User Guide of CDMlib

Cartesian Data Management Library

Advanced Institute for Computational Science
RIKEN

http://www.aics.riken.jp/

November 2015

目次

第 1 章

CDMlib の概要

CDMlib の概要と本ユーザガイドについて説明します.

1.1　CDMlib

CDMlib(Cartesian Data Management Library) は直交格子データのファイル入出力管理を行う C++ クラスライブラリです．ユーザーは，C++ で本ライブラリを利用できます．

CDMlib は，以下の機能を有します．

- DFI ファイル（メタ情報）による格子，領域分割情報の管理
- BOV，PLOT3D,（BVX），NetCDF4(/w HDF5)（※）ファイル形式に対応
- MxN ロード対応（並列数が異なる場合のロード処理
- 粗→密ロード対応（各方向の格子数が 1/2（1/8@3 次元）の場合のロード処理）
- ステージング対応（外部プログラムによる，ランク毎のディレクトリへのファイルコピー機能）
- 並列分散ファイルコンバータ対応
  (外部プログラムによる, 並列ファイル変換機能．BOV,PLOT3D,NetCDF4 → BOV,PLOT3D,SPH,AVS,VTK,NetCDF4)（※）

※ NetCDF4(/w HDF5) の入出力を行うためには，NetCDF4 ライブラリと HDF5 ライブラリが別途必要になります．

1.2　この文書について

1.2.1　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．

1.2.2　動作環境

CDM 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 以降
  - MacOS X Snow Leopard
  - Intel C++/Fortran Compiler Version 11 以降 (icpc/ifort)
- 京コンピュータ

第 2 章

パッケージのビルド

この章では，CDMlib のコンパイルについて説明します.

2.1　パッケージのビルド

2.1.1　パッケージの構造

CDM 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)
`CDMlib-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.

CDMlib-*X.X.X*/
- AUTHORS
- BUILD_DIR/
- COPYING
- ChangeLog
- INSTALL
- LISENCE
- Makefile.am
- Makefile.in
- NEWS
- README
- aclocal.m4
- cdm-config.in
- config.h.in
- configure
- compile
- configure.ac
- depcomp
- doc/
  - Makefile.am
  - Makefile.in
  - cdmlib_ug.pdf
  - doxygen/
    - Doxyfile
    - makepdf.sh
  - reference.pdf
- include/
  - inline/
- install-sh
- missing
- src/
- tools/
  - frm/
    - README
    - include/
    - src/
  - fconv/
    - README
    - include/
    - src/

これらのディレクトリ構造は，次の様になっています．

- BUILD_DIR
  パッケージをビルドするディレクトリです．このディレクトリをカレントにして configure，make を実行します．

- doc
  この文書を含む CDMlib 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 libCDM.a が作成され，make install で $prefix/lib にインストールされます．

- tools
  ファイルのランクディレクトリ割り当てを行うユーティリティ，並列分散ファイルコンバータを行うユーティリティが収められています．

2.1.2　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 zxvf CDMlib-X.X.X.tar.gz
   ```

3. CDMlib-X.X.X ディレクトリに移動
   先ほどの解凍で作成された CDMlib-X.X.X ディレクトリ配下の BUILD_DIR ディレクトリに移動します．

   ```
   $ cd CDMlib-X.X.X
   $ cd BUILD\_DIR
   ```

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，2.1.3 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/libCDM.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
   ```

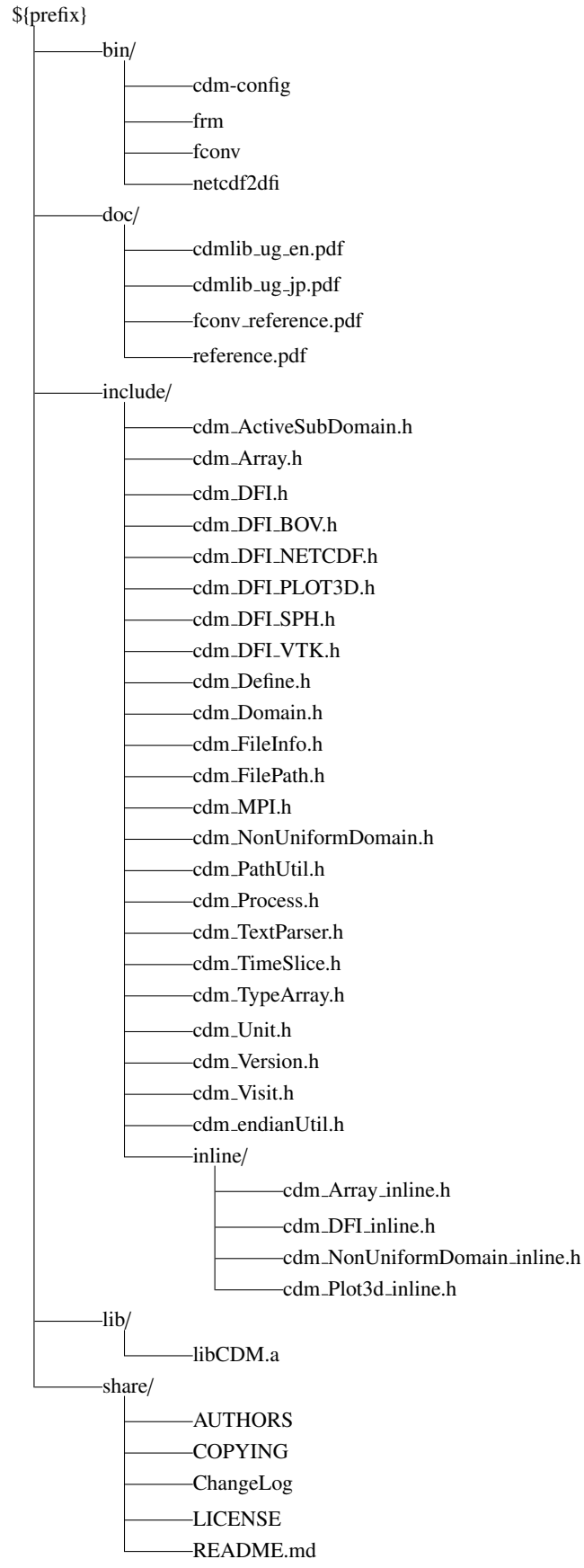
   ただし，インストール先のディレクトリへの書き込みに管理者権限が必要な場合は，sudo コマンドを用いるか，管理者でログインして `make install` を実行します.

   ```
   $ sudo make install
   ```

   または,

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

   インストールされる場所とファイルは以下の通りです.



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.

2.1.3　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/CDMlib の時,

ライブラリ：/usr/local/CDMlib/lib
ヘッダファイル：/usr/local/CDMlib/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/CDMlib 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.

F90

Fortran90 コンパイラのコマンドパスです.

`F90FLAGS`
Fortran90 コンパイラに渡すコンパイルオプションです.

・ ライブラリ指定のオプション
CDM 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．また，並列分散ファイルコンバータをコンパイル，リンク，する場合は CPM ライブラリを指定する必要があります．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-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． OpenMPI で提供されるラッパーコンパイラ（mpicc, mpicxx, mpif90 など）を利用する場合には，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，このオプションは必要ありません.

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

`--with-cpm=`*dir*
CPM ライブラリのインストール先を指定します．CDMlib 自体のコンパイルには CPMlib は利用しませんが，並列分散ファイルコンバータに必要になります．本オプションが指定されない場合，並列分散ファイルコンバータはコンパイル，リンク，インストールされません．京の場合は，並列分散ファイルコンバータをログインノードで–with-MPI=no, –with-frm=yes として，インストールを行います.

`--host=`*hostname*
クロスコンパイル時にアーキテクチャを指定します.

`--with-MPI=`*(yes | no)*
並列動作を指定します．並列時には並列ファイルコンバータがインストールされます.

`--with-frm=`*(no | yes)*
frm ツールのインストールを指定します．ただし，クロスコンパイル時には yes を指定してもインストールできません．ログインノード用に別途コンパイルする必要があります.

`--with-nc=`*dir*
NEtCDF4 ライブラリのインストール先を指定します．このオプションが指定された場合のみ，NetCDF4 ファイルの入出力が可能になります.

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，CDM 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.

2.1.4　configure 実行時オプションの例

・ Linux / MacOS X の場合

CDM ライブラリの prefix：/opt/CDMlib
MPI ライブラリ：OpenMPI，/usr/local/openmpi
TextParser ライブラリ：/usr/local/textparser
CPM ライブラリ：/usr/local/cpmlib
NetCDF4 ライブラリ：/usr/local/netcdf-4.2
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/CDMlib \
              --with-ompi=/usr/local/openmpi \
              --with-parser=/usr/local/textparser \
              --with-cpm=/usr/local/cpmlib \
              --with-nc=/usr/local/netcdf-4.2 \
              CXX=icpc \
              CXXFLAGS=-O3 \
              F90=ifort \
              F90FLAGS=-O3
```

・ 京コンピュータの場合

CDM ライブラリの prefix：/home/userXXXX/CDMlib
TextParser ライブラリ：/home/userXXXX/textparser
CPM ライブラリ：/home/usreXXXX/cpmlib
NetCDF4 ライブラリ：/opt/aics/netcdf/k
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/CDMlib \
              --with-parser=/home/userXXXX/textparser \
              --with-cpm=/home/usreXXXX/cpmlib \
              --with-nc=yes \
              CXX=mpiFCCpx \
              CXXFLAGS=-Kfast \
              F90=mpifrtpx \
              F90FLAGS=-Kfast \
              CPPFLAGS="-I/opt/aics/netcdf/k/include" \
              LDFLAGS="-L/opt/aics/netcdf/k/lib-static" \
              LIBS="-lnetcdf -lhdf5_hl -lhdf5 -lsz -lz"
```

※京コンピュータでは，NetCDF4 ライブラリをスタティックリンクするために，CPPFLAGS,LDFLAGS,LIBS でインクルードファイル，ライブラリのサーチパス，ライブラリの指定を直接記述する必要があります．また，NetCDF4 ライブラリを利用することを示すために，--with-nc オプションに「yes」を指定します．

2.1.5　cdm-config コマンド

CDM ライブラリをインストールすると，$prefix/bin/cdm-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，CDM 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．

cdm-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
CDM 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．

--cflags
C++ コンパイラオプションを取得します．

--libs
CDM 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．

ただし，cdm-config コマンドで取得できるオプションは，CDM 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 cdm-config コマンドの使用方法は，2.2 章を参照してください．

2.1.6　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

```
CDMlib-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)

2.1.7　フロントエンドでステージングツールを使用する場合のビルド方法

京コンピュータ等のクロスコンパイル環境でステージングツールを使用する場合，フロントエンド用のネイティブコンパイラを用いて CDM ライブラリをビルドする必要があります．
また，フロントエンドに MPI ライブラリがインストールされていない場合，CDM ライブラリの configure スクリプト

実行時に MPI ライブラリを未実行とするオプションとステージングツールのインストールオプション「–with-MPI=no, –with-frm=yes」を付けてビルドする必要があります.

・京コンピュータフロントエンド用の configure 実行例

```
$ ./configure --prefix=/home/userXXXX/CDMlib_frontend \
              --with-MPI=no \
              --with-frm=yes \
              --with-parser=/home/userXXXX/textparser \
              CXX=g++ \
              F90=gfortran
```

なお，この場合リンクする textparser もフロントエンドのネイティブコンパイラでビルドしておく必要があります.

2.1.8　NetCDF4(/w HDF5) の利用について

CDMlib で NetCDF4(/w HDF5) フォーマットを利用するためには，CDMlib ビルド時に NEtCDF4 ライブラリを指定する必要があります．また，指定する NetCDF4 ライブラリは HDF5 が有効である必要があります.

NetCDF4 の nc-config コマンドが利用可能な場合は，以下の方法で確認することができます.

```
$ nc-config --has-nc4
yes
```

このコマンドの戻り値が「no」の場合，CDMlib では利用できません.

NetCDF4 ライブラリは configure コマンドの実行時に–with-nc パラメータにより指定します.

・nc-config コマンドが利用可能な場合（通常）
CDMlib の configure 実行時に–with-nc オプションで NetCDF4 ライブラリのインストールパスを指定します．make 時に NetCDF4 ライブラリ内の nc-config コマンドを自動的に実行し，NetCDF4 ライブラリのリンクに必要なコンパイル/リンクオプションを付与します.

・京コンピュータの場合
京コンピュータでは，NetCDF4 ライブラリをスタティックにリンクするために，コンパイル/リンクオプションを直接指定する必要があります．また，NetCDF4 を利用することを指定するために，–with-nc オプションに「yes」を指定します.

CDM **ライブラリの** prefix：/home/userXXXX/CDMlib
TextParser **ライブラリ**：/home/userXXXX/textparser
CPM **ライブラリ**：/home/usreXXXX/cpmlib
NetCDF4 **ライブラリ**：/opt/aics/netcdf/k
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/CDMlib \
              --with-parser=/home/userXXXX/textparser \
```

```
                --with-cpm=/home/usreXXXX/cpmlib \
                --with-nc=yes \
                CXX=mpiFCCpx \
                CXXFLAGS=-Kfast \
                F90=mpifrtpx \
                F90FLAGS=-Kfast \
                CPPFLAGS="-I/opt/aics/netcdf/k/include" \
                LDFLAGS="-L/opt/aics/netcdf/k/lib-static" \
                LIBS="-lnetcdf -lhdf5_hl -lhdf5 -lsz -lz"
```

・Modules 環境で NetCDF4，HDF5 を利用する環境の場合
module コマンドを利用して NetCDF4，HDF5 ライブラリを有効にする環境では，CDMlib の configure，make 実行前にこれらのライブラリを有効にしておく必要があります.

（例）

```
$ module load HDF5
$ module load netCDF
```

また，ライブラリの参照順の関係で，configure 実行時にリンクするライブラリを直接指定する必要がある場合があります．その場合，次のように configure コマンドを実行します.

```
$ module load HDF5
$ module load netCDF
$ ./configure --host=sparc64-unknown-linux-gnu \
              --prefix=/home/userXXXX/CDMlib \
              --with-parser=/home/userXXXX/textparser \
              --with-cpm=/home/usreXXXX/cpmlib \
              --with-nc=yes \
              CXX=mpiFCCpx \
              CXXFLAGS=-Kfast \
              F90=mpifrtpx \
              F90FLAGS=-Kfast \
              LIBS="-lnetcdf -lhdf5_hl -lhdf5"
```

2.2　CDM ライブラリの利用方法

ユーザーが作成する CDM 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.

