文部科学省次世代ＩＴ基盤構築のための研究開発
「イノベーション基盤シミュレーションソフトウェアの研究開発」

CISS フリーソフトウェア

HEC-MW

Ver. 3.0

理論マニュアル

お問い合わせ先
（契約窓口）　（財）生産技術研究奨励会
〒153-8505　東京都目黒区駒場４－６－１
（ソフトウェア管理元）　東京大学生産技術研究所　革新的シミュレーション研究センター
〒153-8505　東京都目黒区駒場４－６－１
Fax：03-5452-6662

目次

1. 線形ソルバー機能

1.1 ソフトウェアの概要

「HEC ミドルウェア（HEC-MW)」プロジェクトにおいては、有限要素法（FEM)、有限体積法（FVM）など、非構造格子を使用した科学技術シミュレーションにおける計算手法の計算処理パターンは：

- データ入出力
- マトリクス生成
- 線形ソルバー
- 領域間通信

などのいくつかの典型的なプロセスから構成されている。

本マニュアルは PC クラスタ向け HEC-MW ライブラリの一部である反復法による線形ソルバーに関する説明書である。
HEC-MW では、反復法としては：

- Conjugate Gradient （CG）
- Bi-Conjugate Gradient Stabilized （BiCGSTAB）
- Generalized Minimal Residual （GMRES）
- Generalized Product-type Bi-Conjugate Gradient（GPBiCG）

を利用できる。1 節点に 1 自由度のスカラソルバーのほか、1 節点に多自由度が存在する場合に関するブロックタイプのソルバーも用意する。ブロック内の自由度が変化する場合、1 つのプログラムの中でブロックサイズが変化する場合に関しても対応する。

前処理手法としては、以下の手法に対応する。

- 局所 ILU(k)／IC(k)法
- Symmetric Successive Overrelaxation (SSOR)
- Block Scaling，Point Jacobi
- Multigrid

1.2　反復法を使用した有限要素法の並列化

有限要素法（Finite Element Method, FEM）における典型的な処理プロセスは以下のとおりである（図 1.2.1）:

プリ･プロセッシング（例：メッシュ生成）
シミュレーション本体（例：構造解析，熱流体解析）
ポスト・プロセッシング（例：可視化，データマイニング）

並列計算で扱うデータのサイズ（メッシュ数）は非常に大きいため、全体領域を一括して取り扱うことは困難で、全体データを部分領域（局所データ）に分割する必要がある。それぞれの領域（domain，partition）は並列計算機の各プロセッサ（Processing Element，PE）に割り当てられる（図 1.2.2)。有限要素法は差分法などと比較して並列化が困難であると考えられてきた。間接データ参照があるため、1 プロセッサ（Processing Element，PE）あたりの計算効率は差分法と比較して低いが、有限要素法の処理は基本的に要素単位の局所的な処理が中心となるため並列化には適している。

FEM の処理は、線形、非線形いずれの場合も、支配方程式を線形化し、要素単位で重みつき残差法を適用して得られる要素剛性マトリクスを足し合わせて得られる、疎な全体剛性マトリクスを係数とする大規模連立一次方程式を解くことに帰着される。FEM の計算のほとんどの部分は：

- 係数マトリクス生成
- 線形ソルバーによる大規模連立一次方程式求解

に費やされる。係数マトリクス生成に関しては、要素単位に実行が可能なため、並列化は容易であり、領域間の通信無しに実行することが可能である。並列 FEM の計算において、通信が発生する可能性があるのは線形ソルバーの部分のみである。したがって、線形ソルバーの部分を除くと 1CPU の PC 向けに開発された FEM コードは並列計算機上で容易に動かすことが可能である。

領域間の通信は図 1.2.2 に示すように線形ソルバーの部分だけで生じる。この特性を最大限利用し、適切なデータ構造を設定、並列計算に適した反復法を採用することによって後述するように 95%を越えるような並列化効率を達成することも可能である。

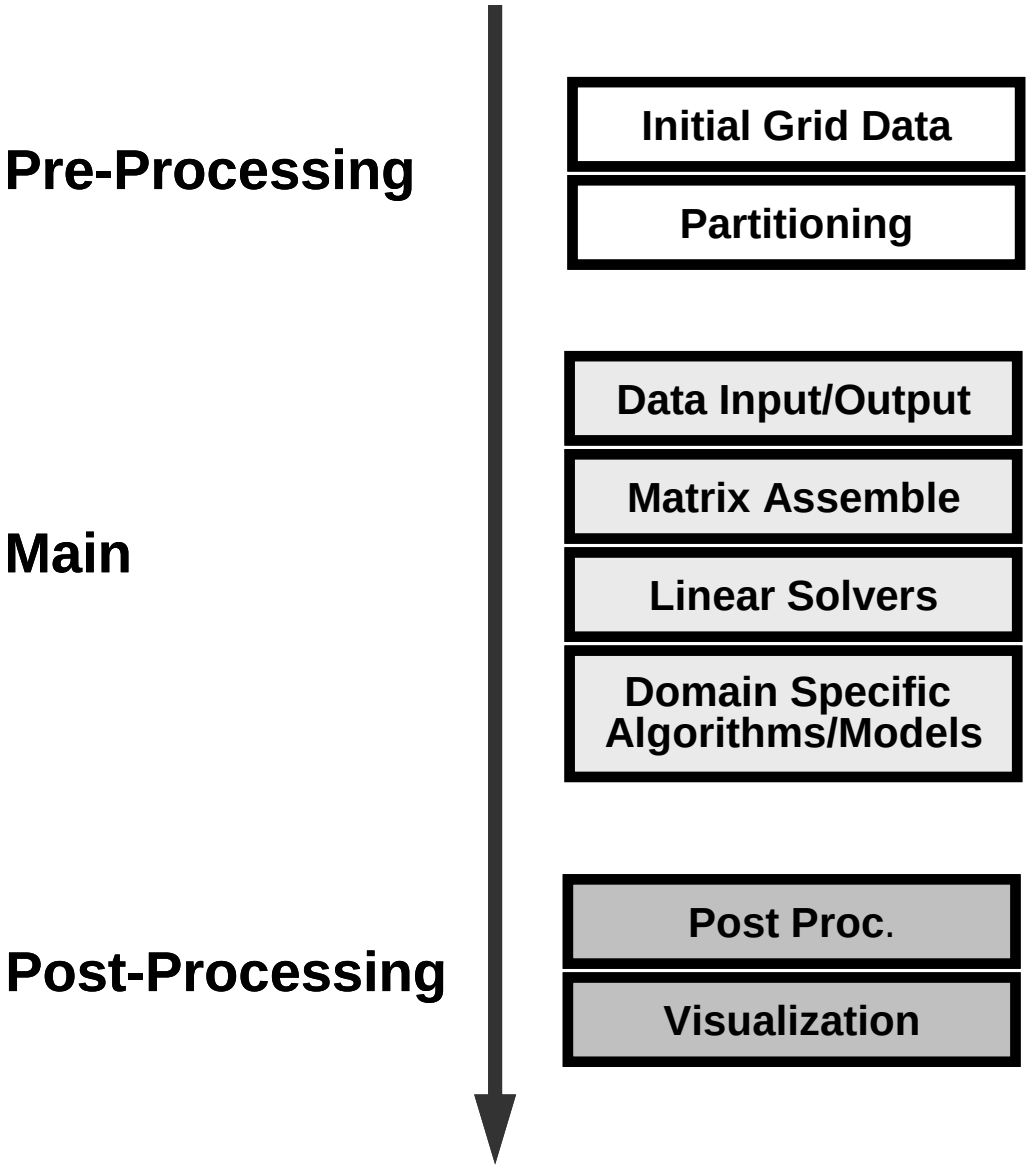


図 1.2.1 Parallel FEM Procedure

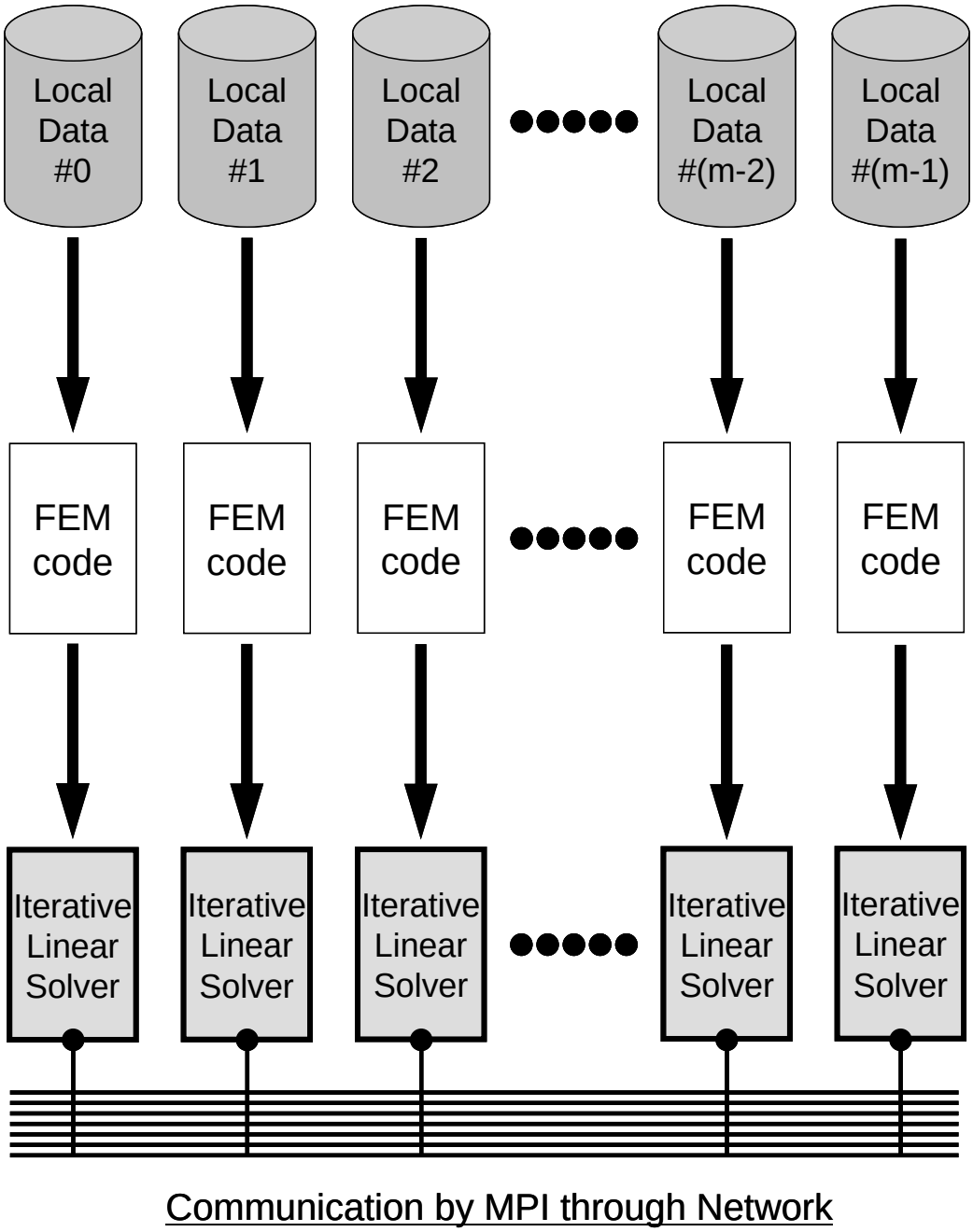


図 1.2.2　Parallel FEM procedure and distributed local data sets in HEC-MW〔5,6,7〕

1.3 分散データ構造

FEM に代表される非構造格子（Unstructured Mesh）を使用したアプリケーションにおいて、適切な分散データ構造を決定することは、並列計算を効率的に実施する上で重要である。HEC-MW の局所メッシュデータは節点ベース（node-based）および要素ベース（element-based）の領域分割の両者をサポートしているが、ここでは、領域間オーバーラップ要素を含む節点ベース領域分割を前提とする。

FEM において、速度、温度、変位など、線形方程式の解となるような変数は節点において定義される。従って、並列計算における効率という観点からは領域間の節点数が均等であることが望ましい。節点ベースの領域分割を使用した場合、剛性マトリクス生成に代表されるような要素単位の処理を各領域において局所的に実施するためには、領域間のオーバーラップ要素が必要である。図 1.3.1 はこのようなオーバーラップ要素の例を示したものである。ここで、灰色に塗られた要素は複数の領域によって共有されており、各領域における各節点に対する剛性マトリクスの足し込みなどの処理を、並列に実施するためにはこれらオーバーラップ要素の情報が必要である。

HEC-MW では領域間の通信の記述には MPI〔15〕を使用している。差分法などに使用されている構造格子（Structured Grids）に関しては MPI 固有の領域間通信用のサブルーチンが準備されているが、有限要素法に代表される非構造格子（Unstructured Grids）では、プログラム開発者が独自にデータ構造と領域間通信を設計しなくてはならない。

HEC-MW において、各領域は以下の情報を含んでいる：

- 各領域に割り当てられた節点
- 各領域に割り当てられた節点を含む要素
- 他の領域に割り当てられているが上記の要素に含まれている節点
- 領域間の通信テーブル（送信，受信用）
- 節点グループ，要素グループ，面グループ
- 材料物性

節点は、通信という観点から以下の 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）：他領域の外点となっている節点

図 1.3.2 は、局所データの実例である。図 1.3.2 と図 1.3.3 における PE#2 において、節点は以下のように分類される：

- 内点　　{1,2,3,4,5,6}
- 外点　　{7,8,9,10,11,12}
- 境界点　{1,2,5,6}

局所データには領域間の通信テーブルの情報も含まれる。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。図 1.3.2 に示す局所データ構造と図 1.3.3 に示す領域間通信によって非常に高い並列性能が達成されている 〔3,5,6,7,8,9〕。

HEC-MW では全体データから局所データを自動的に生成するためのツールとして領域分割用ユーティリティが用意されている。利用者は実際には上記の通信テーブルについては陽に意識することなく並列有限要素法コードの開発、利用が可能である。領域分割にあたっては：

- 各領域の負荷が均等となっていること
- 領域間の通信が少ないこと

が重要である。特に前処理つき反復法を使用する場合には収束を速めるために（2）が重要なポイントである〔7〕。この両条件を満たす手法としては METIS〔14〕が良く知られている。HEC-MW の領域分割ツールでは文献〔12〕で紹介されている RCB 法（Recursive Coordinate Bisection）等のほか、METIS による領域分割も利用できる。

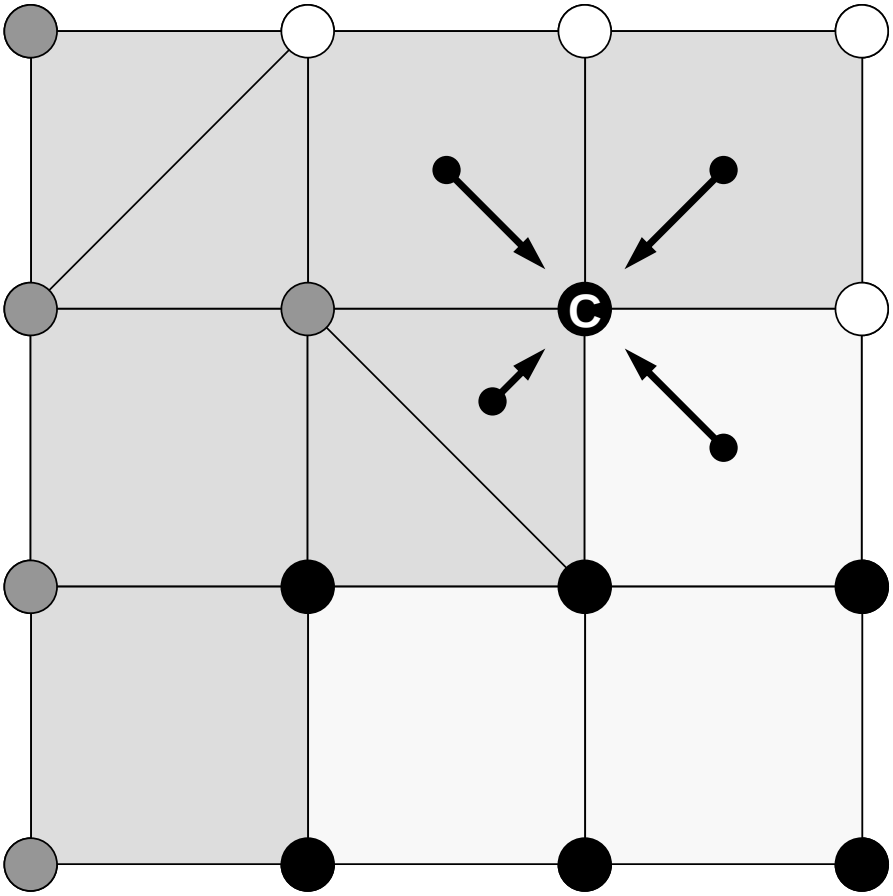


図 1.3.1 Element-by-element operations around node C.
Gray meshes are overlapped among domains.

