Cartesian Partition Manager Library

Ver. 1.0.3

User's Manual

Center of Research on Innovative Simulation Software
Institute of Industrial Science
the University of Tokyo

http://www.iis.u-tokyo.ac.jp/

July 2012

目次

1 この文書について

この文書は，構造格子空間の領域分割情報管理クラスライブラリ（以下，CPM ライブラリ）のユーザーマニュアルです.

1.1 CPM ライブラリについて

本ライブラリは，構造格子空間の領域分割処理，領域情報の管理及び領域間通信機能を提供する，C++ で記述されたクラスライブラリです.

ユーザーは，C++/Fortran90 で本ライブラリを利用可能です.

1.2 書式について

次の書式で表されるものは，Shell のコマンドです.

$ コマンド（コマンド引数）

または,

 # コマンド（コマンド引数）

“$” で始まるコマンドは一般ユーザーで実行するコマンドを表し，“#” で始まるコマンドは管理者（主に root）で実行するコマンドを表しています.

1.3 動作環境

CPM ライブラリは，以下の環境について動作を確認しています.

- Linux/Intel コンパイラ
  - CentOS6.2 i386/x86_64
  - Intel C++/Fortran Compiler Version 12 (icpc/ifort)
- MacOS X Snow Leopard
  - MacOS X Snow Leopard
  - Intel C++/Fortran Compiler Version 11 (icpc/ifort)
- 京コンピュータ

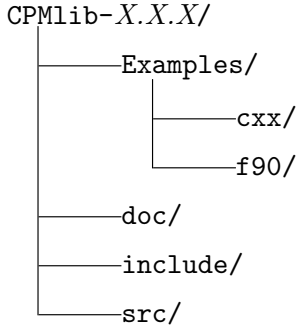
2 パッケージのビルド

2.1 パッケージの構造

CPM ライブラリのパッケージは次のようなファイル名で保存されています.
(*X.X.X* にはバージョンが入ります)

`CPMlib-`*X.X.X*`.tar.gz`

このファイルの内部には，次のようなディレクトリ構造が格納されています.



これらのディレクトリ構造は，次の様になっています.

- `Examples`
  CPM ライブラリの利用例（C++/Fortran90）のサンプルソースコードが収められています.
  - `cxx`
    C++ ユーザープログラムから CPM ライブラリを利用するサンプルソースコードが収められています.
  - `f90`
    Fortran90 ユーザープログラムから CPM ライブラリを利用するサンプルソースコードが収められています.

- `doc`
  この文書を含む TextParser ライブラリの文書が収められています.

- `include`
  ヘッダファイルが収められています．ここに収められたファイルは `make install` で

$prefix/include にインストールされます.

- src
  ソースが格納されたディレクトリです. ここにライブラリ libcpmlib.a が作成され, make install で$prefix/lib にインストールされます.

2.2 パッケージのビルド

いずれの環境でも shell で作業するものとします．以下の例では bash を用いていますが，shell によって環境変数の設定方法が異なるだけで，インストールの他のコマンドは同一です．適宜，環境変数の設定箇所をお使いの環境でのものに読み替えてください．

以下の例では，作業ディレクトリを作成し，その作業ディレクトリに展開したパッケージを用いてビルド，インストールする例を示しています．

1. 作業ディレクトリの構築とパッケージのコピー
   まず，作業用のディレクトリを用意し，パッケージをコピーします．ここでは，カレントディレクトリに work というディレクトリを作り，そのディレクトリにパッケージをコピーします．

   ```
   $ mkdir work
   $ cp [パッケージのパス] work
   ```

2. 作業ディレクトリへの移動とパッケージの解凍
   先ほど作成した作業ディレクトリに移動し，パッケージを解凍します．

   ```
   $ cd work
   $ tar CPMlib-X.X.X.tar.gz
   ```

3. CPMlib-X.X.X ディレクトリに移動先ほどの解凍で作成された CPMlib-X.X.X ディレクトリに移動します．

   ```
   $ cd CPMlib-X.X.X
   ```

4. configure スクリプトを実行
   次のコマンドで configure スクリプトを実行します．

   ```
   $ ./configure [option]
   ```

   configure スクリプトの実行時には，お使いの環境に合わせたオプションを指定する必要があります．configure オプションに関しては，2.3 章を参照してください．configure スクリプトを実行することで，環境に合わせた Makefile が作成されます．

5. make の実行
   make コマンドを実行し，ライブラリをビルドします．

   ```
   $ make
   ```

make コマンドを実行すると，次のファイルが作成されます.

```
src/libcpmlib.a
```

ビルドをやり直す場合は，`make clean` を実行して，前回の make 実行時に作成されたファイルを削除します.

```
$ make clean
$ make
```

また，configure スクリプトによる設定，Makefile の生成をやり直すには，`make distclean` を実行して，全ての情報を削除してから，configure スクリプトの実行からやり直します.

```
$ make distclean
$ ./configure [option]
$ make
```

6. インストール
次のコマンドで configure スクリプトの`--prefix` オプションで指定されたディレクトリに，ライブラリ，ヘッダファイルをインストールします.

```
$ make install
```

インストールされる場所とファイルは以下の通りです.

```
${prefix}
├──bin/
│   └──cpm-config
├──doc/
│   ├──manual.pdf
│   └──reference.pdf
├──include/
│   ├──cpm_Base.h
│   ├──cpm_Define.h
│   ├──cpm_DomainInfo.h
│   ├──cpm_ObjList.h
│   ├──cpm_ParaManager.h
│   ├──cpm_TextParser.h
│   ├──cpm_TextParserDomain.h
│   ├──cpm_Version.h
│   ├──cpm_VoxelInfo.h
│   ├──cpm_fparam.fi
│   └──inline/
│       ├──cpm_ParaManager_BndComm.h
│       ├──cpm_ParaManager_BndCommEx.h
│       └──cpm_ParaManager_inline.h
└──lib/
    └──libcpmlib.a
```

ただし，インストール先のディレクトリへの書き込みに管理者権限が必要な場合は，sudo コマンドを用いるか，管理者でログインして make install を実行します.

```
$ sudo make install
```

または,

```
$ su
password:
# make install
# exit
```

7. アンインストール
   アンインストールするには，書き込み権限によって，

```
$ make uninstall
```

   または，

```
$ sudo make uninstall
```

   または，

```
$ su
passward:
# make uninstall
# exit
```

   を実行します．

2.3 configure スクリプトのオプション

- `--prefix=`*dir*
  prefix は，パッケージをどこにインストールするかを指定します．prefix で設定した場所が `--prefix=/usr/local/CPMlib` の時，

  ライブラリ：`/usr/local/CPMlib/lib`
  ヘッダファイル：`/usr/local/CPMlib/include`

  にインストールされます．
  prefix オプションが省略された場合は，デフォルト値として/usr/local/CPMlib が採用され，インストールされます．

- コンパイラ等のオプション
  コンパイラ，リンカやそれらのオプションは，configure スクリプトで半自動的に探索します．ただし，標準ではないコマンドやオプション，ライブラリ，ヘッダファイルの場所は探索出来ないことがあります．また，標準でインストールされたものでないコマンドやライブラリを指定して利用したい場合があります．そのような場合，これらの指定を configure スクリプトのオプションとして指定することができます．

  `CXX`
  C++ コンパイラのコマンドパスです．

  `CXXFLAGS`
  C++ コンパイラへ渡すコンパイルオプションです．

  `LDFLAGS`
  リンク時にリンカに渡すリンク時オプションです．例えば，使用するライブラリが標準でないの場所 <`libdir`> にある場合，`-L`<`libdir`> としてその場所を指定します．

  `LIBS`
  利用したいライブラリをリンカに渡すリンク時オプションです．例えば，ライブラリ <`library`> を利用する場合，`-l`<`library`> として指定します．

  `FC`
  Fortran90 コンパイラのコマンドパスです．

`FCFLAGS`
Fortran90 コンパイラに渡すコンパイルオプションです.

なお，CPM ライブラリは C++ で記述されているため，C++ 以外のコンパイラの指定は必須ではありませんが，Fortran90 用のサンプルコードを make する場合は，Fortran90 コンパイラとコンパイルオプションを指定する必要があります.

・ライブラリ指定のオプション
CPM ライブラリを利用する場合，コンパイル，リンク時に，MPI ライブラリと TextParser ライブラリが必ず必要になります．これらのライブラリのインストールパスは，次に示す configure オプションで指定する必要があります.

`--with-mpich`=*dir*
MPI ライブラリとして mpich を使用する場合に，mpich のインストール先を指定します.

`--with-ompi`=*dir*
MPI ライブラリとして OpenMPI を使用する場合に，OpenMPI のインストール先を指定します．`--with-mpich` オプションと同時に指定された場合，`--with-mpich` が有効になります.

`--with-parser`=dir
TextParser ライブラリのインストール先を指定します.

なお、`mpic++` 等の mpi ライブラリに付属のコンパイララッパーを使用する場合は，mpi に関する設定がラッパー内で自動的に設定されるため，`--with-mpich` や`--with-ompi` の指定は必要ありません.

・その他のオプション
必要に応じて，次に示すオプションを指定できます.

`--with-real`=(float|double)
CPM ライブラリ内で定義されている実数型 REAL_TYPE の型を指定します.
`--with-real=double` を指定した場合，REAL_TYPE は double 型として扱われます．`--with-real=float` を指定したか，`--with-real` オプションの指定が無い場合，REAL_TYPE は float 型として扱われます.

`--with-comp`=(INTEL|FJ)
使用するコンパイラのベンダーを指定します.
Intel コンパイラの場合 INTEL を，富士通コンパイラのとき FJ を指定します．該当しない場合は指定する必要はありません.
本オプションを指定した場合, `--with-real` オプションの指定内容に従い, cpm-config コマンド (2.5 章を参照) の`--f90flags` で取得できる Fortran コンパイルオプションに，デフォルトの実数型オプションが自動的に付加されます.

`--with-f90example`=(yes|no)
CPM ライブラリ内の make 時に一緒に一緒に make される Fortran90 用のサンプルコードについて，make するかどうかを指定します．Fortran90 コンパイラが存在しない場合や，Fortran90 用のサンプルが必要無い場合は「no」を指定してください．（デフォルトは yes）

`--host`=*hosttype*
クロスコンパイル環境の場合に指定します.

なお，configure オプションの詳細は，`./configure --help` コマンドで表示されますが，CPM ライブラリでは，上記で説明したオプション以外は無効となります.

2.4 configure 実行時オプションの例

・Linux / MacOS X の場合

CPM ライブラリの prefix：/opt/CPMlib
MPI ライブラリ：OpenMPI，/usr/local/openmpi
TextParser ライブラリ：/usr/local/textparser
REAL_TYPE：double
C++ コンパイラ：icpc
F90 コンパイラ：ifort

の環境の場合，次のように configure コマンドを実行します．

```
$ ./configure --prefix=/opt/CPMlib \
              --with-ompi=/usr/local/openmpi \
              --with-parser=/usr/local/textparser \
              --with-real=double \
              --with-comp=INTEL \
              CXX=icpc \
              FC=ifort
```

・京コンピュータの場合

CPM ライブラリの prefix：/home/userXXXX/CPMlib
TextParser ライブラリ：/home/userXXXX/textparser
REAL_TYPE：double
C++ コンパイラ：mpiFCCpx
F90 コンパイラ：mpifrtpx

の環境の場合，次のように configure コマンドを実行します．

```
$ ./configure --host=sparc64-unknown-linux-gnu \
              --prefix=/home/userXXXX/CPMlib \
              --with-parser=/home/userXXXX/textparser \
              --with-real=double \
              --with-comp=FJ \
              CXX=mpiFCCpx \
              FC=mpifrtpx
```

2.5 cpm-config コマンド

CPM ライブラリをインストールすると，`$prefix`/bin/cpm-config コマンド（シェルスクリプト）が生成されます.

このコマンドを利用することで，ユーザーが作成したプログラムをコンパイル，リンクする際に，CPM ライブラリを参照するために必要なコンパイルオプション，リンク時オプションを取得することができます.

`cpm-config` コマンドは，次に示すオプションを指定して実行します.

`--cxx`
CPM ライブラリの構築時に使用した C++ コンパイラを取得します.

`--fc`
CPM ライブラリの構築時に使用した Fortran90 コンパイラを取得します.

`--fclink`
main が Fortran90 のユーザープログラムを CPM ライブラリとリンクする場合に使用するリンカコマンドを取得します.

`--cflags`
C++ コンパイラオプションを取得します.

`--fcflags`
Fortran90 コンパイラオプションを取得します.

`--libs`
CPM ライブラリのリンクに必要なリンク時オプションを取得します.

`--fclibs`
main が Fortran90 のユーザープログラムを CPM ライブラリとリンクする場合に必要なリンク時オプションを取得します.

ただし，`cpm-config` コマンドで取得できるオプションは，CPM ライブラリを利用する上で最低限必要なオプションのみとなります.

最適化オプション等は必要に応じて指定してください.

また，具体的な `cpm-config` コマンドの使用方法は，3 章を参照してください.

