FrontISTR のビルド 虎の巻 (Windows/Metis4 の書)

海洋研究開発機構 地球情報基盤センター

小川 道夫

はじめに

FrontISTR は非線形構造解析機能が充実したオープンソースの構造解析ソフトウェアです。

大規模並列 FEM 基盤ミドルウェア上に構築され、先進性と実用性を兼ね備えています。

京・地球シミュレータ・FX10 などのスーパーコンピュータや各種クラウドサービスから身近にあるパソコンまでのスケーラビリティを備え、並列環境をあまり意識しないシンプルで使いやすい解析手順が提供されています。

ソースコードは公開され、ドキュメントも充実しているため必要であれば新たに機能を実装し、独自のニーズに対応することもできます。

以下では FrontISTR のビルド手順を説明します。

- Windows (64 ビット版)環境へのインストール方法を説明します。
- fistr1、fistr2 をビルドします。
- FrontISTR に含まれるインストールマニュアルの補助資料としてご利用ください。
- 作成する FrontISTR は MPI、OpenMP、パーティショナ、リファイナー、Metis、LAPACK、MUMPS、ML、paracon(並列接触解析)を有効にします。

ソフトウェアのダウンロード

以下のリストを参照して、構築に必要なソフトウェアをダウンロードして下さい。

ソフトウェア名	ダウンロードサイト	ダウンロードするもの
MSYS2	https://msys2.github.io	msys2-x86_64-20150916.exe
Microsoft MPI v7	https://www.microsoft.com/en-us/download/details.aspx?id=49926	msmpi.msi MSMpiSetup.exe
gendef	http://sourceforge.net/projects/mingw/files/MinGW/Extension/gendef/gendef-1.0.1346/	gendef-1.0.1346-1.mingw32-bin.tar.lzma
OpenBLAS	http://www.openblas.net/	OpenBLAS-0.2.15.tar.gz
METIS	http://glaros.dtc.umn.edu/gkhome/metis/metis/download	metis-5.1.0.tar.gz
ScaLAPACK	http://www.netlib.org/scalapack/	scalapack-2.0.2.tgz
MUMPS	http://mumps.enseeiht.fr/ ※要ユーザ登録	MUMPS_5.0.1.tar.gz
ML(Trilinos)	https://trilinos.org/ ※要ユーザ登録	trilinos-12.4.2-Source.tar.bz2
REVOCAP_Refiner	http://www.multi.k.u-tokyo.ac.jp/FrontISTR/reservoir_f/revisions.php FrontISTR リザーバ内 ※要ユーザ登録	REVOCAP_Refiner-1.1.03.tar.gz
FrontISTR	http://www.multi.k.u-tokyo.ac.jp/FrontISTR/reservoir_f/revisions.php FrontISTR リザーバ内 ※要ユーザ登録	FrontISTR_V44.tar.gz
FrontISTR 用パッチ	文中参照	fix_omp_for_gfortran.patch

