

文部科学省次世代IT基盤構築のための研究開発

「イノベーション基盤シミュレーションソフトウェアの研究開発」

CISS フリーソフトウェア

FrontISTR

HEC_MW Ver.4.2

ユーザマニュアル

本ソフトウェアは文部科学省次世代IT基盤構築のための研究開発「イノベーション基盤シミュレーションソフトウェアの研究開発」プロジェクトによる成果物です。本ソフトウェアを無償でご使用になる場合「CISSフリーソフトウェア使用許諾条件」をご了承頂くことが前提となります。営利目的の場合には別途契約の締結が必要です。これらの契約で明示されていない事項に関して、或いは、これらの契約が存在しない状況においては、本ソフトウェアは著作権法など、関係法令により、保護されています。

お問い合わせ先

(契約窓口)

(財)生産技術研究奨励会

〒153-8505 東京都目黒区駒場4-6-1

(ソフトウェア管理元) 東京大学生産技術研究所 革新的シミュレーション研究センター

〒153-8505 東京都目黒区駒場4-6-1

Fax : 03-5452-6662

目次

1.	はじめに.....	1
2.	インストール方法.....	1
2.1	インストールできるソフトウェア.....	1
2.2	インストールに必要なソフトウェアおよび動作確認を行っている環境.....	1
2.3	ファイルの解凍.....	2
2.4	コンパイルおよび実行の準備.....	2
2.5	サンプルプログラムの実行.....	4
2.5.1	インストールディレクトリから実行する場合.....	4
2.5.2	ディレクトリ example から実行する場合.....	4
2.6	再コンパイルをする場合.....	5
2.7	Windows で解析コードをビルドする.....	5
3.	システム概要.....	12
3.1	概要.....	12
3.2	システムの構成.....	12
4.	HEC-MW の機能.....	14
4.1	データ構造.....	14
4.2	領域分割.....	18
4.3	MPI.....	19
4.4	線形代数ソルバー.....	20
4.4.1	CG.....	20
4.4.2	マルチグリッド CG.....	21
4.4.3	MPC 前処理付き反復法.....	22
4.5	ベクトル管理.....	24
4.5.1	自由度混在ベクトルの管理.....	24
4.5.2	基礎演算機能.....	24
4.6	マトリックス管理.....	25
4.6.1	自由度混在型マトリックスの管理.....	25
4.6.2	非ゼロプロファイルの生成.....	25
4.6.3	要素剛性行列の足し込み.....	26
4.6.4	基礎演算機能.....	26
4.7	要素ライブラリ.....	26
4.7.1	Hexa 要素.....	26
4.7.2	Tetra 要素.....	27
4.7.3	Prism 要素.....	28

4.7.4	Quad 要素 エンティティ番号	30
4.7.5	Triangle 要素エンティティ番号	31
4.7.6	Beam 要素エンティティ番号	31
4.7.7	入力ファイル	32
4.7.8	出力ファイル	34
5.	HEC_MW を利用したプログラムの記述方法	35
5.1	HECMW を利用した線形静解析のサンプルプログラム	35
5.1.1	インクルード・ヘッダー	35
5.1.2	初期化处理	36
5.1.3	メッシュファイルの読み込み、モデルデータの構築	36
5.1.4	材料定数の定義	36
5.1.5	方程式の確保	37
5.1.6	FEM メッシュデータにアクセスするためのループ処理部分	37
5.1.7	形状関数処理部分	38
5.1.8	要素剛性行列の生成	40
5.1.9	要素剛性行列の足しこみ	41
5.1.10	境界条件の設定	41
5.1.11	線形ソルバーの実行	43
5.1.12	計算結果の出力	44
5.1.13	HECMW 終了処理	44
5.2	使用例 1 ; 単一メッシュパーツ	45
5.2.1	計算条件	45
5.2.2	計算結果	45
5.3	使用例 2 ; 2 つのメッシュパーツ	47
5.3.1	計算条件	47
5.3.2	実行結果	48
5.4	使用例 3 ; マルチグリッド	49
5.4.1	計算条件	49
5.4.2	計算結果	49
6.	API	51
6.1	初期化・終了処理	51
6.2	バナー表示	51
6.3	File 関連	52
6.4	線形代数方程式管理	54
6.5	線形代数方程式の行列・ベクトル	55
6.6	線形代数ソルバー	58
6.7	階層メッシュ構造の構築	58
6.8	メッシュデータ・アクセス	59

6.9	要素タイプ	62
6.10	形状関数	65
6.11	境界条件	73
6.12	MPI ラップ関数&通信節点	80
6.13	ロガー	83
7.	HECMW 中間メッシュ・ファイル形式	85
7.1	ファイル形式の定義	85
7.1.1	アセンブルモデル	85
7.1.2	ノード	85
7.1.3	エレメント	85
7.1.4	通信 Mesh	86
7.1.5	通信ノード	86
7.1.6	通信面	86
7.1.7	接合メッシュ(ContactMesh)	87
7.1.8	境界条件	87
7.2	要素エンティティ番号	91
7.2.1	Hexa 要素 エンティティ番号	91
7.2.2	Tetra 要素 エンティティ番号	92
7.2.3	Prism 要素 エンティティ番号	93
7.2.4	Quad 要素 エンティティ番号	94
7.2.5	Triangle 要素エンティティ番号	94
7.2.6	Beam 要素エンティティ番号	95
7.3	ユーティリティ	95
7.3.1	ロガー機能	95
8.	ユーティリティ・プログラム	96
8.1	ファイル形式変換ツール	96
8.1.1	コンパイル手順	96
8.1.2	利用方法	96
8.2	領域分割ツール	97
8.2.1	コンパイル手順	97
8.2.2	利用方法	97
8.2.3	実行例	97
例 2	97
	制限事項	99
図 2.3.1	ファイルの解凍方法	2
図 2.4.1	コンパイルおよび実行の準備方法	2
図 2.4.2	Makefile.in の内容	3

図 2.5.1	ライブラリおよび実行体の作成方法	4
図 2.6.1	サンプルプログラム実行方法 (その 1)	4
図 2.6.2	サンプルプログラム実行方法 (その 2)	4
図 2.7.1	再コンパイルの方法	5
図 3.2.1	本システムの全体構成	13
図 4.1.2	データ構造	15
図 4.1.3	階層型データ構造	16
図 4.1.4	アセンブリーデータ構造の接合面	16
図 4.1.5	八分木とマスタースレーブ面	17
図 4.1.6	Bounding Box による絞り込み	17
図 4.2.1	領域分割ファイル	19
図 4.2.2	通信テーブル	19
図 4.4.1	CG 法アルゴリズムおよび前処理つき CG 法アルゴリズム	21
図 4.4.2	MGCG 法のアルゴリズム	22
図 4.6.1	CRS Format の例	25
図 5.2.1	階層データを用いたデータ形状	45
図 5.2.2	階層データを利用した解析結果	46
図 5.2.3	階層データを利用したケースの収束状況	46
図 5.2.4	レベル 5 の格子	47
図 5.2.5	レベル 5 の計算結果	47
図 5.3.1	接合面で接続された 2 つのメッシュパーツ	48
図 5.3.21	接合面で接続された 2 つのメッシュパーツの計算結果	48
図 5.4.11	MGCG のテストで利用した各レベルの形状	49
図 5.4.2	MGCG の収束状況	49
図 5.4.3	MGCG の計算結果	50
図 5.4.4	MGCG 収束状況	50
図 7.2.1	Hexa 要素	91
図 7.2.2	Tetra 要素	92
図 7.2.3	Prism 要素	93
図 7.2.4	Quad 要素	94
図 7.2.5	Triangle 要素	94
図 7.2.6	Beam 要素	95
表 2.2.1	インストールに必要なソフトウェアおよび動作確認を行っている環境	1
表 2.3.1	インストールディレクトリ内容	2
表 2.4.1	Makefile.in の COPT に設定可能なオプション	4
表 2.6.1	テストデータの内容	5
表 4.3.1	ラップする MPI の機能は下記の関数群	19

表 4.5.1	ベクトルに関する基礎演算機能.....	24
表 4.6.1	マトリクスに関する基礎演算機能.....	26
表 4.7.1	HECMW が管理するファイル	32
表 4.7.2	FrontISTR タイプのファイル名管理規則.....	33
表 6.13.1	ログレベル	83
表 7.3.1	ログレベルによる動作.....	95

1. はじめに

大規模並列有限要素基盤 HEC-MW の公開版ユーザズマニュアルである。

2. インストール方法

2.1 インストールできるソフトウェア

本ソフトウェアは、有限要素法（FEM）による大規模シミュレーションコードを開発する際、共通に利用される機能を提供することにより、解析コード開発者がアプリケーションソフトの開発に専念できるようにするためのソフトウェアである。本ソフトウェアをインストールすると、本ソフトウェアの機能が集約されたライブラリが作成される。また、サンプルプログラム、ファイル形式変換ツール、メッシュ領域分割プログラムも作成される。

本ライブラリをユーザの作成するプログラムにリンクすることで本ソフトウェアの機能を利用することができる。

2.2 インストールに必要なソフトウェアおよび動作確認を行っている環境

本ソフトウェアを正しく動作させるために、計算機上には、OS の他に、次のソフトウェアが必要である。

- C++コンパイラ
- boost
- MPI

これらのソフトウェアについては、表 2.2.1 に示すバージョンで動作確認を行っている。

表 2.2.1 インストールに必要なソフトウェアおよび動作確認を行っている環境

項目	Windows	Linux
OS	Windows XP (32bit), Windows 7 (32 bit モード)	Cent OS 5.3 (64bit), Ubuntu 9.10 (64bit)
C++	Visual C++ 2010 Express	GNU g++ 4.3, Intel Compiler 12.0.3
boost	boost 1.46.1	boost 1.34
MPI	MPICH2 1.3.2p1	MPICH2, OpenMPI 1.4.1

2.3 ファイルの解凍

サンプルプログラムを含めたプログラム一式が `tar + gzip` の形式でひとつのファイルになっている (ファイル名 `hecmw.tar.gz`). これを `tar` コマンドで解凍する。

```
% tar zxvf hecmw.tar.gz
% cd hecmw
% ls
Makefile Makefile.in doc example lib src
```

図 2.3.1 ファイルの解凍方法

解凍されたファイル群は、ソースプログラム (`src`)、ライブラリ (`lib`)、サンプル (`test`)、ドキュメント (`doc`) の 4 つのディレクトリから構成される。ソースプログラムはサブルーチンソースの集まりであり、インストールを実行するとライブラリのディレクトリにライブラリが作成される。テストプログラムは 2 つの並列化レベルに対応したサンプルプログラムが作成されている。`Makefile.in` に機種依存情報が格納されている。

表 2.3.1 インストールディレクトリ内容

ディレクトリ名	内容
<code>lib</code>	インストールディレクトリで <code>make</code> を実行するとライブラリ (<code>libmw3.a</code>)がこのディレクトリ内に作成される。ファイル解凍時は何もない
<code>src</code>	本ソフトウェアのライブラリとなるサブルーチン群のソースコードが格納されている
<code>example</code>	線型弾性の構造解析プログラムおよびそのテストデータが格納されている
<code>doc</code>	本ドキュメントが格納されている
<code>Makefile</code>	ライブラリおよびテストプログラムの実行体をビルドするためのファイル
<code>Makefile.in</code>	計算機環境および並列計算環境の違いによるインストール方法の違いを記述したファイル

2.4 コンパイルおよび実行の準備

以下、Linux の例で、記述する。次のような流れでコンパイルおよび実行準備をする。

```
% vi Makefile.in ← 利用者の環境にしたがって Makefile.in を編集
% make clean
% make ← ./src をコンパイルし ./lib に*.a を作成し ./example のサンプルプログラムをビルド
```

図 2.4.1 コンパイルおよび実行の準備方法

使用する計算機の MPI 環境について調べ、include ファイル、l ライブラリ、および MPI 実行コマンドの所在を記述にし、インストールディレクトリのファイル「Makefile.in」の MPI 関連を編集する。ほとんどの環境では赤字の部分のみの編集で充分である。

```
#
# Where is my home?
#
MW3_HOME= (tar で展開した内容のルートディレクトリ)
MPICH_HOME= (mpi のインストールディレクトリ)

#
# Which compiler to use
#
CPC= $(MPICH_HOME)/bin/mpicxx

#
# What options to be used by the compiler
#
COPT= -DHAVE_MPI (MPI や OpenMP の有無 (後述)、その他コンパイラに渡すオプションを記述)
INC_DIR= $(MW3_HOME)/src

#
# What options to be used by the loader
#
LOPT=
LIB= $(MW3_HOME)/lib/libmw3.a
LLIB= -L$(MW3_HOME)/lib -lmw3

#
# What archiving to use
#
AR = ar rv

#
# What to use for indexing the archive
#
RANLIB = ar -ts
#RANLIB = ranlib
#RANLIB =

#
# Which mpirun to use
#
RUN= $(MPICH_HOME)/bin/mpirun
EXE= $(MW3_HOME)/test/a.out

#
# How to compile c++ codes
#
.SUFFIXES: .cxx .o
.cxx.o:
    $(CPC) -c $(COPT) -I$(INC_DIR) $< -o $@
```

図 2.4.2 Makefile.in の内容

表 2.4.1 に、Makefile.in 中の COPT 変数に設定可能なオプションを示す。これらのオプションは自由に組み合わせて使用可能である。また、最適化パラメータ等の C++コンパイラへの他のオプショ

ンを追加して記述することが可能である。

例：MPI + OpenMP で最適化レベル 2

```
COPT= -DHAVE_MPI -openmp -O3
```

表 2.4.1 Makefile.in の COPT に設定可能なオプション

オプション **説明** `-DHAVE_MPI` 本オプションを指定することで、MPI 版となる `-DINTELMKL` 本オプションを指定することで、IntelMKL の Boost サポートがソルバーで利用される `-openmp` 本オプションを指定することで、CG ソルバがマルチスレッド対応となる

ライブラリおよびサンプルプログラムのビルド

インストールディレクトリで make を実行すると、各ディレクトリでコンパイルが実行され、プラットフォームのライブラリ、およびテストプログラムの実行体が作成される。

```
% make ← ./src をコンパイルし./lib に*.a を作成し ./example のサンプルプログラムをビルド
```

図 2.4.3 ライブラリおよび実行体の作成方法

2.5 サンプルプログラムの実行

2.5.1 インストールディレクトリから実行する場合

以下では、代表的なプログラムのチェック方法を示す。いずれの例題も数十から数千自由度の問題であり、テスト例題の実行時間は、数秒から数分程度の例題である。本ソフトウェアに含まれる全テスト例題の実行が行われる。

動作方法はそれぞれの Makefile 中の check の項目に記述されている。正常に起動しない場合には、MPI コマンドのコマンドパスおよび Makefile.in で指定したライブラリの位置を確認すること。

```
% make check
```

図 2.5.1 サンプルプログラム実行方法(その 1)

2.5.2 ディレクトリ example から実行する場合

次のような方法で、本資料に記載された例題を実行することができる。

```
% cd example
% make test1
```

図 2.5.2 サンプルプログラム実行方法(その 2)

表 2.5.1 テストデータの内容

コマンド	内容	メッシュファイル
% make test1	MPC を用いた複数パーツ 1 プロセスでの実行	testPartsB.0.msn
% make test2	片持ち梁（4要素）2 プロセスでの実行	beamPara.0.msh beamPara.1.msh

2.6 再コンパイルをする場合

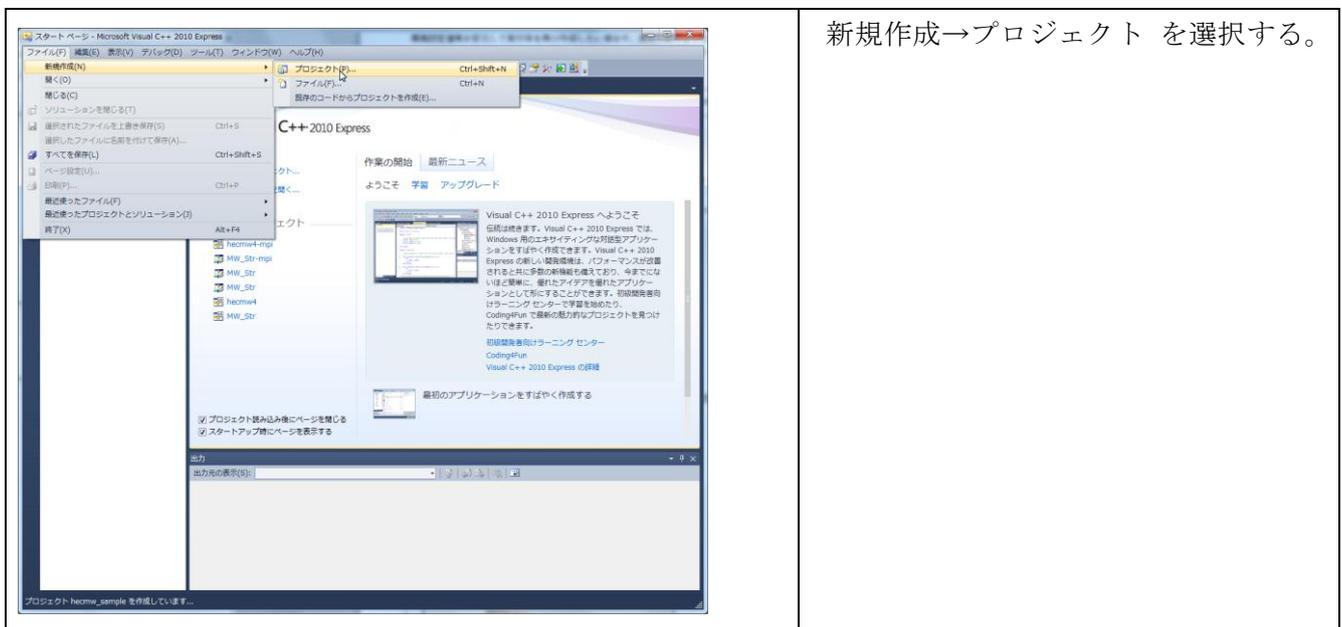
環境設定値等が変化して実行体を再び作成したい場合や、異常な動作をするためにコンパイルし直したい場合には、次の操作を行う。必要に応じて現状の Makefile.in のバックアップをとっておくことを推奨する。

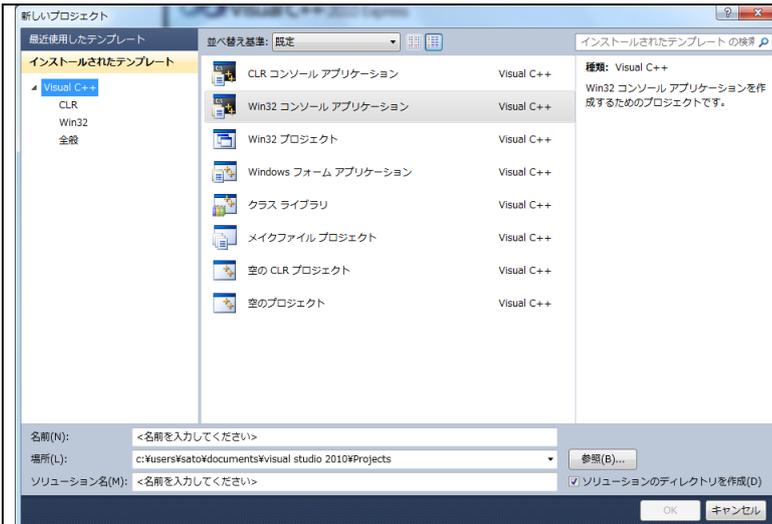
```
% vi Makefile.in ← 利用者の環境にしたがって Makefile.in を編集
% make clean ← 既にコンパイルしたオブジェクト、ライブラリ、プログラムを削除
% make
```

図 2.6.1 再コンパイルの方法

2.7 Windows で解析コードをビルドする

Visual C++ Express で HEC_MW ライブラリをリンクさせて解析コードをビルドする手順を説明する。

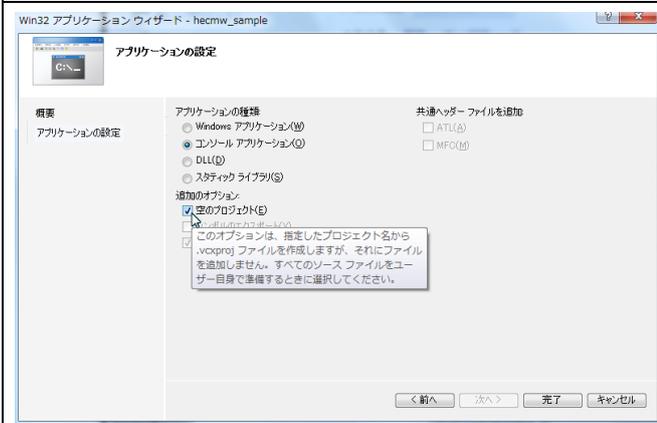




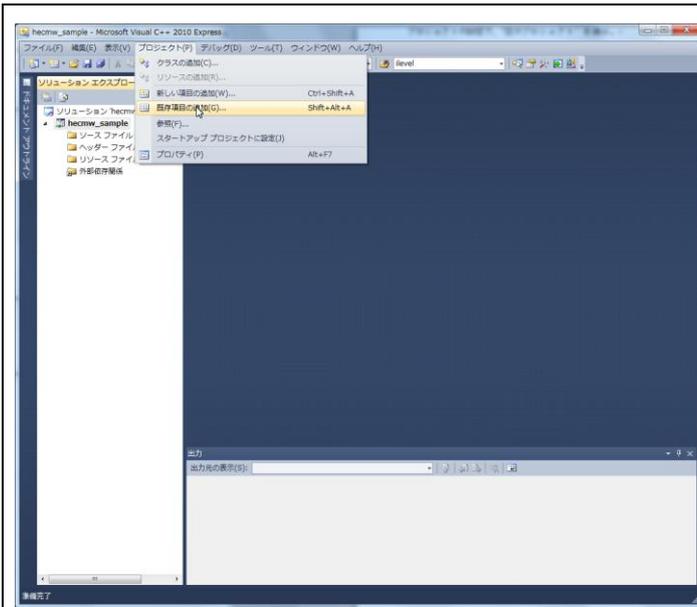
Win32 コンソールアプリケーションを選択し、プロジェクト名を入力する。



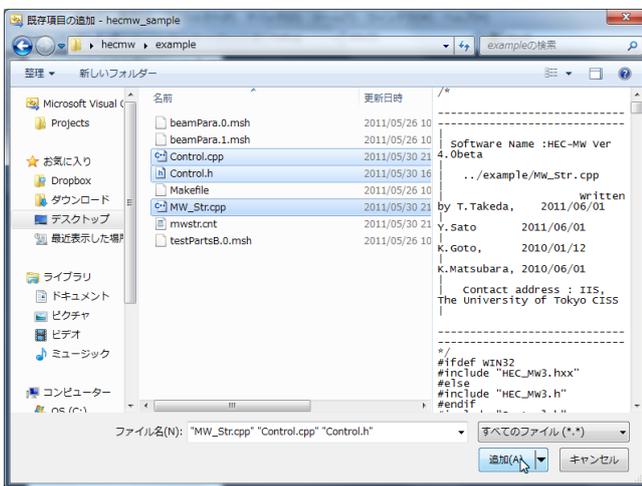
ウィザードが始まるので次へ。



アプリケーションの種類は“コンソールアプリケーション”、追加のオプションで“空のプロジェクト”を選択し、完了を押す。

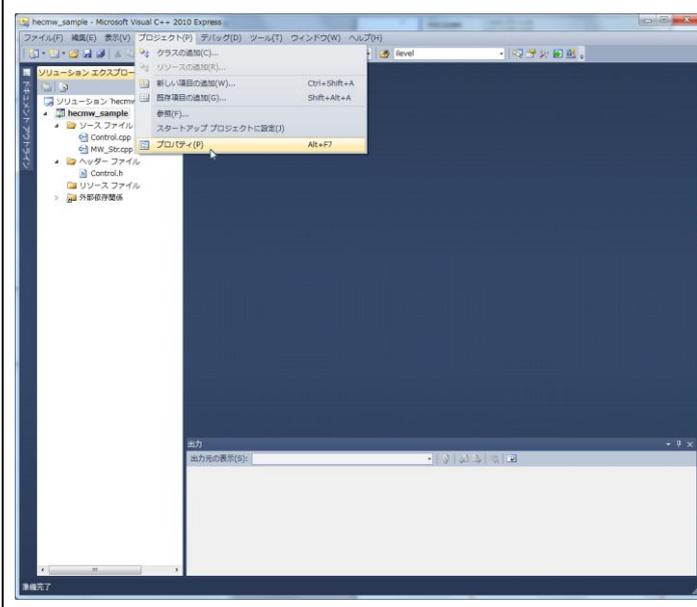


プロジェクト→既存項目の追加 を選ぶ。

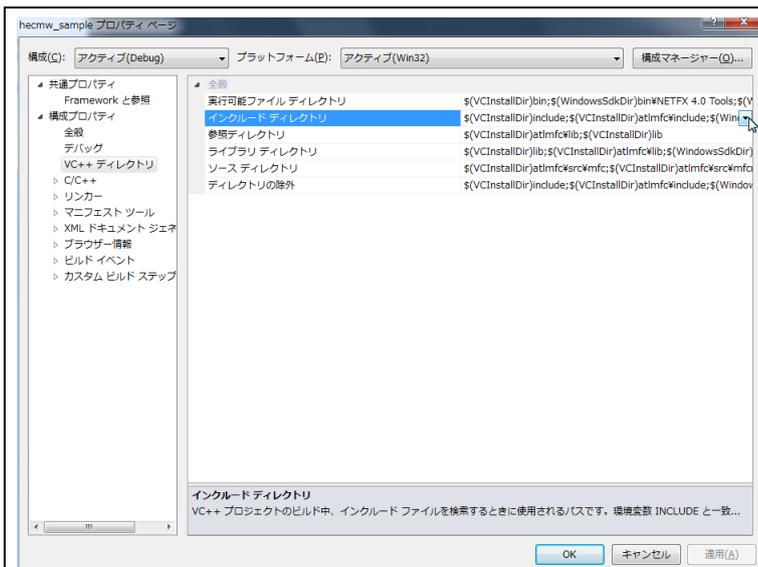


ダイアログより、ダウンロードしたHECMW を解凍したフォルダを開き、example ディレクトリ内の

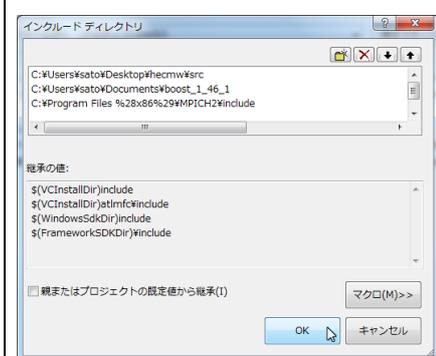
- Control.cpp
 - Control.h
 - MW_Str.cpp
- をプロジェクトに追加する。