2.6 提供環境の作成

提供環境の作成を行うには，configure スクリプト実行後に，以下のコマンドを実行します．

```
$ ./make dist
```

上記コマンドを実行すると，提供環境が

```
CPMlib-X.X.X.tar.gz
```

という圧縮ファイルに保存されます．($X.X.X$ にはバージョンが入ります)

3 CPM ライブラリの利用法 (ビルド)

CPM ライブラリは，C++ 及び Fortran90 プログラム内で利用できます．以下に，ユーザーが作成する CPM ライブラリを利用するプログラムのビルド方法を示します．

以下の例では，configure スクリプトで"`--prefix=/usr/local/CPMlib`" を指定して CPM ライブラリをビルド，インストールしているものとして示します．

3.1 C++

CPM ライブラリを利用している C++ のプログラム main.C を icpc でコンパイルする場合は，次のようにコンパイル，リンクします．

```
$ icpc -o prog main.C ‘/usr/local/CPMlib/bin/cpm-config --cflags‘ \
  ‘/usr/local/CPMlib/bin/cpm-config --libs‘
```

3.2 Fortran 90

CPM ライブラリを利用している Fortran90 のプログラム main.f90 を ifort でコンパイルする場合は，次のようにコンパイル，リンクします．

```
$ ifort -o prog main.f90 ‘/usr/local/CPMlib/bin/cpm-config --fcflags‘ \
  ‘/usr/local/CPMlib/bin/cpm-config --fclibs‘
```

ただし，プラットフォームによっては，C++/Fortran90 が混在したコードをリンクする場合，C++ コンパイラをリンカとして指定する必要がある場合があります．その場合は，以下のようにコンパイル，リンクします．

```
$ mpifrtpx -c main.f90 ‘/usr/local/CPMlib/bin/cpm-config --fcflags‘
$ mpiFCCpx -o prog main.o ‘/usr/local/CPMlib/bin/cpm-config --fclibs‘
```

4 C++ ユーザープログラムでの利用方法

以下に，CPM ライブラリの C++ API の説明を示します．

4.1 サンプルプログラム

Examples/c++ ディレクトリに，C++ ユーザープログラムでの CPM ライブラリ使用例のサンプルソースコードが収められています．

- `main.C`
  領域分割情報ファイルの読み込み，領域分割実行，各種通信処理のテストを行うサンプルです．
  このサンプルコードは make 時に一緒にコンパイル，リンクされ，`exampleCXX` という実行ファイルが作成されます．`data` ディレクトリに実行サンプルが収められています．

4.2 cpm_ParaManager.h のインクルード

CPM ライブラリの C++ API 関数群は，CPM ライブラリが提供するヘッダファイル cpm_ParaManager.h で定義されています．CPM ライブラリの API 関数を使う場合は，このヘッダーファイルをインクルードします．

cpm_ParaManager.h には，ユーザーが利用可能な本ライブラリの API がまとめられている cpm_ParaManager クラスのインターフェイスが記述されています．ユーザープログラムから本ライブラリを使用する場合，このクラスのメソッドを用います．

cpm_ParaManager.h は，configure スクリプト実行時の設定 `prefix` 配下の`${prefix}`/include に `make install` 時にインストールされます．

4.3 cpm_TextParserDomain.h のインクルード

CPM ライブラリは，領域分割情報の生成，管理，各種通信機能等の他に，TextParser ライブラリフォーマットの領域分割情報ファイルの読み込み機能を提供しています．読み込んだ領域分割情報を CPM ライブラリの領域分割処理にそのまま渡すことで，領域分割処理を行うことができます．

領域分割情報ファイルの読み込み機能を利用するためには，cpm_TextParserDomain.h をインクルードします．cpm_TextParserDomain.h には，領域分割情報ファイルの読み込み処理が記述されています．

4.4 マクロ，列挙型，エラーコード

CPM ライブラリ内で使用されるマクロ，列挙型，エラーコードについては，cpm_Define.h に定義されています.

・REAL_TYPE マクロ
REAL_TYPE マクロは，cpm_Define.h で表 2 のように定義されています.
configure 時の`--with-real` オプションの指定に従い，REAL_TYPE 型を float/double のいずれかに設定しています.

表 1　REAL_TYPE マクロ

`--with-real` オプション	定義
指定無し	#define REAL_TYPE float
`--with-real=float`	#define REAL_TYPE float
`--with-real=double`	#define REAL_TYPE double

・cpm_FaceFlag 列挙型
cpm_FaceFlag 列挙型は，cpm_Define.h で表 2 のように定義されています.
CPM ライブラリから取得した隣接領域番号配列等の 6word の配列を参照する際の，配列インデクスの定義です.

表 2　cpm_FaceFlag 列挙型

cpm_FaceFlag 要素	値	意味
X_MINUS	0	-X 面
X_PLUS	1	+X 面
Y_MINUS	2	-Y 面
Y_PLUS	3	+Y 面
Z_MINUS	4	-Z 面
Z_PLUS	5	+Z 面

・cpm_DirFlag 列挙型
cpm_DirFlag 列挙型は，cpm_Define.h で表 3 のように定義されています.
CPM ライブラリから取得した VOXEL 数配列等の 3word の配列を参照する際の，配列インデクスの定義です．また，周期境界通信関数の周期境界方向を指定するフラグとしても使

われます.

表 3　cpm_DirFlag 列挙型

cpm_DirFlag 要素	値	意味
X_DIR	0	X 方向
Y_DIR	1	Y 方向
Z_DIR	2	Z 方向

・ cpm_PMFlag 列挙型
cpm_PMFlag 列挙型は，cpm_Define.h で表 4 のように定義されています.
周期境界通信関数の周期境界方向を指定するフラグとして使われます.

表 4　cpm_PMFlag 列挙型

cpm_PMFlag 要素	値	意味
PLUS2MINUS	0	+ 側から-側
MINUS2PLUS	1	-側から + 側
BOTH	2	双方向

・ cpm_ErrorCode 列挙型
cpm_ErrorCode 列挙型は，cpm_Define.h で表 5，6 のように定義されています.
CPM ライブラリの API 関数のエラーコードは，全てこの列挙型で定義されています.
Fortan90 インターフェイスのエラーコードにも，この列挙型の値（整数値）がセットされます.

・ CPM_Datatype，CPM_Op 列挙型
CPM_Datatype，CPM_Op 列挙型は，cpm_Define.h で定義されていますが，これらの列挙型は Fortran90 インターフェイスメソッド内で使用されるため，ユーザーが C++ コード内で直接使用することはありません.

表 5　cpm_ErrorCode 列挙型 その 1

cpm_ErrorCode 要素	値	意味
CPM_SUCCESS	0	正常終了
CPM_ERROR	1000	その他のエラー
CPM_ERROR_INVALID_PTR	1002	ポインタのエラー
CPM_ERROR_INVALID_DOMAIN_NO	1003	領域番号が不正
CPM_ERROR_INVALID_OBJKEY	1004	指定登録番号のオブジェクトが存在しない
CPM_ERROR_REGIST_OBJKEY	1005	オブジェクト登録に失敗
CPM_ERROR_TEXTPARSER	2000	テキストパーサーに関するエラー
CPM_ERROR_NO_TEXTPARSER	2001	テキストパーサーを組み込んでいない
CPM_ERROR_TP_INVALID_G_ORG	2004	領域分割情報ファイルのドメイン原点情報が不正
CPM_ERROR_TP_INVALID_G_VOXEL	2005	領域分割情報ファイルのドメイン VOXEL 数情報が不正
CPM_ERROR_TP_INVALID_G_PITCH	2006	領域分割情報ファイルのドメインピッチ情報が不正
CPM_ERROR_TP_INVALID_G_RGN	2007	領域分割情報ファイルのドメイン空間サイズ情報が不正
CPM_ERROR_TP_INVALID_G_DIV	2008	領域分割情報ファイルのドメイン領域分割数情報が不正
CPM_ERROR_TP_INVALID_POS	2009	領域分割情報ファイルのサブドメイン位置情報が不正
CPM_ERROR_VOXELINIT	3000	VoxelInit でエラー
CPM_ERROR_NOT_IN_PROCGROUP	3001	自ランクがプロセスグループに含まれていない
CPM_ERROR_ALREADY_VOXELINIIT	3002	指定されたプロセスグループが既に領域分割済み
CPM_ERROR_MISMATCH_NP_SUBDOMAIN	3003	並列数とサブドメイン数が一致していない
CPM_ERROR_CREATE_RANKMAP	3004	ランクマップ生成に失敗
CPM_ERROR_CREATE_NEIGHBOR	3005	隣接ランク情報生成に失敗
CPM_ERROR_CREATE_LOCALDOMAIN	3006	ローカル領域情報生成に失敗
CPM_ERROR_INSERT_VOXELMAP	3007	領域情報のマップへの登録失敗
CPM_ERROR_CREATE_PROCGROUP	3008	プロセスグループ生成に失敗
CPM_ERROR_INVALID_VOXELSIZE	3009	VOXEL 数が不正
CPM_ERROR_INVALID_REGION	3010	全体空間サイズが不正
CPM_ERROR_INVALID_DIVNUM	3011	領域分割数が不正
CPM_ERROR_GET_INFO	4000	情報取得系関数でエラー
CPM_ERROR_GET_DIVNUM	4001	領域分割数の取得エラー
CPM_ERROR_GET_PITCH	4002	ピッチの取得エラー
CPM_ERROR_GET_GLOBALVOXELSIZE	4003	全体ボクセル数の取得エラー
CPM_ERROR_GET_GLOBALORIGIN	4004	全体空間の原点の取得エラー
CPM_ERROR_GET_GLOBALREGION	4005	全体空間サイズの取得エラー
CPM_ERROR_GET_LOCALVOXELSIZE	4006	自ランクのボクセル数の取得エラー
CPM_ERROR_GET_LOCALORIGIN	4007	自ランクの空間原点の取得エラー
CPM_ERROR_GET_LOCALREGION	4008	自ランクの空間サイズの取得エラー
CPM_ERROR_GET_DIVPOS	4009	自ランクの領域分割位置の取得エラー
CPM_ERROR_GET_HEADINDEX	4011	始点インデクスの取得エラー
CPM_ERROR_GET_TAILINDEX	4012	終点インデクスの取得エラー
CPM_ERROR_GET_NEIGHBOR_RANK	4013	隣接ランク番号の取得エラー
CPM_ERROR_GET_PERIODIC_RANK	4014	周期境界位置の隣接ランク番号の取得エラー
CPM_ERROR_GET_MYRANK	4015	ランク番号の取得エラー
CPM_ERROR_GET_NUMRANK	4016	ランク数の取得エラー

表 6　cpm_ErrorCode 列挙型 その 2

cpm_ErrorCode 要素	値	意味
CPM_ERROR_MPI	9000	MPI のエラー
CPM_ERROR_NO_MPI_INIT	9001	MPI_Init がコールされていない
CPM_ERROR_MPI_BARRIER	9003	MPI_Barrier でエラー
CPM_ERROR_MPI_BCAST	9004	MPI_Bcast でエラー
CPM_ERROR_MPI_SEND	9005	MPI_Send でエラー
CPM_ERROR_MPI_RECV	9006	MPI_Recv でエラー
CPM_ERROR_MPI_ISEND	9007	MPI_Isend でエラー
CPM_ERROR_MPI_IRECV	9008	MPI_Irecv でエラー
CPM_ERROR_MPI_WAIT	9009	MPI_Wait でエラー
CPM_ERROR_MPI_WAITALL	9010	MPI_Waitall でエラー
CPM_ERROR_MPI_ALLREDUCE	9011	MPI_Allreduce でエラー
CPM_ERROR_MPI_GATHER	9012	MPI_Gather でエラー
CPM_ERROR_MPI_ALLGATHER	9013	MPI_Allgather でエラー
CPM_ERROR_MPI_GATHERV	9014	MPI_Gatherv でエラー
CPM_ERROR_MPI_ALLGATHERV	9015	MPI_Allgatherv でエラー
CPM_ERROR_MPI_DIMSCREATE	9016	MPI_Dims_create でエラー
CPM_ERROR_BNDCOMM	9500	BndComm でエラー
CPM_ERROR_BNDCOMM_VOXELSIZE	9501	VoxelSize 取得でエラー
CPM_ERROR_BNDCOMM_BUFFER	9502	袖通信バッファ取得でエラー
CPM_ERROR_BNDCOMM_BUFFERLENGTH	9503	袖通信バッファサイズが足りない
CPM_ERROR_PERIODIC	9600	PeriodicComm でエラー
CPM_ERROR_PERIODIC_INVALID_DIR	9601	不正な軸方向フラグが指定された
CPM_ERROR_PERIODIC_INVALID_PM	9602	不正な正負方向フラグが指定された
CPM_ERROR_MPI_INVALID_COMM	9100	MPI コミュニケータが不正
CPM_ERROR_MPI_INVALID_DATATYPE	9101	対応しない型が指定された
CPM_ERROR_MPI_INVALID_OPERATOR	9102	対応しないオペレータが指定された
CPM_ERROR_MPI_INVALID_REQUEST	9103	不正なリクエストが指定された

4.5 インスタンスの取得

cpm_ParaManager クラスのインスタンスは，Singleton パターンによってプログラム内でただ 1 つ生成されます．そのインスタンスへのポインタを取得するメソッドは，引数の違いによる複数のメソッドが用意されており，cpm_ParaManager.h 内で次のように定義されています．

インスタンスの生成，インスタンスへのポインタの取得

```
static cpm_ParaManager* cpm_ParaManager::get_instance();
```

唯一の cpm_ParaManager クラスのインスタンスへのポインタを取得します．

戻り値 唯一の cpm_ParaManager クラスのインスタンスへのポインタ

インスタンスの生成，インスタンスへのポインタの取得（MPI_Init も実行）

```
static cpm_ParaManager*
cpm_ParaManager::get_instance(int &argc, char**& argv);
```

唯一の cpm_ParaManager クラスのインスタンスへのポインタを取得する．
main 関数の引数を渡すことで，MPI_Init が未実行の場合に，インスタンス生成と同時に MPI_Init も実行する．また，4.6 章の CPM ライブラリ初期化処理も実行する．

`argc`[*input*] main 関数の第 1 引数
`argv`[*input*] main 関数の第 2 引数

戻り値 唯一の cpm_ParaManager クラスのインスタンスへのポインタ

ユーザーの作成するプログラム内では，このメソッドで得られたインスタンスへのポインタを用いて，各メンバ関数へアクセスします．

4.6 CPM ライブラリ初期化処理

CPM ライブラリを利用する上で，プログラムの先頭で 1 回だけ初期化メソッドを呼び出す必要があります．初期化処理を行うメソッドは，引数の違いによる複数のメソッドが用意されており，cpm_ParaManager.h 内で次のように定義されています．

CPM ライブラリ初期化処理

```
cpm_ErrorCode cpm_ParaManager::Initialize();
```

CPM ライブラリ初期化処理を行う．
この関数をコールする前に，MPI_Init がコールされている必要がある．

戻り値 エラーコード（表 5，6 を参照）

CPM ライブラリ初期化処理

```
cpm_ErrorCode cpm_ParaManager::Initialize(int &argc, char**& argv);
```

CPM ライブラリ初期化処理を行う．
main 関数の引数を渡すことで，MPI_Init が未実行の場合に，メソッド内部で MPI_Init を呼び出してから初期化処理を行う．

`argc`[*input*] main 関数の第 1 引数
`argv`[*input*] main 関数の第 2 引数

戻り値 エラーコード（表 5，6 を参照）

CPM ライブラリを利用する C++ プログラムでは，main 関数の先頭で 4.5 章，4.6 章で示す，インスタンス取得と初期化処理を行う必要があります．

API 関数では，それぞれ引数有り/無しの関数が用意されていますが，以下の組み合わせで使用してください．

- パターン 1
  MPI_Init(argc, argv);
  cpm_ParaManager *paraMngr = cpm_ParaManager::get_instance();
  paraMngr->Initialize();

- パターン 2
  cpm_ParaManager *paraMngr = cpm_ParaManager::get_instance();
  paraMngr->Initialize(argc, argv);

- パターン 3
  cpm_ParaManager *paraMngr = cpm_ParaManager::get_instance(argc, argv);

4.7 領域分割情報ファイルの読み込み

CPM ライブラリでは，TextParser ライブラリを用いた領域分割情報ファイルの読み込みをサポートしています.

