座標、論理Y座標、論理Z座標の値をそれぞれcoords[0]、coords[1]、coords[2]に指定します。指定したcomm、coordsに存在するプロセスのランクがranks[0]に格納されます。outppnには1が格納されます。

- Tofu座標を対象とし、対象の座標に4プロセス存在する場合

  view = FJMPI_TOFU_SYS または FJMPI_TOFU_REL、maxppn = 4以上の値を指定します。また、Tofu座標のX座標、Y座標、Z座標、A座標、B座標、C座標の値をそれぞれcoords[0]、coords[1]、coords[2]、coords[3]、coords[4]、coords[5]に指定します。指定したcomm、coordsに存在するプロセスのランクがranks[0]、ranks[1]、ranks[2]、ranks[3]に格納されます。outppnには4が格納されます。

型	変数	説明	IN/OUT
MPI_Comm	comm	コミュニケータを指定	IN
int	view	論理座標かTofu座標かを表すマクロを指定 FJMPI_LOGICAL: 論理座標 FJMPI_TOFU_SYS:Tofu座標(実際に割り当てられた座標) FJMPI_TOFU_REL:Tofu座標(引数commのランク0を基準とした相対座標、commがグループ間コミュニケータの場合リモートグループのランク0を基準とした相対座標)	IN
int[]	coords	取得するランクのプロセスが割り当てられている座標の値を指定	IN
int	maxppn	ランクを取得する最大のプロセス数を指定	IN
int*	outppn	ランクを取得したプロセス数	OUT
int[]	ranks	取得したランクを格納した配列 グループ間コミュニケータを指定した場合は、リモートグループのランクを格納	OUT

<復帰値>

正常時	FJMPI_SUCCESS	―
異常時	FJMPI_ERR_TOPOLOGY_NO_PROCESS	指定した座標に並列プロセスが配置されていない場合
	FJMPI_ERR_TOPOLOGY_NODE_SHARED_JOB	ジョブ種別がノード共有ジョブの場合

		(ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

引数 coords は、引数 view に FJMPI_LOGICAL を指定した場合、ジョブと同じ次元数の要素分、値を設定する必要があります。引数 viewに FJMPI_TOFU_SYS または FJMPI_TOFU_REL を指定した場合は、6要素分、値を設定する必要があります。正しく設定されていない場合の動作は不定となり保証されません。

引数 coords に指定した座標に存在する、実際のプロセス数を取得するためには、引数 maxppn の値を大きくした状態で本ルーチンを実行し、引数 outppn の値を参照します。

実際に存在するプロセスの数によっては、引数 maxppn に指定した値よりも引数 outppn に格納される値の方が小さくなります。この場合は、outppn の値と同じ要素数分だけ、引数 ranks に指定した配列内の値が更新されます。

以下のどちらかの場合、動作は不定となり保証されません。

- MPI_INITルーチンの実行前に本ルーチンが呼び出された
- MPI_FINALIZEルーチンの実行後に本ルーチンが呼び出された

5.1.4 カルテシアン構造をもつコミュニケータのランク付け問合せ

5.1.4.1 FJMPI_TOPOLOGY_CART_REORDER

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

カルテシアン構造をもつコミュニケータのトポロジー情報から、ランク付けが実行されたかどうかを判別するための情報を返します。引数 reorderの値が1の場合、ランクの並べ替えが行われたことを表します。引数reorderの値が0の場合、ランクの並べ替えが行われていないことを表します。

型	変数	説明	IN/OUT
MPI_Comm	comm	ランク付けの判別を行うコミュニケータ	IN
int*	reorder	コミュニケータのランク付けの情報	OUT

<復帰値>

正常時	FJMPI_SUCCESS	－
異常時	FJMPI_ERR_TOPOLOGY_INVALID_COMM	・ グループ間コミュニケータが指定された場合

		・ カルテシアン構造をもたないコミュニケータが指定された場合

<備考>

以下のどちらかの場合、動作は不定となり保証されません。

－ MPI_INITルーチンの実行前に本ルーチンが呼び出された

－ MPI_FINALIZEルーチンの実行後に本ルーチンが呼び出された

5.1.5 サンプルプログラム

ランク問合せインターフェースのサンプルプログラムを以下に示します。

本プログラムでは、3次元のプロセス形状で実行されているものとし、次のような処理が行われています。

1. MPIプロセスの次元数およびプロセス形状を問い合わせる
2. 座標ごとに、自プロセスと隣接するプロセスの座標を求め、その座標からランク情報を取得する
3. 座標ごとに、2.で問い合わせたランク情報から座標を取得し、元の座標と一致することを確認する

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

#define FAILURE 1

int main(int argc, char *argv[])
{
    int coords[3], i, size, rank;
    int rc, dim;
    int shape_x, shape_y, shape_z;
    int tmp_coords[3];
    int next_coords[3];
    int ans;
    int outppn;
    char host[255];

    gethostname(host, 255);

    MPI_Init(&argc, &argv);

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

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

    /* 結果チェック */
    if (3 != dim) {
        fprintf(stderr, "[%s] Dimension size ERROR¥n", host);
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }

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

```
}

/*****************************************************************************
 * 自身の座標を取得                                                          *
 *****************************************************************************/
rc = FJMPI_Topology_get_coords(MPI_COMM_WORLD, rank, FJMPI_LOGICAL, 3, coords);
if (FJMPI_SUCCESS != rc) {
    fprintf(stderr, "[%s] FJMPI_Topology_get_coords ERROR¥n", host);
    MPI_Abort(MPI_COMM_WORLD, FAILURE);
}

/*****************************************************************************
 * 座標ごとに前後の隣接するプロセスを取得                                    *
 *****************************************************************************/
for(i = 0; i < 6; i++)
{
    switch(i)
    {
        case 0: /* X軸に隣接 */
        case 1:
            tmp_coords[0] = (0 == i) ?
            (coords[0] - 1 >= 0) ? coords[0] - 1 : shape_x - 1
            : (coords[0] + 1 < shape_x) ? coords[0] + 1 : 0;
            tmp_coords[1] = coords[1];
            tmp_coords[2] = coords[2];
            break;
        case 2: /* Y軸に隣接 */
        case 3:
            tmp_coords[1] = (2 == i) ?
            (coords[1] - 1 >= 0) ? coords[1] - 1 : shape_y - 1
            : (coords[1] + 1 < shape_y) ? coords[1] + 1 : 0;
            tmp_coords[0] = coords[0];
            tmp_coords[2] = coords[2];
            break;
        case 4: /* Z軸に隣接 */
        case 5:
            tmp_coords[2] = (4 == i) ?
            (coords[2] - 1 >= 0) ? coords[2] - 1 : shape_z - 1
            : (coords[2] + 1 < shape_z) ? coords[2] + 1 : 0;
            tmp_coords[0] = coords[0];
            tmp_coords[1] = coords[1];
            break;
    }

    rc = FJMPI_Topology_get_ranks(MPI_COMM_WORLD, FJMPI_LOGICAL, tmp_coords,
                                  1, &outppn, &ans);
    switch (rc)
    {
        case FJMPI_SUCCESS:
            break;
        case FJMPI_ERR_TOPOLOGY_NO_PROCESS:
            /* もっとも近いMPIプロセスを見つけるまで探す. */
            while (rc == FJMPI_ERR_TOPOLOGY_NO_PROCESS) {
                coords[0] = coords[0] - 1;
                rc = FJMPI_Topology_get_ranks(MPI_COMM_WORLD, FJMPI_LOGICAL, coords,
                                              1, &outppn, &ans);
            }
            break;
        default:
            fprintf(stderr, "[%s] FATAL ERROR¥n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }
```

```
        /*****************************************************************************
         * 使用座標と取得座標の一致を確認                                            *
         *****************************************************************************/
        rc = FJMPI_Topology_get_coords(MPI_COMM_WORLD, ans, FJMPI_LOGICAL, 3, next_coords);
        if (MPI_SUCCESS != rc) {
            fprintf(stderr, "[%s] FJMPI_Topology_rank2xyz ERROR¥n", host);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
        if ((next_coords[0] != tmp_coords[0]) ||
            (next_coords[1] != tmp_coords[1]) ||
            (next_coords[2] != tmp_coords[2])) {
            fprintf(stderr, "[%s] PARAM ERROR¥n", host);
            fprintf(stderr, "[%s %d] [user:%u-%u-%u] [get:%u-%u-%u] [rank|next:%d|%d]¥n",
            host, i, tmp_coords[0], tmp_coords[1], tmp_coords[2],
            next_coords[0], next_coords[1], next_coords[2], rank, ans);
            MPI_Abort(MPI_COMM_WORLD, FAILURE);
        }
    }

    MPI_Finalize();

    return 0;
}
```

5.2 区間指定MPI統計情報インターフェース

本インターフェースを使用することでユーザーが指定する位置で統計情報のデータを採取します。

本処理系においてサポートする区間指定MPI統計情報インターフェースについて、具体的なルーチンの一覧を下表に示します。

本インターフェースを利用するには、MCAパラメーター mpi_print_statsに3または4を指定する必要があります。

表5.2 区間指定MPI統計情報インターフェースでサポートするルーチン一覧

ルーチン名	機能概要
FJMPI_COLLECTION_START	区間指定MPI統計情報の採取開始
FJMPI_COLLECTION_STOP	区間指定MPI統計情報の採取停止
FJMPI_COLLECTION_PRINT	区間指定MPI統計情報の採取データ出力
FJMPI_COLLECTION_CLEAR	区間指定MPI統計情報の採取データ初期化

5.2.1 区間指定MPI統計情報インターフェース仕様

5.2.1.1 FJMPI_COLLECTION_START

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

```
USE mpi_f08_ext
FJMPI_Collection_start()
```

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
FJMPI_COLLECTION_START()
```

<説明>

区間指定MPI統計情報の採取を開始します。

MCAパラメーター mpi_print_statsに3または4を指定すると有効になります。

ただし、以下のどちらかで本ルーチンを呼び出した場合は、無効になります。

－ MPI_INITルーチンの実行前

－ MPI_FINALIZEルーチンの実行後

<復帰値>

ありません。

<備考>

本ルーチンを連続実行した場合、最初の開始指示が有効となります。

5.2.1.2 FJMPI_COLLECTION_STOP

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

```
USE mpi_f08_ext
FJMPI_Collection_stop()
```

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
FJMPI_COLLECTION_STOP()
```

<説明>

区間指定MPI統計情報の採取を停止します。

MCAパラメーター mpi_print_statsに3または4を指定すると有効になります。

ただし、以下のどちらかで本ルーチンを呼び出した場合は、無効になります。

－ MPI_INITルーチンの実行前

－ MPI_FINALIZEルーチンの実行後

<復帰値>

ありません。

<備考>

データ採取が繰り返された場合、採取結果は累積されます。

本ルーチンを連続実行した場合、最初の停止指示が有効となります。

5.2.1.3 FJMPI_COLLECTION_PRINT

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

型	変数	説明	IN/OUT
char*	str	区間ごとに標準エラー出力へ出力させたい文字列	IN

引数strは、統計情報を識別するための文字列で、印字可能な半角英数字記号を30文字まで指定できます。30文字を超えた文字列は切り捨てられます。

区間指定MPI統計情報の採取を標準エラー出力へ出力します。

MCAパラメーター mpi_print_statsに3または4を指定すると有効になります。

ただし、以下のどちらかで本ルーチンを呼び出した場合は、無効になります。

- MPI_INITルーチンの実行前
- MPI_FINALIZEルーチンの実行後

MCAパラメーター mpi_print_statsの値により一部動作が異なります。

mpi_print_stats	動作	説明
3	集団的操作	同じ MPI_COMM_WORLD に属する全プロセスが同時に呼び出す必要があります。正しく呼び出しを行わないと、デッドロックの原因となります。
4	非集団的操作	特定のプロセスだけ呼び出すことが可能です。

<復帰値>

ありません。

<備考>

文字列には、印字可能な半角英数記号だけを使用してください。

区間指定出力モードでは、本ルーチンによって統計情報が出力されるので、一度も呼ばれないと統計情報は出力されません。

5.2.1.4 FJMPI_COLLECTION_CLEAR

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

```
USE mpi_f08_ext
FJMPI_Collection_clear()
```

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
FJMPI_COLLECTION_CLEAR()
```

<説明>

区間指定MPI統計情報のすべてのデータを初期化します。

MCAパラメーター mpi_print_statsに3または4を指定すると有効になります。

ただし、以下のどちらかで本ルーチンを呼び出した場合は、無効になります。

- MPI_INITルーチンの実行前
- MPI_FINALIZEルーチンの実行後

<復帰値>

ありません。

<備考>

本ルーチンは、採取データ、開始、終了時刻などすべての統計情報を初期化します。

5.2.2 サンプルプログラム

区間指定MPI統計情報インターフェースのサンプルプログラムを示します。本プログラムはMPI_ALLTOALLルーチン実行だけの区間指定と、MPI_ALLGATHERルーチンを含むルーチン自体を区間指定するプログラムです。

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

int test_aa = 200;
int test_ag = 300;

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

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

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

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

    FJMPI_Collection_clear();

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

    MPI_Finalize();

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

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

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

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

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

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

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

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

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

区間指定MPI統計情報インターフェースのサンプルプログラムを示します。本プログラムはMPI_BCASTルーチンの区間指定と、MPI_ALLREDUCEルーチンの区間指定をするプログラムです。

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

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

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

```
call MPI_BARRIER(MPI_COMM_WORLD,ierr)

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

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

5.3 拡張持続的通信要求インターフェース

本インターフェースを利用することで、MPIの持続的な通信要求では実現できない通信処理を非同期に開始し、演算処理とオーバーラップさせることが可能になります。

本処理系において提供する拡張持続的通信要求インターフェースについて、具体的なルーチンの一覧を下表に示します。

表5.3 拡張持続的通信要求インターフェースのルーチン一覧

ルーチン名	機能概要
FJMPI_PREQUEST_SEND_INIT	拡張持続的通信要求インターフェースを使用した送信の初期化
FJMPI_PREQUEST_RECV_INIT	拡張持続的通信要求インターフェースを使用した受信の初期化
FJMPI_PREQUEST_START	拡張持続的通信要求インターフェースを使用した通信の開始
FJMPI_PREQUEST_STARTALL	拡張持続的通信要求インターフェースを使用した通信の一括開始

5.3.1 概要

拡張持続的通信要求インターフェースは、以下に示す仕組みにより、実際の通信処理と演算処理部分をオーバーラップします。

表5.4 拡張持続的通信要求インターフェースの仕組み

FJMPI_Prequest_send_init... (1)相手プロセスに、情報を通知する	FJMPI_Prequest_recv_init... (1)相手プロセスに、情報を通知する
FJMPI_Prequest_start (2)startで実データの送信を開始する	FJMPI_Preqeust_start (2)startで実データを受信する
(演算処理)	(演算処理)
MPI_Wait	MPI_Wait

メモリ上でメッセージが連続しない派生データ型を指定すると、MPIルーチンを使用した場合よりも通信性能が劣る場合がありますので、注意してください。

5.3.2 拡張持続的通信要求インターフェース仕様

5.3.2.1 FJMPI_PREQUEST_SEND_INIT

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

引数は、MPI_SEND_INITルーチンに準じます。

本ルーチンで作成した送信要求は、備考に記載された事項を除いて、MPI_SEND_INITルーチンで作成した送信要求と同じようにほかのMPIルーチンで使用できます。

<復帰値>

正常時	0が返ります。
異常時	0以外の値が返ります。
ノード共有ジョブで実行された場合	0以外の値が返ります。 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

- 本ルーチンで作成した送信要求は、FJMPI_PREQUEST_STARTルーチンまたはFJMPI_PREQUEST_STARTALLルーチンで開始できます。
- 本ルーチンで作成した送信要求を用いて送信した通信は、FJMPI_PREQUEST_RECV_INITルーチンで作成した受信要求を用いて受信しなければなりません。
- MPI_CANCELルーチンによるキャンセルはできません。その場合、MPI_CANCELルーチンはエラー復帰します。
- 本ルーチンを複数回呼び出す場合、同じコミュニケータとランクとタグをFJMPI_PREQUEST_SEND_INITルーチンで同時に利用することはできません。同じランクとコミュニケータとタグで作成された送信要求がすでに存在する場合、以下のメッセージが出力されます。

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

5.3.2.2 FJMPI_PREQUEST_RECV_INIT

<書式>

C言語書式

```
#include <mpi-ext.h>
int FJMPI_Prequest_recv_init(void *buf, int count, MPI_Datatype datatype,
```

```
                              int source, int tag, MPI_Comm comm,
                              MPI_Request *request)
```

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

引数は、MPI_RECV_INITルーチンに準じます。

本ルーチンで作成した受信要求は、備考に記載された事項を除いて、MPI_RECV_INITルーチンで作成した受信要求と同じようにほかのMPIルーチンで使用できます。

<復帰値>

正常時	0が返ります。
異常時	0以外の値が返ります。
ノード共有ジョブで実行された場合	0以外の値が返ります。 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

- 引数 source に、MPI_ANY_SOURCEを指定することはできません。
- 本ルーチンで作成した受信要求を用いて受信する通信は、FJMPI_PREQUEST_SEND_INITルーチンで作成した送信要求を用いて送信されなければなりません。
- 本ルーチンで作成した受信要求は、FJMPI_PREQUEST_STARTルーチンまたはFJMPI_PREQUEST_STARTALLルーチンで開始できます。
- MPI_CANCELルーチンによるキャンセルはできません。その場合、MPI_CANCELルーチンはエラー復帰します。
- 本ルーチンで宣言した通信は、MPI_PROBE関連ルーチンを使用して受信可能かどうかの判断はできません。
- 本ルーチンを複数回呼び出す場合、同じコミュニケータとランクとタグをFJMPI_PREQUEST_RECV_INITルーチンで同時に利用することはできません。同じランクとコミュニケータとタグで作成された受信要求がすでに存在する場合、以下のメッセージが出力されます。

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

5.3.2.3 FJMPI_PREQUEST_START

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

引数は、MPI_STARTルーチンに準じます。

<復帰値>

正常時	0が返ります。
異常時	0以外の値が返ります。
ノード共有ジョブで実行された場合	0以外の値が返ります。 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

- 使用できる通信要求はFJMPI_PREQUEST_SEND_INITルーチンまたはFJMPI_PREQUEST_RECV_INITルーチンで作成した通信要求だけとなります。
- FJMPI_PREQUEST_SEND_INITルーチンで獲得した送信要求を指定した場合は、処理がローカルではなく、受信プロセスで対応するFJMPI_PREQUEST_STARTルーチンまたはFJMPI_PREQUEST_STARTALLルーチンが呼ばれるのを待ちます。

5.3.2.4 FJMPI_PREQUEST_STARTALL

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

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

<説明>

引数は、MPI_STARTALLルーチンに準じます。

<復帰値>

正常時	0が返ります。
異常時	0以外の値が返ります。

ノード共有ジョブで実行された場合	0以外の値が返ります。 (ノード共有ジョブの詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください)

<備考>

- 使用できる通信要求はFJMPI_PREQUEST_SEND_INITルーチンまたはFJMPI_PREQUEST_RECV_INITルーチンで獲得した通信要求だけとなります。
- FJMPI_PREQUEST_SEND_INITルーチンで獲得した送信要求が含まれている場合は、その送信要求への処理はローカルではなく、受信プロセスで対応するFJMPI_PREQUEST_STARTルーチンまたはFJMPI_PREQUEST_STARTALLルーチンが呼ばれるのを待ちます。

5.3.3 サンプルプログラム

拡張持続的通信要求インターフェースのサンプルプログラムを示します。

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

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

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

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

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

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

    MPI_Barrier(MPI_COMM_WORLD);

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

    MPI_Barrier(MPI_COMM_WORLD);
    stime = MPI_Wtime();

    FJMPI_Prequest_startall(2,&reqs[0]);

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

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

    etime = MPI_Wtime();

    MPI_Barrier(MPI_COMM_WORLD);

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

```
    }

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

    return (0);
}
```

5.4 区間指定MPI非同期通信促進インターフェース

本インターフェースを使用することで、ユーザーが指定する区間だけでアシスタントコアを用いて非同期通信の促進を行うことができます。本機能の詳細については“6.2 アシスタントコアを用いた非同期通信の促進”をお読みください。

本インターフェースは、MCAパラメーターopal_progress_thread_modeの値が1または2のときだけ効果があります。

本処理系においてサポートする区間指定MPI非同期通信促進インターフェースについて、具体的なルーチンの一覧を下表に示します。

表5.5 区間指定MPI非同期通信促進インターフェースでサポートするルーチン一覧

ルーチン名	機能概要
FJMPI_PROGRESS_START	非同期通信の促進処理の開始
FJMPI_PROGRESS_STOP	非同期通信の促進処理の停止

5.4.1 区間指定MPI非同期通信促進インターフェース仕様

5.4.1.1 FJMPI_PROGRESS_START

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

```
USE mpi_f08_ext
FJMPI_Progress_start()
```

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
FJMPI_PROGRESS_START()
```

<説明>

MPI非同期処理進行スレッドの指定区間(MPI呼び出しなし)モードおよび指定区間(MPI呼び出しあり)モードにおいて、非同期通信の促進処理を開始することを指示します。

指定区間(MPI呼び出しなし)では、対応するFJMPI_PROGRESS_STOPルーチンを呼ぶまでに、MPIルーチンや拡張インターフェースを呼ぶことはできません。呼んだかどうかのチェックは行われず、呼んだ場合の動作は不定です。指定区間(MPI呼び出しあり)では、呼ぶことはできますが、呼び出し時に性能のオーバーヘッドがかかります。

自プロセス内に未完了のノンブロッキング通信が2つ以上存在する状態でこのルーチンを呼ぶこともできます。また、未完了のノンブロッキング通信が1つも存在しない状態でこのルーチンを呼ぶこともできます。後者の場合は、ほとんど何の効果もありません。

以下のどちらかの場合は、このルーチンの呼び出しは無視されます。

- 動作モードが指定区間(MPI呼び出しなし)モードまたは指定区間(MPI呼び出しあり)モードでない場合
- すでに非同期通信の促進処理を開始している場合。すなわち、FJMPI_PROGRESS_STARTルーチンがすでに呼ばれていて、FJMPI_PROGRESS_STOPルーチンがまだ呼ばれていない場合

<復帰値>

ありません。

5.4.1.2 FJMPI_PROGRESS_STOP

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

```
USE mpi_f08_ext
FJMPI_Progress_stop()
```

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
FJMPI_PROGRESS_STOP()
```

<説明>

MPI非同期処理進行スレッドの指定区間(MPI呼び出しなし)モードおよび指定区間(MPI呼び出しあり)モードにおいて、非同期通信の促進処理を停止することを指示します。

以下のどちらかの場合は、このルーチンの呼び出しは無視されます。

- 動作モードが指定区間(MPI呼び出しなし)モードまたは指定区間(MPI呼び出しあり)モードでない場合
- まだ非同期通信の促進処理を開始していない場合。すなわち、FJMPI_PROGRESS_STARTルーチンがまだ一度も呼ばれていないか、FJMPI_PROGRESS_STARTルーチンが呼ばれているが対応するFJMPI_PROGRESS_STOPルーチンもすでに呼ばれている場合

<復帰値>

ありません。

5.4.2 サンプルプログラム

区間指定MPI非同期通信促進インターフェースのサンプルプログラムを以下に示します。本プログラムは演算と4つの通信を並行して行うプログラムです。

MCAパラメーターopal_progress_thread_modeの値が0の場合と1の場合とで、実行時間の差を見ることができます。ただし、一般に、演算単独の時間と通信単独の時間が大きく異なると、演算と通信のオーバーラップの効果は小さくなります。そのため、プログラム翻訳時のコンパイラオプションやプログラム実行時のプロセス配置などによって、効果の程度が大きく変わります。

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

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

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

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

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

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

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

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

        stime = MPI_Wtime();

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

        FJMPI_Progress_start();

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

        FJMPI_Progress_stop();

        MPI_Waitall(4, reqs, MPI_STATUSES_IGNORE);

        etime = MPI_Wtime();

    }

    MPI_Finalize();

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

    return 0;
}
```

5.5 持続的集団通信要求インターフェース

5.5.1 概要

本処理系では、MPI-4.0規格に向けたドラフト版(https://www.mpi-forum.org/docs/drafts/mpi-2018-draft-report.pdf)に含まれる持続的集団通信要求のインターフェースが、MPI_ではなくMPIX_から始まるルーチン名で実装されています。

5.5.2 持続的集団通信要求インターフェース仕様

以下に本処理系で実装されている持続的集団通信要求のルーチン一覧を示します。これらのルーチンの挙動や引数の意味についてはMPI規格のドラフトをご覧ください。

<書式>

C言語書式

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

int MPIX_Allgatherv_init(const void *sendbuf, int sendcount, MPI_Datatype sendtype,
    void *recvbuf, const int recvcounts[], const int displs[], MPI_Datatype recvtype,
```

```
    MPI_Comm comm, MPI_Info info, MPI_Request *request)

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Fortran (USE mpi_f08_ext)書式

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

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

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

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

 MPIX_Alltoallv_init(sendbuf, sendcounts, sdispls, sendtype, recvbuf,
```

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

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

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

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

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

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

MPIX_Gatherv_init(sendbuf, sendcount, sendtype, recvbuf, recvcounts, displs,
recvtype, root, comm, info, request, ierror)
    TYPE(*), DIMENSION(..), INTENT(IN), ASYNCHRONOUS :: sendbuf
    TYPE(*), DIMENSION(..), ASYNCHRONOUS :: recvbuf
```

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

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

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

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

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

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

```
    TYPE(MPI_Info), INTENT(IN) :: info
    TYPE(MPI_Request), INTENT(OUT) :: request
    INTEGER, OPTIONAL, INTENT(OUT) :: ierror

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

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

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

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

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

MPIX_Neighbor_alltoallw_init(sendbuf, sendcounts, sdispls, sendtypes,
```

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

Fortran (USE mpi_ext)書式

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

<説明>

これらのルーチンで作成した通信要求は、MPI規格で定義されている通信要求操作ルーチンで利用できます。

これらのルーチンでは、引数infoによるヒントの指定は無効です。

<復帰値>

正常時	MPI_SUCCESSが返ります。
異常時	MPI_SUCCESS以外の値が返ります。

<備考>

これらのルーチンはMPI規格のドラフト版をもとにした実装となっており、将来の改版で動作や引数が変更される場合があります。

本処理系の将来の改版において、MPIX_から始まるルーチン名の本インターフェースとMPI_から始まるルーチン名のMPI規格のインターフェースの両方が使用可能になり、その一定期間の後に本インターフェースは廃止されます。

5.5.3 通信と演算のオーバーラップ

持続的集団通信要求から行われる一部の通信については、Tofuインターコネクトの機能を用い通信と演算のオーバーラップを行える場合があります。また、それ以外の場合でも、アシスタントコアを用いた非同期通信の促進が行われていれば通信と演算のオーバーラップが行われます。

Tofuインターコネクトの機能を用いた通信と演算のオーバーラップが適用されている場合、MPI_STARTルーチンまたはMPI_STARTALLルーチンを呼び出したとき直ちに通信が開始され、MPI_WAITルーチンまたはMPI_WAITALLルーチンが呼ばれるまで計算コードとのオーバーラップが可能です。

アシスタントコアを用いた非同期通信の促進が適用されている場合の挙動については、“6.2 アシスタントコアを用いた非同期通信の促進”をご覧ください。

5.5.3.1 通信と演算のオーバーラップの適用条件

持続的集団通信要求では、次の条件をすべて満たして作成された通信要求で、Tofuインターコネクトの機能を用いて通信と演算をオーバーラップさせることができます。

- MPIX_ALLGATHER_INITルーチンで作成された通信要求である
- 通信要求に対する開始操作が2回目以降である
- 送信データ型と受信データ型が基本データ型である
- コミュニケータがグループ内コミュニケータである
- ノード内プロセス数(proc_per_node)、コミュニケータサイズ(comm_size)、および送信メッセージサイズ(msg_size)が以下の不等式を満たす

$$comm\_size-2+ceil(ceil(msg\_size \div (floor((15-proc\_per\_node) \div 15)+2)) \div 16777215) \times (comm\_size-1) \leqq 2048$$

- MPIジョブのノード内プロセス数が30以下である
- 通信要求の作成のタイミングで、上記の条件を満たして作成された通信要求が、プロセス内にほかに存在しない

これらの条件は将来の改版により変更される場合があります。

5.5.3.2 備考

Tofuインターコネクトの機能を用いた通信と演算のオーバーラップが適用される場合、その通信要求に関してMCAパラメーターmpi_check_buffer_writeによる通信バッファの書込み破壊の監視は行われません。

5.6 追加の定義済みデータ型

5.6.1 概要

富士通Cコンパイラと富士通C++コンパイラのclangモードでは、半精度(16ビット)浮動小数点型として以下の2種類の型をサポートしています。

- _Float16
- __fp16

MPI-3.1規格ではこれらの型に対応する定義済みデータ型を定義していないため、本処理系では独自の定義済みデータ型を利用可能にしています。

富士通Cコンパイラと富士通C++コンパイラにおける半精度浮動小数点型の詳細については、C言語使用手引書とC++言語使用手引書をお読みください。

5.6.2 半精度浮動小数点型用定義済みデータ型仕様

<書式>

C言語書式

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

Fortran (USE mpi_f08_ext)書式

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

Fortran (USE mpi_ext)書式