2.2.1　C++

CDM 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/CDMlib/bin/cdm-config --cflags‘ \
  ‘/usr/local/CDMlib/bin/cdm-config --libs‘
```

第 3 章

API 利用方法

この章では，CDMlib の API の利用方法について説明します.

3.1　ユーザープログラムでの利用方法

以下に，CDM ライブラリの C++ API の説明を示します．

3.1.1　cdm_DFI.h のインクルード

CDM ライブラリの C++ API 関数群は，CDM ライブラリが提供するヘッダファイル cdm_DFI.h で定義されています．CDM 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．

cdm_DFI.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 cdm_DFI 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．

cdm_DFI.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．

3.1.2　マクロ，列挙型，エラーコード

CDM 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，cdm_Define.h に定義されています．

・ D_CDM_XXXX マクロ

表 3.1　D_CDM_XXXX マクロ

マクロ名	内容	マクロ名	内容	マクロ名	内容
D_CDM_EXT_SPH	”sph”	D_CDM_LITTLE	”little”	D_CDM_UINT8	”UInt8”
D_CDM_EXT_BOV	”bov”	D_CDM_BIG	”big”	D_CDM_UINT16	”UInt16”
D_CDM_EXT_BOV_DATAFILE	”dat”	D_CDM_INT8	”Int8”	D_CDM_UINT32	”UInt32”
D_CDM_EXT_FUNC	”fun”	D_CDM_INT16	”Int16”	D_CDM_FLOAT32	”Float32”
D_CDM_ON	”on”	D_CDM_INT32	”Int32”	D_CDM_FLOAT64	”Float64”
D_CDM_OFF	”off”	D_CDM_INT64	”Int64”	D_CDM_EXT_NC	”nc”

・ E_CDM_ONOFF 列挙型

E_CDM_ONOFF 列挙型は，cdm_Define.h で表 3.2 のように定義されています．

フィールドデータを時刻毎にディレクトリを作成して出力するかなどの、オン，オフを判断する際に，使われます．

表 3.2　E_CDM_ONOFF 列挙型

E_CDM_ONOFF 要素	値	意味
E_CDM_OFF	0	スイッチオフ
E_CDM_ON	1	スイッチオン

・ E_CDM_FORMAT 列挙型

E_CDM_FORMAT 列挙型は，cdm_Define.h で表 3.3 のように定義されています．

フィールドデータのファイルフォーマットを指定するフラグとして使われます．

表 3.3　E_CDM_FORMAT 列挙型

E_CDM_FORMAT 要素	値	意味
E_CDM_UNKNOWN	-1	未定義
E_CDM_SPH	0	SPH 形式
E_CDM_BOV	1	BOV 形式
E_CDM_AVS	2	AVS 形式
E_CDM_PLOT3D	3	PLOT3D 形式
E_CDM_VTK	4	VTK 形式
E_CDM_NETCDF4	5	NetCDF(/w HDF5) 形式

・ E_CDM_DTYPE 列挙型

E_CDM_DTYPE 列挙型は，cdm_Define.h で表 3.4 のように定義されています．
フィールドデータのデータ形式を指定するフラグとして使われます．

表 3.4　E_CDM_DTYPE 列挙型

E_CDM_DTYPE 要素	値	意味
E_CDM_DTYPE_UNKNOWN	0	未定義
E_CDM_INT8	1	char
E_CDM_INT16	2	short
E_CDM_INT32	3	int
E_CDM_INT64	4	long long
E_CDM_UINT8	5	unsigned char
E_CDM_UINT16	6	unsigned short
E_CDM_UINT32	7	unsigned int
E_CDM_UINT64	8	unsigned long long
E_CDM_FLOAT32	9	float
E_CDM_FLOAT64	10	double

・ E_CDM_DFITYPE 列挙型

E_CDM_DFITYPE 列挙型は，cdm_Define.h で表 3.5 のように定義されています．
フィールドデータの格子タイプを指定するフラグとして使われます．

表 3.5　E_CDM_DFITYPE 列挙型

E_CDM_DFITYPE 要素	値	意味
E_CDM_DFITYPE_UNKNOWN	-1	未定義
E_CDM_DFITYPE_CARTESIAN	0	直交等間隔格子
E_CDM_DFITYPE_NON_UNIFORM_CARTESIAN	1	直交不等間隔格子

・ E_CDM_ENDIANTYPE 列挙型

E_CDM_ENDIANTYPE 列挙型は，cdm_Define.h で表 3.6 のように定義されています．
フィールドデータのエンディアン形式を指定するフラグとして使われます．

表 3.6　E_CDM_ENDIANTYPE 列挙型

E_CDM_ENDIANTYPE 要素	値	意味
E_CDM_ENDIANTYPE_UNKNOWN	-1	未定義
E_CDM_LITTELE	0	リトルエンディアン形式
E_CDM_BIG	1	ビッグエンディアン形式

・E_CDM_READTYPE 列挙型

E_CDM_READTYPE 列挙型は，cdm_Define.h で表 3.7 のように定義されています．
リスタート時のフィールドデータの読込み形式を指定するフラグとして使われます．

表 3.7　E_CDM_READTYPE 列挙型

E_CDM_READTYPE 要素	値	意味
E_CDM_SAMEDIV_SAMERES	1	同一分割，同一密度
E_CDM_SAMEDIV_REFINEMENT	2	同一分割，粗密
E_CDM_DIFFDIV_SAMERES	3	MxN，同一密度
E_CDM_DIFFDIV_REFINEMENT	4	MxN，粗密
E_CDM_READTYPE_UNKNOWN	5	エラー

・E_CDM_FILE_TYPE 列挙型

E_CDM_FILE_TYPE 列挙型は，cdm_Define.h で表 3.8 のように定義されています．
フィールドデータのファイルタイプを指定するフラグとして使われます．

表 3.8　E_CDM_FILE_TYPE 列挙型

E_CDM_FILE_TYPE 要素	値	意味
E_CDM_FILE_TYPE_DEFAULT	-1	デフォルト (binary)
E_CDM_FILE_TYPE_ASCII	0	ascii 形式
E_CDM_FILE_TYPE_BINARY	1	binary 形式
E_CDM_FILE_TYPE_FBINARY	2	Fortran Binary 形式

・E_CDM_OUTPUT_FNAME 列挙型

E_CDM_OUTPUT_FNAME 列挙型は，cdm_Define.h で表 3.9 のように定義され¿ています．
フィールドデータの出力ファイル名の命名順を指定するフラグとして使われます．

表 3.9　E_CDM_OUTPUT_FNAME 列挙型

E_CDM_OUTPUT_FNAME 要素	値	意味
E_CDM_OUTPUT_FNAME_DEFAULT	-1	デフォルト (step_rank)
E_CDM_OUTPUT_FNAME_STEP_RANK	0	step_rank
E_CDM_OUTPUT_FNAME_RANK_STEP	1	rank_step
E_CDM_OUTPUT_FNAME_RANK	2	rank(NetCDF のみ)

・E_CDM_ERRORCODE 列挙型

E_CDM_ERRORCODE 列挙型は，cdm_Define.h で表 3.10，3.11 のように定義されています．

CDM 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．

表 3.10　E_CDM_ERRORCODE 列挙型 その 1

E_CDM_ERRORCODE 要素	値	意味
E_CDM_SUCCESS	0	正常終了
E_CDM_ERROR	-1	その他のエラー
E_CDM_ERROR_READ_DFI_GLOBALORIGIN	1000	DFI GlobalOrigin 読込みエラー
E_CDM_ERROR_READ_DFI_GLOBALREGION	1001	DFI GlobalRegion 読込みエラー
E_CDM_ERROR_READ_DFI_GLOBALVOXEL	1002	DFI GlobalVoxel 読込みエラー
E_CDM_ERROR_READ_DFI_GLOBALDIVISION	1003	DFI GlobalDivison 読込みエラー
E_CDM_ERROR_READ_DFI_DIRECTORYPATH	1004	DFI DirectoryPath 読込みエラー
E_CDM_ERROR_READ_DFI_TIMESLICEDIRECTORY	1005	DFI TimeSliceDirectoryPath 読込みエラー
E_CDM_ERROR_READ_DFI_PREFIX	1006	DFI Prefix 読込みエラー
E_CDM_ERROR_READ_DFI_FILEFORMAT	1007	DFI FileFormat 読込みエラー
E_CDM_ERROR_READ_DFI_GUIDECELL	1008	DFI GuideCell 読込みエラー
E_CDM_ERROR_READ_DFI_DATATYPE	1009	DFI DataType 読込みエラー
E_CDM_ERROR_READ_DFI_ENDIAN	1010	DFI Endian 読込みエラー
E_CDM_ERROR_READ_DFI_NUMVARIABLES	1012	DFI NumVariables 読込みエラー
E_CDM_ERROR_READ_DFI_FILEPATH_PROCESS	1013	DFI FilePath/Process 読込みエラー
E_CDM_ERROR_READ_DFI_NO_RANK	1014	DFI Rank 要素なし
E_CDM_ERROR_READ_DFI_ID	1015	DFI ID 読込みエラー
E_CDM_ERROR_READ_DFI_HOSTNAME	1016	DFI HoatName 読込みエラー
E_CDM_ERROR_READ_DFI_VOXELSIZE	1017	DFI VoxelSize 読込みエラー
E_CDM_ERROR_READ_DFI_HEADINDEX	1018	DFI HeadIndex 読込みエラー
E_CDM_ERROR_READ_DFI_TAILINDEX	1019	DFI TailIndex 読込みエラー
E_CDM_ERROR_READ_DFI_NO_SLICE	1020	DFI TimeSlice 要素なし
E_CDM_ERROR_READ_DFI_STEP	1021	DFI Step 読込みエラー
E_CDM_ERROR_READ_DFI_TIME	1022	DFI Time 読込みエラー
E_CDM_ERROR_READ_DFI_NO_MINMAX	1023	DFI MinMax 要素なし
E_CDM_ERROR_READ_DFI_MIN	1024	DFI Min 読込みエラー
E_CDM_ERROR_READ_DFI_MAX	1025	DFI Max 読込みエラー
E_CDM_ERROR_READ_DFI_DFITYPE	1026	DFI DFIType 読込みエラー
E_CDM_ERROR_READ_DFI_FIELDFILENAMEFORMAT	1027	DFI FieldFilenameFormat 読込みエラー
E_CDM_ERROR_READ_DFI_COORDINATEFILE	1028	DFI Coordinate File 読込みエラー
E_CDM_ERROR_READ_DFI_COORDINATEFILETYPE	1029	DFI Coordinate File Type 読込みエラー
E_CDM_ERROR_READ_DFI_COORDINATEFILEPRECISION	1030	DFI Coordinate File Precision 読込みエラー
E_CDM_ERROR_READ_DFI_COORDINATEFILEENDIAN	1031	DFI Coordinate File Endian 読込みエラー
E_CDM_ERROR_OPEN_COORDINATEFILE	1032	Coordinate File オープンに失敗
E_CDM_ERROR_READ_COORDINATEFILE	1033	Coordinate File 読込みエラー
E_CDM_ERROR_READ_INDEXFILE_OPENERROR	1050	Index ファイルオープンエラー
E_CDM_ERROR_TEXTPARSER	1051	TextParser エラー
E_CDM_ERROR_READ_FILEINFO	1052	FileInfo 読込みエラー
E_CDM_ERROR_READ_FILEPATH	1053	FilePath 読込みエラー
E_CDM_ERROR_READ_UNIT	1054	UNIT 読込みエラー
E_CDM_ERROR_READ_TIMESLICE	1055	TimeSlice 読込みエラー
E_CDM_ERROR_READ_PROCFILE_OPENERROR	1056	Proc ファイルオープンエラー
E_CDM_ERROR_READ_DOMAIN	1057	Domain 読込みエラー
E_CDM_ERROR_READ_MPI	1058	MPI 読込みエラー
E_CDM_ERROR_READ_PROCESS	1059	Process 読込みエラー
E_CDM_ERROR_READ_DFI_NETCDF	1060	NetCDF 読込みエラー
E_CDM_ERROR_READ_NETCDF_MISMATCH_TYPE	1061	DFI と NetCDF のデータ型の不一致エラー
E_CDM_ERROR_READ_FIELDDATA_FILE	1900	フィールドデータファイル読込みエラー
E_CDM_ERROR_READ_SPH_FILE	2000	SPH ファイル読込みエラー
E_CDM_ERROR_READ_SPH_REC1	2001	SPH ファイルレコード 1 読込みエラー
E_CDM_ERROR_READ_SPH_REC2	2002	SPH ファイルレコード 2 読込みエラー

表 3.11　E_CDM_ERRORCODE 列挙型 その 2

E_CDM_ERRORCODE 要素	値	意味
E_CDM_ERROR_READ_SPH_REC3	2003	SPH ファイルレコード 3 読込みエラー
E_CDM_ERROR_READ_SPH_REC4	2004	SPH ファイルレコード 4 読込みエラー
E_CDM_ERROR_READ_SPH_REC5	2005	SPH ファイルレコード 5 読込みエラー
E_CDM_ERROR_READ_SPH_REC6	2006	SPH ファイルレコード 6 読込みエラー
E_CDM_ERROR_READ_SPH_REC7	2007	SPH ファイルレコード 7 読込みエラー
E_CDM_ERROR_UNMATCH_VOXELSIZE	2050	SPH のボクセルサイズと DFI のボクセルサイズが合致しない
E_CDM_ERROR_NOMATCH_ENDIAN	2051	出力 Fornat が合致しない（Endian 形式が Big,Little 以外）
E_CDM_ERROR_UNMATCH_NUM_OF_VARIABLES	2052	フィールドデータの変数の個数と登録された変数名の個数が合致しない
E_CDM_ERROR_READ_BOV_FILE	2100	BOV ファイル読込みエラー
E_CDM_ERROR_READ_FIELD_HEADER_RECORD	2102	フィールドデータのヘッダーレコード読込み失敗
E_CDM_ERROR_READ_FIELD_DATA_RECORD	2103	フィールドデータのデータレコード読込み失敗
E_CDM_ERROR_READ_FIELD_AVERAGED_RECORD	2104	フィールドデータの Averaged レコード読込み失敗
E_CDM_ERROR_READ_NETCDF_FUNC	2200	NetCDF の nc 関数でエラー
E_CDM_ERROR_READ_NETCDF_VAR_1D	2201	NetCDF の 1 次元配列として読み込む variable が 1 次元で無い
E_CDM_ERROR_MISMATCH_NP_SUBDOMAIN	3003	並列数とサブドメイン数が一致していない
E_CDM_ERROR_INVALID_DIVNUM	3011	領域分割数が不正
E_CDM_ERROR_OPEN_SBDM	3012	ActiveSubdomain ファイルのオープンに失敗
E_CDM_ERROR_READ_SBDM_HEADER	3013	ActiveSubdomain ファイルのヘッダー読み込みに失敗
E_CDM_ERROR_READ_SBDM_FORMAT	3014	ActiveSubdomain ファイルのフォーマットエラー
E_CDM_ERROR_READ_SBDM_DIV	3015	ActiveSubdomain ファイルの領域分割数読み込みに失敗
E_CDM_ERROR_READ_SBDM_CONTENTS	3016	ActiveSubdomain ファイルの Contents 読み込みに失敗
E_CDM_ERROR_SBDM_NUMDOMAIN_ZERO	3017	ActiveSubdomain ファイルの活性ドメイン数が 0
E_CDM_ERROR_MAKEDIRECTORY	3100	Directory 生成で失敗
E_CDM_ERROR_OPEN_FIELDDATA	3101	フィールドデータのオープンに失敗
E_CDM_ERROR_WRITE_FIELD_HEADER_RECORD	3102	フィールドデータのヘッダーレコード出力失敗
E_CDM_ERROR_WRITE_FIELD_DATA_RECORD	3103	フィールドデータのデータレコード出力失敗
E_CDM_ERROR_WRITE_FIELD_AVERAGED_RECORD	3104	フィールドデータの Average レコード出力失敗
E_CDM_ERROR_WRITE_SPH_REC1	3201	SPH ファイルレコード 1 出力エラー
E_CDM_ERROR_WRITE_SPH_REC2	3202	SPH ファイルレコード 2 出力エラー
E_CDM_ERROR_WRITE_SPH_REC3	3203	SPH ファイルレコード 3 出力エラー
E_CDM_ERROR_WRITE_SPH_REC4	3204	SPH ファイルレコード 4 出力エラー
E_CDM_ERROR_WRITE_SPH_REC5	3205	SPH ファイルレコード 5 出力エラー
E_CDM_ERROR_WRITE_SPH_REC6	3206	SPH ファイルレコード 6 出力エラー
E_CDM_ERROR_WRITE_SPH_REC7	3207	SPH ファイルレコード 7 出力エラー
E_CDM_ERROR_WRITE_PROCFILENAME_EMPTY	3500	proc dfi ファイル名が未定義
E_CDM_ERROR_WRITE_PROCFILE_OPENERROR	3501	proc dfi ファイルオープン失敗
E_CDM_ERROR_WRITE_DOMAIN	3502	Domain 出力失敗
E_CDM_ERROR_WRITE_MPI	3503	MPI 出力失敗
E_CDM_ERROR_WRITE_PROCESS	3504	Process 出力失敗
E_CDM_ERROR_WRITE_RANKID	3505	出力ランク以外
E_CDM_ERROR_WRITE_INDEXFILENAME_EMPTY	3510	index dfi ファイル名が未定義
E_CDM_ERROR_WRITE_PREFIX_EMPTY	3511	Prefix が未定義
E_CDM_ERROR_WRITE_INDEXFILE_OPENERROR	3512	proc dfi ファイルオープン失敗
E_CDM_ERROR_WRITE_FILEINFO	3513	FileInfo 出力失敗
E_CDM_ERROR_WRITE_UNIT	3514	Unit 出力失敗
E_CDM_ERROR_WRITE_TIMESLICE	3515	TimeSlice 出力失敗
E_CDM_ERROR_WRITE_FILEPATH	3516	FilePath 出力失敗
E_CDM_ERROR_WRITE_DFI_NETCDF	3600	NetCDF の DFI 出力エラー
E_CDM_WARN_GETUNIT	4000	Unit の単位がない

3.2　入力機能

3.2.1　機能概要

CDM ライブラリでは，下図 (図 3.1, 図 3.2, 図 3.3, 図 3.4) に示すようフィールドデータファイルの読み込み機能として、1 対 1 データの読込み，MxN データの読込み，リファインメントデータ（粗い格子で計算した結果を 1 段階細かい格子（１：２）にマッピング）の読込みの 4 種類をサポートしています．CDM ではこれらを自動的に把握して，読込み処理を行います．

- 同一格子密度での 1 対 1 の読込み
  空間全体の格子数が一致しており，かつ，領域分割位置が一致している場合，各プロセスは対応する１つのフィールドデータを読込みます．

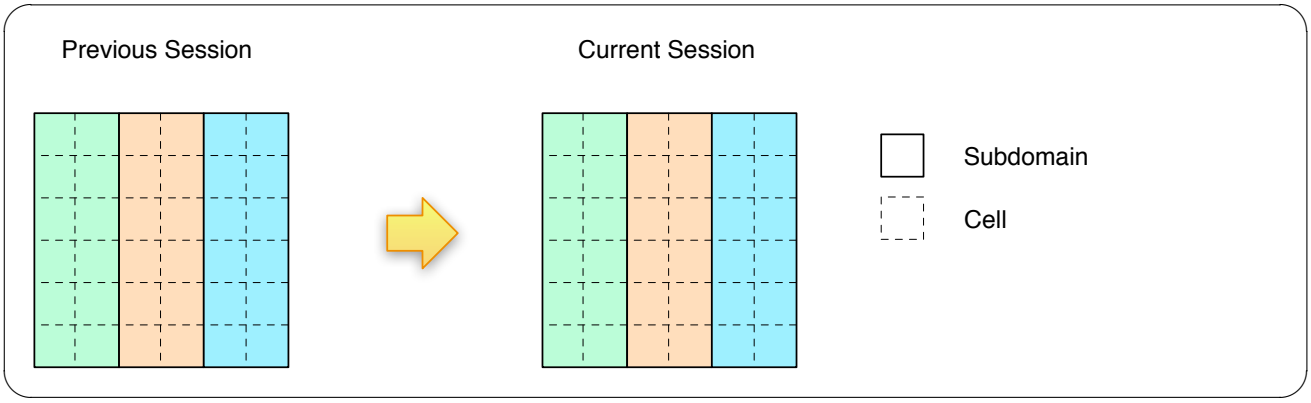


図 3.1　同一格子密度での 1 対 1 読込み

- 同一格子密度での M 対 N の読込み
  空間全体の格子数は一致しているが，領域分割数または領域分割位置が一致していない場合，１つのプロセスが対応する１～複数のフィールドデータを読込みます．

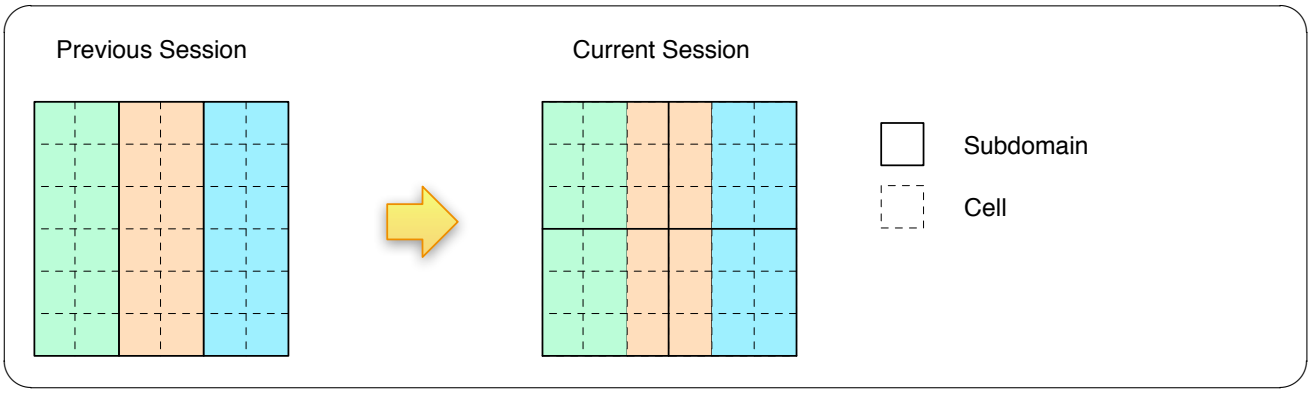


図 3.2　同一格子密度での M 対 N 読込み

- リファインメントデータで 1 対 1 の読込み
  格子が 1 段階細かい格子（1 ： 2）で，読込みフィールドデータが 1 対 1 に対応している場合，各プロセスは対応する 1 つのフィールドデータを読込み，補間処理 (※ 1) をします.

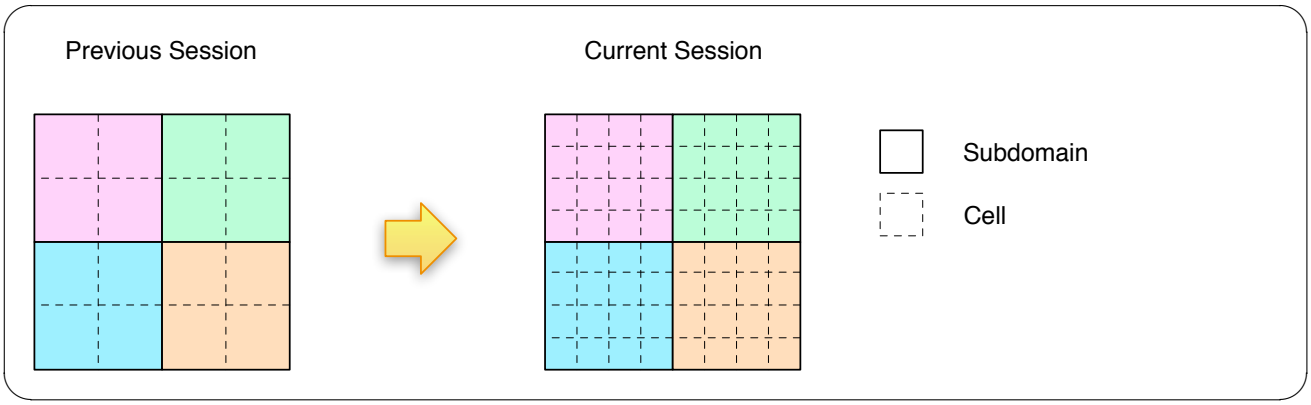


図 3.3　リファインメントで 1 対 1 読込み

- リファインメントデータで M 対 N の読込み
  格子が 1 段階細かい格子（1 ： 2）で，領域分割数が一致していない場合（フィールドデータが 1 対 1 に対応していない），1 つのプロセスが対応する 1 ～複数のフィールドデータを読込み，補間処理 (※ 1) をします.

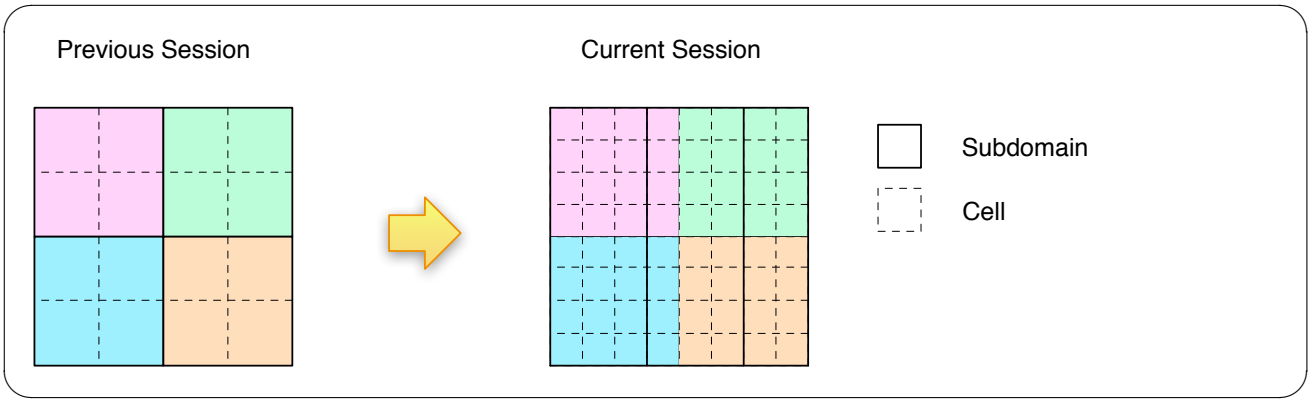


図 3.4　リファインメントで M 対 N 読込み

(※ 1) リファインメントデータの補間処理については 3.2.6 章を参照
(※ 2) リファインメントデータの読込み，補間処理は実数型（単精度/倍精度）のみを対象としています.

3.2.2 入力処理手順

CDM では以下の手順で，フィールドデータ及び DFI データの入力処理を行います．

1. 読込み用インスタンスのポインタ取得 (3.2.2 章参照)
2. 読込んだ DFI ファイルからの情報取得 (3.2.3 章参照)
3. フィールドデータの読込み (3.2.5 章参照)

(注) 読込み用インスタンスポインタは不要になったら，必ずユーザで削除を行う必要があります．

cdm_DFI クラスのインスタンスは，DFI ファイルの種類毎にいくつでも生成可能です．そのインスタンスへのポインタを取得するメソッドは，cdm_DFI.h 内で次のように定義されています．

読込み用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 cdm_DFI* cdm_DFI::ReadInit(const MPI_Comm comm,
                                  const std::string dfifile,
                                  const int G_Voxel[3],
                                  const int G_Div[3],
                                  CDM::E_CDM_ERRORCODE &ret);
```

cdm_DFI クラスのインスタンスへのポインタを取得します．

comm	[*input*]	MPI コミュニケータ
dfifile	[*input*]	index.dfi ファイル名
G_Voxel	[*input*]	X,Y,Z 方向の計算空間全体のボクセルサイズ（3word の配列）
G_Div	[*input*]	X,Y,Z 方向の領域分割数（3word の配列）
ret	[*output*]	エラーコード（表 3.10，3.11 を参照）
戻り値	cdm_DFI クラスのインスタンスへのポインタ	

(注) インスタンスされたポインタは，不要になった時にユーザが delete する必要があります．

```
//DFI のインスタンス
cdmDFI *DFI_IN_PRS = cdm_DFI::ReadInit(,,,);
                :
            (処理)
                :
//不要になったので delete
delete DFI_IN_PRS;
```

3.2.3　DFI 情報の取得

読込んだ DFI の情報を取得するためには CDM のメソッドを使用します．以下に DFI 情報取得するメソッドを説明します．

DFI 情報の取得処理を行うメソッドは cdm_DFI.h 内で次のように定義されています．

1. フィールドデータのデータ型の取得
   フィールドデータのデータ型は文字列 (表 3.1 参照) と列挙型 (表 3.4 参照) それぞれで取得するメソッドが定義されています．

フィールドデータのデータ型の取得（文字列）

```
std::string
cdm_DFI::GetDataTypeString();
```

戻り値　フィールドデータのデータ型，文字列 (表 3.1 参照)

フィールドデータのデータ型の取得（列挙型）

```
CDM::E_CDM_DTYPE
cdm_DFI::GetDataType();
```

戻り値　フィールドデータのデータ型，列挙型 (表 3.4 参照)

2. フィールドデータの変数の個数の取得

フィールドデータの変数の個数の取得

```
int
cdm_DFI::GetNumVariables();
```

戻り値　フィールドデータの変数の個数

3. データ型の変換 (文字列から列挙型)
   フィールドデータのデータ型を文字列から列挙型 (表 3.4 参照) に変換します．

データ型の変換 (文字列から列挙型)

```
static CDM::E_CDM_DTYPE
cdm_DFI::ConvDatatypeS2E(const std::string datatype);
```

datatype	[*input*]	DFI ファイルから取得したデータ型　表 3.4 参照
戻り値		変換された列挙型のデータ型 (表 3.4 参照)

4. データ型の変換 (列挙型から文字列)
   フィールドデータのデータ型を列挙型から文字列 (表 3.1 参照) に変換します．

データ型の変換 (列挙型から文字列)

```
static std::string
cdm_DFI::ConvDatatypeE2S(const CDM::E_CDM_DTYPE Dtype);
```

`Dtype`　[*input*]　DFI ファイルから取得したデータ型　表 3.4 参照
戻り値　変換された文字列のデータ型 (表 3.1 参照)

5. DFI Domain の GlobalVoxel(計算領域全体のボクセル数) の取得

   DFI ファイルの Domain の仕様は 7.1.2 章「プロセス情報ファイル (proc.dfi) 仕様」参照.

