

User Guide of CIOlib

Cartesian Input / Output Library

Ver. 1.5.2

Advanced Institute for Computational Science

RIKEN

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

February 2014





(c) Copyright 2012-2014

Advanced Institute for Computational Science, RIKEN. All rights reserved.

7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, 650-0047, JAPAN.

目次

第 1 章	CIOlib の概要	1
1.1	CIOlib	2
1.2	この文書について	2
1.2.1	書式について	2
1.2.2	動作環境	2
第 2 章	パッケージのビルド	3
2.1	パッケージのビルド	4
2.1.1	パッケージの構造	4
2.1.2	パッケージのビルド	5
2.1.3	configure スクリプトのオプション	8
2.1.4	configure 実行時オプションの例	9
2.1.5	cio-config コマンド	10
2.1.6	提供環境の作成	11
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	11
2.2	CIO ライブラリの利用方法	12
2.2.1	C++	12
第 3 章	API 利用方法	13
3.1	ユーザプログラムでの利用方法	14
3.1.1	cio.DFI.h のインクルード	14
3.1.2	マクロ, 列挙型, エラーコード	14
3.2	入力機能	19
3.2.1	機能概要	19
3.2.2	入力処理手順	21
3.2.3	DFI 情報の取得	22
3.2.4	DFI クラスポインタの取得	25
3.2.5	フィールドデータファイルの読み込み	27
3.2.6	リファインメントデータ補間メソッド	29
3.2.7	入力処理のサンプルコード	33
3.3	出力機能	36
3.3.1	機能概要	36
3.3.2	出力処理手順	36
3.3.3	出力用インスタンスのポインタ取得	36
3.3.4	DFI 情報の追加登録	39
3.3.5	proc.dfi ファイル出力	40

3.3.6	フィールドデータファイル出力	41
3.3.7	出力処理のサンプルコード	42
第 4 章	ステージングツール	44
4.1	ステージングツール	45
4.1.1	機能概要	45
4.1.2	ステージングツールのインストール	45
4.1.3	使用方法	46
	コマンド引数	46
	引数の説明	46
	実行例	47
第 5 章	並列分散ファイルコンバータ	50
5.1	並列分散ファイルコンバータ	51
5.1.1	機能概要	51
	ファイル形式変換機能	51
	M 対 M データの変換機能	51
	M 対 1 データの変換機能	51
	M 対 N データの変換機能	51
5.1.2	FCONV のインストール	51
5.1.3	使用方法	51
	コマンド引数	52
	引数の説明	52
	実行例	52
5.1.4	出力形式	52
5.1.5	ファイルフォーマット毎の対応データ型	52
5.1.6	ファイルフォーマット毎の対応配列型	54
5.1.7	定義点	54
	ファイル形式毎の定義点	54
	格子点への補間	54
5.1.8	ファイルフォーマット毎の出力ファイル	55
5.1.9	間引き	55
5.1.10	ファイル割振り	56
	step 基準	56
	rank 基準	56
第 6 章	ファイル仕様	58
6.1	ファイル仕様	59
6.1.1	インデックスファイル (index.dfi) 仕様	59
6.1.2	プロセス情報ファイル (proc.dfi) 仕様	60
6.1.3	フィールドデータファイルの仕様	62
	SPH 形式	62
	BOV 形式	66
6.1.4	サブドメイン情報ファイルの仕様	67
6.1.5	DFI ファイルのサンプル	67

	index.dfi ファイルのサンプル	67
	proc.dfi ファイルのサンプル	68
6.2	ファイル仕様 (ツール)	70
6.2.1	ステージング用領域分割情報ファイルの仕様	70
6.2.2	並列分割ファイルコンバータ用入力ファイルの仕様	71
第 7 章	アップデート情報	74
7.1	アップデート情報	75
第 8 章	Appendix	77
8.1	API メソッド一覧	78

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)
- ・ 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: SPH,BOV → SPH,BOV,PLOT3D,AVS,VTK)

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

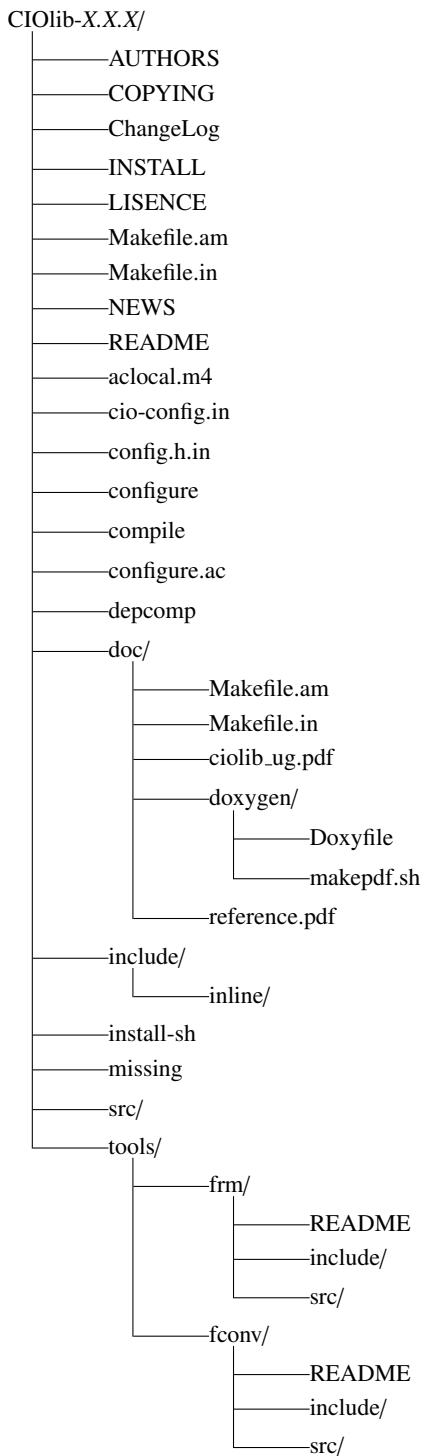
2.1.1 パッケージの構造

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。



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
ファイルのリ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 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/libCI0.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、

