User Guide of HDMlib

Hierarchical Data Management library

Ver 0.3.0

Advanced Institute for Computational Science

RIKEN

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

March 2014

目 次

第1章　HDMlibの概要

HDMlib の概要と本ユーザガイドについて説明します.

1.1 HDMlib

HDMlib(Hierarchical Data Management library) は階層型直交格子構造データの入出力管理を行う C++ クラスライブラリです. ユーザーは、C++で本ライブラリを利用できます.
HDMlib は、以下の機能を有します.

- 分散ファイル管理
- M x N ロード
- ボクセルデータ圧縮
- 結果データ圧縮
- ファイル入出力

1.2 この文書について

1.2.1 書式について

次の書式で表されるものは、Shell のコマンドです.

$ コマンド (コマンド引数)

または

 # コマンド (コマンド引数)

"$"で始まるコマンドは一般ユーザーで実行するコマンドを表し、"#"で始まるコマンドは管理者 (主に root) で実行するコマンドを表しています.

1.2.2 動作環境

HDM ライブラリは、以下の環境について動作を確認しています.

- Linux/Intel コンパイラ
  - CentOS6.2 i386/x86 64
  - Intel C++/Fortran Compiler Version 12 (icpc/ifort)
- MacOS X Snow Leopard 以降
  - MacOS X Snow Leopard
  - Intel C++/Fortran Compiler Version 11 以降 (icpc/ifort)
- 京コンピュータ

第2章　パッケージのビルド

この章では、HDMlib のコンパイルについて説明します.

2.1 パッケージのビルド

2.1.1 パッケージの構造

HDM ライブラリのパッケージは次のようなファイル名で保存されています.
(X.X.X にはバージョンが入ります)
HDMlib-$X.X.X$.tar.gz

このファイルの内部には, 次のようなディレクトリ構造が格納されています.

```
HDMlib-X.X.X
├── AUTHORS
├── COPYING
├── ChangeLog
├── INSTALL
├── LICENSE
├── Makefile.am
├── Makefile.in
├── NEWS
├── README
├── README.md
├── aclocal.m4
├── compile
├── config.h.in
├── configure
├── configure.ac
├── depcomp
├── doc
│   ├── Makefile
│   ├── Makefile.am
│   ├── Makefile.in
│   ├── dox_HDMlib.conf
│   ├── hdmlib_ug.pdf
│   ├── hdmlib_ug.tex
│   └── reference.pdf
├── examples
│   ├── README.txt
│   ├── SampleCreator
│   └── SampleLoader
├── hdm-config.in
├── include
├── install-sh
├── missing
└── src
```

これらのディレクトリ構造は, 次の様になっています.

・doc
この文書を含む HDMlib ライブラリの文書が収められています.

・examples
サンプルソースが収められています.

・include
ヘッダファイルが収められています. ここに収められたファイルは make install で$prefix/include にインストールされます.

・src
ソースが格納されたディレクトリです. ここにライブラリ libHDM.a が作成され ,make install で $prefix/lib にインストールされます.

2.1.2 パッケージのビルド

いずれの環境でも shell で作業するものとします. 以下の例では bash を用いていますが,shell によって環境変数の設定方法が異なるだけで, インストールの他のコマンドは同一です. 適宜, 環境変数の設定箇所をお使いの環境でのものに読み替えてください.
以下の例では, 作業ディレクトリを作成し, その作業ディレクトリに展開したパッケージを用いてビルド, インストールする例を示しています.

1. 作業ディレクトリの構築とパッケージのコピー
   まず, 作業用のディレクトリを用意し, パッケージをコピーします. ここでは, カレントディレクトリに work というディレクトリを作り, そのディレクトリにパッケージをコピーします.

   ```
   $ mkdir work
   $ cp [パッケージのパス] work
   ```

2. 作業ディレクトリへの移動とパッケージの解凍先ほど作成した作業ディレクトリに移動し, パッケージを解凍します.

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

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

   ```
   $ cd HDMlib-X.X.X
   ```

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

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

configure スクリプトの実行時には, お使いの環境に合わせたオプションを指定する必要があります.configure オプションに関しては,2.1.3 章を参照してください.configure スクリプトを実行することで, 環境に合わせた Makefile が作成されます.

5. make の実行
   make コマンドを実行し, ライブラリをビルドします.

```
$ make
```

make コマンドを実行すると, 次のファイルが作成されます.

```
src/libHDM.a
```

ビルドをやり直す場合は,make clean を実行して, 前回の make 実行時に作成されたファイルを削除します.

```
$ make clean
$ make
```

また,configure スクリプトによる設定,Makefile の生成をやり直すには,make distclean を実行して, 全ての情報を削除してから,configure スクリプトの実行からやり直します.