DFI Domain の GlobalVoxel の取得

```
const int*
cdm_DFI::GetDFIGlobalVoxel();
```

戻り値　GlobalVoxel のポインタ

6. DFI Domain の GlobalDivion(計算領域の分割数) の取得

   DFI ファイルの Domain の仕様は 7.1.2 章「プロセス情報ファイル (proc.dfi) 仕様」参照.

DFI Domain の GlobalDivision の取得

```
const int*
cdm_DFI::GetDFIGlobalDivision();
```

戻り値　GlobalDivision のポインタ

7. DFI FileInfo の変数名を取得

   DFI ファイルの FileInfo の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

DFI FileInfo の変数名を取得

```
std::string
cdm_DFI::getVariableName(int pvari);
```

`pvari`　[*input*]　変数位置　0:u 1:v 2:w
戻り値　変数名

8. DFI TimeSlice の minmax 値を取得

   DFI ファイルの TimeSlice の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

DFI TimeSlice の minmax 値を取得

```
CDM::E_CDM_ERRORCODE
cdm_DFI::getMinMax(const unsigned step,
                   const int variNo,
                   double &min_value,
                   double &max_value);
```

step	[*input*]	対象となるステップ番号
variNo	[*input*]	対象となる変数番号 (0〜n)
min_value	[*output*]	min
max_value	[*output*]	max
戻り値		エラーコード（表 3.10，3.11 を参照）

9. DFI UnitList から単位系を取得する

   DFI ファイルの UnitList の単位系の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

UnitList から単位系を取得

```
CDM::E_CDM_ERRORCODE
cdm_DFI::GetUnitElem(const std::string Name,
                     cdm_UnitElem &unit);
```

Name	[*input*]	取得する単位系
unit	[*output*]	取得した単位系
戻り値		エラーコード（表 3.10，3.11 を参照）

10. DFI UnitList にセットされている各値を取得する

    DFI ファイルの UnitList の単位系の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

UnitList にセットされている各値を取得

```
CDM::E_CDM_ERRORCODE
cdm_DFI::GetUnit(const std::string Name,
                 std::string &unit,
                 double &ref,
                 double &diff,
                 bool &bSetDiff);
```

Name	[*input*]	取得する単位系
unit	[*output*]	取得した単位文字列
ret	[*output*]	取得した reference 値
diff	[*output*]	取得した difference 値
bSetDiff	[*output*]	取得した difference の有無フラグ
戻り値		エラーコード（表 3.10，3.11 を参照）

3.2.4　DFI クラスポインタの取得

読込んだ DFI の情報をセットした各クラスのポインタを取得するためには CDM のメソッドを使用します．以下に各クラスのポインタを取得するメソッドを説明します.

DFI 情報をセットした各クラスのポインタ取得処理を行うメソッドは cdm_DFI.h 内で次のように定義されています.

1. cdm_FileInfo クラスポインタの取得

cdm_FileInfo クラスポインタの取得

```
const cdm_FileInfo* GetcdmFileInfo();
```

戻り値 FileInfo の情報がセットされたクラスのポインタ

2. cdm_FilePath クラスポインタの取得

cdm_FilePath クラスポインタの取得

```
const cdm_FilePath* GetcdmFilePath();
```

戻り値 FilePath の情報がセットされたクラスのポインタ

3. cdm_Unit クラスポインタの取得

cdm_Unit クラスポインタの取得

```
const cdm_Unit* GetcdmUnit();
```

戻り値 Unit の情報がセットされたクラスのポインタ

4. cdm_Domain クラスポインタの取得

cdm_Domain クラスポインタの取得

```
const cdm_Domain* GetcdmDomain();
```

戻り値 Domain の情報がセットされたクラスのポインタ

5. cdm_MPI クラスポインタの取得

cdm_MPI クラスポインタの取得

```
const cdm_MPI* GetcdmMPI();
```

戻り値 MPI の情報がセットされたクラスのポインタ

6. cdm_TimeSlice クラスポインタの取得

cdm_TimeSlice クラスポインタの取得

```
const cdm_TimeSlice* GetcdmTimeSlice();
```

戻り値 TimeSlice の情報がセットされたクラスのポインタ

7. cdm_Process クラスポインタの取得

cdm_Process クラスポインタの取得

```
const cdm_Process* GetcdmProcess();
```

戻り値 Process の情報がセットされたクラスのポインタ

3.2.5　フィールドデータファイルの読み込み

フィールドデータファイルの形式は，SPH 形式，BOV 形式，PLOT3D 形式，NetCDF4(/w HDF5) 形式ファイルです．（詳細は，7.1.3 章を参照してください）

フィールドデータファイルの読み込み処理は，読込んだデータのポインタを戻すメソッドとユーザが指定した配列ポインタにデータを読込むメソッドの２つが cdm_DFI.h 内で次のように定義されています．

フィールドデータファイルの読み込み

```
template<class TimeT, class TimeAvrT> void*
ReadData(CDM::E_CDM_ERRORCODE &ret,
         const unsigned step,
         const int gc,
         const int Gvoxel[3],
         const int Gdivision[3],
         const int head[3],
         const int tail[3],
         TimeT &time,
         const bool mode,
         unsigned &step_avr,
         TimeAvrT &time_avr);
```

フィールドデータファイルの読み込みを行います．

ret	[*output*]	エラーコード（表 3.10，3.11 を参照）
step	[*input*]	読込むフィールドデータのステップ番号
gc	[*input*]	計算空間の仮想セル数
Gvoxel	[*input*]	X,Y,Z 方向の計算空間全体のボクセルサイズ（3word の配列）
Gdivision	[*input*]	X,Y,Z 方向の領域分割数（3word の配列）
head	[*input*]	X,Y,Z 方向の計算領域の開始位置（3word の配列）
tail	[*input*]	X,Y,Z 方向の計算領域の終了位置（3word の配列）
time	[*output*]	読込んだ時間
mode	[*input*]	平均時間，平均化したステップ読込みフラグ （false : 読込む，true : 読込まない）
step_avr	[*output*]	読込んだ平均化したステップ
time_avr	[*output*]	読込んだ平均時間
戻り値		読込んだフィールドデータのポインタ

（注）取得したフィールドデータのポインタは，不要になった時にユーザーが delete する必要があります．

```
// フィールドデータファイルの読み込み
float* data = (float *)dfi->Read(引数);
      :
   （処理）
      :
// 不要になったので delete
delete [] data;
```

フィールドデータファイルの読み込み

```
template<class T, class TimeT, class TimeAvrT>
CDM::E_CDM_ERRORCODE
cdm_DFI::ReadData(T* val,
                  const unsigned step,
                  const int gc,
                  const int Gvoxel[3],
                  const int Gdivision[3],
                  const int head[3],
                  const int tail[3],
                  TimeT &time,
                  const bool mode,
                  unsigned &step_avr,
                  TimeAvrT &time_avr);
```

フィールドデータファイルの読み込みを行います.

val	[*output*]	読込み先の配列のポインタ
step	[*input*]	読込むフィールドデータのステップ番号
gc	[*input*]	計算空間の仮想セル数
Gvoxel	[*input*]	X,Y,Z 方向の計算空間全体のボクセルサイズ（3word の配列）
Gdivision	[*input*]	X,Y,Z 方向の領域分割数（3word の配列）
head	[*input*]	X,Y,Z 方向の計算領域の開始位置（3word の配列）
tail	[*input*]	X,Y,Z 方向の計算領域の終了位置（3word の配列）
time	[*output*]	読込んだ時間
mode	[*input*]	平均時間，平均化したステップ読込みフラグ （false : 読込む，true : 読込まない）
step_avr	[*output*]	読込んだ平均化したステップ
time_avr	[*output*]	読込んだ平均時間
戻り値		エラーコード（表 3.10，3.11 を参照）

3.2.6　リファインメントデータ補間メソッド

CDM のリファインメントデータの読込み処理では，以下の Fortran サブルーチン (cdm_interp.f90) により，単純な補間処理を行います.

1. IJKN 配列

cdm_interp_ijkn_r4 ： IJKN 配列，単精度実数版

```
subroutine cdm_interp_ijkn_r4(szS,gcS,szD,gcD,nc,src,dst)
  implicit none
  integer :: szS(3),gcS,szD(3),gcD,nc
  real*4,dimension(1-gcS:szS(1)+gcS,1-gcS:szS(2)+gcS,1-gcS:szS(3)+gcS,nc) :: src
  real*4,dimension(1-gcD:szD(1)+gcD,1-gcD:szD(2)+gcD,1-gcD:szD(3)+gcD,nc) :: dst
  integer :: i,j,k,n
  integer :: ii,jj,kk
  real*4  :: q

  include 'cdm_interp_ijkn.h'

  return
end subroutine cdm_interp_ijkn_r4
```

cdm_interp_ijkn_r8 ： IJKN 配列，倍精度実数版

```
subroutine cdm_interp_ijkn_r8(szS,gcS,szD,gcD,nc,src,dst)
  implicit none
  integer :: szS(3),gcS,szD(3),gcD,nc
  real*8,dimension(1-gcS:szS(1)+gcS,1-gcS:szS(2)+gcS,1-gcS:szS(3)+gcS,nc) :: src
  real*8,dimension(1-gcD:szD(1)+gcD,1-gcD:szD(2)+gcD,1-gcD:szD(3)+gcD,nc) :: dst
  integer :: i,j,k,n
  integer :: ii,jj,kk
  real*8  :: q

  include 'cdm_interp_ijkn.h'

  return
end subroutine cdm_interp_ijkn_r8
```

これらのサブルーチンの実際の補間処理部分は，外部のインクルードファイルに記述されています．補間アルゴリズムを変更する場合はこちらのインクルードファイルを修正してください.

2. NIJK 配列

cdm_interp_nijk_r4 ： NIJK 配列，単精度実数版

```
subroutine cdm_interp_nijk_r4(szS,gcS,szD,gcD,nc,src,dst)
  implicit none
  integer :: szS(3),gcS,szD(3),gcD,nc
  real*4,dimension(nc,1-gcS:szS(1)+gcS,1-gcS:szS(2)+gcS,1-gcS:szS(3)+gcS) :: src
  real*4,dimension(nc,1-gcD:szD(1)+gcD,1-gcD:szD(2)+gcD,1-gcD:szD(3)+gcD) :: dst
  integer :: i,j,k,n
  integer :: ii,jj,kk
  real*4  :: q

  include 'cdm_interp_nijk.h'

  return
end subroutine cdm_interp_nijk_r4
```

cdm_interp_nijk_r8 ： NIJK 配列，倍精度実数版

```
subroutine cdm_interp_nijk_r8(szS,gcS,szD,gcD,nc,src,dst)
  implicit none
  integer :: szS(3),gcS,szD(3),gcD,nc
  real*8,dimension(nc,1-gcS:szS(1)+gcS,1-gcS:szS(2)+gcS,1-gcS:szS(3)+gcS) :: src
  real*8,dimension(nc,1-gcD:szD(1)+gcD,1-gcD:szD(2)+gcD,1-gcD:szD(3)+gcD) :: dst
  integer :: i,j,k,n
  integer :: ii,jj,kk
  real*8  :: q

  include 'cdm_interp_nijk.h'

  return
end subroutine cdm_interp_nijk_r8
```

これらのサブルーチンの実際の補間処理部分は，外部のインクルードファイルに記述されています．補間アルゴリズムを変更する場合はこちらのインクルードファイルを修正してください．

3. インクルードファイルのループインデックスと参照インデックスの関係
   - src と dst は仮想セルを含めて単純に各方向 2 倍にした配列
   - ループは補間元 (src) 配列インデックスループしている (仮想セル含む)
   - i,j,k が src，ii,jj,kk が dst

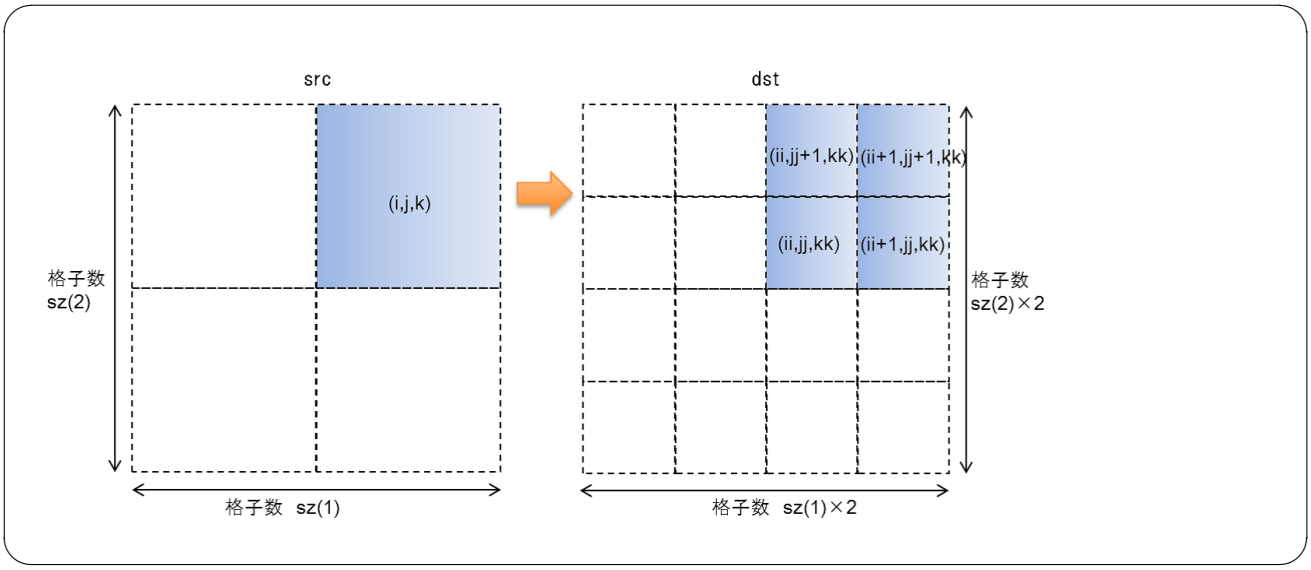


図 3.5　補間処理

cdm_interp_ijkn.h ： IJKN 配列 補間処理部分

```
do n=1,nc
do k=1-gcS,szS(3)+gcS
  kk=(k-1)*2+1
do j=1-gcS,szS(2)+gcS
  jj=(j-1)*2+1
do i=1-gcS,szS(1)+gcS
  ii=(i-1)*2+1
  q = src(i,j,k,n)
  dst(ii  ,jj  ,kk  ,n) = q
  dst(ii+1,jj  ,kk  ,n) = q
  dst(ii  ,jj+1,kk  ,n) = q
  dst(ii+1,jj+1,kk  ,n) = q
  dst(ii  ,jj  ,kk+1,n) = q
  dst(ii+1,jj  ,kk+1,n) = q
  dst(ii  ,jj+1,kk+1,n) = q
  dst(ii+1,jj+1,kk+1,n) = q
enddo
enddo
enddo
enddo
```

cdm_interp_nijk.h ： NIJK 配列 補間処理部分

```
do k=1-gcS,szS(3)+gcS
  kk=(k-1)*2+1
do j=1-gcS,szS(2)+gcS
  jj=(j-1)*2+1
do i=1-gcS,szS(1)+gcS
  ii=(i-1)*2+1
do n=1,nc
  q = src(n,i,j,k)
  dst(n,ii  ,jj  ,kk  ) = q
  dst(n,ii+1,jj  ,kk  ) = q
  dst(n,ii  ,jj+1,kk  ) = q
  dst(n,ii+1,jj+1,kk  ) = q
  dst(n,ii  ,jj  ,kk+1) = q
  dst(n,ii+1,jj  ,kk+1) = q
  dst(n,ii  ,jj+1,kk+1) = q
  dst(n,ii+1,jj+1,kk+1) = q
enddo
enddo
enddo
enddo
```

［補足］

src　：　読込んだ粗データ配列

szS　：　src の実ボクセルのサイズが入った配列

gcS　：　src の仮想セル数

dst　：　粗データを補間処理した密データ配列

3.2.7　入力処理のサンプルコード

1. 引数で渡された配列のポインタにフィールドデータを読込む

```
#include "cdm_DFI.h"
int main( int argc, char **argv )
{
  //CDM のエラーコード
  CDM::E_CDM_ERRORCODE ret = CDM::E_CDM_SUCCESS;

  //MPI Initialize
  if( MPI_Init(&argc,&argv) != MPI_SUCCESS )
  {
     std::cerr << "MPI_Init error." << std::endl;
     return 0;
  }

  //引数で渡された dfi ファイル名をセット
  if( argc != 2 ) {
    //エラー、DFI ファイル名が引数で渡されない
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CDM::E_CDM_ERROR;
  }
  std::string dfi_fname = argv[1];

  //計算空間の定義
  int GVoxel[3] = {64, 64, 64}; ///<計算空間全体のボクセルサイズ
  int GDiv[3]   = {1, 1, 1};    ///<領域分割数（並列数）
  int head[3]   = {1, 1, 1};    ///<計算領域の開始位置
  int tail[3]   = {64, 64, 64}; ///<計算領域の終了位置
  int gsize     = 2;            ///<計算空間の仮想セル数
  //読込み配列のサイズ
  size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

  //読込み用インスタンスのポインタ取得
  cdm_DFI* DFI_IN = cdm_DFI::ReadInit(MPI_COMM_WORLD, ///<MPI コミュニケータ
                                      dfi_fname,      ///<dfi ファイル名
                                      GVoxel,         ///<計算空間全体のボクセルサイズ
                                      GDiv,           ///<領域分割数
                                      ret);           ///<エラーコード
  //エラー処理
  if( ret != CDM::E_CDM_SUCCESS || DFI_IN == NULL ) {
    //エラーインスタンス失敗
    std::cerr << "Error Readinit." << std::endl;
    return ret;
  }

  //読込みフィールドデータ型のチェック
  if( DFI_IN->GetDataType() != CDM::E_CDM_FLOAT32 ) {
    //データの型違い
    std::cerr << "Error Datatype unmatch." << std::endl;
    return CDM::E_CDM_ERROR;
  }

  //読込みフィールドデータの変数の個数を取得
  int nvari=DFI_IN->GetNumVariables();

  //単位系の取得
  std::string Lunit;
  double Lref,Ldiff;
  bool LBset;
  ret=DFI_IN->GetUnit("Length",Lunit,Lref,Ldiff,LBset);
  if( ret==CDM::E_CDM_SUCCESS ) {
    printf("Length\n");
    printf("  Unit      : %s\n",Lunit.c_str());
    printf("  reference : %e\n",Lref);
```

```
    if( LBset ) {
      printf("  difference: %e\n",Ldiff);
    }
  }

  //読込み配列のアロケート
  float *d_v = new float[size*nvari];
  //読込み配列のゼロクリア
  memset(d_v, 0, sizeof(float)*size*nvari);
  //読込みフィールドデータのステップ番号をセット
  unsigned step = 10;

  float r_time;       ///<dfi から読込んだ時間
  unsigned i_dummy;   ///<平均化ステップ
  float f_dummy;      ///<平均時間

  //フィールドデータの読込み
  ret =  DFI_IN->ReadData(d_v,    ///<読込み先配列のポインタ
                          step,   ///<読込みフィールドデータのステップ番号
                          gsize,  ///<計算空間の仮想セル数
                          GVoxel, ///<計算空間全体のボクセルサイズ
                          GDiv,   ///<領域分割数
                          head,   ///<計算領域の開始位置
                          tail,   ///<計算領域の終了位置
                          r_time, ///<dfi から読込んだ時間
                          true,   ///<平均を読込まない
                          i_dummy,
                          f_dummy );
  //エラー処理
  if( ret != CDM::E_CDM_SUCCESS ) {
    //フィールドデータの読込み失敗
    std::cerr << "Error ReadData." << std::endl;
    delete [] d_v;
    delete DFI_IN;
    return ret;
  }

  //正常終了処理
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<配列ポインタの削除
  delete DFI_IN;  ///<読込み用インスタンスのポインタの削除
  return CDM::E_CDM_SUCCESS;
}
```

2. 読込んだフィールドデータの配列ポインタを戻す

```
#include "cdm_DFI.h"
int main( int argc, char **argv )
{
  //CDM のエラーコード
  CDM::E_CDM_ERRORCODE ret = CDM::E_CDM_SUCCESS;

  //MPI Initialize
  if( MPI_Init(&argc,&argv) != MPI_SUCCESS )
  {
     std::cerr << "MPI_Init error." << std::endl;
     return 0;
  }

  //引数で渡された dfi ファイル名をセット
  if( argc != 2 ) {
    //エラー、DFI ファイル名が引数で渡されない
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CDM::E_CDM_ERROR;
  }
  std::string dfi_fname = argv[1];
```

```
  //計算空間の定義
  int GVoxel[3] = {64, 64, 64}; ///<計算空間全体のボクセルサイズ
  int GDiv[3]   = {1, 1, 1};    ///<領域分割数（並列数）
  int head[3]   = {1, 1, 1};    ///<計算領域の開始位置
  int tail[3]   = {64, 64, 64}; ///<計算領域の終了位置
  int gsize     = 2;            ///<計算空間の仮想セル数

  //読込み用インスタンスのポインタ取得
  cdm_DFI* DFI_IN = cdm_DFI::ReadInit(MPI_COMM_WORLD, ///<MPI コミュニケータ
                                      dfi_fname,      ///<dfi ファイル名
                                      GVoxel,         ///<計算空間全体のボクセルサイズ
                                      GDiv,           ///<領域分割数
                                      ret);           ///<エラーコード
  //エラー処理
  if( ret != CDM::E_CDM_SUCCESS || DFI_IN == NULL ) {
    //エラーインスタンス失敗
    std::cerr << "Error Readinit." << std::endl;
    return ret;
  }
  //読込みフィールドデータ型のチェック
  if( DFI_IN->GetDataType() != CDM::E_CDM_FLOAT32 ) {
    //データの型違い
    std::cerr << "Error Datatype unmatch." << std::endl;
    return CDM::E_CDM_ERROR;
  }
  //単位系の取得
  cdm_UnitElem unit;
  ret=DFI_IN->GetUnitElem("Pressure",unit);
  if( ret==CDM::E_CDM_SUCCESS ) {
    printf("Pressure\n");
    printf("  Unit      : %s\n",unit.Unit.c_str());
    printf("  reference : %e\n",unit.reference);
    if( unit.BsetDiff ) {
      printf("  diferrence: %e\n",unit.difference);
    }
  }

  unsigned step = 10; ///<読込みフィールドデータのステップ番号
  float r_time;       ///<dfi から読込んだ時間
  unsigned i_dummy;   ///<平均化ステップ
  float f_dummy;      ///<平均時間

  //フィールドデータの読込み
  float* d_v = (float *)DFI_IN->ReadData(
                        ret,    ///<リターンコード
                        step,   ///<読込みフィールドデータのステップ番号
                        gsize,  ///<計算空間の仮想セル数
                        GVoxel, ///<計算空間全体のボクセルサイズ
                        GDiv,   ///<領域分割数
                        head,   ///<計算領域の開始位置
                        tail,   ///<計算領域の終了位置
                        r_time, ///<dfi から読込んだ時間
                        true,   ///<平均を読込まない
                        i_dummy,
                        f_dummy );
  //エラー処理
  if( ret != CDM::E_CDM_SUCCESS ) {
    std::cerr << "Error ReadData." << std::endl;
    delete [] d_v;
    delete DFI_IN;
    return ret;
  }

  //正常終了処理
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<配列ポインタの削除
  delete DFI_IN;  ///<読込み用インスタンスのポインタの削除
  return CDM::E_CDM_SUCCESS;
}
```

3.3　出力機能

3.3.1　機能概要

CDM ライブラリでは，フィールドデータファイルの出力機能として 1 対 1 のみの出力をサポートしています.

