MPIによる並列アプリケーション開発法入門(III)

2010年5月13日・14日

中島研吾
東京大学情報基盤センター

T2Kオープンスパコン(東大)並列プログラミング講習会(試行)

方　針

- 〔II〕で定義した局所分散データ構造
- MPIの処理をできるだけ「隠蔽」
  - 初期化等環境設定
  - 通信
- hpcmw_eps_fvm_・・・という関数名
  - HPC-MW(HPC Middlewareに由来)
    - マルチフィジックスシミュレーション向け大規模並列計算コード開発基盤
  - 並列アプリケーションにおける並列処理のソフトウェア的な隠蔽
    - http://hpcmw.tokyo.rist.or.jp/

マルチフィジックスシミュレーション向け大規模並列計算コード開発基盤

- マルチフィジックス・マルチスケールの並列大規模連成シミュレーションを円滑に実施するためのフレームワーク
- 連立一次方程式ソルバー，形状処理，可視化，コード間連成などの共通処理に関する機能を提供し，並列大規模連成シミュレーションコードを開発するための支援環境
- PC上で開発された個別のプログラムを「plug-in」すれば，PCクラスタから「地球シミュレータ」，「ペタスケール計算機」まで様々なハードウェアに対して最適化された並列プログラムが自動的に生成される・・・というのが理想
  - HPC-MW(Middleware)，HEC-MW

大規模並列計算コード開発基盤
これまで関連して来たプロジェクト

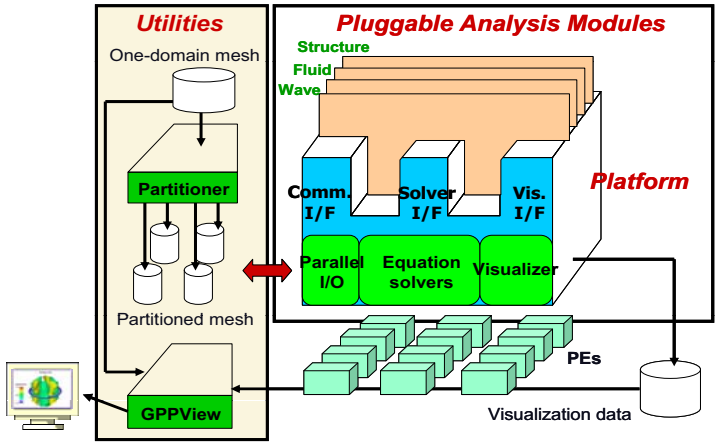
- **GeoFEM(FY.1998-FY.2002)**
  - 科学技術振興調整費　科学技術総合研究委託「高精度の固体地球変動予測のための並列ソフトウェアの開発に関する研究」
  - 固体地球シミュレーション用並列有限要素法プラットフォーム
  - 地球シミュレータ
- **HPC-MW(FY.2002-FY.2003)(プロジェクトはFY.2007迄)**
  - 文部科学省ITプログラム「戦略的基盤ソフトウェアの開発」
    - RSS21「革新的シミュレーションソフトウェアの研究開発(FY.2005-)」
- **観測・計算を融合した階層連結地震・津波予測システム(FY.2005-FY.2010(予定))**
  - 科学技術振興機構戦略的創造研究推進事業(CREST)
  - 「階層連結シミュレーション」，「連成」を重視

GeoFEM: FY.1998-2002

http://geofem.tokyo.rist.or.jp/

- 文部科学省「科学技術振興調整費総合研究」
  - 「高精度の地球変動予測のための並列ソフトウェア開発に関する研究」の一部
  - リーダー：奥田洋司教授（東大・人工物）
- 固体地球シミュレーション用並列有限要素法プラットフォーム
  - 並列I/O，並列線形ソルバー，並列可視化をサポート
- HPCと自然科学の緊密な協力

GeoFEM：「plug-in」の発想

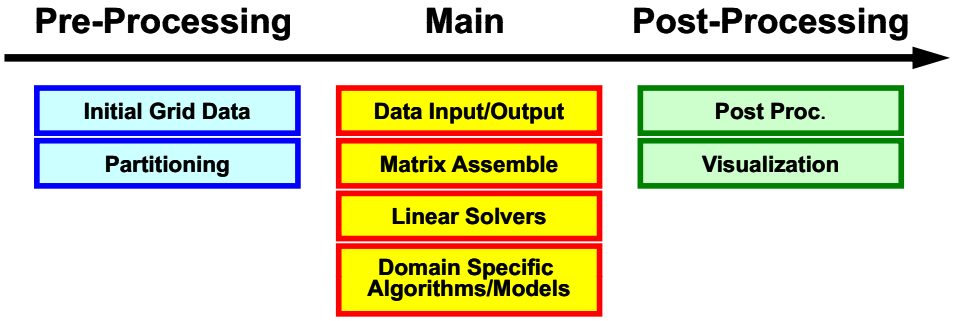


HPC-MW

PCクラスタから「地球シミュレータ」まで

- GeoFEMにおける「Plug-in」のアイディアに基づく
- 科学技術計算（特に有限要素法）における共通プロセスの最適化チューニング，並列化等のライブラリ化による隠蔽
- 「HPC-MW」によって，PC上で開発されたコードを，PCクラスタから「ES」まで，様々なハードウェアで最適な性能によって稼動させることができる
  - GeoFEMで「ES」向け最適化をやったのが契機
  - 当時は「ベクトル」，「スカラー」が並立しそうな気配もあり
- HPC-MWはGeoFEMと比較して，より広い機能をカバーし，さらに各ハードウェアへの最適化の考えを導入している

並列有限要素法のプロセス

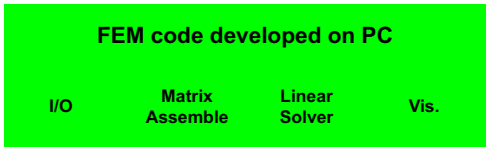


HPC-MWのサポートする機能

- データ入出力
- 適応格子，動的負荷分散
- 並列可視化
- 線形ソルバー
- 有限要素処理（コネクティビティ処理，係数行列生成）
- カップリング
- 関連ユーティリティ（領域分割等）

HPC-MWの利用イメージ

PC（単独CPU）上で開発されたFEMコード

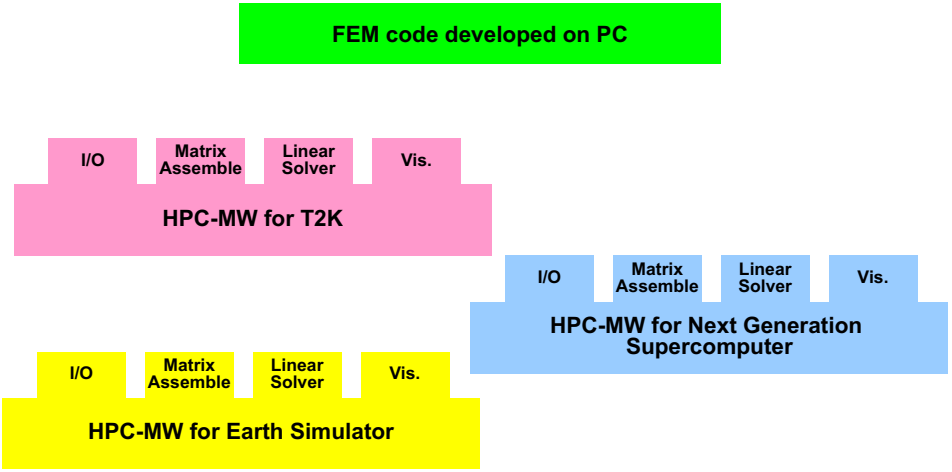


HPC-MWの利用イメージ

HPC-MWを使用する場合，共通部分は不要

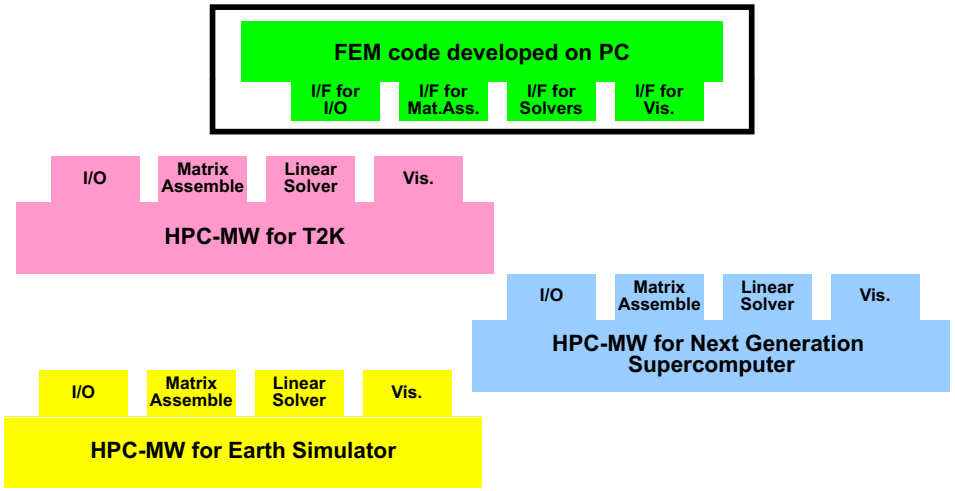
HPC-MWの利用イメージ

各H/W用に最適化されたライブラリ



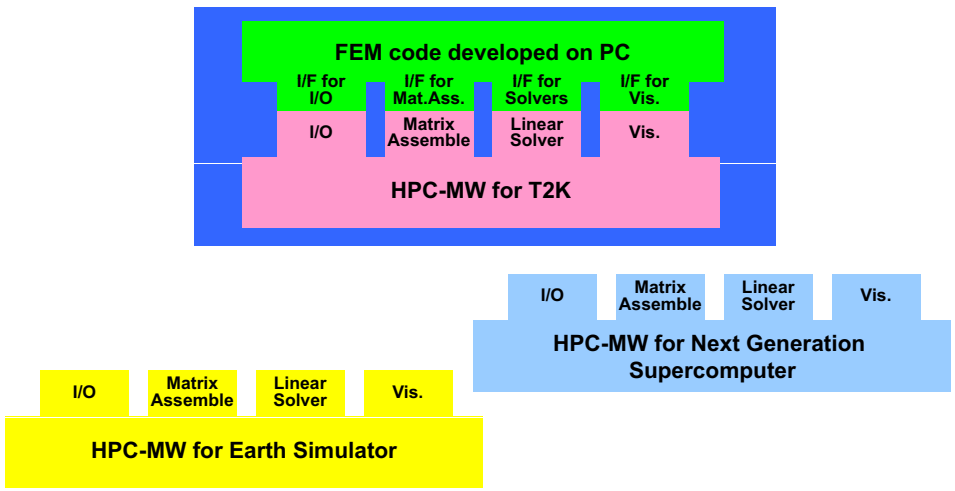
HPC-MWの利用イメージ

各ライブラリに対して同じインタフェース

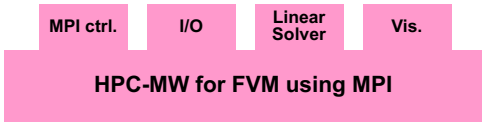


HPC-MWの利用イメージ

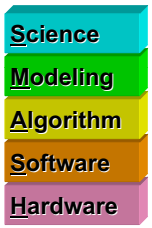
「地球シミュレータ」用最適化コード



hpcmw_eps_fvm



- 今回は，このようなミドルウェア的な機能も想定して並列プログラムを開発している
  - ミドルウェアとして切り離し，他の並列アプリケーションにも使いまわせるような機能
- SMASHのSHをカバー
- hpcmw_eps_fvmで始まる関数群
  - MPI関連
  - I/O



プログラム類のインストール

- ソースコード
  - `<$FVM>/src`
  - ここで「`make`」すると「`<$FVM>/run`」 に「`sol`」という実行形式ができる。以下これを使用する。

```
$> cd <$FVM>/src
$> make
$> ls -l ../run/sol
     sol
```

- チュートリアル

http://nkl.cc.u-tokyo.ac.jp/tutorial/parallel_lib_tutorial/

http://nkl.cc.u-tokyo.ac.jp/tutorial/parallel_lib_tutorial.tar

ファイル準備

```
$> cd <$FVM>/ex
$> cat fvmmg.ctrl
   32  32  32
$> cat fvmpart.ctrl
   !INITIAL FILE
   fvm_entire_mesh.dat

   !METHOD
   RCB
   X,Y,Z

   !REGION NUMBER
   8

   !MESH FILE
   mesh.rcb

   !COMMUNICATION FILE
   comm.rcb

   !UCD
   32-32-32-rcb-8.inp
$> eps_fvm_mg
$> eps_fvm_part
```

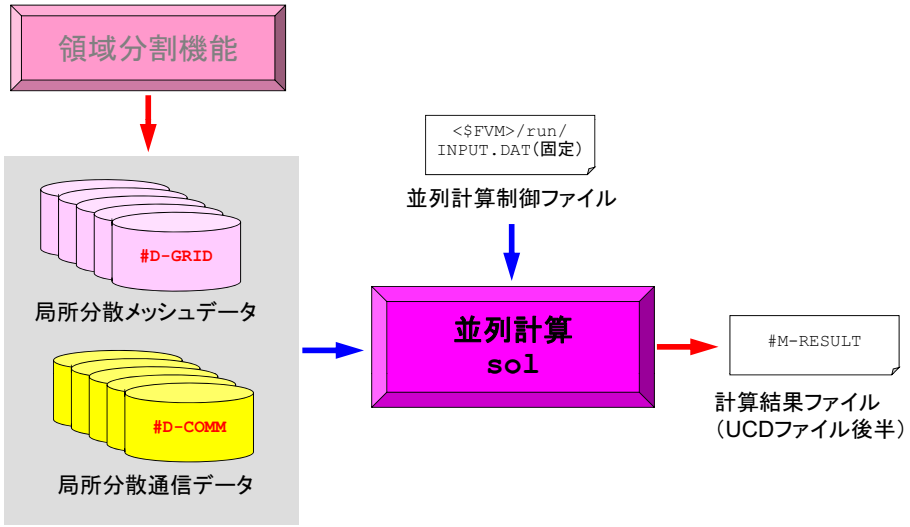
```
$> ls -l mesh.rcb.*
   mesh.rcb.0 … mesh.rcb.7
$> ls -l comm.rcb.*
   comm.rcb.0 … comm.rcb.7
```


並列計算制御ファイル

- **INPUT.DAT**（名称固定）
- 実行形式「**sol**」と同じディレクトリになければならない（この場合は**<$FVM>/run**）。
- 全ての項目は省略不可。

```
../ex/mesh.rcb   局所分散メッシュファイルのヘッダ名
../ex/comm.rcb   局所分散通信ファイルのヘッダ名
../ex/result     可視化用出力ファイル名（後述）
1                可視化用出力の有無（=1のとき出力）
```


並列シミュレーションにおけるI/O



計算実行

```
$> cd <$FVM>/run
$> cat INPUT.DAT
   ../ex/mesh.rcb
   ../ex/comm.rcb
   ../ex/result
   1

「go.sh」を書き換える

$> qsub go.sh
$> ls -l ../ex/result
   result
```

「eps_fvm」の並列化：変更点：test.f

```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

- 実はほとんど無い
- 通信関連ファイル読み込み
- 内点，外点
  - 内点　　intNODE_tot
  - 内点＋外点　NODE_tot
- MPIコールはできるだけ隠蔽
  - 初期化，Finalize
    - hpcmw_eps_fvm_util.*
  - 通信用サブルーチン群
    - hpcmw_eps_fvm_comm.*