領域分割情報ファイルは，TextParser ライブラリがサポートする書式で規定された，領域分割数や空間サイズを記述したテキスト形式ファイルです.（詳細は，7 章を参照してください）

領域分割情報ファイルの読み込み処理を行うメソッドは，cpm_TextParserDomain.h 内で次のように定義されています.

領域分割情報ファイルの読み込み

```
static cpm_GlobalDomainInfo*
cpm_TextParserDomain::Read( std::string filename, int &errorcode );
```

領域分割情報ファイルの読み込みを行う.
取得した領域情報のポインタは，4.9 章の領域分割処理にそのまま渡すことが可能．領域分割情報ファイルを利用する場合，実行時の並列数（MPI プロセス並列数）は，活性サブドメイン数以上である必要がある.

`filename`[*input*] 領域分割情報ファイル名
`errorcode`[*output*] エラーコード（表 5，6 を参照）

戻り値 領域情報のポインタ

（注）取得した領域情報のポインタは，不要になった時（VoxelInit 実行後など）にユーザーが delete する必要があります.

```
// 領域分割情報ファイルの読み込み
cpm_GlobalDomainInfo* dInfo = cpm_TextParserDomain::Read( fname, err );
      :
   （処理）
      :
// 不要になったので delete
delete dInfo;
```

4.8 プロセスグループ

CPM ライブラリの領域情報管理には，プロセスグループの概念があります．

プロセスグループを使用することで，複数の計算空間を同時に扱うことができます．プロセスグループは CPM ライブラリ内部で番号で管理されています．初期化後に，全ランクを含むデフォルトのプロセスグループが生成されており，そのプロセスグループ番号は 0 に規定されています．あるプロセスグループに対する処理を行う場合は，API 関数にプロセスグループ番号を指定して呼び出します．API 関数に指定するプロセスグループ番号は省略可能となっており，省略した場合はデフォルトのプロセスグループ番号 0 に対する処理を行います．

デフォルトでは無い，新規のプロセスグループを作成する場合は，cpm_ParaManager.h 内で次のように定義されている API メソッドを使用します．

プロセスグループの作成

```
int cpm_ParaManager::CreateProcessGroup( int nproc, int *proclist
                                       , int parentProcGrpNo=0 );
```

指定されたランクリストを使用してプロセスグループを生成する．

`nproc`[*input*] 生成するプロセスグループのランク数
`proclist`[*input*] 生成するプロセスグループに含むランク番号の配列（サイズ:nproc）
`parentProcGrpNo`[*input*] 親とするプロセスグループの番号（省略時 0）

戻り値 0 以上:生成されたプロセスグループの番号，負値:エラー

4.9 領域分割処理

領域分割処理を行うメソッドは，引数の違いによる複数のメソッドが用意されており，cpm_ParaManager.h 内で次のように定義されています

領域分割処理

```
cpm_ErrorCode
cpm_ParaManager::VoxelInit( cpm_GlobalDomainInfo* domainInfo
                          , size_t maxVC=1, size_t maxN=3
                          , int procGrpNo=0 );
```

領域分割処理を行う．
領域情報のポインタを渡すことで，領域分割情報ファイルで定義された領域情報と活性サブドメイン情報を用いた領域分割を行う．
領域分割情報の活性サブドメイン数と，procGrpNo で指定されたランク数が一致している必要がある．ただし，領域情報内の活性サブドメイン情報が空の場合は，領域分割数で指定された全領域が活性サブドメインに設定される．

`domainInfo`[*input*] 領域情報のポインタ
`maxVC`[*input*] 袖通信バッファ確保用の最大袖層数
`maxN`[*input*] 袖通信バッファ確保用の最大成分数
`procGrpNo`[*input*] 領域分割を行うプロセスグループの番号

戻り値 エラーコード（表 5，6 を参照）

領域分割処理

```
cpm_ErrorCode
cpm_ParaManager::VoxelInit( int div[3], int vox[3], REAL_TYPE origin[3]
                          , REAL_TYPE region[3]
                          , size_t maxVC=1, size_t maxN=3
                          , int procGrpNo=0 );
```

領域分割処理を行う．
引数で指定した X,Y,Z 方向の領域分割数，VOXEL 数，空間原点座標，空間サイズを用いた領域分割を行う．
領域分割数と，procGrpNo で指定されたランク数が一致している必要があり，このメソッドを用いて領域分割を行った場合，全てのサブドメインが活性サブドメインとなる．

`div[`*input*`]`　X,Y,Z 方向の領域分割数（3word の配列，表 3 参照）
`vox[`*input*`]`　X,Y,Z 方向の全体 VOXEL 数（3word の配列，表 3 参照）
`origin[`*input*`]`　X,Y,Z 方向の全体空間原点座標（3word の配列，表 3 参照）
`region[`*input*`]`　X,Y,Z 方向の全体空間サイズ（3word の配列，表 3 参照）
`maxVC[`*input*`]`　袖通信バッファ確保用の最大袖層数
`maxN[`*input*`]`　袖通信バッファ確保用の最大成分数
`procGrpNo[`*input*`]`　領域分割を行うプロセスグループの番号

戻り値　エラーコード（表 5，6 を参照）

領域分割処理

```
cpm_ErrorCode
cpm_ParaManager::VoxelInit( int vox[3], REAL_TYPE origin[3]
                          , REAL_TYPE region[3]
                          , size_t maxVC=1, size_t maxN=3
                          , int procGrpNo=0 );
```

領域分割処理を行う．
指定したプロセスグループのランク数で，自動的に領域分割数を決定し，引数で指定した X,Y,Z 方向の VOXEL 数，空間原点座標，空間サイズを用いた領域分割を行う．
このメソッドを用いて領域分割を行った場合，全てのサブドメインが活性サブドメインとなる．
領域分割数は，隣接領域間の袖通信点数の総数が最小値になるような最適な分割数となる．

`vox`[*input*]　X,Y,Z 方向の全体 VOXEL 数（3word の配列，表 3 参照）
`origin`[*input*]　X,Y,Z 方向の全体空間原点座標（3word の配列，表 3 参照）
`region`[*input*]　X,Y,Z 方向の全体空間サイズ（3word の配列，表 3 参照）
`maxVC`[*input*]　袖通信バッファ確保用の最大袖層数
`maxN`[*input*]　袖通信バッファ確保用の最大成分数
`procGrpNo`[*input*]　領域分割を行うプロセスグループの番号

戻り値　エラーコード（表 5，6 を参照）

4.10 並列情報の取得

並列関連の各種情報の取得関数は，cpm_ParaManager.h 内で次のように定義されています．

並列実行であるかチェックする

```
bool cpm_ParaManager::IsParallel();
```

並列実行であるかチェックする．
mpirun 等で実行していても，並列数が 1 のときは false を返す．

戻り値 true:並列実行，false:逐次実行

ランク数の取得

```
int cpm_ParaManager::GetNumRank( int procGrpNo=0 );
```

指定したプロセスグループのランク数を取得する．

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 ランク数

ランク番号の取得

```
int cpm_ParaManager::GetMyRankID( int procGrpNo=0 );
```

指定したプロセスグループ内の自分自身のランク番号を取得する．

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 ランク番号

NULL のランク番号を取得

```
static int cpm_Base::getRankNull();
```

NULL のランク番号を取得する．

戻り値 MPI_PROC_NULL

NULL のランクかどうかを確認

```
static bool cpm_Base::IsRankNull( int rankNo );
```

NULL のランクかどうかを確認する．無効なランク番号（負値）のとき，true を返す．

`rankNo`[*input*] ランク番号

戻り値 true:NULL，false:有効なランク番号

MPI コミュニケータの取得

```
MPI_Comm cpm_ParaManager::GetMPI_Comm( int procGrpNo=0 );
```

指定したプロセスグループの MPI コミュニケータを取得する．
ユーザーが MPI 関数を用いた独自処理を記述する場合に，本メソッドを用いて，プロセスグループに関連付けされた MPI_Comm を取得する．

`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 MPI コミュニケータ

NULL の MPI コミュニケータを取得

```
static MPI_Comm cpm_Base::getCommNull();
```

NULL の MPI コミュニケータを取得する．

戻り値 MPI_COMM_NULL

NULL の MPI コミュニケータかどうかを確認

```
static bool cpm_Base::IsCommNull( MPI_Comm comm );
```

NULL の MPI コミュニケータかどうかを確認する．無効な MPI コミュニケータ（MPI_COMM_NULL）のとき，true を返す．

`rankNo`[*input*] ランク番号

戻り値 true:NULL，false:有効な MPI コミュニケータ

領域分割数の取得

```
const int* cpm_ParaManager::GetDivNum( int procGrpNo=0 );
```

指定したプロセスグループの領域分割数を取得する.

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 領域分割数配列のポインタ（3word の配列，表 3 参照）

自ランクの領域分割位置を取得

```
const int* cpm_ParaManager::GetDivPos( int procGrpNo=0 );
```

指定したプロセスグループ内での領域分割位置を取得する.

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 領域分割位置配列のポインタ（3word の配列，表 3 参照）

自ランクのホスト名を取得

```
std::string cpm_ParaManager::GetHostName();
```

自ランクのホスト名を取得する.

戻り値 自ランクのホスト名

4.11 全体空間の領域情報取得

全体空間の領域情報取得関数は，cpm_ParaManager.h 内で次のように定義されています.

ピッチの取得

```
const REAL_TYPE* cpm_ParaManager::GetPitch( int procGrpNo=0 );
```

指定したプロセスグループ内での VOXEL ピッチを取得する.

procGrpNo[*input*]　プロセスグループ番号（省略時 0）

戻り値　VOXEL ピッチ配列のポインタ（3word の配列，表 3 参照）

全体空間ボクセル数を取得

```
const int* cpm_ParaManager::GetGlobalVoxelSize( int procGrpNo=0 );
```

指定したプロセスグループ内での全体空間の VOXEL 数を取得する.

procGrpNo[*input*]　プロセスグループ番号（省略時 0）

戻り値　ボクセル数配列のポインタ（3word の配列，表 3 参照）

全体空間の原点座標を取得

```
const REAL_TYPE* cpm_ParaManager::GetGlobalOrigin( int procGrpNo=0 );
```

指定したプロセスグループ内での全体空間の原点座標を取得する.

procGrpNo[*input*]　プロセスグループ番号（省略時 0）

戻り値　原点座標配列のポインタ（3word の配列，表 3 参照）

全体空間の空間サイズを取得

```
const REAL_TYPE* cpm_ParaManager::GetGlobalRegion( int procGrpNo=0 );
```

指定したプロセスグループ内での全体空間の空間サイズを取得する.

procGrpNo[*input*]　プロセスグループ番号（省略時 0）

戻り値　空間サイズ配列のポインタ（3word の配列，表 3 参照）

4.12 ローカル空間の領域情報取得

ローカル空間の領域情報取得関数は，cpm_ParaManager.h 内で次のように定義されています．

自ランクのボクセル数を取得

```
const int* cpm_ParaManager::GetLocalVoxelSize( int procGrpNo=0 );
```

指定したプロセスグループ内での自ランクの VOXEL 数を取得する．

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 ボクセル数配列のポインタ（3word の配列，表 3 参照）

自ランクの原点座標を取得

```
const REAL_TYPE* cpm_ParaManager::GetLocalOrigin( int procGrpNo=0 );
```

指定したプロセスグループ内での自ランクの原点座標を取得する．（図 1 参照）

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 原点座標配列のポインタ（3word の配列，表 3 参照）

自ランクの空間サイズを取得

```
const REAL_TYPE* cpm_ParaManager::GetLocalRegion( int procGrpNo=0 );
```

指定したプロセスグループ内での自ランクの空間サイズを取得する．

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 空間サイズ配列のポインタ（3word の配列，表 3 参照）

自ランクの始点 VOXEL の全体空間でのインデクスを取得

```
const int* cpm_ParaManager::GetVoxelHeadIndex( int procGrpNo=0 );
```

指定したプロセスグループ内での，自ランクの始点 VOXEL の全体空間でのインデクスを取得する．
全体空間における始点 VOXEL のインデクスを 0 としたインデクスが取得される．（図 1 参照）

procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 始点 VOXEL インデクス配列のポインタ（3word の配列，表 3 参照）

自ランクの終点 VOXEL の全体空間でのインデクスを取得

```
const int* cpm_ParaManager::GetVoxelTailIndex( int procGrpNo=0 );
```

指定したプロセスグループ内での，自ランクの終点 VOXEL の全体空間でのインデクスを取得する．
全体空間における始点 VOXEL のインデクスを 0 としたインデクスが取得される．（図 1 参照）