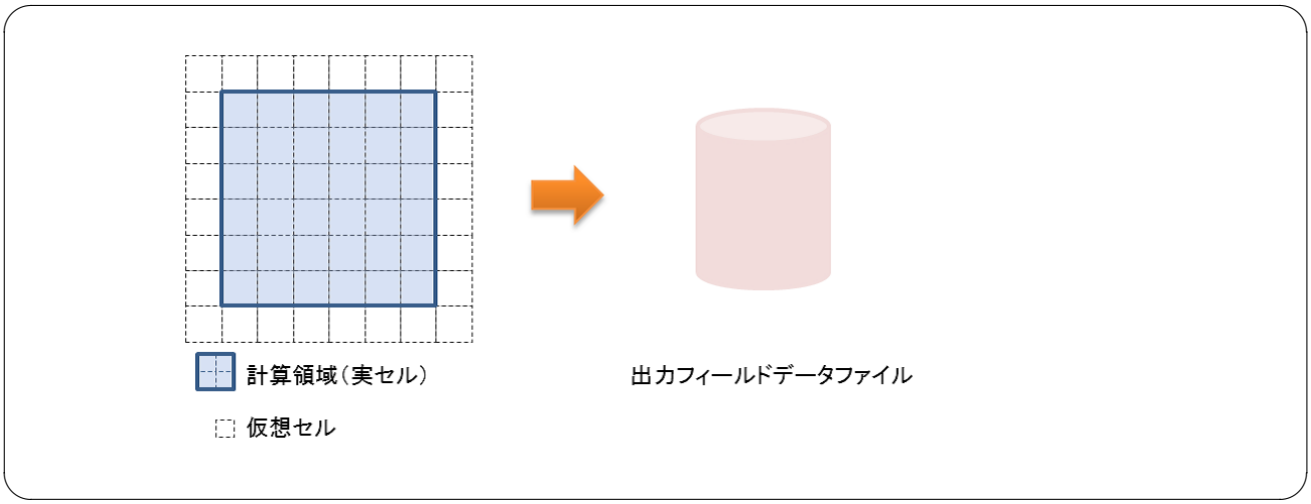


図 3.6　1 対 1 の出力

3.3.2　出力処理手順

CDM では以下の手順で，フィールドデータ及び DFI データの出力処理を行います.

1. 出力用インスタンスのポインタ取得 (3.3.3 章参照)
2. 出力する DFI の情報を登録 (3.3.4 章参照)
3. 格子ファイル出力 (PLOT3D 形式の場合) (3.3.5 章参照)
4. proc.dfi ファイル出力 (3.3.7 章参照)
5. フィールドデータファイル出力 (3.3.8 章参照)

3.3.3 出力用インスタンスのポインタ取得

cdm_DFI クラスのインスタンスは，DFI ファイルの種類毎にいくつでも生成可能です．そのインスタンスへのポインタを取得するメソッドは，cdm_DFI.h 内で次のように定義されています．

出力用インスタンスの生成，インスタンスへのポインタの取得（等間隔格子用）

```
template<typename T>
static cdm_DFI*
cdm_DFI::WriteInit(const MPI_Comm comm,
                   const std::string DfiName,
                   const std::string Path,
                   const std::string prefix,
                   const CDM::E_CDM_FORMAT format,
                   const int GCell,
                   const CDM::E_CDM_DTYPE DataType,
                   const int nVari,
                   const std::string proc_fname,
                   const int G_size[3],
                   const T pitch[3],
                   const T G_origin[3],
                   const int division[3],
                   const int head[3],
                   const int tail[3],
                   const std::string hostname,
                   const CDM::E_CDM_ONOFF TSliceOnOff);
```

cdm_DFI クラスのインスタンスへのポインタを取得します．

comm	[*input*]	MPI コミュニケータ
DfiName	[*input*]	出力する index.dfi ファイル名
Path	[*input*]	出力するフィールドデータのディレクトリ
prefix	[*input*]	ベースファイル名
format	[*input*]	フィールドデータのファイルフォーマット（表 3.3 参照）
GCell	[*input*]	出力する仮想セルの数
DataType	[*input*]	フィールドデータのデータ型（表 3.4 参照）
nVari	[*input*]	フィールドデータの変数の個数
proc_fname	[*input*]	出力する proc.dfi ファイル名
G_size	[*input*]	X,Y,Z 方向の計算空間全体のボクセルサイズ（3word の配列）
pitch	[*input*]	X,Y,Z 方向のボクセルピッチ（3word の配列，float 型もしくは double 型）
G_origin	[*input*]	計算空間全体の原点座標値（3word の配列，float 型もしくは double 型）
division	[*input*]	X,Y,Z 方向の領域分割数（3word の配列）
head	[*input*]	X,Y,Z 方向の計算領域の開始位置（3word の配列）
tail	[*input*]	X,Y,Z 方向の計算領域の終了位置（3word の配列）
hostname	[*input*]	ホストノード名
TSliceOnOff	[*input*]	タイムスライス毎のディレクトリに出力させるフラグ（表 3.2 参照）
戻り値	cdm_DFI クラスのインスタンスへのポインタ	

(注) インスタンスされたポインタは，不要になった時にユーザが delete する必要があります．また，テンプレート引数の型 (float か double) を明示的に与えて，格子のピッチおよび原点座標の型を指定するようにします．

```
//DFI のインスタンス
cdmDFI *DFI_OUT_PRS = cdm_DFI::WriteInit<float>(,,,);
                :
          (処理)
                :
//不要になったので delete
```

```
delete DFI_OUT_PRS;
```

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.

出力用インスタンスの生成，インスタンスへのポインタの取得（不等間隔格子用）

```
template<typename T>
static cdm_DFI*
cdm_DFI::WriteInit(const MPI_Comm comm,
                   const std::string DfiName,
                   const std::string Path,
                   const std::string prefix,
                   const CDM::E_CDM_FORMAT format,
                   const int GCell,
                   const CDM::E_CDM_DTYPE DataType,
                   const int nVari,
                   const std::string proc_fname,
                   const int G_size[3],
                   const T* coord_X,
                   const T* coord_Y,
                   const T* coord_Z,
                   const std::string coord_file,
                   const CDM::E_CDM_FILE_TYPE coord_filetype,
                   const CDM::E_CDM_ENDIANTYPE coord_fileEndian,
                   const int division[3],
                   const int head[3],
                   const int tail[3],
                   const std::string hostname,
                   const CDM::E_CDM_ONOFF TSliceOnOff);
```

cdm_DFI クラスのインスタンスへのポインタを取得します.

comm	[*input*]	MPI コミュニケータ
DfiName	[*input*]	出力する index.dfi ファイル名
Path	[*input*]	出力するフィールドデータのディレクトリ
prefix	[*input*]	ベースファイル名
format	[*input*]	フィールドデータのファイルフォーマット（表 3.3 参照）
GCell	[*input*]	出力する仮想セルの数
DataType	[*input*]	フィールドデータのデータ型（表 3.4 参照）
nVari	[*input*]	フィールドデータの変数の個数
proc_fname	[*input*]	出力する proc.dfi ファイル名
G_size	[*input*]	X,Y,Z 方向の計算空間全体のボクセルサイズ（3word の配列）
coord_X	[*input*]	格子点の X 座標データポインタ（※ 1）
coord_Y	[*input*]	格子点の Y 座標データポインタ（※ 1）
coord_Z	[*input*]	格子点の Z 座標データポインタ（※ 1）
coord_file	[*input*]	座標ファイル名
coord_filetype	[*input*]	座標ファイルのファイルタイプ
coord_fileEndian	[*input*]	座標ファイルのエンディアンタイプ
division	[*input*]	X,Y,Z 方向の領域分割数（3word の配列）
head	[*input*]	X,Y,Z 方向の計算領域の開始位置（3word の配列）
tail	[*input*]	X,Y,Z 方向の計算領域の終了位置（3word の配列）
hostname	[*input*]	ホストノード名
TSliceOnOff	[*input*]	タイムスライス毎のディレクトリに出力させるフラグ（表 3.2 参照）
戻り値		cdm_DFI クラスのインスタンスへのポインタ

(※ 1) 小さい順に座標値を格納した配列のポインタ．座標値は float 型もしくは double 型.

(注) インスタンスされたポインタは，不要になった時にユーザが delete する必要があります．また，テンプレート引数の型 (float か double) を明示的に与えて，格子の座標データの型を指定するようにします.

```
//DFI のインスタンス
cdmDFI *DFI_OUT_PRS = cdm_DFI::WriteInit<float>(,,,);
                :
            (処理)
                :
//不要になったので delete
delete DFI_OUT_PRS;
```

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.

3.3.4　DFI 情報の追加登録

時系列データの出力時，インスタンスした DFI 情報に各出力ステップにおける情報を追加登録するには CDM のメソッドを使用します．以下に DFI 情報を登録するメソッドを説明します.

DFI 情報の登録処理を行うメソッドは cdm_DFI.h 内で次のように定義されています.

1. 単位系の登録

   単位系は DFI ファイルの Unit に出力します.

   DFI ファイルの Unit の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

単位系の登録

```
void
cdm_DFI::AddUnit(const std::string Name,
                 const std::string Unit,
                 const double reference,
                 const double difference = 0.0,
                 const bool BsetDiff = false);
```

Unit に単位系を登録します.

Name	[*input*]	登録する単位系	(”Length”,”Velocity”,,,,)
Unit	[*input*]	単位につけるラベル	(”M”,”CM”,”MM”,”M/S”,,)
reference	[*input*]	規格化したスケール値	(”L0”,”V0”,,,,)
difference	[*input*]	差の値 (※ 1)	
BsetDiff	[*input*]	difference の有無 (※ 2)	

(※ 1) 省略可．ただし省略した場合 BsetDiff は無効.
(※ 2) 省略可．　省略した場合 false

2. 時系列データの登録

   時系列データは DFI ファイルの TimeSlice に出力します.

   DFI ファイルの TimeSlice の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

時系列データの登録

```
template<class T, class TimeT, class TimeAvrT>
void
cdm_DFI::AddTimeSlice(const unsigned step,
                      TimeT time,
                      T* minmax = NULL,
                      bool avr_mode = true,
                      unsigned step_avr = 0,
                      TimeAvrT time_avr = 0.0);
```

TimeSlice に時系列データを登録します.

step	[*input*]	登録するステップ番号
time	[*input*]	登録する時刻
minmax	[*input*]	登録するデータの MinMax（※ 1）
avr_mode	[*input*]	平均ステップ，時間の出力指示 false：出力（※ 2）
step_avr	[*input*]	平均ステップ（※ 3）
time_avr	[*input*]	平均時刻（※ 4）

(※ 1) 省略可．ただし省略した場合は登録無効.
(※ 2) 省略可．省略した場合は ture
(※ 3) 省略可．省略した場合は 0
(※ 4) 省略可．省略した場合は 0.0

3. TimeSlice 毎のディレクトリ出力指示を登録する

TimeSlice 毎のディレクトリ出力指示を登録する

```
void
cdm_DFI::SetTimeSliceFlag(const CDM::E_CDM_ONOFF ONOFF);
```

ONOFF	[*input*]	出力指示フラグ (表 3.2 参照)

4. DFI FileInfo の変数名を登録する

DFI ファイルの FileInfo の仕様は 7.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

DFI FileInfo の変数名を登録する

```
void
cdm_DFI::setVariableName(int pvari,
                         std::string variName);
```

pvari	[*input*]	登録する変数名の位置
variName	[*input*]	登録する変数名

5. 読込みランクリストを登録する

読込みランクリストを登録する

```
CDM::E_CDM_ERRORCODE
cdm_DFI::CheakReadRank(cdm_Domain dfi_domain,
                       const int head[3],
                       const int tail[3],
                       CDM::E_CDM_READTYPE readflag,
                       vector<int> &readRankList);
```

dfi_domain	[*input*]	DFI の Domian 情報
head	[*input*]	計算領域開始インデックス (3word の配列)
tail	[*input*]	計算領域終了インデックス (3word の配列)
readflag	[*input*]	フィールドデータの読込み方法 (表 3.7 参照)
readRankList	[*output*]	読込みランクリスト
戻り値	エラーコード（表 3.10，3.11 を参照）	

6. 出力フィールドファイル名の命名規約を登録する

   出力されるフィールドファイル名の命名規約をセットします.

   NetCDF4(/w HDF5) 形式ファイルの場合，CDM::E_CDM_OUTPUT_FNAME_RANK を登録することで，1 ファイルに全時系列情報を出力する方式に変更できます.

   本登録を行わない場合のデフォルトは CDM::E_CDM_OUTPUT_FNAME_STEP_RANK です.

出力フィールドファイル名の命名規約を登録する

```
void
cdm_DFI::set_output_fname(CDM::E_CDM_OUTPUT_FNAME output_fname)
```

output_fname	[*input*]	フィールドファイル名の命名規約 (表 3.9 参照)

3.3.5 格子ファイル出力

格子点の座標を記述したファイルを出力することができます．本ライブラリで扱うファイル形式のうち，PLOT3D 形式のみ格子ファイルが存在し，xyz ファイルと呼ばれています (xyz ファイルについては，7.1.3 章「フィールドデータファイルの仕様」参照).

格子ファイルの出力の処理を行うメソッドは cdm_DFI.h 内で次のように定義されています.

格子ファイルの出力

```
CDM::E_CDM_ERRORCODE
cdm_DFI::WriteGridFile(const int* iblank = NULL);
```

格子ファイルの出力を行います.

iblank	[*input*]	IBLANK 値を格納した一次元配列のポインタ（※ 1）
戻り値	エラーコード（表 3.10，3.11 を参照）	

(※ 1) 格子点 (i,j,k) の IBLANK 値を，出力するガイドセルも含めて，表 3.12 に示した順で一次元配列に格納. IBLANK 値の意味については，表 7.13 を参照．ポインタを省略した場合，IBLANK の設定はなし.

表 3.12　一次元配列 v_ib における IBLANK 値の格納順序

配列要素	説明
v_ib(0)	格子点 (0,0,0) の IBLANK 値
v_ib(1)	格子点 (1,0,0) の IBLANK 値
・・・	
v_ib(imax-1)	格子点 (imax-1,0,0) の IBLANK 値
v_ib(imax)	格子点 (0,1,0)IBLANK 値
・・・	
v_ib(imax*jmax-1)	格子点 (imax-1,jmax-1,0) の IBLANK 値
v_ib(imax*jmax)	格子点 (0,0,1) の IBLANK 値
・・・	
v_ib(imax*jmax*kmax-1)	格子点 (imax-1,jmax-1,kmax-1) の IBLANK 値

3.3.6　index.dfi ファイル出力

index.dfi ファイルの出力の処理を行うメソッドは cdm_DFI.h 内で次のように定義されています.

index.dfi ファイルの出力

```
CDM::E_CDM_ERRORCODE
cdm_DFI::WriteIndexDfiFile();
```

index.dfi ファイルの出力を行います.
戻り値　エラーコード（表 3.10，3.11 を参照）

3.3.7　proc.dfi ファイル出力

proc.dfi ファイルの出力の処理を行うメソッドは cdm_DFI.h 内で次のように定義されています.

proc.dfi ファイルの出力

```
CDM::E_CDM_ERRORCODE
cdm_DFI::WriteProcDfiFile(const MPI_Comm comm,
                          bool out_host);
```

proc.dfi ファイルの出力を行います.

comm	[*input*]	MPI コミュニケータ	
out_host	[*input*]	ホスト名出力指示フラグ	false：出力させない
			true ：出力させる
戻り値		エラーコード（表 3.10，3.11 を参照）	

3.3.8　フィールドデータファイル出力

フィールドデータファイルの形式は，SPH 形式，BOV 形式，PLOT3D 形式，NetCDF4(/w HDF5) 形式ファイルです．（詳細は，7.1.3 章を参照してください）

フィールドデータファイルの出力の処理を行うメソッドは cdm_DFI.h 内で次のように定義されています．

フィールドデータファイルの出力（index.dfi ファイルの出力を含む）

```
template<class T, class TimeT, class TimeAvrT>
CDM::E_CDM_ERRORCODE
cdm_DFI::WriteData(const unsigned step,
                   TimeT time,
                   const int sz[3],
                   const int nVari,
                   const int gc,
                   T* val,
                   T* minmax = NULL,
                   bool avr_mode = true,
                   unsigned step_avr = 0,
                   TimeAvrT time_avr = 0.0);
```

フィールドデータファイルの出力を行います．

step	[*input*]	出力するステップ番号
time	[*input*]	出力時刻
sz	[*input*]	出力するデータの配列 val の X,Y,Z 方向の実ボクセル数（3word の配列）
nVari	[*input*]	出力するデータの変数の個数
gc	[*input*]	出力するデータの配列 val の仮想セル数
val	[*input*]	出力するデータの配列のポインタ
minmax	[*input*]	出力するデータの MinMax（※ 1） minmax[0]=変数 1 の minX minmax[1]=変数 1 の maxX ： minmax[2n-2]=変数 n の maxX minmax[2n-1]=変数 n の minX minmax[2n]=合成値の min（※ 2） minmax[2n+1]=合成値の max（※ 2）
avr_mode	[*input*]	平均ステップ，時間の出力指示 false：出力（※ 3）
step_avr	[*input*]	平均ステップ（※ 4）
time_avr	[*input*]	平均時刻（※ 5）
戻り値		エラーコード（表 3.10，3.11 を参照）

(※ 1) 省略可．ただし省略した場合は登録無効．
(※ 2) SPH 形式で変数の個数が 3 の場合のみ．
(※ 3) 省略可．省略した場合は ture
(※ 4) 省略可．省略した場合は 0
(※ 5) 省略可．省略した場合は 0.0

フィールドデータファイルの出力（index.dfi ファイルの出力を含まない）

```
template<class T, class TimeT, class TimeAvrT>
CDM::E_CDM_ERRORCODE
cdm_DFI::WriteFieldDataFile(const unsigned step,
                            TimeT time,
                            const int sz[3],
                            const int nVari,
                            const int gc,
                            T* val,
                            bool avr_mode = true,
                            unsigned step_avr = 0,
                            TimeAvrT time_avr = 0.0);
```

フィールドデータファイルの出力を行います.

step	[*input*]	出力するステップ番号
time	[*input*]	出力時刻
sz	[*input*]	出力するデータの配列 val の X,Y,Z 方向の実ボクセル数（3word の配列）
nVari	[*input*]	出力するデータの変数の個数
gc	[*input*]	出力するデータの配列 val の仮想セル数
val	[*input*]	出力するデータの配列のポインタ
avr_mode	[*input*]	平均ステップ，時間の出力指示 false：出力（※ 1）
step_avr	[*input*]	平均ステップ（※ 2）
time_avr	[*input*]	平均時刻（※ 3）
戻り値		エラーコード（表 3.10，3.11 を参照）