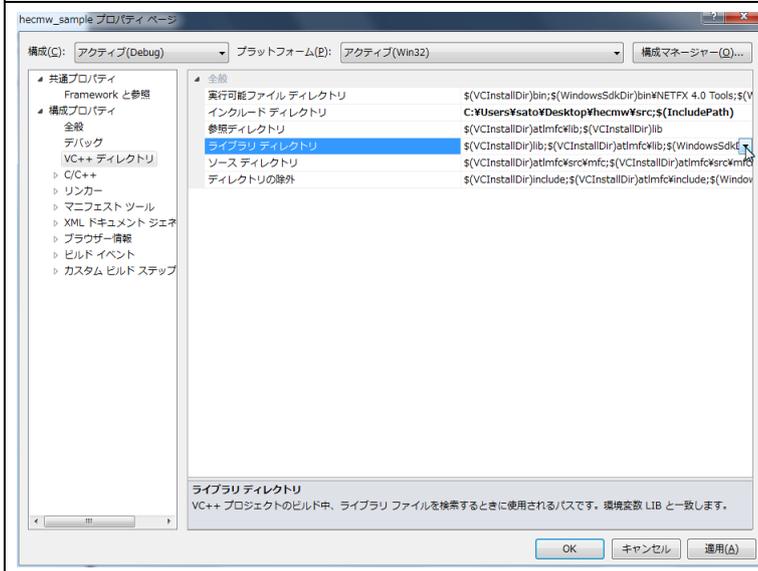
プロジェクト→プロパティ を開く。



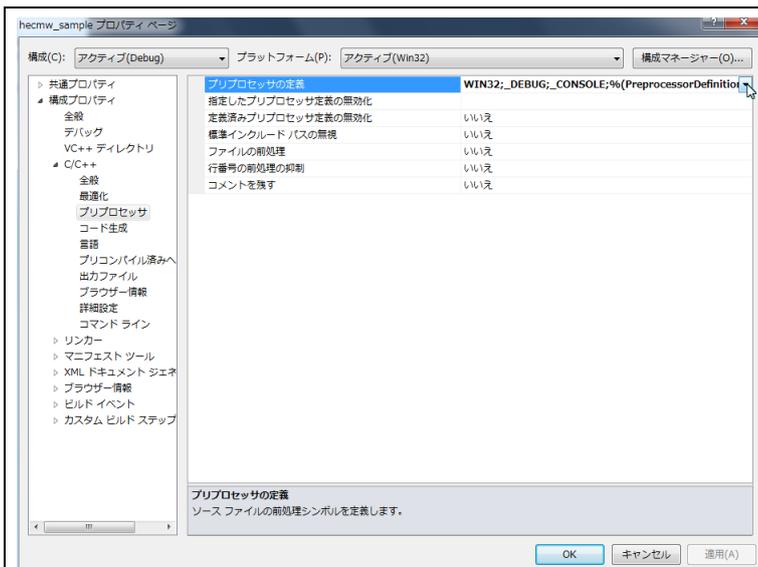
左側メニューより“構成プロパティ”の“VC++ディレクトリ”を選び、“インクルードディレクトリ”を編集する。



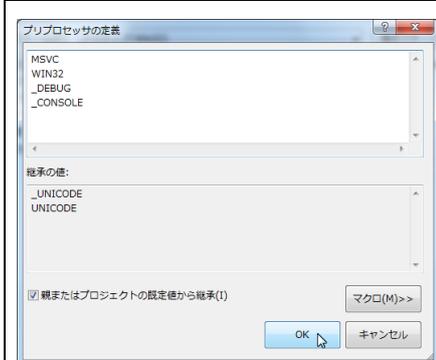
インクルードディレクトリに HECMW を解凍したディレクトリ内の src ディレクトリを追加する。
また、システムのインクルードパスに追加していない場合には Boost と MPICH2 のインクルードディレクトリも追加する。



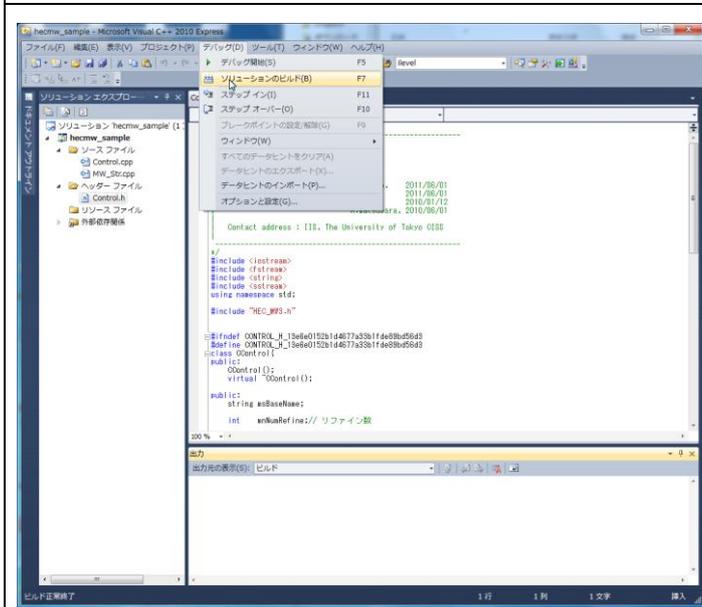
続いて、左側メニューの“VC++ディレクトリ”内の“ライブラリディレクトリ”を編集する。



左側メニューの“構成プロパティ” → “C/C++” → “プリプロセッサ”内の“プリプロセッサの定義”を編集する。



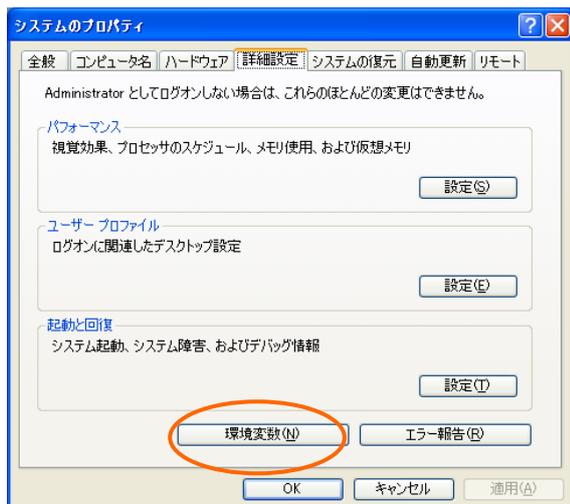
プリプロセッサの定義に“MSVC”を追加する。



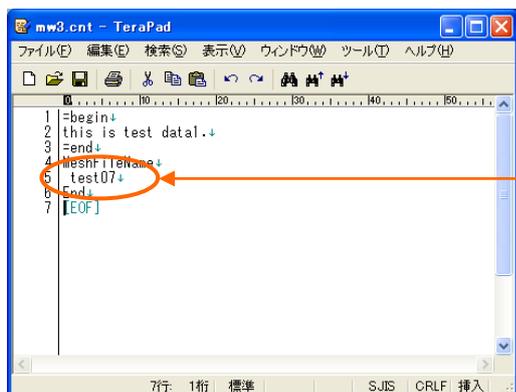
プロジェクトのプロパティを閉じ、“デバッグ” → “ソリューションのビルド” または F7 キーを押してビルドする。ビルドしたプロダクトはプロジェクトフォルダ内の Debug フォルダに生成される。

ビルドした解析コードを実行する

Windows の環境変数の PATH に **MPICH2\bin** と **hecmw4-mpi.dll**、**hecmw4-serial.dll** のディレクトリを記述して、パスを通しておく。 *シングル版を使う場合は MPICH2 へのパスを通す必要はない。



テキストエディタを使って **mwstr.cnt** の **MeshFileName** ブロックに使用するファイル名を記述する。 *テキストエディタはノートパッドでもワードパッドでも何でも良い。



入力ファイル名

コマンドプロンプトを開き、実行コマンドを入力する。

a. 並列版の場合のコマンド入力

```
C:\¥..... > mpiexec -n 2 mw_str mwstr.cnt
User credentials needed to launch processes:
account (domain¥user) [SS141¥FSTR]: △△△△
password: *****
```

PC のユーザー・アカウント(ユーザー名)

パスワード

b. シングル版の場合のコマンド入力

```
C:\¥ ..... > mw_strs mwstr.cnt
```

3. システム概要

3.1 概要

本システムは、解析コード開発に利用するライブラリ・プログラムである。

有限要素法（FEM）による大規模シミュレーションコードを開発する際、共通に利用される機能を提供することにより、解析コード開発者がアプリケーションソフトの開発に専念できるようにする。

本ライブラリの API（Application Programming Interface）を通じて、並列線形代数ソルバー、ファイル入出力、階層型アセンブリー構造データ、行列・ベクトル、要素ライブラリ、境界条件データなどの機能が提供される。

本ライブラリを使用して開発された FEM プログラムは自動的に階層化、並列化され、不連続なメッシュデータを接合（アセンブル構造）することができる。

3.2 システムの構成

ダウンロードされた、本システムは以下の各部分から構成される。

- a. 「HEC_MW ライブラリ」
 - ・ 階層型アセンブリーデータ構造の管理
 - ・ 行列およびベクトル データの管理
 - ・ 要素ライブラリ
 - ・ 境界条件データ管理
 - ・ 並列線形ソルバー
 - ・ ファイル入出力
 - ・ Fortran/C & C++言語用 アプリケーションプログラミングインターフェイス(API)
- b. 「並列メッシュ領域分割用ユーティリティソフトウェア」
- c. 例題プログラム

図 3.2.1 に本システムの全体構成を示す。図中で、「並列メッシュ領域分割用ユーティリティソフトウェア」は、並列有限要素法の計算に使用する、大規模非構造メッシュの領域分割を効率的に実施するためのユーティリティを示す。「例題プログラム」はこれらの「HEC-MW ライブラリ」の利用方法を示すために、HEC-MW を使用して開発される並列有限要素法によるプログラムを示す。

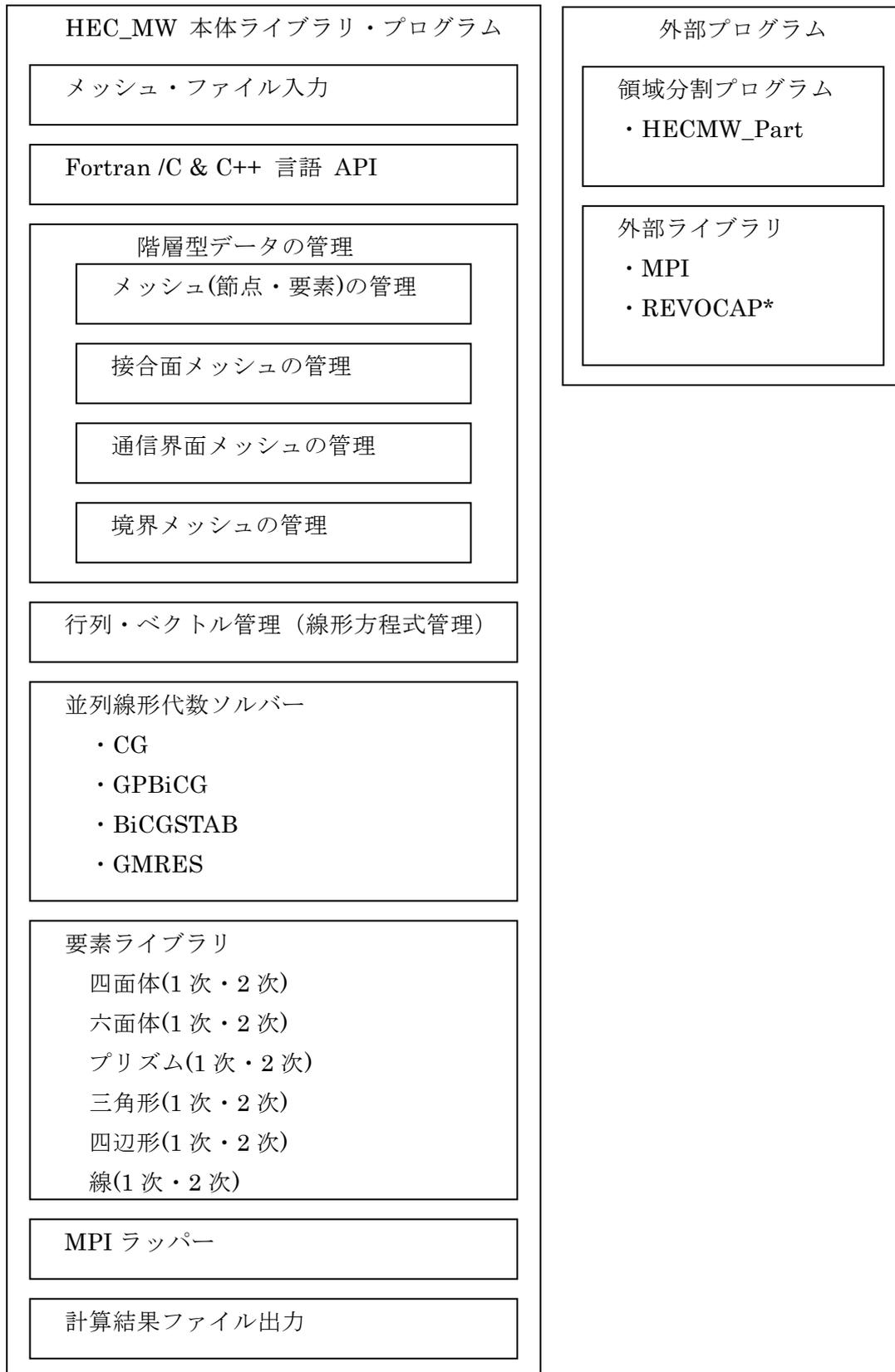


図 3.2.1 本システムの全体構成

4. HEC-MW の機能

4.1 データ構造

HEC_MW は、階層型アセンブリデータ構造を構築できる。

アセンブリデータ構造とは、個別にメッシュ生成したデータを多点拘束により接合していくデータ構造であり、階層型データ構造とは、入力されたメッシュデータを再分割することで幾何学的マルチグリッドソルバーに対応させたデータ構造のことである。

アセンブリデータ構造を生成するには、各アセンブリパーツの接合面を、それぞれ要素面グループとして定義しておき、それらのペアの一方をマスター面、もう一方をスレーブ面とし、スレーブ面上の各節点の自由度をマスター面上の対応する要素面に固定することで実現している。

上記の関係式は、HEC_MW 内部にて自動生成され、多点拘束(MPC)条件として線形代数ソルバーに組み込まれることによりアセンブリ部品を結合している。

階層構造は、アセンブリパーツを表すメッシュデータの各要素を再分割することで実現している。

再分割のために新たに節点を生成追加し、その節点データをもとに要素を分割し、上位階層の要素として新たに要素生成を行っている。

二つのメッシュパーツからなるアセンブリデータ構造を模式的に書くと、図 4.1.1 になる。

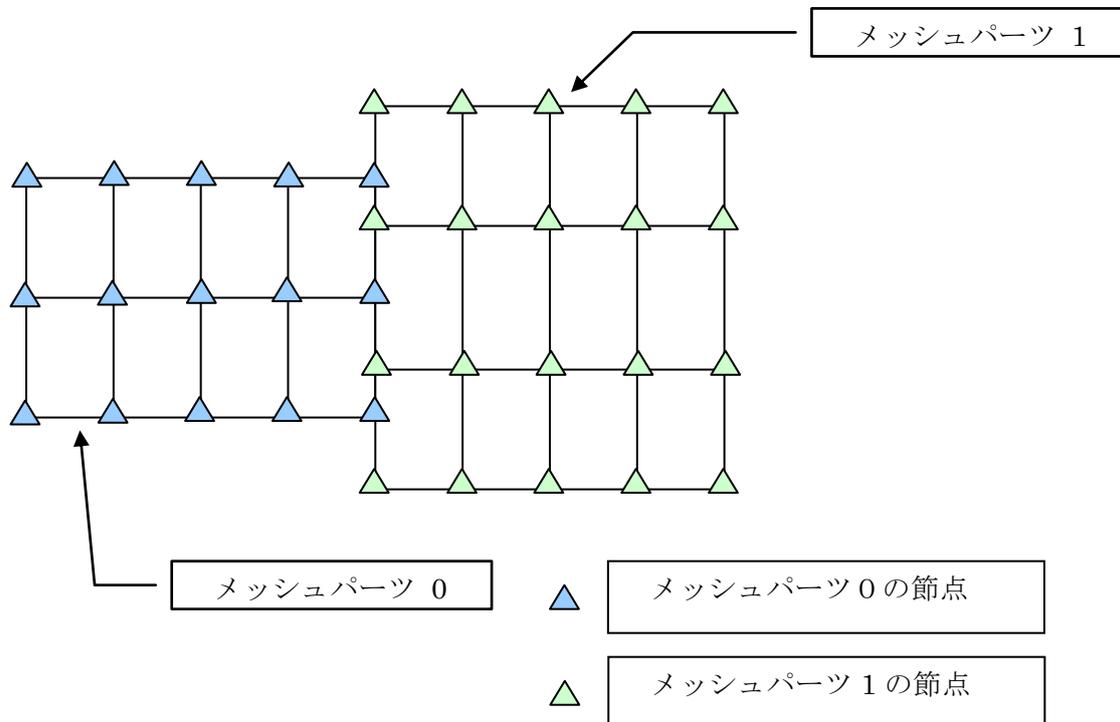


図 4.1.1 アセンブリデータ構造の模式図

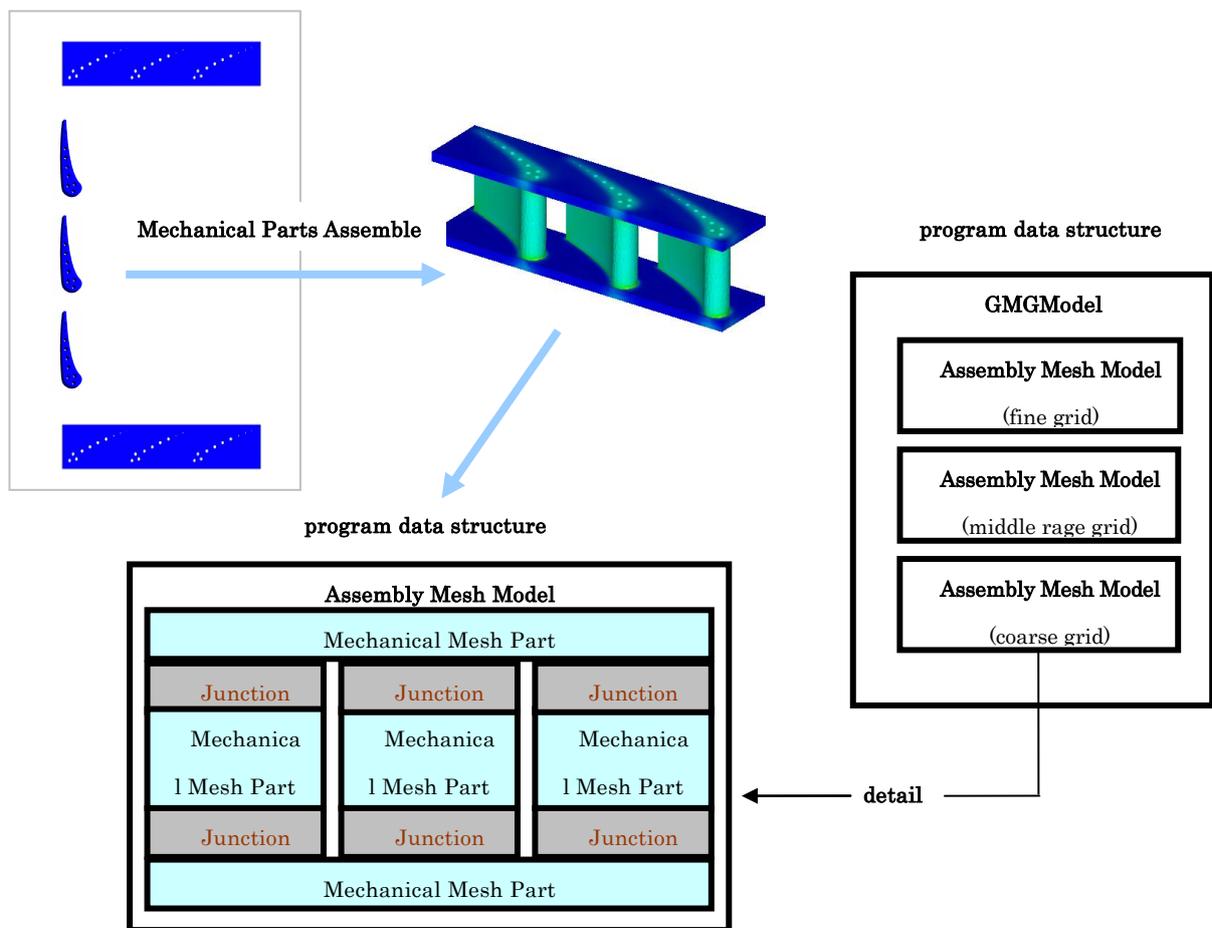


図 4.1.2 データ構造

階層型データ構造について更に記すと、ソリッド要素については全ての要素を Hexa 要素になるように再分割していく、これは FEM の解析において Hexa 要素の研究が進んでおり、現象によっては Hexa 要素でしか計算できない分野も存在することから、解析モジュール開発の適用範囲を広げることがを目的とした処理である。

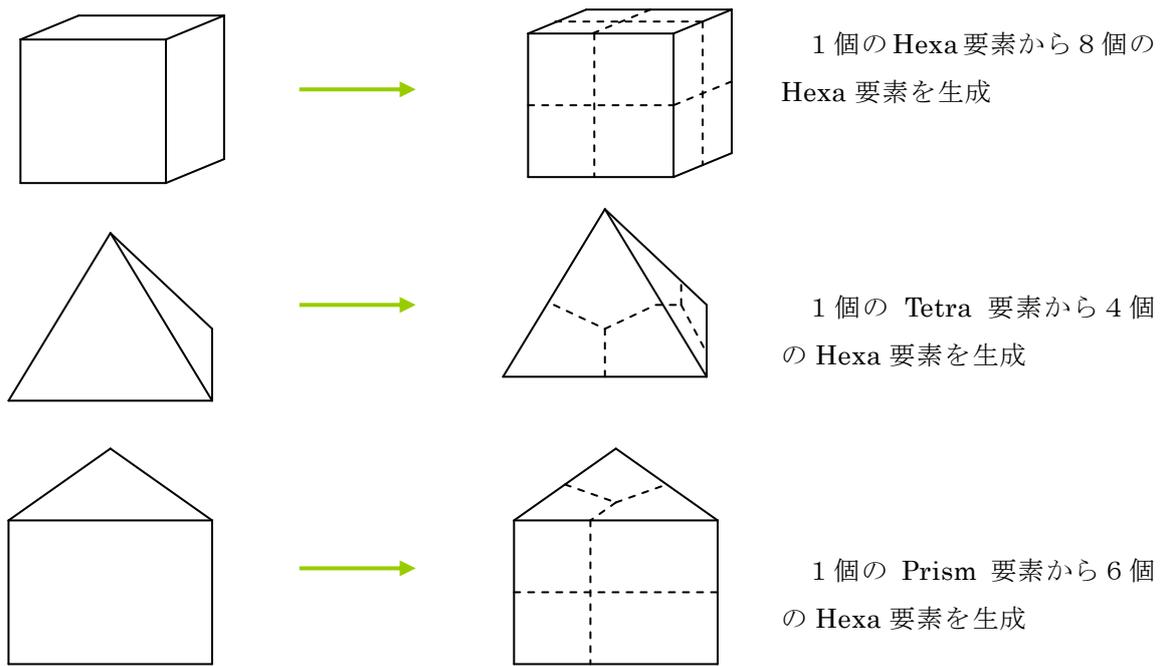


図 4.1.3 階層型データ構造

アセンブリーデータ構造の内部的な扱いについて、更に記しておく。

アセンブリーデータを構築するためには、接合面の“検索”とマスター・スレーブの“関係式”を生成する必要があり、検索について説明する。

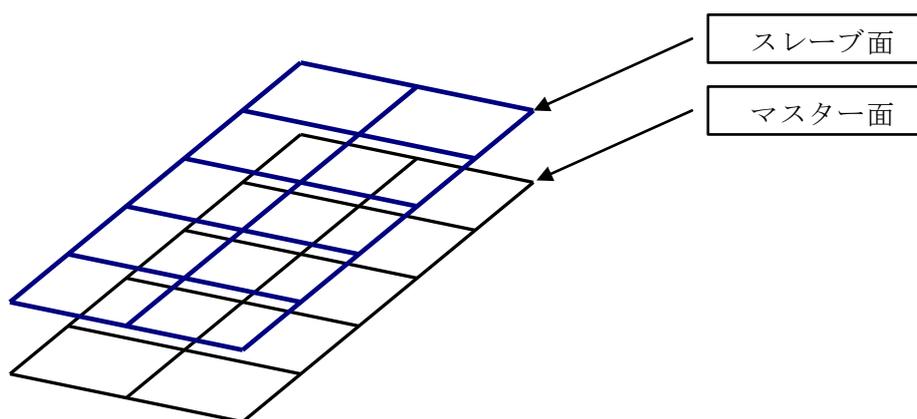


図 4.1.4 アセンブリーデータ構造の接合面

上図のようにマスター・スレーブ面が3次元空間に存在するとして、任意のマスター面に拘束されるスレーブ点をスレーブ面全体から検索する必要がある。この検索を高速に行うために、八分木を用いて対象スレーブ点の絞り込みを行う。

八分木は、マスター・スレーブ面全体（接合メッシュ）を覆うように定義し、接合メッシュの節点数の数に応じて八分木分割数レベルを一段ずつあげていき、マスター面と同一空間に存在するスレーブ点を検索対象としてリストアップする。

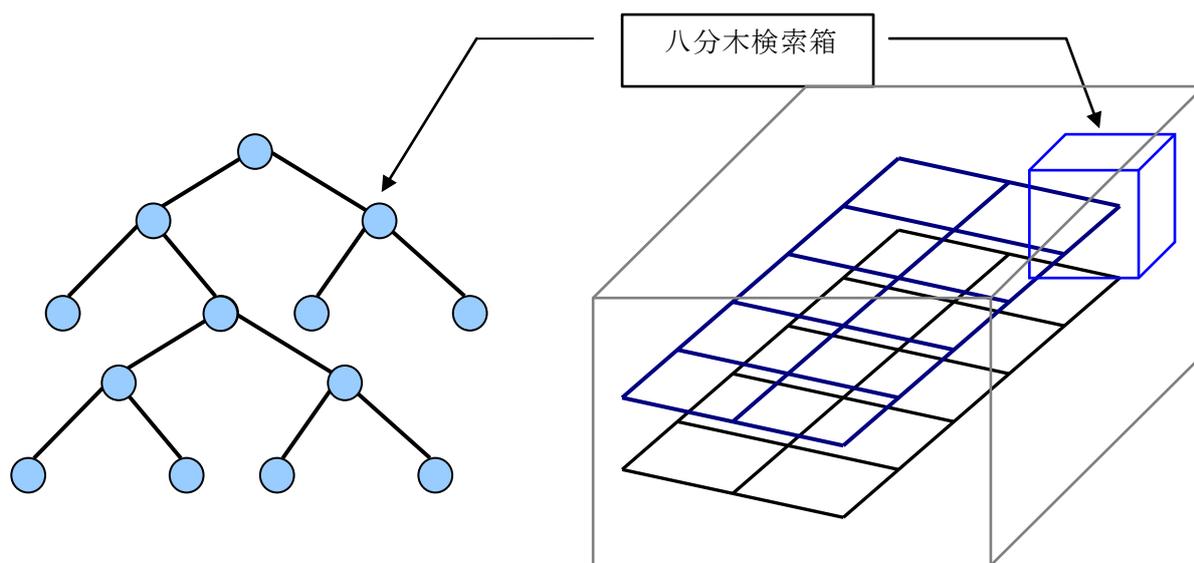


図 4.1.5 八分木とマスター・スレーブ面

八分木により絞り込まれたスレーブ点を対象として、マスター面に平行な局所座標を持つ厚さ方向が薄い Bounding Box を定義し、更に絞り込みを行う。

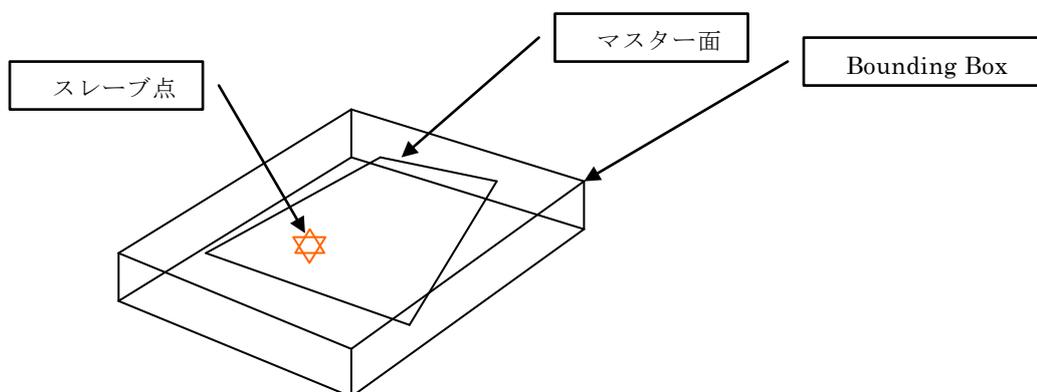


図 4.1.6 Bounding Box による絞り込み

Bounding Box 内に存在するスレーブ点をリストアップしておき、マスター面の平面内に存在するとして、面の内外判定を行い、面内に存在するスレーブ点をマスター面に所属するスレーブ点として登録して検索を終了する。

