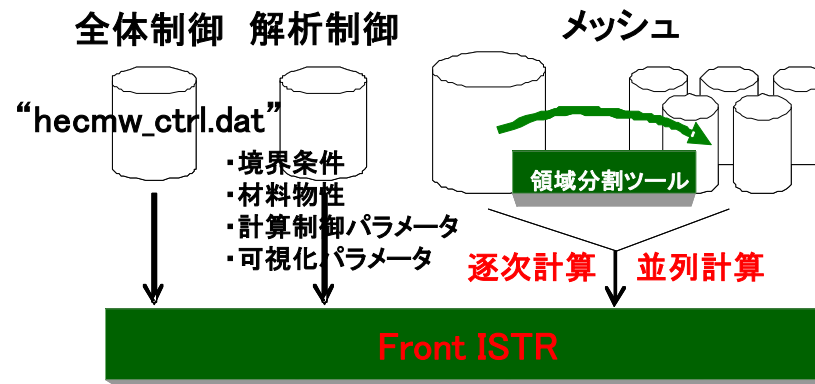
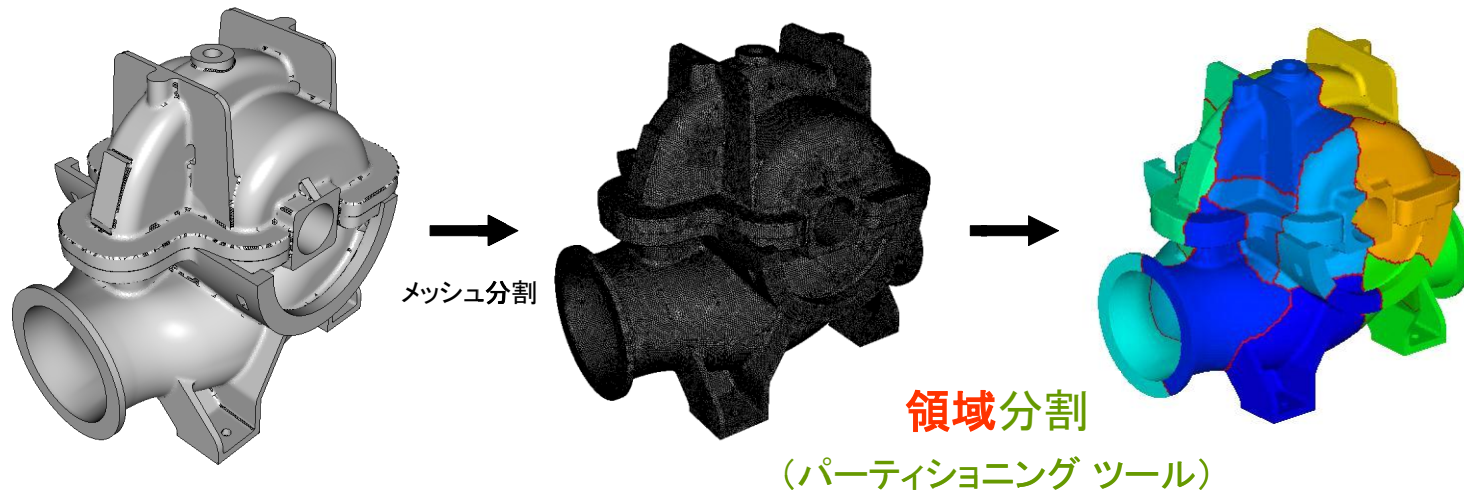