(※ 1) 省略可．省略した場合は ture
(※ 2) 省略可．省略した場合は 0
(※ 3) 省略可．省略した場合は 0.0

3.3.9　出力処理のサンプルコード

1. 等間隔格子のデータ出力を行う

```
#include "cdm_DFI.h"
int main( int argc, char **argv )
{
  //IO のエラーコード
  CDM::E_CDM_ERRORCODE ret = CDM::E_CDM_SUCCESS;

  //MPI Initialize
  if( MPI_Init(&argc,&argv) != MPI_SUCCESS )
  {
     std::cerr << "MPI_Init error." << std::endl;
     return 0;
  }

  //引数で渡された dfi ファイル名をセット
  if( argc != 2 ) {
    //エラー、DFI ファイル名が引数で渡されない
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CDM::E_CDM_ERROR;
  }
  std::string dfi_fname = argv[1];

  //計算空間の定義
  int GVoxel[3] = {64, 64, 64}; ///<計算空間全体のボクセルサイズ
  int GDiv[3]   = {1, 1, 1};    ///<領域分割数（並列数）
  int head[3]   = {1, 1, 1};    ///<計算領域の開始位置
  int tail[3]   = {64, 64, 64}; ///<計算領域の終了位置
  int gsize     = 2;            ///<計算空間の仮想セル数
  float pit[3]  = {1.0/64.0, 1.0/64.0, 1.0/64.0} ; ///<ピッチ
  float org[3]  = {-0.5, -0.5, -0.5};              ///<原点座標値
  //配列のサイズ
  size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

  std::string path  = "./";     ///<出力ディレクトリ
  std::string prefix= "field";  ///<ベースファイル名
  int out_gc        = 1;        ///<出力仮想セル数
  int nvari         = 4;        ///<データの変数の個数
  CDM::E_CDM_FORMAT format  = CDM::E_CDM_FMT_PLOT3D; ///<出力フォーマット
  CDM::E_CDM_DTYPE datatype = CDM::E_CDM_FLOAT32;    ///<データ型
  std::string proc_fname    = "proc.dfi";            ///<proc ファイル名
  std::string hostname      = "";                    ///<ホスト名
  CDM::E_CDM_ONOFF TimeSliceOnOff = CDM::E_CDM_OFF;  ///<タイムスライス出力指示

  //出力用インスタンスのポインタ取得
  cdm_DFI* DFI_OUT = cdm_DFI::WriteInit<float>(MPI_COMM_WORLD,  ///<MPI コミュニケータ
                                               dfi_fname,       ///<dfi ファイル名
                                               path,            ///<出力ディレクトリ
                                               prefix,          ///<ベースファイル名
                                               format,          ///<出力フォーマット
                                               out_gc,          ///<出力仮想セル数
                                               datatype,        ///<データ型
                                               nvari,           ///<データの変数の個数
                                               proc_fname,      ///<proc ファイル名
                                               GVoxel,          ///<計算空間全体のボクセルサイズ
                                               pit,             ///<ピッチ
                                               org,             ///<原点座標値
                                               GDiv,            ///<領域分割数
                                               head,            ///<計算領域の開始位置
                                               tail,            ///<計算領域の終了位置
                                               hostname,        ///<ホスト名
                                               TimeSliceOnOff); ///<タイムスライス出力オプション
  //エラー処理
  if( DFI_OUT == NULL )
```

```
{
  //エラーインスタンス失敗
  std::cerr << "Error Writeinit." << std::endl;
  return CDM::E_CDM_ERROR;
}
//unit の登録
DFI_OUT->AddUnit("Length","NonDimensional",1.0);
DFI_OUT->AddUnit("Velocity","NonDimensional",1.0);
DFI_OUT->AddUnit("Pressure","NonDimensional",0.0,0.0,true);
//proc ファイル出力
DFI_OUT->WriteProcDfiFile(MPI_COMM_WORLD, ///<MPI コミュニケータ
                          false);         ///<ホスト名出力指示

//IBLANK 用配列の設定 (PLOT3D 形式)
size_t size_ib=(GVoxel[0]+2*out_gc)*(GVoxel[1]+2*out_gc)*(GVoxel[2]+2*out_gc);
int *iblank_v;
if (format == CDM::E_CDM_FMT_PLOT3D) {
  //配列 (iblank_v) のアロケート
  iblank_v = new int[size_ib];
  //配列 (iblank_v) に値をセット
  for(int i=0; i<size_ib; i++) {
    iblank_v[i] = 1;
  }
}

//格子ファイルの出力 (PLOT3D 形式)
ret = DFI_OUT->WriteGridFile(iblank_v); ///<plot3d の xyz ファイル出力 (IBLANK 値含む)
//エラー処理
if( ret != CDM::E_CDM_SUCCESS ) {
  //格子ファイルの出力失敗
  std::cerr << "Error WriteGridFile." << std::endl;
  return CDM::E_CDM_ERROR;
}

// IBLANK 用配列のデアロケート
delete [] iblank_v; ///<IBLANK 用配列のポインタの削除

unsigned step=10;  ///<出力ステップ番号
float r_time=0.0; ///<出力時間
float minmax[8];  ///<minmax
//minmax のゼロクリア
for(int i=0; i<8; i++) minmax[i]=0.0;

//変数名の登録
DFI_OUT->setVariableName(0,"u");
DFI_OUT->setVariableName(1,"v");
DFI_OUT->setVariableName(2,"w");
DFI_OUT->setVariableName(3,"P");

//TimeSlice の登録
DFI_OUT->AddTimeSlice(step,    ///<登録ステップ番号
                      r_time,  ///<登録時間
                      minmax,  ///<最小値，最大値
                      true,    ///<平均出力なし
                      0,       ///<平均をとったステップ数
                      0.0);    ///<平均をとった時刻

//index ファイル出力
ret = DFI_OUT->WriteIndexDfiFile();
//エラー処理
if( ret != CDM::E_CDM_SUCCESS ) {
  //index ファイルの出力失敗
  std::cerr << "Error WriteIndexDfiFile." << std::endl;
  return CDM::E_CDM_ERROR;
}

//配列のアロケート
float *d_v = new float[size*nvari];

//フィールドデータの出力
ret = DFI_OUT->WriteFieldDataFile(step,   ///<出力ステップ番号
```

```
                                    r_time, ///<出力時間
                                    GVoxel, ///<d_v の実ボクセル数
                                    nvari,  ///<d_v の成分数
                                    gsize,  ///<d_v の仮想セル数
                                    d_v,    ///<出力するフィールドデータポインタ
                                    true,   ///<平均出力なし
                                    0,      ///<平均をとったステップ数
                                    0.0);   ///<平均をとった時刻
  //エラー処理
  if( ret != CDM::E_CDM_SUCCESS ) {
    //フィールドデータの出力失敗
    std::cerr << "Error WriteFieldDataFile." << std::endl;
    delete [] d_v;
    delete DFI_OUT;
    return ret;
  }

  //正常終了処理
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<配列ポインタの削除
  delete DFI_OUT; ///<出力インスタンスのポインタ削除
  return CDM::E_CDM_SUCCESS;
}
```

2. 不等間隔格子のデータ出力を行う

```
#include "cdm_DFI.h"
int main( int argc, char **argv )
{
  //IO のエラーコード
  CDM::E_CDM_ERRORCODE ret = CDM::E_CDM_SUCCESS;

  //MPI Initialize
  if( MPI_Init(&argc,&argv) != MPI_SUCCESS )
  {
     std::cerr << "MPI_Init error." << std::endl;
     return 0;
  }

  //引数で渡された dfi ファイル名をセット
  if( argc != 2 ) {
    //エラー、DFI ファイル名が引数で渡されない
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CDM::E_CDM_ERROR;
  }
  std::string dfi_fname = argv[1];

  //計算空間の定義
  int GVoxel[3] = {10, 5, 7};   ///<計算空間全体のボクセルサイズ
  int GDiv[3]   = {1, 1, 1};    ///<領域分割数（並列数）
  int head[3]   = {1, 1, 1};    ///<計算領域の開始位置
  int tail[3]   = {10, 5, 7};   ///<計算領域の終了位置
  int gsize     = 1;            ///<計算空間の仮想セル数
  //配列のサイズ
  size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

  std::string path  = "./";     ///<出力ディレクトリ
  std::string prefix= "field";  ///<ベースファイル名
  int out_gc        = 1;        ///<出力仮想セル数
  int nvari         = 4;        ///<データの変数の個数
  CDM::E_CDM_FORMAT format  = CDM::E_CDM_FMT_PLOT3D; ///<出力フォーマット
  CDM::E_CDM_DTYPE datatype = CDM::E_CDM_FLOAT32;    ///<データ型
  std::string proc_fname    = "proc.dfi";            ///<proc ファイル名
  std::string hostname      = "";                    ///<ホスト名
  CDM::E_CDM_ONOFF TimeSliceOnOff = CDM::E_CDM_OFF;  ///<タイムスライス出力指示

  //Domain 情報の設定
```

```
double *coord_X = NULL;
double *coord_Y = NULL;
double *coord_Z = NULL;
coord_X = new double[GVoxel[0]+1];
coord_Y = new double[GVoxel[1]+1];
coord_Z = new double[GVoxel[2]+1];
for(int i=0; i<GVoxel[0]+1; i++) {
  coord_X[i] = double(i*i);
}
for(int j=0; j<GVoxel[1]+1; j++) {
  coord_Y[j] = double(j*j);
}
for(int k=0; k<GVoxel[2]+1; k++) {
  coord_Z[k] = double(k*k);
}
std::string coord_file = "coord.crd";                                  ///<座標ファイル名
CDM::E_CDM_FILE_TYPE coord_filetype = CDM::E_CDM_FILE_TYPE_BINARY; ///<座標ファイルの
                                                                       /// データフォーマット
CDM::E_CDM_ENDIANTYPE coord_fileEndian = CDM::E_CDM_LITTLE;        ///<座標ファイルの
                                                                       /// エンディアンタイプ

//出力用インスタンスのポインタ取得
cdm_DFI* DFI_OUT = cdm_DFI::WriteInit<double>(MPI_COMM_WORLD,   ///<MPI コミュニケータ
                                              dfi_fname,        ///<dfi ファイル名
                                              path,             ///<出力ディレクトリ
                                              prefix,           ///<ベースファイル名
                                              format,           ///<出力フォーマット
                                              out_gc,           ///<出力仮想セル数
                                              datatype,         ///<データ型
                                              nvari,            ///<データの変数の個数
                                              proc_fname,       ///<proc ファイル名
                                              GVoxel,           ///<計算空間全体のボクセルサイズ
                                              coord_X,          ///<X 座標データポインタ
                                              coord_Y,          ///<Y 座標データポインタ
                                              coord_Z,          ///<Z 座標データポインタ
                                              coord_file,       ///<座標ファイル名
                                              coord_filetype,   ///<座標ファイルの
                                                                /// データフォーマット
                                              coord_fileEndian, ///<座標ファイルの
                                                                /// エンディアンタイプ
                                              GDiv,             ///<領域分割数
                                              head,             ///<計算領域の開始位置
                                              tail,             ///<計算領域の終了位置
                                              hostname,         ///<ホスト名
                                              TimeSliceOnOff);  ///<タイムスライス出力オプション
//エラー処理
if( DFI_OUT == NULL )
{
  //エラーインスタンス失敗
  std::cerr << "Error Writeinit." << std::endl;
  delete [] coord_X;
  delete [] coord_Y;
  delete [] coord_Z;
  return CDM::E_CDM_ERROR;
}
//unit の登録
DFI_OUT->AddUnit("Length","NonDimensional",1.0);
DFI_OUT->AddUnit("Velocity","NonDimensional",1.0);
DFI_OUT->AddUnit("Pressure","NonDimensional",0.0,0.0,true);
//proc ファイル出力
DFI_OUT->WriteProcDfiFile(MPI_COMM_WORLD, ///<MPI コミュニケータ
                          false);         ///<ホスト名出力指示

//IBLANK 用配列の設定 (PLOT3D 形式)
size_t size_ib=(GVoxel[0]+2*out_gc)*(GVoxel[1]+2*out_gc)*(GVoxel[2]+2*out_gc);
int *iblank_v;
if (format == CDM::E_CDM_FMT_PLOT3D) {
  //配列 (iblank_v) のアロケート
  iblank_v = new int[size_ib];
  //配列 (iblank_v) に値をセット
  for(int i=0; i<size_ib; i++) {
```

```
    iblank_v[i] = 1;
  }
}

//格子ファイルの出力 (PLOT3D 形式)
ret = DFI_OUT->WriteGridFile(iblank_v); ///<plot3d の xyz ファイル出力 (IBLANK 値含む)
//エラー処理
if( ret != CDM::E_CDM_SUCCESS ) {
  //格子ファイルの出力失敗
  std::cerr << "Error WriteGridFile." << std::endl;
  delete [] iblank_v;
  return CDM::E_CDM_ERROR;
}

// IBLANK 用配列と座標データ配列のデアロケート
delete [] iblank_v; ///<IBLANK 用配列のポインタの削除
delete [] coord_X;  ///<座標データ配列ポインタの削除
delete [] coord_Y;
delete [] coord_Z;

unsigned step=10;  ///<出力ステップ番号
float r_time=0.0; ///<出力時間
float minmax[8];  ///<minmax
//minmax のゼロクリア
for(int i=0; i<8; i++) minmax[i]=0.0;

//変数名の登録
DFI_OUT->setVariableName(0,"u");
DFI_OUT->setVariableName(1,"v");
DFI_OUT->setVariableName(2,"w");
DFI_OUT->setVariableName(3,"P");

//TimeSlice の登録
DFI_OUT->AddTimeSlice(step,    ///<登録ステップ番号
                      r_time,  ///<登録時間
                      minmax,  ///<最小値，最大値
                      true,    ///<平均出力なし
                      0,       ///<平均をとったステップ数
                      0.0);    ///<平均をとった時刻

//index ファイル出力
ret = DFI_OUT->WriteIndexDfiFile();
//エラー処理
if( ret != CDM::E_CDM_SUCCESS ) {
  //index ファイルの出力失敗
  std::cerr << "Error WriteIndexDfiFile." << std::endl;
  return CDM::E_CDM_ERROR;
}

//配列のアロケート
float *d_v = new float[size*nvari];

//フィールドデータの出力
ret = DFI_OUT->WriteFieldDataFile(step,   ///<出力ステップ番号
                                  r_time, ///<出力時間
                                  GVoxel, ///<d_v の実ボクセル数
                                  nvari,  ///<d_v の成分数
                                  gsize,  ///<d_v の仮想セル数
                                  d_v,    ///<出力するフィールドデータポインタ
                                  true,   ///<平均出力なし
                                  0,      ///<平均をとったステップ数
                                  0.0);   ///<平均をとった時刻
//エラー処理
if( ret != CDM::E_CDM_SUCCESS ) {
  //フィールドデータの出力失敗
  std::cerr << "Error WriteFieldDataFile." << std::endl;
  delete [] d_v;
  delete DFI_OUT;
  return ret;
}
```

```
  //正常終了処理
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<配列ポインタの削除
  delete DFI_OUT; ///<出力インスタンスのポインタ削除
  return CDM::E_CDM_SUCCESS;
}
```

第 4 章

ステージングツール

この章では，CDMlib のステージングツールについて説明します．

4.1　ステージングツール

4.1.1　機能概要

ステージングツール frm(File RankMapper) は，大規模並列計算機で CDM ライブラリを使用する上で，各計算ノード (MPI ランク) 毎に必要なファイルを，ランク番号で命名したディレクトリにコピーする，ステージング対応用のプログラムです．

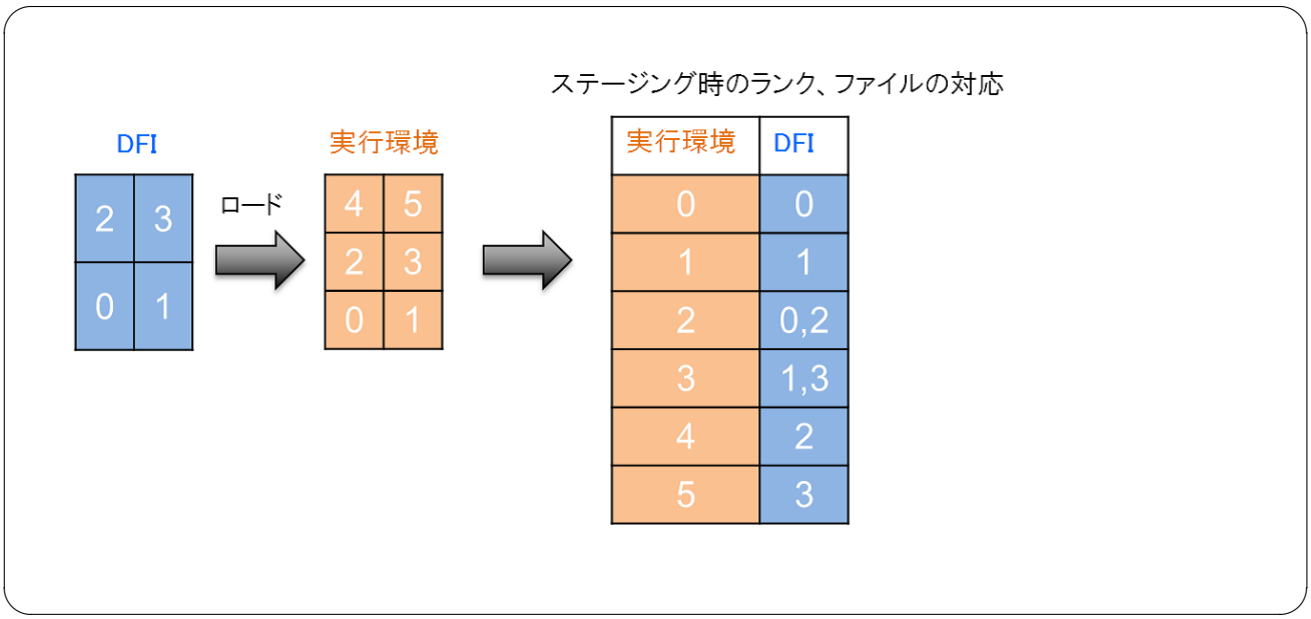


図 4.1　ステージング

4.1.2　ステージングツールのインストール

frm はログインノードで動作し，ステージング機構のあるマシンでの利用を対象としています．したがって，ステージング機構を持たない計算機環境には必要ありません．

ログインノードのコンパイル環境には，２通りあります．ひとつはネイティブ環境で，もう一つはクロスコンパイル環境です．京/FX，BlueGene はクロスコンパイル環境です．Intel 系のクラスタの場合には、ネイティブコンパイラの場合がほとんどでしょう．

frm は，ログインノードで`--with-frm` オプションでビルドします．コンパイル環境により，次のようなインストールとなります．

1. ネイティブコンパイル環境
   `--with-frm=yes` オプションにより，CDMlib のコンパイルと同時に frm をインストールします．`--with-frm` オプションのデフォルトは no です．ネイティブコンパイル環境の frm のビルドは並列動作しますが，通常ログインノードでは逐次で動かします．
2. クロスコンパイル環境
   CDMlib ビルド時に，`--with-frm=yes` オプションを指定しても frm はインストールされません．別途ログインノードのネイティブコンパイラを指定してビルドする必要があります．このとき，前もって TextParser もネイティブでコンパイルしておきます[*1]．クロスコンパイル環境での frm は逐次動作のみです．

[*1] 計算ノード用とは別のモジュールを作成する必要があります

CDM パッケージのビルドの詳細は 2.1.2 章,2.1.7 章を参照してください.

4.1.3　使用方法

frm はコマンドを実行して使用します.

コマンド引数

以下の引数を指定します. ([] は省略可能なオプション)

```
$ frm [-i proc.txt] [-f fconv.tp] [-n np] [-s stepNo] [-o outDir] DFIfile...
```

引数の説明

-i proc.txt (省略可)

これから計算するソルバーの領域分割情報が記述されたファイル名を指定します.

proc.txt にソルバーの Domain 情報が入ったファイル名 (TextParser 形式) を指定します.

領域分割情報が記述されたファイル proc.txt の仕様は 7.2.1 参照.

省略した場合は -f で FCONV 入力ファイルの指定が必須となります.

-i オプションとの併用はできません.

-f fconv.tp (省略可)

FCONV 用の入力ファイル名を指定します.

省略した場合は -i で領域分割情報ファイルが必須となります.

-f オプションとの併用はできません.

-n [np] (省略可)

FCONV の Mx1,MxM の実行並列数を指定します.

省略された場合は FCONV が並列数 1 の実行を指定した事になります.

-s stepNo (省略可)

振り分け対象とするステップ番号を指定します.

stepNo に対象とするステップ番号を指定します.

省略した場合は全ステップが対象となり，各ランク用のディレクトリにコピーされます.

(例) -s 100

DFIfile で指定したファイル中の 100 ステップのファイルについて各ランクのディレクトリにコピーされます.

-o outDir (省略可)

振り分け結果のコピー先のディレクトリ名を指定します.

outDir にディレクトリ名を指定します.

省略した場合はカレントディレクトリが出力先となります.

(例 1) -o hoge

カレントディレクトリに hoge/ディレクトリが生成され，そのディレクトリ配下に各ランク用の 000000/,000001/,... ディレクトリが生成されます.

(例 2) 省略時

カレントディレクトリに各ランク用の 000000/,000001/,... ディレクトリが生成されます.

DFIfile... (省略可)

振り分け対象とする DFI ファイル名を指定します.

複数の DFI ファイルを指定することが出来ます.

-i オプションで proc.txt ファイルが指定された場合は必須になります.

-f オプションとの併用はできません.

(例) vel.dfi prs.dfi を指定

vel.dfi，prs.dfi の両方の DFI ファイルが振り分け対象となり，同じ出力ディレクトリにコピーされます.

実行例

4 分割 (2,1,2) の結果を 8 分割 (2,2,2) でリスタートする例

・ ソルバーの Domain 情報格納ファイル (solvproc.txt)

```
Domain {
  GlobalVoxel=(64,64,64)
  GlobalDivision=(2,2,2)
  ActiveSubdomainFile=""
}
```

・ 振り分け対象の DFI ファイル

old ディレクトリ配下の prs.dfi,vel.dfi

実体の sph ファイルは SPH/ディレクトリに存在.

```
old/
  prs.dfi           <--DirectoryPath="SPH"
  vel.dfi           <--DirectoryPath="SPH"
  proc.dfi          <--prs.dfi, vel.dfi から参照
  SPH/
    prs_0000000000_id000000.sph
    prs_0000000000_id000001.sph
    prs_0000000000_id000002.sph
    prs_0000000000_id000003.sph
    prs_0000000100_id000000.sph
    prs_0000000100_id000001.sph
    prs_0000000100_id000002.sph
    prs_0000000100_id000003.sph
    vel_0000000000_id000000.sph
    vel_0000000000_id000001.sph
    vel_0000000000_id000002.sph
    vel_0000000000_id000003.sph
    vel_0000000100_id000000.sph
    vel_0000000100_id000001.sph
    vel_0000000100_id000002.sph
    vel_0000000100_id000003.sph
```

・ 振り分け対象ステップ番号

ステップ 100 のファイル

・出力先ディレクトリ
hoge/

・実行コマンド

```
$ frm -i solvproc.txt -s 100 -o hoge old/prs.dfi old/vel.dfi
```

・出力結果
hoge/ディレクトリが生成され，その配下に 6 桁のランク番号ディレクトリが生成されます．各ランク用ディレクトリ配下にそれぞれ必要なファイルがコピーされます．

```
hoge/000000/
  prs.dfi                          <--DirectoryPath="./"
  prs_0000000100_id000000.sph
  prs_proc.dfi                     <--proc.dfi からコピーされる
  vel.dfi                          <--DirectoryPath="./"
  vel_0000000100_id000000.sph
  vel_proc.dfi                     <--proc.dfi からコピーされる

hoge/000001/
  prs.dfi
  prs_0000000100_id000001.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000001.sph
  vel_proc.dfi

hoge/000002/
  prs.dfi
  prs_0000000100_id000000.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000000.sph
  vel_proc.dfi

hoge/000003/
  prs.dfi
  prs_0000000100_id000001.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000001.sph
  vel_proc.dfi

hoge/000004/
  prs.dfi
  prs_0000000100_id000002.sph
```

```
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000002.sph
  vel_proc.dfi

hoge/000005/
  prs.dfi
  prs_0000000100_id000003.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000003.sph
  vel_proc.dfi

hoge/000006/
  prs.dfi
  prs_0000000100_id000002.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000002.sph
  vel_proc.dfi

hoge/000007/
  prs.dfi
  prs_0000000100_id000003.sph
  prs_proc.dfi
  vel.dfi
  vel_0000000100_id000003.sph
  vel_proc.dfi
```

第 5 章

並列分散ファイルコンバータ

この章では，CDMlib の並列分散ファイルコンバータについて説明します．

5.1　並列分散ファイルコンバータ

5.1.1　機能概要

並列分散ファイルコンバータツール (FCONV) は，SPH/BOV 形式の分散ファイルを結合し，ファイル形式の変換を行い出力するプログラムで，MPI 並列での処理が可能です.