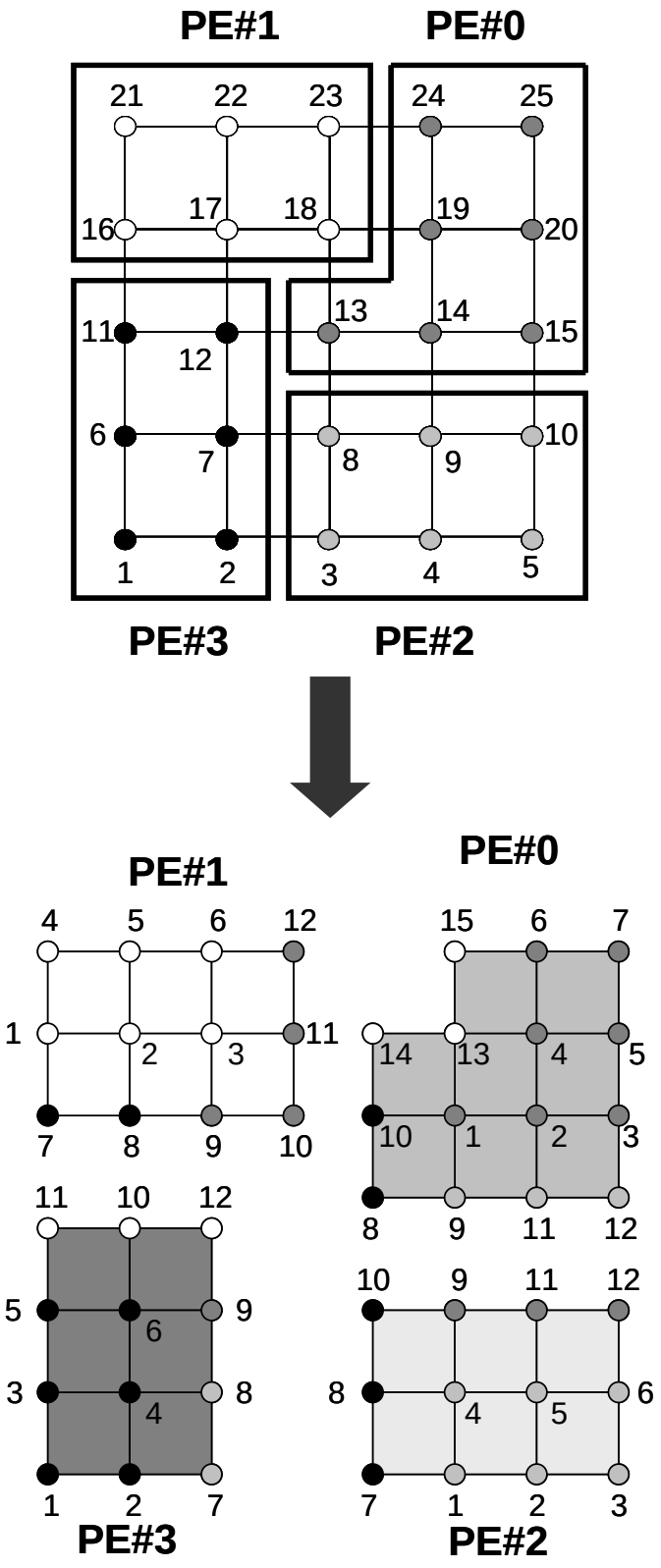
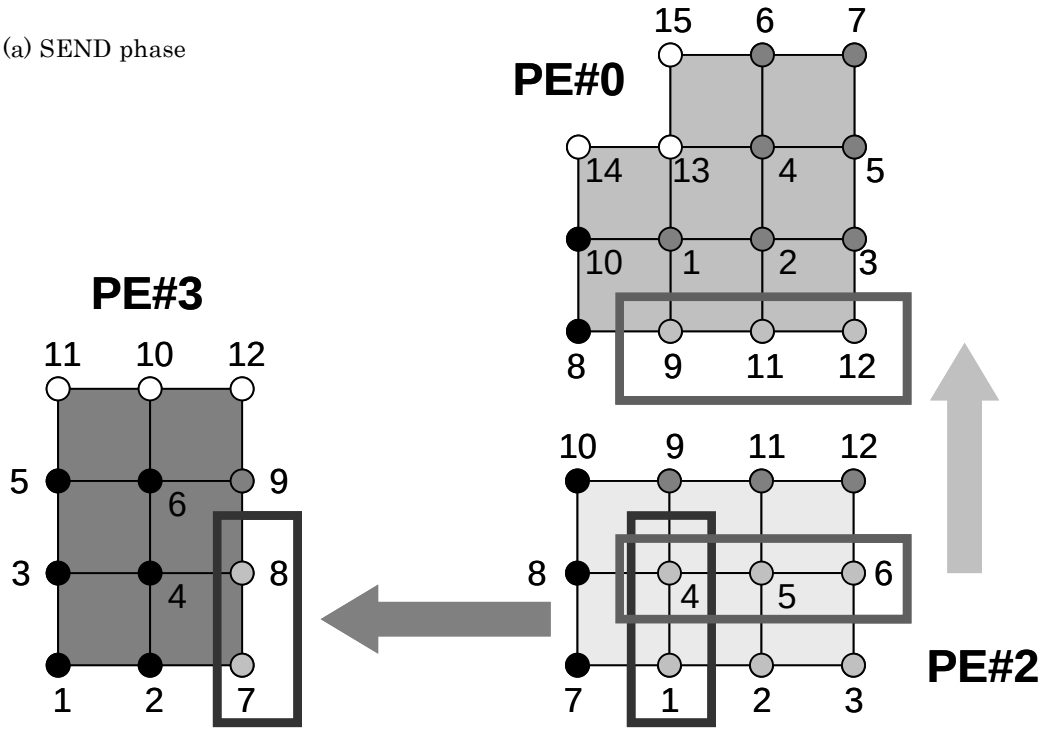


図 1.3.2　Node-based partitioning into 4 PEs 〔5,6,7〕

(a) SEND phase



(b) RECEIVE phase

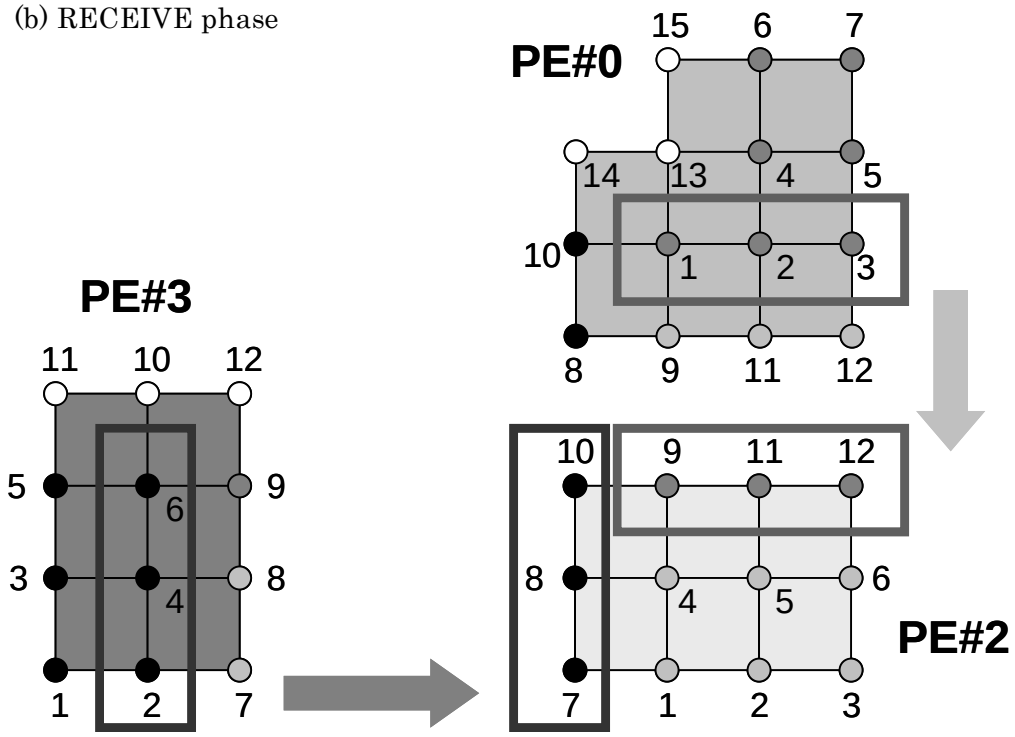


図 1.3.3 Communication among processors 〔5,6,7〕

1.4　前処理付反復法

連立一次方程式の解法としてはガウスの消去法などの直接法（Direct Method）〔2〕が広く使用されてきたが、大域的（Global）処理が生ずるため、並列計算には適していない。並列計算に適した手法として共役勾配法（Conjugate Gradient Method，CG）〔2〕などの Krylov 型反復法（Krylov Iterative Method）が注目されている。

反復法の収束特性は係数行列の固有値分布に依存するため、実用的な問題に適用するためには前処理（Preconditioning）を施し、固有値分布を変えた行列を解く手法が一般的である。反復法の前処理手法としては不完全 LU 分解（Incomplete LU Factorization，ILU）あるいは不完全コレスキー分解（Incomplete Cholesky Factorization，IC）などがよく使用される〔2〕。

前処理つき反復法における計算プロセスは以下の 4 種類に分類される：

（1） 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）
（4） 前処理

図 1.4.2 は前処理付 CG 法 〔1,2〕の処理手順を示したものである。これによると、1 回の反復で以下に示す回数だけ各プロセスが発生する：

行列ベクトル積　　1
ベクトル～ベクトル内積　　2
ベクトル（およびその実数倍）の加減（DAXPY）　　3
前処理　　1

このうち（3）を除く各プロセスでは領域間の通信が発生する。（1）は計算前に 2．で述べた通信を実施すれば局所的な処理が可能である。（2）は「MPI_ALLREDUCE」などのサブルーチンを使用して容易に達成可能である。図 1.4.3 は（1）～（3）の各プロセスについて MPI を使用して FORTRAN で記述した場合のプログラム例である。

（4）については前処理手法によって異なる。例えば ILU などの手法は前進／後退代入（Foward Backward Substitution，FBS）により大域的な変数の依存性が生じるため、並列化が困難である。単独プロセッサを使用した計算の場合、Fill-in のない ILU（0）法を前処理として使用すると、前処理計算部分が全体の 50%程度を占めるため〔7〕、前進／後退代入部分の並列化は計算効率の向上のために不可欠である。前進／後退代入は以下のような処理である。

前進代入　　$y_k = b_k - \sum_{j=1}^{k-1} L_{kj}\, y_j \quad (k=2,\cdots N)$

後退代入　　$x_k = \tilde{D}_k \left(b_k - \sum_{j=k+1}^{N} U_{kj} y_j \right)$ $(k=N,N-1,\cdots 1)$

HEC-MW ではブロックヤコビ法に基づく、局所前処理法（局所 ILU（0）法, Localized ILU（0））〔7〕を使用している。局所 ILU（0）法は一種の「擬似」ILU（0）法である。局所 ILU（0）法では前進／後退代入計算時に領域外からの影響（すなわち外点の影響）を 0 とすることによって、前処理の局所化を行い、並列性の高いアルゴリズムを実現している（図 1.4.4 参照）。

この処理は、各プロセッサにおいて、境界＝0 のディリクレ型境界条件のもとで前処理を実施することと同等であり、完全に並列な前処理を実施することが可能となる。この考え方は不完全ヤコビ前処理〔2,12〕に基づくものである。図 1.4.5 は局所 ILU（0）法を使用した場合の CG 法の計算アルゴリズムである。1 回の反復計算あたりで領域間通信が生じるのは 3 回であり、そのうち 2 つは内積計算のためにスカラを BROADCAST するだけであり、残りの 1 回が通信テーブルを使用したオー場ラップ領域情報の通信である。

局所 ILU（0）法は並列性能には優れていると考えられるが、グローバルな処理によっている本来の ILU（0）法と比較すると前処理の機能は強力ではない。一般的に局所 ILU（0）型の前処理では、領域数が増加するほど収束が悪くなる〔3,5,6,7〕。領域数と自由度数が同じになると、対角スケーリングと同じになってしまう。表 1.4.1 は 3×44^3 自由度の一様な立方体領域における三次元線形弾性解析を局所 IC（0）前処理つきの CG 法で計算した例である。計算には東京大学情報基盤センターの Hitachi SR2201 を使用した。領域数の増加とともに収束までの反復回数は増加しているが、1PE から 32PE まで増えた場合でも 30%程度の増加である。

同様に Hitachi SR2201 を使用して、並列性能の評価を実施した。図 1.4.6～図 1.4.8 は一様な立方体領域における三次元線形弾性解析を様々な問題規模において解いた場合の並列性能（work ratio＝計算時間／合計実行時間）である〔8,9〕。これらのケースでは 1PE あたりの問題規模は固定されている。最大規模の問題では 1024 PE を使用して 196,608,000 自由度の問題を解いている。図 1.4.6～図 1.4.8 によると 1PE あたりの問題規模が充分大きければ、並列性能は 95%以上である。

文献〔10〕などに見られるように、適切なオーダリング、領域分割を実施することによって、並列計算の場合の ILU／IC 処理においても完全にグローバルな処理を達成することは可能である。しかしながらこれが可能となるのはあらかじめ全体マトリクスが得られているときのみである。並列有限要素法においては、局所領域ごとに係数マトリクスが生成されるため、このような手法は本研究においては採用していない。文献〔4〕においては深さ k の Fill-in レベルをもつ ILU（k）法のグローバル並列化に関して検討されている。非常に安定な収束を示しているが、並列性能は低い。

(a) Calling interface for communication among domains (1x1 scalar and 3x3 block)

1x1 Scalar

```
      allocate (WS(NP), WR(NP), X(NP)
      call SOLVER_SEND_RECV                                          &
     &    ( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_NODE,       &
     &      EXPORT_INDEX, EXPORT_NODE, WS, WR, X , SOLVER_COMM,     &
     &      my_rank)
```

3x3 Block

```
      allocate (WS(3*NP), WR(3*NP), X(3*NP)
      call SOLVER_SEND_RECV_3                                        &
     &    ( NP, NEIBPETOT, NEIBPE, IMPORT_INDEX, IMPORT_NODE,       &
     &      EXPORT_INDEX, EXPORT_NODE, WS, WR, X , SOLVER_COMM,     &
     &      my_rank)
```

(b) Subroutines for communication among domains

- SEND phase

```
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
```

- RECEIVE phase

```
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, sta1, ierr)
```

図 1.4.1 Communication procedures among domains in HEC-MW 〔5,6,7,8,9〕

```
compute r(0)= b - Ax(0) for some initial guess x(0)
for i= 1,2,...
   solve M z(i-1)= r(i-1) (M: preconditioning matrix)      Preconditioning
   ρi-1= r(i-1)T z(i-1)                                    Dot Product (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)                            DAXPY (1)
   endif
   q(i)= A p(i)                                            MATVEC
   αi= ρi-1/(p(i)T q(i))                                   Dot Product (2)
   x(i)= x(i-1) + αi p(i)                                  DAXPY (2)
   r(i)= r(i-1) - αi q(i)                                  DAXPY (3)
   check convergence; continue if necessary
end
```

図 1.4.2　Procedures in CG iterative method 〔1,2〕

(a) Matrix-vector products

```
 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
```

(b) Inner dot products

```
 RHO0= 0.0
 do i= 1, N
   RHO0= RHO0 + WW(i,R)*WW(i,Z)
 enddo

 call MPI_allREDUCE (RHO0, RHO, 1, MPI_DOUBLE_PRECISION, &
&                    MPI_SUM, SOLVER_COMM, ierr)
```

(c) DAXPY

```
 do i= 1, N
   X (i)  = X (i)   + ALPHA * WW(i,P)
   WW(i,R)= WW(i,R) - ALPHA * WW(i,Q)
 enddo
```

図 1.4.3 Parallelization of typical processes in iterative solvers in FORTRAN with MPI 〔5,6,7,8,9〕

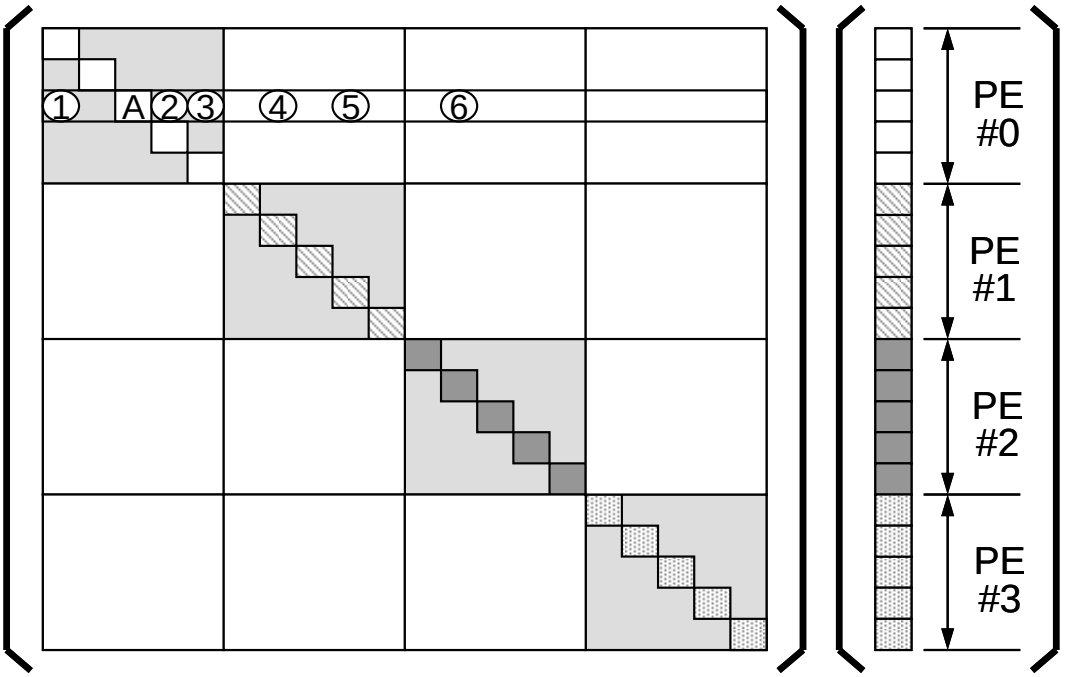


図 1.4.4　Localized ILU(0) Operation: Matrix components whose column numbers are outside the processor are ignored (set equal to 0) at localized ILU(0) factorization. For example the element A on PE#0 has 6 non-zero components but only 1,2,3 are considered and 4,5,6 are ignored and set to 0 〔5,6,7,8,9〕