`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　終点 VOXEL インデクス配列のポインタ（3word の配列，表 3 参照）

自ランクの隣接ランク番号を取得

```
const int* cpm_ParaManager::GetNeighborRankID( int procGrpNo=0 );
```

指定したプロセスグループ内での自ランクの隣接ランク番号を取得する．
隣接領域が存在しない面方向には，NULL のランクがセットされている．

`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　隣接ランク番号配列のポインタ（6word の配列，表 2 参照）

ランクの周期境界位置の隣接ランク番号を取得

```
const int* cpm_ParaManager::GetPeriodicRankID( int procGrpNo=0 );
```

指定したプロセスグループ内での自ランクの周期境界の隣接ランク番号を取得する．
内部境界および周期境界位置に隣接領域が存在しない面方向には，NULL のランクがセットされている．

`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　周期境界隣接ランク番号配列のポインタ（6word の配列，表 2 参照）

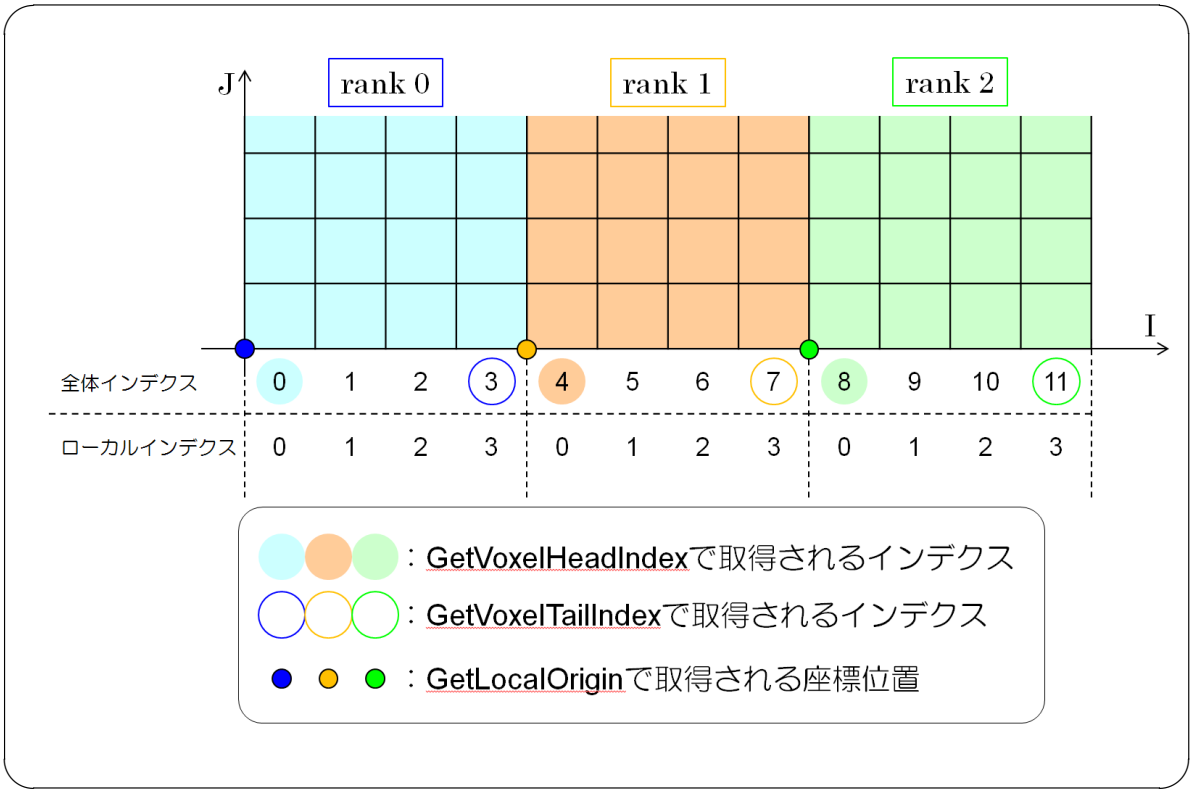


図 1　インデクスと原点座標

指定 id を含む全体ボクセル空間のインデクス範囲を取得

```
bool
cpm_ParaManager::GetBndIndexExtGc( int id, int *array
                                 , int imax, int jmax, int kmax, int vc
                                 , int &ista, int &jsta, int &ksta
                                 , int &ilen, int &jlen, int &klen
                                 , int procGrpNo=0 );
```

指定 id を含む全体ボクセル空間のインデクス範囲を取得する．（図 2 参照）
指定 id がどの領域にも含まれない場合，false が返る．

`id`[*input*] 判定する id
`array`[*input*] 判定対象の配列ポインタ（S3D 形式）
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`ista`[*output*] I 方向範囲のスタートインデクス
`jsta`[*output*] J 方向範囲のスタートインデクス
`ksta`[*output*] K 方向範囲のスタートインデクス
`ilen`[*output*] I 方向範囲の長さ
`jlen`[*output*] J 方向範囲の長さ
`klen`[*output*] K 方向範囲の長さ
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 指定 id が存在したとき true．存在しなかったとき false.

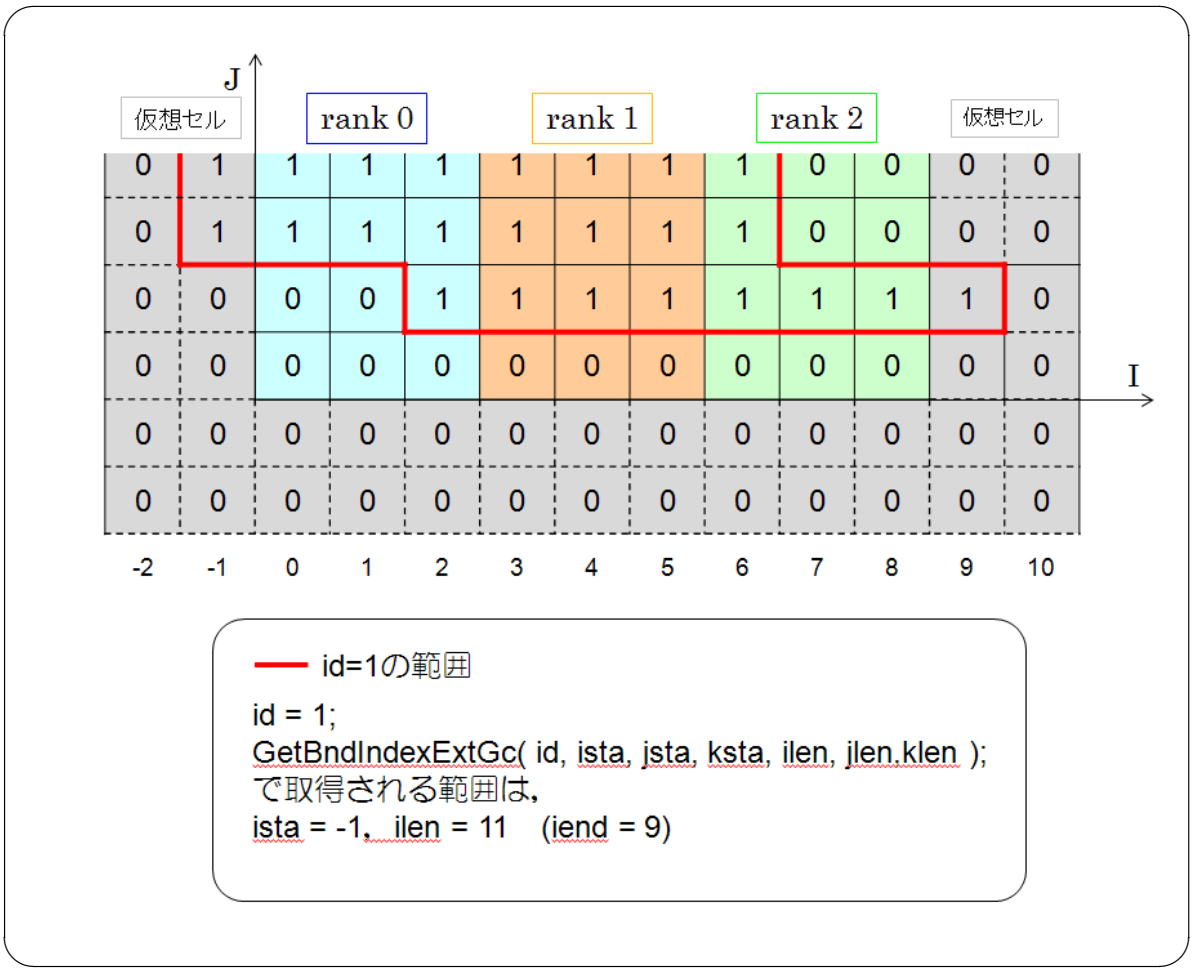


図 2　GetBndIndexExtGc で取得される範囲

4.13 MPI 通信関数

CPM ライブラリでは，プロセスグループ内での並列通信処理メソッドを提供しています．
プロセスグループ内での並列通信処理メソッドは，MPI 関数をラップする形で実装されており，MPI 関数とほぼ同じインターフェイスで利用することができます．

また，一部のメソッドは C++ のテンプレート関数として実装されており，送受信データの MPI_Datatype をメソッド内部で決定しているため，ユーザーは MPI_Datatype を意識せずに利用することができます．

プロセスグループ内での並列通信処理メソッドは，cpm_ParaManager.h 内で次のように定義されています．

MPI_Abort のインターフェイス

```
void cpm_ParaManager::Abort( int errorcode );
```

MPI_Abort のインターフェイスメソッド．

`errorcode`[*input*] MPI_Abort に渡すエラーコード

MPI_Barrier のインターフェイス

```
cpm_ErrorCode cpm_ParaManager::Barrier( int procGrpNo=0 );
```

MPI_Barrier のインターフェイスメソッド．

`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

MPI_Wait のインターフェイス

```
cpm_ErrorCode cpm_ParaManager::Wait( MPI_Request *request );
```

MPI_Wait のインターフェイスメソッド．

`request`[*input*] MPI_Request のポインタ

戻り値 エラーコード（表 5，6 を参照）

MPI_Waitall のインターフェイス

```
cpm_ErrorCode cpm_ParaManager::Waitall( int count, MPI_Request requests[] );
```

MPI_Waitall のインターフェイスメソッド.

`count`[*input*] MPI_Request 数
`requests`[*input*] MPI_Request の配列（サイズ:count）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Bcast のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Bcast( T *buf, int count, int root, int procGrpNo=0 );
```

MPI_Bcast のインターフェイスメソッド.

`buf`[*input/output*] 送受信バッファ
`count`[*input*] 送受信する配列要素数
`root`[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Send のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Send( T *buf, int count, int dest, int procGrpNo=0 );
```

MPI_Send のインターフェイスメソッド.

`buf`[*input*] 送信バッファ
`count`[*input*] 送信する配列要素数
`dest`[*input*] 送信先ランク番号（procGrpNo 内でのランク番号）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Recv のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Recv( T *buf, int count, int source, int procGrpNo=0 );
```

MPI_Recv のインターフェイスメソッド.

buf[*output*] 受信バッファ
count[*input*] 受信する配列要素数
source[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Isend のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Isend( T *buf, int count, int dest, MPI_Request *request
                      , int procGrpNo=0 );
```

MPI_Isend のインターフェイスメソッド.

buf[*input*] 送信バッファ
count[*input*] 送信する配列要素数
dest[*input*] 送信先ランク番号（procGrpNo 内でのランク番号）
request[*output*] MPI_Request のポインタ
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Irecv のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Irecv( T *buf, int count, int source, MPI_Request *request
                      , int procGrpNo=0 );
```

MPI_Irecv のインターフェイスメソッド.

buf[*output*] 受信バッファ
count[*input*] 受信する配列要素数
source[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
request[*output*] MPI_Request のポインタ
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Allreduce のインターフェイス

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::Allreduce( T *sendbuf, T *recvbuf, int count, MPI_Op op
                          , int procGrpNo=0 );
```

MPI_Allreduce のインターフェイスメソッド.

sendbuf[*input*] 送信バッファ
recvbuf[*output*] 受信バッファ
count[*input*] 送受信する配列要素数
op[*input*] オペレータ（MPI_Op）
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Gather のインターフェイス

```
template<class Ts, class Tr>
cpm_ErrorCode cpm_ParaManager::Gather( Ts *sendbuf, int sendcnt
                                     , Tr *recvbuf, int recvcnt
                                     , int root, int procGrpNo=0 );
```

MPI_Gather のインターフェイスメソッド.

sendbuf[*input*] 送信バッファ
sendcnt[*input*] 送信バッファの配列要素数
recvbuf[*output*] 受信バッファ
recvcnt[*input*] 受信バッファの配列要素数
root[*input*] 受信するランク番号（procGrpNo 内でのランク番号）
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Allgather のインターフェイス

```
template<class Ts, class Tr>
cpm_ErrorCode cpm_ParaManager::Allgather( Ts *sendbuf, int sendcnt
                                        , Tr *recvbuf, int recvcnt
                                        , int procGrpNo=0 );
```

MPI_Allgather のインターフェイスメソッド.

`sendbuf`[*input*] 送信バッファ
`sendcnt`[*input*] 送信バッファの配列要素数
`recvbuf`[*output*] 受信バッファ
`recvcnt`[*input*] 受信バッファの配列要素数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Gatherv のインターフェイス

```
template<class Ts, class Tr>
cpm_ErrorCode cpm_ParaManager::Gatherv( Ts *sendbuf, int sendcnt
                                      , Tr *recvbuf, int *recvcnts
                                      , int *displs, int root
                                      , int procGrpNo=0 );
```

MPI_Gatherv のインターフェイスメソッド.