検索の結果登録されたスレーブ点は、マスター面の各頂点への線形補間係数を幾何学的関係から計算し、MPC ソルバーに用いる拘束式の係数を決定する。

境界条件については、境界値を付与する範囲を幾何学的属性からグループ化して階層データとして管理する。幾何学的属性とは、面、辺、体積、点の 4 種類で、このうち点については階層化はなされない。各グループは、幾何学的属性ごとに境界条件 ID 番号と名前が付与され、領域分割によって分割された場合でも、階層化された場合であっても、境界 ID 番号と名前は維持される。

また、HECMW は、任意の自由度の線型方程式を複数個管理するようになっている。例えば、図 4.1.7 のように剛性方程式と熱伝導方程式の二つの方程式が解かれる場合である。

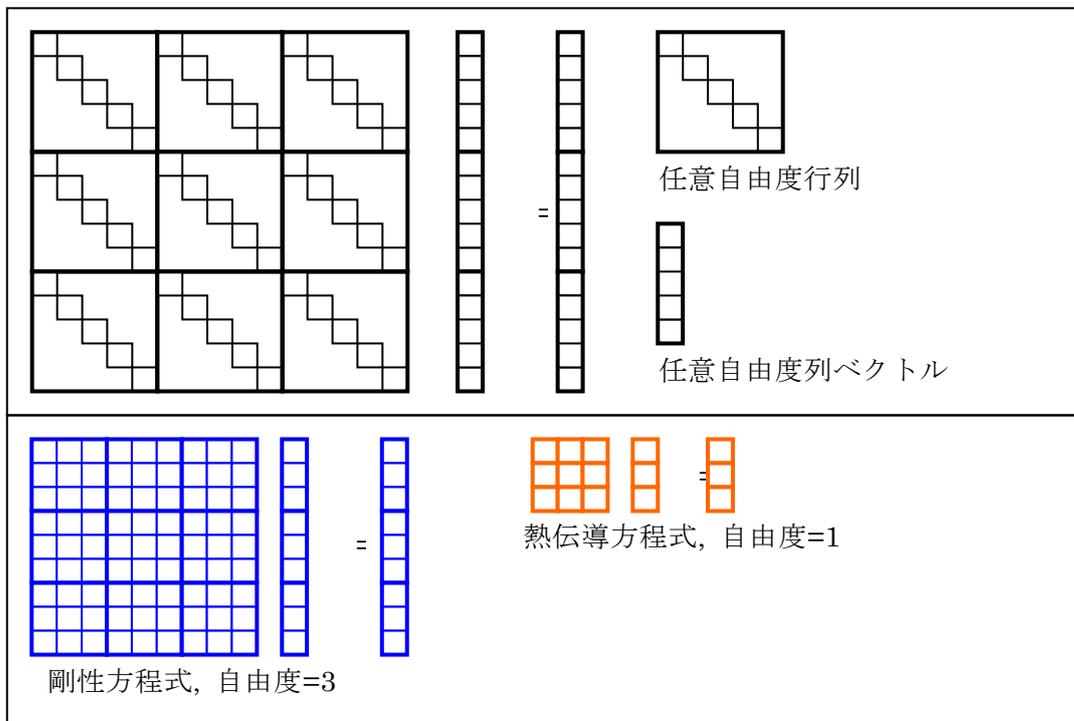


図 4.1.7 複数の線型方程式の例

4.2 領域分割

並列計算のための領域分割機能を提供する。

領域分割は、並列プロセス数に合わせてメッシュを分割してプロセス単位にメッシュを提供する機能である。領域分割プログラムは、HECMW 書式のファイルを読み込み、分割された HECMW ファイルを出力する。

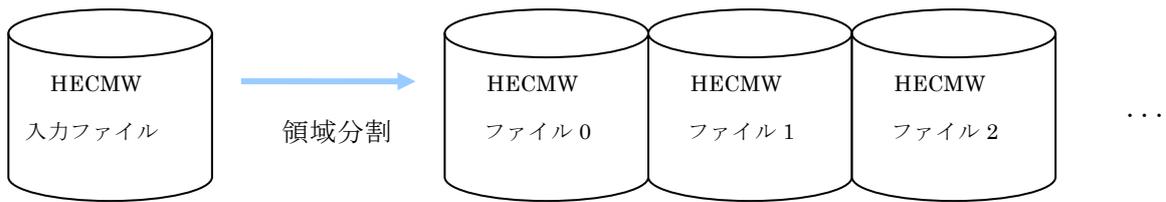


図 4.2.1 領域分割ファイル

領域分割は、オーバーラップ領域は存在せず輸出節点と輸入節点は同一であり、階層化を行うと通信界面が分割されていくことになる。その際に新たに追加された節点の並びが同一になるように階層化を行う。マルチグリッドソルバーに対応するためにこの通信方式が選択された。

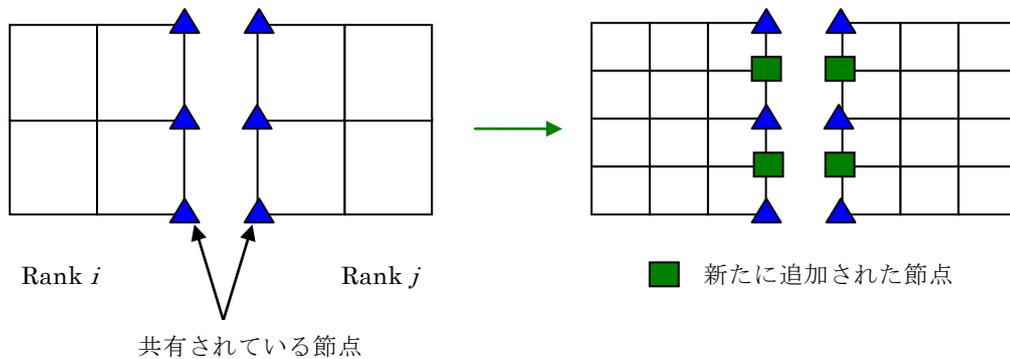


図 4.2.2 通信テーブル

4.3 MPI

本ライブラリでは、並列処理ライブラリとして **MPI** を用いている。

MPI ライブラリは並列処理を前提としているライブラリであるため、シングルプロセスの場合は、通常のプロセス起動となり **MPI** ライブラリは不要で、並列プロセスとは異なる。

しかし本ライブラリにおいては、シングル・並列プロセスどちらの場合でも、同一実装にするために、**MPI** をラップしたモジュールを提供する。

表 4.3.1 ラップする MPI の機能は下記の関数群

HEC_MW 関数	MPI 関数	備考
CHeCMPI:: Initialize	MPI_Init	MPI 使用の有無に関わりなく CHeCMPI 関数は起動可能

CHecMPI:: Finalize	MPI_Finalize	〃
CHecMPI:: Isend	MPI_Isend	〃
CHecMPI:: Irecv	MPI_Irecv	〃
CHecMPI:: Wait	MPI_Wait	〃
CHecMPI:: Allreduce	MPI_Allreduce	〃
CHecMPI:: Barrier	MPI_Barrier	〃
CHecMPI:: Abort	MPI_Abort	〃
CHecMPI:: Send	MPI_Send	〃
CHecMPI:: Recv	MPI_Recv	〃
CHecMPI:: Allgather	MPI_Allgather	〃
CHecMPI:: Gather	MPI_Gather	〃
CHecMPI:: Scatter	MPI_Scatter	〃
CHecMPI:: Bcast	MPI_Bcast	〃
CHecMPI:: getRank	MPI_Comm_rank	〃
CHecMPI:: getNumOfProcess	MPI_Comm_size	〃

4.4 線形代数ソルバー

並列線形ソルバーを本ライブラリは提供する。

並列計算では、解析に先立って、全体領域のメッシュデータを領域分割し、同じフォーマットの部分領域ごとのメッシュデータを作成する。

この部分領域ごとのデータは分散領域メッシュと呼ばれ、これを並列計算に使う各プロセッサに持たせる。各プロセッサにおいては独立に剛性行列の作成がなされ、線形代数ソルバーの中で MPI を使用した領域間通信を行うことによって領域全体の整合性をとり、並列計算を実現する。

本ライブラリで利用可能な線形代数ソルバーの種類は以下である。

- ・ 前処理付き反復法 ; CG、GMRES、BiCGSTAB、GPBiCG
- ・ 反復法に用いる前処理 ; Jacobi、ILU、SSOR、幾何学的マルチグリッドマルチグリッド CG

4.4.1 CG

基本となる CG 法についてのアルゴリズムを次に示す。アルゴリズムの詳細は理論説明書を参照されたい。

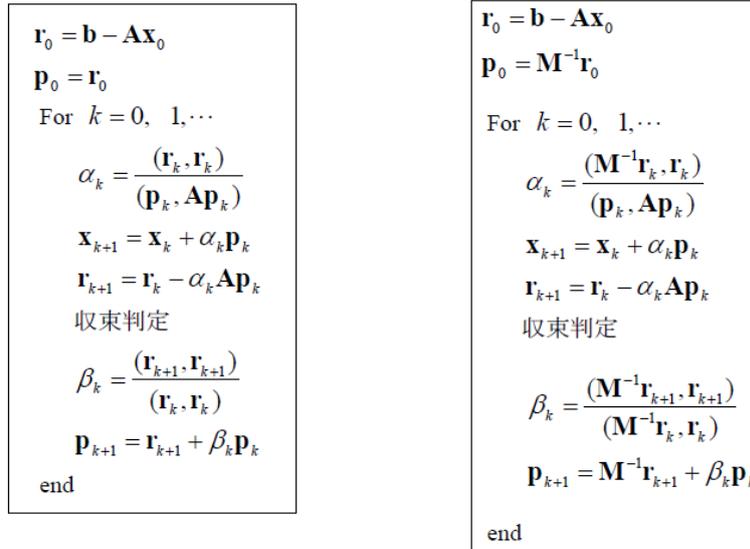


図 4.4.1 CG 法アルゴリズムおよび前処理つき CG 法アルゴリズム

4.4.2 マルチグリッド CG

通常の反復法はメッシュサイズに相当する波長を持った誤差成分の減衰には適しているが、誤差の成分のうち、長い波長の成分は反復を繰り返してもなかなか収束しない。粗い格子を使用することで、長波長成分を効率的に減衰させることを目的とした手法がマルチグリッド法である。マルチグリッド法の各反復における手順の概要を以下に示す。

- ① 細メッシュで残差を計算
- ② 残差を粗メッシュに制限近似 (restriction)
- ③ 粗メッシュで補正式の反復計算
- ④ 細メッシュへ補正量を延長補間 (prolongation)
- ⑤ 細メッシュでの解を更新

MGCG 法は、CG 法に上記の MG 法を前処理として適用した方法であり、本ソフトウェアは、自動メッシュ細分化による MGCG 法をサポートしている。

MGCG 法のアルゴリズムを下図に示す。図中で左枠内はマルチグリッド前処理を含む MGCG 法のアルゴリズムであり、右枠内はマルチグリッド前処理のアルゴリズムである。

<pre> r₀ = f - Ku₀ for <i>k</i> = 0..<i>maxiter</i> z^{<i>k</i>} = MG(K, r^{<i>k</i>}, initial_z^{<i>k</i>}, γ, μ_1, μ_2) $\rho_k = (\mathbf{r}_k, \mathbf{z}_k)$ if <i>k</i> ≡ 0 p₀ = r₀ else $\beta_k = \frac{\rho_k}{\rho_{k-1}}$ p_{<i>k</i>} = z_{<i>k</i>} + β_kp_{<i>k-1</i>} endif q_{<i>k</i>} = Kp_{<i>k</i>} $\alpha_k = \frac{\rho_k}{(\mathbf{p}_k, \mathbf{q}_k)}$ u_{<i>k+1</i>} = u_{<i>k</i>} + α_kp_{<i>k</i>} r_{<i>k+1</i>} = r_{<i>k</i>} - α_kq_{<i>k</i>} check convergence; continue if necessary end </pre>	<pre> MG(K_{<i>level</i>}, f, u, γ, μ_1, μ_2) { if <i>level</i> ≡ coarsest_level solve K_{<i>level</i>} u = f else u = pre_smoothing(K_{<i>level</i>}, f, u, μ_1) r = restrict(f - K_{<i>level</i>} u) $\Delta \mathbf{u}_C = \text{initial_}\Delta \mathbf{u}$ for <i>n</i> = 1..γ $\Delta \mathbf{u}_C = \text{MG}(\mathbf{K}_{\text{level-1}}, \mathbf{r}, \Delta \mathbf{u}_C, \gamma, \mu_1, \mu_2)$ end u = u + prolongate($\Delta \mathbf{u}_C$) u = post_smoothing(K_{<i>level</i>}, f, u, μ_2) endif return u } </pre>
MGCG 法	MG 前処理

図 4.4.2 MGCG 法のアルゴリズム

プログラムの内部では、前処理のひとつの機能として呼ばれているため、ユーザはマルチグリッドであることを意識してプログラミングする必要はない。実際のプログラム内部の処理は、つぎの3つの処理をことを行っている。

- ① 粗格子から密格子に至る全レベルの格子情報の生成。レベル間のインデクスも作成する必要がある (restriction and prolongation)。
- ② 上記インデクスを利用した全レベルの剛性行列の作成をユーザ側プログラムで行う(サンプルプログラムを参照)。
- ③ ソルバの前処理では、①の上下関係を表すインデクスを利用して、②で作成した剛性行列を呼び出しながら機能する。

なお、CG ソルバー及び対角スケーリング前処理においては、OpenMP を利用したマルチスレッド計算に対応しており、MPI 並列計算との併用も可能となっている。

4.4.3 MPC 前処理付き反復法

MPC 条件の組み込みに際しては、組み込みの容易さから、ペナルティ法が採用されることが多い。しかし、この場合、方程式が悪条件となり、反復法による求解が難しく、直接法の利用がより適しています。一方、大規模問題を並列環境で解く場合、直接法の利用は不可能であり、反復法の利用が必

須となる。

このため、ペナルティ法に変わる MPC 条件の組み込み方法として、本ソフトウェアでは自由度消去法を導入した。自由度消去法を用いた場合、MPC により接合される各パーツが一体のものとしてモデル化された場合と等価な方程式となるため、ペナルティ法のように方程式が悪条件となることはない。

通常、自由度消去法による MPC 条件の組み込みは、マトリックスの非ゼロのプロファイルが変更される。HEC-MW のマトリックスは非ゼロ成分のみを保持している (CRS : Compressed Row Storage) ため、非ゼロのプロファイルを変更するためには煩雑な処理が必要となり、望ましくない。そこで、以下に示す方法により、マトリックス自体には MPC 条件を組み込まず、ソルバーの各反復の中で MPC による拘束自由度を消去する方法を採用した。

解くべき方程式は以下のように表される。

$$\mathbf{Ku} = \mathbf{f} \quad (4.1)$$

$$\mathbf{Bu} = \mathbf{g} \quad (4.2)$$

ここで、 \mathbf{K} は係数マトリックス ($N \times N$)、 \mathbf{u} は未知数ベクトル ($1 \times N$)、 \mathbf{f} は右辺ベクトル ($1 \times N$)、 \mathbf{B} および \mathbf{g} はそれぞれ多点拘束条件を表す係数マトリックスおよび定数ベクトルであり、多点拘束条件の数を M とすると、大きさはそれぞれ $M \times N$ および $1 \times M$ である。

(4.2) から M 個の従属自由度を決め、それらを消去し、それ以外の独立自由度についての方程式を解くことを考える。

全自由度のうち、独立自由度のみが意味を持つ未知数ベクトルを \mathbf{u}' ($1 \times N$)、 \mathbf{u} と \mathbf{u}' との間の変換行列を \mathbf{T} とすると、(4.2) の拘束条件は

$$\mathbf{u} = \mathbf{T}\mathbf{u}' + \mathbf{u}_g \quad (4.3)$$

の形で表すことができる。ただし、 \mathbf{u}_g は \mathbf{B} および \mathbf{g} から決まる定数ベクトルである。

たとえば、 $N = 5$ 、 $u_5 - u_4 = 1$ の場合、多点拘束条件は、(4.2) の形式では、

$$\begin{bmatrix} 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{Bmatrix} u_4 \\ u_5 \end{Bmatrix} = [1]$$

(1.8-3) の形式では、

$$\begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{Bmatrix} u'_1 \\ u'_2 \\ u'_3 \\ u'_4 \\ u'_5 \end{Bmatrix} + \begin{Bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{Bmatrix}$$

と表される。

(4.3)を(4.1)に代入し、定数項を右辺に移項すると、

$$\mathbf{K}\mathbf{T}\mathbf{u}' = \mathbf{f} - \mathbf{K}\mathbf{u}_g$$

さらに、係数行列を対称にするため、両辺の左から \mathbf{T}^T をかけると

$$\mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{u}' = \mathbf{T}^T(\mathbf{f} - \mathbf{K}\mathbf{u}_g) \tag{4.4}$$

が得られる。(4.4)が、最終的に解くべき、多点拘束を組み込んだ方程式である。

(4.4)に対して反復解法を適用する。その際、 $\mathbf{T}^T\mathbf{K}\mathbf{T}$ を陽には保持せず、 \mathbf{T} や \mathbf{T}^T との掛け算を必要に応じてその都度行う。通常反復法におけるマトリックス・ベクトル積1回について、本手法では \mathbf{T} および \mathbf{T}^T との積の計算が増えることになるが、 \mathbf{T} はほとんど単位行列であるため、その計算コストは低く抑えることが可能である。

なお、(4.4)において、 $\mathbf{T}^T\mathbf{K}\mathbf{T}$ の従属自由度に関する行および列にはすべて0が入ることになり、解が唯一ではない。しかし、反復解法を適用した場合、未知ベクトルの初期値として従属自由度成分を0とすることで、修正ベクトル・残差ベクトルともに従属自由度成分は常に0となり、従属自由度成分を無視した形での求解が可能である。

したがって、これらの処理をプログラム内部で実行する場合には、行列ベクトル積の

$$\mathbf{K}\mathbf{x}$$

の計算をする代わりに

$$\mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{x}$$

とすればいいことがわかる。

4.5 ベクトル管理

本ライブラリで解くべき線形代数方程式 $\mathbf{A}\mathbf{x} = \mathbf{b}$ の、 \mathbf{x} , \mathbf{b} ベクトル配列の管理を行う。

4.5.1 自由度混在ベクトルの管理

ベクトル管理機能において、任意自由度のベクトル列を管理する機能を有する。

4.5.2 基礎演算機能

本ライブラリは、ベクトルのノルム計算、ベクトル同士の内積計算、ベクトルの代入機能を提供する。

表 4.5.1 ベクトルに関する基礎演算機能

関数名	機能
CAssyVector::add	ベクトル同士の和

CAsyVector::subst	ベクトルの代入
CAsyVector::norm2	ベクトルのノルムの2乗
CAsyVector::innerProd	ベクトルのペアの内積

4.6 マトリックス管理

本ライブラリで解くべき線形代数方程式 $Ax = b$ の、行列 A 疎行列の管理を行う。

4.6.1 自由度混在型マトリックスの管理

マトリックス管理機能において、任意自由度のメッシュデータのマトリックスを管理する機能を有する。ただし、アセンブリー構造に存在する複数メッシュブロックのうち、ひとつのメッシュブロックでは自由度数は一致していなければならない。

4.6.2 非ゼロプロファイルの生成

FEM のアルゴリズムにより生成される行列は疎行列と呼ばれ、そのほとんどを 0 が占めているが、実際の計算に必要な値は、非ゼロの行列成分であり、非ゼロプロファイルを生成することで計算の実行時間の短縮が図れる。そのため、非ゼロプロファイルの生成と管理を行う機能を本ライブラリは有する。

	1	2	3	4	5	6	7	8
1	1		1		3	4		
2		2		2	3	5		
3	1		4		4		5	
4		9		3		10		
5			32		4		2	
6		1		5		6		7
7	1		1				1	8
8		3		5	6			2

```

NP, NPL, NPU  8, 10, 12
D              {1, 2, 4, 3, 4, 6, 1, 2}
AL             {1, 9, 32, 1, 5, 1, 1, 3, 5, 6}
indexL        {0: 0, 0, 1, 2, 3, 5, 7, 10}
itemL         {1, 2, 3, 2, 4, 1, 3, 2, 4, 5}
AU            {1, 3, 4, 2, 3, 5, 4, 5, 10, 2, 7, 8}
indexU        {0: 3, 6, 8, 9, 10, 11, 12, 12}
itemU         {3, 5, 6, 4, 5, 6, 5, 7, 6, 7, 8, 8}

```

図 4.6.1 CRS Format の例

4.6.3 要素剛性行列の足し込み

FEM アルゴリズムにおいて、最終的な線形代数方程式 $A x = b$ の A を求めるためには、要素剛性行列を全体行列に足し込む機能が必要となる、この機能を提供する。

4.6.4 基礎演算機能

本ライブラリは、マトリクスとベクトルの積を求める機能を提供する。また、不完全 LU 分解を行う機能を有する。

表 4.6.1 マトリクスに関する基礎演算機能

関数名	機能
CAssyMatrix::multVector	マトリクスとベクトルの積
CAssyMatrix::multMPC	MPC 行列とベクトルの積
CAssyMatrix::residual	マトリクスとベクトルの積とベクトルを比較する(主に $Ax=b$ を計算する場合に利用する)
CAssyMatrix::setupPreconditioner	不完全 LU 分解を実行する
CAssyMatrix::precond	不完全 LU 分解に基づく前進消去後退代入を行う

4.7 要素ライブラリ

HEC_MW は、Hexa(1 次、2 次)、Tetra(1 次、2 次)、Prism(1 次、2 次)、Quad(1 次、2 次)、Triangle(1 次、2 次)、Line(1 次、2 次)の 12 種類の形状関数を所有している。

以下に、形状関数を列挙する。

4.7.1 Hexa 要素

1 次要素

節点数	8 節点
積分点数	1 点、8 点

形状関数(積分点座標 : r, s, t)

$$N_0 = 0.125 (1 - r) (1 - s) (1 - t)$$

$$N_1 = 0.125 (1 + r) (1 - s) (1 - t)$$

$$N_2 = 0.125 (1 + r) (1 + s) (1 - t)$$

$$N_3 = 0.125 (1 - r) (1 + s) (1 - t)$$

$$N_4 = 0.125 (1 - r) (1 - s) (1 + t)$$

$$N_5 = 0.125 (1 + r) (1 - s) (1 + t)$$

$$N_6 = 0.125 (1 + r) (1 + s) (1 + t)$$

$$N_7 = 0.125 (1 - r) (1 + s) (1 + t)$$

2 次要素

節点数	20 節点
積分点数	1 点、8 点、27 点

形状関数(積分点座標 : r, s, t)

$$N_0 = -0.125(1 - r) (1 - s) (1 - t) (2+r+s+t)$$

$$N_1 = -0.125(1 + r) (1 - s) (1 - t) (2-r+s+t)$$

$$N_2 = -0.125(1 + r) (1 + s) (1 - t) (2-r-s+t)$$

$$N_3 = -0.125(1 - r) (1 + s) (1 - t) (2+r-s+t)$$

$$N_4 = -0.125(1 - r) (1 - s) (1 + t) (2+r+s-t)$$

$$N_5 = -0.125(1 + r) (1 - s) (1 + t) (2-r+s-t)$$

$$N_6 = -0.125(1 + r) (1 + s) (1 + t) (2-r-s-t)$$

$$N_7 = -0.125(1 - r) (1 + s) (1 + t) (2+r-s-t)$$

$$N_8 = 0.25(1 - r^2) (1 - s) (1 - t)$$

$$N_9 = 0.25(1 + r) (1 - s^2) (1 - t)$$

$$N_{10} = 0.25(1 - r^2) (1 + s)(1 - t)$$

$$N_{11} = 0.25(1 - r) (1 - s^2) (1 - t)$$

$$N_{12} = 0.25(1 - r^2) (1 - s) (1 + t)$$

$$N_{13} = 0.25(1 + r) (1 - s^2) (1 + t)$$

$$N_{14} = 0.25(1 - r^2) (1 + s) (1 + t)$$

$$N_{15} = 0.25(1 - r) (1 - s^2) (1 + t)$$

$$N_{16} = 0.25(1 - r) (1 - s) (1 - t^2)$$

$$N_{17} = 0.25(1 + r) (1 - s) (1 - t^2)$$

$$N_{18} = 0.25(1 + r) (1 + s) (1 - t^2)$$

$$N_{19} = 0.25(1 - r) (1 + s) (1 - t^2)$$

4.7.2 Tetra 要素

1 次要素

節点数	4 節点
積分点数	1 点

形状関数(積分点座標 : L₁, L₂, L₃)

$$N_0 = 1 - L_1 - L_2 - L_3$$

$$N_1 = L_1$$

$$N_2 = L_2$$

$$N_3 = L_3$$

2 次要素

節点数	10 節点
積分点数	1 点、4 点、15 点

形状関数(積分点座標 : L_1, L_2, L_3)

$$a = 1.0 - L_1 - L_2 - L_3$$

$$N_0 = (2a - 1.0)a$$

$$N_1 = L_1(2L_1 - 1.0)$$

$$N_2 = L_2(2L_2 - 1.0)$$

$$N_3 = L_3(2L_3 - 1.0)$$

$$N_4 = 4L_1 a$$

$$N_5 = 4L_1 L_2$$

$$N_6 = 4L_2 a$$

$$N_7 = 4L_3 a$$

$$N_8 = 4L_1 L_3$$

$$N_9 = 4L_2 L_3$$

4.7.3 Prism 要素

1 次要素

節点数	6 節点
積分点数	2 点

形状関数(積分点座標 : L_1, L_2, ξ)

$$a = 1.0 - L_1 - L_2$$

$$N_0 = 0.5a(1.0 - \xi)$$

$$N_1 = 0.5L_1(1.0 - \xi)$$

$$N_2 = 0.5L_2(1.0 - \xi)$$

$$N_3 = 0.5a(1.0 + \xi)$$

$$N_4 = 0.5L_1(1.0 + \xi)$$

$$N_5 = 0.5L_2(1.0 + \xi)$$

2 次要素

節点数	15 節点
-----	-------

形状関数(積分点座標 : L_1, L_2, ξ)

$$a = 1.0 - L_1 - L_2$$

$$N_0 = 0.5a(1 - \xi)(2a - 2 - \xi)$$

$$N_1 = 0.5L_1(1 - \xi)(2L_1 - 2 - \xi)$$

$$N_2 = 0.5L_2(1 - \xi)(2L_2 - 2 - \xi)$$

$$N_3 = 0.5a(1 + \xi)(2a - 2 + \xi)$$

$$N_4 = 0.5L_1(1 + \xi)(2L_1 - 2 + \xi)$$

$$N_5 = 0.5L_2(1 + \xi)(2L_2 - 2 + \xi)$$

$$N_6 = 2L_1a(1 - \xi)$$

$$N_7 = 2L_1L_2(1 - \xi)$$

$$N_8 = 2L_2a(1 - \xi)$$

$$N_9 = 2L_1a(1 + \xi)$$

$$N_{10} = 2L_1L_2(1 + \xi)$$

$$N_{11} = 2L_2a(1 + \xi)$$

$$N_{12} = a(1 - \xi^2)$$

$$N_{13} = L_1(1 - \xi^2)$$

$$N_{14} = L_2(1 - \xi^2)$$

4.7.4 Quad 要素 エンティティ番号

1 次要素

節点数	4 節点
積分点数	1 点

形状関数(積分点座標 : r, s)

$$N_0 = 0.25(1-r)(1-s)$$

$$N_1 = 0.25(1+r)(1-s)$$

$$N_2 = 0.25(1+r)(1+s)$$

$$N_3 = 0.25(1-r)(1+s)$$

2 次要素

節点数	8 節点
積分点数	4 点、9 点

形状関数(積分点座標 : r, s)

$$N_0 = 0.25(1-r)(1-s)(-1-r-s)$$

$$N_1 = 0.25(1+r)(1-s)(-1+r-s)$$

$$N_2 = 0.25(1+r)(1+s)(-1+r+s)$$

$$N_3 = 0.25(1-r)(1+s)(-1-r+s)$$

$$N_4 = 0.5(1-r^2)(1-s)$$

$$N_5 = 0.5(1-s^2)(1+r)$$

$$N_6 = 0.5(1-r^2)(1+s)$$

$$N_7 = 0.5(1-s^2)(1-r)$$

4.7.5 Triangle 要素エンティティ番号

1 次要素

節点数	3 節点
積分点数	1 点

形状関数(積分点座標 : r, s)

$$N_0 = r$$

$$N_1 = s$$

$$N_2 = 1 - r - s$$

2 次要素

節点数	6 節点
積分点数	3 点

形状関数(積分点座標 : r, s)

$$t = 1 - r - s;$$

$$N_0 = t(2t-1)$$

$$N_1 = r(2r-1)$$

$$N_2 = s(2s-1)$$

$$N_3 = 4rt$$

$$N_4 = 4rs$$

$$N_5 = 4st$$

4.7.6 Beam 要素エンティティ番号

1 次要素

節点数	2 節点
積分点数	1 点

形状関数(積分点座標 : r)

$$N_0 = 0.5(1-r)$$

$$N_1 = 0.5(1+r)$$

2 次要素

節点数	3 節点
積分点数	2 点

形状関数(積分点座標 : r)

$$N_0 = -0.5(1-r)r$$

$$N_1 = 0.5(1+r)r$$

$$N_2 = 1-r^2$$

4.7.7 入力ファイル

HECMW ファイル管理は、2 種類存在する。一つは、HECMW 標準の命名規則でのファイル名管理で、もう一方は FrontISTR の命名規則に応じたファイル名管理である

