User Guide of CIOlib

Cartesian Input / Output Library

Ver. 1.3.8

Advanced Institute for Computational Science
RIKEN

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

October 2013

Version 1.3.8	10 Oct.	2013
Version 1.3.7	2 Oct.	2013
Version 1.3.6	29 Sep.	2013
Version 1.3.5	9 Aug.	2013
Version 1.3.4	20 July	2013
Version 1.3.3	27 Jun.	2013
Version 1.3.2	27 Jun.	2013
Version 1.3.1	26 Jun.	2013
Version 1.3.0	25 Jun.	2013
Version 1.2.0	10 Jun.	2013
Version 1.1.0	8 Jun.	2013
Version 1.0.0	6 Jun.	2013

目次

第 1 章

CIOlib の概要

CIOlib の概要と本ユーザガイドについて説明します.

1.1　CIOlib

CIOlib(Cartesian Input/Output 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.

CIOlib 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
- SPH，BOV,（BVX）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）

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

CIO 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，CIOlib 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

CIO 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)
`CIOlib-`*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.

```
CIOlib-X.X.X
   ├─── AUTHORS
   ├─── COPYING
   ├─── ChangeLog
   ├─── INSTALL
   ├─── LICENSE
   ├─── Makefile.am
   ├─── Makefile.in
   ├─── NEWS
   ├─── README
   ├─── aclocal.m4
   ├─── cio-config.in
   ├─── config.h.in
   ├─── config_cio.sh
   ├─── config_cio_nompi.sh
   ├─── configure
   ├─── configure.ac
   ├─── depcomp
   ├─── doc/
   │      ├─── Makefile.am
   │      ├─── Makefile.in
   │      ├─── doxygen/
   │      │      ├─── Doxyfile
   │      │      └─── makepdf.sh
   │      └─── reference.pdf
   ├─── include/
   │      └─── inline/
   ├─── install-sh
   ├─── missing
   ├─── src/
   └─── tools/
          └─── frm/
                 ├─── 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．

- doc
  この文書を含む CIOlib 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 libCIO.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 CIOlib-X.X.X.tar.gz
   ```

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

   ```
   $ cd CIOlib-X.X.X
   ```

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

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

   configure スクリプトの実行時には，お使いの環境に合わせたオプションを指定する必要があります．configure オプションに関しては，2.1.3 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/libCIO.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.

```
${prefix}
├──bin/
│   └──cio-config
├──doc/
│   └──CIOlib.pdf
├──include/
│   ├──cio_ActiveSubDomain.h
│   ├──cio_Array.h
│   ├──cio_DFI.h
│   ├──cio_DFI_BOV.h
│   ├──cio_DFI_SPH.h
│   ├──cio_Define.h
│   ├──cio_Domain.h
│   ├──cio_FileInfo.h
│   ├──cio_FilePath.h
│   ├──cio_Interval_Mngr.h
│   ├──cio_MPI.h
│   ├──cio_PathUtil.h
│   ├──cio_Process.h
│   ├──cio_TextParser.h
│   ├──cio_TimeSlice.h
│   ├──cio_TypeArray.h
│   ├──cio_Unit.h
│   ├──cio_Version.h
│   ├──cio_endianUtil.h
│   └──inline/
│       ├──cio_Array_inline.h
│       └──cio_DFI_inline.h
├──lib/
│   └──libCIO.a
└──share/
    ├──AUTHORS
    ├──COPYING
    ├──ChangeLog
    ├──LICENSE
    └──README
```

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
password:
# 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/CIOlib の時,

ライブラリ：/usr/local/CIOlib/lib
ヘッダファイル：/usr/local/CIOlib/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/CIOlib 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
CIO ライブラリを利用する場合，コンパイル，リンク時に，MPI ライブラリと TextParser ライブラリが必ず必要になります．これらのライブラリのインストールパスは，次に示す configure オプションで指定する必要があります.

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

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

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

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

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，CIO 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

CIO **ライブラリの** prefix：/opt/CIOlib
MPI **ライブラリ**：OpenMPI，/usr/local/openmpi
TextParser **ライブラリ**：/usr/local/textparser
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/CIOlib \
              --with-ompi=/usr/local/openmpi \
              --with-parser=/usr/local/textparser \
              --with-comp=INTEL \
              CXX=icpc \
              FC=ifort
```

・京コンピュータの場合

CIO **ライブラリの** prefix：/home/userXXXX/CIOlib
TextParser **ライブラリ**：/home/userXXXX/textparser

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/CIOlib \
              --with-parser=/home/userXXXX/textparser \
              --with-comp=FJ \
              CXX=mpiFCCpx \
              FC=mpifrtpx
```

2.1.5　cio-config コマンド

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

cio-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
CIO 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
CIO 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，cio-config コマンドで取得できるオプションは，CIO 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 cio-config コマンドの使用方法は，**??**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

CIOlib-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 CIO 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，CIO 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「–without-MPI」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/CIOlib_frontend \
              --without-MPI \
              --with-parser=/home/userXXXX/textparser \
              CXX=g++ \
              FC=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.