```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


変数ブロック：hpcmw_eps_fvm_all

```
!C
!C***
!C*** hpcmw_eps_fvm_all
!C***
!C
      module hpcmw_eps_fvm_all
        use hpcmw_eps_fvm_util
        use hpcmw_eps_fvm_pcg
        use appl_cntl
      end module hpcmw_eps_fvm_all
```


module hpcmw_eps_fvm_util

- 変数ブロック
  - メッシュ
  - 通信
- MPI初期化・終了等に関連したサブルーチン群
  - hpcmw_eps_fvm_init
    - MPI_Init
  - hpcmw_eps_fvm_finalize
    - MPI_Finalize
  - hpcmw_eps_fvm_abort
    - MPI_Abort
  - hpcmw_eps_fvm_define_file_name
    - 分散ファイル名定義

hpcmw_eps_fvm_util（1/3）メッシュ関連

変数名	型	配列サイズ	内 容
NODE_tot	I	-	内点数＋外点数
intNODE_tot	**I**	**-**	**内点数**
NODE_GLOBAL(:)	**I**	**NODE_tot**	**グローバル要素番号**
NODE_VOL(:)	R	NODE_tot	要素体積
NODE_COND(:)	R	NODE_tot	要素熱伝導率
NODE_XYZ(:)	R	3*NODE_tot	要素重心座標（3次元）
CONN_tot	I	-	コネクティビティ総数
CONN_node(:)	I	2*CONN_tot	コネクティビティ構成要素
CONN_COEF(:)	R	CONN_tot	コネクティビティ係数
FIX_NODE_tot	I	-	ディリクレ境界条件適用要素数
FIX_NODE_ID(:)	I	FIX_NODE_tot	ディリクレ境界条件適用要素番号
FIX_NODE_COEF(:)	R	FIX_NODE_tot	ディリクレ境界条件係数
FIX_NODE_VAL(:)	R	FIX_NODE_tot	ディリクレ境界条件値
SURF_NODE_tot	I	-	ノイマン境界条件適用要素数
SURF_NODE_ID(:)	I	SURF_NODE_tot	ノイマン境界条件適用要素番号
SURF_NODE_FLUX(:)	R	SURF_NODE_tot	ノイマン境界条件フラックス
BODY_NODE_tot	I	-	体積発熱境界条件適用要素数
BODY_NODE_ID(:)	I	BODY_NODE_tot	体積発熱境界条件適用要素番号
BODY_NODE_FLUX(:)	R	BODY_NODE_tot	体積発熱境界条件フラックス

hpcmw_eps_fvm_util（2/3）通信関連

変数名	型	配列サイズ	内 容
PETOT	I	-	プロセッサ数
errno	I	-	エラーもどり値
my_rank	I	-	ランク番号
n_neighbor_pe	I	-	隣接領域数
neighbor_pe(:)	I	n_neighbor_pe	隣接領域ID
import_index(:)	I	0:n_neighbor_pe	受信テーブル用インデックス
import_item(:)	I	import_index(n_neighbor_pe)	受信テーブル
export_index(:)	I	0:n_neighbor_pe	送信テーブル用インデックス
export_item(:)	I	export_index(n_neighbor_pe)	送信テーブル
HPCMW_NAME_LEN	I	-	NAME lengthパラメータ（＝63）
HPCMW_HEADER_LEN I	I	-	ヘッダー長さパラメータ（＝127）
HPCMW_MSG_LEN	I	-	メッセージ長さパラメータ（＝255）
HPCMW_FILNAME_LEN	I	-	ファイル名長さパラメータ（＝1023）
hpcmw_eps_fvm_files(:)	**C**	**4**	**分散ファイル名，(1)メッシュファイル，(2)結果ファイル，(4)通信ファイル**

hpcmw_eps_fvm_util（3/3）
MPI用パラメータ

変数名	型	パラメータ値	内 容
hpcmw_sum	I	46801	MPI_SUM
hpcmw_prod	I	46802	MPI_PROD
hpcmw_max	I	46803	MPI_MAX
hpcmw_min	I	46804	MPI_MIN
hpcmw_integer	I	53951	MPI_INTEGER
hpcmw_single_precision	I	53952	MPI_SINGLE_PRECISION
hpcmw_double_presision	I	53953	MPI_DOUBLE_PRECISION
hpcmw_character	I	53954	MPI_CHARACTER

マトリクス関連（hpcmw_eps_fvm_pcg）

変数名	**型**	**サイズ**	**内容**
NPLU	I	-	連立一次方程式係数マトリクス非対角成分総数
D(:)	R	NODE_tot	連立一次方程式係数マトリクス対角成分
PHI(:)	R	NODE_tot	連立一次方程式未知数ベクトル
BFORCE(:)	R	NODE_tot	連立一次方程式右辺ベクトル
index(:)	I	0:NODE_tot	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分数）
item(:)	I	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分要素番号）
AMAT(:)	R	NPLU	係数マトリクス非対角成分要素番号用一次元圧縮配列（非対角成分）

```
do i= 1, N
  q(i)= D(i)*p(i)
  do k= index(i-1)+1, index(i)
    q(i)= q(i) + AMAT(k)*p(item(k))
  enddo
enddo
```

変数ブロック：appl_cntl

```
!C
!C***
!C*** appl_cntl
!C***
!C
      module appl_cntl
      use hpcmw_eps_fvm_util

!C
!C-- FILE NAME
      character(len=HPCMW_HEADER_LEN):: HEADERgrid
      character(len=HPCMW_HEADER_LEN):: HEADERresult
      character(len=HPCMW_HEADER_LEN):: HEADERcomm
      character(len=HPCMW_HEADER_LEN):: AVSfile

!C
!C-- MESH info.
      integer(kind=kint) :: NX, NY, NZ, NXP1, NYP1, NZP1
      integer(kind=kint) :: PVISFLAG

(略)
      end module appl_cntl
```



```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


初期化（hpcmw_eps_fvm_util.f）

```
!C
!C***
!C*** HPCMW_EPS_FVM_INIT
!C***
!C
!C    INIT. HPCMW-FEM process's
!C
      subroutine HPCMW_EPS_FVM_INIT
      integer :: ierr

      call MPI_INIT      (ierr)
      call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT   , ierr)
      call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank , ierr)

      if (my_rank.eq.0) then
        zero= 1
       else
        zero= 0
      endif

      end subroutine hpcmw_eps_fvm_init
```


Finalize（hpcmw_eps_fvm_util.f）

```
!C
!C***
!C*** HPCMW_EPS_FVM_FINALIZE
!C***
!C
      subroutine HPCMW_EPS_FVM_FINALIZE
      integer :: ierr

      call MPI_FINALIZE (ierr)
      if (my_rank.eq.0) stop ' * normal termination'

      end subroutine hpcmw_eps_fvm_finalize
```

Abort（hpcmw_eps_fvm_util.f）

```
!C
!C***
!C*** HPCMW_EPS_FVM_ABORT
!C***
!C
      subroutine HPCMW_EPS_FVM_ABORT
      integer :: ierr

      call MPI_BARRIER (MPI_COMM_WORLD, ierr)
      call MPI_ABORT   (MPI_COMM_WORLD, ierr)

      end subroutine hpcmw_eps_fvm_abort
```



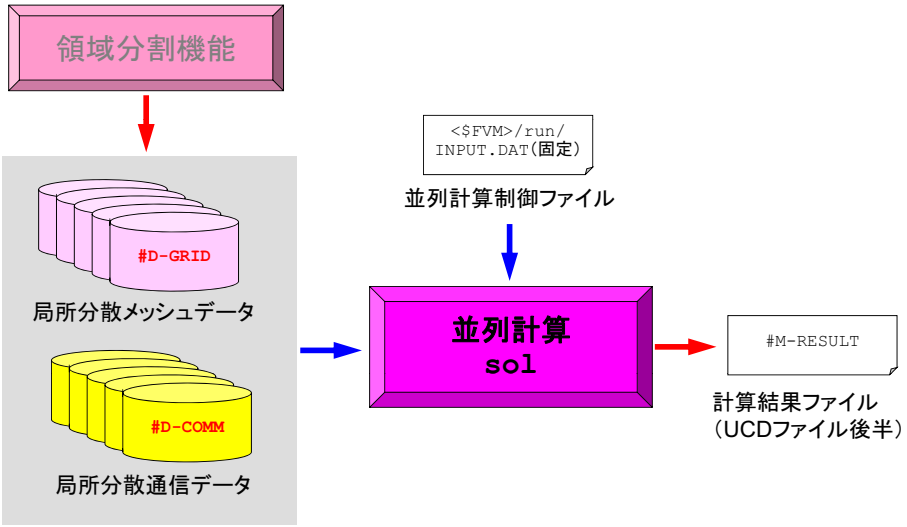
```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


データ入力部分（1/5）

```
      subroutine hpcmw_eps_fvm_input_grid
      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      character(len=HPCMW_NAME_LEN) :: member
      character(len=80            ) :: LINE

!C
!C +-------+
!C | FILES |                                          制御ファイル「INPUT.DAT」
!C +-------+
!C===
      open (11, file='INPUT.DAT', status='unknown')
        read (11,'(a127)') HEADERgrid
        read (11,'(a127)') HEADERcomm
        read (11,'(a127)') AVSfile
        read (11,*)        PVISFLAG
      close (11)

      allocate (hpcmw_eps_fvm_files(4))

      member= 'gridfile'                                                    #D-GRID
      call hpcmw_eps_fvm_define_file_name (member, HEADERgrid)

      member= 'commfile'
      call hpcmw_eps_fvm_define_file_name (member, HEADERcomm)              #D-COMM
!C===
```

ヘッダーを与えて分散ファイル名を生成する。

並列シミュレーションにおけるI/O



局所分散ファイル名生成(1/2)
(hpcmw_eps_fvm_util.f)

```
!C
!C***
!C*** HPCMW_EPS_FVM_DEFINE_FILE_NAME
!C***
!C
      subroutine HPCMW_EPS_FVM_DEFINE_FILE_NAME (member, HEADERo)

      character (len=HPCMW_HEADER_LEN) ::  HEADERo, FILENAME
      character (len=HPCMW_NAME_LEN)     ::  member
      character (len=HPCMW_NAME_LEN)     ::  HEADER
      character (len= 1) ::  SUBindex1
      character (len= 2) ::  SUBindex2
      character (len= 3) ::  SUBindex3
      character (len= 4) ::  SUBindex4
      character (len= 5) ::  SUBindex5
      character (len= 6) ::  SUBindex6
      integer:: LENGTH, ID

      HEADER= adjustL (HEADERo)
      LENGTH= len_trim(HEADER)
```


局所分散ファイル名生成(2/2)
(hpcmw_eps_fvm_util.f)

```
      if (my_rank.le.9) then
        ID= 1
        write(SUBindex1 ,'(i1.1)') my_rank
       else if (my_rank.le.99) then
        ID= 2
        write(SUBindex2 ,'(i2.2)') my_rank
       else if (my_rank.le.999) then
        ID= 3
        write(SUBindex3 ,'(i3.3)') my_rank
       else if (my_rank.le.9999) then
        ID= 4
        write(SUBindex5 ,'(i4.4)') my_rank
       else if (my_rank.le.99999) then
        ID= 5
        write(SUBindex6 ,'(i5.5)') my_rank
       else if (my_rank.le.999999) then
        ID= 6
        write(SUBindex4 ,'(i6.6)') my_rank
      endif

      if (ID.eq.1) filename= HEADER(1:LENGTH)//'.'//SUBindex1
      if (ID.eq.2) filename= HEADER(1:LENGTH)//'.'//SUBindex2
      if (ID.eq.3) filename= HEADER(1:LENGTH)//'.'//SUBindex3
      if (ID.eq.4) filename= HEADER(1:LENGTH)//'.'//SUBindex4
      if (ID.eq.5) filename= HEADER(1:LENGTH)//'.'//SUBindex5
      if (ID.eq.6) filename= HEADER(1:LENGTH)//'.'//SUBindex6

      if (member.eq. 'gridfile') hpcmw_eps_fvm_files(1)= filename
      if (member.eq. 'commfile') hpcmw_eps_fvm_files(4)= filename

      end subroutine hpcmw_eps_fvm_define_file_name
```

領域数1,000,000まで
対応可能

データ入力部分(2/5)

#D-GRID, ここらへんはserialと同じ
各PEでファイル名を書き出して見よ

```
!C
!C +------------+
!C | MESH INPUT |
!C +------------+
!C===
      IUNIT= 11
      open (IUNIT,file= hpcmw_eps_fvm_files(1), status='unknown')

!C
!C-- NODE
      read (IUNIT, '(10i10)') NODE_tot

      allocate (NODE_VOL(NODE_tot), NODE_COND(NODE_tot),                &
     &          NODE_XYZ(3*NODE_tot))

      do i= 1, NODE_tot
        read (IUNIT,'(i10,5e16.6)') ii, NODE_VOL(i), NODE_COND(i),      &
     &                              (NODE_XYZ(3*i-3+k), k=1, 3)
      enddo

!C
!C-- CONNECTION
      read (IUNIT,'(10i10)') CONN_tot
      allocate (CONN_NODE(2*CONN_tot), CONN_COEF(CONN_tot))
      do i= 1, CONN_tot
        read (IUNIT,'( 2i10, 3e16.6)') (CONN_NODE(2*i-2+k), k= 1, 2),   &
     &                                 AREA, D1, D2
        in1= CONN_NODE(2*i-1)
        in2= CONN_NODE(2*i  )
        C1 = NODE_COND(in1)
        C2 = NODE_COND(in2)
        CONN_COEF(i)= AREA / ( D1/C1 + D2/C2 )
      enddo
```

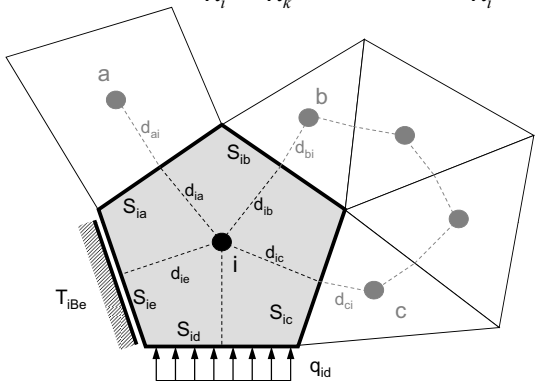

有限体積法:隣接メッシュの情報が必要

熱流束に関するつりあい式

隣接要素との熱伝導　　温度固定境界

$$\sum_k \frac{S_{ik}}{\frac{d_{ik}}{\lambda_i}+\frac{d_{ki}}{\lambda_k}}\left(T_k - T_i\right) + \sum_e \frac{S_{ie}}{\frac{d_{ie}}{\lambda_i}}\left(T_{iBe} - T_i\right) + \sum_d S_{id}\dot{q}_{id} + V_i\dot{Q}_i = 0$$

要素境界面通過熱流束　　体積発熱



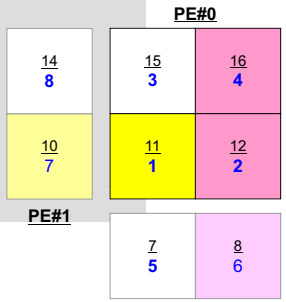
- λ　:熱伝導率
- V_i :要素体積
- S　:表面面積
- d_{ij} :要素中心から表面までの距離
- q　:表面フラックス
- Q　:体積発熱
- T_{iB} :境界温度

データ入力部分(3/5)

```
!C
!C-- DIRICHLET
      read (IUNIT,'(10i10)') FIX_NODE_tot
      allocate (FIX_NODE_ID(FIX_NODE_tot), FIX_NODE_COEF(FIX_NODE_tot))
      allocate (FIX_NODE_VAL(FIX_NODE_tot))

      do i= 1, FIX_NODE_tot
        read (IUNIT, '(i10, 3e16.6)')                                    &
     &        FIX_NODE_ID(i), AREA, DIST, FIX_NODE_VAL(i)
        icel= FIX_NODE_ID(i)
        COND= NODE_COND(icel)
        FIX_NODE_COEF(i)= AREA / (DIST/COND)
      enddo
!C
!C-- NEUMANN
      read (IUNIT,'(10i10)') SURF_NODE_tot
      allocate                                                           &
     &  (SURF_NODE_ID  (SURF_NODE_tot), SURF_NODE_FLUX(SURF_NODE_tot))

      do i= 1, SURF_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') SURF_NODE_ID(i), AREA, FLUX
        SURF_NODE_FLUX(i)= AREA*FLUX
      enddo
!C
!C-- BODY FLUX
      read (IUNIT,'(10i10)') BODY_NODE_tot

      allocate (BODY_NODE_FLUX(NODE_tot))
      do i= 1, BODY_NODE_tot
        read (IUNIT, '(i10, 3e16.6)') icel, FLUX
        BODY_NODE_FLUX(icel)= FLUX * NODE_VOL(icel)
      enddo
      close (IUNIT)
```