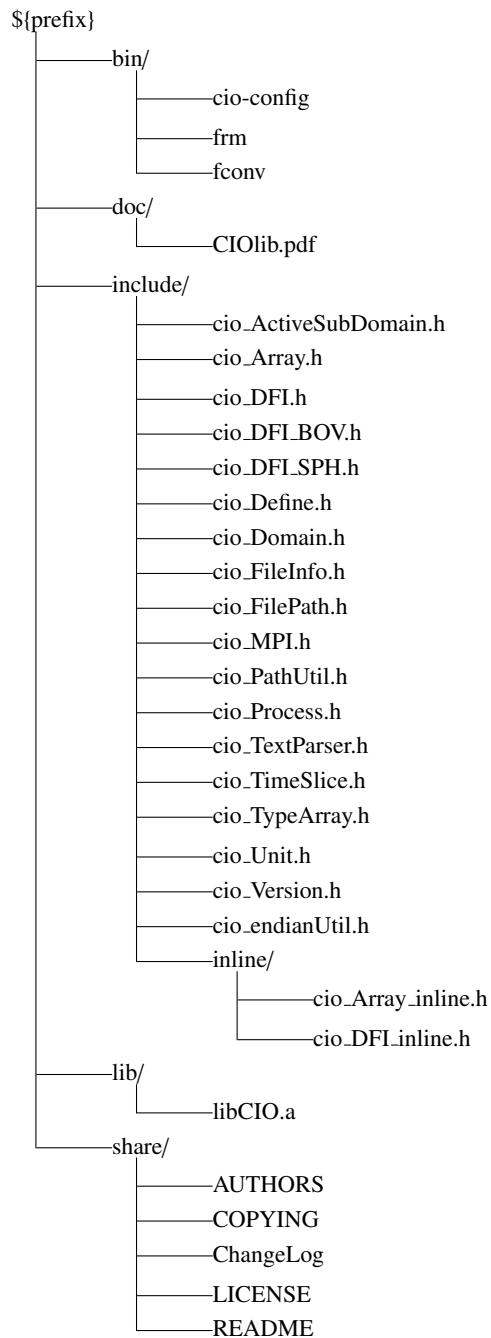
```
$ su
```

```
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
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。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

必要があります。

`--with-mpi=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。CIOlib 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。

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`

CPM ライブラリ : `/usr/local/cpmlib`

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-mpi=/usr/local/openmpi \  
               --with-parser=/usr/local/textparser \  
               --with-cpm=/usr/local/cpmlib \  
               --with-frm=yes
```

```
--with-cpm=/usr/local/cpmlib
CXX=icpc \
CXXFLAGS=-O3 \
F90=ifort \
F90FLAGS=-O3
```

・ 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
CPM ライブラリ : /home/usreXXXX/cpmlib
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-cpm=/home/usreXXXX/cpmlib \
CXX=mpiFCCpx \
CXXFLAGS=-Kfast \
F90=mpifrtpx \
F90FLAGS=-Kfast
```

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、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

```
CI0lib-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/CIOlib_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。

2.2 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。以下の指定では、並列 CIO ライブラリ、ファイル変換ツール、frm ツールがインストールされます。

```
$ ./configure --prefix=/usr/local/CIOlib \  
    --with-frm=yes \  
    --with-parser=/usr/local/textparser \  
    --with-cpm=/usr/local/cpmlib \  
    CXX=mpicxx \  
    CXXFLAGS=-O3 \  
    F90=mpif90 \  
    F90FLAGS=-O3
```

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	"dat"	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_AVS	2	AVS 形式
E_CIO_PLOT3D	3	PLOT3D 形式
E_CIO_VTK	4	VTK 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_ENDIANATYPE 列挙型

E_CIO_ENDIANATYPE 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。

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_LITTLE	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

- 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み込み形式を指定するフラグとして使われます。

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_OUTPUT_TYPE 列挙型

E_CIO_OUTPUT_TYPE 列挙型は、cio.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出力形式を指定するフラグとして使われます。

表 3.8 E_CIO_OUTPUT_TYPE 列挙型

E_CIO_OUTPUT_TYPE 要素	値	意味
E_CIO_OUTPUT_TYPE_DEFAULT	-1	デフォルト (binary)
E_CIO_OUTPUT_TYPE_ASCII	0	ascii 形式
E_CIO_OUTPUT_TYPE_BINARY	1	binary 形式
E_CIO_OUTPUT_TYPE_FBINAR	2	Fortran Binary 形式

- E_CIO_OUTPUT_FNAME 列挙型

E_CIO_OUTPUT_FNAME 列挙型は、cio.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_CIO_OUTPUT_FNAME 列挙型

E_CIO_OUTPUT_FNAME 要素	値	意味
E_CIO_OUTPUT_FNAME_DEFAULT	-1	デフォルト (step_rank)
E_CIO_OUTPUT_FNAME_STEP_RANK	0	step_rank
E_CIO_OUTPUT_FNAME_RANK_STEP	1	rank_step

