RSS2108-PJ7-ﾕｰｻﾞﾏﾆｭｱﾙ-001b

文部科学省次世代ＩＴ基盤構築のための研究開発
「革新的シミュレーションソフトウエアの研究開発」

RSS21 フリーソフトウエア

HEC ミドルウェア（HEC-MW）

PC クラスタ用ライブラリ型 HEC-MW

（hecmw-PC-cluster） バージョン 2.01

HEC-MW を用いたプログラム作成手法

お問い合わせ先
（公開／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 計算科学技術連携研究センター
〒153-8505 東京都目黒区駒場４－６－１
Fax：03-5452-6662
E-mail：software@rss21.iis.u-tokyo.ac.jp

目　　次

1. はじめに

HEC-MW のほとんどの機能は API として提供されています。ユーザは、必要な機能が HEC－MW で提供されていれば、それをサブルーチンコールとして利用することができます。

以下では、HEC-MW を利用したプログラムの作成方法を紹介します。

2. コンパイル・リンク方法

HEC-MW ライブラリを使用したプログラムのコンパイル方法を示します。ただし、HEC-MW のインストールは既に完了していることを前提としています。インストール方法の詳細については、「HEC-MW インストールマニュアル」をご覧ください。

インストールされた HEC-MW の各ディレクトリが以下のようになっているとします。

HEC-MW インクルードディレクトリ	/usr/local/hecmw/include
HEC-MW ライブラリディレクトリ	/usr/local/hecmw/lib

また、HEC-MW をコンパイルしたコンパイラは、以下のものであったとします。

Fortran コンパイラ	mpif90

この場合、HEC-MW を利用したプログラム"test.f90"は、

```
mpif90 test.f90 -I/usr/local/hecmw/include -L/usr/local/hecmw/lib -lfhecmw -lhecmw
```

でコンパイル・リンクできます。

Fortran コンパイラが mpif90 でない場合、MPI のインクルードディレクトリとライブラリディレクトリ、ライブラリを明示する必要があります。開発環境における MPI の利用方法を確認してください。

3. 最小の HEC-MW 利用プログラム

HEC-MW を利用するためには以下のことをプログラム内で行わなければなりません。

1. hecmw モジュールを use する
2. プログラムの最初に hecmw_init をコールする
3. プログラムの最後で hecmw_finalize をコールする

これらに従って記述したプログラムは、以下のようになります。

test.f90

```
program test
  use hecmw
  call hecmw_init
  call hecmw_finalize
end program test
```

まず、hecmw というモジュールを use して、HEC-MW で提供される全てのサブルーチンを利用可能にします。このモジュールにより、プログラム開発者は HEC-MW のサブルーチンがどのモジュールに属するかを気にする必要がなくなります。

次に、プログラムの開始直後にサブルーチン hecmw_init をコールします。これは、HEC-MW を利用するために必要不可欠な初期化処理を行います。

そして最後に、サブルーチン hecmw_finalize をコールします。この呼び出し以降、何らかの処理を行うことは推奨されません。

以上が、HEC-MW を利用した最も小さなプログラムです。このプログラムは、初期化と終了処理以外何もしていません。

実際に、コンパイル・リンクを行います。

```
mpif90 test.f90 -I/usr/local/hecmw/include -L/usr/local/hecmw/lib -lfhecmw -lhecmw
```

これで、実行モジュールが生成されます。しかし、全体制御ファイルが存在しないため正常に実行できません。

HEC-MW では、実行時カレントディレクトリに必ず全体制御ファイルが必要となります。

とりあえずこのプログラムを実行するために、”hecmw_ctrl.dat”という空のファイルを作成しておきまず。

```
touch hecmw_ctrl.dat
```

この状態でプログラムを実行すると、何も表示しないで正常終了します。

4. メッシュデータ入力方法

ここでは、最小の HEC-MW 利用プログラムに修正を加えて、メッシュデータの入力を可能にします。

修正を加えたプログラムを以下に示します。

```
program test
  use hecmw
  implicit none
  character(len=HECMW_NAME_LEN) :: name_ID
  type(hecmwST_local_mesh) :: mesh
  call hecmw_init
  name_ID = 'test'
  call hecmw_get_mesh(name_ID, mesh)
  call hecmw_dist_free(mesh)
  call hecmw_finalize
end program test
```

このプログラム実行時に必要となる全体制御ファイルの内容を示します。

```
!MESH, NAME=test, TYPE=HECMW-ENTIRE
mesh.dat
```