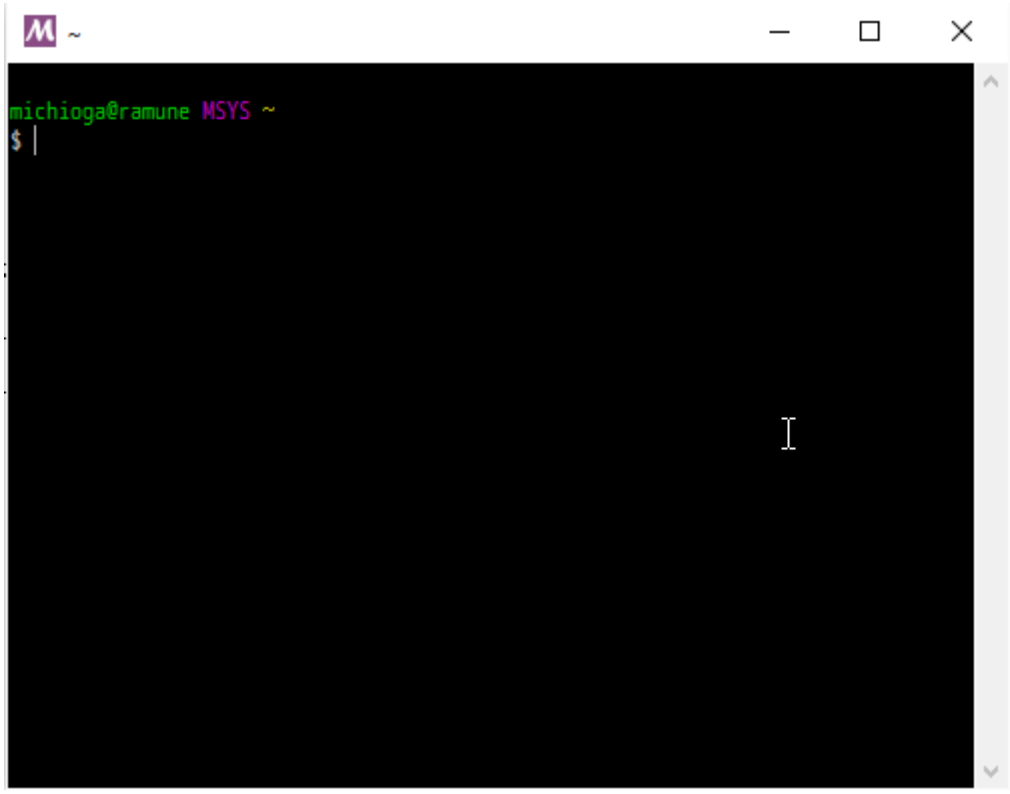
ビルド環境の構築

MSYS2 を用いて MinGW 64 ビット版の環境を構築します。

Unix 系のソフトウェアを Windows 上でビルドする環境として Cygwin が有名ですが、ランタイムへの依存があるため今回は用いません。

ダウンロードしたインストーラ msys2-x86_64-20150916.exe をダブルクリックし、ガイドに従ってインストールして下さい。

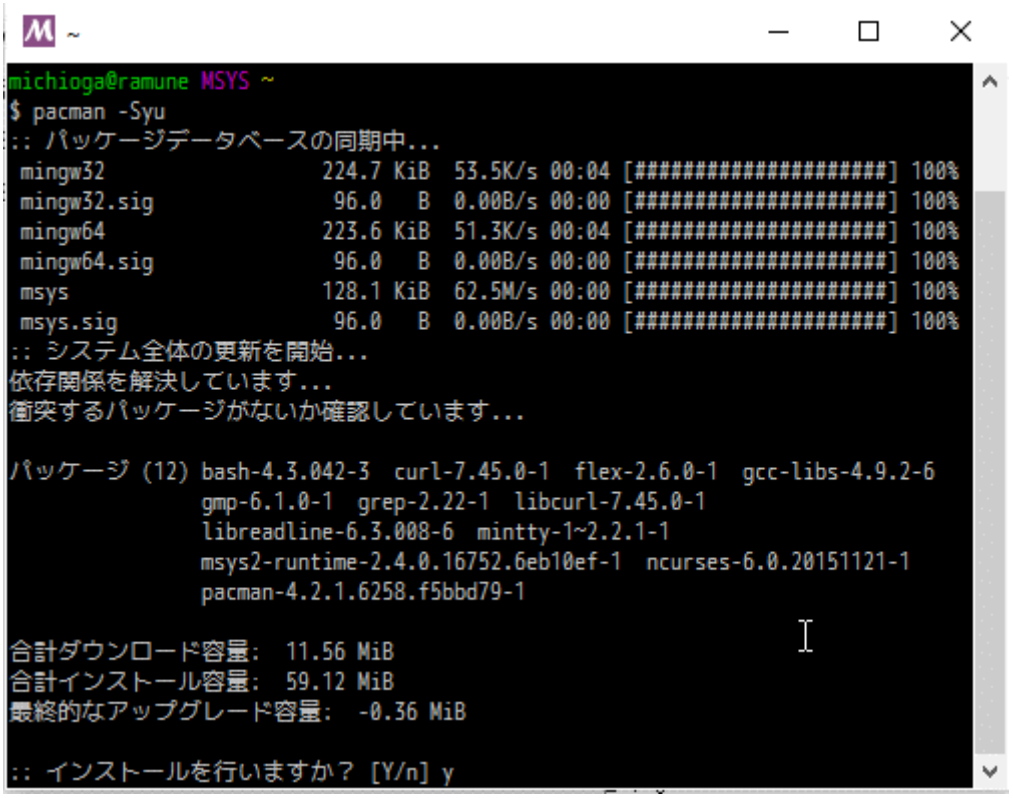
インストーラが終了すると「MSYS2 Shell」が実行されます。



MSYS2 は archlinux 由来の「pacman」というパッケージマネージャで管理されています。

pacman を使って環境を最新の状態にしてください。

```
(MSYS)$ pacman -Syu
```



パッケージのアップデートが完了したら、一度ウインドウを閉じて下さい。

再度「MSYS2 Shell」を起動し、その他のパッケージをインストールしていきます。

```
(MSYS)$ pacman -S base-devel mingw-w64-x86_64-toolchain mingw-w64-x86_64-cmake \
        mingw-w64-x86_64-extra-cmake-modules mingw-w64-x86_64-boost
```

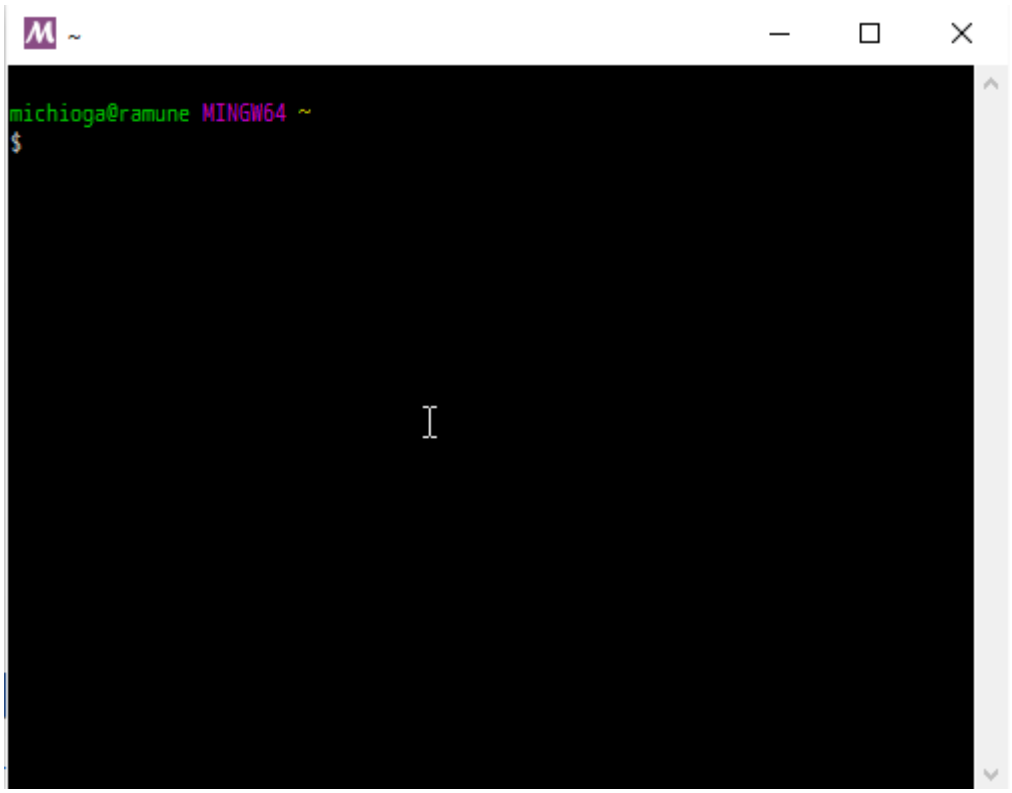
pacman を使い、構築済みのパッケージをインストールする事も出来ます。

```
キーワードでパッケージ名を検索
(MSYS)$ pacman -Ss <キーワード>


パッケージをインストール
(MSYS)$ pacman -S <パッケージ名>
```