FrontISTRにおける 並列計算のしくみ <領域分割に基づく並列FEM>



FrontISTRにおける 並列計算のしくみ <領域分割に基づく並列FEM>

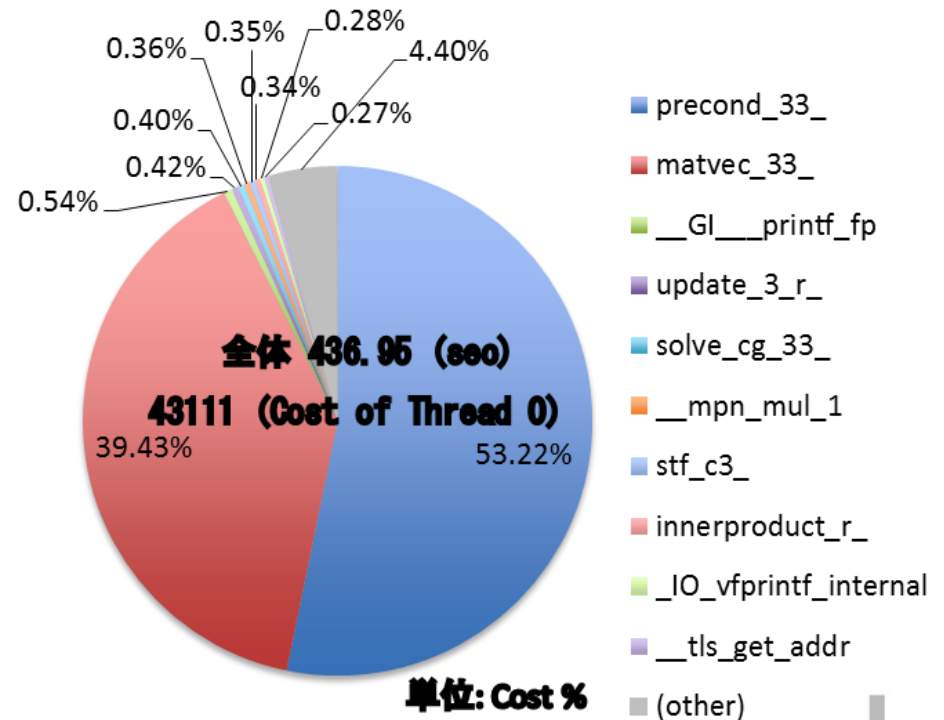
FEMの主な演算

- 剛性マトリックスの作成 → 部分領域(要素)ごとに並列処理可
- 剛性行列の求解 {反復法ソルバー, 直接法ソルバー}

反復法ソルバー

4種類の演算からなる

- (1) 疎行列・ベクトル積
- (2) ベクトル・ベクトル内積
- (3) ベクトルの加減(DAXPY)
- (4) 前処理



前処理付きCG法のアルゴリズム

compute $r^{(0)} = b - Ax^{(0)}$ for some initial guess $x^{(0)}$

for $i = 1, 2, \dots$

solve $M z^{(i-1)} = r^{(i-1)}$ (M: preconditioning matrix)

Preconditioning

$\rho_{i-1} = r^{(i-1)T} z^{(i-1)}$

Dot Product (1)

if $i=1$

$p^{(1)} = z^{(0)}$

else

$\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$

$p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$

DAXPY (1)

endif

$q^{(i)} = A p^{(i)}$

MATVEC

$\alpha_i = \rho_{i-1} / (p^{(i)T} q^{(i)})$

Dot Product (2)

$x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$

DAXPY (2)

$r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$

DAXPY (3)

check convergence; continue if necessary

end

■ 反復法ソルバーの並列処理

■ 4種類の演算からなる → 通信しながら部分領域ごとに並列処理可

- (1) 疎行列・ベクトル積
- (2) ベクトル・ベクトル内積
- (3) ベクトル(およびその実数倍)の加減(DAXPY) 通信不要
- (4) 前処理

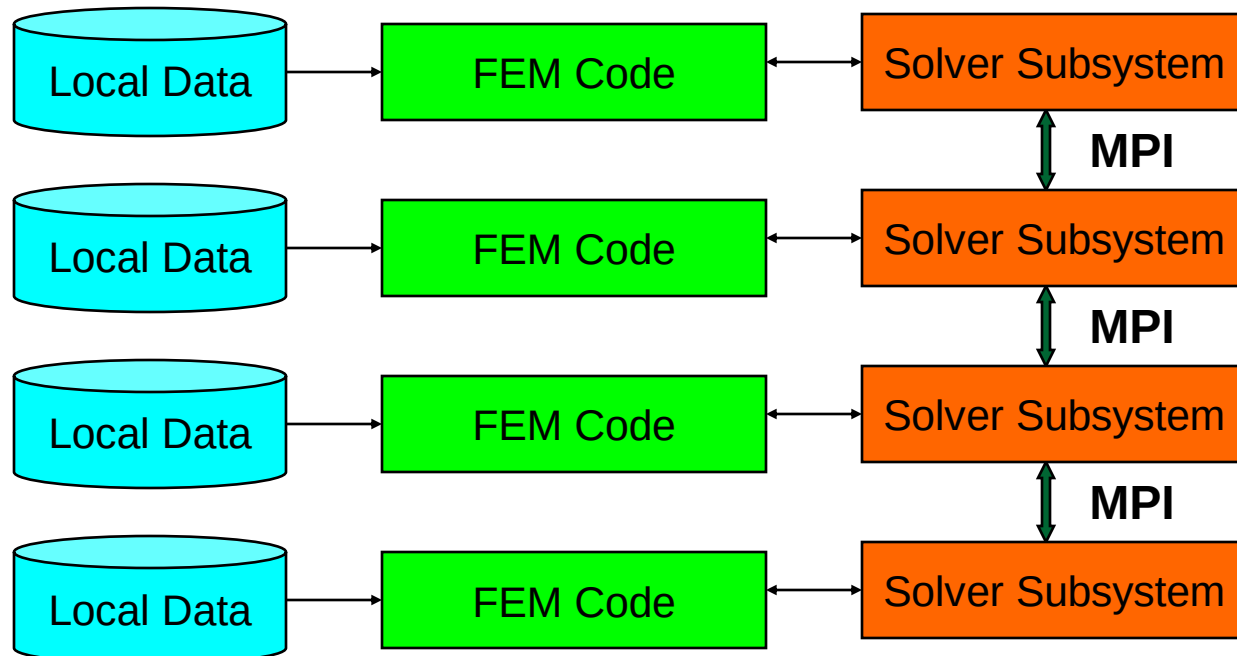


FrontISTRにおける 並列計算のしくみ <領域分割に基づく並列FEM>

- 領域分割 (domain decomposition, partitioning)
 - 分散メッシュ = (通常の)メッシュ情報 + 通信テーブル
- SPMD (Single Program Multiple Data) プログラム
 - 部分領域ごとに(通常の)FEM計算 + 通信
 - 通信はMPIによる
 - 通信部分のプログラムは HEC-MW によって”隠蔽”

