User Guide of CIOlib

Cartesian Input / Output Library

Advanced Institute for Computational Science

RIKEN

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

June 2014

Contents

Chapter 1

Overview of CIOlib

This chapter provides an overview of CIOlib and this user guide.

1.1 CIOlib

The Cartesian Input/Output Library (CIOlib) is a C++ class library that manages file I/O for Cartesian grid data.

CIOlib offers the following functions:

- Management of grid and domain decomposition information using DFI metadata
- SPH and BOV (BVX) file format support
- MxN loading (for different numbers of parallel executions)
- Data loading from coarse to fine meshes
  (for cases in which the fraction of grids in each direction is 1/2 – or 1/8 for 3D)
- Staging (copying files to a directory by rank using an external program)
- External distributed parallel file conversion support
  (SPH,BOV → SPH,BOV,PLOT3D,AVS,VTK)

1.2 About This Document

1.2.1 Format

The following format represents a Shell command:

```
$ command (command parameter)
```

or

```
# command (command parameter)
```

A command that starts with "$" is to be executed by a user. A command that starts with "#" is to be executed by the administrator (root user in most cases).

1.2.2 Supported Environments

CIOlib supports the following environments:

- Linux/Intel Compiler
  - CentOS6.2 i386/x86_64
  - Intel C++/Fortran Compiler Version 12 (icpc/ifort)
- MacOS X Snow Leopard or later
  - MacOS X Snow Leopard
  - Intel C++/Fortran Compiler Version 11 or later (icpc/ifort)
- The K computer

Chapter 2

Building CIOlib

This chapter explains how to compile CIOlib.

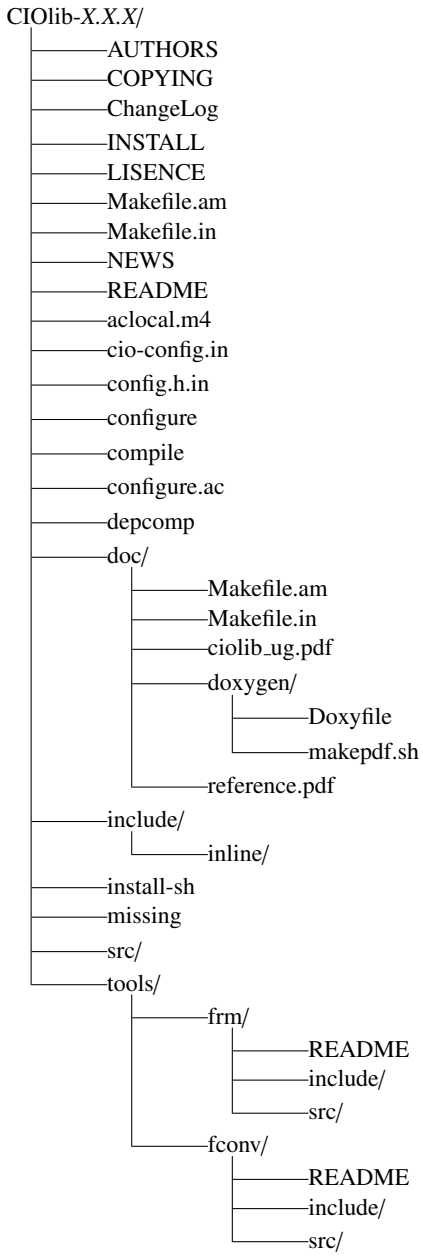
2.1 Building the CIOlib Package

2.1.1 Package Structure

The CIOlib package is stored in a file with the following name format:

`CIOlib-`*X.X.X*`.tar.gz` (where *X.X.X* is the version designation).

When the file is expanded, the following directories and files should appear:



- `doc`
  Contains all CIOlib documents, including this user guide.

- `include`
  Contains header files. The contents of this directory will be installed in `$prefix/include` after `make install` is invoked.

- `src`
  Contains source files. A library, `LibCIO.a`, will be created here and installed in `$prefix/lib` after `make install` is invoked.

- `tools`
  Contains the utility that allocates files to rank directories and the utility that executes distributed parallel file conversion.

2.1.2 Building the Package

Use a shell environment to build the package. The syntax for setting environmental variables will differ according to the shell you use. In the following example, a working directory is created and an unzipped package is built and installed in this working directory using bash:

1. Make a working directory (here named "work") into which the CIOlib package can be copied.

   ```
   $ mkdir work
   $ cp [package path] work
   ```

2. Change to the working directory and unzip the package.

   ```
   $ cd work
   $ tar zxvf CIOlib-X.X.X.tar.gz
   ```

3. Change to CIOlib-X.X.X directory generated by unzipping.

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

4. Execute the configure script, specifying an appropriate configuration option.

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

   When you execute the configure script with an appropriate option, a Makefile adapted to your environment will be created. For details on configuration options, see subsection 2.1.3.

5. Execute `make` command to build the library.

   ```
   $ make
   ```

   This will create a file named:

   ```
   src/libCIO.a
   ```

   If you want to re-build CIOlib, execute a `make clean` to delete the files created by the previous make command, then execute `make` again.

   ```
   $ make clean
   $ make
   ```

   If you want to rerun the configure script or recreate the makefile, execute `make distclean` to delete all information and begin again from the configure script.

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

6. Install
   To install libraries and header files into the directory specified by the `--prefix` option, use `make install`.

   ```
   $ make install
   ```

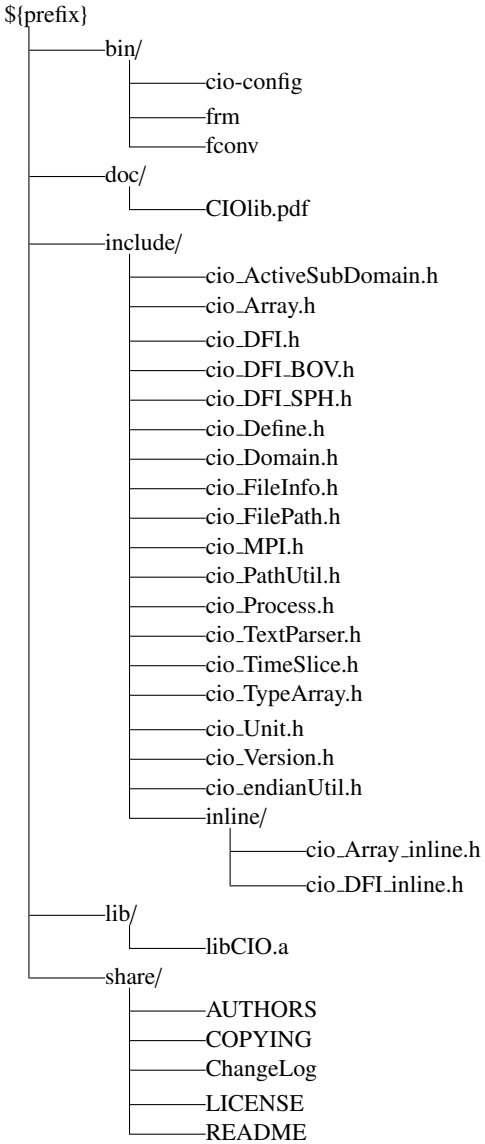
   If administrator rights are required to write under the installation directory, use the `sudo` command, or login as the administrator and execute `make install`.

   ```
   $ sudo make install
   ```

   or

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

   The location and files for installation are as follows:



7. Uninstall
   The uninstall command is different depending on your write permissions.

   ```
   $ make uninstall
   ```

   or

   ```
   $ sudo make uninstall
   ```

   or

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

2.1.3 Configure Script Options

- `--prefix=`*dir*
  Specifies where to install the package. When `--prefix=/usr/local/CIOlib` is specified, the libraries and header files will be installed under the following directories:

  ```
  Library: /usr/local/CIOlib/lib
  Header file: /usr/local/CIOlib/include
  ```

  If this option is not specified, /usr/local/CIOlib is used as the default value for the installation.

- Compiler and linker options
  Compilers, linkers and their options will be found semi-automatically. If you want to use nonstandard commands, options, libraries or header files, you must specify them with the following configure script options:

  `CXX`
  Command path of the C++ compiler.

  `CXXFLAGS`
  Compile option to pass to the C++ compiler.

  `LDFLAGS`
  Link option to pass to the linker. For example, if the library is in nonstandard location `<libdir>`, you can specify it with `-L<libdir>`.

  `LIBS`
  Link option to pass a library that you want to use to the linker. For example, If you want to use library `<library>`, you can specify it with `-l<library>`.

  `F90`
  Command path of the Fortran90 compiler.

  `F90FLAGS`
  Compile option to pass to the Fortran90 compiler.

- Library options
  Both the MPI library and the TextParser library are required when compiling and linking CIOlib. If you want to compile and link the distributed parallel file converter as well, specify the CPM library. Ensure that the installation paths of all necessary libraries are specified with the following configure options:

  `--with-ompi=`*dir*
  To use OpenMPI as the MPI library, specify its installation path using this option. If you are going to use the wrapper compiler attached to OpenMPI (mpicc, mpicxx, or mpif90), this option is unnecessary because the mpi

setting will be found automatically in the wrapper.

`--with-parser`=*dir*
Specify the installation path of the TextParser library.

`--with-cpm`=*dir*
Specify the installation path of the CPM library. CPMlib is required for the distributed parallel file converter. If this option is not specified, the distributed parallel file converter will not be compiled, linked, or installed. For the K computer, specify "`--with-MPI=no`, `--with-frm=yes`" at a login node to install the distributed parallel file converter.

`--host`=*hostname*
Specify the architecture for cross compilation.

`--with-MPI`=*(yes | no)*
Specify parallel calculation. The distributed parallel file converter will be installed for parallel computing.

`--with-frm`=*(no | yes)*
Specify the installation path of the FileRankMapper (frm) tool. Note that this option is invalid for cross compilation. In that case, you must manually compile the frm tool for a login node.

For more details on configure options, see the `./configure --help` command. Note, however, that options other than those described above are invalid for CIOlib.

2.1.4 Example of Configure Options

- Linux / MacOS X

```
CIOlib prefix: /opt/CIOlib
MPI library: OpenMPI, /usr/local/openmpi
TextParser library: /usr/local/textparser
CPM library: /usr/local/cpmlib
C++ compiler: icpc
F90 compiler: ifort
```

For the above environment, execute the following configure command:

```
$ ./configure --prefix=/opt/CIOlib \
              --with-ompi=/usr/local/openmpi \
              --with-parser=/usr/local/textparser \
              --with-cpm=/usr/local/cpmlib
              CXX=icpc \
              CXXFLAGS=-O3 \
              F90=ifort \
              F90FLAGS=-O3
```

- The K computer

```
CIOlib prefix: /home/userXXXX/CIOlib
TextParser library: /home/userXXXX/textparser
CPM library: /home/usreXXXX/cpmlib
C++ compiler: mpiFCCpx
F90 compiler: mpifrtpx
```

For the above environment, execute the following configure command:

```
$ ./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 Command

A shell script, triggered by the `$prefix/bin/cio-config` command, is generated when CIOlib is installed. This command lets you obtain compile options and link options for programs that reference CIOlib.

Execute `cio-config` command with the following options:

`--cxx`
Gets the C++ compiler that was used when CIOlib was built.

`--cflags`
Gets C++ compiler options.

`--libs`
Gets link options that are required for linking CIOlib.

Note that the options obtained using the `cio-config` command are the minimum options for CIOlib. Specify optimization options as necessary.

For more details on how to use the `cio-config` command in detail, see Section 2.2.

2.1.6 How to Create a Distribution Package

To create a distribution package, run the following command after executing the configure script:

```
$ ./make dist
```

After that, the environment will be compressed into a file of the form:

`CIOlib-X.X.X.tar.gz` (where *X.X.X* is the version indicator.)

2.1.7 How to Build CIOlib in an Environment Where a Staging Tool Is Used

When using a staging tool in a cross compilation environment such as the K computer, you need to build CIOlib with a front end native compiler.
If MPI library is not installed in the front end, be sure to specify the option "`--with-MPI=no`, `--with-frm=yes`" when running the CIOlib configure script.

・Example of running the configure script for the K computer front end:

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

In this case, you also need to build TextParser for linking by the front end native compiler.

2.2 How to Use CIOlib

2.2.1 C++

To build your own program referencing CIOlib, compile and link it as follows:
(Note that main.c is compiled using icpc in this example.)

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

Chapter 3

How to Use the CIOlib API

3.1 How to Use the C++ API in a User Program

3.1.1 Including cio_DFI.h

CIOlib defines its C++ API functional groups in the header file cio_DFI.h. Include this header file to use the API functions of CIOlib in your programs. This header file exposes the interfaces of the cio_DFI class, where all of the user-available API functions are gathered. The cio_DFI.h file is installed under the `${prefix}`/include directory, where `${prefix}` is whatever you specified for the configure script when invoking `make install`.

3.1.2 Macro, Enumerated Type, and Error Code

The macros, enumerated types, and error codes of CIOlib are defined in cio_Define.h.

- D_CIO_XXXX Macro

Table. 3.1 D_CIO_XXXX Macro

Name	Contents	Name	Contents	Name	Contents
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 Enumerated Type
  This toggles whether the system makes a directory and outputs the field data into that directory for each recorded period of time.
  It is defined as shown in Table 3.2.

Table. 3.2 E_CIO_ONOFF Enumerated Type

E_CIO_ONOFF Element	Value	Meaning
E_CIO_OFF	0	Switch OFF
E_CIO_ON	1	Switch ON

- E_CIO_FORMAT Enumerated Type
  This is a flag that specifies the file format of the field data.
  It is defined as shown in Table 3.3.

Table. 3.3 E_CIO_FORMAT Enumerated Type

E_CIO_FORMAT Element	Value	Meaning
E_CIO_UNKNOWN	-1	Undefined
E_CIO_SPH	0	SPH format
E_CIO_BOV	1	BOV format
E_CIO_AVS	2	AVS format
E_CIO_PLOT3D	3	PLOT3D format
E_CIO_VTK	4	VTK format

- E_CIO_DTYPE Enumerated Type
  This is a flag that specifies the data format of the field data.
  It is defined as shown in Table 3.4.

Table. 3.4 E_CIO_DTYPE Enumerated Type

E_CIO_DTYPE Element	Value	Meaning
E_CIO_DTYPE_UNKNOWN	0	Undefined
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 Enumerated Type
  This is a flag to specify the array shape of the field data.
  It is defined as shown in Table 3.5.

Table. 3.5 E_CIO_ARRAYSHAPE Enumerated Type

E_CIO_ARRAYSHAPE Element	Value	Meaning
E_CIO_ARRAYSHAPE_UNKNOWN	-1	Undefined
E_CIO_IJKN	0	(i,j,k,n)
E_CIO_NIJK	1	(n,i,j,k)

- E_CIO_ENDIANTYPE Enumerated Type
  This is a flag to specify the endian type of the field data.
  It is defined as shown in Table 3.6.

Table. 3.6 E_CIO_ENDIANTYPE Enumerated Type

E_CIO_ENDIANTYPE Element	Value	Meaning
E_CIO_ENDIANTYPE_UNKNOWN	-1	Undefined
E_CIO_LITTELE	0	Little endian type
E_CIO_BIG	1	Big endian type

- E_CIO_READTYPE Enumerated Type
  This is a flag to specify how the field data is read.
  It is defined as shown in Table 3.7.

Table. 3.7 E_CIO_READTYPE Enumerated Type

E_CIO_READTYPE Element	Value	Meaning
E_CIO_SAMEDIV_SAMERES	1	Same division, same density
E_CIO_SAMEDIV_REFINEMENT	2	Same division, refinement
E_CIO_DIFFDIV_SAMERES	3	MxN, same density
E_CIO_DIFFDIV_REFINEMENT	4	MxN, refinement
E_CIO_READTYPE_UNKNOWN	5	Error

- E_CIO_OUTPUT_TYPE Enumerated Type
  This is a flag to specify the output format of the field data.
  It is defined as shown in Table 3.8.

Table. 3.8 E_CIO_OUTPUT_TYPE Enumerated Type

E_CIO_OUTPUT_TYPE Element	Value	Meaning
E_CIO_OUTPUT_TYPE_DEFAULT	-1	Default (binary)
E_CIO_OUTPUT_TYPE_ASCII	0	ascii format
E_CIO_OUTPUT_TYPE_BINARY	1	binary format
E_CIO_OUTPUT_TYPE_FBINARY	2	Fortran Binary format

- E_CIO_OUTPUT_FNAME Enumerated Type
  This is a flag to specify the order for naming the output file of the field data.
  It is defined as shown in Table 3.9.

Table. 3.9 E_CIO_OUTPUT_FNAME Enumerated Type

E_CIO_OUTPUT_FNAME Element	Value	Meaning
E_CIO_OUTPUT_FNAME_DEFAULT	-1	Default (step_rank)
E_CIO_OUTPUT_FNMAE_STEP_RANK	0	step_rank
E_CIO_OUTPUT_FNMAE_RANK_STEP	1	rank_step

- E_CIO_ERRORCODE Enumerated Type
  All error codes for API functions are provided in this enumerated type.
  It is defined as shown in Tables 3.10 and 3.11.

Table. 3.10 E_CIO_ERRORCODE Enumerated Type 1

E_CIO_ERRORCODE Element	Value	Meaning
E_CIO_SUCCESS	0	Success
E_CIO_ERROR	-1	Error
E_CIO_ERROR_READ_DFI_GLOBALORIGIN	1000	DFI GlobalOrigin read error
E_CIO_ERROR_READ_DFI_GLOBALREGION	1001	DFI GlobalRegion read error
E_CIO_ERROR_READ_DFI_GLOBALVOXEL	1002	DFI GlobalVoxel read error
E_CIO_ERROR_READ_DFI_GLOBALDIVISION	1003	DFI GlobalDivison read error
E_CIO_ERROR_READ_DFI_DIRECTORYPATH	1004	DFI DirectoryPath read error
E_CIO_ERROR_READ_DFI_TIMESLICEDIRECTORY	1005	DFI TimeSliceDirectoryPath read error
E_CIO_ERROR_READ_DFI_PREFIX	1006	DFI Prefix read error
E_CIO_ERROR_READ_DFI_FILEFORMAT	1007	DFI FileFormat read error
E_CIO_ERROR_READ_DFI_GUIDECELL	1008	DFI GuideCell read error
E_CIO_ERROR_READ_DFI_DATATYPE	1009	DFI DataType read error
E_CIO_ERROR_READ_DFI_ENDIAN	1010	DFI Endian read error
E_CIO_ERROR_READ_DFI_ARRAYSHAPE	1011	DFI ArrayShape read error
E_CIO_ERROR_READ_DFI_COMPONENT	1012	DFI Component read error
E_CIO_ERROR_READ_DFI_FILEPATH_PROCESS	1013	DFI FilePath/Process read error
E_CIO_ERROR_READ_DFI_NO_RANK	1014	DFI Rank no Element
E_CIO_ERROR_READ_DFI_ID	1015	DFI ID read error
E_CIO_ERROR_READ_DFI_HOSTNAME	1016	DFI HoatName read error
E_CIO_ERROR_READ_DFI_VOXELSIZE	1017	DFI VoxelSize read error
E_CIO_ERROR_READ_DFI_HEADINDEX	1018	DFI HeadIndex read error
E_CIO_ERROR_READ_DFI_TAILINDEX	1019	DFI TailIndex read error
E_CIO_ERROR_READ_DFI_NO_SLICE	1020	DFI TimeSlice no Element
E_CIO_ERROR_READ_DFI_STEP	1021	DFI Step read error
E_CIO_ERROR_READ_DFI_TIME	1022	DFI Time read error
E_CIO_ERROR_READ_DFI_NO_MINMAX	1023	DFI MinMax no Element
E_CIO_ERROR_READ_DFI_MIN	1024	DFI Min read error
E_CIO_ERROR_READ_DFI_MAX	1025	DFI Max read error
E_CIO_ERROR_READ_DFI_DFITYPE	1026	DFI DFIType read error
E_CIO_ERROR_READ_DFI_FIELDFILENAMEFORMAT	1027	DFI FieldFilenameFormat read error
E_CIO_ERROR_READ_INDEXFILE_OPENERROR	1050	Index file open error
E_CIO_ERROR_TEXTPARSER	1051	TextParser error
E_CIO_ERROR_READ_FILEINFO	1052	FileInfo read error
E_CIO_ERROR_READ_FILEPATH	1053	FilePath read error
E_CIO_ERROR_READ_UNIT	1054	UNIT read error
E_CIO_ERROR_READ_TIMESLICE	1055	TimeSlice read error
E_CIO_ERROR_READ_PROCFILE_OPENERROR	1056	Proc file open error
E_CIO_ERROR_READ_DOMAIN	1057	Domain read error
E_CIO_ERROR_READ_MPI	1058	MPI read error
E_CIO_ERROR_READ_PROCESS	1059	Process read error
E_CIO_ERROR_READ_FIELDDATA_FILE	1900	Field data file read error
E_CIO_ERROR_READ_SPH_FILE	2000	SPH file read error
E_CIO_ERROR_READ_SPH_REC1	2001	SPH file record1 read error
E_CIO_ERROR_READ_SPH_REC2	2002	SPH file record2 read error

Table. 3.11 E_CIO_ERRORCODE Enumerated Type 2

cpm_ErrorCode Element	Value	Meaning
E_CIO_ERROR_READ_SPH_REC3	2003	SPH file record 3 read error
E_CIO_ERROR_READ_SPH_REC4	2004	SPH file record 4 read error
E_CIO_ERROR_READ_SPH_REC5	2005	SPH file record 5 read error
E_CIO_ERROR_READ_SPH_REC6	2006	SPH file record 6 read error
E_CIO_ERROR_READ_SPH_REC7	2007	SPH file record 7 read error
E_CIO_ERROR_UNMATCH_VOXELSIZE	2050	Voxel size of SPH and DFI are different
E_CIO_ERROR_NOMATCH_ENDIAN	2051	Output format is different (Endian format is not either of Big or Little)
E_CIO_ERROR_READ_BOV_FILE	2100	BOV file read error
E_CIO_ERROR_READ_FIELD_HEADER_RECORD	2102	Failed to read header record of field data
E_CIO_ERROR_READ_FIELD_DATA_RECORD	2103	Failed to read data record of field data
E_CIO_ERROR_READ_FIELD_AVERAGED_RECORD	2104	Failed to read Averaged record of field data
E_CIO_ERROR_MISMATCH_NP_SUBDOMAIN	3003	Different parallel number and subdomain number
E_CIO_ERROR_INVALID_DIVNUM	3011	Invalid number of subdomains in domain decomposition
E_CIO_ERROR_OPEN_SBDM	3012	Failed to open ActiveSubdomain file
E_CIO_ERROR_READ_SBDM_HEADER	3013	Failed to read the header of ActiveSubdomain file
E_CIO_ERROR_READ_SBDM_FORMAT	3014	ActiveSubdomain file format error
E_CIO_ERROR_READ_SBDM_DIV	3015	Failed to read the number of subdomains in Active Subdomain file
E_CIO_ERROR_READ_SBDM_CONTENTS	3016	Failed to read Contents of ActiveSubdomain file
E_CIO_ERROR_SBDM_NUMDOMAIN_ZERO	3017	Active domain number of ActiveSubdomain file is 0
E_CIO_ERROR_MAKEDIRECTORY	3100	Failed to make Directory
E_CIO_ERROR_OPEN_FIELDDATA	3101	Failed to open field data
E_CIO_ERROR_WRITE_FIELD_HEADER_RECORD	3102	Failed to output the header record of field data
E_CIO_ERROR_WRITE_FIELD_DATA_RECORD	3103	Failed to output data record of field data
E_CIO_ERROR_WRITE_FIELD_AVERAGED_RECORD	3104	Failed to output Average record of field data
E_CIO_ERROR_WRITE_SPH_REC1	3201	SPH file record 1 output error
E_CIO_ERROR_WRITE_SPH_REC2	3202	SPH file record 2 output error
E_CIO_ERROR_WRITE_SPH_REC3	3203	SPH file record 3 output error
E_CIO_ERROR_WRITE_SPH_REC4	3204	SPH file record 4 output error
E_CIO_ERROR_WRITE_SPH_REC5	3205	SPH file record 5 output error
E_CIO_ERROR_WRITE_SPH_REC6	3206	SPH file record 6 output error
E_CIO_ERROR_WRITE_SPH_REC7	3207	SPH file record 7 output error
E_CIO_ERROR_WRITE_PROCFILENAME_EMPTY	3500	proc dfi name is Undefined
E_CIO_ERROR_WRITE_PROCFILE_OPENERROR	3501	Failed to open proc dfi file
E_CIO_ERROR_WRITE_DOMAIN	3502	Failed to output Domain
E_CIO_ERROR_WRITE_MPI	3503	Failed to output MPI
E_CIO_ERROR_WRITE_PROCESS	3504	Failed to output Process
E_CIO_ERROR_WRITE_RANKID	3505	Other than output rank
E_CIO_ERROR_WRITE_INDEXFILENAME_EMPTY	3510	index dfi file name undefined
E_CIO_ERROR_WRITE_PREFIX_EMPTY	3511	Prefix undefined
E_CIO_ERROR_WRITE_INDEXFILE_OPENERROR	3512	Failed to open proc.dfi file
E_CIO_ERROR_WRITE_FILEINFO	3513	FileInfo output failed
E_CIO_ERROR_WRITE_UNIT	3514	Unit output failed
E_CIO_ERROR_WRITE_TIMESLICE	3515	TimeSlice output failed
E_CIO_ERROR_WRITE_FILEPATH	3516	FilePath output failed
E_CIO_WARN_GETUNIT	4000	No unit of Unit

3.2 Input Function

3.2.1 Overview

CIOlib supports four types of field data file reading functions: 1 to 1 data reading, MxN data reading, and two types of refinement data reading that map the computation result of a coarse mesh onto a fine mesh in the ratio of 1:2. CIOlib identifies the type of the field data file automatically.

- 1 to 1 reading between grids of the same density
  When the number of grids for the entire domain is the same and the positioning of the domain decomposition is the same, each process reads a piece of corresponding field data.

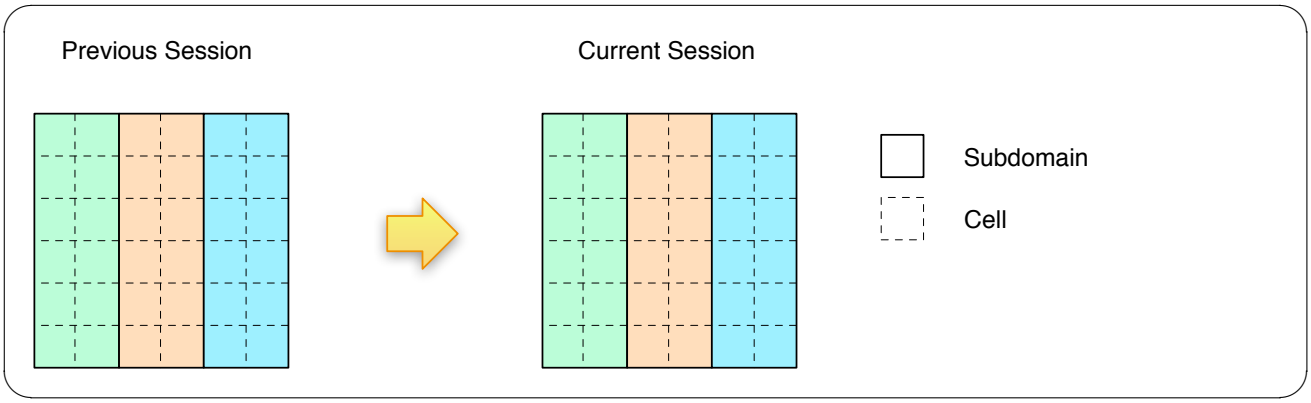


Fig. 3.1 1 to 1 reading between grids of the same density

- MxN reading between grids of the same density
  When the number of grids for the entire domain is the same and the number or positioning of the domain decomposition is different, one process reads either one or multiple piece(s) of corresponding field data.

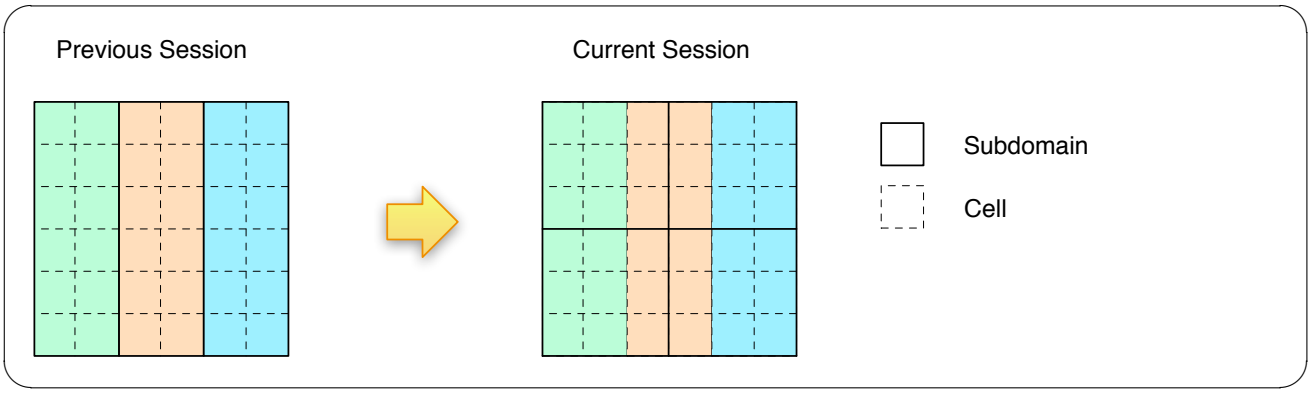


Fig. 3.2 MxN reading between grids of the same density

- 1 to 1 reading using refinement data
  When a grid is refined by a ratio of 1:2 and the field data is in one-to-one correspondence, each process reads a piece of corresponding field data and interpolates it*.

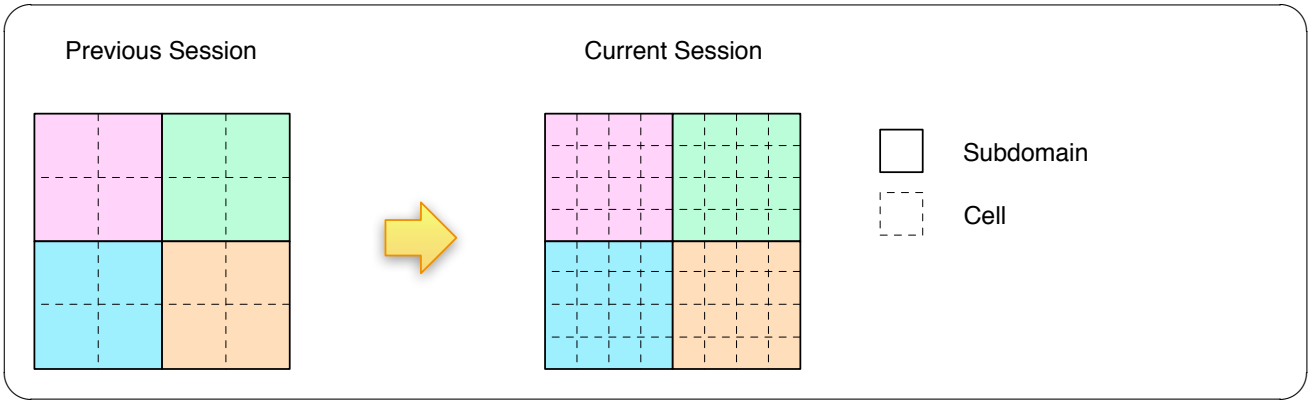


Fig. 3.3 1 to 1 reading with refinement data

- MxN reading using refinement data
  When a grid is refined by a ratio of 1:2 and the number of subdomains in the domain decomposition is different (in other words, field data is not in a one-to-one correspondence), one process reads one or multiple piece(s) of corresponding field data and interpolates it/them**.

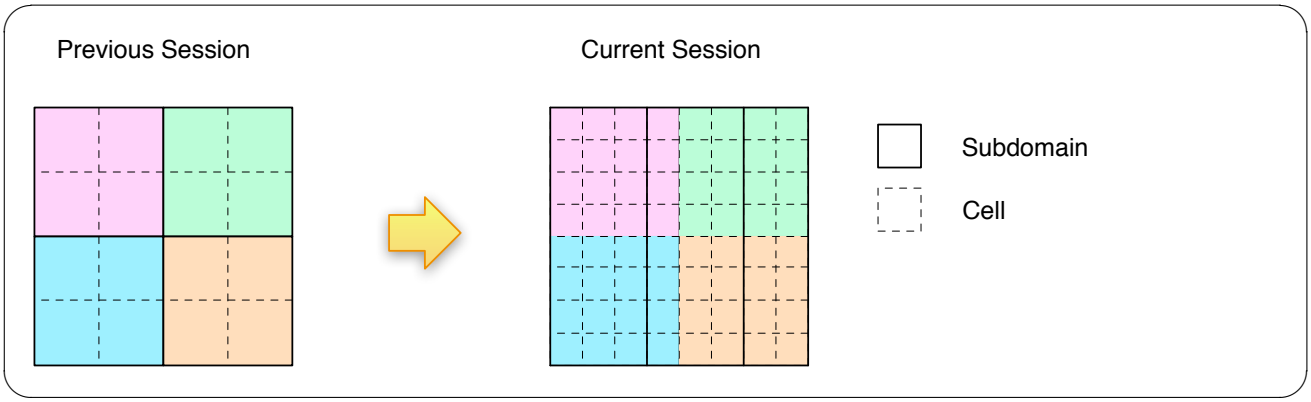


Fig. 3.4 MxN reading with refinement data

* For details on the interpolation of refinement data, see subsection 3.2.6
** Only real numbers (single-precision / double-precision) are available for the process of reading refinement data and interpolation.

3.2.2 Input Processing

CIOlib takes input of the field data and the DFI data according to the following steps:

1. Get a pointer to an instance of the reading process (see subsection 3.2.2)
2. Get information from the DFI file (see subsection 3.2.3)
3. Read the field data (see subsection 3.2.5)

Note that you should delete the instance pointer to the reading process when it is no longer needed.

Any number of cio_DFI class instances can be created for each kind of DFI file. The method for getting the pointer to the instance is defined inside cio_DFI.h, as follows:

Creating an instance of the reading process and getting the pointer to the instance