```
USE MPI_EXT
INTEGER mpix_c_float16
```

<説明>

C言語とC++の_Float16型と__fp16型に対応する名前付き定義済みデータ型です。

このデータ型は、MPI_FLOATなどのほかの浮動小数点型に対応する名前付き定義済みデータ型と同様に、基本データ型としてMPIルーチンで使用できます。ただし、MPI_TYPE_GET_NAMEルーチンで取得できるこのデータ型の名前は、“MPIX_C_FLOAT16”とは異なる名前になります。

参考

- Fortranの半精度浮動小数点型であるREAL(2)型に対しては、MPI-3.1規格で定義されている名前付き定義済みデータ型MPI_REAL2が使用可能です。
- C言語やC++のこれらの型のデータを言語間結合によりFortranプログラムで使用する場合に、Fortran書式が必要になります。言語間結合については、各コンパイラのマニュアルをお読みください。

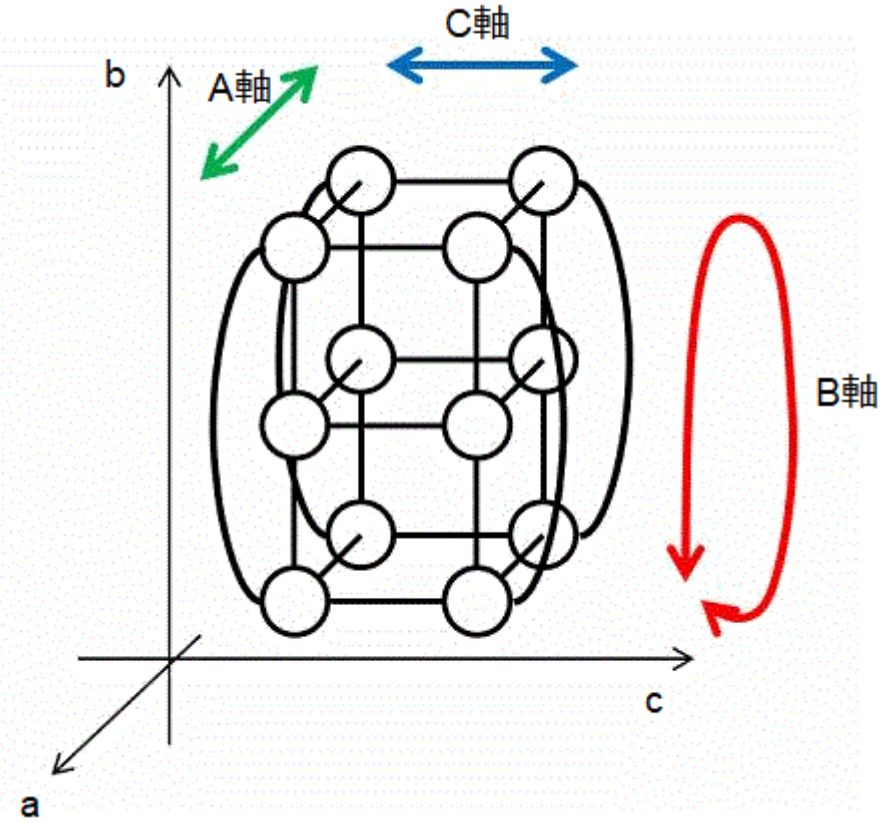
第6章 補足事項

6.1 Tofuインターコネクト

6.1.1 Tofuインターコネクトの構成

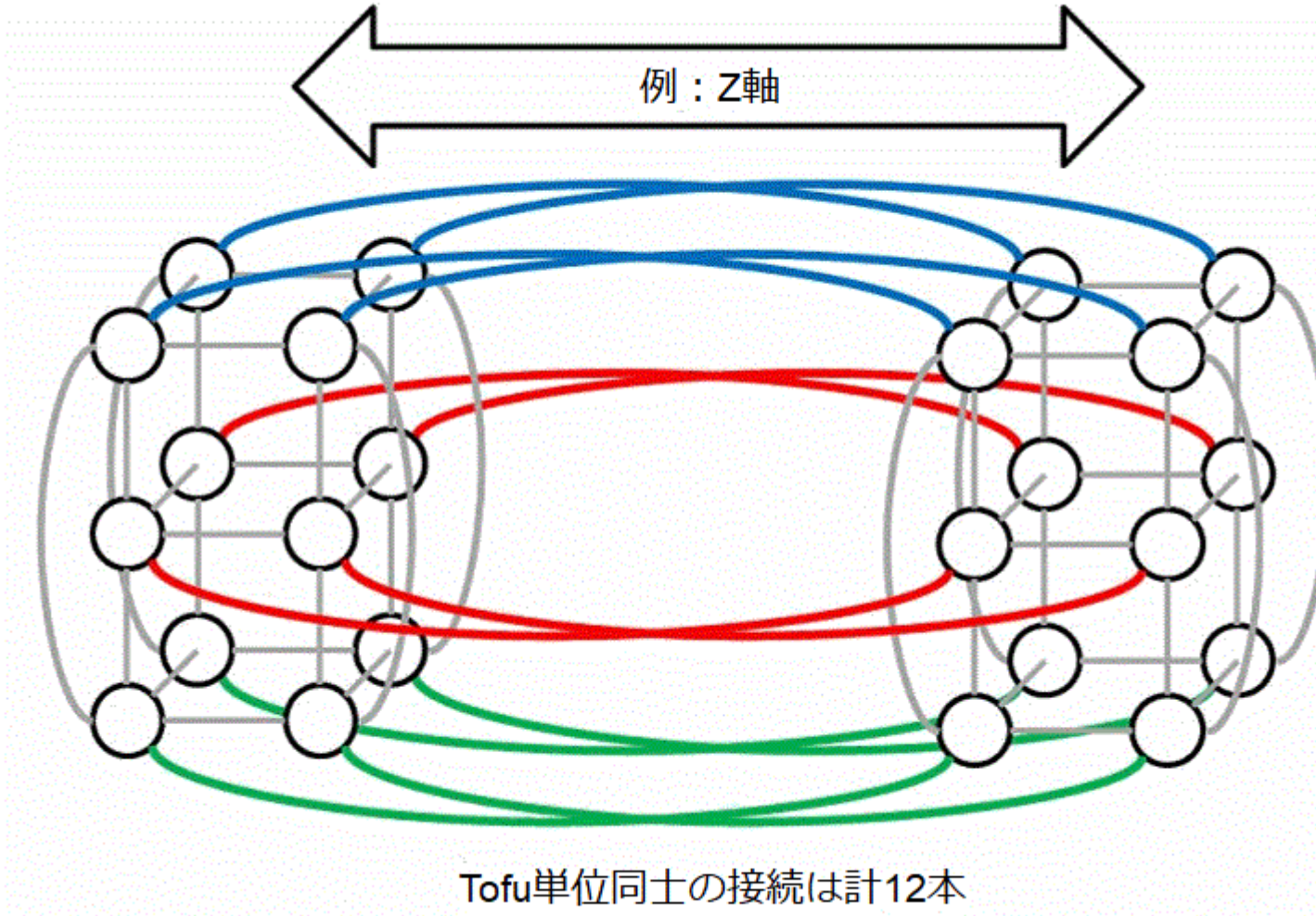
Tofuインターコネクトは、物理的には6次元メッシュ／トーラスネットワークで構成されています。6次元メッシュ／トーラスネットワークの座標はX, Y, Z, A, B, Cの6次元で与えられます。この6次元で与えられた座標をTofu座標と呼称します。また、大きさ2×3×2のA, B, C軸で構成される単位をTofu単位と呼称します。

図6.1 Tofu単位



隣接するTofu単位同士はX, Y, Z軸で接続されており、A, B, C軸座標を同じくするノード同士がつながるように構成されています。このため、Tofu単位間はX, Y, Z軸方向それぞれに関し12本の接続を持っています。

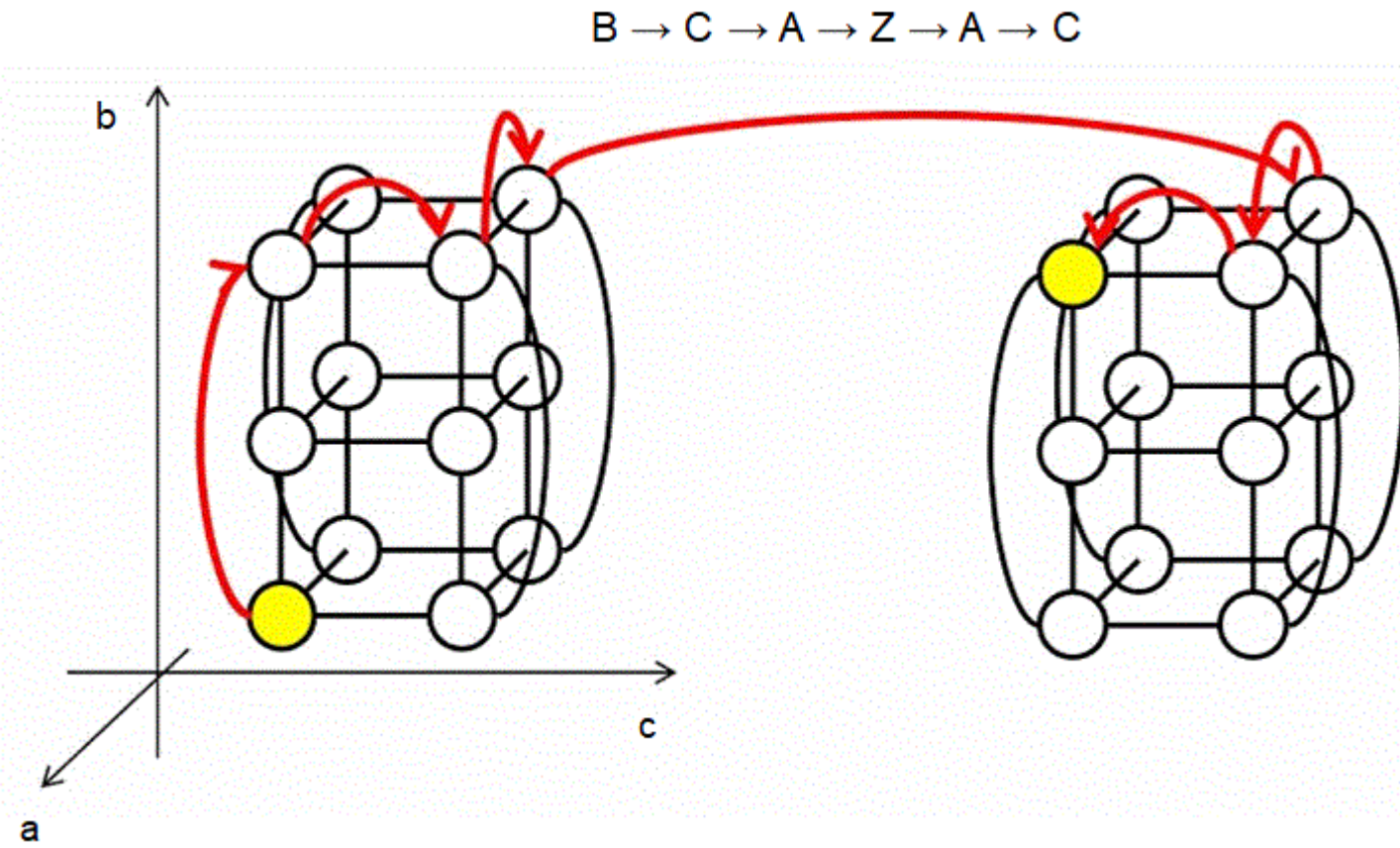
図6.2 Tofu単位同士の接続



6.1.2 ルーチング

Tofuインターコネクトにおいて、パケットは座標軸をB, C, A, X, Y, Z, A, C, Bの順に移動します。最初のABC軸の経路は故障ノードの回避や経路分散のために、残りの経路は目的のノードへ到達するためのものです。MPIライブラリが、最初の12通りの経路のうちどの経路が使われるかを決定します。一般的な次元オーダ・ルーチングと比べ、経路の数が増加しているため、これを拡張次元オーダ・ルーチングと呼称します。

図6.3 最初のABC軸ルーチングでABC軸をすべて移動するような例

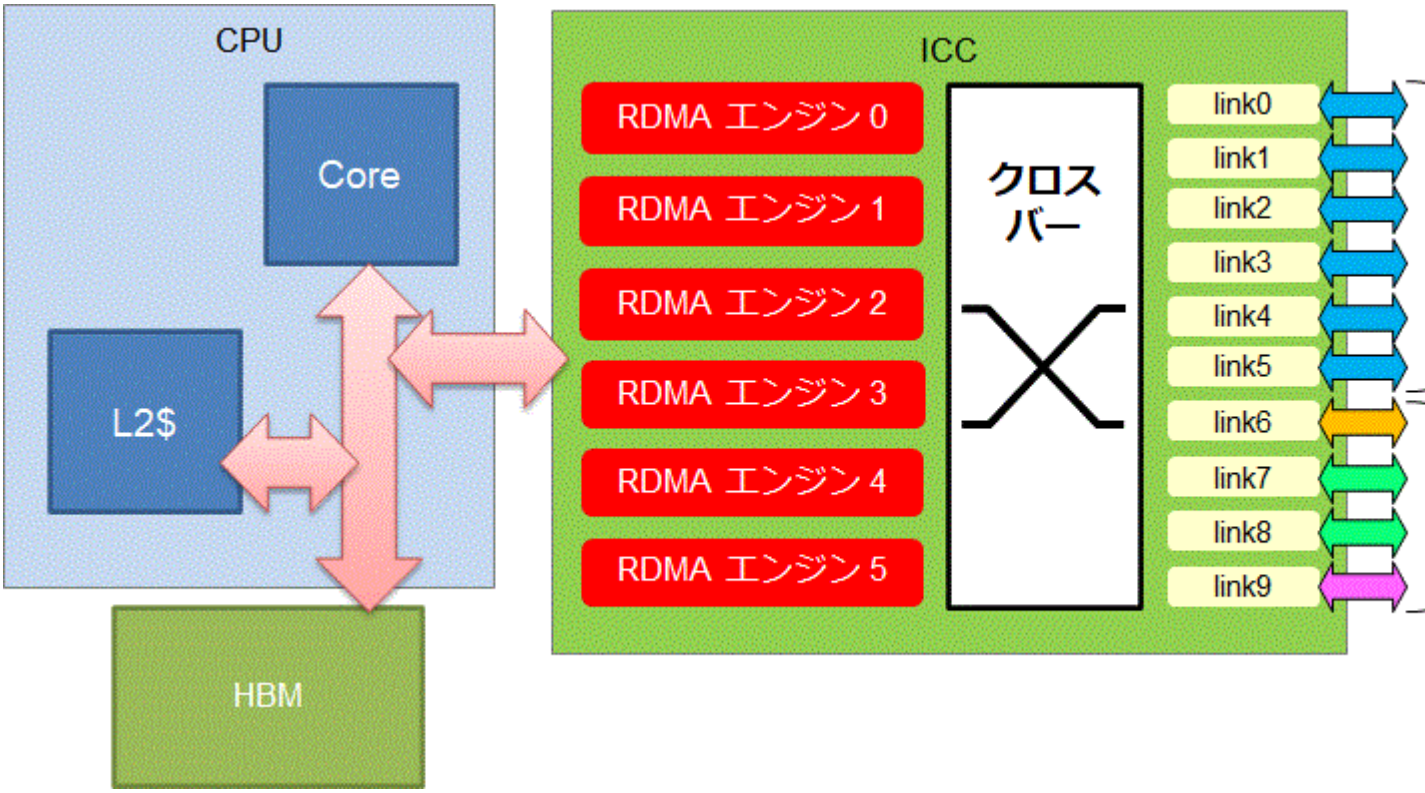


6.1.3 ノード内の構成

各ノードにはInterconnect Controller(ICC)と呼ばれるモジュールが存在し、他ノードとの通信を司っています。内部にはTofu Network Interface(TNI)というRDMAエンジンが6つ存在します。各TNIは同時に1送信・1受信が可能であり、6つのTNIをあわせると、ICCは同時に6送信・6受信が可能です。ポート数はXYZ軸方向に6本、ABC軸方向に4本の計10本存在します。

MPIライブラリは、この6つのRDMAエンジンを利用して、RDMA通信を行います。

図6.4 ICCの構成



6.2 アシスタントコアを用いた非同期通信の促進

本処理系では、演算と通信がオーバーラップすることを期待してノンブロッキング通信を使用しても、通信が演算と完全には非同期に進行されず、MPI_WAITルーチンなどの完了待ちルーチンが呼び出されてからメッセージ本体の転送が始まり、実際にはオーバーラップしないことがあります。本システムのCPUであるA64FXには、ジョブとして投入されたユーザープログラムの実行を担当する演算コア(48個)の他に、OSやIO処理などを担当するアシスタントコア(2個または4個)が存在します。

そこで、本処理系では、ノンブロッキング通信を使用したMPIプログラムに対して、アシスタントコアを用いて非同期通信を促進する機能を用意しています。本機能では、MPIプログラムの実行時に、ノンブロッキング通信を進行するためのスレッド(MPI非同期処理進行スレッド)を、アシスタントコア上に作成します。これにより、演算コアでのユーザープログラムの演算と同時に、アシスタントコアで通信が非同期に進行され、すなわち演算と通信のオーバーラップが促進され、結果的にMPIプログラムの実行時間が短縮される可能性があります。この効果は、ノンブロッキング通信で転送しようとするメッセージのサイズが大きい場合、特に“しきい値”以上である場合に、大きくなる傾向があります。

ただし、アシスタントコアは計算ノード内の複数のユーザープロセス、デーモンプロセス、カーネルなどで共有して使用されるため、MPIプログラムの性能のバラつきや実行時間が逆に増加するといった副作用が生じる場合があります。特に、目安として1ノード内に4つ以上の並列プロセスが割り当てられる場合は、これらの副作用が大きくなる可能性が高くなります。

また、MPIプログラムがMPIルーチンを呼び出したときの、演算コア上のMPIルーチンを処理するスレッドとアシスタントコア上のMPI非同期処理進行スレッドとの間で必要となる排他処理は、コストの高いものになります。そのため、MPIプログラムの実装容易さと性能の出やすさのバランスの観点から、“表6.1 MPI非同期処理進行スレッドの動作モード”に示す3つの動作モードを用意しています。表の「値」の欄は、MCAパラメーターopal_progress_thread_modeに指定する値を示します。

表6.1 MPI非同期処理進行スレッドの動作モード

動作モード名	値	動作モードの説明
指定区間(MPI呼び出しなし)モード	1	MPIプログラムの中で区間指定MPI非同期通信促進インターフェースによって指定した区間だけで非同期通信の促進処理を行います。その区間でMPIプログラムからMPIルーチンや拡張インターフェースを呼ぶことはできません。使用するにはMPIプログラムの修正が必要です。区間指定MPI非同期通信促進インターフェースを呼び出したときだけ、スレッド間の排他処理を行います。そのため、性能のオーバーヘッドがもっとも小さくなります。

動作モード名	値	動作モードの説明
指定区間(MPI呼び出しあり)モード	2	MPIプログラムの中で区間指定MPI非同期通信促進インターフェースによって指定した区間だけで非同期通信の促進処理を行います。その区間でMPIプログラムからMPIルーチンや拡張インターフェースを呼ぶことができます。使用するにはMPIプログラムの修正が必要です。区間指定MPI非同期通信促進インターフェースを呼び出したときと、その区間の中でMPIルーチンを呼び出したときだけ、スレッド間の排他処理を行います。そのため、その区間内でMPIルーチンの呼び出しが少なければ、性能のオーバーヘッドが比較的小さくなります。しかし、その区間の外であっても、指定区間(MPI呼び出しなし)モードと比較して、MPIルーチンの呼び出し時に若干のオーバーヘッドがあります。
自動区間モード	3	ノンブロッキング通信の未完了の要求が1つ以上存在している区間で非同期通信の促進処理を行います。その区間でMPIプログラムからMPIルーチンや拡張インターフェースを呼ぶことができます。MPIプログラムの修正は不要です。その区間は本処理系が自動的に判定し、その区間の中でMPIルーチンを呼び出したときだけ、スレッド間の排他処理を行います。そのため、MPI_ISENDルーチンなどによるノンブロッキング通信の開始からMPI_WAITルーチンなどによる完了確認までの間にMPIルーチンを多数呼んでいる場合は、オーバーヘッドが大きくなります。そうでない場合であっても、指定区間(MPI呼び出しあり)モードと比較して、MPIルーチンの呼び出し時に若干のオーバーヘッドがあります。

区間指定MPI非同期通信促進インターフェースの詳細は、“5.4 区間指定MPI非同期通信促進インターフェース”をお読みください。

本機能を用いる場合はMCAパラメーターopal_progress_thread_modeにより動作モードを指定する必要があります。詳細は“表4.42 opal_progress_thread_mode (MPI非同期処理進行スレッドの動作モードを指定)”をお読みください。

なお、本機能を“5.3 拡張持続的通信要求インターフェース”と組み合わせて使用することはできますが、組み合わせる効果はほとんどありません。

例

非同期通信が促進されるプログラムの抜粋

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

6.3 MPIライブラリ内メモリコピー処理のスレッド並列化

本処理系では、MPIライブラリ内部の処理は基本的にそのMPIルーチンを呼んだスレッドで行われますが、特定のメモリコピー処理に限り、複数のスレッドで並列化できるようにする機能を用意しています。本機能を利用するには、“表6.2 MPIライブラリ内メモリコピー処理のスレッド並列化機能の利用方法”に示すコンパイラオプション、環境変数、およびMCAパラメーターの指定が必要です。

スレッド並列化の対象となる主な処理は、派生データ型のデータを用いたpack処理とunpack処理です。pack処理とは、メモリ上で不連続な配置となっているデータを、連続な配置となるようにする処理です。unpack処理は、pack処理を行ったデータを元の不連続な配置に戻す処理です。本機能を有効にすることで、MPI_PACKルーチンやMPI_UNPACKルーチンのほか、派生データ型のデータを用いた通信ルーチンなどの処理を高速化できる場合があります。スレッド並列時のスレッド数は、MPIプログラムの実行時に指定した環境変数によって決まります。MPIプログラム内でOpenMPのomp_set_num_threadsルーチンを呼んでいる場合は、そのルーチンで設定したスレッド数で並列処理を行います。

ただし、以下のどちらかの場合は、対象の処理であってもスレッド並列化されません。

- スレッド並列化の対象となるMPIルーチンが、OpenMPのparallelリージョンから呼ばれている場合
- スレッド数やデータサイズなどの条件から、スレッド並列化しない方が速いとMPIライブラリが判断した場合

表6.2 MPIライブラリ内メモリコピー処理のスレッド並列化機能の利用方法

指定するタイミング	内容
MPIプログラムの翻訳/結合時	翻訳/結合コマンドのオプションとして、-Kparallelと-Kopenmpの両方またはどちらか1つと、-Nlibompを指定してください。 -Kparallelオプションおよび-Kopenmpオプションの詳細については、各コンパイラのマニュアルをお読みください。
MPIプログラムの実行時	・ 環境変数OMP_NUM_THREADSに、スレッド並列時のスレッド数を指定してください。環境変数によるスレッド数の指定がない場合、スレッド並列時のスレッド数はジョブで利用できるCPU数と同じになります。 ・ mpiexecコマンドのオプションとして、MCAパラメーターopal_mt_memcpyに値1を指定してください。 MCAパラメーターopal_mt_memcpyの詳細については、“表4.41 opal_mt_memcpy (MPIライブラリ内で行われる特定のメモリコピー処理をスレッド並列化)”をお読みください。

注意

本機能利用時に複数の方法でスレッド数が指定されている場合、以下の優先順位でスレッド並列時のスレッド数が決まります。

1. MPIプログラム内のomp_set_num_threadsルーチンで設定された値
2. 環境変数OMP_NUM_THREADSに指定された値

6.4 MPI規格仕様に関する注意事項

6.4.1 MPI規格への対応レベル

本処理系で提供するMPIライブラリは、MPI-3.1規格およびMPI-4.0規格の一部に準拠しています。

C++ bindingは、MPI-2.2規格の範囲でサポートされています。

Javaにおけるインターフェースは、“MPI User's Guide Additional Volume Java Interface”をお読みください。

MPI-3.1規格の範囲で定義されたルーチンのうち、Javaプログラムからの利用時にサポートされていないものを“表6.3 本処理系のJava bindingで提供しないルーチン”に示します。

MPI規格で廃止されたインターフェースは、本処理系でも今後の版数アップで廃止される可能性があります。使用しないようにしてください。
MPI規格で廃止されたインターフェースとそれらの代替インターフェースについては、MPI規格を参照してください。

表6.3 本処理系のJava bindingで提供しないルーチン

ルーチン
MPI_AINT_ADD
MPI_AINT_DIFF
MPI_ALLOC_MEM
MPI_COMM_CREATE_ERRHANDLER
MPI_COMM_JOIN
MPI_ERRHANDLER_FREE
MPI_FILE_CREATE_ERRHANDLER
MPI_FREE_MEM
MPI_GET_ADDRESS
MPI_GREQUEST_COMPLETE
MPI_GREQUEST_START
MPI_INEIGHBOR_ALLTOALLW
MPI_INFO_GET_VALUELEN
MPI_NEIGHBOR_ALLTOALLW
MPI_PACK_EXTERNAL

ルーチン
MPI_PACK_EXTERNAL_SIZE
MPI_PCONTROL
MPI_REGISTER_DATAREP
MPI_TYPE_CREATE_DARRAY
MPI_TYPE_CREATE_HINDEXED_BLOCK
MPI_TYPE_CREATE_INDEXED_BLOCK
MPI_TYPE_CREATE_SUBARRAY
MPI_TYPE_GET_CONTENTS
MPI_TYPE_GET_ENVELOPE
MPI_T_CATEGORY_CHANGED
MPI_T_CATEGORY_GET_CATEGORIES
MPI_T_CATEGORY_GET_CVARS
MPI_T_CATEGORY_GET_INDEX
MPI_T_CATEGORY_GET_INFO
MPI_T_CATEGORY_GET_NUM
MPI_T_CATEGORY_GET_PVARS
MPI_T_CVAR_GET_INDEX
MPI_T_CVAR_GET_INFO
MPI_T_CVAR_GET_NUM
MPI_T_CVAR_HANDLE_ALLOC
MPI_T_CVAR_HANDLE_FREE
MPI_T_CVAR_READ
MPI_T_CVAR_WRITE
MPI_T_ENUM_GET_INFO
MPI_T_ENUM_GET_ITEM
MPI_T_FINALIZE
MPI_T_INIT_THREAD
MPI_T_PVAR_GET_INDEX
MPI_T_PVAR_GET_INFO
MPI_T_PVAR_GET_NUM
MPI_T_PVAR_HANDLE_ALLOC
MPI_T_PVAR_HANDLE_FREE
MPI_T_PVAR_READ
MPI_T_PVAR_READRESET
MPI_T_PVAR_RESET
MPI_T_PVAR_SESSION_CREATE
MPI_T_PVAR_SESSION_FREE
MPI_T_PVAR_START
MPI_T_PVAR_STOP
MPI_T_PVAR_WRITE
MPI_UNPACK_EXTERNAL
MPI_WIN_CREATE_ERRHANDLER
MPI_WIN_SHARED_QUERY

6.4.2 本処理系で利用できるMPI定義済みデータ型

本処理系で利用できるMPI定義済みデータ型は、MPIプログラムで使用するMPIの言語bindingによって異なります。MPIの定義済みデータ型と各言語のデータ型の対応について、“表6.4 Fortran bindingで利用可能なデータ型”、“表6.5 C bindingで利用可能なデータ型”、“表6.6 C++ bindingで利用可能なデータ型”、“表6.7 Java bindingで利用可能なデータ型”に、それぞれ示します。

“表6.7 Java bindingで利用可能なデータ型”には、リダクション演算で対応するデータ型の種別を記しています。リダクション演算でのデータ型の種別についてはMPI規格を参照してください。

表6.4 Fortran bindingで利用可能なデータ型

定義済みデータ型	Fortranのデータ型
[基本データ型]	
MPI_CHARACTER	CHARACTER
MPI_LOGICAL	LOGICAL
MPI_LOGICAL1	LOGICAL(1)
MPI_LOGICAL2	LOGICAL(2)
MPI_LOGICAL4	LOGICAL(4)
MPI_LOGICAL8	LOGICAL(8)
MPI_INTEGER	INTEGER
MPI_INTEGER1	INTEGER(1)
MPI_INTEGER2	INTEGER(2)
MPI_INTEGER4	INTEGER(4)
MPI_INTEGER8	INTEGER(8)
MPI_REAL	REAL
MPI_REAL2	REAL(2)
MPI_REAL4	REAL(4)
MPI_REAL8	REAL(8)
MPI_REAL16	REAL(16)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION
MPI_COMPLEX	COMPLEX
MPI_COMPLEX4	COMPLEX(2)
MPI_COMPLEX8	COMPLEX(4)
MPI_COMPLEX16	COMPLEX(8)
MPI_COMPLEX32	COMPLEX(16)
MPI_DOUBLE_COMPLEX	COMPLEX(8)
MPI_CHAR	char (C言語)
MPI_SHORT	signed short int (C言語)
MPI_INT	signed int (C言語)
MPI_LONG	signed long int (C言語)
MPI_LONG_LONG_INT	signed long long int (C言語)
MPI_LONG_LONG	signed long long int (C言語)
MPI_SIGNED_CHAR	signed char (C言語)
MPI_UNSIGNED_CHAR	unsigned char (C言語)
MPI_UNSIGNED_SHORT	unsigned short int (C言語)
MPI_UNSIGNED	unsigned int (C言語)
MPI_UNSIGNED_LONG	unsigned long int (C言語)
MPI_UNSIGNED_LONG_LONG	unsigned long long int (C言語)
MPI_FLOAT	float (C言語)
MPI_DOUBLE	double (C言語)
MPI_LONG_DOUBLE	long double (C言語)
MPI_WCHAR	wchar_t (C言語)
MPI_BYTE	
MPI_PACKED	
MPI_C_BOOL	_Bool (C言語)
MPI_C_COMPLEX	float _Complex (C言語)
MPI_C_FLOAT_COMPLEX	float _Complex (C言語)
MPI_C_DOUBLE_COMPLEX	double _Complex (C言語)
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex (C言語)
MPI_INT8_T	int8_t (C言語)
MPI_INT16_T	int16_t (C言語)
MPI_INT32_T	int32_t (C言語)
MPI_INT64_T	int64_t (C言語)

定義済みデータ型	Fortranのデータ型
MPI_UINT8_T	uint8_t (C言語)
MPI_UINT16_T	uint16_t (C言語)
MPI_UINT32_T	uint32_t (C言語)
MPI_UINT64_T	uint64_t (C言語)
MPIX_C_FLOAT16	_Float16, __fp16 (C言語)
MPI_AINT	INTEGER (KIND=MPI_ADDRESS_KIND)
MPI_OFFSET	INTEGER (KIND=MPI_OFFSET_KIND)
MPI_COUNT	INTEGER (KIND=MPI_COUNT_KIND)
MPI_CXX_BOOL	bool (C++)
MPI_CXX_FLOAT_COMPLEX	std::complex<float> (C++)
MPI_CXX_DOUBLE_COMPLEX	std::complex<double> (C++)
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double> (C++)
[基本データ型以外] MPI_LB MPI_UB	
MPI_FLOAT_INT	float(C言語)とsigned int(C言語)のペア
MPI_DOUBLE_INT	double(C言語)とsigned int(C言語)のペア
MPI_LONG_INT	signed long int(C言語)とsigned int(C言語)のペア
MPI_2INT	signed int(C言語)のペア
MPI_SHORT_INT	signed short int(C言語)とsigned int(C言語)のペア
MPI_LONG_DOUBLE_INT	long double(C言語)とsigned int(C言語)のペア
MPI_2REAL	REALのペア
MPI_2DOUBLE_PRECISION	DOUBLE PRECISIONのペア
MPI_2INTEGER	INTEGERのペア
MPI_2COMPLEX	COMPLEXのペア
MPI_2DOUBLE_COMPLEX	DOUBLE COMPLEXのペア

表6.5 C bindingで利用可能なデータ型

定義済みデータ型	C言語のデータ型
[基本データ型]	
MPI_CHARACTER	CHARACTER (Fortran)
MPI_LOGICAL	LOGICAL (Fortran)
MPI_LOGICAL1	LOGICAL(1) (Fortran)
MPI_LOGICAL2	LOGICAL(2) (Fortran)
MPI_LOGICAL4	LOGICAL(4) (Fortran)
MPI_LOGICAL8	LOGICAL(8) (Fortran)
MPI_INTEGER	INTEGER (Fortran)
MPI_INTEGER1	INTEGER(1) (Fortran)
MPI_INTEGER2	INTEGER(2) (Fortran)
MPI_INTEGER4	INTEGER(4) (Fortran)
MPI_INTEGER8	INTEGER(8) (Fortran)
MPI_REAL	REAL (Fortran)
MPI_REAL2	REAL(2) (Fortran)
MPI_REAL4	REAL(4) (Fortran)
MPI_REAL8	REAL(8) (Fortran)
MPI_REAL16	REAL(16) (Fortran)
MPI_DOUBLE_PRECISION	DOUBLE PRECISION (Fortran)
MPI_COMPLEX	COMPLEX (Fortran)
MPI_COMPLEX4	COMPLEX(2) (Fortran)
MPI_COMPLEX8	COMPLEX(4) (Fortran)

定義済みデータ型	C言語のデータ型
MPI_COMPLEX16	COMPLEX(8) (Fortran)
MPI_COMPLEX32	COMPLEX(16) (Fortran)
MPI_DOUBLE_COMPLEX	COMPLEX(8) (Fortran)
MPI_CHAR	char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_LONG_LONG_INT	signed long long int
MPI_LONG_LONG	signed long long int
MPI_SIGNED_CHAR	signed char
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_UNSIGNED_LONG_LONG	unsigned long long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_WCHAR	wchar_t
MPI_BYTE	
MPI_PACKED	
MPI_C_BOOL	_Bool
MPI_C_COMPLEX	float _Complex
MPI_C_FLOAT_COMPLEX	float _Complex
MPI_C_DOUBLE_COMPLEX	double _Complex
MPI_C_LONG_DOUBLE_COMPLEX	long double _Complex
MPI_INT8_T	int8_t
MPI_INT16_T	int16_t
MPI_INT32_T	int32_t
MPI_INT64_T	int64_t
MPI_UINT8_T	uint8_t
MPI_UINT16_T	uint16_t
MPI_UINT32_T	uint32_t
MPI_UINT64_T	uint64_t
MPIX_C_FLOAT16	_Float16, __fp16
MPI_AINT	MPI_Aint
MPI_OFFSET	MPI_Offset
MPI_COUNT	MPI_Count
MPI_CXX_BOOL	bool (C++)
MPI_CXX_FLOAT_COMPLEX	std::complex<float> (C++)
MPI_CXX_DOUBLE_COMPLEX	std::complex<double> (C++)
MPI_CXX_LONG_DOUBLE_COMPLEX	std::complex<long double> (C++)
[基本データ型以外] MPI_LB MPI_UB	
MPI_FLOAT_INT	floatとsigned intのペア
MPI_DOUBLE_INT	doubleとsigned intのペア
MPI_LONG_INT	signed long intとsigned intのペア

定義済みデータ型	C言語のデータ型
MPI_2INT	signed intのペア
MPI_SHORT_INT	signed short intとsigned intのペア
MPI_LONG_DOUBLE_INT	long doubleとsigned intのペア
MPI_2REAL	REAL(Fortran)のペア
MPI_2DOUBLE_PRECISION	DOUBLE PRECISION(Fortran)のペア
MPI_2INTEGER	INTEGER(Fortran)のペア
MPI_2COMPLEX	COMPLEX(Fortran)のペア
MPI_2DOUBLE_COMPLEX	COMPLEX(8)(Fortran)のペア

表6.6 C++ bindingで利用可能なデータ型

定義済みデータ型	C++のデータ型
[基本データ型]	
MPI::INTEGER	INTEGER (Fortran)
MPI::INTEGER1	INTEGER(1) (Fortran)
MPI::INTEGER2	INTEGER(2) (Fortran)
MPI::INTEGER4	INTEGER(4) (Fortran)
MPI::REAL	REAL (Fortran)
MPI::REAL4	REAL(4) (Fortran)
MPI::REAL8	REAL(8) (Fortran)
MPI::DOUBLE_PRECISION	DOUBLE PRECISION (Fortran)
MPI::F_COMPLEX	COMPLEX (Fortran)
MPI::LOGICAL	LOGICAL (Fortran)
MPI::CHARACTER	CHARACTER(1) (Fortran)
MPI::CHAR	char
MPI::SHORT	signed short int
MPI::INT	signed int
MPI::LONG	signed long int
MPI::LONG_LONG	signed long long int
MPI::SIGNED_CHAR	signed char
MPI::UNSIGNED_CHAR	unsigned char
MPI::UNSIGNED_SHORT	unsigned short int
MPI::UNSIGNED	unsigned int
MPI::UNSIGNED_LONG	unsigned long int
MPI::UNSIGNED_LONG_LONG	unsigned long long int
MPI::FLOAT	float
MPI::DOUBLE	double
MPI::LONG_DOUBLE	long double
MPI::WCHAR	wchar_t
MPI::BYTE	
MPI::PACKED	
MPI::BOOL	bool
MPI::COMPLEX	Complex<float>
MPI::DOUBLE_COMPLEX	Complex<double>
MPI::LONG_DOUBLE_COMPLEX	Complex<long double>
[基本データ型以外]	
MPI::LB	
MPI::UB	
MPI::FLOAT_INT	floatとsigned intのペア
MPI::DOUBLE_INT	doubleとsigned intのペア
MPI::LONG_INT	signed long intとsigned intのペア
MPI::TWOINT	signed intのペア

定義済みデータ型	C++のデータ型
MPI::SHORT_INT	signed short intとsigned intのペア
MPI::LONG_DOUBLE_INT	long doubleとsigned intのペア
MPI::TWOREAL	REAL(Fortran)のペア
MPI::TWODOUBLE_PRECISION	DOUBLE PRECISION(Fortran)のペア
MPI::TWOINTEGER	INTEGER(Fortran)のペア

表6.7 Java bindingで利用可能なデータ型

定義済みデータ型	Javaのデータ型	データ型の種別
[基本データ型]		
MPI.CHAR	char	C言語の整数型
MPI.SHORT	short	C言語の整数型
MPI.INT	int	C言語の整数型
MPI.LONG	long	C言語の整数型
MPI.FLOAT	float	浮動小数点型
MPI.DOUBLE	double	浮動小数点型
MPI.BYTE	byte	バイト型
MPI.PACKED		
MPI.BOOLEAN	boolean	論理型
[基本データ型以外]		
MPI.FLOAT_COMPLEX	java.nio.FloatBuffer (mpi.FloatComplexクラスを使うことでjava.nio.FloatBufferのデータをfloat[]で取得できる)	
MPI.DOUBLE_COMPLEX	java.nio.DoubleBuffer (mpi.DoubleComplexクラスを使うことでjava.nio.DoubleBufferのデータをdouble[]で取得できる)	
MPI.FLOAT_INT	floatとintのペア	
MPI.DOUBLE_INT	doubleとintのペア	
MPI.LONG_INT	longとintのペア	
MPI.INT2	intのペア	
MPI.SHORT_INT	shortとintのペア	

6.4.3 予約済のコミュニケータ

本処理系では、MPI規格の仕様どおり、次のコミュニケータが予約されています。

- MPI_COMM_WORLD
- MPI_COMM_SELF

また、MPI_COMM_NULLが定義済み定数として予約されています。

6.4.4 本処理系で設定される定数値

本処理系では、MPI規格で規定された以下のFortran名前付き定数の値は .FALSE. です。

- MPI_SUBARRAYS_SUPPORTED
- MPI_ASYNC_PROTECTS_NONBLOCKING

6.4.5 マルチスレッド環境での動作

本処理系は、マルチスレッド環境での動作に対応しています。本処理系でのスレッドサポートのレベルは、MPI_THREAD_SERIALIZEDです。

MPI_THREAD_SERIALIZEDとは、複数のスレッドからMPIの呼出しが可能ですが、その場合には同時に呼び出すことはできないことを意味します。すなわち、複数のスレッドからのMPIの呼出しは、すべて逐次化されていなければなりません。このMPI呼出しの逐次化は、利用者アプリケーションプログラム内で対応する必要があります。MPI呼出しの逐次化がされていないMPIプログラムの動作については保証されませんので、ご注意ください。詳細は、MPI規格書をお読みください。

6.4.6 シグナル動作の変更

本処理系では、MPI_INITルーチン、MPI_INIT_THREADルーチン、またはMPI_T_INIT_THREADルーチンが呼び出されたときに、以下に示すシグナルそれぞれに対して、デフォルト以外のハンドラがすでに設定されていなければ本処理系独自のハンドラを設定しています。

システム標準の動作を変更しているシグナル名

- SIGABRT
- SIGBUS
- SIGFPE
- SIGSEGV
- SIGXCPU

また、本処理系では、リアルタイムシグナルを1つ使用しています。リアルタイムシグナルについての詳細は、市販されているLinux関連の書籍などを参照してください。

6.4.7 片側通信

本処理系における片側通信についての注意事項を説明します。

6.4.7.1 最適化情報

本処理系において、MPI_WIN_POSTルーチン、MPI_WIN_STARTルーチン、MPI_WIN_FENCEルーチン、MPI_WIN_LOCKルーチン、およびMPI_WIN_LOCK_ALLルーチンのassert引数は、最適化のために使用されます。本処理系において指定可能な最適化情報を、下表に示します。

表6.8 片側通信の最適化情報

MPIルーチン	指定可能な最適化情報	本処理系での動作
MPI_WIN_POST	MPI_MODE_NOCHECK	無視されます。
	MPI_MODE_NOSTORE	無視されます。
	MPI_MODE_NOPUT	無視されます。
MPI_WIN_START	MPI_MODE_NOCHECK	無視されます。
MPI_WIN_FENCE	MPI_MODE_NOSTORE	無視されます。
	MPI_MODE_NOPUT	無視されます。
	MPI_MODE_NOPRECEDE	完了待ちをするRMA操作がないことを示します。ウィンドウに関連するグループに属するすべてのプロセスで指定しなければなりません。
	MPI_MODE_NOSUCCEED	fence以降に、RMA操作がないことを示します。ウィンドウに関連するグループに属するすべてのプロセスで指定しなければなりません。
MPI_WIN_LOCK MPI_WIN_LOCK_ALL	MPI_MODE_NOCHECK	無視されます。

6.4.7.2 info引数

本処理系において、MPI_WIN_CREATEルーチン、MPI_WIN_ALLOCATEルーチン、MPI_WIN_CREATE_DYNAMICルーチン、およびMPI_WIN_ALLOCATE_SHAREDルーチンのinfo引数は、これらのルーチンで作成されたウィンドウの動作を決めるために使用されます。

本処理系において指定可能なinfoキーを下記に示します。

表6.9 MPI_WIN_CREATE、MPI_WIN_ALLOCATE、MPI_WIN_CREATE_DYNAMICにおけるinfoキー

infoキー	本処理系での動作
accumulate_ops	same_opとsame_op_no_opの動作は同じです。 省略値はsame_op_no_opです。

表6.10 MPI_WIN_ALLOCATE_SHAREDにおけるinfoキー

infoキー	本処理系での動作
alloc_shared_noncontig	trueが指定された場合は、各プロセスに近い場所にメモリを割り当てます。 falseが指定された場合は、プロセス間で隣接するようにメモリを割り当てます。 省略値はfalseです。

6.4.8 コミュニケータを共有しないグループ間での通信の確立

本処理系における、コミュニケータを共有しない2つのMPIプロセスグループ間での通信の確立についての注意事項を説明します。

本処理系において、上述の通信の確立は、1つのジョブ実行の中でバックグラウンド実行を使用して行うことができます。具体的には、バックグラウンド実行を利用して、ジョブスクリプト内に対応する複数のmpiexecコマンドの実行を指定することで可能となります。バックグラウンド実行により複数のmpiexecコマンドを実行する場合、mpiexecコマンドに-vcoordfileオプションまたは--vcoordfileオプションを指定する必要があります。

これらのオプションの詳細は、“4.1 実行コマンドの形式”をお読みください。

6.4.8.1 info引数の値

本処理系では、MPI_OPEN_PORTルーチン、MPI_COMM_ACCEPTルーチン、MPI_COMM_CONNECTルーチン、MPI_PUBLISH_NAMEルーチン、MPI_UNPUBLISH_NAMEルーチン、およびMPI_LOOKUP_NAMEルーチンの入力引数infoには、MPI_INFO_NULLを指定してください。

6.4.8.2 MPI_COMM_JOINの返却値

本処理系におけるMPI_COMM_JOINルーチンの出力は、MPI_COMM_NULLです。

6.4.8.3 MPI_PUBLISH_NAMEにおけるサービス名

本処理系におけるサービス名の文字数の上限値は、63文字(63バイト)です(C言語およびC++の場合はNULL文字を除く)。また、本処理系では、公開されるサービス名の有効範囲は、ジョブ単位です。1つのジョブ内で同じサービス名を公開しようとした場合、エラークラスMPI_ERR_SERVICEのエラーが発生することがあります。

6.4.9 動的プロセス生成

本処理系における動的プロセス生成についての注意事項を説明します。

6.4.9.1 info引数の値

本処理系によって提供されるMPI_COMM_SPAWNルーチンのinfo引数およびMPI_COMM_SPAWN_MULTIPLEルーチンのarray_of_info引数において、指定可能なキーの一覧を下表に示します。なお、MPI_COMM_SPAWN_MULTIPLEルーチンのarray_of_info引数にvcoordfileキー、num_nodesキー、またはrank_mapキーを指定する場合は、array_of_info引数の先頭の要素にキーと値を設定してください。

表6.11 動的プロセス生成機能におけるInfoキー

infoキー	値	説明
wdir	ディレクトリのパス名	動的プロセス生成実行時のカレントディレクトリを指定
vcoordfile	ファイルのパス名	動的プロセスの生成先ノードの論理座標と、各プロセスで使用するCPU(コア)数などが書かれたVCOORDファイルのパスを指定

infoキー	値	説明
num_nodes	文字列(1以上の整数)	動的プロセス生成に利用するノード数を指定
rank_map	文字列bychipまたはbynode	動的プロセスのランク割り当て規則を指定
env	文字列(NAME=VALUE)	動的プロセスに対して追加で設定する環境変数を指定
fjprof_spawn_dir_name	ディレクトリのパス名	動的プロセスに対するプロファイリングデータの出力先を指定
fjdbg_spawn_dir_name	任意の文字列	動的プロセスに対してデッドロック検出機能を使用したときの出力先として使用される識別名

6.4.9.1.1 wdirキーによるカレントディレクトリの指定

info引数にwdirキーを指定することによって、生成されるプロセスのカレントディレクトリのパス名を渡すことができます。

相対パスを指定した場合、MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンが呼び出されたときのカレントディレクトリからの相対パスとなります。

6.4.9.1.2 vcoordfileキーによるプロセスの生成先ノードおよび使用CPU(コア)数の指定

vcoordfileキーは、動的プロセスの生成先ノードおよび各プロセスが使用するCPU(コア)数を指定するために用います。値として、絶対パス・相対パスの両方を指定できます。ファイル名だけまたは相対パスを指定した場合、MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンが呼び出されたときのカレントディレクトリからの相対パスとなります。vcoordfileキーで指定するVCOORDファイルは、プロセスごとの生成先ノードと使用CPU(コア)数を記述することができるため、適切に記述されたVCOORDファイルを指定することで、動的プロセスの生成先ノード、ノード形状、ノード内プロセス数、使用CPU(コア)数、ランク配置をユーザーが指定できます。

VCOORDファイルの記述方法については"4.5 VCOORDファイルの記述形式"を参照してください。

注意

- 以下の場合、エラークラスMPI_ERR_SPAWNをエラーコードとして返却します。
  - VCOORDファイルに不正な座標が記載されている
  - VCOORDファイルが存在しない
  - MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンの引数maxprocsで指定したプロセス数が、VCOORDファイルで指定した座標数より多い
  - 1ノードあたりのプロセス生成可能数を超えている
  - CPU(コア)数指定だけの場合、必要な数のノードが存在しない
  - 1ノードあたりのCPU(コア)数を超えている
  - VCOORDファイルのフォーマットが不正
  - numanode_assign_policyの値に従った、プロセスへのCPU(コア)割り当てができない
- 以下の場合、FJMPI_ERR_SPAWN_NO_AVAILABLE_NODESをエラーコードとして返却します。エラーコードについての詳細は、"6.4.9.5.1 動的プロセス生成の失敗原因を識別するエラーコード"を参照してください。
  - VCOORDファイルに指定した座標がすでに使用中であり、CPU(コア)に空きがない

6.4.9.1.3 num_nodesキーによるノード数の指定

num_nodesキーは、動的プロセスの生成に用いるノード数を指定するために用います。

num_nodesキーを指定したとき、生成される動的プロセスのノード内プロセス数は、maxprocs ÷ num_nodes で算出されます。ここで、maxprocsは、MPI_COMM_SPAWNルーチン/MPI_COMM_SPAWN_MULTIPLEルーチンの引数として指定する動的生成プロセス数を表します。

注意

num_nodesキーによるノード数の指定では、以下の注意が必要です。

- vcoordfileキーと同時に指定された場合、num_nodesキーによるノード数の指定は無効となります。
- 使用するノードはジョブ運用ソフトウェアが自動で決定します。ジョブ実行開始時に指定したノード空間の中で効率よく割り当てる必要がある場合は、vcoordfileキーにより使用するノードを細かく指定してください。
- 指定したノード数が誤りである(ジョブで確保したノード数を超えているなど)場合、エラークラスMPI_ERR_SPAWNをエラーコードとして返却します。
- 指定したノード数が、一部ノードが使用中で確保できない場合、エラーコードFJMPI_ERR_SPAWN_NO_AVAILABLE_NODESを返却します。エラーコードについての詳細は、“6.4.9.5.1 動的プロセス生成の失敗原因を識別するエラーコード”を参照してください。

6.4.9.1.4 rank_mapキーによるプロセスのランク割り当て規則の指定

rank_mapキーは、動的プロセス生成時のランクの割り当て規則を指定するために用います。

rank_mapキーには下表に示す値が指定可能です。省略時は、bychipを指定したものとみなします。

表6.12 rank_mapキーに指定可能な値

値	説明
bychip	ランクは、同じノード内のCPU(コア)から順に設定されます。 MPI_COMM_SPAWNルーチンでmaxprocs = 8、num_nodes = 2を指定した場合は、図のような順でランクを割り当てます。
bynode	ランクは、ノードのノードID順に1つずつ(ラウンドロビン方式で)設定されます MPI_COMM_SPAWNルーチンでmaxprocs = 8、num_nodes = 2を指定した場合は、図のような順でランクを割り当てます。

図6.5 bychipを指定した場合のランク割り当て

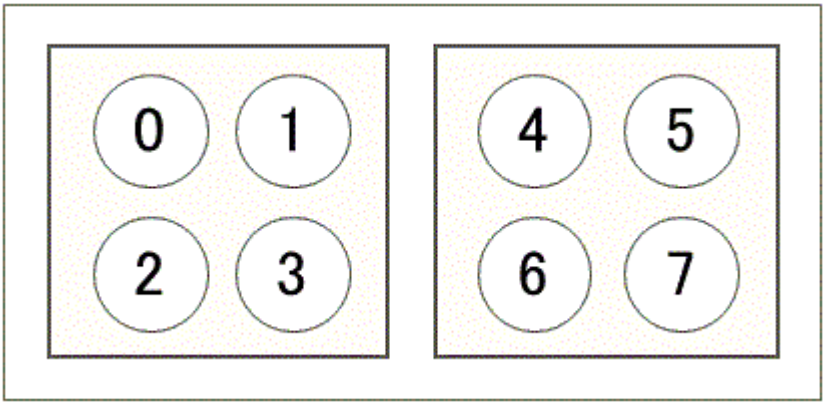
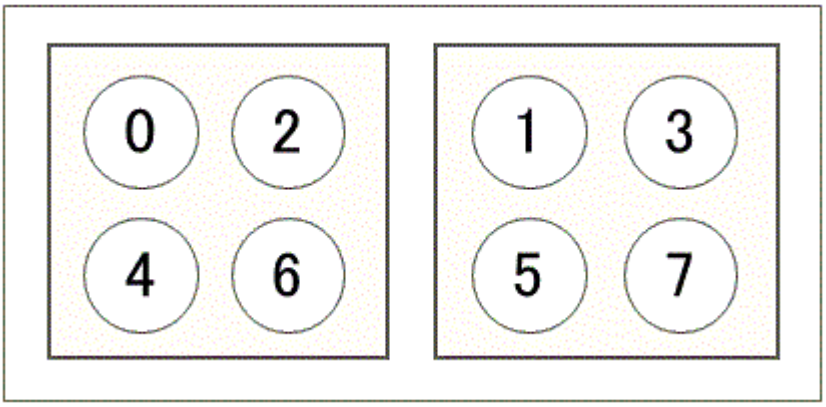


図6.6 bynodeを指定した場合のランク割り当て



注意

rank_mapキーによる、ランク割り当て規則の指定では以下の注意が必要です。

- vcoordfileキーと同時に指定された場合、rank_mapキーによるランク割り当て規則の指定は無効となります。
- num_nodesキーと同時に指定された場合、どちらの指定も有効になります。

6.4.9.1.5 envキーによる環境変数の指定

envキーは、動的プロセス生成機能によって作成されるプロセスに対して、新たに環境変数を設定したい場合に指定します。envキーに対する値の指定は、“環境変数名=環境変数に設定する値”の形式で行います。また、複数の環境変数を指定したい場合は、改行文字(¥n)を区切り文字に使用します。

6.4.9.1.6 fjprof_spawn_dir_nameキーによるプロファイリングデータの出力先の指定

fjprof_spawn_dir_nameキーは、動的プロセス生成機能によって作成されるプロセスに対し、プロファイラを用いて採取した格納場所を指定するために設定します。そのキー値には出力先ディレクトリのパスを指定します。プロファイラの詳細については、プロファイラ使用手引書を参照してください。

6.4.9.1.7 fjdbg_spawn_dir_nameキーによるデッドロック調査機能の結果出力先の指定

fjdbg_spawn_dir_nameキーは、動的プロセス生成機能によって作成されるプロセスに対し、デバッガが提供する異常終了調査機能およびデッドロック調査機能を用いて採取した結果を格納するため、識別名を出力先ディレクトリの一部として指定します。異常終了調査機能およびデッドロック調査機能の詳細については、並列実行デバッガ使用手引書を参照してください。

6.4.9.2 実行可能ファイルの検索

MPI_COMM_SPAWNルーチンのcommand引数、MPI_COMM_SPAWN_MULTIPLEルーチンのarray_of_commands引数に指定される実行可能ファイルは、以下のように検索されます。

- infoキー wdir を指定した場合

  指定したディレクトリに対して、検索を行います。

- infoキー wdir を指定しない場合

  最初に、MPI_COMM_SPAWNルーチン、MPI_COMM_SPAWN_MULTIPLEルーチンが呼び出されたときのカレントディレクトリに対して、検索を行います。次に、環境変数PATHに定義されているディレクトリに対して検索を行います。

6.4.9.3 MPI_UNIVERSE_SIZE

MPI_COMM_WORLDの定義済み属性MPI_UNIVERSE_SIZEの値は、以下によって算出されています。