- E.CIO_ERRORCODE 列挙型

E.CIO_ERRORCODE 列挙型は, cio.Define.h で表 3.10, 3.11 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 3.10 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 読みみエラー
E.CIO.ERROR.READ.DFI_GLOBALREGION	1001	DFI GlobalRegion 読みみエラー
E.CIO.ERROR.READ.DFI_GLOBALVOXEL	1002	DFI GlobalVoxel 読みみエラー
E.CIO.ERROR.READ.DFI_GLOBALDIVISION	1003	DFI GlobalDivison 読みみエラー
E.CIO.ERROR.READ.DFI_DIRECTORYPATH	1004	DFI DirectoryPath 読みみエラー
E.CIO.ERROR.READ.DFI_TIMESLICEDIRECTORY	1005	DFI TimeSliceDirectoryPath 読みみエラー
E.CIO.ERROR.READ.DFI_PREFIX	1006	DFI Prefix 読みみエラー
E.CIO.ERROR.READ.DFI_FILEFORMAT	1007	DFI FileFormat 読みみエラー
E.CIO.ERROR.READ.DFI_GUIDECCELL	1008	DFI GuideCell 読みみエラー
E.CIO.ERROR.READ.DFI_DATATYPE	1009	DFI DataType 読みみエラー
E.CIO.ERROR.READ.DFI_ENDIAN	1010	DFI Endian 読みみエラー
E.CIO.ERROR.READ.DFI_ARRAYSHAPE	1011	DFI ArrayShape 読みみエラー
E.CIO.ERROR.READ.DFI_COMPONENT	1012	DFI Component 読みみエラー
E.CIO.ERROR.READ.DFI_FILEPATH.PROCESS	1013	DFI FilePath/Process 読みみエラー
E.CIO.ERROR.READ.DFI_NO_RANK	1014	DFI Rank 要素なし
E.CIO.ERROR.READ.DFI_ID	1015	DFI ID 読みみエラー
E.CIO.ERROR.READ.DFI_HOSTNAME	1016	DFI HoatName 読みみエラー
E.CIO.ERROR.READ.DFI_VOXELSIZE	1017	DFI VoxelSize 読みみエラー
E.CIO.ERROR.READ.DFI_HEADINDEX	1018	DFI HeadIndex 読みみエラー
E.CIO.ERROR.READ.DFI_TAILINDEX	1019	DFI TailIndex 読みみエラー
E.CIO.ERROR.READ.DFI_NO_SLICE	1020	DFI TimeSlice 要素なし
E.CIO.ERROR.READ.DFI_STEP	1021	DFI Step 読みみエラー
E.CIO.ERROR.READ.DFI_TIME	1022	DFI Time 読みみエラー
E.CIO.ERROR.READ.DFI_NO_MINMAX	1023	DFI MinMax 要素なし
E.CIO.ERROR.READ.DFI_MIN	1024	DFI Min 読みみエラー
E.CIO.ERROR.READ.DFI_MAX	1025	DFI Max 読みみエラー
E.CIO.ERROR.READ.DFI_DFITYPE	1026	DFI DFIType 読みみエラー
E.CIO.ERROR.READ.DFI_FIELDFILENAMEFORMAT	1027	DFI FieldFilenameFormat 読みみエラー
E.CIO.ERROR.READ.INDEXFILE_OPENERERROR	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 読みみエラー
E.CIO.ERROR.READ.FILEPATH	1053	FilePath 読みみエラー
E.CIO.ERROR.READ.UNIT	1054	UNIT 読みみエラー
E.CIO.ERROR.READ.TIMESLICE	1055	TimeSlice 読みみエラー
E.CIO.ERROR.READ.PROCFILE_OPENERERROR	1056	Proc ファイルオープンエラー
E.CIO.ERROR.READ.DOMAIN	1057	Domain 読みみエラー
E.CIO.ERROR.READ.MPI	1058	MPI 読みみエラー
E.CIO.ERROR.READ.PROCESS	1059	Process 読みみエラー
E.CIO.ERROR.READ.FIELDDATA_FILE	1900	フィールドデータファイル読みみエラー
E.CIO.ERROR.READ.SPH_FILE	2000	SPH ファイル読みみエラー
E.CIO.ERROR.READ.SPH_REC1	2001	SPH ファイルレコード 1 読みみエラー
E.CIO.ERROR.READ.SPH_REC2	2002	SPH ファイルレコード 2 読みみエラー

表 3.11 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 読みエラー
E_CIO_ERROR_READ_SPH_REC4	2004	SPH ファイルレコード 4 読みエラー
E_CIO_ERROR_READ_SPH_REC5	2005	SPH ファイルレコード 5 読みエラー
E_CIO_ERROR_READ_SPH_REC6	2006	SPH ファイルレコード 6 読みエラー
E_CIO_ERROR_READ_SPH_REC7	2007	SPH ファイルレコード 7 読みエラー
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 Format 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_OPENERERROR	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_OPENERERROR	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 データの読み込み、 $M \times N$ データの読み込み、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) の読み込みの 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、読み込み処理を行います。

- 同一格子密度での 1 対 1 の読み込み

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み込みます。

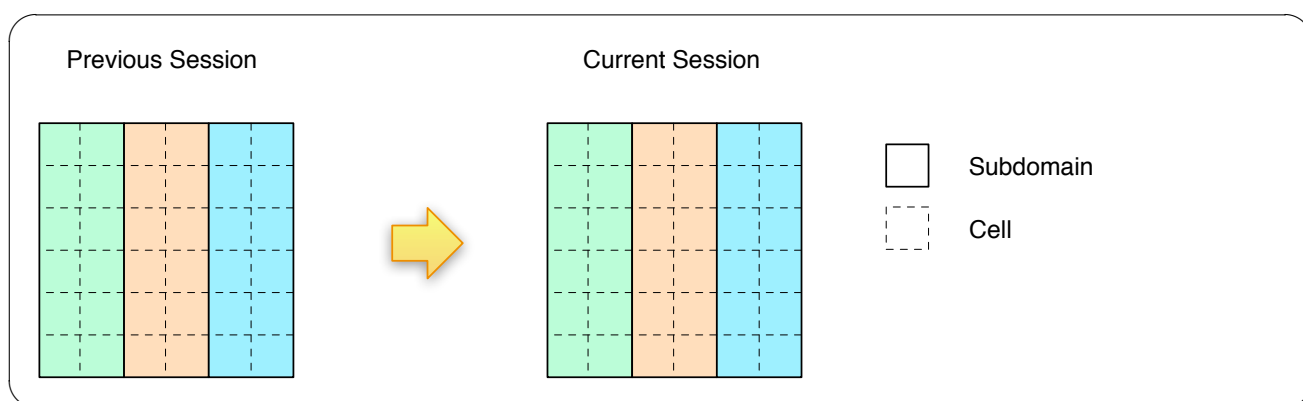


図 3.1 同一格子密度での 1 対 1 読み込み

- 同一格子密度での M 対 N の読み込み

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み込みます。

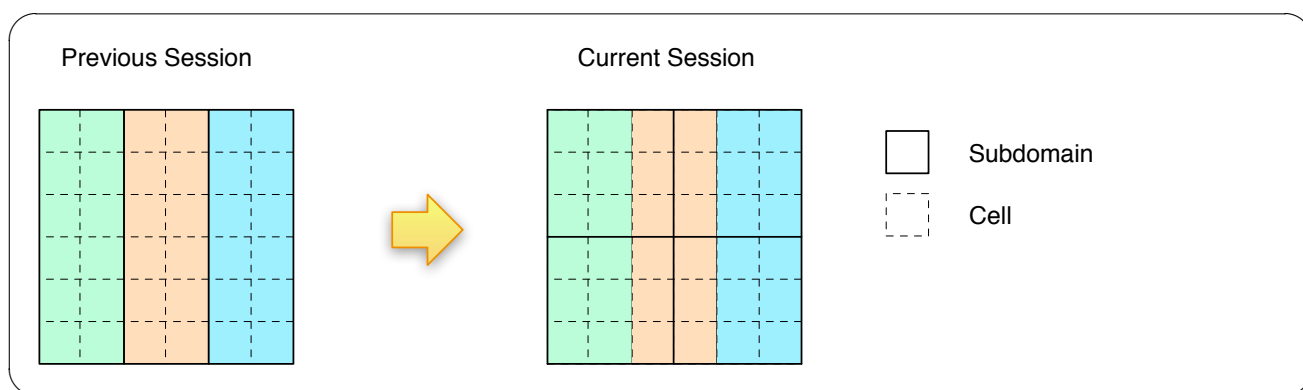


図 3.2 同一格子密度での M 対 N 読み込み

- リファインメントデータで1対1の読み込み

格子が1段階細かい格子（1：2）で、読み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み込み、補間処理（1）をします。