```
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);
```

This gets the pointer to cio_DFI class's instance. The method arguments are as follows:

comm	[*input*]	MPI communicator
dfifile	[*input*]	index.dfi file name
G_Voxel	[*input*]	voxel size of the entire computational domain for X, Y, and Z directions (an array of 3 words)
G_Div	[*input*]	the number of the subdomains in the domain decomposition for X, Y, and Z directions (an array of 3 words)
ret	[*output*]	error code（see Tables 3.10 and 3.11.）
Return Value		pointer to the cio_DFI class's instance

Note that you should delete the instance pointer when it is no longer needed.

```
//DFI instance
cioDFI *DFI_IN_PRS = cio_DFI::ReadInit(,,,);
             :
           (Processing)
             :
//Delete the DFI instance when it is no longer needed
delete DFI_IN_PRS;
```

3.2.3 Getting DFI Information

Use methods in CIOlib to get the imported DFI information. These methods are defined as follows:

1. The methods for getting the array shape for the field data are defined for String and Enumerated type:

Getting the array shape of the field data (String)

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

Return Value the array shape of the field data as a String (see Table 3.1.)

Getting the array shape of the filed data (Enumerated type)

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

Return Value the array shape of the field data as an Enumerated type (see Table 3.5.)

2. The methods for getting the data type of the field data are defined for String and Enumerated type:

Getting the data type of the field data (String)

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

Return Value the data type of the filed data as a String (see Table 3.1.)

Getting the data type of the field data (Enumerated type)

```
CIO::E_CIO_DTYPE
cio_DFI::GetDataType();
```

Return Value the data type of the field data as an Enumerated type (see Table 3.4.)

3. Get the number of the components of the field data:

Getting the number of the components of the filed data

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

Return Value the number of the components of the field data

4. Convert the data type (from String to Enumerated type):

Converting the data type (from String to Enumerated type)

```
static CIO::E_CIO_DTYPE
cio_DFI::ConvDatatypeS2E(const std::string datatype);
```

`datatype` [*input*] data type, as retrieved from the DFI file. See Table 3.4.
Return Value Enumerated Type (see Table 3.4.)

5. Convert the data type (from Enumerated type to String):

Converting the data type (from Enumerated type to String)

```
static std::string
cio_DFI::ConvDatatypeE2S(const CIO::E_CIO_DTYPE Dtype);
```

Dtype	[*input*]	data type, as retrieved from the DFI file	See Table 3.4.
Return Value	String data (see Table 3.1.)		

6. Get GlobalVoxel, the number of voxels in the entire computational domain, for the DFI domain:

Getting GlovalVoxel of DFI Domain

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

Return Value A pointer to GlobalVoxel

Make sure to delete this pointer when it is no longer needed.

For specification of the DFI file's Domain, see subsection 6.1.2, "Specification for the Process Information File (proc.dfi)."

7. Get GlobalDivision, the number of subdomains in the computational domain, for the DFI domain:

Getting GlobalDivision of DFI Domain

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

Return Value A pointer to GlobalDivision

Make sure to delete this pointer when it is no longer needed.

For specification of the DFI file's Domain, see subsection 6.1.2, "Specification for the Process Information File (proc.dfi)."

8. Get the component name of the DFI FileInfo:

Getting the component name of DFI FileInfo

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

pcomp	[*input*]	component position	0:u 1:v 2:w
Return Value	component name		

For specification of the DFI file's FileInfo, see subsection 6.1.1, "Specification for the Index File (index.dfi)."

9. Get the minmax composite value for the DFI TimeSlice:

Getting the minmax composite value of DFI TimeSlice

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

step	[*input*]	target step number
vec_min	[*output*]	min composite Value
vec_max	[*output*]	max composite Value
Return Value	error code	(see Tables 3.10 and 3.11)

For specification of DFI file's TimeSlice, see subsection 6.1.1, "Specification for the Index File (index.dfi)."

10. Get the minmax value of the DFI TimeSlice:

Getting the minmax value of DFI TimeSlice

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

step	[*input*]	target step number
compNo	[*input*]	target component number (0 – n)
min_value	[*output*]	min
max_value	[*output*]	max
Return Value	error code	(see Tables 3.10 and 3.11)

For specification of the DFI file's TimeSlice, see subsection 6.1.1, "Specification for the Index File (index.dfi)."

11. Gets the unit from the DFI UnitList:

Getting the unit from UnitList

```
CIO::E_CIO_ERRORCODE
cio_DFI::GetUnitElem(const std::string Name,
                     cio_UnitElem &unit);
```

Name	[*input*]	unit to get
unit	[*output*]	got unit
Return Value	error code	(see Tables 3.10 and 3.11)

For specification of the unit of DFI file's UnitList, see subsection 6.1.1, "Specification for the Index File (index.dfi)."

12. Gets every value of the DFI UnitList

Getting every value of UnitList

```
CIO::E_CIO_ERRORCODE
cio_DFI::GetUnit(const std::string Name,
                 std::string &unit,
                 double &ref,
                 double &diff,
                 bool &bSetDiff);
```

Name	[*input*]	unit to get
unit	[*output*]	got unit strings
ret	[*output*]	got referenceValue
diff	[*output*]	got differenceValue
bSetDiff	[*output*]	got difference flag
Return Value	error code (see Tables 3.10 and 3.11)	

For specification of the unit of DFI file's UnitList, see subsection 6.1.1, "Specification for the Index File (index.dfi)."

3.2.4 Getting the DFI Class Pointer

Use methods in CIOlib to get pointers to each class where the input DFI information is set.

1. Get the cio_FileInfo class pointer:

Get the cio_FileInfo class pointer:

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

Return Value pointer to the class where FileInfo information is set.

2. Get the cio_FilePath class pointer:

Getting the cio_FilePath class pointer

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

Return Value pointer to the class where FilePath information is set.

3. Get the cio_Unit class pointer:

Getting the cio_Unit class pointer

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

Return Value pointer to the class where Unit information is set.

4. Get the cio_Domain class pointer:

Getting the cio_Domain class pointer

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

Return Value pointer to the class where Domain information is set.

5. Get the cio_MPI class pointer:

Getting the cio_MPI class pointer

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

Return Value pointer to the class where MPI information is set.

6. Get the cio_TimeSlice class pointer:

Getting the cio_TimeSlice class pointer

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

Return Value pointer to the class where TimeSlice information is set.

7. Get the cio_Process class pointer:

Getting the cio_Process class pointer

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

Return Value pointer to the class where Process information is set.

3.2.5 Reading the Field Data File

The field data file formats supported in CIOlib are SPH and BOV. (For details, see subsection 6.1.3, "Specification for the Field Data File.")

Two methods of reading field data file are defined in cio_DFI.h: a method that returns a pointer to the input data and a method that reads the data into the array pointer specified by the user.

Reading field data file

```
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);
```

Reads the field data file

ret	[*output*]	error code (see Tables 3.10 and 3.11.)
step	[*input*]	step number of the field data file to read
gc	[*input*]	the number of virtual cells in the computational domain
Gvoxel	[*input*]	voxel size of the entire computational domain of X, Y, and Z directions (an array of 3 words)
Gdivision	[*input*]	the number of subdomains of X, Y, and Z directions (an array of 3 words)
head	[*input*]	start position of the computational domain of X, Y, and Z directions (an array of 3 words)
tail	[*input*]	end position of the computational domain of X, Y, and Z directions (an array of 3 words)
time	[*output*]	time to read
mode	[*input*]	average time, averaged step read flag (false: read, true: not read)
step_avr	[*output*]	averaged step to read
time_avr	[*output*]	average time to read
Return Value	pointer of the input field data	

Be sure to delete the field data pointer when it is no longer used.

```
// Reading field data file
float* data = (float *)dfi->Read(Parameter);
      :
    (Processing)
      :
// Delete the pointer
delete [] data;
```

Reading field data file

```
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);
```

Reads the field data file.

`val`	[*output*]	pointer of array to read
`step`	[*input*]	step number of the field data file to read
`gc`	[*input*]	the number of virtual cells in the computational domain
`Gvoxel`	[*input*]	voxel size of the entire computational domain of X, Y, and Z directions (an array of 3 words)
`Gdivision`	[*input*]	the number of subdomains of X, Y, and Z directions (an array of 3 words)
`head`	[*input*]	start position of the computational domain of X, Y, and Z directions (an array of 3 words)
`tail`	[*input*]	end position of the computational domain of X, Y, and Z directions (an array of 3 words)
`time`	[*output*]	time to read
`mode`	[*input*]	average time, averaged step read flag (false: read, true: not read)
`step_avr`	[*output*]	averaged step to read
`time_avr`	[*output*]	average time to read
Return Value	error code (see Tables 3.10 and 3.11.)	

3.2.6 Interpolation Method for Refinement Data

While reading refinement data, CIOlib interpolates the data using the following Fortran subroutines (cio_interp.f90):

1. IJKN array

cio_interp_ijkn_r4: IJKN array, single-precision real number version

```
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_r3
```

cio_interp_ijkn_r8: IJKN array, double-precision real number version

```
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
```

The interpolation processes for these subroutines are written in an external include file. If you want to change the interpolation algorithm, modify this file.

2. NIJK array

cio_interp_nijk_r4: NIJK array, single-precision real number version

```
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 array, double-precision real number version

```
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
```

The interpolation processes for these subroutines are written in an external include file. If you want to change the interpolation algorithm, modify this file.

