Particle Data Manegement Library

User's Guide

Ver. 0.1

Advanced Visualization Research Team
Advanced Institute for Computational Science
RIKEN

7-1-26, Minatojima-minami-machi, Chuo-ku, Kobe, Hyogo, 650-0047, Japan

http://aics.riken.jp/

March 2014

Release

Version 0.1 20 March 2014

目次

第 1 章

PDMlib の概要

本章では PDMlib の概要と本ユーザガイドについて記述します。

1.1 PDMlib

PDMlib(Particle Data Manegiment Library) とは粒子データのファイル入出力管理を行う C++ クラスライブラリです. ユーザは C++ から本ライブラリを利用することができます。PDMlib には以下の機能が含まれています。

- DFI ファイル（メタ情報) による、解析領域、データ分割情報の管理
- zip, fpzip, RLE など複数のアルゴリズムによるデータ圧縮/伸長
- MxN ロード機能（並列数が異なるファイルからの入力およびデータ再分散）
- ポストプロセッサで読み込み可能な形式に変換するデータコンバータツール
- 京で Rank ディレクトリへのステージング指示行を生成するステージングツール

1.2 この文書について

1.2.1 文書中で使われる書式

以下の書式はシェルからのコマンド入力を表わします。前者は一般ユーザ権限でのコマンド入力、後者は root 権限でのコマンド入力を表します。

```
$ コマンド（コマンド引数）
# コマンド（コマンド引数）
```

1.2.2 動作環境

PDMlib は以下の環境で動作を確認しています。

- Linux PC
  - GNU/C++ ver 4.8.2 + OpenMPI
- Linux PC クラスタ
  - Intel C++ コンパイラ Version 12 + OpenMPI
- 京コンピュータ
  - 富士通コンパイラ + 富士通 MPI

第 2 章

パッケージのビルド

本章では、PDMlib のビルド方法について記述します。

2.1 PDMlib が依存するライブラリのビルド

PDMlib は以下の 5 つのライブラリに依存しているため、これらのライブラリがシステムにインストールされていない場合は PDMlib 自身のビルドの前に、これらのライブラリをビルドする必要があります。

- zlib
- fpzip
- Zoltan
- TextParser
- HDF5(optional)

HDF5 ライブラリは、PDMlib 本体からは使われておらず、H5PartConverter のみが依存しています。したがって、HDF5 ライブラリが無い場合でも H5PartConverter 以外の機能は全て利用することができます。

2.1.1 zlib のビルド

zlib は一般的な Linux ディストリビューション向けにバイナリパッケージが用意されていますので、そちらを使用してください。

2.1.2 fpzip のビルド

fpzip を配布元の URL(http://computation.llnl.gov/casc/fpzip/fpzip.html) よりダウンロード、展開した後に以下のコマンドを順に実行してください。

```
$cd src
$make
```

makefile 内では C++ コンパイラとして g++ が指定されているので、これ以外のコンパイラを使用する場合は CXX および CXXFLAGS 変数を変更してから make コマンドを実行してください。

正常にビルドできた場合は、” src ” と同じレベルにある”lib”ディレクトリの下に libfpzip.a が生成されています。

install スクリプトは用意されていませんので、以下の各ファイルを適切なディレクトリにコピーしてください。

- inc/fpzip.h
- lib/libfpzip.a

2.1.3　Zoltan のビルド

Zoltan を配布元の URL(http://www.cs.sandia.gov/ web1400/1400_download.html) からダウンロードし以下のコマンドを順に実行してください。

```
$tar xfz zoltan_distrib_v3.8.tar.gz
$cd Zoltan_v3.8
$mkdir BUILD
$cd BUILD/
$../configure --with-id-type=ulong --with-mpi-compilers=yes
$make everything
#make install
```

2.1.4　TextParser のビルド

TextParser のビルド方法は、TextParser に同梱されているドキュメントをご参照ください。

2.1.5　HDF5 のビルド

HDF5 ライブラリは一般的な Linux ディストリビューション向けにバイナリパッケージが用意されていますので、そちらを使用してください。

2.2　PDMlib のビルド

PDMlib 本体のビルドには Cmake を利用します。ビルドの実行

以下のコマンドを順に実行することで、ヘッダファイル、ライブラリ本体、付属ツール類がインストールされます。実行例中の PATH_TO_PDMLIB_DIR には CMakeLists.txt ファイルがあるディレクトリを指定してください。

```
$mkdir BUILD
$cd BUILD
$cmake PATH_TO_PDMLIB_DIR
$make install
```

cmake に指定可能なオプションの一覧は以下のとおりです。

オプション	内容	デフォルト値
CMAKE_CXX_COMPILER	使用する C++ コンパイラ	mpicxx
CMAKE_CXX_FLAGS	C++ コンパイラに渡すオプション	-g
CMAKE_INSTALL_PREFIX	PDMlib のインストール先	
build_samples	サンプルコードをコンパイルするかどうか指定するフラグ (on or off)	off
build_tests	テストプログラムをコンパイルするかどうか指定するフラグ (on or off)	off
FPZIP_ROOT	fpzip がインストールされているパス	
ZOLTAN_ROOT	Zoltan がインストールされているパス	
TP_ROOT	TextParser がインストールされているパス	