2.2　CIO ライブラリの利用方法

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

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

2.2.1　C++

CIO 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/CIOlib/bin/cio-config --cflags‘ \
  ‘/usr/local/CIOlib/bin/cio-config --libs‘
```

第 3 章

API 利用方法

この章では，CIOlib 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，CIO ライブラリの C++ API の説明を示します．

3.1.1　cio_DFI.h のインクルード

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

cio_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 cio_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．

cio_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

CIO 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，cio_Define.h に定義されています．

・D_CIO_XXXX マクロ

表 3.1　D_CIO_XXXX マクロ

マクロ名	内容	マクロ名	内容	マクロ名	内容
D_CIO_EXT_SPH	”sph”	D_CIO_LITTLE	”little”	D_CIO_UINT8	”UInt8”
D_CIO_EXT_BOV	”bov”	D_CIO_BIG	”big”	D_CIO_UINT16	”UInt16”
D_CIO_ON	”on”	D_CIO_INT8	”Int8”	D_CIO_UINT32	”UInt32”
D_CIO_OFF	”off”	D_CIO_INT16	”Int16”	D_CIO_FLOAT32	”Float32”
D_CIO_IJNK	”ijkn”	D_CIO_INT32	”Int32”	D_CIO_FLOAT64	”Float64”
D_CIO_NIJK	”nijk”	D_CIO_INT64	”Int64”		

・E_CIO_ONOFF 列挙型

E_CIO_ONOFF 列挙型は，cio_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_CIO_ONOFF 列挙型

E_CIO_ONOFF 要素	値	意味
E_CIO_OFF	0	スイッチオフ
E_CIO_ON	1	スイッチオン

・E_CIO_FORMAT 列挙型

E_CIO_FORMAT 列挙型は，cio_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_CIO_FORMAT 列挙型

E_CIO_FORMAT 要素	値	意味
E_CIO_UNKNOWN	-1	未定義
E_CIO_SPH	0	SPH 形式
E_CIO_BOV	1	BOV 形式

・E_CIO_DTYPE 列挙型
E_CIO_DTYPE 列挙型は，cio_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_CIO_DTYPE 列挙型

E_CIO_DTYPE 要素	値	意味
E_CIO_DTYPE_UNKNOWN	0	未定義
E_CIO_INT8	1	char
E_CIO_INT16	2	short
E_CIO_INT32	3	int
E_CIO_INT64	4	long long
E_CIO_UINT8	5	unsigned char
E_CIO_UINT16	6	unsigned short
E_CIO_UINT32	7	unsigned int
E_CIO_UINT64	8	unsigned long long
E_CIO_FLOAT32	9	float
E_CIO_FLOAT64	10	double

・E_CIO_ARRAYSHAPE 列挙型
E_CIO_ARRAYSHAPE 列挙型は，cio_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配列形式を指定するフラグとして使われます.

表 3.5　E_CIO_ARRAYSHAPE 列挙型

E_CIO_ARRAYSHAPE 要素	値	意味
E_CIO_ARRAYSHAPE_UNKNOWN	-1	未定義
E_CIO_IJKN	0	(i,j,k,n)
E_CIO_NIJK	1	(n,i,j,k)

・E_CIO_ENDIANTYPE 列挙型
E_CIO_ENDIANTYPE 列挙型は，cio_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.

・E_CIO_READTYPE 列挙型
E_CIO_READTYPE 列挙型は，cio_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 3.6　E_CIO_ENDIANTYPE 列挙型

E_CIO_ENDIANTYPE 要素	値	意味
E_CIO_ENDIANTYPE_UNKNOWN	-1	未定義
E_CIO_LITTELE	0	リトルエンディアン形式
E_CIO_BIG	1	ビッグエンディアン形式

表 3.7　E_CIO_READTYPE 列挙型

E_CIO_READTYPE 要素	値	意味
E_CIO_SAMEDIV_SAMERES	1	同一分割，同一密度
E_CIO_SAMEDIV_REFINEMENT	2	同一分割，粗密
E_CIO_DIFFDIV_SAMERES	3	MxN，同一密度
E_CIO_DIFFDIV_REFINEMENT	4	MxN，粗密
E_CIO_READTYPE_UNKNOWN	5	エラー

・E_CIO_ERRORCODE 列挙型

E_CIO_ERRORCODE 列挙型は，cio_Define.h で表 3.8，3.9 のように定義されています．

CIO 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 3.8　E_CIO_ERRORCODE 列挙型 その 1

E_CIO_ERRORCODE 要素	値	意味
E_CIO_SUCCESS	0	正常終了
E_CIO_ERROR	-1	その他のエラー
E_CIO_ERROR_READ_DFI_GLOBALORIGIN	1000	DFI GlobalOrigin 読込みエラー
E_CIO_ERROR_READ_DFI_GLOBALREGION	1001	DFI GlobalRegion 読込みエラー
E_CIO_ERROR_READ_DFI_GLOBALVOXEL	1002	DFI GlobalVoxel 読込みエラー
E_CIO_ERROR_READ_DFI_GLOBALDIVISION	1003	DFI GlobalDivison 読込みエラー
E_CIO_ERROR_READ_DFI_DIRECTORYPATH	1004	DFI DirectoryPath 読込みエラー
E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY	1005	DFI TimeSliceDirectoryPath 読込みエラー
E_CIO_ERROR_READ_DFI_PREFIX	1006	DFI Prefix 読込みエラー
E_CIO_ERROR_READ_DFI_FILEFORMAT	1007	DFI FileFormat 読込みエラー
E_CIO_ERROR_READ_DFI_GUIDECELL	1008	DFI GuideCell 読込みエラー
E_CIO_ERROR_READ_DFI_DATATYPE	1009	DFI DataType 読込みエラー
E_CIO_ERROR_READ_DFI_ENDIAN	1010	DFI Endian 読込みエラー
E_CIO_ERROR_READ_DFI_ARRAYSHAPE	1011	DFI ArrayShape 読込みエラー
E_CIO_ERROR_READ_DFI_COMPONENT	1012	DFI Component 読込みエラー
E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS	1013	DFI FilePath/Process 読込みエラー
E_CIO_ERROR_READ_DFI_NO_RANK	1014	DFI Rank 要素なし
E_CIO_ERROR_READ_DFI_ID	1015	DFI ID 読込みエラー
E_CIO_ERROR_READ_DFI_HOSTNAME	1016	DFI HoatName 読込みエラー
E_CIO_ERROR_READ_DFI_VOXELSIZE	1017	DFI VoxelSize 読込みエラー
E_CIO_ERROR_READ_DFI_HEADINDEX	1018	DFI HeadIndex 読込みエラー
E_CIO_ERROR_READ_DFI_TAILINDEX	1019	DFI TailIndex 読込みエラー
E_CIO_ERROR_READ_DFI_NO_SLICE	1020	DFI TimeSlice 要素なし
E_CIO_ERROR_READ_DFI_STEP	1021	DFI Step 読込みエラー
E_CIO_ERROR_READ_DFI_TIME	1022	DFI Time 読込みエラー
E_CIO_ERROR_READ_DFI_NO_MINMAX	1023	DFI MinMax 要素なし
E_CIO_ERROR_READ_DFI_MIN	1024	DFI Min 読込みエラー
E_CIO_ERROR_READ_DFI_MAX	1025	DFI Max 読込みエラー
E_CIO_ERROR_READ_INDEXFILE_OPENERROR	1050	Index ファイルオープンエラー
E_CIO_ERROR_TEXTPARSER	1051	TextParser エラー
E_CIO_ERROR_READ_FILEINFO	1052	FileInfo 読込みエラー
E_CIO_ERROR_READ_FILEPATH	1053	FilePath 読込みエラー
E_CIO_ERROR_READ_UNIT	1054	UNIT 読込みエラー
E_CIO_ERROR_READ_TIMESLICE	1055	TimeSlice 読込みエラー
E_CIO_ERROR_READ_PROCFILE_OPENERROR	1056	Proc ファイルオープンエラー
E_CIO_ERROR_READ_DOMAIN	1057	Domain 読込みエラー
E_CIO_ERROR_READ_MPI	1058	MPI 読込みエラー
E_CIO_ERROR_READ_PROCESS	1059	Process 読込みエラー
E_CIO_ERROR_READ_FIELDDATA_FILE	1900	フィールドデータファイル読込みエラー
E_CIO_ERROR_READ_SPH_FILE	2000	SPH ファイル読込みエラー
E_CIO_ERROR_READ_SPH_REC1	2001	SPH ファイルレコード 1 読込みエラー
E_CIO_ERROR_READ_SPH_REC2	2002	SPH ファイルレコード 2 読込みエラー

表 3.9　E_CIO_ERRORCODE 列挙型 その 2

cpm_ErrorCode 要素	値	意味
E_CIO_ERROR_READ_SPH_REC3	2003	SPH ファイルレコード 3 読込みエラー
E_CIO_ERROR_READ_SPH_REC4	2004	SPH ファイルレコード 4 読込みエラー
E_CIO_ERROR_READ_SPH_REC5	2005	SPH ファイルレコード 5 読込みエラー
E_CIO_ERROR_READ_SPH_REC6	2006	SPH ファイルレコード 6 読込みエラー
E_CIO_ERROR_READ_SPH_REC7	2007	SPH ファイルレコード 7 読込みエラー
E_CIO_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_CIO_ERROR_NOMATCH_ENDIAN	2051	出力 Fornat が合致しない（Endian 形式が Big,Little 以外）
E_CIO_ERROR_READ_BOV_FILE	2100	BOV ファイル読込みエラー
E_CIO_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_CIO_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_CIO_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_CIO_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_CIO_ERROR_INVALID_DIVNUM	3011	領域分割数が不正
E_CIO_ERROR_OPEN_SBDM	3012	ActiveSubdomain ファイルのオープンに失敗
E_CIO_ERROR_READ_SBDM_HEADER	3013	ActiveSubdomain ファイルのヘッダー読み込みに失敗
E_CIO_ERROR_READ_SBDM_FORMAT	3014	ActiveSubdomain ファイルのフォーマットエラー
E_CIO_ERROR_READ_SBDM_DIV	3015	ActiveSubdomain ファイルの領域分割数読み込みに失敗
E_CIO_ERROR_READ_SBDM_CONTENTS	3016	ActiveSubdomain ファイルの Contents 読み込みに失敗
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO	3017	ActiveSubdomain ファイルの活性ドメイン数が 0
E_CIO_ERROR_MAKEDIRECTORY	3100	Directory 生成で失敗
E_CIO_ERROR_OPEN_FIELDDATA	3101	フィールドデータのオープンに失敗
E_CIO_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_CIO_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_CIO_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_CIO_ERROR_WRITE_SPH_REC1	3201	SPH ファイルレコード 1 出力エラー
E_CIO_ERROR_WRITE_SPH_REC2	3202	SPH ファイルレコード 2 出力エラー
E_CIO_ERROR_WRITE_SPH_REC3	3203	SPH ファイルレコード 3 出力エラー
E_CIO_ERROR_WRITE_SPH_REC4	3204	SPH ファイルレコード 4 出力エラー
E_CIO_ERROR_WRITE_SPH_REC5	3205	SPH ファイルレコード 5 出力エラー
E_CIO_ERROR_WRITE_SPH_REC6	3206	SPH ファイルレコード 6 出力エラー
E_CIO_ERROR_WRITE_SPH_REC7	3207	SPH ファイルレコード 7 出力エラー
E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY	3500	proc dfi ファイル名が未定義
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR	3501	proc dfi ファイルオープン失敗
E_CIO_ERROR_WRITE_DOMAIN	3502	Domain 出力失敗
E_CIO_ERROR_WRITE_MPI	3503	MPI 出力失敗
E_CIO_ERROR_WRITE_PROCESS	3504	Process 出力失敗
E_CIO_ERROR_WRITE_RANKID	3505	出力ランク以外
E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY	3510	index dfi ファイル名が未定義
E_CIO_ERROR_WRITE_PREFIX_EMPTY	3511	Prefix が未定義
E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR	3512	proc dfi ファイルオープン失敗
E_CIO_ERROR_WRITE_FILEINFO	3513	FileInfo 出力失敗
E_CIO_ERROR_WRITE_UNIT	3514	Unit 出力失敗
E_CIO_ERROR_WRITE_TIMESLICE	3515	TimeSlice 出力失敗
E_CIO_ERROR_WRITE_FILEPATH	3516	FilePath 出力失敗
E_CIO_WARN_GETUNIT	4000	Unit の単位がない

3.2　入力機能

3.2.1　機能概要

CIO 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．CIO ではこれらを自動敵に把握して，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．

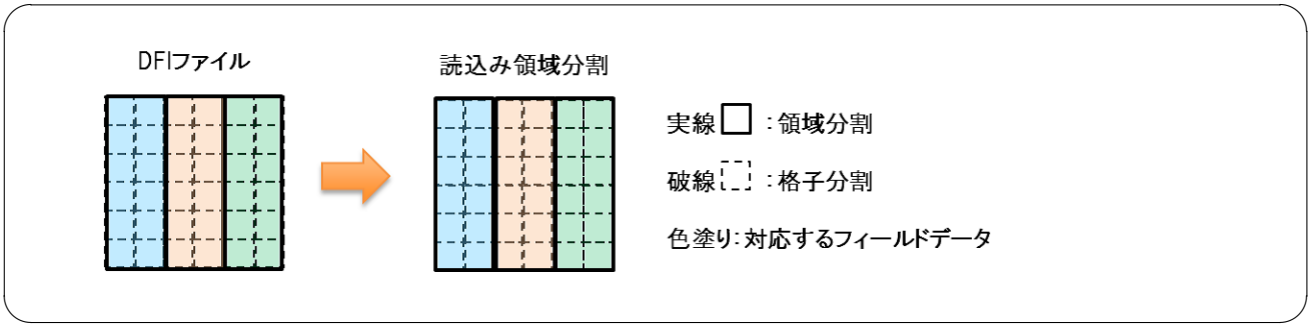

図 3.1　同一格子密度での 1 対 1 読込み

- 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．

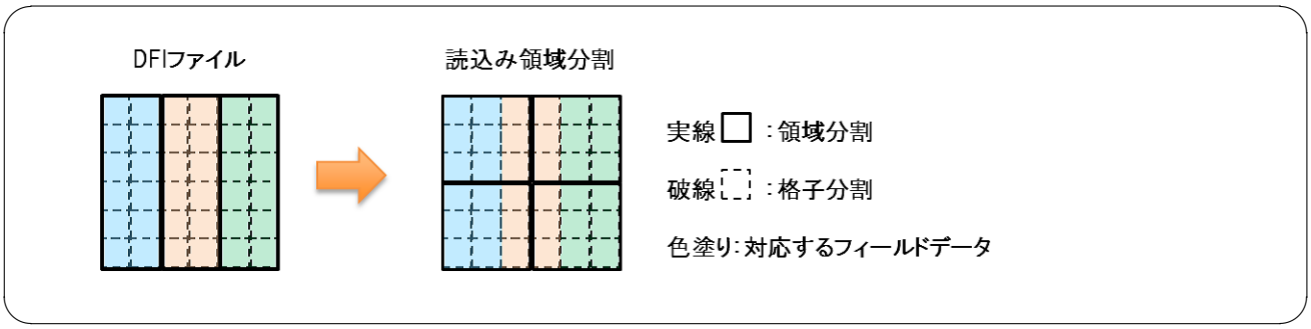

図 3.2　同一格子密度での M 対 N 読込み

- 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.

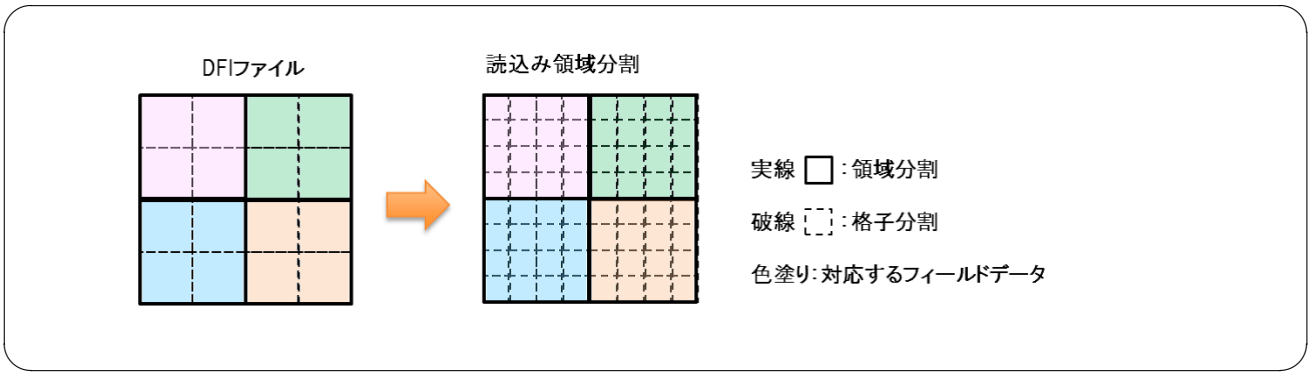

図 3.3　リファインメントで 1 対 1 読込み

- 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.

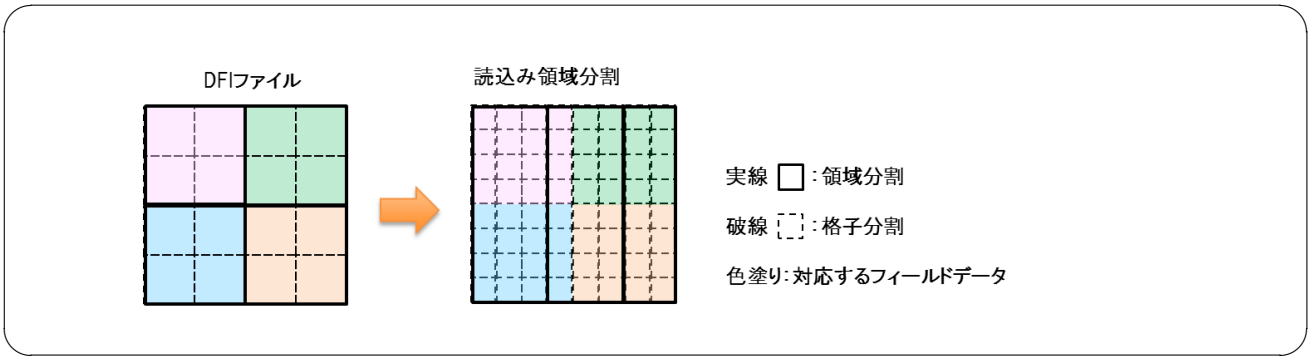

図 3.4　リファインメントで M 対 N 読込み

(※ 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

CIO 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 (**??**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．

cio_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，cio_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 cio_DFI* cio_DFI::ReadInit(const MPI_Comm comm,
                                  const std::string dfifile,
                                  const int G_Voxel[3],
                                  const int G_Div[3],
                                  CIO::E_CIO_ERRORCODE &ret);
```