3. About the do loop index in the include file
   - The array size for dst, the interpolation destination, is just twice the size of the array size for src, which is the source of interpolation, including virtual cells.
   - The range of the do loop index is determined by the index of src.
   - i,j,k indicates the index of src and ii.jj.kk indicates the index of dst.

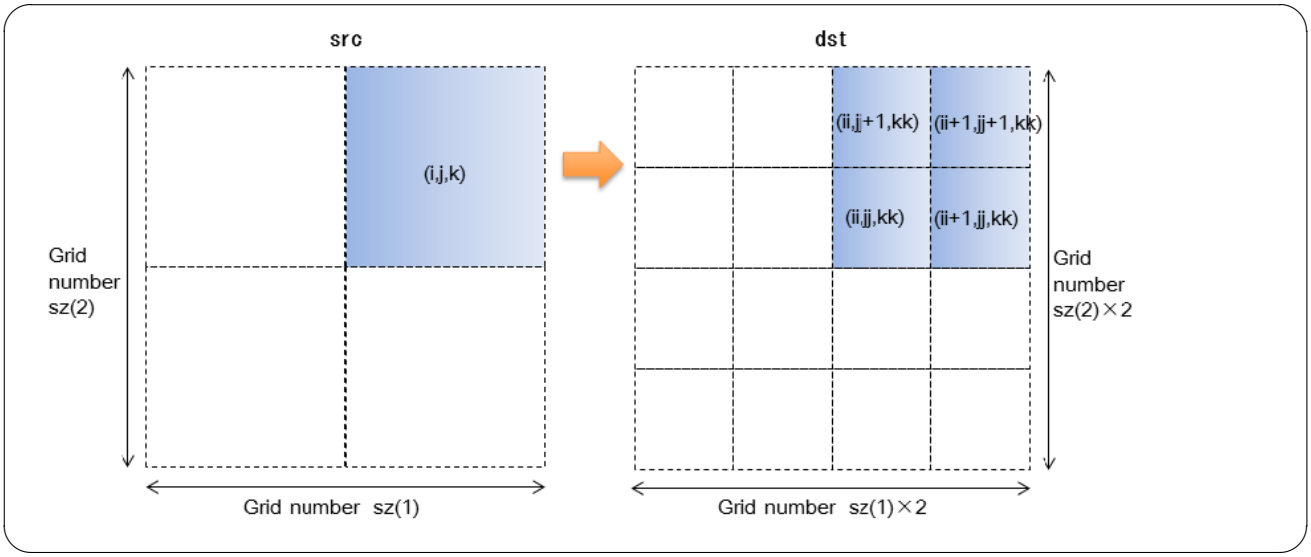


Fig. 3.5 Interpolation process

cio_interp_ijkn.h: IJKN array, interpolation process

```
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 array, interpolation process

```
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
```

Definitions for the above listings are as follows:

src	:	array of the input coarse data
szS	:	array contains the real voxel size of the src
gcS	:	the number of virtual cells in the src
dst	:	array of fine data interpolated from coarse data

3.2.7 Sample Code for the Input Processing

1. Read the field data into the array pointer passed as a parameter:

```
include "cio_DFI.h"
int main( int argc, char **argv )
{
  //CIO's error code
  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;
  }

  //Sets the dfi file name passed as a parameter
  if( argc != 2 ) {
    // Error: if DFI file name is not passed as a parameter
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CIO::E_CIO_ERROR;
  }
  std::string dfi_fname = argv[1];

  //Defines computational domain
  int GVoxel[3] = {64, 64, 64}; ///<Voxel size of the entire computational domain
  int GDiv[3]   = {1, 1, 1};    ///<the number of subdomains
                           /// (i.e. the number of parallel executions)
  int head[3]   = {1, 1, 1};    ///<Starting point for the computational domain
  int tail[3]   = {64, 64, 64}; ///<End point for the computational domain
  int gsize     = 2;            ///<the number of virtual cells in the computational domain
  //Size of the array to read
  size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

  //Gets the pointer to the instance for reading
  cio_DFI* DFI_IN = cio_DFI::ReadInit(MPI_COMM_WORLD, ///<MPI communicator
                                      dfi_fname,      ///<dfi file name
                                      GVoxel,         ///<voxel size of
                                             ///  the entire computational domain
                                      GDiv,           ///<the number of subdomains
                                      ret);           ///< error code
  // Error processing
  if( ret != CIO::E_CIO_SUCCESS || DFI_IN == NULL ) {
    // Error: if the instance fails
    std::cerr << "Error Readinit." << std::endl;
    return ret;
  }

  //Checks the type of the field data to read
  if( DFI_IN->GetDataType() != CIO::E_CIO_FLOAT32 ) {
    // If data type is different
    std::cerr << "Error Datatype unmatch." << std::endl;
    return CIO::E_CIO_ERROR;
  }

  //Gets the number of components of the field data to read
  int ncomp=DFI_IN->GetNumComponent();

  //Gets the unit
  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);
    }
  }

  //Allocates the array to read
  float *d_v = new float[size*ncomp];
  //Clears the array to read (sets to zero)
  memset(d_v, 0, sizeof(float)*size*ncomp);
  //Sets the step number for reading the field data
  unsigned step = 10;

  float r_time;       ///<Time to read from dfi
  unsigned i_dummy;   ///<Average step
  float f_dummy;      ///<Average time

  //Reads the field data
  ret =  DFI_IN->ReadData(d_v,    ///<Pointer of the array to read
                          step,   ///<Step number of the field data to read
                          gsize,  ///<the number of virtual cells in the computational domain
                          GVoxel, ///<Voxel size of the entire computational domain
                          GDiv,   ///<the number of subdomains
                          head,   ///<Starting point of the computational domain
                          tail,   ///<End point of the computational domain
                          r_time, ///<Read time from dfi
                          true,   ///<Not read average
                          i_dummy,
                          f_dummy );
  // Error Processing
  if( ret != CIO::E_CIO_SUCCESS ) {
    //If reading the field data fails
    std::cerr << "Error ReadData." << std::endl;
    delete [] d_v;
    delete DFI_IN;
    return ret;
  }

  //Normal end processing
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<Deletes the array pointer
  delete DFI_IN;  ///<Deletes the pointer to the instance for reading
  return CIO::E_CIO_SUCCESS;
}
```

2. Return the array pointer to the field data after reading:

```
#include "cio_DFI.h"
int main( int argc, char **argv )
{
  //CIO's error code
  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;
```

```
}

//Sets the dfi file name passed as a parameter
if( argc != 2 ) {
  // Error: if DFI file name is not passed as a parameter
  std::cerr << "Error undefined DFI file name." << std::endl;
  return CIO::E_CIO_ERROR;
}
std::string dfi_fname = argv[1];

//Defines computational domain
int GVoxel[3] = {64, 64, 64}; ///<Voxel size of the entire computational domain
int GDiv[3]   = {1, 1, 1};    ///<the number of subdomains
                               ///(i.e. the number of parallel executions)
int head[3]   = {1, 1, 1};    ///<Starting point of the computational domain
int tail[3]   = {64, 64, 64}; ///<End point of the computational domain
int gsize     = 2;            ///<the number of virtual cells in the computational domain

//Gets the pointer to the instance for reading
cio_DFI* DFI_IN = cio_DFI::ReadInit(MPI_COMM_WORLD, ///<MPI communicator
                                    dfi_fname,      ///<dfi file name
                                    GVoxel,         ///<Voxel size of
                                                    /// the entire computational domain
                                    GDiv,           ///<the number of subdomains
                                    ret);           ///< error code
// Error processing
if( ret != CIO::E_CIO_SUCCESS || DFI_IN == NULL ) {
  // Error: if the instance fails
  std::cerr << "Error Readinit." << std::endl;
  return ret;
}
//Checks the type of the field data to read
if( DFI_IN->GetDataType() != CIO::E_CIO_FLOAT32 ) {
  //If data type is different
  std::cerr << "Error Datatype unmatch." << std::endl;
  return CIO::E_CIO_ERROR;
}
//Gets the unit
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; ///<Step number for reading the field data
float r_time;       ///<Time to read from dfi
unsigned i_dummy;   ///<Average step
float f_dummy;      ///<Average time

//Reads the field data
float* d_v = (float *)DFI_IN->ReadData(
                      ret,    ///<Return code
                      step,   ///<Step number of the field data to read
                      gsize,  ///<the number of virtual cells in the computational domain
                      GVoxel, ///<Voxel size of the entire computational domain
                      GDiv,   ///<the number of subdomains
                      head,   ///<Starting point of the computational domain
                      tail,   ///<End point of the computational domain
                      r_time, ///<Reads time from dfi
                      true,   ///<Not read average
                      i_dummy,
                      f_dummy );
```

```
  // Error processing
  if( ret != CIO::E_CIO_SUCCESS ) {
    std::cerr << "Error ReadData." << std::endl;
    delete [] d_v;
    delete DFI_IN;
    return ret;
  }

  // Normal end processing
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<Deletes the array pointer
  delete DFI_IN;  ///<Deletes the pointer to the instance for reading
  return CIO::E_CIO_SUCCESS;
}
```

3.3 Output Function

3.3.1 Overview

CIOlib supports only 1 to 1 output of the field data file.

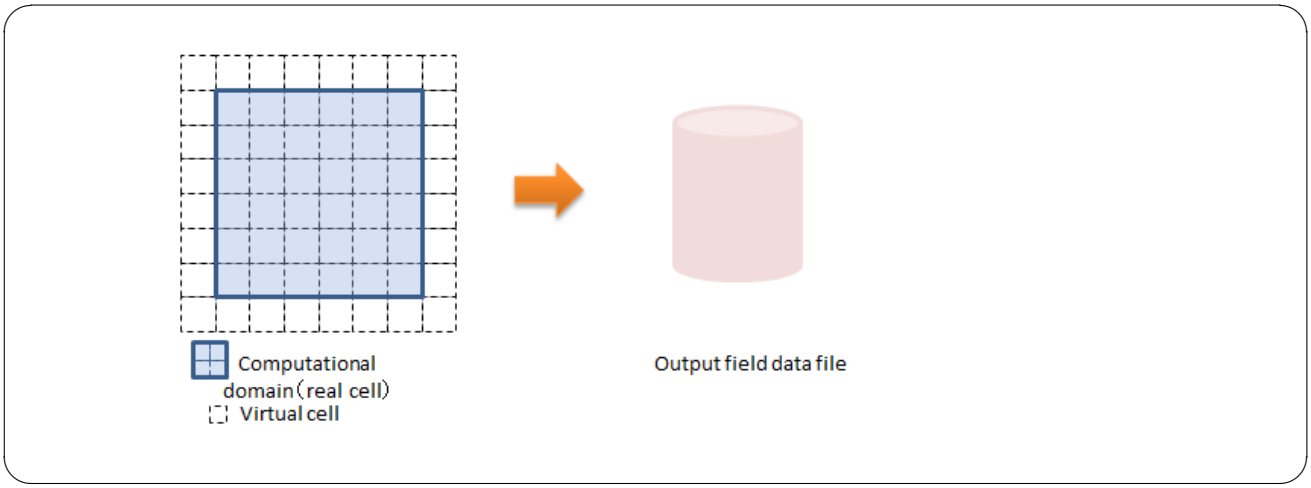


Fig. 3.6 1 to 1 output

3.3.2 Output Processing

CIOlib outputs the field data and the DFI data according to the following steps (See Fig.3.7.):

1. Gets a pointer to an instance of the output process(see subsection 3.3.3)
2. Registers the DFI information to output(see subsection 3.3.4)
3. Outputs the proc.dfi file(see subsection 3.3.5)
4. Outputs the field data file(see subsection 3.3.6)

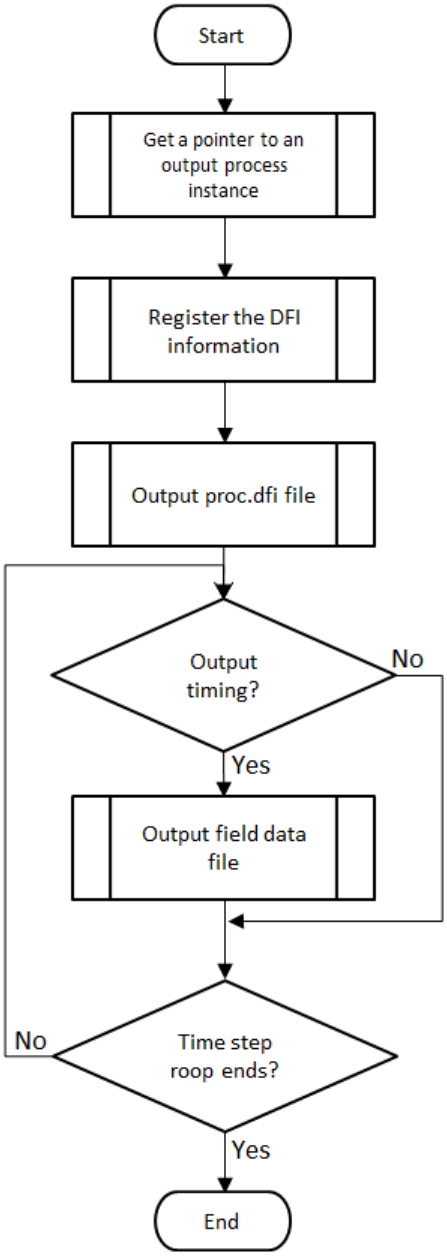


Fig. 3.7 Output Process

3.3.3 Getting the Pointer to an Output Instance

Any number of cio_DFI class instances can be created for each kind of DFI file. The method for getting the pointer to the instance is defined in cio_DFI.h as follows:

Creating an instance of the output process and getting the pointer to the instance (float type)

```
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);
```

Gets the pointer to cio_DFI class's instance.

comm	[*input*]	MPI communicator
DfiName	[*input*]	name of the index.dfi file to output
Path	[*input*]	directory where the field data to output exists
Prefix	[*input*]	base file name
format	[*input*]	file format of the field data (see Table 3.3)
GCell	[*input*]	the number of virtual cells to output
DataType	[*input*]	data type of the field data (see Table 3.4)
ArrayShape	[*input*]	array shape of the field data (see Table 3.5)
nComp	[*input*]	the number of components of the field data (scalar is 1 and vector is 3)
proc_fname	[*input*]	name of the proc.dfi file to output
G_size	[*input*]	voxel size of the entire computational domain of X, Y, and Z directions (an array of 3 words)
pitch	[*input*]	voxel pitch of X, Y, and Z directions (an array of 3 words float type)
G_origin	[*input*]	original point's coordinate value for the entire computational domain (an array of 3 words float type)
division	[*input*]	the number of subdomains for the X, Y, and Z directions (an array of 3 words)
head	[*input*]	starting point of the computational domain for X, Y, and Z directions (an array of 3 words)
tail	[*input*]	end point of the computational domain for X, Y, and Z directions (an array of 3 words)
hostname	[*input*]	host node name
TSliceOnOff	[*input*]	flag telling process to output field data into the time slice directory (see Table 3.2.)
Return Value		pointer to the cio_DFI class's instance

Be sure to delete the instance pointer when it is no longer needed.