HECMW 標準のファイル管理は、アプリケーション・プログラムから渡されるファイルベース名 (Basename) を用いて、拡張子を追加付与することで全てのファイル名を管理する。つまり基本となるファイル名は一つで、用途別に拡張子を変えて区別する。この場合に HECMW が管理するファイルの種類を表 4.7.1 に示す。

表 4.7.1 HECMW が管理するファイル

ファイル種類	ファイル名
メッシュファイル	Basename .PE#.msh
リスタートファイル	Basename.PE#.Step#.res
計算結果出力ファイル	Basename.PE#.Part#.inp

注 1) Basename は、ファイル読み込み API 関数(mw_file_read, CMW::FileRead)の引数で、アプリケーション側から指定する。

注 2) PE#は、領域分割のプロセス番号、Part#はメッシュパーツ番号、Step#はアプリケーション側から指定するステップ番号で任意の番号になる。

アプリケーション・プログラムで管理することを想定しているファイル

- ・ 解析制御ファイル：ファイル名称、ファイル内容は、アプリケーション側の裁量
- ・ 材質データファイル：ファイル名称、ファイル内容は、アプリケーション側の裁量

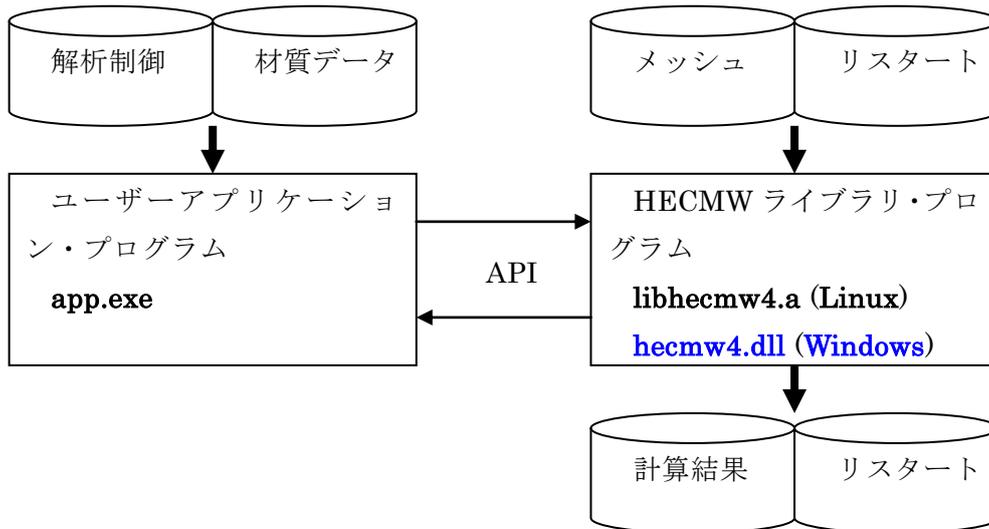


図 4.7.1 HECMW、アプリケーション・プログラムとそれぞれが管理するデータの関係

FrontISTR タイプのファイル管理は、全体制御ファイルに記述された用途別のファイル名に従う。全体制御ファイル名は、標準では“hecmw_ctrl.dat”となっているが、自由なファイル名を用いることができる。全体制御ファイルで記述された各ファイル名をそれぞれ“MeshName”、“CntName”、“RestartName”、“ResultName”、“PartInName”、“PartOutName”、“VisMeshName”、“VisInName”、“VisOutName”とした場合のファイル名管理を表 4.7.2 に記す。

表 4.7.2 FrontISTR タイプのファイル名管理規則

ファイル種類	ファイル名
メッシュファイル	MeshName.PE# 拡張子なし
解析制御ファイル	CntName 拡張子なし
リスタートファイル	RestartName.PE# 拡張子なし
リザルトファイル	ResultName.PE#.Step# 拡張子なし
パーティショナー 入力ファイル	PartInName 拡張子なし
パーティショナー 出力ファイル	PartOutName.PE# 拡張子なし
ビジュアライザー メッシュファイル	VisMeshName.PE# 拡張子なし
ビジュアライザー 入力ファイル	VisInName.PE#.Stet# 拡張子なし
ビジュアライザー 出力ファイル	VisOutName.Step#.拡張子*

注 1)ビジュアライザーの出力ファイル拡張子は、MicroAVS の場合は“*.inp”、ビットマップの場合は“.bmp”となる。

注 2)PE#は、領域分割のプロセス番号、Step#はアプリケーション側から指定するステップ番号で任意の番号になる。

注 3)FrontISTR タイプでの解析制御ファイル管理は、解析制御ファイル名を HECMW から取得して管理することになる。

FrontISTR タイプのファイル管理の詳細については、FrontISTR ユーザーズマニュアルを参照のこと。

4.7.8 出力ファイル

4.7.8.1. 計算結果ファイル

MicroAVS UCD ファイル書式での出力に対応。

4.7.8.2. リスタートファイル

リスタートファイルは、アプリケーション指定により生成された、方程式の数、方程式ごとの自由度、および基礎変数計算結果により構成される。

5. HEC_MW を利用したプログラムの記述方法

本節では、HEC_MW の使い方をサンプルコードを用いて説明する。

5.1 HECMW を利用した線形静解析のサンプルプログラム

HECMW API を利用して 3 次元の線型弾性プログラムを記述した例を示す。このサンプルプログラムは、ダウンロード後に解凍した `examle` ディレクトリに入っている。

この例題プログラムを使用して、API の使用方法を説明する。

API 呼び出し部分を□で囲って記述し、実際の実装部分を示す。

5.1.1 インクルード・ヘッダー

HEC_MW ライブラリを利用するために” HEC_MW3.h ” をインクルードする。

```
#include "HEC_MW3.h"
```

```
17 #include "HEC_MW3.h"
18 #include "Control.h"
```

その下でインクルードされているファイルは、例題プログラムの一部であり、解析制御ファイルを読むための実装がなされている `Control` クラスのヘッダーである。

`CMW::Instance` により HECMW ライブラリのオブジェクト・インスタンスを取得する。これにより得られた HECMW のインスタンスは、例題プログラム中では `pMW` という変数名で記述されている、よって `pMW` をキーワードに例題プログラムソースを追うと API が判別できる。

API;

```
CMW *pMW = CMW::Instance();
```

尚、`CMW::Instance` 関数の呼び出しは、C++特有であり、Fortran / C 言語については、`mw_initialize` 関数、もしくは `mw_initialize_fstr` 関数内部で HECMW が生成されている。

実装は下記の部分である。

```
22 int main(int argc, char** argv)
23 {
24     if( argc == 1 ){
25         cout << "control file must be specified";
26         exit(EXIT_FAILURE);
27     }
28
29     CMW *pMW = CMW::Instance();// construct MW3
```

5.1.2 初期化処理

HECMW を初期化する。

API;

```
pMW->Initialize(argc, argv);
```

実装は下記の部分である。

```
60      // --
61      // standard style : app initialize func
62      // --
63      pMW->Initialize(argc, argv);
64      pMW->Banner_Display();
```

5.1.3 メッシュファイルの読み込み、モデルデータの構築

FileRead 関数でメッシュファイルを読み込み、Refine 関数でメッシュモデルの構築を行っている。関数名が Refine となっているが、階層数が 0 であっても、この関数の呼び出しは必須である。

Refine 後に FinalizeRefine で例題プログラムに不要なデータを解放している。

API;

```
pMW->FileRead(sBaseName, ASCII);
```

```
pMW->Refine(nnRefine);
```

```
pMW->FinalizeRefine();
```

実装は下記の部分である。

```
162      // --
163      // read mesh file
164      // --
165      if(nMeshFileBINARY==0){
166          pMW->FileRead(sBaseName, ASCII);
167      }else{
```

```
194          pMW->Refine(nNumRefine); // construct multigrid model
195          pMW->FinalizeRefine(); // memory release
```

このあと、解析制御ファイルの読み込み処理を行うが、この処理部分は、API に関係なくアプリケーション開発者が独自に実装する部分であるので、説明は省略する。この例題では Control クラスのインスタンスにより解析制御ファイルを読み込んでいる。

5.1.4 材料定数の定義

例題プログラムでは、ソース内に材料定数をリテラルとして直接定義している。

この材料データを用いて構成則 \mathbf{D} 行列を作成している。

実装は下記の部分である。

```
202 // --
203 // Material Properties
204 // --
205 double nyu = 0.3;
206 double E = 1.0e9;
207 double coef = ((1.0-nyu)*E)/((1.0+nyu)*(1.0-2*nyu));
208 double D[6][6] = { {coef, coef*nyu/(1.0-nyu), coef*nyu/(1.0-nyu),
0, 0, 0},
209 {coef*nyu/(1.0-nyu), coef, coef*nyu/(1.0-nyu), 0, 0,
0},
210 {coef*nyu/(1.0-nyu), coef*nyu/(1.0-nyu), coef, 0, 0,
0},
211 {0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu)), 0, 0},
212 {0, 0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu)), 0},
213 {0, 0, 0, 0, 0, coef*(1-2*nyu)/(2*(1.0-nyu))} };
```

5.1.5 方程式の確保

HECMW 内部に生成する線型方程式の領域を確保する。例題プログラムでは自由度3の方程式を一つ生成している。

API;

線型方程式(行列,ベクトル)を生成: `pMW->GeneLinearAlgebra(nNumOfAlgebra, vDOF);`

実装は下記の部分である。

```
214 // --
215 // Ax=b generate
216 // --
217 pMW->GeneLinearAlgebra(nNumEquation, nGlobalNumMesh, vvDOF);
218 pMW->LoggerInfo(nInfo, "%s%d", "MW_Str:F ", nPE);
```

5.1.6 FEM メッシュデータにアクセスするためのループ処理部分

最初のループ処理として、階層を指定する。ソースの中では、MG-Level loop とコメントしてある部分で、iAssemble をインデックスとしてループ処理されている部分である。

次のループ処理は、メッシュパーツの指定をするためのループ処理である。その次に要素を指定するためのループ処理を記述する。これで全ての階層のメッシュパーツの要素を指定することができる。

API;

マルチグリッド階層レベルを指定: `pMW->SelectAssembleModel(iAssemble);`

線型方程式番号を指定: `pMW->SelectAlgebra(0);`

メッシュパーツを指定: `pMW->SelectMeshPart_IX(iMesh);`

要素を指定: `pMW->SelectElement_IX(iElem);`

実装は下記の部分である。

```
253 // --
254 // ### FEM start ###
255 // --
256 uint iAssembleMax, iMeshMax, iElemMax;
257 iAssembleMax = pMW->GetNumOfAssembleModel(); //Num of MultiGrid
```

```

Level
258      //
259      // - start of MG-Level loop
260      //
261      for(uiint iAssemble = 0; iAssemble < iAssembleMax; iAssemble++)
{
262          pMW->LoggerInfo(nInfo, "%s%d%s%d", "- MG-Level loop ",
iAssemble, "MaxLevel:", iAssembleMax-1);
263
264          // select MG-Level && Algebra-Number
265          pMW->SelectAssembleModel( iAssemble );// current MG-Level
( MG-Level==iAssemble )
266          pMW->SelectAlgebra(0);          // algebra_eqation num
( number==0 )
267          //
268          // -- start of mesh-part loop :## Local_Mesh in each PE
269          //
270          iMeshMax = pMW->GetNumOfMeshPart();
271          for(uiint iMesh = 0; iMesh < iMeshMax; iMesh++){
272              pMW->LoggerInfo(nInfo, "%s%d%s%d", "-- Mesh loop", iMesh,
"MaxMeshParts:", iMeshMax-1);
273
274              pMW->SelectMeshPart_IX( iMesh );// current MeshPart ( independent
Mesh )
275
276          // --- start of element loop
277          iElemMax = pMW->getElementSize( iMesh );
278          for(uiint iElem =0; iElem < iElemMax; iElem++){

```

5.1.7 形状関数処理部分

メッシュファイルに記述されている要素形状タイプに対応した形状関数を指定する部分である。

GetElementType 関数により、入力された要素タイプを取得し、そのタイプに応じた形状関数を指定する。

ここで指定されている形状関数は、HECMW ライブラリに備わっている要素ライブラリから取得できるようになっている。例えば要素形状が六面体一次の場合は、shapetype_hexa82 を指定している。shapetype_hexa82 は、六面体 1 次関数・積分点 8 点の形状関数である。

例題プログラムでは、取得した形状関数タイプを shapeType 変数に代入している。

形状関数の取得後に、弾性解析の剛性行列の前段階の行列として、歪行列 B と構成則 D 行列を用いた B^TDB 行列の配列確保も行っている。

API;

要素を指定: `pMW->SelectElement_IX(iElem);`

指定要素の”要素形状番号”を取得: `pMW->GetElementType();`

六面体の”要素形状番号”を取得: `pMW->elemtype_hexa()`

六面体 1 次要素・積分点 8 個の”形状関数番号”を取得: `pMW->shapetype_hexa82();`

!注)他の要素形状番号、形状関数番号は、省略する。下記の実装部分と、API の章を参照のこと。

実装は下記の部分である。

```
280     uiint numOfLocalNode;
281     double **B, **DB, **BDB, *ElemMatrix, weight, detJ;
282
283     uiint shapeType, elemType;
284     pMW->SelectElement_IX(iElem);
285
286     elemType = pMW->GetElementType();
287
288     uiint mat_size;
289     // Hexa Element
290     if(elemType == pMW->elemtype_hexa() ){
291         shapeType = pMW->shapetype_hexa82();
292         numOfLocalNode = 8;
293         mat_size = numOfLocalNode * 3; // 8Node * 3dof == 24
294         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
295     }
296     if(elemType == pMW->elemtype_hexa2() ){
297         shapeType = pMW->shapetype_hexa202();
298         numOfLocalNode = 20;
299         mat_size = numOfLocalNode * 3; // 20Node * 3dof == 60
300         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
301     }
302     // Tetra Element
303     if(elemType == pMW->elemtype_tetra() ){
304         shapeType = pMW->shapetype_tetra41();
305         numOfLocalNode = 4;
306         mat_size = numOfLocalNode * 3;
307         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
308     }
309     if(elemType == pMW->elemtype_tetra2() ){
310         shapeType = pMW->shapetype_tetra104();
311         numOfLocalNode = 10;
312         mat_size = numOfLocalNode * 3;
313         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
314     }
315     // Prism Element
316     if(elemType == pMW->elemtype_prism() ){
317         shapeType = pMW->shapetype_prism62();
318         numOfLocalNode = 6;
319         mat_size = numOfLocalNode * 3;
320         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
321     }
322     if(elemType == pMW->elemtype_prism2() ){
323         shapeType = pMW->shapetype_prism159();
324         numOfLocalNode = 15;
325         mat_size = numOfLocalNode * 3;
326         B = new double*[6]; DB = new double*[6]; BDB = new
double*[mat_size]; ElemMatrix = new double[mat_size*mat_size];
327     }
328     for(uiint i=0; i < 6; i++){ B[i] = new double[mat_size];
DB[i] = new double[mat_size];}
329     for(uiint i=0; i < mat_size; i++){ BDB[i] = new
double[mat_size];}
```

5.1.8 要素剛性行列の生成

形状関数の積分点ごとに、要素の各節点の空間座標での形状関数勾配を API から取得し、歪行列 B を生成している。

更に形状関数の積分点ごとにヤコビアン行列式と形状関数の重みを取得し、要素剛性行列を生成している。

API;

引数で指定された、形状関数の積分点数を取得: `pMW->NumOfIntegPoint(shapeType);`

形状関数の空間座標での勾配を取得: `pMW->dNdx(elemType, numOfInteg, iElem, dNdx);`

!注) for ループの "iElem" 番目の要素の勾配を取得している。

実装は下記の部分である。

```
331         uint numOfInteg = pMW->NumOfIntegPoint(shapeType);
332
333         vvvdouble dNdx;
334         pMW->dNdx(elemType, numOfInteg, iElem, dNdx);
335
336         // 0 clear
337         for(uint i=0; i < mat_size; i++) for(uint j=0; j < mat_size;
j++) ElemMatrix[mat_size*i+j] = 0.0;
338
339         // ---- start of Gauss points loop
340         for(uint igauss=0; igauss < numOfInteg; igauss++){
341             for(uint iLocalNode=0; iLocalNode < numOfLocalNode;
iLocalNode++){
342                 B[0][iLocalNode*3  ] = dNdx[igauss][iLocalNode][0]; //dNdx
343                 B[0][iLocalNode*3 +1] = 0.0;
344                 B[0][iLocalNode*3 +2] = 0.0;
345                 B[1][iLocalNode*3  ] = 0.0;
346                 B[1][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][1]; //dNdY
347                 B[1][iLocalNode*3 +2] = 0.0;
348                 B[2][iLocalNode*3  ] = 0.0;
349                 B[2][iLocalNode*3 +1] = 0.0;
350                 B[2][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][2]; //dNdZ
351                 B[3][iLocalNode*3  ] = dNdx[igauss][iLocalNode][1]; //dNdY
352                 B[3][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][0]; //dNdX
353                 B[3][iLocalNode*3 +2] = 0.0;
354                 B[4][iLocalNode*3  ] = 0.0;
355                 B[4][iLocalNode*3 +1] = dNdx[igauss][iLocalNode][2]; //dNdZ
356                 B[4][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][1]; //dNdY
357                 B[5][iLocalNode*3  ] = dNdx[igauss][iLocalNode][2]; //dNdZ
358                 B[5][iLocalNode*3 +1] = 0.0;
359                 B[5][iLocalNode*3 +2] = dNdx[igauss][iLocalNode][0]; //dNdX
360             }
```

要素剛性行列の生成部分の実装は下記である。

```
362         for(uint i=0; i < 6; i++){
363             for(uint j=0; j < mat_size; j++){
364                 DB[i][j] = 0.0;
365                 for(uint k=0; k < 6; k++) DB[i][j] += D[i][k]*B[k][j];
366             }
367         }
368         for(uint i=0; i < mat_size; i++){
369             for(uint j=0; j < mat_size; j++){
```

```

370      BDB[i][j] = 0.0;
371      for(uiint k=0; k < 6; k++) BDB[i][j] += B[k][i]*DB[k][j];
372      }
373      }
374      pMW->detJacobian(elemType, numOfInteg, igauss, detJ);
375      pMW->Weight(elemType, numOfInteg, 0, weight);
376
377      for(uiint i=0; i < mat_size; i++)
378          for(uiint j=0; j < mat_size; j++)
379              ElemMatrix[mat_size*i+j] += BDB[i][j] * weight * detJ;
380
381      };// ---- end of Gauss points loop

```

5.1.9 要素剛性行列の足しこみ

生成した要素剛性行列を HECMW の全体行列に足し込む。

API;

選択されている線型方程式番号に要素剛性行列を足しこんでいく:

```
pMW->Matrix_Add_Elem(iMesh, iElem, ElemMatrix);
```

!注) 線型方程式の選択は、5.1.6 で示している

実装は下記の部分である。

```

383      pMW->Matrix_Add_Elem(iMesh, iElem, ElemMatrix);// --- 要素剛性
行列の足し込み
384
385      for(uiint i=0; i < 6; i++){ delete[] B[i]; delete[] DB[i];}
386      for(uiint i=0; i < mat_size; i++) delete[] BDB[i];
387      delete[] B; delete[] DB; delete[] BDB;
388      delete[] ElemMatrix;
389
390      };// --- end of element loop

```

5.1.10 境界条件の設定

境界条件は、辺、面、体積、節点の各グループごとにメッシュデータとして管理されている。

それらは、各メッシュパーツの境界条件として管理されているのでメッシュパーツ・ループの中に境界条件を記述する。

モデル管理部分から、境界条件を取得し、右辺ベクトルに境界値をセットする。

辺グループに境界値がセットされている場合を例に API を説明する。

API;

```
境界条件・辺グループ数を取得: pMW->GetNumOfBoundaryEdgeMesh();
```

```
境界条件・辺グループの境界条件種類番号を取得: pMW->GetBNDType_BEEdgeMesh(iBEMesh);
```

```
境界条件・辺グループの自由度数を取得: pMW->GetNumOfDOF_BEEdgeMesh(iBEMesh);
```

!)自由度数であって、自由度番号ではない。境界条件として自由度数が 1 で合った場合を例にとると、一つだけの自由度番号の境界値を持っていることになる。

境界条件・辺グループの節点数を取得:`pMW->GetNumOfBNode_BEEdgeMesh(iBEMesh);`

境界条件・辺グループの境界節点番号から、メッシュ本体の節点 ID を取得;

`pMW->GetNodeID_BNode_BEEdgeMesh(iBEMesh, iBNode);`

境界条件・辺グループの自由度インデックスの自由度番号を取得:

`pMW->GetDOF_BEEdgeMesh(iBEMesh,idof);`

!)自由度インデックスとは、自由度番号の通し番号で、例えば自由度番号が“2”と“3”の二つがあった場合は、通し番号としてインデックス番号 0 の自由度番号=2、インデックス番号 1 の自由度番号=3 といった具合になる。

境界値を取得:`pMW->GetBNodeValue_BEEdgeMesh(iBEMesh,iBNode, nDOF, iAssemble);`

境界条件のタイプ番号を取得:`pMW->getNeumannType()`

!注) 境界条件の種類は、ディレクレ境界とノイマン境界の 2 種類。

右辺ベクトルへ境界値を代入:`pMW->Set_BC_RHS(iMesh, iBNodeID, nDOF, dBndVal);`

!注)例題プログラムでは、ノイマン境界条件の場合に使用。

ディレクレ境界条件の直接代入:

`pMW->Set_BC_Mat_RHS2(iMesh, iBNodeID, nDOF, dDiagonal, dBndVal);`

!注)境界値を持つ節点の対角項の値を指定し、境界値を右辺ベクトルに代入。

例題プログラムの中の辺グループ境界値設定の実装部分を示す。

```
435 // 2.boundary edge
436 uint nNumOfBEMesh = pMW->GetNumOfBoundaryEdgeMesh();
437 uint iBEMesh;
438 for(iBEMesh=0; iBEMesh < nNumOfBEMesh; iBEMesh++) {
439
440     uint nBndType = pMW->GetBNDType_BEEdgeMesh(iBEMesh);
441     uint nBDof = pMW->GetNumOfDOF_BEEdgeMesh(iBEMesh);
442     uint nNumOfBNode =
pMW->GetNumOfBNode_BEEdgeMesh(iBEMesh);
443     uint iBNode;
444     for(iBNode=0; iBNode < nNumOfBNode; iBNode++) {
445         uint iBNodeID =
pMW->GetNodeID_BNode_BEEdgeMesh(iBEMesh, iBNode);
446         // Neumann
447         if(nBndType==pMW->getNeumannType()){
448             for(uint idof=0; idof < nBDof; idof++) {
449                 uint nDOF =
pMW->GetDOF_BEEdgeMesh(iBEMesh,idof);
450                 double dBndVal =
pMW->GetBNodeValue_BEEdgeMesh(iBEMesh,iBNode, nDOF, iAssemble); //boundary
value
451
452                 pMW->Set_BC_RHS( iMesh, iBNodeID, nDOF,
dBndVal); // boundary set for RHS_vector
453             };
```

```

454         }
455         // Dirichlet
456         if(nBndType==pMW->getDirichletType()){
457             for(uiint idof=0; idof < nBDof; idof++) {
458                                     uiint nDof =
pMW->GetDof_BEdgeMesh(iBEMesh,idof);
459                                     double dBndVal =
pMW->GetBNodeValue_BEdgeMesh(iBEMesh, iBNode, nDof, iAssemble);//boundary
value
460
461                                     // direct set ( matrix off-diagonal=0, matrix
diagonal=1.0 )
462                                     double dDiagonal = 1.0;
463                                     pMW->Set_BC_Mat_RHS2(iMesh, iBNodeID, nDof,
dDiagonal, dBndVal);
464
465                                     //
466                                     ////double dDiagonal = 1.0E+6;
467                                     ////pMW->Set_BC_Mat_RHS(iMesh, iBNodeID, nDof,
dDiagonal, dBndVal);
468                                     };
469             }
470     };
471     };// --- end of boundary_edge_mesh loop

```

5.1.11 線形ソルバーの実行

階層レベルを指定し、線形方程式番号を指定した後で、線形ソルバーを実行する。

マルチグリッドの階層レベルの指定: `pMW->SelectAssembleModel(iAssembleMax - 1);`

線形方程式番号の指定: `pMW->SelectAlgebra(0);`

!)引数で指定しているのが、方程式番号。

線形方程式とは、行列と列ベクトルで構成されている一組の $Ax=b$ のことである。

線形ソルバーの実行: `pMW->Solve(nnMaxIter, dTolerance, nnType, nnPre);`

type = 1:CG, 2:BiCBSTAB, 3:GPBiCG, 4:GMRES || pre= 1:Jacobi, 2:MG, 3:SSOR, 4:ILU

実装は下記の部分である。

```

545         // --
546         // select MG-Level && AlgebraEquation-Number
547         // --
548         pMW->SelectAssembleModel( iAssembleMax - 1 );// finest MG-Level
(MG-Level==iAssembleMax-1)
549         pMW->SelectAlgebra(0); // algebra_eqation num
( number==0 )

602         // solve the algebra_equation : solv= 1:CG, 2:BiCBSTAB, 3:GPBiCG,
4:GMRES || pre= 1:Jacobi, 2:MG, 3:SSOR, 4:ILU
603         // --
604         pMW->Solve(nMaxIter, dTolerance, nSolvType, nPreType);