局所分散メッシュファイル(mesh.0)

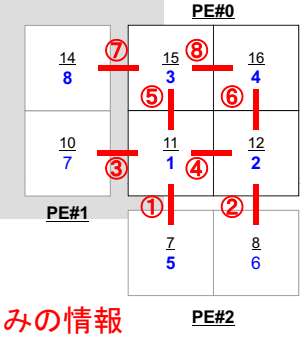
```
要素
8
1   1.000000E+00   1.000000E+00   2.500000E+00   2.500000E+00   5.000000E-01
2   1.000000E+00   1.000000E+00   3.500000E+00   2.500000E+00   5.000000E-01
3   1.000000E+00   1.000000E+00   2.500000E+00   3.500000E+00   5.000000E-01
4   1.000000E+00   1.000000E+00   3.500000E+00   3.500000E+00   5.000000E-01
5   1.000000E+00   1.000000E+00   2.500000E+00   1.500000E+00   5.000000E-01
6   1.000000E+00   1.000000E+00   3.500000E+00   1.500000E+00   5.000000E-01
7   1.000000E+00   1.000000E+00   1.500000E+00   2.500000E+00   5.000000E-01
8   1.000000E+00   1.000000E+00   1.500000E+00   3.500000E+00   5.000000E-01
コネクティビティ
8
5       1   1.000000E+00   5.000000E-01   5.000000E-01
6       2   1.000000E+00   5.000000E-01   5.000000E-01
7       1   1.000000E+00   5.000000E-01   5.000000E-01
1       2   1.000000E+00   5.000000E-01   5.000000E-01
1       3   1.000000E+00   5.000000E-01   5.000000E-01
2       4   1.000000E+00   5.000000E-01   5.000000E-01
8       3   1.000000E+00   5.000000E-01   5.000000E-01
3       4   1.000000E+00   5.000000E-01   5.000000E-01
境界条件
2
2   1.000000E+00   5.000000E-01   0.000000E+00
4   1.000000E+00   5.000000E-01   0.000000E+00
0
1
1   1.000000E+00
```



基本的に初期全体メッシュファイルと同じ
局所要素番号による記述
境界条件(ディリクレ，ノイマン，体積発熱):「内点」のみの情報

局所分散メッシュファイル(mesh.0)

```
要素
8
1   1.000000E+00   1.000000E+00   2.500000E+00   2.500000E+00   5.000000E-01
2   1.000000E+00   1.000000E+00   3.500000E+00   2.500000E+00   5.000000E-01
3   1.000000E+00   1.000000E+00   2.500000E+00   3.500000E+00   5.000000E-01
4   1.000000E+00   1.000000E+00   3.500000E+00   3.500000E+00   5.000000E-01
5   1.000000E+00   1.000000E+00   2.500000E+00   1.500000E+00   5.000000E-01
6   1.000000E+00   1.000000E+00   3.500000E+00   1.500000E+00   5.000000E-01
7   1.000000E+00   1.000000E+00   1.500000E+00   2.500000E+00   5.000000E-01
8   1.000000E+00   1.000000E+00   1.500000E+00   3.500000E+00   5.000000E-01
コネクティビティ
8
5       1   1.000000E+00   5.000000E-01   5.000000E-01  ①
6       2   1.000000E+00   5.000000E-01   5.000000E-01  ②
7       1   1.000000E+00   5.000000E-01   5.000000E-01  ③
1       2   1.000000E+00   5.000000E-01   5.000000E-01  ④
1       3   1.000000E+00   5.000000E-01   5.000000E-01  ⑤
2       4   1.000000E+00   5.000000E-01   5.000000E-01  ⑥
8       3   1.000000E+00   5.000000E-01   5.000000E-01  ⑦
3       4   1.000000E+00   5.000000E-01   5.000000E-01  ⑧
境界条件
2
2   1.000000E+00   5.000000E-01   0.000000E+00
4   1.000000E+00   5.000000E-01   0.000000E+00
0
1
1   1.000000E+00
```



基本的に初期全体メッシュファイルと同じ
局所要素番号による記述
コネクティビティ:「内点～内点」,「内点～外点」のみの情報

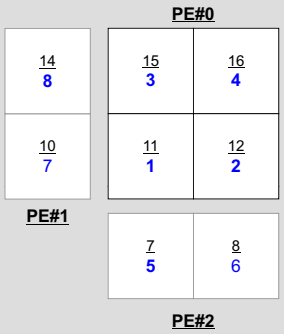
データ入力部分(4/5)

分散通信データ
#D-COMM

```
!C
!C +------------+
!C | COMM INPUT |
!C +------------+
!C===
      IUNIT= 12
      open (IUNIT,file= hpcmw_eps_fvm_files(4), status='unknown')

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') n_neighbor_pe

      allocate (neighbor_pe(n_neighbor_pe))
      allocate (import_index(0:n_neighbor_pe))
      allocate (export_index(0:n_neighbor_pe))

      import_index= 0
      export_index= 0

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (neighbor_pe(k), k= 1, n_neighbor_pe)


      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (import_index(k), k= 1, n_neighbor_pe)
      nn= import_index(n_neighbor_pe)
      allocate (import_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (import_item(k), k= 1, nn)
```

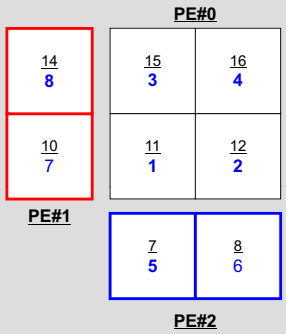
局所分散通信ファイル（comm.0）
隣接領域

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



局所分散通信ファイル（comm.0）
受信テーブル，外点情報

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

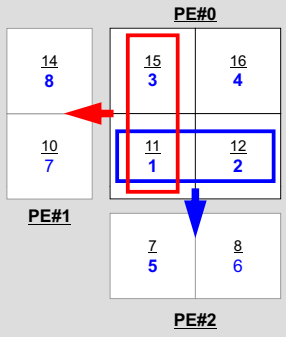


データ入力部分（5/5）

```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```


局所分散通信ファイル（comm.0）
送信テーブル，境界点情報

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```



データ入力部分(5/5)

```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```

内点数

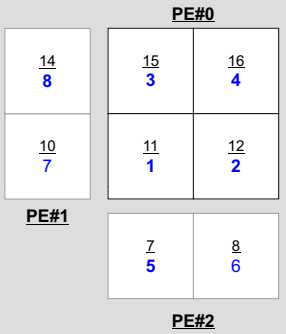
内点＋外点数(NODE_tot)

局所分散通信ファイル(comm.0)

内点数，総要素数(内点＋外点)，全体要素番号

```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID
  11  12  15  16   7   8
  10  14
```

全体要素番号(局所番号順)



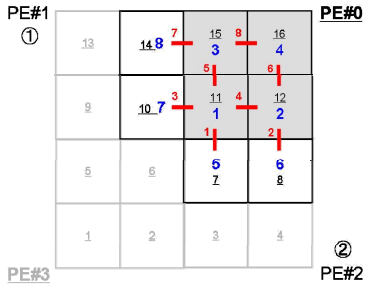

```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


poi_gen(1/2)

- 係数行列生成部は1PEの場合とほとんど同じ
- 理由
  - 係数行列生成に必要な情報が全て局所分散メッシュファイルに含まれている。

局所分散メッシュファイル

```
8
1    1.00    1.00    2.50E+00    2.50E+00    5.00E-01
2    1.00    1.00    3.50E+00    2.50E+00    5.00E-01
3    1.00    1.00    2.50E+00    3.50E+00    5.00E-01
4    1.00    1.00    3.50E+00    3.50E+00    5.00E-01
5    1.00    1.00    2.50E+00    1.50E+00    5.00E-01
6    1.00    1.00    3.50E+00    1.50E+00    5.00E-01
7    1.00    1.00    1.50E+00    2.50E+00    5.00E-01
8    1.00    1.00    1.50E+00    3.50E+00    5.00E-01
8
5        1    1.000000E+00    5.000000E-01    5.000000E-01
6        2    1.000000E+00    5.000000E-01    5.000000E-01
7        1    1.000000E+00    5.000000E-01    5.000000E-01
1        2    1.000000E+00    5.000000E-01    5.000000E-01
1        3    1.000000E+00    5.000000E-01    5.000000E-01
2        4    1.000000E+00    5.000000E-01    5.000000E-01
8        3    1.000000E+00    5.000000E-01    5.000000E-01
3        4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```

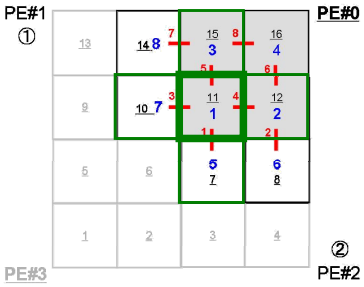


poi_gen（2/2）

- 例えば1番の要素（全体番号11番）における係数行列生成に必要な隣接要素の情報（2,3,5,7番（全体番号：12,15,7,10））は全て局所分散メッシュデータに全て含まれている。
  - うち，5番，7番は外点

局所分散メッシュファイル

```
8
1    1.00    1.00    2.50E+00    2.50E+00    5.00E-01
2    1.00    1.00    3.50E+00    2.50E+00    5.00E-01
3    1.00    1.00    2.50E+00    3.50E+00    5.00E-01
4    1.00    1.00    3.50E+00    3.50E+00    5.00E-01
5    1.00    1.00    2.50E+00    1.50E+00    5.00E-01
6    1.00    1.00    3.50E+00    1.50E+00    5.00E-01
7    1.00    1.00    1.50E+00    2.50E+00    5.00E-01
8    1.00    1.00    1.50E+00    3.50E+00    5.00E-01
8
5         1    1.000000E+00    5.000000E-01    5.000000E-01
6         2    1.000000E+00    5.000000E-01    5.000000E-01
7         1    1.000000E+00    5.000000E-01    5.000000E-01
1         2    1.000000E+00    5.000000E-01    5.000000E-01
1         3    1.000000E+00    5.000000E-01    5.000000E-01
2         4    1.000000E+00    5.000000E-01    5.000000E-01
8         3    1.000000E+00    5.000000E-01    5.000000E-01
3         4    1.000000E+00    5.000000E-01    5.000000E-01
2
2    1.000000E+00    5.000000E-01    0.000000E+00
4    1.000000E+00    5.000000E-01    0.000000E+00
0
1
1    1.000000E+00
```




```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


前処理付き共役勾配法の並列化

Preconditioned Conjugate Gradient Method（CG）

```
Compute r(0)= b-[A]x(0)
for i= 1, 2, …
    solve [M]z(i-1)= r(i-1)
    ρi-1= r(i-1) z(i-1)
    if i=1
      p(1)= z(0)
    else
      βi-1= ρi-1/ρi-2
      p(i)= z(i-1) + βi-1 p(i-1)
    endif
    q(i)= [A]p(i)
    αi  = ρi-1/p(i)q(i)
    x(i)= x(i-1) + αip(i)
    r(i)= r(i-1) - αiq(i)
    check convergence |r|
end
```

並列計算，領域間通信が必要な部分

- 行列ベクトル積
- 内積

hpcmw_eps_fvm_comm.*

- MPIに関連した共通サブルーチン群
  - 通信
- MPIサブルーチンの代替
- 「eps_fvm」における1対1通信
  - 一般化された通信テーブル

- 利点
  - MPIサブルーチンを直接呼ぶよりも引数を減らすことができる。
  - 1対1通信においては「送信バッファへの代入，送信，受信，受信バッファからの読み出し」という操作をそのたびに記述する必要が無くなる。

hpcmw_eps_fvm_commの内容

http://nkl.cc.u-tokyo.ac.jp/tutorial/parallel_lib_tutorial/

MPI_BARRIER
```
 subroutine hpcmw_eps_fvm_barrier
```
MPI_ALLREDUCE(スカラー)
```
 subroutine hpcmw_eps_fvm_allreduce_R ( VAL, ntag)
 subroutine hpcmw_eps_fvm_allreduce_I ( VAL, ntag)
```
MPI_BCAST(スカラー)
```
 subroutine hpcmw_eps_fvm_bcast_R ( VAL, nbase)
 subroutine hpcmw_eps_fvm_bcast_I ( VAL, nbase)
 subroutine hpcmw_eps_fvm_bcast_C ( VAL, n, nbase)
```
MPI_ALLREDUCE(ベクトル)
```
 subroutine hpcmw_eps_fvm_allreduce_RV ( VAL, n, ntag)
 subroutine hpcmw_eps_fvm_allreduce_IV ( VAL, n, ntag)
```
MPI_BCAST(ベクトル)
```
 subroutine hpcmw_eps_fvm_bcast_RV ( VAL, n, nbase)
 subroutine hpcmw_eps_fvm_bcast_IV ( VAL, n, nbase)
 subroutine hpcmw_eps_fvm_bcast_CV ( VAL, n, nn, nbase)
```
一対一通信用(ベクトル)
```
 subroutine hpcmw_eps_fvm_update_1_R (X, n)
```


hpcmw_eps_fvm_allreduce_R

```
!C
!C***
!C*** hpcmw_eps_fvm_allREDUCE_R
!C***
!C
      subroutine hpcmw_eps_fvm_allreduce_R ( VAL, ntag)
      use hpcmw_eps_fvm_util
      implicit REAL*8 (A-H,O-Z)
      integer :: ntag, ierr
      real(kind=kreal) :: VAL, VALM

      if (ntag .eq. hpcmw_sum) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_SUM,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_max) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_MAX,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_min) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, 1, MPI_DOUBLE_PRECISION, MPI_MIN,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      VAL= VALM

      end subroutine hpcmw_eps_fvm_allreduce_R
```


hpcmw_eps_fvm_allreduce_RV

```
!C
!C***
!C*** hpcmw_eps_fvm_allREDUCE_RV
!C***
!C
      subroutine hpcmw_eps_fvm_allreduce_RV ( VAL, n, ntag)
      use hpcmw_eps_fvm_util
      implicit REAL*8 (A-H,O-Z)
      integer :: n, ntag, ierr
      real(kind=kreal), dimension(n) :: VAL
      real(kind=kreal), dimension(:), allocatable :: VALM

      allocate (VALM(n))
      if (ntag .eq. hpcmw_sum) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_SUM,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_max) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_MAX,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      if (ntag .eq. hpcmw_min) then
        call MPI_allREDUCE                                              &
     &       (VAL, VALM, n, MPI_DOUBLE_PRECISION, MPI_MIN,              &
     &        MPI_COMM_WORLD, ierr)
      endif

      VAL= VALM
      deallocate (VALM)

      end subroutine hpcmw_eps_fvm_allreduce_RV
```


hpcmw_eps_fvm_update_1_R(1/2)