修正プログラムには、サブルーチン hecmw_get_mesh が追加されています。このサブルーチンは、メッシュデータをファイルから読み込みます。

ここで注意すべきことは、引数 name_ID に指定されている文字列がファイル名ではないことです。name_ID は、全体制御ファイルで定義されている!MESH の NAME 識別子を表しています。よって、この場合、実際に読み込まれるファイルは"mesh.dat"となります。

Note:

!MESH で指定するファイル名は、初期メッシュデータについては全て実際のファイル名ですが、HEC-MW 分散メッシュデータに関しては、ファイル名から末尾の「.<rank>」を除いたものとなります。従って、分散メッシュファイルのファイル名は、「全体制御ファイルから得られたファイル名.<rank>」でなければなりません。

これにより、プログラムが並列に動作している場合は、各プロセスが入力するファイルをファイル名により区別することができます。

また、TYPE には HECMW-ENTIRE と記述されています。これは、"mesh.dat"が HEC-MW

単一領域メッシュデータであることを示しています。

TYPE には以下を指定することが可能です。

HECMW-DIST	HEC-MW 分散メッシュデータ
HECMW-ENTIRE	HEC-MW 単一領域メッシュデータ
GEOFEM	GeoFEM データ

これらの指定は全て全体制御ファイルで行うため、メッシュデータ入力のためのサブルーチンをタイプごとに記述する必要はありません。全て、hecmw_get_mesh で読み込むことができます。

読み込まれたメッシュデータは、hecmw_get_mesh の第 2 引数 mesh で示される変数に格納されます。データ格納のために必要なメモリ領域は、hecmw_get_mesh 内で自動的に確保されます。

分散メッシュデータ構造体 hecmwST_local_mesh は、HEC-MW が内部的に使用するデータ構造です。メッシュデータからの情報は全てこの構造体に格納され、その後の処理に利用されます。プログラム開発者もその情報を利用することができます。

以後、hecmw_get_mesh によって取得した分散メッシュデータ構造体をもとに、様々な処理を行うことになります。

また、サブルーチン hecmw_dist_free がコールされていますが、これは分散メッシュデータ構造体に確保されているメモリ領域を開放するためのものです。

5. メッシュデータ出力方法

この章では、メッシュデータ入力プログラムに修正を加え、メッシュデータの出力を可能にします。

修正を加えたプログラムを以下に示します。

```
program test
  use hecmw
  implicit noen
  character(len=HECMW_NAME_LEN) :: name_ID
  type(hecmwST_local_mesh) :: mesh
  call hecmw_init
  name_ID = 'test'
  call hecmw_get_mesh(name_ID, mesh)
  name_ID = 'test-out'
  call hecmw_put_mesh(name_ID, mesh)
  call hecmw_dist_free(mesh)
  call hecmw_finalize
end program test
```

新たに、サブルーチン hecmw_put_mesh が加わりました。これにより、第 2 引数に渡されている分散メッシュデータ構造体に格納されているデータがファイルに出力されます。

hecmw_put_mesh では、ファイルに出力を行うため、そのファイルを全体制御ファイルで定義しておく必要があります。

全体制御ファイルの内容を以下に示します。

```
!MESH, NAME=test, TYPE=HECMW-ENTIRE
mesh.dat

!MESH, NAME=test-out, TYPE=HECMW-DIST
mesh.out
```

入力ファイル指定のための!MESH(NAME は"test"）に加えて、出力ファイル指定のための!MESH（NAME は"test-out"）が定義されていることが分かります。

出力ファイルは、必ず分散メッシュデータとして出力されます。従って、全体制御ファイルの TYPE 指定にも HECMW-DIST を指定しなければなりません。

また、出力されるファイル名は、全体制御ファイルから取得したファイル名に「.<rank>」を付加したものとなります。これにより、プログラムが並列に動作している場合は、複数の分散ファイルが出力されることになります。

6. 結果データ出力方法

解析処理を行った後に、結果データを POST 処理用に残す場合があります。HEC-MW では、結果データをファイルに出力する方法を提供しています。

結果データ出力のサンプルプログラムの一部を以下に示します。

```
program test
  use hecmw
  implicit none
  character(len=HECMW_NAME_LEN) :: name_ID
  type(hecmwST_local_mesh) :: mesh
  integer(kind=kint) :: tstep, n_dof
  character(len=HECMW_HEADER_LEN) :: header
  character(len=HECMW_NAME_LEN) :: label
  real(kind=kreal),dimension(:),pointer :: displacement
  real(kind=kreal),dimension(:),pointer :: strain
  real(kind=kreal),dimension(:),pointer :: stress

  ...

  header = 'result sample'
  call hecmw_result_init(mesh, tstep, header)
  label = 'DISPLACEENT'
  n_dof = 3
  call hecmw_result_add(1, n_dof, label, displacement)
  label = 'STRAIN'
  n_dof = 6
```

```
  call hecmw_result_add(1, n_dof, label, strain)
  label = 'STRESS'
  n_dof = 7
  call hecmw_result_add(1, n_dof, label, stress)
  call hecmw_result_write()
  call hecmw_result_finalize()

  ...

end program test
```