```
ジョブに割り当てられたノード数 × mpiexecコマンドによって生成されたプロセスにおける1ノードあたりのプロセス数
  または
ジョブに割り当てられたノード数 × --mpi max-proc-per-node によって指定された値
```

6.4.9.4 max-proc-per-nodeの指定

動的プロセスの1ノードあたりのプロセス数が、mpiexecコマンドによって生成されたプロセスにおける1ノードあたりのプロセス数を上回る場合、ジョブ運用ソフトウェアのオプション --mpi max-proc-per-node を必ず指定してください。値には動的プロセスの1ノードあたりのプロセス数を設定します。同オプション指定時の注意事項については、“4.6 同一ノード上で複数のMPIプログラムを実行”を参照してください。

6.4.9.5 エラーコードによる動的プロセス生成の失敗原因の識別

本処理系では、MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンを用いた動的プロセス生成に失敗した場合、その失敗理由が空きノード不足によるものなのか、それ以外の理由によるものなのかをエラーコードで識別できます。

本機能とともにMPI_COMM_GET_ERRHANDLERルーチンおよびMPI_COMM_SET_ERRHANDLERルーチンを用いることで、動的プロセス生成に失敗した場合にプログラムの処理を継続するかどうか、選択することが可能となります。

6.4.9.5.1 動的プロセス生成の失敗原因を識別するエラーコード

本処理系独自のエラーコードを下表に示します。

表6.13 動的プロセス生成機能における独自エラーコード

エラーコード	エラークラス	説明
FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES	MPI_ERR_SPAWN	空きノード不足により動的プロセス生成に失敗

6.4.9.5.2 利用例

エラーコードFJMPI_ERR_SPAWN_NO_AVAILABLE_NODESを利用して、ノードが空き次第、次の動的プロセス生成が行われるプログラムの概要を以下に示します。

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

    MPI_Comm_get_errhandler(…); /* 元のエラーハンドラを取得 */
    MPI_Comm_set_errhandler(…); /* エラーハンドラを切り替え */
    while (1) {
        while (1) {
            ret = MPI_Comm_spawn(…, &inter_comm, …);
            if (MPI_SUCCESS == ret) {
                break;
            } else if (FJMPI_ERR_SPAWN_NO_AVAILABLE_NODES != ret) {
                /* ノード不足以外で失敗した場合は，異常終了 */
                MPI_Abort(…);
            }
        }
        MPI_Comm_disconnect(&inter_comm);
        if (…) {
            break;
        }
    }
    MPI_Comm_set_errhandler(…); /* エラーハンドラを元に戻す */
```

6.4.9.6 注意事項

- 本処理系では、以下の場合、動的プロセス生成を行えません。
  - mpiexecコマンドの1回の実行において、MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチンの呼出し回数の合計が4294967295回を超える場合

    動的プロセス生成を行うと、ジョブ運用ソフトウェアによってエラーメッセージが出力された後にMPIプログラムが異常終了します。

ー 同時に存在する動的プロセスのMPI_COMM_WORLDの数が65535個を超える場合

動的プロセス生成を行うと、ジョブ運用ソフトウェアによってエラーメッセージが出力された後にMPIプログラムが異常終了します。

ー 生成するプロセスがMPIプログラム以外の場合

動的プロセス生成の対象にMPIプログラム以外を指定した場合の動作は保証されません。

ジョブ運用ソフトウェアについての詳細は、ジョブ運用ソフトウェアのマニュアルをお読みください。

- 同一プロセスが動的プロセス生成を繰り返し実行した場合、生成されたプロセス生成情報およびコミュニケータ情報がメモリに蓄積されていくため、メモリ不足を引き起こすことがあります。ご注意ください。

6.4.9.7 Javaプログラムからの動的プロセス生成

JavaプログラムでMPI_COMM_SPAWNルーチンおよびMPI_COMM_SPAWN_MULTIPLEルーチンを呼び出す場合、起動するJavaクラスファイルが存在するクラスパスは、-classpathオプションではなく環境変数CLASSPATHで設定する必要があります。

Javaプログラムからの実行時の指定例とプログラム例を以下に示します。“*製品インストールパス*”は、システム管理者にお問い合わせください。

例

実行時の指定例

```
CLASSPATH=/home/user1/bin:$CLASSPATH
export CLASSPATH
PATH=/製品インストールパス/bin:$PATH
export PATH
LD_LIBRARY_PATH=/製品インストールパス/lib64:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH

mpiexec -n 2 java Spawn
```

Javaプログラムの例

```
import mpi.*;

public class Spawn {
    private final static String CMD_ARGV1 = "THIS IS ARGV 1"; /* 子プロセスに渡す引数1 */
    private final static String CMD_ARGV2 = "THIS IS ARGV 2"; /* 子プロセスに渡す引数2 */
    private final static String CMD = "Spawn"; /* Spawn対象のクラスファイル名 */

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

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

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

        Intercomm parent;
        parent = Intercomm.getParent();

        if (!parent.isNull()) {
           /* 子プロセスの処理 */
           System.out.println("I am a child");
        }else{
           /* 親プロセスの処理 : 自分自身をSpawnする */
```

```
            MPI.COMM_WORLD.spawn("java", spawn_argv, count, MPI.INFO_NULL, 0, errcode);
            System.out.println("I am a parent");
        }

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

6.4.10 カルテシアン・トポロジーによるランク並べ替え

本処理系では、MPI_CART_CREATEルーチンによって、カルテシアン・トポロジーを付加した新しいコミュニケータを生成する場合、引数reorderにtrueを指定することで、その生成したコミュニケータに属する並列プロセスに対して、カルテシアン座標に基づいた順番で新しいランクを割り振ることができます。すなわち、カルテシアン座標に基づいたランク並べ替えを行うことができます。

以降、カルテシアン座標に基づくランク並べ替えが行われるための条件およびランク並べ替えの規則について説明します。

6.4.10.1 ランク並べ替えが可能な条件

MPI_CART_CREATEルーチンによって、ランク並べ替えを行うには、いくつかの条件を満たす必要があります。ここでは、ランク並べ替えを行うための条件について説明します。

なお、ランク並べ替えは、ジョブ運用ソフトウェアによって各並列プロセスに割り当てられたノード空間の座標に基づいて行われます。本書では、各並列プロセスに対応するこの座標を、カルテシアン座標とは区別する意味で、「論理座標」または単に「座標」と呼びます。

MPI_CART_CREATEルーチンによるランク並べ替えは、以下のすべての条件を満たす場合に行われます。

- reorder引数にtrueが指定される
- 入力コミュニケータcomm_oldがもつプロセス形状、すなわち次元数および各次元のプロセス数が、新しく付加しようとするカルテシアン・トポロジーの形状と一致する
- 入力コミュニケータcomm_oldがもつプロセス形状において、論理座標の重複またはすき間のどちらもない

なお、本処理系では、ランク並べ替えを行う場合の形状の一致について、MPI_CART_CREATEルーチンの引数dims配列の先頭から順番に論理座標のX軸、Y軸、Z軸に対応させています。

6.4.10.2 ランク並べ替えの規則

MPI_CART_CREATEルーチンによってカルテシアン・トポロジーを生成する場合、カルテシアン座標は、入力コミュニケータcomm_oldに属する並列プロセスに対して、そのプロセス形状の論理座標に基づく以下の順番に従って、0から始まる昇順のランクが割り振られます。

- 形状が1次元(X軸)の場合、論理座標の原点(0)に最も近い座標にあるプロセスをランク0とし、そこを起点にX軸に沿って遠ざかる順番
- 形状が2次元(X軸Y軸)の場合、論理座標の原点(0,0)に最も近い座標にあるプロセスをランク0とし、そこを起点にY軸→X軸の順に遠ざかる順番
- 形状が3次元(X軸Y軸Z軸)の場合、論理座標の原点(0,0,0)に最も近い座標にあるプロセスをランク0とし、そこを起点にZ軸→Y軸→X軸の順に遠ざかる順番

6.4.10.3 ランク並べ替えの確認方法

本処理系では、ランク並べ替えが行われたかどうかを確認するための拡張インターフェースFJMPI_TOPOLOGY_CART_REORDERルーチンを用意しています。拡張インターフェースFJMPI_TOPOLOGY_CART_REORDERルーチンの詳細は、“5.1 ランク問合せインターフェース”をお読みください。

6.4.10.4 サンプルプログラム

以下のMPIプログラムは、次の条件のコミュニケータに対して、MPI_Cart_create関数を実行し、ランク並べ替えが実際に行われたかどうかを確認するプログラムの例です。

- 次元:3次元

・ ノード形状(X:2,Y:3,Z:4)

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

#define FAILURE 1

void error_handler(MPI_Comm *comm, int *error_code, ...)
{
    if (*error_code == MPI_ERR_DIMS) {
        fprintf(stderr, "Abort MPI_ERR_DIMS: MPI_COMM_SIZE < 24¥n");
        MPI_Finalize();
    } else {
        fprintf(stderr, "Abort ERROR: [%d]¥n", *error_code);
        MPI_Abort(MPI_COMM_WORLD, FAILURE);
    }
}

int main(int argc, char *argv[])
{
    MPI_Comm cart_comm_on;
    MPI_Comm cart_comm_off;
    MPI_Errhandler ehhdl;
    int size, rank;
    int dims_on[3] = {2, 3, 4};    /* 3次元2x3x4 並べ替えられる */
    int dims_off[3] = {3, 4, 2};   /* 3次元3x4x2 並べ替えられない */
    int periods[3] = {1, 1, 1};    /* cyclic 固定 */
    int cart_result;

    MPI_Init(&argc, &argv);

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

    fprintf(stderr, "MPI_COMM_WORLD's MPI_Comm_size : %d¥n", size);
    fprintf(stderr, "rank of MPI_COMM_WORLD = %d¥n", rank);

    MPI_Errhandler_create(error_handler, &ehhdl);

    MPI_Errhandler_set(MPI_COMM_WORLD, ehhdl);

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

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

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

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

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

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

```
    }

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

    MPI_Errhandler_free(&ehhdl);

    MPI_Finalize();

    return 0;

}
```

最初のMPI_Cart_create関数の呼出しによってランクの並べ替えが行われますが、2番目のMPI_Cart_create関数の呼出しではランクの並べ替えが行われません。2番目のMPI_Cart_create関数の呼出しでは、引数に指定したカルテシアン・トポロジーの形状が、MPIプログラムを実行したときに指定したプロセス形状と一致しないためです。

実際、プログラムでは、それぞれのMPI_Cart_create関数を呼び出した後で、FJMPI_Topology_cart_reorder関数を使って、実際にランクの並べ替えが行われたかどうかを判定しています。

FJMPI_Topology_cart_reorder関数の第2引数(プログラムでは変数cart_result)に判定結果が返ります。この判定結果の値が1の場合、ランク並べ替えが行われていることを示します。判定結果の値が0の場合、ランク並べ替えが行われていないことを示します。

6.4.11 送信バッファおよび受信バッファの注意事項

MPIプログラムの並列プロセスが書込みできないメモリ領域を、送信バッファまたは受信バッファに指定できません。送信バッファにそのようなメモリ領域を指定した場合、動作は保証されません。

例えば、MPIプログラムの並列プロセスが書込みできないメモリ領域には、MPIプログラムの命令領域(.textセクション)などがあります。

6.4.12 MPIの入出力

本処理系におけるMPIの入出力の動作については、ROMIOの実装に準じています。

ROMIOに関する情報は、http://www.mcs.anl.gov/projects/romio/ から入手できます。また、下記の制約があります。

- 1回の入出力において扱うことができるファイル入出力データサイズは、「2GiB － 64KiB」以下となります。データサイズは、「1要素のデータ型サイズ × 要素数」です。

本処理系において、MPIの入出力として扱えるファイルシステムは、LLIO、FEFS、およびNFSです。ただし、FEFS固有の機能は動作しません。その他のファイルシステムはサポートしていません。また、NFSはデータ更新遅れによる内容の不整合を防止するためmountオプションにnoacを指定してください。LLIOとFEFSの詳細は、LLIOとFEFSのマニュアルをお読みください。

本処理系ではデータ表現としてnativeだけをサポートします。

本処理系で提供するMPI_FILE_OPENルーチンは、既存ファイルのサイズを変更しません。

本処理系では、MPIの入出力ルーチンで利用者の入出力ファイルと同じディレクトリにテンポラリファイルを作成することがあります。これは、MPI_FILE_OPENルーチンによってファイルをオープンしたときに生成されるものであり、1つのサイズは8バイト程度です。このテンポラリファイルは、通常、MPI_FILE_CLOSEルーチンで削除されますが、プログラムの強制終了が行われた場合、システムに異常が発生した場合などにおいて、残ることがあります。

MPIプログラムの実行が終了した後で以下のような名前のファイルが存在していた場合、手動でファイルを削除してください。

```
.[MPI I/Oのファイル名].shfp.数字
```

MPIの集団的な入出力ルーチン(MPI_FILE_READ_ALLルーチンなど)のstatusには、常に呼出し時の引数情報に基づいた情報が設定されます。

本処理系におけるMPIの入出力で使用可能なInfoオブジェクトのkeyとvalueを下表に示します。

表6.14 MPIの入出力で使用可能なInfoオブジェクトのkeyとvalue

Key	valueの省略値	意味
cb_buffer_size	16777216	集団的アクセスに使用する一時バッファ域の大きさを指定します。

Key	valueの省略値	意味
cb_nodes	入出力を行うコミュニケータに割り当てられたホスト数	集団的アクセスで実際に入出力を行うプロセス数を指定します。
ind_rd_buffer_size	4194304	プロセス個別読込み時のバッファ域の大きさを指定します。
ind_wr_buffer_size	524288	プロセス個別書込み時のバッファ域の大きさを指定します。
romio_ds_write	disable	Data sievingをwrite時に行うかどうかを指定します。 行う場合は“enable”または“ENABLE”、行わない場合は“disable”または“DISABLE”、MPIライブラリに判断させる場合は“automatic”または“AUTOMATIC”を指定してください。
romio_ds_read	automatic	Data sievingをread時に行うかどうかを指定します。 行う場合は“enable”または“ENABLE”、行わない場合は“disable”または“DISABLE”、MPIライブラリに判断させる場合は“automatic”または“AUTOMATIC”を指定してください。

上記に加え、ファイルシステムがLLIOの場合には、下表のkeyを指定できます。

表6.15 LLIOの場合に使用可能なInfoオブジェクトのkeyとvalue

Key	valueの省略値	意味
direct_read	false	ダイレクトI/O(read)を行うかどうかを指定します。 ダイレクトI/O(read)を使用する場合は“true”または“TRUE”を指定してください。
direct_write	false	ダイレクトI/O(write)を行うかどうかを指定します。 ダイレクトI/O(write)を使用する場合は“true”または“TRUE”を指定してください。
bypass_cn_file_cache	automatic	計算ノード内キャッシュをバイパスしてI/Oを行うかどうかを指定します。 バイパスする場合は“enable”または“ENABLE”、バイパスしない場合は“disable”または“DISABLE”、MPIライブラリに判断させる場合は“automatic”または“AUTOMATIC”を指定してください。
romio_llio_ds_in_coll	disable	集団的I/Oに対しData sievingの適用をするかどうかを指定します。 このKeyの設定値はromio_ds_writeやromio_ds_readに優先して集団的I/Oに対し適用されます。 適用する場合は“enable”または“ENABLE”、適用しない場合は“disable”または“DISABLE”を指定してください。

6.4.13 プロファイリングインターフェースの利用

- C言語のプログラムにおいてプロファイリングインターフェースを利用する場合、C言語のインターフェースを用いてフックできます。
- Fortranプログラムにおいてプロファイリングインターフェースを利用する場合、Fortranのインターフェースを用いてフックできます。
- C++プログラムにおいてプロファイリングインターフェースを利用する場合、C言語のインターフェースを用いてフックできます。

6.4.14 MPIツール情報インターフェース

本処理系では、MPIツール情報インターフェースはサポートされていますが、制御変数と性能変数は公開されていません。

6.4.15 マクロで実装されているルーチン

以下に示すルーチンは、C言語のmpi.hにマクロとして定義されています。

- MPI_AINT_ADD

- MPI_AINT_DIFF
- PMPI_AINT_ADD
- PMPI_AINT_DIFF

6.4.16 ユーザー定義エラー処理ルーチンの引数

MPIプログラムの実行中にMPIライブラリがエラーを検出すると、通常はMPIライブラリ内部で定義されたエラー処理ルーチンが呼ばれます。このとき呼ばれるエラー処理ルーチンは、ユーザーが定義したルーチンに変更することもできます。C言語の場合、ユーザー定義エラー処理ルーチンの書式は、以下のように可変個引数をとるものとして定義されています。

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

MPI規格では、エラー処理ルーチンの第1、第2引数以外の引数はMPIライブラリの実装依存とされています。本処理系では、エラーを検出したMPIルーチン名を表すconst char *型の文字列が、エラー処理ルーチンの第3引数に渡されます。第4引数以降に渡される値はありません。

6.4.17 グループ間コミュニケーションにおける集団通信

本処理系におけるグループ間コミュニケータでの集団通信の注意事項を説明します。

本項では、rootにMPI_ROOTを指定しているプロセスをrootプロセスと呼びます。また、rootプロセスがあるグループを第1グループ、そうでないグループを第2グループと呼びます。

要素数

下表に示す条件に該当する場合、本処理系では動作を保証しません。

ルーチン名	動作保証しない条件
MPI_Allgather	第2グループのサイズと第2グループのrank 0で指定された第2引数sendcountの積、または、第1グループのサイズと第2グループのrank 0で指定された第5引数recvcountの積が、2147483647を超える場合
MPI_Allgatherv	あるグループのランクが指定した第2引数sendcountの総和が、2147483647を超える場合
MPI_Gather	第2グループのサイズと第2グループのrank 0で指定された第2引数sendcountの積、または、第1グループのサイズと第2グループのrank 0で指定された第5引数recvcountの積が、2147483647を超える場合
MPI_Gatherv	第2グループに含まれるランクが指定した第2引数sendcountの総和が、2147483647を超える場合
MPI_Reduce_scatter	あるグループのランクが指定した第3引数recvcountsの要素の総和が、2147483647を超える場合
MPI_Reduce_scatter_block	あるランクにおける第3引数recvcountとそのランクの属するグループのサイズの積が、2147483647を超える場合
MPI_Scatter	第2グループのサイズと第1グループのrootプロセスでの第2引数sendcountの積、または、第2グループのサイズと第2グループのrank 0で指定された第5引数recvcountの積が、2147483647を超える場合
MPI_Scatterv	第2グループに含まれるあるランクが指定した第2引数sendcountsの要素の総和が、2147483647を超える場合
MPI_Iallgather MPIX_Allgather_init	あるランクが指定した第5引数recvcountと、そのランクが属していない方のグループのサイズから1引いた値の積が、2147483647を超える場合
MPI_Ialltoall MPIX_Alltoall_init	あるランクで指定された第2引数sendcountと、そのランクが属していない方のグループのサイズから1引いた値の積、または、あるランクで指定された第5引数recvcountと、そ

ルーチン名	動作保証しない条件
	のランクが属していない方のグループのサイズから1引いた値の積が、2147483647を超える場合
MPI_Igather MPIX_Gather_init	第1グループのrootプロセスで指定された第5引数recvcountと第2グループのサイズから1引いた値の積が、2147483647を超える場合
MPI_Ineighbor_allgather MPIX_Neighbor_allgather_init	あるランクの入次数から1引いた値とそのランクの第5引数recvcountの積が、2147483647を超える場合
MPI_Ineighbor_alltoall MPIX_Neighbor_alltoall_init	あるランクの出次数から1引いた値とそのランクの第2引数sendcountの積、または、あるランクの入次数から1引いた値とそのランクの第5引数recvcountの積が、2147483647を超える場合
MPI_Ireduce_scatter MPIX_Reduce_scatter_init	あるランクにおける第5引数recvcountsの要素の総和が、2147483647を超える場合
MPI_Ireduce_scatter_block MPIX_Reduce_scatter_block_init	あるランクにおける第5引数recvcountとそのランクが属するローカルコミュニケータのサイズの積が、2147483647を超える場合
MPI_Iscatter MPIX_Scatter_init	第1グループのrootプロセスで指定された第2引数sendcountと第2グループのサイズから1引いた値の積が、2147483647を超える場合

データ型

下表に示す条件に該当する場合、本処理系では動作を保証しません。

ルーチン名	動作保証しない条件
MPI_Allgatherv	同一グループのランク間のデータ型のサイズが異なる場合
MPI_Gatherv	同一グループのランク間のデータ型のサイズが異なる場合
MPI_Scatterv	同一グループのランク間のデータ型のサイズが異なる場合

6.5 Eager通信方式とRendezvous通信方式

本処理系では、メッセージの通信方式を2つ実装しています。

Eager通信方式

Eager通信方式は、送信側が受信側の状態にかかわらずメッセージを送信する、いわゆる非同期型の通信方式です。

Eager通信方式では、利用者プログラムの受信先メモリ領域の情報なしで送信側プロセスがメッセージを送信します。そのため、そのメッセージのサイズ以上の大きさのバッファ領域をMPIライブラリ内部にあらかじめ用意します。Eager通信方式では、送信側と受信側との連携処理のオーバーヘッドはありませんが、内部バッファ領域と利用者プログラムのメモリ空間の間でのコピーのオーバーヘッドがあります。

Rendezvous通信方式

Rendezvous通信方式は、送信側と受信側とで連携処理を行い、送信側が受信側のメッセージ格納先が確定するまでメッセージを送信しない、いわゆる同期型の通信方式です。

Rendezvous通信方式では、送信側と受信側とで事前に連携処理が行われ、利用者プログラムの受信先メモリ領域が確定してから送信側プロセスがメッセージを送信します。そのため、サイズの大きいメッセージを送信する場合でも、大きな内部バッファ領域を必要としません。特に、メッセージが連続データの場合は、内部バッファ領域を使用せずに、利用者プログラムの送信側と受信側のメモリ空間の間で直接にコピーできます。

本処理系では、実際、サイズの小さいメッセージの送信にはEager通信方式を選択し、サイズの大きいメッセージの通信にはRendezvous通信方式を選択するよう、送信するメッセージのサイズによって内部的に通信方式を切り替えています。Tofu通信の場合、厳密には、メッセージのサイズだけでなく、メッセージ通信を行う距離(ホップ数)についても考慮しています。本システム内の計算ノードは物理的にメッシュ状またはトーラス状に接続されており、ある2つの計算ノード間のメッセージ通信は、必要に応じて別の複数の計算ノードを経由して行わ

れます。この経由する計算ノードの数に1を加えた数がホップ数です。本処理系では、Tofu通信の高速型通信モードにおけるEager通信方式とRendezvous通信方式を切り替えるときの“しきい値”(バイト数)について、次の式で決めています。

```
しきい値 ＝ 38600 ＋ ホップ数 × 296
```

省メモリ型通信モードにおける“しきい値”については、メモリ使用量が少なくなるように、本処理系によって適切な値が自動的に設定されます。高速型通信モードと省メモリ型通信モードの詳細は、“6.10 メモリ使用量を抑えるための考慮”をお読みください。

メッセージのサイズに関係なく、次のような場合にもRendezvous通信方式の方が高速になることがあります。

- ノンブロッキング通信により複数の通信を同時に行うようなMPIプログラム
- 受信ルーチン(MPI_RECVルーチンなど)が送信ルーチン(MPI_SENDルーチンなど)よりも早く実行されるようなMPIプログラム

そのような場合、この“しきい値”を変更することで、MPIプログラムの性能向上が期待できることがあります。“しきい値”は、MCAパラメーターbtl_tofu_eager_limitによって変更できます。MCAパラメーターbtl_tofu_eager_limitについての詳細は、“表4.6 btl_tofu_eager_limit (通信方式を切り替えるしきい値を変更)”をお読みください。

ノード内通信についても、Eager通信方式とRendezvous通信方式を切り替えるときの“しきい値”が存在しますが、Tofu通信向けの“しきい値”とは値が異なり、32768に固定されています。この“しきい値”はユーザーが変更することはできません。

6.6 Stride RDMA通信

一般的な派生データ型のメッセージ通信では、内部的に複数のメッセージに分割しEager通信方式のSend通信によってそれらのメッセージをパイプライン的に送信しています。すなわち、内部的なバッファ領域を確保し、分割して切り出したメッセージをそのバッファ領域に一旦コピーすることになるため、サイズの大きいメッセージの通信では、その通信時間は非常に長くなります。

本処理系では、パイプライン的に処理するEager通信方式のSend通信の代わりにTofuインターコネクトのRDMA通信機能を使用することで、内部的なバッファ領域を使わずに利用者プログラムのデータ領域から直接にコピーするように改善しています。本処理系では、この仕組みをStride RDMA通信と呼びます。Stride RDMA通信は、派生型データの構成が比較的単純であり、全体のメッセージのサイズが大きい場合には、通信性能の向上が期待できます。
具体的には、以下の条件をすべて満たす1対1通信について、Stride RDMA通信が有効となります。

- 送信側と受信側のデータ型が同じである
- メッセージがメモリ上で連続していない
- 異なる計算ノード間の通信である
- 送信するメッセージのサイズ(バイト数)の合計が通信方式を切り替えるときの“しきい値”以上である

通信方式を切り替えるときの“しきい値”の詳細は、“6.5 Eager通信方式とRendezvous通信方式”をお読みください。

MCAパラメーターpml_ob1_use_stride_rdmaに0を指定することにより、Stride RDMA通信の抑止もできます。MCAパラメーターpml_ob1_use_stride_rdmaの詳細は、“表4.47 pml_ob1_use_stride_rdma (Stride RDMA通信を使用)”をお読みください。

例

Stride RDMA通信が適用される場合のプログラムの抜粋

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

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

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