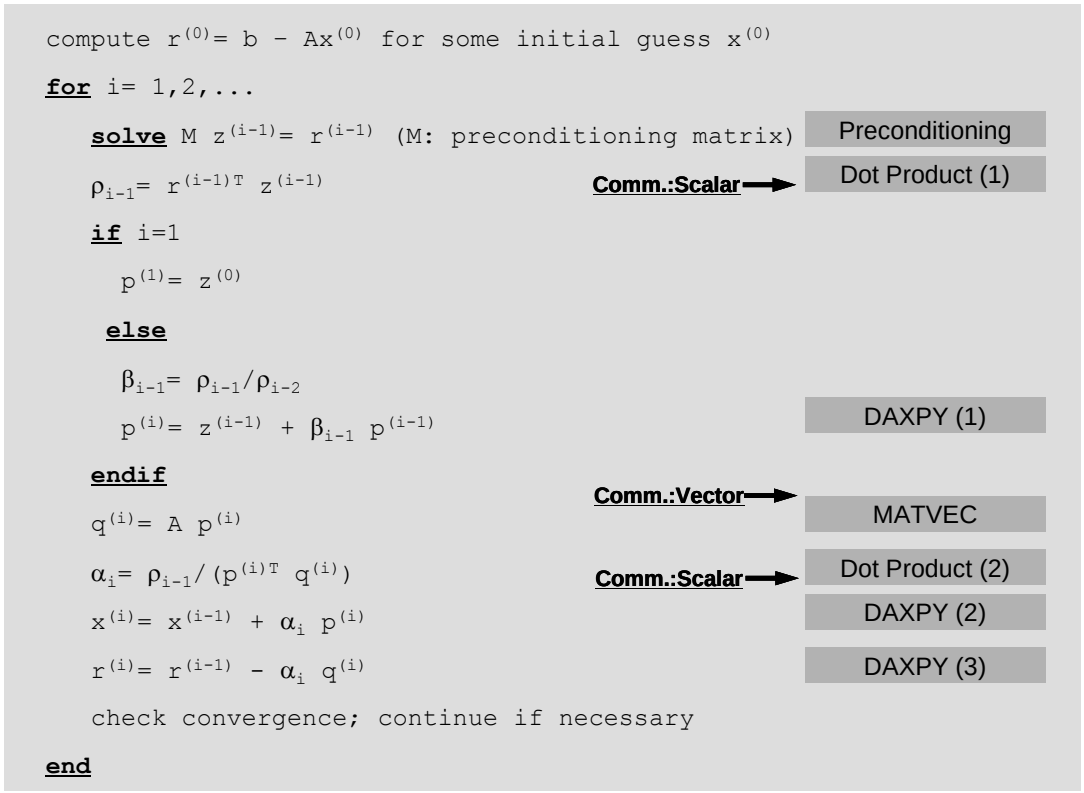


図 1.4.5 Parallel CG iterative method by localized preconditioning in HEC-MW

表 1.4.1 Homogeneous solid mechanics example with 3×44^3 DOF on Hitachi SR2201 solved by CG method with localized IC(0) preconditioning (Convergence Criteria $\varepsilon=10^{-8}$).

PE #	Iter. #	sec.	Speed Up
1	204	233.7	-
2	253	143.6	1.63
4	259	74.3	3.15
8	264	36.8	6.36
16	262	17.4	13.52
32	268	9.6	24.24
64	274	6.6	35.68

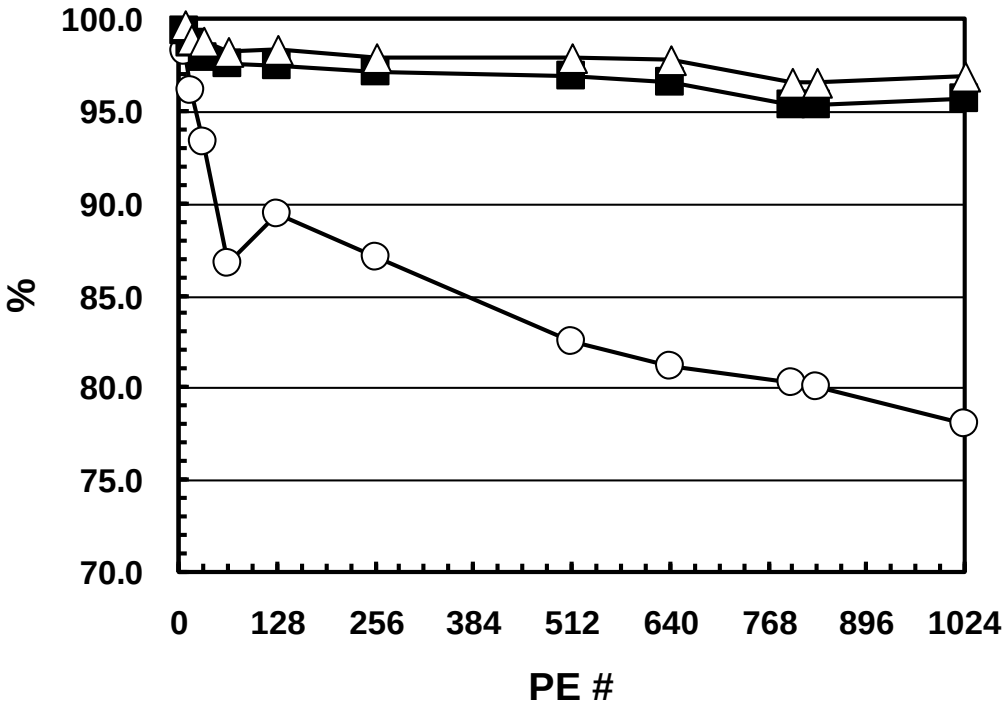
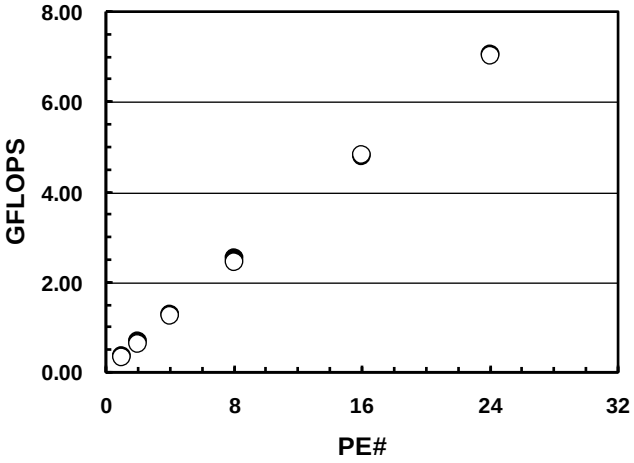
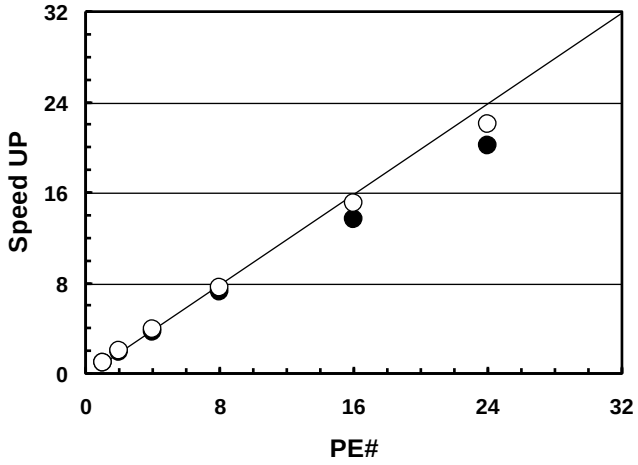


図 1.4.6 Parallel performance for various problem sizes for simple 3D elastic solid mechanics on Hitachi SR2201. Problem size/PE is fixed. Largest case is 196,608,000 DOF on 1024 PEs. (Circles: 3×16^3 (= 12,288) DOF/PE,
Squares: 3×32^3 (= 98,304), Triangles: 3×40^3 (= 192,000)).

(a) GFLOPS



(b) Parallel Speed-UP



(c) Parallel Work Ratio

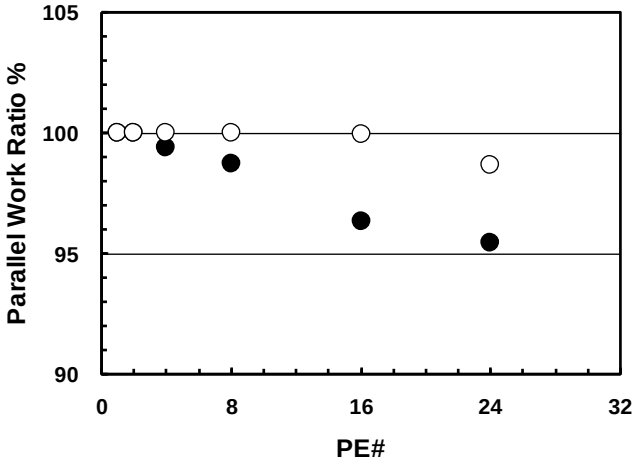
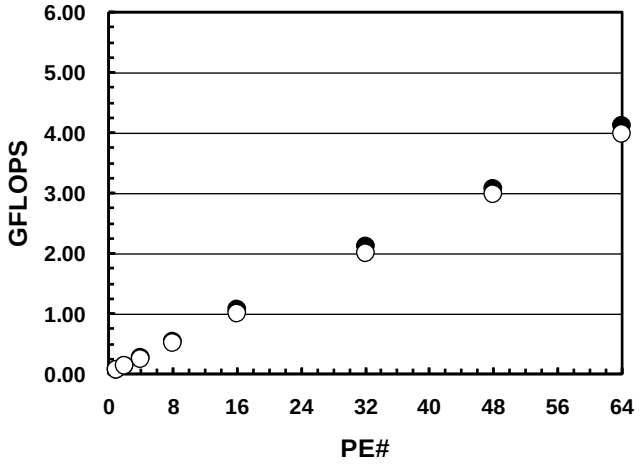
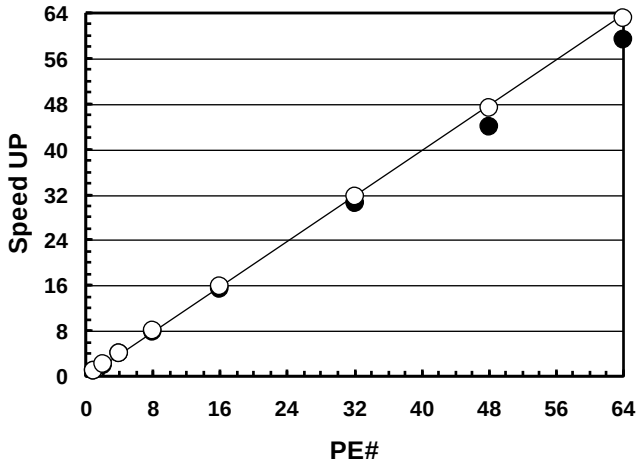


図 1.4.7 Parallel performance for various problem sizes for simple 3D elastic solid mechanics on Xeon 2.8GHz Cluster. Problem size/PE is fixed. Largest case is 2,359,296 DOF on 24 PEs. (Black Circles: 3×16^3 (= 12,288) DOF/PE, White Circles: 3×32^3 (= 98,304)).

(a) GFLOPS



(b) Parallel Speed-UP



(c) Parallel Work Ratio

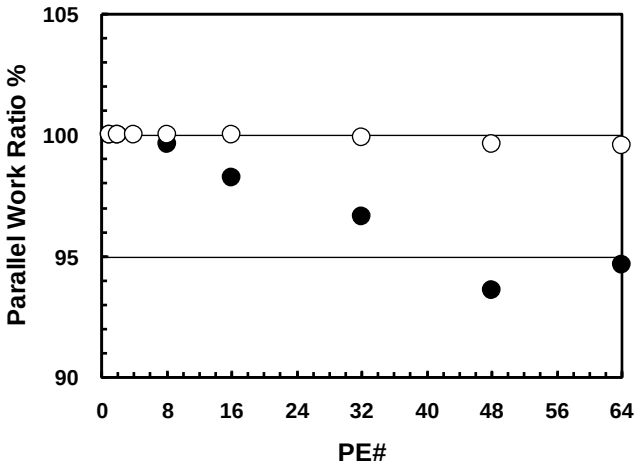


図 1.4.8 Parallel performance for various problem sizes for simple 3D elastic solid mechanics on Hitachi SR2201. Problem size/PE is fixed. Largest case is 6,291,456 DOF on 64 PEs. (Black Circles: 3×16^3 (= 12,288) DOF/PE, White Circles: 3×32^3 (= 98,304)).

1.5 Additive Schwartz Domain Decomposition

局所 ILU(0)前処理法を安定化させる手法として、領域間オーバーラップ領域に加法型 Schwartz 領域分割法（Additive Schwartz Domain Decomposition，ASDD）〔12〕を適用した。ASDD の手順は以下のとおりである：

以下の前処理を実施する $Mz = r$ ここで M：前処理行列， r,z：ベクトル、である。

全体領域が図 1.5-14（a）に示すように Ω_1 および Ω_2 の 2 領域に分かれているものと仮定すると前処理は各領域において、以下のように局所的に実施される。

$$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$$

局所的な前処理を実行したのちに、領域間オーバーラップ領域 ▪ 1 および ▪ 2 において以下の計算を実施する（図 1.5.1（b））：

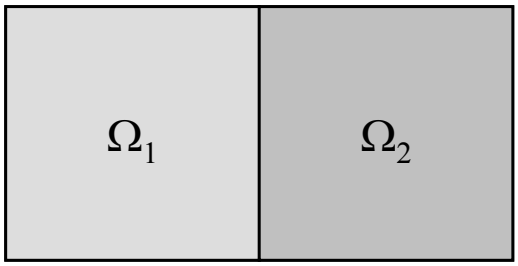
$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1}) \quad z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}(r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1})$$

ここで n は ADSS の繰り返し数である。

（2）および（3）のプロセスを収束まで繰り返す。

表 1.5.1 は ASDD の効果を 0 で紹介した 3×44^3 自由度の一様な立方体領域における三次元線形弾性解析に適用した例である。計算には東京大学情報基盤センターの Hitachi SR2201 を使用した。すでに示したように、ASDD 無しの場合は領域数とともに反復回数は増加するが、ASDD の導入により、1 回の反復計算あたりの計算量は増加するものの、領域数増加にともなう反復回数の増加はわずかに抑制される。

(a) Local operation



(b) Blobal nesting correction

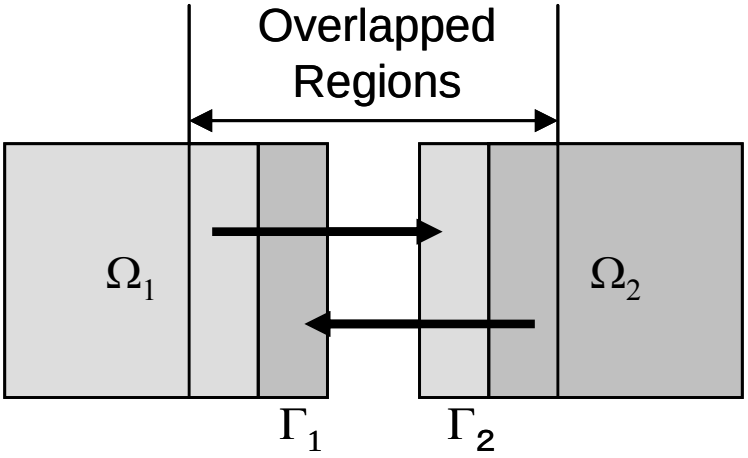


図 1.5.1　Operations in ASDD for 2 domains 〔12〕

表 1.5.1 Effect of ASDD for solid mechanics with 3×44^3 DOF on a Hitachi SR2201.