```
      subroutine hpcmw_eps_fvm_update_1_R (X, n)
      use hpcmw_eps_fvm_util

      implicit REAL*8 (A-H,O-Z)
      integer :: n, nn, ierr
      real(kind=kreal), dimension(n) :: X
      real(kind=kreal), dimension(:), allocatable :: WS, WR

      integer(kind=kint ), dimension(:,:), allocatable :: sta1
      integer(kind=kint ), dimension(:,:), allocatable :: sta2
      integer(kind=kint ), dimension(:  ), allocatable :: req1
      integer(kind=kint ), dimension(:  ), allocatable :: req2

      nn= max (n, import_index(n_neighbor_pe),                          &
     &            export_index(n_neighbor_pe))

      allocate (WS(nn), WR(nn))
!C
!C-- INIT.
      allocate (sta1(MPI_STATUS_SIZE,n_neighbor_pe))
      allocate (sta2(MPI_STATUS_SIZE,n_neighbor_pe))
      allocate (req1(n_neighbor_pe))
      allocate (req2(n_neighbor_pe))
```

送信，受信バッファの定義

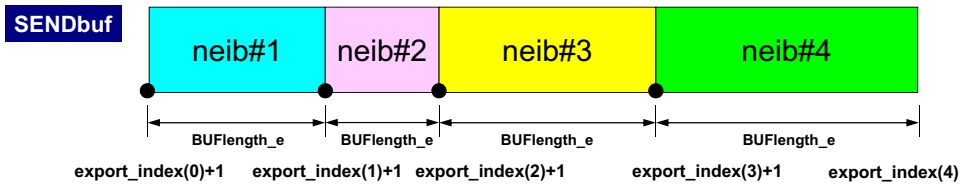
hpcmw_eps_fvm_update_1_R(2/2)

送信バッファへの代入

送信

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        istart= export_index(neib-1)
        inum  = export_index(neib  ) - istart

        do k= istart+1, istart+inum
           WS(k)= X(export_item(k))
        enddo
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        do k= istart+1, istart+inum
          X(import_item(k))= WR(k)
        enddo
      enddo

      call MPI_WAITALL (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```


送信(MPI_Isend/Irecv/Waitall)




```
 do neib= 1, NEIBPETOT
   do k= export_index(neib-1)+1, export_index(neib)
     kk= export_item(k)
     SENDbuf(k)= VAL(kk)
   enddo
 enddo

 do neib= 1, NEIBPETOT
   iS_e= export_index(neib-1) + 1
   iE_e= export_index(neib  )
   BUFlength_e= iE_e + 1 - iS_e

   call MPI_Isend                                                 &
&          (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&           MPI_COMM_WORLD, request_send(neib), ierr)
 enddo

 call MPI_Waitall (NEIBPETOT, request_send, stat_recv, ierr)
```

送信バッファへの代入
温度などの変数を直接送信，受信に使うのではなく，このようなバッファへ一回代入して計算することを勧める。

hpcmw_eps_fvm_update_1_R(2/2)

受信

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        istart= export_index(neib-1)
        inum  = export_index(neib  ) - istart

        do k= istart+1, istart+inum
           WS(k)= X(export_item(k))
        enddo
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        do k= istart+1, istart+inum
          X(import_item(k))= WR(k)
        enddo
      enddo

      call MPI_WAITALL (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```

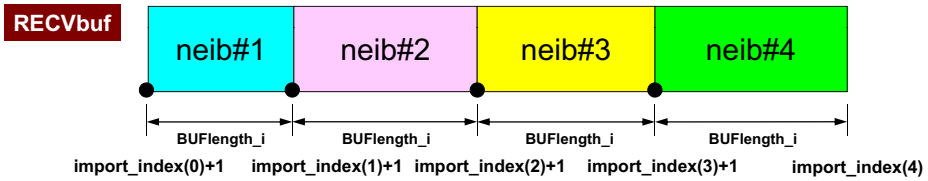

hpcmw_eps_fvm_update_1_R(2/2)

受信バッファからのデータ取り出し

```
!C
!C-- SEND
      do neib= 1, n_neighbor_pe
        istart= export_index(neib-1)
        inum  = export_index(neib  ) - istart

        do k= istart+1, istart+inum
           WS(k)= X(export_item(k))
        enddo
        call MPI_ISEND (WS(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req1(neib), ierr)
      enddo
!C
!C-- RECEIVE
      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        call MPI_IRECV (WR(istart+1), inum, MPI_DOUBLE_PRECISION,       &
     &                  neighbor_pe(neib), 0, MPI_COMM_WORLD,           &
     &                  req2(neib), ierr)
      enddo

      call MPI_WAITALL (n_neighbor_pe, req2, sta2, ierr)

      do neib= 1, n_neighbor_pe
        istart= import_index(neib-1)
        inum  = import_index(neib  ) - istart
        do k= istart+1, istart+inum
          X(import_item(k))= WR(k)
        enddo
      enddo

      call MPI_WAITALL (n_neighbor_pe, req1, sta1, ierr)

      deallocate (sta1, sta2, req1, req2, WS, WR)]
      end subroutine hpcmw_eps_fvm_update_1_R
```

受信（MPI_Isend/Irecv/Waitall）

```
 do neib= 1, NEIBPETOT
   iS_i= import_index(neib-1) + 1
   iE_i= import_index(neib  )
   BUFlength_i= iE_i + 1 - iS_i

   call MPI_Irecv                                              &
&         (RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, request_recv(neib), ierr)
  enddo

  call MPI_Waitall (NEIBPETOT, request_recv, stat_recv, ierr)

  do neib= 1, NEIBPETOT
     do k= import_index(neib-1)+1, import_index(neib)
       kk= import_item(k)
       VAL(kk)= RECVbuf(k)                                   受信バッファから代入
     enddo
  enddo
```



hpcmw_solver.f

```
      subroutine hpcmw_eps_fvm_solver

      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      EPS = 1.d-8
      ITR = NODE_tot

      call hpcmw_eps_fvm_solver_CG                                  &
     &     ( intNODE_tot, NODE_tot, NPLU, D, BFORCE, DELPHI, EPS,   &
     &       ITR,  IER, index, item, AMAT, COMMtime)

      open (11, file='fvmmg.ctrl', status='unknown')
        read (11,*) NX, NY, NZ
      close (11)

      iS= NX*NY*NZ/2 + NX*NY/2
      do i= iS+1, iS+NX
        write (*,'(i8,3(1pe16.6))') i, DELPHI(i)
      enddo

      end subroutine hpcmw_eps_fvm_solver
```


共役勾配法

```
!C
!C***
!C*** CG
!C***
!C
      subroutine hpcmw_eps_fvm_solver_CG                            &
     &         ( N, NP, NPLU, D, B, X, EPS, ITR, IER,               &
     &           index, item, COEF, Tcomm)

      use hpcmw_eps_fvm_util
      implicit REAL*8 (A-H,O-Z)

      real(kind=kreal), dimension(N ) :: D
      real(kind=kreal), dimension(NP) :: B
      real(kind=kreal), dimension(NP) :: X

      integer            , dimension(0:N) :: index
      integer            , dimension(NPLU):: item
      real   (kind=kreal), dimension(NPLU):: COEF

      real(kind=kreal) :: EPS, Tcomm

      integer :: ITR, IER
      integer :: P, Q, R, Z, DD

      real(kind=kreal), dimension(:,:), allocatable, save :: W

        N :  intNODE_tot     COEF: AMAT
        NP:  NODE_tot        B   : RHS
```


共役勾配法の「並列化」(1/4)

- 行列ベクトル積

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===

      exchange W(i,P)

      do i= 1, N
        W(i,Q) = D(i) * W(i,P)
        do j= index(i-1)+1, index(i)
          W(i,Q) = W(i,Q) + COEF(j) * W(item(j),P)
        enddo
      enddo
!C===
```

共役勾配法の「並列化」(1/4)

- 行列ベクトル積

```
!C
!C +-------------+
!C | {q}= [A]{p} |
!C +-------------+
!C===
      call hpcmw_eps_fvm_update_1_R (W(1,P), NP)

      do i= 1, N
        W(i,Q) = D(i) * W(i,P)
        do j= index(i-1)+1, index(i)
          W(i,Q) = W(i,Q) + COEF(j) * W(item(j),P)
        enddo
      enddo
!C===
```


共役勾配法の「並列化」(2/4)

- 内積：MPI_ALLREDUCE

```
!C
!C +-------------+
!C | RHO= {r}{z} |
!C +-------------+
!C===
      RHO= 0.d0

      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO

!C===
```

```
!C
!C +-------------+
!C | RHO= {r}{z} |
!C +-------------+
!C===
      RHO= 0.d0

      do i= 1, NP
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      allreduce RHO

!C===
```

こうしてはいけない・・・何故か?
(N：内点数，NP：内点＋外点)

```
N : intNODE_tot
NP: NODE_tot
```


共役勾配法の「並列化」(2/4)

- 内積：MPI_ALLREDUCE

```
!C
!C +-------------+
!C | RHO= {r}{z} |
!C +-------------+
!C===
      RHO= 0.d0

      do i= 1, N
        RHO= RHO + W(i,R)*W(i,Z)
      enddo

      call hpcmw_eps_fvm_allreduce_R (RHO, hpcmw_sum)
!C===
```


共役勾配法の「並列化」(3/4)

- N(**intNODE_tot**)とNP(**NODE_tot**)の違いに注意。
  - 基本的に計算はN個分やればよい(ループは`do i=1,N`)。
  - 外点の値を変えるような(左辺にくる)計算はしない。
    - 外点の値が必要なときは行列ベクトル積のみ。
    - そのときに通信して外点の値をもらってくれば良い。

共役勾配法の「並列化」(4/4)

- 最後に，従属変数(X)の外点における最新値をもらっておくことを忘れないように。

```
      DNRM20= 0.d0
      do i= 1, N
        X(i)   = X(i)    + ALPHA * W(i,P)
        W(i,R)= W(i,R) - ALPHA * W(i,Q)
      enddo
      DNRM2 = 0.0
      do i= 1, N
        DNRM2= DNRM2 + W(i,R)**2
      enddo

      call hpcmw_eps_fvm_allreduce_R (DNRM2, hpcmw_sum)
      RESID= dsqrt(DNRM2/BNRM2)

        if ( RESID.le.EPS) goto 900
        RHO1 = RHO
    enddo
900 continue

      call hpcmw_eps_fvm_update_1_R (X, NP)

      return
      end subroutine hpcmw_eps_fvm_solver_CG
```



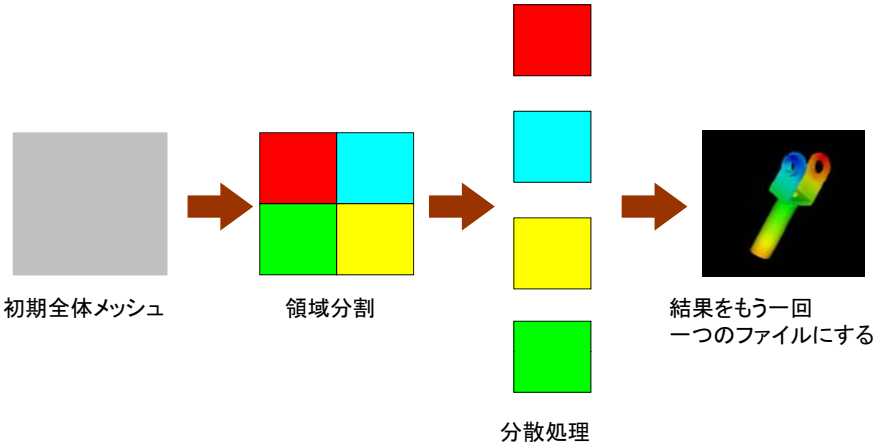
```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call output_ucd

call hpcmw_eps_fvm_finalize

end program eps_fvm
```


可視化

- いわゆる，「並列可視化」をやってもらうつもりだったのであるが，プログラムが完全構造格子用に作られたものであるため，「eps_fvm」のような非構造格子的なものでは対応が難しい。

- 幸い，MicroAVSが各自で使えるので，少し違ったアプローチを試みることにした。

やるべきこと



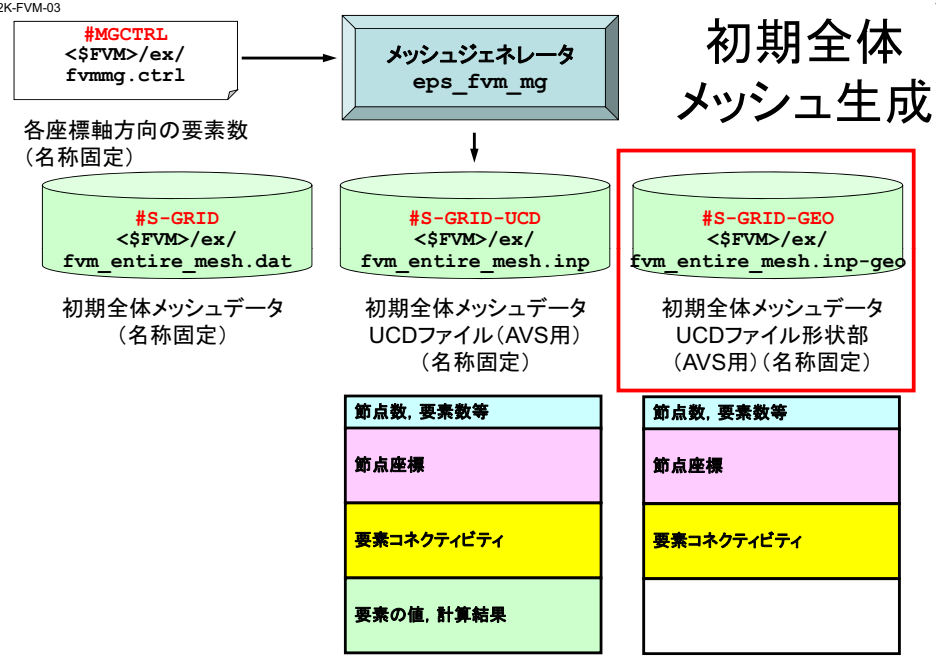
可視化

- いわゆる，「並列可視化」をやってもらうつもりだったのであるが，プログラムが完全構造格子用に作られたものであるため，「eps_fvm」のような非構造格子的なものでは対応が難しい。
- 幸い，MicroAVSが各自で使えるので，少し違ったアプローチを試みることにした。
- 本当の「並列可視化」とは言えないが，「**MPI_Gatherv**」等の関数を利用する機会にもなるので，このようなやりかたを試みる。

AVS UCDファイル

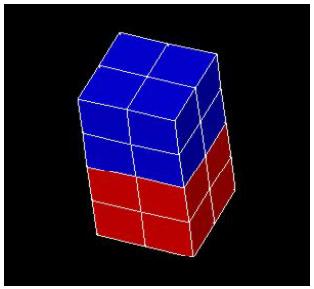
- 最終出力はUCD（Unstructured Cell Data）フォーマットとして，これをMicroAVSで読み込む。
- ファイル名は「`*.inp`」でなければならない。
- UCDファイルは以下の2部分から構成されているのであるが，今回は「形状」については，初期全体メッシュジェネレータで自動的に生成されている（`fvm_entire_mesh.inp-geo`）。
  - 形状（節点，要素）
  - 結果
- 結果のみについてアペンドすればよい。