```
    MPI_Send(sendbuf, 1, newtype, 0, tag, MPI_COMM_WORLD);
}
```

6.6.1 Stride RDMA通信に関する注意事項

連続していることが保証されないメモリ領域を派生データ型の通信で使用すると、Stride RDMA通信では以下のようなエラーメッセージを出して異常終了することがあります。

- “[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag.”を含むメッセージ

この場合は、MCAパラメーターpml_ob1_use_stride_rdmaに0を指定してStride RDMA通信を抑止してください。ただし、通信性能が低下することがあります。ご注意ください。

MCAパラメーターpml_ob1_use_stride_rdmaの詳細は、“表4.47 pml_ob1_use_stride_rdma (Stride RDMA通信を使用)”をお読みください。

6.7 複数TNIの利用

本システムはTofuインターコネクトによって接続されており、各計算ノードにはTNI(Tofu Network Interface)と呼ばれるネットワークインターフェース装置が配備されています。本システムでは各計算ノードに6つのTNIがあり、各計算ノードに割り当てられたMPIプロセスは、この複数のTNIを使用することで、効率よいメッセージの送受信が可能となります。同時に通信する数が少ないMPIプログラムでは、使用するTNIの数を少なくすることで、サイズの小さいメッセージの通信時間が短くなることがあります。TNIの使用数の上限は、MCAパラメーターcommon_tofu_max_tnisによって変更できます。MCAパラメーターcommon_tofu_max_tnisの詳細は、“表4.21 common_tofu_max_tnis (TNIの使用数の上限値を変更)”をお読みください。

また、複数のTNIを使用することで、1対1通信のスループット性能を向上させることもできます。すなわち、1対1通信において、複数のTNIを使用することで結果的に複数の通信経路を使う機会が増え、大きなデータサイズのメッセージでも、それら複数の通信経路を利用してメッセージを分割して効率よく通信できる可能性が広がるためです。本処理系では、これをトランキングと呼びます。このトランキングは、MCAパラメーターcommon_tofu_use_multi_pathに値1を指定することで利用できます。MCAパラメーターcommon_tofu_use_multi_pathの使用方法については、“表4.30 common_tofu_use_multi_path (複数の通信経路を利用した1対1通信を行う)”をお読みください。

上述のとおり、トランキングの利用により、通信性能のスループット向上が期待できることがあります。しかし、一方で、複数の通信相手が存在する場合でも、1つの通信相手に複数のTNIを占有してしまう可能性があり、異なる通信相手とのメッセージ通信を行う場合の全体のスループット性能に影響を与えることもあります。さらに、利用しようとする通信経路が他の通信処理によってすでに使用されている場合には、通信の衝突が発生し、逆に全体の通信性能が悪くなることもあります。このように、アプリケーションプログラムまたはその他の通信環境の状況によっては、トランキングの効果が得られないこともあります。ご利用にあたっては、十分な注意が必要です。

トランキングにおいて利用できるTNIの数は、同一ノード内で実行するプロセス数またはMCAパラメーター common_tofu_max_tnis の指定によって変動します。例えば、common_tofu_max_tnis に値2が指定される場合、MPIプロセスでは最大2個のTNIを利用するようになります。この場合、トランキングにおいて、使用するTNI数も最大2個までとなります。

6.8 集団通信におけるリダクション演算の順序保証

本処理系では、リダクション演算を行う集団通信MPI_REDUCEルーチン、MPI_IREDUCEルーチン、MPI_ALLREDUCEルーチン、MPI_IALLREDUCEルーチン、MPI_REDUCE_SCATTERルーチン、MPI_IREDUCE_SCATTERルーチン、MPI_REDUCE_SCATTER_BLOCKルーチン、MPI_IREDUCE_SCATTER_BLOCKルーチン、およびMPI_SCANルーチンにおいて、コミュニケータのサイズ、メッセージのサイズ、ランク配置の形状などの通信条件に応じて、集団通信の実行時間が最適になるように内部的に演算順序を変えることがあります。浮動小数点数データの場合には、リダクション演算の順序が変わることで、結果的に計算結果の精度に影響を与えることもあります。

本処理系では、このようなリダクション演算による計算結果への影響を抑えるため、リダクション演算の順序を固定化し保証する機能を用意しています。具体的には、MCAパラメーターcoll_base_reduce_commute_safeに1を指定することで、リダクション演算の順序を常に固定化し保証できます。しかし、この機能を利用する場合には、リダクション演算の順序が固定化されるため、集団通信の実行時間が長くなることにご注意ください。MCAパラメーターcoll_base_reduce_commute_safeの詳細は、“表4.7 coll_base_reduce_commute_safe (リダクション演算の順序を保証)”をお読みください。

通常、MCAパラメーターcoll_base_reduce_commute_safeの省略値は0に設定されています。リダクション演算の順序が変わることを気にする必要がない限り、なるべく省略時の設定でご利用することをお薦めします。

なお、MPIプログラムの中で、MPI_OP_CREATEルーチンを使うことで本機能と同等の効果を得ることもできます。具体的には、MPI_OP_CREATEルーチンの引数commuteにfalseを指定することで可換でない新たな演算を定義し、その新しい演算によってリダクション演算を行うことで本機能と同等の効果が得られます。

また、リダクション演算の順序を明示的に指定したい場合は、演算に用いるデータを特定のランクに集めて、MPI_REDUCE_LOCALルーチンを用いて、順序を指定しながら実行してください。

6.9 MPIプログラム内からのプロセス生成について

本処理系では、システムコール(forkなど)、ライブラリ関数(systemなど)、またはFortran処理系のサービスルーチン(FORKなど)などを使用して、MPIプログラムから子プロセスの生成を行う場合、以下のような制約があります。

- プロセス生成のタイミングで、MPI通信の送受信バッファなどとして使用中のメモリ領域が存在する場合は、そのメモリ領域を含むページは子プロセスに引き継がれないことがあります。そのため、子プロセスはそのメモリ領域を含むページにアクセスできないことがあります。

使用中かどうかは以下の基準で判定されます。

- 1対1通信、集団通信、隣接集団通信のブロッキング通信の送受信バッファでは、そのルーチンを呼び出してからそのルーチンが復帰するまでが使用中となります。
- 1対1通信、集団通信、隣接集団通信のノンブロッキング通信の送受信バッファでは、MPI_ISENDルーチンなどで通信を開始してからMPI_WAITルーチンなどで通信の完了を確認するまでが使用中となります。
- 1対1通信、集団通信、隣接集団通信の持続的通信要求の送受信バッファでは、MPI_SEND_INITルーチンなどで要求を作成してからMPI_REQUEST_FREEルーチンで要求を解放するまでが使用中となります。ただし、要求を解放する前にMPI_WAITルーチンなどで個々の通信の完了を確認している必要があります。
- MPI_WIN_CREATE_DYNAMICルーチン以外で作成した片側通信のウィンドウでは、MPI_WIN_CREATEルーチンなどでウィンドウを作成してからMPI_WIN_FREEルーチンでウィンドウを解放するまでが使用中となります。
- MPI_WIN_ATTACHルーチンでアタッチしたメモリ領域は、MPI_WIN_ATTACHルーチンでアタッチしてからMPI_WIN_DETACHルーチンでデタッチするまでまたはMPI_WIN_FREEルーチンでウィンドウを解放するまでが使用中となります。
- 片側通信のRMA通信のoriginバッファでは、通信を開始してからMPI_WIN_FENCE、MPI_WIN_FLUSH、MPI_WAITなどのルーチンで通信のローカルでの完了を確認するまでが使用中となります。
- MPI_ALLOC_MEMルーチンで確保したメモリは、MPI_ALLOC_MEMルーチンでメモリを確保してからMPI_FREE_MEMルーチンで解放するまでが使用中となります。

この制約のため、例えば、forkシステムコールによって生成された子プロセスが、execveシステムコールや_exitシステムコールを呼ぶまでの間に該当するメモリ領域に対してアクセスを行うと、子プロセスにおいてsegmentation faultのエラーが発生することがあります。

アクセスの可否はOSの管理するページの単位で判定されるため、あるメモリ領域が使用中の場合にその周辺のアドレスのメモリ領域にもアクセスできないこともありますので、ご注意ください。

また、生成された子プロセスのメモリ領域が、MPIライブラリ内部で通信バッファとして使用中のメモリ領域と同じページ上に割り当たることがあります。この場合は、前述の判定基準で使用中と判定されなくても、子プロセスにおいてsegmentation faultのエラーが発生することがあります。これを発生しにくくするためには、MCAパラメーターcommon_tofu_use_memory_poolに1を指定してください。ただし、このオプションを指定するとメモリ使用量が大きくなる場合がありますので、ご注意ください。MCAパラメーターcommon_tofu_use_memory_poolについては、“表4.31 common_tofu_use_memory_pool (MPIライブラリ用のメモリプールを使用)”を参照してください。

6.10 メモリ使用量を抑えるための考慮

本処理系では、MPIプログラムが実行されると、各並列プロセス内で受信用のバッファなどMPIライブラリ自身が必要とするメモリが内部的に確保されます。これらのメモリは、ある1つの並列プロセスに着目すると、通信相手となるプロセスごとに確保される必要があります。本処理系では、不要なメモリの確保を避けるために、通信相手となる各プロセスと初めて通信を行う時点でメモリを確保するようにしています。本書では、この方式を動的コネクションと呼びます。この動的コネクションによる方式を採用することで、メモリ使用量をある程度まで抑えることができるようになっています。しかし、動的コネクションの方式を採用しても、MPI_INITルーチンが実行された直後のメモリ使用量は少ないものの、MPIプログラムの実行が進み、通信相手となるプロセスの数が増加すれば、結果的にメモリ使用量も多くなっていきます。

本節では、本処理系が使用するメモリの使用量を可能な限り抑えるための方法について説明しています。

6.10.1 高速型通信モードと省メモリ型通信モードの切り替え

本処理系では、全体の通信性能を大きく損なわず、通信相手となるプロセス数に伴って増加するメモリ使用量を可能な限り抑えるために、Tofu通信において高速型通信モードと省メモリ型通信モードという2つの通信モードを用意し、通信相手となるプロセスごとにこれら2つの通信モードを内部的に使い分けられるようになっています。

高速型通信モードでは、Large受信バッファと呼ぶ比較的大きな受信バッファとSmall受信バッファと呼ぶ追加の受信バッファを、通信相手プロセスごとに用意します。これらの受信バッファを使用し、可能な限り高速に通信を行います。一方、省メモリ型通信モードでは、メモリ使用量がなるべく少なくなるように通信を行います。省メモリ型通信モードには2種類の方式があります。1つ目は、Medium受信バッファと呼ぶ比較的小さな受信バッファだけを使用する方式です。Medium受信バッファは、通信相手プロセスごとに用意されます。この方式では、通信性能がやや落ちる代わりにメモリ使用量を抑えて通信を行うことができます。2つ目は、Shared受信バッファと呼ぶ受信バッファだけを使用する方式です。Shared受信バッファは各プロセスに1つだけ用意され、省メモリ型通信モードで通信するすべての相手プロセスで共有して使われます。このため、この方式では、受信バッファの合計メモリ使用量が通信相手プロセス数によらず一定になります。したがって、通信相手プロセス数が特に多い場合は、この方式を使うことでよりメモリ使用量を抑えることができます。ただし、通信性能は大きく落ちるため注意が必要です。省メモリ型通信モードで使う方式は、MCAパラメーターcommon_tofu_memory_saving_methodによって選択できます。このMCAパラメーターの詳細は、“表4.25 common_tofu_memory_saving_method (省メモリ型通信モードで使用する方式を変更)”をお読みください。通信頻度の高いプロセスとの通信を高速型通信モードで行い、通信頻度の低いプロセスとの通信を省メモリ型通信モードで行うなど、これら2つの通信モードをバランスよく使い分けることが重要になります。

実際にどのプロセスと高速型通信モードで通信するかは、MPIプログラムの通信パターンによって実行時に決まります。通常、最初はどのプロセスが相手でも省メモリ型通信モードでの通信になります。そして、あるプロセスとの通信回数が基準値に達すると、そのプロセスとのそれ以降の通信では高速型通信モードが使われます。ただし、高速型通信モードで通信可能なプロセス数には上限が設定できるようになっており、高速型通信モードによる通信に切り替えられたプロセスの数がこの上限に達した場合、それ以降の切り替えは行われません。

通常、高速型通信モードで通信を行うことができるプロセス数の上限は256に設定されています。この上限値はMCAパラメーターcommon_tofu_max_fastmode_procsによって変更できます。このMCAパラメーターの設定方法によって、すべての通信を高速型通信モードで行うこともできます。また、逆に、すべての通信を省メモリ型通信モードだけ行うような設定も行うことができます。MCAパラメーターcommon_tofu_max_fastmode_procsの詳細は、“表4.20 common_tofu_max_fastmode_procs (高速型通信モードで通信できるプロセス数の上限を変更)”をお読みください。

省メモリ型通信モードから高速型通信モードに切り替わる条件となる、通信回数の基準値は、通常16に設定されています。この基準値は、MCAパラメーターcommon_tofu_fastmode_thresholdによって変更できます。ただし、この回数には、MPIライブラリ内部で行われる制御通信の回数も含みます。そのため、指定した回数より少ない回数のMPIルーチンの呼び出しで切り替わることがあります。MCAパラメーターcommon_tofu_fastmode_thresholdの詳細は、“表4.18 common_tofu_fastmode_threshold (高速型通信モードに切り替わる条件を変更)”をお読みください。

Large受信バッファのサイズは、MCAパラメーターcommon_tofu_large_recv_buf_sizeによって変更できます。MCAパラメーターの詳細は、“表4.19 common_tofu_large_recv_buf_size (Large受信バッファのサイズを変更)”をお読みください。
Medium受信バッファのサイズは、MCAパラメーターcommon_tofu_medium_recv_buf_sizeによって変更できます。MCAパラメーターの詳細は、“表4.22 common_tofu_medium_recv_buf_size (Medium受信バッファのサイズを変更)”をお読みください。
Shared受信バッファのサイズは、MCAパラメーターcommon_tofu_shared_recv_buf_sizeによって変更できます。MCAパラメーターの詳細は、“表4.29 common_tofu_shared_recv_buf_size (Shared受信バッファのサイズを変更)”をお読みください。

並列プロセス数が大きい場合には、MPIプログラム自体が必要とするメモリ量および求められる通信性能に応じて、これらのMCAパラメーターによるチューニングが重要になります。詳細は、“6.11 メモリ使用量の見積り式とチューニング”をお読みください。

6.10.2 動的コネクションの性能への影響

本処理系で採用している動的コネクションでは、通信相手となる各プロセスと初めて通信を行う時点で、通信相手プロセスとの制御通信を伴う処理が発生します。そのため、初回の通信は、それ以降の通常の通信と比較して遅くなります。

特に、ループの中でルートランクを毎回変更してブロードキャスト通信を行うような処理を、MPI_BCASTルーチン/MPI_IBCASTルーチンを使用せずに、行うようなMPIプログラムを書いた場合、ループの各回転においてルートランクがボトルネックとなるため、大幅に通信性能が低下することがあります。そのような場合には、通信相手となるプロセスとの初回の通信をまとめてループの外側(前)に移動してください。通信性能の低下の程度を緩和できることがあります。

また、コネクション確立中の相手プロセス数が130を上回ると、通信処理のリトライの多発に伴って通信性能が低下することがあります。その場合は、同時に発行される各相手プロセスへの初回の通信要求数が130以下になるように、MPI_WAITALLルーチン等の呼び出しを分割してください。

6.11 メモリ使用量の見積り式とチューニング

ジョブには、ノード単位の使用メモリ制限値が設定されており、MPIプログラム自体およびMPIライブラリが使用するメモリ量の合計は、その制限値を超えることはできません。この制限値に達することで、MPIプログラムが正常に実行できない場合は、メモリ使用量の観点からチューニングを行う必要があります。

ここでは、そのチューニングに必要となるメモリ使用量の見積り式、チューニングの指針、および制限値の指定について説明します。

6.11.1 メモリ使用量の見積り式

あるMPIプロセスにおけるMPIライブラリのメモリ使用量は、“図6.7 概算式”を使って見積もることができます。ただし、以下の点にご注意ください。

- “図6.7 概算式”でメモリ使用量を見積もることができるのは、ラージページを使用する場合のみです。ラージページについては、“ジョブ運用ソフトウェア エンドユーザ向けガイド HPC拡張機能編”をお読みください。
- OSはページ単位でメモリを管理するため、ジョブを実行させるには、ページサイズによってより多くのメモリを必要とすることがあります。
- MPIライブラリ以外の要因によるメモリ使用量の変動により、“図6.7 概算式”による見積り結果とMPIジョブ実行時の実測値との誤差が大きくなる場合があります。
- “図6.7 概算式”は、MPIプログラムの並列プロセス数が約200以上の場合にお使いください。並列プロセス数が200以下の場合には、計算結果の誤差が大きくなるため、あまり参考にならないことに注意ください。また、並列プロセス数が200以上であっても、必ずしも正確な値が求められるわけではありませんので、あわせてご注意ください。
- MPIプログラムにおいて片側通信を使用している場合、動的プロセス生成を使用している場合、またはコミュニケータを共有しないMPIプロセスグループ間での通信の確立を使用している場合は、“図6.7 概算式”では見積もることができません。
- MCAパラメーターcommon_tofu_use_memory_poolに1が指定されている場合も、“図6.7 概算式”で見積もることができません。MCAパラメーターcommon_tofu_use_memory_poolについては、“表4.31 common_tofu_use_memory_pool (MPIライブラリ用のメモリプールを使用)”を参照してください。
- MCAパラメーターcommon_tofu_conv_dimに2または3が指定されていると、“図6.7 概算式”の計算結果との誤差が大きくなる場合があります。MCAパラメーターcommon_tofu_conv_dimについては、“表4.16 common_tofu_conv_dim (各プロセスの座標の次元をより高次元に変換してMPIプログラムを実行)”を参照してください。
- 1つの計算ノード内に複数のプロセスが存在する場合は、それだけMPIプロセスによって使用される計算ノード上のメモリ量の割合が大きくなります。このため、計算ノード内のプロセス数が多い場合は、特にメモリ使用量にご注意ください。例えばジョブ実行時に使用する計算ノード数が25600で、計算ノード内のMPIプロセス数が48の場合、MPI_INITルーチンが呼び出された直後のメモリ使用量だけで計算ノード上のメモリ量の90%に達し、アプリケーションとしては動作しない可能性が高くなります。

この概算式の各変数の意味については、“表6.16 メモリ使用量の概算式の変数”に示します。

図6.7 概算式

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

“図6.7 概算式”の1行目は、どのようなMPIプログラムでも最低限消費するメモリ量を表します。MPIプログラムの実行を開始したときおよびMPI_INITルーチンが呼び出されたときに確保されるメモリ量です。このメモリ量は、総プロセス数に依存します。

“図6.7 概算式”の2行目から5行目は、通信相手となる総プロセス数、高速型通信モードで通信されるプロセス数、および自プロセスへの通信が存在するかによって値が変わります。新しく通信相手となるプロセスと初めて通信が行われる時点で増加します。

“図6.7 概算式”の6行目は、コミュニケータごとに消費するメモリ量の総和を表します。これらのコミュニケータには、MPI_COMM_WORLDも含まれます。各コミュニケータが作成された時点およびそのコミュニケータで通信が発生した時点で増加します。

“図6.7 概算式”の7行目は、Unexpected messageが発生した時点で増えます。Unexpected messageは、MPI_SENDルーチンなど送信系のルーチンに対応するMPI_RECVルーチンなど受信系のルーチンの呼出しが遅れたメッセージです。受信側のプロセスにおいて、受信したメッセージを一時待避するためにメモリを使用します。

表6.16 メモリ使用量の概算式の変数

変数	変数の意味	変数の説明	値
N_{Proc}	総プロセス数	そのMPIプロセスと同じコミュニケータMPI_COMM_WORLDに属するプロセスの数です。	―
N_{Peer}	通信相手となるプロセス数	そのMPIプロセスの通信相手となるプロセスの数です。MPI_INITルーチンを呼び出した直後は0ですが、新しく通信相手となるプロセスと初めて通信を行う時点で増加します。MPIプログラムに記述した1対1通信によって行われるプロセスだけでなく、集団通信などによってMPIライブラリの内部で通信を行うプロセスも含まれます。	―
N_{FastPeer}	高速型通信モードによる通信相手となるプロセス数	通信相手となるプロセス数のうち、高速型通信モードで通信を行うプロセスの数です。省メモリ型通信モードを使用しない場合、通信相手となるプロセス数と一致します。MCAパラメーター	―

変数	変数の意味	変数の説明	値
		common_tofu_max_fastmode_procsで指定した値が上限値になります。	
$N_{Loopback}$	自プロセスへの通信が存在するか	自プロセスへの通信が存在する場合は1、存在しない場合は0になります。MPI_INITルーチンを呼び出した直後は0ですが、自プロセスへの通信を行う時点で1になります。	－
N_{Member}	そのコミュニケータに属するプロセス数	プロセス数は、コミュニケータによって異なります。	－
$N_{PeerMember}$	そのコミュニケータで通信相手となるプロセス数	そのコミュニケータを使用して通信を行う通信相手となるプロセスの数です。値は、コミュニケータによって異なります。コミュニケータが作成された直後は0ですが、新しく通信相手となるプロセスと初めて通信を行う時点で増加します。MPIプログラムに記述した1対1通信によって行われるプロセスだけでなく、集団通信などによってMPIライブラリの内部で通信を行うプロセスも含まれます。あるMPIプロセスと複数のコミュニケータで通信を行う場合、それぞれのコミュニケータで加算されます。	－
B_{Large}	Large受信バッファのサイズ	MCAパラメーターcommon_tofu_large_recv_buf_sizeで指定された値です。	省略時の値: 1MiB
B_{Small}	Small受信バッファのサイズ	定数です。	64KiB
B_{Medium}	Medium受信バッファのサイズ	MCAパラメーターcommon_tofu_medium_recv_buf_sizeで指定された値です。	省略時の値: 2KiB ただし、MCAパラメーターcommon_tofu_memory_saving_methodに2が指定されている場合は、MCAパラメーターcommon_tofu_medium_recv_buf_sizeの指定値に関わらず0Bとなります。 MCAパラメーターcommon_tofu_memory_saving_methodについては、“表4.25 common_tofu_memory_saving_method (省メモリ型通信モードで使用する方式を変更)”をお読みください。
B_{Shared}	Shared受信バッファのサイズ	MCAパラメーターcommon_tofu_shared_recv_buf_sizeで指定された値です。	省略時の値: 16MiB ただし、MCAパラメーターcommon_tofu_memory_saving_methodに1が指定されている場合は、MCAパラメーターcommon_tofu_shared_recv_buf_sizeの指定値に関わらず0Bとなります。 MCAパラメーターcommon_tofu_memory_saving_methodについては、“表4.25 common_tofu_memory_saving_method (省メモリ型通信モードで使用する方式を変更)”をお読みください。

変数	変数の意味	変数の説明	値
$B_{Loopback}$	自プロセスへの通信用のバッファのサイズ	定数です。	4MiB
$B_{UnexpectedMessage}$	Unexpected messageの量	Unexpected messageが蓄積された時点で増えます。	―
K_{Buffer}	メモリ割り当て効率	定数です。	1.0
C_{Base}	係数	1ノードに割り当てるプロセスの数(ppn: processes per node)によって決まる定数です。	1-4ppn: 89MiB 5ppn: 67MiB 6-9ppn: 45MiB 10-15ppn: 30MiB 16-48ppn: 20MiB
C_{Proc}	係数	1ノードに割り当てるプロセスの数(ppn: processes per node)によって決まる定数です。	1ppn: 1702B 2ppn: 941B 3ppn: 841B 4ppn: 740B 5ppn: 673B 6-15ppn: 606B 16-48ppn: 514B
C_{Peer}	係数	定数です。	1952B
C_{Member}	係数	1ノードに割り当てるプロセスの数(ppn: processes per node)によって決まる定数です。	1ppn: 408B 2ppn: 184B 3-4ppn: 144B 5-8ppn: 80B 9-16ppn: 56B 17-48ppn: 32B
$C_{PeerMember}$	係数	定数です。	0

6.11.2 メモリ使用量のチューニングの指針

MPIプログラムのパターンによって、チューニングの観点に違いがあります。下表に、それぞれのパターンに応じたチューニングの指針を示します。

表6.17 チューニングの指針

パターン	チューニングの観点
通信性能を求められる通信相手プロセス数が、全通信相手プロセス数と比較して少ない場合	MCAパラメーターcommon_tofu_max_fastmode_procsにより、高速型通信モードで通信を行うプロセス数の上限値を設定してください。 ただし、高速型通信モードで通信を行うプロセス数が上限値に達したことで、高速型通信モードによる通信に切り替えることができなかった通信相手(プロセス)との通信性能に注意する必要があります。 高速型通信モードおよび省メモリ型通信モードの割り当てプロセスが想定どおりにならない場合、MCAパラメーターcommon_tofu_fastmode_thresholdを使って、省メモリ型通信モードから高速型通信モードに切り替えられる通信回数を調整してください。
通信相手となるほぼすべてのプロセスに対し、均等に通信性能が要求される場合	MCAパラメーターcommon_tofu_large_recv_buf_sizeにより、高速型通信モードのLarge受信バッファのサイズを調整してください。 ただし、データサイズが数KiBから数十KiBの場合は通信性能が一律に落ちることがありますので、ご注意ください。

パターン	チューニングの観点
上記の対応で不十分な場合	上記のとおり、MCAパラメーターcommon_tofu_max_fastmode_procsおよびcommon_tofu_large_recv_buf_sizeによる調整に加え、MCAパラメーターcommon_tofu_medium_recv_buf_sizeによって、省メモリ型通信モードのMedium受信バッファのサイズを調整してください。それでも不十分な場合は、MCAパラメーターcommon_tofu_memory_saving_methodに2を指定し、必要に応じてMCAパラメーターcommon_tofu_shared_recv_buf_sizeによって省メモリ型通信モードのShared受信バッファのサイズを調整してください。

6.11.3 メモリ使用量の制限値の指定

“表6.17　チューニングの指針”において述べたとおり、省メモリ型通信モードを使用する場合、MCAパラメーターcommon_tofu_max_fastmode_procsに、高速型通信モードを行う通信相手プロセスの上限数を指定する必要があります。また、高速型通信モードの上限数の指定だけでは、性能とメモリ使用量の両立ができない場合もあります。その場合、さらに受信バッファのサイズも指定する必要があります。

見方を変えると、利用者がMPIプログラム自身の使用するメモリ量を知っている場合、MPIライブラリが残りのメモリの範囲で動作できれば、上述のようなMCAパラメーターの値を利用者が計算する必要はなくなることになります。実際、本処理系では、MPIライブラリが使用してよいメモリ使用量を指定できるようになっています。

すなわち、MPIライブラリが使用可能なメモリ使用量を、利用者が制限することで、本処理系は、内部的に各MCAパラメーターを自動的にチューニングし、可能な限り指定されたメモリ使用量の制限値の範囲で動作するようになります。MPIプログラムを実行する場合の実際のメモリ使用量は、MPIプログラム内で使用しているMPIルーチン、並列数、実行方法などによって変わることがあります。必ずしも指定されたメモリ使用量の制限値の範囲で動作できるとは限りません。ご利用にあたっては、“6.11.3.3 メモリ使用量の制限値指定による実行における注意事項”に記載した注意事項をお読みください。

なお、動的プロセス生成を使用する場合またはコミュニケータを共有しないMPIプロセスグループ間での通信の確立を使用する場合、メモリ使用量の制限値は指定できません。指定した場合の動作は不定になります。

6.11.3.1 メモリ使用量の制限値の指定方法

具体的には、MCAパラメーターcommon_tofu_memory_limitに1以上の値を指定する場合にメモリ使用量制限が有効になります。そのMCAパラメーターに指定された値を、MPIライブラリが使用可能なメモリ使用量の制限値(MiB)と解釈し、他のMCAパラメーターを自動チューニングします。このとき、MCAパラメーターcommon_tofu_memory_limit_peersで指定された値を、通信相手プロセス数として使用します。MCAパラメーターcommon_tofu_memory_limit_peersが指定されていない場合、通信相手プロセス数として、同じコミュニケータMPI_COMM_WORLDに属するプロセスの数が使用されます。MCAパラメーターcommon_tofu_memory_limitについては“表4.23 common_tofu_memory_limit (メモリ使用量の制限値を指定)”を、common_tofu_memory_limit_peersについては“表4.24 common_tofu_memory_limit_peers (メモリ使用量を制限する際に想定する通信相手プロセス数を指定)”をお読みください。

なお、デバッグ用のMPIライブラリを使用している場合、指定されたメモリ使用量の制限値を超えるメモリを使用するなど、必ずしも指定どおりには動作しないことにご注意ください。

6.11.3.2 自動チューニングの対象となるMCAパラメーター

自動的なチューニングの対象となるMCAパラメーターを下表に示します。

表6.18 メモリ使用量の制限値指定により自動チューニングされるMCAパラメーター

優先順位	MCAパラメーター	意味
1	common_tofu_max_fastmode_procs	高速型通信を行う通信相手プロセスの上限数
2	common_tofu_large_recv_buf_size	Large受信バッファのサイズ
3	common_tofu_medium_recv_buf_size	Medium受信バッファのサイズ。本パラメーターはMCAパラメーターcommon_tofu_memory_saving_methodの値が1の場合に有効
	common_tofu_shared_recv_buf_size	Shared受信バッファのサイズ。本パラメーターはMCAパラメーターcommon_tofu_memory_saving_methodの値が2の場合に有効

備考：優先順位の値は、値が小さいほど優先順位が高いことを表します。

優先順位の高いMCAパラメーターだけで、指定されたメモリ使用量の自動チューニングを達成できる場合、残りの優先順位の低いMCAパラメーターの値は、チューニングが不要となるため省略値のままとなります。

MCAパラメーターcommon_tofu_memory_limitに1以上の値が指定されているにもかかわらず、上表に記載されるMCAパラメーターのうち2つ以下のMCAパラメーターが同時に指定されている場合には、それら指定されたMCAパラメーターについては、指定された値がそのまま有効となります。そして、上表のうち残りの指定されていないMCAパラメーターだけを対象に、優先順位の高い(数値が小さい)順に自動チューニングされます。

上表に記載されている3つのMCAパラメーターがすべて指定され、かつMCAパラメーターcommon_tofu_memory_limitが同時に指定された場合、自動チューニングは行われません。

例えば、MPI統計情報などにより1プロセスが通信する相手プロセスの最大数がわかっていて、MCAパラメーターcommon_tofu_memory_limit、common_tofu_memory_limit_peers、およびcommon_tofu_max_fastmode_procsを指定したとします。その場合、まずMCAパラメーターcommon_tofu_large_recv_buf_sizeの値が自動チューニングされ、それでも十分でなければ、さらにMCAパラメーターcommon_tofu_medium_recv_buf_sizeも自動チューニングされます。MCAパラメーターcommon_tofu_memory_saving_methodに2を指定している場合は、common_tofu_medium_recv_buf_sizeの代わりにcommon_tofu_shared_recv_buf_sizeが自動チューニングされます。

6.11.3.3 メモリ使用量の制限値指定による実行における注意事項

自動チューニングは、"図6.7 概算式"を逆算して行われます。そのため、メモリ使用量の制限値を指定してMPIプログラムを実行する際には、以下の注意事項があります。

本処理系が最低限必要とするメモリ使用量は、並列プロセス数などの条件によって変わります。そのため、利用者が指定したメモリ使用量の制限値を超えてしまうことがあります。

また、Unexpected messageが発生した場合は、それをMPIライブラリ内で待避させるために、Unexpected messageの量に応じたメモリを使用することになります。MPIライブラリは、MPIプログラムの実行中にどれだけの量のUnexpected messageが発生するかを把握できないため、自動チューニングではUnexpected messageの発生が考慮されていません。すなわち、Unexpected messageが大量に発生するMPIプログラムでは、利用者が指定したメモリ使用量の制限値を超えてしまうことがあります。

その他にも、動的プロセス生成を行った場合、コミュニケータを作成した場合、などにもメモリが使用されますが、これらのメモリ使用量についても本処理系では事前に把握できないため、自動チューニングの際に内部的に行われる計算の対象外となります。この場合にも、利用者が指定したメモリ使用量の制限値を超えてしまうことがありますので、ご注意ください。

6.12 Tofuバリア通信による高速化

本処理系では、MPI_BARRIERルーチン、MPI_BCASTルーチン、MPI_REDUCEルーチン、およびMPI_ALLREDUCEルーチンの実行時に、Tofuインターコネクトのハードウェア機能として提供されるバリア通信機能を利用し、高速化を実現できます。

本節では、MPIルーチンごとにバリア通信機能が適用される条件について説明します。また、バリア通信の注意事項について説明します。

6.12.1 MPI_BARRIER

MPI_BARRIERルーチンでは、次の条件をすべて満たす場合にバリア通信機能が適用されます。

- MCAパラメーターcollの値に^tbiが指定されていない。
- コミュニケータがグループ内コミュニケータ(intra-communicator)である、かつ、MPI_INTERCOMM_MERGEルーチンで作成されたものではない。
- 以下の条件a.またはb.を満たす。
  a. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が2である。かつ、コミュニケータに割り当てられた計算ノード数が4以上である。
  b. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が3または4である。かつ、以下の条件1.または2.を満たす。
     1. コミュニケータに属するプロセス数が4以上である。かつ、コミュニケータに割り当てられた計算ノード数が3以下ならば、すべての計算ノードでノード内プロセス数が2以上である。
     2. コミュニケータに属するプロセス数とコミュニケータ割り当てられた計算ノード数が両方とも4以上である。
- "6.12.4 バリア通信の注意事項"において説明するバリアゲートの必要数が確保できる。

6.12.2 MPI_BCAST

MPI_BCASTルーチンでは、次の条件をすべて満たす場合にバリア通信機能が適用されます。

- MCAパラメーターcollの値に^tbiが指定されていない。
- MCAパラメーターcoll_tbi_use_on_bcastの値に0が指定されていない。
- コミュニケータがグループ内コミュニケータ(intra-communicator)である、かつ、MPI_INTERCOMM_MERGEルーチンで作成されたものではない。
- 以下の条件a.またはb.を満たす。
  a. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が2である。かつ、コミュニケータに割り当てられた計算ノード数が4以上である。
  b. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が3または4である。かつ、以下の条件1.または2.を満たす。
    1. コミュニケータに属するプロセス数が4以上である。かつ、コミュニケータに割り当てられた計算ノード数が3以下ならば、すべての計算ノードでノード内プロセス数が2以上である。
    2. コミュニケータに属するプロセス数とコミュニケータ割り当てられた計算ノード数が両方とも4以上である。
- “6.12.4 バリア通信の注意事項”において説明するバリアゲートの必要数が確保できる。
- データ型、1要素あたりのバイト数、およびメッセージの要素数上限の組み合わせが“表6.19 MPI_BCASTルーチンでバリア通信が適用可能な組み合わせ”のどれかである。
  - 基本データ型ではないデータ型を利用する場合、メッセージの要素数は、構成要素にした基本データ型の要素数が“表6.19 MPI_BCASTルーチンでバリア通信が適用可能な組み合わせ”において説明するメッセージの要素数上限を超えない範囲である。
  - “表4.10 coll_tbi_repeat_max (バリア通信により通信を行うメッセージの長さの範囲を制御)”において説明されている、MCAパラメーターcoll_tbi_repeat_maxと組み合わせた場合、メッセージの要素数は、メッセージの要素数上限をcoll_tbi_repeat_max倍した値を超えない範囲である。

表6.19 MPI_BCASTルーチンでバリア通信が適用可能な組み合わせ

データ型	1要素あたりのバイト数	メッセージの要素数上限
整数型 浮動小数点型 論理型	8バイト以内	6 (各要素が8バイトの場合) 12 (各要素が4バイトの場合) 24 (各要素が2バイトの場合) 48 (各要素が1バイトの場合)
複素数型	4バイト(2バイト × 2) または 8バイト(4バイト × 2) または 16バイト(8バイト × 2)	12(各要素が4バイトの場合) または 6(各要素が8バイトの場合) または 3(各要素が16バイトの場合)
バイト型	1バイト	48
多言語型	8バイト	6

MPI_BCASTルーチンの送信側と受信側とで型仕様(type signature)が異なる場合(MPI規格ではプログラムの誤り)、バリア通信機能の適用が正しく認識できず、プログラムが異常終了するなど正しく動作しないことがあります。これは、送信側と受信側のどちらか一方がバリア通信の使用条件を満たさないことで、バリア通信の実行環境が整合しなくなるために発生する問題です。本来、MPIプログラムとしては正しくないプログラムですが、MCAパラメーターcoll_tbi_use_on_bcastの値を0に設定することによって、MPI_BCASTルーチンにおいてバリア通信機能を適用しないようにできます。また、MCAパラメーターcollの値を^tbiに設定することによって、MPIルーチンに関係なくバリア通信機能の適用を無効化することもできます。

MCAパラメーターcoll_tbi_use_on_bcastの詳細は、“表4.11 coll_tbi_use_on_bcast (MPI_BCASTルーチンにおいてバリア通信機能を適用)”をお読みください。

MCAパラメーターcollの詳細は、“表4.9 coll (すべての集団通信に共通して適用される設定を変更)”をお読みください。

6.12.3 MPI_REDUCEおよびMPI_ALLREDUCE

MPI_REDUCEルーチンまたはMPI_ALLREDUCEルーチンでは、次の条件をすべて満たす場合にバリア通信機能が適用されます。

- MCAパラメーターcollの値に^tbiが指定されていない。
- コミュニケータがグループ内コミュニケータ(intra-communicator)である、かつ、MPI_INTERCOMM_MERGEルーチンで作成されたものではない。
- 以下の条件a.またはb.を満たす。
  a. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が2である。かつ、コミュニケータに割り当てられた計算ノード数が4以上である。
  b. MCAパラメーターcoll_tbi_intra_node_reductionに指定された値が3または4である。かつ、以下の条件1.または2.を満たす。
     1. コミュニケータに属するプロセス数が4以上である。かつ、コミュニケータに割り当てられた計算ノード数が3以下ならば、すべての計算ノードでノード内プロセス数が2以上である。
     2. コミュニケータに属するプロセス数とコミュニケータ割り当てられた計算ノード数が両方とも4以上である。
- “6.12.4 バリア通信の注意事項”において説明するバリアゲートの必要数が確保できる。
- MCAパラメーターcoll_base_reduce_commute_safeによるリダクション演算の順序を保証する指定がない。
- リダクション演算の演算、データ型、1要素あたりのバイト数、およびメッセージの要素数上限の組み合わせが“表6.20 MPI_REDUCEルーチンおよびMPI_ALLREDUCEルーチンでバリア通信が適用可能な演算の組み合わせ”のどれかである。
  - “表4.10 coll_tbi_repeat_max (バリア通信により通信を行うメッセージの長さの範囲を制御)”において説明されている、MCAパラメーターcoll_tbi_repeat_maxと組み合わせた場合、メッセージの要素数は、メッセージの要素数上限をcoll_tbi_repeat_max倍した値を超えない範囲である。

表6.20 MPI_REDUCEルーチンおよびMPI_ALLREDUCEルーチンでバリア通信が適用可能な演算の組み合わせ

MPI定義済み演算	データ型	1要素あたりのバイト数	メッセージの要素数上限
[C/Fortran] MPI_MAX MPI_MIN [C++] MPI::MAX MPI::MIN [Java] MPI.MAX MPI.MIN	整数型 浮動小数点型	8バイト以内	6
	多言語型	8バイト	6
[C/Fortran] MPI_SUM [C++] MPI::SUM [Java] MPI.SUM	整数型	8バイト以内	6
	浮動小数点型	8バイト以内	3
	複素数型	4バイト (2バイト × 2) または 8バイト (4バイト × 2) または 16バイト (8バイト × 2)	1
	多言語型	8バイト	6
[C/Fortran] MPI_LAND MPI_LOR MPI_LXOR [C++] MPI::LAND MPI::LOR MPI::LXOR	整数型 論理型	8バイト以内	384

MPI定義済み演算	データ型	1要素あたりのバイト数	メッセージの要素数上限
[Java] MPI.LAND MPI.LOR MPI.LXOR			
[C/Fortran] MPI_BAND MPI_BOR MPI_BXOR [C++] MPI::BAND MPI::BOR MPI::BXOR [Java] MPI.BAND MPI.BOR MPI.BXOR	整数型	8バイト以内	6 (各要素が8バイトの場合) 12 (各要素が4バイトの場合) 24 (各要素が2バイトの場合) 48 (各要素が1バイトの場合)
	バイト型	1バイト	48
	多言語型	8バイト	6
[C/Fortran] MPI_MAXLOC MPI_MINLOC [C++] MPI::MAXLOC MPI::MINLOC [Java] MPI.MAXLOC MPI.MINLOC	[C] MPI_2INT MPI_LONG_INT MPI_SHORT_INT [Fortran] MPI_2INTEGER [C++] MPI::TWOINT MPI::LONG_INT MPI::SHORT_INT MPI::TWOINTEGER [Java] MPI.INT2 MPI.LONG_INT MPI.SHORT_INT	8バイト (4バイト + 4バイト) 12バイト (8バイト + 4バイト) 6バイト (2バイト + 4バイト)	3

6.12.4 バリア通信の注意事項

- バリア通信は、各計算ノードに複数個装備されているTofuインターコネクトのバリアゲートのうち必要数を確保し、対応するコミュニケータ内でバリアネットワークを構成することで行われます。バリア通信を行うためのバリアネットワークを構成するには、各ノードに始点/終点用の1個のバリアゲートおよび中継点用の複数個のバリアゲートを確保する必要があります。

  これらのバリアゲートは、各ノードの同一TNIから確保される必要があります。

  始点と終点の役目を果たすバリアゲートは、各ノードでTNIごとに16個、最大総数96個まで利用できます。また、中継点の役目を果たすバリアゲートは、各ノードでTNIごとに32個、最大総数192個まで利用できます。このバリアゲートの最大数は、システムの状況または今後の製品の版数アップによって変動する可能性がありますので、ご注意ください。最大数は、あくまでも目安と考えてください。

- MPI_BARRIERルーチン、MPI_BCASTルーチン、MPI_REDUCEルーチン、およびMPI_ALLREDUCEルーチンでバリア通信が適用される場合、中継点用バリアゲートの使用数の目安は、1回のバリア通信において使用するノード数をN、1ノード内のプロセス数をPとすると以下のように表されます。

  ー coll_tbi_intra_node_reductionに2が指定された場合

  各ノードで最大で$\log_2 N$個程度のバリアゲートが使用されます。

  ー coll_tbi_intra_node_reductionに3が指定された場合

  各ノードで最大で$\log_2 N+(3P-3)$個程度のバリアゲートが使用されます。

ー coll_tbi_intra_node_reductionに4が指定された場合

各ノードで最大で($\log_2 N+\log_2 P$)×P個程度のバリアゲートが使用されます。

プログラム実行時に実際に使用できるバリアゲートの数は、各ノードのバリアゲート使用状況によって変わります。

同じプログラムを実行する場合でも、他のジョブがバリアゲートを使用している場合には、使用できるバリアゲートの数が実行ごとに変化します。

また、使用できるバリアゲート数に伴い、プログラム実行時に指定したcoll_tbi_intra_node_reductionの値どおりのアルゴリズムが選択されない場合もあります。

- coll_tbi_intra_node_reductionに4を指定しても、使用できるバリアゲート数によっては、このMCAパラメーターに2または3を指定したときのアルゴリズムが使われる場合があります。また、バリア通信を使用しない、ソフトウェアによるアルゴリズムが選択される場合もあります。したがって、想定したアルゴリズムに比べて実行時間が長くなる場合があることに注意が必要です。

  また、アルゴリズムごとにリダクション演算順序が異なるため、精度誤差により計算結果が異なる場合があることにも注意が必要です。

- バリアネットワークの構成はコミュニケータやウィンドウの作成時に行われます。

  以下の場合、バリアネットワークの構成が頻繁に行われ実行時間が長くなることがありますが、MCAパラメーターcollの値を^tbiに設定するか、MCAパラメーターcoll_tbi_use_on_comm_dupの値を0に設定すると、実行時間を改善できる場合があります。

  ー MPI_COMM_DUPルーチン、MPI_COMM_IDUPルーチン、MPI_COMM_DUP_WITH_INFOルーチンによるコミュニケータの複製を繰り返すようなMPIプログラム

  ー コミュニケータの生成・解放を頻繁に繰り返すようなMPIプログラム

  ー ウィンドウの作成を繰り返すようなMPIプログラム

  MCAパラメーターcollの詳細は、“表4.9 coll (すべての集団通信に共通して適用される設定を変更)”をお読みください。

  MCAパラメーターcoll_tbi_use_on_comm_dupの詳細は、“表4.12 coll_tbi_use_on_comm_dup (MPI_COMM_DUPルーチン、MPI_COMM_IDUPルーチン、MPI_COMM_DUP_WITH_INFOルーチンで作成されたコミュニケータにおいてバリア通信機能を適用)”をお読みください。

6.13 ランク間で要素数が同一の場合のMPI_BCASTルーチン/MPI_IBCASTルーチン

本処理系では、MPI_BCASTルーチンまたはMPI_IBCASTルーチンにおいて、ランク間の要素数が同じである場合、通信の高速化を実現する機能を用意しています。この機能は、MCAパラメーターcoll_tuned_bcast_same_countに1を指定することで利用できます。

しかし、MPI_BCASTルーチンまたはMPI_IBCASTルーチンに指定した要素数がランク間で異なるMPIプログラムでこの機能を利用した場合、MPIライブラリ内で不整合が生じることがあり、その結果デッドロックが発生して正常に動作しません。

MPIの規格では、ランク間でデータ型と要素数が異なる場合でも、型仕様(type signature)が一致していれば、正しいプログラムとなります。例えば、ランク0はMPI_INTを2要素、ランク1は2個のMPI_INTから成る派生データ型を1要素で実行する場合です。このようなMPIプログラムが正常に実行できるように、通常、MCAパラメーターcoll_tuned_bcast_same_countの省略値は0に設定されています。

ランク間の要素数が同じであることが保証されている場合は、通信の高速化のために1を指定することを推奨します。

MCAパラメーターcoll_tuned_bcast_same_countの詳細は、“表4.14 coll_tuned_bcast_same_count (ランク間で同じ要素数を用いたMPI_BCASTルーチン/MPI_IBCASTルーチンの通信を高速化)”をお読みください。

注意

MPI_ALLGATHERルーチンおよびMPI_ALLGATHERVルーチンには、内部でMPI_BCASTルーチンの処理を呼ぶアルゴリズムが存在します。

このため、以下のプログラムでは、MCAパラメーターcoll_tuned_bcast_same_countに1を指定した場合、MPIライブラリ内で不整合が生じることがあります。

- MPI_ALLGATHERルーチンの場合
  送信側と受信側の要素数がランク間で異なるMPIプログラム

- MPI_ALLGATHERVルーチンの場合
  受信側の要素数(送信側の各ランクに対する受信の要素数を配列で個々に指定)がランクごとで異なるMPIプログラム

不整合が生じないようにするには、以下の対応が必要です。

- MPI_ALLGATHERルーチンの場合
  送信側と受信側に同一の要素数にする必要があります。
- MPI_ALLGATHERVルーチンの場合
  受信側の要素数については配列で指定します。その配列内では送信側の各ランクに対する受信の要素数を個々に指定します。このため、すべてのランクに対して同一の要素数にする必要があります。送信側の要素数についてはランク間で異なっていても構いません。

6.14 集団通信のアルゴリズムとコミュニケータに割り当てられた計算ノードの形状

集団通信ではTofuインターコネクトの機能を利用するアルゴリズムが複数存在します。また、これらのアルゴリズムの中には、コミュニケータに割り当てられた計算ノードの形状が特定の形状のときに呼び出しが可能となるものがあります。本節では、この特定の形状を「直方体的」と呼びます。

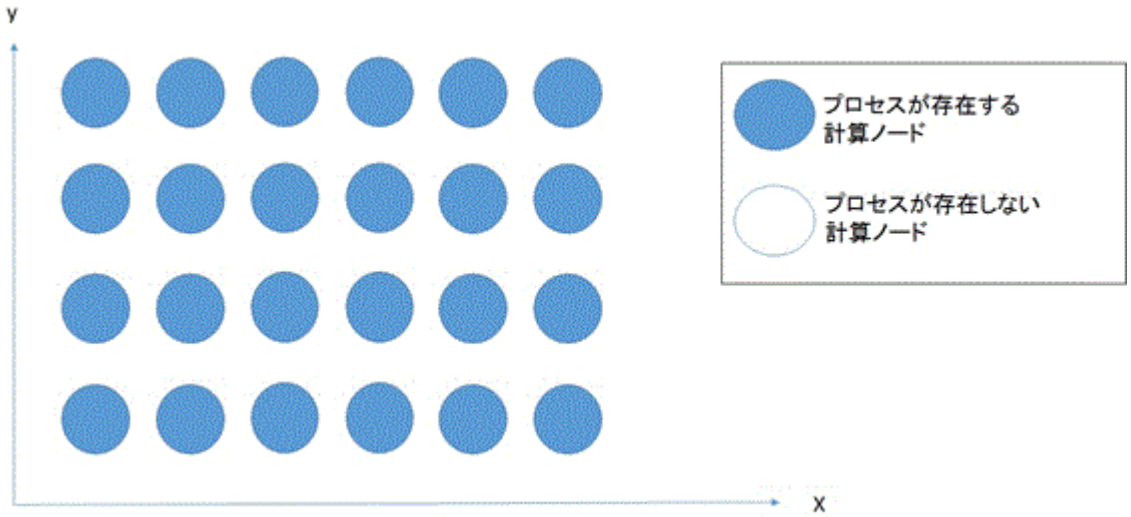
MPI統計情報には、コミュニケータに割り当てられた計算ノードの形状を表示する機能があります。コミュニケータに割り当てられた計算ノードの形状は、“6.16 MPI統計情報”で確認してください。

6.14.1 MPI_COMM_WORLDに割り当てられている計算ノードの形状

ユーザーがジョブ実行時に、プロセスの形状を指定することができます。“プロセスの形状の指定”の詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。ユーザーが指定したプロセスの形状に従って、MPI_COMM_WORLDが生成されます。MPI_COMM_WORLD向けに指定されたノードすべてに、1つ以上のプロセスが生成される場合、このノードの形状を「直方体的」とします。「直方体的」か否かは、計算ノード内のプロセス数によらず、計算ノードにプロセスが存在しているか否かで判断します。

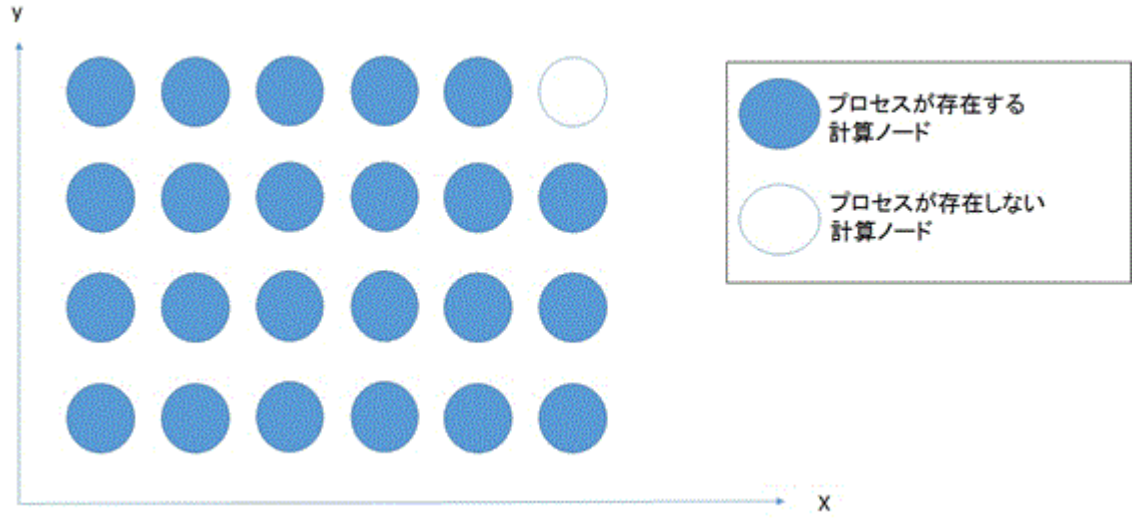
2次元でプロセスの形状をユーザーが指定した場合の例で説明します。“図6.8 6×4の平面に24プロセスが配置された例”では、X×Y=6×4の平面に24プロセスが存在しています。このとき、すべての計算ノードに1プロセスずつ割り当てられます。よって、MPI_COMM_WORLDは「直方体的」です。

図6.8 6×4の平面に24プロセスが配置された例



“図6.9 6×4の平面に23プロセスが配置された、不定形になる例”では、X×Y=6×4の平面に23プロセスが存在しています。このときプロセスが割り当てられない計算ノードが存在します。よって、MPI_COMM_WORLDは「直方体的」ではありません。

図6.9 6×4の平面に23プロセスが配置された、不定形になる例



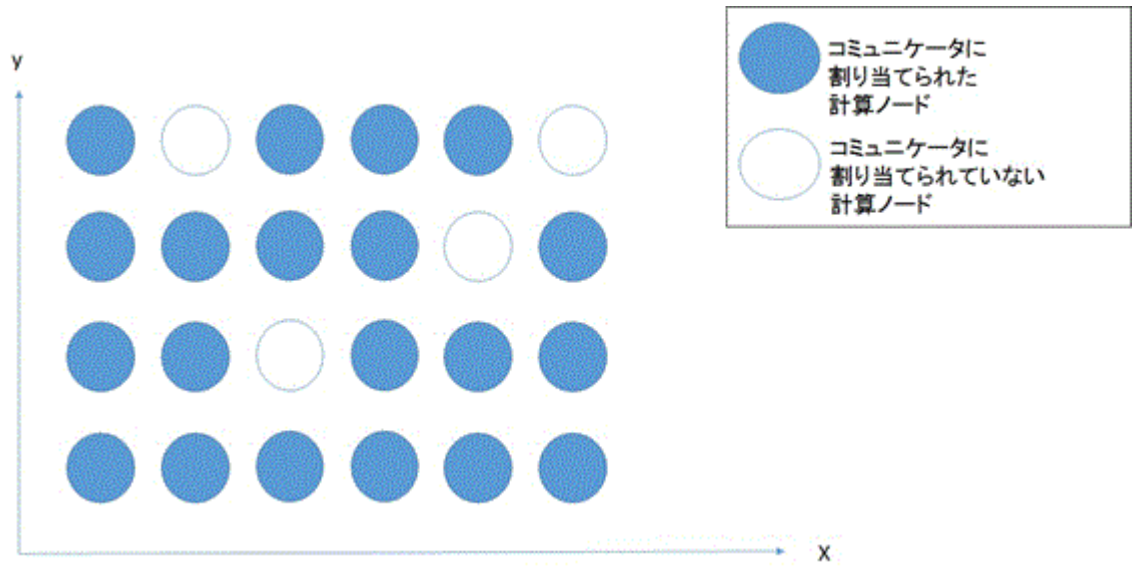
以降は、MPI_COMM_WORLDに割り当てられた計算ノードは「直方体的」であるという前提で説明します。

6.14.2 グループ内コミュニケータに割り当てられた計算ノード

グループ内コミュニケータは、MPI_COMM_WORLDの複製か、その一部のプロセスから構成されます。MPI_COMM_WORLDの一部から生成された新しいグループ内コミュニケータは、「直方体的」にならない可能性があります。“図6.8 6×4の平面に24プロセスが配置された例”では、MPI_COMM_WORLDとして、X×Y=6×4の平面に24プロセスが存在しています。

“図6.10 6×4の平面にランダムにコミュニケータを生成した例”は、“図6.8 6×4の平面に24プロセスが配置された例”のMPI_COMM_WORLDからランダムにコミュニケータを生成した例です。これは、「直方体的」ではありません。このような形状の場合、Tofuインターコネクトを意識したアルゴリズムを呼び出すことができません。

図6.10 6×4の平面にランダムにコミュニケータを生成した例



6.14.3 Tofuインターコネクトを意識したアルゴリズムのための計算ノードの形状

グループ内コミュニケータにおける「直方体的」について説明します。

ユーザーが指定したプロセスの形状の次元によって、「直方体的」であるかどうかの判定方法が変わります。

- 1次元を指定した場合

  どのようなグループ内コミュニケータを生成しても、「直方体的」であると判定します。

- 2次元を指定した場合

  プロセスの形状をX×Yと指定したとします。1..X−1の値をI、1..Y−1の値をJとします。

  以下の条件を満たすように、コミュニケータのプロセスが割り当てられている場合、そのコミュニケータに割り当てられた計算ノードの形状は「直方体的」であると判断します。

  ー X×Yそのもの

  ー X×Yから、IxYあるいはXxJの長方形を除く処理を繰り返すことでできる形状

- 3次元を指定した場合

  プロセスの形状をX×Y×Zと指定したとします。1..X−1の値をI、1..Y−1の値をJ、1..Z−1の値をKとします。

  以下の条件を満たすように、コミュニケータのプロセスが割り当てられている場合、そのコミュニケータに割り当てられた計算ノードの形状は「直方体的」であると判定します。

  ー X×Y×Zそのもの

  ー X×Y×ZからI×Y×Z、X×J×Z、あるいは、X×Y×Kの直方体を除く処理を繰り返すことでできる形状

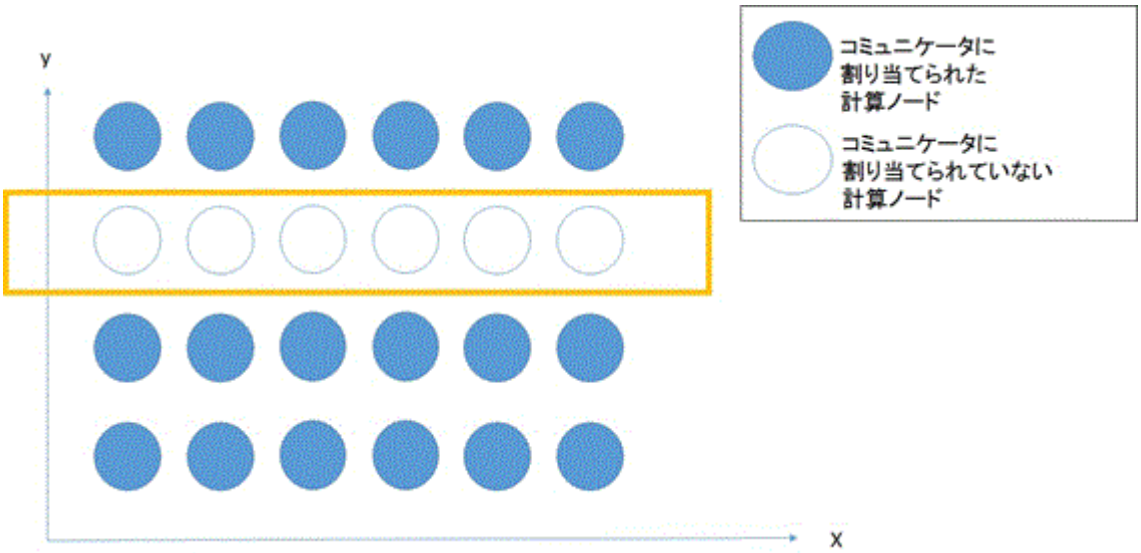
6次元のTofu座標でも同様の考え方で「直方体的」であるかを判定しているアルゴリズムが存在します。

例として、2次元でプロセスの形状を指定した場合について、説明をします。

“図6.8 6×4の平面に24プロセスが配置された例”では、MPI_COMM_WORLDとして、X×Y=6×4の平面にプロセスが配置されています。このとき、MPI_COMM_WORLDは「直方体的」です。

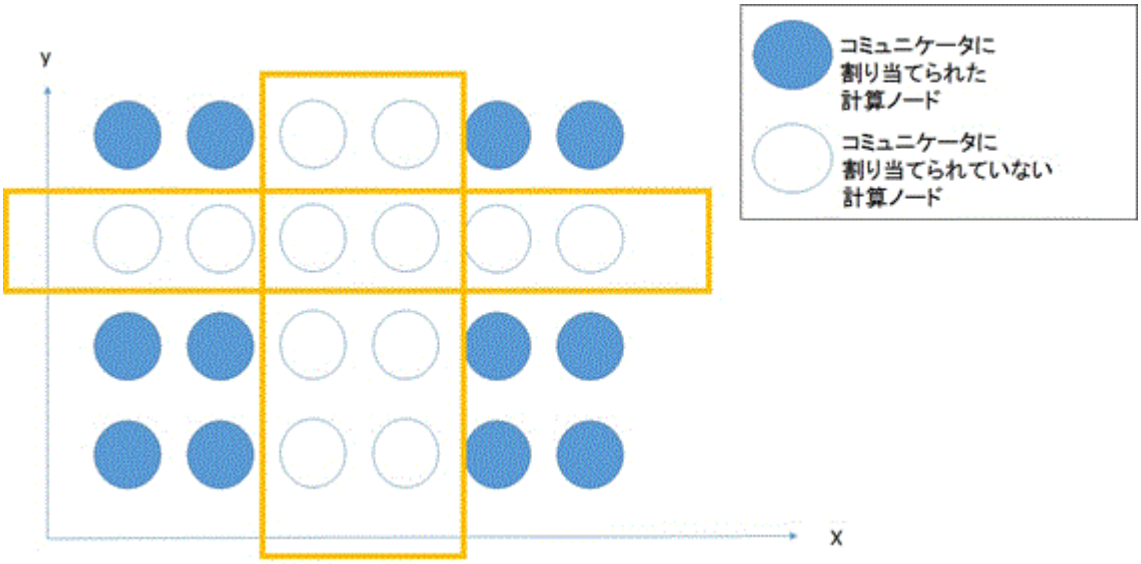
“図6.11 6×4の特定の1行が除かれた例”のように、Y=2を満たす計算ノードは、新しいコミュニケータXに割り当てられない場合を考えます。このとき、コミュニケータXは「直方体的」です。

図6.11 6×4の特定の1行が除かれた例



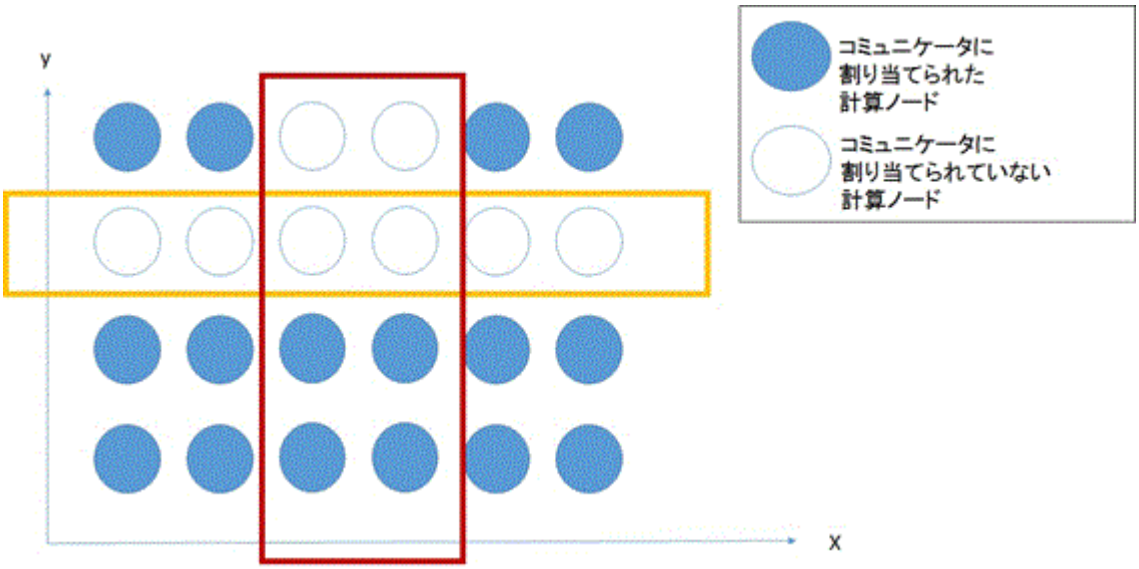
“図6.12「6×4の特定の1行が除かれた例」から特定の2列が除かれた例”は、コミュニケータXの中で、X=1またはX=2を満たす計算ノードすべてが、新しいコミュニケータYに割り当てられない場合です。このとき、コミュニケータYは「直方体的」です。

図6.12「6×4の特定の1行が除かれた例」から特定の2列が除かれた例



“図6.13 不定形の例”は、コミュニケータXの中で、X=1またはX=2を満たす計算ノードの一部が、新しいコミュニケータYに割り当てられない場合です。このとき、コミュニケータYは「直方体的」ではありません。

図6.13 不定形の例



6.14.4 直方体的な形状の軸の長さ

軸の長さは、各軸上に存在するコミュニケータに割り当てられた計算ノードの数で表現します。

“図6.12「6×4の特定の1行が除かれた例」から特定の2列が除かれた例”の場合、X軸はX=0,1,4,5上にコミュニケータに割り当てられた計算ノードが存在するので、X軸の長さは4になります。同様にY軸はY=0,2,3にコミュニケータに割り当てられた計算ノードが存在するので、Y軸の長さは3になります。これより、この形状は4×3としてMPI統計情報で表示されます。

6.15 ジョブ次元変換機能

6.15.1 ジョブ次元変換機能の概要

MCAパラメーターcommon_tofu_conv_dimにジョブの次元より高次元の値を指定してMPIプログラムを実行した場合、可能であれば、MPIプロセスに割り当てられた論理座標を指定された次元の座標に置き換えて実行します。これにより、ジョブ実行開始までの時間が短いという低次元ジョブのメリットと、集団通信性能が高いという高次元ジョブのメリットを両立することが可能です。ただし、ジョブ次元変換機能は適用

できない条件があり、その条件に該当する場合はジョブ本来の次元を用いて実行を継続します。適用条件については、“表6.21 ジョブ次元変換機能のログにおけるreasonの説明”を参照してください。

注意

メモリ使用量に関する注意事項

ジョブ次元変換機能が適用されると、MPIライブラリのメモリ使用量が増加します。MPIプロセスあたりの増加量は、ジョブ内のノード数に24バイトを乗じた値となります。

注意

集団通信アルゴリズムに関する注意事項

一般的には集団通信は低次元ジョブよりも高次元ジョブの場合の方が高速に動作しますが、必ず高速になることが保証されているわけではありません。また、リダクション演算を行う集団通信では、高次元ジョブと低次元ジョブの座標やアルゴリズムの違いにより、リダクション演算の順序が変わることがあります。これにより、計算結果の精度にも影響を与えることがあります。リダクション演算の順序の詳細については“6.8 集団通信におけるリダクション演算の順序保証”をお読みください。

6.15.2 ジョブ次元変換機能のログ出力

MCAパラメーターcommon_tofu_conv_dim_logを用いてログ出力を行うことで、ジョブ次元変換機能が適用されたか否かと、適用できなかった場合の理由を知ることができます。

適用された場合は以下のメッセージが表示されます。

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

ここで、*dim1*は本来のジョブ次元、*dim2*は変換を試みたジョブ次元です。

適用されなかった場合は、以下のメッセージが表示されます。

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

ここで、*dim1*は変換を試みたジョブ次元、*dim2*は本来のジョブ次元、そして*reason*は適用できなかった理由を示します。*reason*の詳細な説明は“表6.21 ジョブ次元変換機能のログにおけるreasonの説明”をお読みください。

表6.21 ジョブ次元変換機能のログにおけるreasonの説明

reason	説明
1	ジョブ次元変換機能はジョブの次元がMCAパラメーターcommon_tofu_conv_dimの指定値未満の場合のみ利用可能です。MCAパラメーターの指定またはノード形状の指定を確認してください。ノード形状の指定方法については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。
2	ジョブ次元変換機能はトーラスモードで実行したジョブでのみ利用可能です。ノードの配置方法としてトーラスモードを指定しているか確認してください。ノードの配置方法については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。
3	ジョブの要求ノード数が12の倍数になっていないため、ジョブ次元変換機能を利用できません。ノード形状の指定を確認してください。ノード形状の指定方法については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。
4	ジョブ運用ソフトウェアが割り当てた領域に含まれるノード数がジョブの要求ノード数と一致していないため、ジョブ次元変換機能を利用できません。ノードの配置方法としてトーラスモードを指定しているか確認してください。ただし、トーラスモードを指定していても、ジョブ形状やシステム内の空きノードの状況によってはジョブ次元変換機能を適用できない領域がジョブ運用ソフトウェアにより割り当てられることがあります。本機能の適用条件を満たす領域が割り当てられる確率を変える方法としては、例えば以下のような方法が考えられます。ただし、いずれの方法も必ず本機能の適用条件を満たす領域が割り当てられることを保証するものではありません。 ・ジョブ形状の次元を変更する

reason	説明
	・ノード数を変更する ・各軸長を変更する ・リソースグループを変更する ・システムの利用状況を確認し、領域割り当ての自由度を下げるような長時間ジョブが走行している時間帯を避ける ノード形状、ノードの配置方法、およびリソースグループの指定方法については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。
5	ジョブ運用ソフトウェアのshapeパラメーターでプロセス形状が指定され、かつ、その大きさがnodeパラメーターで指定されたノード形状よりも小さい場合はジョブ次元変換機能を利用できません。shapeの指定を削除するか、shapeの指定値を確認してください。shapeパラメーターやnodeパラメーターの詳細については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。
6	ジョブ運用ソフトウェアが割り当てた領域に故障ノードが含まれているため、ジョブ次元変換機能を利用できません。故障ノードを回避したい場合は、ジョブ運用ソフトウェアのオプション--mpi "assign-online-node"を指定してください。--mpi "assign-online-node"オプションの詳細については、“ジョブ運用ソフトウェア エンドユーザ向けガイド”をお読みください。

6.16 MPI統計情報

本処理系では、MPI通信に関する統計情報を表示できます。本書では、このMPI通信に関する統計情報をMPI統計情報と呼びます。

MPI統計情報の出力方法には全体出力モード、区間指定出力モードの2種類があります。MPI統計情報の出力は、MCAパラメーターmpi_print_statsによって制御できます。MCAパラメーターmpi_print_statsの詳細は、“表4.37 mpi_print_stats (MPI統計情報を出力)”をお読みください。“表6.22 全体出力モードMPI統計情報の出力内容”に、本処理系において出力可能なMPI統計情報の内容を示します。

MCAパラメーターmpi_print_statsに値1が指定された場合には、“表6.22 全体出力モードMPI統計情報の出力内容”に示すMPI統計情報の出力内容のうちMPI Information、Collective Communication Information、およびProcess Mappingを除くそれぞれの出力項目について、すべての並列プロセスに対する最大値(MAX)、最小値(MIN)、および平均値(AVE)が出力されます。また、最大値および最小値については、それぞれ対応する並列プロセスのランクが括弧内に出力されます。

MCAパラメーターmpi_print_statsに値2が指定された場合には、“表6.22 全体出力モードMPI統計情報の出力内容”に示すMPI統計情報の出力内容のうちMPI InformationおよびProcess Mappingを除くそれぞれの出力項目について、対応する並列プロセスについての情報が出力されます。この場合、MCAパラメーターmpi_print_stats_ranksによって、MPI統計情報の出力を対象とする並列プロセスを特定できます。MCAパラメーターmpi_print_stats_ranksの詳細は、“表4.38 mpi_print_stats_ranks (MPI統計情報を出力する並列プロセスを特定)”をお読みください。なお、これらは、標準エラー出力に出力されます。複数の並列プロセスに対して出力を行う場合、出力の重なりを回避するために、mpiexecコマンドのオプション-of-proc/-std-proc、--of-proc/--std-proc、-oferr-proc/-stderr-proc、または--oferr-proc/--stderr-procを合わせて指定することを推奨します。

MCAパラメーターmpi_print_statsに値3が指定された場合には、“表6.23 区間指定出力モードMPI統計情報の出力内容”に示すMPI統計情報の出力内容のうちMPI Information、Collective Communication Information、およびProcess Mappingを除くそれぞれの出力項目について、mpi_print_statsに値1を指定したときと同様に最大値(MAX)、最小値(MIN)、および平均値(AVE)が出力されます。また、最大値および最小値については、それぞれ対応する並列プロセスのランクが括弧内に出力されます。ただし、ヘッダー部、ボディー部、フッター部に分けて出力される違いがあります。

MCAパラメーターmpi_print_statsに値4が指定された場合には、“表6.23 区間指定出力モードMPI統計情報の出力内容”に示すMPI統計情報の出力内容のうちMPI InformationおよびProcess Mappingを除くそれぞれの出力項目について、mpi_print_statsに値2を指定したときと同様に出力されます。ただし、ヘッダー部、ボディー部、フッター部に分けて出力される違いがあります。

MPI統計情報の区間指定出力についての詳細は、“5.2 区間指定MPI統計情報インターフェース”をお読みください。

MPI統計情報の出力項目Process Mappingについては、上述のMCAパラメーターmpi_print_stats_ranksの値によらず、ランク0の並列プロセスだけによって出力されます。

MPI統計情報の出力項目Connection、Max_Hop、およびAverage_Hopは、区間指定出力モードを利用した場合、区間指定が無効となります。常に、FJMPI_COLLECTION_PRINTルーチンまたはMPI_FINALIZEルーチンの実行時点の情報が出力されます。

MPI統計情報の出力項目 Per-protocol Nonblocking/Persistent Communication CountおよびPer-protocol Nonblocking/Persistent Communication Count Started in Waitについては、MCAパラメーターopal_progress_thread_modeを使用するときに参考となる情報です。詳細は“6.2 アシスタントコアを用いた非同期通信の促進”をお読みください。

表6.22 全体出力モードMPI統計情報の出力内容

出力タイトルおよび出力項目名	出力内容
MPI Information	
Dimension	MPI_COMM_WORLDに属する並列プロセスが配置されているトーラス構成の次元数
Shape	MPI_COMM_WORLDに属する並列プロセスが配置されているトーラス構成のプロセス形状
MPI Memory Usage	
Estimated_Memory_Size	MPIライブラリのメモリ使用量の概算値 “6.11.1 メモリ使用量の見積り式”に記載されているメモリ使用量の見積り値
Per-peer Communication Count	
In_Node	1対1通信におけるノード内の通信回数
Neighbor	1対1通信における隣接するノード間の通信回数
Not_Neighbor	1対1通信における隣接しないノード間の通信回数
Total_Count	1対1通信における通信回数の総計
Connection	Tofu通信のためのコネクション数
Max_Hop	Tofu通信によるプロセス間の最大ホップ数
Average_Hop	Tofu通信によるプロセス間の平均ホップ数
Per-peer Transmission size	
In_Node	1対1通信におけるノード内の送信データ量
Neighbor	1対1通信における隣接するノード間の送信データ量
Not_Neighbor	1対1通信における隣接しないノード間の送信データ量
Total_Size	1対1通信における送信データ量
Per-protocol Communication Count	
Eager	1対1通信において、送信側でEager通信方式を利用した回数
Rendezvous	1対1通信において、送信側でRendezvous通信方式を利用した回数
Persistent_Extended_IF	1対1通信において、送信側で拡張持続的通信要求インターフェースを利用した回数
Unexpected_Message	1対1通信において、内部バッファに一時的に退避したメッセージの最大数
Barrier Communication Count	
Tofu	Tofuインターコネクトのバリア通信機能を利用したバリアの回数
Soft	ソフトウェア的に実行したバリアの回数
Tofu Barrier Collective Communication Count	
Bcast	Tofuインターコネクトのバリア通信機能を利用した集団通信MPI_BCASTルーチン呼出し回数
Reduce	Tofuインターコネクトのバリア通信機能を利用した集団通信MPI_REDUCEルーチンの呼出し回数
Allreduce	Tofuインターコネクトのバリア通信機能を利用した集団通信MPI_ALLREDUCEルーチンの呼出し回数
6D-Tofu-specific Collective Communication Count	
Alltoall	集団通信MPI_ALLTOALLルーチンの呼び出しのうち、Tofu座標を意識してチューニングしたアルゴリズムである、MPI_ALLTOALLルーチンのblacc6dアルゴリズムが実行された回数となります。

出力タイトルおよび出力項目名	出力内容
Tofu-specific Collective Communication Count	
Bcast	集団通信MPI_BCASTルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Reduce	集団通信MPI_REDUCEルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Gather	集団通信MPI_GATHERルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Gatherv	集団通信MPI_GATHERVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Allreduce	集団通信MPI_ALLREDUCEルーチン呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Alltoall	集団通信MPI_ALLTOALLルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Alltoallv	集団通信MPI_ALLTOALLVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Allgather	集団通信MPI_ALLGATHERルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Allgatherv	集団通信MPI_ALLGATHERVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムが呼び出された回数
Non-Tofu-specific Collective Communication Count	
Bcast	集団通信MPI_BCASTルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Reduce	集団通信MPI_REDUCEルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Gather	集団通信MPI_GATHERルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Gatherv	集団通信MPI_GATHERVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Allreduce	集団通信MPI_ALLREDUCEルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Alltoall	集団通信MPI_ALLTOALLルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Alltoallv	集団通信MPI_ALLTOALLVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Allgather	集団通信MPI_ALLGATHERルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Allgatherv	集団通信MPI_ALLGATHERVルーチンの呼出しのうち、Tofuインターコネクト向けにチューニングされたアルゴリズムを利用できなかった回数
Per-protocol Nonblocking/Persistent Communication Count	
Eager	ノンブロッキング1対1通信または持続的要求を用いた通信において、送信側でEager通信方式を利用した回数
Rendezvous	ノンブロッキング1対1通信または持続的要求を用いた通信において、送信側でRendezvous通信方式を利用した回数
Collective	ノンブロッキング集団通信を利用した回数
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	

出力タイトルおよび出力項目名	出力内容
Eager	ノンブロッキング1対1通信または持続的要求を用いた通信において、送信側でEager通信方式を利用し、MPI_WAITルーチン、MPI_WAITANYルーチン、MPI_WAITALLルーチン、またはMPI_WAITSOMEルーチンが呼び出されたときにメッセージ本体の転送が開始された回数
Rendezvous	ノンブロッキング1対1通信または持続的要求を用いた通信において、送信側でRendezvous通信方式を利用し、MPI_WAITルーチン、MPI_WAITANYルーチン、MPI_WAITALLルーチン、またはMPI_WAITSOMEルーチンが呼び出されたときにメッセージ本体の転送が開始された回数
Collective Communication Information	
COLLECTIVE	実行されたブロッキング集団通信ルーチンの名前 各ブロック集団通信ルーチン名からMPI_を除いたものが表示されます。
ALG	実行されたブロッキング集団通信ルーチンのアルゴリズム番号の数字 数字に該当するアルゴリズムは“8.3.4 アルゴリズムと適用に必要な条件一覧”の表をお読みください。
MSGMIN - MSGMAX	ブロッキング集団通信ルーチンのメッセージサイズの範囲 各ブロッキング集団通信ルーチンに指定されたメッセージサイズが、MSGMINからMSGMAXの範囲内であったことを表しています。 各ブロッキング集団通信ルーチンのメッセージサイズの定義は、“表8.4 Infoオブジェクトで使用するブロッキング集団通信ルーチン別メッセージサイズの定義”をお読みください。ただし、以下の集団通信ルーチンについては、出力するランクの要素数をもとに概算で表示しています。 ・ MPI_GATHERVルーチン ・ MPI_SCATTERVルーチン
NODE	実行されたブロッキング集団通信を呼び出したコミュニケータに割り当てられた計算ノードの形状 形状については“6.14 集団通信のアルゴリズムとコミュニケータに割り当てられた計算ノードの形状”をお読みください。 表示として、以下の4種類があります。 ・ 形状が不定形になったものは“計算ノード数(I)”で示しています。 ・ 1次元の形状は“計算ノード数(R)”で示しています。 ・ 2次元の形状は“X軸長xY軸長”で示しています。 ・ 3次元の形状は“X軸長xY軸長xZ軸長”で示しています。
COMM	実行されたブロッキング集団通信を呼び出したコミュニケータのプロセス数
COUNT	実行回数
AVE.TIME(MSEC)	平均実行時間(単位はミリ秒)
Process Mapping	
	MPI_COMM_WORLDに属するすべての並列プロセスのランクと座標の対応一覧 ただし、本情報は、ランクが0の並列プロセスだけによって出力されます

例

MCAパラメーターmpi_print_statsに値1を指定した場合のMPI統計情報の出力例

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

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

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

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

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

----------------- Per-protocol Communication Count -----------------
                               MAX               MIN               AVE
Eager                            0 [   0]          0 [   0]        0.0
Rendezvous                       0 [   0]          0 [   0]        0.0
Persistent_Extended_IF           0 [   0]          0 [   0]        0.0
Unexpected_Message               2 [   0]          1 [   1]        1.5

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

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

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

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

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

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

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


--------------- Collective Communication Information --------------
COLLECTIVE      ALG    MSGMIN -    MSGMAX              NODE     COMM    COUNT AVE.TIME(MSEC)
Allgather       101   16777216 -  67108863     2x     2x     2       32        1 33.773
Allgatherv      101   67108864 -       inf     2x     2x     2       32        1 137.815
Alltoall        100    1048576 -   4194303     2x     2x     2       32        1 25.393
Barrier         200          0 -       inf     2x     2x     2       32        6 2.438
Gatherv         100    4194304 -  16777215     2x     2x     2       32        1 4.499
Scan              1          0 -      4095     2x     2x     2       32        1 0.022
Scatter         300     262144 -   1048575     2x     2x     2       32        1 16.054
Allreduce         3      16384 -     65535                5(I)       5        1 0.809
Barrier         200          0 -       inf                5(I)       5        4 0.101
Bcast             5       4096 -     16383                5(I)       5        1 0.257
Exscan            1          0 -      4095                5(I)       5        1 0.050
Reduce            6      65536 -    262143                5(I)       5        1 3.170

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

表6.23 区間指定出力モードMPI統計情報の出力内容

出力タイトルおよび出力項目名	出力内容
ヘッダー部 1回目のFJMPI_COLLECTION_PRINTルーチン実行時点で出力される内容	
MPI Information	
Dimension	※全体出力モード参照
Shape	
ボディー部	

出力タイトルおよび出力項目名	出力内容
FJMPI_COLLECTION_PRINTルーチン実行単位で出力される内容	
Section	
Time(Sec)	区間指定ごとの実行時間(秒)
Per-peer Communication Count	
In_Node	※全体出力モード参照
Neighbor	
Not_Neighbor	
Total_Count	
Connection	
Max_Hop	
Average_Hop	
Per-peer Transmission size	
In_Node	※全体出力モード参照
Neighbor	
Not_Neighbor	
Total_Size	
Per-protocol Communication Count	
Eager	※全体出力モード参照
Rendezvous	
Persistent_Extended_IF	
Unexpected_Message	
Barrier Communication Count	
Tofu	※全体出力モード参照
Soft	
Tofu Barrier Collective Communication Count	
Bcast	※全体出力モード参照
Reduce	
Allreduce	
6D-Tofu-specific Collective Communication Count	
Alltoall	※全体出力モード参照
Tofu-specific Collective Communication Count	
Bcast	※全体出力モード参照
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	

出力タイトルおよび出力項目名	出力内容
Allgatherv	
Non-Tofu-specific Collective Communication Count	
Bcast	※全体出力モード参照
Reduce	
Gather	
Gatherv	
Allreduce	
Alltoall	
Alltoallv	
Allgather	
Allgatherv	
Per-protocol Nonblocking/Persistent Communication Count	
Eager	※全体出力モード参照
Rendezvous	
Collective	
Per-protocol Nonblocking/Persistent Communication Count Started in Wait	
Eager	※全体出力モード参照
Rendezvous	
Collective Communication Information	
COLLECTIVE	※全体出力モード参照
ALG	
MSGMIN - MSGMAX	
NODE	
COMM	
COUNT	
AVE.TIME(MSEC)	
フッター部 MPI_FINALIZEルーチン実行時に出力される内容	
MPI Memory Usage	
Estimated_Memory_Size	※全体出力モード参照
Process Mapping	
	MPI_COMM_WORLDに属するすべての並列プロセスのランクと座標の対応一覧 ただし、本情報は、ランクが0の並列プロセスだけによって出力されます

例

MCAパラメーターmpi_print_statsに値3を指定した場合のMPI統計情報の出力例

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

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

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

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

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

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

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

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

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

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

--------- Non-Tofu-specific Collective Communication Count --------
                             MAX              MIN              AVE
Bcast                         0 [   0]         0 [   0]        0.0
Reduce                        0 [   0]         0 [   0]        0.0
```

```
Gather                          0 [   0]         0 [   0]       0.0
Gatherv                         0 [   0]         0 [   0]       0.0
Allreduce                       0 [   0]         0 [   0]       0.0
Alltoall                        0 [   0]         0 [   0]       0.0
Alltoallv                       0 [   0]         0 [   0]       0.0
Allgather                       0 [   0]         0 [   0]       0.0
Allgatherv                      0 [   0]         0 [   0]       0.0

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

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

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

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

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

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

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

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

----------- Tofu Barrier Collective Communication Count -----------
                              MAX              MIN              AVE
```

```
Bcast                           0 [   0]         0 [   0]       0.0
Reduce                          0 [   0]         0 [   0]       0.0
Allreduce                       0 [   0]         0 [   0]       0.0

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

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

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

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

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

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

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


------------------------- Process Mapping --------------------------
(0,0,0)                     0,1,2,3
(1,0,0)                     4,5,6,7
(0,1,0)                     8,9,10,11
(1,1,0)                     12,13,14,15
(0,0,1)                     16,17,18,19
(1,0,1)                     20,21,22,23
```