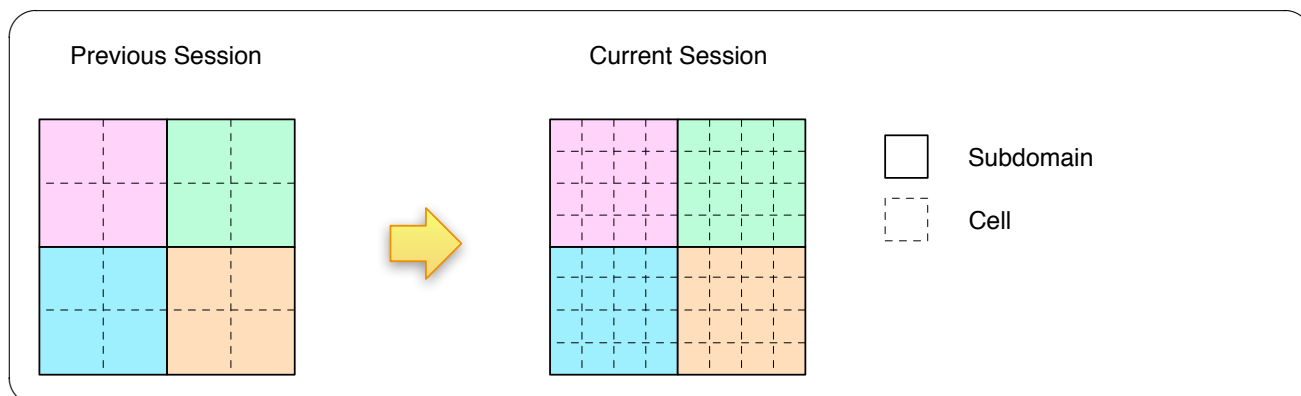


図 3.3 リファインメントで1対1読み込み

- リファインメントデータでM対Nの読み込み

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み込み、補間処理（1）をします。

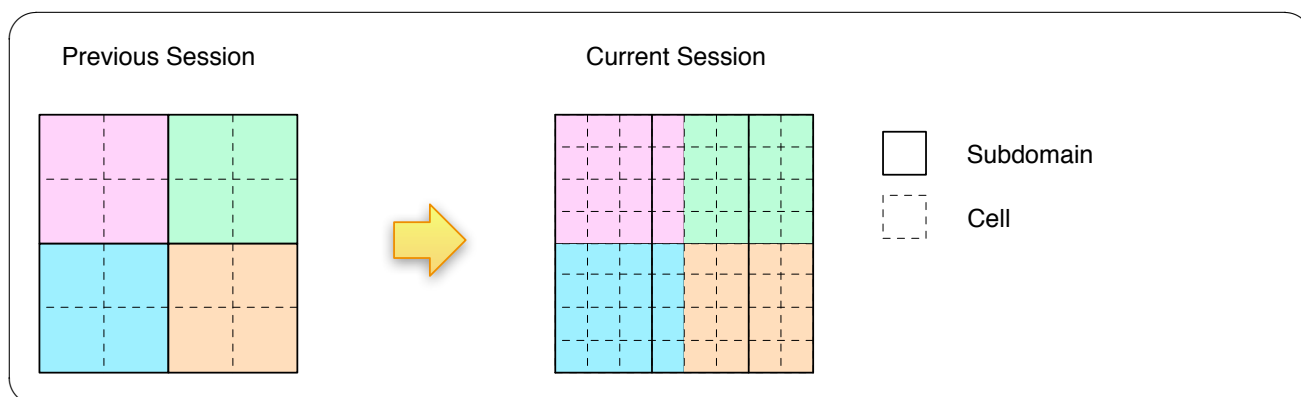


図 3.4 リファインメントでM対N読み込み

- （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み込み、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. 読み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み込み (3.2.5 章参照)

(注) 読み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。

読み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 コミュニケーター
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		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
cio_DFI *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 6.1.2 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 GlobalDivision(計算領域の分割数) の取得

DFI ファイルの Domain の仕様は 6.1.2 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 6.1.1 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 6.1.1 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 3.10, 3.11 を参照)

10. DFI TimeSlice の minmax 値を取得

DFI ファイルの TimeSlice の仕様は 6.1.1 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 3.10, 3.11 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 6.1.1 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 3.10, 3.11 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 6.1.1 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 3.10, 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 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、6.1.3 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 3.10, 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み込みフラグ (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]	読み込み先の配列のポインタ
step	[input]	読み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]	読み込んだ時間
mode	[input]	平均時間, 平均化したステップ読み込みフラグ (false : 読み込む, true : 読み込まない)
step_avr	[output]	読み込んだ平均化したステップ
time_avr	[output]	読み込んだ平均時間
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