	NO Additive Schwarz			WITH Additive		
PE #	Iter. #	Sec.	Speedup	Iter.#	Sec.	Speedup
1	204	233.7	-	144	325.6	-
2	253	143.6	1.63	144	163.1	1.99
4	259	74.3	3.15	145	82.4	3.95
8	264	36.8	6.36	146	39.7	8.21
16	262	17.4	13.52	144	18.7	17.33
32	268	9.6	24.24	147	10.2	31.80
64	274	6.6	35.68	150	6.5	50.07

Number of ASDD cycle/iteration = 1, Convergence Criteria $\varepsilon=10^{-8}$

1.6　悪条件問題に関する前処理手法

1.6.1 各前処理手法について

1.6.1.1　概 要

接触条件に基づく境界非線形性を有するアプリケーションは、工学、自然科学の広い分野にわたって非常に重要である。有限要素法で接触問題を解く場合には、拡大ラグランジェ法（Augmented Lagrange Method，ALM）とペナルティ法が使用されることが多く、接触による拘束条件は大きなペナルティ数λを導入することによって表される〔9〕。非線形過程は Newton-Raphson 法（NR 法）によって線形化され、反復的に解かれている。

一般に、λの値が大きいほど、精度よく接触条件を模擬することができ、Newton-Raphson 法の非線形計算に要する反復回数は減少するが、係数行列の条件数が大きくなる。したがって、反復法ソルバーの収束のためには多くの反復を必要とする。条件数の大きい悪条件問題を解く場合には安定した信頼性の高い前処理手法が不可欠である。

ILU/IC 系の前処理手法を使用する場合、このような悪条件問題に関する対処法としては以下のようなものがあげられる：

- ブロッキング
- 深い Fill-in レベル
- オーダリング

1.6.1.2　ブロッキング

三次元固体力学の問題においては、各節点あたり、3 方向の変位成分を自由度としてもっている。そこで、図 1.6.1 に示すように 3×3 ブロックに関する完全 LU 分解を、対角ブロックに導入する。これが、ILU/IC 系前処理のブロック化バージョンである、ブロック ILU/IC（BILU/BIC）前処理である。このような処理を導入することによって、各節点上の三自由度は同時に計算されるため、各自由度を独立に計算する場合と比較してより効果的な前処理が可能となる。

1.6.1.3　深い Fill-in レベル

続いて、BILU/BIC 前処理において深いレベルの Fill-in を考慮する BILU（n）/BIC（n）前処理について検討する（「n」は Fill-in レベル）。直接法で使用される完全 LU 分解（あるいはガウスの消去法）のアルゴリズムは以下のとおりである：

```
Gaussian Elimination
  do i= 2, n
    do k= 1,   i-1
      ajk := ajk/akk
      do j= k+1, n
        aij := aij - aik*akj
      enddo
    enddo
  enddo
```

完全 LU 分解では、分解のたびに多量の Fill-in が発生するため、もとの行列が疎でも完全 LU 分解によって生成される逆行列は密行列となる可能性がある。ILU（n）または IC（n）は「n」レベル分の Fill-in を許容する手法である。「n」の値が大きいほど、分解の精度は高くなり、より安定した前処理行列が得られるが、計算量、メモリのコストは高くなる。一般工学分野においては、Fill-in を許さず、オリジナル行列と同じ非ゼロ成分パターンを保持する以下のに示す ILU（0）/IC（0）前処理が、効率の観点から広く使用されている。

```
ILU(0)
  do i= 2, n
    do k= 1,   i-1
      if ((i,k)  ∈  NonZero(A)) then
        ajk := ajk/akk
      endif
      do j= k+1, n
        if ((i,j)  ∈  NonZero(A)) then
          aij := aij - aik*akj
        endif
      enddo
    enddo
  enddo
```

一方「n」レベルの Fill-in を考慮する ILU（n）/IC（n）法のアルゴリズムは以下のとおりである：

```
ILU(n)
  LEV_ij=0 if ((i,j) ∈ NonZero(A)) otherwise LEV_ij= p+1
  do i= 2, n
    do k= 1,  i-1
      if (LEV_ik ≤ p) then
        a_jk := a_jk/a_kk
      endif
      do j= k+1, n
        if (LEV_ij = min(LEV_ij,1+LEV_ik+ LEV_kj) ≤ p) then
          a_ij := a_ij - a_ik*a_kj
        endif
      enddo
    enddo
  enddo
```

1.6.1.4 Selective Blocking

Fill-in レベルを深くとる手法に加えて、接触問題向けに「Selective Blocking」法を開発した〔9〕。「Selective Blocking」法は、接触要素によって決定され、ペナルティ数により強くカップルした「接触節点グループ」〔9〕に属する節点が、連続する節点番号をもって同じグループ（「Selective Block」または「Super Node」）になるようにオーダリングし、この「Selective Block」内に完全 LU 分解を適用する手法である。三次元問題の場合、ある「Selective Block」に含まれる節点数を NB とすると、(3×NB) × (3×NB) ブロックに関する完全 LU 分解が必要となる（図 1.6.2 および図 1.6.3 参照)。つまり、前処理計算のプロセスにおいて、接触節点グループに属する節点群に関しては直接法的な処理が適用される。

この「Selective Block」の考え方は、「Clustered Element-by-Element（CEBE)」法あるいはブロック ICT 法と類似したものである。これらの手法は、全体領域を複数の節点のクラスタ（Cluster）に分割し、各クラスタ内に直接法的な処理を導入するものである。クラスタの大きさは任意に選ぶことが可能であるが、最適な計算効率を得るためのパラメータとして考えることができる。CEBE 法はクラスタの大きさが節点数と同じになった場合に直接法と同等と見なすことができる。一般的にクラスタの大きさが大きいほど安定した収束が得られるが、図 1.6.4 に示すように各反復における計算コストは増大する。収束と計算コストのトレードオフは必ずしも明確ではないが、クラスタの大きさが大きいほど概して効率的である。

「Selective Blocking」では、CEBE 法などにおけるクラスタは接触節点グループの情報によって決定される。個々のクラスタの大きさは CEBE 法と比較すると一般的に小さい。例えば、接触節点グループに属さない節点の場合、その節点単独で「大きさ＝1」のクラスタを形成することになる。「Selective Blocking」法におけるサイクルあたりの計算コストは、クラスタ間の Fill-in を考慮しない場合は、BILU（0）/BIC（0）とほぼ同等である。

1.6.1.5 各手法の評価

表 1.6.1 は、図 1.6.6 に示す形状の問題について、ブロック間の接触面の各節点について MPC（Multi-Point Constraint）条件を課した、弾性静解析についての計算結果である。3×3 ブロック処理を導入することにより、Fill-in を考慮しない IC 法前処理（BIC（0））にておいても$\lambda=10^6$の場合に計算を実施することが可能となった。Fill-in を深くすることにより、さらに効率的に解を得ることが可能であるが、SB-BIC（0）前処理（BIC（0）前処理と「Selective Blocking」オーダリングを組み合わせた手法）によるものが最も効率が良いことがわかった（表 1.6.1）。

SB-BIC（0）は収束までの反復回数は BIC（1）や BIC（2）（Fill-in レベル＝1 または 2 のブロック IC 前処理）と比較して多いが、1 回の反復あたりの計算コストが低いため、計算時間が少なくなる。SB-BIC（0）では、「Selective Block」間の Fill-in は考慮されておらず、「Selective Block」内の Fill-in のみ考慮されているため、計算コスト、必要メモリ量は既に述べたように BILU（0）/BIC（0）とほぼ同等である。

「Selective Blocking」を考慮した前処理による Krylov 反復解法は反復法と直接法のハイブリッド解法と見なすことができる。すなわち、接触節点グループに属する節点に関しては、前処理においては直接法的な処理が適用されている。この手法は反復法の効率、スケーラビリティと直接法の安定性の両方を備えた解法と考えることができる。

1.6.2 並列計算手法

局所 ILU/IC 前処理は効率的な並列前処理手法であるが、悪条件問題に関しては安定な手法ではない。表 1.6.2 は表 1.6.1 で示した問題によるマトリクスに関して、8 PE の PC クラスタを使用して、局所 ICCG 法によって計算を実施した例である。なお、領域分割は METIS の中の k-way METIS を使用している。計算結果によるとペナルティ数の増加とともに収束は悪化し、$\lambda=10^6$ の場合は収束までの反復回数が 10 倍のオーダーで増加している。これは同じ接触節点グループに属する節点から構成される要素（すなわち「接触要素」）が、領域間境界に存在しているため、これらの辺で「edge-cut」が生じているためであると考えられる〔9〕。

このような「edge-cut」を解消するため、同じ接触節点グループに属する節点が必ず同じ領域に属するような特殊な領域分割手法を開発した。さらに、領域間の負荷が均等となるような、負荷再配分手法も併せて開発した（図 1.6.5）。

表 1.6.2 は、この新しい領域分割法を適用して得られる計算結果である。反復回数は、いずれの前処理手法においても従来と比較して著しく減少していることがわかる。

1.6.3 ベンチマーク

1.6.3.1 概 要

接触問題用に開発された前処理手法および領域分割手法の効率と安定性について、二種類の三次元計算例を使用して検証を実施した。

図 1.6.6 は、第一の計算対象である、3 つの単純形状ブロックから構成される対象（簡易ブロックモデル）と、三次元弾性静解析のための境界条件について説明したものである。この計算例では、以下に示すように、線形多点拘束（Multiple Point Constraint，MPC）の条件が接触節点グループ内の各節点に適用されている。

- 接触節点グループを構成する各節点の座標は同一である。
- 接触節点グループを構成する各節点の変位は三方向において拘束されている。
- 固体力学における微小弾性変形理論に基づき、節点の座標は不変であり、したがって接触に関する関係も不変である。
- ペナルティ数による拘束が接触節点グループを構成する各節点に適用される。

「111」型トラス要素が図 1.6.7 に示すように各接触節点グループの節点によって形成される。このトラス要素の剛性はペナルティ数に対応する。図 1.6.8 は接触節点グループの行列処理を示す。前述のように（x,y,z）三方向の変位が拘束されている。

ペナルティ数が大きくなるほど、強い拘束条件となるが係数行列の条件数は大きくなる。したがって、ペナルティ数が大きくなると、反復法の収束は悪化する。このベンチマークで扱う問題自体は線形の弾性解析であるが、得られる連立一次方程式は非線形接触問題を解く場合と同様に、反復法で解くことは困難である。解析モデルと境界条件は以下のとおりである：

- ヤング率＝1.00，ポアソン比＝0.30 の一様物性からなる 3 つのブロックを含んでいる。一次線形アイソパラメトリック六面体要素を使用している。
- 各ブロックの境界に沿って一様な MPC 条件が適用されている。したがって、図 1.6.8 に示すように各接触節点グループに属する節点数は異なる（2 または 3）。
- x=0 および y=0 の面においては対称境界条件が適用されている。
- x=Xmax および y=Ymax の面においては拘束条件は適用されていない。
- z=0 の面においては固定境界条件が適用されている。
- z=Zmax の面においては z 方向に一様分布荷重がかけられている。
- 接触面における摩擦を考慮しない場合には、係数行列は対称正定となる。したがって共益勾配法（CG 法）の適用が可能である。

この計算例における要素の形状は全て立法体である。

図 1.6.9 に示す二番目の解析モデルは、西南日本域における地震シミュレーション〔9〕に使用

される複雑な形状をもったメッシュである。このモデルは大陸側プレート地殻（濃い灰色）と沈み込みプレート（淡い灰色）の 2 つの部分から構成されている。節点数は 27,195 であり、一次線形アイソパラメトリック六面体要素 23,831 個から構成されている。簡易ブロックモデルと同様の境界条件を適用しているが、この西南日本モデルでは、z=Zmax の面における z 方向への表面分布荷重ではなく、各要素に z 方向に大きさ-1.0 の体積分布荷重が適用されている。また、x および y 方向への対称境界条件も適用されていない。西南日本モデルでは、各要素の形状は不規則で、場所によっては非常に大きく歪んでいるものもある。物性は簡易ブロックモデルと同様一様で、ヤング率＝1.00，ポアソン比＝0.30 である。

以下、これら 2 種類の計算モデルについて、1 PE（Compaq Alpha 21164/600MHz）によって小規模問題を解いた場合、日立 SR2201 において Flat-MPI 並列プログラミングモデルを使用して並列化した場合の大規模計算例を示す。

1.6.3.2 ベンチマーク I（簡易ブロックモデル）

まず最初に図 1.6.6 に示す簡易ブロックモデルについて、ペナルティ数をパラメータとして、様々な前処理手法を適用した。計算は 1 領域について実施し、Compaq Alpha 21164/600MHz を使用した。このベンチマークでは、以下のようなモデルを使用している：

NX1＝20，NX2=20，NY=15，NZ1=20，NZ2=20（図 1.6.6（a））
要素数＝24,000，節点数＝27,888，自由度数＝83,664

表 1.6.3 は反復法の収束状況を示したものである。ペナルティ数が 10^4 より大きくなると BIC(0) は収束しなくなる。BIC（1），BIC（2）と SB-BIC（0）は広範囲のペナルティ数に対して安定である。SB-BIC（0）は BIC（1），BIC（2）と比較して収束までの反復回数は多いが計算時間は短く、最も効率的である。

続いて、前処理手法の安定性を$[M]^{-1}[A]$の固有値分布に基づき文献〔1,3〕に示された手法によって推測した。ここで[A]はもとの係数行列であり、$[M]^{-1}$ は前処理行列の逆行列である。

対称正定行列において、条件数κは

$$\kappa = E_{max} / E_{min}$$

によって得られる。ここで E_{max}，E_{min} は$[M]^{-1}[A]$の最大および最小固有値である。

表 1.6.4 は各前処理手法において、様々なペナルティ数に関して得られる E_{max}，E_{min} および κ の値である。表 1.6.3～表 1.6.4 によると、BIC（0）以外の前処理手法では、広範囲のペナルティ数において全ての固有値はほぼ同じ値であり、1.00 に近い。BIC（1）と BIC（2）からは SB-BIC（0）と比較して若干小さい κ が得られる。

1.6.3.3 ベンチマーク II（西南日本モデル）

図 1.6.9 に示すように複雑な形状をもった西南日本モデルについて、ペナルティ数をパラメータとして、様々な前処理手法を適用した。計算は 1 領域について実施し、Compaq Alpha 21164/600MHz を使用した。このベンチマークでは、以下のようなモデルを使用している：

表 1.6.5 は反復法の収束状況を示したものである。ペナルティ数が 10^4 より大きくなると BIC(0) は収束しなくなる。BIC（1)、BIC（2）と SB-BIC（0）は広範囲のペナルティ数に対して安定であるが、BIC (1) と BIC (2) はペナルティ数が 10^2 から 10^4 に増加すると反復回数が増加する（BIC（1）は 201 から 259 に、BIC（2）は 176 から 232)。SB-BIC（0）は BIC（1)、BIC（2）と比較して収束までの反復回数は多いが計算時間は短く、最も効率的である。

表 1.6.6 は各前処理手法において、様々なペナルティ数に関して、$[M]^{-1}[A]$の固有値分布から得られる E_{max}, E_{min} および κの値である。SB-BIC (0) についてはペナルティ数が変化しても、E_{max}, E_{min} および κの値は変化しないが、BIC（1）と BIC（2）については、ペナルティ数が 10^2 から 10^4 に増加すると κの値が増加する。これは表 1.6.5 に示した BIC（1）と BIC（2）の反復回数が増加したのに対応している。この計算例では、形状が簡易ブロックモデルと比較すると形状は複雑であり、メッシュ形状も不規則で歪んでいるものもある。要素の歪みは、係数行列[A]、$[M]^{-1}[A]$の固有値分布に直接に影響する。SB-BIC（0）はこのような条件下においても安定している。

簡易ブロックおよび西南日本のいずれのモデルにおいても、$[M]^{-1}[A]$の条件数は前処理手法の収束性の評価に役立つパラメータである。簡易ブロックモデルにおいては、BIC（1）および BIC（2)に関する条件数は SB-BIC(0)のそれと比較して小さく、収束までの反復回数は少ない(表 1.6.2～表 1.6.4)。一方、西南日本モデルにおいては、ペナルティ数が 10^4 よりも大きくなると、BIC (1) および BIC（2）に関する条件数は SB-BIC（0）のそれよりも大きくなるが、収束までの反復回数は少ない（表 1.6.5 および表 1.6.6)。

1.6.4 大規模並列計算

1.6.4.1 簡易ブロックモデル

図 1.6.6 に示す簡易ブロックモデルについて、大規模計算を実施した。以下のようなモデルを使用している：

NX1＝70，NX2=70，NY=40，NZ1=70，NZ2=70（図 1.6.6（a））
要素数＝784,000，節点数＝823,813，自由度数＝2,471,439

各種前処理手法について、様々なペナルティ数に関して並列計算を実施した。並列計算にあたっては、1.6.3 で述べた接触問題用領域分割手法を使用した。計算にあたっては日立 SR2201 を 16～128 PE 使用した。並列プログラミングモデルとしては Flat-MPI を使用した。