全体制御ファイルの内容のうち、関係する部分を以下に示します。

```
!RESULT, NAME=result-out, IO=OUT
result.out
```

結果データの出力を容易にするために、HEC-MW では結果データ出力用サブルーチンを用意しています。それらのサブルーチンによって、以下の処理を行います。

1.	hecmw_result_init	結果データ出力準備
2.	hecmw_result_add	結果データ指定
3.	hecmw_result_write	結果データ
4.	hecmw_result_finalize	結果データ出力終了

まず、hecmw_result_init によって初期化を行います。これは、結果データの出力に必要な情報を事前に取得する役目があります。

取得するデータは以下のとおりです。

メッシュデータ
タイムステップ
結果ファイルヘッダ

結果ファイルヘッダは､結果ファイルの識別しやすくするために指定する任意の文字列であり、結果ファイルの先頭行に出力されます。

初期化が終了した後は、hecmw_result_add によって、出力したい結果データの情報を指定します。指定する内容は以下のとおりです。

指定する結果データは節点値なのか、要素値なのか（節点：1, 要素：2）
自由度数
結果データにつけるラベル
結果データそのもの（浮動小数点型一次元配列）

この指定は複数回行うことができます。

また、指定の時点ではファイルへの出力は行われず、HEC-MW 内部に情報が蓄積されるだけです。従って、ここで指定したデータの内容は、実際にファイルへの出力が終了するまで変更してはなりません。

出力したい結果データを全て指定し終わったら、hocmw_result_write で実際にファイルに出力します。

ここで、出力するファイル名は全体制御ファイルから取得します。取得するファイル名は、!RESULT で IO パラメータが OUT のもののうち、最初に定義されているものとなります。もし、出力ファイルを明示的に指定したいのなら、hecmw_result_write_by_name を使用して、引数に name_ID を指定すると、定義されている!RESULT から NAME が該当するものを取得することができます。この場合、IO パラメータは無視されます。

実際に出力するファイルの名前は、取得したファイル名に「.<rank>.<tstep>」を付加したものとなります。

最後に、後始末として hecmw_result_finalize をコールします。これによって、hecmw_result_add された情報がクリアされます。

7. 結果データ入力方法

結果データをファイルから入力するのは非常に簡単です。

サンプルプログラムの一部を以下に示します。

```
...
integer(kind=kint) :: tstep
type(hecmwST_result_data) :: result
...
call hecmw_result_read(tstep, result)
...
```

全体制御データの内容のうち、関係するものを以下に示します。

```
!RESULT, NAME=result-in, IO=IN
result.in
```

結果データをファイルから入力するためには、hpmcw_result_read をコールするだけです。その結果、結果データは結果データ構造体に格納され、返されます。

結果データ構造体は以下のようになっています。

```
type hecmwST_result_data
    integer(kind=kint) :: nn_component
    integer(kind=kint) :: ne_component
    integer(kind=kint),pointer :: nn_dof(:)
    integer(kind=kint),pointer :: ne_dof(:)
    character(len=HECMW_NAME_LEN),pointer :: node_label(:)
    character(len=HECMW_NAME_LEN),pointer :: elem_label(:)
    real(kind=kreal),pointer :: node_val_item(:)
    real(kind=kreal),pointer :: elem_val_item(:)
end type hecmwST_result_data
```

変数名	内容
Nn_component	節点コンポーネント数
Ne_component	要素コンポーネント数
Nn_dof	節点自由度数
Ne_dof	要素自由度数
node_label	節点ラベル
elem_label	要素ラベル
node_val_item	節点値
elem_val_item	要素値

node_val_item の並びは、節点ごとに全コンポーネントが順に格納されている。

elem_val_item の並びは、要素ごとに全コンポーネントが順に格納されている。

結果ファイルのデータは全てこの構造体に格納されるので、これを参照して処理を行うことができます。

8. リスタートデータ出力方法

HEC-MW では、リスタートデータを出力する方法を提供しています。

リスタートファイルに出力可能なデータは特に決まっておらず、ユーザが自由に選択することができます。

リスタートデータ出力の手順を以下に示します。

1. ファイルに出力したいデータを指定する（複数回可能）
2. ファイルに出力する

以下にサンプルプログラムの一部を示します。

```
...

integer(kind=kint),pointer,dimension(:) ::  int_data
real(kind=kreal),pointer,dimension(:) ::  real_data
real(kind=kreal),pointer,dimension(:) ::  real_data2

...

allocate(int_data(100))
allocate(real_data(200))
allocate(real_data2(300))

...

call hecmw_restart_add_int(int_data, size(int_data))
call hecmw_restart_add_real(real_data, size(real_data))
call hecmw_restart_add_real(real_data2, size(real_data2))
call hecmw_restart_write()

...
```