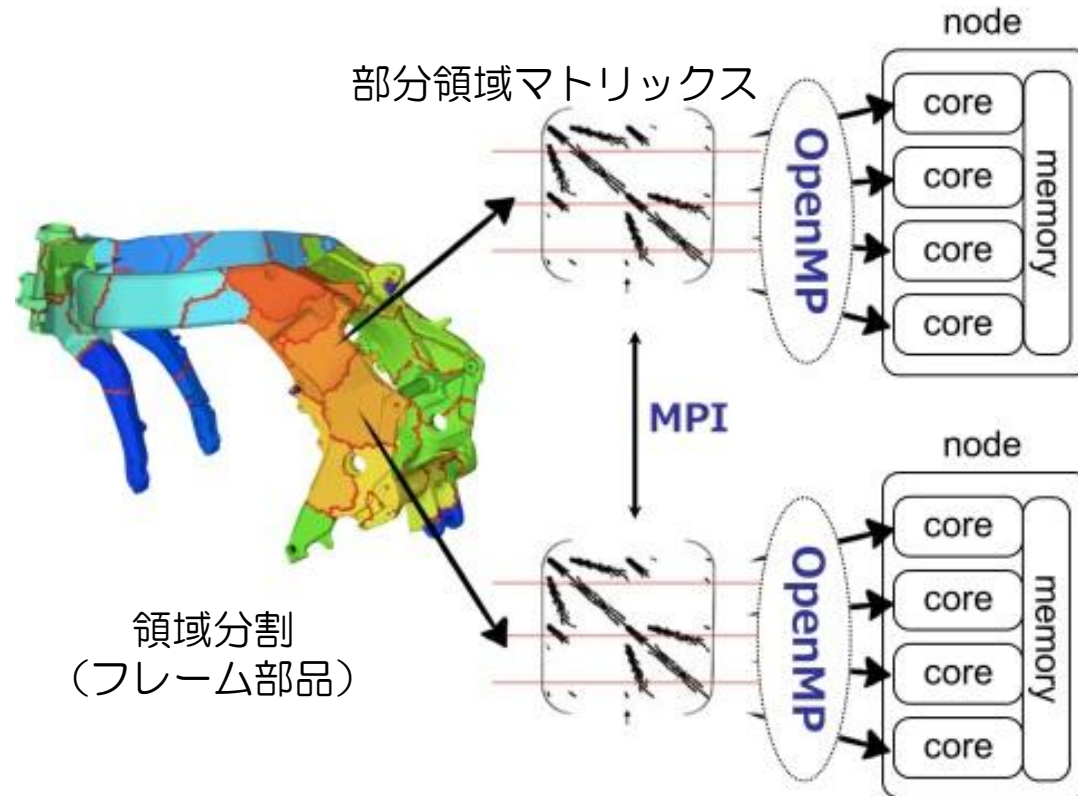
SPMD Programming Style

- Large file handling → Local distributed data
- FE analysis modules just consider local operation (element matrix assemble)
- Global operation occurs only in linear solver.