表 1.6.7 は 128 PE を使用した場合の各種前処理における計算結果である。ペナルティ数が 10^4 より大きくなると BIC（0）は収束しなくなる。BIC（1）、BIC（2）と SB-BIC（0）は広範囲のペナルティ数に対して安定である。SB-BIC（0）は BIC（1）、BIC（2）と比較して収束までの反復回数は多いが計算時間は短く、最も効率的である。表 1.6.8 と図 1.6.10 は同じ問題を PE 数を 16 から 128 まで変化させて解いた場合の結果である。BIC（1）は PE 数が 64 より小さいとメモリ不足で動かず、BIC（2）に至っては 128 PE の場合のみ計算が可能であった。表 1.6.8 と図 1.6.10 からわかるように、BIC（0）と SB-BIC（0）において、PE 数を 16 から 128 に変化させると、局所前処理の影響で反復回数は増大するが、16 PE から 128 PE での増加は 11%程度である。16 PE の場合を基準とすると、128PE における加速率は 120 以上である。

図 1.6.11 は各前処理手法における必要メモリサイズに関して比較したものである。日立 SR2201 の各プロセッサのメモリは 256MB であるが、このうちアプリケーションに利用可能なのは 224MB である。例えば、BIC（2）は 14.4 GB のメモリを必要とするが、64 PE では利用可能メモリの合計は、224MB×64／1000＝14.34GB であるため、64 PE では動かないということになる。SB-BIC（0）の必要メモリ容量は BIC（0）とほぼ同じであり、BIC（1）の 50%以下、BIC（2）の 25%程度である。SB-BIC（0）の必要メモリ容量は接触要素の数や、各「Selective Block」のサイズによって変化しうるが、いずれにせよ、BIC（1）や BIC（2）と比較すると少ない。

1.6.4.2 西南日本モデル

西南日本モデルについて、SB-BIC（0）前処理を使用して、大規模並列計算を実施した。計算にあたっては日立 SR2201 を 16～128 PE 使用した。図 1.6.9 に示すメッシュをグローバルに 2 回細分化し得られた 997,422 節点，960,509 要素の解析モデルを使用した。総自由度数は 2,992,264 である。

表 1.6.9 と図 1.6.12 は同じ問題を PE 数を 16 から 128 まで変化させて解いた場合の結果である。BIC（0）と SB-BIC（0）において、PE 数を 16 から 128 に変化させると、局所前処理の影響で反復回数は増大するが、16 PE から 128 PE での増加は 15%程度である。16 PE の場合を基準とすると、128PE における加速率は 107 である。

表 1.6.1 Iterations/computation time for convergence ($\varepsilon=10^{-8}$) on a single PE of Intel Xeon 2.8 GHz by preconditioned CG for the 3D elastic fault-zone contact problem in 図 1.6.6 (83,664 DOF).: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iterations	Set-up (sec.)	Solve (sec.)	Set-up+Solve (sec.)	Single Iteration (sec.)	Memory Size (MB)
Diagonal	10^2	1531	<0.01	75.1	75.1	0.049	119
Scaling	10^6	No Conv.	-	-	-	-	
IC(0)	10^2	401	0.02	39.2	39.2	0.098	119
(Scalar Type)	10^6	No Conv.	-	-	-	-	
BIC(0)	10^2	388	0.02	37.4	37.4	0.097	59
	10^6	2590	0.01	252.3	252.3	0.097	
BIC(1)	10^2	77	8.5	11.7	20.2	0.152	176
	10^6	78	8.5	11.8	20.3	0.152	
BIC(2)	10^2	59	16.9	13.9	30.8	0.236	319
	10^6	59	16.9	13.9	30.8	0.236	
SB-BIC(0)	10^0	114	0.10	12.9	13.0	0.113	67
	10^6	114	0.10	12.9	13.0	0.113	

Mp=q where $M=(L+D)D^{-1}(D+U)$

Forward Substitution

$(L+D)p= q$: $p= D^{-1}(q-Lp)$

Backward Substitution

$(I+ D^{-1} U)p_{new}= p_{old}$: $p= p – D^{-1} Up$

```
do i= 1, N
  SW1= Z(3*i-2)
  SW2= Z(3*i-1)
  SW3= Z(3*i  )
  isL= INL(i-1)+1
  ieL= INL(i)
  do j= isL, ieL
      k= IAL(j)
     X1= Z(3*k-2,Z)
     X2= Z3*k-1,Z)
     X3= WW(3*k  ,Z)
    SW1= SW1 - AL(1,1,j)*X1 - AL(1,2,j)*X2 - AL(1,3,j)*X3
    SW2= SW2 - AL(2,1,j)*X1 - AL(2,2,j)*X2 - AL(2,3,j)*X3
    SW3= SW3 - AL(3,1,j)*X1 - AL(3,2,j)*X2 - AL(3,3,j)*X3
  enddo

  X1= SW1
  X2= SW2
  X3= SW3
  X2= X2 - ALU(2,1,i)*X1
  X3= X3 - ALU(3,1,i)*X1 - ALU(3,2,i)*X2
  X3= ALU(3,3,i)*  X3
  X2= ALU(2,2,i)*( X2 - ALU(2,3,i)*X3 )
  X1= ALU(1,1,i)*( X1 - ALU(1,3,i)*X3 - ALU(1,2,i)*X2)
  WW(3*i-2,Z)= X1
  WW(3*i-1,Z)= X2
  WW(3*i  ,Z)= X3
enddo
```

Full LU factorization for 3x3 block

```
WW (:,Z) : work vector
INL(:)   : coefficient index of the lower triangular matrix (LTM)
IAL(:)   : connected nodes of LTM

AL (3,3,:): 3x3 coefficient matrix component of the LTM
ALU(3,3,:): 3x3 LU factorization for each 'node'
```

図 1.6.1 Procedure of the 3×3 block ILU(0) preconditioning: forward substitution[9]

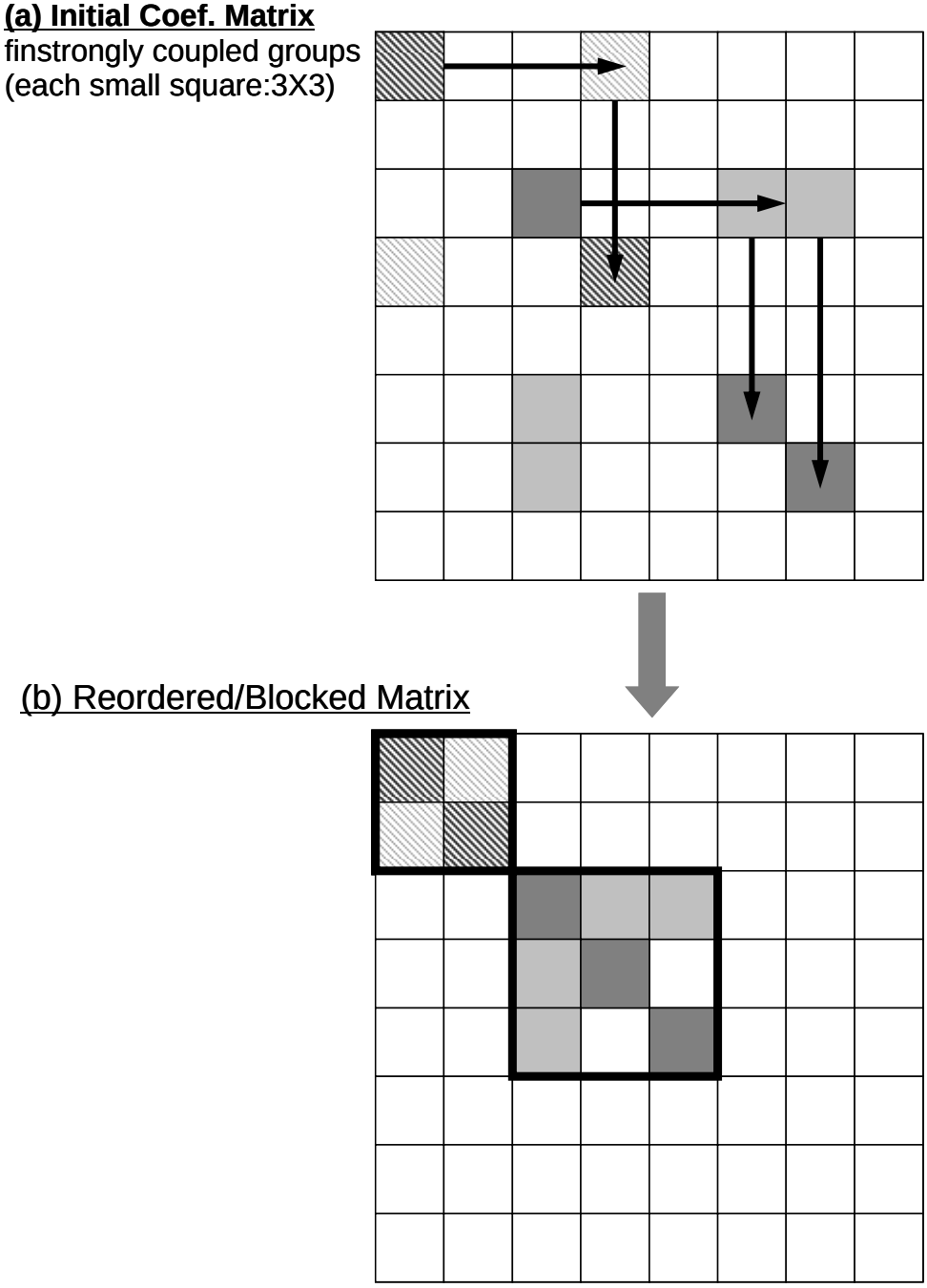


図 1.6.2　Procedure of the *selective blocking* : Strongly coupled elements are put into the same *selective block*. (a) searching for strongly coupled components and (b) reordering and selective blocking. Full LU factorization procedure is applied to each *selective block*. Coupled finite-element nodes in contact groups can be solved in direct method during preconditioning procedure. In SB-BIC(0) (BIC(0) preconditioning combined with the *selective blocking* reordering), no *inter-block* fill-in is considered. Only *inter-node* fill-in in each *selective block* is considered in SB-BIC(0) [9].

```
      do ib= 1, NBLOCKtot
        NB0size= BLOCKstack(ib) - BLOCKstack(ib-1)
        (FORWARD SUBSTITUTIONS)
        do i= 1, NB0size
          ii= i + iBS
          WVAL1= 0.d0
          WVAL2= 0.d0
          WVAL3= 0.d0
          do j= 1, NB0size
            WR1= WKB(3*j-2)
            WR2= WKB(3*j-1)
            WR3= WKB(3*j  )
            WVAL1= WVAL1 + ALU(3*i-2,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i-2,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i-2,3*j  ,ib) * WR3
            WVAL2= WVAL2 + ALU(3*i-1,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i-1,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i-1,3*j  ,ib) * WR3
            WVAL3= WVAL3 + ALU(3*i  ,3*j-2,ib) * WR1                    &
     &                   + ALU(3*i  ,3*j-1,ib) * WR2                    &
     &                   + ALU(3*i  ,3*j  ,ib) * WR3
          enddo
          WW(3*ii-2,Z)= WVAL1
          WW(3*ii-1,Z)= WVAL2
          WW(3*ii  ,Z)= WVAL3
        enddo
      enddo
```

図 1.6.3 Procedure of the *selective blocking*: Full LU factorization procedure for a (3×NB) × (3×NB) size *selective block.* [9]

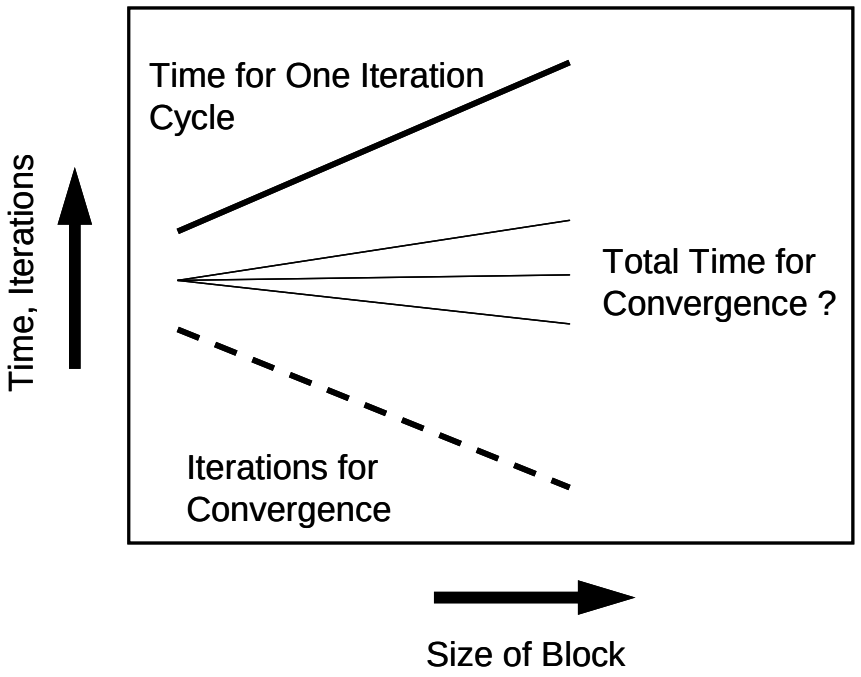


図 1.6.4 Trade-off between convergence and computational cost per one iteration cycle according to block size in CEBE type method.

表 1.6.2 Iterations/computation time for convergence ($\varepsilon=10^{-8}$) on 8 PEs of Intel Xeon 2.8 GHz cluster by preconditioned CG for the 3D elastic fault-zone contact problem (83,664 DOF).: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering. Effect of repartitioning method in 図 1.6.6 is evaluated.

		ORIGINAL Partitioning		**IMPROVED** Partitioning	
Preconditioning	λ	Iterations	Set-up+Solve (sec.)	Iterations	Set-up+Solve (sec.)
BIC(0)	10^2	703	7.5	489	5.3
	10^6	4825	50.6	3477	37.5
BIC(1)	10^2	613	11.3	123	2.7
	10^6	2701	47.7	123	2.7
BIC(2)	10^2	610	19.5	112	4.7
	10^6	2448	73.9	112	4.7
SB-BIC(0)	10^0	655	10.9	165	2.9
	10^6	3498	58.2	166	2.9