以上でビルド環境の構築は完了しましたので、「MSYS2 Shell」を閉じて下さい。

これからの作業は「MinGW-w64 Win64」で行います。



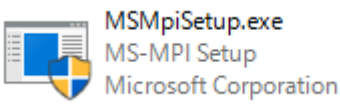
また、ビルド作業は「$HOME/Software」で行います。

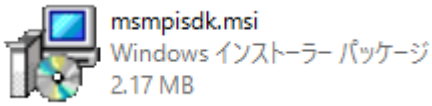
```
(MINGW64)$ cd $HOME
(MINGW64)$ mkdir Software
```

Microsoft MPI v7 のインストールと調整

Windows 用 MPI ライブラリとして、MPICH を改良した Microsoft MPI を利用します。

ダウンロードした、msmpi.msi と MSMpiSetup.exe をダブルクリックしインストールして下さい。





次に、インストールした各種ファイルを$HOME/msmpi 以下にコピーして下さい。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ mkdir -p msmpi/Lib
(MINGW64)$ cd msmpi
```

```
(MINGW64)$ cp -r /c/Program\ Files\ \(x86\)/Microsoft\ SDKs/MPI/Include/ .
(MINGW64)$ cp Include/x64/mpifptr.h Include/mpifptr.h
(MINGW64)$ cp /c/Windows/System32/msmpi.dll Lib/
```

コピーした msmpi.dll から MinGW で使えるライブラリを生成します。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf gendef-1.0.1345-1-mingw32-bin.tar.lzma
(MINGW64)$ cd $HOME/Software/msmpi/Lib
(MINGW64)$ $HOME/Software/bin/gendef msmpi.dll
(MINGW64)$ ls
msmpi.def  msmpi.dll
```

msmpi.def ファイルが新たに生成されます。

```
(MINGW64)$ dlltool -d msmpi.def -l libmsmpi.a -D msmpi.dll
```

これで、MinGW 環境でリンク可能なライブラリ libmsmpi.a が生成されます。コピーした msmpi.dll は必要ないので削除して下さい。

```
(MINGW64)$ rm msmpi.dll msmpi.def
```

次は、ヘッダファイル mpi.h に変更を加えます。

```
(MINGW64)$ cd ../Include
(MINGW64)$ vi mpi.h
#ifndef MPI_INCLUDE
#define MPI_INCLUDE
のすぐ下に #include <stdint.h> を追加し
#ifndef MPI_INCLUDE
#define MPI_INCLUDE
#include <stdint.h>
の様に変更
```

同様に mpif.h にも変更を加えます。

```
(MINGW64)$ vi mpif.h
PARAMETER (MPI_ADDRESS_KIND=INT_PTR_KIND())
を
PARAMETER (MPI_ADDRESS_KIND=8)
へ変更
```

これで Microsoft MPI の準備は完了です。

OpenBLAS のビルド

Windows 版 OpenBLAS はバイナリ配布がありますが、OpenMP を有効にしたライブラリを作成するためソースからビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf OpenBLAS-0.2.15.tar.gz
(MINGW64)$ cd OpenBLAS-0.2.15
(MINGw64)$ make BINARY=64 NO_SHARED=1 USE_OPENMP=1
(MINGW64)$ make PREFIX=$HOME/Software install
```

以上でライブラリとヘッダファイルがインストールされます。

他のマシンで実行させる場合、その CPU に合わせた TARGET を指定する必要があるかもしれません。

TargetList.txt にサポートされている CPU の一覧からキーワードを探し、動作させるマシンの TARGET を指定してください。

```
Nehalem なら
(MINGW64)$ make BINARY=64 NO_SHARED=1 USE_OPENMP=1 USE_THREAD=1 TARGET=NEHALEM
Sandy Bridge なら
(MINGW64)$ make BINARY=64 NO_SHARED=1 USE_OPENMP=1 USE_THREAD=1 TARGET=SANDYBRIDGE
```

新しい CPU を TARGET に指定したライブラリは、古い CPU 上では正常に動かない可能性があります。

また、他の CPU の最適化に対応したライブラリを作成する事も出来ます。

```
(MINGW64)$ make BINARY=64 NO_SHARED=1 USE_OPENMP=1 USE_THREAD=1 DYNAMIC_ARCH=1
```

DYNAMIC_ARCH=1 を指定した場合、ビルドにかかる時間が若干増え、出来上がったライブラリのサイズも大きくなります。

また、上記のオプションでビルドしたライブラリには LAPACK も含まれます。LAPACK をリンクしたい場合、-llapack の代わりに-lopenblas を指定します。

METIS のビルド

古い metis-4.0.3 をビルドします。

```
(MINGW64)$ cd $HOME/Software
```

```
(MINGW64)$ tar xvf metis-4.0.3.tar.gz
(MINGW64)$ metis-4.0.3
```

Makefile.in を修正します。