参考：FrontISTRにおける 並列計算のしくみ <ハイブリッド並列>

注意：本日の話題は **MPI 並列のみ**です

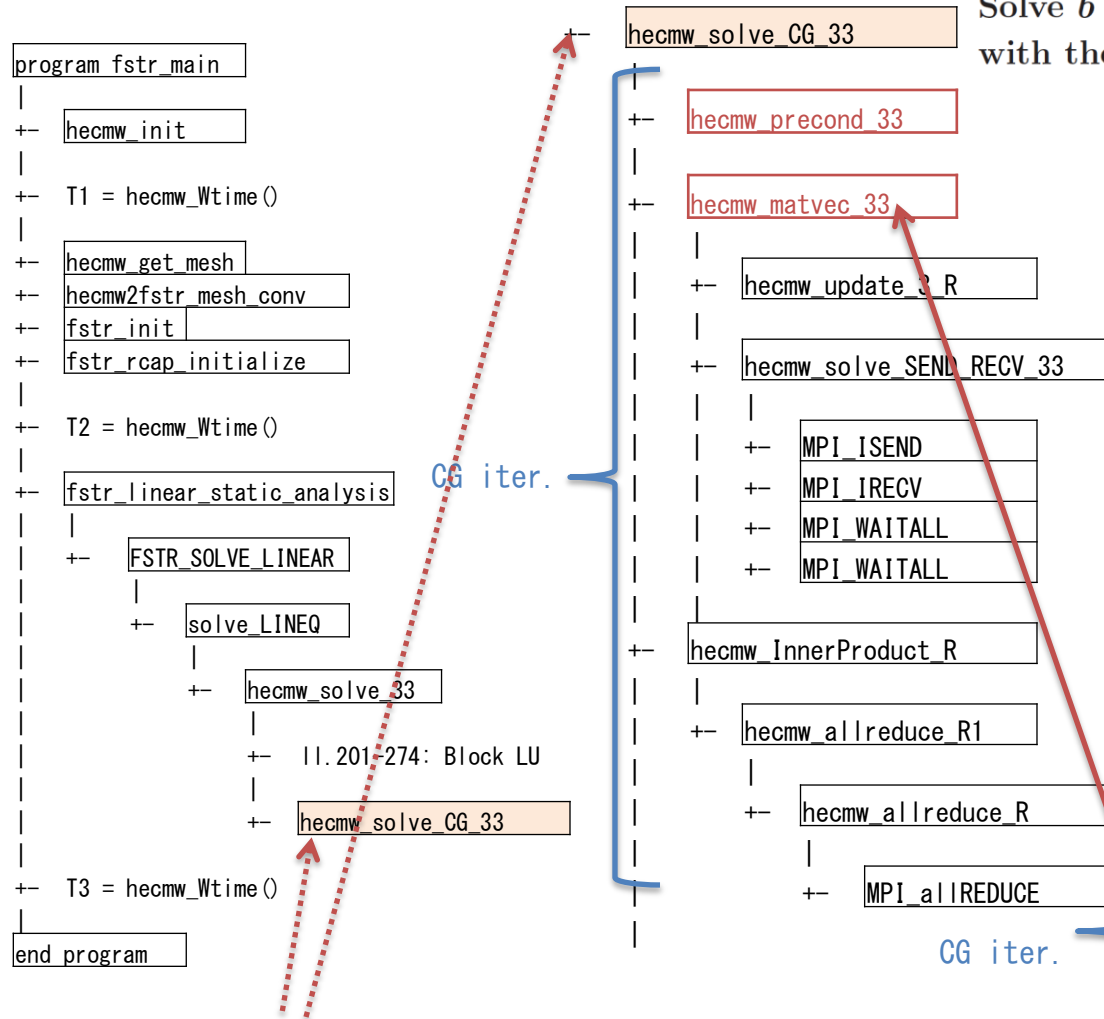


部分領域間通信 スレッド並列

MPI - OpenMP ハイブリッド並列

Solve $b = Ax$ by the Conjugate Gradient (CG) method with the left preconditioner:

0. Choose the preconditioner M such that $M \simeq A$
 - 0-a. by Incomplete LU factorization, $M = LU$,
 - 0-b. by Diagonal block scaling, or others
1. Set $x = x_{\text{ini}}$: initial guess, and b : rhs vector.
2. Set MAXIT: maximum number of iterations, and TOL: tolerance of convergence.
3. $r = b - Ax$
4. do iter = 1, MAXIT
 5. $z = M^{-1}r$
 6. $\rho = (r, z)$
 7. if (iter .eq. 1) then
 8. $p = z$
 9. else
 10. $\beta = \rho / \rho_1$
 11. $p = z + \beta p$
 12. endif
 13. $q = Ap$ 行列ベクトル積
 14. $\alpha = \rho / (p, q)$
 15. $x = x + \alpha p$, $r = r - \alpha q$
 16. Compute the scaled residual norm, $\text{RESID} = \|r\| / \|b\|$
 17. if (RESID .le. TOL) exit do-loop
 18. $\rho_1 = \rho$
 19. enddo



連立一次方程式の求解 (CG法)

FrontISTRプログラムの構造 -- 並列CG法

HEC-MW構造体(1) 分散メッシュ構造体

(PEおよび通信情報)

type hecmwST_local_mesh :: hecMESH

内容) 節点・要素・材料情報, PEおよび通信情報

特徴) hecmw_get_mesh によりセット.

HPC-MW ライブラリのハンドルの役割も

局所データには部分領域間の通信テーブルも含まれている

構造体成分(抜粋)

(2) PEおよび通信情報

integer(kind=kint)	:: zero	#0領域か?
integer(kind=kint)	:: my_rank	ランク
integer(kind=kint)	:: PETOT	総領域数
integer(kind=kint)	:: n_subdomain	総領域数 (局所分散データからの読み込み)
integer(kind=kint)	:: n_neighbor_pe	隣接領域数
integer(kind=kint), pointer	:: neighbor_pe(:)	隣接領域ID
integer(kind=kint), pointer	:: import_index(:)	受信テーブル用一次元インデックス
integer(kind=kint), pointer	:: import_item(:)	受信テーブル配列
integer(kind=kint), pointer	:: export_index(:)	受信テーブル用一次元インデックス
integer(kind=kint), pointer	:: export_item(:)	受信テーブル配列
integer(kind=kint), pointer	:: shared_index(:)	送受信テーブル用一次元インデックス
integer(kind=kint), pointer	:: shared_item(:)	受信テーブル配列