初期全体メッシュ生成



AVS UCDファイルの例（1/2）
45節点，16要素の六面体

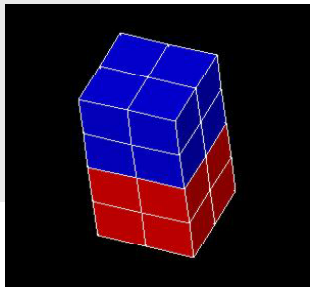
```
45      16       0       1       0
 1  0.000000E+00  0.000000E+00  0.000000E+00
 2  1.000000E+00  0.000000E+00  0.000000E+00
 3  2.000000E+00  0.000000E+00  0.000000E+00
 4  0.000000E+00  1.000000E+00  0.000000E+00
 5  1.000000E+00  1.000000E+00  0.000000E+00
 6  2.000000E+00  1.000000E+00  0.000000E+00
 7  0.000000E+00  2.000000E+00  0.000000E+00
 8  1.000000E+00  2.000000E+00  0.000000E+00
 9  2.000000E+00  2.000000E+00  0.000000E+00
10  0.000000E+00  0.000000E+00  1.000000E+00
11  1.000000E+00  0.000000E+00  1.000000E+00
12  2.000000E+00  0.000000E+00  1.000000E+00
13  0.000000E+00  1.000000E+00  1.000000E+00
14  1.000000E+00  1.000000E+00  1.000000E+00
15  2.000000E+00  1.000000E+00  1.000000E+00
16  0.000000E+00  2.000000E+00  1.000000E+00
17  1.000000E+00  2.000000E+00  1.000000E+00
18  2.000000E+00  2.000000E+00  1.000000E+00
19  0.000000E+00  0.000000E+00  2.000000E+00
20  1.000000E+00  0.000000E+00  2.000000E+00
21  2.000000E+00  0.000000E+00  2.000000E+00
22  0.000000E+00  1.000000E+00  2.000000E+00
23  1.000000E+00  1.000000E+00  2.000000E+00
24  2.000000E+00  1.000000E+00  2.000000E+00
25  0.000000E+00  2.000000E+00  2.000000E+00
26  1.000000E+00  2.000000E+00  2.000000E+00
27  2.000000E+00  2.000000E+00  2.000000E+00
28  0.000000E+00  0.000000E+00  3.000000E+00
29  1.000000E+00  0.000000E+00  3.000000E+00
30  2.000000E+00  0.000000E+00  3.000000E+00
31  0.000000E+00  1.000000E+00  3.000000E+00
32  1.000000E+00  1.000000E+00  3.000000E+00
33  2.000000E+00  1.000000E+00  3.000000E+00
34  0.000000E+00  2.000000E+00  3.000000E+00
35  1.000000E+00  2.000000E+00  3.000000E+00
36  2.000000E+00  2.000000E+00  3.000000E+00
37  0.000000E+00  0.000000E+00  4.000000E+00
38  1.000000E+00  0.000000E+00  4.000000E+00
39  2.000000E+00  0.000000E+00  4.000000E+00
40  0.000000E+00  1.000000E+00  4.000000E+00
41  1.000000E+00  1.000000E+00  4.000000E+00
42  2.000000E+00  1.000000E+00  4.000000E+00
43  0.000000E+00  2.000000E+00  4.000000E+00
44  1.000000E+00  2.000000E+00  4.000000E+00
45  2.000000E+00  2.000000E+00  4.000000E+00
```



AVS UCDファイルの例(2/2)
45節点，16要素の六面体

```
    1  1 hex          1          2          5          4         10         11         14         13
    2  1 hex          2          3          6          5         11         12         15         14
    3  1 hex          4          5          8          7         13         14         17         16
    4  1 hex          5          6          9          8         14         15         18         17
    5  1 hex         10         11         14         13         19         20         23         22
    6  1 hex         11         12         15         14         20         21         24         23
    7  1 hex         13         14         17         16         22         23         26         25
    8  1 hex         14         15         18         17         23         24         27         26
    9  1 hex         19         20         23         22         28         29         32         31
   10  1 hex         20         21         24         23         29         30         33         32
   11  1 hex         22         23         26         25         31         32         35         34
   12  1 hex         23         24         27         26         32         33         36         35
   13  1 hex         28         29         32         31         37         38         41         40
   14  1 hex         29         30         33         32         38         39         42         41
   15  1 hex         31         32         35         34         40         41         44         43
   16  1 hex         32         33         36         35         41         42         45         44
  1  1
COLOR, color
        1         2.00e+00
        2         2.00e+00
        3         2.00e+00
        4         2.00e+00
        5         2.00e+00
        6         2.00e+00
        7         2.00e+00
        8         2.00e+00
        9         1.00e+00
       10         1.00e+00
       11         1.00e+00
       12         1.00e+00
       13         1.00e+00
       14         1.00e+00
       15         1.00e+00
       16         1.00e+00
```

この部分のみ自分で作成すれば良い



手 順

- 各プロセッサにおける計算結果(**PHI(:)**に格納される)，グローバル要素番号を，あるプロセッサに集める：MPI_Gatherv使用
  - 「内点」の結果のみで良い。
- そのプロセッサから，結果を以下の書式で，グローバル番号順に書き出す(「,」は入れない)：

  <グローバル要素番号><計算結果>

- ファイルを「**fvm_entire_mesh.inp-geo**」にアペンドしてUCDファイルを生成する。
- ヒント
  - MPI_Gathervを使う
  - グローバルIDは局所分散通信ファイルより得る

MPI_Gatherv

MPI_Gatherv：
局所ベクトルから全体ベクトル生成

- **cd <$FVM>/gatherv**
- 「a2.0~a2.3」から局所ベクトル情報を読み込み，「全体ベクトル」情報を各プロセッサに生成するプログラムを作成する。MPI_Gathervを使用する。

PE#0

```
  8
101.0
103.0
105.0
106.0
109.0
111.0
121.0
151.0
```

PE#1

```
  5
201.0
203.0
205.0
206.0
209.0
```

PE#2

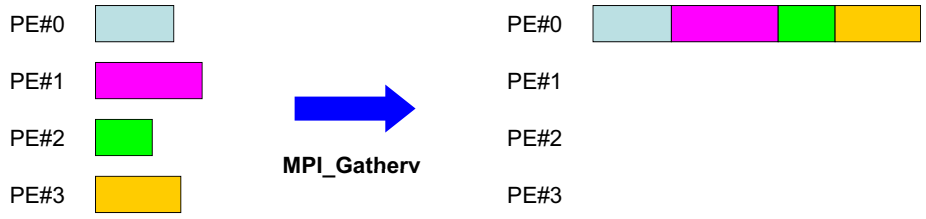
```
  7
301.0
303.0
305.0
306.0
311.0
321.0
351.0
```

PE#3

```
  3
401.0
403.0
405.0
```


MPI_Gatherv

# 局所ベクトルから全体ベクトル生成
MPI_Gathervを使う場合(1/5)

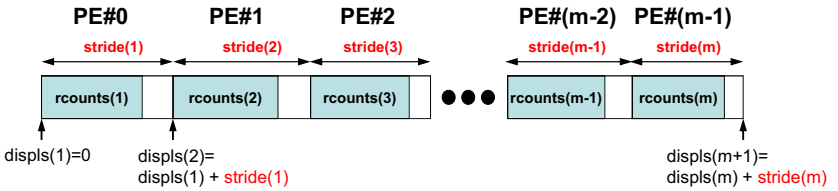


MPI_Gatherv

- MPI_Gather の可変長さベクトル版
  - 「局所データ」から「全体データ」を生成する
- **call MPI_Gatherv (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, root, comm, ierr)**
  - sendbuf 任意 I 送信バッファの先頭アドレス，
  - scount 整数 I 送信メッセージのサイズ
  - sendtype 整数 I 送信メッセージのデータタイプ
  - recvbuf 任意 O 受信バッファの先頭アドレス，
  - **rcounts 整数 I 受信メッセージのサイズ（配列：サイズ＝PETOT）**
  - **displs 整数 I 受信メッセージのインデックス（配列：サイズ＝PETOT+1）**
  - recvtype 整数 I 受信メッセージのデータタイプ
  - root 整数 I 受信メッセージ受信元（ランク）
  - comm 整数 I コミュニケータを指定する
  - ierr 整数 O 完了コード

MPI_Gatherv（続き）

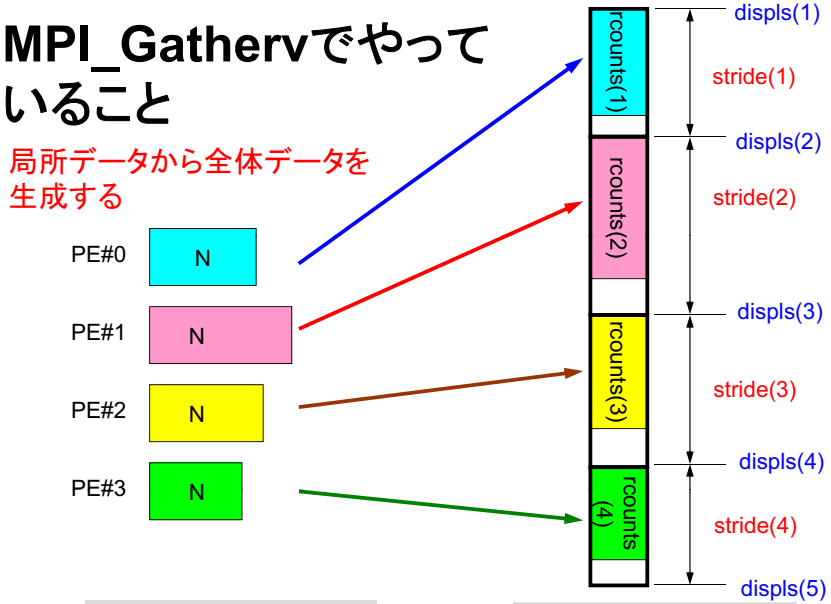
- **call MPI_Gatherv (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, root, comm, ierr)**
  - **rcounts 整数 I 受信メッセージのサイズ（配列：サイズ＝PETOT）**
  - **displs 整数 I 受信メッセージのインデックス（配列：サイズ＝PETOT+1）**
  - **この2つの配列は，最終的に生成される「全体データ」のサイズに関する配列であるため，各プロセスで配列の全ての値が必要になる：**
    - **もちろん各プロセスで共通の値を持つ必要がある。**
  - **通常はstride(i)=rcounts(i)**



size(recvbuf)= displs(PETOT+1)= sum(stride)

MPI_Gathervでやっていること

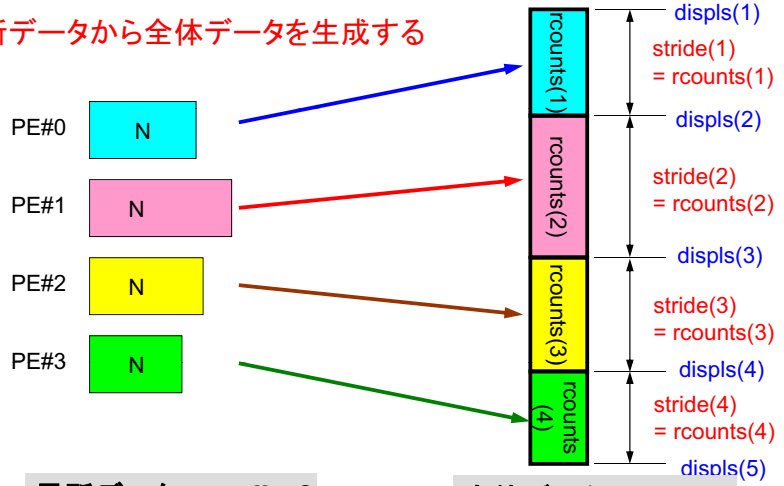
局所データから全体データを生成する



局所データ：sendbuf　全体データ：recvbuf

MPI_Gathervでやっていること

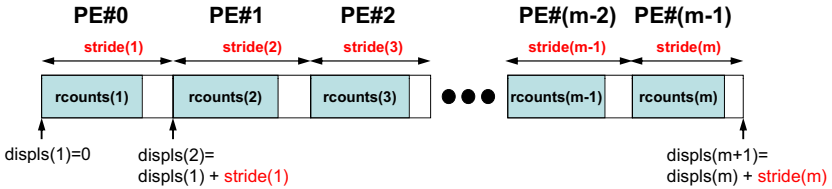
局所データから全体データを生成する



局所データ：sendbuf　全体データ：recvbuf

MPI_Gatherv詳細(1/2)

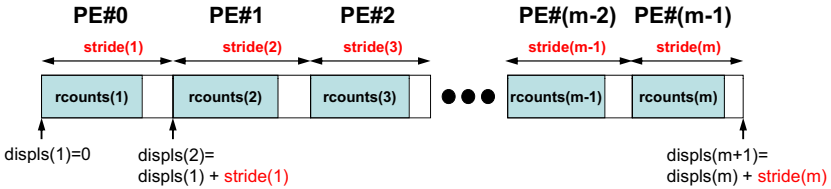
- call MPI_ALLGATHERV (sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, root, comm, ierr)
  - rcounts 整数 I 受信メッセージのサイズ(配列:サイズ=PETOT)
  - displs 整数 I 受信メッセージのインデックス(配列:サイズ=PETOT+1)
- rcounts
  - 各PEにおけるメッセージサイズ:局所データのサイズ
- displs
  - 各局所データの全体データにおけるインデックス
  - displs(PETOT+1)が全体データのサイズ



size(recvbuf)= displs(PETOT+1)= sum(stride)

MPI_Gatherv詳細(2/2)

- rcountsとdisplsは各プロセスで共通の値が必要
  - 各プロセスのベクトルの大きさ N をallgatherして,rcountsに相当するベクトルを作る。
  - rcountsから各プロセスにおいてdisplsを作る(同じものができる)。
    - stride(i)= rcounts(i) とする
  - rcountsの和にしたがってrecvbufの記憶領域を確保する。



size(recvbuf)= displs(PETOT+1)= sum(stride)

MPI_Gatherv使用準備

- “a2.0”~”a2.3”から,全体ベクトルを生成する。
- 各ファイルのベクトルのサイズが,8,5,7,3であるから,長さ23(=8+5+7+3)のベクトルができることになる。

PE#0
```
  8
101.0
103.0
105.0
106.0
109.0
111.0
121.0
151.0
```

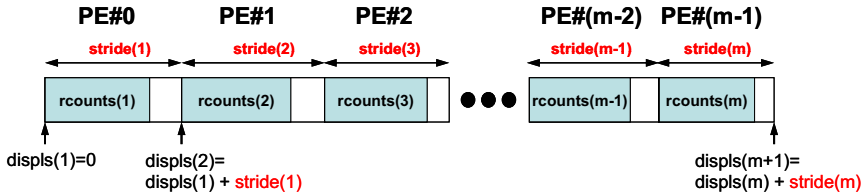
PE#1
```
  5
201.0
203.0
205.0
206.0
209.0
```

PE#2
```
  7
301.0
303.0
305.0
306.0
311.0
321.0
351.0
```

PE#3
```
  3
401.0
403.0
405.0
```

S1-2

局所⇒全体ベクトル生成:手順



size(recvbuf)= displs(PETOT+1)= sum(stride)

- 局所ベクトル情報を読み込む
- 「rcounts」,「displs」を作成する
- 「recvbuf」を準備する
- Gatherv

S1-2

MPI_Gatherv

局所⇒全体ベクトル生成(1/2)

<$FVM>/gatherv/mpigathertest.*

```
implicit REAL*8 (A-H,O-Z)
include 'mpif.h'
integer :: PETOT, my_rank, SOLVER_COMM, ierr
real(kind=8), dimension(:), allocatable :: VEC, VEC2, VECg
integer (kind=4), dimension(:), allocatable :: COUNT, COUNTindex
character(len=80)          :: filename

call MPI_INIT      (ierr)
call MPI_COMM_SIZE (MPI_COMM_WORLD, PETOT, ierr )
call MPI_COMM_RANK (MPI_COMM_WORLD, my_rank, ierr )
call MPI_COMM_DUP  (MPI_COMM_WORLD, SOLVER_COMM, ierr)

if (my_rank.eq.0) filename= 'a2.0'
if (my_rank.eq.1) filename= 'a2.1'
if (my_rank.eq.2) filename= 'a2.2'
if (my_rank.eq.3) filename= 'a2.3'

open (21, file= filename, status= 'unknown')
  read (21,*) N
  allocate (VEC(N))
  do i= 1, N
    read (21,*) VEC(i)
  enddo

allocate (COUNT(PETOT), COUNTindex(PETOT+1))
call MPI_allGATHER ( N     , 1, MPI_INTEGER,
&                    COUNT, 1, MPI_INTEGER,
&                    MPI_COMM_WORLD, ierr)
COUNTindex(1)= 0

do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo
```

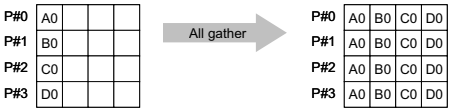
中身を書き出して見よう

各PEにおけるベクトル長さの情報が「COUNT」に入る(「rcounts」)

中身を書き出して見よう

MPI_Gatherv

MPI_Allgather



- MPI_Gather+MPI_Bcast
- **call MPI_Allgather (sendbuf, scount, sendtype, recvbuf, rcount, recvtype, comm, ierr)**
  - sendbuf 任意 I 送信バッファの先頭アドレス,
  - scount 整数 I 送信メッセージのサイズ
  - sendtype 整数 I 送信メッセージのデータタイプ
  - recvbuf 任意 O 受信バッファの先頭アドレス,
  - rcount 整数 I 受信メッセージのサイズ
  - recvtype 整数 I 受信メッセージのデータタイプ
  - comm 整数 I コミュニケータを指定する
  - ierr 整数 O 完了コード