```
//DFI instance
cioDFI *DFI_OUT_PRS = cio_DFI::WriteInit(,,,);
              :
            (Processing)
              :
//Delete the DFI instance
delete DFI_OUT_PRS;
```

Get the instance pointer with the above method and use it to access member functions from your program.

Creating an instance of the output process and getting the pointer to the instance (double type)

```
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);
```

Gets the pointer to cio_DFI class's instance.

comm	[*input*]	MPI communicator
DfiName	[*input*]	name of the index.dfi file to output
Path	[*input*]	directory where the field data to output exists
Prefix	[*input*]	base file name
format	[*input*]	file format of the field data (see Table 3.3)
GCell	[*input*]	the number of virtual cells to output
DataType	[*input*]	data type of the field data (see Table 3.4)
ArrayShape	[*input*]	array shape of the field data (see Table 3.5)
nComp	[*input*]	the number of components of the field data (scalar is 1 and vector is 3)
proc_fname	[*input*]	file name of proc.dfi to output
G_size	[*input*]	voxel size of the entire computational domain for X, Y, and Z directions (an array of 3 words)
pitch	[*input*]	voxel pitch for X, Y, and Z directions (an array of 3 words double type)
G_origin	[*input*]	original point's coordinates value for the entire computational domain (an array of 3 words double type)
division	[*input*]	the number of subdomains for X, Y, and Z directions (an array of 3 words)
head	[*input*]	starting point of the computational domain for X, Y, and Z directions (an array of 3 words)
tail	[*input*]	end point of the computational domain for X, Y, and Z directions (an array of 3 words)
hostname	[*input*]	host node name
TSliceOnOff	[*input*]	flag telling process to output field data into the time slice directory (see Table 3.2)
Return Value		pointer to the cio_DFI class instance

Be sure to delete the instance pointer when it is no longer needed.

```
//DFI instance
cioDFI *DFI_OUT_PRS = cio_DFI::WriteInit(,,,);
              :
            (Processing)
              :
//Delete the DFI instance
delete DFI_OUT_PRS;
```

Get the instance pointer with the above method and use it to access member functions from your program.

3.3.4 Additional Registration of DFI Information

When outputting time-sliced data, information from each output step can be combined into DFI output using data registration. The methods for registering step-wise data for DFI output are defined in cio_DFI.h as follows:

1. Register the unit (outputs the unit in Unit of the DFI file):

Registering the unit

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

Registers the unit to Unit

Name	[*input*]	unit to register	(”Length”,”Velocity”,,,,)
Unit	[*input*]	label to attach to the unit	(”M”,”CM”,”MM”,”M/S”,,)
reference	[*input*]	standardized scale value	(”L0”,”V0”,,,,)
difference	[*input*]	difference value*	
BsetDiff	[*input*]	whether difference exists or not**	

* Omittable. If omitted, BsetDiff becomes invalid.
** Omittable. If omitted, the setting will default to false.

For specification of DFI file's Unit, see subsection 6.1.1, “Specification for the Index File (index.dfi).”

2. Register the instruction for outputting the field data into each TimeSlice directory:

Registers the instruction of outputting the field data into each TimeSlice directory

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

ONOFF	[*input*]	output instruction flag (see Table 3.2.)

3. Register the name of the DFI FileInfo's component:

Registers the name of DFI FileInfo's component

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

pcomp	[*input*]	position of the name of the component to register
compName	[*input*]	name of the component to register

For specification of DFI file's FileInfo, see subsection 6.1.1, “Specification for the Index File (index.dfi).”

4. Register the rank list to read:

Registers the rank list to read

```
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's Domain information
head	[*input*]	start index of the computational domain (an array of 3 words)
tail	[*input*]	end index of the computational domain (an array of 3 words)
readflag	[*input*]	how to read the field data (see Table 3.7.)
readRankList	[*output*]	rank list to read
Return Value		error code (see Tables 3.10 and 3.11)

3.3.5 Proc.dfi File Output

The method to output the proc.dfi file is defined in cio_DFI.h as follows:

proc.dfi file output

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

Outputs proc.dfi file.

comm	[*input*]	MPI communicator	
out_host	[*input*]	host name output instruction flag	false: not output true: output
Return Value	error code (see Tables 3.10 and 3.11)		

3.3.6 Field Data File Output

The field data file formats supported in CIOlib are SPH and BOV. (For details, see subsection 6.1.3.)

The method to output the field data file is defined in cio_DFI.h as follows:

Field data file output

```
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);
```

Outputs the field data file.

step	[*input*]	output step number	
time	[*input*]	output time	
sz	[*input*]	real voxel size of X, Y, and Z directions for the output data array, val (an array of 3 words)	
nComp	[*input*]	the number of components for the output data (scalar: 1, vector: 3)	
gc	[*input*]	the number of virtual cells for the output data array, val	
val	[*input*]	pointer to the output data array	
minmax	[*input*]	MinMax of the output data	
		for scalar	minmax[0]=min minmax[1]=max
		for vector	minmax[0]=component 1's minX minmax[1]=component 1's maxX : minmax[2n-2]=component n's maxX minmax[2n-1]=component n's minX minmax[2n]=min of composite value minmax[2n+1]=max of composite value
avr_mode	[*input*]	average step, time output instruction false: output	
step_avr	[*input*]	average step	
time_avr	[*input*]	average time	
Return Value	error code （see Tables 3.10 and 3.11.）		

3.3.7 Sample Code for the Output Processing

```
#include "cio_DFI.h"
int main( int argc, char **argv )
{
  //CIO's error code
  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;
  }

  //Sets the dfi file name passed as a parameter
  if( argc != 2 ) {
    //Error: if DFI file name is not passed as a parameter
    std::cerr << "Error undefined DFI file name." << std::endl;
    return CIO::E_CIO_ERROR;
  }
  std::string dfi_fname = argv[1];

  //Defines the computational domain
  int GVoxel[3] = {64, 64, 64}; ///<Voxel size of the entire computational domain
  int GDiv[3]   = {1, 1, 1};    ///<The number of subdomains (i.e. the number of parallel execution)
  int head[3]   = {1, 1, 1};    ///<Starting point for the computational domain
  int tail[3]   = {64, 64, 64}; ///<End point for the computational domain
  int gsize     = 2;            ///<The number of virtual cells in the computational domain
  float pit[3]  = {1.0/64.0, 1.0/64.0, 1.0/64.0} ; ///<pitch
  float org[3]  = {-0.5, -0.5, -0.5};              ///<Original point's coordinates value
  //array size
  size_t size=(GVoxel[0]+2*gsize)*(GVoxel[1]+2*gsize)*(GVoxel[2]+2*gsize);

  std::string path  = "./";     ///<output directory
  std::string prefix= "vel";    ///<base file name
  int out_gc        = 0;        ///<the number of output virtual cell
  int ncomp         = 3;        ///<the number of data components
  CIO::E_CIO_FORMAT format  = CIO::E_CIO_FMT_SPH; ///<output format
  CIO::E_CIO_DTYPE datatype = CIO::E_CIO_FLOAT32; ///<data type
  std::string proc_fname    = "proc.dfi";         ///<proc file name
  std::string hostname      = "";                 ///<host name
  CIO::E_CIO_ONOFF TimeSliceOnOff = CIO::E_CIO_OFF; ///<TimeSlice output instruction

  //Gets the pointer to the output instance
  cio_DFI* DFI_OUT = cio_DFI::WriteInit(MPI_COMM_WORLD,  ///<MPI communicator
                                        dfi_fname,       ///<dfi file name
                                        path,            ///<output directory
                                        prefix,          ///<base file name
                                        format,          ///<output format
                                        out_gc,          ///<the number of output virtual cell
                                        datatype,        ///<data type
                                        CIO::E_CIO_NIJK, ///<array shape
                                        ncomp,           ///<the number of data components
                                        proc_fname,      ///<proc file name
                                        GVoxel,          ///<Voxel size for the all computational domain
                                        pit,             ///<pitch
                                        org,             ///<Original point's coordinates value
                                        GDiv,            ///<The number of subdomains
                                        head,            ///<Starting point for the computational domain
                                        tail,            ///<End point for the computational domain
                                        hostname,        ///<host name
                                        TimeSliceOnOff); ///<TimeSlice output option
```

```
  // Error processing
  if( DFI_OUT == NULL )
  {
    // Error: if the instance fails
    std::cerr << "Error Writeinit." << std::endl;
    return CIO::E_CIO_ERROR;
  }
  //Registers units
  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 file output
  DFI_OUT->WriteProcDfiFile(MPI_COMM_WORLD, ///<MPI communicator
                            false,          ///<host name output instruction
                            (float *)NULL); ///<Original point's coordinates value to output.
                                            ///<If it is null, the WriteInit value.

  //Allocates the array
  float *d_v = new float[size*ncomp];

  unsigned step=0;  ///<output step number
  float r_time=0.0; ///<output time
  float minmax[8];  ///<minmax
  //Clears minmax (sets to zero)
  for(int i=0; i<8; i++) minmax[i]=0.0;

  //Registers the component name
  DFI_OUT->setComponentVariable(0,"u");
  DFI_OUT->setComponentVariable(1,"v");
  DFI_OUT->setComponentVariable(2,"w");

  //Field data output
  ret = DFI_OUT->WriteData(step,   ///<output step number
                           r_time, ///<output time
                           GVoxel, ///<the real voxel number of d_v
                           ncomp,  ///<the real component number of d_v
                           gsize,  ///<the real virtual cell number of d_v
                           d_v,    ///<field data pointer to output
                           minmax, ///<the minimum value and the maximum value
                           true,   ///<not output the average
                           0,      ///<averaged step number
                           0.0);   ///<averaged time

  //Error processing
  if( ret != CIO::E_CIO_SUCCESS ) {
    //if field data output fails
    std::cerr << "Error WriteData." << std::endl;
    delete [] d_v;
    delete DFI_OUT;
    return ret;
  }

  //End normal processing
  std::cout << "Normal End." << std::endl;
  delete [] d_v;  ///<Deletes the array pointer
  delete DFI_OUT; ///<Deletes the pointer to the output instance
  return CIO::E_CIO_SUCCESS;
}
```

Chapter 4

Staging Tool

This chapter explains the use of a staging tool with CIOlib.

4.1 Staging Tool

4.1.1 Staging Tool

File RankMapper (frm) is a batch program for staging. When CIOlib is being used on a large-scale parallel computer, the staging tool copies files required for each computational node, or MPI rank, to the directory named according to the rank number.

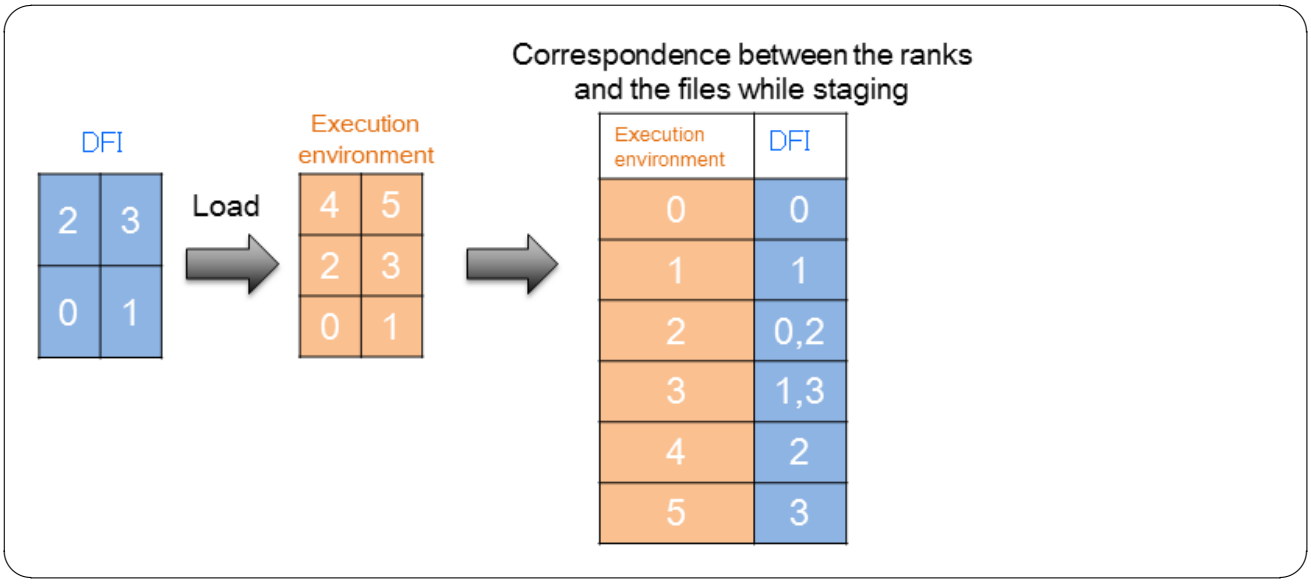


Fig. 4.1 Staging

4.1.2 Installation of Staging Tool

Frm operates on a login node. It is available on any machine with a staging system.

Two compilation environments exist on a login node: a native compilation environment and a cross compilation environment. Intel clusters use native compilation in most cases. The K computer/FX and BlueGene use cross compilation.

Frm is built by using the `-with-frm` option at a login node. It will be installed differently according to the compilation environments:

1. Native compilation environment
   Frm will be installed while compiling CIOlib by specifying the `--with-frm=yes` option. This option is set to no by default. Frm can be built through parallel operation in a native compilation environment; the frm available at a login node, however, is usually operated sequentially.
2. Cross compilation environment
   Frm will not be installed even if the `--with-frm=yes` option is specified during the CIOlib build. Therefore, it needs to be built manually. TextParser must be compiled with the native compiler beforehand [*1]. Frm in a cross compilation environment only operates sequentially.