`sendbuf`[*input*] 送信バッファ
`sendcnt`[*input*] 送信バッファの配列要素数
`recvbuf`[*output*] 受信バッファ
`recvcnts`[*input*] 各ランクからの受信データサイズ
`displs`[*input*] 各ランクからの受信データ配置位置
`root`[*input*] 受信するランク番号（procGrpNo 内でのランク番号）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

MPI_Allgatherv のインターフェイス

```
template<class Ts, class Tr>
cpm_ErrorCode cpm_ParaManager::Allgatherv( Ts *sendbuf, int sendcnt
                                         , Tr *recvbuf, int *recvcnts
                                         , int *displs, int procGrpNo=0 );
```

MPI_Allgatherv のインターフェイスメソッド.

`sendbuf`[*input*] 送信バッファ
`sendcnt`[*input*] 送信バッファの配列要素数
`recvbuf`[*output*] 受信バッファ
`recvcnts`[*input*] 各ランクからの受信データサイズ
`displs`[*input*] 各ランクからの受信データ配置位置
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

4.14 内部境界袖通信メソッド

CPM ライブラリでは，領域分割空間での隣接領域間（内部境界）の袖領域の通信メソッドを提供しています．袖領域の通信メソッドは，配列形状毎に用意されています．

また，C++ のテンプレート関数として実装されており，送受信データの MPI_Datatype をメソッド内部で決定しているため，ユーザーは MPI_Datatype を意識せずに利用することができます．

袖領域の通信メソッドは，cpm_ParaManager.h 内で次のように定義されています．

袖通信バッファのセット

```
cpm_ErrorCode
cpm_ParaManager::SetBndCommBuffer( size_t maxVC, size_t maxN
                                 , int procGrpNo=0 );
```

袖通信で使用する転送データ格納用バッファの確保を行う．
VoxelInit（4.9 章を参照）実行時にも実行されるが，プログラムの途中でバッファサイズを変更したい場合に呼び出す．

maxVC[*input*] 袖通信バッファ確保用の最大袖層数
maxN[*input*] 袖通信バッファ確保用の最大成分数
procGrpNo[*input*] バッファを確保するプロセスグループの番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

袖通信バッファサイズの取得

```
size_t cpm_ParaManager::GetBndCommBufferSize( int procGrpNo=0 );
```

袖通信で使用する転送データ格納用バッファの確保済みサイズを取得する．

procGrpNo[*input*] バッファサイズを取得するプロセスグループの番号（省略時 0）

戻り値 確保済みのバッファサイズ（byte）

袖通信 (Scalar3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS3D( T *array, int imax, int jmax, int kmax
                           , int vc, int vc_comm, int procGrpNo=0 );
```

Scalar3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

袖通信 (Vector3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommV3D( T *array, int imax, int jmax, int kmax
                           , int vc, int vc_comm, int procGrpNo=0 );
```

Vector3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う.（成分数=3）

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

袖通信 (Scalar4D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS4D( T *array, int imax, int jmax, int kmax
                           , int nmax, int vc, int vc_comm
                           , int procGrpNo=0 );
```

Scalar4D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`nmax`[*input*] 成分数
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

袖通信 (Vector3DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommV3DEx( T *array, int imax, int jmax, int kmax
                             , int vc, int vc_comm, int procGrpNo=0 );
```

Vector3DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=3）

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

袖通信 (Scalar4DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS4DEx( T *array, int nmax
                             , int imax, int jmax, int kmax
                             , int vc, int vc_comm, int procGrpNo=0 );
```

Scalar4DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`nmax`[*input*] 成分数
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

4.15 内部境界袖通信メソッド（非同期版）

非同期版の内部境界袖通信メソッドは，送信データのパックと非同期送受信処理をするメソッドと，通信完了の待機と受信データの展開をするメソッドに分かれており，非同期通信中に他の計算処理を実行することが可能です.

※ 袖通信メソッドは通信用の送受信バッファを共有しているため，非同期版の袖通信メソッドを使用する場合，その非同期通信中に他の袖通信メソッド（内部境界，周期境界，同期，非同期版の全て）を呼び出すと，通信結果が保証されません.
なお，袖通信以外の通信メソッドは袖通信用の共有バッファを使用しないため，非同期袖通信中であっても使用することができます.

非同期版の袖領域通信メソッドは，cpm_ParaManager.h 内で次のように定義されています.

非同期袖通信 (Scalar3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS3D_nowait( T *array, int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , MPI_Request req[12], int procGrpNo=0 );
```

Scalar3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.
通信完了待ちと受信データの展開は行わない.

`array`[*input*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*output*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar3D 版) の待機，展開処理

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::wait_BndCommS3D( T *array, int imax, int jmax, int kmax
                                , int vc, int vc_comm
                                , MPI_Request req[12], int procGrpNo=0 );
```

BndCommS3D_nowait メソッドの通信完了待ちと受信データの展開を行う．

array[*input/output*] 袖通信をする配列の先頭ポインタ
imax[*input*] 配列サイズ (I 方向)
jmax[*input*] 配列サイズ (J 方向)
kmax[*input*] 配列サイズ (K 方向)
vc[*input*] 仮想セル数
vc_comm[*input*] 袖通信を行う仮想セル数
req[*input*] MPI_Request 配列（サイズ 12）
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommV3D_nowait( T *array, int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , MPI_Request req[12], int procGrpNo=0 );
```

Vector3D 形状の配列の内部境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う．（成分数=3）
通信完了待ちと受信データの展開は行わない．

array[*input*] 袖通信をする配列の先頭ポインタ
imax[*input*] 配列サイズ (I 方向)
jmax[*input*] 配列サイズ (J 方向)
kmax[*input*] 配列サイズ (K 方向)
vc[*input*] 仮想セル数
vc_comm[*input*] 袖通信を行う仮想セル数
req[*output*] MPI_Request 配列（サイズ 12）
procGrpNo[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3D 版) の待機，展開処理

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::wait_BndCommV3D( T *array, int imax, int jmax, int kmax
                                , int vc, int vc_comm
                                , MPI_Request req[12], int procGrpNo=0 );
```

BndCommV3D_nowait メソッドの通信完了待ちと受信データの展開を行う.

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*input*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS4D_nowait( T *array, int imax, int jmax, int kmax
                                  , int nmax, int vc, int vc_comm
                                  , MPI_Request req[12]
                                  , int procGrpNo=0 );
```

Scalar4D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）
通信完了待ちと受信データの展開は行わない.

`array`[*input*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`nmax`[*input*] 成分数
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*output*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4D 版) の待機，展開処理

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::wait_BndCommS4D( T *array
                                , int imax, int jmax, int kmax, int nmax
                                , int vc, int vc_comm
                                , MPI_Request req[12], int procGrpNo=0 );
```

BndCommS4D_nowait メソッドの通信完了待ちと受信データの展開を行う．

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`nmax`[*input*] 成分数
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*input*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommV3DEx_nowait( T *array
                                    , int imax, int jmax, int kmax
                                    , int vc, int vc_comm
                                    , MPI_Request req[12], int procGrpNo=0 );
```

Vector3DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=3）
通信完了待ちと受信データの展開は行わない.

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*input*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3DEx 版) の待機，展開処理

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::wait_BndCommV3DEx( T *array
                                  , int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , MPI_Request req[12], int procGrpNo=0 );
```

BndCommV3DEx_nowait メソッドの通信完了待ちと受信データの展開を行う.

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*input*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::BndCommS4DEx_nowait( T *array
                                    , int nmax, int imax, int jmax, int kmax
                                    , int vc, int vc_comm
                                    , MPI_Request req[12], int procGrpNo=0 );
```

Scalar4DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）
通信完了待ちと受信データの展開は行わない.

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`nmax`[*input*] 成分数
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*output*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

非同期袖通信 (Scalar4DEx 版) の待機，展開処理

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::wait_BndCommS4DEx( T *array
                                  , int nmax, int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , MPI_Request req[12], int procGrpNo=0 );
```

BndCommS4DEx_nowait メソッドの通信完了待ちと受信データの展開を行う．

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`nmax`[*input*] 成分数
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`req`[*input*] MPI_Request 配列（サイズ 12）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5，6 を参照）

4.16 周期境界袖通信メソッド

CPM ライブラリでは，領域分割空間での周期境界（外部境界）の袖領域の通信メソッドを提供しています．袖領域の通信メソッドは，配列形状毎に用意されています．

また，C++ のテンプレート関数として実装されており，送受信データの MPI_Datatype をメソッド内部で決定しているため，ユーザーは MPI_Datatype を意識せずに利用することができます．

周期境界の袖通信メソッドは，cpm_ParaManager.h 内で次のように定義されています．

周期境界袖通信 (Scalar3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::PeriodicCommS3D( T *array
                                , int imax, int jmax, int kmax
                                , int vc, int vc_comm
                                , cpm_DirFlag dir, cpm_PMFlag pm
                                , int procGrpNo=0 );
```

Scalar3D 形状の配列の周期境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う．

`array`[*input/output*]　袖通信をする配列の先頭ポインタ
`imax`[*input*]　配列サイズ (I 方向)
`jmax`[*input*]　配列サイズ (J 方向)
`kmax`[*input*]　配列サイズ (K 方向)
`vc`[*input*]　仮想セル数
`vc_comm`[*input*]　袖通信を行う仮想セル数
`dir`[*input*]　袖通信を行う軸方向フラグ（表 3 を参照）
`pm`[*input*]　袖通信を行う正負方向フラグ（表 4 を参照）
`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　エラーコード（表 5，6 を参照）

周期境界袖通信 (Vector3D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::PeriodicCommV3D( T *array
                                , int imax, int jmax, int kmax
                                , int vc, int vc_comm
                                , cpm_DirFlag dir, cpm_PMFlag pm
                                , int procGrpNo=0 );
```

Vector3D 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う（成分数=3).

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`dir`[*input*] 袖通信を行う軸方向フラグ（表 3 を参照）
`pm`[*input*] 袖通信を行う正負方向フラグ（表 4 を参照）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

周期境界袖通信 (Scalar4D 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::PeriodicCommS4D( T *array
                                , int imax, int jmax, int kmax, int nmax
                                , int vc, int vc_comm
                                , cpm_DirFlag dir, cpm_PMFlag pm
                                , int procGrpNo=0 );
```

Scalar4D 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う（成分数=nmax）.

`array`[*input/output*]　袖通信をする配列の先頭ポインタ
`imax`[*input*]　配列サイズ (I 方向)
`jmax`[*input*]　配列サイズ (J 方向)
`kmax`[*input*]　配列サイズ (K 方向)
`nmax`[*input*]　成分数
`vc`[*input*]　仮想セル数
`vc_comm`[*input*]　袖通信を行う仮想セル数
`dir`[*input*]　袖通信を行う軸方向フラグ（表 3 を参照）
`pm`[*input*]　袖通信を行う正負方向フラグ（表 4 を参照）
`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　エラーコード（表 5, 6 を参照）

周期境界袖通信 (Vector3DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::PeriodicCommV3DEx( T *array
                                  , int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , cpm_DirFlag dir, cpm_PMFlag pm
                                  , int procGrpNo=0 );
```

Vector3DEx 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う（成分数=3).

`array`[*input/output*] 袖通信をする配列の先頭ポインタ
`imax`[*input*] 配列サイズ (I 方向)
`jmax`[*input*] 配列サイズ (J 方向)
`kmax`[*input*] 配列サイズ (K 方向)
`vc`[*input*] 仮想セル数
`vc_comm`[*input*] 袖通信を行う仮想セル数
`dir`[*input*] 袖通信を行う軸方向フラグ（表 3 を参照）
`pm`[*input*] 袖通信を行う正負方向フラグ（表 4 を参照）
`procGrpNo`[*input*] プロセスグループ番号（省略時 0）

戻り値 エラーコード（表 5, 6 を参照）

周期境界袖通信 (Scalar4DEx 版)