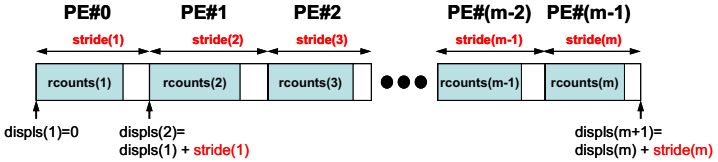
MPI_Gatherv

局所⇒全体ベクトル生成(2/2)

<$FVM>/gatherv/mpigathertest.*

```
do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0

call MPI_Gatherv                                              &
&     ( VEC , N, MPI_DOUBLE_PRECISION,                        &
&       VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,        &
&       0, MPI_COMM_WORLD, ierr)

if (my_rank.eq.0) then
  do i= 1, COUNTindex(PETOT+1)
    write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
  enddo
endif

call MPI_FINALIZE (ierr)

stop
end
```

「displs」に相当するものを生成。



size(recvbuf)= displs(PETOT+1)= sum(stride)

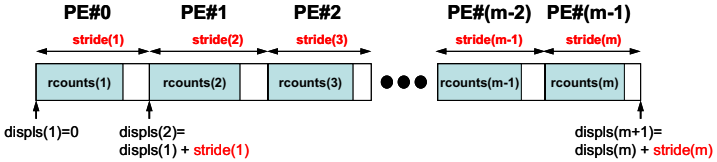
MPI_Gatherv

局所⇒全体ベクトル生成(2/2)

<$FVM>/gatherv/mpigathertest.*

```
do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0

call MPI_Gatherv                                              &
&     ( VEC , N, MPI_DOUBLE_PRECISION,                        &
&       VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,        &
&       0, MPI_COMM_WORLD, ierr)

if (my_rank.eq.0) then
  do i= 1, COUNTindex(PETOT+1)
    write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
  enddo
endif

call MPI_FINALIZE (ierr)

stop
end
```

「recbuf」のサイズ



size(recvbuf)= displs(PETOT+1)= sum(stride)

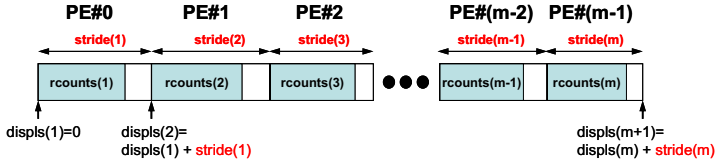
MPI_Gatherv

局所⇒全体ベクトル生成(2/2)

<$FVM>/gatherv/mpigathertest.*

```
do ip= 1, PETOT
  COUNTindex(ip+1)= COUNTindex(ip) + COUNT(ip)
enddo

allocate (VECg(COUNTindex(PETOT+1)))
VECg= 0.d0

call MPI_Gatherv                                              &
&    ( VEC , N, MPI_DOUBLE_PRECISION,                         &
&      VECg, COUNT, COUNTindex, MPI_DOUBLE_PRECISION,         &
&      0, MPI_COMM_WORLD, ierr)

if (my_rank.eq.0) then
  do i= 1, COUNTindex(PETOT+1)
    write (*,'(2i8,f10.0)') my_rank, i, VECg(i)
  enddo
endif

call MPI_FINALIZE (ierr)

stop
end
```

call MPI_Gatherv
(sendbuf, scount, sendtype, recvbuf, rcounts, displs, recvtype, root, comm, ierr)



size(recvbuf)= displs(PETOT+1)= sum(stride)

実 行

FORTRAN

```
$ cd <$FVM>/gatherv
$ mpif90 -Oss -noparallel mpigathertest.f

(go.sh修正)
$ qsub go.sh
```

C

```
$ cd <$FVM>/gatherv
$ mpicc -Os -noparallel mpigathertest.c

(go.sh修正)
$ qsub go.sh
```


手 順

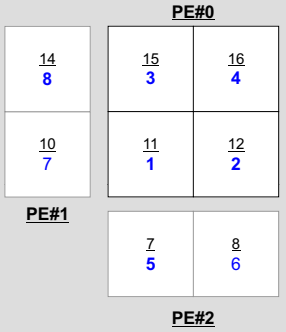
- 各プロセッサにおける計算結果(**PHI(:)**に格納される)を，あるプロセッサに集める：MPI_Gatherv使用
  - 「内点」の結果のみで良い。
- そのプロセッサから，結果を以下の書式で，グローバル番号順に書き出す(「,」は入れない)：

  <グローバル要素番号><計算結果>
- ファイルを「**fvm_entire_mesh.inp-geo**」にアペンドしてUCDファイルを生成する。
- ヒント
  - MPI_Gathervを使う
  - グローバルIDは局所分散通信ファイルより得る

局所分散通信ファイル(comm.0)

内点数，総要素数(内点＋外点)，全体要素番号

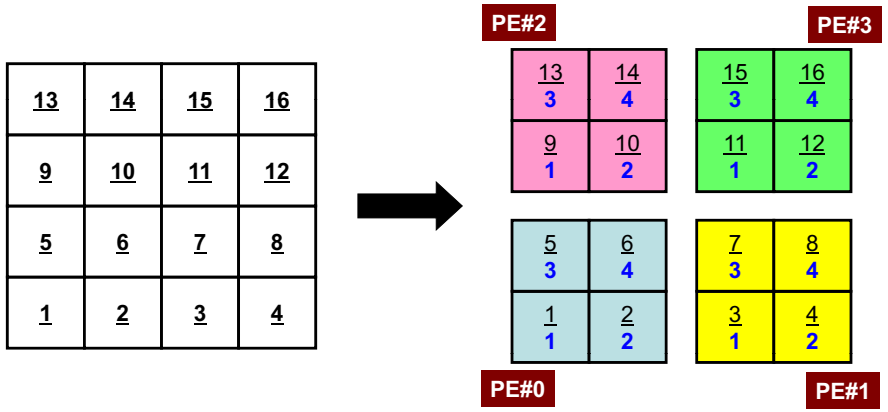
```
#NEIBPEtot
   2
#NEIBPE
   1   2
#IMPORT index
   2   4
#IMPORT items
   7   8   5   6
#EXPORT index
   2   4
#EXPORT items
   1   3   1   2
#INTERNAL NODE
   4
#TOTAL NODE
   8
#GLOBAL NODE ID                    全体要素番号(局所番号順)
  11  12  15  16   7   8
  10  14
```



要素のグローバルID hpcmw_eps_fvm_input_grid

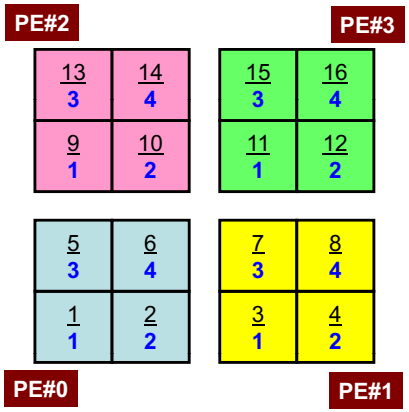
```
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_index(k), k= 1, n_neighbor_pe)
      nn= export_index(n_neighbor_pe)
      allocate (export_item(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (export_item(k), k= 1, nn)

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') intNODE_tot

      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') nn

      allocate (NODE_GLOBAL(nn))
      read (IUNIT,'(a)') LINE
      read (IUNIT,'(6i12)') (NODE_GLOBAL(k), k= 1, nn)

      close (IUNIT)
!C===

      end subroutine hpcmw_eps_fvm_input_grid
```


例題



内点

例題：計算結果例：PHIの中身



PE#	要素番号（ローカル）	要素番号（グローバル）	PHI(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
0	3	5	1.150000E+01
0	4	6	9.500000E+00
2	1	3	6.500000E+00
2	2	4	2.500000E+00
2	3	7	6.500000E+00
2	4	8	2.500000E+00
1	1	9	1.150000E+01
1	2	10	9.500000E+00
1	3	13	1.150000E+01
1	4	14	9.500000E+00
3	1	11	6.500000E+00
3	2	12	2.500000E+00
3	3	15	6.500000E+00
3	4	16	2.500000E+00

hpcmw_eps_fvm_global_output（1/2）

```
      subroutine hpcmw_eps_fvm_global_output
      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      integer, dimension(:), allocatable :: rcounts, displs
      integer, dimension(:), allocatable :: NODE_ID_G
      integer, dimension(:), allocatable :: NEWtoOLD
      real(kind=kreal), dimension(:), allocatable :: VAL
!C
!C-- INIT.
      allocate (rcounts(PETOT), displs(0:PETOT))
      rcounts= 0
      displs = 0

      call MPI_Allgather                                              &
     &     (intNODE_tot, 1, MPI_INTEGER, rcounts, 1, MPI_INTEGER,     &
     &      MPI_COMM_WORLD, ierr)

      do ip= 1, PETOT
        displs(ip)= displs(ip-1) + rcounts(ip)
      enddo

      NODE_tot_G= displs(PETOT)

      allocate (NODE_ID_G(NODE_tot_G), NEWtoOLD(NODE_tot_G))
      allocate (VAL(NODE_tot_G))
```

MPI_Gathervの準備

領域全体の要素数（内点数の和）

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_GATHERv                                                  &
     &     (NODE_GLOBAL, intNODE_tot,        MPI_INTEGER,               &
     &      NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,               &
     &      0, MPI_COMM_WORLD, ierr)

      call MPI_GATHERv                                                  &
     &     (PHI, intNODE_tot,        MPI_DOUBLE_PRECISION,              &
     &      VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,           &
     &      0, MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

グローバル要素番号 NODE_ID_G

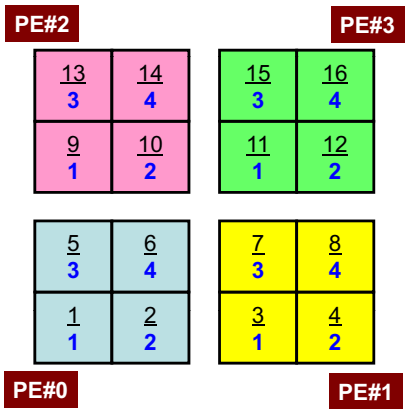
hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_GATHERv                                                  &
     &     (NODE_GLOBAL, intNODE_tot,        MPI_INTEGER,               &
     &      NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,               &
     &      0, MPI_COMM_WORLD, ierr)

      call MPI_GATHERv                                                  &
     &     (PHI, intNODE_tot,        MPI_DOUBLE_PRECISION,              &
     &      VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,           &
     &      0, MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

計算結果ベクトル VAL

この状態のVALの中身

PE番号順に格納されており，
グローバル要素番号順になっていない



PE#	i	要素番号（グローバル）NODE_ID_G(i)	VAL(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
0	3	5	1.150000E+01
0	4	6	9.500000E+00
1	5	3	6.500000E+00
1	6	4	2.500000E+00
1	7	7	6.500000E+00
1	8	8	2.500000E+00
2	9	9	1.150000E+01
2	10	10	9.500000E+00
2	11	13	1.150000E+01
2	12	14	9.500000E+00
3	13	11	6.500000E+00
3	14	12	2.500000E+00
3	15	15	6.500000E+00
3	16	16	2.500000E+00

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_GATHERv                                                  &
     &     (NODE_GLOBAL, intNODE_tot,        MPI_INTEGER,               &
     &      NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,               &
     &      0, MPI_COMM_WORLD, ierr)

      call MPI_GATHERv                                                  &
     &     (DELPHI, intNODE_tot,        MPI_DOUBLE_PRECISION,          &
     &      VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,           &
     &      0, MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown', position='append')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

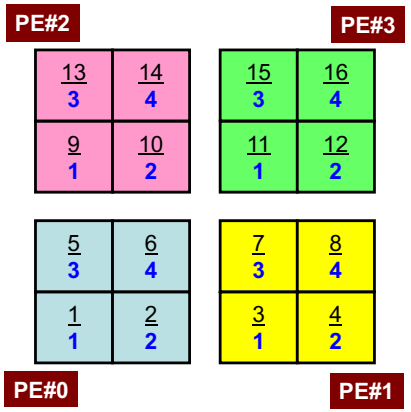
Global番号順に
並べ替え（my_rank=0のみ）

hpcmw_eps_fvm_global_output（2/2）

```
!C
!C-- GLOBAL ARRAY
      call MPI_GATHERv                                                  &
     &    (NODE_GLOBAL, intNODE_tot,         MPI_INTEGER,              &
     &     NODE_ID_G  , rcounts, displs(0), MPI_INTEGER,                &
     &     0, MPI_COMM_WORLD, ierr)

      call MPI_GATHERv                                                  &
     &    (DELPHI, intNODE_tot,         MPI_DOUBLE_PRECISION,           &
     &     VAL   , rcounts, displs(0), MPI_DOUBLE_PRECISION,            &
     &     0, MPI_COMM_WORLD, ierr)

      if (my_rank.eq.0) then
        do i= 1, NODE_tot_G
          j= NODE_ID_G(i)
          NEWtoOLD(j)= i
        enddo

        IUNIT= 12
        open (IUNIT, file= AVSfile, status='unknown')
          do j= 1, NODE_tot_G
            ii= NEWtoOLD(j)
            write (IUNIT,'(i8,1pe16.6)') j, VAL(ii)
          enddo
        close (IUNIT)
      endif

      end subroutine hpcmw_eps_fvm_global_output
```

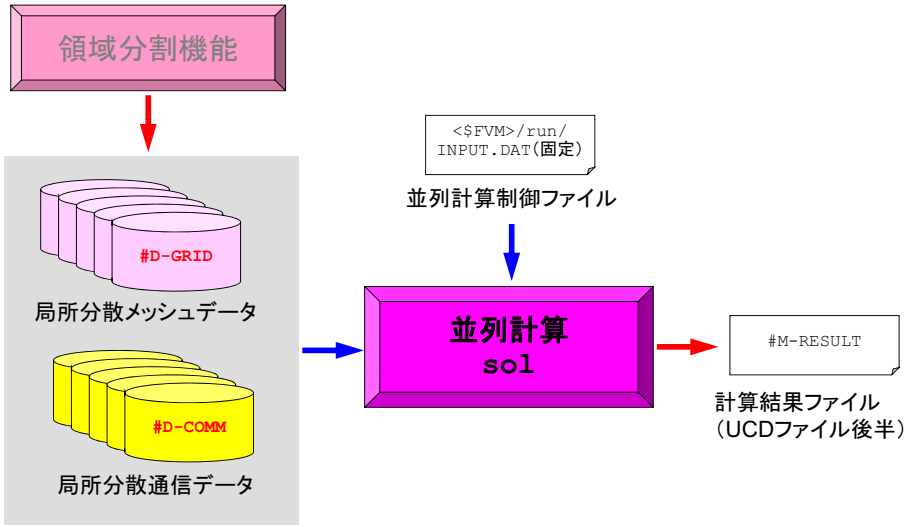
書き出し（my_rank=0のみ）

並び替えた後のVALの中身



PE#	i	要素番号（グローバル）	VAL(i)
0	1	1	1.150000E+01
0	2	2	9.500000E+00
1	3	3	6.500000E+00
1	4	4	2.500000E+00
0	5	5	1.150000E+01
0	6	6	9.500000E+00
1	7	7	6.500000E+00
1	8	8	2.500000E+00
2	9	9	1.150000E+01
2	10	10	9.500000E+00
3	11	11	6.500000E+00
3	12	12	2.500000E+00
2	13	13	1.150000E+01
2	14	14	9.500000E+00
3	15	15	6.500000E+00
3	16	16	2.500000E+00