To see how to build the CIO package, see subsections 2.1.2 and 2.1.7.

4.1.3 How to Use Frm

Command Parameters

To use frm, use the following command with associated parameters (`[]` indicates an optional parameter).

[*1] Another module, other than the module for a computational node, must be created

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

Explanation of Parameters

-i proc.txt (optional)
Specify the name of the file with the domain decomposition information of the solver.
Specify the name of the TextParser format file where the solver's Domain information is provided in proc.txt.
For specification of proc.txt with the domain decomposition information, see subsection 6.2.1.
If this option is omitted, a FCONV input file must be specified with -f.
If the option is not omitted, you cannot specify the -f option.

-f fconv.tp (optional)
Specify the input file name for FCONV.
If this parameter is omitted, the file with the domain decomposition information must be specified with -i.
This parameter cannot be specified together with the -i option.

-n [np] (optional)
Specify the number of parallel executions of Mx1 or MxM of FCONV.
Omitting this parameter is the same as specifying 1 as the number of parallel executions.

-s stepNo (optional)
Specify the number of steps to allocate using "stepNo".
If omitted, all steps will be the target step and copied to each rank directory.

Example: If -s 100 is specified
A file at the 100th step of the files specified in the DFI file will be copied in each rank directory.

-o outDir (optional)
Specify the name of the directory to which the allocation result should be copied, using "outDir".
If omitted, the current directory will be the output directory.

Example 1: If -o hoge is specified
a hoge/ directory will be created in the current directory, and the directories for each rank (e.g. 000000/, 000001/, etc.) will be created under that.

Example 2: If omitted
directories for each rank (e.g. 000000/, 000001/, etc.) will be created in the current directory.

DFIfile... (optional)
Specify the name of the DFI file to allocate. Multiple DFI files can be specified.
This parameter is required if the proc.txt file is specified using the -i option.
It cannot be specified together with the -f option.

Example: If vel.dfi and prs.dfi are specified
both vel.dfi and prs.dfi will be the allocation targets and copied to the same output directory.

Execution Example

This is an example of restarting the result of dividing by four (2, 1, 2) by dividing by eight (2, 2, 2):

- The following is content of the solver's Domain information stored file (solvproc.txt)

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

- DFI files to allocate
  The prs.dfi and vel.dfi under old/ directory are the allocation target.
  Actual sph files exist under SPH/ directory.

```
old/
  prs.dfi           <--DirectoryPath="SPH"
  vel.dfi           <--DirectoryPath="SPH"
  proc.dfi          <--referenced by prs.dfi and vel.dfi
  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
```

- Target step number to allocate
  The file at step number 100 is the target to allocate.

- Output directory
  hoge/

- Execution command

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

- Output result
  The hoge/ directory is created, and six-digit rank number directories are created under that. All necessary files are copied to these rank directories.

```
hoge/000000/
  prs.dfi                       <--DirectoryPath="./"
  prs_0000000100_id000000.sph
  prs_proc.dfi                  <--copied from proc.dfi
  vel.dfi                       <--DirectoryPath="./"
  vel_0000000100_id000000.sph
  vel_proc.dfi                  <--copied from proc.dfi

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
```

Chapter 5

Distributed Parallel File Converter

This chapter explains distributed parallel file conversion using CIOlib.

5.1 Distributed Parallel File Converter

5.1.1 Overview

The distributed parallel file converter tool (FCONV) combines files in SPH/BOV format, converts them, and then outputs the files using parallel MPI.

FCONV offers the following functions:

- File format conversion
- M to M data conversion
- M to 1 data conversion
- M to N data conversion

File format conversion

This takes SPH or BOV format files and converts them to SPH, BOV, PLOT3D, AVS, or VTK format file.

M to M data conversion

This takes the CIOlib distributed file output and converts it to the specified format without any change in the distributed state.

M to 1 data conversion

This takes the CIOlib distributed file output and converts it to the specified format as a single, combined file.

M to N data conversion

This takes the M distributed files output by CIOlib and converts them to specified format as N combined files.

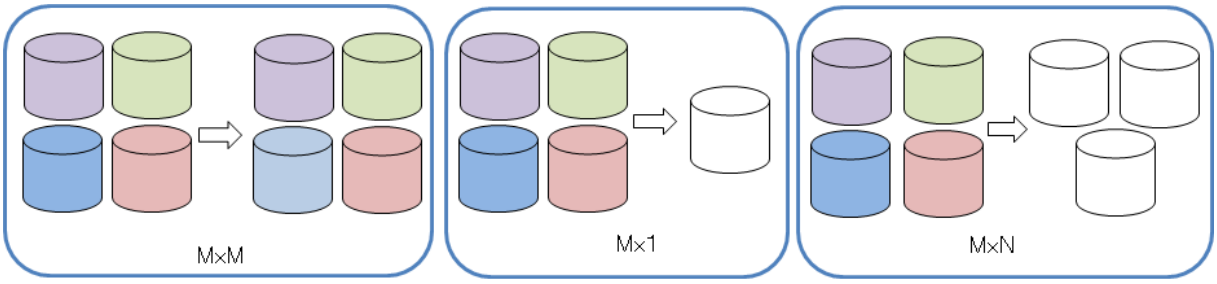


Fig. 5.1 Conversion image

5.1.2 Installation of FCONV

FCONV is built when the CIOlib package is built (or when configure, make, and makeinstall are invoked). It will be installed under `${prefix}`/bin when `make install` is invoked (where `prefix` is the directory you specified in the configure script). FCONV will not operate at a login node of the K computer or FX, as it only works in a parallel environment. For details on building the CIOlib package, see subsections 2.1.2 and 2.1.7.

5.1.3 How to Use FCONV

Execute the following command to use FCONV.

Excecution Command

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

Explanation of Parameters

-f conv.tp (required)
Specify the name of the FCONV input file where parameters are provided.
For specification of input files, see subsection 6.2.2.

-l logfile (optional)
Output a log into a file.

-v (optional)
Output a log to the screen.

Execution Example

When converting the result of division by three (1,1,3) to division by two (2,1,1) using 2 as the “thinning out” number:

- Input file(conv.tp)

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

- Execution command

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

- Conversion results
See Fig.5.2 and 5.3.

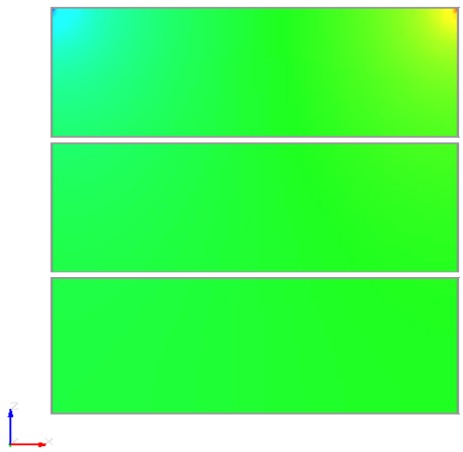


Fig. 5.2 Before conversion

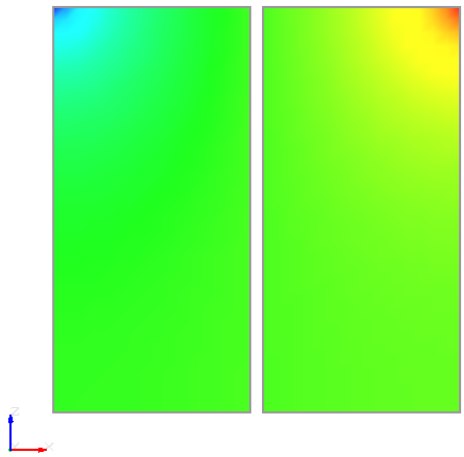


Fig. 5.3 After conversion

5.1.4 Output Format

Possible output formats according to file formats are indicated in Table 5.1.

Table. 5.1 Output format for each file format

	OutputFormatType		
	ascii	binary	Fortran_Binary
sph	-	✔	-
bov	-	✔	-
avs	X	✔	-
plot3d	✓	✔	✓
vtk	✓	✔	-

-: Unsupported
✓: Supported
✔: Default
X: Unsupported in FCONV

5.1.5 Possible Data Type for Each File Format

Possible data types and specifications for each file format are indicated in Table 5.2:

Table. 5.2 Data type for each file format

-: Unsupported ✓: Supported

	OutputDataType	bov header	sph	bov	avs	plot3d	vtk
bit	-	-	-	-	-	-	N/A*
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	✓	✓	✓	✓	✓

* The conversion source formats “sph” or “bov” do not support this, although the data type does exist.

5.1.6 Array Shape Supported by Each File Format

Possible array shapes for each file format are as follows:

Table. 5.3 Array shape for each file format

File Format	the number of components
sph	nijk(n=1or3)
bov	nijk,ijkn(n=arbitrary)
avs	nijk(n=arbitrary)
plot3d	ijkn(n=arbitrary)
vtk	nijk(n=arbitrary)

5.1.7 Defined Point

The position of a defined point differs according to file format, but will be either a centroid or a grid point.
The defined point for each file format and interpolation are as follows:

Interpolation to grid point

Interpolation from a cell center to grid points differs according to the presence of virtual cells.

Table. 5.4 Defined point for each file format

File format	Defined point
sph	centroid (Cell center)
bov	centroid (Cell center)
avs	Grid point
plot3d	Grid point
vtk	Grid point

Interpolation to grid point

Interpolation from a cell center to grid points differs according to the presence of virtual cells.

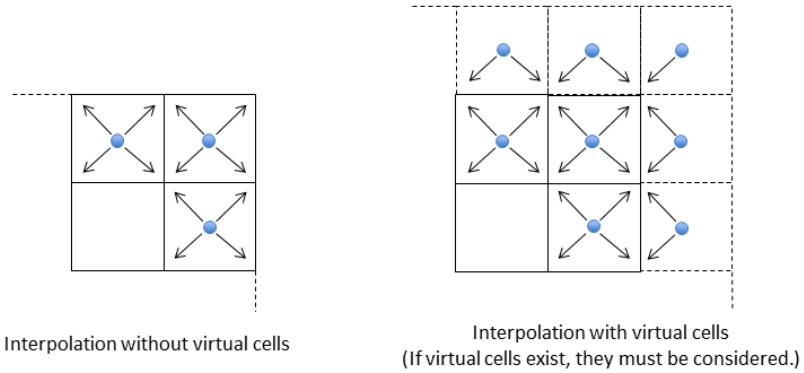


Fig. 5.4 Interpolation to grid points

5.1.8 Output File for Each File Format

Output files and file extensions for each file format are as follows:

Table. 5.5 Output file for each file format

File format	Field data	dfi	Header file	Others (coordinates value)
sph	*.sph	*.dfi	-	-
bov	*.dat	*.dfi	*.bov	-
avs	*.dat	-	*.fld	*.cod
plot3d	*.func	-	-	*.xyz
vtk	*.vtk	-		

5.1.9 Thinning Out

FCONV can be output after thinning out. When the thinning out number is two, it will proceed as follows:

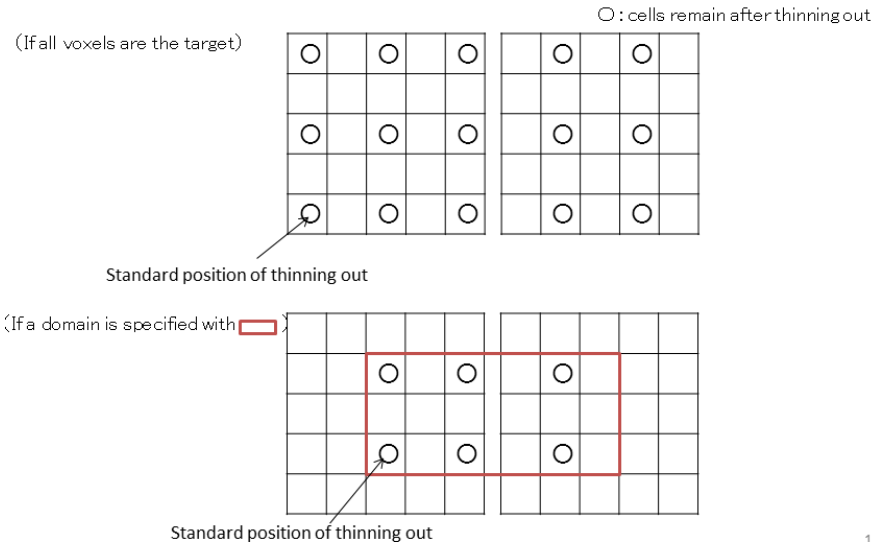


Fig. 5.5 Example for the case in which the thinning out number is two

5.1.10 Allocation of Files to Each Process

FCONV offers step standard and rank standard as its file allocation methods:

step standard

1. Make a list sorted according to the step number for each input DFI.
2. Distribute the sorted list equally to ranks according to the number of parallel executions of FCONV.
3. If the list is not divisible by the number of parallel executions of FCONV, the step number to be allocated to each rank will be increased in ascending order of the ranks of FCONV.
4. All the DFI rank files for the step number are allocated to each rank of FCONV.

For example, when prs and vel are distributed using four steps and three divisions, and the result is converted using five parallels, the process is as shown in Fig. 5.6.

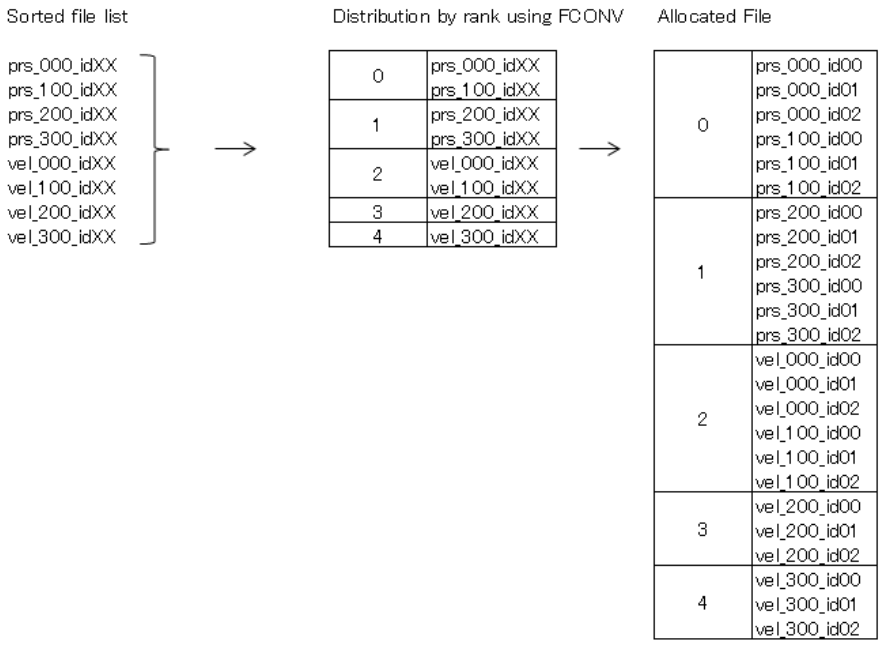


Fig. 5.6 Example of step standard

Supplement:

1 File allocation of this example will be more efficient if a common divisor of the number of DFI multiplied by the step number is used (8,4,2 parallel in this example).