```
template<class T>
cpm_ErrorCode
cpm_ParaManager::PeriodicCommS4DEx( T *array
                                  , int nmax, int imax, int jmax, int kmax
                                  , int vc, int vc_comm
                                  , cpm_DirFlag dir, cpm_PMFlag pm
                                  , int procGrpNo=0 );
```

Scalar4DEx 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う（成分数=nmax）.

`array`[*input/output*]　袖通信をする配列の先頭ポインタ
`nmax`[*input*]　成分数
`imax`[*input*]　配列サイズ (I 方向)
`jmax`[*input*]　配列サイズ (J 方向)
`kmax`[*input*]　配列サイズ (K 方向)
`vc`[*input*]　仮想セル数
`vc_comm`[*input*]　袖通信を行う仮想セル数
`dir`[*input*]　袖通信を行う軸方向フラグ（表 3 を参照）
`pm`[*input*]　袖通信を行う正負方向フラグ（表 4 を参照）
`procGrpNo`[*input*]　プロセスグループ番号（省略時 0）

戻り値　エラーコード（表 5, 6 を参照）

5 Fortran90 ユーザープログラムでの利用方法

以下に，CPM ライブラリの Fortran90 インターフェイスの説明を示します．

なお，以降の Fortran90 インターフェイスの説明では，各メソッドを Fortran90 サブルーチンの形式で記述していますが，実際には「`extern "C"`」定義の C 言語型インターフェイスの C++ メソッドとして記述されています．インターフェイスメソッドでは，cpm_ParaManager クラスの唯一のインスタンスを経由して，CPM ライブラリの各種機能にアクセスしています．

5.1 実数型

CPM ライブラリの Fortran90 インターフェイスで扱われる実数型は，全て REAL_TYPE 型（2.3 章および表 1 参照）に対応した実数型になります．

この文書内で説明のある Fortran90 インターフェイスの real 型については，configure 時オプションで指定した REAL_TYPE と同じ型を使用してください．

- `--with-real=float` 指定もしくは未指定の場合
  real 型は，real*4 もしくはデフォルト kind=4 とした real を使用する．

- `--with-real=double` 指定の場合
  real 型は，real*8 もしくはデフォルト kind=8 とした real を使用する．

5.2 MPI リクエスト番号

CPM ライブラリの Fortran90 インターフェイスが提供する MPI 通信関数は，ライブラリ内部で C++ コードから MPI 関数を呼び出しています．

Fortran90 では，C++ で扱う MPI_Request 型を扱うことができないため，Fortran90 インターフェイスの非同期通信関数で用いる MPI リクエストは，CPM ライブラリ内部で番号管理しています．

5.3 サンプルプログラム

Examples/f90 ディレクトリに，Fortran90 ユーザープログラムでの CPM ライブラリ使用例のサンプルソースコードが収められています．

- `mconvp_CPM.f90`
  入力ファイルの読み込み，全ランクへの展開，領域分割実行を行い，poisson 方程式を計算するサンプルです．

このサンプルコードは make 時に一緒にコンパイル，リンクされ，`mconvp_CPM` という実行ファイルが作成されます．`data` ディレクトリに実行サンプルが収められています．

5.4 cpm_fparam.fi のインクルード

CPM ライブラリの Fortran90 インターフェイスで使用する各種定義は，CPM ライブラリが提供するヘッダファイル cpm_fparam.fi で定義されています．CPM ライブラリの Fortran90 インターフェイスを使う場合は，この cpm_fparam.fi ファイルをインクルードします．

cpm_fparam.fi は，configure スクリプト実行時の設定 `prefix` 配下の`${prefix}`/include に `make install` 時にインストールされます．

5.5 parameter 定義

CPM ライブラリの Fortran90 インターフェイスで使用される定数は，cpm_fparam.fi に定義されています．

・戻り値
Fortran90 インターフェイス関数では，引数に必ず処理の戻り値を設定します．処理が正常終了した場合，この戻り値には 0 がセットされます．インターフェイス関数内でエラーが発生した場合，この戻り値には cpm_ErrorCode 列挙型（5，6 を参照）の値が整数値としてセットされます．
cpm_fparam.fi では，正常終了の場合の戻り値 0 を CPM_SUCCESS 定数として定義しています．（表 7）

表 7　戻り値定数

定数名	値	意味
CPM_SUCCESS	0	正常終了

・デフォルトのプロセスグループ番号
CPM ライブラリは，プロセスグループの機能を提供しており，並列プロセス内に存在する複数の計算空間をプロセスグループとして定義しています．プロセスグループは CPM ライブラリ内部で番号で管理されており，デフォルトのグループ番号は 0 とされています．
cpm_fparam.fi では, このデフォルトのプロセスグループ番号を, CPM_DEFAULT_GROUP 定数として定義しています．（表 8）

表 8　プロセスグループ番号定数

定数名	値	意味
CPM_DEFAULT_GROUP	0	デフォルトのプロセスグループ番号

・面フラグ

CPM ライブラリから取得した隣接領域番号配列等の 6word の配列を参照する際の，配列インデクスの定義です．

配列の宣言方法により，2 種類のインデクス定義を使い分ける必要があります．（表 9，10）

表 9　面フラグ定数（0 スタート）

定数名	値	意味
X_MINUS	0	-X 面
X_PLUS	1	+X 面
Y_MINUS	2	-Y 面
Y_PLUS	3	+Y 面
Z_MINUS	4	-Z 面
Z_PLUS	5	+Z 面

表 10　面フラグ定数（1 スタート）

定数名	値	意味
X_MINUS1	1	-X 面
X_PLUS1	2	+X 面
Y_MINUS1	3	-Y 面
Y_PLUS1	4	+Y 面
Z_MINUS1	5	-Z 面
Z_PLUS1	6	+Z 面

・軸方向フラグ

CPM ライブラリから取得した VOXEL 数配列等の 3word の配列を参照する際の，配列インデクスの定義です．また，周期境界通信関数の周期境界方向を指定するフラグとしても使われます．（表 11）

・正負方向フラグ

周期境界通信関数の周期境界方向を指定するフラグとして使われます．（表 12）

表 11　軸方向フラグ定数

定数名	値	意味
X_DIR	0	X 方向
Y_DIR	1	Y 方向
Z_DIR	2	Z 方向

表 12　正負方向フラグ定数

定数名	値	意味
PLUS2MINUS	0	+ 側から-側
MINUS2PLUS	1	-側から + 側
BOTH	2	双方向

・Fortran データ型

CPM ライブラリが提供する MPI 通信関係の Fortran90 インターフェイスでは，通信データのデータ型を指定する必要があります.

データ型は MPI_Datatype に対応した独自定数として定義されています.（表 13）

表 13　Fortan データ型定数

定数名	値	意味
CPM_INT	6	MPI_INTEGER に対応
CPM_INTEGER	6	MPI_INTEGER に対応
CPM_REAL	52	REAL_TYPE に対応
CPM_FLOAT	10	MPI_REAL4 に対応
CPM_REAL4	10	MPI_REAL4 に対応
CPM_DOUBLE	11	MPI_REAL8 に対応
CPM_REAL8	11	MPI_REAL8 に対応

・Fortran オペレータタイプ

CPM ライブラリが提供する reduce 通信関係の Fortran90 インターフェイスでは，reduce のオペレータタイプを指定する必要があります.

オペレータタイプは MPI_Op に対応した独自定数として定義されています.（表 14）

表 14　Fortran オペレータタイプ

定数名	値	意味
CPM_MAX	100	MPI_MAX に対応
CPM_MIN	101	MPI_MIN に対応
CPM_SUM	102	MPI_SUM に対応
CPM_PROD	103	MPI_PROD に対応
CPM_LAND	104	MPI_LAND に対応
CPM_BAND	105	MPI_BAND に対応
CPM_LOR	106	MPI_LOR に対応
CPM_BOR	107	MPI_BOR に対応
CPM_LXOR	108	MPI_LXOR に対応
CPM_BXOR	109	MPI_BXOR に対応

5.6 CPM ライブラリ初期化処理

CPM ライブラリを利用する上で，プログラムの先頭で 1 回だけ初期化メソッドを呼び出す必要があります.

CPM ライブラリ初期化処理

```
subroutine cpm_Initialize( ierr )
```

CPM ライブラリ初期化処理を行う.
この関数をコールする前に，ユーザーの Fortran90 プログラム内で MPI_INIT がコールされている必要がある.

integer ierr[*output*] エラーコード（表 5，6 を参照）

5.7 領域分割処理

領域分割処理を行うメソッドは，引数の違いによる複数のメソッドが用意されており，次のように定義されています

領域分割処理

```
subroutine cpm_VoxelInit( div, vox, origin, region, maxVC, maxN
                        , procGrpNo, ierr )
```

領域分割処理を行う.
引数で指定した X,Y,Z 方向の領域分割数，VOXEL 数，空間原点座標，空間サイズを用いた領域分割を行う.
領域分割数と，procGrpNo で指定されたランク数が一致している必要があり，このメソッドを用いて領域分割を行った場合，全てのサブドメインが活性サブドメインとなる.

integer div[*input*] X,Y,Z 方向の領域分割数（3word の配列）
integer vox[*input*] X,Y,Z 方向の全体 VOXEL 数（3word の配列）
real origin[*input*] X,Y,Z 方向の全体空間原点座標（3word の配列）
real region[*input*] X,Y,Z 方向の全体空間サイズ（3word の配列）
integer maxVC[*input*] 袖通信バッファ確保用の最大袖層数
integer maxN[*input*] 袖通信バッファ確保用の最大成分数
integer procGrpNo[*input*] 領域分割を行うプロセスグループの番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

領域分割処理

```
subroutine cpm_VoxelInit_nodiv( vox, origin, region, maxVC, maxN
                              , procGrpNo, ierr )
```

領域分割処理を行う.
指定したプロセスグループのランク数で，自動的に領域分割数を決定し，引数で指定した X,Y,Z 方向の VOXEL 数，空間原点座標，空間サイズを用いた領域分割を行う.
このメソッドを用いて領域分割を行った場合，全てのサブドメインが活性サブドメインとなる.
領域分割数は，隣接領域間の袖通信点数の総数が最小値になるような最適な分割数となる.

integer `vox`[*input*] X,Y,Z 方向の全体 VOXEL 数（3word の配列）
real `origin`[*input*] X,Y,Z 方向の全体空間原点座標（3word の配列）
real `region`[*input*] X,Y,Z 方向の全体空間サイズ（3word の配列）
integer `maxVC`[*input*] 袖通信バッファ確保用の最大袖層数
integer `maxN`[*input*] 袖通信バッファ確保用の最大成分数
integer `procGrpNo`[*input*] 領域分割を行うプロセスグループの番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

5.8 並列情報の取得

並列関連の各種情報の取得関数は，次のように定義されています．

並列実行であるかチェックする

```
subroutine cpm_IsParallel( ipara, ierr )
```

並列実行であるかチェックする．
並列実行時は ipara に 1 がセットされ，逐次実行時は ipara に 1 以外がセットされる．
mpirun 等で実行していても，並列数が 1 のときは逐次実行と判断される．

integer ipara[*output*] 1:並列実行，1 以外:逐次実行
integer ierr[*output*] エラーコード（表 5，6 を参照）

ランク数の取得

```
subroutine cpm_GetNumRank( nrank, procGrpNo, ierr )
```

指定したプロセスグループのランク数を取得する．

integer nrank[*output*] ランク数
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

ランク番号の取得

```
subroutine cpm_GetMyRankID( idm procGrpNo, ierr )
```

指定したプロセスグループ内の自分自身のランク番号を取得する．

integer id[*output*] ランク番号
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

領域分割数の取得

```
subroutine cpm_GetDivNum( div, procGrpNo, ierr )
```

指定したプロセスグループの領域分割数を取得する．

integer div[*output*] 領域分割数（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの領域分割位置を取得

```
subroutine cpm_GetDivPos( pos, procGrpNo, ierr )
```

指定したプロセスグループ内での領域分割位置を取得する.

integer pos[*output*] 領域分割位置（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

5.9 全体空間の領域情報取得

全体空間の領域情報取得関数は，cpm_ParaManager.h 内で次のように定義されています.

ピッチの取得

```
subroutine cpm_GetPitch( pch, procGrpNo, ierr )
```

指定したプロセスグループ内での VOXEL ピッチを取得する.

real pch[*output*] VOXEL ピッチ（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

全体空間ボクセル数を取得

```
subroutine cpm_GetGlobalVoxelSize( wsz, procGrpNo, ierr )
```

指定したプロセスグループ内での全体空間の VOXEL 数を取得する.

integer wsz[*output*] VOXEL 数（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

全体空間の原点座標を取得

```
subroutine cpm_GetGlobalOrigin( worg, procGrpNo, ierr )
```

指定したプロセスグループ内での全体空間の原点座標を取得する.

real worg[*output*] 原点座標（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

全体空間の空間サイズを取得

```
subroutine cpm_GetGlobalRegion( wrgn, procGrpNo, ierr )
```

指定したプロセスグループ内での全体空間の空間サイズを取得する.

real wrgn[*output*] 空間サイズ（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

5.10 ローカル空間の領域情報取得

ローカル空間の領域情報取得関数は，次のように定義されています．

自ランクのボクセル数を取得

```
subroutine cpm_GetLocalVoxelSize( lsz, procGrpNo, ierr )
```

指定したプロセスグループ内での自ランクの VOXEL 数を取得する．

integer lsz[*output*] VOXEL 数（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの原点座標を取得

```
subroutine cpm_GetLocalOrigin( lorg, procGrpNo, ierr )
```

指定したプロセスグループ内での自ランクの原点座標を取得する．

real lorg[*output*] 原点座標（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの空間サイズを取得

```
subroutine cpm_GetLocalRegion( lrgn, procGrpNo, ierr )
```

指定したプロセスグループ内での自ランクの空間サイズを取得する．

real lrgn[*output*] 空間サイズ（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの始点 VOXEL の全体空間でのインデクスを取得

```
subroutine cpm_GetVoxelHeadIndex( idx, procGrpNo, ierr )
```

指定したプロセスグループ内での，自ランクの始点 VOXEL の全体空間でのインデクスを取得する．
全体空間における始点 VOXEL のインデクスを 0 としたインデクスが取得される．

integer idx[*output*] 始点 VOXEL インデクス（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの終点 VOXEL の全体空間でのインデクスを取得

```
subroutine cpm_GetVoxelTailIndex( idx, procGrpNo, ierr )
```

指定したプロセスグループ内での，自ランクの終点 VOXEL の全体空間でのインデクスを取得する．
全体空間における始点 VOXEL のインデクスを 0 としたインデクスが取得される．

integer idx[*output*] 終点 VOXEL インデクス（3word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの隣接ランク番号を取得

```
subroutine cpm_GetNeighborRankID( nID, procGrpNo, ierr )
```

指定したプロセスグループ内での自ランクの隣接ランク番号を取得する．
隣接領域が存在しない面方向には，NULL のランクがセットされている．

integer nID[*output*] 隣接ランク番号（6word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

自ランクの周期境界位置の隣接ランク番号を取得

```
subroutine cpm_GetPeriodicRankID( nID, procGrpNo, ierr )
```

指定したプロセスグループ内での自ランクの周期境界の隣接ランク番号を取得する．
内部境界および周期境界位置に隣接領域が存在しない面方向には，NULL のランクがセットされている．

integer nID[*output*] 周期境界位置の隣接ランク番号（6word の配列）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

5.11 MPI 通信関数

CPM ライブラリでは，プロセスグループ内での並列通信処理メソッドを提供しています．

プロセスグループ内での並列通信処理メソッドは，MPI 関数をラップする形で実装されており，MPI 関数とほぼ同じインターフェイスで利用することができます．

プロセスグループ内での並列通信処理メソッドは，次のように定義されています．

MPI_Abort のインターフェイス

```
subroutine cpm_Abort( errorcode )
```

MPI_Abort のインターフェイスメソッド．

errorcode[*input*] MPI_Abort に渡すエラーコード

MPI_Barrier のインターフェイス

```
subroutine cpm_Barrier( procGrpNo, ierr )
```

MPI_Barrier のインターフェイスメソッド．

integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

MPI_Wait のインターフェイス

```
subroutine cpm_Wait( reqNo, ierr )
```

MPI_Wait のインターフェイスメソッド．

integer reqNo[*input*] リクエスト番号（5.2 章を参照）
integer ierr[*output*] エラーコード（表 5，6 を参照）

MPI_Waitall のインターフェイス