CIO のリファインメントデータの読み込み処理では、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

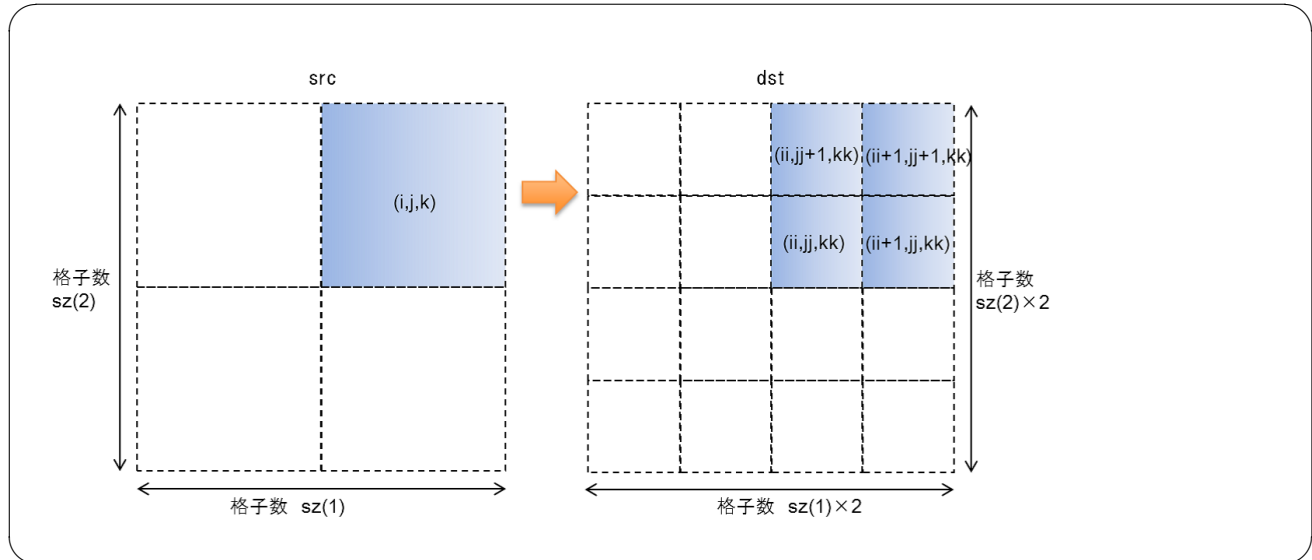


図 3.5 補間処理

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み込む

```

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
    //読み込み配列のサイズ
    size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

    //読み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, ///

```

```

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

//読み込み配列のアロケート
float *d_v = new float[size*ncomp];
//読み込み配列のゼロクリア
memset(d_v, 0, sizeof(float)*size*ncomp);
//読み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;        ///

```

2. 読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

//読み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, ///

```


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。

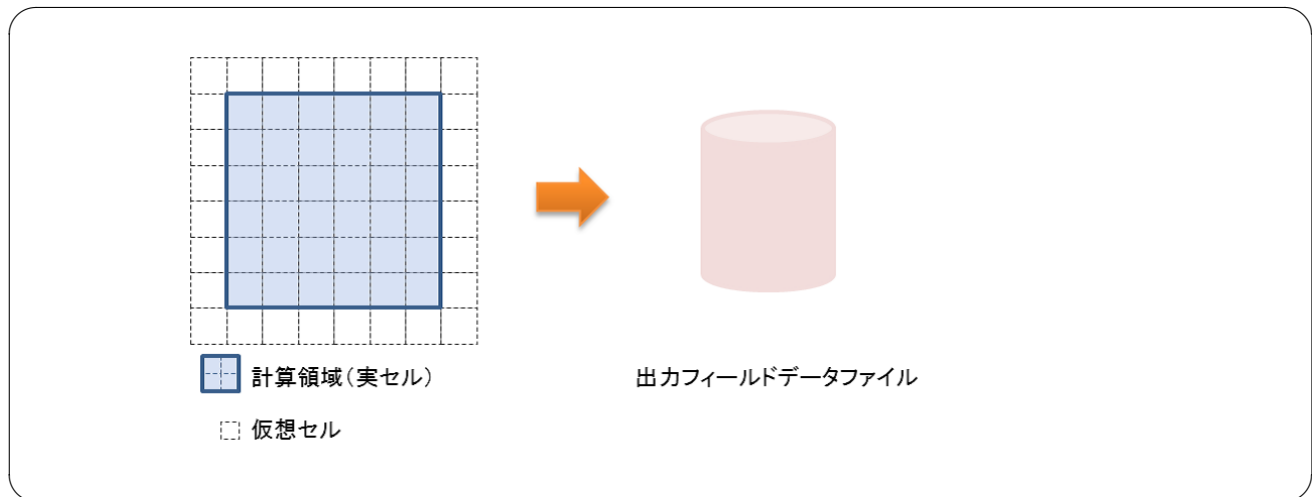


図 3.6 1対1の出力

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 (??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 コミュニケーター
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 コミュニケーター
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 6.1.1 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 6.1.1 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. 読み込みランクリストを登録する

読み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み込み方法 (表 3.7 参照)
readRankList	[output]	読み込みランクリスト
戻り値		エラーコード (表 3.10, 3.11 を参照)

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

```
CIO::E_CIO_ERRORCODE
cio_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)

3.3.6 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、6.1.3 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 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
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 3.10, 3.11 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";          ///

```