以下の機能有します.

- ファイル形式変換機能
- M 対 M データの変換機能
- M 対 1 データの変換機能
- M 対 N データの変換機能

ファイル形式変換機能

SPH，BOV，NetCDF4(/w HDF5) 形式のファイルから SPH，BOV，PLOT3D，AVS，VTK，NetCDF4(/w HDF5) 形式へ変換する機能です.

M 対 M データの変換機能

CDMlib を用いて出力した分散ファイルについて，分散状態はそのままでファイル形式のみ，指定形式に変換する機能です.

M 対 1 データの変換機能

CDMlib を用いて出力した分散ファイルについて，１つの結合ファイルとして指定形式に変換する機能です.

M 対 N データの変換機能

CDMlib を用いて出力した M 個の分散ファイルについて，N 個の分散ファイルとして指定形式に変換する機能です.

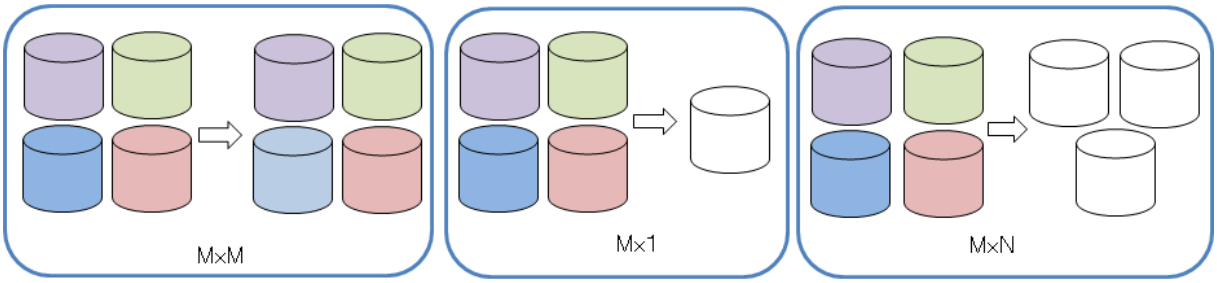


図 5.1　変換イメージ図

5.1.2　FCONV のインストール

FCONV は，CDM パッケージのビルド (configure，make，makeinstall) が行われるときに同時にビルドされ，configure スクリプト実行時の設定 `prefix` 配下の`${prefix}`/bin に `make install` 時にインストールされます．並列動作のみを想定しており，京や FX のログインノードでは動作しません．CDM パッケージのビルドは 2.1.2 章,2.1.7 章参照.

5.1.3　使用方法

FCONV はコマンドを実行して使用します.

コマンド引数

以下の引数を指定します.

```
$ fconv -f conv.tp [-l logfile] [-v]
```

引数の説明

-f conv.tp (必須)
　変換する DFI ファイル名，変換形式等のパラメータを指定する FCONV の入力ファイル名を指定します.
　入力ファイルの仕様は 7.2.2 参照.

-l logfile (省略可)
　ログをファイル出力させます.

-v (省略可)
　ログを画面出力させます.

実行例

3 分割 (1,1,3) の結果を間引き数 2 で 2 分割 (2,1,1) にコンバートする例

・ 入力ファイル (conv.tp)

```
ConvData{
  InputDFI[@]="prs.dfi"
  ConvType="MxN"
  OutputDivision=(2,1,1)
  OutputFormat="sph"
  OutputDataType="Float32"
  OutputFormatType="binary"
  OutputDir="hoge_sph"
  ThinningOut=2
}
```

・ 実行コマンド

```
$ mpirun -np 2 fconv -f conv.tp
```

・ 変換結果
　図 5.2, 5.3 を参照.

5.1.4　出力形式

ファイル形式毎に可能な出力形式は表 5.1 のとおりになります.

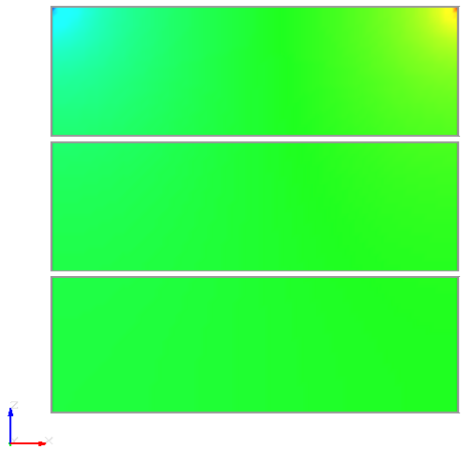


図 5.2　変換前

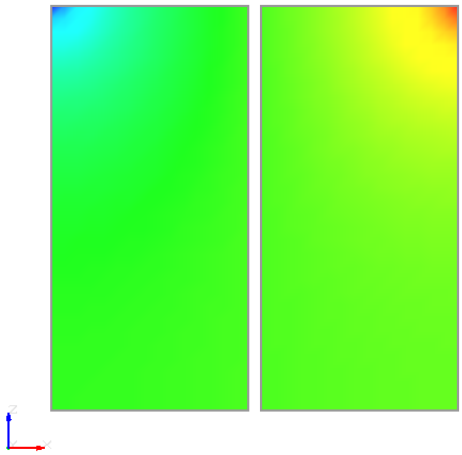


図 5.3　変換後

表 5.1　ファイルフォーマット毎の出力形式

	OutputFormatType		
	ascii	binary	Fortran_Binary
sph	-	◎	-
bov	-	◎	-
avs	×	◎	-
plot3d	○	◎	○
vtk	○	◎	-
NetCDF4(/w HDF5)	-	◎	-

-：未対応
○：対応
◎：デフォルト
×：FCONV では未対応

5.1.5　ファイルフォーマット毎の対応データ型

ファイル形式毎に可能なデータ形式及び指定形式は表 5.2 のとおりになります.

表 5.2　ファイルフォーマット毎のデータ型

- : 未対応 ○ : 対応 × : 対象外 （※）

	OutputDataType	bov ヘッダ	sph	bov	avs	plot3d	vtk	NetCDF4(/w HDF5)
bit	-	-	-	-	-	-	×	-
unsigned char	UInt8	UInt8	-	○	-	-	○	○
char(byte)	Int8	BYTE	-	○	○	-	○	○
unsigned short	UInt16	UInt16	-	○	-	-	○	○
short	Int16	Int16	-	○	○	-	○	○
unsigned int	UInt32	UInt32	-	○	-	-	○	○
int	Int32	INT	-	○	○	-	○	○
unsigned long	UInt64	UInt64	-	○	-	-	○	○
long	Int64	Int64	-	○	-	-	○	○
float	Float32	FLOAT	○	○	○	○	○	○
double	Float64	DOUBLE	○	○	○	○	○	○

（※）フォーマットとしては存在するが変換元の sph,bov が対応していない型のため，対象外とする．

5.1.6　ファイルフォーマット毎の対応配列型

ファイル形式毎に可能な配列形状は以下になります.

表 5.3　ファイルフォーマット毎の配列形状

ファイルフォーマット	成分数
sph	nijk(n=1or3)
bov	nijk,ijkn(n=任意)
avs	nijk(n=任意)
plot3d	ijkn(n=任意)
vtk	nijk(n=任意)
NetCDF4(/w HDF5)	ijkn(n=任意)

5.1.7　定義点

ファイル形式毎に出力される定義点の位置が図心，格子点と異なります.
ファイル形式毎の定義点と図心から格子点への補間方法を説明します.

ファイル形式毎の定義点

ファイル形式毎の定義点は以下になります.

表 5.4　ファイルフォーマット毎の定義点

ファイルフォーマット	定義点
sph	図心（セルセンター）
bov	図心（セルセンター）
avs	格子点
plot3d	格子点
vtk	格子点
NetCDF4(/w HDF5)	図心（セルセンター）

格子点への補間

図心から格子点への補間方法は仮想セルの有無で以下のようになります.

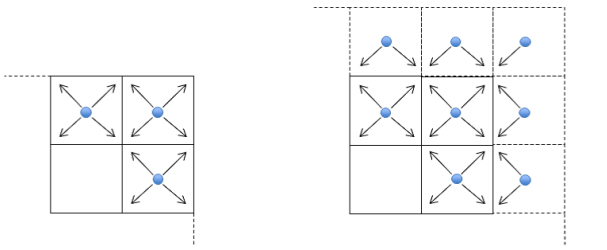


図 5.4　格子点への補間

5.1.8　ファイルフォーマット毎の出力ファイル

ファイル形式毎に出力されるファイルおよびファイルの拡張子は以下になります.

表 5.5　ファイルフォーマット毎の出力ファイル

ファイルフォーマット	フィールドデータ	dfi	ヘッダファイル	その他（座標値)
sph	*.sph	*.dfi	-	-
bov	*.dat	*.dfi	*.bov	-
avs	*.dat	-	*.fld	*.cod
plot3d	*.func	-	-	*.xyz
vtk	*.vtk	-	-	-
NetCDF4(/w HDF5)	*.nc	*.dfi	-	-

5.1.9　間引き

FCONV では，間引き処理をして出力する事が出来ます．以下に間引き数を 2 とした例で間引きの処理方法を説明します.

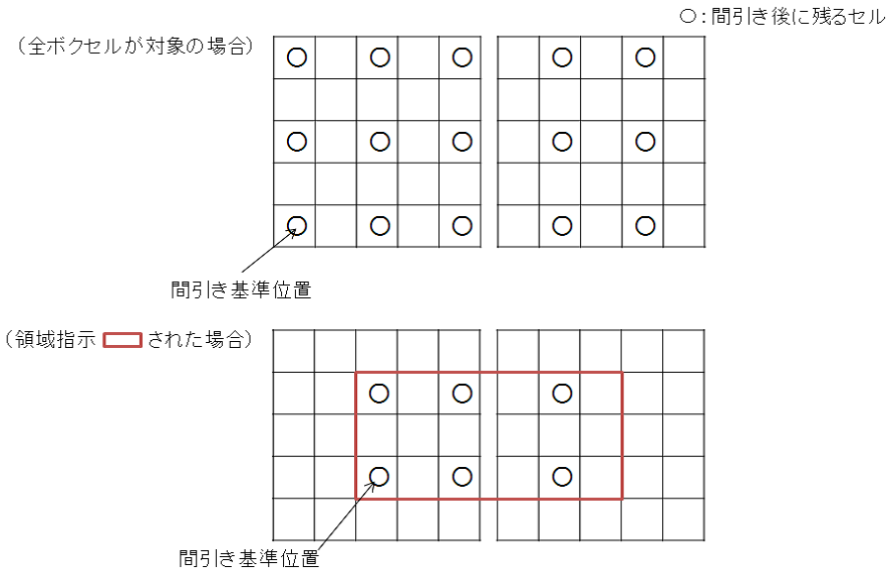


図 5.5　間引き数を 2 とした例

5.1.10　ファイル割振り

FCONV では，ファイルの割振り方法として，step 基準と rank 基準があります.
以下にそれぞれの方法について説明します.

step 基準

1. 入力 DFI 毎にステップ番号でソートしたリストを作る
2. ソートしたリストを FCONV の並列数で均等に分散させる
3. FCONV の並列数で割り切れない場合は FCONV のランクの若い順に担当するステップ数を増やす
4. FCONV のあるランクに振り分けられたステップ番号の全ての DFI のランクファイルをそのランクで担当する

(例)prs,vel が 4 ステップ 3 分割で出力された結果を 5 並列でコンバートする場合

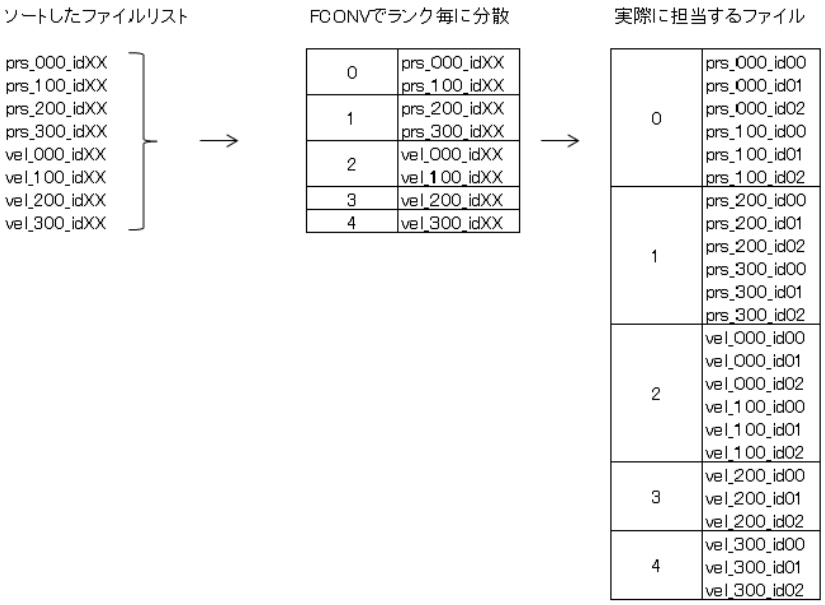


図 5.6　step 基準の例

(※ 1) DFI 数×ステップ数の公約数で処理させると効率が良い
(この例の場合は 8,4,2 並列)
(※ 2) M 対 1 は必ず step 基準となる

rank 基準

1. 入力 DFI 毎に DFI ランク番号でソートしたリストを作る
2. ソートしたリストを FCONV 並列数で均等に分散させる
3. FCONV 並列数で割り切れない場合は FCONV ランクの若い順に担当させる
4. 同一の DFI ランクの全ステップデータを同一 FCONV ランクが担当する

(例)prs,vel が 4 ステップ 3 分割で出力された結果を 5 並列でコンバートする場合

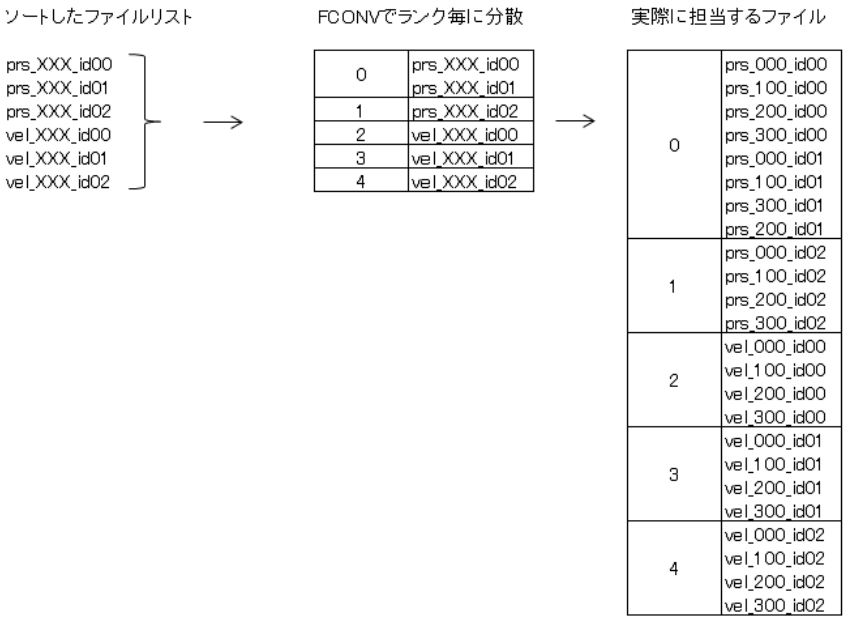


図 5.7　rank 基準の例

(※ 1) DFI ×ランク数の公約数で処理させると効率が良い
　　(この例の場合は 6,3,2 並列)
(※ 2) ステップ数が少なく，ランク数が多い場合，実行並列数を上げたいときに有効

5.1.11　NetCDF4(/w HDF5) に対応について

NetCDF4(/w HDF5) 形式では，1 ファイル内に全時系列データを保持している場合に，DFI ファイルに File-Info/FieldFilenameFormat=”rank”の記述をすることで対応が可能ですが，FCONV から NetCDF4(/w HDF5) に出力する場合は，”step_rank”もしくは”rank_step”のいずれかのみに対応します.

変換元として NetCDF4 ファイルを使用する場合，その入力データについては”rank”を使用することが可能です.

第 6 章

NetCDF4 用 DFI 生成ツール

この章では，NetCDF4(/w HDF5) ファイル群から CDMlib の DFI ファイルを生成するツールについて説明します.

6.1　NetCDF4 用 DFI 生成ツール

6.1.1　機能概要

NetCDF4 用 DFI 生成ツール (netcdf2dfi) は，メタ情報 (DFI ファイル) が存在しない NetCDF4(/w HDF5) ファイル群から自動的に DFI ファイルを生成するツールです.

※ CDMlib ビルド時に NetCDF4(/w HDF5) を有効にしていない場合，本ツールはインストールされません．(2.1.8 章を参照のこと)

6.1.2　NetCDF4 用 DFI 生成ツールのインストール

NetCDF4 用 DFI 生成ツールは NetCEF4 ライブラリを使用しているため，CDMlib ビルド時に NetCDF4(/w HDF5) を有効にしていない場合，本ツールはインストールされません．詳細は 2.1.8 章を参照してください.

6.1.3　使用方法

netcdf2dfi はコマンドを実行して使用します.

コマンド引数

以下の引数を指定します．([] は省略可能なオプション)

```
$ netcdf2dfi input.tp
```

引数の説明

input.tp

netcdf2dfi 用の入力ファイル名を指定します.

入力ファイルのフォーマットの詳細については，7.2.3 章を参照してください.

実行例

36 分割 (6,6,1) の NetCDF4 ファイル群から DFI ファイルを生成する例

1. DFI ファイルのみを生成する場合
   - 入力ファイル (input.tp)

```
NetCDF2DFI
{
  num_ncfiles         = 36
  DirectoryPath       = "./data"
  Prefix              = "history_d01"
  FieldFilenameFormat = "rank"
  RankNoPrefix        = ".pe"
  GuideCell           = 0
  Variable[@] {name = "U"}
  Variable[@] {name = "V"}
```

```
  Variable[@] {name = "W"}
  Variable[@] {name = "PRES"}

  index_fname         = "index.dfi"
  proc_fname          = "proc.dfi"

  VariableName
  {
    x    = "x"
    y    = "y"
    z    = "z"
    time = "time"
  }
}
```

・入力ファイル構成
入力ファイルのファイル/ディレクトリ構成は以下の通りです.

```
input.tp
data/
  history_d01.pe000000.nc
  history_d01.pe000001.nc
     :
  history_d01.pe000035.nc
```

・実行コマンド

```
$ netcdf2dfi input.tp
```

・出力結果
カレントディレクトリに index.dfi と proc.dfi が出力されます.

```
index.dfi
proc.dfi
data/                        <--dfi からは元の NetCDF4 ファイルを参照
  history_d01.pe000000.nc
  history_d01.pe000001.nc
     :
  history_d01.pe000035.nc
```

2. DFI ファイル生成と同時に，必要な成分のみを抽出した NetCDF4 ファイルを変換出力する場合
・入力ファイル (input.tp)

```
NetCDF2DFI
{
  num_ncfiles         = 36
  DirectoryPath       = "./data"
  Prefix              = "history_d01"
  FieldFilenameFormat = "rank"
```

```
  RankNoPrefix        = ".pe"
  GuideCell           = 0
  Variable[@] {name = "U"}
  Variable[@] {name = "V"}
  Variable[@] {name = "W"}
  Variable[@] {name = "PRES"}

  //OutputDirectoryPath を指定すると，このディレクトリに Variable で指定した
  //成分のみを抽出した NetCDF4 ファイルが生成される
  OutputDirectoryPath = "./output"

  index_fname         = "index.dfi"
  proc_fname          = "proc.dfi"

  VariableName
  {
    x    = "x"
    y    = "y"
    z    = "z"
    time = "time"
  }
}
```

・入力ファイル構成

入力ファイルのファイル/ディレクトリ構成は以下の通りです.

```
input.tp
data/
  history_d01.pe000000.nc
  history_d01.pe000001.nc
     :
  history_d01.pe000035.nc
```

・実行コマンド

```
$ netcdf2dfi input.tp
```

・出力結果

カレントディレクトリに index.dfi と proc.dfi が出力されます.

また，成分を抽出した NetCDF4 ファイルが output ディレクトリに出力されます.

```
index.dfi
proc.dfi
output/                    <--dfi からは成分を抽出した NetCDF4 ファイルを参照
  history_d01.pe000000.nc
  history_d01.pe000001.nc
     :
  history_d01.pe000035.nc
```

第 7 章

ファイル仕様

CDMlib で使用しているファイルの仕様について説明します.