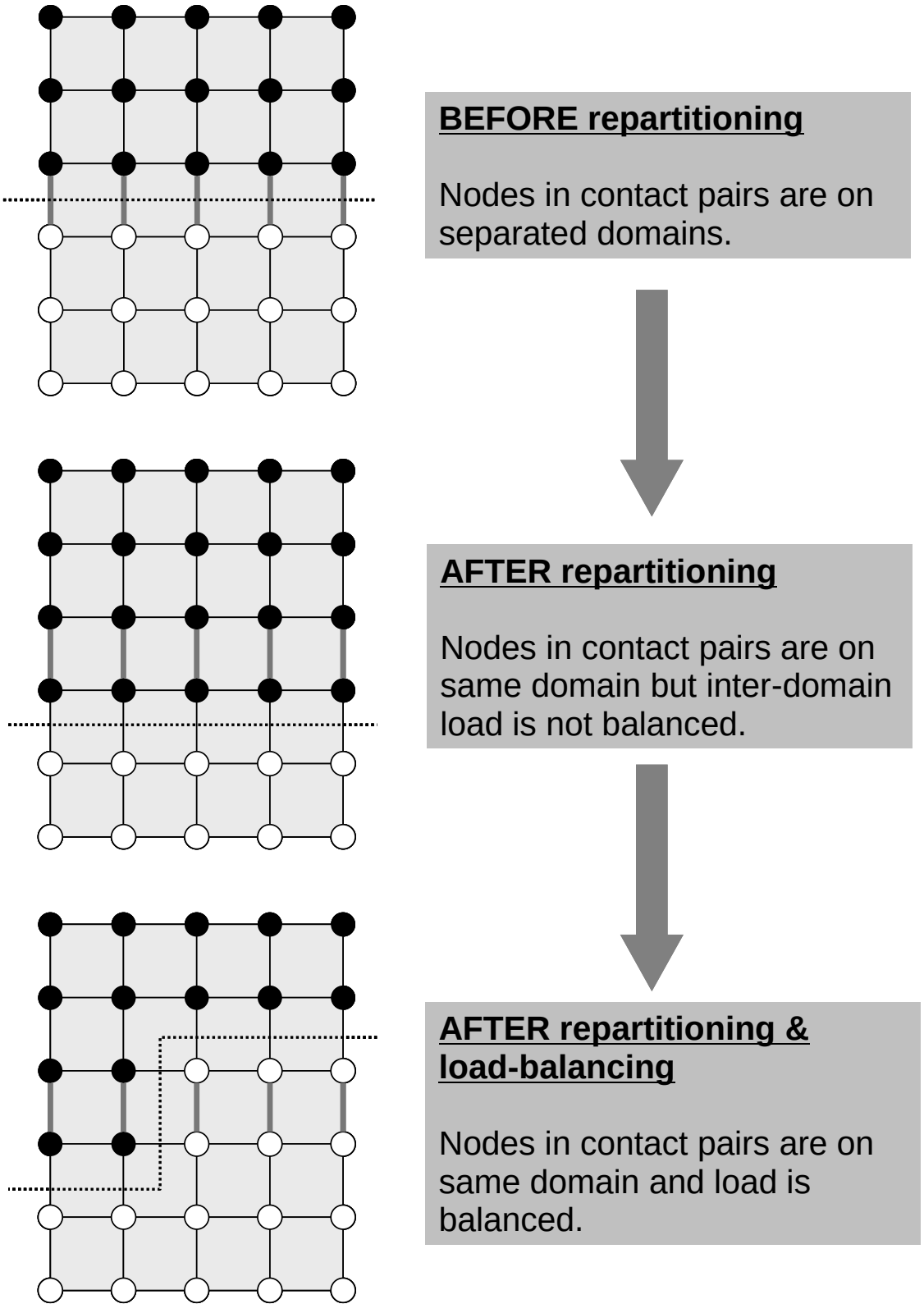
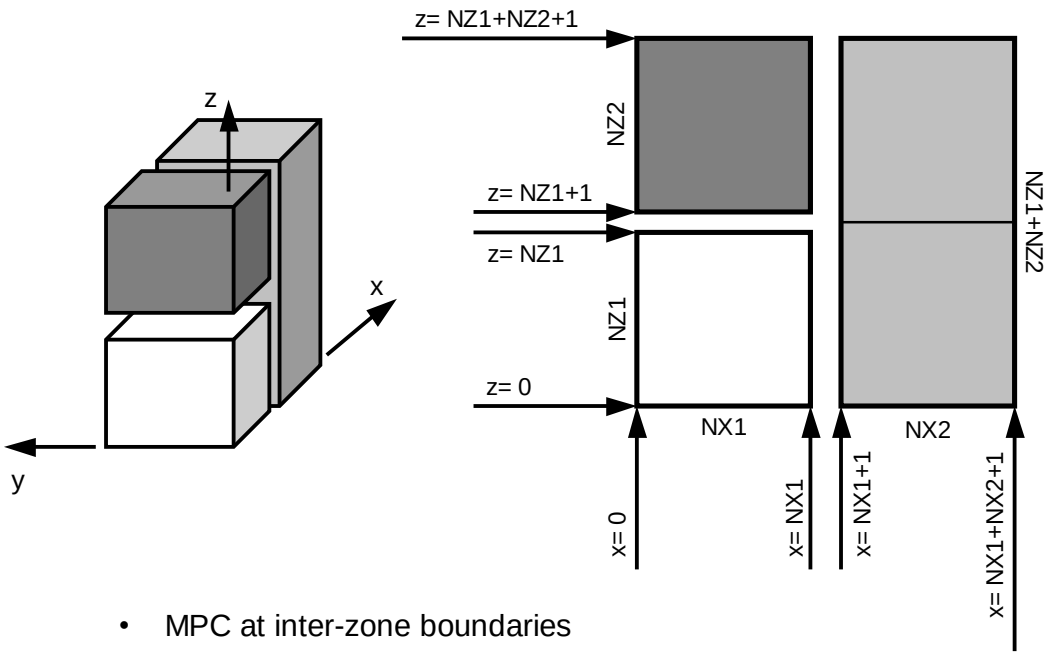
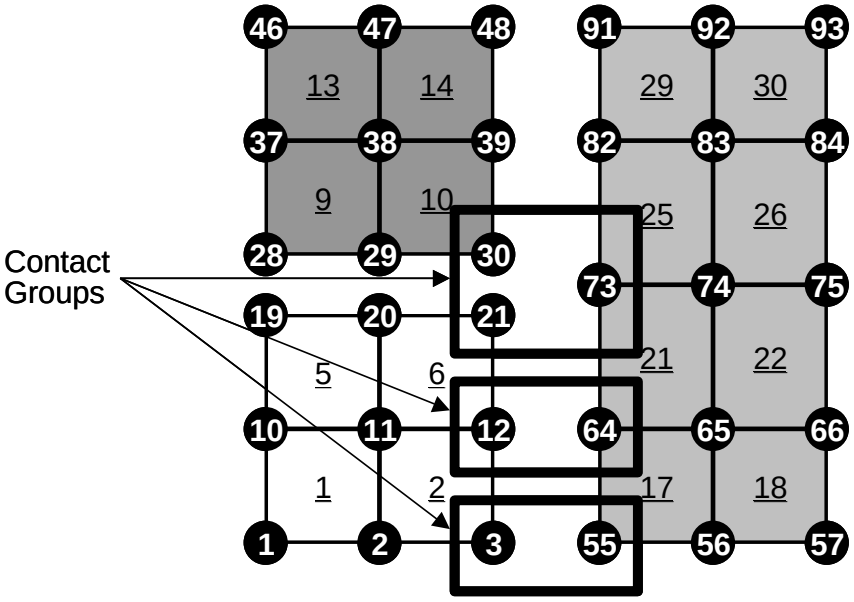


図 1.6.5 Partitioning strategy for the nodes in contact groups [9]



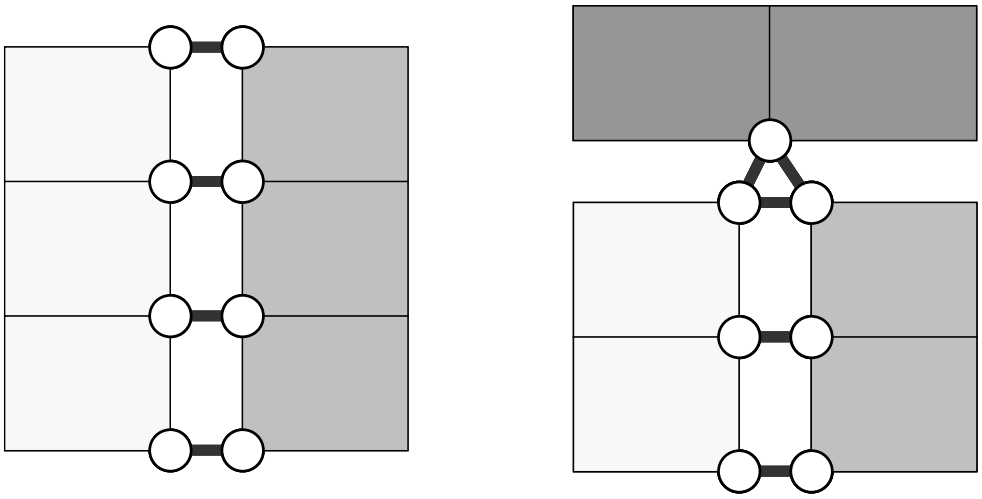
- MPC at inter-zone boundaries
- Symmetric condition at the x=0 and y=0 surfaces
- Dirichlet fixed condition at the z=0 surface
- Uniform distributed load at the z= Zmax surface

(a) Model and boundary conditions



(b) Node, elements and contact groups

図 1.6.6 Description of the simple block model [9]



put 111-type element with Large stiffness for contact pairs.

図 1.6.7　111-type element (Rod/Beam) is put in each contact group and very large stiffness corresponding to penalty is applied [9]

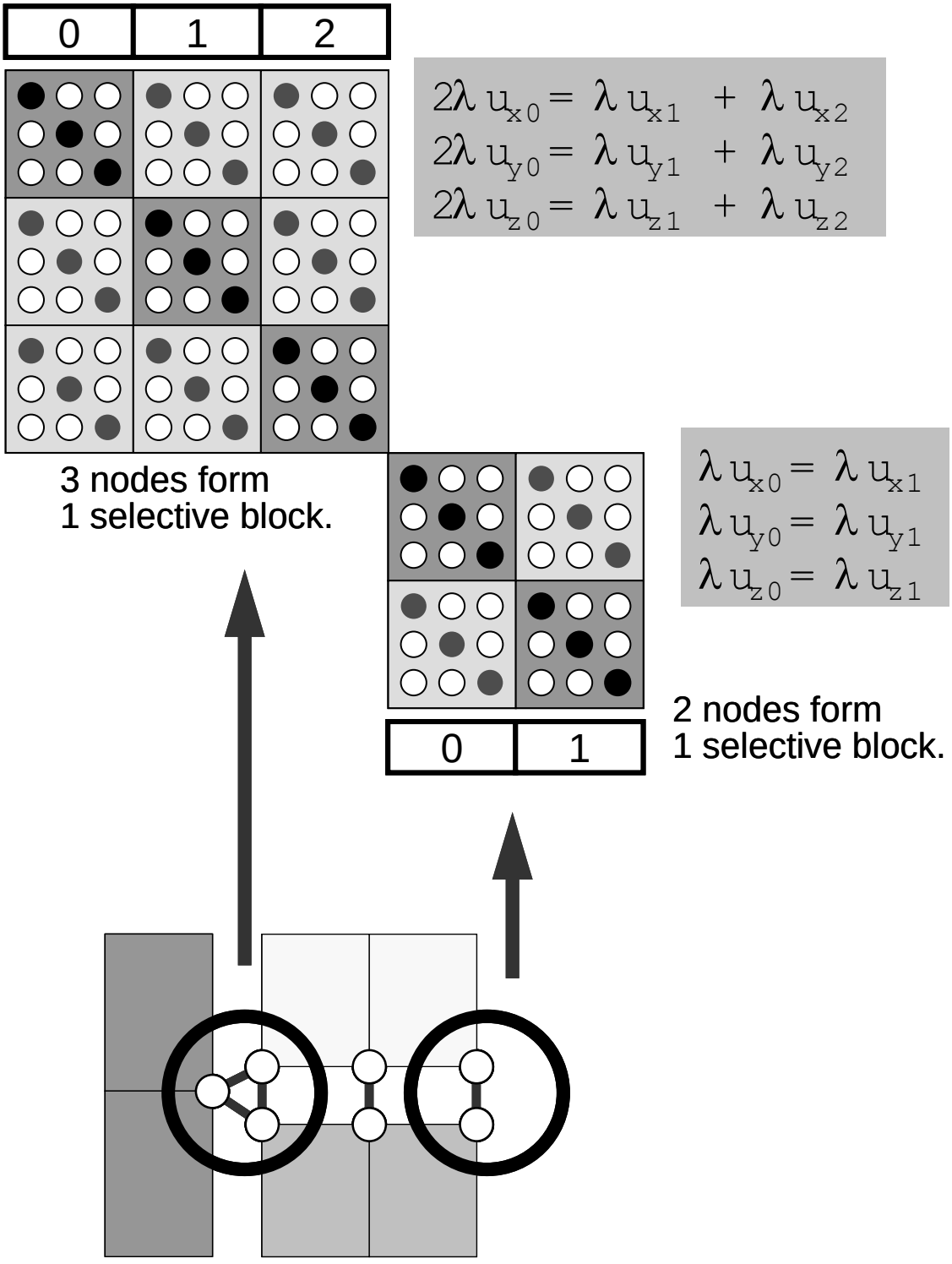


図 1.6.8　Matrix operation of nodes in a contact group [9]

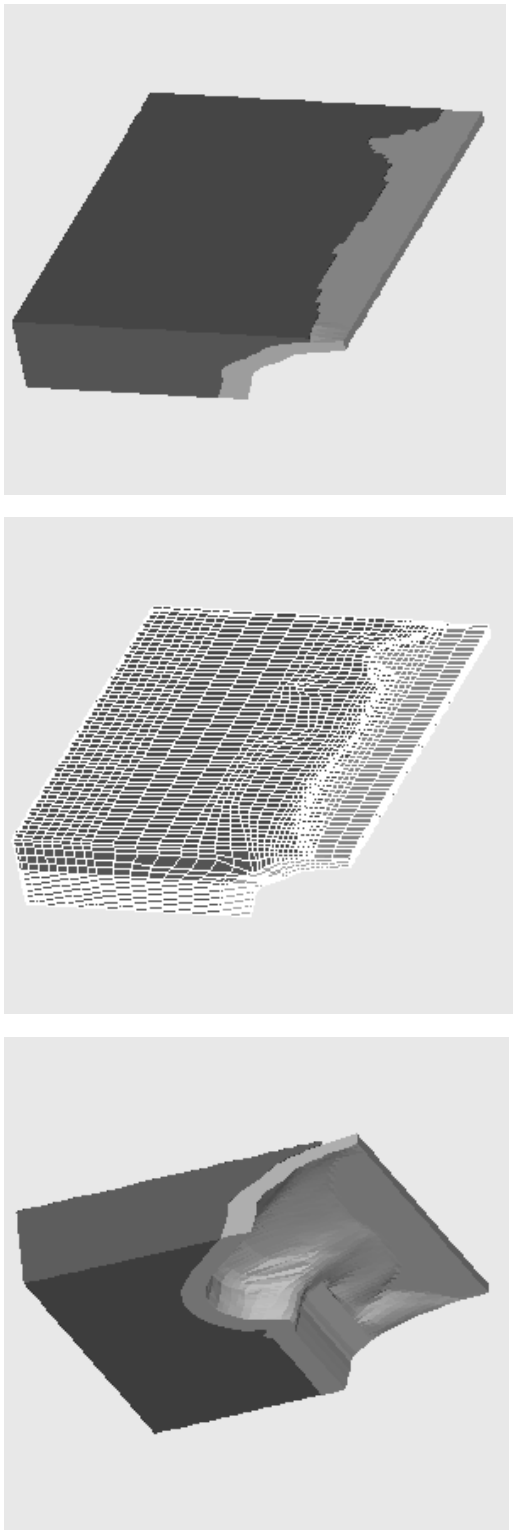


図 1.6.9　Description of the Southwest Japan model This model consists crust (dark gray) and subduction plate (light gray). 27,195 nodes and 23,831 tri-linear (1st order) hexahedral elements are included [9].

表 1.6.3　Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a single PE of COMPAQ Alpha 21164/600MHz by preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in 図 1.6.6 (83,664DOF).: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	388	202.
	10^4	No Conv.	N/A
BIC(1)	10^2	77	89.
	10^6	77	89.
	10^{10}	78	90.
BIC(2)	10^2	59	135.
	10^6	59	135.
	10^{10}	60	137.
SB-BIC(0)	10^2	114	61.
	10^6	114	61.
	10^{10}	114	61.

表 1.6.4　Largest and smallest eigenvalues (E_{min}, E_{max}) and $\kappa= E_{max}/E_{min}$ of $[M]^{-1}[A]$ for a wide range of penalty parameter values: 3D elastic contact problem for simple block model with MPC condition in 図 1.6.6 (83,664DOF).

Preconditioning		$\lambda=\tilde{1}^2$	$\lambda=\tilde{1}^6$	$\lambda=\tilde{1}^{\tilde{1}}$
BIC(0)	E_{min}	4.845568E-03	4.865363E-07	4.865374E-11
	E_{max}	1.975620E+00	1.999998E+00	2.000000E+00
	κ	4.077170E+02	4.110686E+06	4.110681E+10
BIC(1)	E_{min}	8.901426E-01	8.890643E-01	8.890641E-01
	E_{max}	1.013930E+00	1.013863E+00	1.013863E+00
	κ	1.139065E+00	1.140371E+00	1.140371E+00
BIC(2)	E_{min}	9.003662E-01	8.992896E-01	8.992895E-01
	E_{max}	1.020256E+00	1.020144E+00	1.020144E+00
	κ	1.133157E+00	1.134388E+00	1.134389E+00
SB-BIC(0)	E_{min}	6.814392E-01	6.816873E-01	6.816873E-01
	E_{max}	1.005071E+00	1.005071E+00	1.005071E+00
	κ	1.474924E+00	1.474387E+00	1.474387E+00

表 1.6.5　Iterations/CPU time (includes factorization) for convergence ($\varepsilon=10^{-8}$) on a single PE of COMPAQ Alpha 21164/600MHz by preconditioned CG for the 3D elastic contact problem for Southwestern Japan model with MPC condition in 図 1.6.9 (81,585DOF).: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	344	172.
	10^4	No Conv.	N/A
BIC(1)	10^2	201	192.
	10^4	256	237.
	10^6	256	237.
	10^8	258	240.
	10^{10}	259	241.
BIC(2)	10^2	176	288.
	10^4	229	360.
	10^6	230	361.
	10^8	230	361.
	10^{10}	232	364.
SB-BIC(0)	10^2	297	149.
	10^4	295	148.
	10^6	295	148.
	10^8	295	148.
	10^{10}	295	148.

表 1.6.6 Largest and smallest eigenvalues (E_{min}, E_{max}) and $\kappa = E_{max}/E_{min}$ of $[M]^{-1}[A]$ for a wide range of penalty parameter values: 3D elastic contact problem for Southwestern Japan model with MPC condition in 図 1.6.9 (81,585DOF).

Preconditioning		$\lambda=\tilde{1}^{2}$	$\lambda=\tilde{1}^{4}$	$\lambda=\tilde{1}^{6}$	$\lambda=\tilde{\tilde{1}}^{1}$
BIC(0)	E_{min}	1.970395E-02	1.999700E-04	1.999997E-06	2.000000E-10
	E_{max}	1.005194E+00	1.005194E+00	1.005194E+00	1.005194E+00
	κ	5.101486E+01	5.026725E+03	5.025979E+05	5.025971E+09
BIC(1)	E_{min}	3.351178E-01	2.294832E-01	2.286390E-01	2.286306E-01
	E_{max}	1.142246E+00	1.142041E+00	1.142039E+00	1.142039E+00
	κ	3.408491E+00	4.976580E+00	4.994944E+00	4.995128E+00
BIC(2)	E_{min}	3.558432E-01	2.364909E-01	2.346180E-01	2.345990E-01
	E_{max}	1.058883E+00	1.088397E+00	1.089189E+00	1.089196E+00
	κ	2.975702E+00	4.602277E+00	4.642391E+00	4.642800E+00
SB-BIC(0)	E_{min}	2.380572E-01	2.506369E-01	2.507947E-01	2.507963E-01
	E_{max}	1.005194E+00	1.005455E+00	1.005465E+00	1.005466E+00
	κ	4.222491E+00	4.011600E+00	4.009117E+00	4.009092E+00

表 1.6.7 Iterations/elapsed execution time (includes factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 128 PEs using preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in 図 1.6.6 (2,471,439 DOF). Domains are partitioned according to the contact group information.: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering.