```

    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, ///

```


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。

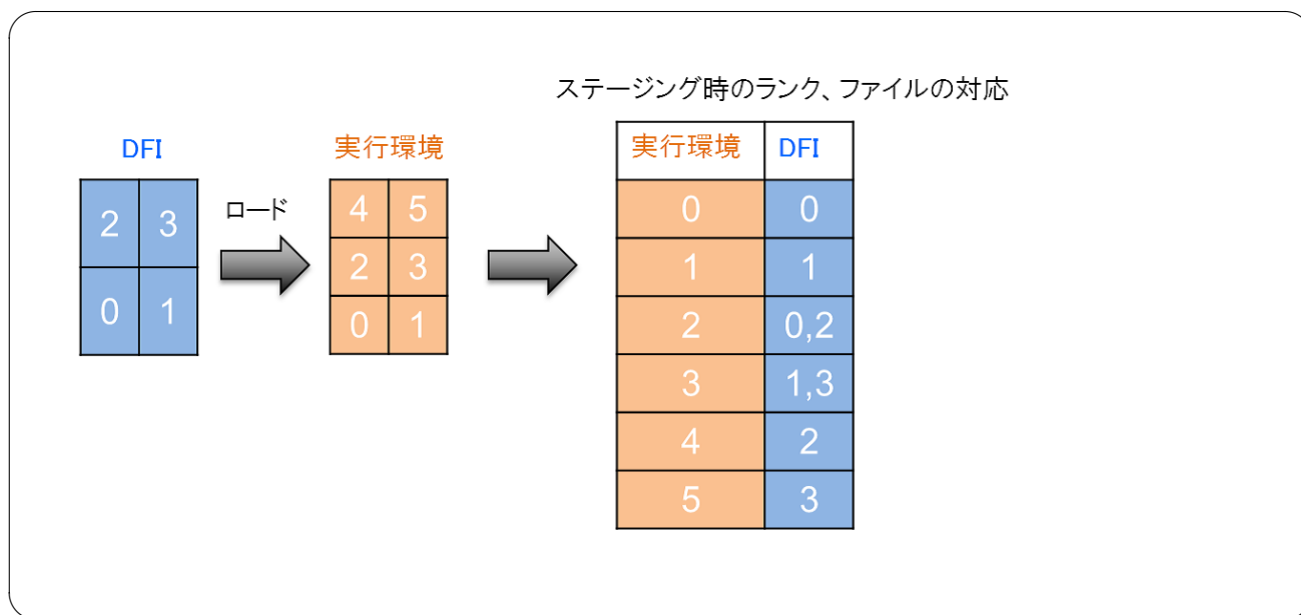


図 4.1 ステージング

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、CIOlib 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

CIOlib 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

CIO 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 6.2.1 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_00000000100_id000000.sph  
  prs_proc.dfi                          <--proc.dfi からコピーされる  
  vel.dfi                                <--DirectoryPath="./"  
  vel_00000000100_id000000.sph  
  vel_proc.dfi                          <--proc.dfi からコピーされる
```

```
hoge/000001/  
  prs.dfi  
  prs_00000000100_id000001.sph  
  prs_proc.dfi  
  vel.dfi  
  vel_00000000100_id000001.sph  
  vel_proc.dfi
```

```
hoge/000002/  
  prs.dfi  
  prs_00000000100_id000000.sph  
  prs_proc.dfi  
  vel.dfi  
  vel_00000000100_id000000.sph  
  vel_proc.dfi
```

```
hoge/000003/  
  prs.dfi  
  prs_00000000100_id000001.sph  
  prs_proc.dfi  
  vel.dfi  
  vel_00000000100_id000001.sph  
  vel_proc.dfi
```

```
hoge/000004/  
  prs.dfi  
  prs_00000000100_id000002.sph
```

```
prs_proc.dfi  
vel.dfi  
vel_00000000100_id0000002.sph  
vel_proc.dfi
```

```
hoge/0000005/  
prs.dfi  
prs_00000000100_id0000003.sph  
prs_proc.dfi  
vel.dfi  
vel_00000000100_id0000003.sph  
vel_proc.dfi
```

```
hoge/0000006/  
prs.dfi  
prs_00000000100_id0000002.sph  
prs_proc.dfi  
vel.dfi  
vel_00000000100_id0000002.sph  
vel_proc.dfi
```

```
hoge/0000007/  
prs.dfi  
prs_00000000100_id0000003.sph  
prs_proc.dfi  
vel.dfi  
vel_00000000100_id0000003.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、CIOLib 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 形式、BOV 形式のファイルから SPH、BOV、PLOT3D、AVS、VTK 形式へ変換する機能です。

M 対 M データの変換機能

CIOlib 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

CIOlib 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