(0, 1, 1)	24, 25, 26, 27
(1, 1, 1)	28, 29, 30, 31

6.17 MPIプログラム実行時の動的デバッグ

本処理系では、MPIプログラムのデバッグのために、次のような機能を用意しています。

- 通信タイムアウト設定 (通信待ちの打切り)
- MPI通信バッファの書込み破壊の監視
- 引数チェック機能

なお、これらのデバッグ機能を利用する場合、MPIプログラムの実行時間が遅くなることがあります。ご利用にあたっては十分にご注意ください。

6.17.1 通信タイムアウト設定

MPIプログラムの実行中にメッセージの通信待ち状態が続く場合、そのプログラムは通信のデッドロックなどによりハングアップしている可能性があります。ハングアップしているプログラムが長時間残り続けるのは、システムの計算資源を有効利用するうえで好ましくありません。そこで本処理系では、想定以上に通信待ちが続く状態を回避する機能を用意しています。MPIプログラムの実行時に通信待ち時間の上限値を指定しておけば、プログラムの実行中に通信待ち時間がこの上限値を超えたときに、その旨のメッセージを出力してプログラムの実行を終了させることができます。以下では、通信待ち時間の上限値を超えた状態を「通信タイムアウト」と呼びます。

この通信待ち時間の上限値(秒)は、MCAパラメーターopal_progress_timeoutで設定できます。出力されるメッセージには、スタックのトレース情報も含まれます。そのトレース情報にシンボル名(ルーチン名)を表示させたい場合は、MPIプログラムの翻訳/結合時に、翻訳/結合コマンドへのオプションとして“-Wl,-export-dynamic”を指定してください。MCAパラメーターopal_progress_timeoutの詳細は、“表4.43 opal_progress_timeout (通信待ちの打切り時間を指定)”をお読みください。

ただし、この通信タイムアウト設定機能では、MPIプログラムの記述に誤りがなく本当に通信待ち時間が長くなるだけの場合でも、指定した通信待ち時間の上限値を超えるとプログラムの実行が打ち切られます。ご利用にあたっては十分な注意が必要です。また、通信タイムアウトとなった時点におけるプロセスまたはMPIプログラム内の位置が、ハングアップの原因箇所とは限りません。したがって、ハングアップの原因箇所を特定するには、プログラムを遡って見直す必要があります。

本機能を使っても、すべてのハングアップを検出できるわけではありません。ハングアップの原因箇所を特定するには、プログラム中で通信ルーチンを呼んでいる箇所の前後に、MPI_BARRIERルーチンを挿入してみるのも1つの方法です。

また、MCAパラメーターopal_abort_delayを使用すると、通信タイムアウトとなってから実際にプログラムを終了させるまでの時間を調整できます。プログラムが終了するタイミングを遅らせると、他のプロセスでも通信タイムアウトになる可能性があります。このようにして複数のプロセスの通信タイムアウト時の情報を取得することも、ハングアップの原因特定に役立つ場合があります。MCAパラメーターopal_abort_delayの詳細は、“表4.39 opal_abort_delay (異常を検出したときのプログラム終了を遅延)”をお読みください。

6.17.2 通信バッファの書込み破壊の監視

ノンブロッキング通信の送信が完了する前に、その送信バッファに対して別の書込みが発生した場合、結果異常や領域破壊など再現性に乏しい誤動作を引き起こす可能性があります。本処理系では、このような誤動作を検出するために、ノンブロッキング通信の送信バッファへの書込みによる領域破壊を監視する手段を用意しています。

このノンブロッキング通信の送信バッファの監視は、MCAパラメーターmpi_check_buffer_writeによって行うことができます。この監視において、ノンブロッキング通信の送信バッファへの書込みがあった場合、その旨のメッセージが出力され、MPIプログラムの実行は終了します。出力されるメッセージには、スタックのトレース情報も出力されますが、そのトレース情報にシンボル名(ルーチン名)を表示させたい場合、あらかじめMPIプログラムの翻訳時にリンカのオプション-export-dynamicを指定してください。具体的には、MPIプログラムの翻訳/結合コマンドにコンパイラに渡すオプションとして、“-Wl,-export-dynamic”を指定してください。MCAパラメーターmpi_check_buffer_writeの詳細は、“表4.33 mpi_check_buffer_write (通信バッファの書込み破壊を監視)”をお読みください。

なお、ノンブロッキング通信のうち、バッファモードでの送信(MPI_IBSENDルーチン)については、監視の対象外となりますので、ご注意ください。

例

使用例

次のようなプログラムについて、送信バッファの監視を行うものとします。

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

MPIプログラムの翻訳/結合コマンドであるmpifccpxコマンドにより、オプション“-Wl,-export-dynamic”を指定し、上記プログラムを翻訳します。次に、MCAパラメーターmpi_check_buffer_writeに値1を指定して、上記プログラムを実行すると、次のようなエラーメッセージが標準エラー出力に出力されます。“*製品インストールパス*”は、システム管理者にお問い合わせください。

```
[mpi::opal-util::check-buffer-write] The buffer was destroyed in this process.
/製品インストールパス/lib64/libmpi.so.0(PMPI_Wait+0x50) [0xffffffff20296ad0] (1)
./a.out(main+0x4a4) [0x1012e4](2)
/…/… (  …       )  [0xfffffffffa07f381c]
./a.out(  …       )  [0x100cec]
```

上記のエラーメッセージの2行目から5行目は、スタックのトレース情報です。このスタックのトレース情報は、通信中に送信バッファへの書込みが発生したノンブロッキング通信の送信処理が完了する場所を示しています。今回の例では、上記メッセージの(1)と(2)から、そのようなノンブロッキング通信の送信処理はmain関数から呼ばれるMPI_Wait関数で完了確認を行うことがわかります。次に、(1)に含まれるアドレス情報をもとにプログラムを確認することで、その送信処理を行った箇所(MPI_Isend関数)がわかります。最後に、MPI_Isend関数とMPI_Wait関数の呼び出し間を確認することで、送信バッファbufへの書込みが行われた箇所を特定できます。

6.17.3 引数チェック機能

本処理系では、デバッグ用のMPIライブラリを使用することで、MPIプログラムの実行時に、MPIルーチン呼出しの引数が正しいかどうかの簡易的な検査を行うことができます。

この引数チェック機能のご利用にあたっては、mpiexecコマンドの実行時にオプション-debuglibまたは--debuglibを指定してください。指定方法については、“4.1 実行コマンドの形式”をお読みください。

引数チェック機能によってエラーが検出される場合、その旨のメッセージが出力され、以下のエラークラスで復帰します。本処理系で出力されるエラークラスの一覧については、“付録A エラークラス一覧”を参照してください。

MPI_ERR_COMM	無効なコミュニケータ
MPI_ERR_COUNT	無効なカウント
MPI_ERR_TAG	無効なタグ
MPI_ERR_RANK	無効なランク
MPI_ERR_TYPE	無効なデータタイプ型
MPI_ERR_BUFFER	無効なバッファポインタ
MPI_ERR_REQUEST	無効なリクエスト
MPI_ERR_TOPOLOGY	無効なトポロジー
MPI_ERR_DIMS	無効な次元
MPI_ERR_ROOT	無効なルート
MPI_ERR_GROUP	無効なグループ
MPI_ERR_OP	無効な操作
MPI_ERR_ARG	その他無効な引数