```
subroutine cpm_Waitall( count, reqlist, ierr )
```

MPI_Waitall のインターフェイスメソッド．

integer count[*input*] MPI_Request 数
integer reqlist[*input*] リクエスト番号配列（5.2 章を参照）
integer ierr[*output*] エラーコード（表 5，6 を参照）

MPI_Bcast のインターフェイス

```
subroutine cpm_Bcast( buf, count, datatype, root, procGrpNo, ierr )
```

MPI_Bcast のインターフェイスメソッド.

void buf[*input/output*] 送受信バッファ
integer count[*input*] 送受信する配列要素数
integer datatype[*input*] 送受信バッファのデータタイプ（表 13 を参照）
integer root[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Send のインターフェイス

```
subroutine cpm_Send( buf, count, datatype, dest, procGrpNo, ierr )
```

MPI_Send のインターフェイスメソッド.

void buf[*input*] 送信バッファ
integer count[*input*] 送信する配列要素数
integer datatype[*input*] 送信バッファのデータタイプ（表 13 を参照）
integer dest[*input*] 送信先ランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Recv のインターフェイス

```
subroutine cpm_Recv( buf, count, datatype, source, procGrpNo, ierr )
```

MPI_Recv のインターフェイスメソッド.

void buf[*output*] 受信バッファ
integer count[*input*] 受信する配列要素数
integer datatype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer source[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Isend のインターフェイス

```
subroutine cpm_Isend( buf, count, datatype, dest, procGrpNo, reqNo, ierr )
```

MPI_Isend のインターフェイスメソッド.

void buf[*input*] 送信バッファ
integer count[*input*] 送信する配列要素数
integer datatype[*input*] 送信バッファのデータタイプ（表 13 を参照）
integer dest[*input*] 送信先ランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer reqNo[*output*] リクエスト番号（5.2 章を参照）
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Irecv のインターフェイス

```
subroutine cpm_Irecv( buf, count, datatype, source, procGrpNo, reqNo
                    , ierr )
```

MPI_Irecv のインターフェイスメソッド.

void buf[*output*] 受信バッファ
integer count[*input*] 受信する配列要素数
integer datatype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer source[*input*] 送信元ランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer reqNo[*output*] リクエスト番号（5.2 章を参照）
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Allreduce のインターフェイス

```
subroutine cpm_Allreduce( sendbuf, recvbuf, count, datatype, op, procGrpNo
                        , ierr )
```

MPI_Allreduce のインターフェイスメソッド.

void sendbuf[*input*] 送信バッファ
void recvbuf[*output*] 受信バッファ
integer count[*input*] 送受信する配列要素数
integer datatype[*input*] 送受信バッファのデータタイプ（表 13 を参照）
integer op[*input*] オペレータ（表 14 を参照）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Gather のインターフェイス

```
subroutine cpm_Gather( sendbuf, sendcnt, sendtype
                     , recvbuf, recvcnt, recvtype
                     , root, procGrpNo, ierr )
```

MPI_Gather のインターフェイスメソッド.

void sendbuf[*input*] 送信バッファ
integer sendcnt[*input*] 送信バッファの配列要素数
integer sendtype[*input*] 送信バッファのデータタイプ（表 13 を参照）
void recvbuf[*output*] 受信バッファ
integer recvcnt[*input*] 受信バッファの配列要素数
integer recvtype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer root[*input*] 受信するランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Allgather のインターフェイス

```
subroutine cpm_Allgather( sendbuf, sendcnt, sendtype
                        , recvbuf, recvcnt, recvtype
                        , procGrpNo, ierr )
```

MPI_Allgather のインターフェイスメソッド.

void sendbuf[*input*] 送信バッファ
integer sendcnt[*input*] 送信バッファの配列要素数
integer sendtype[*input*] 送信バッファのデータタイプ（表 13 を参照）
void recvbuf[*output*] 受信バッファ
integer recvcnt[*input*] 受信バッファの配列要素数
integer recvtype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Gatherv のインターフェイス

```
subroutine cpm_Gatherv( sendbuf, sendcnt, sendtype
                      , recvbuf, recvcnts, displs, recvtype
                      , root, procGrpNo, ierr )
```

MPI_Gatherv のインターフェイスメソッド.

void sendbuf[*input*] 送信バッファ
integer sendcnt[*input*] 送信バッファの配列要素数
integer sendtype[*input*] 送信バッファのデータタイプ（表 13 を参照）
void recvbuf[*output*] 受信バッファ
integer recvcnts[*input*] 各ランクからの受信データサイズ
integer displs[*input*] 各ランクからの受信データ配置位置
integer recvtype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer root[*input*] 受信するランク番号（procGrpNo 内でのランク番号）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

MPI_Allgatherv のインターフェイス

```
subroutine cpm_Allgatherv( sendbuf, sendcnt, sendtype
                         , recvbuf, recvcnts, displs, recvtype
                         , procGrpNo, ierr )
```

MPI_Allgatherv のインターフェイスメソッド.

void sendbuf[*input*] 送信バッファ
integer sendcnt[*input*] 送信バッファの配列要素数
integer sendtype[*input*] 送信バッファのデータタイプ（表 13 を参照）
void recvbuf[*output*] 受信バッファ
integer recvcnts[*input*] 各ランクからの受信データサイズ
integer displs[*input*] 各ランクからの受信データ配置位置
integer recvtype[*input*] 受信バッファのデータタイプ（表 13 を参照）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5, 6 を参照）

5.12 内部境界袖通信メソッド

CPM ライブラリでは，領域分割空間での隣接領域間（内部境界）の袖領域の通信メソッドを提供しています．袖領域の通信メソッドは，配列形状毎に用意されています．

袖領域の通信メソッドは，次のように定義されています．

袖通信バッファのセット

```
subroutine cpm_SetBndCommBuffer( maxVC, maxN, procGrpNo, ierr )
```

袖通信で使用する転送データ格納用バッファの確保を行う．
VoxelInit（4.9 章を参照）実行時にも実行されるが，プログラムの途中でバッファサイズを変更したい場合に呼び出す．

integer maxVC[*input*] 袖通信バッファ確保用の最大袖層数
integer maxN[*input*] 袖通信バッファ確保用の最大成分数
integer procGrpNo[*input*] バッファを確保するプロセスグループの番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

袖通信 (Scalar3D 版)

```
subroutine cpm_BndCommS3D( array, imax, jmax, kmax, vc, vc_comm, datatype
                         , procGrpNo, ierr )
```

Scalar3D 形状の配列の内部境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う．

void array[*input/output*] 袖通信をする配列の先頭ポインタ
integer imax[*input*] 配列サイズ (I 方向)
integer jmax[*input*] 配列サイズ (J 方向)
integer kmax[*input*] 配列サイズ (K 方向)
integer vc[*input*] 仮想セル数
integer vc_comm[*input*] 袖通信を行う仮想セル数
integer datatype[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer procGrpNo[*input*] プロセスグループ番号
integer ierr[*output*] エラーコード（表 5，6 を参照）

袖通信 (Vector3D 版)

```
subroutine cpm_BndCommV3D( array, imax, jmax, kmax, vc, vc_comm, datatype
                         , procGrpNo, ierr )
```

Vector3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う.（成分数=3）

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

袖通信 (Scalar4D 版)

```
subroutine cpm_BndCommS4D( array, imax, jmax, kmax, nmax, vc, vc_comm
                         , datatype, procGrpNo, ierr )
```

Scalar4D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `nmax`[*input*] 成分数
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

袖通信 (Vector3DEx 版)

```
subroutine cpm_BndCommV3DEx( array, imax, jmax, kmax, nmax, vc, vc_comm
                           , datatype, procGrpNo, ierr )
```

Vector3DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=3）

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

袖通信 (Scalar4DEx 版)

```
subroutine cpm_BndCommS4DEx( array, nmax, imax, jmax, kmax, vc, vc_comm
                           , datatype, procGrpNo, ierr )
```

Scalar4DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `nmax`[*input*] 成分数
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

5.13 内部境界袖通信メソッド（非同期版）

非同期版の内部境界袖通信メソッドは，送信データのパックと非同期送受信処理をするメソッドと，通信完了の待機と受信データの展開をするメソッドに分かれており，非同期通信中に他の計算処理を実行することが可能です.

※ 袖通信メソッドは通信用の送受信バッファを共有しているため，非同期版の袖通信メソッドを使用する場合，その非同期通信中に他の袖通信メソッド（内部境界，周期境界，同期，非同期版の全て）を呼び出すと，通信結果が保証されません.
なお，袖通信以外の通信メソッドは袖通信用の共有バッファを使用しないため，非同期袖通信中であっても使用することができます.

非同期版の袖領域通信メソッドは，次のように定義されています.

非同期袖通信 (Scalar3D 版)

```
subroutine cpm_BndCommS3D_nowait( array, imax, jmax, kmax, vc, vc_comm
                                , datatype, reqlist, procGrpNo, ierr )
```

Scalar3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.
通信完了待ちと受信データの展開は行わない.

void `array`[*input*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*output*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar3D 版) の待機，展開処理

```
subroutine cpm_wait_BndCommS3D( array, imax, jmax, kmax, vc, vc_comm
                              , datatype, reqlist, procGrpNo, ierr )
```

cpm_BndCommS3D_nowait メソッドの通信完了待ちと受信データの展開を行う.

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*input*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3D 版)

```
subroutine cpm_BndCommV3D_nowait( array, imax, jmax, kmax, vc, vc_comm
                                , datatype, reqlist, procGrpNo, ierr )
```

Vector3D 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う.（成分数=3）
通信完了待ちと受信データの展開は行わない.

void `array`[*input*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*output*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3D 版) の待機，展開処理

```
subroutine cpm_wait_BndCommV3D( array, imax, jmax, kmax, vc, vc_comm
                              , datatype, reqlist, procGrpNo, ierr )
```

cpm_BndCommV3D_nowait メソッドの通信完了待ちと受信データの展開を行う．

void `array`[*input/output*]　袖通信をする配列の先頭ポインタ
integer `imax`[*input*]　配列サイズ (I 方向)
integer `jmax`[*input*]　配列サイズ (J 方向)
integer `kmax`[*input*]　配列サイズ (K 方向)
integer `vc`[*input*]　仮想セル数
integer `vc_comm`[*input*]　袖通信を行う仮想セル数
integer `datatype`[*input*]　袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*input*]　リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*]　プロセスグループ番号
integer `ierr`[*output*]　エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4D 版)

```
subroutine cpm_BndCommS4D_nowait( array, imax, jmax, kmax, nmax, vc
                                , vc_comm, datatype, reqlist, procGrpNo
                                , ierr )
```

Scalar4D 形状の配列の内部境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う．（成分数=nmax）
通信完了待ちと受信データの展開は行わない．

void `array`[*input*]　袖通信をする配列の先頭ポインタ
integer `imax`[*input*]　配列サイズ (I 方向)
integer `jmax`[*input*]　配列サイズ (J 方向)
integer `kmax`[*input*]　配列サイズ (K 方向)
integer `nmax`[*input*]　成分数
integer `vc`[*input*]　仮想セル数
integer `vc_comm`[*input*]　袖通信を行う仮想セル数
integer `datatype`[*input*]　袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*output*]　リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*]　プロセスグループ番号
integer `ierr`[*output*]　エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4D 版) の待機，展開処理

```
subroutine cpm_wait_BndCommS4D( array, imax, jmax, kmax, nmax, vc, vc_comm
                              , datatype, reqlist, procGrpNo, ierr )
```

cpm_BndCommS4D_nowait メソッドの通信完了待ちと受信データの展開を行う．

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `nmax`[*input*] 成分数
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*input*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3DEx 版)

```
subroutine cpm_BndCommV3DEx_nowait( array, imax, jmax, kmax, vc, vc_comm
                                  , datatype, reqlist, procGrpNo, ierr )
```

Vector3DEx 形状の配列の内部境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う．（成分数=3）
通信完了待ちと受信データの展開は行わない．

void `array`[*input*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*output*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Vector3DEx 版) の待機，展開処理

```
subroutine cpm_wait_BndCommV3DEx( array, imax, jmax, kmax, vc, vc_comm
                                , datatype, reqlist, procGrpNo, ierr )
```

cpm_BndCommV3DEx_nowait メソッドの通信完了待ちと受信データの展開を行う.

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*input*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4DEx 版)

```
subroutine cpm_BndCommS4DEx_nowait( array, nmax, imax, jmax, kmax, vc
                                  , vc_comm, datatype, reqlist, procGrpNo
                                  , ierr )
```

Scalar4DEx 形状の配列の内部境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う.（成分数=nmax）
通信完了待ちと受信データの展開は行わない.

void `array`[*input*] 袖通信をする配列の先頭ポインタ
integer `nmax`[*input*] 成分数
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*output*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

非同期袖通信 (Scalar4DEx 版) の待機，展開処理

```
subroutine cpm_wait_BndCommS4DEx( array, nmax, imax, jmax, kmax, vc
                                , vc_comm, datatype, reqlist, procGrpNo
                                , ierr )
```

cpm_BndCommS4DEx_nowait メソッドの通信完了待ちと受信データの展開を行う．

void `array`[*input/output*] 袖通信をする配列の先頭ポインタ
integer `nmax`[*input*] 成分数
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `reqlist`[*input*] リクエスト番号配列（サイズ 12，5.2 章を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

5.14 周期境界袖通信メソッド

CPM ライブラリでは，領域分割空間での周期境界（外部境界）の袖領域の通信メソッドを提供しています．袖領域の通信メソッドは，配列形状毎に用意されています．

周期境界の袖通信メソッドは，次のように定義されています．

周期境界袖通信 (Scalar3D 版)