並列シミュレーションにおけるI/O

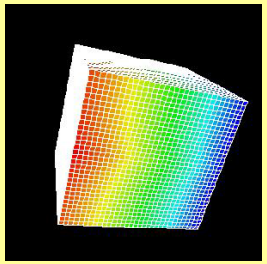


並列計算制御ファイル

- **INPUT.DAT**（名称固定）
- 実行形式「**sol**」と同じディレクトリになければならない（この場合は**<$FVM>/run**）。
- 全ての項目は省略不可。

```
../ex/mesh.rcb  局所分散メッシュファイルのヘッダ名
../ex/comm.rcb  局所分散通信ファイルのヘッダ名
../ex/result    可視化用出力ファイル名（後述）
1               可視化用出力の有無（=1のとき出力）
```

計算実行，ファイル生成



```
$> cd <$FVM>/run
$> cat INPUT.DAT
  ../ex/mesh.rcb
  ../ex/comm.rcb
  ../ex/result
  1

$> qsub go.sh
$> ls -l ../ex/result
  result
```

これが書きだされた結果ファイル

```
$> cd ../ex
$> cat fvm_entire_mesh.inp-geo result > test.inp
```

形状部分に結果ファイルをアペンドして，AVS出力用のファイルを生成する。

「eps_fvm」の並列化

```
program eps_fvm
use hpcmw_eps_fvm_all

implicit REAL*8 (A-H,O-Z)

call hpcmw_eps_fvm_init
call hpcmw_eps_fvm_input_grid
call poi_gen
call hpcmw_eps_fvm_solver
call hpcmw_eps_fvm_global_output

call hpcmw_eps_fvm_finalize

end program eps_fvm
```

可視化処理の部分を除くと，並列計算特有の処理は，共役勾配法の内積，行列ベクトル積程度。それも，hpcmw_eps_fvm_comm.＊のようなサブルーチン群を使用すれば，一行の挿入で済んでしまう。

並列計算の「隠蔽」

並列分散データ構造の適切な設計が全て：一般化された通信テーブル

並列化した「eps-fvm」

- 並列分散データ入力
- 共役勾配法
- 可視化
- 課題

課題1

- Strong Scaling
  - 問題サイズを固定，PE数を変化させて時間（全体，各部分）を測定。
- Weak Scaling
  - PEあたりの問題サイズを固定，1反復あたりの計算時間を求める。
- 考慮すべき項目
  - 問題サイズ
  - 領域分割手法（RCB，METIS，1D～3D）の影響。
- 注意
  - 通信時間を計測する工夫をして見よ。

反復回数を固定したい場合

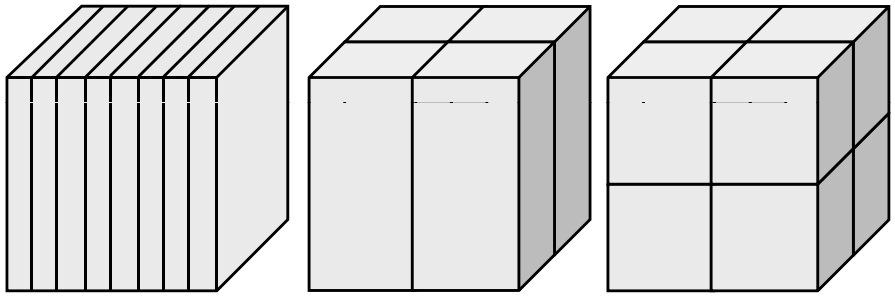
- hpcmw_eps_fvm_solver

```
!C
!C***
!C*** hpcmw_solver
!C***
!C
      subroutine hpcmw_eps_fvm_solver

      use hpcmw_eps_fvm_all
      implicit REAL*8 (A-H,O-Z)

      ISET= 0
      EPS = 1.d-8
      ITR = intNODE_tot
      call hpcmw_eps_fvm_allreduce_R (ITR, hpcmw_max)
      ITR = PETOT * ITR

      S_TIME= MPI_WTIME()
      call hpcmw_eps_fvm_solver_CG                                  &
     &    ( intNODE_tot, NODE_tot, N2, D, BFORCE, DELPHI, EPS, &
     &      ITR, IER, WAindex, WAcoef, COMMtime)
      E_TIME= MPI_WTIME()
      ISET= ISET + 1
```

この部分を例えば「ITR=50」のようにする。

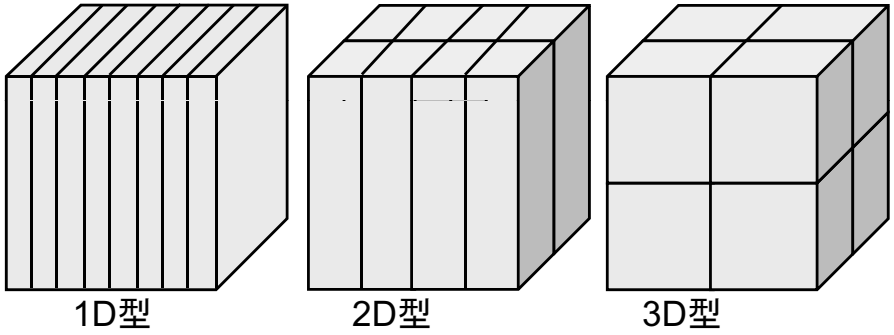
1D～3D分割

どの方法が良いか考えて見よ



1D～3D分割

通信量の総和（各辺4N，8領域とする）



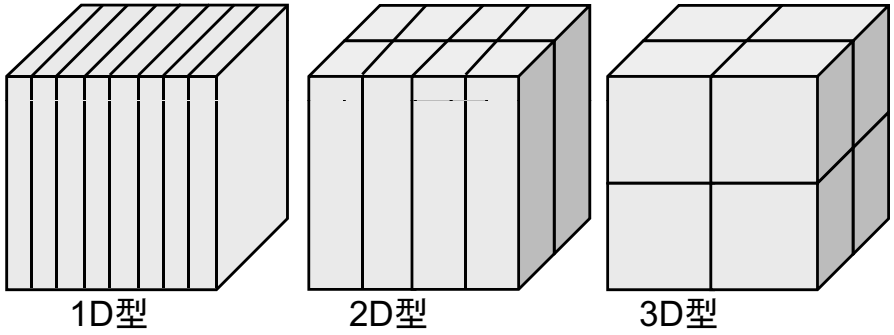
16 N^2 x 7 = 112 N^2

16 N^2 x 4 = 64 N^2

16 N^2 x 3 = 48 N^2

1D～3D分割

80x80x80要素，8領域の計算例



16 N^2 x 7 = 112 N^2

16 N^2 x 4 = 64 N^2

16 N^2 x 3 = 48 N^2

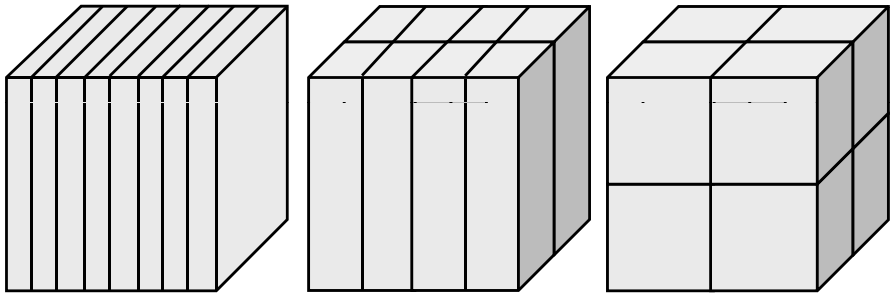
```
### solver time      8.867188E+00
### comm.  time      3.488281E+00
### work ratio       6.066079E+01
```

```
### solver time      6.574219E+00
### comm.  time      1.285156E+00
### work ratio       8.045157E+01
```

```
### solver time      6.230469E+00
### comm.  time      1.031250E+00
### work ratio       8.344828E+01
```

1D～3D分割

fvmpart.ctrl




```
!METHOD
RCB
X,X,X

!REGION NUMBER
8
```

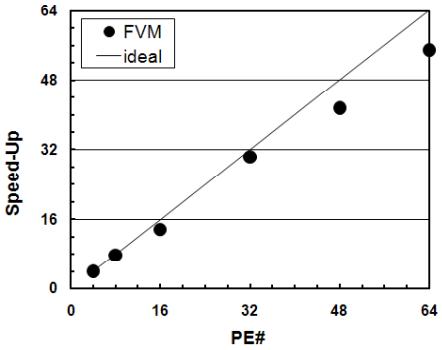
```
!METHOD
RCB
X,Y,X

!REGION NUMBER
8
```

```
!METHOD
RCB
X,Y,Z

!REGION NUMBER
8
```


ジョブスクリプト：go.sh

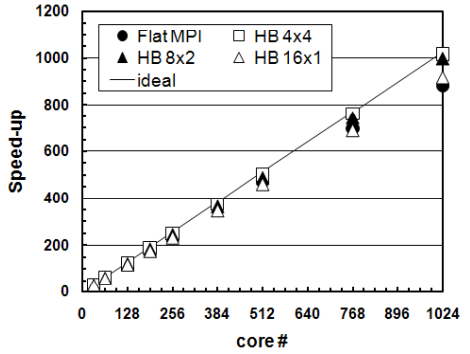
```
#@$-r test
#@$-q tutorial
#@$-N 1
#@$-J T8
#@$-e err
#@$-o aaa.lst
#@$-lM 28GB
#@$-lT 00:05:00
#@$-s /bin/sh
#@$

cd $PBS_O_WORKDIR
mpirun numactl --localalloc ./sol
```


計算例：Strong Scaling：T2K東大



- 100^3要素
  - 問題規模固定
- 4コアの場合を基準

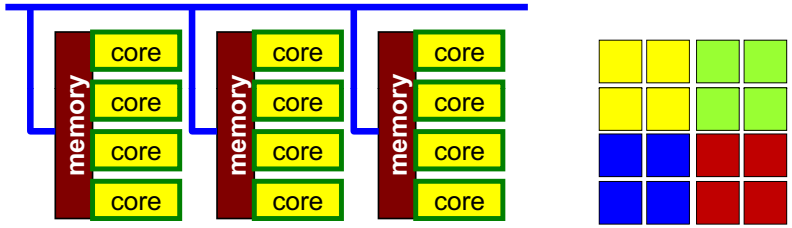
計算例（三次元弾性問題）（T2K）



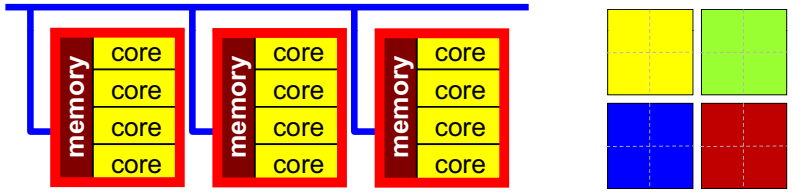
- 8,957,952自由度（3×144^3），Strong Scaling
- 32～1024コア（32コア Flat MPIの場合を基準）

Flat MPI vs. Hybrid

Flat-MPI：Each PE -> Independent

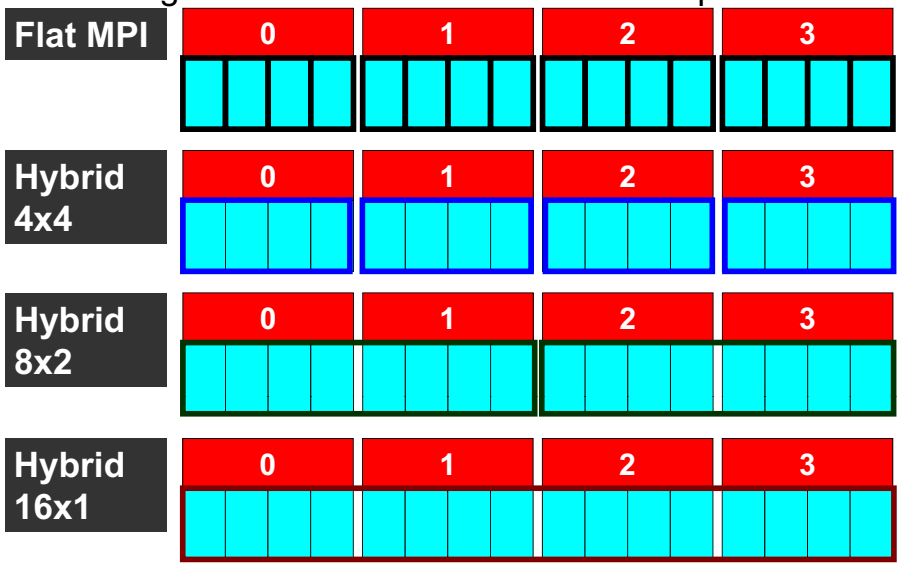


Hybrid：Hierarchal Structure (OpenMP+MPI)



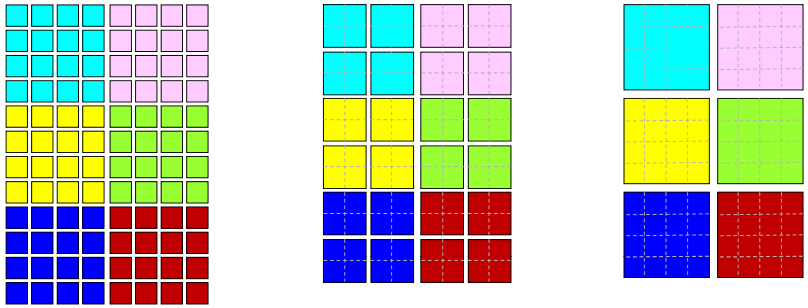
Flat MPI, Hybrid (4x4, 8x2, 16x1)

Higher Performance of HB16x1 is important



Domain Decomposition

example: 6 nodes, 24 sockets, 96 cores



Flat MPI　　HB 4x4　　HB 16x1

理想値からのずれ

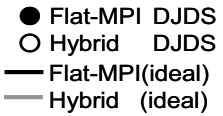
- MPI通信そのものに要する時間
  - データを送付している時間
  - ノード間においては通信バンド幅によって決まる
  - 通信時間は送受信バッファのサイズに比例
- MPIの立ち上がり時間
  - latency
  - 送受信バッファのサイズによらない
    - 呼び出し回数依存，プロセス数が増加すると増加する傾向
  - 通常，数～数十μsecのオーダー
- MPIの同期のための時間
  - プロセス数が増加すると増加する傾向
- 計算時間が小さい場合（問題規模が小さい場合）はこれらの効果を無視できない。
  - 特に，送信メッセージ数が小さい場合は，「Latency」が効く。

研究例

- Weak Scaling
  - プロセッサあたりの問題規模を固定
  - プロセッサ数を増加させる
  - 「計算性能」としては不変のはずであるが，普通はプロセッサ数を増やすと性能は悪化

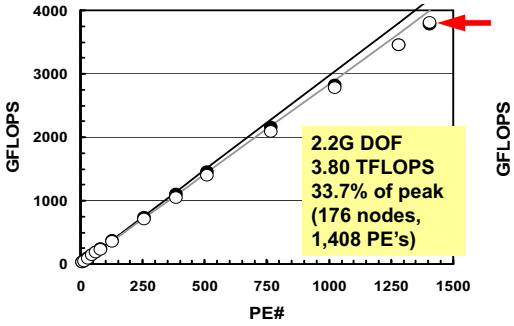
- Nakajima, K. (2007), The Impact of Parallel Programming Models on the Linear Algebra Performance for Finite Element Simulations, Lecture Notes in Computer Science 4395, 334-348.
  - 並列有限要素法（特に latency の影響大）

Weak Scaling: LARGE



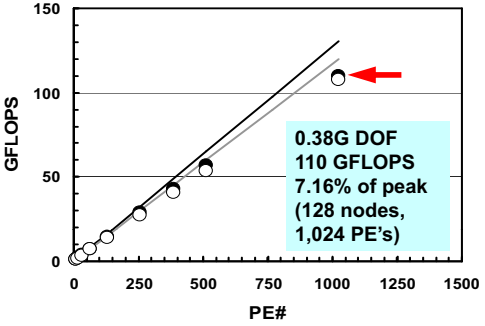
Earth Simulator

1,572,864 DOF/PE
(=3x128x64x64)

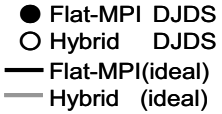


IBM SP-3 (Seaborg at LBNL)