例

使用例

```
#include <mpi.h>

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

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

このMPIプログラム例では、MPI_Reduce関数の引数に誤ったルートプロセスが指定されています。このMPIプログラムを使って引数チェックを行う場合、次のようなエラーメッセージが出力されます。2行目以降の各行の先頭に出力されている[em99-cn071:18342]は、ホスト名とpidの情報です。

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

6.18 他者(または他社)ツール利用上の注意

6.18.1 Valgrind 利用上の注意

不正なメモリアクセスチェックなどを行えるオープンソースソフトウェアの1つにValgrindがあります。本処理系では、Valgrindを利用した場合の動作は保証していません。しかし、Valgrindの版数によっては、MPIプログラムの実行時に利用できる場合があります。

本処理系で利用するには、デバッグ用のMPIライブラリを指定する必要があります。mpiexecコマンドのオプションとして-debuglibまたは--debuglibを指定してください。

実行例を以下に示します。

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

MPIプログラム実行時にValgrindを利用すると、MPIプログラムだけでなく、MPIライブラリ内部で呼んでいる関数などに対しても、Valgrindがエラーや警告を検出することがあります。

Valgrindによる出力結果を確認する際は、そのエラーや警告がMPIプログラムに対するものかどうかに注意してください。

参考

MPIライブラリ内部で呼んでいる関数名の多くは、以下の接頭語で始まります。

- opal
- ompi
- orte
- mca
- MPI

6.19 リンクダウン時のジョブ実行継続機能の注意事項

リンクダウン時のジョブ実行継続機能の注意事項を説明します。本機能の詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

6.19.1 通信性能

本機能が有効な場合、以下を使用すると、本機能が無効な場合より通信性能が低下することがあります。

- バリア通信が適用されなくなることの影響を受ける以下のルーチン
  - MPI_ALLREDUCEルーチン
  - MPI_BARRIERルーチン
  - MPI_BCASTルーチン
  - MPI_REDUCEルーチン
- 片側通信
- MPIX_ALLGATHER_INITルーチン

6.19.2 MPI_ALLTOALLルーチンを実行する場合の条件

本機能が有効な場合、MPI_ALLTOALLルーチンを実行する時は、以下の2条件を満たす必要があります。2条件が満たされないと、ジョブがMPI_ALLTOALLルーチンで停止し、時間超過で終了する可能性があります。

1. 引数として指定する送信バッファのデータ型が、MPI_ALLTOALLルーチンを実行するすべてのランクで等しい
2. 引数として指定する受信バッファのデータ型が、MPI_ALLTOALLルーチンを実行するすべてのランクで等しい

6.19.3 適用されないアルゴリズム

本機能が有効な場合、MPI_ALLGATHERルーチン、MPI_ALLGATHERVルーチン、およびMPI_ALLTOALLルーチンでは、それぞれ、下表のアルゴリズムが適用されません。

表6.24 適用されない集団通信のアルゴリズム一覧

集団通信ルーチン名	適用されないアルゴリズム
MPI_ALLGATHER	3dtorus_sm
MPI_ALLGATHERV	3dtorus_sm
MPI_ALLTOALL	blacc3d blacc6d

6.19.4 MCAパラメーターおよびオプション指定の注意

本機能では、“local_optionsの形式とオプション説明”の-tuneオプションは指定できません。-tuneオプションを指定した場合、本機能は無効になります。

本機能が有効な場合、下表のMCAパラメーターでは、特定の指定値が有効になりません。指定値を省略したとみなされます。

表6.25 MCAパラメーターの指定が有効にならない場合の動作

MCAパラメーター名	有効にならない指定値	適用される指定値
	指定値の意味	指定値の意味
common_tofu_memory_saving_method	2	1 (省略値)
	省メモリ型通信モードでの通信時に、Shared受信バッファを使用する方式を行う。	省メモリ型通信モードでの通信時に、Medium受信バッファを使用する方式を行う。
common_tofu_use_multi_path	1	0 (省略値)
	1対1通信においてトランキングを行う。	1対1通信において、複数の通信経路を利用しない。

6.19.5 動的プロセス生成

本機能が有効な場合、動的プロセス生成機能を使用する時は、動的プロセスのMPI_FINALIZEルーチンの実行が完了するまで、その生成を行った元の並列プロセスが終了しないようにしてください。

第7章 エラーメッセージ

この章では、本処理系で検出するエラーメッセージについて説明しています。

7.1 並列プロセス関連情報の出力形式

以降に説明するメッセージのうち、特定の並列プロセスを対象として出力されるメッセージについては、各メッセージに先行して、その並列プロセスに対応するホスト名(host)およびプロセスID(pid)が出力される場合があります。その場合、次の形式で出力されます。

[host:pid] メッセージIDおよびメッセージ本文の文字列

7.2 mpiexecコマンドのエラーメッセージ

--
[mpi::mca-base::duplicated-mca-params]
The following MCA parameter has been listed multiple times on the
command line:

MCA param: *MCA parameter*

MCA parameters can only be listed once on a command line to ensure there
is no ambiguity as to its value. Please correct the situation and
try again.
--

- メッセージの説明

  mpiexecコマンドに同じMCAパラメーターを複数回指定できません。

- パラメーターの説明

  MCA parameter : 該当MCAパラメーター

- 利用者の処置

  MCA パラメーターを確認してください。

--
[mpi::mca-base::find-available:not-valid]
A requested component was not found, or was unable to be opened. This
means that this component is either not installed or is unable to be
used on your system (e.g., sometimes this means that shared libraries
that the component requires are unable to be found/loaded). Note that
Open MPI stopped checking at the first component that it did not find.

Host: *host*
Framework: *frame*
Component: *comp*
--

- メッセージの説明

  指定されたMPIライブラリの機能(フレームワークのコンポーネント)を選択できませんでした。サポートされていない機能を指定している可能性があります。mpiexecコマンドまたはMPIプログラムの実行を終了します。

- パラメーターの説明

  host : ホスト名

  frame : フレームワーク名

comp : コンポーネント名

- 利用者の処置

  MCAパラメーターの指定値を確認してください。

[mpi::mca-base::getcwd-error] Error: Unable to get the current working directory

- メッセージの説明

  システムコールgetcwdの処理に失敗しました。現在のパスをカレントディレクトリとして設定します。mpiexecコマンドの実行を継続します。

- 利用者の処置

  システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::mca-var::invalid-value-enum]
An invalid value was supplied for an enum variable.

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

- メッセージの説明

  MCAパラメーターに指定された値が正しくありません。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  name : MCAパラメーター名

  val : MCAパラメーターに指定された値

  values : MCAパラメーターに指定可能な値の一覧

- 利用者の処置

  MCAパラメーターの指定値を確認してください。

[mpi::mca-var::missing-param-file]
Process *pid* Unable to locate the variable file "*file*" in the following search path:
** *wdir***

- メッセージの説明

  AMCAパラメーターファイル(MCAパラメーターの設定ファイル)が指定されたパスでは見つかりません。メッセージの出力後、mpiexecコマンドまたはMPIプログラムを継続して実行します。

- パラメーターの説明

  pid : プロセスID

  file : 指定されたファイルパス

  wdir : mpiexecコマンドが実行されたディレクトリパス

- 利用者の処置

  AMCAパラメーターファイル(MCAパラメーターの設定ファイル)の指定を確認してください。

[mpi::opal-runtime::opal_init:startup:internal-failure]
It looks like opal_init failed for some reason; your parallel process is
likely to abort. There are many reasons that a parallel process can

fail during opal_init; some of which are due to configuration or environment problems. This failure appears to be an internal failure; here's some additional information (which may only be relevant to an Open MPI developer):

***fun* failed**
--> Returned value *errinfo* (*errno*) instead of OPAL_SUCCESS

- メッセージの説明

  mpiexecコマンドまたはMPIプログラムの初期化処理に失敗しました。mpiexecコマンドまたはMPIプログラムの実行を終了します。

- パラメーターの説明

  fun : エラールーチン名

  errinfo : エラー詳細

  errno : エラー番号

- 利用者の処置

  MCAパラメーターの指定に誤りがないか確認してください。指定に誤りがない場合には、出力されたメッセージと合わせて、担当保守員(SE)にご連絡ください。

[mpi::opal-util::keyval-error] keyval parser: error *num* reading file *file* at line *lineno*: *code*

- メッセージの説明

  AMCAパラメーターファイル(MCAパラメーターの設定ファイル)に使用できない文字が含まれています。メッセージの出力後、mpiexecコマンドまたはMPIプログラムの実行を継続します。

- パラメーターの説明

  num : エラー番号

  file : AMCAパラメーターファイル(MCAパラメーターの設定ファイル)のファイルパス

  lineno : 行番号

  code : 使用できない文字

- 利用者の処置

  AMCAパラメーターファイル(MCAパラメーターの設定ファイル)に使用できない文字がないか確認してください。

[mpi::opal-util::memory-error] Unable to allocate memory for the private addresses array

- メッセージの説明

  プライベート・アドレス配列のためのメモリを割り当てることができません。メモリの獲得に失敗しました。メッセージの出力後、mpiexecコマンドまたはMPIプログラムの実行を継続します。

- 利用者の処置

  プログラムの最大メモリサイズリミット値を確認してください。問題がない場合、システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::opal-util::param-option] *mpiexec* Error: option "*opt*" did not have enough parameters (*num*)

- メッセージの説明

  mpiexecコマンドに指定したオプションの引数が不足しています。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  mpiexec : mpiexecコマンド

  opt : 該当オプション

num : 必要な引数の数

- 利用者の処置

  mpiexecコマンドの該当オプションに必要となる引数を指定してください。

[mpi::opal-util::private-ipv4-error] FOUND BAD!

- メッセージの説明

  サポートされていないMCAパラメーターが指定されている可能性があります。(OMPI_MCA_opal_net_private_ipv4)メッセージの出力後、mpiexecコマンドまたはMPIプログラムの実行を継続します。

- 利用者の処置

  MCAパラメーターの指定値を確認してください。

[mpi::opal-util::unknown-option] *mpiexec* Error: unknown option "*opt*"

- メッセージの説明

  mpiexecコマンドにサポートされていないオプションが指定されました。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  mpiexec : mpiexec コマンド

  opt : 該当オプション

- 利用者の処置

  mpiexecコマンドに正しいオプションを指定してください。

[mpi::orterun::event-def-failed]
mpiexec was unable to define an event required for proper operation of the system. The reason for this error was:
Error: *syserr*

- メッセージの説明

  システムコールの実行に失敗しました。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  syserr : システムエラーの詳細

- 利用者の処置

  システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::orterun::multi-apps-and-zero-np]
mpiexec found multiple applications specified on the command line, with at least one that failed to specify the number of processes to execute. When specifying multiple applications, you must specify how many processes of each to launch via the -np argument.

- メッセージの説明

  MPMDモデルでの実行時に、各MPIプログラムに対して並列プロセス数の指定がないため、処理を継続できません。mpiexecコマンドの実行を終了します。

- 利用者の処置

  mpiexecコマンドで指定した各MPIプログラムに対して並列プロセス数を指定してください。

[mpi::orterun::nothing-to-do]
mpiexec could not find anything to do.

- メッセージの説明

  内部エラーを検知しました。mpiexecコマンドの実行を終了します。

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::orterun::orterun:appfile-not-found]
Unable to open the appfile:

file

Double check that this file exists and is readable.

- メッセージの説明

  実行定義ファイル指定方式で指定されたファイルが見つかりません。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  file : 指定されたファイルパス

- 利用者の処置

  ファイルパスを確認してください。

[mpi::orterun::orterun:executable-not-specified]
No executable was specified on the mpiexec command line.

Aborting.

- メッセージの説明

  mpiexecコマンドにMPIプログラムの指定がありません。mpiexecコマンドの実行を終了します。

- 利用者の処置

  mpiexecコマンドにMPIプログラムを指定してください。

[mpi::orterun::precondition]
mpiexec was unable to precondition transports
Returned value *errno* instead of ORTE_SUCCESS.

- メッセージの説明

  内部エラーを検知しました。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  errno : エラー番号

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

--
[mpi::orte-runtime::orte_init:startup:internal-failure]
It looks like orte_init failed for some reason; your parallel process is likely to abort. There are many reasons that a parallel process can fail during orte_init; some of which are due to configuration or environment problems. This failure appears to be an internal failure; here's some additional information (which may only be relevant to an Open MPI developer):

***fun* failed**
--> Returned value *errinfo* (*errno*) instead of ORTE_SUCCESS
--

- メッセージの説明

  mpiexecコマンドまたはMPIプログラムの初期化処理に失敗しました。mpiexecコマンド またはMPIプログラムの実行を終了します。

- パラメーターの説明

  fun : エラールーチン名

  errinfo : エラー詳細

  errno : エラー番号

- 利用者の処置

  MCAパラメーターの指定に誤りがないか確認してください。指定に誤りがない場合には、出力されたメッセージと合わせて、担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  plexecコマンドの実行に失敗しました。ジョブ運用ソフトウェアの並列実行環境(PLE)が正しく動作していない可能性があります。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  errinfo : エラー詳細

  errno : エラー番号

  com : 実行コマンド

- 利用者の処置

  ジョブ運用ソフトウェアの並列実行環境(PLE)が正常に動作しているかシステム管理者に確認してください。正常に動作している場合には、内部エラーの可能性がありますので、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  mpiexecコマンドの並列プロセス数の指定に誤りがあります。mpiexecコマンドの実行を終了します。

- 利用者の処置

  mpiexecコマンドの並列プロセス数を確認してください。

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

- メッセージの説明

  mpiexecコマンドからmpiexecコマンドを2重に起動できません。mpiexecコマンドの実行を終了します。

- 利用者の処置

  mpiexecコマンドには、MPIプログラムを指定してください。

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

- メッセージの説明

  plexecコマンドがシグナルを受信して異常終了しました。ジョブ運用ソフトウェアの並列実行環境(PLE)が正しく動作していない可能性があります。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  signo : 子プロセスが受信したシグナル番号

- 利用者の処置

  ジョブ運用ソフトウェアの並列実行環境(PLE)が正常に動作しているかシステム管理者に確認してください。正常に動作している場合には、内部エラーの可能性がありますので、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  システムコールwaitpidの動作に失敗しました。mpiexecコマンドの実行を終了します。

- パラメーターの説明

  errinfo : エラー詳細

  errno : エラー番号

- 利用者の処置

  システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::schizo-ompi::param-env] Warning: could not find environment variable "*env*"

- メッセージの説明

  指定した環境変数の値が設定されていません。メッセージの出力後、mpiexecコマンドの実行を継続します。

- パラメーターの説明

  env : 指定した環境変数

- 利用者の処置

  mpiexecコマンドのオプション(-x)で指定した環境変数の値を確認してください。

7.3 通信ライブラリのエラーメッセージ

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

- メッセージの説明

  Tofuインターコネクトによる通信時にメモリの獲得に失敗しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

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

- メッセージの説明

  メモリの獲得に失敗しました。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

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

- メッセージの説明

  メモリの獲得に失敗しました。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

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

- メッセージの説明

  アルゴリズム選択の初期化の途中で問題が発生しました。プログラムを終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  アルゴリズム選択の終了処理の途中で問題が発生しました。プログラムを終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  システムのアルゴリズム選択に問題が発生しました。この際、MPI統計情報の一部機能が正しく表示できません。

- パラメーターの説明

  reason : 理由を表す数字

reason	理由の内容
0	システムによるアルゴリズム選択で使用するファイルに記載している文字列または文法が誤っています。
1	システムによるアルゴリズム選択で使用するファイルに記載可能なルール数を超えています。
2	システムによるアルゴリズム選択で使用するファイルに記載可能な条件文の数を超えています。
3	システムによるアルゴリズム選択で使用するファイルに記載している割り算でゼロ除算が存在します。
4	システムによるアルゴリズム選択で使用するファイルに記載している変数が誤っています。
5	アルゴリズム名に誤りがあります。
6	実行可能なアルゴリズムが存在しません。
7	ブロッキング集団通信のルールが網羅されていません。

reason	理由の内容
8	バリア通信をシステムによるアルゴリズム選択で使用するファイル内で指定しています。
9	システムによるアルゴリズム選択で使用するファイルの解析中にMPIライブラリ内部でエラーが発生しています。

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  バリアネットワークの作成に失敗しました。

- パラメーターの説明

  reason : 失敗の内容

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  バリアネットワークの解放処理に失敗しました。

- パラメーターの説明

  reason : 失敗の内容

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  バリアが内部的に使用するファイルへのアクセスに失敗しました。

- パラメーターの説明

  operate : 操作

  file : ファイルパス

- 利用者の処置

  ファイルシステムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

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

- メッセージの説明

  バリアがメモリの獲得に失敗しました。

- パラメーターの説明

  errno : エラー番号

- 利用者の処置

  メモリの使用量を確認してください。問題がない場合には、システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::coll-tbi::operation-error] Operation error is reported by Tofu barrier communication. [*rank function arguments*] [*data*] *<スタックのトレース情報>*

- メッセージの説明

  バリア通信で演算エラーが検出されました。エラークラスMPI_ERR_OPのエラーを発生させます。

- パラメーターの説明

  rank : ランク

  function : MPIルーチン

  arguments : MPIルーチンの引数

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  表示されたMPIルーチンで、以下のどちらかの記述がないかMPIプログラムを確認してください。

  リダクション演算はarguments中にopとして表示されます。

  ー プロセス間で異なるリダクション演算を指定した

  ー プロセス間で異なる集団通信ルーチンを指定した

  問題がない場合、内部エラーの可能性がありますので、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  集団通信のサブコミュニケータ処理の初期化処理に失敗しました。

- パラメーターの説明

  reason : 失敗の内容

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  メモリの獲得に失敗しました。

- 利用者の処置

  メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

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

- メッセージの説明

  Tofuインターコネクトによる通信のコネクション確立処理でエラーが発生しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  Tofuインターコネクトによる通信時にメモリの獲得に失敗しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

[mpi::common-tofu::mrq-error] Communication error is reported by Tofu MRQ. [*data*]

- メッセージの説明

  Tofuインターコネクトによって通信エラーが検出されました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::mrq-memory-error] Communication memory error is reported by Tofu MRQ. [*data*]

- メッセージの説明

  Tofuインターコネクトによって通信時のメモリ指定エラーが検出されました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  MPIの通信ルーチンに指定した送信バッファおよび受信バッファにおける、先頭アドレス、データ型、または要素数に誤りがないかを確認してください。

[mpi::common-tofu::mrq-peer-error] Communication peer error is reported by Tofu MRQ. This error may be caused by abort of peer process. [*data*]

- メッセージの説明

  Tofuインターコネクトによって通信先のノード・プロセスでのエラーが検出されました。通信先プロセスの停止または通信先プロセスでの送受信バッファの解放によるエラーの可能性があります。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  ジョブを強制的に終了した場合またはMPIプログラムがMPI_FINALIZEルーチンを呼ぶ前に終了した場合、このメッセージが出力されることがあります。その場合、このメッセージを無視してください。また、通信が完了する前に送受信バッファを解放した場合、このメッセージが出力されることがあります。その場合、送受信バッファの解放処理を見直してください。どちらでもない場合、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::tcq-error] Communication error is reported by Tofu TCQ. [*data*]

- メッセージの説明

  Tofuインターコネクトによって通信エラーが検出されました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::tcq-memory-error] Communication memory error is reported by Tofu TCQ. [*data*]

- メッセージの説明

  Tofuインターコネクトによって通信時のメモリ指定エラーが検出されました。MPIプログラムの実行を終了します。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  MPIの通信ルーチンに指定した送信バッファおよび受信バッファにおける、先頭アドレス、データ型、または要素数に誤りがないかを確認してください。

[mpi::common-tofu::tofu-async-error] Tofu interconnect detected an asynchronous error. [Event: *description* (*eventno*), TNI: *tniid*, BG: *bgid*]

- メッセージの説明

  Tofuバリア通信で内部エラーを検知しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  description : 通信ライブラリ(Tofuライブラリ)から通知されたエラーメッセージ

  eventno : 検知した内部エラーに対応する通信ライブラリ(Tofuライブラリ)側の管理番号

  tniid : エラーが発生したネットワークインターフェース装置(TNI)のID

  bgid : エラーが発生したバリアゲート(BG)のID

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::tofu-async-error] Tofu interconnect detected an asynchronous error. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- メッセージの説明

  Tofuワンサイド通信で内部エラーを検知しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  description : 通信ライブラリ(Tofuライブラリ)から通知されたエラーメッセージ

  eventno : 検知した内部エラーに対応する通信ライブラリ(Tofuライブラリ)側の管理番号

  tniid : エラーが発生したネットワークインターフェース装置(TNI)のID

  cqid : エラーが発生した完了キュー(CQ)のID

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::tofu-init-failure] Internal error. [*reason*]

- メッセージの説明

  Tofuインターコネクトの初期化処理に失敗しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  reason : 初期化処理失敗の内容

- 利用者の処置

  システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

[mpi::common-tofu::tofu-mrq-overflow] Tofu interconnect detected an MRQ overflow. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- メッセージの説明

  Tofuインターコネクトによる完了通知の数が完了キューのエントリ数を超えました。MPIプログラムのノンブロッキング通信処理の発行回数に問題があります。MPIプログラムの実行を終了します。

- パラメーターの説明

  description : 通信ライブラリ(Tofuライブラリ)から通知されたエラーメッセージ

  eventno : 検知した内部エラーに対応する通信ライブラリ(Tofuライブラリ)側の管理番号

  tniid : エラーが発生したネットワークインターフェース装置(TNI)のID

  cqid : エラーが発生した完了キュー(CQ)のID

- 利用者の処置

  MPIプログラムのノンブロッキング通信処理またはEager通信方式を用いた送信に対する通信完了処理を見直してください。または、“表4.26 common_tofu_num_mrq_entries (完了キューのエントリ数を指定)”を参照してTofuインターコネクトの完了キューのエントリ数を増やしてください。

[mpi::common-tofu::tofu-congestion] Tofu interconnect detected the congestion. [Event: *description* (*eventno*), TNI: *tniid*, CQ: *cqid*]

- メッセージの説明

  Tofuインターコネクトによる輻輳が検知されました。MPIプログラムの実行を終了します。

- パラメーターの説明

  description : 通信ライブラリ(Tofuライブラリ)から通知されたエラーメッセージ

  eventno : 検知した内部エラーに対応する通信ライブラリ(Tofuライブラリ)側の管理番号

  tniid : エラーが発生したネットワークインターフェース装置(TNI)のID

  cqid : エラーが発生した完了キュー(CQ)のID

- 利用者の処置

  MPIプログラムの処理を見直して、特定のプロセスに通信が集中しないようにしてください。

[mpi::common-tofu::tofu-stag-error] Failed to query/register Tofu STag. [*data*]

- メッセージの説明

  バッファの使用の誤りまたはTofuインターコネクトのメモリ管理資源の枯渇が検出されました。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  MPIの通信ルーチンに指定した送信バッファおよび受信バッファにおける、先頭アドレス、データ型、または要素数に誤りがないかを確認してください。MPIプログラムの各並列プロセスが書込みできないメモリ領域を、送信バッファまたは受信バッファに指定できません。また、ラージページを使用していない場合、ラージページを使用するか、MPIの通信ルーチンに指定する送信バッファおよび受信バッファにおける、先頭アドレスと要素数のパターンを削減してください。指定に誤りがなく、ラージページを使用している場合、内部エラーの可能性がありますので、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::common-tofu::tofu-stag-release-error] Failed to release Tofu STag. [data]

- メッセージの説明

  Tofuインターコネクトのメモリ管理資源の解放時に不整合が検出されました。

- パラメーターの説明

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::dpm::num-nodes-invalid] [[*jobid*,*snum*],*rank*] The specified number of nodes is invalid.

- メッセージの説明

  動的プロセス生成のinfoキーnum_nodesに指定したノード数が、ジョブに割り当てたノード数を超えている可能性があります。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成ルーチンのinfoキーnum_nodesに指定したノード数の値を見直してください。

[mpi::dpm::rankmap-invalid] [[*jobid*,*snum*],*rank*] The value specified with the info key rank_map is invalid.

- メッセージの説明

  動的プロセス生成ルーチンのinfoキーrank_mapに指定した値が不正です。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  infoキーrank_mapの値には、bychipまたはbynodeを指定してください。

[mpi::dpm::spawn-exist-error] [[*jobid*,*snum*],*rank*] The number of MPI_COMM_WORLD for the dynamic processes that can exist at the same time exceeds the upper limit.

- メッセージの説明

  同時に存在可能な動的プロセスのMPI_COMM_WORLDの数の上限値を超えました。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  同時に存在する動的プロセスのMPI_COMM_WORLDの数を65535以下としてください。

[mpi::dpm::spawn-limit-error] [[*jobid*,*snum*],*rank*] The total invocation count of the MPI_COMM_SPAWN routine or the MPI_COMM_SPAWN_MULTIPLE routine exceeds the upper limit.

- メッセージの説明

  動的プロセス生成の回数の上限値を超えました。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

rank : ランク

- 利用者の処置

  動的プロセス生成の回数を4294967295以下としてください。

[mpi::dpm::spawn-resource-error] [[*jobid*,*snum*],*rank*] There are not enough compute nodes to create processes dynamically according to the requirement.

- メッセージの説明

  動的プロセス生成において、プロセスを生成するために必要な未使用のノードが不足しています。

  MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成に利用するノード数またはノードあたりの生成可能プロセス数の指定が正しいか確認してください。各プロセスに割り当てるCPUコア数だけをVCOORDファイルに指定している場合は、その指定順序が原因の可能性があります。

[mpi::dpm::vcoord-core-num-invalid] [[*jobid*,*snum*],*rank*] The number of CPUs (cores) on a compute node is not sufficient.

- メッセージの説明

  動的プロセス生成において、割り当てに必要な数の CPU コアが存在しません。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成で指定するVCOORDファイルについて、1つのノードで生成するプロセスに割り当てるCPUコアの総数は、そのノードに搭載されているCPUコア数以下になるようにしてください。

[mpi::dpm::vcoord-format-error] [[*jobid*,*snum*],*rank*] The VCOORD file format is invalid.

- メッセージの説明

  動的プロセス生成において、infoキーvcoordfileに指定されたVCOORDファイルの内容は、書き方が間違っています。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成で指定するVCOORDファイルについて、ファイルの内容を見直してください。

[mpi::dpm::vcoord-invalid] [[*jobid*,*snum*],*rank*] The specified logical coordinates are invalid.

- メッセージの説明

  動的プロセス生成のinfoキーvcoordfileに指定したVCOORDファイルに不正な論理座標が指定されています。指定された論理座標がジョブに割り当てたノードの範囲を超えている可能性があります。

  MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成で指定するVCOORDファイルについて、ファイルの内容を見直してください。

[mpi::dpm::vcoord-maxproc-invalid] [[*jobid*,*snum*],*rank*] The number of processes exceeds the number of logical coordinates specified in the VCOORD file.

- メッセージの説明

  VCOORDファイルの行数が、動的プロセス生成ルーチンの引数maxprocsに指定したプロセス数に対して不足しています。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成で指定するVCOORDファイルの行数は、動的プロセス生成ルーチンの引数maxprocsで指定するプロセス数以上となるようにしてください。

[mpi::dpm::vcoord-not-exist] [[*jobid*,*snum*],*rank*] The file specified with the vcoordfile info key does not exist.

- メッセージの説明

  動的プロセス生成ルーチンのinfoキーvcoordfileに指定したVCOORDファイルが存在しません。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  infoキーvcoordfileに指定したVCOORDファイルのパスを見直してください。

[mpi::dpm::vcoord-numa-error] [[*jobid*,*snum*],*rank*] The process cannot be bound to CPUs (cores) under numanode_assign_policy.

- メッセージの説明

  動的プロセス生成において、numanode_assign_policyの値に従った、プロセスへのCPUコア割り当てができません。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  動的プロセス生成で指定するVCOORDファイルで指定したnumanode_assign_policyの値を見直してください。

[mpi::dpm::vcoord-ppn-invalid] [[*jobid*,*snum*],*rank*] The number of processes exceeds the limit on a compute node.

- メッセージの説明

  動的プロセス生成において、1つの計算ノードで生成可能なプロセス数を超えました。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  VCOORDファイルの内容を見直し、1つの計算ノードで生成可能なプロセス数を超えないように指定してください。

[mpi::dpm::vcoord-use-error] [[*jobid*,*snum*],*rank*] The specified logical coordinates are already used.

- メッセージの説明

  動的プロセス生成において、VCOORDファイルで指定した論理座標の一部がすでに使用中です。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

- 利用者の処置

  別の論理座標を指定してください。または、すでに実行中のプロセスが終了するのを待ち合わせて再実行してください。エラーコードとして、FJMPI_ERR_SPAWN_NO_AVAILABLE_NODESが返ります。

[mpi::dpm::violated-establishing-communication] MPI_COMM_CONNECT or MPI_COMM_ACCEPT is called although the specified value for the MCA parameter mpi_no_establish_communication is 1.

- メッセージの説明

  MCAパラメーターmpi_no_establish_communicationに1を指定したにもかかわらず、コミュニケータを共有しないグループ間での通信の確立を行おうとしました。MPIプログラムの実行を終了します。

- 利用者の処置

  MCAパラメーターの指定値と、コミュニケータを共有しないグループ間での通信の確立(MPI_COMM_CONNECTルーチンまたはMPI_COMM_ACCEPTルーチン)を使用していないかどうかを、確認してください。

[mpi::dpm::violated-spawn] MPI_COMM_SPAWN or MPI_COMM_SPAWN_MULTIPLE is called although the specified value for the MCA parameter mpi_no_establish_communication is 1.

- メッセージの説明

  MCAパラメーターmpi_no_establish_communicationに1を指定したにもかかわらず、動的プロセス生成を実行しようとしました。MPIプログラムの実行を終了します。

- 利用者の処置

  MCAパラメーターの指定値と、動的プロセス生成(MPI_COMM_SPAWNルーチンまたはMPI_COMM_SPAWN_MULTIPLEルーチン)を使用していないかどうかを、確認してください。

[mpi::errmgr-base::orte-error] [[*jobid*,*snum*],*rank*] ORTE_ERROR_LOG: *error* in file *file* at line *lineno*

- メッセージの説明

  内部エラーを検知しました。MPIプログラムの実行を終了します。

- パラメーターの説明

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

  error : エラー詳細

  file : エラーファイルパス

  lineno : 行番号

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

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

- メッセージの説明

  引数に指定されたコミュニケータとランクとタグから作成された通信要求がすでに存在します。

- 利用者の処置

  FJMPI_PREQUEST_SEND_INITルーチンまたはFJMPI_PREQUEST_RECV_INITルーチンに指定したタグ、コミュニケータ、ランクを見直してください。または、すでに存在する通信要求をMPI_REQUEST_FREEルーチンで開放してから、FJMPI_PREQUEST_SEND_INITルーチンまたはFJMPI_PREQUEST_RECV_INITルーチンを実行してください。

[mpi::mpi-api::mpi-abort]
MPI_ABORT was invoked on rank *rank* in communicator *comm*
with errorcode *rc*.

NOTE: invoking MPI_ABORT causes Open MPI to kill all MPI processes.
You may or may not see output from other processes, depending on
exactly when Open MPI kills them.

- メッセージの説明

  MPI_ABORTルーチンが呼ばれました。MPIプログラムの実行を終了します。

- パラメーターの説明

  rank : ランク

  comm : MPI_ABORTルーチンの第1引数のコミュニケータの詳細情報

  rc : MPI_ABORTルーチンの第2引数

- 利用者の処置

  MPIプログラムの内容に誤りがないか確認してください。

[mpi::mpi-runtime::mpi_init: already finalized]

Open MPI has detected that this process has attempted to initialize MPI (via MPI_INIT or MPI_INIT_THREAD) after MPI_FINALIZE has been called.
This is erroneous.
--

- メッセージの説明

  MPI_FINALIZEルーチンの後で、MPIの規格上呼び出してはならないMPIルーチンが呼び出されました。

- 利用者の処置

  プログラム中でMPI_FINALIZEルーチンの後に、MPIルーチンを呼び出していないかをご確認ください。

  もし呼び出している場合はMPIの規格に反しますので、プログラムを修正してください。

  ただし、下記のルーチンは、MPI_FINALIZEルーチンの後でも呼び出し可能です。

  ー MPI_INITIALIZEDルーチン

  ー MPI_FINALIZEDルーチン

  ー MPI_GET_VERSIONルーチン

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

- メッセージの説明

  MPI_INITルーチンとMPI_INIT_THREADルーチンが合計2回以上呼び出されました。

- 利用者の処置

  プログラム中で、MPI_INITルーチンとMPI_INIT_THREADルーチンの呼出し回数が合計2回以上でないか、ご確認ください。
  MPIの規格上、MPI_INITルーチンとMPI_INIT_THREADルーチンは、どちらかを1回しか呼び出すことができません。もし呼び出されていた場合、プログラムを修正してください。

--
[mpi::mpi-runtime::mpi-param-check-enabled-but-compiled-out]
WARNING: The MCA parameter mpi_param_check has been set to true, but parameter checking has been compiled out of Open MPI. The mpi_param_check value has therefore been ignored.
--

- メッセージの説明

  MCAパラメーター「mpi_param_check」が設定されました。MCAパラメーターチェックは、デバッグ用MPIライブラリでなければ指定できません。メッセージの出力後、MPIプログラムの実行を継続します。

- 利用者の処置

  MCAパラメーター「mpi_param_check」の指定を解除してください。

--
[mpi::mpi-errors::mpi_errors_are_fatal]
[*info*] * An error occurred [*msg*]**
[*info*] * reported by process [[*jobid*,*rank*]]**
[*info*] * on [*type*]**
[*info*] * [*error class*]**
[*info*] * MPI_ERRORS_ARE_FATAL (processes in this [*type*] will now abort,**

[*info*] *** and potentially your MPI job)

- メッセージの説明

  MPIプログラム実行中に重大な問題が発生しました。プログラムを終了します。

- パラメーターの説明

  info : ホスト名とpidの情報

  msg : 問題となった現象の説明

  jobid : MPIジョブID

  rank : ランク

  type : コミュニケータ、ファイル、またはウィンドウの情報(問題となった現象に依存します)

  error class : “付録A エラークラス一覧”をご覧ください

- 利用者の処置

  msgと***error class***から、プログラムに問題がないかをご確認ください。もし問題がない場合は、出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::opal-free-list::memory-error] Out of memory.

- メッセージの説明

  メモリの獲得に失敗しました。MPIプログラムの実行を終了します。

- 利用者の処置

  ー メモリの使用量およびプログラムの最大メモリサイズリミット値を確認してください。メモリの使用量が多い場合は、削減してください。

  ー MPIライブラリは通信中に処理しきれないデータをメモリ上に退避するため、退避した量に応じたメモリを使用することになります。このため、通信相手となるプロセス数が高速型通信モードで通信を行う通信相手プロセス数の上限を超えていて、Unexpected messageが大量に発生しているようなMPIプログラムでは、以下の手順で通信を高速化することで回避できる場合があります。

  1. MCAパラメーターmpi_print_statsに2を指定して再実行します。MCAパラメーターmpi_print_stats_ranksを指定しないでください。指定する場合は-1を設定し、すべての並列プロセスからの出力を有効にしてください。MPI統計情報を出力しUnexpected messageの数を確認してください。MPI統計情報の詳細については“6.16 MPI統計情報”をお読みください。
  2. Unexpected messageの数が増加していれば、Medium受信バッファのサイズを大きくしてください。Medium受信バッファのサイズの変更方法は“表4.22 common_tofu_medium_recv_buf_size (Medium受信バッファのサイズを変更)”をお読みください。

[mpi::opal-util::check-buffer-write] The buffer was destroyed in this process.
<スタックのトレース情報>

- メッセージの説明

  ノンブロッキング通信の送信バッファに、別の書込みが発生しました。スタックのトレース情報を出力してMPIプログラムの実行を終了します。

- 利用者の処置

  スタックのトレース情報を参考に、ノンブロッキング通信の送信バッファに上書きするような処理がないかMPIプログラムを見直してください。

[*info*] Possible hang-up (no progress) is detected on [[*jobid*,*snum*],*rank*]
<スタックのトレース情報>
[*data*]

- メッセージの説明

  通信待ち時間が、利用者が指定した上限値(秒)を超えました。デッドロック状態に陥った可能性がありますので、スタックのトレース情報と担当保守員(SE)による解析のためのデータを出力してMPIプログラムの実行を終了します。

- パラメーターの説明

  info : ホスト名とpidの情報

  jobid : MPIジョブID

  snum : spawn番号

  rank : ランク

  data : 担当保守員(SE)による解析のためのデータ

- 利用者の処置

  スタックのトレース情報を参考に、デッドロック状態を引き起こすような記述がないか、MPIプログラムを見直してください。

[mpi::opal-util::dynamic-debug-failure] Internal error. [*reason*]

- メッセージの説明

  動的デバッグ機能の実行に失敗しました。

- パラメーターの説明

  reason : 失敗の内容

- 利用者の処置

  出力されたメッセージと合わせて担当保守員(SE)にご連絡ください。

[mpi::opal-util::dynamic-debug-memory-error] Unable to allocate memory. [*errno*]

- メッセージの説明

  動的デバッグ機能が使用するメモリの獲得に失敗しました。

- パラメーターの説明

  errno : エラー番号

- 利用者の処置

  メモリの使用量を確認してください。問題がない場合、システムが正しく動作していない可能性がありますので、システム管理者にご連絡ください。

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

- メッセージの説明

  *time*で指定された時間(秒)の間、プログラムの終了を遅延させます。

- パラメーターの説明

  info : ホスト名とpidの情報

  time : MCAパラメーターopal_abort_delayに指定した値

7.4 翻訳/結合コマンドのエラーメッセージ

comm unrecognized option: -showme:*param*

- メッセージの説明

  翻訳/結合コマンドに誤ったパラメーターが指定されました。

- パラメーターの説明

  comm : 翻訳/結合コマンド

  param : 該当パラメーター

・ 利用者の処置

-showmeオプションに正しいパラメーターを指定してください。

第8章 ブロッキング集団通信の高速化

この章では、高度なチューニングを行う方法として、ブロッキング集団通信ルーチンのアルゴリズムのパラメーターチューニング方法とアルゴリズム選択方法について説明しています。このため、集団通信について以下の知識を有するユーザーが対象となります。

- 本処理系で使用可能なMCAパラメーターの機能を理解していること
- 各アルゴリズムとその特徴および性能特性を理解していること
- “6.14 集団通信のアルゴリズムとコミュニケータに割り当てられた計算ノードの形状”を理解していること
- 性能分析ができること

注意

アルゴリズムの種類と特徴については、以下の参考文献およびOpen MPIのソースコード内のompi/mca/coll/baseディレクトリも参照してください。

- Performance analysis of MPI collective operations

  “https://link.springer.com/article/10.1007/s10586-007-0012-0”
- The design of ultra scalable MPI collective communication on the K computer

  “https://link.springer.com/article/10.1007/s00450-012-0211-7”

8.1 概要

本システムには、1つのブロッキング集団通信ルーチンに対して、1つまたは複数のアルゴリズムが実装されています。本システムでは、ブロッキング集団通信ルーチンが呼び出された際に、集団通信ルーチンの引数やコミュニケータの形状の情報をもとに、高速なアルゴリズムを選択します。アルゴリズムによっては、パラメーターを変更することによって、そのアルゴリズムの性能が変化します。このように、アルゴリズムの選択とアルゴリズム自体のパラメーターを変化させることによって性能が変化します。

8.2 アルゴリズムのMCAパラメーターチューニング

アルゴリズムの中には、MCAパラメーターを変化させることによって、性能が変化するものがあります。

8.2.1 セグメントサイズの変更

8.2.1.1 セグメントサイズの変更によるアルゴリズムの性能の変化

アルゴリズムの中には、パイプライン転送を行うものがあります。パイプライン転送は、メッセージを一定サイズのセグメントごとに分割して転送する方法です。パイプライン転送の性能は、メッセージサイズとセグメントサイズに依存します。本システムでは、メッセージサイズごとに、性能が引き出せるようにセグメントサイズを調整しています。

指定可能なMCAパラメーターは、coll_select_allreduce_algorithm_segmentsize、coll_select_bcast_algorithm_segmentsize、およびcoll_select_reduce_algorithm_segmentsizeの3つです。MCAパラメーターと対応するアルゴリズムは下表の通りです。

表8.1 セグメントサイズを指定するMCAパラメーターと指定可能なアルゴリズムの一覧

MCAパラメーター	集団通信ルーチン名	指定可能なアルゴリズム
coll_select_allreduce_algorithm_segmentsize	MPI_ALLREDUCE	segmented_ring trinaryx6 (trix6) trinaryx3 (trix3)
coll_select_bcast_algorithm_segmentsize	MPI_BCAST	chain pipeline

MCAパラメーター	集団通信ルーチン名	指定可能な アルゴリズム
		split_binary_tree binary_tree binomial knomial trinaryx6 (trix6) bintree3d (bin3d) trinaryx3 (trix3) bintree6d (bin6d)
coll_select_reduce_algorithm_segmentsize	MPI_REDUCE	chain pipeline binary binomial in-order_binary trinaryx6 (trix6) trinaryx3 (trix3)

使用する際には、各集団通信ルーチンでのアルゴリズムを指定するMCAパラメーターと併用して使用してください。

例

MPI_BCASTルーチンのbinomialアルゴリズムのセグメントサイズを1024バイトと指定した例

MPI_BCASTルーチンにおいては、セグメント分割を可能にするために、MCAパラメーターcoll_tuned_bcast_same_countに1を指定してください。MCAパラメーターcoll_tuned_bcast_same_countについては“表4.14 coll_tuned_bcast_same_count (ランク間で同じ要素数を用いたMPI_BCASTルーチン/MPI_IBCASTルーチンの通信を高速化)”をお読みください。

```
$ mpiexec --mca coll_tuned_bcast_same_count 1 ¥
--mca coll_select_bcast_algorithm binomial ¥
--mca coll_select_bcast_algorithm_segmentsize 1024  ./a.out
```

8.2.1.2 セグメントサイズの変更による注意事項

セグメントサイズを変化させることによって、パイプライン転送の回数が変化します。特に、セグメントサイズを小さくする場合、パイプライン転送の回数は増加します。パイプライン転送が増加した結果、通信資源の不足による異常終了が発生する可能性があります。通信資源の不足により発生した異常終了は以下の文字列から始まるエラーメッセージを出力します。

```
[mpi::common-tofu::tofu-mrq-overflow]
```

セグメントサイズの変更により、上記のエラーメッセージが出るようになった場合、セグメントサイズを増やすようにしてください。

セグメントサイズの上限値は16776960です。セグメントサイズの変更については、“8.4.3 アルゴリズム自体をチューニングするMCAパラメーター”をお読みください。

8.3 アルゴリズム選択によるチューニング

本システムでは、アルゴリズムをユーザーが指定できる機能を提供します。本機能は、ブロッキング集団通信の性能が変化する機能です。ここでは、アルゴリズムの選択方法、確認方法、注意事項を説明します。

アルゴリズム選択によりブロッキング集団通信の性能が変化します。ただし、必ず性能が向上するとは限りません。ユーザーの責任において利用してください。

8.3.1 アルゴリズム選択方法

アルゴリズムは以下の2つの処理を繰り返すことによって選択されます。

1. アルゴリズムの候補を選択

   複数のアルゴリズムの中から、条件に応じて、適切なアルゴリズムの候補を選択します。

2. アルゴリズムが呼出し可能であるかの判定

   1で選択したアルゴリズムの候補を実行しても問題がないかを判定します。アルゴリズムによっては、条件判定が存在しない場合もあります。

ここでは、本システムでのアルゴリズム選択のフロー、フロー中に出てくる複数のアルゴリズム選択方法、およびアルゴリズムの呼出し可能かの判定、について説明します。

8.3.1.1 アルゴリズム選択のフロー

本システムの中でどのようにアルゴリズムが選択されるかのフローを説明します。

アルゴリズムの選択方法には5種類あります。選択方法には“表8.2 アルゴリズム選択のフロー”に示す優先度があり、優先度の高い選択方法からアルゴリズムの選択を行います。

以下の条件のいずれかを満たした場合は、ユーザーが指定したアルゴリズム選択方法は適用できません。

1. コミュニケータのサイズが1の場合
2. コミュニケータがMPI_INTERCOMM_MERGEルーチンを使用して作成された場合
3. コミュニケータがグループ間コミュニケータ(inter-communicator)の場合
4. ユーザーがアルゴリズムを指定したブロッキング集団通信が、別のブロッキング集団通信の内部処理として行われる場合

また、アルゴリズムの選択方法には、ユーザーが指定可能な選択方法が3種類あります。ただし、以下のブロッキング集団通信ルーチンでは、アルゴリズムが1種類しかないため、アルゴリズムを指定することはできません。

- MPI_ALLTOALLW
- MPI_EXSCAN
- MPI_REDUCE_SCATTER_BLOCK

表8.2 アルゴリズム選択のフロー

優先度	選択方法	概要	ユーザー指定の可否	詳細内容の参照場所
1	特別な場合の選択	特定の条件を満たした場合、本システムによって自動的に選択される。	×	“8.3.1.2 特別な場合のアルゴリズム選択”
2	MCAパラメーターによる指定	MCAパラメーターcoll_select_(*集団通信の種別*)_algorithmを指定することで、アルゴリズムを指定する。	○	“8.3.1.3 MCAパラメーターによるアルゴリズム選択”
3	Infoオブジェクトによる指定	プログラムを修正し、Infoオブジェクトを使用することで、アルゴリズムを指定する。Infoオブジェクトの指定方法は2種類存在する。	○	“8.3.1.4 Infoオブジェクトによるアルゴリズム選択”
4	外部入力ファイルによる指定	外部入力ファイルにて、アルゴリズムの選択ルールを作成し、実行時にMCAパラメーターを指定することでアルゴリズムを指定する。	○	“8.3.1.5 外部入力ファイルによるアルゴリズム選択”
5	本システムによる選択	本システムによって自動的に選択される。	×	なし

ユーザーが指定可能なアルゴリズムの選択方法を指定しない場合は、本システムによる自動的なアルゴリズム選択となります。この場合、実行時の条件によっては最適なアルゴリズムが選ばれるとは限りません。そのため、ユーザーが指定可能なアルゴリズムの選択方法を使用することによって、性能が改善できることがあります。

下表は、ユーザーが指定可能な3種類のアルゴリズムの選択方法のメリットとデメリットをまとめた表です。以降の節で詳細を説明します。

表8.3 ユーザーが指定可能なアルゴリズムの選択方法のメリットとデメリット

選択方法	メリット	デメリット
MCAパラメーターによる指定	再翻訳不要である。	アプリケーションプログラム全体に、指定したアルゴリズムが適用される。
Infoオブジェクトによるアルゴリズム指定	集団通信ルーチンごとにアルゴリズムを指定可能である。	ソースファイルの修正・再翻訳が必要である。
外部入力ファイルによるアルゴリズム指定	再翻訳不要である。 条件ごとにアルゴリズムを指定可能である。	入力ファイル作成が必要であるため、入力ファイルの作成に時間がかかる可能性がある。

8.3.1.2 特別な場合のアルゴリズム選択

本選択方法は、特殊な条件がある場合に、他の選択方法よりも優先的に選択を行います。特殊な場合とは、以下の3つのいずれかの場合を指します。

1. バリア通信が選択可能な場合
2. リダクション演算が、順序交換不可能な場合(以下のいずれかの条件を満たした場合)
    a. ユーザー定義演算作成時に、ユーザーが演算を非可換であると指定した場合
    b. ユーザーがMCAパラメーターcoll_base_reduce_commute_safeに1を指定した場合
3. MPI_ALLTOALLルーチン、MPI_ALLTOALLVルーチンにおいて送信バッファにMPI_IN_PLACEが指定されていた場合

このいずれかの条件を満たした場合は、あらかじめ本システムで実装しているアルゴリズム選択を行います。1の条件の場合は、常にバリア通信によるアルゴリズムを選択します。バリア通信の適用に必要な条件につきましては"6.12 Tofuバリア通信による高速化"をお読みください。2のいずれかの条件を満たす場合は、特別なアルゴリズムを選択します。MCAパラメーターcoll_base_reduce_commute_safeについては、"表4.7 coll_base_reduce_commute_safe (リダクション演算の順序を保証)"をお読みください。3の条件の場合は、MPI_IN_PLACEに対応したアルゴリズムを選択します。

8.3.1.3 MCAパラメーターによるアルゴリズム選択

ブロッキング集団通信ルーチンが常に同じ引数で呼ばれる場合に有効な選択方法です。MCAパラメーターによるアルゴリズム選択を使用すると、ある集団通信のアルゴリズムが、すべてMCAパラメーターで指定されたアルゴリズムになります。この選択方法では、アプリケーションプログラムの再翻訳は不要です。

例

MCAパラメーターによるアルゴリズム選択の例

```
$ mpiexec --mca coll_select_allgather_algorithm bruck ./a.out
```

この例では、MPI_ALLGATHERルーチンのアルゴリズムとして常にbruckアルゴリズムを選択します。MCAパラメーターに指定可能な値については、"8.4.1 アルゴリズム選択を指定するMCAパラメーター"を参照してください。

8.3.1.4 Infoオブジェクトによるアルゴリズム選択

本システムでは、Infoオブジェクトを使用した、アルゴリズム選択方法をサポートしています。この選択方法でブロッキング集団通信ルーチンのチューニングを有効にするためには、使用するアプリケーションプログラムを、以下の手順で修正する必要があります。修正したアプリケーションプログラムは再翻訳が必要です。

1. Infoオブジェクトを作成
2. Infoオブジェクトにkeyとvalueを設定
3. コミュニケータにInfoオブジェクトを設定
4. チューニングしたいブロッキング集団通信ルーチンに、3で設定したコミュニケータを引数に指定

5. Infoオブジェクトを解放

例

Infoオブジェクトによるアルゴリズム選択の疑似コードの例

Infoオブジェクトによるアルゴリズム選択の疑似コードです。2行目からの各行は、上記の1から5に対応しています。

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, key, value);
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm);
MPI_Info_free(&info);
```

Infoオブジェクトでは、keyとvalueが指定できます。アルゴリズム選択において、keyは、アルゴリズムを指定したいブロッキング集団通信ルーチン名とその使用範囲を設定できます。valueには、指定するアルゴリズム名とそれを使用するメッセージサイズの範囲を設定できます。また、一部のアルゴリズムにおいては、valueにセグメントサイズも指定できます。

keyに指定する文字列については“8.3.1.4.1 Infoオブジェクトでのkeyで指定するパラメーター”をお読みください。valueに指定する文字列については“8.3.1.4.4 アルゴリズム選択ルールの指定”をお読みください。

8.3.1.4.1 Infoオブジェクトでのkeyで指定するパラメーター

アルゴリズム選択ルールの使用範囲を設定するためには、Infoオブジェクトに渡すkeyを指定する必要があります。keyに指定できる使用範囲は、集団通信ルーチン呼出しごとの指定とコミュニケータごとの指定の2種類があります。

8.3.1.4.2 集団通信ルーチン呼出しごとの指定

ある集団通信ルーチンにおいて、特定のアルゴリズムを1回だけ使用する場合に、(*集団通信の種別*)_oneshot_ruleを指定します。集団通信の種別の指定については、“表8.9 外部入力ファイルで指定可能な集団通信の種別”を参照してください。

例

Infoオブジェクトを使用した集団通信ルーチンごとのアルゴリズムの指定例

集団通信ルーチンのアルゴリズムを一度だけ変更する場合の擬似コードになります。

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "allgather_oneshot_rule", "bruck");
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm); //(1)
MPI_Allgather(..., comm); //(2)
MPI_Info_free(&info);
```

(*集団通信の種別*)_oneshot_ruleは、MPI_COMM_SET_INFOルーチンの直後に指定した集団通信ルーチンに対してのみ有効なアルゴリズムの選択方法です。この例において、(1)の行のMPI_ALLGATHERルーチンで、bruckアルゴリズムを実行します。しかし、(1)の行を実行した後は、(*集団通信の種別*)_oneshot_ruleの指定は無効になるため、(2)の行のMPI_ALLGATHERルーチンでは、異なる手段によるアルゴリズム選択が行われます。

指定可能なkeyについては“表8.5 集団通信ルーチンのアルゴリズムを一度だけ変更する場合のkey”をお読みください。

8.3.1.4.3 コミュニケータごとの指定

集団通信ルーチンのアルゴリズムをコミュニケータ単位で変更する場合に、(*集団通信の種別*)_rulesを指定します。集団通信の種別の指定については、“表8.9 外部入力ファイルで指定可能な集団通信の種別”を参照してください。

例

Infoオブジェクトを使用したコミュニケータごとのアルゴリズムの指定例

ある集団通信ルーチンのアルゴリズムをコミュニケータ単位で変更する場合の擬似コードになります。

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "allgather_rules", "bruck");
MPI_Comm_set_info(comm, info);
MPI_Allgather(..., comm); //(1)
MPI_Allgather(..., comm); //(2)
MPI_Info_free(&info);
```