```
subroutine cpm_PeriodicCommS3D( array, imax, jmax, kmax, vc, vc_comm
                              , dir, pm, datatype, procGrpNo, ierr )
```

Scalar3D 形状の配列の周期境界袖通信を行う．
VOXEL 数が [imax,jmax,kmax] の領域に対して，配列形状が array(imax,jmax,kmax) の Fortran 型の配列の袖通信を行う．

void `array`[*input/output*] 周期境界袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `dir`[*input*] 袖通信を行う軸方向フラグ（表 11 を参照）
integer `pm`[*input*] 袖通信を行う正負方向フラグ（表 12 を参照）
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5，6 を参照）

周期境界袖通信 (Vector3D 版)

```
subroutine cpm_PeriodicCommV3D( array, imax, jmax, kmax, vc, vc_comm
                              , dir, pm, datatype, procGrpNo, ierr )
```

Vector3D 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,3) の Fortran 型の配列の袖通信を行う（成分数=3).

void `array`[*input/output*] 周期境界袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `dir`[*input*] 袖通信を行う軸方向フラグ（表 11 を参照）
integer `pm`[*input*] 袖通信を行う正負方向フラグ（表 12 を参照）
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

周期境界袖通信 (Scalar4D 版)

```
subroutine cpm_PeriodicCommS4D( array, imax, jmax, kmax, nmax, vc, vc_comm
                              , dir, pm, datatype, procGrpNo, ierr )
```

Scalar4D 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(imax,jmax,kmax,nmax) の Fortran 型の配列の袖通信を行う（成分数=nmax).

void `array`[*input/output*] 周期境界袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `nmax`[*input*] 成分数
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `dir`[*input*] 袖通信を行う軸方向フラグ（表 11 を参照）
integer `pm`[*input*] 袖通信を行う正負方向フラグ（表 12 を参照）
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

周期境界袖通信 (Vector3DEx 版)

```
subroutine cpm_PeriodicCommV3DEx( array, imax, jmax, kmax, vc, vc_comm
                                , dir, pm, datatype, procGrpNo, ierr )
```

Vector3DEx 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(3,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う（成分数=3).

void `array`[*input/output*] 周期境界袖通信をする配列の先頭ポインタ
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `dir`[*input*] 袖通信を行う軸方向フラグ（表 11 を参照）
integer `pm`[*input*] 袖通信を行う正負方向フラグ（表 12 を参照）
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

周期境界袖通信 (Scalar4DEx 版)

```
subroutine cpm_PeriodicCommS4DEx( array, nmax, imax, jmax, kmax, vc
                                , vc_comm, dir, pm, datatype, procGrpNo
                                , ierr )
```

Scalar4DEx 形状の配列の周期境界袖通信を行う.
VOXEL 数が [imax,jmax,kmax] の領域に対して, 配列形状が array(nmax,imax,jmax,kmax) の Fortran 型の配列の袖通信を行う（成分数=nmax).

void `array`[*input/output*] 周期境界袖通信をする配列の先頭ポインタ
integer `nmax`[*input*] 成分数
integer `imax`[*input*] 配列サイズ (I 方向)
integer `jmax`[*input*] 配列サイズ (J 方向)
integer `kmax`[*input*] 配列サイズ (K 方向)
integer `vc`[*input*] 仮想セル数
integer `vc_comm`[*input*] 袖通信を行う仮想セル数
integer `dir`[*input*] 袖通信を行う軸方向フラグ（表 11 を参照）
integer `pm`[*input*] 袖通信を行う正負方向フラグ（表 12 を参照）
integer `datatype`[*input*] 袖通信をする配列のデータタイプ（表 13 を参照）
integer `procGrpNo`[*input*] プロセスグループ番号
integer `ierr`[*output*] エラーコード（表 5, 6 を参照）

6 API メソッド一覧

以下に，CPM ライブラリが提供する API メソッドの一覧を示します．（表 15，16）

表 15 メソッド一覧その 1（クラス名の無い C++ メソッドは cpm_ParaManager クラスメンバ）

機能	C++ API	Fortan API
初期化関連		
インスタンス生成	get_instance	—
CPM ライブラリ初期化	Initialize	cpm_Initialize
領域分割情報ファイル読み込み	cpm_TextParserDomain::Read	—
プロセスグループ作成	CreateProcessGroup	—
領域分割	VoxelInit	cpm_VoxelInit cpm_VoxelInit_nodiv
情報取得関連		
並列実行チェック	IsParallel	cpm_IsParallel
ランク数取得	GetNumRank	cpm_GetNumRank
ランク番号取得	GetMyRankID	cpm_GetMyRankID
ホスト名取得	GetHostName	—
NULL のランク番号取得	cpm_Base::getRankNull	—
NULL のランク番号チェック	cpm_Base::IsRankNull	—
MPI コミュニケータ取得	GetMPI_Comm	—
NULL の MPI_Comm 取得	cpm_Base::getCommNull	—
NULL の MPI_Comm チェック	cpm_Base::IsCommNull	—
領域分割数取得	GetDivNum	cpm_GetDivNum
自ランク分割位置取得	GetDivPos	cpm_GetDivPos
VOXEL ピッチ取得	GetPitch	cpm_GetPitch
全体 VOXEL 数取得	GetGlobalVoxelSize	cpm_GetGlobalVoxelSize
全体原点座標取得	GetGlobalOrigin	cpm_GetGlobalOrigin
全体空間サイズ取得	GetGlobalRegion	cpm_GetGlobalRegion
ローカル VOXEL 数取得	GetLocalVoxelSize	cpm_GetLocalVoxelSize
ローカル原点座標取得	GetLocalOrigin	cpm_GetLocalOrigin
ローカル空間サイズ取得	GetLocalRegion	cpm_GetLocalRegion
ローカル BCID 取得	GetBCID	cpm_GetBCID
始点 VOXEL インデクス取得	GetVoxelHeadIndex	cpm_GetVoxelHeadIndex
終点 VOXEL インデクス取得	GetVoxelTailIndex	cpm_GetVoxelTailIndex
隣接ランク番号取得	GetNeighborRankID	cpm_GetNeighborRankID
周期境界隣接ランク番号取得	GetPeriodicRankID	cpm_GetPeriodicRankID
ID 範囲全体インデクス取得	GetBndIndexExtGc	—

表 16　メソッド一覧その 2（クラス名の無い C++ メソッドは cpm_ParaManager クラスメンバ）

機能	C++ API	Fortan API
MPI 通信のインターフェイス関連		
MPI_Abort の I/F	Abort	cpm_Abort
MPI_Barrier の I/F	Barrier	cpm_Barrier
MPI_Wait の I/F	Wait	cpm_Wait
MPI_Waitall の I/F	Waitall	cpm_Waitall
MPI_Bcast の I/F	Bcast	cpm_Bcast
MPI_Send の I/F	Send	cpm_Send
MPI_Recv の I/F	Recv	cpm_Recv
MPI_Isend の I/F	Isend	cpm_Isend
MPI_Irecv の I/F	Irecv	cpm_Irecv
MPI_Allreduce の I/F	Allreduce	cpm_Allreduce
MPI_Gather の I/F	Gather	cpm_Gather
MPI_Allgather の I/F	Allgather	cpm_Allgather
MPI_Gatherv の I/F	Gatherv	cpm_Gatherv
MPI_Allgatherv の I/F	Allgatherv	cpm_Allgatherv
袖通信関連		
袖通信バッファのセット	SetBndCommBuffer	cpm_SetBndCommBuffer
袖通信バッファサイズ取得	GetBndCommBufferSize	—
袖通信 (Scalar3D 版)	BndCommS3D	cpm_BndCommS3D
袖通信 (Vector3D 版)	BndCommV3D	cpm_BndCommV3D
袖通信 (Scalar4D 版)	BndCommS4D	cpm_BndCommS4D
袖通信 (Vector3DEx 版)	BndCommV3DEx	cpm_BndCommV3DEx
袖通信 (Scalar4DEx 版)	BndCommS4DEx	cpm_BndCommS4DEx
袖通信（非同期版）関連		
非同期袖通信 (Scalar3D 版)	BndCommS3D_nowait	cpm_BndCommS3D_nowait
非同期袖通信 (Vector3D 版)	BndCommV3D_nowait	cpm_BndCommV3D_nowait
非同期袖通信 (Scalar4D 版)	BndCommS4D_nowait	cpm_BndCommS4D_nowait
非同期袖通信 (Vector3DEx 版)	BndCommV3DEx_nowait	cpm_BndCommV3DEx_nowait
非同期袖通信 (Scalar4DEx 版)	BndCommS4DEx_nowait	cpm_BndCommS4DEx_nowait
非同期袖通信の待機 (Scalar3D 版)	wait_BndCommS3D	cpm_wait_BndCommS3D
非同期袖通信の待機 (Vector3D 版)	wait_BndCommV3D	cpm_wait_BndCommV3D
非同期袖通信の待機 (Scalar4D 版)	wait_BndCommS4D	cpm_wait_BndCommS4D
非同期袖通信の待機 (Vector3DEx 版)	wait_BndCommV3DEx	cpm_wait_BndCommV3DEx
非同期袖通信の待機 (Scalar4DEx 版)	wait_BndCommS4DEx	cpm_wait_BndCommS4DEx
周期境界袖通信関連		
袖通信 (Scalar3D 版)	PeriodicCommS3D	cpm_PeriodicCommS3D
袖通信 (Vector3D 版)	PeriodicCommV3D	cpm_PeriodicCommV3D
袖通信 (Scalar4D 版)	PeriodicCommS4D	cpm_PeriodicCommS4D
袖通信 (Vector3DEx 版)	PeriodicCommV3DEx	cpm_PeriodicCommV3DEx
袖通信 (Scalar4DEx 版)	PeriodicCommS4DEx	cpm_PeriodicCommS4DEx

7 領域分割情報ファイルの仕様

以下に，領域分割情報ファイルの仕様とサンプルを示します.

以下の例では，活性サブドメイン数が 20 ですので，実行時の MPI プロセス数は 20 以上である必要があります.

領域分割情報ファイルの仕様

```
"DomainInfo"
{
  "G_origin" = (1.0   , 2.0   , 3.0   ) //全計算領域の基点（実数、必須）
  "G_region" = (100.0, 80.0 , 50.0 ) //計算領域の大きさ（実数、必須）
  "G_voxel"  = (20    , 24    , 16    ) //格子分割数（整数、オプション）
  "G_pitch"  = (1.0   , 0.8   , 0.5   ) //格子分割幅（実数、オプション）
  "G_div"    = (2     , 3     , 4     ) //全計算領域の分割数（整数、オプション）
  /* G_region の指定があるとき，G_voxel の指定もあれば pitch を自動計算する．
     G_region の指定が無く，G_voxel と G_pitch の両方の指定があるとき，
     region を自動計算する．
   */
}

"SubdomainInfo" //サブドメイン情報
{
  "filename" = "subdomain1.dat"       //サブドメイン情報ファイル名
}
```

8 アップデート情報

本文書のアップデート情報について記します.

Version 1.0.3 2012/7/3

- GetHostName メソッドの追加
  自ランクのホスト名を取得する GetHostName 関数を cpm_ParaManager クラスに追加.

- GetBndIndexExtGc メソッドの追加
  指定 id を含む全体ボクセル空間のインデクス範囲を取得する GetBndIndexExtGc 関数を cpm_ParaManager クラスに追加.

- インデクスと原点座標に関する図追加
  インデクスと原点座標に関する説明図を，図 1 に追加.

- CPM_Op，CPM_Datatype 列挙型に関する補足を追記
  4.4 章に，CPM_Op，CPM_Datatype 列挙型に関する補足を追加.
  - CPM_Datatype，CPM_Op 列挙型
    CPM_Datatype，CPM_Op 列挙型は，cpm_Define.h で定義されていますが，これらの列挙型は Fortran90 インターフェイスメソッド内で使用されるため，ユーザーが C++ コード内で直接使用することはありません.

Version 1.0.2 2012/6/27

- 非同期通信に関する注記の追記
  4.15，5.13 章に非同期通信に関する注記を追加.
  ※ 袖通信メソッドは通信用の送受信バッファを共有しているため，非同期版の袖通信メソッドを使用する場合，その非同期通信中に他の袖通信メソッド（内部境界，周期境界，同期，非同期版の全て）を呼び出すと，通信結果が保証されません．なお，袖通信以外の通信メソッドは袖通信用の共有バッファを使用しないため，非同期袖通信中であっても使用することができます.

- 領域分割情報ファイルフォーマット修正
  以下のように，領域分割情報ファイルのフォーマットを変更.
  - G_org を G_origin に名称変更.
  - G_origin，G_region を必須項目とし，それ以外の項目はオプションとする.
  - サブドメイン情報は別ファイルからの読み込みに変更し，ActiveSubDomains は廃止とする．指定が無い場合は全サブドメインを活性サブドメインとする.

- cpm_ParaManager::VoxelInit の仕様変更
  「領域分割情報ファイルフォーマット修正」に伴い，VoxelInit 関数の仕様を変更.
  - サブドメイン情報の bcid を廃止.
  - VoxelInit メソッドの引数から bcid を削除.
  - G_region を必須とした事から，VoxelInit メソッドの引数 pitch を region に変更.

- 活性サブドメイン情報配列が空のとき，全ランクを活性サブドメインとする．

※ 1.0.2 では全領域に活性サブドメインが存在するものとして扱う．

Version 1.0.1 2012/6/25

- cpm_FaceFlag 列挙型の修正定義の値を変更．
  変更前

```
enum cpm_FaceFlag
{
  X_MINUS = 0  ///< -X face
, Y_MINUS = 1  ///< -Y face
, Z_MINUS = 2  ///< -Z face
, X_PLUS  = 3  ///< +X face
, Y_PLUS  = 4  ///< +Y face
, Z_PLUS  = 5  ///< +Z face
};
```

  変更後

```
enum cpm_FaceFlag
{
  X_MINUS = 0  ///< -X face
, X_PLUS  = 1  ///< +X face
, Y_MINUS = 2  ///< -Y face
, Y_PLUS  = 3  ///< +Y face
, Z_MINUS = 4  ///< -Z face
, Z_PLUS  = 5  ///< +Z face
};
```

Version 1.0.0 2012/6/18

- Version 1.0.0 リリース

表目次

図目次