```

5.1.12 計算結果の出力

MicroAVS 書式 (UCD フォーマット) で計算結果を出力する。

メッシュパーツ毎に計算結果を出力するので、メッシュパーツを選択した後に、それぞれのメッシュパーツにラベルと出力値を登録し、ファイル出力を行う。

API;

メッシュパーツ数の取得: `pMW->GetNumOfMeshPart();`

メッシュパーツの選択: `pMW->SelectMeshPart_IX(iMesh);`

MicroAVS 出力ファイルのラベル登録: `pMW->recAVS_Label(iMesh, cLabel, cUnit, nDOF);`

MicroAVS 出力ファイルの出力値を登録: `pMW->recAVS_Variable(iMesh, nNumOfNode, cLabel, vValue);`

MicroAVS ファイル出力: `pMW->PrintMicroAVS_FEM();`

実装は下記の部分である。

```
606 // The output sample as same as PrintMicroAVS_Basis function.
607 iMeshMax = pMW->GetNumOfMeshPart();
608 for(uiint iMesh = 0; iMesh < iMeshMax; iMesh++){
609     char cLabel[5]="disp"; char cUnit[3]="mm"; uiint nDOF=3;
610     pMW->recAVS_Label(iMesh, cLabel, cUnit, nDOF); //recording
Label
611
612     pMW->SelectMeshPart_IX( iMesh );
613     uiint nNumOfNode = pMW->getNodeSize();
614     double *vValue = new double[nNumOfNode*nDOF];
615
616     for(uiint iNode=0; iNode < nNumOfNode; iNode++)
617         for(uiint iDOF=0; iDOF < nDOF; iDOF++) vValue[iNode*nDOF
+ iDOF]= pMW->GetSolutionVector_Val(iMesh, iNode, iDOF);
618
619     pMW->recAVS_Variable(iMesh, nNumOfNode, cLabel,
vValue); //recording value[]
620 };
621 pMW->PrintMicroAVS_FEM();
```

5.1.13 HECMW 終了処理

Finalize 関数を呼び出し、HECMW を終了する。

API;

HECMW を終了する: `pMW->Finalize();`

実装は下記の部分である。

```
670 pMW->Finalize(); // mpi finalize, logger close
```

5.2 使用例 1；単一メッシュパーツ

5.2.1 計算条件

次のような片持ち梁の形状で階層データを生成し、各リファインレベルでテストを実施した。

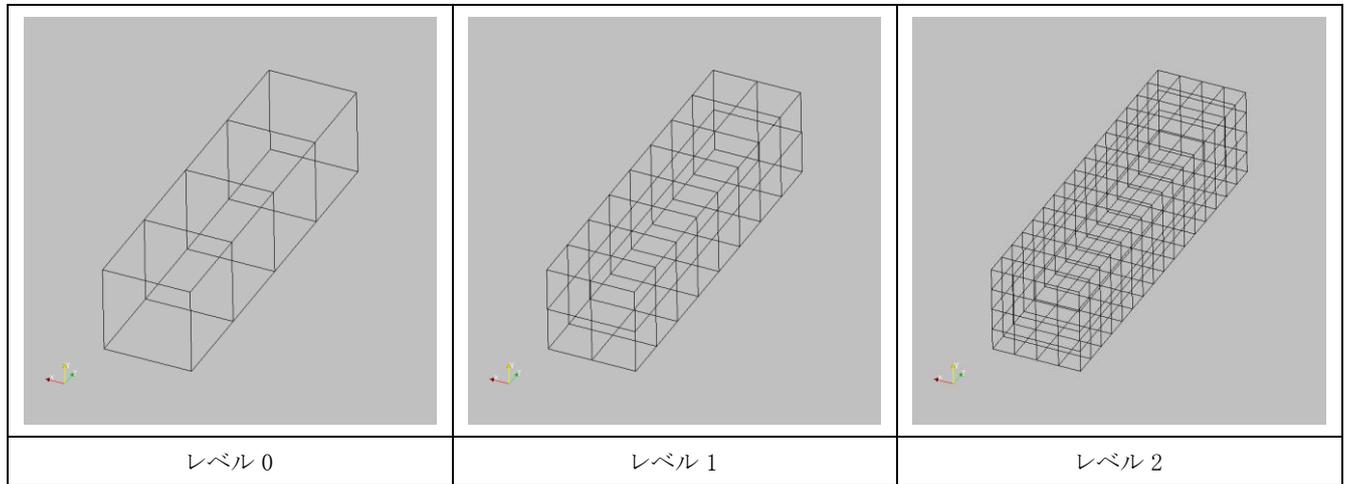
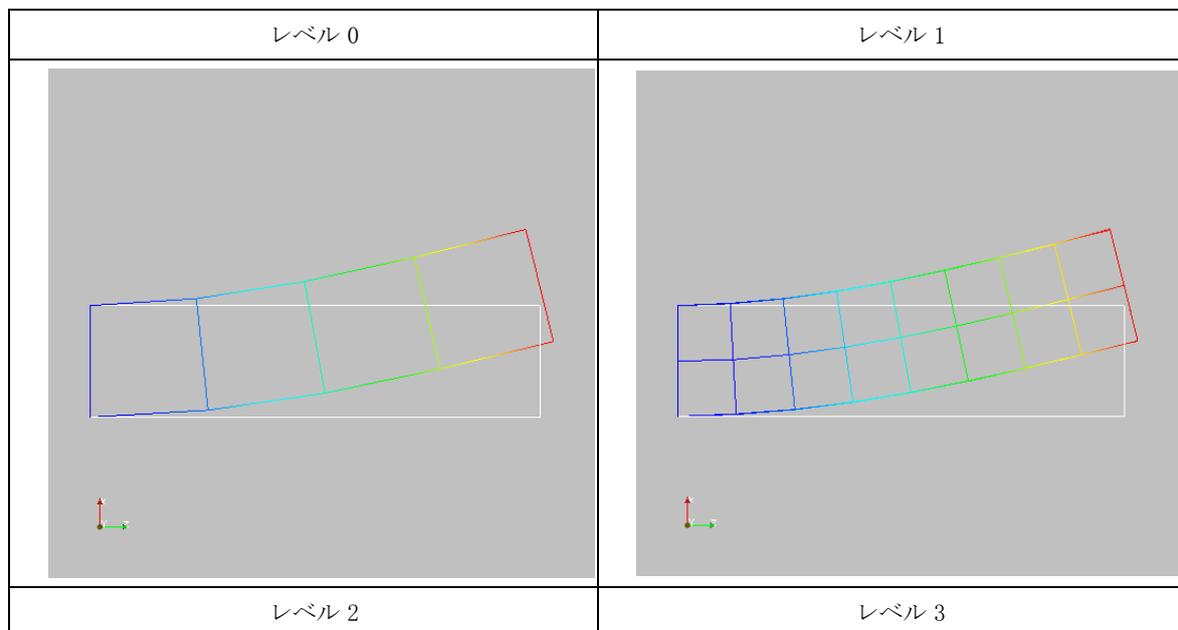


図 5.2.1 階層データを用いたデータ形状

5.2.2 計算結果

各リファインレベルの結果を示す。(図において、変位は倍率を乗じているが、この図では、その倍率はレベル毎に異なる値を利用している。)



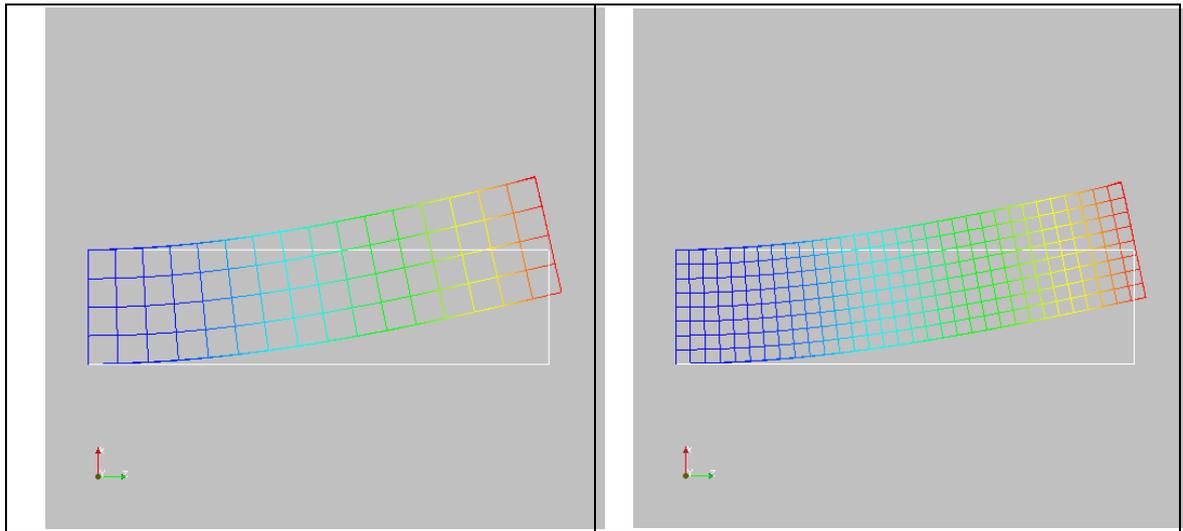


図 5.2.2 階層データを利用した解析結果

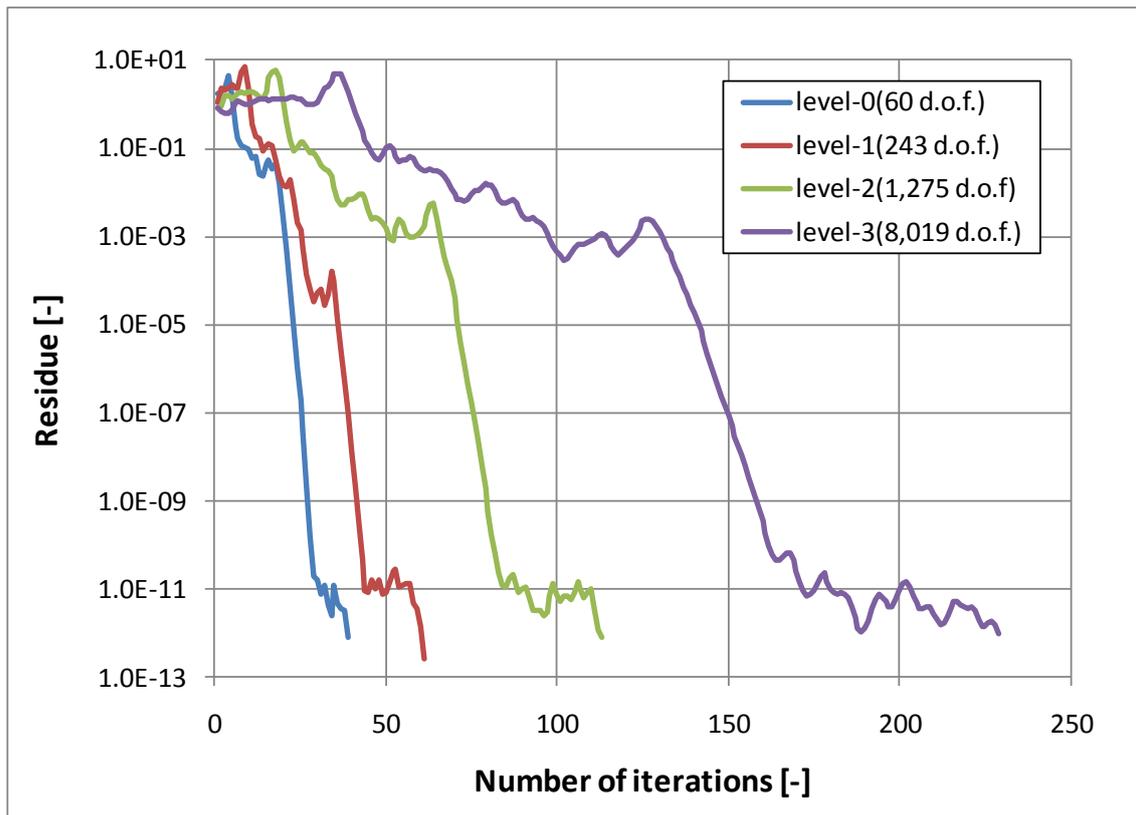


図 5.2.3 階層データを利用したケースの収束状況

また、参考までにレベル 5 の結果を示す。

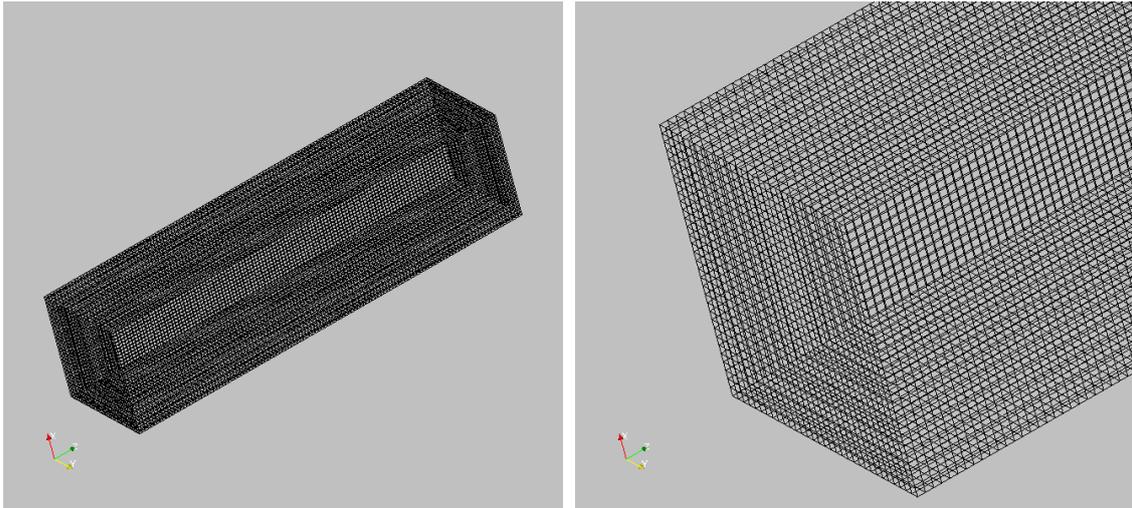


図 5.2.4 レベル 5 の格子

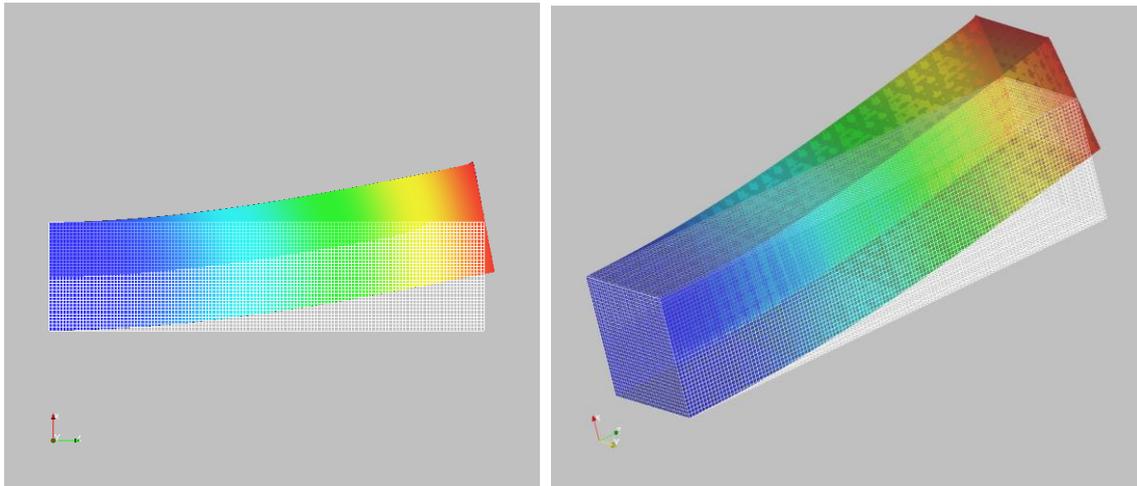


図 5.2.5 レベル 5 の計算結果

5.3 使用例 2； 2つのメッシュパーツ

5.3.1 計算条件

2つのメッシュパーツで作成した片持ち梁の形状でテストを実施した。つぎにそのメッシュの形状を示す。

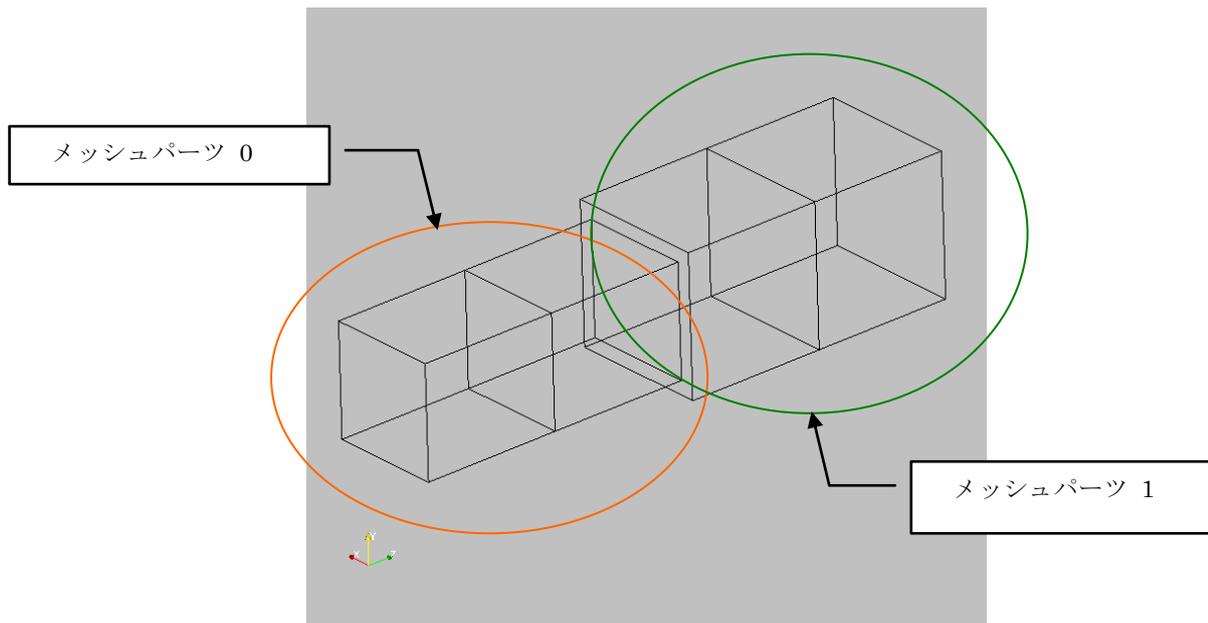


図 5.3.1 接合面で接続された2つのメッシュパーツ

5.3.2 実行結果

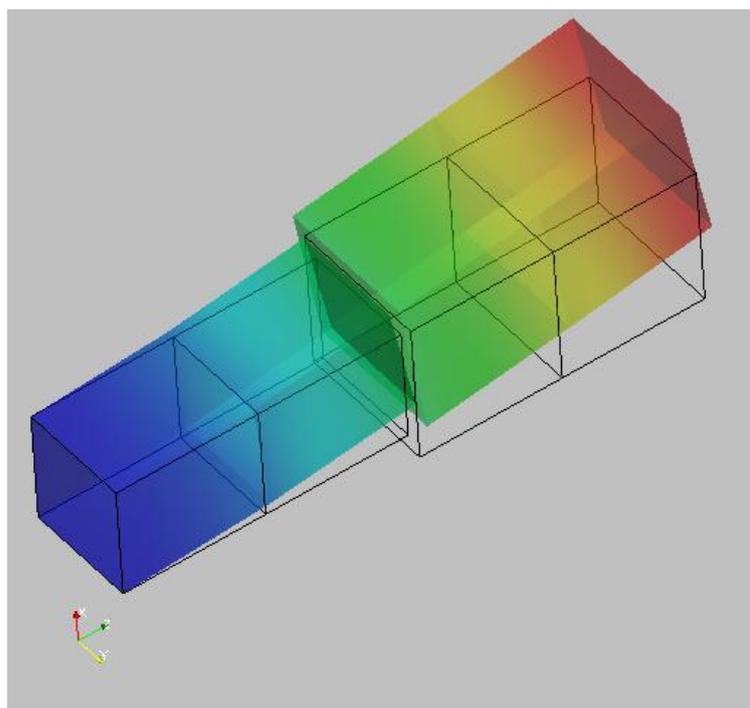


図 5.3.21 接合面で接続された2つのメッシュパーツの計算結果

5.4 使用例 3 ; マルチグリッド

5.4.1 計算条件

片持ち梁で、3 レベルでのマルチグリッドの基本的なテストを行った。各レベルのメッシュ形状と入力データを示す。

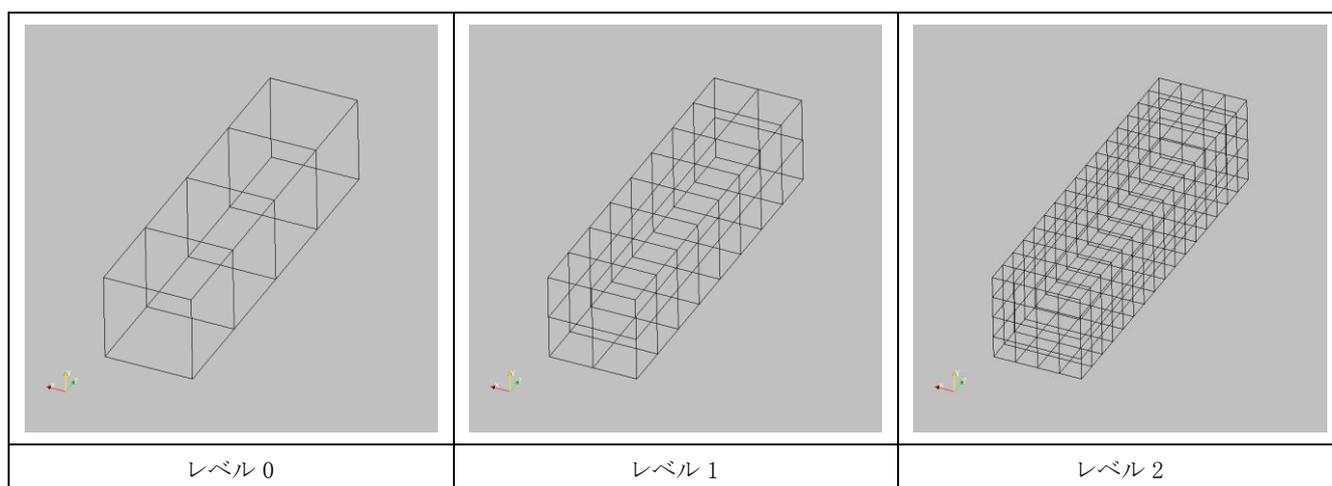


図 5.4.11 MGCG のテストで利用した各レベルの形状

5.4.2 計算結果

マルチグリッドと従来のソルバーの場合の結果を比較し、完全に一致していることを確認した。

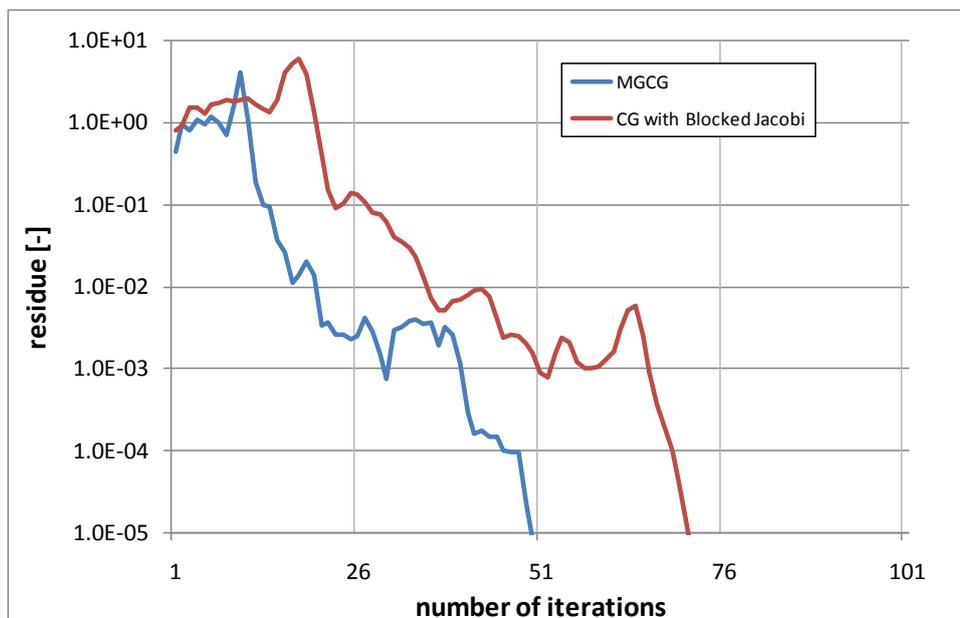


図 5.4.2 MGCG の収束状況

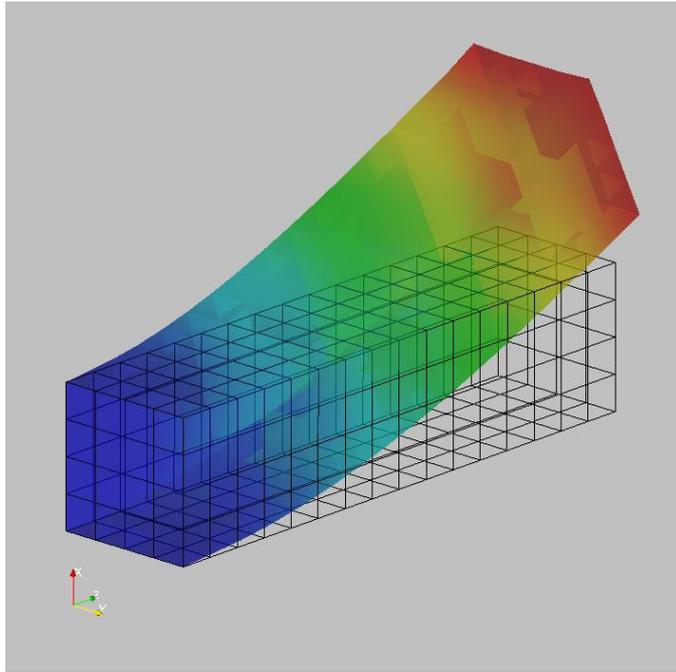


図 5.4.3 MGCG の計算結果

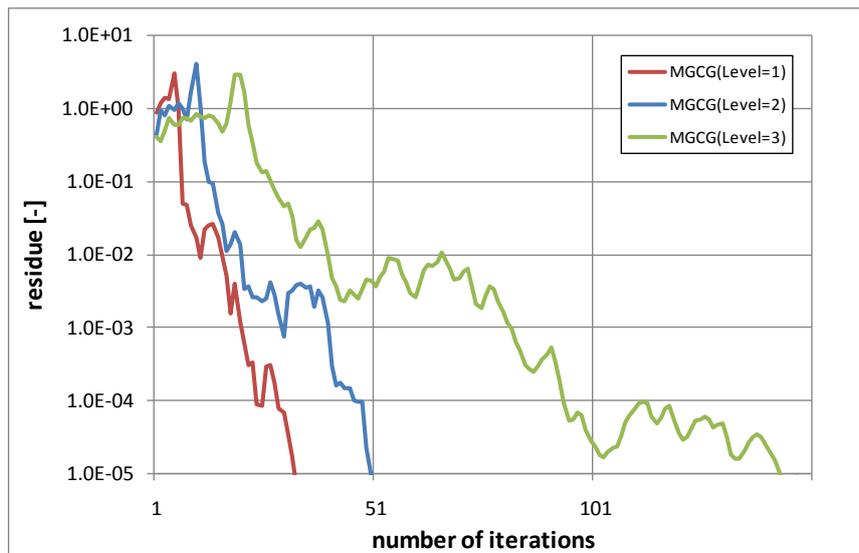


図 5.4.4 MGCG 収束状況

6. API

HECMW の API について記述する。

API は C++ と C 言語の 2 種類存在する、Fortran は C 言語の API をそのまま呼び出すだけであるので省略する。

C++ の API は全て `namespace pmw` 下の `class CMW` 下に存在する。下記の API の記述では namespace および class 名は省略する。