この例において、(1)の行のMPI_ALLGATHERルーチンで、bruckアルゴリズムを実行します。(*集団通信の種別*)_oneshot_ruleとは異なり、(2)の行のMPI_ALLGATHERルーチンでもbruckアルゴリズムを実行します。

指定可能なkeyについては"表8.6 集団通信ルーチンのアルゴリズムをコミュニケータ単位で変更する場合のkey"をお読みください。

8.3.1.4.4 アルゴリズム選択ルールの指定

ここでは、valueでアルゴリズムの呼出し条件を指定する方法について説明します。アルゴリズムの呼出し条件の指定には、以下の2種類があります。

1. (*アルゴリズム名*)

   メッセージサイズに関係なく、指定したアルゴリズムが常に呼ばれます。

2. (*メッセージサイズの範囲*):(*アルゴリズム名*)

   指定したメッセージサイズの範囲に限り、指定したアルゴリズムが呼ばれます。

この指定をする場合には、以下の点にご注意ください。

- (*集団通信の種別*)_oneshot_ruleがkeyに指定されている場合は、valueに複数のアルゴリズムを指定しても、最初のアルゴリズムだけが有効になります。集団通信の種別の指定については、"表8.9 外部入力ファイルで指定可能な集団通信の種別"を参照してください。

指定可能なアルゴリズム名は、MCAパラメーターで指定可能な文字列と同じです。詳細は、"8.4.1 アルゴリズム選択を指定するMCAパラメーター"をお読みください。

アルゴリズムによってはパラメーターを設定できるものがあります。現在指定可能なパラメーターはセグメントサイズのみです。セグメントサイズと指定可能なアルゴリズムについては"8.2 アルゴリズムのMCAパラメーターチューニング"をお読みください。セグメントサイズを指定する場合は、以下のようにvalueを指定してください。

例

Infoオブジェクトを使用してセグメントサイズを指定するアルゴリズムの指定例

セグメントサイズを指定可能なアルゴリズムに、セグメントサイズを指定した疑似コードの例です。

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "bcast_rules", "chain(segsize=2048)");
MPI_Comm_set_info(comm, info);
MPI_Bcast(..., comm); //chainアルゴリズムがセグメントサイズ2048バイトで実行される
MPI_Info_free(&info);
```

例

Infoオブジェクトを使用してメッセージサイズを指定するアルゴリズムの指定例

メッセージサイズを指定した疑似コードの例です。

```
MPI_Info info;
MPI_Info_create(&info);
MPI_Info_set(info, "bcast_rules", "1024:chain:2048-4096:binomial");
MPI_Comm_set_info(comm, info);
MPI_Bcast(..., comm); //メッセージサイズが1024バイトのMPI_BCASTルーチンで、chainアルゴリズムが実行される
MPI_Bcast(..., comm); //メッセージサイズが3072バイトのMPI_BCASTルーチンで、binomialアルゴリズムが実行される
MPI_Info_free(&info);
```

この例では、メッセージサイズ(要素数 × データ型のサイズ)が1024バイトの時はchainアルゴリズムを使います。また、2048バイト以上4096バイト以下ではbinomialアルゴリズムを使います。それ以外のメッセージサイズの場合、異なる方法でアルゴリズムを選択します。

valueに指定可能なメッセージサイズの定義について説明します。メッセージサイズの定義は、以下の"表8.4 Infoオブジェクトで使用するブロッキング集団通信ルーチン別メッセージサイズの定義"の通りです。ただし、使用不可能と書いてある集団通信ルーチンでは、メッセージサイズを使用しない、あるいはメッセージサイズがランク間で異なり定義ができないため、valueにメッセージサイズの指定ができません。該当する集団通信ルーチンで、メッセージサイズの指定をしても無効となります。

表8.4 Infoオブジェクトで使用するブロッキング集団通信ルーチン別メッセージサイズの定義

集団通信ルーチン名	メッセージサイズの定義
MPI_ALLGATHER	要素数 × データ型のサイズ × コミュニケータのサイズ
MPI_ALLGATHERV	要素数の総和 × データ型のサイズ
MPI_ALLREDUCE	要素数 × データ型のサイズ
MPI_ALLTOALL	要素数 × データ型のサイズ × コミュニケータのサイズ
MPI_ALLTOALLV	使用不可能
MPI_BARRIER	使用不可能
MPI_BCAST	要素数 × データ型のサイズ
MPI_GATHER	送信要素数 × データ型のサイズ × コミュニケータのサイズ
MPI_GATHERV	rootランクの受信要素数の総和 × データ型のサイズ
MPI_REDUCE	要素数 × データ型のサイズ
MPI_REDUCE_SCATTER	要素数 × データ型のサイズ × コミュニケータのサイズ
MPI_SCAN	要素数 × データ型のサイズ
MPI_SCATTER	受信要素数 × データ型のサイズ × コミュニケータのサイズ
MPI_SCATTERV	rootランクの送信要素数の総和 × データ型のサイズ

次にメッセージサイズごとにアルゴリズム選択を変更する方法について説明します。メッセージサイズの指定方法は2種類あります。

1. 特定のメッセージサイズ

   特定のメッセージサイズのときのみ、指定のアルゴリズムを選択する

2. msgmin - msgmax

   msgminバイト以上msgmaxバイト以下のメッセージサイズなら、指定のアルゴリズムを選択する

複数のアルゴリズムを選択する場合は、以下の"Infoオブジェクトによるメッセージサイズを指定するアルゴリズム選択"のようにvalueを記載します。valueに、アルゴリズムの呼出し条件を";"で区切った文字列を記載します。

Infoオブジェクトによるメッセージサイズを指定するアルゴリズム選択

```
value = "(アルゴリズムの呼出し条件1); … ;(アルゴリズムの呼出し条件N)"
```

8.3.1.4.5 Infoオブジェクトによるアルゴリズム選択の注意事項

ここでは、Infoオブジェクトによるアルゴリズム選択の注意事項を記載します。

- Infoオブジェクトにはコミュニケータの全プロセスで等しいkeyとvalueを指定してください。

- valueとして入力可能な文字列は、MPI_MAX_INFO_VALとして定義されている文字列長までとなります。本システムでは半角英数字255文字未満となります。
- 一度だけアルゴリズムを変更する(*集団通信の種別*)_oneshot_ruleは使用後、本システム側で(*集団通信の種別*)_oneshot_ruleのkeyとvalueを削除します。このため、再度(*集団通信の種別*)_oneshot_ruleを使用したい場合は、コミュニケータに対してInfoオブジェクトを再度設定してください。集団通信の種別の指定については、“表8.9 外部入力ファイルで指定可能な集団通信の種別”を参照してください。
- MPI_GATHERVルーチンとMPI_SCATTERVルーチンにおいては、本システムがメッセージサイズを知るために、内部でMPI_BCASTルーチン相当の処理を行います。このため、実行時間がかかります。

8.3.1.4.6 Infoオブジェクトのkeyに指定可能な値一覧

ここでは、Infoオブジェクトに指定可能なkeyを記載します。

表8.5 集団通信ルーチンのアルゴリズムを一度だけ変更する場合のkey

key	意味
allgather_oneshot_rule	MPI_ALLGATHERルーチンのアルゴリズムを一度だけ固定します。
allgatherv_oneshot_rule	MPI_ALLGATHERVルーチンのアルゴリズムを一度だけ固定します。
allreduce_oneshot_rule	MPI_ALLREDUCEルーチンのアルゴリズムを一度だけ固定します。
alltoall_oneshot_rule	MPI_ALLTOALLルーチンのアルゴリズムを一度だけ固定します。
alltoallv_oneshot_rule	MPI_ALLTOALLVルーチンのアルゴリズムを一度だけ固定します。
barrier_oneshot_rule	MPI_BARRIERルーチンのアルゴリズムを一度だけ固定します。
bcast_oneshot_rule	MPI_BCASTルーチンのアルゴリズムを一度だけ固定します。
gather_oneshot_rule	MPI_GATHERルーチンのアルゴリズムを一度だけ固定します。
gatherv_oneshot_rule	MPI_GATHERVルーチンのアルゴリズムを一度だけ固定します。
reduce_oneshot_rule	MPI_REDUCEルーチンのアルゴリズムを一度だけ固定します。
reduce_scatter_oneshot_rule	MPI_REDUCE_SCATTERルーチンのアルゴリズムを一度だけ固定します。
scan_oneshot_rule	MPI_SCANルーチンのアルゴリズムを一度だけ固定します。
scatter_oneshot_rule	MPI_SCATTERルーチンのアルゴリズムを一度だけ固定します。
scatterv_oneshot_rule	MPI_SCATTERVルーチンのアルゴリズムを一度だけ固定します。

表8.6 集団通信ルーチンのアルゴリズムをコミュニケータ単位で変更する場合のkey

key	意味
allgather_rules	MPI_ALLGATHERルーチンのアルゴリズムをコミュニケータ単位で固定します。
allgatherv_rules	MPI_ALLGATHERVルーチンのアルゴリズムをコミュニケータ単位で固定します。
allreduce_rules	MPI_ALLREDUCEルーチンのアルゴリズムをコミュニケータ単位で固定します。
alltoall_rules	MPI_ALLTOALLルーチンのアルゴリズムをコミュニケータ単位で固定します。
alltoallv_rules	MPI_ALLTOALLVルーチンのアルゴリズムをコミュニケータ単位で固定します。
barrier_rules	MPI_BARRIERルーチンのアルゴリズムをコミュニケータ単位で固定します。
bcast_rules	MPI_BCASTルーチンのアルゴリズムをコミュニケータ単位で固定します。
gather_rules	MPI_GATHERルーチンのアルゴリズムをコミュニケータ単位で固定します。
gatherv_rules	MPI_GATHERVルーチンのアルゴリズムをコミュニケータ単位で固定します。
reduce_rules	MPI_REDUCEルーチンのアルゴリズムをコミュニケータ単位で固定します。
reduce_scatter_rules	MPI_REDUCE_SCATTERルーチンのアルゴリズムをコミュニケータ単位で固定します。
scan_rules	MPI_SCANルーチンのアルゴリズムをコミュニケータ単位で固定します。
scatter_rules	MPI_SCATTERルーチンのアルゴリズムをコミュニケータ単位で固定します。

key	意味
scatterv_rules	MPI_SCATTERVルーチンのアルゴリズムをコミュニケータ単位で固定します。

8.3.1.5 外部入力ファイルによるアルゴリズム選択

本システムでは、外部入力ファイルによるアルゴリズム選択をサポートしています。外部入力ファイルには、ユーザーがアルゴリズムの選択ルールを“8.3.1.5.5 外部入力ファイルの記載内容”に従って記載をします。ここでは、外部入力ファイルによるアルゴリズム選択の方法について説明します。ここでは、外部入力ファイルの使用方法、3パターンの外部入力ファイルの記載例、外部入力ファイルの記載内容、および外部入力ファイル指定時の注意事項について説明します。

8.3.1.5.1 使用方法

外部入力ファイルに記載したアルゴリズム選択のルールは、MCAパラメーターcoll_select_dectree_fileに指定をすることで実施できます。MCAパラメーターについては、詳細は“8.4.2.1 coll_select_dectree_file(アルゴリズム選択の外部入力ユーザー定義ファイルの指定)”をお読みください。

8.3.1.5.2 外部入力ファイルの記載例

ここでは、基本的な外部入力ファイルの記載例について説明します。

基本的な外部入力ファイル

外部入力ファイルは、「ヘッダ」と「集団通信のルール群」からなります。ヘッダは、外部入力ファイルの先頭に必ず記載してください。

MPI_GATHERルーチンのアルゴリズムを指定した単純な外部入力ファイルの例

```
header:                 <--+
  version: 1.0             | ヘッダ
  require: mtofu        <--+

gather: simple          <--- 集団通信のルール群
```

ヘッダ部分は、このファイルがアルゴリズム選択用のファイルであることを示しています。

アルゴリズム選択のルールのファイルを作成する際に、必ず先頭にこの3行を記載してください。外部入力ファイルに記載されたルールを解析する機能のバージョンナンバーは1.0です。requireには、どのコンポーネントに存在するアルゴリズムを選択するかを宣言するため、コンポーネント情報を指定します。“MPI_GATHERルーチンのアルゴリズムを指定した単純な外部入力ファイルの例”ではTofuインターコネクト向けにチューニングしたアルゴリズムを使用するため、requireにmtofuを記載します。Open MPIのアルゴリズムを使用する場合はbaseを記載してください。両方を使用する場合には、mtofu, baseと記載してください。

集団通信のルール群には、アルゴリズムを指定したい集団通信と指定するアルゴリズムを記載します。ここで記載がなかった集団通信は、本システムがアルゴリズムを選択します。 例えば、1つのアルゴリズムに固定したい場合、以下のように記載します。

```
（集団通信の種別）：（アルゴリズム名）

または

（集団通信の種別）：
    （アルゴリズム名）
```

複数のアルゴリズムを使い分けたい場合は“8.3.1.5.3 アルゴリズムの複数指定とパラメーター指定の記載例”の説明をご覧ください。

“MPI_GATHERルーチンのアルゴリズムを指定した単純な外部入力ファイルの例”の「集団通信のルール群」では、MPI_GATHERルーチンは常にsimpleアルゴリズムを呼ぼうとします。simpleアルゴリズムを呼び出す条件を満たさなかった場合、本システムがアルゴリズムを選択します。

8.3.1.5.3 アルゴリズムの複数指定とパラメーター指定の記載例

ここでは、複数のアルゴリズムの指定およびアルゴリズムのパラメーター指定をするときの記載について説明します。

“8.3.1 アルゴリズム選択方法”の通り、アルゴリズムによっては呼び出す際の条件が存在するため、期待するアルゴリズムが選択されるとは限りません。このため、同じ条件下でも呼び出すアルゴリズムの優先度を変えて記載することができます。

アルゴリズムを複数指定した例

```
header:
  version: 1.0
  require: mtofu, base

gather: simple, binomial
```

この例は、MPI_GATHERルーチンのアルゴリズムを指定しています。最初にsimpleアルゴリズムを呼ぼうとします。simpleアルゴリズムを呼び出せなかった場合には、binomialアルゴリズムが選択されます。このように、左側に記載されたアルゴリズムから順番に判定をしていきます。もし、記載されているすべてのアルゴリズムが選択できない場合は、本システムがアルゴリズムを選択します。

また、アルゴリズムによってはパラメーターを設定できるものがあります。現在指定可能なパラメーターはセグメントサイズのみです。例はMPI_BCASTルーチンのセグメントサイズも指定する選択方法です。セグメントサイズと指定可能なアルゴリズムについては“8.2 アルゴリズムのMCAパラメーターチューニング”をお読みください。

アルゴリズムのパラメーターを指定した例

```
header:
  version: 1.0
  require:base

bcast: binomial(segsize=1024)
```

セグメントサイズが指定されていなかった場合は、0が指定されたものとして処理をします。0が指定された場合は、メッセージを分割せずに通信を行います。この例では、binomialアルゴリズムをセグメントサイズ1024バイトとして実行します。

8.3.1.5.4 条件文の記載例

外部入力ファイルの集団通信のルール群において、選択するアルゴリズムを条件によって変更したい場合があります。この場合の使用可能な条件文の書き方について説明します。条件文は以下の2種類に分かれています。

1. ifとelseを使用した、2つの条件を記載する場合
2. caseとmatchを使用した複数の条件を記載する場合

はじめにifを使用した条件文について説明します。if文は以下のように使用します。必ず、else:の部分が必要となります。

if文の概要

```
  if (条件式) :
    文
  else :
    文
```

if文を使用した例

```
header:
  version: 1.0
  require: base

allreduce:
  if msg_size <= 2097152:
    recursive_doubling
  else:
    ring
```

この例では、以下のようにメッセージサイズによってアルゴリズムの選択が変更されます。

- メッセージサイズが2097152バイト以下の場合はrecursive_doublingアルゴリズムを実行する
- メッセージサイズが2097152バイトより大の場合はringアルゴリズムを実行する

次にcaseとmatchを使用した例を説明します。

caseとmatchの概要

```
    case (式):
        match A .. B:
            文1
        match C .. D:
            文2
        match _:
            文3
```

上記では、式がA以上B以下だと文1を実施し、式がC以上D以下だと文2を実施します。どの範囲にも入らない場合は文3を実施します。match (範囲)の行は3つ以上あっても構いません。ただし、最後はmatch _ :と記載する必要があります。

caseとmatchを使用した例

```
header:
  version: 1.0
  require: mtofu, base

bcast:
  case msg_size:
    match 1 .. 65536
      split_binary_tree
    match 65537 .. 1048576
      bintree3d(segsize = 2048)
    match _:
      bintree3d(segsize = 16384)
```

この例では、以下のようにメッセージサイズによってアルゴリズムの選択が変更されます。

- メッセージサイズが1バイト以上65536バイト以下の場合はsplit_binary_treeアルゴリズムを実行する
- メッセージサイズが65537バイト以上1048576バイト以下の場合はbintree3dアルゴリズムをセグメントサイズ2048バイトで実行する
- メッセージサイズがそれ以外の場合はbintree3dアルゴリズムをセグメントサイズ16384バイトで実行する

case文は前から評価を行っているため、式に合致するmatch文が複数ある場合は、前の条件が優先されます。

8.3.1.5.5 外部入力ファイルの記載内容

ここでは外部入力ファイルの記載内容を説明します。

“基本的な外部入力ファイル”で説明をした通り、外部入力ファイルには、「ヘッダ部分」と「集団通信のルール群」を記載します。

外部入力ファイルの概要

```
header:
    version: 1.0
    (コンポーネント情報)
(集団通信のルール群)
```

ヘッダ部分は“header”という文字列、“: ”、バージョン情報、コンポーネント情報から成り立ちます。バージョン情報は、“version”、“: ”、バージョンナンバーから成り立ちます。外部入力ファイルに記載されたルールを解析する機能のバージョンナンバーは1.0です。

なお、外部入力ファイルの記載では、以下の注意が必要です。

- 単語や記号の間は空白を使用してください
- コメントは#で記述してください。#が記載された後ろの文字すべてが意味のないものと認識されます。

ー 行の先頭から半角空白での字下げも記載可能です。ただし、下例に示すように、字下げの位置を誤って記載した場合は、文法エラーになります。

```
header:
    version: 1.0
    require: base

allreduce:
    if msg_size <= 2097152:
        recursive_doubling
#elseの字下げの位置がこれより前に記載されているifの行と合っていないため
#文法エラーとなる。この場合は、elseをifの字下げの位置に合わせる必要が
#ある。
  else:
        ring
```

詳細な例は、“8.3.1.5.2 外部入力ファイルの記載例”をお読みください。

外部入力ファイルに記載できる項目の記載方法を下表に示します。下表における、記号については以下の通りです。

表8.7 記載方法に表記されている記号と意味

記号	意味
" "	この記号で囲まれた項目名は、記載方法に書かれている書式に従って、ユーザーが選択条件を作成します。
[ ]	[ ]の中で囲んだ内容を1回行います。
[ ]+	[ ]の中で囲んだ内容を1回以上繰り返します。
|	この記号で区切りられている項目の中から、どれか1つを選択します。

下表の記載方法にある" "で囲まれた項目名は、その項目名の記載方法を適用することを示します。

例えば、項目がヘッダの場合は、記載方法が「header: "バージョン情報" "コンポーネント情報"」となっているため、“外部入力ファイルの概要”の例のように、「header:」の後には、バージョン情報およびコンポーネント情報の記載方法の書式に従った内容を外部入力ファイルへ記載します。

表8.8 外部入力ファイルの定義

外部 入力ファイル	項目	記載方法	備考
ヘッダ部分	ヘッダ	header: "バージョン情報" "コンポーネント情報"	
	バージョン情報	version: "バージョンナンバー"	
	バージョンナンバー	1.0	
	コンポーネント情報	require: "コンポーネント場所"	
	コンポーネント場所	mtofu | base | mtofu, base	
集団通信の ルール群	集団通信のルール	["集団通信の種別"] : "文章"	
	集団通信の種別	“表8.9 外部入力ファイルで指定可能な集団通信の種別”を参照してください。	
	文章	"if-else文" | "case-match文" | "アルゴリズム文"	
	if-else文	if "比較文" : "文章" else : "文章"	

外部 入力ファイル	項目	記載方法	備考
	case-match文	case "式" : [match "範囲" :"文章"]+ match _: "文章"	

	範囲	"下限値の正の整数".."上限値の正の整数"
		|"下限値の正の小数".."上限値の正の小数"
			case-match文で式が必要とする範囲を指定します。式によって範囲の意味が異なります。
			
			式がメッセージサイズを表す場合は、以下のように指定します。
			例:1024バイトから2048バイトの範囲を表す指定の場合
			`case msg_size:`
			`  match 1024..2048:`
			
			式がsharpnessの場合は、Tofu座標における軸の面積の比率を以下のように指定します。
			例:1.0から2.0の比率を表す場合
			`case sharpness:`
			`  match 1.0..2.0:`

	アルゴリズム文	"アルゴリズム名" |"アルゴリズム名" , "アルゴリズム文" |"アルゴリズム名" (segsize = "正の整数") |"アルゴリズム名" (segsize = "正の整数") , "アルゴリズム文" |"アルゴリズム名" (segsize = "正の整数" , "正の整数") |"アルゴリズム名" (segsize = "正の整数" , "正の整数") , "アルゴリズム文"	segsizeは原則として1つを指定します。 ただし、MPI_ALLREDUCEルーチンのアルゴリズムのtrinaryx3とtrinaryx6は、内部でreduceとbcastを呼んでいるため、segsizeを2つ記載することができます。 例: `trinaryx3(segsize=4096, 2048)` この例の場合のセグメントサイズは、reduceでは4096バイトであり、bcastでは2048バイトとなります。segsizeを1つだけ指定した場合は、reduceとbcastは同じセグメントサイズとなります。
	比較文	"bool型の変数" |"変数" is even_num |"変数" is odd_num |"式" "比較演算子" "項" |"項" "比較演算子" "式"	比較文に変数だけを指定する場合は、bool型の変数だけを指定してください。 「is even_num」と「is odd_num」の前の"変数"には、bool型やdouble型の変数は指定できません。 "項"には、bool型の変数は指定できません。
	比較演算子	<= |< |>= |>	

外部 入力ファイル	項目	記載方法	備考
		| == | !=	
	項	"正の整数" | "正の小数" | "変数"	"変数"には、bool型の変数は指定できません。
	演算子	+ | - | * | /	
	式	"項" | "項" "演算子" "項" | "項" "演算子" "項" "演算子" "項" | ("項" "演算子" "項") "演算子" "項" | "項" "演算子" ("項" "演算子" "項")	式は項と演算子と括弧で構成されます。項は3つまで記載可能です。演算子は2つまで記載可能です。括弧のくくりは1つだけ記載可能です。 例: `(proc_count + 1024) * msg_size`
	アルゴリズム名	“8.4.1 アルゴリズム選択を指定するMCAパラメーター”の表に記載されているアルゴリズム名を参照してください。	
	変数	“表8.10 外部入力ファイルで指定可能な変数名”を参照してください。	
	bool型の変数	log_cuboid | pow_two | 6d_cuboid | equal_proc	変数の意味については、“表8.10 外部入力ファイルで指定可能な変数名”を参照してください。

外部入力ファイルに指定可能な集団通信の種別を下表に示します。

表8.9 外部入力ファイルで指定可能な集団通信の種別

外部入力ファイルに記載可能な集団通信の種別	対応するブロッキング集団通信ルーチン
allgather	MPI_ALLGATHER
allgatherv	MPI_ALLGATHERV
allreduce	MPI_ALLREDUCE
alltoall	MPI_ALLTOALL
alltoallv	MPI_ALLTOALLV
barrier	MPI_BARRIER
bcast	MPI_BCAST
gather	MPI_GATHER
gatherv	MPI_GATHERV
reduce	MPI_REDUCE
reduce_scatter	MPI_REDUCE_SCATTER
scan	MPI_SCAN

外部入力ファイルに記載可能な集団通信の種別	対応するブロッキング集団通信ルーチン
scatter	MPI_SCATTER
scatterv	MPI_SCATTERV

外部入力ファイルの変数に指定可能な変数名を下表に示します。また、変数名に対するデータ型についても下表に示します。特に、比較文においてbool型の変数に指定可能な変数は、4種類(log_cuboid、pow_two、6d_cuboid、equal_proc)だけとなりますので、ご注意ください。

MPI_GATHERVとMPI_SCATTERVの集団通信のメッセージサイズ(total_msg_size)は、本システムがtotal_msg_sizeを知るために、内部でMPI_BCASTルーチン相当の処理を行います。このため、実行時間がかかります。

表8.10 外部入力ファイルで指定可能な変数名

変数名	データ型	意味
node_count	int32_t	コミュニケータ内のノード数。
proc_count	int32_t	コミュニケータ内のプロセス数。
max_proc	int32_t	コミュニケータ内のノード内の最大プロセス数。
log_cuboid	bool	コミュニケータ内のノードの形状が直方体的か否か(特定の論理軸の抜けは許す)。
x_len、y_len、z_len	int32_t	コミュニケータ内のプロセスが属するノードの各論理軸長。
comm_dim	int32_t	コミュニケータ内のプロセスが存在するノードの形状の次元数をベースとした値。以下の条件で値が決定する。 comm_dim=3: log_cuboidがtrueであり、x_len、y_len、z_lenがすべて2以上の場合 comm_dim=2: log_cuboidがtrueであり、x_len、y_len、z_lenのうち2つが2以上の場合 comm_dim=1: log_cuboidがtrueであり、x_len、y_len、z_lenのうち1つが2以上の場合 comm_dim=0: log_cuboidがfalse、または、x_len、y_len、z_lenがすべて1以下の場合
msg_count	int32_t	自プロセスの要素数。MPI_ALLREDUCEルーチンのみで使用可能。
msg_size	int64_t	集団通信のメッセージサイズ。メッセージサイズの定義については、“表8.11 ブロッキング集団通信ルーチン別msg_sizeの定義”を参照。
total_msg_size	int64_t	集団通信のメッセージサイズの総和。単純にmsg_size * proc_countで表現できない場合に使用する。具体的には、以下の集団通信ルーチンでの値を示す。 ・ MPI_ALLGATHERVルーチンの受信バッファのサイズ ・ MPI_REDUCE_SCATTERルーチンの送信バッファ ・ MPI_GATHERVルーチンのrootランクの受信バッファのサイズ ・ MPI_SCATTERVルーチンのrootランクの送信バッファのサイズ
pow_two	bool	コミュニケータ内のプロセス数が2べき乗か否か。
sharpness	double	鮮鋭度(コミュニケータの物理最長軸と残りの2軸の面積の比)。コミュニケータを内包する形状のTofu座標におけるX軸、Y軸、Z軸に対して、最長軸と残りの軸の面積の比率を表す。
6d_cuboid	bool	コミュニケータ内のノードの形状がTofu座標において直方体的か否か(特定の物理軸の抜けは許す)。
equal_proc	bool	ノード内のプロセス数が均一か否か。
job_topo	int32_t	Tofu座標におけるノードの配置方法の種類。ジョブを実行させる時のノードの配置方法を指定できる。指定しない場合は、ジョブ運用ソフトで設定されているノードの配置方法に従いジョブが実行される。 0: トーラスモード 1: メッシュモード 2: 離散割り当て

変数名	データ型	意味
tni_count	int32_t	各プロセスで使用可能なTNI数。
world_size	int32_t	MPI_COMM_WORLDのプロセス数。

コミュニケータ内のノード形状については“6.14 集団通信のアルゴリズムとコミュニケータに割り当てられた計算ノードの形状”を参考にしてください。また、Tofu座標におけるノード配置方法の詳細については、ジョブ運用ソフトウェアのマニュアルをお読みください。

表8.11 ブロッキング集団通信ルーチン別msg_sizeの定義

集団通信ルーチン名	メッセージサイズの定義
MPI_ALLGATHER	要素数 × データ型のサイズ
MPI_ALLGATHERV	使用不可能
MPI_ALLREDUCE	要素数 × データ型のサイズ
MPI_ALLTOALL	要素数 × データ型のサイズ
MPI_ALLTOALLV	使用不可能
MPI_BARRIER	使用不可能
MPI_BCAST	要素数 × データ型のサイズ
MPI_GATHER	送信要素数 × データ型のサイズ
MPI_GATHERV	使用不可能
MPI_REDUCE	要素数 × データ型のサイズ
MPI_REDUCE_SCATTER	使用不可能
MPI_SCAN	要素数 × データ型のサイズ
MPI_SCATTER	受信要素数 × データ型のサイズ
MPI_SCATTERV	使用不可能

外部入力ファイルの比較文に指定可能な“is”について下表に示します。

表8.12 外部入力ファイルの比較文に指定可能な“is”

指定	意味
式 is even_num	式の値が偶数の場合にtrueを返し、そうでない場合はfalseを返す。
式 is odd_num	式の値が奇数の場合にtrueを返し、そうでない場合はfalseを返す。

8.3.1.5.6 外部入力ファイル指定時の注意事項

ここでは、外部入力ファイルを指定する際の注意事項について説明します。

注意が必要な指定	注意事項
外部入力ファイルに記載可能な入力数	外部入力ファイルに記載可能な入力数には上限があります。入力数は、記載されたアルゴリズムの総数と、記載された条件文の数で決定します。入力数を求める式は以下の通りです。 **入力数 = 記載したアルゴリズムの総数 + if文の数 + case-match文で書かれたcaseとmatchの総和** 本システムに記載可能な入力数の上限は約3500です。MCA パラメーター coll_select_show_decision_processに1か2が指定されている場合、入力数の上限数を超えると、以下の警告文を出力します。 `[mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [2]`
外部入力ファイルに一度に記載可能な条件文の数	外部入力ファイルに一度に記載可能な条件文の数には上限があります。

注意が必要な指定	注意事項
	条件文の数は、if文の数とcase-match文で書かれたmatchの数の和であり、条件文の数は30個までとなります。 MCA パラメーターcoll_select_show_decision_processの指定が1か2の場合、条件文の上限数を超えると、以下の警告文を出力します。 [mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [3]

8.3.1.6 アルゴリズムの適用に必要な条件

ここでは、アルゴリズムの特徴と適用に必要な条件について説明します。アルゴリズムを実行する際には、プロセス数やコミュニケータ形状などの必要条件で判定する場合があります。実行時に、呼ぼうとしているアルゴリズムを実際に呼び出して実行できるかを、本システムが判定します。判定は、以下の2種類に分類されます。

1. 自プロセスだけで判定が可能な場合
2. コミュニケータを構成するすべてのプロセスの判定が必要な場合

1は、プロセスによらず判定が一意に行うことができる場合に使用します。例えば、プロセス数のような、コミュニケータに依存する項目です。

2は、コミュニケータ以外の引数が条件の場合に使用します。例えば、データ型です。MPIの規格において、ほとんどのブロッキング集団通信では、ランク間で異なるデータ型を持つことを条件付きで許されています。このため、自プロセスの引数だけを見て、データ型の条件を満たしているかどうかの判定ができません。引数の条件については、MPIの規格書をお読みください。判定を行う際は、コミュニケータ全体で同じアルゴリズムを選択しなければならないため、コミュニケータ内で判定結果を一致させる必要があります。このため、判定を行う際には、本システムの内部でMPI_ALLREDUCEルーチン相当の処理を実施します。各アルゴリズムの適用に必要な条件につきましては、“8.3.4 アルゴリズムと適用に必要な条件一覧”をお読みください。

8.3.2 選択結果の確認方法

選択したアルゴリズムが想定通り実行されているかを確認するための機能を説明します。

8.3.2.1 アルゴリズム選択過程の表示

MCAパラメーターcoll_select_show_decision_processに1か2を指定すると、標準エラー出力に、アルゴリズムの選択過程が表示されます。MCAパラメーターについては、“8.4.2.2 coll_select_show_decision_process(アルゴリズム選択の過程の出力)”をお読みください。出力されるメッセージについては、“8.5.2 アルゴリズム選択過程表示機能による出力メッセージ(警告)”をお読みください。

8.3.2.2 Infoオブジェクトを使用したアルゴリズムの選択結果の取得

アルゴリズムの選択と同様に、Infoオブジェクトを使用することで、アルゴリズムの選択結果を取得することができます。アルゴリズム選択結果の取得の際は、MCAパラメーターcoll_select_get_tuning_infoに1を指定してください。MCAパラメーターcoll_select_get_tuning_infoについては、“8.4.4.1 coll_select_get_tuning_info(直前に実行した集団通信のアルゴリズム情報を取得)”をお読みください。

例

Infoオブジェクトを使用したアルゴリズム取得の疑似コード

```
char val[MPI_MAX_INFO_VAL];
int flag = 0;
MPI_Info info;
MPI_Allgather(..., comm)//(1)
MPI_Comm_get_info(comm, &info);
MPI_Info_get(info, "last_allgather_algorithm", MPI_MAX_INFO_VAL, val, &flag);
if(flag){//flagが1の時は、Infoを取得できた時
    printf("val is %s¥n",val);//(2)
}
MPI_Info_free(&info);
```

アルゴリズム番号の取得結果

```
val is 5(segsize=0)
val is 1
```

“Infoオブジェクトを使用したアルゴリズム取得の疑似コード”において、(2)の行で出力される値は、(1)の行で実行したアルゴリズムの番号となります。

“アルゴリズム番号の取得結果”の各行は、それぞれ“Infoオブジェクトを使用したアルゴリズム取得の疑似コード”に記載されている(2)の行で出力される値の例です。セグメントサイズが設定可能なアルゴリズムについては、セグメントサイズをsegsizeとして表示します。それ以外の場合は、アルゴリズムの番号のみが出力されます。

“表8.13 アルゴリズム選択の結果取得のためのkey一覧”は、MPI_INFO_GETルーチンで取得するためのkey一覧です。

表8.13 アルゴリズム選択の結果取得のためのkey一覧

key	意味
last_allgather_algorithm	MPI_ALLGATHERルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_allgatherv_algorithm	MPI_ALLGATHERVルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_allreduce_algorithm	MPI_ALLREDUCEルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_alltoall_algorithm	MPI_ALLTOALLルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_alltoallv_algorithm	MPI_ALLTOALLVルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_barrier_algorithm	MPI_BARRIERルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_bcast_algorithm	MPI_BCASTルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_gather_algorithm	MPI_GATHERルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_gatherv_algorithm	MPI_GATHERVルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_reduce_algorithm	MPI_REDUCEルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_reduce_scatter_algorithm	MPI_REDUCE_SCATTERルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_scan_algorithm	MPI_SCANルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_scatter_algorithm	MPI_SCATTERルーチンにおいて、最後に使用したアルゴリズムを取得します。
last_scatterv_algorithm	MPI_SCATTERVルーチンにおいて、最後に使用したアルゴリズムを取得します。

8.3.2.3 MCAパラメーターで結果のみを取得する

MPI統計情報の機能として、ブロッキング集団通信ルーチンで実行されたアルゴリズムを表示可能です。MPI統計情報については、“6.16 MPI統計情報”をお読みください。

8.3.3 アルゴリズム選択の注意事項

1つの計算ノードに同時に数千以上の通信が集中すると、その計算ノード付近のTofuネットワークが混雑し、ネットワーク処理が著しく遅くなることがあります。アルゴリズムによっては、1つのプロセスに同時に通信が集中します。このようなアルゴリズムは、数千プロセス以上の規模では使用しないでください。

また1つのプロセスに通信が集中するような集団通信ルーチンを連続して実行すると、通信資源が枯渇する可能性があります。通信資源の不足により発生した異常終了は以下の文字列から始まるエラーメッセージを出力します。