CIOlib 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。

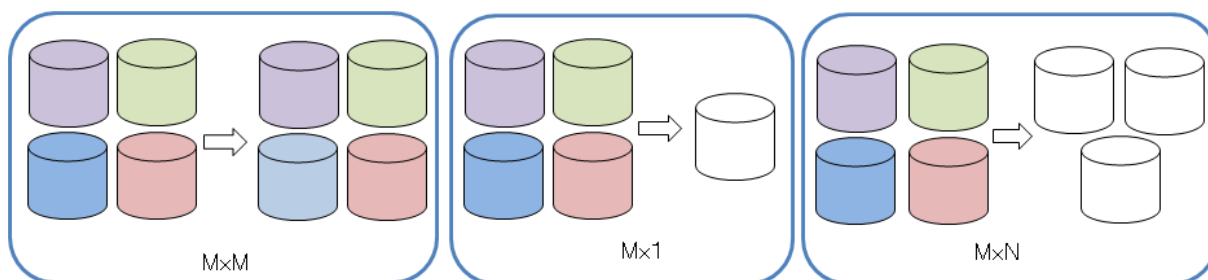


図 5.1 変換イメージ図

5.1.2 FCONV のインストール

FCONV は、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。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。CIO 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 6.2.2 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

5.1.4 出力形式

ファイル形式毎に可能な出力形式は以下になります。

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以下になります。

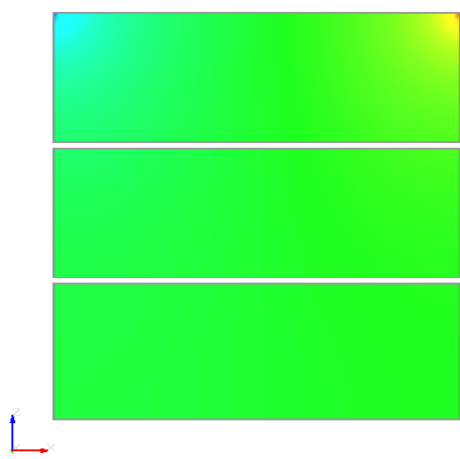


図 5.2 変換前

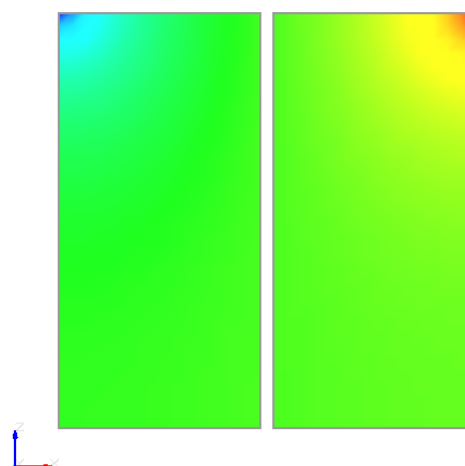


図 5.3 変換後

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	○		-

- : 未対応

○ : 対応

: デフォルト

× : FCONV では未対応

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
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)

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

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。

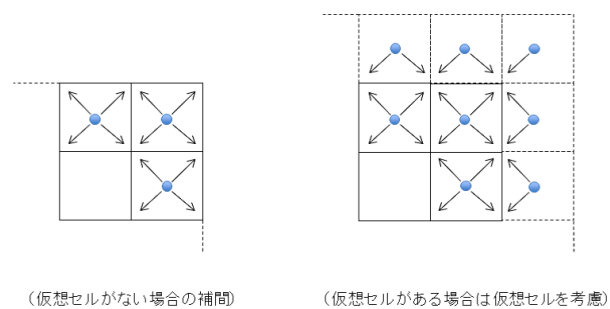


図 5.4 格子点への補間

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	-	-	-

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。

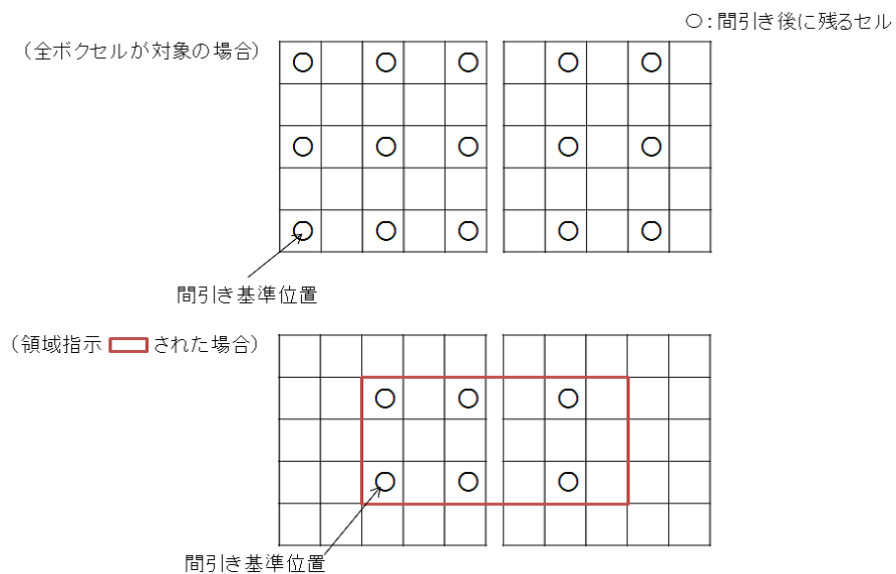


図 5.5 間引き数を 2 とした例

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

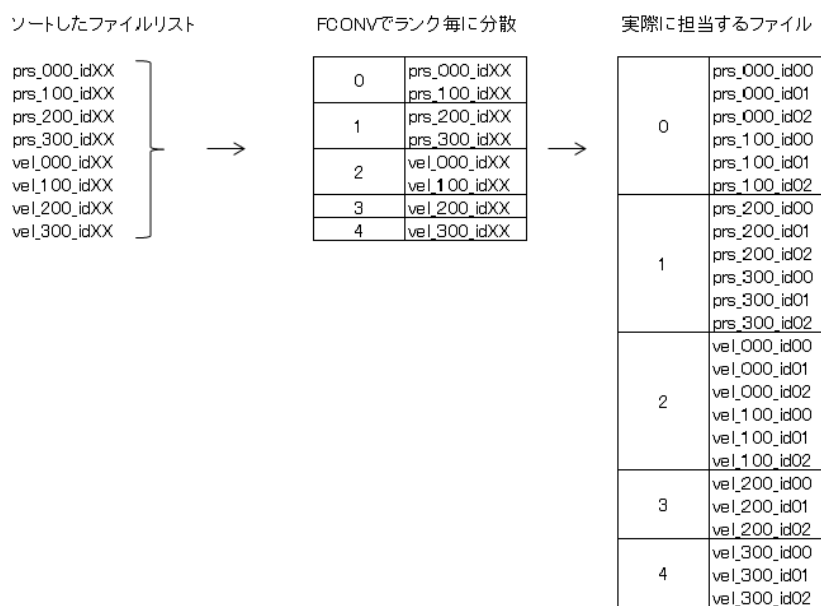


図 5.6 step 基準の例

(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

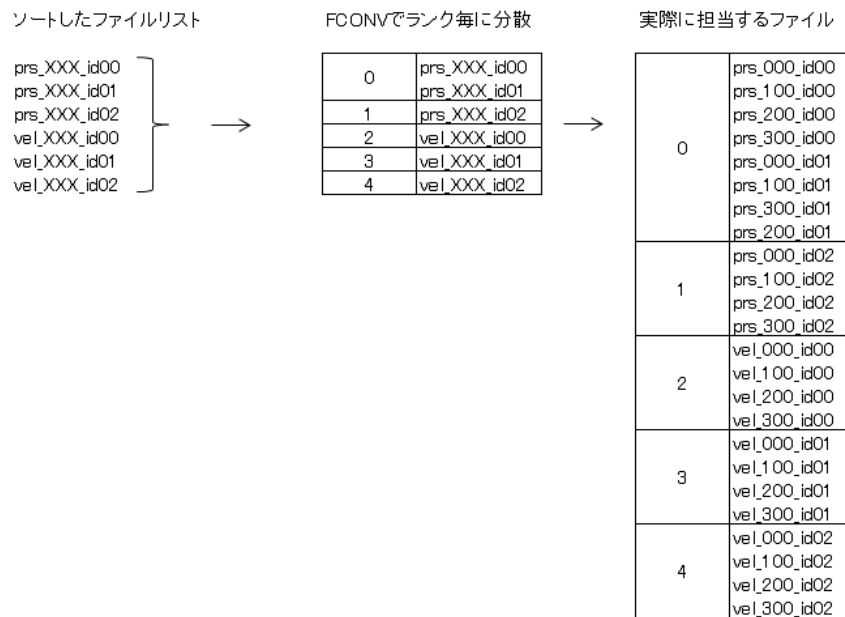


図 5.7 rank 基準の例

- (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

第 6 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 .

6.1 ファイル仕様

6.1.1 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            = "vel"          // ベースファイル名 ( 3)
    FileFormat        = "sph"          // ファイルタイプ, 拡張子 ( 3)
    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 ( 4)
    Endian            = "little"       // データのエンディアン ( 5)
    ArrayShape        = "nijk"        // 配列形状 ( 6)
    Component         = 3              // 成分数 (スカラーは不要)
    Variable[@]       {name = "u"}     // 成分名 (Component 個)
    Variable[@]       {name = "v"}     //
    Variable[@]       {name = "w"}     //
}
```