```
(MINGW64)$ vi Makefile.in
CC = gcc
COPTIONS = -D__VC__
※変更点のみ記載
```

また、MinGW には srand48 と drand48 の関数がありませんので、代わりに srand と drand を使います(乱数の品質は下がります)。

以上の修正が済んだらビルドします。

```
(MINGW64)$ make
```

以上で METIS のビルドは終了です。

ScaLAPACK のビルド

scalapack-2.0.2 をビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf scalapack-2.0.2.tgz
(MINGW64)$ cd scalapack-2.0.2
```

サンプルの SLmake.inc.example をコピーし、環境に合わせた内容に書き換えます。

```
(MINGW64)$ cp SLmake.inc.example SLmake.inc
(MINGW64)$ vi SLmake.inc
FC            = gfortran -fopenmp -fno-range-check
CC            = gcc -fopenmp
FCFLAGS       = -O3 -I$(HOME)/Software/msmpi/Include
CCFLAGS       = -O3 -I$(HOME)/Software/msmpi/Include
FCLOADER      = $(FC) -L$(HOME)/Software/msmpi/Lib -lmsmpi
CCLOADER      = $(CC) -L$(HOME)/Software/msmpi/Lib -lmsmpi

BLASLIB       = -L$(HOME)/Software/lib -lopenblas
LAPACKLIB     =
※変更点のみ記載
```

変更が済んだらビルドをします。

```
(MINGW64)$ make
```

BLACS/TESTING 以下のビルドでエラーが出ます。

```
../../libscalapack.a(BI_Rsend.o):BI_Rsend.c:(.text+0x24): undefined reference to
`MPI_Rsend'
collect2.exe: error: ld returned 1 exit status
Makefile:18: ターゲット 'xCbtest' のレシピで失敗しました
make[2]: *** [xCbtest] エラー 1
make[2]: ディレクトリ '/home/michioga/Software/scalapack-2.0.2/BLACS/TESTING' から出ます
Makefile:8: ターゲット 'tester' のレシピで失敗しました
make[1]: *** [tester] エラー 2
make[1]: ディレクトリ '/home/michioga/Software/scalapack-2.0.2/BLACS' から出ます
Makefile:74: ターゲット 'blacsexe' のレシピで失敗しました
make: *** [blacsexe] エラー 2
```

これは無視して構いません。

以上で ScaLAPACK のビルドは完了です。

MUMPS のビルド

MUMPS_5.0.1 をビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf MUMPS_5.0.1.tar.gz
(MINGW64)$ cd MUMPS_5.0.1
```

テンプレート Makefile.inc.generic を Makefile.inc としてコピーし、環境に合わせた内容に書き換えます。

```
(MINGW64)$ cp Make.inc/Makefile.inc.generic Makefile.inc
(MINGW64)$ vi Makefile.inc
LMETISDIR = $(HOME)/Software/metis-4.0.3
IMETIS = -I$(LMETISDIR)/Lib
LMETIS = -L$(LMETISDIR) -lmetis

ORDERINGSF = -Dmetis4 -Dpord

CC = gcc -fopenmp
FC = gfortran -fopenmp -fno-range-check
FL = gfortran -fopenmp
```

```
SCALAP = -L$(HOME)/Software/scalapack-2.0.2 -lscalapack

INCPAR = -I$(HOME)/Software/msmpi/Include
LIBPAR = $(SCALAP) -L$(HOME)/Software/msmpi/Lib -lmsmpi

LIBBLAS = -L$(HOME)/Software/lib -lopenblas
LIBOTHERS = -lpthread -fopenmp

OPTF    = -O -DMUMPS_OPENMP
OPTC    = -O -I. -DMUMPS_OPENMP
OPTL    = -O
※変更点のみ記載
```

変更が済んだらビルドしします。

```
(MINGW64)$ make
```

以上で MUMPS のビルドは完了です。

ML(Trilinos)のビルド

trilinos-12.4.2 をビルドします。FrontISTR では ML を用いますが、zoltan も一緒にビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf trilinos-12.4.2-Source.tar.bz2
(MINGW64)$ cd trilinos-12.4.2-Source
(MINGW64)$ mkdir build
```

ファイルを展開したら cmake、make をして下さい。

```
(MINGW64)$ mkdir build
(MINGW54)$ cd build
(MINGW64)$ cmake -G "MSYS Makefiles" \
-DCMAKE_INSTALL_PREFIX=$HOME/Software/trilinos \
-DCMAKE_CXX_FLAGS="-DNO_TIMES" \
-DCMAKE_C_FLAGS="-DNO_TIMES" \
-DTrilinos_ENABLE_ML=ON -DTrilinos_ENABLE_Zoltan=ON -DTrilinos_ENABLE_OpenMP=ON \
-DTrilinos_ENABLE_Kokkos=OFF -DTrilinos_ENABLE_Teuchos=OFF -
DTrilinos_ENABLE_AztecOO=OFF -DTrilinos_ENABLE_Epetra=OFF \
-DTPL_BLAS_LIBRARIES=$HOME/Software/lib/libopenblas.a -
DTPL_LAPACK_LIBRARIES=$HOME/Software/lib/libopenblas.a \
```

```
-DML_ENABLE_MPI=ON \
-DTPL_ENABLE_MPI=ON \
-DMPI_CXX_COMPILER=g++ \
-DMPI_CXX_INCLUDE_PATH=$HOME/Software/msmpi/Include \
-DMPI_CXX_LIBRARIES=$HOME/Software/msmpi/Lib/libmsmpi.a \
-DMPI_C_COMPILER=gcc \
-DMPI_C_INCLUDE_PATH=$HOME/Software/msmpi/Include \
-DMPI_C_LIBRARIES=$HOME/Software/msmpi/Lib/libmsmpi.a \
..
(MINGW64)$ make
(MINGW64)$ make install
```