7.1　ファイル仕様

7.1.1　インデックスファイル（index.dfi）仕様

index.dfi ファイルはファイル情報（FileInfo），ファイルパス情報（FilePath），単位系（Unit），時系列データ（TimeSlice）の４つのブロックで構成されています．

以下に，index.dfi ファイルの仕様とサンプルをブロック毎に示します．

ファイル情報（FileInfo）の仕様

```
FileInfo
{
  DFIType             = "Cartesian"    // dfi の種別 (※ 1)
  DirectoryPath       = "./"           // フィールドデータの存在するディレクトリ (※ 2)
  TimeSliceDirectory = "off"           // 時刻毎のディレクトリ作成オプション
  Prefix              = "field"        // ベースファイル名 (※ 3)
  FileFormat          = "bov"          // ファイル形式 (※ 4)
  FieldFilenameFormat= "step_rank"     // ファイル命名順 (※ 3)
  GuideCell           = 0              // 仮想セル数
  DataType            = "Float32"      // データタイプ (※ 5)
  Endian              = "little"       // データのエンディアン (※ 6)
  NumVariables        = 4              // 変数の個数
  Variable[@]  {name = "u"}            // 変数名 (※ 7)
  Variable[@]  {name = "v"}            //
  Variable[@]  {name = "w"}            //
  Variable[@]  {name = "P"}            //
  RankNoPrefix        = ".pe"          // フィールドファイル名のランク番号文字列 (※ 8)
}
```

(※ 1) ”Cartesian”,”Non_Uniform_Cartesian”
(※ 2) index.dfi ファイルからの相対パス，もしくは絶対パス
(※ 3) ファイル名
　　step_rank : [Prefix]_[ステップ番号:10 桁]_id[RankID:6 桁].[ext]
　　rank_step : [Prefix]_id[RankID:6 桁]_[ステップ番号:10 桁].[ext]
　　rank : [Prefix]_id[RankID:6 桁].[ext] ※ NetCDF4(/w HDF5) 形式で時系列 1 ファイルの場合のみ
　　逐次時 : [Prefix]_[ステップ番号:10 桁].[ext]
　　(拡張子 ext は，BOV 形式⇒ bov，PLOT3D 形式⇒ fun，SPH 形式⇒ sph，NetCDF4(/w HDF5) 形式⇒ nc)
(※ 4) bov,plot3d,sph,netcdf4
(※ 5) Int8,UInt8,Int16,Uint16,Int32,Uint32,Int64,Uint64,Float32,Float64
(※ 6) little,big, 省略時:実行プラットフォームと同じ
(※ 7) フィールドデータと登録順に変数の個数分だけ登録
(※ 8) フィールドファイルのランク番号前の文字列を指定
　　上記の例の場合，[Prefix]_[ステップ番号:10 桁].pe[RankID:6 桁].[ext] のように，ランク番号前の文字列を指定可能となる．(省略時は”_id”)

ファイルパス（FilePath）の仕様

```
FilePath
{
  Process             = "proc.dfi"     // proc ファイル名 (※ 1)
}
```

(※ 1) index.dfi ファイルからの相対パス，もしくは絶対パス

単位系（UnitList）の仕様

```
UnitList
{
  Length {
    Unit              = "M"           // (NonDimensional, m, cm, mm)
    Reference         = 1.0           // 規格化に用いた長さスケール
  }
  Velocity {
    Unit              = "m/s"         // (NonDimensional, m/s)
    Reference         = 3.4           // 代表速度 (m/s)
  }
  Pressure {
    Unit              = "Pa"          // (NonDimensional, Pa)
    Reference         = 0.0           // 基準圧力 (Pa)
    Difference        = 510.0         // 圧力差 (Pa)
  }
  Temperature {
    Unit              = "C"           // (NonDimensional, C, K)
    Reference         = 10.0          // 基準温度 (C)
    Difference        = 510.0         // 温度差 (C)
  }
}
```

時系列データ（TimeSlice）の仕様

```
TimeSlice
{
  Slice{@} {          // ファイル出力ステップ数分
    Step              = 0             // 出力ステップ
    Time              = 0.0           // 出力時刻
    AverageTime       = 0.0           // 平均時間（必要に応じて出力）
    AverageStep       = 0             // 平均化したステップ数（必要に応じて出力）
    VectorMinMax {    // ベクトル合成値の min/max 値 (※ 1)
      Min             = 0.0           // 最小値
      Max             = 0.0           // 最大値
    }
    MinMax{@} {                       // 変数の min/max 値 (※ 2)
      Min             = 0.0           // 1 番目の変数の最小値
      Max             = 0.0           // 1 番目の変数の最大値
    }
    MinMax{@} {
      Min             = 0.0           // 2 番目の変数の最小値
      Max             = 0.0           // 2 番目の変数の最大値
    }
    MinMax{@} {
      Min             = 0.0           // 3 番目の変数の最小値
      Max             = 0.0           // 3 番目の変数の最大値
    }
    ・・・任意のアノテーション追加可能
  }
  Slice{@} {
    :
    :
  }
}
```

(※ 1) SPH 形式出力で変数の個数が 3 の場合のみ記述
(※ 2) FileInfo で登録された変数順に記述

NetCDF4 用のデータ（NetCDF4）の仕様 (※ 1)

```
NetCDF4
{
  VariableName     // NetCDF4 ファイル内の必須 dimension，variable の名称 (※ 2)
  {
    x = "x"        // x 格子数 (dimension)，格子図心 X 座標値 (variable) の名称
    y = "y"        // y 格子数 (dimension)，格子図心 Y 座標値 (variable) の名称
    z = "z"        // z 格子数 (dimension)，格子図心 Z 座標値 (variable) の名称
    time = "time"  // ステップ数 (dimension)，各ステップの時刻 (variable) の名称
  }
}
```

(※ 1) ファイル形式が NetCDF4(/w HDF5) のときのみ有効
(※ 2) 省略時は”x”，”y”，”z”，”time”を使用

7.1.2 プロセス情報ファイル（proc.dfi）仕様

proc.dfi ファイルはドメイン情報（Domain），並列情報（MPI），プロセス情報（Process）の 3 つのブロックで構成されています.

以下に，proc.dfi ファイルの仕様とサンプルをブロック毎に示します.

ドメイン情報（Domain）の仕様

```
Domain
{
  GlobalOrigin        = (-3.00,-3.00,-3.00)   // 計算空間の起点座標
  GlobalRegion        = ( 6.00, 6.00, 6.00)   // 計算空間の各軸方向の長さ
  GlobalVoxel         = ( 64, 64, 64 )        // 計算領域全体のボクセル数
  GlobalDivision      = ( 1, 1, 1)            // 計算領域全体の分割数
  ActiveSubdomainFile = "subdomain.dat"       // ActiveSubdomain ファイル名 (※ 1)
  CoordinateFile      = "coord.crd"           // 座標ファイル名 (※ 1, ※ 2, ※ 3)
  CoordinateFileType  = "ascii"               // 座標ファイルのファイルタイプ (※ 2, ※ 4)
  CoordinateFilePrecision = "Float64"         // 座標ファイルのデータ精度 (※ 2, ※ 5)
  CoordinateFileEndian = "little"             // 座標ファイルのエンディアンタイプ (※ 2, ※ 6)

}
```

(※ 1) index.dfi ファイルからの絶対パス，もしくは相対パス
(※ 2) 直交不等間隔格子の場合に記述
(※ 3) 座標ファイルの詳細は，7.1.5 章「座標ファイルの仕様」を参照
(※ 4) ascii,binary
(※ 5) Float32,Float64 (binary の場合のみ)
(※ 6) little,big (binary の場合のみ)

並列情報（MPI）の仕様

```
MPI
{
  NumberOfRank        = 128                  // プロセス数
  NumberOfGroup       = 1                    // グループ数
}
```

プロセス情報（Process）の仕様

```
Process
{
  Rank{@} {   // NumberOfRank 個
    ID        = 0                  // ランク番号
    HostName  = "Strontium.local" // ホストノード名（必須ではない）
    VoxelSize = ( 64, 64, 64 )    // ボクセルサイズ
    HeadIndex = ( 1, 1, 1 )       // 始点インデックス，グローバルで (1,1,1) からスタート
    TailIndex = ( 64, 64, 64 )    // 終点インデックス，グローバルで (64, 64, 64) が終端
  }
  Rank{@} {
    :
    :
  }
}
```

7.1.3　フィールドデータファイルの仕様

以下に，CDM で対応しているフィールドデータファイルの形式を示します．

SPH 形式

SPH データ（V-Sphere Simple Voxel データ）ファイルは，Solver フレームワーク V-Sphere の計算結果を格納するバイナリ形式のファイルです．SPH データファイルは，１ファイルに各レコードが順に１レコードずつ記述されています．(表 7.1 を参照)

表 7.1　SPH ファイルレコード形式

レコード名	意味
データ属性レコード	データの属性を記述する （単精度 or 倍精度） （スカラー or ベクトル）
ボクセルサイズレコード	ボクセルサイズを記述する
原点座標レコード	原点座標を記述する
ボクセルピッチレコード	ボクセルピッチを記述する
時刻レコード	タイムステップと時刻を記述する
データレコード	データを記述する

- **データ属性レコード**
  データ属性を記述するレコードで，データ種別とデータ型を指します．データ種別は記述されるデータがスカラーなのかベクトルなのかを区別します．データ型は記述されるデータの精度（単精度 or 倍精度）を区別します．

表 7.2　データ属性レコード

名称	表現	サイズ	説明
Size	整数	4 bytes	レコード長（＝ 8）（※ 1）
svType	整数	4 bytes	データ種別フラグ（※ 2）
dType	整数	4 bytes	データ型フラグ（※ 3）
Size	整数	4 bytes	レコード長（＝ 8）（※ 1）

（※ 1 ）**レコード長**
Fortran の書式なし出力の形式に合わせた項目で，データレコード長のバイト数でデータをはさむ形式をとります．

（※ 2 ）**データ種別フラグ**
スカラーデータかベクトルデータかを判断するフラグです．表 7.3 の値をとります．

（※ 3 ）**データ型フラグ**
記述されるデータの型（単精度/倍精度）を判断するフラグです． 表 7.4 の値をとります．

表 7.3　データ種別フラグ

データ種別	svType の値
スカラーデータ	1
ベクトルデータ	2

表 7.4　データ型フラグ

データ型	dType の値
単精度	1
倍精度	2

- **ボクセルサイズレコード**

  ボクセルサイズ（計算空間のボクセル数）を記述するレコードです.

表 7.5　ボクセルサイズレコード

名称	表現	サイズ	説明
Size	整数	4 bytes	レコード長（= 12 or 24）（※ 2）
IMAX	整数	4 or 8 bytes （※ 1）	I 方向ボクセル数
JMAX	整数	4 or 8 bytes （※ 1）	J 方向ボクセル数
KMAX	整数	4 or 8 bytes （※ 1）	K 方向ボクセル数
Size	整数	4 bytes	レコード長（= 12 or 24）（※ 2）

（※ 1 ）**データ型フラグ（dType）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType = 1）: 4 bytes

倍精度の場合（dType = 2）: 8 bytes

（※ 2 ）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType = 1）: 12 bytes

倍精度の場合（dType = 2）: 24 bytes

- **原点座標レコード**

  計算空間の原点座標を記述するレコードです.

表 7.6　原点座標レコード

名称	表現	サイズ	説明
Size	整数	4 bytes	レコード長（= 12 or 24）（※ 2）
XORG	整数	4 or 8 bytes （※ 1）	X 軸方向原点座標
YORG	整数	4 or 8 bytes （※ 1）	Y 軸方向原点座標
ZORG	整数	4 or 8 bytes （※ 1）	Z 軸方向原点座標
Size	整数	4 bytes	レコード長（= 12 or 24）（※ 2）

（※ 1 ）**データ型フラグ（(dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType = 1）: 4 bytes

倍精度の場合（dType ＝ 2）：8 bytes

（※ 2 ）**データ型フラグ（(dtype）の値（単精度 or 倍精度）により異なります．**

単精度の場合（dType ＝ 1）：12 bytes

倍精度の場合（dType ＝ 2）：24 bytes

- **ボクセルピッチレコード**

  1 ボクセルのピッチを記述するレコードです.

表 7.7　ボクセルピッチレコード

名称	表現	サイズ	説明
Size	整数	4 bytes	レコード長（＝ 12 or 24）（※ 2）
XPITCH	整数	4 or 8 bytes （※ 1）	X 方向ボクセルピッチ
YPITCH	整数	4 or 8 bytes （※ 1）	Y 方向ボクセルピッチ
ZPITCH	整数	4 or 8 bytes （※ 1）	Z 方向ボクセルピッチ
Size	整数	4 bytes	レコード長（＝ 12 or 24）（※ 2）

（※ 1 ）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります．**

単精度の場合（dType ＝ 1）：4 bytes

倍精度の場合（dType ＝ 2）：8 bytes

（※ 2 ）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります．**

単精度の場合（dType ＝ 1）：12 bytes

倍精度の場合（dType ＝ 2）：24 bytes

- **時刻レコード**

  タイムステップと時刻を記述するレコードです.

表 7.8　時刻レコード

名称	表現	サイズ	説明
Size	整数	4 bytes	レコード長（＝ 8 or 12）（※ 2）
STEP	整数	4 or 8 bytes （※ 1）	タイムステップ
TIME	整数	4 or 8 bytes （※ 1）	時刻
Size	整数	4 bytes	レコード長（＝ 8 or 12）（※ 2）

（※ 1 ）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります．**

単精度の場合（dType ＝ 1）：4 bytes

倍精度の場合（dType ＝ 2）：8 bytes

（※ 2 ）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります．**

単精度の場合（dType ＝ 1）：8 bytes

倍精度の場合（dType ＝ 2）：16 bytes

- **データレコード**

  データを記述するレコードです.

  – スカラーデータの場合（svType ＝ 1 のとき）

名称	表現	サイズ（※ 1）	説明
Size	整数	4 bytes	レコード長（※ 2）
DATA(0,0,0)	実数	4 or 8 bytes	格子点 (0,0,0) のデータ値
DATA(1,0,0)	実数	4 or 8 bytes	格子点 (1,0,0) のデータ値
DATA(2,0,0)	実数	4 or 8 bytes	格子点 (2,0,0) のデータ値
・・・			
DATA(IMAX-1,JMAX-1,KMAX-1)	実数	4 or 8 bytes	格子点 (IMAX-1,JMAX-1,Kmax-1) のデータ値
Size	整数	4 bytes	レコード長（※ 2）

（※ 1）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType ＝ 1）: 4 bytes

倍精度の場合（dType ＝ 2）: 8 bytes

（※ 2）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType ＝ 1）: IMAX × JMAX × KMAX × 4 (bytes)

倍精度の場合（dType ＝ 2）: IMAX × JMAX × KMAX × 8 (bytes)

  – ベクトルデータの場合（svType ＝ 2 のとき）

名称	表現	サイズ（※ 1）	説明
Size	整数	4 bytes	レコード長（※ 2）
U(0,0,0)	実数	4 or 8 bytes	格子点 (0,0,0) の U データ値
V(0,0,0)	実数	4 or 8 bytes	格子点 (0,0,0) の V データ値
W(0,0,0)	実数	4 or 8 bytes	格子点 (0,0,0) の W データ値
U(1,0,0)	実数	4 or 8 bytes	格子点 (1,0,0) の U データ値
V(1,0,0)	実数	4 or 8 bytes	格子点 (1,0,0) の V データ値
W(1,0,0)	実数	4 or 8 bytes	格子点 (1,0,0) の W データ値
・・・			
U(IMAX-1,JMAX-1,KMAX-1)	実数	4 or 8 bytes	格子点 (IMAX-1,JMAX-1,Kmax-1) の U データ値
V(IMAX-1,JMAX-1,KMAX-1)	実数	4 or 8 bytes	格子点 (IMAX-1,JMAX-1,Kmax-1) の V データ値
W(IMAX-1,JMAX-1,KMAX-1)	実数	4 or 8 bytes	格子点 (IMAX-1,JMAX-1,Kmax-1) の W データ値
Size	整数	4 bytes	レコード長（※ 2）

（※ 1）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType ＝ 1）: 4 bytes

倍精度の場合（dType ＝ 2）: 8 bytes

（※ 2）**データ型フラグ（dtype）の値（単精度 or 倍精度）により異なります.**

単精度の場合（dType ＝ 1）: IMAX × JMAX × KMAX × 4 × 3（bytes）

倍精度の場合（dType ＝ 2）: IMAX × JMAX × KMAX × 8 × 3（bytes）

BOV 形式

可視化ソフトウエア「VisIt」の Brick of Values 形式ファイル
データ配列のみが単純に格納されています．(表 7.9,7.10 を参照)

表 7.9　（例 1）ijkn 配列 v(i,j,k,n) の記述例

配列要素	説明
v(0,0,0,0)	格子点 (0,0,0) の成分 0 のデータ値
v(1,0,0,0)	格子点 (1,0,0) の成分 0 のデータ値
・・・	
v(imax-1,jmax-1,kmax-1,0)	格子点 (imax-1,jmax-1,kmax-1) の成分 0 のデータ値
v(0,0,0,1)	格子点 (0,0,0) の成分 1 のデータ値
・・・	
v(imax-1,jmax-1,kmax-1,n-1)	格子点 (imax-1,jmax-1,kmax-1) の成分 n-1 のデータ値

表 7.10　（例 2）nijk 配列 v(n,i,j,k) の記述例

配列要素	説明
v(0,0,0,0)	格子点 (0,0,0) の成分 0 のデータ値
v(1,0,0,0)	格子点 (0,0,0) の成分 1 のデータ値
・・・	
v(n-1,0,0,0)	格子点 (0,0,0) の成分 n-1 のデータ値
v(0,0,0,1)	格子点 (0,0,1) の成分 0 のデータ値
・・・	
v(n-1,imax-1,jmax-1,kmax-1)	格子点 (imax-1,jmax-1,kmax-1) の成分 n-1 のデータ値

PLOT3D 形式

PLOT3D のデータ形式は，NASA で開発されたソフトウェア「PLOT3D」で適用されていたデータ形式であり，計算結果データや格子データを格納するための標準的なデータ形式です．PLOT3D には，xyz File，Q File，Function File の三つのデータ形式があります．

本ライブラリでは，計算結果データを Function File に，格子データを xyz File に格納します．これらのデータは，Fortran Binary 形式で格納されています．

Function File には，データ配列 v(i,j,k,n) が表 7.11 の順で格納されています．

表 7.11　Function File に格納されるデータ配列 v(i,j,k,n)

配列要素	説明
v(0,0,0,0)	格子点 (0,0,0) の変数 0 のデータ値
v(1,0,0,0)	格子点 (1,0,0) の変数 0 のデータ値
・・・	
v(imax-1,jmax-1,kmax-1,0)	格子点 (imax-1,jmax-1,kmax-1) の変数 0 のデータ値
v(0,0,0,1)	格子点 (0,0,0) の変数 1 のデータ値
・・・	
v(imax-1,jmax-1,kmax-1,n-1)	格子点 (imax-1,jmax-1,kmax-1) の変数 n-1 のデータ値

xyz File には，格子点 (i,j,k) の x,y,z 座標が格納されています．また，xyz File に IBLANK の情報を追加することもできます．xyz File に格納されているデータ内容を表 7.12 に示します．IBLANK に格納される値とその意味については，表 7.13 に示します．

表 7.12　xyz File に格納されるデータ内容

データ内容
格子点 (0,0,0) の x 座標
格子点 (1,0,0) の x 座標
・・・
格子点 (imax-1,jmax-1,kmax-1) の x 座標
格子点 (0,0,0) の y 座標
格子点 (1,0,0) の y 座標
・・・
格子点 (imax-1,jmax-1,kmax-1) の y 座標
格子点 (0,0,0) の z 座標
格子点 (1,0,0) の z 座標
・・・
格子点 (imax-1,jmax-1,kmax-1) の z 座標
格子点 (0,0,0) の IBLANK の値
格子点 (1,0,0) の IBLANK の値
・・・
格子点 (imax-1,jmax-1,kmax-1) の IBLANK の値

表 7.13　IBLANK に格納される値とその意味

IBLANK の値	値の意味
0	非計算格子
1	計算格子（デフォルト）
2	壁面格子

NetCDF4(/w HDF5) 形式

NetCDF 形式は，Unidata Program Center で開発された「NetCDF ライブラリ」のファイル形式であり，配列指向型のデータを自己記述的でかつポータブルなフォーマットとして作成・アクセス・共有することを目的としたデータ形式です．また，HDF5 は NCSA で開発されたフリーな階層データフォーマットです．

CDMlib では，NetCDF4 with HDF5 形式についてサポートしますが，本形式は自由な記述が可能なため，以下の記述内容についてのみサポートすることとしています．

・対応するデータ型

CDMlib では，以下の NetCDF データ型に対応します．

表 7.14　dimension（必須）の項目

NetCDF データ型	対応する CDMlib データ型
NC_BYTE	CDM_INT8
NC_SHORT	CDM_INT16
NC_INT	CDM_INT32
NC_INT64	CDM_INT64
NC_UBYTE	CDM_UINT8
NC_USHORT	CDM_UINT16
NC_UINT	CDM_UINT32
NC_UINT64	CDM_UINT64
NC_FLOAT	CDM_FLOAT32
NC_DOUBLE	CDM_FLOAT64

・dimension（必須）

dimension として以下の次元が記述されている必要があります．なお，dimension 名は DFI ファイルの

表 7.15　dimension（必須）の項目

dimension 名	内容
x	x 方向格子数
y	y 方向格子数
z	z 方向格子数
time	ステップ数

NetCDF4/VariableName の記述により変更することが可能です．

・variable（必須）

variable として，以下の配列が記述されている必要があります．

表 7.16　variable（必須）の項目

variable 名	配列サイズ (dimension)	データ型	内容
x	(x)	NC_FLOAT or NC_DOUBLE	x 方向各格子の図心（セルセンター）X 座標値
y	(y)	NC_FLOAT or NC_DOUBLE	y 方向各格子の図心（セルセンター）Y 座標値
z	(z)	NC_FLOAT or NC_DOUBLE	z 方向各格子の図心（セルセンター）Z 座標値
time	(time)	NC_FLOAT or NC_DOUBLE	各ステップの時刻

・ variable（任意）
速度場，圧力場等の物理量は任意名の variable として記述します．variable は必ず 1 成分のデータとして記述する必要があります．以下の例は，速度場（U,V,W の 3 成分），圧力場（PRES の 1 成分）の場合の記述内容になります．

表 7.17　variable（必須）の項目

variable 名	配列サイズ (dimension)	データ型	内容
U	(time,z,y,x)	表 7.14 で対応するデータ型	速度場 U 成分
V	(time,z,y,x)	表 7.14 で対応するデータ型	速度場 V 成分
W	(time,z,y,x)	表 7.14 で対応するデータ型	速度場 W 成分
PRES	(time,z,y,x)	表 7.14 で対応するデータ型	圧力場