- (1) CIOlib では "Cartesian" のみ
- (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]
 逐次時 [Prefix]_[ステップ番号:10 桁].[ext]
- (4) Int8, UInt8, Int16, UInt16, Int32, UInt32, Int64, UInt64, Float32, Float64
- (5) little, big, 省略時:実行プラットフォームと同じ
- (6) 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{@} {
    Step          = 0            // ファイル出力ステップ数分
    Time          = 0.0          // 出力ステップ
    AverageTime    = 0.0          // 出力時刻
    AverageStep    = 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{@} {
    :
    :
  }
}

```

6.1.2 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{@} {
    :
    :
  }
}
```

6.1.3 フィールドデータファイルの仕様

以下に、CIO に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 6.1 を参照)

表 6.1 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 6.2 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 6.3 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 6.4 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 6.5 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 6.6 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): $IMAX \times JMAX \times KMAX \times 4$ (bytes)

倍精度の場合 (dtype = 2): $IMAX \times JMAX \times KMAX \times 8$ (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): $IMAX \times JMAX \times KMAX \times 4 \times 3$ (bytes)

倍精度の場合 (dtype = 2): $IMAX \times JMAX \times KMAX \times 8 \times 3$ (bytes)

BOV 形式

可視化ソフトウェア「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 6.7, 6.8 を参照)

表 6.7 (例 1) ijk 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
...	
$v(\text{imax}-1, \text{jmax}-1, \text{kmax}-1, 0)$	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
...	
$v(\text{imax}-1, \text{jmax}-1, \text{kmax}-1, n-1)$	格子点 (imax-1, jmax-1, kmax-1) の成分 n-1 のデータ値

表 6.8 (例 2) nij 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
...	
$v(n-1, \text{imax}-1, \text{jmax}-1, \text{kmax}-1)$	格子点 (imax-1, jmax-1, kmax-1) の成分 n-1 のデータ値

6.1.4 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 6.9 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。

6.1.5 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 = 1.0000000e+00
  }
  Pressure {
    Unit      = "NonDimensional"
    Reference = 0.0000000e+00
    Difference = 1.176300e+00
  }
  Velocity {
    Unit      = "NonDimensional"
    Reference = 1.0000000e+00
  }
}
TimeSlice {
  Slice[@] {
    Step = 0
    Time = 0.0000000e+00
    VectorMinMax {
      Min = 0.0000000e+00
      Max = 0.0000000e+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)
}
}
```

6.2 ファイル仕様 (ツール)

6.2.1 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 (CorpIndexStart,CorpIndeEnd) 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) FCONVInfor 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 (CorpIndexStart,CorpIndexEnd) を読み込みます。
- (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 ではないとき 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 $\text{int}(NV/ND)$ 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。

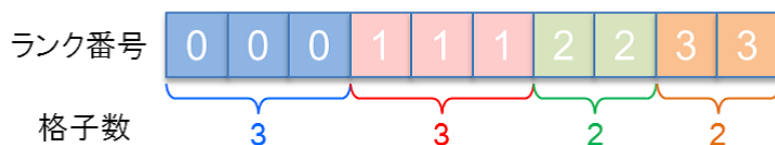


図 6.1 (例) 格子数 10, 領域分割 4

6.2.2 並列分割ファイルコンバータ用入力ファイルの仕様

以下に、並列分割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。

並列分割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 5.1.1 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”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”, ”ijnk”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, CIOlib 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. CorpIndexStart, CorpIndexEnd

- ・ 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
- ・ CorpIndexStart, CorpIndexEnd とともに指定されている場合はその領域
- ・ CorpIndexStart のみ指定されている場合は CorpIndexStart から最後まで領域
- ・ CorpIndexEnd のみ指定されている場合は先頭から CorpIndexEnd までの領域
- ・ 両方省略された場合は全部の領域

第 7 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。

7.1 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 14 2014/1/10

- 並列分散ファイルコンバートツール (fconv) の追加
- 上記ツール追加に伴う CIOlib の修正

Revision 13 2013/10/12

- cio.Interval_Mngr クラスを分離

Revision 12 2013/10/10

- DFI ファイルの単位系の Format 変更に伴う修正

Revision 11 2013/10/02

- modify for intel mpi

Revision 10 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

Revision 9 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

Revision 8 2013/07/20

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

Revision 7 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

Revision 6 2013/06/27

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

Revision 5 2013/06/26

- TextParser のアーカイブ名の修正に対応

Revision 4 2013/06/25

- dfi ファイルの相対パス処理の修正

- refinement, MxN の読み込み処理の再チェック

Revision 3 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

Revision 2 2013/06/08

- 体裁とパッケージングを変更

Revision 1 2013/06/06

- リリース

第 8 章

Appendix

8.1 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 8.1)

表 8.1 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
読み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み込み	ReadData	読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	GetDataTypeInfoString	文字列を取得
	GetDataTypeInfo	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	
読み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 DeltaT を無次元化	normalizeDeltaT	
CIO のバージョン No の取り出し	getVersionInfo	static メソッド

表目次

3.1	D_CIO.XXXX マクロ	14
3.2	E_CIO.ONOFF 列挙型	14
3.3	E_CIO.FORMAT 列挙型	15
3.4	E_CIO.DTYPE 列挙型	15
3.5	E_CIO.ARRAYSHAPE 列挙型	15
3.6	E_CIO.ENDIANTYPE 列挙型	16
3.7	E_CIO.READTYPE 列挙型	16
3.8	E_CIO.OUTPUT_TYPE 列挙型	16
3.9	E_CIO.OUTPUT_FNAME 列挙型	16
3.10	E_CIO.ERRORCODE 列挙型 その 1	17
3.11	E_CIO.ERRORCODE 列挙型 その 2	18
5.1	ファイルフォーマット毎の出力形式	53
5.2	ファイルフォーマット毎のデータ型	53
5.3	ファイルフォーマット毎の配列形状	54
5.4	ファイルフォーマット毎の定義点	54
5.5	ファイルフォーマット毎の出力ファイル	55
6.1	SPH ファイルレコード形式	62
6.2	データ属性レコード	62
6.3	ボクセルサイズレコード	63
6.4	原点座標レコード	63
6.5	ボクセルピッチレコード	64
6.6	時刻レコード	64
6.7	(例 1) ijkn 配列 $v(i,j,k,n)$ の記述例	66
6.8	(例 2) nijk 配列 $v(n,i,j,k)$ の記述例	66
6.9	サブドメイン情報ファイル仕様	67
8.1	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)	78

3.1	同一格子密度での 1 対 1 読み込み	19
3.2	同一格子密度での M 対 N 読み込み	19
3.3	リファインメントで 1 対 1 読み込み	20
3.4	リファインメントで M 対 N 読み込み	20
3.5	補間処理	31
3.6	1 対 1 の出力	36
4.1	ステージング	45
5.1	変換イメージ図	51
5.2	変換前	53
5.3	変換後	53
5.4	格子点への補間	54
5.5	間引き数を 2 とした例	55
5.6	step 基準の例	56
5.7	rank 基準の例	57
6.1	(例) 格子数 10 , 領域分割 4	71