6.1 初期化・終了処理

HECMW の初期化処理と終了処理である。HECMW 標準の初期化関数と、FrontISTR 仕様のファイル管理を使用する場合は、初期化関数は異なる。

C 言語 API

```
iint mw_initialize(int* argc, char** argv);
```

HECMW 標準仕様の初期化 ファイル名の命名規則が fstr 仕様とは異なる。

```
iint mw_initialize_fstr(int* argc, char** argv, char* ctrlname);
```

FrontISTR 仕様の初期化

```
iint mw_finalize_0();
```

HECMW の終了処理

C++ 言語 API { namespace pmw, class CMW }

```
uuint Initialize(int argc, char** argv);
```

HECMW 標準仕様の初期化 ファイル名の命名規則が fstr 仕様とは異なる。

```
uuint Initialize_fstr(int argc, char** argv, string& ctrlname);
```

FrontISTR 仕様の初期化

```
uuint Finalize();
```

HECMW の終了処理

6.2 バナー表示

CISS コピーライトの表示

C 言語 API

```
void mw_banner_0();
```

標準出力にコピーライトを表示

C++言語 API { namespace pmw, class CMW }

```
void Banner_Display();
```

標準出力にコピーライトを表示

6.3 File 関連

メッシュファイルの入力、計算結果の出力を扱う。

HECMW 標準のメッシュファイル入力では、HECMW の命名規則に則り拡張子が判別されるが、fstr タイプでは、FrontISTR の命名規則に則り拡張子が判別される。

C 言語 API

```
iint mw_file_read(char* basename);
```

メッシュファイルの読み込み。標準ファイル管理仕様

```
iint mw_file_read_fstr();
```

メッシュファイルの読み込み。FrontISTR ファイル管理仕様

```
iint mw_file_debug_write();
```

メッシュファイルのデータチェック用ファイル出力

```
iint mw_get_fstr_filename_length_mesh();
```

```
iint mw_get_fstr_filename_length_control();
```

```
iint mw_get_fstr_filename_length_result();
```

```
iint mw_get_fstr_filename_length_restart();
```

```
iint mw_get_fstr_filename_length_part_in();
```

```
iint mw_get_fstr_filename_length_part_out();
```

```
iint mw_get_fstr_filename_length_vis_mesh();
```

```
iint mw_get_fstr_filename_length_vis_in();
```

```
iint mw_get_fstr_filename_length_vis_out();
```

FrontISTR 仕様のファイル名の文字列長の取得

```
void mw_get_fstr_filename_mesh(char name[], iint* len);
```

```
void mw_get_fstr_filename_control(char name[], iint* len);
```

```
void mw_get_fstr_filename_result(char name[], iint* len);
```

```
void mw_get_fstr_filename_restart(char name[], iint* len);
```

```
void mw_get_fstr_filename_part_in(char name[], iint* len);
```

```
void mw_get_fstr_filename_part_out(char name[], iint* len);
```

```
void mw_get_fstr_filename_vis_mesh(char name[], iint* len);
```

```
void mw_get_fstr_filename_vis_in(char name[], iint* len);
```

void mw_get_fstr_filename_vis_out(char name[], iint* len);

FrontISTR 仕様のファイル名の取得

void mw_rlt_start(iint* step);

自由書式ファイル出力の初期化

iint mw_rlt_print(iint* width, char* format, ...);

自由書式ファイル出力

void mw_rlt_end();

自由書式ファイル出力の終了

void mw_print_avs_basis(iint* ieq);

基礎変数の MicroAVS 書式ファイル出力

void mw_print_avs_fem();

登録変数の MicroAVS 書式ファイル出力

void mw_rec_avs_label(iint* imesh, char* label, char* unit, iint* ndof);

MicroAVS 書式ファイルのラベル登録

void mw_rec_avs_variable(iint* imesh, iint* num_of_node, char* label, double* value);

MicroAVS 書式ファイルの出力値登録

iint mw_file_write_res(iint* step);

リスタートファイル出力

iint mw_set_restart(iint* step);

リスタートファイルの値をソリューションベクトルへセット

C++言語 API { namespace pmw, class CMW }

uuint FileRead(string& basename, bool bBinary);

メッシュファイルの読み込み。標準ファイル管理仕様

uuint FileRead_fstr(bool bBinary);

メッシュファイルの読み込み。FrontISTR ファイル管理仕様

uuint FileDebugWrite();

メッシュファイルのデータチェック用ファイル出力

uuint FileWriteRes(const uuint& nStep, bool bBinary);

リスタートファイル出力

uuint SetRestart(const uuint& nStep, bool bBinary);

リスタートファイルの値をソリューションベクトルへセット

void PrintRlt_Start(const uuint& nStep, bool bBinary);

自由書式ファイル出力の初期化

```
void PrintRlt_P(const uiint& width, const char* format, ... );
```

```
void PrintRlt_R(const uiint& width, const char* format, ... );
```

自由書式ファイル出力

```
void PrintRlt_End();
```

自由書式ファイル出力の終了

```
void PrintMicroAVS_Basis(const uiint& ieq);
```

基礎変数の MicroAVS 書式ファイル出力

```
void PrintMicroAVS_FEM();
```

登録変数の MicroAVS 書式ファイル出力

```
void recAVS_Label(const uiint& iMesh, char* cLabel, char* cUnit, const uiint& nNumOfDOF);
```

MicroAVS 書式ファイルのラベル登録

```
void recAVS_Variable(const uiint& iMesh, const uiint& nNumOfNode, char* cLabel, double* pvValue);
```

MicroAVS 書式ファイルの出力値登録

```
string getFstr_FileName_Mesh();
```

```
string getFstr_FileName_Control();
```

```
string getFstr_FileName_Result();
```

```
string getFstr_FileName_Restart();
```

```
string getFstr_FileName_PartIn();
```

```
string getFstr_FileName_PartOut();
```

```
string getFstr_FileName_VisMesh();
```

```
string getFstr_FileName_VisIn();
```

```
string getFstr_FileName_VisOut();
```

FrontISTR 仕様のファイル名の取得。

6.4 線形代数方程式管理

C 言語 API

```
void mw_gene_linear_algebra(iint* num_of_algebra, iint dof[]);
```

線型方程式を構成する行列と二つの列ベクトルを生成する、つまり $\mathbf{Ax}=\mathbf{b}$ の生成である。

```
void mw_select_algebra(iint* ieq);
```

生成されている線型方程式の選択を行う。ここで選択された線型方程式に対して線形ソルバーの実行、剛性行列の足し込みが行われる。

C++ API { namespace pmw, class CMW }

```
void GeneLinearAlgebra(const uint& nNumOfAlgebra, uint* vDOF);
```

線型方程式を構成する行列と二つの列ベクトルを生成する、つまり $Ax=b$ の生成である。

```
void SelectAlgebra(const uint& iequ);
```

生成されている線型方程式の選択を行う。ここで選択された線型方程式に対して線形ソルバーの実行、剛性行列の足し込みが行われる。

6.5 線形代数方程式の行列・ベクトル

C 言語 API

```
iint mw_matrix_add_elem(iint* imesh, iint* ielem, double elem_matrix[]);
```

要素剛性行列の全体剛性行列への足し込み

```
iint mw_matrix_add_node(iint* imesh, iint* i_index, iint* j_index, double nodal_matrix[]);
```

節点剛性の全体剛性行列への足し込み

```
void mw_matrix_clear(iint* imesh);
```

全体剛性行列のクリア

```
void mw_vector_clear(iint* imesh);
```

列ベクトル(解ベクトル、右辺ベクトル)のクリア

```
iint mw_matrix_add_elem_24(iint* imesh, iint* ielem, double elem_matrix[][24]);
```

```
iint mw_matrix_add_elem_60(iint* imesh, iint* ielem, double elem_matrix[][60]);
```

```
iint mw_matrix_add_elem_12(iint* imesh, iint* ielem, double elem_matirx[][12]);
```

```
iint mw_matrix_add_elem_30(iint* imesh, iint* ielem, double elem_matirx[][30]);
```

```
iint mw_matrix_add_elem_18(iint* imesh, iint* ielem, double elem_matirx[][18]);
```

```
iint mw_matirx_add_elem_45(iint* imesh, iint* ielem, double elem_matirx[][45]);
```

```
iint mw_matirx_add_elem_20(iint* imesh, iint* ielem, double elem_matirx[][20]);
```

```
iint mw_matrix_add_elem_40(iint* imesh, iint* ielem, double elem_matirx[][40]);
```

```
iint mw_matrix_add_elem_15(iint* imesh, iint* ielem, double elem_matirx[][15]);
```

```
iint mw_matirx_add_elem_9(iint* imesh, iint* ielem, double elem_matirx[][9]);
```

```
iint mw_matirx_add_elem_48(iint* imesh, iint* ielem, double elem_matirx[][48]);
```

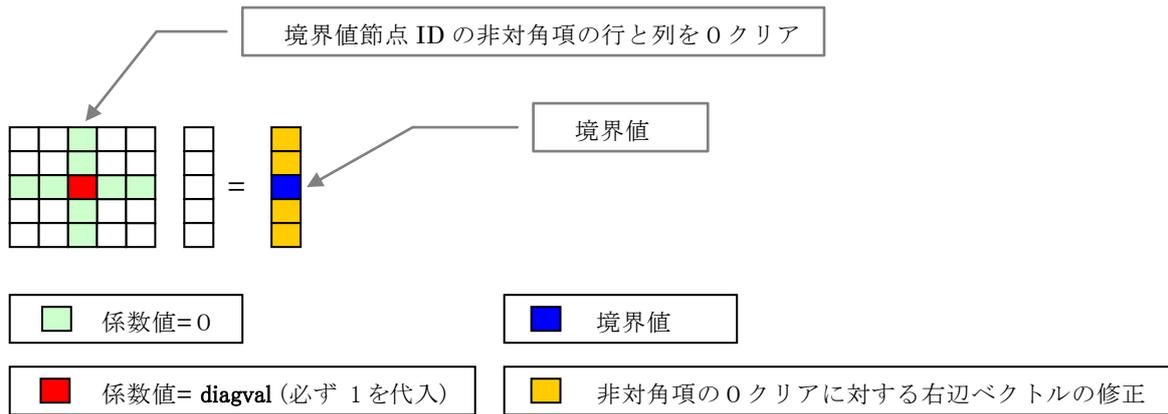
```
iint mw_matirx_add_elem_6(iint* imesh, iint* ielem, double elem_matirx[][6]);
```

```
iint mw_matirx_add_elem_10(iint* imesh, iint* ielem, double elem_matirx[][10]);
```

要素剛性行列の全体剛性行列への足し込み、二重配列バージョン

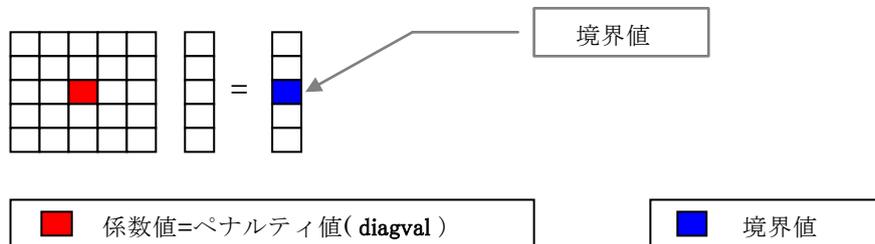
```
iint mw_matrix_rhs_set_bc2(iint* imesh, iint* inode, iint* dof, double* diagval, double* solval);
```

直接代入によるディレクレ境界値のセット、下図参照。



```
iint mw_matrix_rhs_set_bc(iint* imesh, iint* inode, iint* dof, double* diagval, double* rhsval);
```

ペナルティ法によるディレクレ境界値のセット、下図参照。



```
iint mw_rhs_set_bc(iint* imesh, iint* inode, iint* dof, double* value);
```

右辺ベクトルへ境界値をセット

```
iint mw_rhs_add_bc(iint* imesh, iint* inode, iint* dof, double* value);
```

右辺ベクトルへ値を加算

```
void mw_get_solution_vector(double buf[], iint* imesh);
```

解ベクトルの取得

```
void mw_get_solution_assy_vector(double buf[]);
```

アセンブル解ベクトルの取得

```
void mw_get_rhs_vector(double buf[], iint* imesh);
```

右辺ベクトルの取得

```
void mw_get_rhs_assy_vector(double buf[]);
```

アセンブル右辺ベクトルの取得

```
double mw_get_solution_assy_vector_val(iint* imesh, iint* inode, iint* idof);
```

アセンブル解ベクトルの inode, idof の値の取得

```
double mw_get_rhs_assy_vector_val(iint* imesh, iint* inode, iint* idof);
```

アセンブル右辺ベクトルの inode, idof の値の取得

```
iint mw_get_solution_assy_vector_dof();
```

アセンブル解ベクトルの自由度数の取得

```
iint mw_get_rhs_assy_vector_dof();
```

アセンブル右辺ベクトルの自由度数の取得


```

void GetSolution_AssyVector(double* buf);
    アセンブル解ベクトルの取得
void GetRHS_Vector(double* buf, const uiint& imesh);
    右辺ベクトルの取得
void GetRHS_AssyVector(double* buf);
    アセンブル右辺ベクトルの取得
double& GetSolutionVector_Val(const uiint& imesh, const uiint& inode, const uiint& idof);
    アセンブル解ベクトルの inode, idof の値の取得
double& GetRHSVector_Val(const uiint& imesh, const uiint& inode, const uiint& idof);
    アセンブル右辺ベクトルの inode, idof の値の取得
uiint& GetSolutionVector_DOFF();
    アセンブル解ベクトルの自由度数の取得
uiint& GetRHSVector_DOFF();
    アセンブル右辺ベクトルの自由度数の取得
void multVector(double* vX, double* vB);
    行列と引数vB 列ベクトルの積vX の取得

```

6.6 線形代数ソルバー

線形ソルバーの実行

C 言語 API

```

iint mw_solve(iint* iter_max, double* tolerance, iint* method, iint* pre_condition);
method = 1:CG, 2:BiCBSTAB, 3:GPBiCG, 4:GMRES
pre_condition = 1:Jacobi, 2:MG, 3:SSOR, 4:ILU

```

C++言語 API { namespace pmw, class CMW }

```

uiint Solve(iint& iter_max, double& tolerance, iint& method, iint& precondition);
method = 1:CG, 2:BiCBSTAB, 3:GPBiCG, 4:GMRES
precondition = 1:Jacobi, 2:MG, 3:SSOR, 4:ILU

```

6.7 階層メッシュ構造の構築

マルチグリッド階層構造を構築する関数である。

階層数がゼロであっても、メッシュ内部関係を構築するために呼び出す必要がある。

※必須関数

C 言語 API

```
iint mw_refine(int* num_of_refine);
```

入力メッシュを指定の階層数多階層化する。

```
iint mw_mg_construct(int* num_of_refine);
```

mw_refine と同一関数。

```
void mw_finalize_refine();
```

階層構造の構築で使用した隣接要素などをメモリーから解放する。

```
void mw_finalize_mg_construct();
```

mw_finalize_refine と同一関数。

C++言語 API { namespace pmw, class CMW }

```
uiint Refine(const uiint& nNumOfRefine);
```

入力メッシュを指定の階層数多階層化する。

```
void FinalizeRefine();
```

階層構造の構築で使用した隣接要素などをメモリーから解放する。

6.8 メッシュデータ・アクセス

HECMW のメッシュデータにアクセスするための関数について説明する。

メッシュデータ API を利用して HECMW 内部のメッシュデータにアクセスする手順を大まかに記すと、選択順序は；

1. Assemble Model(階層構造から必要な階層(Level)のアセンブル・モデルを取得)
2. Mesh (Assemble Model から、必要な Mesh パーツを取得)
3. Element (Mesh から、必要な要素を取得)
4. Node (要素の構成 NodeID から Node を取得)

のように呼び出すことで、メッシュのデータにアクセスすることができる。

C 言語 API

```
iint mw_get_num_of_assemble_model();
```

Assemble Model の個数つまり、階層数(mMGLLevel+1)の取得する関数である。

```
void mw_select_assemble_model(iint* mglevel);
```

Assemble Model の選択をする関数である。引数は階層レベル番号である。

```
iint mw_get_num_of_mesh_part();
```

選択されている Assemble Model の Mesh パーツの個数を取得する関数である。

```
void mw_select_mesh_part_with_id(iint* mesh_id);
```

選択されている Assemble Model の Mesh パーツの選択をする関数である。

引数はメッシュパーツ ID 番号である。

```
void mw_select_mesh_part(iint* index);
```

選択されている Assemble Model の Mesh パーツの選択をする関数である。

引数は Assemble Mode に存在する Mesh パーツの配列インデックス番号である。

void mw_select_element_with_id(iint* elem_id);

選択されている Mesh パーツの要素の選択をする関数である。引数は要素 ID 番号である。

void mw_select_element(iint* index);

選択されている Mesh パーツの要素の選択をする関数である。

引数は Mesh パーツに存在する要素の配列インデックス番号である。

iint mw_get_element_type();

要素タイプの取得をする関数である。

iint mw_get_num_of_element_vert();

選択されている要素の頂点数を取得する関数である。

void mw_get_element_vert_node_id(iint v_node_id[]);

選択されている要素の頂点のノード ID を取得する関数である。引数 v_node_id[] に代入される。

iint mw_get_num_of_element_edge();

選択されている要素の辺数を取得する関数である。

void mw_get_element_edge_node_id(iint v_node_id[]);

選択されている要素の辺のノード ID を取得する関数である。引数 v_node_id[] に代入される。

void mw_get_node_coord(iint* node_id, double* x, double* y, double* z);

選択された Mesh 内の節点座標を取得する関数である。引数 x, y, z に代入される。

iint mw_get_num_of_node();

選択されている Mesh パーツの節点数を取得する関数である。

iint mw_get_num_of_node_with_mesh(iint* imesh);

選択されている Mesh パーツの節点数を取得する関数である。

iint mw_get_num_of_element();

選択されている Mesh パーツの要素数を取得する関数である。

iint mw_get_num_of_element_with_mesh(iint* imesh);

選択されている Mesh パーツの要素数を取得する関数である。

iint mw_get_node_id(iint* index);

配列インデックス番号から節点の ID 番号を取得する

iint mw_get_element_id(iint* index);

配列インデックス番号から要素の ID 番号を取得する

iint mw_get_node_index(iint* id);

節点 ID 番号から配列インデックス番号を取得する

iint mw_get_element_index(iint* id);

要素 ID 番号から配列インデックス番号を取得する

iint mw_get_num_of_aggregate_element(iint* node_id);

指定節点 ID の周囲に接続する要素数を取得する

iiint mw_get_aggregate_element_id(iiint* node_id, iiint* index);

指定節点 ID の周囲に接続する要素 ID を取得する

C++言語 API { namespace pmw, class CMW }

uiint GetNumOfAssembleModel();

Assemble Model の個数つまり、階層数(mMGLevel+1)の取得する関数である。

void SelectAssembleModel(const uiint& mgLevel);

Assemble Model の選択をする関数である。引数は階層レベル番号である。

uiint GetNumOfMeshPart();

選択されている Assemble Model の Mesh パーツの個数を取得する関数である。

void SelectMeshPart_ID(const uiint& mesh_id);

選択されている Assemble Model の Mesh パーツの選択をする関数である。

引数はメッシュパーツ ID 番号である。

void SelectMeshPart_IX(const uiint& index);

選択されている Assemble Model の Mesh パーツの選択をする関数である。

引数は Assemble Mode に存在する Mesh パーツの配列インデックス番号である。

void SelectElement_ID(const uiint& elem_id);

選択されている Mesh パーツの要素の選択をする関数である。引数は要素 ID 番号である。

void SelectElement_IX(const uiint& index);

選択されている Mesh パーツの要素の選択をする関数である。

引数は Mesh パーツに存在する要素の配列インデックス番号である。

uiint GetElementType();

要素タイプの取得をする関数である。

uiint GetNumOfElementVert();

選択されている要素の頂点数を取得する関数である。

void GetElementVertNodeID(iiint* vNodeID);

選択されている要素の頂点のノード ID を取得する関数である。引数 vNodeID に代入される。

uiint GetNumOfElementEdge();

選択されている要素の辺数を取得する関数である。

void GetElementEdgeNodeID(iiint* vNodeID);

選択されている要素の辺のノード ID を取得する関数である。引数 vNodeID に代入される。

void GetNodeCoord(const uiint& node_id, double& x, double& y, double& z);

選択された Mesh 内の節点座標を取得する関数である。引数 x, y, z に代入される。

uiint getNodeSize();

選択されている Mesh パーツの節点数を取得する関数である。

uiint getElementSize();

選択されている Mesh パーツの要素数を取得する関数である。

uiint getNodeSize(uiint iMesh);

選択されている Mesh パーツの節点数を取得する関数である。

```
uiint getElementSize(uiint iMesh);
```

選択されている Mesh パーツの要素数を取得する関数である。

```
uiint& getNodeID(const uiint& index);
```

配列インデックス番号から節点の ID 番号を取得する

```
uiint& getElementID(const uiint& index);
```

配列インデックス番号から要素の ID 番号を取得する

```
uiint& getNodeIndex(const uiint& id);
```

節点 ID 番号から配列インデックス番号を取得する

```
uiint& getElementIndex(const uiint& id);
```

要素 ID 番号から配列インデックス番号を取得する

```
uiint getNumOfAggregateElement(const uiint& node_id);
```

指定節点 ID の周囲に接続する要素数を取得する

```
uiint& getAggregateElementID(const uiint& node_id, const uiint& ielem);
```

指定節点 ID の周囲に接続する要素 ID を取得する

6.9 要素タイプ

要素形状のタイプ番号を返す。

C 言語 API

```
iint mw_elemtype_hexa_0();
```

六面体 1 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_hexa2_0();
```

六面体 2 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_tetra_0();
```

四面体 1 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_tetra2_0();
```

四面体 2 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_prism_0();
```

プリズム 1 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_prism2_0();
```

プリズム 2 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_quad_0();
```

四辺形 1 次要素形状を現す要素形状番号を返す

```
iint mw_elemtype_quad2_0();
```

四辺形 2 次要素形状を現す要素形状番号を返す

```

iint mw_elemtypе_triangle_0();
    三角形 1 次要素形状を現す要素形状番号を返す
iint mw_elemtypе_triangle2_0();
    三角形 2 次要素形状を現す要素形状番号を返す
iint mw_elemtypе_line_0();
    線 1 次要素形状を現す要素形状番号を返す
iint mw_elemtypе_line2_0();
    線 2 次要素形状を現す要素形状番号を返す
iint mw_fistr_elemtypе_hexa_0();
    六面体 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_hexa2_0();
    六面体 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_tetra_0();
    四面体 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_tetra2_0();
    四面体 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_prism_0();
    プリズム 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_prism2_0();
    プリズム 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_quad_0();
    四辺形 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_quad2_0();
    四辺形 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_triangle_0();
    三角形 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_triangle2_0();
    三角形 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_line_0();
    線 1 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_line2_0();
    線 2 次要素形状を現す FrontISTR 要素形状番号を返す
iint mw_fistr_elemtypе_to_mw3_elemtypе_(iint* fistr_elemtypе);
    FrontISTR 要素形状番号から HECMW 要素形状番号を取得
iint mw_mw3_elemtypе_to_fistr_elemtypе_(iint* mw3_elemtypе);
    HECMW 要素形状番号から FrontISTR 要素形状番号を取得

```

C++言語 API { namespace pmw, class CMW }

uint elemtype_hexa();

六面体 1 次要素形状を現す要素形状番号を返す

uint elemtype_hexa2();

六面体 2 次要素形状を現す要素形状番号を返す

uint elemtype_tetra();

四面体 1 次要素形状を現す要素形状番号を返す

uint elemtype_tetra2();

四面体 2 次要素形状を現す要素形状番号を返す

uint elemtype_prism();

プリズム 1 次要素形状を現す要素形状番号を返す

uint elemtype_prism2();

プリズム 2 次要素形状を現す要素形状番号を返す

uint elemtype_quad();

四辺形 1 次要素形状を現す要素形状番号を返す

uint elemtype_quad2();

四辺形 2 次要素形状を現す要素形状番号を返す

uint elemtype_triangle();

三角形 1 次要素形状を現す要素形状番号を返す

uint elemtype_triangle2();

三角形 2 次要素形状を現す要素形状番号を返す

uint elemtype_line();

線 1 次要素形状を現す要素形状番号を返す

uint elemtype_line2();

線 2 次要素形状を現す要素形状番号を返す

uint fistr_elemtype_hexa();

六面体 1 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_hexa2();

六面体 2 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_tetra();

四面体 1 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_tetra2();

四面体 2 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_prism();

プリズム 1 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_prism2();

プリズム 2 次要素形状を現す FrontISTR 要素形状番号を返す

uint fistr_elemtype_quad();

四辺形 1 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_quad2();

四辺形 2 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_triangle();

三角形 1 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_triangle2();

三角形 2 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_line();

線 1 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_line2();

線 2 次要素形状を現す FrontISTR 要素形状番号を返す

uuint fistr_elemtype_to_mw3_elemtype(const uuint& fistr_elemtype);

FrontISTR 要素形状番号から HECMW 要素形状番号を取得

uuint mw3_elemtype_to_fistr_elemtype(const uuint& mw3_elemtype);

HECMW 要素形状番号から FrontISTR 要素形状番号を取得

6.10 形状関数

HECMW が所有する形状関数を取得する関数である。

C 言語 API

iint mw_get_num_of_integ_point(iint* shape_type);

形状関数種類別の積分点数の取得関数である。

void mw_shape_function_on_pt(iint* shape_type, iint* igauss, double N[]);

引数で指定された形状関数タイプの積分点での形状関数を取得する。

積分点における全ての形状関数が配列で返される。

void mw_shape_function_hexa81(iint* igauss, iint* ishape, double* N);

六面体 1 次・積分点数 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_hexa82(iint* igauss, iint* ishape, double* N);

六面体 1 次・積分点数 8 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_hexa201(iint* igauss, iint* ishape, double* N);

六面体 2 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_hexa202(iint* igauss, iint* ishape, double* N);

六面体 2 次・積分点 8 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_hexa203(iint* igauss, iint* ishape, double* N);

六面体 2 次・積分点 27 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tetra41(iint* igauss, iint* ishape, double* N);

四面体 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tetra101(iint* igauss, iint* ishape, double* N);

四面体 2 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tetra104(iint* igauss, iint* ishape, double* N);

四面体 2 次・積分点 4 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tetra1015(iint* igauss, iint* ishape, double* N);

四面体 2 次・積分点 15 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_prism62(iint* igauss, iint* ishape, double* N);

プリズム 1 次・積分点 2 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_prism156(iint* igauss, iint* ishape, double* N);

プリズム 2 次・積分点 6 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_prism159(iint* igauss, iint* ishape, double* N);

プリズム 2 次・積分点 9 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_prism1518(iint* igauss, iint* ishape, double* N);