375,000 DOF/PE
(=3x50^3)

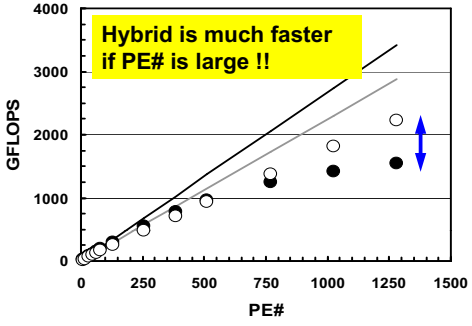


Weak Scaling: SMALL



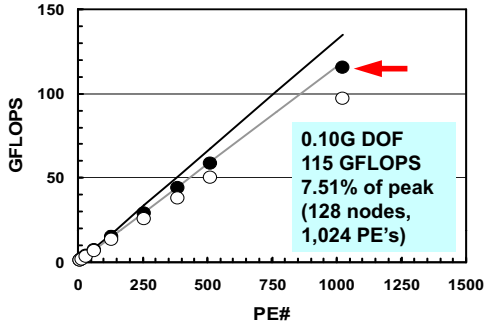
Earth Simulator

98,304 DOF/PE
(=3x32^3)



IBM SP-3 (Seaborg at LBNL)

98,304 DOF/PE
(=3x32^3)



通信：メモリーコピー

実は意外にメモリの負担もかかる

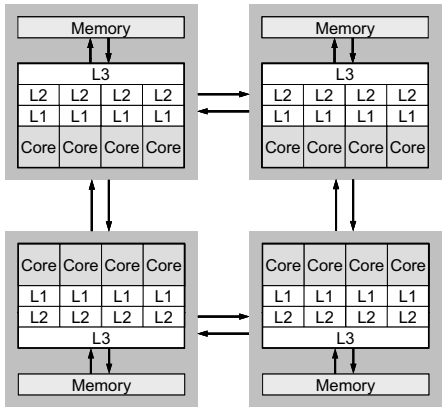
```
do neib= 1, NEIBPETOT
  do k= export_index(neib-1)+1, export_index(neib)
    kk= export_item(k)
    SENDbuf(k)= VAL(kk)
  enddo
enddo

do neib= 1, NEIBPETOT
  iS_e= export_index(neib-1) + 1
  iE_e= export_index(neib  )
  BUFlength_e= iE_e + 1 - iS_e
  iS_i= import_index(neib-1) + 1
  iE_i= import_index(neib  )
  BUFlength_i= iE_i + 1 - iS_i

  call MPI_SENDRECV                                          &
&         (SENDbuf(iS_e), BUFlength_e, MPI_INTEGER, NEIBPE(neib), 0,&
&          RECVbuf(iS_i), BUFlength_i, MPI_INTEGER, NEIBPE(neib), 0,&
&          MPI_COMM_WORLD, stat_sr, ierr)
  enddo

  do neib= 1, NEIBPETOT
    do k= import_index(neib-1)+1, import_index(neib)
      kk= import_item(k)
      VAL(kk)= RECVbuf(k)
    enddo
  enddo
```

課題2

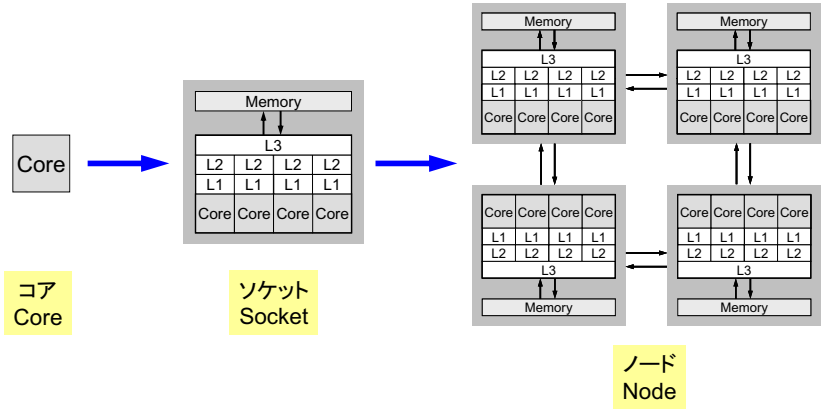
- 更にnumactlの影響について検討せよ
  - 問題サイズの影響
    - 全体の問題サイズ，1コアあたりの問題サイズ
  - 4ノードまで使用して見よ
  - 「**--physcpubind**」の影響についても検討せよ：コアを指定できる
  - 1ノードのうち8コアを選択して使用する場合について検討してみよ
  - プロセス番号～コア番号の関係についても検討してみよ

Quad Core Opteron: NUMA Architecture

- AMD Quad Core Opteron 2.3GHz
  - Quad Coreのソケット×4 ⇒ 1ノード（16コア）
- 各ソケットがローカルにメモリを持っている
  - NUMA：Non-Uniform Memory Access
  - できるだけローカルのメモリをアクセスして計算するようなプログラミング，データ配置，実行時制御（numactl）が必要



NUMA Architecture



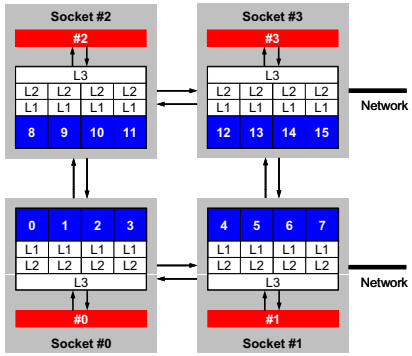
numactl

- NUMA(Non Uniform Memory Access) 向けのメモリ割付のためのコマンドライン：Linuxでサポート
  - http://www.novell.com/collateral/4621437/4621437.pdf

```
>$ numactl --show

policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
cpubind: 0 1 2 3
nodebind: 0 1 2 3
membind: 0 1 2 3
```

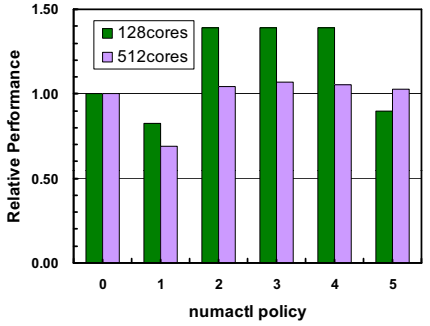
numactl --show

```
>$ numactl --show

policy: default
preferred node: current
physcpubind: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15     コア
cpubind: 0 1 2 3                                       ソケット
nodebind: 0 1 2 3                                      ソケット
membind: 0 1 2 3                                       メモリ
```



numactlの影響

- T2K，有限要素法アプリケーション，Strong Scaling
- 128コア：1コアあたりの問題サイズは512コアの4倍
  - メモリへの負担大
- POLICY=0：何も指定しない場合
- 相対性能：大きいほど良い



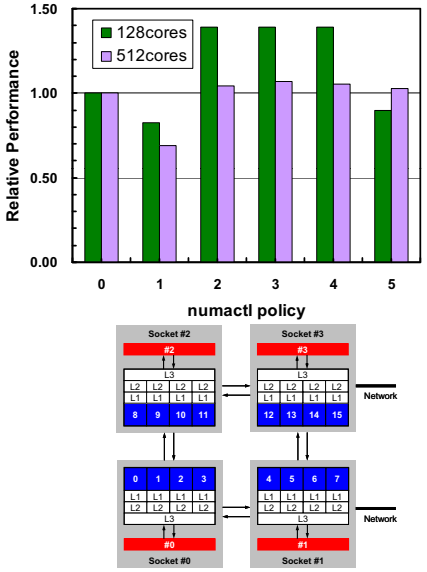

```
#@$-r HID-org
#@$-q h08nkl32
#@$-N 24
#@$-J T16
#@$-e err
#@$-o x384-40-1-a.lst
#@$-lM 27GB
#@$-lE 03:00:00
#@$-s /bin/sh
#@$

cd /XXX
mpirun ./numarun.sh ./sol
exit
```


numarun.shの実例

- T2K上

- **<$FVM>/run/go2.sh**
- **<$FVM>/run/numarun.sh**
- **<$FVM>/run/n1.sh, n2.sh**

numarun.shの中身



Policy:1
```
#!/bin/bash
MYRANK=$MXMPI_ID MPIのプロセス番号
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --interleave=all $@
```

Policy:2
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --interleave=$SOCKET $@
```

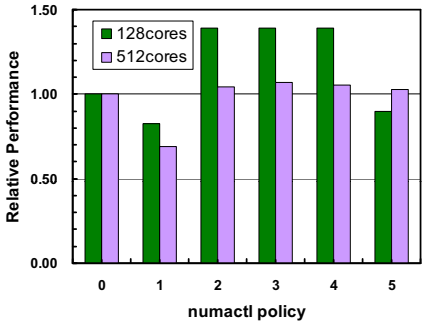
Policy:3
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```

Policy:4
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --localalloc $@
```

Policy:5
```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --localalloc $@
```

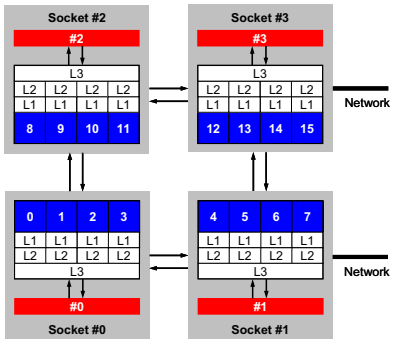
numactlの影響

- 128コア：1コアあたりの問題サイズは512コアの4倍
  - メモリへの負担大
- メモリへの負担が大きい場合，特にnumactl指定の影響は大




```
#@$-r HID-org
#@$-q h08nkl32
#@$-N 24
#@$-J T16
#@$-e err
#@$-o x384-40-1-a.lst
#@$-lM 27GB
#@$-lE 03:00:00
#@$-s /bin/sh
#@$

cd /XXX
mpirun ./numarun.sh ./sol
exit
```


いくつかのヒント（1/3）

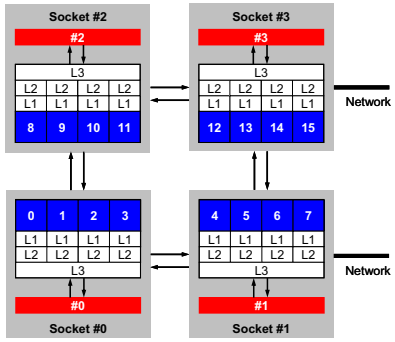


Policy-3での割付はこのようになっている

```
Policy:3
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```

MPI process	Socket	Memory	Core
0	0	0	
1	0	0	
2	0	0	
3	0	0	
4	1	1	
5	1	1	
6	1	1	
7	1	1	
8	2	2	
9	2	2	
10	2	2	
11	2	2	
12	3	3	
13	3	3	
14	3	3	
15	3	3	

いくつかのヒント（2/3）

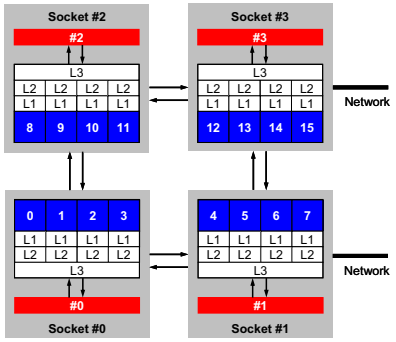


こんな風にもできる

```
#!/bin/bash
MYRANK=$MXMPI_ID
SOCKET=$(expr $MYRANK % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```

MPI process	Socket	Memory	Core
0	0	0	
1	1	1	
2	2	2	
3	3	3	
4	0	0	
5	1	1	
6	2	2	
7	3	3	
8	0	0	
9	1	1	
10	2	2	
11	3	3	
12	0	0	
13	1	1	
14	2	2	
15	3	3	

いくつかのヒント（3/3）

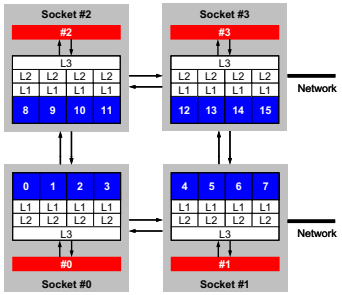


こんな風にもできる

```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
CORE=$(expr $MYRANK % 16)
numactl --physcpubind=$CORE --membind=$SOCKET $@
```

MPI process	Socket	Memory	Core
0		0	0
1		0	1
2		0	2
3		0	3
4		1	4
5		1	5
6		1	6
7		1	7
8		2	8
9		2	9
10		2	10
11		2	11
12		3	12
13		3	13
14		3	14
15		3	15

8コア使う場合はどうする？

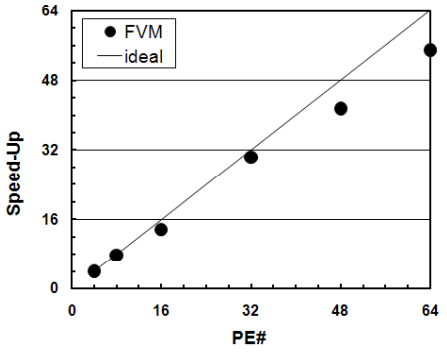


MPI process	Socket	Memory	Core
0		0	0
1		0	2
2		1	4
3		1	6
4		2	8
5		2	10
6		3	12
7		3	14

MPI process	Socket	Memory	Core
0		0	0
1		0	1
2		0	2
3		0	3
4		1	4
5		1	5
6		1	6
7		1	7

MPI process	Socket	Memory	Core
0		0	0
1		0	1
2		1	4
3		1	5
4		2	8
5		2	9
6		3	12
7		3	13

計算例：Strong Scaling：T2K東大



- 100^3要素
- 4コアの場合を基準
- Policy-3適用

Policy:3

```
#!/bin/bash
MYRANK=$MXMPI_ID
MYVAL=$(expr $MYRANK / 4)
SOCKET=$(expr $MYVAL % 4)
numactl --cpunodebind=$SOCKET --membind=$SOCKET $@
```


終わりに

- 「並列化」
  - 分散データ構造の重要性
  - 元のアプリケーションに対する深い理解が必要
  - アプリケーションによって並列化の戦略も異なる

- 5月24日まで利用できます。
- 質問，相談等は随時（5月24日以降もOK，ただ最近レスポンスが悪くて不評）
- アンケートにご協力ください

- 領域分割ツール等
  - /home/t00000/pre.tar

今後の技術的動向

- マルチコア
- GPU

- ハイブリッド並列プログラミングモデル
  - 例：OpenMP+MPI
  - MPIの部分は本講習会のものと基本的に変わらない

今後の講習会開催予定

名称	期間	時期（予定）	内容
MPI基礎	1日半 ～2日	2010年7月22・23日 2010年9月 2・ 3日 2011年3月17・18日	MPIによる並列プログラミングの基礎に関する講習，実習 • 並列化の基礎知識 • MPIのAPI説明 • 行列積の並列化実習 • makeを使った分割コンパイルと並列処理 • T2K（東大）による実習
MPI応用	1日半	2010年5月13・14日 2010年12月（未定）	MPIを使用した並列アプリケーション開発手法に関する講習，実習 • 有限体積法によるポアソン方程式ソルバーの概要 • 並列データ構造の考え方 • 領域分割手法 • 並列化手法 • T2K（東大）による実習
OpenMP （基礎＋応用）	1日半 ～2日	2010年9月（未定）	OpenMPによるマルチコアプロセッサ向け並列プログラミング，最適化手法に関する，実アプリケーションに基づく講習，実習 • 有限体積法によるポアソン方程式ソルバー，ICCG法の概要 • OpenMPの基礎 • リオーダリングによる並列化，最適化 • T2K（東大）による実習
ライブラリ利用	2日	2010年9月27・28日 2011年2月 2・ 3日	密行列ライブラリBLAS, LAPACK, ScaLAPACK、および、疎行列ライブラリPETsc, Lisの利用法に関する講習，実習 • 数値解法の原理と特徴の説明 • 数理的モデリング，離散化，データ格納 • ブロック化、データ分散の考え方 • T2K（東大）による実習