(1) 全体情報 ...前出

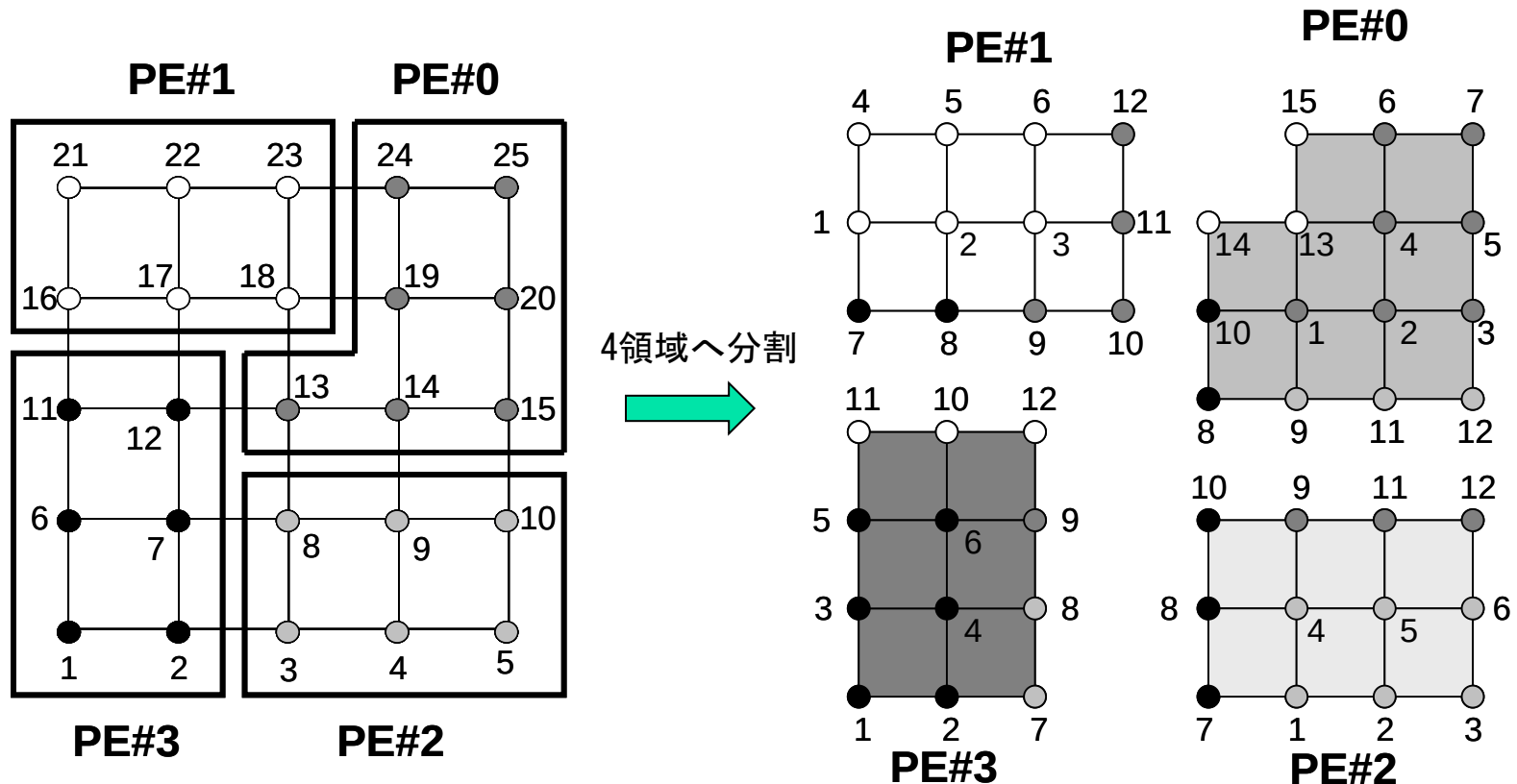
(3) 節点情報 ...前出

(4) 要素情報 ...前出

(5) 下部構造 ...前出

領域間オーバーラップ要素を含む節点ベース領域分割

剛性マトリクスの足し込みなどの処理を**各領域で並列に**実施するためには、オーバーラップ要素の情報が必要



節点は、通信の観点から以下の3種類に分類される：

- ・ 内点 (Internal Nodes) : 各領域に割り当てられた節点
- ・ 外点 (External Nodes) : 他の領域に属しているが、各領域の要素に含まれている節点
- ・ 境界点 (Boundary Nodes) : 他の領域の外点となっている内点