プリズム 2 次・積分点 18 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_quad41(iint* igauss, iint* ishape, double* N);

四辺形 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_quad84(iint* igauss, iint* ishape, double* N);

四辺形 1 次・積分点 4 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_quad89(iint* igauss, iint* ishape, double* N);

四辺形 2 次・積分点 9 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tri31(iint* igauss, iint* ishape, double* N);

三角形 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_tri63(iint* igauss, iint* ishape, double* N);

三角形 2 次・積分点 3 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_line21(iint* igauss, iint* ishape, double* N);

線 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_shape_function_line32(iint* igauss, iint* ishape, double* N);

線 2 次・積分点 2 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void mw_dndr(iint* shape_type, double dndr[]);

自然座標勾配 dN/dr を一括で取得する。dNdr は、一列の配列である。

void mw_dndr_hexa81(iint* igauss, iint* ishape, iint* iaxis, double* dndr);

六面体 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

void mw_dndr_hexa82(iint* igauss, iint* ishape, iint* iaxis, double* dndr);

六面体 1 次・積分点数 8 の形状関数の自然座標での勾配を取得する

void mw_dndr_hexa201(iint* igauss, iint* ishape, iint* iaxis, double* dndr);

六面体 2 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_hexa202(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

六面体 2 次・積分点数 8 の形状関数の自然座標での勾配を取得する

`void mw_dndr_hexa203(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

六面体 2 次・積分点数 27 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tetra41(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四面体 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tetra101(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四面体 2 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tetra104(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四面体 2 次・積分点数 4 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tetra1015(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四面体 2 次・積分点数 15 の形状関数の自然座標での勾配を取得する

`void mw_dndr_prism62(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

プリズム 1 次・積分点数 2 の形状関数の自然座標での勾配を取得する

`void mw_dndr_prism156(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

プリズム 2 次・積分点数 6 の形状関数の自然座標での勾配を取得する

`void mw_dndr_prism159(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

プリズム 2 次・積分点数 9 の形状関数の自然座標での勾配を取得する

`void mw_dndr_prism1518(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

プリズム 2 次・積分点数 18 の形状関数の自然座標での勾配を取得する

`void mw_dndr_quad41(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四辺形 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_quad84(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四辺形 2 次・積分点数 4 の形状関数の自然座標での勾配を取得する

`void mw_dndr_quad89(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

四辺形 2 次・積分点数 9 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tri31(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

三角形 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_tri63(iint* igauss, iint* ishape, iint* iaxis, double* dndr);`

三角形 2 次・積分点数 3 の形状関数の自然座標での勾配を取得する

`void mw_dndr_line21(iint* igauss, iint* ishape, double* dndr);`

線 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`void mw_dndr_line32(iint* igauss, iint* ishape, double* dndr);`

線 2 次・積分点数 2 の形状関数の自然座標での勾配を取得する

`void mw_dndx(iint* elem_type, iint* num_of_integ, iint* ielem, double dndx[]);`

指定した要素 Index 番号の要素について空間座標での導関数を計算し、導関数を一括取得する。値は引数 dNdx に代入される。dNdx は一列の配列である。

※要素タイプと積分点数を指定することで形状関数を指定する。

void mw_det_jacobian(iint* elem_type, iint* num_of_integ, iint* igauss, double* det_j);

Jacobian 行列式を取得する関数である。引数 detJ に代入される。

この関数を呼び出す直前に使用した dNdx 計算の detJ の値が代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

void mw_weight(iint* elem_type, iint* num_of_integ, iint* igauss, double* w);

Gauss 積分点の重みを取得する関数である。引数 w に代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

iint mw_shapetype_hexa81_0();

形状関数 Hexa81 のタイプ番号を返す。

iint mw_shapetype_hexa82_0();

形状関数 Hexa82 のタイプ番号を返す。

iint mw_shapetype_hexa201_0();

形状関数 Hexa201 のタイプ番号を返す。

iint mw_shapetype_hexa202_0();

形状関数 Hexa202 のタイプ番号を返す。

iint mw_shapetype_hexa203_0();

形状関数 Hexa203 のタイプ番号を返す。

iint mw_shapetype_tetra41_0();

形状関数 Tetra41 のタイプ番号を返す。

iint mw_shapetype_tetra101_0();

形状関数 Tetra101 のタイプ番号を返す。

iint mw_shapetype_tetra104_0();

形状関数 Tetra104 のタイプ番号を返す。

iint mw_shapetype_tetra1015_0();

形状関数 Tetra1015 のタイプ番号を返す。

iint mw_shapetype_prism62_0();

形状関数 Prism62 のタイプ番号を返す。

iint mw_shapetype_prism156_0();

形状関数 Prism156 のタイプ番号を返す。

iint mw_shapetype_prism159_0();

形状関数 Prism159 のタイプ番号を返す。

iint mw_shapetype_prism1518_0();

形状関数 Prism1518 のタイプ番号を返す。

iint mw_shapetype_quad41_0();

形状関数 Quad41 のタイプ番号を返す。

```
iint mw_shapetype_quad84_0);
```

形状関数 Quad84 のタイプ番号を返す。

```
iint mw_shapetype_quad89_0);
```

形状関数 Quad89 のタイプ番号を返す。

```
iint mw_shapetype_tri31_0);
```

形状関数 Triangle31 のタイプ番号を返す。

```
iint mw_shapetype_tri63_0);
```

形状関数 Triangle63 のタイプ番号を返す。

```
iint mw_shapetype_line21_0);
```

形状関数 Line21 のタイプ番号を返す。

```
iint mw_shapetype_line32_0);
```

形状関数 Line32 のタイプ番号を返す。

C++言語 API { namespace pmw, class CMW }

```
uiint& NumOfIntegPoint(const uiint& shapeType);
```

形状関数種類別の積分点数の取得関数である。

```
void ShapeFunc_on_pt(const uiint& shapeType, const uiint& igauss, vdouble& N);
```

引数で指定された形状関数タイプの積分点での形状関数を取得する。

積分点における全ての形状関数が配列で返される。

```
void ShapeFunc_on_pt(uiint shapeType, uiint igauss, double N[]);
```

積分点での形状関数を取得する。積分点における全ての形状関数が配列で返される。

```
void ShapeFunc(const uiint& shapeType, vdouble& N);
```

引数で指定された形状関数タイプについて、全積分点の形状関数を一括で取得する、[積分点]-[形状関数番号]の順序の 2 重配列である。

```
double& ShapeFunc_Hexa81(uiint igauss, uiint ishape);
```

六面体 1 次・積分点数 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

```
double& ShapeFunc_Hexa82(uiint igauss, uiint ishape);
```

六面体 1 次・積分点数 8 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

```
double& ShapeFunc_Hexa201(uiint igauss, uiint ishape);
```

六面体 2 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

```
double& ShapeFunc_Hexa202(uiint igauss, uiint ishape);
```

六面体 2 次・積分点 8 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

```
double& ShapeFunc_Hexa203(uiint igauss, uiint ishape);
```

六面体 2 次・積分点 27 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Tetra41(uiint igauss, uiint ishape);

四面体 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Tetra101(uiint igauss, uiint ishape);

四面体 2 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Tetra104(uiint igauss, uiint ishape);

四面体 2 次・積分点 4 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Tetra1015(uiint igauss, uiint ishape);

四面体 2 次・積分点 15 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Prism62(uiint igauss, uiint ishape);

プリズム 1 次・積分点 2 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Prism156(uiint igauss, uiint ishape);

プリズム 2 次・積分点 6 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Prism159(uiint igauss, uiint ishape);

プリズム 2 次・積分点 9 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Prism1518(uiint igauss, uiint ishape);

プリズム 2 次・積分点 18 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Quad41(uiint igauss, uiint ishape);

四辺形 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Quad84(uiint igauss, uiint ishape);

四辺形 1 次・積分点 4 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Quad89(uiint igauss, uiint ishape);

四辺形 2 次・積分点 9 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Triangle31(uiint igauss, uiint ishape);

三角形 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Triangle63(uiint igauss, uiint ishape);

三角形 2 次・積分点 3 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Line21(uiint igauss, uiint ishape);

線 1 次・積分点 1 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

double& ShapeFunc_Line32(uiint igauss, uiint ishape);

線 2 次・積分点 2 の形状関数の積分点 igauss、節点 ishape での形状関数を取得する

void dNdr_on_pt(const uiint& shapeType, const uiint& igauss, vvdouble& dNdr);

自然座標での導関数を取得する関数である。引数 dNdr に代入される。

積分点ごと dN/dr を取得する。dNdr は [形状関数]-[座標方向]の 2 重配列である。

void dNdr(const uiint& shapeType, vvdouble& dNdr);

自然座標勾配 dN/dr を一括で取得する。dNdr は [積分点]- [形状関数]-[座標方向]の 3 重配列である。

void dNdr(const uiint& shapeType, double dNdr[]);

自然座標勾配 dN/dr を一括で取得する。dNdr は、一列の配列である。

`double& dNdr_Hexa81_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
六面体 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Hexa82_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
六面体 1 次・積分点数 8 の形状関数の自然座標での勾配を取得する

`double& dNdr_Hexa201_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
六面体 2 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Hexa202_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
六面体 2 次・積分点数 8 の形状関数の自然座標での勾配を取得する

`double& dNdr_Hexa203_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
六面体 2 次・積分点数 27 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tetra41_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四面体 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tetra101_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四面体 2 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tetra104_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四面体 2 次・積分点数 4 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tetra1015_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四面体 2 次・積分点数 15 の形状関数の自然座標での勾配を取得する

`double& dNdr_Prism62_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
プリズム 1 次・積分点数 2 の形状関数の自然座標での勾配を取得する

`double& dNdr_Prism156_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
プリズム 2 次・積分点数 6 の形状関数の自然座標での勾配を取得する

`double& dNdr_Prism159_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
プリズム 2 次・積分点数 9 の形状関数の自然座標での勾配を取得する

`double& dNdr_Prism1518_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
プリズム 2 次・積分点数 18 の形状関数の自然座標での勾配を取得する

`double& dNdr_Quad41_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四辺形 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Quad84_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四辺形 2 次・積分点数 4 の形状関数の自然座標での勾配を取得する

`double& dNdr_Quad89_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
四辺形 2 次・積分点数 9 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tri31_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
三角形 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

`double& dNdr_Tri63_on_pt_on_shape(uiint igauss, uiint ishape, uiint iaxis);`
三角形 2 次・積分点数 3 の形状関数の自然座標での勾配を取得する

`double& dNdr_Line21_on_pt_on_shape(uiint igauss, uiint ishape);`
線 1 次・積分点数 1 の形状関数の自然座標での勾配を取得する

double& dNdr_Line32_on_pt_on_shape(uiint igauss, uiint ishape);

線 2 次・積分点数 2 の形状関数の自然座標での勾配を取得する

void Calculate_dNdx(const uiint& elemType, const uiint& numOfInteg, const uiint& elem_index);

空間座標での導関数を計算-取得する関数である。

指定した要素 Index 番号の要素について空間座標での導関数を計算する。

※要素タイプと積分点数を指定することで形状関数を指定する。

void dNdx_on_pt(const uiint& igauss, vvdoubles& dNdx);

直前に計算された積分点の導関数を取得する。値は引数 dNdx に代入される。[形状関数]-[座標方向]の 2 重配列である。

※dNdx_on_pt()は、Calculate_dNdx(...)を実行後に使用する必要がある。

void dNdx(const uiint& elemType, const uiint& numOfInteg, const uiint& elem_index, vvdoubles& dNdx);

指定した要素 Index 番号の要素について空間座標での導関数を計算し、導関数を一括取得する。値は引数 dNdx に代入される。dNdx は[積分点]- [形状関数]-[座標方向]の 3 重配列である。

※要素タイプと積分点数を指定することで形状関数を指定する。

void dNdx(const uiint& elemType, const uiint& numOfInteg, const uiint& ielem, double dNdx[]);

指定した要素 Index 番号の要素について空間座標での導関数を計算し、導関数を一括取得する。値は引数 dNdx に代入される。dNdx は一列の配列である。

※要素タイプと積分点数を指定することで形状関数を指定する。

void detJacobian(const uiint& elemType, const uiint& numOfInteg, const uiint& igauss, double& detJ);

Jacobian 行列式を取得する関数である。引数 detJ に代入される。

この関数を呼び出す直前に使用した dNdx 計算の detJ の値が代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

void Weight(const uiint& elemType, const uiint& numOfInteg, const uiint& igauss, double& w);

Gauss 積分点の重みを取得する関数である。引数 w に代入される。

※要素タイプと積分点数を指定することで形状関数を指定する。

uiint shapetype_hexa81();

形状関数 Hexa81 のタイプ番号を返す。

uiint shapetype_hexa82();

形状関数 Hexa82 のタイプ番号を返す。

uiint shapetype_hexa201();

形状関数 Hexa201 のタイプ番号を返す。

uiint shapetype_hexa202();

形状関数 Hexa202 のタイプ番号を返す。

uiint shapetype_hexa203();

形状関数 Hexa203 のタイプ番号を返す。

uiint shapetype_tetra41();

形状関数 Tetra41 のタイプ番号を返す。

uiint shapetype_tetra101();

形状関数 Tetra101 のタイプ番号を返す。

uiint shapetype_tetra104();

形状関数 Tetra104 のタイプ番号を返す。

uiint shapetype_tetra1015();

形状関数 Tetra1015 のタイプ番号を返す。

uiint shapetype_prism62();

形状関数 Prism62 のタイプ番号を返す。

uiint shapetype_prism156();

形状関数 Prism156 のタイプ番号を返す。

uiint shapetype_prism159();

形状関数 Prism159 のタイプ番号を返す。

uiint shapetype_prism1518();

形状関数 Prism1518 のタイプ番号を返す。

uiint shapetype_quad41();

形状関数 Quad41 のタイプ番号を返す。

uiint shapetype_quad84();

形状関数 Quad84 のタイプ番号を返す。

uiint shapetype_quad89();

形状関数 Quad89 のタイプ番号を返す。

uiint shapetype_tri31();

形状関数 Triangle31 のタイプ番号を返す。

uiint shapetype_tri63();

形状関数 Triangle63 のタイプ番号を返す。

uiint shapetype_line21();

形状関数 Line21 のタイプ番号を返す。

uiint shapetype_line32();

形状関数 Line32 のタイプ番号を返す。

6.11 境界条件

C 言語 API

```

iint mw_get_num_of_boundary_bnode_mesh_0();
    点グループ境界条件のグループ数
iint mw_get_num_of_boundary_bface_mesh_0();
    面グループ境界条件のグループ数
iint mw_get_num_of_boundary_bedge_mesh_0();
    辺グループ境界条件のグループ数
iint mw_get_num_of_boundary_bvolume_mesh_0();
    体積グループ境界条件のグループ数
iint mw_get_bnd_type_bnode_mesh(iint* ibmesh);
    点グループ境界条件のタイプ、ノイマン型かディレクレ型
iint mw_get_bnd_type_bface_mesh(iint* ibmesh);
    面グループ境界条件のタイプ、ノイマン型かディレクレ型
iint mw_get_bnd_type_bedge_mesh(iint* ibmesh);
    辺グループ境界条件のタイプ、ノイマン型かディレクレ型
iint mw_get_bnd_type_bvolume_mesh(iint* ibmesh);
    体積グループ境界条件のタイプ、ノイマン型かディレクレ型
iint mw_get_neumann_type_0();
    ノイマン型を表すタイプ番号
iint mw_get_dirichlet_type_0();
    ディレクレ型を表すタイプ番号
iint mw_get_num_of_bnode_in_bnode_mesh(iint* ibmesh);
    点グループ境界条件の節点数
iint mw_get_num_of_bnode_in_bface_mesh(iint* ibmesh);
    面グループ境界条件の節点数
iint mw_get_num_of_bnode_in_bedge_mesh(iint* ibmesh);
    辺グループ境界条件の節点数
iint mw_get_num_of_bnode_in_bvolume_mesh(iint* ibmesh);
    体積グループ境界条件の節点数
iint mw_get_num_of_dof_in_bnode_mesh(iint* ibmesh, iint* ibnode);
    点グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
iint mw_get_num_of_dof_in_bface_mesh(iint* ibmesh);
    面グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
iint mw_get_num_of_dof_in_bedge_mesh(iint* ibmesh);
    辺グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
iint mw_get_num_of_dof_in_bvolume_mesh(iint* ibmesh);
    体積グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
iint mw_get_dof_bnode_mesh(iint* ibmesh, iint* ibnode, iint* idof);
    点グループ境界条件の境界値の自由度番号

```

```

iint mw_get_dof_bface_mesh(iint* ibmesh, iint* idof);
    面グループ境界条件の境界値の自由度番号
iint mw_get_dof_bedge_mesh(iint* ibmesh, iint* idof);
    辺グループ境界条件の境界値の自由度番号
iint mw_get_dof_bvolume_mesh(iint* ibmesh, iint* idof);
    体積グループ境界条件の境界値の自由度番号
double mw_get_bnode_value_in_bnode_mesh(iint* ibmesh, iint* ibnode, iint* dof);
    点グループ境界条件の節点境界値
double mw_get_bnode_value_in_bface_mesh(iint* ibmesh, iint* ibnode, iint* dof, iint* mlevel);
    面グループ境界条件の節点境界値
double mw_get_bnode_value_in_bedge_mesh(iint* ibmesh, iint* ibnode, iint* dof, iint* mlevel);
    辺グループ境界条件の節点境界値
double mw_get_bnode_value_in_bvolume_mesh(iint* ibmesh, iint* ibnode, iint* dof, iint* mlevel);
    体積グループ境界条件の節点境界値
iint mw_get_node_id_in_bnode_mesh(iint* ibmesh, iint* ibnode);
    点グループ境界条件の節点 ID
iint mw_get_node_id_in_bface_mesh(iint* ibmesh, iint* ibnode);
    面グループ境界条件の節点 ID
iint mw_get_node_id_in_bedge_mesh(iint* ibmesh, iint* ibnode);
    辺グループ境界条件の節点 ID
iint mw_get_node_id_in_bvolume_mesh(iint* ibmesh, iint* ibnode);
    体積グループ境界条件の節点 ID
iint mw_get_num_of_bface(iint* ibmesh);
    面グループ境界条件の面の数
double mw_get_bface_value(iint* ibmesh, iint* ibface, iint* dof);
    面グループ境界条件の面の境界値
iint mw_get_num_of_bedge(iint* ibmesh);
    辺グループ境界条件の辺の数
double mw_get_bedge_value(iint* ibmesh, iint* ibedge, iint* dof);
    辺グループ境界条件の辺の境界値
iint mw_get_num_of_bvolume(iint* ibmesh);
    体積グループ境界条件の体積数
double mw_get_bvolume_value(iint* ibmesh, iint* ibvol, iint* dof);
    体積グループ境界条件の体積中心境界値
iint mw_get_num_of_node_bface(iint* ibmesh, iint* ibface);
    面グループ境界の面の節点数
iint mw_get_node_id_bface(iint* ibmesh, iint* ibface, iint* ibnode);
    面グループ境界の面の節点 ID

```

```

iint mw_get_num_of_node_bedge(iint* ibmesh, iint* ibedge);
    辺グループ境界の辺の節点数
iint mw_get_node_id_bedge(iint* ibmesh, iint* ibedge, iint* ibnode);
    辺グループ境界の辺の節点 ID
iint mw_get_num_of_node_bvolume(iint* ibmesh, iint* ibvol);
    体積グループ境界の体積の節点 ID
iint mw_get_node_id_bvolume(iint* ibmesh, iint* ibvol, iint* ibnode);
    体積グループ境界の体積の節点 ID

iint mw_get_bnode_mesh_namelenlength(iint* ibmesh);
    点グループ境界の境界条件名の文字列長
void mw_get_bnode_mesh_name(iint* ibmesh, char* name, iint* name_len);
    点グループ境界の境界条件名
iint mw_get_bface_mesh_namelenlength(iint* ibmesh);
    面グループ境界の境界条件名の文字列長
void mw_get_bface_mesh_name(iint* ibmesh, char* name, iint* name_len);
    面グループ境界の境界条件名
iint mw_get_bvolume_mesh_namelenlength(iint* ibmesh);
    体積グループ境界の境界条件名の文字列長
void mw_get_bvolume_mesh_name(iint* ibmesh, char* name, iint* name_len);
    体積グループ境界の境界条件名
iint mw_get_bedge_mesh_namelenlength(iint* ibmesh);
    辺グループ境界の境界条件名の文字列長
void mw_get_bedge_mesh_name(iint* ibmesh, char* name, iint* name_len);
    辺グループ境界の境界条件名

iint mw_get_edge_id_bedge(iint* ibmesh, iint* ibedge);
    辺グループ境界の辺番号
iint mw_get_elem_id_bedge(iint* ibmesh, iint* ibedge);
    辺グループ境界の辺が所属する要素 ID
iint mw_get_face_id_bface(iint* ibmesh, iint* ibface);
    面グループ境界の面番号
iint mw_get_elem_id_bface(iint* ibmesh, iint* ibface);
    面グループ境界の面が所属する要素 ID
iint mw_get_elem_id_bvolume(iint* ibmesh, iint* ibvol);
    体積グループ境界の体積の要素 ID

```

C++言語 API { namespace pmw, class CMW }

```

uiint GetNumOfBoundaryNodeMesh();
    点グループ境界条件のグループ数
uiint GetNumOfBoundaryFaceMesh();
    面グループ境界条件のグループ数
uiint GetNumOfBoundaryEdgeMesh();
    辺グループ境界条件のグループ数
uiint GetNumOfBoundaryVolumeMesh();
    体積グループ境界条件のグループ数
uiint GetBNDType_BNodeMesh(const uiint& ibmesh);
    点グループ境界条件のタイプ、ノイマン型かディレクレ型
uiint GetBNDType_BFaceMesh(const uiint& ibmesh);
    面グループ境界条件のタイプ、ノイマン型かディレクレ型
uiint GetBNDType_BEdgeMesh(const uiint& ibmesh);
    辺グループ境界条件のタイプ、ノイマン型かディレクレ型
uiint GetBNDType_BVolumeMesh(const uiint& ibmesh);
    体積グループ境界条件のタイプ、ノイマン型かディレクレ型

uiint getNeumannType();
    ノイマン型を表すタイプ番号
uiint getDirichletType();
    ディレクレ型を表すタイプ番号

uiint GetNumOfBNode_BNodeMesh(const uiint& ibmesh);
    点グループ境界条件の節点数
uiint GetNumOfBNode_BFaceMesh(const uiint& ibmesh);
    面グループ境界条件の節点数
uiint GetNumOfBNode_BEdgeMesh(const uiint& ibmesh);
    辺グループ境界条件の節点数
uiint GetNumOfBNode_BVolumeMesh(const uiint& ibmesh);
    体積グループ境界条件の節点数

uiint GetNumOfDOF_BNodeMesh(const uiint& ibmesh, const uiint& ibnode);
    点グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
uiint GetNumOfDOF_BFaceMesh(const uiint& ibmesh);
    面グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
uiint GetNumOfDOF_BEdgeMesh(const uiint& ibmesh);
    辺グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)
uiint GetNumOfDOF_BVolumeMesh(const uiint& ibmesh);

```

体積グループ境界条件の自由度数(節点へ与える境界条件パラメータ数)

uiint GetDOF_BNodeMesh(const uiint& ibmesh, const uiint& ibnode, const uiint& idof);

点グループ境界条件の境界値の自由度番号

uiint GetDOF_BFaceMesh(const uiint& ibmesh, const uiint& idof);

面グループ境界条件の境界値の自由度番号

uiint GetDOF_BEdgeMesh(const uiint& ibmesh, const uiint& idof);

辺グループ境界条件の境界値の自由度番号

uiint GetDOF_BVolumeMesh(const uiint& ibmesh, const uiint& idof);

体積グループ境界条件の境界値の自由度番号

double& GetBNodeValue_BNodeMesh(const uiint& ibmesh, const uiint& ibnode, const uiint& dof);

点グループ境界条件の節点境界値

double& GetBNodeValue_BFaceMesh(const uiint& ibmesh, const uiint& ibnode, const uiint& dof, const uiint& mgLevel);

面グループ境界条件の節点境界値

double& GetBNodeValue_BEdgeMesh(const uiint& ibmesh, const uiint& ibnode, const uiint& dof, const uiint& mgLevel);

辺グループ境界条件の節点境界値

double& GetBNodeValue_BVolumeMesh(const uiint& ibmesh, const uiint& ibnode, const uiint& dof, const uiint& mgLevel);

体積グループ境界条件の節点境界値

uiint& GetNodeID_BNode_BNodeMesh(const uiint& ibmesh, const uiint& ibnode);

点グループ境界条件の節点 ID

uiint& GetNodeID_BNode_BFaceMesh(const uiint& ibmesh, const uiint& ibnode);

面グループ境界条件の節点 ID

uiint& GetNodeID_BNode_BEdgeMesh(const uiint& ibmesh, const uiint& ibnode);

辺グループ境界条件の節点 ID

uiint& GetNodeID_BNode_BVolumeMesh(const uiint& ibmesh, const uiint& ibnode);

体積グループ境界条件の節点 ID

uiint GetNumOfBFace(const uiint& ibmesh);

面グループ境界条件の面の数

double& GetBFaceValue(const uiint& ibmesh, const uiint& ibface, const uiint& dof);

面グループ境界条件の面の境界値

uiint GetNumOfBEdge(const uiint& ibmesh);

辺グループ境界条件の辺の数

double& GetBEdgeValue(const uiint& ibmesh, const uiint& ibedge, const uiint& dof);

辺グループ境界条件の辺の境界値

uiint GetNumOfBVolume(const uiint& ibmesh);

体積グループ境界条件の体積数

```
double& GetBVVolumeValue(const uiint& ibmesh, const uiint& ibvol, const uiint& dof);
```

体積グループ境界条件の体積中心境界値

```
uiint GetNumOfNode_BFace(const uiint& ibmesh, const uiint& ibface);
```

面グループ境界の面の節点数

```
uiint& GetNodeID_BFace(const uiint& ibmesh, const uiint& ibface, const uiint& ibnode);
```

面グループ境界の面の節点 ID

```
uiint GetNumOfNode_BEEdge(const uiint& ibmesh, const uiint& ibedge);
```

辺グループ境界の辺の節点数

```
uiint& GetNodeID_BEEdge(const uiint& ibmesh, const uiint& ibedge, const uiint& ibnode);
```

辺グループ境界の辺の節点 ID

```
uiint GetNumOfNode_BVolume(const uiint& ibmesh, const uiint& ibvol);
```

体積グループ境界の体積の節点 ID

```
uiint& GetNodeID_BVolume(const uiint& ibmesh, const uiint& ibvol, const uiint& ibnode);
```

体積グループ境界の体積の節点 ID


```
uiint GetBNodeMesh_NameLength(const uiint& ibmesh);
```

点グループ境界の境界条件名の文字列長

```
string& GetBNodeMesh_Name(const uiint& ibmesh);
```

点グループ境界の境界条件名

```
uiint GetBFaceMesh_NameLength(const uiint& ibmesh);
```

面グループ境界の境界条件名の文字列長

```
string& GetBFaceMesh_Name(const uiint& ibmesh);
```

面グループ境界の境界条件名

```
uiint GetBVVolumeMesh_NameLength(const uiint& ibmesh);
```

体積グループ境界の境界条件名の文字列長

```
string& GetBVVolumeMesh_Name(const uiint& ibmesh);
```

体積グループ境界の境界条件名

```
uiint GetBEEdgeMesh_NameLength(const uiint& ibmesh);
```

辺グループ境界の境界条件名の文字列長

```
string& GetBEEdgeMesh_Name(const uiint& ibmesh);
```

辺グループ境界の境界条件名


```
uiint GetEdgeID_BEEdge(const uiint& ibmesh, const uiint& ibedge);
```