全体制御ファイルを以下に示します。

```
!RESTART,NAME=restart-out,IO=OUT
restart.out
```

サブルーチン hecmw_restart_add_int,hecmw_restart_add_real は、それぞれ第 1 引数に整数型、浮動小数点型の一次元配列をリスタートデータとして指定することができます。これらで指定された情報は、HEC-MW 内部に蓄えられ、ファイルに書き出されるまで保持されます。従って、実際にファイルに出力されるまで指定したデータを書き換えてはいけません。

また、第 2 引数には、指定した配列の要素数を与えます。

hecmw_restart_add_int, hecmw_restart_add_real で指定されたデータは、hecmw_restart_write がコールされた時点でファイルに出力されます。

このとき出力されるファイルの名前は、全体制御ファイルから取得されます。実際に出力されるファイルの名前は、「全体制御ファイルから取得したファイル名.<rank>」となります。hecmw_rsetart_write で取得されるファイル名は、!RESTART で、IO パラメータが OUT または INOUT のもののうち、最初に定義されたものです。hecmw_restart_write_by_name を使用すると、NAME が指定できます。この場合、IO パラメータは無視されます。

9. リスタートデータ入力方法

リスタートファイルからリスタートデータを入力するには、以下のようにします。

1. リスタートファイルをオープンする
2. リスタートファイルから入力（出力した時と同じ回数）
3. リスタートファイルをクローズする

以下にサンプルプログラムの一部を示します。

```
...
integer(kind=kint),pointer,dimension(:) :: int_data
real(kind=kreal),pointer,dimension(:) ::  real_data
real(kind=kreal),pointer,dimension(:) ::  real_data2

allocate(int_data(100))
allocate(real_data(200))
allocate(real_data2(300))
call hecmw_restart_open()
call hecmw_restart_read_int(int_data)
call hecmw_restart_read_real(real_data)
call hecmw_restart_read_real(real_data2)
call hecmw_restart_close()

...
```

以下に全体制御ファイルを示します。

```
!RESTART,NAME=restart-in,IO=IN
restart.in
```

リスタートファイルのオープンは、hecmw_restart_open で行います。

このときオープンされるファイルは、!RESTART で IO パラメータが IN または INOUT のもののうち、最初に定義されているものとなります。hecmw_restart_open_by_name を使用すれば、NAME 指定ができます。この場合、IO パラメータは無視されます。

オープンがすんだ後は、hecmw_restart_read_int, hecmw_restart_read_real によって、実際にファイルからデータを入力し、変数に格納していきます。このとき注意すべきことは、リスタートの入力と出力は整合性を取らなければならないということです。つまり、出力と入力の順番、型、配列要素数は全て一致していなければなりません。また、格納先の領域は事前にユーザが確保しておく必要があります。

全てのリスタートデータを入力し終えたら、hecmw_restart_close によってファイルを閉じます。

10. 可視化方法（メモリ渡し）

可視化には、ファイル渡しとメモリ渡しの二通りの方法があります。

ファイル渡しは、解析の結果データを結果ファイルとしてファイルに出力しておき、それを別プロセスが可視化しますが、メモリ渡しでは、解析終了後、メモリ上に存在している結果データを使用して同一プロセスで可視化まで行います。

可視化の手順は以下のとおりです。

1. 可視化データ構造体の作成
2. 可視化初期化
3. 可視化
4. 可視化終了処理

ここでは可視化の詳細には触れませんが、以上の手順を踏めばよいことになります。

サンプルプログラムの一部を以下に示します。

```
...
integer(kind=kint) :: step, max_step, is_force
type(hecmwST_result_data) :: result
...
call hecmw_visualize_init
call hecmw_visualize(mesh, result, step,max_step,is_force)
call hecmw_visualize_finalize
...
```

まず、可視化を行うために結果データが必要です。この結果データは、ユーザが結果データ構造体に格納しておく必要があります。

hecmw_visualize_init によって可視化の初期化を行います。

次に、hecmw_visualize をコールすると、実際に可視化が行われます。

可視化実行時に与える情報は以下のとおりです。

メッシュデータ

結果データ

タイムステップ

最大タイムステップ

最後のステップで強制的に描くかどうかのフラグ

最後に可視化の後始末として、hecmw_visualize_finalize をコールします。
以上で、可視化が終了します。