{ } 内は PE#0 の例

{1, 2, 3, 4, 5, 6, 7}

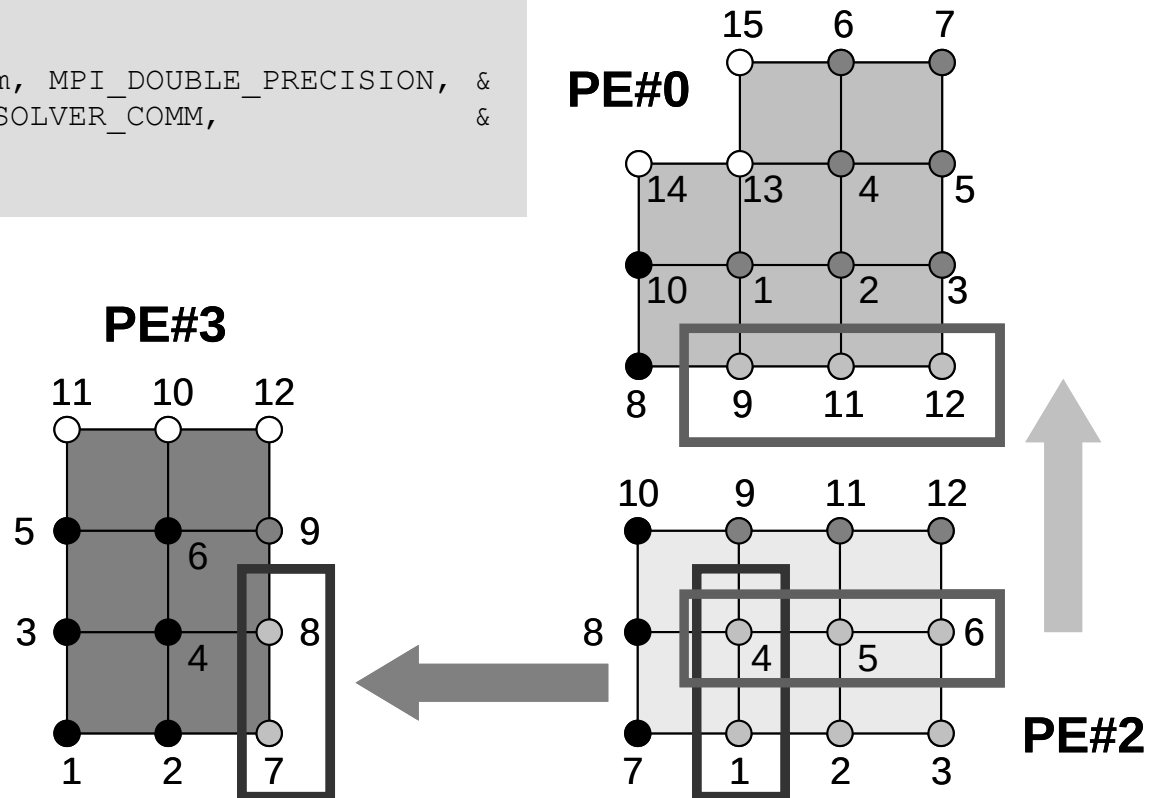
{8, 9, 10, 11, 12, 13, 14, 15}

{1, 2, 3, 4, 6}

境界点における値は隣接領域へ「送信 (send)」され、送信先では外点として「受信 (receive)」される

SEND phase (PE#2 の例)

```
do neib= 1, NEIBPETOT
  istart= EXPORT_INDEX(neib-1)
  inum  = EXPORT_INDEX(neib) - istart
  do k= istart+1, istart+inum
    WS(k)= X(EXPORT_NODE(k))
  enddo
  call MPI_ISEND
    (WS(istart+1), inum, MPI_DOUBLE_PRECISION, &
     NEIBPE(neib), 0, SOLVER_COMM,
     req1(neib), ierr)
enddo
```



(つづき)

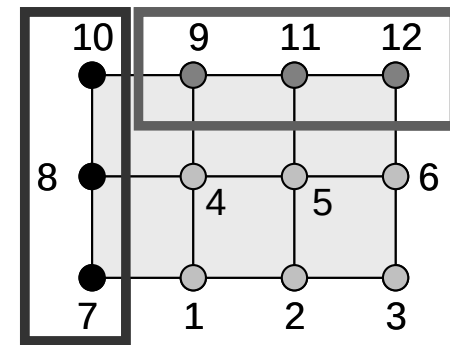
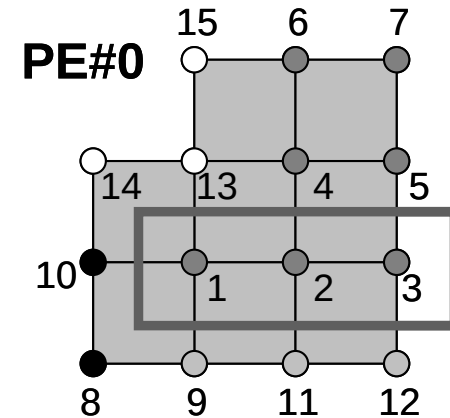
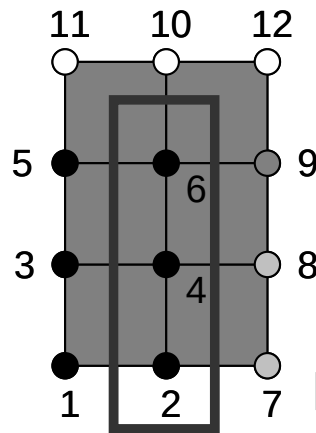
RECEIVE phase (PE#2 の例)

```
do neib= 1, NEIBPETOT
  istart= IMPORT_INDEX(neib-1)
  inum  = IMPORT_INDEX(neib ) - istart
  call MPI_Irecv
    (WR(istart+1), inum, MPI_DOUBLE_PRECISION, &
     NEIBPE(neib), 0, SOLVER_COMM,
     req2(neib), ierr)
enddo

call MPI_WAITALL (NEIBPETOT, req2, sta2, ierr)

do neib= 1, NEIBPETOT
  istart= IMPORT_INDEX(neib-1)
  inum  = IMPORT_INDEX(neib ) - istart
  do k= istart+1, istart+inum
    X(IMPORT_NODE(k))= WR(k)
  enddo
enddo

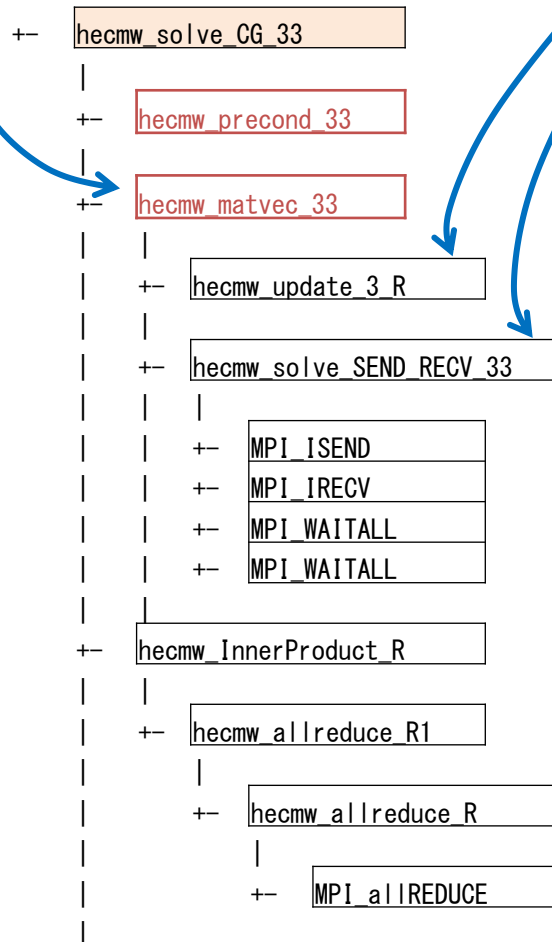
call MPI_WAITALL (NEIBPETOT, req1, stal, ierr)
```