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

6. インストール
   次のコマンドで configure スクリプトの–prefix オプションで指定されたディレクトリに, ライブラリ, ヘッダファイルをインストールします.

```
$ make install
```

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

```
$ sudo make install
```

または,

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

インストールされる場所とファイルは以下の通りです.

```
${prefix}
├── bin
│   └── hdm-config
├── doc
│   ├── hdmlib_ug.pdf
│   └── reference.pdf
├── include
│   ├── BCMFileCommon.h
│   ├── BCMFileLoader.h
│   ├── BCMFileSaver.h
│   ├── BCMRLE.h
│   ├── BCMTypes.h
│   ├── BitVoxel.h
│   ├── DirUtil.h
│   ├── ErrorUtil.h
│   ├── FileSystemUtil.h
│   ├── IdxBlock.h
│   ├── IdxStep.h
│   ├── LeafBlockLoader.h
│   ├── LeafBlockSaver.h
│   ├── Logger.h
│   ├── PartitionMapper.h
│   ├── Vec3.h
│   ├── hdmVersion.h
│   ├── hdmVersion.h.in
│   └── mpi_stubs.h
├── lib
│   └── libHDM.a
└── share
    ├── AUTHORS
    ├── COPYING
    ├── ChangeLog
    ├── INSTALL
    ├── LICENSE
    ├── NEWS
    └── README
```

7. アンインストール
   アンインストールするには、書き込み権限によって、

   ```
   $ make uninstall
   ```

   または,

   ```
   $ su
   password:
   ```

```
# make uninstall
# exit
```

を実効します。

2.1.3 configure スクリプトのオプション

- −−prefix=*dir*
  prefix は, パッケージをどこにインストールするかを指定します.prefix で設定した場所が −−prefix=/usr/local/HDMlib の時,

  ライブラリ:/usr/local/HDMlib/lib
  ヘッダファイル:/usr/local/HDMlib/include

  にインストールされます.
  prefix オプションが省略された場合は, デフォルト値として/usr/local/HDMlib が採用され, インストールされます.

- コンパイラ等のオプション
  コンパイラ, リンカやそれらのオプションは, configure スクリプトで半自動的に探索します. ただし, 標準ではないコマンドやオプション, ライブラリ, ヘッダファイルの場所は探索出来ないことがあります. また, 標準でインストールされたものでないコマンドやライブラリを指定して利用したい場合があります. そのような場合, これらの指定を configure スクリプトのオプションとして指定することができます.

  CXX
  C++ コンパイラのコマンドパスです.

  CXXFLAGS
  C++ コンパイラへ渡すコンパイルオプションです.

  LDFLAGS
  リンク時にリンカに渡すリンク時オプションです. 例えば, 使用するライブラリが標準でないの場所 <libdir> にある場合,-L<libdir> としてその場所を指定します.

  LIBS
  利用したいライブラリをリンカに渡すリンク時オプションです. 例えば, ライブラリ <library> を利用する場合,-l<library> として指定します.

  F90
  Fortran90 コンパイラのコマンドパスです.

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

- ライブラリ指定のオプション
  HDM ライブラリを利用する場合, コンパイル, リンク時に, MPI ライブラリと TextParser ライブラリと BCM ライブラリが必ず必要になります. これらのライブラリのインストールパスは, 次に示す configure オプションで指定する必要があります.

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

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

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

  −−with-bcm=dir
  BCM ライブラリのインストール先を指定します.

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

  なお,configure オプションの詳細は,./configure –help コマンドで表示されますが,HDM ライブラリでは, 上記で説明したオプション以外は無効となります.

2.1.4 configure 実行時オプションの例

- Linux/Max OS X の場合

  HDM ライブラリの prefix: /opt/HDMlib
  MPI ライブラリ:OpenMPI, /usr/local/openmpi
  TextParser ライブラリ: /usr/local/textparser
  BCM ライブラリ: /usr/local/bcmlib
  C++ コンパイラ:icpc
  F90 コンパイラ:ifort

  の環境の場合, 次のように configure コマンドを実行します.

```
$ ./configure   −−prefix=/opt/HDMlib \
                −−with-ompi=/usr/local/openmpi \
                −−with-parser=/usr/local/textparser \
                −−with-bcm=/usr/local/bcmlib \
```

```
−−with-comp=INTEL \
CXX=icpc \
FC=ifort
```