cio_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 3.8，3.9 を参照）
戻り値	cio_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
cioDFI *DFI_IN_PRS = cio_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 CIO 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 cio_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配列形状の取得
   フィールドデータの配列形状は文字列 (表 3.1 参照) と列挙型 (表 3.5 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配列形状の取得（文字列）

```
std::string
cio_DFI::GetArrayShapeString();
```

戻り値 フィールドデータの配列形状，文字列 (表 3.1 参照)

フィールドデータの配列形状の取得（列挙型）

```
CIO::E_CIO_ARRAYSHAPE
cio_DFI::GetArrayShapeString();
```

戻り値　フィールドデータの配列形状，列挙型 (表 3.5 参照)

2. 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
cio_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）

```
CIO::E_CIO_DTYPE
cio_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)

3. フィールドデータの成分数の取得

フィールドデータの成分数の取得

```
int
cio_DFI::GetNumComponent();
```

戻り値　フィールドデータの成分数

4. 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 CIO::E_CIO_DTYPE
cio_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)

5. 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
cio_DFI::ConvDatatypeE2S(const CIO::E_CIO_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)

6. 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 5.1.2 章「プロセス情報ファイル (proc.dfi) 仕様」参照.

DFI Domain の GlobalVoxel の取得

```
int*
cio_DFI::GetDFIGlobalVoxel();
```

戻り値　GlobalVoxel のポインタを取得します.

(注) 取得したポインタは，不要になったときにユーザが delete する必要があります.

7. 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 5.1.2 章「プロセス情報ファイル (proc.dfi) 仕様」参照.

DFI Domain の GlobalDivision の取得

```
int*
cio_DFI::GetDFIGlobalDivision();
```

戻り値　GlobalDivision のポインタを取得します.

(注) 取得したポインタは，不要になったときにユーザが delete する必要があります.

8. DFI FileInfo の成分名を取得

   DFI ファイルの FileInfo の仕様は 5.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

DFI FileInfo の成分名を取得

```
std::string
cio_DFI::getComponentVariable(int pcomp);
```

 pcomp　[*input*]　成分位置　0:u 1:v 2:w
 戻り値　成分名

9. DFI TimeSlice の minmax 合成値を取得

   DFI ファイルの TimeSlice の仕様は 5.1.1 章「インデックスファイル (index.dfi) 仕様」参照.

DFI TimeSlice の minmax 合成値を取得