以上で ML(Trilinos)のビルドは完了です。

REVOCAP_Refiner のビルド

REVOCAP_Refiner-1.1.03 をビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf REVOCAP_Refiner-1.1.03.tar.gz
(MINGW64)$ cd REVOCAP_Refiner-1.1.03
```

MakefileConfig.in を環境に合わせた内容に書き換えます。

```
(MINGW64)$ vi MakefileConfig.in
ARCH = x86_64-mingw-w64
CXXFLAGS = -O -Wall $(DEBUGFLAG)
※変更点のみ記載
```

更に足りない記述を kmbMeshOperation.cpp に追加します。

```
(MINGW64)$ $ vi MeshDB/kmbMeshOperation.h
#include <cstdlib>
#include <cstring>
を追加
$ vi MeshDB/kmbMeshBrep.h
#include <cstdlib>
を追加
```

変更が済んだらビルドします。

```
(MINGW64)$ make
```

以上で REVOCAP_Refiner のビルドは完了です。

FrontISTR のビルド

FrontISTR_V44.tar.gz の中には、Version4.6(fistr2)と Version3.6(fistr1)が同梱されています。

FrontISTR はコンパイル時に有効にする機能が多数あります。今回は

- MPI
- OpenMP
- パーティショナ等のツール類
- リファイナー
- METIS
- MUMPS
- LAPACK
- ML
- paracon(並列接触解析)

を有効にします。

FrontISTR_V44 をビルドします。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ tar xvf FrontISTR_V44.tar.gz
(MINGW64)$ cd FrontISTR_V44
```

OpenMP 用パッチ(fix_omp_gfortran.patch)

OpenMP に対応した FrontISTR を gfortran でビルドするためのパッチを適用します。

このパッチは、合同会社 PExProCS(http://www.pexprocs.jp)の後藤様より提供して頂きました。

fix_omp_gfortran.patch

```
diff --git a/hecmw1/src/solver/matrix/hecmw_mat_ass.f90
b/hecmw1/src/solver/matrix/hecmw_mat_ass.f90
index e22b422..363ca2f 100644
--- a/hecmw1/src/solver/matrix/hecmw_mat_ass.f90
```

```
+++ b/hecmw1/src/solver/matrix/hecmw_mat_ass.f90
@@ -302,9 +302,9 @@ module hecmw_matrix_ass
subroutine hecmw_mat_ass_bc(hecMAT, inode, idof, RHS, conMAT)
type (hecmwST_matrix)     :: hecMAT
integer(kind=kint) :: inode, idof
-       real(kind=kreal) :: RHS
+       real(kind=kreal) :: RHS, val
type (hecmwST_matrix),optional     :: conMAT
-       integer(kind=kint) :: NDOF, in, i, ii, iii, ndof2, k, iS, iE, iiS, iiE, ik
+       integer(kind=kint) :: NDOF, in, i, ii, iii, ndof2, k, iS, iE, iiS, iiE, ik, idx

NDOF = hecMAT%NDOF
if( NDOF < idof ) return
@@ -318,13 +318,14 @@ module hecmw_matrix_ass

DO i = NDOF-1,0,-1
IF( i .NE. NDOF-idof ) THEN
+          idx = NDOF*inode-i
+          val = hecMAT%D(ndof2*inode-ii)*RHS
!$omp atomic
-          hecMAT%B(NDOF*inode-i) = hecMAT%B(NDOF*inode-i)        &
-                                 - hecMAT%D(ndof2*inode-ii)*RHS
+          hecMAT%B(idx) = hecMAT%B(idx) - val
if(present(conMAT)) then
+            val = conMAT%D(ndof2*inode-ii)*RHS
!$omp atomic
-            conMAT%B(NDOF*inode-i) = conMAT%B(NDOF*inode-i)        &
-                                   - conMAT%D(ndof2*inode-ii)*RHS
+            conMAT%B(idx) = conMAT%B(idx) - val
endif
ENDIF
ii = ii - NDOF
@@ -373,14 +374,15 @@ module hecmw_matrix_ass
if (hecMAT%itemU(ik) .eq. inode) then
iii = ndof2 - idof
DO i = NDOF-1,0,-1
+              idx = NDOF*in-i
+              val = hecMAT%AU(ndof2*ik-iii)*RHS
!$omp atomic
-              hecMAT%B(NDOF*in-i) = hecMAT%B(NDOF*in-i)      &
```

```
-                                - hecMAT%AU(ndof2*ik-iii)*RHS
+              hecMAT%B(idx) = hecMAT%B(idx) - val
hecMAT%AU(ndof2*ik-iii)= 0.d0
if(present(conMAT)) then
+                val = conMAT%AU(ndof2*ik-iii)*RHS
!$omp atomic
-                conMAT%B(NDOF*in-i) = conMAT%B(NDOF*in-i)      &
-                                    - conMAT%AU(ndof2*ik-iii)*RHS
+                conMAT%B(idx) = conMAT%B(idx) - val
conMAT%AU(ndof2*ik-iii)= 0.d0
endif
iii = iii - NDOF
@@ -412,14 +414,15 @@ module hecmw_matrix_ass
iii  = ndof2 - idof

DO i = NDOF-1, 0, -1
+              idx = NDOF*in-i
+              val = hecMAT%AL(ndof2*ik-iii)*RHS
!$omp atomic
-              hecMAT%B(NDOF*in-i) = hecMAT%B(NDOF*in-i)      &
-                                  - hecMAT%AL(ndof2*ik-iii)*RHS
+              hecMAT%B(idx) = hecMAT%B(idx) - val
hecMAT%AL(ndof2*ik-iii) = 0.d0
if(present(conMAT)) then
+                val = conMAT%AL(ndof2*ik-iii)*RHS
!$omp atomic
-                conMAT%B(NDOF*in-i) = conMAT%B(NDOF*in-i)      &
-                                    - conMAT%AL(ndof2*ik-iii)*RHS
+                conMAT%B(idx) = conMAT%B(idx) - val
conMAT%AL(ndof2*ik-iii) = 0.d0
endif
iii = iii - NDOF
```