辺グループ境界の辺番号

```
uiint GetElemID_BEEdge(const uiint& ibmesh, const uiint& ibedge);
```

辺グループ境界の辺が所属する要素 ID

```
uiint GetFaceID_BFace(const uiint& ibmesh, const uiint& ibface);
```

面グループ境界の面番号

```
uiint GetElemID_BFace(const uiint& ibmesh, const uiint& ibface);
```

面グループ境界の面が所属する要素 ID

```
uiint GetElemID_BVolume(const uiint& ibmesh, const uiint& ibvol);
```

体積グループ境界の体積の要素 ID

6.12 MPI ラップ関数&通信節点

C 言語 API

```
int mw_mpi_int_0();
```

MPI_INT のタイプ番号の取得

```
int mw_mpi_double_0();
```

MPI_DOUBLE のタイプ番号の取得

```
int mw_mpi_comm_0();
```

MPI_COMM_WORLD のタイプ番号の取得

```
int mw_mpi_sum_0();
```

MPI_SUM のタイプ番号の取得

```
int mw_mpi_max_0();
```

MPI_MAX のタイプ番号の取得

```
int mw_mpi_min_0();
```

MPI_MIN のタイプ番号の取得

```
int mw_get_rank_0();
```

自身のプロセス番号の取得

```
int mw_get_num_of_process_0();
```

全体のプロセス数の取得

```
void mw_allreduce_r(double val[], int* val_size, int* op);
```

MPI_Allreduce のラッパー関数、実数型

```
void mw_allreduce_i(int val[], int* val_size, int* op);
```

MPI_Allreduce のラッパー関数、整数型

```
int mw_barrier_0();
```

MPI_Barrier のラッパー関数

```
int mw_abort(int* error);
```

MPI_Abort のラッパー関数

```
int mw_allgather_r(double sendbuf[], int* sendcnt, double recvbuf[], int* recvcnt);
```

MPI_Allgather のラッパー関数の実数型

```
int mw_allgather_i(int sendbuf[], int* sendcnt, int recvbuf[], int* recvcnt);
```

MPI_Allgather のラッパー関数の整数型

```

int mw_gather_r(double sendbuf[], int* sendcnt, double recvbuf[], int* recvcnt, int* root);
    MPI_Gather のラッパー関数の実数型
int mw_gather_i(int sendbuf[], int* sendcnt, int recvbuf[], int* recvcnt, int* root);
    MPI_Gather のラッパー関数の整数型
int mw_scatter_r(double sendbuf[], int* sendcnt, double recvbuf[], int* recvcnt, int* root);
    MPI_Scatter のラッパー関数の実数型
int mw_scatter_i(int sendbuf[], int* sendcnt, int recvbuf[], int* recvcnt, int* root);
    MPI_Scatter のラッパー関数の整数型
int mw_bcast_r(double buf[], int* cnt, int* root);
    MPI_Bcast のラッパー関数の実数型
int mw_bcast_i(int buf[], int* cnt, int* root);
    MPI_Bcast のラッパー関数の整数型
int mw_bcast_s(char buf[], int* cnt, int* root);
    MPI_Bcast のラッパー関数の文字列型
int mw_send_r(double buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Send のラッパー関数、実数型
int mw_send_i(int buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Send のラッパー関数、整数型
int mw_recv_r(double buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Recv のラッパー関数、実数型
int mw_recv_i(int buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Recv のラッパー関数、整数型
int mw_get_num_of_neibpe(int* imesh);
    自身のプロセスが通信するプロセス数
int mw_get_transrank(int* imesh, int* ipe);
    自身が通信する相手プロセスのプロセス番号(ランク番号)
void mw_send_recv_r(double buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Send と MPI_Recv 関数 実数型
void mw_send_recv_i(int buf[], int* num_of_node, int* dof_size, int* trans_rank);
    MPI_Send と MPI_Recv 関数 整数型
iint mw_get_num_of_comm_mesh();
    通信テーブル数
iint mw_get_num_of_comm_node(iint* icmesh);
    通信テーブルの通信節点数
iint mw_get_node_id_comm_node(iint* icmesh, iint* icnode);
    通信節点のメッシュパーツの節点 ID

```

C++言語 API { namespace pmw, class CMW }

int& GetRank();
 自身のプロセス番号の取得

int& GetNumOfProcess();
 全体のプロセス数の取得

int AllReduce(void* sendbuf, void* recvbuf, int buf_size, int datatype, int op, int commworld);
 MPI_Allreduce のラッパー関数

int Barrier(int commworld);
 MPI_Barrier のラッパー関数

int Abort(int commworld, int error);
 MPI_Abort のラッパー関数

int AllGather(void* sendbuf, int sendcnt, MPI_Datatype sendtype, void* recvbuf, int recvcnt, MPI_Datatype recvtype, MPI_Comm comm);
 MPI_Allgather のラッパー関数

int Gather(void* sendbuf, int sendcnt, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm comm);
 MPI_Gather のラッパー関数

int Scatter(void* sendbuf, int sendcnt, MPI_Datatype sendtype, void* recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm);
 MPI_Scatter のラッパー関数

int Recv(void* buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status);
 MPI_Recv のラッパー関数

int Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm);
 MPI_Send のラッパー関数

int Bcast(void* buf, int cnt, MPI_Datatype type, int root, MPI_Comm comm);
 MPI_Bcast のラッパー関数

uiint GetNumOfNeibPE(const uiint& imesh);
 自身のプロセスが通信するプロセス数

uiint& GetTransRank(const uiint& imesh, const uiint& ipe);
 自身が通信する相手プロセスのプロセス番号(ランク番号)

void Send_Recv_R(double* buf, const int& num_of_node, const int& dof_size, const int& trans_rank);
 MPI_Send と MPI_Recv 関数 実数型

void Send_Recv_I(int* buf, const int& num_of_node, const int& dof_size, const int& trans_rank);
 MPI_Send と MPI_Recv 関数 整数型

uiint GetNumOfCommMesh();
 通信テーブル数

uiint GetNumOfCommNode(const uiint& icmesh);
 通信テーブルの通信節点数

uiint& GetNodeID_CommNode(const uiint& icmesh, const uiint& icnode);

6.13 ロガー

ロガーは、4つのレベル(Debug,Error,Warn,Info)の指定が出来、アプリケーション・プログラムの開発者が、これを指定する。またデバイス指定によって、出力先をコンソール端末とファイルの2種類を選択することができる。

表 6.13.1 ログレベル

Logger_Mode で指定するログレベル	動作するログレベル
Debug	Debug, Error, Warn, Info
Error	Error, Warn, Info
Warn	Warn, Info
Info	Info

例えば、**mw_logger_info** の Mode を **Error** 指定で文字列出力をソースに埋め込んだ場合に、設定モードが **Warn** であった場合には、文字列は出力されない。**Error** 指定の文字列出力は、モード設定が **Error** もしくは **Debug** の場合に出力される。

C 言語 API

```
void mw_logger_set_mode(iint* mode);
```

ロガーモードの設定

```
void mw_logger_set_device(iint* mode, iint* device);
```

モード別の出力デバイスの設定

```
void mw_logger_info_ (iint* mode, char* message, iint* str_len);
```

モード別の文字列出力

```
iint mw_get_error_mode_0();
```

モード番号の取得(エラー・モード)

```
iint mw_get_warn_mode_0();
```

モード番号の取得(ワーニング・モード)

```
iint mw_get_info_mode_0();
```

モード番号の取得(インフォ・モード)

```
iint mw_get_debug_mode_0();
```

モード番号の取得(デバッグ・モード)

```
iint mw_get_disk_device_0();
```

出力デバイス番号の取得(ファイル出力)

uint mw_get_display_device();

出力デバイス番号の取得(画面出力)

C++言語 API { namespace pmw, class CMW }

void LoggerMode(const uint& mode);

ロガーモードの設定

void LoggerDevice(const uint& mode, const uint& device);

モード別の出力デバイスの設定

void LoggerInfo(const uint& mode, char* message);

モード別の文字列出力

void LoggerInfo(const uint& mode, const char* message);

モード別の文字列出力

uint getErrorMode();

モード番号の取得(エラー・モード)

uint getWarnMode();

モード番号の取得(ワーニング・モード)

uint getInfoMode();

モード番号の取得(インフォ・モード)

uint getDebugMode();

モード番号の取得(デバッグ・モード)

uint getDiskDevice();

出力デバイス番号の取得(ファイル出力)

uint getDisplayDevice();

出力デバイス番号の取得(画面出力)

7. HECMW 中間メッシュ・ファイル形式

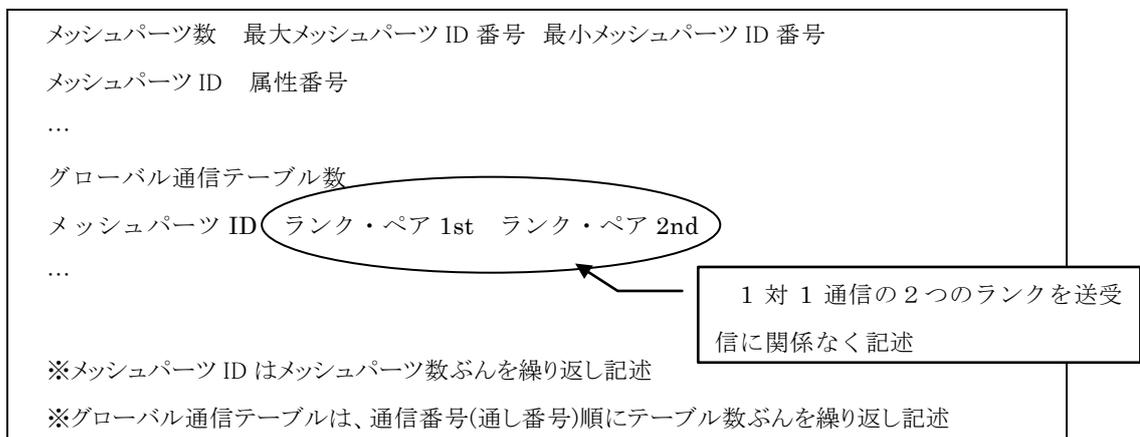
パーティショナーから出力される HEC_MW Ver4.0 中間ファイルの書式を説明する。

7.1 ファイル形式の定義

ブロックデータ構造をとり、各ブロックの内容は以下である。デリミタはスペースを用いる。各ブロックデータは、"ブロック名～End"の中に記述する。

7.1.1 アセンブルモデル

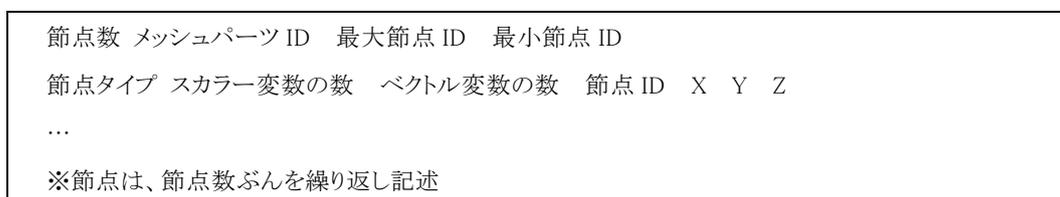
ブロック名：AssyModel～End



7.1.2 ノード

各メッシュパーツ内の節点情報を記述

ブロック名：Node～End



7.1.3 エレメント

各メッシュパーツ内の要素情報を記述

ブロック名：Element～End

要素数	メッシュパーツ ID	最大要素 ID	最小要素 ID
要素タイプ	要素 ID	要素を構成する節点 ID…要素を構成する節点 ID	
…			
※要素数ぶんを繰り返し記述			

7.1.4 通信 Mesh

通信領域のメッシュデータを記述

ブロック名：CommMesh2～End

メッシュパーツ ID	メッシュパーツ毎の通信メッシュ数
通信メッシュ ID	通信面の数 通信節点の数 自身のランク番号 通信先ランク番号
…	

7.1.5 通信ノード

通信領域の節点データを記述

ブロック名：CommNodeCM2～End

要素分割型通信メッシュ ID	メッシュパーツ ID	通信節点数
通信節点 ID	節点 ID	X Y Z
…		

7.1.6 通信面

通信領域の界面データを記述

ブロック名：CommFace～End

要素分割型通信メッシュ ID	メッシュパーツ ID	通信界面数
面タイプ	通信面 ID	要素 ID エンティティ番号 構成通信節点 ID… 構成通信節点 ID
…		

面タイプ:”Quad”, ”Quad2”, ”Triangle”, ”Triangle2”, ”Beam”, ”Beam2”, ”Point”

エンティティ番号は、面の場合:局所面番号、辺の場合:局所辺番号、点の場合:局所節点番号

7.1.7 接合メッシュ(ContactMesh)

メッシュパーツを接合する面を記述

ブロック名：ContactMesh～End

接合メッシュ数	最大接合メッシュ ID	最小接合メッシュ ID								
接合メッシュ ID	自身のランク番号	属性番号								
...										
接合節点数	最大接合節点 ID	最小接合節点 ID								
接合節点 ID	X	Y	Z	メッシュパーツ ID	節点 ID	所属ランク番号	maslave 番号	接合節点のタイプ	自由度数	ダミー
...										
...										
マスター接合面数	最大接合面 ID	最小接合面 ID								
マスター接合面 ID	メッシュパーツ ID	要素 ID	要素面番号	面形状タイプ	面を構成する接合節点 ID	...	面を構成する接合節点 ID	所属ランク		
...										
...										
スレーブ接合面数	最大接合面 ID	最小接合面 ID								
スレーブ接合面 ID	メッシュパーツ ID	要素 ID	要素面番号	面形状タイプ	面を構成する接合節点 ID	...	面を構成する接合節点 ID	所属ランク		
...										
...										

maslave 番号 : 0:マスター、1:スレーブ

面形状の値は、文字列: "Quad", "Quad2", "Triangle", "Triangle2"

* [メッシュパーツ ID](#) [節点 ID](#) [要素 ID](#) [面番号](#) は対応するデータが存在しない場合は、マイナス記号 "-" を使用。

対応しないメッシュパーツとは、並列計算のために分割された別領域に属している面

接合節点のタイプ: "s", "S", "v", "V", "sv", "SV"

7.1.8 境界条件

メッシュパーツに所属する境界条件について記述する。

境界条件は、節点境界、辺境界、面境界、体積境界に別れており、種類は、ディレクレ、ノイマンの 2 種類である。

点グループ境界条件

ブロック名: BoundaryNodeMesh

メッシュパーツ ID	境界条件数	
節点境界メッシュ ID	境界条件種類	名称
...		
(繰り返し記述)		

ブロック名: BoundaryNode

節点境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数			
境界節点 ID	節点 ID	X	Y	Z	自由度番号	境界値
...						
(繰り返し記述)						

面グループ境界条件

ブロック名: BoundaryFaceMesh

メッシュパーツ ID	境界条件数					
面境界メッシュ ID	境界条件種類	名称	自由度数	自由度番号	自由度番号	...(繰り返し)
...						
(繰り返し記述)						

ブロック名: BoundaryFace

ノイマン型境界

面境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	Face 数		
境界節点 ID	節点 ID					
...						
(繰り返し記述)						
面形状タイプ	境界面 ID	要素 ID	面番号	自由度番号	境界節点 ID...(繰り返し)	境界値
...						
(繰り返し記述)						

ディレクレ型境界

面境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	Face 数
境界節点 ID	節点 ID	DOF 数	DOF 番号 境界値	DOF 番号 境界値
...			DOF 番号 境界値	...
(繰り返し記述)				
面形状タイプ	境界面 ID	要素 ID	面番号	境界節点 ID...(繰り返し)
...				
(繰り返し記述)				

辺グループ境界条件

ブロック名: BoundaryEdgeMesh

メッシュパーツ ID	境界条件数
辺境界メッシュ ID	境界条件種類
名称	自由度数
自由度番号	自由度番号
...	...(繰り返し)
(繰り返し記述)	

ブロック名: BoundaryEdge

ノイマン型

辺境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	Edge 数
境界節点 ID	節点 ID			
...				
(繰り返し記述)				
辺形状タイプ	境界辺 ID	要素 ID	辺番号	自由度番号
				境界節点 ID
				境界節点 ID
				境界値
...				
(繰り返し記述)				

ディレクレ型

辺境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	Edge 数	
境界節点 ID	節点 ID	DOF 数	DOF 番号 境界値	DOF 番号 境界値	DOF 番号 境界値...
...					
(繰り返し記述)					
辺形状タイプ	境界辺 ID	要素 ID	辺番号	境界節点 ID 境界節点 ID	
...					
(繰り返し記述)					

体積グループ境界条件

ブロック名: BoundaryVolumeMesh

メッシュパーツ ID	境界条件数					
体積境界メッシュ ID	境界条件種類	名称	自由度数	自由度番号	自由度番号	...(繰り返し)
...						
(繰り返し記述)						

ブロック名: BoundaryVolume

ノイマン型

体積境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	要素数		
境界節点 ID	節点 ID					
...						
(繰り返し記述)						
形状タイプ	境界体積 ID	要素 ID	0	自由度番号	境界節点 ID...(繰り返し)	境界値
...						
(繰り返し記述)						

要素 ID の後ろに記述する”0”は定数。

ディレクレ型

体積境界メッシュ ID	境界条件種類	メッシュパーツ ID	境界節点数	要素数
境界節点 ID	節点 ID	DOF 数	DOF 番号 境界値	DOF 番号 境界値 DOF 番号 境界値...
...				
(繰り返し記述)				
形状タイプ	境界体積 ID	要素 ID 0	境界節点 ID...(繰り返し)	
...				
(繰り返し記述)				

7.2 要素エンティティ番号

7.2.1 Hexa 要素 エンティティ番号

(1) 形状定義

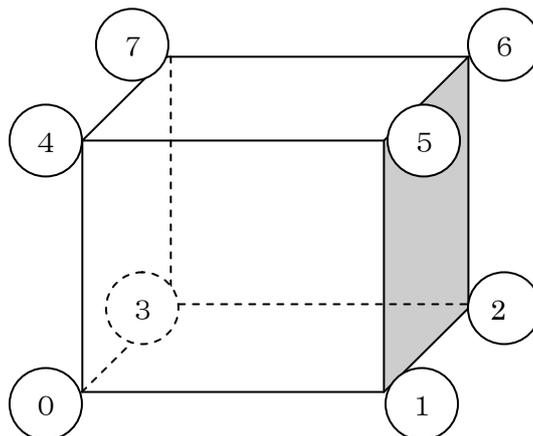


図 7.2.1 Hexa 要素

(2) Hexa 面番号

- 面 0: 0-1-2-3(図の底面)
- 面 1: 4-5-6-7(図の上面)
- 面 2: 1-5-6-2(図の右面)
- 面 3: 0-3-7-4(図の左面)
- 面 4: 0-4-5-1(図の手前)
- 面 5: 2-6-7-3(図の奥)

(3) Hexa 辺番号

辺0:0-1	辺4:4-5	辺8:0-4
辺1:1-2	辺5:5-6	辺9:1-5
辺2:2-3	辺6:6-7	辺10:2-6
辺3:3-0	辺7:7-4	辺11:3-7

7.2.2 Tetra 要素 エンティティ番号

(1) 形状定義

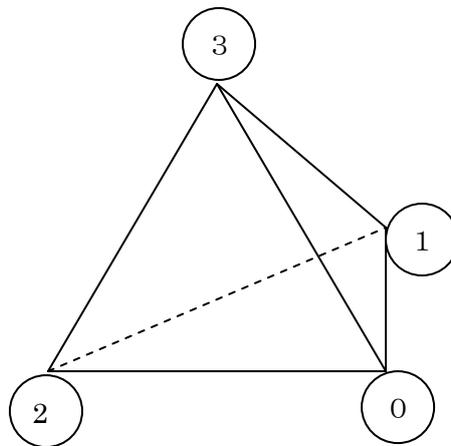


図 7.2.2 Tetra 要素

(2) Tetra 面番号

- 面 0: 0-1-2(図の底面)
- 面 1: 0-3-1(図の右面)
- 面 2: 1-3-2(図の奥)
- 面 3: 0-2-3(図の手前)

(3) Tetra 辺番号

辺0:0-1	辺3:0-3
辺1:1-2	辺4:1-3
辺2:2-0	辺5:2-3

7.2.3 Prism 要素 エンティティ番号

(1) 形状定義

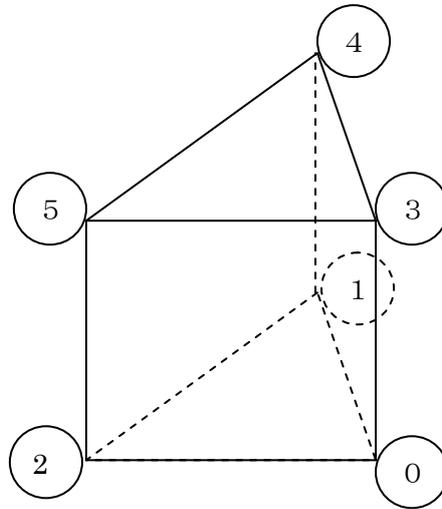


図 7.2.3 Prism 要素

(2) Prism 面番号

面 0: 0-1-2(図の底面)

面 1: 3-4-5(図の上面)

面 2: 0-3-4-1(図の奥右)

面 3: 1-4-5-2(図の奥左)

面 4: 0-2-5-3(図の手前)

(3) Prism 辺番号

辺0:0-1	辺3:0-3	辺6:3-4
辺1:2-0	辺4:1-4	辺7:4-5
辺2:1-2	辺5:2-5	辺8:5-3

7.2.4 Quad 要素 エンティティ番号

(1) 形状定義

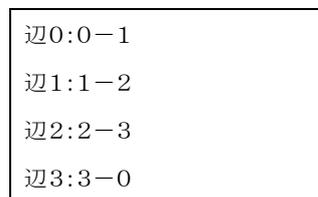


図 7.2.4 Quad 要素

(2) Quad 面番号

面 0: 0-1-2-3

(3) Quad 辺番号



7.2.5 Triangle 要素エンティティ番号

(1) 形状定義

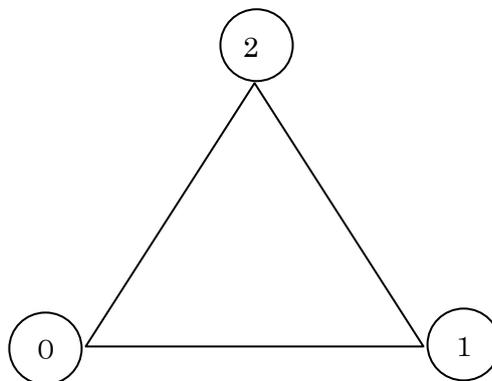


図 7.2.5 Triangle 要素

(2) Triangle 面番号

面 0: 0-1-2

(3) Triangle 辺番号

辺0:0-1

辺1:1-2

辺2:2-0

7.2.6 Beam 要素エンティティ番号

7.2.6.1. Beam 面番号

(1) 形状定義



図 7.2.6 Beam 要素

(2) Beam 面番号

なし

(3) Beam 辺番号

辺0:0-1

7.3 ユーティリティ

7.3.1 ロガー機能

プログラム開発をする上で必要なログ機能を提供する。

ロガーは、プログラム中にロガー呼び出しルーチンを記述することで使用できる。

ロガーは、4つのレベル(**Debug, Error, Warn, Info**)により動作が異なり、FEM解析コードの開発者が、これを指定する。ステートによる動作の差異は、下表に示す。

ロガーは、出力先をコンソール端末とファイルの2種類を選択することができる。

ロガーのモードによる動作を次の表にまとめておく。

表 7.3.1 ログレベルによる動作

Logger_Mode で指定するログレベル+	Logger_Monitor が動作するログレベル	備考
-------------------------	---------------------------	----

Debug	Debug, Error, Warn, Info	HEC_MW を用いたプログラム開発に必要な情報は Debug 指定しておく。リリース時にはログレベルを Error 以下に変更すること。
Error	Error, Warn, Info	論理的なエラーなどに利用。
Warn	Warn, Info	ユーザへの警告。
Info	Info	著作権等の情報表示に利用。

8. ユーティリティ・プログラム

8.1 ファイル形式変換ツール

8.1.1 コンパイル手順

util/conv2mw3 ディレクトリで、make を実行する。

8.1.2 利用方法

ファイル変換プログラム"conv2mw3"の置いてあるカレント・ディレクトリを Directory とした場合に下記のように使用する。

c:¥Directory>conv2mw3 <入力ファイル名> <出力ファイル名> <[-u]>

- ・ 入力ファイル名 : FrontISTR v 4 のメッシュデータ名
- ・ 出力ファイル名 : MW 3 のメッシュデータ名
- ・ -u : オプションは、各グループデータを Dirichlet か、Neumann にするかの選択をユーザーが行う。
- ・ オプションをつけない場合は、NGROUP⇒Dirichlet、その他の GROUP⇒Neumann になる。

c:¥Directory>conv2mw3 だけを入力した場合は、上記の使い方がモニターに表示される。

注意)

旧版 REVOCAP_PrePost を使用していて、FrontISTR メッシュデータを出力させた場合はタグを追加する必要がある出てくる。

例として、NUM とか、PARTNAME とかを追加する。詳細は FrontISTR マニュアルを参照すること。

8.2 領域分割ツール

8.2.1 コンパイル手順

METIS ライブラリにインクルードパス、ライブラリパスを通じた状態で、`util/part` ディレクトリ内で `make` する。

領域分割ツールのコンパイルには本ライブラリが必要なため、先にコンパイルしておく必要がある。このときに読み込まれる `Makefile.in` は、本ライブラリのコンパイルの際に用いられるものと共通である。

8.2.2 利用方法

実行時オプションは以下の通りである。

- i: 入力ファイル名 (領域番号と拡張子 `.0.msh` を除いたもの)
- O: 出力ファイル名 (領域番号と拡張子 `?.msh` を除いたもの)
- n: 領域分割数

たとえば、

```
$ ./part -i out -o para -n 4
```

上記コマンドで、`out.0.msh` から `para.[0-3].msh` が出力される。

8.2.3 実行例

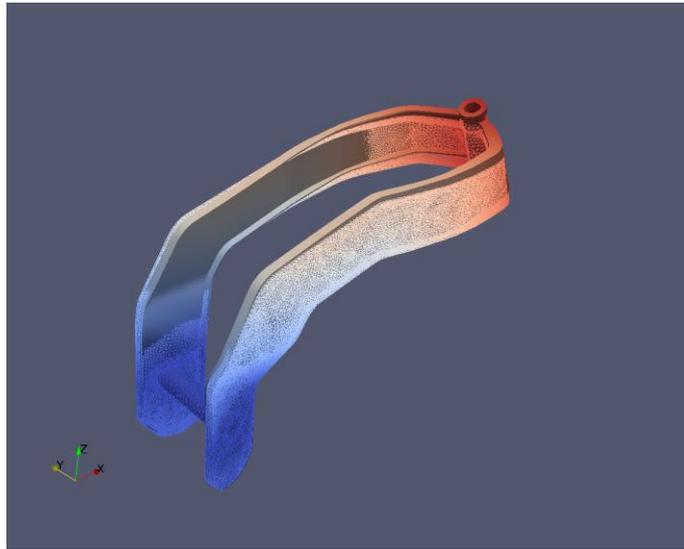
例 1

要素タイプ	341
節点数	100
要素数	261
リファイン回数	2
リファイン方法	MW
リファイン後節点数	10,597
リファイン後要素数	8,352

例 2

要素タイプ	341
-------	-----

節点数	27,114
要素数	81,754
リファイン回数	0



制限事項

本プログラムは、つぎのような位置付けで公開している。

- ▶ 解析アプリケーションプログラム開発者にライブラリプログラムを提供する。
- ▶ そのライブラリプログラムを利用した例題プログラムも合わせて提供する。

インストール方法の章に記述した計算機環境で動作確認を行っている。