これらのオプションを使用する場合は cmake の-D オプションを使用して以下のように値を指定します。

```
$cmake -DCMAKE_CXX_COMPILER=mpiicpc -Dbuild_tests=on
```

第 3 章

PDMlib の機能説明

本章では PDMlib の機能を解説します。

3.1 メタ情報の管理

3.2 ファイル圧縮機能

3.3 ロードバランス機能

第 4 章

API 利用方法

本章では PDMlib の API の利用方法について記述します

4.1 ユーザプログラムへの組込み

ユーザプログラムへ組込む際には、以下の 3 つのソース修正を行う必要があります。

- ヘッダファイルのインクルード
- 入力機能 API の組込み（optional)
- 出力機能 API の組込み

4.2 ヘッダファイルのインクルード

PDMlib のヘッダファイルは”PDMlib.h” のみですので、PDMlib の API を呼び出すソースコード内に以下の include 文を追加してください。

```
#inclde <PDMlib.h>
```

4.3 入出力機能共通の注意

PDMlib はシングルトンパターンを使用しており、ユーザコード内で明示的にインスタンスを生成することはできません。PDMlib の API を呼び出す時は、静的メンバ関数である PDMlib::GetInstance() が返す PDMlib オブジェクトの参照を通じて関数を呼び出してください。

4.4 入力機能

PDMlib のファイル入力機能は次の手順で使用します。

1. PDMlib の初期化
2. コンテナ情報の取得
3. フィールドデータの読み取り

4.4.1　PDMlib の初期化方法

以下の関数を呼び出すことで PDMlib の初期化を行います。入力機能を使う場合は、この関数の第四引数に読み込む dfi ファイルのファイル名を指定します。この引数のデフォルト値は””となっており、未指定の場合は入力機能は使えません。

```
void Init(const int& argc, char** argv, const std::string& WriteMetaDataFile,
          const std::string& ReadMetaDataFile = "");
```

4.4.2　コンテナ情報の取得方法

コンテナ情報は以下に示す ContainerInfo 構造体として表わされています。

```
struct ContainerInfo
 {
   std::string Name;
   std::string Annotation;
   std::string Compression;
   SupportedType Type;
   std::string Suffix;
   int nComp;
   StorageOrder VectorOrder;
};
```

以下の関数を使うことで、DFI ファイル内で指定されているコンテナ情報の一覧を取得することができます。

```
std::vector<ContainerInfo> GetContainerInfo(void);
```

4.4.3　フィールドデータの読み取り

以下の関数によりフィールドデータを読み取ります。

```
template<typename T> int Read(const std::string& Name, size_t* ContainerLength,
                              T** Container, int* TimeStep = NULL,
                               bool read_all_files = false);
```

Name で指定した名前のコンテナを Container で指定したポインタのポインタが指す領域に対して読み込みます。**Container が NULL の時はライブラリ内部で動的に領域を確保して返すので、ユーザコード側で必ず delete してください。TimeStep として NULL ポインタが渡された場合と、*TimeStep が負の値だった場合はカレントディレクトリ内の最新のタイムステップのデータを読み込みます。それ以外の場合は、指定された TimeStep のデータを読み込みます。オプション引数の read_all_files が true の時は全 Rank が重複して全てのファイルを読み込みます。デフォルトでは、ファイル単位でブロック分割して自 Rank が担当するファイルのみを読み込むので、これ以外の方法でデータ分散を独自に行う場合にお使いください。

4.4.4　フィールドデータの一括読み取り

Read() 関数を用いる方法では、1 コンテナづつファイルを読み込みますが、読み込み対象のコンテナを事前に登録しておいて 1 回のルーチンコールで複数のファイルを読み込むこともできます。この場合は、読み込んだデータに対して

ロードバランスをとるためのマイグレーション処理を行うことも可能です。マイグレーション処理では座標データをもとに、近い位置にあるデータをなるべく同じプロセスが担当するように移動させます