2 Mx1 is always step standard.

Rank standard

1. Make a list sorted according to the DFI rank number for each input DFI.
2. Distribute the sorted list equally to ranks according to the number of parallel executions of FCONV.
3. If the list is not divisible by the number of parallel executions of FCONV, the DFI rank number to be allocated to each rank will be increased in ascending order of the ranks of FCONV.
4. All the step data at one DFI rank is allocated to the rank of FCONV.

For example, when prs and vel are distributed according to four steps and three divisions, and the result is converted using five parallels, the process is as shown in Fig. 5.7.

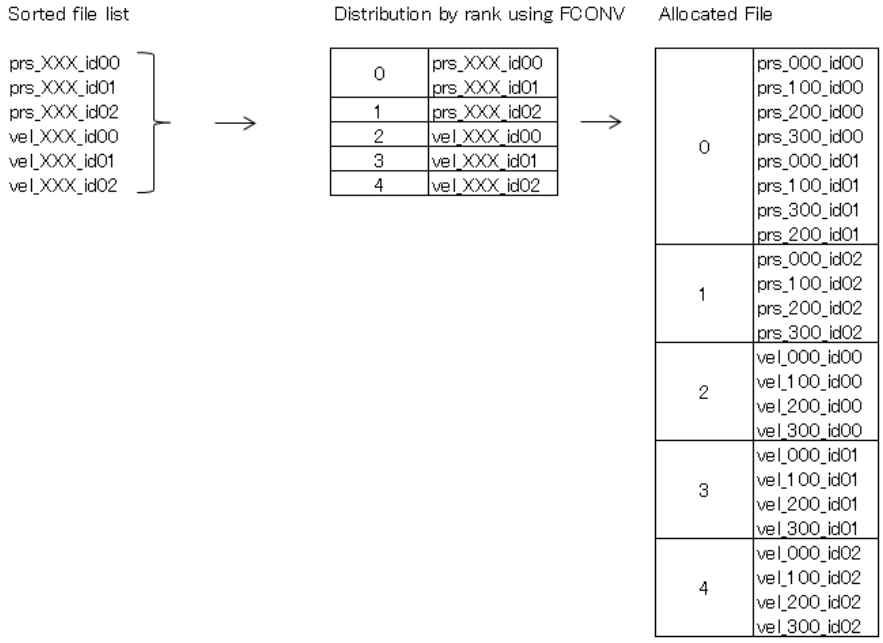


Fig. 5.7 Example of rank standard

Supplement:

1 File allocation of this example will be more efficient if a common divisor of the number of DFI multiplied by the rank number is used (6,3,2 parallel in this example).

2 Rank standard is effective when the step number is lower and the rank number is higher, and when you want to increase the number of parallel executions.

Chapter 6

File Specifications

This chapter explains file specifications of CIOlib.

6.1 File Specifications

6.1.1 Specifications for the Index File (index.dfi)

The index.dfi file consists of the following four blocks: file information (FileInfo), file path information (FilePath), unit (Unit), and time slice data (TimeSlice).

Specifications and samples of index.dfi are shown by block in the follow:

Specifications for file information (FileInfo)

```
FileInfo
{
  DFIType             = "Cartesian"   // Type of dfi (*1)
  DirectoryPath       = "./"          // Directory where the field data exists (*2)
  TimeSliceDirectory = "off"          // Option to create a directory every hour
  Prefix              = "vel"         // Base file name (*3)
  FileFormat          = "sph"         // File type, extension (*3)
  FieldFilenameFormat= "step_rank"    // File name format (*3)
  GuideCell           = 0             // The number of virtual cells
  DataType            = "Float32"     // Data type (*4)
  Endian              = "little"      // Data endian (*5)
  ArrayShape          = "nijk"        // Array shape (*6)
  Component           = 3             // component number (unneccesary for scalar)
  Variable[@]  {name = "u"}           // Name of components (the number of Component)
  Variable[@]  {name = "v"}           //
  Variable[@]  {name = "w"}           //
}
```

(*1) Only ”Cartesian” for CIOlib.
(*2) A relative path or an absolute path from index.dfi
(*3) File name
step_rank : [Prefix]_[step number: ten-digit]_id[RankID: six-digit].[ext]
rank_step : [Prefix]_id[RankID: six-digit]_[[step number: ten-digit].[ext]
When sequentially processed: [Prefix]_[step number: ten-digit].[ext]
(*4) Int8,UInt8,Int16,Uint16,Int32,Uint32,Int64,Uint64,Float32,Float64
(*5) little,big, If omitted, this is the same as the execution platform.
(*6) ijkn,nijk
ijkn:(imax,jmax,kmax,Component)
nijk:(Component,imax,jmax,kmax)

Specifications for file path (FilePath)

```
FilePath
{
  Process             = "proc.dfi"    //proc file name*
}
```

* A relative or absolute path from the index.dfi file

Specifications for units (UnitList)

```
UnitList
{
  Length {
    Unit              = "M"           // (NonDimensional, m, cm, mm)
    Reference         = 1.0           // Length scale used in standardization
  }
  Velocity {
    Unit              = "m/s"         // (NonDimensional, m/s)
    Reference         = 3.4           // Characteristic velocity(m/s)
  }
  Pressure {
    Unit              = "Pa"          // (NonDimensional, Pa)
    Reference         = 0.0           // Standard pressure(Pa)
    Difference        = 510.0         // Difference of pressure(Pa)
  }
  Temperature {
    Unit              = "C"           // (NonDimensional, C, K)
    Reference         = 10.0          // Standard temperature(C)
    Difference        = 510.0         // Difference of temperature(C)
  }
}
```

Specifications for time slice data (TimeSlice)

```
TimeSlice
{
  Slice{@} {          // the number of file output steps
    Step              = 0             // output step
    Time              = 0.0           // output time
    AverageTime       = 0.0           // average time (output as necessary)
    AverageStep       = 0             // averaged step number (output as necessary)
    VectorMinMax {    // min/max of the composite value of u, v, and w (only when Component>1)
      Min             = 0.0           // minimum value
      Max             = 0.0           // maximum value
    }
    MinMax{@} {       // the number of Component
      Min             = 0.0           // u minimum value
      Max             = 0.0           // u maximum value
    }
    MinMax{@} {
      Min             = 0.0           // v minimum value
      Max             = 0.0           // v maximum value
    }
    MinMax{@} {
      Min             = 0.0           // w minimum value
      Max             = 0.0           // w maximum value
    }
    • • • Additional arbitrary annotations available.
  }
  Slice{@} {
    ⋮
    ⋮
  }
}
```

6.1.2 Specifications for the Process Information File (proc.dfi)

The proc.dfi file consists of the following three blocks: domain information (Domain), parallel information (MPI), and process information (Process).

Specifications and samples of proc.dfi are shown by block below:

Domain specification (Domain)

```
Domain
{
  GlobalOrigin        = (-3.00,-3.00,-3.00)   // starting point coordinates for the
                                              //  computational domain
  GlobalRegion        = ( 6.00, 6.00, 6.00)   // length of each axis direction for the
                                              //  computational domain
  GlobalVoxel         = ( 64, 64, 64 )        // the number of voxels for the entire
                                              //  computational domain
  GlobalDivision      = ( 1, 1, 1)            // the number of subdomains for the entire
                                              //  computational domain
  ActiveSubdomainFile = "subdomain.dat"       // ActiveSubdomain file name*
}
```

* An absolute or relative path from index.dfi

Specifications for parallel information (MPI)

```
MPI
{
  NumberOfRank        = 128                   // Process number
  NumberOfGroup       = 1                     // group number
}
```

Specifications for process information (Process)

```
Process
{
  Rank{@} {   // the value of NumberOfRank
    ID        = 0                  // rank number
    HostName  = "Strontium.local" // host node name (not essential)
    VoxelSize = ( 64, 64, 64 )    // voxel size
    HeadIndex = ( 1, 1, 1 )       // index of starting point, start from (1, 1, 1) in global
    TailIndex = ( 64, 64, 64 )    // index of end point, end at (64, 64, 64) in global
  }
  Rank{@} {
    :
    :
  }
}
```

6.1.3 Specifications for Field Data Files

Specifications for field data files supported in CIOlib are provided below:

SPH format

V-Sphere Simple Voxel (SPH) is a binary file format that stores computation results from Solver flamework's V-Sphere. For details of the file format, see Table 6.1.

Table. 6.1 SPH file record format

Record name	Meaning
Data attribute record	Describes the data attribute （simple precision or double precision） （scalar or vector）
Voxel size record	Describes the voxel size
Original point's coordinates record	Describes the original point's coordinates
Voxel pitch record	Describes the voxel pitch
Time record	Describes the time step and the time
Data record	Describes the data

- `Data attribute record`
  This is the record of the data attribute consisting of the data classification flag and the data type flag along with their respective record lengths. The data classification flag indicates whether the data is scalar or vector. The data type flag indicates whether the data is simple precision or double precision.

Table. 6.2 Data attributes record

Name	Expression	Size	Explanation
Size	integer	4 bytes	Record length（=8）（*1）
svType	integer	4 bytes	Data classification flag（*2）
dType	integer	4 bytes	Data type flag（*3）
Size	integer	4 bytes	Record length（=8）（*1）

（*1） `Record length`
This puts the data between the byte numbers of the data record length in order to adjust to Fortran output without format.

（*2） `Data classification flag`
This determines whether the data is scalar or vector. Possible values are shown in Table 6.3:

Table. 6.3 Data classification flag

Data classification	svType value
scalar data	1
vector data	2

（*3） `Data type flag`
This determines whether the data type is single precision or double precision. Possible values are shown in Table 6.4:

Table. 6.4 Data type flag

Data type	dType value
single precision	1
double precision	2

- `Voxel size record`
  This is the record of the voxel size (the number of voxels in the computational domain). Possible values are found in Table 6.5.

Table. 6.5 Voxel size record

Name	Expression	Size	Explanation
Size	integer	4 bytes	record length（= 12 or 24）**
IMAX	integer	4 or 8 bytes *	I direction voxel number
JMAX	integer	4 or 8 bytes *	J direction voxel number
KMAX	integer	4 or 8 bytes *	K direction voxel number
Size	integer	4 bytes	record lenght（=12 or 24）**

  * `Differs according to the value of the data type flag (dType):`
    If the data type flag is single precision (dType=1), then 4 bytes
    If the data type flag is double precision (dType=2), then 8 bytes
  ** `Differs according to the value of the data type flag (dType):`
    If the data type flag is single precision (dType =1), then 12 bytes
    If the data type flag is double precision (dType=2), then 24bytes

- `Original coordinates point record`
  Used to describe the original coordinate point of the computational domain. Possible values are found in Table tbl:origin.

Table. 6.6 Original coordinates point record

Name	Expression	Size	Explanation
Size	integer	4 bytes	record length（= 12 or 24）**
XORG	integer	4 or 8 bytes *	X axis direction original coordinates point
YORG	integer	4 or 8 bytes *	Y axis direction original coordinates point
ZORG	integer	4 or 8 bytes *	Z axis direction original coordinates point
Size	integer	4 bytes	record lenght（=12 or 24）**

  * `Differs according to the value of the data type flag (dType)`
    If the data type flag is single precision (dType=1), then 4 bytes
    If the data type flag is double precision (dType=2), then 8 bytes
  ** `Differs according to the value of the data type flag (dType)`
    If the data type flag is single precision (dType =1), then 12 bytes
    If the data type flag is double precision (dType=2), then 24 bytes

- `Voxel pitch record`
  This is the record of 1 voxel pitch. Possible values are given in Table 6.7.

Table. 6.7 Voxel pitch record

Name	Expression	Size	Explanation
Size	integer	4 bytes	recored length（=12 or 24） **
XPITCH	integer	4 or 8 bytes *	X direction voxel pitch
YPITCH	integer	4 or 8 bytes *	Y direction voxel pitch
ZPITCH	integer	4 or 8 bytes *	Z direction voxel pitch
Size	integer	4 bytes	record length（=12 or 24） **

* `Differs according to the value of the data type flag (dType):`
If the data type flag is single precision (dType=1), then 4 bytes
If the data type flag is double precision (dType=2), then 8 bytes
** `Differs according to the value of the data type flag (dType):`
If the data type flag is single precision (dType =1), then 12 bytes
If the data type flag is double precision (dType=2), then 24bytes

- `Time record`
  This is the record of the time step and the time.