```
[mpi::common-tofu::tofu-mrq-overflow]
```

連続して実行した結果、上記のメッセージが出力された場合、MCAパラメーターcommon_tofu_memory_saving_methodを2に指定してください。ただし、実行性能が低下する可能性があります。MCAパラメーターcommon_tofu_memory_saving_methodについては“表4.25 common_tofu_memory_saving_method (省メモリ型通信モードで使用する方式を変更)”をお読みください。

バリア通信はユーザーが使用の有無を選択できません。必要に応じて本システムで選択します。

MPI_REDUCEルーチンで、アルゴリズムの選択方法を指定した場合の演算順序は、アルゴリズムの選択方法を指定しない場合の演算順序とは変わります。このため、アルゴリズムの選択方法を指定しない場合と同じ引数で実行をしても、演算結果が異なることがあります。MPI規格では、MPI_REDUCEルーチンは、「同じ引数で実行した場合、同じ結果になることが強く望ましい」とされているため、演算順序を変えないようにする場合は、順序保証モードを使用してください。順序保証モードについては、“6.8 集団通信におけるリダクション演算の順序保証”をお読みください。

8.3.4 アルゴリズムと適用に必要な条件一覧

ここでは、以下に関する情報について説明します。

- MCAパラメーターに対応する、アルゴリズムの出力機能で表示される番号
- アルゴリズム選択として、指定可能なMCAパラメーターの値
- アルゴリズムの適用に必要な条件

アルゴリズムの出力機能で表示される番号には、以下の規則性があります。

番号	内容
1 ～ 6	Open MPIで公開されているアルゴリズム
100 ～ 103	TofuインターコネクトやシステムのCPU向けにチューニングされたアルゴリズム
200	バリア通信
300	本システムで開発したアルゴリズム

8.3.4.1 MPI_ALLGATHERルーチンで選択されるアルゴリズム

表8.14 MPI_ALLGATHERルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_allgather_algorithm	適用に必要な条件
1	linear	なし
2	bruck	なし
3	recursive_doubling	pow_twoが真であること
4	ring	なし
5	neighbor	proc_countが偶数であること
6	two_proc	proc_countが2であること
100	gtbc	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること
101	3dtorus, 3dtorus_fm	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること
102	3dtorus_sm	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること MPI_COMM_WORLDにおけるノード内プロセス数が24以下であること proc_count × max_procが2048未満であること msg_sizeが16776960B以下であること

MPIX_ALLGATHER_INITルーチンを呼んでから作成した要求に対応したMPI_REQUEST_FREEルーチンを呼ぶまでの間は、3dtorus_smアルゴリズムを指定しても使用できない場合があります。この場合は、本システムによって自動的に選択されます。

pow_two、proc_count、log_cuboid、max_proc、およびmsg_sizeについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.2 MPI_ALLGATHERVルーチンで選択されるアルゴリズム

表8.15 MPI_ALLGATHERVルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_allgatherv_algorithm	適用に必要な条件
1	default	なし
2	bruck	なし
3	ring	なし
4	neighbor	proc_countが偶数であること
5	two_proc	proc_countが2であること
100	gtvbc	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること 受信バッファの受信するデータの配置に隙間がないこと
101	3dtorus, 3dtorus_fm	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること
102	3dtorus_sm	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること MPI_COMM_WORLDにおけるノード内プロセス数が24以下であること proc_count × max_procが2048未満であること 受信要素数の最大値と受信データ型のサイズの積が16776960B以下であること

MPIX_ALLGATHER_INITルーチンを呼んでから作成した要求に対応したMPI_REQUEST_FREEルーチンを呼ぶまでの間は、3dtorus_smアルゴリズムを指定しても使用できない場合があります。この場合は、本システムによって自動的に選択されます。

proc_count、log_cuboid、およびmax_procについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.3 MPI_ALLREDUCEルーチンで選択されるアルゴリズム

表8.16 MPI_ALLREDUCEルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_allreduce_algorithm	適用に必要な条件
1	basic_linear	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
2	nonoverlapping	

アルゴリズムの番号	MCAパラメーター coll_select_allreduce_algorithm	適用に必要な条件
		注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
3	recursive_doubling	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
4	ring	msg_countがproc_count以上あること
5	segmented_ring	msg_countがproc_count以上あること セグメントサイズが1以上であること msg_count × データ型のサイズが、セグメントサイズ × proc_countよりも大きなこと
6	rabenseifner	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
100	rdbc	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 演算が定義済み演算であること log_cuboidがtrueであること 注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
101	trinaryx6, trix6	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 演算が定義済み演算であること log_cuboidがtrueであること tni_countがcomm_dim×2以上あること x_len、y_len、z_lenが以下の条件を満たすこと ・ x_lenが1、または3以上、かつ、 ・ y_lenが0、1、または3以上、かつ、 ・ z_lenが0、1、または3以上

アルゴリズムの番号	MCAパラメーター coll_select_allreduce_algorithm	適用に必要な条件
		注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
102	trinaryx3, trix3	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 演算が定義済み演算であること log_cuboidがtrueであること tni_countがcomm_dim以上あること 注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
200	なし	バリア通信のMPI_ALLREDUCEルーチンが呼び出せること (バリア通信の呼出し条件につきましては"6.12 Tofuバリア通信による高速化"をお読みください。)

msg_count、proc_count、log_cuboid、tni_count、comm_dim、x_len、y_len、およびz_lenについては、"表8.10 外部入力ファイルで指定可能な変数名"をお読みください。

8.3.4.4 MPI_ALLTOALLルーチンで選択されるアルゴリズム

表8.17 MPI_ALLTOALLルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_alltoall_algorithm	適用に必要な条件
1	linear	proc_countが1536以下であること 注意 本アルゴリズムは、特定のプロセスに多数の通信が同時に集中する可能性があります。この結果、ネットワーク処理が著しく遅くなる可能性があるため、数千プロセス以上の規模では使用しないでください。
2	pairwise	なし
3	modified_bruck	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
4	linear_sync	なし
5	two_proc	proc_countが2であること
100	doublespread	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること

アルゴリズムの番号	MCAパラメーター coll_select_alltoall_algorithm	適用に必要な条件
101	blacc3d	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること コミュニケータのcomm_dimが3であること コミュニケータのx_len、y_len、z_lenがすべて偶数であること msg_sizeが33553920B以下であること equal_procがtrueであること
102	blacc6d	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 6d_cuboidがtrueであること コミュニケータの形状がTofu座標において、abc軸に隙間がないこと job_topoが2ではないこと equal_procがtrueであること tni_countが6であること **注意** このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
103	crp	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること node_countが1ノードであること

proc_count、log_cuboid、tni_count、comm_dim、x_len、y_len、z_len、msg_size、6d_cuboid、equal_proc、job_topo、およびnode_countについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.5 MPI_ALLTOALLVルーチンで選択されるアルゴリズム

表8.18 MPI_ALLTOALLVルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_alltoallv_algorithm	適用に必要な条件
1	basic_linear	proc_countが1536以下であること **注意** 本アルゴリズムは、特定のプロセスに多数の通信が同時に集中する可能性があります。この結果、ネットワーク処理が著しく遅くなる可能性があるため、数千プロセス以上の規模では使用しないでください。
2	pairwise	なし
100	doublespread	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること

proc_countについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.6 MPI_BARRIERルーチンで選択されるアルゴリズム

表8.19 MPI_BARRIERルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_barrier_algorithm	適用に必要な条件
1	linear	なし
3	recursive_doubling	なし
4	bruck	なし
5	two_proc	proc_countが2であること
6	tree	なし
200	なし	バリア通信のMPI_BARRIERルーチンが呼び出せること (バリア通信の呼出し条件につきましては“6.12 Tofuバリア通信による高速化”をお読みください。)

proc_countについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.7 MPI_BCASTルーチンで選択されるアルゴリズム

表8.20 MPI_BCASTルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_bcast_algorithm	適用に必要な条件
1	basic_linear	なし **注意** これらのアルゴリズムは、連続的に、数千プロセス以上で使用すると通信資源の枯渇が発生する可能性があります。詳細は“8.3.3 アルゴリズム選択の注意事項”をお読みください。
2	chain	
3	pipeline	
4	split_binary_tree	MCAパラメーターcoll_tuned_bcast_same_countの値が1であること 要素数が2以上であること セグメントサイズが、要素数 ÷ 2 × データ型 のサイズ以下であること(要素数が奇数の場合は要素数÷2の小数点以下を切り捨てる)
5	binary_tree	なし
6	binomial	なし
7	knomial	なし
100	trinaryx6, trix6	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること tni_countがcomm_dim×2以上あること x_len、y_len、z_lenが以下の条件を満たすこと ・ x_lenが1、または3以上、かつ、 ・ y_lenが0、1、または3以上、かつ、 ・ z_lenが0、1、または3以上

アルゴリズムの番号	MCAパラメーター coll_select_bcast_algorithm	適用に必要な条件
101	bintree3d, bin3d	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること
102	trinaryx3, trix3	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること
103	bintree6d, bin6d	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること log_cuboidがtrueであること tni_countがcomm_dim×2以上あること
200	なし	バリア通信のMPI_BCASTルーチンが呼び出せること (バリア通信の呼出し条件につきましては“6.12 Tofuバリア通信による高速化”をお読みください。)

log_cuboid、tni_count、comm_dim、x_len、y_len、およびz_lenについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.8 MPI_GATHERルーチンで選択されるアルゴリズム

表8.21 MPI_GATHERルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_gather_algorithm	適用に必要な条件
1	basic_linear	なし
2	binomial	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、ルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
3	linear_sync	なし 注意 ランク間で同じ要素数の場合に高速に実行するためのアルゴリズムです。ランク間で要素数が異なる場合に、実行すると異常終了することがあります。ランク間の要素数が一致することを保証してアルゴリズムを呼び出してください。
100	simple	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること

8.3.4.9 MPI_GATHERVルーチンで選択されるアルゴリズム

表8.22 MPI_GATHERVルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_gatherv_algorithm	適用に必要な条件
1	default	なし

アルゴリズムの番号	MCAパラメーター coll_select_gatherv_algorithm	適用に必要な条件
100	simple	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること

8.3.4.10 MPI_REDUCEルーチンで選択されるアルゴリズム

表8.23 MPI_REDUCEルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_reduce_algorithm	適用に必要な条件
1	default	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
2	chain	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
3	pipeline	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
4	binary	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
5	binomial	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
6	in-order_binary	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
7	rabenseifner	

アルゴリズムの番号	MCAパラメーター coll_select_reduce_algorithm	適用に必要な条件
		注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
100	trinaryx6, trix6	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 演算が定義済み演算であること log_cuboidがtrueであること tni_countがcomm_dim×2以上あること x_len、y_len、z_lenが以下の条件を満たすこと ・ x_lenが1、または3以上、かつ、 ・ y_lenが0、1、または3以上、かつ、 ・ z_lenが0、1、または3以上 注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
101	trinaryx3, trix3	データ型がMPI_PACKEDではなく、隙間のない定義済みデータ型であること 演算が定義済み演算であること log_cuboidがtrueであること tni_countがcomm_dim以上あること 注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファとルートプロセスの受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
200	なし	バリア通信のMPI_REDUCEルーチンが呼び出せること (バリア通信の呼出し条件につきましては“6.12 Tofuバリア通信による高速化”をお読みください。)

log_cuboid、tni_count、comm_dim、x_len、y_len、およびz_lenについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.11 MPI_REDUCE_SCATTERルーチンで選択されるアルゴリズム

表8.24 MPI_REDUCE_SCATTERルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_reduce_scatter_algorithm	適用に必要な条件
1	non-overlapping	

アルゴリズムの番号	MCAパラメーター coll_select_reduce_scatter_algorithm	適用に必要な条件
		注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
2	recursive_halving	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
3	ring	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。

8.3.4.12 MPI_SCANルーチンで選択されるアルゴリズム

表8.25 MPI_SCANルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_scan_algorithm	適用に必要な条件
1	linear	なし
2	recursive_doubling	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、送信バッファと受信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。

8.3.4.13 MPI_SCATTERルーチンで選択されるアルゴリズム

表8.26 MPI_SCATTERルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_scatter_algorithm	適用に必要な条件
1	basic_linear	なし
2	binomial	注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、ルートプロセスの送信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。
300	use_bcast	msg_size × proc_count が2147483647以下であること

アルゴリズムの番号	MCAパラメーター coll_select_scatter_algorithm	適用に必要な条件
		注意 このアルゴリズムでは、MPIライブラリが一時的に多くのメモリを確保します。このため、ルートプロセスの送信バッファのサイズが大きな場合にメモリ不足を引き起こすことがあります。

msg_sizeおよびproc_countについては、“表8.10 外部入力ファイルで指定可能な変数名”をお読みください。

8.3.4.14 MPI_SCATTERVルーチンで選択されるアルゴリズム

表8.27 MPI_SCATTERVルーチンにおける、アルゴリズムの番号と対応するMCAパラメーターとその適用条件

アルゴリズムの番号	MCAパラメーター coll_select_scatterv_algorithm	適用に必要な条件
1	basic_linear	なし
300	linear_sync	なし

8.4 アルゴリズム選択に関連するMCAパラメーター

ここでは、アルゴリズム選択に関連するMCAパラメーターの種類について説明します。MCAパラメーターの指定方法と優先順位については、“表4.5 MCAパラメーターの指定方法と優先順位”をお読みください。

8.4.1 アルゴリズム選択を指定するMCAパラメーター

8.4.1.1 coll_select_allgather_algorithm(MPI_ALLGATHERルーチンのアルゴリズムを指定)

プログラム中のMPI_ALLGATHERルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
3dtorus_sm	Tofuインターコネクト向けにチューニングされたアルゴリズム3dtorus_smを使用します。
3dtorus_fm、3dtorus	Tofuインターコネクト向けにチューニングされたアルゴリズム3dtorus_fmを使用します。3dtorus_fm、3dtorusのどちらの文字列でも有効です。
gtbc	Tofuインターコネクト向けにチューニングされたアルゴリズムgtbcを使用します。
two_proc	Open MPIで実装されたアルゴリズムtwo_procを使用します。
neighbor	Open MPIで実装されたアルゴリズムneighborを使用します。
ring	Open MPIで実装されたアルゴリズムringを使用します。
recursive_doubling	Open MPIで実装されたアルゴリズムrecursive_doublingを使用します。
bruck	Open MPIで実装されたアルゴリズムbruckを使用します。
linear	Open MPIで実装されたアルゴリズムlinearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.2 coll_select_allgatherv_algorithm(MPI_ALLGATHERVルーチンのアルゴリズムを指定)

プログラム中のMPI_ALLGATHERVルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
3dtorus_sm	Tofuインターコネクト向けにチューニングされたアルゴリズム3dtorus_smを使用します。
3dtorus_fm, 3dtorus	Tofuインターコネクト向けにチューニングされたアルゴリズム3dtorus_fmを使用します。3dtorus_fm、3dtorusのどちらの文字列でも有効です。
gtvbc	Tofuインターコネクト向けにチューニングされたアルゴリズムgtvbcを使用します。
two_proc	Open MPIで実装されたアルゴリズムtwo_procを使用します。
neighbor	Open MPIで実装されたアルゴリズムneighborを使用します。
ring	Open MPIで実装されたアルゴリズムringを使用します。
bruck	Open MPIで実装されたアルゴリズムbruckを使用します。
default	Open MPIで実装されたアルゴリズムdefaultを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.3 coll_select_allreduce_algorithm(MPI_ALLREDUCEルーチンのアルゴリズムを指定)

プログラム中のMPI_ALLREDUCEルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
trinaryx3, trix3	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx3を使用します。trinaryx3、trix3のどちらの文字列でも有効です。
trinaryx6, trix6	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx6を使用します。trinaryx6、trix6のどちらの文字列でも有効です。
rdbc	Tofuインターコネクト向けにチューニングされたアルゴリズムrdbcを使用します。
rabenseifner	Open MPIで実装されたアルゴリズムrabenseifnerを使用します。
segmented_ring	Open MPIで実装されたアルゴリズムsegmented_ringを使用します。
ring	Open MPIで実装されたアルゴリズムringを使用します。
recursive_doubling	Open MPIで実装されたアルゴリズムrecursive_doublingを使用します。
nonoverlapping	Open MPIで実装されたアルゴリズムnonoverlappingを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.4 coll_select_alltoall_algorithm(MPI_ALLTOALLルーチンのアルゴリズムを指定)

プログラム中のMPI_ALLTOALLルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
crp	Tofuインターコネクト向けにチューニングされたアルゴリズムcrpを使用します。

MCAパラメーターの値	内容
blacc6d	Tofuインターコネクト向けにチューニングされたアルゴリズムblacc6dを使用します。
blacc3d	Tofuインターコネクト向けにチューニングされたアルゴリズムblacc3dを使用します。
doublespread	Tofuインターコネクト向けにチューニングされたアルゴリズムdoublespreadを使用します。
two_proc	Open MPIで実装されたアルゴリズムtwo_procを使用します。
linear_sync	Open MPIで実装されたアルゴリズムlinear_syncを使用します。
modified_bruck	Open MPIで実装されたアルゴリズムmodified_bruckを使用します。
pairwise	Open MPIで実装されたアルゴリズムpairwiseを使用します。
linear	Open MPIで実装されたアルゴリズムlinearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.5 coll_select_alltoallv_algorithm(MPI_ALLTOALLVルーチンのアルゴリズムを指定)

プログラム中のMPI_ALLTOALLVルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
doublespread	Tofuインターコネクト向けにチューニングされたアルゴリズムdoublespreadを使用します。
pairwise	Open MPIで実装されたアルゴリズムpairwiseを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.6 coll_select_barrier_algorithm(MPI_BARRIERルーチンのアルゴリズムを指定)

プログラム中のMPI_BARRIERルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
tree	Open MPIで実装されたアルゴリズムtreeを使用します。
two_proc	Open MPIで実装されたアルゴリズムtwo_procを使用します。
bruck	Open MPIで実装されたアルゴリズムbruckを使用します。
recursive_doubling	Open MPIで実装されたアルゴリズムrecursive_doublingを使用します。
linear	Open MPIで実装されたアルゴリズムlinearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.7 coll_select_bcast_algorithm(MPI_BCASTルーチンのアルゴリズムを指定)

プログラム中のMPI_BCASTルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
bintree6d, bin6d	Tofuインターコネクト向けにチューニングされたアルゴリズムbintree6dを使用します。bintree6dでもbin6dでも有効です。
trinaryx3, trix3	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx3を使用します。trinaryx3でもtrix3でも有効です。

MCAパラメーターの値	内容
bintree3d, bin3d	Tofuインターコネクト向けにチューニングされたアルゴリズムbintree3dを使用します。bintree3dでもbin3dでも有効です。
trinaryx6, trix6	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx6を使用します。trinaryx6でもtrix6でも有効です。
knomial	Open MPIで実装されたアルゴリズムknomialを使用します。
binomial	Open MPIで実装されたアルゴリズムbinomialを使用します。
binary_tree	Open MPIで実装されたアルゴリズムbinary_treeを使用します。
split_binary_tree	Open MPIで実装されたアルゴリズムsplit_binary_treeを使用します。
pipeline	Open MPIで実装されたアルゴリズムpipelineを使用します。
chain	Open MPIで実装されたアルゴリズムchainを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.8 coll_select_gather_algorithm(MPI_GATHERルーチンのアルゴリズムを指定)

プログラム中のMPI_GATHERルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
simple	Tofuインターコネクト向けにチューニングされたアルゴリズムsimpleを使用します。
linear_sync	Open MPIで実装されたアルゴリズムlinear_syncを使用します。
binomial	Open MPIで実装されたアルゴリズムbinomialを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.9 coll_select_gatherv_algorithm(MPI_GATHERVルーチンのアルゴリズムを指定)

プログラム中のMPI_GATHERVルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
simple	Tofuインターコネクト向けにチューニングされたアルゴリズムsimpleを使用します。
default	Open MPIで実装されたアルゴリズムdefaultを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.10 coll_select_reduce_algorithm(MPI_REDUCEルーチンのアルゴリズムを指定)

プログラム中のMPI_REDUCEルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
trinaryx3, trix3	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx3を使用します。trinaryx3でもtrix3でも有効です。

MCAパラメーターの値	内容
trinaryx6, trix6	Tofuインターコネクト向けにチューニングされたアルゴリズムtrinaryx6を使用します。trinaryx6でもtrix6でも有効です。
rabenseifner	Open MPIで実装されたアルゴリズムrabenseifnerを使用します。
in-order_binary	Open MPIで実装されたアルゴリズムin-order_binaryを使用します。
binomial	Open MPIで実装されたアルゴリズムbinomialを使用します。
binary	Open MPIで実装されたアルゴリズムbinaryを使用します。
pipeline	Open MPIで実装されたアルゴリズムpipelineを使用します。
chain	Open MPIで実装されたアルゴリズムchainを使用します。
linear	Open MPIで実装されたアルゴリズムlinearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.11 coll_select_reduce_scatter_algorithm(MPI_REDUCE_SCATTERルーチンのアルゴリズムを指定)

プログラム中のMPI_REDUCE_SCATTERルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
ring	Open MPIで実装されたアルゴリズムringを使用します。
recursive_halving	Open MPIで実装されたアルゴリズムrecursive_halvingを使用します。
non-overlapping	Open MPIで実装されたアルゴリズムnon-overlappingを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.12 coll_select_scan_algorithm(MPI_SCANルーチンのアルゴリズムを指定)

プログラム中のMPI_SCANルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
recursive_doubling	Open MPIで実装されたアルゴリズムrecursive_doublingを使用します。
linear	Open MPIで実装されたアルゴリズムlinearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.13 coll_select_scatter_algorithm(MPI_SCATTERルーチンのアルゴリズムを指定)

プログラム中のMPI_SCATTERルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
use_bcast	本システム向けに作成されたアルゴリズムuse_bcastを使用します。
binomial	Open MPIで実装されたアルゴリズムbinomialを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。

MCAパラメーターの値	内容
	MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.1.14 coll_select_scatterv_algorithm(MPI_SCATTERVルーチンのアルゴリズムを指定)

プログラム中のMPI_SCATTERVルーチンで実行するアルゴリズムを、常に特定のアルゴリズムに固定します。

MCAパラメーターの値	内容
linear_sync	本システム向けに作成されたアルゴリズムlinear_syncを使用します。
basic_linear	Open MPIで実装されたアルゴリズムbasic_linearを使用します。 MCAパラメーターによるアルゴリズム選択については、“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.2 アルゴリズム選択そのものに関するMCAパラメーター

8.4.2.1 coll_select_dectree_file(アルゴリズム選択の外部入力ユーザー定義ファイルの指定)

MCAパラメーターの値	内容
記載したファイルのパス	集団通信のアルゴリズム選択を記載したファイルを指定します。ファイルの記載方法および各アルゴリズムの詳細は“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をお読みください。 ユーザーが記載したアルゴリズム選択のルールに基づいてアルゴリズム選択を行います。ただし、ルールに誤りがある場合や、記載されていない集団通信については、本システムで選択したアルゴリズムを実施します。 本パラメーターの省略値はNULLです。 外部入力ファイルによるアルゴリズム選択については、“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.2.2 coll_select_show_decision_process(アルゴリズム選択の過程の出力)

MCAパラメーターの値	内容
2	集団通信のアルゴリズム選択の情報を出力します。標準エラー出力に出力されます。 それぞれのコミュニケータにおけるランク0が情報を出力します。
1	MPI_COMM_WORLDにおいて、ランク0が情報を出力します。
0	情報を出力しません。本パラメーターの省略値は0です。 詳細な情報については“8.3.2.1 アルゴリズム選択過程の表示”をお読みください。

備考：MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.3 アルゴリズム自体をチューニングするMCAパラメーター

8.4.3.1 coll_select_allreduce_algorithm_segmentsize(MPI_ALLREDUCEルーチンのセグメントサイズの指定)

MCAパラメーターの値	内容
1以上16776960以下の整数値	MPI_ALLREDUCEルーチンの特定のアルゴリズムで転送するときのセグメントサイズを、バイト数で指定します。セグメントサイズに16776960を超える値を指定した場合は、16776960が指定されたものとみなします。特定のアルゴリズムは、MCAパラメーター“8.4.1.3 coll_select_allreduce_algorithm(MPI_ALLREDUCEルーチンのアルゴリズムを指定)”で指定します。 指定可能なアルゴリズムは、以下になります。 segmented_ring、trinaryx6 (trix6)、trinaryx3 (trix3) MPI_ALLREDUCEルーチンの引数に指定したデータ型のサイズで、本パラメーターに指定した値が割り切れない場合、割り切れる値に変更します。本パラメーターを調整することで、特定のアルゴリズムの実行時間が変化します。パラメーターを指定しない場合と比べて性能が劣化する可能性もあります。
0	本パラメーターの省略値は0です。 メッセージサイズが、16776960B以下の場合は、セグメント分割を実施しません。メッセージサイズが、16776960Bを超える場合は、本パラメーターに16776960を指定した時の同じ動作になります。 詳細な情報については、“8.2 アルゴリズムのMCAパラメーターチューニング”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.3.2 coll_select_bcast_algorithm_segmentsize(MPI_BCASTルーチンのセグメントサイズの指定)

MCAパラメーターの値	内容
1以上16776960以下の整数値	MPI_BCASTルーチンの特定のアルゴリズムで転送するときのセグメントサイズを、バイト数で指定します。セグメントサイズに16776960を超える値を指定した場合は、16776960が指定されたものとみなします。特定のアルゴリズムは、MCAパラメーター“8.4.1.7 coll_select_bcast_algorithm(MPI_BCASTルーチンのアルゴリズムを指定)”で指定します。 指定可能なアルゴリズムは以下になります。 chain、pipeline、split_binary_tree、binary_tree、binomial、knomial、trinaryx6 (trix6)、bintree3d (bin3d)、trinaryx3 (trix3)、bintree6d (bin6d) MPI_BCASTルーチンの引数に指定したデータ型のサイズで、本パラメーターに指定した値が割り切れない場合、割り切れる値に変更します。 本パラメーターを調整することで、特定のアルゴリズムの実行時間が変化します。パラメーターを指定しない場合と比べて性能が劣化する可能性もあります。 MPI_BCASTルーチンの引数で、すべてのランクの要素数が同じであることが前提です。MCAパラメーターcoll_tuned_bcast_same_countに1を指定する必要があります。MCAパラメーターcoll_tuned_bcast_same_countについては、“表4.14 coll_tuned_bcast_same_count (ランク間で同じ要素数を用いたMPI_BCASTルーチン/MPI_IBCASTルーチンの通信を高速化)”をお読みください。
0	本パラメーターの省略値は0です。 メッセージサイズが、16776960B以下の場合は、セグメント分割を実施しません。 メッセージサイズが、16776960Bを超える場合は、以下のように動作します。 ・ 指定されたアルゴリズムが、chain、pipeline、split_binary_tree、binary_tree、binomial、knomialの場合は、セグメント分割を実施しません。 ・ 指定されたアルゴリズムが、trinaryx6 (trix6)、bintree3d (bin3d)、trinaryx3 (trix3)、bintree6d (bin6d)の場合は、本パラメーターに16776960を指定した時と同じ動作になります。 詳細な情報については、“8.2 アルゴリズムのMCAパラメーターチューニング”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.3.3 coll_select_reduce_algorithm_segmentsize(MPI_REDUCEルーチンのセグメントサイズの指定)

MCAパラメーターの値	内容
1以上16776960以下の整数値	MPI_REDUCEルーチンの特定のアルゴリズムで転送するときのセグメントサイズを、バイト数で指定します。セグメントサイズに16776960を超える値を指定した場合は、16776960が指定されたものとみなします。特定のアルゴリズムは、MCAパラメーター“8.4.1.10 coll_select_reduce_algorithm(MPI_REDUCEルーチンのアルゴリズムを指定)”で指定します。 指定可能なアルゴリズムは以下になります。 chain、pipeline、binary、binomial、in-order_binary、trinaryx6 (trix6)、trinaryx3 (trix3) MPI_REDUCEルーチンの引数に指定したデータ型のサイズで、本パラメーターに指定した値が割り切れない場合、割り切れる値に変更します。本パラメーターを調整することで、特定のアルゴリズムの実行時間が変化します。パラメーターを指定しない場合と比べて性能が劣化する可能性もあります。
0	本パラメーターの省略値は0です。 メッセージサイズが、16776960B以下の場合は、セグメント分割を実施しません。メッセージサイズが、16776960Bを超える場合は、本パラメーターに16776960を指定した時と同じ動作になります。 詳細な情報については、“8.2 アルゴリズムのMCAパラメーターチューニング”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.4.4 アルゴリズム選択の結果を取得するためのMCAパラメーター

8.4.4.1 coll_select_get_tuning_info(直前に実行した集団通信のアルゴリズム情報を取得)

MCAパラメーターの値	内容
1	MPI_Comm_get_infoを使用することで、直前に実行した集団通信のアルゴリズムが取得可能になります。 本パラメーターを使用した場合、通常に比べ実行時間が長くなります。
0	MPI_Comm_get_infoを使用しても、直前に実行した集団通信のアルゴリズムを取得できません。本パラメーターの省略値は0です。 詳細な情報については、“8.3.2.2 Infoオブジェクトを使用したアルゴリズムの選択結果の取得”をお読みください。

備考:MCAパラメーター名に使われている文字列collの"l"(エル)はどちらも英小文字です。

8.5 出力メッセージ

ここでは、アルゴリズム選択に関連する警告メッセージおよびアルゴリズム選択過程表示機能による出力メッセージについて説明します。

8.5.1 アルゴリズム選択に関する出力メッセージ(警告)

[mpi::coll-select::user-file-warn] Unable to open file. [*path*]

- メッセージの説明

  MCAパラメーターcoll_select_dectree_fileで指定されたファイルを参照できません。外部入力によるファイル指定を無視して、処理を継続します。

- パラメーターの説明

  path : MCAパラメーターcoll_select_dectree_fileで指定されたパス名

- 利用者の処置

  MCAパラメーターで指定した文字列が正しいかをご確認ください。正しい場合、出力されたエラーメッセージと合わせて担当保守員(SE)にご相談ください。正しくない場合、MCAパラメーターでファイルを正しく指定してください。

8.5.2 アルゴリズム選択過程表示機能による出力メッセージ(警告)

[mpi-coll::show-decision-process::warn] Algorithm selection does not work. [*reason*]

- メッセージの説明

  このコミュニケータにおいては、アルゴリズム選択の機能を使用できません。また、MPI統計情報の一部情報の取得もできません。

- パラメーターの説明

  reason : 理由を表す数字

reason	理由の内容
0	コミュニケータがグループ間コミュニケータです。
1	コミュニケータのサイズが1です。
2	コミュニケータがMPI_INTERCOMM_MERGEルーチンで作成されたものです。
3	その他の理由でアルゴリズム選択の機能を使用できません。

- 利用者の処置

  作成したコミュニケータが、上記の理由を満たすかをご確認ください。満たした場合は対処不要です。満たしていない場合は、エラーメッセージと合わせて担当保守員(SE)にご相談ください。

[mpi-coll::show-decision-process::warn] The algorithm is judged as not applicable. [*collname*] [*algnum*] [*reason*] [*new_algnum*]

- メッセージの説明

  指定されたアルゴリズムは適用条件を満たしていません。異なるアルゴリズムを選択します。

- パラメーターの説明

  collname : 集団通信の種別

  algnum : アルゴリズムを表す番号

  reason : 理由を表す数字

  new_algnum : *algnum*が1から6の場合、または300の場合に選択される、代替のアルゴリズム番号

reason	理由の内容
0	コミュニケータのサイズが条件を満たしていません。
1	データ型を除く、アルゴリズムの適用条件を満たしていません。
2	データ型の条件を満たしていません。
3	データ配置に隙間が存在するため、gtvbcアルゴリズムを使用できません。
4	MCAパラメーターcoll_tuned_bcast_same_countが無効になっているため、データ分割を行えません。セグメントサイズを0にします。
5	その他の理由で選択できません。

- 利用者の処置

  意図したアルゴリズムがこのメッセージにより選択されない場合は、“8.3.4 アルゴリズムと適用に必要な条件一覧”に記載してある条件をお読みください。

[mpi::coll-select::show-decision-process::warn] Algorithm selection by the user file does not work. [*reason*]

- メッセージの説明

  外部入力ファイルによるアルゴリズム選択に問題が発生しました。システムのアルゴリズム選択に変更します。

- パラメーターの説明

  reason : 理由を表す数字

reason	理由の内容
0	外部入力ファイルが参照できません。
1	外部入力ファイルに記載している文字列または文法が誤っています。
2	外部入力ファイルに記載可能な入力数を超えています。
3	外部入力ファイルに記載可能な条件文の数を超えています。
4	外部入力ファイルに記載している割り算でゼロ除算が存在します。
5	外部入力ファイルに記載している変数が誤っています。
6	アルゴリズム名に誤りがあります。
7	実行可能なアルゴリズムが存在しません。
8	外部入力ファイルの解析中にMPIライブラリ内部でエラーが発生しています。

- 利用者の処置

  *reason*が8以外の場合には、*reason*に従って、各種設定が正しいかどうかをご確認ください。*reason*が8の場合には、外部入力ファイルと出力メッセージを合わせて、担当保守員(SE)にご連絡ください。

[mpi::coll-select::show-decision-process::warn] Algorithm selection by the Info object does not work. [*method*]

- メッセージの説明

  Infoオブジェクトによるアルゴリズム選択に問題が発生しました。次の優先度のアルゴリズム選択に変更します。

- パラメーターの説明

  method : アルゴリズムの選択方法

  - info-oneshot

    Infoオブジェクトによるアルゴリズムを1回だけ指定する選択方法。"8.3.1.4.2 集団通信ルーチン呼出しごとの指定"をご確認ください。

  - info-rules

    Infoオブジェクトによるアルゴリズムをコミュニケータ単位で指定する選択方法。"8.3.1.4.3 コミュニケータごとの指定"をご確認ください。

- 利用者の処置

  Infoオブジェクトで指定した*method*に対応するvalueが正しいかをご確認ください。Infoオブジェクトに指定したvalueが正しい場合は、該当部分の処理と警告文を合わせて担当保守員(SE)にご相談ください。

[mpi::coll-select::show-decision-process::warn] Tofu Barrier Communication can not be applicable. [*method*]

- メッセージの説明

  バリア通信は指定できません。次の優先度のアルゴリズム選択に変更します。

- パラメーターの説明

  method : アルゴリズムの選択方法

- mca-param

  MCAパラメーターによるアルゴリズム選択。“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をご確認ください。

- info-oneshot

  Infoオブジェクトによるアルゴリズムを1回だけ指定する選択方法。“8.3.1.4.2 集団通信ルーチン呼出しごとの指定”をご確認ください。

- info-rules

  Infoオブジェクトによるアルゴリズムをコミュニケータ単位で指定する選択方法。“8.3.1.4.3 コミュニケータごとの指定”をご確認ください。

- user-file

  外部入力ファイルによるアルゴリズム選択。“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をご確認ください。

・ 利用者の処置

*method*にバリア通信を指定しないでください。バリア通信はMPIライブラリによる自動選択でのみ使用可能です。

[mpi::coll-select::show-decision-process::warn]. An algorithm different from the selected one was used. [*collname*] [*algnum*] [*reason*] [*new_algnum*]

・ メッセージの説明

3dtorus_smアルゴリズムを選択しようとしましたが、アルゴリズムで使う通信資源の競合により選択できませんでした。そのため、異なるアルゴリズムを選択しました。

・ パラメーターの説明

collname : 集団通信の種別

algnum : アルゴリズムを表す番号

reason : 理由を表す数字

new_algnum : 実際に選択されたアルゴリズムを表す番号

reason	理由の内容
0	通信資源の競合により選択できませんでした。

・ 利用者の処置

意図したアルゴリズムがこのメッセージにより選択されない場合は、“8.3.4 アルゴリズムと適用に必要な条件一覧”に記載してある条件をお読みください。

8.5.3 アルゴリズム選択過程表示機能による出力メッセージ(情報)

[mpi::coll-select::show-decision-process::info] Any algorithm is not selected. [*method*] [*collname*]

・ メッセージの説明

*method*によるアルゴリズム選択を行いませんでした。

・ パラメーターの説明

method : アルゴリズムの選択方法

- mca-param

  MCAパラメーターによるアルゴリズム選択。“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をご確認ください。

- info-oneshot

  Infoオブジェクトによるアルゴリズムを1回だけ指定する選択方法。“8.3.1.4.2 集団通信ルーチン呼出しごとの指定”をご確認ください。

— info-rules

Infoオブジェクトによるアルゴリズムをコミュニケータ単位で指定する選択方法。“8.3.1.4.3 コミュニケータごとの指定”をご確認ください。

— user-file

外部入力ファイルによるアルゴリズム選択。“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をご確認ください。

collname : 集団通信の種別

- 利用者の処置

対処不要です。

[mpi::coll-select::show-decision-process::info] An algorithm is selected. [*method*] [*collname*] [*algnum*] [*reason*]

- メッセージの説明

*method*によるアルゴリズム選択を行いました。これから適用に必要な条件の判定を行います。

- パラメーターの説明

method : アルゴリズムの選択方法

— special-route

特別な場合によるアルゴリズム選択。“8.3.1.2 特別な場合のアルゴリズム選択”をご確認ください。

— mca-param

MCAパラメーターによるアルゴリズム選択。“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をご確認ください。

— info-oneshot

Infoオブジェクトによるアルゴリズムを1回だけ指定する選択方法。“8.3.1.4.2 集団通信ルーチン呼出しごとの指定”をご確認ください。

— info-rules

Infoオブジェクトによるアルゴリズムをコミュニケータ単位で指定する選択方法。“8.3.1.4.3 コミュニケータごとの指定”をご確認ください。

— user-file

外部入力ファイルによるアルゴリズム選択。“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をご確認ください。

— system-file

本システムによる自動的なアルゴリズム選択。

collname : 集団通信の種別

algnum : アルゴリズムを表す番号

reason : 選択法がspecial-routeの場合に選択された理由

reason	Specialの理由の内容
0	バリア通信を優先的に選択します。
1	非可換演算や順序保証モードに対応するアルゴリズムを選択します。
2	MPI_IN_PLACEのためのアルゴリズムを選択します。

- 利用者の処置

対処不要です。

[mpi::coll-select::show-decision-process::info] The algorithm is judged as applicable. [*method*] [*collname*] [*algnum*]

- メッセージの説明

  *method*によるアルゴリズム選択を行いました。適用に必要な条件を満たしていたので、このアルゴリズムを実行します。

- パラメーターの説明

  method : アルゴリズムの選択方法

  - mca-param

    MCAパラメーターによるアルゴリズム選択。“8.3.1.3 MCAパラメーターによるアルゴリズム選択”をご確認ください。

  - info-oneshot

    Infoオブジェクトによるアルゴリズムを1回だけ指定する選択方法。“8.3.1.4.2 集団通信ルーチン呼出しごとの指定”をご確認ください。

  - info-rules

    Infoオブジェクトによるアルゴリズムをコミュニケータ単位で指定する選択方法。“8.3.1.4.3 コミュニケータごとの指定”をご確認ください。

  - user-file

    外部入力ファイルによるアルゴリズム選択。“8.3.1.5 外部入力ファイルによるアルゴリズム選択”をご確認ください。

  - system-file

    本システムによる自動的なアルゴリズム選択。

  collname : 集団通信の種別

  algnum : アルゴリズムを表す番号

- 利用者の処置

  対処不要です。

[mpi::coll-select::show-decision-process::info] Rules are not set for this collective communication. [*collname*]

- メッセージの説明

  ユーザーが指定した外部入力ファイルに、この集団通信のルールは記載されていません。

- パラメーターの説明

  collname : 集団通信の種別

- 利用者の処置

  表示された集団通信の種別を外部入力ファイルに記載していないことを確認してください。外部入力ファイルに記載してある場合は、外部入力ファイルとエラーメッセージと合わせて担当保守員(SE)にご相談ください。

付録A　エラークラス一覧

下表に、本処理系で検出するエラークラスを示します。これらのエラークラスはMPI規格で規定されています。詳細は、MPI規格をご覧ください。

表A.1 エラークラス一覧

エラークラス	意味	設定値
MPI_SUCCESS	エラーなし	0
MPI_ERR_BUFFER	無効なバッファポインタ	1
MPI_ERR_COUNT	無効なcount引数	2
MPI_ERR_TYPE	無効なデータ型引数	3
MPI_ERR_TAG	無効なタグ引数	4
MPI_ERR_COMM	無効なコミュニケータ	5
MPI_ERR_RANK	無効なランク	6
MPI_ERR_REQUEST	無効な要求(ハンドル)	7
MPI_ERR_ROOT	無効なルート	8
MPI_ERR_GROUP	無効なグループ	9
MPI_ERR_OP	無効な操作	10
MPI_ERR_TOPOLOGY	無効なトポロジー	11
MPI_ERR_DIMS	無効な次元引数	12
MPI_ERR_ARG	無効な引数	13
MPI_ERR_UNKNOWN	原因不明なエラー	14
MPI_ERR_TRUNCATE	受信でメッセージが切り捨てられた	15
MPI_ERR_OTHER	このリストにないエラー	16
MPI_ERR_INTERN	MPI内部エラー	17
MPI_ERR_IN_STATUS	ステータス中のエラーコード	18
MPI_ERR_PENDING	保留要求	19
MPI_ERR_ACCESS	パーミッションの誤り	20
MPI_ERR_AMODE	無効なamode	21
MPI_ERR_ASSERT	無効なassert引数	22
MPI_ERR_BAD_FILE	無効なファイル名	23
MPI_ERR_BASE	無効なベース引数	24
MPI_ERR_CONVERSION	ユーザー定義のデータ変換中のエラー	25
MPI_ERR_DISP	無効な変位引数	26
MPI_ERR_DUP_DATAREP	datarepがすでに定義されている	27
MPI_ERR_FILE_EXISTS	ファイルがすでに存在している	28
MPI_ERR_FILE_IN_USE	複数プロセスでオープン中のファイル操作が未完了	29
MPI_ERR_FILE	無効なファイルハンドル	30
MPI_ERR_INFO_KEY	無効なInfoキー	31
MPI_ERR_INFO_NOKEY	定義されていないキー	32
MPI_ERR_INFO_VALUE	無効なInfo値	33
MPI_ERR_INFO	無効なInfo引数	34

エラークラス	意味	設定値
MPI_ERR_IO	その他のI/Oエラー	35
MPI_ERR_KEYVAL	無効なキー値	36
MPI_ERR_LOCKTYPE	無効なlocktype引数	37
MPI_ERR_NAME	確立した名前でない	38
MPI_ERR_NO_MEM	有効なメモリがない	39
MPI_ERR_NOT_SAME	collective引数が異なる	40
MPI_ERR_NO_SPACE	十分な容量がない	41
MPI_ERR_NO_SUCH_FILE	ファイルが存在しない	42
MPI_ERR_PORT	ポートが存在しない	43
MPI_ERR_QUOTA	quotaを超えた	44
MPI_ERR_READ_ONLY	読み出し専用のファイルまたはファイルシステム	45
MPI_ERR_RMA_CONFLICT	ウィンドウアクセスの衝突	46
MPI_ERR_RMA_SYNC	RMA呼出しの誤った同期	47
MPI_ERR_SERVICE	サービスはすでに確立している	48
MPI_ERR_SIZE	無効なサイズ引数	49
MPI_ERR_SPAWN	子プロセスが生成できない	50
MPI_ERR_UNSUPPORTED_DATAREP	未サポートのdatarep	51
MPI_ERR_UNSUPPORTED_OPERATION	未サポートのオペレーション	52
MPI_ERR_WIN	無効なウィンドウ引数	53
MPI_T_ERR_MEMORY	メモリ不足	54
MPI_T_ERR_NOT_INITIALIZED	ツールインターフェースが初期化されていない	55
MPI_T_ERR_CANNOT_INIT	ツールインターフェースが初期化できない	56
MPI_T_ERR_INVALID_INDEX	無効なインデックス番号	57
MPI_T_ERR_INVALID_ITEM	無効なアイテムインデックス	58
MPI_T_ERR_INVALID_HANDLE	無効なハンドル	59
MPI_T_ERR_OUT_OF_HANDLES	これ以上ハンドルが利用できない	60
MPI_T_ERR_OUT_OF_SESSIONS	これ以上セッションが利用できない	61
MPI_T_ERR_INVALID_SESSION	無効なセッション	62
MPI_T_ERR_CVAR_SET_NOT_NOW	今現在変数に設定できない	63
MPI_T_ERR_CVAR_SET_NEVER	実行終了まで変数に設定できない	64
MPI_T_ERR_PVAR_NO_STARTSTOP	変数が開始または終了できない	65
MPI_T_ERR_PVAR_NO_WRITE	変数が書込みまたはリセットできない	66
MPI_T_ERR_PVAR_NO_ATOMIC	変数がアトミックに書込みまたは読込みできない	67
MPI_ERR_RMA_RANGE	ターゲットメモリがウィンドウに属していない	68
MPI_ERR_RMA_ATTACH	メモリをアタッチできない	69
MPI_ERR_RMA_FLAVOR	渡されたウィンドウが誤ったフレーバーである	70
MPI_ERR_RMA_SHARED	メモリを共有できない	71
MPI_T_ERR_INVALID	ツールインターフェースの無効な使用またはパラメーターの値が正しくない	72
MPI_T_ERR_INVALID_NAME	無効な変数名またはカテゴリ名	73

エラークラス	意味	設定値
MPI_ERR_LASTCODE	最後のエラーコード	92

用語集

型仕様(type signature)

MPIルーチンによって送受信されるメッセージは、必ず基本データ型のデータの並びに分解できます。型仕様は、この「基本データ型の並び」です。型仕様(type signature)は、MPI規格で定義されている用語です。

グループ間コミュニケータ(inter-communicator)

2つの異なるコミュニケータを接続したコミュニケータを指します。MPIの規格書ではinter-communicatorとして定義されている用語です。例えば、MPI_COMM_SPAWNルーチンにより生成したコミュニケータと、MPI_COMM_SPAWNルーチンを呼び出したコミュニケータ間で通信をする際に用いられます。

グループ内コミュニケータ(intra-communicator)

MPI_COMM_WORLDから生成されたコミュニケータや、MPI_INTERCOMM_CREATE以外のコミュニケータ生成ルーチンで生成されたコミュニケータを指します。MPIの規格書ではintra-communicatorとして定義されている用語です。

コミュニケータに割り当てられた計算ノード

あるコミュニケータにおいて、そのコミュニケータに属する並列プロセスが1つでも存在する計算ノードを指します。

最大転送サイズ

メッセージの転送は、MPIライブラリ内でパケットと呼ばれる単位で送受信が行われます。1つのパケットには上限値があり、この上限値を表します。

最大転送サイズよりも大きいメッセージを転送する場合、各パケットのサイズが最大転送サイズ以下となるように、複数のパケットに分割されて転送されます。

セグメントサイズ

パイプライン転送時の1回あたりのデータ転送量です。MCAパラメーターでは、集団通信のルーチンごとに、セグメントサイズを指定することが可能です。この値を変更することにより、パイプライン転送の転送回数と1回あたりの転送時間が変化します。

ノンブロッキング通信

MPIルーチンの実際の手続が完了する前に、MPIルーチンから復帰する可能性のあるメッセージ送受信を表します。MPIルーチンの呼出し時に指定した利用者バッファは、操作の完了確認を行う前に再利用できません。

パイプライン転送

一度にすべてのメッセージを転送するのではなく、一定サイズのセグメントにメッセージを分割して転送することです。集団通信において、パイプライン転送を実装したアルゴリズムがあります。

バリアゲート

バリア通信を行うためのハードウェア・リソースです。バリアゲートには、始点と終点の役目を果たす入力・出力ゲートおよび中継点の役目を果たす中継ゲートの2種類のゲートがあります。バリアゲートはネットワークインターフェース装置(TNI)ごとに入力・出力ゲート、中継ゲート合わせて48個存在します。MPIライブラリで使用可能な最大のゲート数は、計算ノードごとに入力・出力ゲートが96であり、中継ゲートが192です。これらMPIで使用できるゲート数は、本処理系の製品の版数が変更されるときに、変更される可能性があります。

バリア通信

Tofuインターコネクト上で、48バイト以下のデータのリダクション機能を備え、プロセス間でバリア同期を実現するハードウェアの通信機構です。

ブロッキング通信

MPIルーチンから復帰すれば、MPIルーチンの呼出し時に指定した利用者バッファが再利用できるメッセージ送受信を表します。

並列プロセス

mpiexecコマンドにより計算ノードに起動されたプロセスを並列プロセスと呼びます。各並列プロセスには、0から始まる番号が割り当てられます。本処理系では、これら番号は、MPIプログラムのコミュニケータMPI_COMM_WORLDのランクに対応します。

メッセージのサイズ

バイト数で表したメッセージの大きさです。本書では、“メッセージの長さ”と区別し、この用語を使用しています。

メッセージの長さ

メッセージのデータの要素数を表します。MPI規格で定義されているメッセージの長さ(message length)に準じています。

MPMD

Multiple Program/Multiple Dataの略語です。2つ以上の異なるMPIプログラムを使用して、処理を分担しながら動作する並列プログラミングモデルの1つです。

SPMD

Single Program/Multiple Dataの略語です。各プロセスで同一のMPIプログラムを使用して、処理を分担しながら動作する並列プログラミングモデルの1つです。

Unexpected message

MPI_SENDルーチンなど送信系のルーチンに対応するMPI_RECVルーチンなど受信系のルーチンの呼出しが遅れることに対応して、受信側のプロセスにおいて一時的なバッファに退避しておく必要のあるメッセージです。