このパッチファイルを適用します。

```
(MINGW64)$ cd $HOME/Software
(MINGW64)$ vi fix_omp_for_gfortran.txt
上記のパッチをファイルにする
(MINGW64)$ cd FrontISTR_V44
(MINGW64)$ patch -p1 < ../fix_omp_for_gfortran.txt
```

これで gfortran でも OpenMP 対応の FrontISTR をビルドできるようになりました。

ファイルの編集

テンプレート Makefile.conf.org を Makefile.conf としてコピーし、環境に合わせた内容に書き換えます。

```
(MINGW64)$ cp Makefile.conf.org Makefile.conf
(MINGW64)$ vi Makefile.conf
##################################################
#                                                #
#     Setup Configulation File for FrontISTR     #
#                                                #
##################################################

# MPI
MPIDIR          = $(HOME)/Software/msmpi
MPIBINDIR       = /c/Program\ Files/Microsoft\ MPI/Bin
MPILIBDIR       = $(MPIDIR)/Lib
MPIINCDIR       = $(MPIDIR)/Include
MPILIBS         = -lmsmpi

# for install option only
PREFIX          = $(HOME)/FrontISTR
BINDIR          = $(PREFIX)/bin
LIBDIR          = $(PREFIX)/lib
INCLUDEDIR      = $(PREFIX)/include

# Metis
METISDIR        = $(HOME)/Software/metis-4.0.3
METISLIBDIR     = $(METISDIR)
METISINCDIR     = $(METISDIR)/Lib

# ParMetis
PARMETISDIR     = $(HOME)/ParMetis-3.1
PARMETISLIBDIR  = $(PARMETISDIR)
PARMETISINCDIR  = $(PARMETISDIR)/ParMETISLib

# Refiner
REFINERDIR      = $(HOME)/Software/REVOCAP_Refiner-1.1.03
REFINERINCDIR   = $(REFINERDIR)/Refiner
REFINERLIBDIR   = $(REFINERDIR)/lib/x86_64-mingw-w64
```

```
# Coupler
REVOCAPDIR     = $(HOME)/Software/REVOCAP_Coupler-2.1
REVOCAPINCDIR  = $(REVOCAPDIR)/librcap
REVOCAPLIBDIR  = $(REVOCAPDIR)/librcap

# MUMPS
MUMPSDIR       = $(HOME)/Software/MUMPS_5.0.1
MUMPSINCDIR    = $(MUMPSDIR)/include
MUMPSLIBDIR    = $(MUMPSDIR)/lib
MUMPSLIBS      = -ldmumps -lmumps_common -lpord -L$(HOME)/Software/scalapack-2.0.2 -
lscalapack

# ML
MLDIR          = $(HOME)/Software/trilinos
MLINCDIR       = $(MLDIR)/include
MLLIBDIR       = $(MLDIR)/lib
MLLIBS         = -lml -lzoltan -lws2_32

# C compiler settings
CC             = gcc -fopenmp
CFLAGS         = -D_WINDOWS
LDFLAGS        = -L$(HOME)/Software/lib -lopenblas -lm -lstdc++
OPTFLAGS       = -O3

# C++ compiler settings
CPP            = g++ -fopenmp
CPPFLAGS       =
CPPLDFLAGS     = -L$(HOME)/Software/lib -lopenblas
CPPOPTFLAGS    = -O3

# Fortran compiler settings
F90            = gfortran -fno-range-check -fopenmp
F90FLAGS       =
F90LDFLAGS     = -L$(HOME)/Software/lib -lopenblas -lstdc++
F90OPTFLAGS    = -O2
F90LINKER      = gfortran -fopenmp

MAKE           = make
AR             = ar ruv
CP             = cp -f
```

```
RM              = rm -f
MKDIR           = mkdir -p
```

gfortran でコンパイルする時に問題になる箇所を修正します。

```
(MINGW64)$ vi fistr2/src/analysis/dynamic/transit/dynamic_output.f90
subroutine dynamic_nodal_stress_2d 内の
      real(kind=KREAL) :: s11, s22, s33, s12, s23, s13, ps, smises
の下に
      integer :: tmp1_ielem, tmp2_ielem, tmp3_ielem, tmp4_ielem
宣言を追加。


            fstrSOLID%ESTRAIN(6*ielem-5:6*ielem) = estrain
            fstrSOLID%ESTRESS(7*ielem-6:7*ielem-1) = estress
を
            tmp1_ielem = 6*ielem-5
            tmp2_ielem = 6*ielem
            fstrSOLID%ESTRAIN(tmp1_ielem:tmp2_ielem) = estrain
            tmp3_ielem = 7*ielem-6
            tmp4_ielem = 7*ielem-1
            fstrSOLID%ESTRESS(tmp3_ielem:tmp4_ielem) = estress
と変更
```

ビルド

FrontISTR をビルドします。

```
(MINGW64)$ ./setup.sh -p --with-tools --with-refiner --with-paracon \
                 --with-lapack --with-metis --with-mumps --with-ml
(MINGW64)$ make
(MINGW64)$ make install
```