Preconditioning	λ	Iter #	sec.
BIC(0)	10^2	998	118.
	10^4	No Conv.	N/A
BIC(1)	10^2	419	98.
	10^6	419	98.
	10^{10}	421	99.
BIC(2)	10^2	394	171.
	10^6	394	171.
	10^{10}	396	172.
SB-BIC(0)	10^2	565	71.
	10^6	566	71.
	10^{10}	567	72.

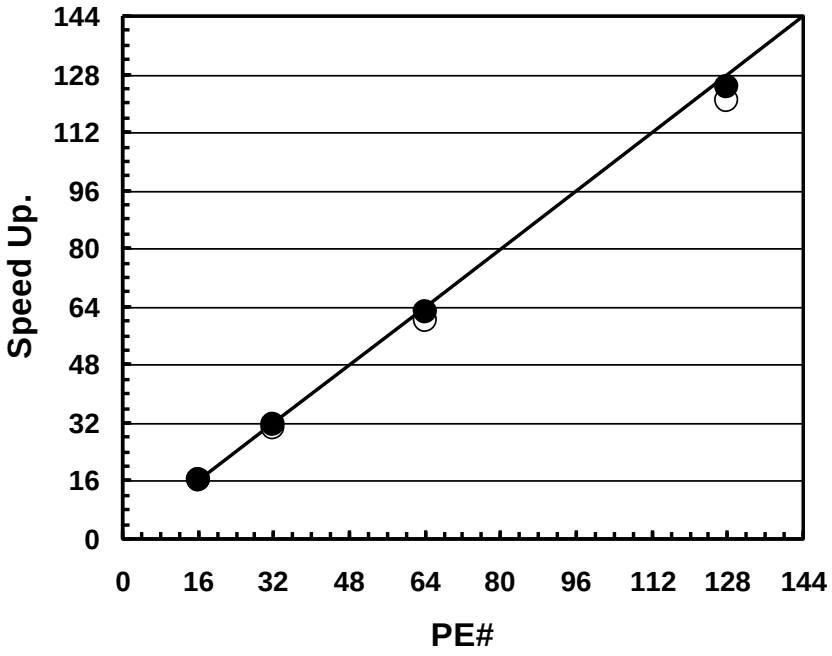
表 1.6.8 Iterations/elapsed execution time (includes factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using preconditioned CG for the 3D elastic contact problem for simple block model with MPC condition in 図 1.6.6 (2,471,439 DOF). Domains are partitioned according to the contact group information.: BIC(n): Block IC with n-level fill-in, SB-BIC(0): BIC(0) with the selective blocking reordering.

$\lambda=10^2$

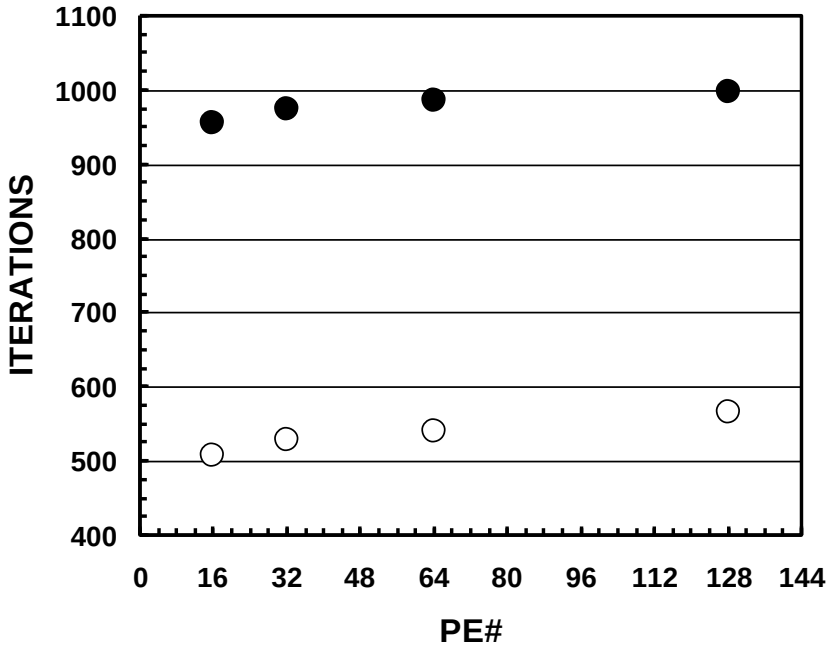
Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
BIC(0)	iters	956	975	986	998
	sec.	919	469	236	118
	ratio	16.0	31.4	62.5	124.4
BIC(1)	iters			396	419
	sec.	N/A	N/A	190	98
	ratio			64.0	124.3
BIC(2)	iters				394
	sec.	N/A	N/A	N/A	171
	ratio				-
SB-BIC(0)	iters	508	529	541	565
	sec.	540	282	144	71
	ratio.	16.0	30.7	60.2	120.7

$\lambda=10^6$

Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
BIC(1)	iters			395	419
	sec.	N/A	N/A	190	98
	ratio			64.0	124.2
BIC(2)	iters				394
	sec.	N/A	N/A	N/A	171
	ratio				-
SB-BIC(0)	iters	510	532	543	566
	sec.	542	283	144	71
	ratio	16.0	30.6	60.5	121.9



(a) Speed-up ratio



(b) Iteration number for convergence

図 1.6.10 Parallel performance based on elapsed execution time including communication and iterations for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using preconditioned CG for the 3D elastic contact problem with MPC condition ($\lambda=10^{2}$) in 図 1.6.6 (2,471,439 DOF). Domains are partitioned according to the contact group information. (White Circles: SB-BIC(0), Black-Circles: BIC(0)).

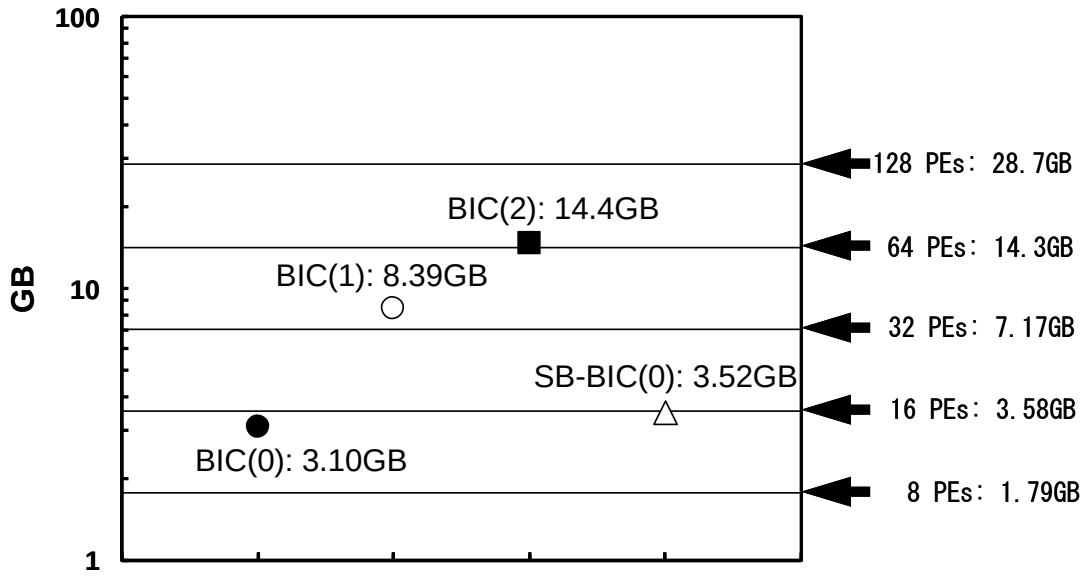
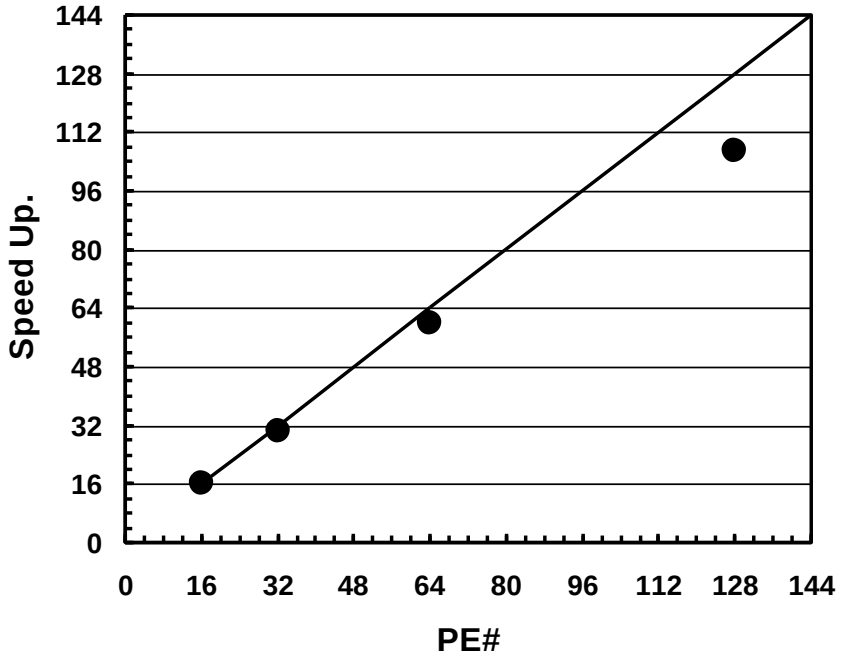


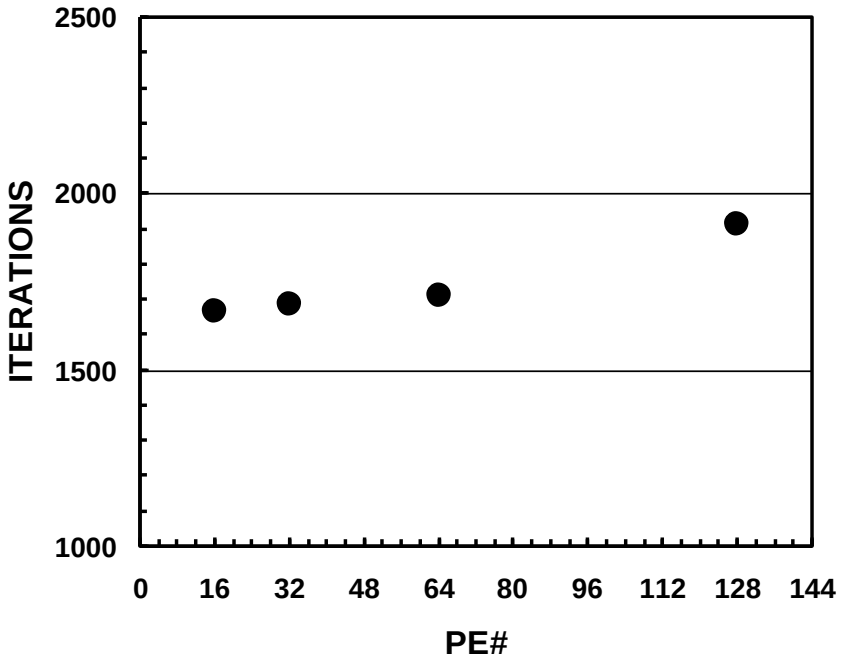
図 1.6.11 Required memory size of CG solvers with various types preconditioners for the 3D elastic contact problem with MPC condition ($\lambda=10^2$) in 図 1.6.6 (2,471,439 DOF) and available memory size on Hitachi SR2201 (Black-Circles: BIC(0), White-Circles: BIC(1), Black-Squares: BIC(2), White Triangles: SB-BIC(0)).

表 1.6.9 Iterations/elapsed execution time (including factorization, communication overhead) for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using SB-BIC(0) CG for the 3D elastic contact problem for Southwest Japan model with MPC condition ($\lambda=10^6$) in 図 1.6.9 (2,992,264 DOF). Domains are partitioned according to the contact group information.

Preconditioning		16 PEs	32 PEs	64 PEs	128 PEs
SB-BIC(0)	iters	1665	1686	1710	1912
	sec.	1901.	993.	506.	284.
	ratio	16.0	30.6	60.1	107.2



(a) Speed-up ratio



(b) Iteration number for convergence

図 1.6.12 Parallel performance based on elapsed execution time including communication and iterations for convergence ($\varepsilon=10^{-8}$) on a Hitachi SR2201 with 16 to 128 PEs using SB-BIC(0) CG for the 3D elastic contact problem with MPC condition ($\lambda=10^{6}$) in 図 1.6.9 (2,992,266 DOF). Domains are partitioned according to the contact group information.

1.7 Sparse Approximate Inverse (SAI)

SAI は「近似逆行列」であり、以下に示す式（1.7-1）の最小二乗問題によって係数行列 A の逆行列を近似するような前処理行列 M^{-1} を求めるものである：

$$\min \left\| AM^{-1} - E \right\|_F^2 = \sum_{k=1}^{n} \min_{m_k \in R^n} \left\| A\mathbf{m}_k - \mathbf{e}_k \right\|_2^2 \tag{1.7.1}$$

ただし、ベクトル $\mathbf{m}_k$ は行列 M^{-1} の k 列目の列成分、ベクトル $\mathbf{e}_k$ は単位行列 E の k 列目の列成分を表している。ベクトル $\mathbf{m}_k$ は式(1.7-1)の右辺を最小化するように求めればよいが、計算量を減少させるために、あらかじめ前処理行列 M^{-1} の非ゼロ要素の場所を決定しておき、その非ゼロ要素のみを取り出すことを考える。最も簡単な方法としては、前処理行列 M^{-1} の非ゼロ要素の位置を係数行列 A と同じにすることである。ILU 系前処理と同様、Fill-in を許せば、前処理行列 M^{-1} の性能は向上するが、計算量、記憶容量ともに増加する。前処理行列 M^{-1} の非ゼロ要素の行インデックス集合を J，J の各々の行インデックスに対応する非ゼロ要素の列インデックス集合を I とすると、式(1.7-1)は：

$$\sum_{k=1}^{n} \min_{m_k(J) \in R^J} \left\| A(I, J)\, \mathbf{m}_k(J) - \mathbf{e}_k(I) \right\|_2^2 \tag{1.7.2}$$

を解くことに帰着できる。ここで式（1.6-2）は、n 本の I×J 次の最小二乗問題：

$$\min_{m_k(J) \in R^J} \left\| A(I, J)\, \mathbf{m}_k(J) - \mathbf{e}_k(I) \right\|_2^2 \qquad (1 \le k \le n) \tag{1.7.3}$$

の計算を独立に行なうことができるので、式（1.7-3）の最小二乗問題を各 CPU に均等に割り当てて、並列に計算することが可能となる。式（1.7-3）の最小二乗問題は Givens 回転行列を使用して QR 分解によって解くことが可能である。

HEC-MW では LAPACK の DGELS〔13〕を使用している。

図 1.6.6 に示すブロック間の接触問題〔9〕について、SAI を適用して計算し、他の前所為手法と比較した例を示す。図 1.6.6 に示す接触グループ（Contact Groups）を構成する節点にペナルティ条件を課する場合、並列計算においてはこれらの節点が同じ領域上にある場合（coupled partition）に収束の速いことが知られている〔9〕。逆に、同じ領域上にない場合（decoupled）は収束が非常に遅い。表 1.7.1 に示すように、SAI は「decoupled partition」の場合も良好な収束を示すことがわかる。表 1.7.2 に示すように、他の手法と比較して、「set-up」に時間を要しているものの、概ね良好な収束を示していることがわかる。打ち切りパラメータを小さくすることによって、反復回数は減るが、計算時間は増加する。

表 1.7.1 Comparison of preconditioners for simple block prblem with 27,888 nodes (83,664 DOF), SAI works well for 8PE case in "decoupled" partitioning, Xeon 2.8GHz Cluster.

1 PE

	SAI/GPBiCG	SAI/BiCGSTAB	SB-BILU(0)CG	BILU(1)-CG
ITERs	190	187	114	78
set-up	10.7	10.7	<0.01	8.2
solver	23.9	17.3	18.6	11.7

8 PEs (decoupled)

	SAI/GPBiCG	SAI/BiCGSTAB	SB-BILU(0)CG	BILU(1)-CG
ITERs	191	193	3498	2701
set-up	1.0	1.0	<0.01	0.5
solver	4.1	3.0	56.9	46.9

8 PEs (coupled)

	SAI/GPBiCG	SAI/BiCGSTAB	SB-BILU(0)CG	BILU(1)-CG
ITERs			166	123
set-up			<0.01	0.5
solver			2.8	2.2

表 1.7.2 Comparison of preconditioners for simple block prblem with 2.4M DOF, coupled partittioing, Xeon 2.8GHz Cluster with 32 PEs.