```
CIO::E_CIO_ERRORCODE
cio_DFI::getVectorMinMax(const unsigned step,
                         double &vec_min,
                         double &vec_max);
```

step	[*input*]	対象となるステップ番号
vec_min	[*output*]	min の合成値
vec_max	[*output*]	max の合成値
戻り値	エラーコード（表 3.8，3.9 を参照）	

10. DFI TimeSlice の minmax 値を取得

    DFI ファイルの TimeSlice の仕様は 5.1.1 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

```
CIO::E_CIO_ERRORCODE
cio_DFI::getMinMax(const unsigned step,
                   const int compNo,
                   double &min_value,
                   double &max_value);
```

step	[*input*]	対象となるステップ番号
compNo	[*input*]	対象となる成分番号 (0〜n)
min_value	[*output*]	min
max_value	[*output*]	max
戻り値	エラーコード（表 3.8，3.9 を参照）	

11. 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 5.1.1 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

```
CIO::E_CIO_ERRORCODE
cio_DFI::GetUnitElem(const std::string Name,
                     cio_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 3.8，3.9 を参照）	

12. 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 5.1.1 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

```
CIO::E_CIO_ERRORCODE
cio_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 3.8，3.9 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 CIO 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 cio_DFI.h 内で次のように定義されています．

1. cio_FileInfo クラスポインタの取得

cio_FileInfo クラスポインタの取得

```
const cio_FileInfo* GetcioFileInfo();
```

戻り値 FileInfo の情報がセットされたクラスのポインタ

2. cio_FilePath クラスポインタの取得

cio_FilePath クラスポインタの取得

```
const cio_FilePath* GetcioFilePath();
```

戻り値 FilePath の情報がセットされたクラスのポインタ

3. cio_Unit クラスポインタの取得

cio_Unit クラスポインタの取得

```
const cio_Unit* GetcioUnit();
```

戻り値 Unit の情報がセットされたクラスのポインタ

4. cio_Domain クラスポインタの取得

cio_Domain クラスポインタの取得

```
const cio_Domain* GetcioDomain();
```

戻り値 Domain の情報がセットされたクラスのポインタ

5. cio_MPI クラスポインタの取得

cio_MPI クラスポインタの取得

```
const cio_MPI* GetcioMPI();
```

戻り値 MPI の情報がセットされたクラスのポインタ

6. cio_TimeSlice クラスポインタの取得

cio_TimeSlice クラスポインタの取得

```
const cio_TimeSlice* GetcioTimeSlice();
```

戻り値 TimeSlice の情報がセットされたクラスのポインタ

7. cio_Process クラスポインタの取得

cio_Process クラスポインタの取得

```
const cio_Process* GetcioProcess();
```

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 形式と BOV 形式ファイルです．（詳細は，5.1.3 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 cio_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(CIO::E_CIO_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 3.8，3.9 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>
CIO::E_CIO_ERRORCODE
cio_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 3.8，3.9 を参照）	

3.2.6 リファインメントデータ補間メソッド

CIO 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 (cio_interp.f90) により，単純な補間処理を行います.

1. IJKN 配列

cio_interp_ijkn_r4 ： IJKN 配列，単精度実数版

```
subroutine cio_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 'cio_interp_ijkn.h'

  return
end subroutine cio_interp_ijkn_r4
```

cio_interp_ijkn_r8 ： IJKN 配列，倍精度実数版

```
subroutine cio_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 'cio_interp_ijkn.h'

  return
end subroutine cio_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

cio_interp_nijk_r4 ： NIJK 配列，単精度実数版

```
subroutine cio_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 'cio_interp_nijk.h'

  return
end subroutine cio_interp_nijk_r4
```

cio_interp_nijk_r8 ： NIJK 配列，倍精度実数版

```
subroutine cio_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 'cio_interp_nijk.h'

  return
end subroutine cio_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

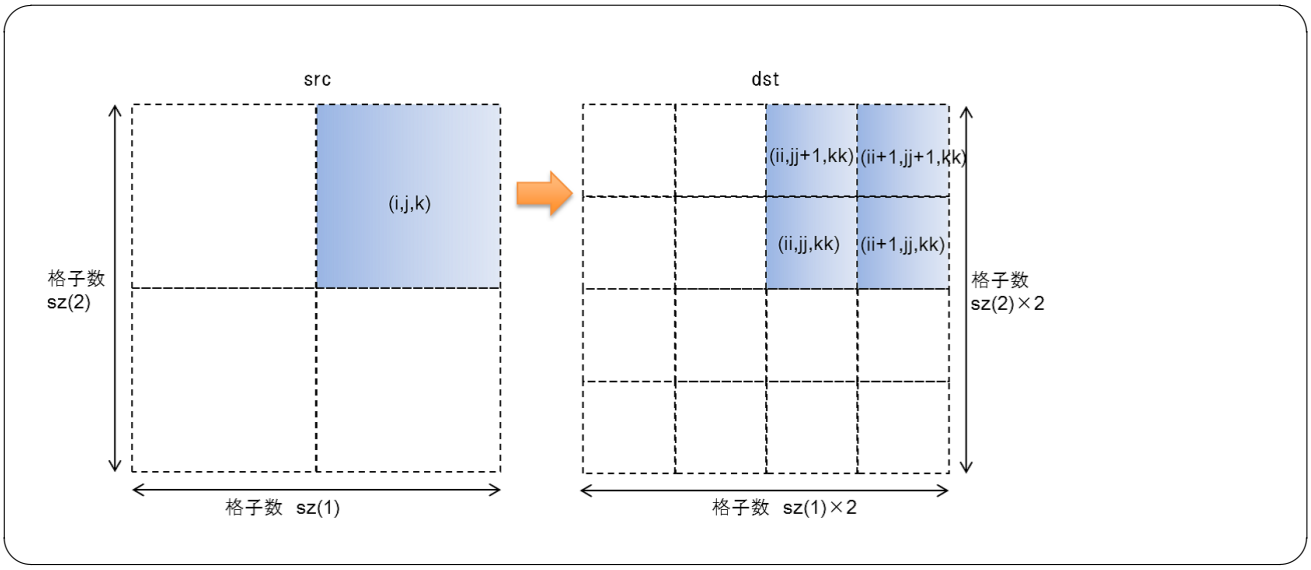

図 3.5　補間処理

cio_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
```

cio_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 "cio_DFI.h"
int main( int argc, char **argv )
{
  //CIO のエラーコード
  CIO::E_CIO_ERRORCODE ret = CIO::E_CIO_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 CIO::E_CIO_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
  cio_DFI* DFI_IN = cio_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 != CIO::E_CIO_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() != CIO::E_CIO_FLOAT32 ) {
    //データの型違い
    std::cerr << "Error Datatype unmatch." << std::endl;
    return CIO::E_CIO_ERROR;
  }

  //読込みフィールドデータの成分数を取得
  int ncomp=DFI_IN->GetNumComponent();

  //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==CIO::E_CIO_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*ncomp];
  //読込み配列のゼロクリア
  memset(d_v, 0, sizeof(float)*size*ncomp);
  //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 != CIO::E_CIO_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 CIO::E_CIO_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 "cio_DFI.h"
int main( int argc, char **argv )
{
  //CIO のエラーコード
  CIO::E_CIO_ERRORCODE ret = CIO::E_CIO_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 CIO::E_CIO_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
  cio_DFI* DFI_IN = cio_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 != CIO::E_CIO_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() != CIO::E_CIO_FLOAT32 ) {
    //データの型違い
    std::cerr << "Error Datatype unmatch." << std::endl;
    return CIO::E_CIO_ERROR;
  }
  //単位系の取得
  cio_UnitElem unit;
  ret=DFI_IN->GetUnitElem("Pressure",unit);
  if( ret==CIO::E_CIO_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 != CIO::E_CIO_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 CIO::E_CIO_SUCCESS;
}
```

3.3　出力機能

3.3.1　機能概要

CIO 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．

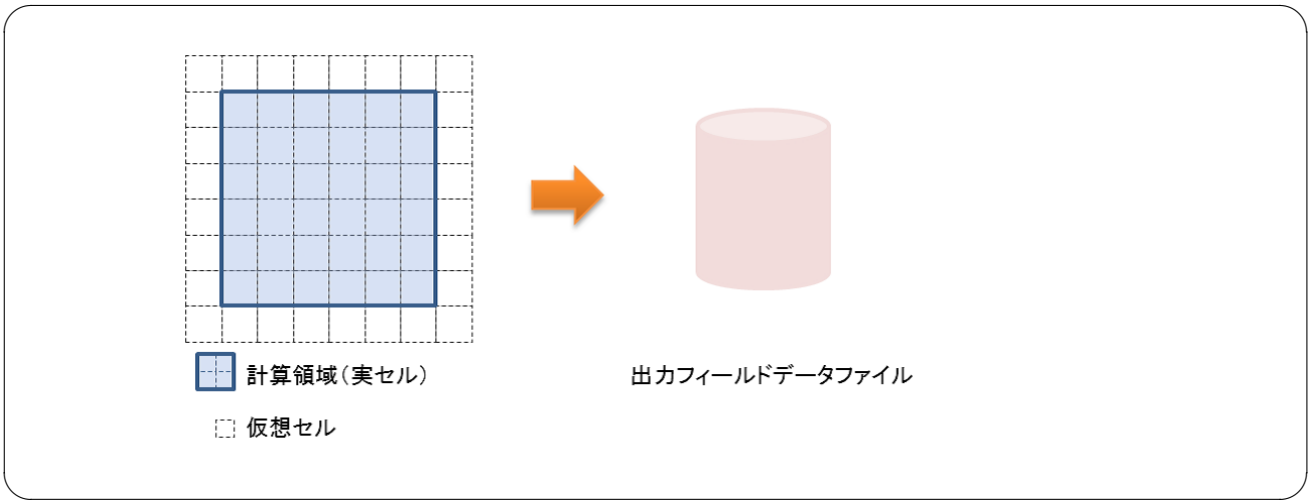

図 3.6　1 対 1 の出力

3.3.2　出力処理手順

CIO 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. proc.dfi ファイル出力 (3.3.5 章参照)
4. 出力インターバルの制御 (3.4 章参照)
5. フィールドデータファイル出力 (3.3.6 章参照)

3.3.3　出力用インスタンスのポインタ取得

cio_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，cio_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（float 型）