$HOME/FrontISTR/bin にソルバー(fistr1.exe, fistr2.exe)やパーティショナ(hecmw_part1.exe, hecmw_part2.exe)がインストールされます。

```
(MINGW64)$ cd $HOME/FrontISTR/bin
(MINGW64)$ ls
conv2mw3.exe  hec2rcap.exe     hecmw_vis1.exe  rconv.exe
fistr1.exe    hecmw_part1.exe  hecmw_vis2.exe  rmerge.exe
fistr2.exe    hecmw_part2.exe  neu2fstr.exe
```

「MinGW-w64 Win64 Shell」内であれば、この状態で動作します。

このシェルを起動せずに実行するには、必要になる DLL をこのディレクトリコピーしておくと便利です。

```
(MINGW64)$ mkdir /c/FrontISTR
(MINGW64)$ cp $HOME/FrontISTR/bin /c/FrontISTR
(MINGW64)$ cd /mingw64/bin
(MINGW64)$ cp libgcc_s_seh-1.dll libgomp-1.dll libstdc++-6.dll $HOME/FrontISTR/bin
(MINGW64)$ cp libgfortran-3.dll libquadmath-0.dll $HOME/FrontISTR/bin
(MINGW64)$ cp libwinpthread-1.dll $HOME/FrontISTR/bin
(MINGW64)$ cd /c/FrontISTR
(MINGW64)$ ls
fistr1.exe       libgcc_s_seh-1.dll  libstdc++-6.dll      rmerge.exe
hec2rcap.exe     libgfortran-3.dll   libwinpthread-1.dll
hecmw_part1.exe  libgomp-1.dll       neu2fstr.exe
hecmw_vis1.exe   libquadmath-0.dll   rconv.exe
```

上の例では C:ドライブ直下に FrontISTR というディレクトリを作り、そこへ実行ファイルと DLL をコピーしました。

Windows の環境変数で C:\FrontISTR へのパスを加えておけば、MinGW をインストールしていない環境でも実行できます。

テスト

スタートメニューから「Windows システム ツール」⇒「コマンドプロンプト」を起動して下さい。

先ほどのディレクトリにパスを通してから、FrontISTR のチュートリアルデータがある場所に移動して下さい。

```
C:>set %PATH%=C:\FrontISTR;%PATH%
C:>cd msys64\home\<ログオン名>\Software\FrontISTR_V44\tutorial
C:>dir /w
 ドライブ C のボリューム ラベルは Windows です
 ボリューム シリアル番号は XXXX-XXXX です

 C:\msys64\home\<ログオン名>\Software\FrontISTR_V44\tutorial のディレクトリ

[.]                              [..]
[01_elastic_hinge]               [02_elastic_hinge_parallel]
[02_elastic_hinge_parallel_02] [03_hyperelastic_cylinder]
```

```
[04_hyperelastic_spring]       [05_plastic_cylinder]
[06_plastic_can]               [07_viscoelastic_cylinder]
[08_creep_cylinder]            [09_contact_hertz]
[10_contact_2tubes]            [11_contact_2beam]
[12_dynamic_beam]              [13_dynamic_beam_nonlinear]
[14_dynamic_plate_contact]     [15_eigen_spring]
[16_heat_block]                [17_freq_beam]
```

この中の 01_elastic_hinge と 02_elastic_hinge_parallel を実行してみて下さい。

```
> cd 01_elastic_hinge
> dir
2015/12/04  17:19    <DIR>          .
2015/12/04  17:19    <DIR>          ..
2015/02/16  18:04               183 hecmw_ctrl.dat
2015/02/16  18:04               511 hinge.cnt
2015/02/16  18:04         9,266,446 hinge.msh
               3 個のファイル           9,267,140 バイト
               2 個のディレクトリ   XXX,XXX,XXX,XXX バイトの空き領域
```

正しくパスが通っていれば fistr1 と打つだけで解析が始まります。

```
> fistr1
Step control not defined! Using default step=1
 fstr_setup: OK
### 3x3 B-SSOR-CG(0)  1
      1    1.903375E+00
      2    1.974378E+00
...
...
   2967    1.072387E-08
   2968    9.994170E-09
### Relative residual = 1.02411E-08

### summary of linear solver
      2968 iterations      9.994170E-09
    set-up time      :     2.124152E-01
    solver time      :     7.884661E+01
    solver/comm time :     4.373577E-02
    solver/matvec    :     3.441607E+01
    solver/precond   :     3.714435E+01
```

```
    solver/1 iter    :     2.656557E-02
    work ratio (%)   :     9.994453E+01


Start visualize PSF 1 at timestep 1


 ====================================
    TOTAL TIME (sec) :     82.02
           pre (sec) :      0.85
         solve (sec) :     81.17
 ====================================
 FrontISTR Completed !!
>
```

何も指定していなければ、コンピュータに搭載されたコア数の OpenMP スレッドで解析をします。タスクマネージャで確認してみてください。

スレッド数を指定する場合は、OMP_NUM_THREADS という環境変数を設定します。

```
> set OMP_NUM_THREADS=4
> echo %OMP_NUM_THREADS%
4
> fistr1
```

Hyper-threading が ON になってと実際の倍のコアが有るように見えますが、スレッドの数は実際のコア数に指定するのが一番効率が良いようです。

次に MPI の実行例を示します。最初に領域分割をします。

```
> cd ..\02_elastic_hinge_parallel
> hecmw_part1
Dec 04 17:38:31 Info: Reading mesh file...
Dec 04 17:38:31 Info: Starting domain decomposition...
Dec 04 17:38:31 Info: Creating local mesh for domain #0 ...
Dec 04 17:38:32 Info: Creating local mesh for domain #1 ...
Dec 04 17:38:32 Info: Creating local mesh for domain #2 ...
Dec 04 17:38:32 Info: Creating local mesh for domain #3 ...
Dec 04 17:38:32 Info: Domain decomposition done
>
```