Table. 6.8 Time record

Name	Expression	Size	Explanation
Size	integer	4 bytes	record length （=8 or 12） **
STEP	integer	4 or 8 bytes *	time step
TIME	integer	4 or 8 bytes *	time
Size	integer	4 bytes	record length （=8 or 12） **

* `Differs according to the value of the data type flag (dType):`
If the data type flag is single precision (dType=1), then 4 bytes
If the data type flag is double precision (dType=2), then 8 bytes
** `Differs according to the value of the data type flag (dType):`
If the data type flag is single precision (dType =1), then 8 bytes
If the data type flag is double precision (dType =1), then 16 bytes

- `Data record`
  This is the record of the data.
  – For scalar data (svType=1)

Name	Expression	Size *	Explanation
Size	integer	4 bytes	record length **
DATA(0,0,0)	real number	4 or 8 bytes	the data value of the grid point (0,0,0)
DATA(1,0,0)	real number	4 or 8 bytes	the data value of the grid point (1,0,0)
DATA(2,0,0)	real number	4 or 8 bytes	the data value of the grid point (2,0,0)
・・・			
DATA(IMAX-1,JMAX-1,KMAX-1)	real number	4 or 8 bytes	the data value of the grid point (IMAX-1,JMAX-1,Kmax-1)
Size	integer	4 bytes	record length **

* `Differs according to the value of the data type flag (dType)`
If the data type flag is single precision (dType=1), then 4 bytes
If the data type flag is double precision (dType=2), then 8bytes
** `Differs according to the value of the data type flag (dType)`
If the data type flag is single precision (dType =1), then IMAX × JMAX × KMAX × 4 (bytes)
If the data type flag is double precision (dType=2), then IMAX × JMAX × KMAX × KMAX × 8 (bytes)

– For vector data (svType=2)

Name	Expression	Size *	Explanation
Size	integer	4 bytes	record length **
U(0,0,0)	real number	4 or 8 bytes	U data value of the grid point (0,0,0)
V(0,0,0)	real number	4 or 8 bytes	V data value of the grid point(0,0,0)
W(0,0,0)	real number	4 or 8 bytes	W data value of the grid point(0,0,0)
U(1,0,0)	real number	4 or 8 bytes	U data value of the grid point (1,0,0)
V(1,0,0)	real number	4 or 8 bytes	V data value of the grid point(1,0,0)
W(1,0,0)	real number	4 or 8 bytes	W data value of the grid point(1,0,0)
・・・			
U(IMAX-1,JMAX-1,KMAX-1)	real number	4 or 8 bytes	U data value of grid point(IMAX-1,JMAX-1,KMAX-1)
V(IMAX-1,JMAX-1,KMAX-1)	real number	4 or 8 bytes	V data value of grid point(IMAX-1,JMAX-1,KMAX-1)
W(IMAX-1,JMAX-1,KMAX-1)	real number	4 or 8 bytes	W data value of grid point(IMAX-1,JMAX-1,KMAX-1)
Size	integer	4 bytes	record length **

* Differs according to the value of the data type flag (dType):
If the data type flag is single precision (dType=1), then 4 bytes
If the data type flag is double precision (dType=2), then 8 bytes

** Differs according to the value of the data type flag (dType):
If the data type flag is single precision (dType =1), then IMAX × JMAX × KMAX × 4 × 3 (bytes)
If the data type flag is double precision (dType=2), then IMAX × JMAX × KMAX × 8 × 3 (bytes)

BOV format

The Brick of Values (BOV) format is used by the visualization software known as "VisIt".
It only uses data arrays (See Tables 6.9 and 6.10.)

Table. 6.9 Example description for ijkn array v(i,j,k,n)

Array element	Explanation
v(0,0,0,0)	Data value of component 0 of the grid point (0,0,0)
v(1,0,0,0)	Data value of component 0 of the grid point (1,0,0)
・・・	
v(imax-1,jmax-1,kmax-1,0)	Data value of component 0 of the grid point (imax-1,jmax-1,kmax-1)
v(0,0,0,1)	Data value of component 1 of the grid point (0,0,0)
・・・	
v(imax-1,jmax-1,kmax-1,n-1)	Data value of component n-1 of the grid point (imax-1,jmax-1,kmax-1)

Table. 6.10 Example description of nijk array v(n,i,j,k)

Array element	Explanation
v(0,0,0,0)	Data value of component 1 of the grid point (0,0,0)
v(1,0,0,0)	Data value of component 0 of the grid point(1,0,0)
・・・	
v(n-1,0,0,0)	Data value of component n-1 of the grid point(0,0,0)
v(0,0,0,1)	Data value of component 1 of the grid point(0,0,0)
・・・	
v(n-1,imax-1,jmax-1,kmax-1)	Data value of component n-1 of the grid point(n-1,imax-1, jmax-1, kmax-1)

6.1.4 Specifications for Subdomain Information File

Table. 6.11 Specifications for subdomain information file

Name	Expression	Type	Size	Explanation
Identifier	String	uchar	4bytes	Endian identifier *
Size X	integer	uint	4bytes	the number of subdomains of X direction
Size Y	integer	uint	4bytes	the number of subdomains of Y direction
Size Z	integer	uint	4bytes	the number of subdomains of Z direction
Contents	integer	uchar	1bytes x SizeX x SizeY x SizeZ	Active subdomain flag**

* Corresponding ASCII codes are set as follows:
for little endian, use the order 'S', 'B', 'D', 'M'
for big endian, use the order 'M', 'D', 'B', 'S'

** Active subdomain flag for each domain is stored in the order of X → Y → Z.
If it is active, 1 is stored. If it is not active, 0 is stored.

6.1.5 Sample of DFI Files

Sample of index.dfi

```
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
    }
  }
}
```

Sample of proc.dfi

```
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 Specifications for File (Tool)

6.2.1 Specifications for Domain Decomposition Information File for Staging

Specifications for the domain decomposition information file

```
Domain (*1)
{
  GlobalVoxel         = ( 64, 64, 64 )        // the number of voxels for the entire
                                              //  computational domain
  GlobalDivision      = ( 1, 1, 1)            // the number of subdomains for the
                                              //  computational domain
  ActiveSubdomainFile = "subdomain.dat"       // ActiveSubdomain file name
}
FCONVInfo
{
  InputFile           = "conv.tp"             // FCONV input file name
  NumberOfProcess     = 4                     // the number of parallel execution for
                                              //  Mx1 or MxM
}
MPI(*3)
{
  NumberOfRank        = 1                     // Process number
}
Process(*3)
{
  Rank[@] {                                   // the NumberOfRank
    ID                = 0                     // rank number
    VoxelSize         =( 64, 64, 64 )         // voxel size
    HeadIndex         =(  1,  1,  1 )         // index of starting point(*4)
    TailIndex         =( 64, 64, 64 )         // index of end point(*4)
  }
}
```

(*1) Domain tag is required, but ActiveSubdomainFile can be arbitrary.
If CropIndexStart and CropIndexEnd are specified in FCONV, the value of GlobalVoxel will be invalid.
GlobalDivision is valid only for MxN.

(*2) FCONVInfo tag is arbitrary.
File conversion type (ConvType), CropIndexStart and CropIndexEnd are read from the file specified by InputFile.

(*3) MPI and Process are arbitrary.
If the rank location direction is not I→J→K or the position of HeadIndex or TailIndex is different, they must be described.

(*4) HeadIndex and TailIndex
If the rank location direction is not I→J→K, HeadIndex and Tailindex must be described.
When CropIndexStart and CropIndexEnd are specified in FCONV, these values are invalid.

When the number of grids is NV and the number of subdomains is ND (the rank number is 0 to ND-1) for a direction, the number of grids of a rank is int (NV/ND). However, when the rank number <NV%ND, the number of grids of the rank is +1.

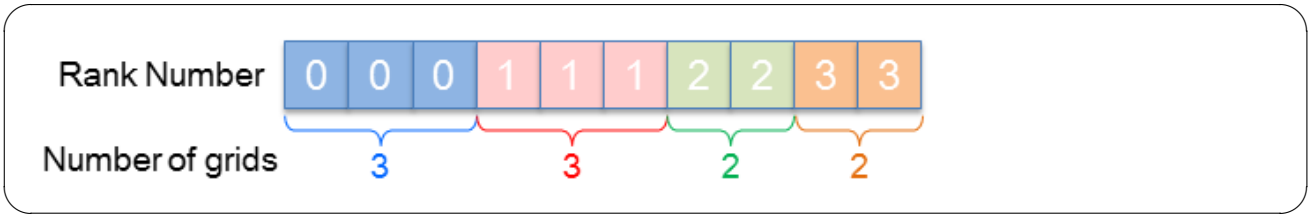


Fig. 6.1 Example for the number of grids equal to 10 and the number of subdomains equal to 4

6.2.2 Specifications for Input File for Distributed Parallel File Converter

Specifications for input file for distributed parallel file converter

```
ConvData{
  InputDFI[@]="prs.dfi"               a list of dfi to convert (required)
  InputDFI[@]="vel.dfi"
     :
  OutputDFI[@]="prs2.dfi"             a list of dfi to output (optional)
  OutputDFI[@]="vel2.dfi"
     :
  OutputProcDFI[@]="prs2_proc.dfi"    a list of proc.dfi to output (optional)
  PutputProcDFI[@]="vel2_proc.dfi"
     :
  ConvType="MxN"                      file conversion type (required)
  OutputDivision=(2,2,2)              output decomposition information (optional)
  OutputFormat="sph"                  output file format (optional)
  OutputDataType="Float32"            output data type (optional)
  OutputFormatType="binary"           output format (optional)
  OutputDir="conv_out"                output directory (required)
  ThinningOut=2                       the thinning out number (optional)
  OutputArrayShape="nijk"             output array shape (optional)
  OutputFilenameFormat="step_rank"    output filename format (optional)
  OutputGuideCell=0                   the number of output guide cell (optional)
  MultiFileCasting="step"             allocation of files at parallel execution (optional)
  CropIndexStart=(1,1,22)             start index for input domain (optional)
  CropIndexEnd=(64,64,32)             end index for input domain (optional)
}
```

1. InputDFI
   - “File name with a relative path”, “File name with the absolute path”, and “only file name” can be specified.
2. OutPutDFI
   - If this is omitted, the DFI file will not be output.
   - The number specified must be the same as that for InputDFI.
   - It must correspond to InputDFI in the description order.
   - Only valid for SPH and BOV
   - Only the file name can be specified. (The file name will be output in the current directory at runtime.)
3. OutputProcDFI
   - Can be specified only when OutputDFI is valid.
   - The number specified must be the same as that for InputDFI.
   - It must correspond to InputDFI in the description order.
   - If omitted, a file name specified at OutputDFI with the ”_proc” suffix will be output (e.g., ”prs2_proc.dfi”).
   - Only the file name can be specified. (The file name will be output in the current directory at runtime.)
4. ConvType
   - Specify ”Mx1”, ”MxN”, or ”MxM” as file conversion type. (For details, see Fig.5.1, “Conversion image.”)
5. OutputDivision
   - Specify output division information for each direction (IDIV, JDIV, and KDIV).
   - Valid only when file conversion type (ConvType) is MxN.
   - If specified, the number of parallel executions must equal to IDIV * JDIV * KDIV.
   - If omitted, the auto-dividing function of CPMlib divides output according to the number of parallel executions.
6. OutputFormat
   - Specify the output format as ”sph”, ”bov”, ”avs”, ”plot3d”, or ”vtk”.
   - If omitted, the file will be output according to the file format specified at the input DFI.
7. OutputDataType
   - Specify the output data type as ”Int8”,”UInt8”,”Int16”,”UInt16”,”Int32”,”UInt32”,”Float32”, or ”Float64”.
   - If omitted, the data type will not be converted.

8. OutputFormatType
   - Specify the output format type as ”ascii”,”binary”, or ”Fortran_Binary”.
     (For details, see Section 5.1.)
9. ThinningOut
   - If the value is one or less, thinning out will not be performed. If the value is two or higher, thinning out will be performed.
     (For details, see the example in subsection 5.1.9.)
   - If omitted, thinning out will not be performed.
10. OutputArrayShape
    - Specify output array shape as ”nijk” or ”ijnk”.
    - Valid only when output file format is BOV.
    - If a format other than the BOV is specified, the array will be output in an automatically corresponding array shape.
    - If omitted, the array will be output in the same shape as the input.
11. OutputFilenameFormat
    - Specify the file name format of output files as ”step_rank” or ”rank_step”.
      step_rank:[Prefix]_[StepNo,10 digit]_id[RankID,6 digit].[ext]
      rank_step:[Prefix]_id[RankID,6 digit]_[StepNo,10 digit].[ext]
    - If omitted, step_rank is used.
    - If an output file is sequential data, the file name will be
      step_rank:[Prefix]_[StepNo,10 digit].[ext] regardless of this specification.
      (RankID will not be output, following the specification of CIOlib.)
12. OutputGuideCell
    - Supports only SPH and BOV.
    - Does not support thinning out or grid points.
    - Available number of outputting guide cells is the number of guide cells output in the input file or fewer.
13. MultiFileCasting
    - Specify the file allocation type during parallel execution as ”step” or ”rank”.
    - Files will be allocated on the basis of the ”step” standard when step is specified, or on the basis of the ”rank” standard when rank is specified. (See subsection 5.1.10.)
    - Only ”step” is valid for Mx1. The ”step” is invalid for MxN.
    - If omitted, ”step” is used.
14. CorpIndexStart,CorpIndexEnd
    - Specify the index of each direction of the input as a whole (I direction, J direction, and K direction.)
    - Does not support MxM.
    - If both CropIndexStart and CropIndexEnd are specified, the domain between them will be the target.
    - If only the CropIndexStart parameter is specified, the target domain will be from CropIndexStart to the end.
    - If only the CropIndexEnd parameter is specified, the target domain will be from the head to CropIndexEnd.
    - If both are omitted, the entire domain will be the target.

Chapter 7

Update Information

This chapter provides information on updates of CIOlib.

7.1 Update Information

Revision 1 14/06/2014

- Initial Release of English version.

Chapter 8

Appendix

8.1 ClOlib API Method List

Table. 8.1 API Method list (C++ method names without class names are cio_DFI class members)

Function	C++ API	Note
create input instance	ReadInit	static method
create output instance	WriteInit	float version, static method
	WriteInit	double version, static method
get cio_FileInfoclass pointer	GetcioFileInfo	
get cio_FilePath class pointer	GetcioFilePath	
get cio_Unit class pointer	GetcioUnit	
get cio_Domain class pointer	GetcioDomain	
get cio_MPI class pointer	GetcioMPI	
get cio_TimeSlice class pointer	GetcioTimeSlice	
get cio_Process class pointer	GetcioProcess	
read field data	ReadData	return the array pointer of the read data
	ReadData	read the data into an array pointer passed as a parameter
output field data	WriteData	
output proc.dfi file	WriteProcDfiFile	float version
	WriteProcDfiFile	double version
get array shape of DFI	GetArrayShapeString	get strings
	GetArrayShape	get enumerated types
get data type of DFI	GetDataTypeString	get strings
	GetDataType	get enumerated type
get number of DFI component	GetNumComponent	
converts data type from String to Enumerated type	ConvDatatypeS2E	static method
converts data type from Enumerated type to String	ConvDatatypeE2S	static method
get DFI GlobalVoxel	GetDFIGlobalVoxel	
get DFI GlobalDivision	GetDFIGlobalDivision	
add unit	AddUnit	
get unit element (by class)	GetUnitElem	
get unit (member variables)	GetUnit	
set component name of FileInfo	setComponentVariable	
get component name of FileInfo	getComponentVariable	
get composite value of DFI MINMAX	getVectorMinMax	
get DFI MinMax	getMinMax	
Create read rank list	CheakReadRank	
set interval step	setIntervalStep	
set interval time	setIntervalTime	
normalize interval time	normalizeTime	normalize base_time,interval_time,start_time, last_time
normalize interval base_time	normalizeBaseTime	
normalize interval interval	normalizeIntervalTime	
normalize interval start_time	normalizeStartTime	
normalize interval last_time	normalizeLastTime	
normalize interval DeltaT	normalizeDelteT	
get version No of CIO	getVersionInfo	static method

List of Tables

List of Figures