```
static cio_DFI* cio_DFI::WriteInit(const MPI_Comm comm,
                                   const std::string DfiName,
                                   const std::string Path,
                                   const std::string prefix,
                                   const CIO::E_CIO_FORMAT format,
                                   const int GCell,
                                   const CIO::E_CIO_DTYPE DataType,
                                   const CIO::E_CIO_ARRAYSHAPE ArrayShape,
                                   const int nComp,
                                   const std::string proc_fname,
                                   const int G_size[3],
                                   const float pitch[3],
                                   const float G_origin[3],
                                   const int division[3],
                                   const int head[3],
                                   const int tail[3],
                                   const std::string hostname,
                                   const CIO::E_CIO_ONOFF TSliceOnOff);
```

cio_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）
ArrayShape	[*input*]	フィールドデータの配列形状（表 3.5 参照）
nComp	[*input*]	フィールドデータの成分数（スカラーは 1，ベクトルは 3）
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）
G_origin	[*input*]	計算空間全体の原点座標値（3word の配列 float 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	cio_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
cioDFI *DFI_OUT_PRS = cio_DFI::WriteInit(,,,);
          :
      (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（double 型）

```
static cio_DFI* cio_DFI::WriteInit(const MPI_Comm comm,
                                   const std::string DfiName,
                                   const std::string Path,
                                   const std::string prefix,
                                   const CIO::E_CIO_FORMAT format,
                                   const int GCell,
                                   const CIO::E_CIO_DTYPE DataType,
                                   const CIO::E_CIO_ARRAYSHAPE ArrayShape,
                                   const int nComp,
                                   const std::string proc_fname,
                                   const int G_size[3],
                                   const double pitch[3],
                                   const double G_origin[3],
                                   const int division[3],
                                   const int head[3],
                                   const int tail[3],
                                   const std::string hostname,
                                   const CIO::E_CIO_ONOFF TSliceOnOff);
```

cio_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）
ArrayShape	[*input*]	フィールドデータの配列形状（表 3.5 参照）
nComp	[*input*]	フィールドデータの成分数（スカラーは 1，ベクトルは 3）
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 double 型）
G_origin	[*input*]	計算空間全体の原点座標値（3word の配列 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	cio_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
cioDFI *DFI_OUT_PRS = cio_DFI::WriteInit(,,,);
             :
        (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 CIO 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 cio_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 5.1.1 章「インデックスファイル (index.dfi) 仕様」参照．

単位系の登録

```
void
cio_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. 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
cio_DFI::SetTimeSliceFlag(const CIO::E_CIO_ONOFF ONOFF);
```

ONOFF	[*input*]	出力指示フラグ (表 3.2 参照)

3. DFI FileInfo の成分名を登録する

   DFI ファイルの FileInfo の仕様は 5.1.1 章「インデックスファイル (index.dfi) 仕様」参照．

DFI FileInfo の成分名を登録する

```
void
cio_DFI::setComponentVariable(int pcomp,
                              std::string compName);
```

pcomp	[*input*]	登録する成分名の位置
compName	[*input*]	登録する成分名

4. 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

```
CIO::E_CIO_ERRORCODE
cio_DFI::CheakReadRank(cio_Domain dfi_domain,
                       const int head[3],
                       const int tail[3],
                       CIO::E_CIO_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 3.8，3.9 を参照）	

3.3.5　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 cio_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（float 版）

```
CIO::E_CIO_ERRORCODE
cio_DFI::WriteProcDfiFile(const MPI_Comm comm,
                          bool out_host,
                          float* org);
```

proc.dfi ファイル（float 版）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
org	[*input*]	原点座標値	(※ 1)
戻り値	エラーコード（表 3.8，3.9 を参照）		

(※ 1)(float *)NULL 指定の場合 WriteInit で指定した原点座標値が出力される.

proc.dfi ファイルの出力（double 版）

```
CIO::E_CIO_ERRORCODE
cio_DFI::WriteProcDfiFile(const MPI_Comm comm,
                          bool out_host,
                          double* org);
```

proc.dfi ファイル（double 版）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
org	[*input*]	原点座標値	(※ 1)
戻り値	エラーコード（表 3.8，3.9 を参照）		

(※ 1)(double *)NULL 指定の場合 WriteInit で指定した原点座標値が出力される.

3.3.6　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 形式と BOV 形式ファイルです．（詳細は，5.1.3 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 cio_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

```
template<class T, class TimeT, class TimeAvrT>
CIO::E_CIO_ERRORCODE
cio_DFI::WriteData(const unsigned step,
                   TimeT time,
                   const int sz[3],
                   const int nComp,
                   const int gc,
                   T* val,
                   T* minmax,
                   bool force,
                   bool avr_mode,
                   unsigned &step_avr,
                   TimeAvrT &time_avr);
```

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）
nComp	[*input*]	出力するデータの成分数（スカラー：１，ベクトル：３）
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
		スカラーのとき　minmax[0]=min
		minmax[1]=max
		ベクトルのとき　minmax[0]=成分 1 の minX
		minmax[1]=成分 1 の maxX
		：
		minmax[2n-2]=成分 n の maxX
		minmax[2n-1]=成分 n の minX
		minmax[2n]=合成値の min
		minmax[2n+1]=合成値の max
force	[*input*]	強制出力指示　true ：インターバルに関係なく出力
avr_mode	[*input*]	平均ステップ，時間の出力指示 false：出力
step_avr	[*input*]	平均ステップ
time_avr	[*input*]	平均時刻
戻り値		エラーコード（表 3.8，3.9 を参照）

3.3.7　出力処理のサンプルコード

```
#include "cio_DFI.h"
int main( int argc, char **argv )
{
  //IO のエラーコード
  CIO::E_CIO_ERRORCODE ret = CIO::E_CIO_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 CIO::E_CIO_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= "vel";    ///<ベースファイル名
  int out_gc        = 0;        ///<出力仮想セル数
  int ncomp         = 3;        ///<データの成分数
  CIO::E_CIO_FORMAT format  = CIO::E_CIO_FMT_SPH; ///<出力フォーマット
  CIO::E_CIO_DTYPE datatype = CIO::E_CIO_FLOAT32; ///<データ型
  std::string proc_fname    = "proc.dfi";         ///<proc ファイル名
  std::string hostname      = "";                 ///<ホスト名
  CIO::E_CIO_ONOFF TimeSliceOnOff = CIO::E_CIO_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
  cio_DFI* DFI_OUT = cio_DFI::WriteInit(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
                                        CIO::E_CIO_NIJK, ///<配列形状
                                        ncomp,           ///<データの成分数
                                        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 CIO::E_CIO_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
                            (float *)NULL); ///<出力する原点座標値，NULL のときは WriteInit の値
  //配列のアロケート
  float *d_v = new float[size*ncomp];

  unsigned step=0;  ///<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;

  //成分名の登録
  DFI_OUT->setComponentVariable(0,"u");
  DFI_OUT->setComponentVariable(1,"v");
  DFI_OUT->setComponentVariable(2,"w");

  //フィールドデータの出力
  ret = DFI_OUT->WriteData(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
                           ncomp,  ///<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
                           minmax, ///<最小値，最大値
                           true,   ///<強制出力
                           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 != CIO::E_CIO_SUCCESS ) {
    //フィールドデータの出力失敗
    std::cerr << "Error WriteData." << 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 CIO::E_CIO_SUCCESS;
}
```