MPI でソルバを実行するときは、OpenMP スレッドの数を 1 つにした方が良いようです(環境によります)。

Microsoft MPI は mpirun へのエイリアスがありません。mpiexec を指定して実行します。

```
> set OMP_NUM_THREADS=1
> mpiexec -n 4 -cores 1 fistr1
 Step control not defined! Using default step=1
 Step control not defined! Using default step=1
 Step control not defined! Using default step=1
 Step control not defined! Using default step=1
 fstr_setup: OK
 fstr_setup: OK
 fstr_setup: OK
 fstr_setup: OK
### 3x3 B-SSOR-CG(0)  2
      1    2.179037E+00
      2    2.412720E+00
...
...
   2084    1.003548E-08
   2085    9.112093E-09
### Relative residual = 9.29985E-09

### summary of linear solver
      2085 iterations      9.112093E-09
    set-up time      :     1.456044E-02
    solver time      :     9.929363E+01
    solver/comm time :     2.988770E+00
    solver/matvec    :     4.605539E+01
    solver/precond   :     4.684918E+01
    solver/1 iter    :     4.762284E-02
    work ratio (%)   :     9.698997E+01


 ====================================
    TOTAL TIME (sec) :    101.14
           pre (sec) :      0.32
         solve (sec) :    100.82
 ====================================
 FrontISTR Completed !!
Start visualize PSF 1 at timestep 1
```

```
>
```

ディレクトリ内を見ると結果が出力されているのが分かります。

```
> dir /w
[.]                     [..]                    0.log
1.log                   2.log                   3.log
FSTR.dbg.0              FSTR.dbg.1              FSTR.dbg.2
FSTR.dbg.3              FSTR.msg                FSTR.sta
hecmw_ctrl.dat          hecmw_part.log          hecmw_part_ctrl.dat
hecmw_vis.ini           hinge.cnt               hinge.msh
hinge.res.0.1           hinge.res.1.1           hinge.res.2.1
hinge.res.3.1           hinge_4.0               hinge_4.1
hinge_4.2               hinge_4.3               hinge_vis_psf.0001.inp
part.inp
>
```

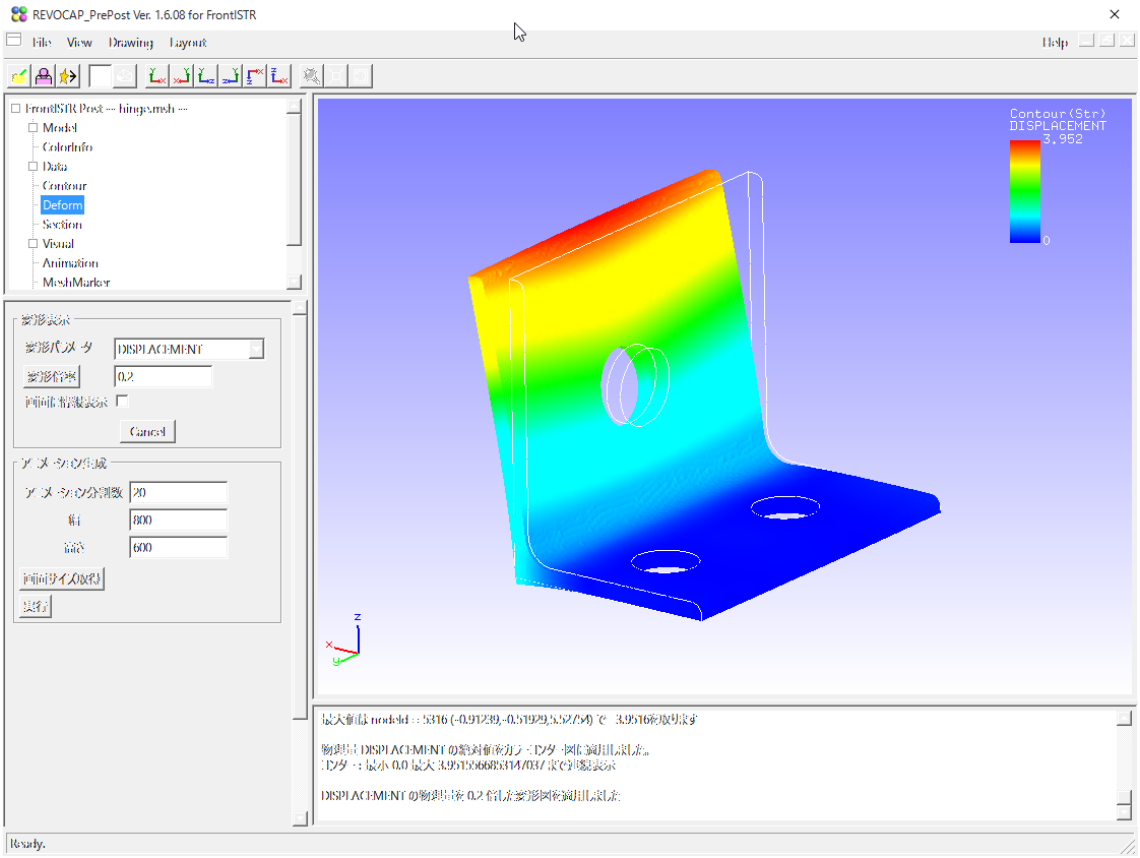
hinge.cnt 内の!output_type を下の様に書き換えて下さい。

```
!output_type = COMPLETE_AVS
を
!output_type = COMPLETE_REORDER_AVS
```

変更後 fistr1 を実行すると、解析結果を ParaView からも見ることが出来ます。

REVOCAP_PrePost はどちらの形式にも対応しているので、変更の必要はありません。

REVOCAP_PrePost での可視化結果は下の図のようになります。



ParaView ではこのように表示されます。