コンテナの登録には以下の関数を用います。

```
template<typename T> int RegisterContainer(const std::string& Name, T** Container) const;
```

Name および Container の意味は Read() と同じです。一括読み取りには以下の関数を用います。

```
size_t ReadAll(int* TimeStep = NULL, const bool& MigrationFlag = false,
               const std::string& CoordinateContainer = "Coordinate");
```

引数の TimeStep は Read() と同様です。MigrationFlag に true を設定すると、前述のマイグレーション処理を行います。CoordinateContainer には座標データを保持するコンテナの名前を指定する必要があります。

4.5　出力機能

PDMlib のファイル出力機能は以下の手順で使用します。

1. PDMlib の初期化
2. コンテナ情報の設定
3. フィールドデータの出力

4.5.1　PDMlib の初期化方法

PDMlib の初期化方法は入力機能と同じく PDMlib::Init() によって行います。入力機能用に初期化済の場合は、改めて Init() を呼び出す必要はありません。

第三引数に出力する dfi ファイルのファイル名を指定してください。入力機能を使わない場合は第四引数の指定は不要です。

```
void Init(const int& argc, char** argv, const std::string& WriteMetaDataFile,
const std::string& ReadMetaDataFile = "");
```

また、出力するフィールドデータのベースファイル名は以下の関数によって設定することができます。

```
int SetBaseFileName(const std::string& FileName);
```

SetBaseFileName 関数で”FileName”を指定すると、実際に出力されるファイル名は、”FileName_Rank 番号_タイムステップ数. 拡張子”となります。拡張子は、後述の ContainerInfo 構造体のメンバ Suffix で設定されます。

4.5.2　コンテナ情報の設定

コンテナ情報は、前述の ContainerInfo 構造体を作成し、以下の関数に渡すことで登録されます。

```
void AddContainer(const ContainerInfo& Container);
```

4.5.3 フィールドデータの出力

以下の関数によりフィールドデータを出力します。

```
template<typename T> int Write(const std::string& Name, const size_t& ContainerLength,
                               T* Container, T MinMax[8], const int& NumComp,
                               const int& TimeStep, const double& Time);
```

第 5 章

付属ユーティリティ

本章では、PDMlib に付属している 2 種類のツールに関して記述します。

5.1 ステージングツール

京コンピュータでリスタート計算を行う時に前回の計算結果をどのプロセスに送り込むかを指定するステージング指示行を生成します。実行方法は以下のとおりです。

```
$ python2 stage.py NPROC DFI_FILE
```

NPROC には、これから実行する時に使用するプロセス数を、DFI_FILE には、リスタート計算に用いる計算結果の DFI_FILE をそれぞれ指定してください。

5.2 ファイルコンバータ

PDMlib の出力を可視化ソフトから読める形式に変換する 2 種類のコンバータが付属しています。

- FV14Converter
  FieldView version 14 以降で使われる Particle Path 形式へのコンバータ
- H5PartConverter
  VisIt などで読み込むことができる H5Part 形式へのコンバータ

両コンバータともに MPI 並列化されており、ファイル単位でのデータ分散を行っていますが、出力するファイル形式の制約からデータ分散の方法が異なります。

FieldView v14 の ParticlePath 形式では解析領域内に存在する全粒子を 1 タイムステップあたり 1 ファイルに出力する必要があります。このため、FV14Converter のデータ分散は時間方向でブロック分割を行なっています。

VisIt の H5Part reader にはこのような制限は無く同一ステップのデータが複数のファイルに分散されていても読み込むことができますが、各ファイルに TimeStep0 のデータが含まれていなければ正常に読み込めません。このため、H5PartConverter はソルバ実行時のデータ分散をそのまま使用し、1 つのファイルに全ステップのデータを出力しています。

なお、プロセス数を変更してリスタート計算を行なった場合のように、タイムステップによってプロセス数（=フィールドデータファイル数）が異なる場合は、全タイムステップを通じた、最小のプロセス数でコンバータが動作するように制限されています。これは、同一ファイル内に不連続なタイムステップが存在すると、VisIt での描画に不具合が生じていたため、この現象を回避するための対策です。

第 6 章

ファイル仕様

本章では PDMlib が出力するファイルの仕様について記述します。

6.1 DFI ファイルの仕様

DFI ファイルは TextParser 形式のファイルで以下の情報を持っています。

6.2 フィールドデータファイルの仕様

フィールドデータファイルは、次図のような合計 12byte のヘッダを持つバイナリファイルで作成されています。コンテナ部分のデータ型の情報などのメタデータは前述の DFI ファイルにしか含まれていないため、フィールドデータファイルのみでは正常にデータを読み取ることはできません。