PE#2

疎行列・ベクトル積

hecmw_matvec の中では、前回の通信を行ったのちに、部分領域ごとに並列に以下の行列・ベクトル積を実行する



```

do i= 1, N
  isL= INL(i-1) + 1
  ieL= INL(i )
  WVAL= WW(i,R)
  do j= isL, ieL
    inod = IAL(j)
    WVAL= WVAL - AL(j) * WW(inod,Z)
  enddo
  WW(i,Z)= WVAL * DD(i)
enddo

do i= N, 1, -1
  SW = 0.0d0
  isU= INU(i-1) + 1
  ieU= INU(i )
  do j= isU, ieU
    inod = IAU(j)
    SW= SW + AU(j) * WW(inod,Z)
  enddo
  WW(i,Z)= WW(i,Z) - DD(i) * SW
enddo
  
```

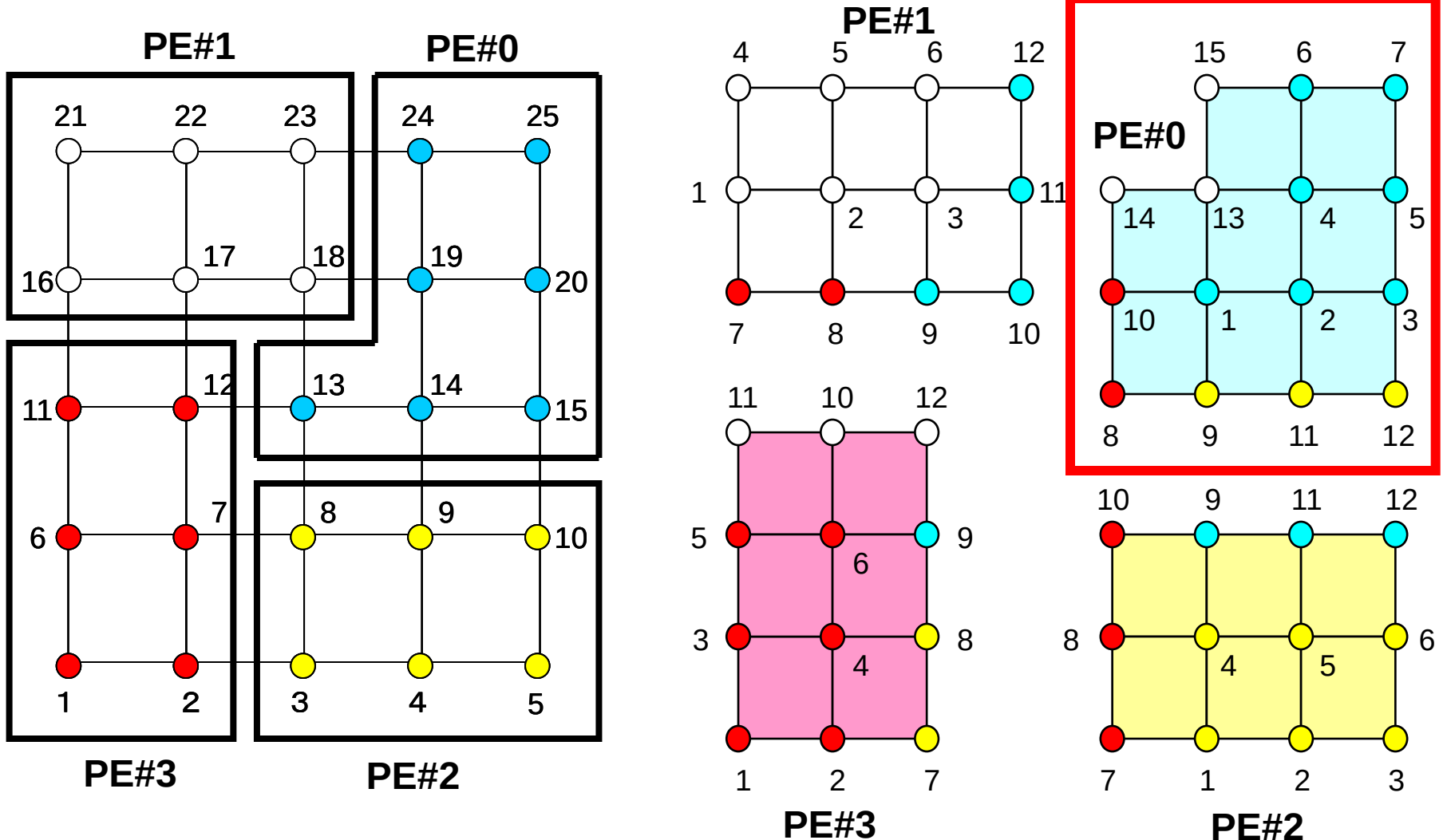
行列・ベクトル積



Local Data Structure

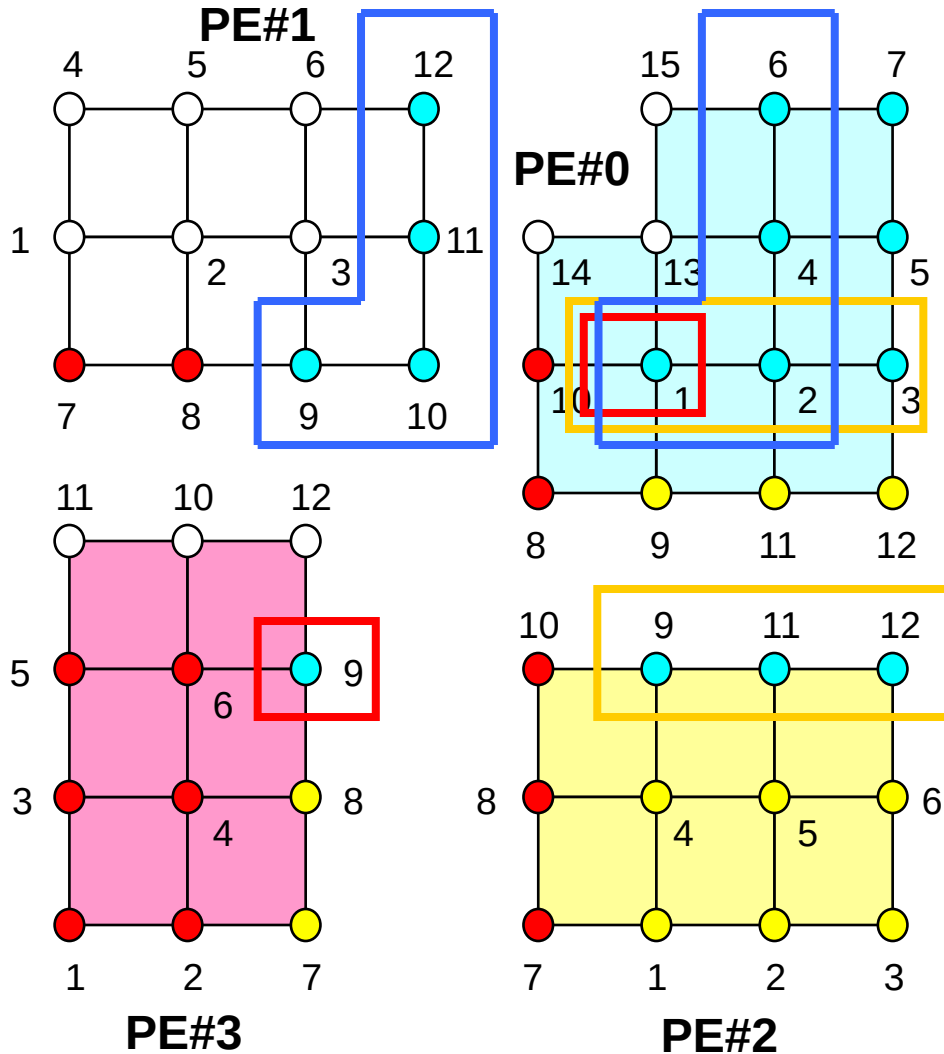
Node-based Partitioning

internal nodes - elements - external nodes



Local Data Structure : PE#0

internal nodes - elements - external nodes



Partitioned nodes themselves
internal nodes

Elements which include “internal nodes”
Provide data locality in order to carry
out element-by-element operation in
each partition

Nodes included in the elements
external nodes
Numbering : internal -> external

Internal nodes which are “external nodes”
for other partitions
boundary nodes

Communication table provides boundary~external node relationship