以下に，対応するファイル内容を示します．

※ NetCDF4 ライブラリ付属の ncdump ツールの出力結果です．
以下のコマンドの出力結果になります．

```
$ ncdump -h test.pe000000.nc
```

```
netcdf test.pe000000 {
dimensions:
        x = 56 ;
        y = 56 ;
        z = 36 ;
        time = UNLIMITED ; // (25 currently)
variables:
        double x(x) ;
                x:units = "m" ;
        double y(y) ;
                y:units = "m" ;
        double z(z) ;
                z:units = "m" ;
        double time(time) ;
                time:units = "seconds_since_1999-01-01_00-00-00" ;
        float U(time, z, y, x) ;
                U:units = "m/s" ;
        float V(time, z, y, x) ;
                V:units = "m/s" ;
        float W(time, z, y, x) ;
                W:units = "m/s" ;
        float PRES(time, z, y, x) ;
                PRES:units = "Pa" ;
}
```

7.1.4　サブドメイン情報ファイルの仕様

以下に，サブドメイン情報ファイルの仕様を示します．

表 7.18　サブドメイン情報ファイル仕様

名称	表現	型	サイズ	説明
Identifier	文字列	uchar	4bytes	エンディアン識別子 (※ 1)
Size X	整数	uint	4bytes	X 方向領域分割数
Size Y	整数	uint	4bytes	Y 方向領域分割数
Size Z	整数	uint	4bytes	Z 方向領域分割数
Contents	整数	uchar	1bytes x SizeX x SizeY x SizeZ	活性サブドメインフラグ (※ 2)

(※ 1) リトルエンディアンのとき'S'，'B'，'D'，'M' の順に，ビッグエンディアンのとき'M'，'D'，'B'，'S' の順に対応する ASCII コードがセットされている．
(※ 2) 各領域の活性サブドメインフラグを X → Y → Z の順に格納．活性状態の場合 1 が，不活性状態の場合 0 が格納されている．

7.1.5　座標ファイルの仕様

座標ファイルとは，不等間隔な計算格子領域の格子点座標を格納したファイルであり，表 7.19 に示した順で格子点の座標情報が格納されています．

表 7.19　座標ファイルに格納されるデータ内容

データ内容
x 方向の格子点の数:Nx
x 方向の 1 番目の x 座標
・・・
x 方向の Nx 番目の x 座標
y 方向の格子点の数:Ny
y 方向の 1 番目の y 座標
・・・
y 方向の Ny 番目の y 座標
z 方向の格子点の数:Nz
z 方向の 1 番目の z 座標
・・・
z 方向の Nz 番目の z 座標

格子点の数は，ドメイン情報で指定される GlobalVoxel の数より 1 大きい点に注意してください．座標ファイルのファイル形式は，ascii 形式もしくは binary 形式です．ファイルの拡張子は " crd " です．

7.1.6　DFI ファイルのサンプル

index.dfi ファイルのサンプル

以下に，index.dfi のサンプルを示します．

```
FileInfo {
  DFIType            = "Non_Uniform_Cartesian"
  DirectoryPath      = "data"
  TimeSliceDirectory = "off"
  Prefix             = "field"
  FileFormat         = "plot3d"
  FieldFilenameFormat= "step_rank"
  GuideCell          = 0
  DataType           = "Float32"
  Endian             = "little"
  NumVariables       = 4
```

```
  Variable[@]{ name  = "u" }
  Variable[@]{ name  = "v" }
  Variable[@]{ name  = "w" }
  Variable[@]{ name  = "P" }
}
FilePath {
  Process = "./proc.dfi"
}
UnitList {
  Length {
    Unit       = "NonDimensional"
    Reference  = 1.000000e+00
  }
  Pressure {
    Unit       = "NonDimensional"
    Reference  = 0.000000e+00
    Difference = 1.176300e+00
  }
  Velocity {
    Unit       = "NonDimensional"
    Reference  = 1.000000e+00
  }
}
TimeSlice {
  Slice[@] {
    Step = 0
    Time = 0.000000e+00
    MinMax[@] {
      Min = 0.000000e+00
      Max = 0.000000e+00
    }
    MinMax[@] {
      Min = 0.000000e+00
      Max = 0.000000e+00
    }
    MinMax[@] {
      Min = 0.000000e+00
      Max = 0.000000e+00
    }
    MinMax[@] {
      Min = 0.000000e+00
      Max = 0.000000e+00
    }
  }
  Slice[@] {
    Step = 10
    Time = 3.125000e-02
    MinMax[@] {
      Min = -4.000939e-05
      Max = 2.169154e-04
    }
    MinMax[@] {
      Min = -4.603719e-07
      Max = 3.829139e-07
    }
    MinMax[@] {
      Min = -1.032495e-04
      Max = 1.032476e-04
    }
    MinMax[@] {
      Min = 2.018320e-09
      Max = 2.169154e-04
    }
  }
}
```

proc.dfi ファイルのサンプル

以下に，proc.dfi のサンプルを示します.

```
Domain {
  GlobalOrigin        = (-5.000000e-01, -5.000000e-01, -5.000000e-01)
  GlobalRegion        = (1.000000e+00, 1.000000e+00, 1.000000e+00)
  GlobalVoxel         = (64, 64, 64)
  GlobalDivision      = (2, 2, 2)
  ActiveSubdomainFile = ""
  CoordinateFile      = "coord.crd"
  CoordinateFileType  = "ascii"
  CoordinateFilePrecision = "Float64"
  CoordinateFileEndian = "little"
}
MPI {
  NumberOfRank   = 8
  NumberOfGroup  = 1
}
Process {
  Rank[@] {
    ID        = 0
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (1, 1, 1)
    TailIndex = (32, 32, 32)
  }
  Rank[@] {
    ID        = 1
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (33, 1, 1)
    TailIndex = (64, 32, 32)
  }
  Rank[@] {
    ID        = 2
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (1, 33, 1)
    TailIndex = (32, 64, 32)
  }
  Rank[@] {
    ID        = 3
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (33, 33, 1)
    TailIndex = (64, 64, 32)
  }
  Rank[@] {
    ID        = 4
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (1, 1, 33)
    TailIndex = (32, 32, 64)
  }
  Rank[@] {
    ID        = 5
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (33, 1, 33)
    TailIndex = (64, 32, 64)
  }
  Rank[@] {
    ID        = 6
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (1, 33, 33)
    TailIndex = (32, 64, 64)
  }
  Rank[@] {
```

```
    ID        = 7
    HostName  = "yakibuta"
    VoxelSize = (32, 32, 32)
    HeadIndex = (33, 33, 33)
    TailIndex = (64, 64, 64)
  }
}
```

7.2　ファイル仕様 (ツール)

7.2.1　ステージング用領域分割情報ファイルの仕様

以下に，ステージングツールで使用する領域分割情報を記述したファイルの仕様を示します.

領域分割情報ファイルの仕様

```
Domain (※ 1)
{
  GlobalVoxel         = ( 64, 64, 64 )        // 計算領域全体のボクセル数
  GlobalDivision      = ( 1, 1, 1)            // 計算領域全体の分割数
  ActiveSubdomainFile = "subdomain.dat"       // ActiveSubdomain ファイル名
}
FCONVInfo
{
  InputFile           = "conv.tp"             // FCONV 入力ファイル名
  NumberOfProcess     = 4                     // Mx1,MxM のときの FCONV 実行並列数
}
MPI(※ 3)
{
  NumberOfRank        = 1                     // プロセス数
}
Process(※ 3)
{
  Rank[@] {                                   // NumberOfRank 個
    ID                = 0                     // ランク番号
    VoxelSize         =( 64, 64, 64 )         // ボクセルサイズ
    HeadIndex         =(  1,  1,  1 )         // 始点インデックス (※ 4)
    TailIndex         =( 64, 64, 64 )         // 終点インデックス (※ 4)
  }
}
```

(※ 1) Domain タグは必須

ただし，ActiveSubdomainFile は任意

また、FCONV で入力領域指示 (CropIndexStart,CropIndeEnd) が指定されているときは，GlobalVoxel の値は無効になります.

GlobalDivision は MxN のみ有効

(※ 2) FCONVInfo タグは任意

InputFile で指定されたファイルから，ファイルの変換方法 (ConvType) と入力領域指示 (CropIndexStart,CropIndexEnd) を読込みます.

(※ 3) MPI,Process タグは任意

ランクの配置方向が I→J→K でない配置の場合，もしくは HeadIndex，TailIndex の位置が異なる場合に記述します.

(※ 4) HeadIndex,TailIndex

ランクの配置方向が I→J→K ではいとき HeadIndex,TailIndex を記述します.

FCONV で入力領域が指示されているときは無効になります.

ある方向について格子数 NV，領域分割数 ND(ランク番号 0〜ND-1) としたとき，あるランクにおける格子数は int(NV/ND) とする．ただし，ランク番号 <NV%ND のランクの格子数は +1 とします.

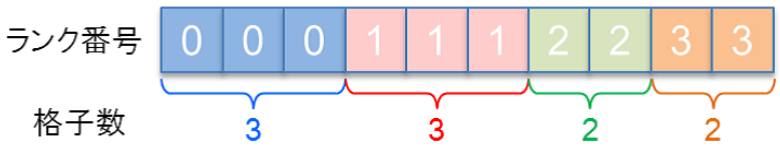


図 7.1　(例) 格子数 10，領域分割 4

7.2.2　並列分散ファイルコンバータ用入力ファイルの仕様

以下に，並列分散ファイルコンバータ（FCONV) で使用する入力ファイルの仕様を示します.

並列分散ファイルコンバータ入力ファイルの仕様

```
ConvData{
  InputDFI[@]="prs.dfi"                変換する dfi ファイルリスト（必須）
  InputDFI[@]="vel.dfi"
     :
  OutputDFI[@]="prs2.dfi"              出力する dfi ファイルリスト（省略可）
  OutputDFI[@]="vel2.dfi"
     :
  OutputProcDFI[@]="prs2_proc.dfi"     出力する proc.dfi ファイルリスト（省略可）
  PutputProcDFI[@]="vel2_proc.dfi"
     :
  ConvType="MxN"                       ファイル変換方法（必須）
  OutputDivision=(2,2,2)               出力分割情報（省略可）
  OutputFormat="sph"                   出力ファイルフォーマット（省略可）
  OutputDataType="Float32"             出力データタイプ（省略可）
  OutputFormatType="binary"            出力形式（省略可）
  OutputDir="conv_out"                 出力先ディレクトリ（必須）
  ThinningOut=2                        間引き数（省略可）
  OutputArrayShape="nijk"              出力配列形状（省略可）
  OutputFilenameFormat="step_rank"     出力ファイル命名順（省略可）
  OutputGuideCell=0                    出力ガイドセル数（省略可）
  MultiFileCasting="step"              並列時のファイル割振り方法（省略可）
  CropIndexStart=(1,1,22)              入力領域のスタート位置（省略可）
  CropIndexEnd=(64,64,32)              入力領域のエンド位置（省略可）
}
```

1. InputDFI
   - 「相対パスつきファイル名」,「絶対パスつきファイル名」,「ファイル名のみ」の 3 つの形式が指定可能.
2. OutPutDFI
   - 省略時は DFI ファイルが出力されない
   - InputDFI と同数の指定が必要
   - InputDFI と同じ記述順での対応
   - SPH,BOV 出力のみで有効
   - ファイル名のみ指定可能 (実行時のカレントディレクトリに出力される)

3. OutputProcDFI
   - OutputDFI が有効なときのみ指定可能
   - InputDFI と同数の指定が必要
   - InputDFI と同じ記述順での対応
   - 省略した場合は OutputDFI で指定したファイル名に”_proc”がついたファイル名で出力される
     (例) prs2_proc.dfi
   - ファイル名のみ指定可能 (実行時のカレントディレクトリに出力される)
4. ConvType
   - ファイル変換方法を”Mx1”,”MxN”,”MxM”で指定する
     (それぞれの変換方法は 5.1 変換イメージ図参照．)
5. OutputDivision
   - 出力分割情報を各方向 (IDIV,JDIV,KDIV) で指定で指定する
   - ファイル変換形式 (ConvType) が MxN のときのみ有効
   - 指定した場合は実行並列数が IDIV × JDIV × KDIV となる必要がある
   - 省略した場合は CPMlib が自動分割機能で実行並列数より自動分割を行う
6. OutputFormat
   - 出力ファイルフォーマットを”sph”,”bov”,”avs”,”plot3d”,”vtk”，”netcdf4”で指定する
   - 省略された場合は入力 DFI で指定されているファイルフォーマットで出力
7. OutputDataType
   - 出力データタイプを”Int8”,”UInt8”,”Int16”,”UInt16”,”Int32”,”UInt32”,”Float32”,”Float64”で指定する
   - 省略した場合は型変換を行わない
8. OutputFormatType
   - 出力形式を”ascii”,”binary”,”Fortran_Binary”で指定する
     (5.1 ファイルフォーマット毎の出力形式参照．)
9. ThinningOut
   - １以下のとき間引きなし，２以上のとき間引きあり
     (5.1.9 間引き数を２とした例参照)
   - 省略した場合は間引きを行わない
10. OutputArrayShape
    - 出力配列形状を”nijk”,”ijkn”で指定する
    - 出力ファイルフォーマットが BOV のときのみ有効
    - BOV 以外のフォーマットで指定しても自動的に対応する配列形状で出力
    - 省略した場合は入力と同じ形式で出力
11. OutputFilenameFormat
    - 出力ファイルの命名順を”step_rank”,”rank_step”で指定する
      step_rank:[Prefix]_[StepNo,10 桁]_id[RankID,6 桁].[ext]
      rank_step:[Prefix]_id[RankID,6 桁]_[StepNo,10 桁].[ext]
    - 省略した場合は step_rank
    - 出力ファイルが逐次データの場合，OutputFilenameFormat の指示によらず，
      step_rank:[Prefix]_[StepNo,10 桁].[ext]
      とする．(RankID は出力しない，CDMlib の仕様に準拠)
12. OutputGuideCell
    - SPH,BOV のみ対応
    - 間引き有り，格子点有りは未対応
    - 出力可能なガイドセル数は入力ファイルに出力されているガイドセル数以下

13. MultiFileCasting
    - 並列実行時のファイル割振り方法を”step”,”rank”で指定する
    - ”step”:step 基準，”rank”:rank 基準でファイル割振りを行う (5.1.10 参照)
    - Mx1 では”step”のみ，MxN では無効
    - 省略した場合は”step”
14. CropIndexStart,CropIndexEnd
    - 各方向で入力全体のインデックス (I 方向，J 方向，K 方向) で指定する
    - MxM は未対応
    - CropIndexStart,CropIndexEnd ともに指定されている場合はその領域
    - CropIndexStart のみ指定されている場合は CropIndexStart から最後までの領域
    - CropIndexEnd のみ指定されている場合は先頭から CropIndexEnd までの領域
    - 両方省略された場合は全部の領域

7.2.3　NetCDF4 用 DFI 生成ツール入力ファイルの仕様

以下に，NetCDF4 用 DFI 生成ツール (netcdf2dfi) で使用する入力ファイルの仕様を示します.

NetCDF4 用 DFI 生成ツール入力ファイルの仕様

```
NetCDF2DFI
{
  num_ncfiles         = 36
  num_steps           = 25
  DirectoryPath       = "./data"
  Prefix              = "history_d01"
  FieldFilenameFormat = "rank"
  RankNoPrefix        = ".pe"
  GuideCell           = 0
  Variable[@] {name = "U"}
  Variable[@] {name = "V"}
  Variable[@] {name = "W"}
  Variable[@] {name = "PRES"}
  OutputDirectoryPath = "./output"
  index_fname         = "index.dfi"
  proc_fname          = "proc.dfi"

  VariableName
  {
    x    = "x"
    y    = "y"
    z    = "z"
    time = "time"
  }
}
```

1. num_ncfiles

   元になる NetCDF4 ファイルのファイル数を指定します.

   1 ステップ 1 ファイルの場合は，元のファイルの分散並列数を指定します.

2. DirectoryPath

元になる NetCDF4 ファイルの存在するディレクトリパスを指定します.
相対パスを指定した場合は，netcdf2dfi 実行時のカレントディレクトリからの相対パスになります.

3. Prefix
   元になる NetCDF4 ファイルのベースファイル名を指定します.
4. FieldFilenameFormat
   元になる NetCDF4 ファイルのファイル名命名順を指定します.
   step_rank : [Prefix]_[ステップ番号:10 桁]_id[RankID:6 桁].[ext]
   rank_step : [Prefix]_id[RankID:6 桁]_[ステップ番号:10 桁].[ext]
   rank : [Prefix]_id[RankID:6 桁].[ext]
   step_rank，rank_step の場合，1 ステップ 1 ファイルとして読み込みます.
   rank の場合，1 ファイル全時系列格納ファイルとして読み込みます.
   NetCDF4 ファイル名のステップ番号，RankID に相当する連番は 0 から始まる 1 刻みの数字である必要があります.
5. num_steps
   元になる NetCDF4 ファイルのステップ数を指定します.
   - FieldFilenameFormat が step_rank，rank_step の場合
     必ず指定する必要があります.
   - FieldFilenameFormat が rank の場合
     省略可能です．NetCDF4 ファイル内の配列サイズからステップ数を自動的に取得するため，本パラメータの指定は必要ありません.
6. RankNoPrefix(省略可)
   フィールドファイルのランク番号前の文字列を指定します.
   省略時のデフォルトは”_id”です.
   指定すると，[Prefix]_[ステップ番号:10 桁][RankNoPrefix][RankID:6 桁].[ext] のように，フィールドファイル名内の文字列を変更できます.
7. GuideCell
   仮想セル数を指定します.
   1 以上が指定された場合，元の NetCDF4 ファイルの格子数のうち，GuideCell 分の格子が仮想セルとして認識されます.
8. Variable(複数指定可)
   抽出する成分名を指定します.
9. OutputDirectoryPath(省略可)
   Variable で指定した成分のみを抽出した NetCDF4 ファイルの出力先ディレクトリを指定します.
   指定した場合，出力 dfi ファイルからはこのディレクトリに出力された NetCDF4 ファイルを参照します.
   省略した場合，出力 dfi ファイルからは元の NetCDF4 ファイル (DirectoryPath で指定したディレクトリに存在するファイル) を参照します.
10. index_fname(省略可)
    出力するインデックスファイル名を指定します.
    省略時のデフォルトは”index.dfi”です.
11. proc_fname(省略可)
    出力するプロセス情報ファイル名を指定します.
    省略時のデフォルトは”proc.dfi”です.
12. VariableName(省略可)
    NetCDF4 ファイル内の必須 dimension 名，variable 名を指定します.
    - x(省略可)

x 格子数 (dimension)，格子図心 X 座標値 (variable) の名称を指定します.
省略時のデフォルトは”x”です.

- y(省略可)
  y 格子数 (dimension)，格子図心 Y 座標値 (variable) の名称を指定します.
  省略時のデフォルトは”y”です.
- z(省略可)
  z 格子数 (dimension)，格子図心 Z 座標値 (variable) の名称を指定します.
  省略時のデフォルトは”z”です.
- time(省略可)
  ステップ数 (dimension)，各ステップの時刻 (variable) の名称を指定します.
  省略時のデフォルトは”time”です.

第 8 章

アップデート情報

アップデート情報について記します.

8.1　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.

Revision 1　　2014/XX/XX

- 　　リリース

第 9 章

Appendix

9.1　API メソッド一覧

以下に，cdm ライブラリが提供する API メソッドの一覧を示します．（表 9.1）

表 9.1　メソッド一覧（クラス名の無い C++ メソッドは cdm_DFI クラスメンバ）

機能	C++ API	備考
読込み用インスタンスの生成	ReadInit	static メソッド
出力用インスタンスの生成	WriteInit	float 版，static メソッド
	WriteInit	double 版，static メソッド
cdm_FileInfo クラスポインタの取得	GetcdmFileInfo	
cdm_FilePath クラスポインタの取得	GetcdmFilePath	
cdm_Unit クラスポインタの取得	GetcdmUnit	
cdm_Domain クラスポインタの取得	GetcdmDomain	
cdm_MPI クラスポインタの取得	GetcdmMPI	
cdm_TimeSlice クラスポインタの取得	GetcdmTimeSlice	
cdm_Process クラスポインタの取得	GetcdmProcess	
フィールドデータの読込み	ReadData	読込んだデータの配列ポインタが戻される
	ReadData	引数で渡された配列ポインタに読み込まれる
フィールドデータの出力	WriteData	
proc.dfi ファイル出力	WriteProcDfiFile	float 版
	WriteProcDfiFile	double 版
DFI の配列形状を取得	GetArrayShapeString	文字列を取得
	GetArrayShape	列挙型を取得
DFI のデータタイプ取得	GetDataTypeString	文字列を取得
	GetDataType	列挙型を取得
DFI の変数の個数取得	GetNumVariables	
データタイプを文字列から列挙型に変換	ConvDatatypeS2E	static メソッド
データタイプを列挙型から文字列に変換	ConvDatatypeE2S	static メソッド
DFI の GlobalVoxel の取得	GetDFIGlobalVoxel	
DFI の GlobalDivision の取得	GetDFIGlobalDivision	
単位系を追加	AddUnit	
単位系を取得（クラス単位）	GetUnitElem	
単位系を取得（メンバ変数）	GetUnit	
FileInfo の変数名を登録する	setVariableName	
FileInfo の変数名を取得する	getVariableName	
DFI の MinMax の合成値を取得する	getVectorMinMax	
DFI の MinMax を取得する	getMinMax	
読込みランクリストの生成	CheakReadRank	
インターバルステップの登録	setIntervalStep	
インターバルタイムの登録	setIntervalTime	
インターバルの時間を無次元化する	normalizeTime	base_time,interval_time,start_time, last_time 全て無次元化する
インターバルの base_time を無次元化	normalizeBaseTime	
インターバルの interval を無次元化	normalizeIntervalTime	
インターバルの start_time を無次元化	normalizeStartTime	
インターバルの last_time を無次元化	normalizeLastTime	
インターバルの DetlaT を無次元化	normalizeDelteT	
cdm のバージョン No の取り出し	getVersionInfo	static メソッド

表目次

図目次