3.4　出力インターバルの制御

CIO の出力機能には，出力インターバルを制御する機能が実装されています．出力インターバルは出力ステップ番号による制御と出力時刻による制御を行うことが可能です．cio_DFI のインスタンスに対して，setIntervalStep メソッド (ステップ制御)，setIntervalTime メソッド (時刻制御) で設定します．出力インターバル制御がこれらのメソッドにより設定されていない場合は WriteData メソッドコール時に必ず出力されます．設定されている場合は WriteData メソッド内で自動的に出力インターバル制御を行います．

設定例

出力インターバル制御を行う場合，以下のパラメータを設定します．

- インターバル (ステップ/タイム)　：　出力インターバル
- ベース (ステップ/タイム)　：　インターバルを計算し始めるベースとなるステップもしくはタイム
- セッションスタート (ステップ/タイム)　：　実際に出力を開始するステップもしくはタイム
  出力インターバルとなるステップもしくはタイムであっても，このセッションスタートより前は出力されません．
- エンド (ステップ/タイム)　：　プログラムの最終ステップもしくはタイム
  エンドが指定されると，プログラムの最終ステップもしはタイムが出力インターバルで無い場合でも出力を行います．

パラメータの設定方法は 3.4.7 章参照．

3.4.1　インターバルステップ指定

以下は，出力インターバルにステップ 100 指定したときの例になります．

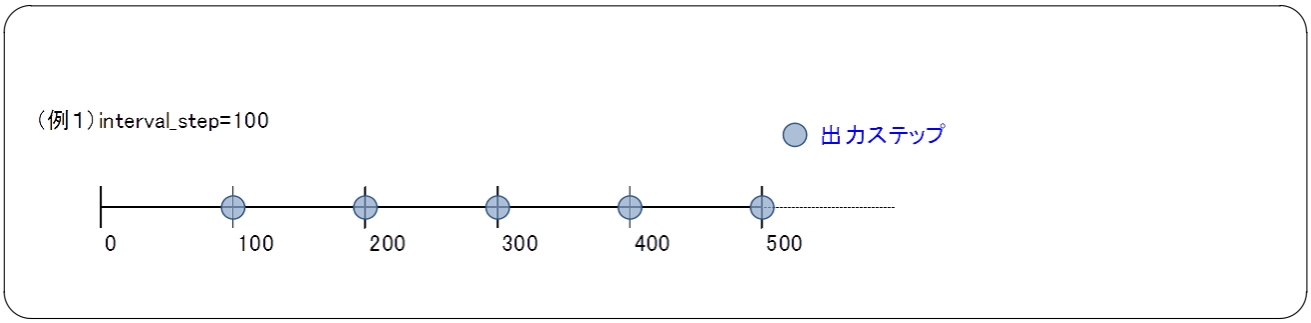

図 3.7　(例 1) interval_step=100

3.4.2　インターバルタイム指定

1. インターバルタイムが Δt で割り切れる
   以下は，出力インターバルにタイム 9.0，Δt 1.0 指定したときの例になります．

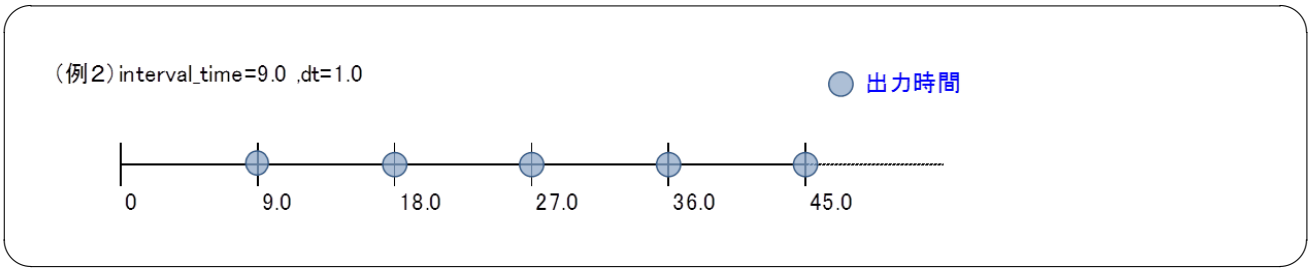

図 3.8　(例 2) interval_time=9.0，dt=1.0

2. インターバルタイムが Δt で割り切れない
   インターバルタイムが Δt で割り切れない場合は現時刻 (t) が出力時刻以下かつ出力時刻が t+Δt より小さい場合に出力されます．（下図)