- 京コンピュータの場合

HDM ライブラリの prefix:/home/userXXXX/HDMlib
TextParser ライブラリ:/home/userXXXX/textparser
C++ コンパイラ:mpiFCCpx
F90 コンパイラ:mpifrtpx

の環境の場合, 次のように configure コマンドを実行します.

```
$ ./configure  –host=sparc64-unknown-linux-gnu \
               −−prefix=/home/userXXXX/HDMlib \
               −−with-parser=/home/userXXXX/textparser \
               −−with-comp=FJ \
               CXX=mpiFCCpx \
               FC=mpifrtpx
```

2.1.5 hdm-config コマンド

HDM ライブラリをインストールすると, $prefix/bin/hdm-config コマンド (シェルスクリプト) が生成されます. このコマンドを利用することで, ユーザーが作成したプログラムをコンパイル, リンクする際に, HDM ライブラリを参照するために必要なコンパイルオプション, リンク時オプションを取得することができます.

hdm-config コマンドは, 次に示すオプションを指定して実行します.

–cxx
: HDM ライブラリの構築時に使用した C++ コンパイラを取得します.

–cflags
: C++コンパイラオプションを取得します.

–libs
: HDM ライブラリのリンクに必要なリンク時オプションを取得します.

ただし, hdm-config コマンドで取得できるオプションは, HDM ライブラリを利用する上で最低限必要なオプションのみとなります.
最適化オプション等は必要に応じて指定してください.

2.1.6 提供環境の作成

提供環境の作成を行うには,configure スクリプト実行後に, 以下のコマンドを実行します.

$ ./make dist

上記コマンドを実行すると, 提供環境が

HDMlib-$X.X.X$.tar.gz

という圧縮ファイルに保存されます.(X.X.X にはバージョンが入ります)

2.2 HDM ライブラリの利用方法

HDM ライブラリは, C++ プログラム内で利用できます. 以下に, ユーザーが作成する HDM ライブラリを利用するプログラムのビルド方法を示します.
以下の例では,configure スクリプトで ”–prefix=/usr/local/HDMlib ” を指定して HDM ライブラリをビルド, インストールしているものとして示します.

2.2.1 C++

HDM ライブラリを利用している C++ のプログラム main.C を icpc でコンパイルする場合は, 次のようにコンパイル, リンクします.

```
$ icpc -o prog main.cpp ‘ /usr/local/HDMlib/bin/hdm-config –cflags‘ \
‘ /usr/local/HDMlib/bin/hdm-config –libs‘
```

第3章　API利用方法

この章では、HDMlib の API の利用方法について説明します.

3.1 FileIO 用ファイル構成

ファイル入出力関連のソースコードは，すべて以下に配置されています.
examples/FileIO/[inlcude—src]

各ファイルの概要を説明します.

ファイル	機能
BCMFileCommon.h	構造体定義等
BCMFileLoader.h	BCM ファイル読込クラス
BCMFileSaver.h	BCM ファイル出力クラス
BCMRLE.h	ランレングス圧縮/展開ライブラリ
BCMTypes.h	データ型宣言群
BitVoxel.h	ビットボクセル圧縮/展開ライブラリ
DirUtil.h	ディレクトリ操作ユーティリティ
ErrorUtil.h	エラー処理関数群
FileSystemUtil.h	ファイル操作ユーティリティ
IdxBlock.h	インデックスファイル用ブロック情報クラス
IdxStep.h	インデックスファイル用タイムステップ情報
LeafBlockLoader.h	LeafBlock ファイル読込関数群
LeafBlockSaver.h	LeafBlock ファイル出力関数群
Logger.h	ログ出力機能
PartitionMapper.h	MxN データロード用データマッピングユーティリティ

3.2 ファイル保存機能使用方法

ファイル保存機能は以下のように使用します.