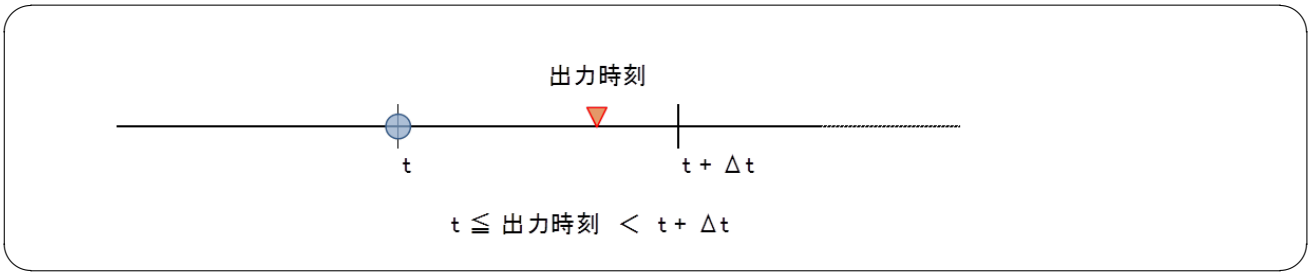

図 3.9　インターバルタイムと出力時刻の関係

以下は，出力インターバルにタイム 0.5，Δt 0.3 指定したときの例になります．

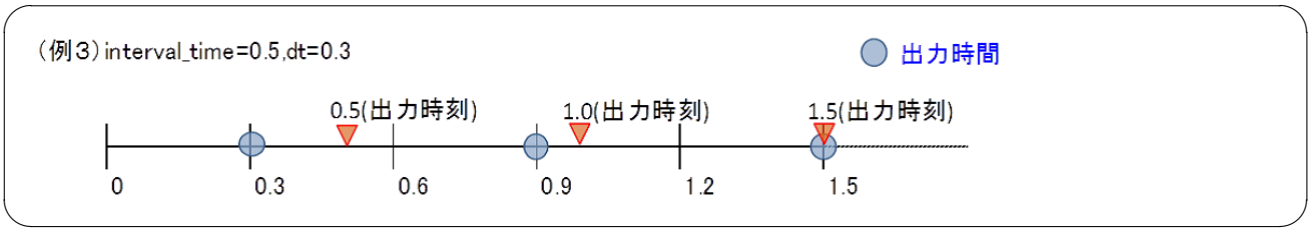

図 3.10　(例 3) interval_time=5.0，dt=0.3

3.4.3　セッション開始指定

以下は，出力インターバルにステップ 100，セッションスタートにステップ 150 指定したときの例になります．

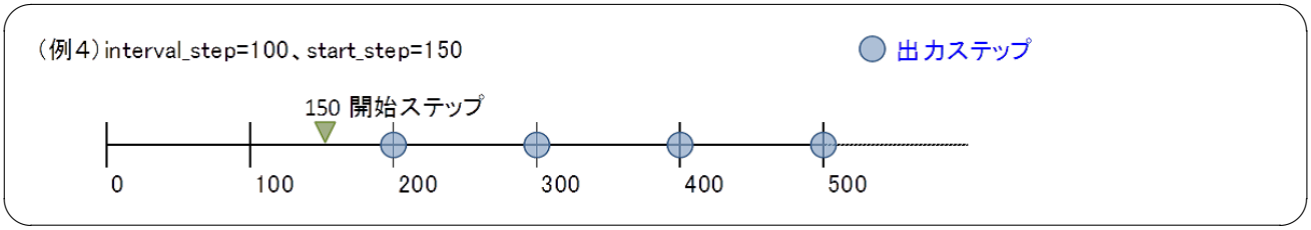

図 3.11　(例 4) interval_step=100，start_step=150

3.4.4　セッション終了指定

以下は，出力インターバルにステップ 100，エンドにステップ 450 指定したときの例になります．

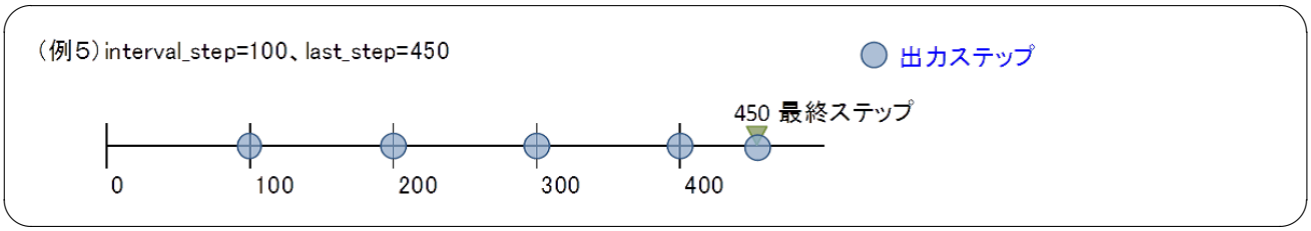

図 3.12　(例 5) interval_step=100，last_step=450

3.4.5　ベース指定

以下は，出力インターバルにステップ 100，ベースにステップ 150 指定したときの例になります．

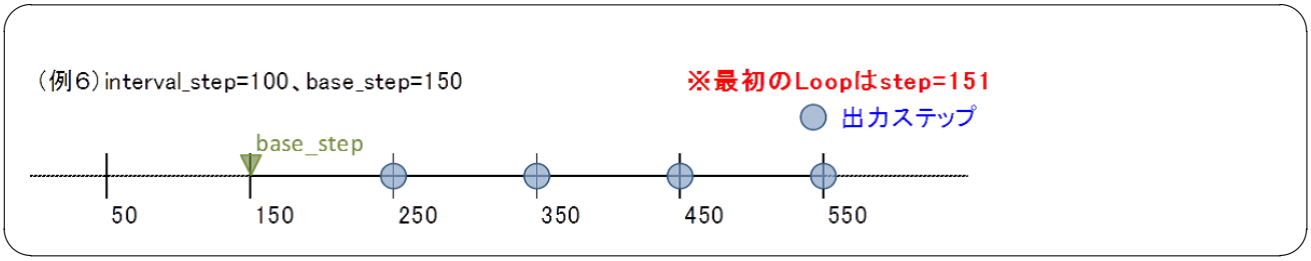

図 3.13　(例 1) interval_step=100, base_step=150

3.4.6　ベースとセッションスタート指定

以下は，出力インターバルにステップ 100，ベースにステップ 150，セッションスタートにステップ 300 指定したときの例になります．

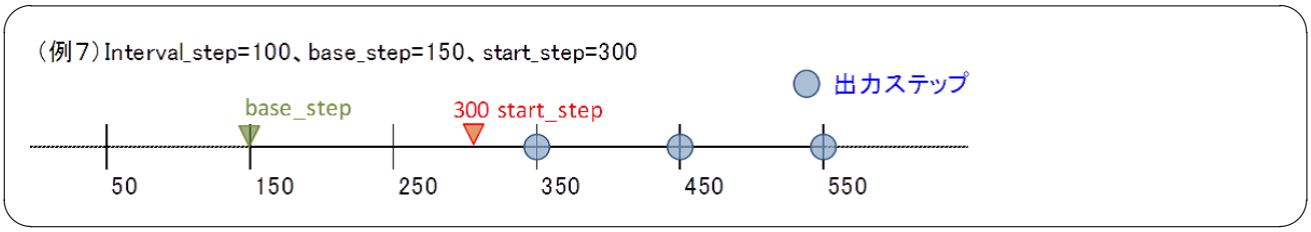

図 3.14　(例 1) interval_step=100 base_step=150 start_step=300

3.4.7　API インターフェイス

出力インターバルの処理を行うメソッドは cio_DFI.h 内で次のように定義されています．

1. 出力インターバルステップの登録

出力インターバルステップの登録

```
void
cio_DFI::setIntervalStep(int interval_step,
                         int base_step =0,
                         int start_step=0,
                         int last_step =-1);
```

出力インターバルステップの登録を行います．

interval_step	[*input*]	インターバルステップ	
base_step	[*input*]	基準となるステップ	デフォルト：0 ステップ（※ 1）
start_step	[*input*]	セッション開始ステップ	デフォルト：0 ステップ（※ 2）
last_step	[*input*]	セッション終了ステップ	デフォルト：-1 （※ 3）

(※ 1) 省略可．ただし省略した場合 start_step，last_step は無効．
(※ 2) 省略可．ただし省略した場合 last_step は無効．
(※ 3) インターバル間隔に関係なく最終ステップを出力させる場合に最終ステップ番号を指定．省略した場合最終ステップがインタバール間隔と合わない場合出力されない．

2. 出力インターバルタイムの登録

出力インターバルタイムの登録

```
void
cio_DFI::setIntervalTime(double interval_time,
                         double dt,
                         double base_time =0.0,
                         double start_time=0.0,
                         double last_time =-1.0);
```

出力インターバルタイムの登録を行います.

interval_time	[*input*]	インターバルステップ	
dt	[*input*]	計算の時間間隔	
base_time	[*input*]	基準となるタイム	デフォルト：0.0 ステップ（※ 1）
start_time	[*input*]	セッション開始タイム	デフォルト：0.0 ステップ（※ 2）
last_time	[*input*]	セッション終了タイム	デフォルト：-1.0 （※ 3）

(※ 1) 省略可．ただし省略した場合 start_time,last_time は無効.
(※ 2) 省略可．ただし省略した場合 last_taime は無効.
(※ 3) ンターバル間隔に関係なく最終タイムを出力させる場合に最終タイムを指定．省略した場合最終タイムがインタバール間隔と合わない場合出力されない.

3. インターバル時間の無次元化

インターバル時間の無次元化

```
bool
cio_DFI::normalizeTime(const double scale);
```

インターバルの計算に使われる全ての時間 (base_time, interval_time, start_time, last_time, DeltaT) をスケール値で無次元化を行います.

sclae	[*input*]	スケール値
戻り値		インターバルの登録がステップの場合は false を返し，無次元化は行われない.

4. base_time の無次元化

base_time の無次元化

```
void
cio_DFI::normalizeBaseTime(const double scale);
```

インターバルの計算に使われ base_time をスケール値で無次元化を行います.

sclae	[*input*]	スケール値

5. interval_time の無次元化

interval_time の無次元化

```
void
cio_DFI::normalizeIntervalTime(const double scale);
```

インターバルの計算に使われ Interval_time をスケール値で無次元化を行います.
 sclae　[*input*]　スケール値

6. start_time の無次元化

start_time の無次元化

```
void
cio_DFI::normalizeStartTime(const double scale);
```

インターバルの計算に使われ Start_time をスケール値で無次元化を行います.
 sclae　[*input*]　スケール値

7. last_time の無次元化

last_time の無次元化

```
void
cio_DFI::normalizeLastTime(const double scale);
```

インターバルの計算に使われ Last_time をスケール値で無次元化を行います.
 sclae　[*input*]　スケール値

8. 計算時間間隔 (DeltaT) の無次元化

DeltaT の無次元化

```
void
cio_DFI::normalizeDeltaT(const double scale);
```

インターバルの計算に使われ計算時間懸隔 (DeltaT) をスケール値で無次元化を行います.
 sclae　[*input*]　スケール値

第 4 章

ステージングツール

この章では，CIOlib 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 CIO 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バッチプログラムです.

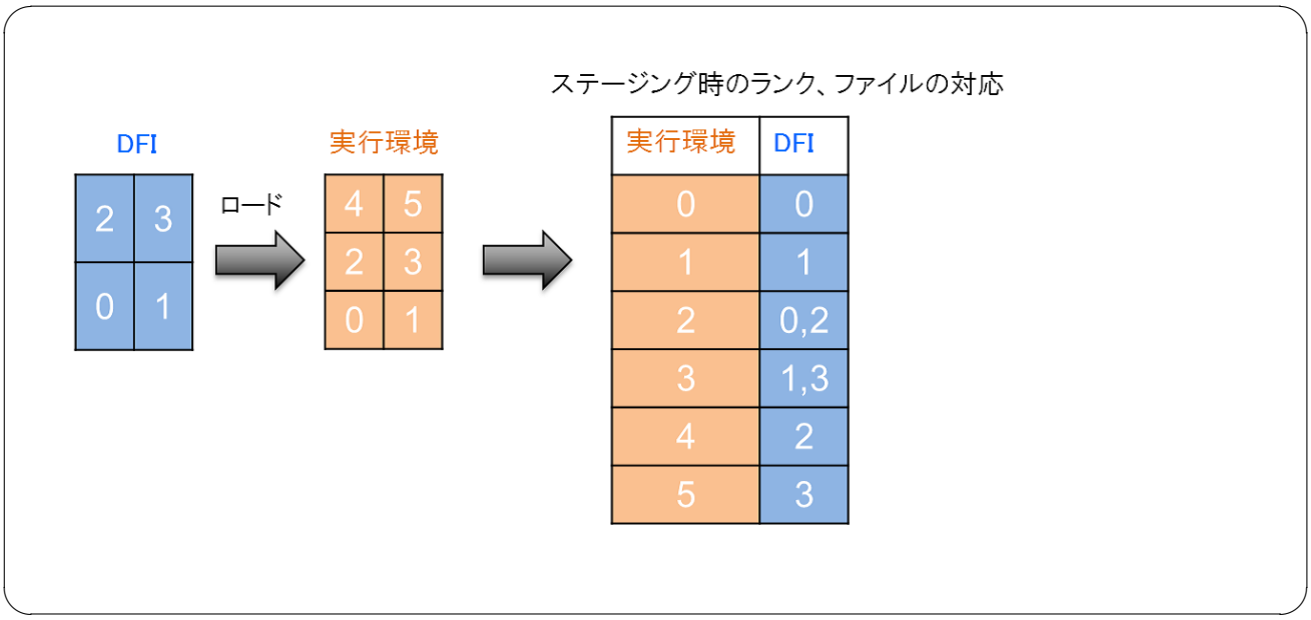

図 4.1　ステージング

4.1.2　ステージングツールのインストール

frm は，CIO 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．CIO パッケージのビルドは 2.1.2 章,2.1.7 章参照.