1. BCMFileIO::BCMFileSaver のインスタンス生成
2. 出力するデータ (ブロック) の情報を登録
3. インデックスファイル/Octree ファイル出力
4. リーフブロックファイル出力

=== 例 ===

```
// BCMFileSaver のインスタンス生成
BCMFileIO::BCMFileSaver saver(
        globalOrigin,               // 計算空間全体の起点座標
        globalRegion,               // 計算空間全体の領域サイズ
        octree,                     // BCMOctree のポインタ
        "out"                       // ファイル出力先ディレクトリ (省略した場合"./")
```

```
);

// 出力するタイムステップ情報を設定
BCMFileIO::IdxStep step(0, 100, 5); // 0 から 100 まで 5 刻みのリストを設定

// 出力するデータの情報を登録 (CellID)
saver.RegisterCellIDInformation(
        dcid_cid,                   // データクラス ID
        bitWidth,                   // セルのビット幅
        "CellID",                   // 系の名称
        "cid",                      // ファイル名の Prefix
        "lb",                       // 拡張子
        "cid",                      // 出力先ディレクトリ ※ 1
        true                        // Gather フラグ (省略した場合 true)
);

// 出力するデータの情報を登録 (Data) ※ 2
saver.RegisterDataInformation(
        &dcid_tmp,                  // データクラス ID (ポインタ) ※ 3
        BCMFileIO::LB_SCALAR, // データの種別
        BCMFioeIO::LB_FLOAT64,// データの型
        virtualCell,                // 仮想セルサイズ
        "Temperature",              // 系の名称
        "tmp",                      // ファイル名の Prefix
        "lb",                       // 拡張子
        step,                       // タイムステップリスト
        "tmp",                      // 出力先ディレクトリ ※ 1
        true                        // タイムステップごとのサブディレクトリフラグ ※ 4
);
```

※ 1 出力先ディレクトリは、コンストラクタで指定したディレクトリからの相対パスとなります.

※ 2 複数の Data を登録する場合、同様に RegisterDataInformation を複数呼び出します.

※ 3 複数コンポーネント (ベクトル・テンソル等) の場合、各要素をそれぞれ Scalar3D<T>として保持するため、 データを登録する際には、要素毎のデータクラス ID を配列にし、先頭アドレスを渡してください. その場合、配列のサイズは、データの種別で指定する要素数と同じにしてください.

※ 4 タイムステップごとのサブディレクトリフラグを true にした場合、出力先として指定したディレクトリの下に、タイムステップ毎のディレクトリが作成され、リーフブロックファイルは群はその下に出力されます.

```
// インデックスファイルと Octree ファイルを出力
// 登録されたデータ (CellID/Data) の情報と Octree をファイルに保存
saver.Save();
```

```
// リーフブロックファイルを出力
saver.SaveLeafBlock("CellID"); // 出力するデータの系の名称
saver.SaveLeafBlock("Temprature", 0); // 出力するデータの系の名称, タイムステップ番号
```

3.3 ファイル読込機能使用方法

ファイル読込機能は以下のように使用します.

1. BCMFileIO::BCMFileLoader のインスタンス生成とメタ情報読込
2. 複数のインデックスファイルを読む場合、追加のインデックスファイルを読み込み
3. リーフブロックファイル読込

※ メタ情報
- インデックスファイルの情報
- Octree

=== 例 ===

```
// BCMFileLoader のインスタンス生成とメタ情報
BCMFileIO::BCMFileLoader loader(
        "cellId.bcm",                   // インデックスファイルのパス
        bcsetter                        // 境界条件設定ファンクタ
);

// 追加のインデックスファイルの読み込み ※ 1
loader.LoadAdditionalIndex("data.bcm");

// リーフブロックファイル読込 (CellID)
loader.LoadLeafBlock(
        &cid_dcid,                      // データクラス ID ※ 2
        "CellID",                       // 系の名称
        vc                              // 仮想セルサイズ
); // return はデータクラス ID

// リーフブロックファイル読込 (Data)
loader.LoadLeafBlock(i
        &tmp_dcid,                      // データクラス ID ※ 2
        "Temperature",                  // 系の名称
        vc,                             // 仮想セルサイズ
```

```
        0                                   // タイムステップ
); // return はデータクラス ID
```

※ 2 LoadLeafBlock を実行すると、系の名称に対応するブロックが生成されます．生成したブロックのデータクラス ID は、LoadLeafBlock の第一引数として出力されます．また、すでに作成されている場合は、そのデータクラス ID を出力します．系の名称に対応するデータが複数コンポーネントの場合、各要素に対応したデータクラス ID を出力するため、第一引数に渡すバッファは配列として、事前に確保してください．また、データを読み込まずブロックの生成だけを実行したい場合、loader.CreateLeafBlock(&dcid, name, vc) を呼び出してください．

3.4 サンプルプログラム

BCMTools FileIO を使ったサンプルを 2 種類作成しました．

・SampleCreator
→ ParallelMeshGeneration のメッシュ生成機能ルーチンを使い、メッシュ生成を行い、結果 (CellID/Scalar/Vecotr) をファイルとして出力

・SampleLoader
→ SampleCreator で作成した BCM ファイル群を読み込み、結果をファイルに出力