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 [-s stepNo] [-o outDir] DFIfile...
```

引数の説明

-i proc.txt (必須)

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 5.1.5 参照.

-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... (必須)

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.

(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

ファイル仕様

CIOlib で使用しているファイルの仕様について説明します.

5.1 ファイル仕様

5.1.1 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
{
  DirectoryPath      = "./"          // フィールドデータの存在するディレクトリ（※ 1)
  TimeSliceDirectory = "off"         // 時刻毎のディレクトリ作成オプション
  Prefix             = "vel"         // ベースファイル名（※ 2)
  FileFormat         = "sph"         // ファイルタイプ，拡張子（※ 2)
  GuideCell          = 0             // 仮想セル数
  DataType           = "Float32"     // データタイプ（※ 3)
  Endian             = "little"      // データのエンディアン（※ 4)
  ArrayShape         = "nijk"        // 配列形状（※ 4)
  Component          = 3             // 成分数（スカラーは不要）（※ 5)
  Variable[@]  {name = "u"}          // 成分名（Component 個）
  Variable[@]  {name = "v"}          //
  Variable[@]  {name = "w"}          //
}
```

(※ 1) index.dfi ファイルからの相対パス，もしくは絶対パス
(※ 2) ファイル名　並列時 [Prefix]_[ステップ番号:10 桁]_id[RankID:6 桁].[ext]
　　逐次時 [Prefix]_[ステップ番号:10 桁].[ext]
(※ 3) Int8,UInt8,Int16,Uint16,Int32,Uint32,Int64,Uint64,Float32,Float64
(※ 4) little,big, 省略時:実行プラットフォームと同じ
(※ 5) ijkn,nijk　ijkn:(imax,jmax,kmax,Component)
　　nijk:(Component,imax,jmax,kmax)

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 {   // u,v,w 合成値の min/max 値（Component>1 のときのみ）
      Min            = 0.0          // 最小値
      Max            = 0.0          // 最大値
    }
    MinMax{@} {      // Component 個
      Min            = 0.0          // u 最小値
      Max            = 0.0          // u 最大値
    }
    MinMax{@} {
      Min            = 0.0          // v 最小値
      Max            = 0.0          // v 最大値
    }
    MinMax{@} {
      Min            = 0.0          // w 最小値
      Max            = 0.0          // w 最大値
    }
    ・・・任意のアノテーション追加可能
  }
  Slice{@} {
    :
    :
  }
}
```

5.1.2　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)
}
```

(※ 1) index.dfi ファイルからの絶対パス，もしくは相対パス

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{@} {
    :
    :
  }
}
```

5.1.3　フィールドデータファイルの仕様

以下に，CIO 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 5.1 を参照)

表 5.1　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 5.2　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．以下の値をとります．

（※ 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． 以下の値をとります．

データ種別	svType の値
スカラーデータ	1
ベクトルデータ	2

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 5.3　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 5.4　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 5.5　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 5.6　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 5.7,5.8 を参照)

表 5.7　（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 5.8　（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 1 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(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 (0,0,0) の成分 1 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

5.1.4　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 5.9　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．

5.1.5　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
}
MPI(※ 2)
{
  NumberOfRank        = 128                   // プロセス数
}
Process(※ 2)
{
  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 (※ 3)
    TailIndex         =( 64, 64, 64 )         // 終点インデックス (※ 3)
  }
}
```

(※ 1) Domain タグは必須
　　ただし，ActiveSubdomainFile は任意
(※ 2) 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．
(※ 3) 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．

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．

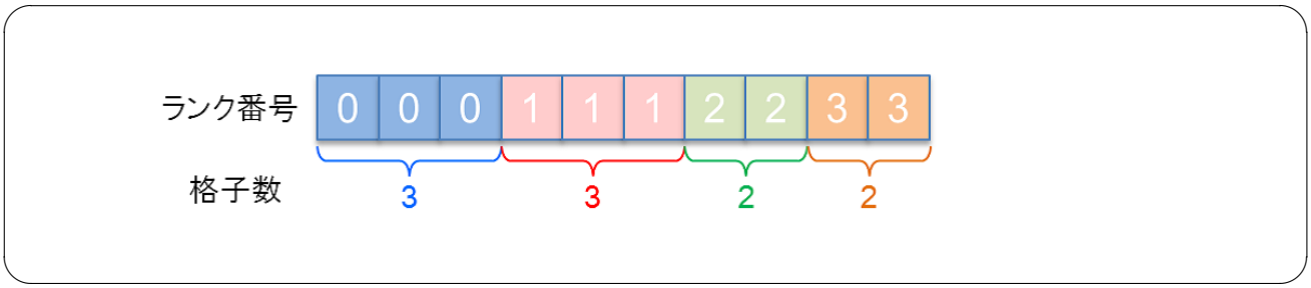

図 5.1　(例) 格子数 10，領域分割 4

5.1.6　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 {
  DirectoryPath      = "data"
  TimeSliceDirectory = "off"
  Prefix             = "vel"
  FileFormat         = "sph"
  GuideCell          = 0
  DataType           = "Float32"
  Endian             = "little"
  ArrayShape         = "nijk"
  Component          = 3
  Variable[@]{ name  = "u" }
  Variable[@]{ name  = "v" }
  Variable[@]{ name  = "w" }
}
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
    VectorMinMax {
      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
    VectorMinMax {
      Min = 2.018320e-09
      Max = 2.169154e-04
    }
    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
    }
  }
}
```

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 = ""
}
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)
  }
}
```

第 6 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.

6.1 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.

Version 1.3.8　　2013/10/10

- DFI ファイルの単位系の Format 変更に伴う修正

Version 1.3.7　　2013/10/02

- modify for intel mpi

Version 1.3.6　　2013/09/09

- ステージングツール (frm) の修正
  - CIO の修正に伴う見直し
- ビルドの一括化
- ステージングツール用の DFI 情報取得関数の追加

Version 1.3.5　　2013/08/09

- データ構造の見直し
  - クラス階層の修正
  - 配列クラスの修正
  - クラス、関数のテンプレート化

Version 1.3.4　　2013/07/20

- Change policy to display version info
  - generate Version no from configure

Version 1.3.3　　2013/06/27

- Change configure.ac
  - TP_CFLAGS=‘$TP_DIR/bin/tp-config –cflags‘
  - TP_LDFLAGS=‘$TP_DIR/bin/tp-config –libs‘
  - remove TP_LIBS from configure.ac & cio-config.in

Version 1.3.2　　2013/06/27

- Add description in INSTALL file.
- Correction of cio-config.in
- Change archive name to libCIO.a

Version 1.3.1　　2013/06/26

- TextParser のアーカイブ名の修正に対応

Version 1.3　　2013/06/25

- dfi ファイルの相対パス処理の修正
- refinment, MxN の読込み処理の再チェック

Version 1.2　　2013/06/10

- コピーライトの様式を統一，ソースとヘッダに挿入
- キーワード変更　”ActiveSubDomain” >> ”ActiveSubDomainFile”
- Version Info の導入
- Temerature の文字綴り修正
- ファイル出力の様式を整形（スペース，配置など）

- Bug fix :
  - `Write_Step(, int) >> Write_Step(, const unsigned)`
  - `Write_OutFileInfo() %d > %u`

Version 1.1　　2013/06/08

- 体裁とパッケージングを変更

Version 1.0　　2013/06/06

- リリース

第 7 章

Appendix

7.1　API メソッド一覧

以下に，CIO ライブラリが提供する API メソッドの一覧を示します．（表 7.1）

表 7.1　メソッド一覧（クラス名の無い C++ メソッドは cio_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
cio_FileInfo クラスポインタの取得	GetcioFileInfo	
cio_FilePath クラスポインタの取得	GetcioFilePath	
cio_Unit クラスポインタの取得	GetcioUnit	
cio_Domain クラスポインタの取得	GetcioDomain	
cio_MPI クラスポインタの取得	GetcioMPI	
cio_TimeSlice クラスポインタの取得	GetcioTimeSlice	
cio_Process クラスポインタの取得	GetcioProcess	
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 の成分数取得	GetNumComponent	
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 の成分名を登録する	setComponentVariable	
FileInfo の成分名を取得する	getComponentVariable	
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	
CIO のバージョン No の取り出し	getVersionInfo	static メソッド

表目次

図目次