Time	BIC(1)-CG	BIC(2)-CG	SB-BIC(0)CG	SAI/BiCGSTAB		
				0.20-0.20	0.10-0.10	0.05-0.05
ITERs	382	333	535	843	671	545
Solver	87.4	117.6	124.0	136.0	121.7	119.1
Set-up+ Solver	108.2	149.6	124.0	155.2	142.7	200.2

1.8 自由度消去 MPC

1.8.1 概要

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

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

通常、自由度消去法による MPC 条件の組み込みは、マトリックスの非ゼロのプロファイルが変更される。HEC-MW のマトリックスは非ゼロ成分のみを保持している（CRS：Compressed Row Storage）ため、非ゼロのプロファイルを変更するためには煩雑な処理が必要となり、望ましくない。

そこで、以下に示す方法により、マトリックス自体には MPC 条件を組み込まず、ソルバーの各反復の中で MPC による拘束自由度を消去する方法を採用する。

1.8.2 MPC 前処理付き反復法

解くべき方程式は以下のように表される。

$$\mathbf{Ku}=\mathbf{f} \tag{1.8.1}$$

$$\mathbf{Bu}=\mathbf{g} \tag{1.8.2}$$

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

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

全自由度のうち、独立自由度のみが意味を持つ未知数ベクトルを $\mathbf{u}'$（$1\times N$） 、$\mathbf{u}$ と $\mathbf{u}'$ との間の変換行列を $\mathbf{T}$ とすると、(1.8.2)の拘束条件は

$$\mathbf{u}=\mathbf{Tu}'+\mathbf{u}_g \tag{1.8.3}$$

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

たとえば、$N=5$、$u_5-u_4=1$ の場合、多点拘束条件は、(1.8.2)の形式では、

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

(1.8-3)の形式では、

$$\begin{Bmatrix}u_1\\u_2\\u_3\\u_4\\u_5\end{Bmatrix}=\begin{bmatrix}1&0&0&0&0\\0&1&0&0&0\\0&0&1&0&0\\0&0&0&1&0\\0&0&0&1&0\end{bmatrix}\begin{Bmatrix}u_1'\\u_2'\\u_3'\\u_4'\\u_5'\end{Bmatrix}+\begin{Bmatrix}0\\0\\0\\0\\1\end{Bmatrix}$$

と表わされる。

(1.8.3)を(1.8.1)に代入し、定数項を右辺に移項すると、

$$\mathbf{KTu}'=\mathbf{f}-\mathbf{Ku}_g$$

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

$$\mathbf{T}^T\mathbf{KTu}'=\mathbf{T}^T\left(\mathbf{f}-\mathbf{Ku}_g\right) \tag{1.8.4}$$

が得られる。(1.8.4)が、最終的に解くべき、多点拘束を組み込んだ方程式である。

(1.8.4)に対して反復解法を適用する。その際、$\mathbf{T}^T\mathbf{KT}$ を陽には保持せず、$\mathbf{T}$ や $\mathbf{T}^T$ との掛け算を必要に応じてその都度行う。図 1.8.1 に共役勾配法を適用した場合のアルゴリズムを示す。通常の反復法におけるマトリックス・ベクトル積１回について、本手法では $\mathbf{T}$ および $\mathbf{T}^T$ との積の計算が増えることになるが、$\mathbf{T}$ はほとんど単位行列であるため、その計算コストは低く抑えることが可能である。

なお、(1.8.4)において、$\mathbf{T}^T\mathbf{KT}$ の従属自由度に関する行および列にはすべて０が入ることになり、解が唯一ではない。しかし、反復解法を適用した場合、未知ベクトルの初期値として従属自由度成分を０とすることで、修正ベクトル・残差ベクトルともに従属自由度成分は常に０となり、従属自由度成分を無視した形での求解が可能である。

$$\mathbf{r}_0 = \mathbf{T}^T\left(\mathbf{f} - \mathbf{K}\mathbf{u}_g\right) - \mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{u}'_0$$
$$\rho_0 = \left(\mathbf{r}_0, \mathbf{r}_0\right)$$
$$\mathbf{p}_0 = \mathbf{r}_0$$
for $k = 0, 1, \cdots$
$$\mathbf{q}_k = \mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{p}_k$$
$$\alpha_k = \frac{\rho_k}{\left(\mathbf{p}_k, \mathbf{q}_k\right)}$$
$$\mathbf{u}'_{k+1} = \mathbf{u}'_k + \alpha_k\mathbf{p}_k$$
$$\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k\mathbf{q}_k$$
check convergence; continue if necessary
$$\rho_{k+1} = \left(\mathbf{r}_{k+1}, \mathbf{r}_{k+1}\right)$$
$$\beta_k = \frac{\rho_{k+1}}{\rho_k}$$
$$\mathbf{p}_{k+1} = \mathbf{r}_{k+1} + \beta_k\mathbf{p}_k$$
end
$$\mathbf{u} = \mathbf{T}\mathbf{u}' + \mathbf{u}_g$$

図 1.8.1 Conjugate Gradient method with direct elimination of multi-point constrained DOFs.

1.8.3 前処理

本手法では、係数行列 $\mathbf{T}^T\mathbf{KT}$ を保持していない。このため、ブロック ILU 前処理およびブロック SSOR 前処理に関しては、通常の前処理を適用することはできない。しかし、$\mathbf{K}$ をもとにした前処理を適用することは可能である。$\mathbf{K}$ と $\mathbf{T}^T\mathbf{KT}$ の違いは、多点拘束条件のかかった自由度に関する行列成分であるが、実際の多点拘束条件付き問題においては、全自由度に対して拘束自由度の数はそれほど大きくない。したがって、$\mathbf{K}$ と $\mathbf{T}^T\mathbf{KT}$ が大きく異なるわけではなく、$\mathbf{K}$ をもとにした前処理によって、前処理の効果を得ることは可能である。

ブロック対角スケーリング前処理に関しては、$\mathbf{T}^T\mathbf{KT}$ の対角ブロックだけを実際に計算・保持するコストは小さく、それを用いた対角スケーリングを適用することにより、効果的な前処理が可能である。

そこで、自由度消去による MPC では、前処理は以下のように行う：

ブロック ILU 前処理：　　$\mathbf{K}$ をもとにした前処理
ブロック SSOR 前処理：　　$\mathbf{K}$ をもとにした前処理
ブロック対角スケーリング：　　$\mathbf{T}^T\mathbf{KT}$ をもとにした前処理

自由度消去による MPC を前処理付き共役勾配法に適用した場合のアルゴリズムを図 1.8.2 に示す。

$$\mathbf{r}_0 = \mathbf{T}^T\left(\mathbf{f} - \mathbf{K}\mathbf{u}_g\right) - \mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{u}'_0$$

for $k = 0, 1, \cdots$

  solve $\mathbf{M}\mathbf{z}^k = \mathbf{r}^k$

  $\rho_k = \left(\mathbf{r}_k, \mathbf{z}_k\right)$

  if $k \equiv 0$

    $\mathbf{p}_0 = \mathbf{r}_0$

  else

    $\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$

    $\mathbf{p}_k = \mathbf{z}_k + \beta_k \mathbf{p}_{k-1}$

  endif

  $\mathbf{q}_k = \mathbf{T}^T\mathbf{K}\mathbf{T}\mathbf{p}_k$

  $\alpha_k = \dfrac{\rho_k}{\left(\mathbf{p}_k, \mathbf{q}_k\right)}$

  $\mathbf{u}'_{k+1} = \mathbf{u}'_k + \alpha_k \mathbf{p}_k$

  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k$

  check convergence; continue if necessary

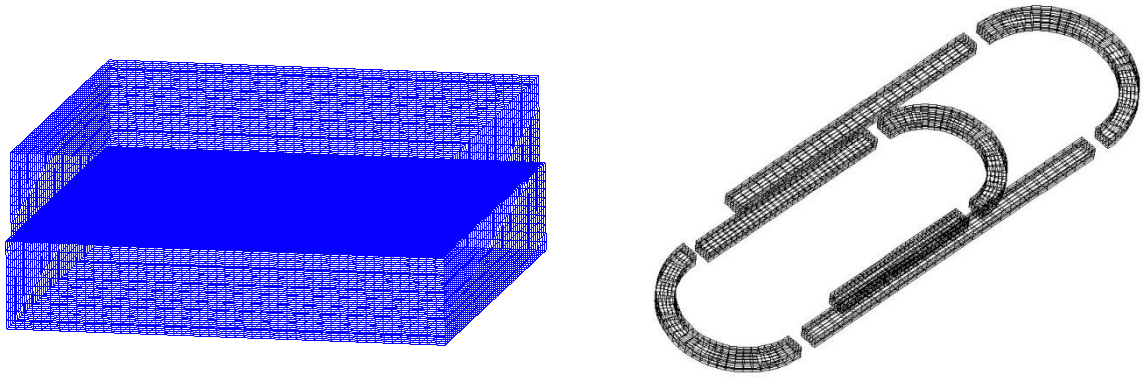
end

$\mathbf{u} = \mathbf{T}\mathbf{u}' + \mathbf{u}_g$

図 1.8.2 Preconditioned Conjugate Gradient method with direct elimination of multi-point constrained DOFs.

1.8.4 ベンチマーク

図 1.8.3 に示すモデルにたいして MPCCG 法を適用し、ペナルティ法（ペナルティ数＝10^4）と比較した。その結果、反復回数・計算時間ともに、50～90%程度減少することを確認した（表 1.8.1 および表 1.8.2）。



(a) Simple block model　　(b) Clip model

図 1.8.3　Test models for MPCCG.

表 1.8.1　Number of iterations and CPU time for the simple block model.

	Iterations	CPU Time (sec)	CPU Time per Iteration (sec)
MPCCG	165	6.21	3.76×10^{-2}
Penalty+CG	1,075	37.35	3.47×10^{-2}

表 1.8.2　Number of iterations and CPU time for the clip model.

	Iterations	CPU Time (sec)	CPU Time per Iteration (sec)
MPCCG	33,349	143	4.18×10^{-3}
Penalty+CG	69,365	279	3.97×10^{-3}

1.9　大規模ノード数、多数コア対応アルゴリズム

有限要素法で解析時間の大部分を占める、反復法ソルバーにおける疎行列ベクトル積に注目し、多数コア CPU で構成される並列計算機に向けた高速化を行う。

多数コア CPU を複数搭載した SMP（symmetric multiprocessor）クラスタでは、各ノードあたりのコア数が大きくなる（図 1.9.1）。ノード数が大きい場合、MPI のみによるフラットな並列化では、通信量が増大し、高い並列化効率が得られなくなることが予想される。このような環境においては、ノード内では OpenMP などによるスレッド並列、ノード間では MPI による並列を用いた、ハイブリッドな並列化が有効と考えられる。

また、計算に使えるコアが増えることで、演算能力は 2 倍、3 倍…と増加していくが、疎行列ベクトル積はメモリのバンド幅によって制限を受けるため、コア数が増えると並列化率は落ちていく。そこで、多数コア環境でのパフォーマンス向上のためには、キャッシュを活用する必要が出てくる。そこで、ブロック CRS（Compressed Row Storage）フォーマットを採用することによりキャッシュを活用した高速化を実現している。

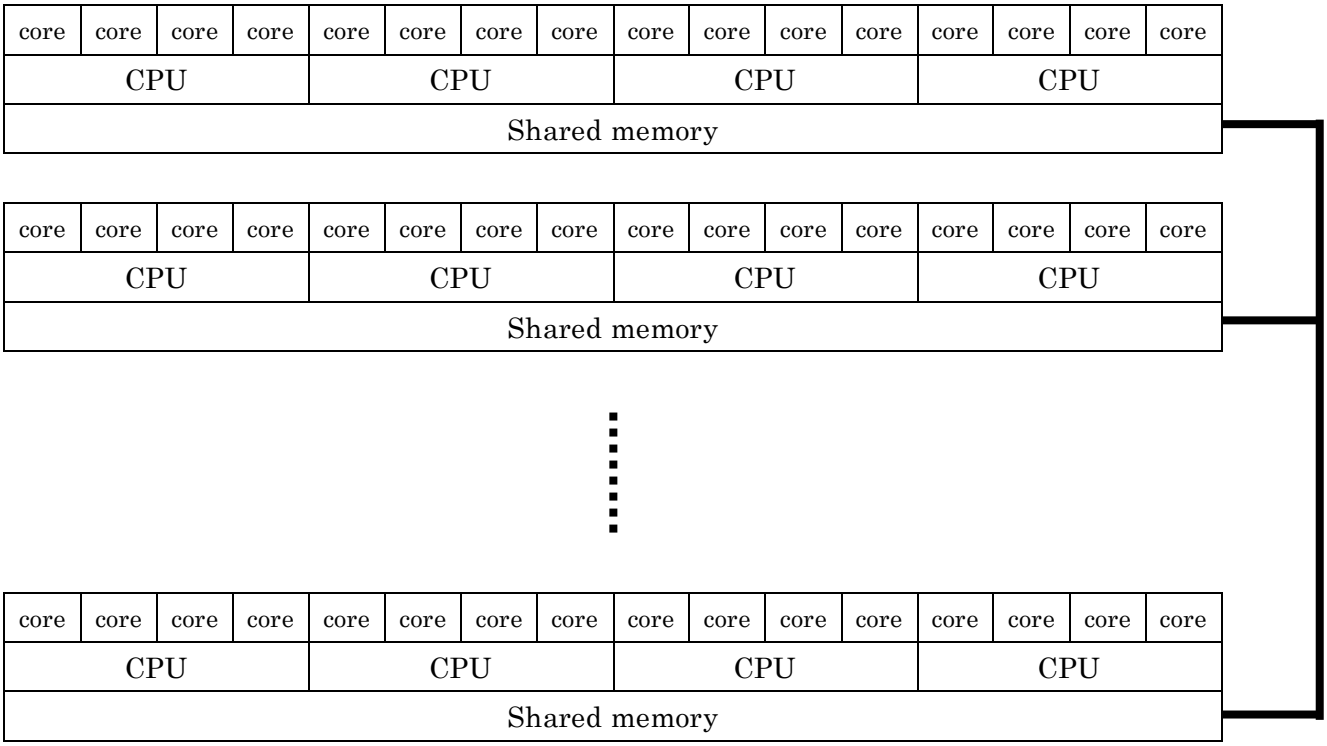


図 1.9.1　SMP cluster with multicore CPUs.

1.10　マルチグリッド法

1.10.1 概要

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

マルチグリッド法の各反復における手順の概要を以下に示す[16][17]：

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

1.10.2 計算手法

細かい格子レベル（fine grid level）において以下の線型方程式を解くことを考える：

$$\mathbf{K}_F \mathbf{u}_F = \mathbf{f}$$

ここで、$\mathbf{K}_C$ を粗い格子レベル（coarse grid level）における係数マトリックスとすると、粗い格子レベルにおける補正は以下のように記述される：

$$\mathbf{u}_F^{(i+1)} = \mathbf{u}_F^{(i)} + \mathbf{R}_{C\Rightarrow F}\mathbf{K}_C^{-1}\mathbf{R}_{F\Rightarrow C}(\mathbf{f} - \mathbf{K}_F\mathbf{u}_F^{(i)})$$

ここで、

$\mathbf{R}_{C\Rightarrow F}$ 粗い格子から細かい格子への補間オペレータ（prolongation operator, 延長補間）
$\mathbf{R}_{F\Rightarrow C}$ 細かい格子から粗い格子への補間オペレータ（restriction operator, 制限補間）

このような補間計算によって、細かい格子で残差を計算し、それを粗い格子で補正し、その結果を細かい格子に補間して誤差を補正するというプロセスを構築できる。

なお、$\mathbf{R}_{C\Rightarrow F}$ と $\mathbf{R}_{F\Rightarrow C}$ には、一般に以下の関係がある：

$$\mathbf{R}_{C\Rightarrow F} = c\left(\mathbf{R}_{F\Rightarrow C}\right)^T$$

ここで、cは実数の定数である。

1. 線形化された方程式$\mathbf{K}_F\mathbf{u}_F = \mathbf{f}$を細かい格子状で緩和し、結果を$\mathbf{u}_F^{(i)} = S_F\left(K_F, f\right)$とする。

   演算子S_F（たとえばGauss-Seidel）は緩和演算子（smoothing operator）と呼ばれる。

2. 細かい格子状で残差$\mathbf{r}_F = \mathbf{f} - \mathbf{K}_F\mathbf{u}_F^{(i)}$を求める。

3. 制限補間オペレータ$\mathbf{R}_{F\Rightarrow C}$によって、細かい格子状での残差を粗い格子に補間する：$\mathbf{r_C} = \mathbf{R}_{F\Rightarrow C}\mathbf{r_F}$

4. 方程式$\mathbf{K}_C\Delta\mathbf{u}_C = \mathbf{r}_C$（修正量に関する方程式なので「修正方程式」と呼ぶ）を粗い格子で解く。

5. 粗い格子状での修正方程式の解$\Delta\mathbf{u}_C$から、延長補間オペレータ$\mathbf{R}_{C\Rightarrow F}$によって、細かい格子における修正量$\Delta\mathbf{u_F^{(i)}} = \mathbf{R}_{C\Rightarrow F}\Delta\mathbf{u_C}$を求める。

6. 細かい格子での解を修正量によって更新する：$\mathbf{u_F^{(i+1)}} = \mathbf{u_F^{(i)}} + \Delta\mathbf{u_F^{(i)}}$

以上のプロセスを残差が基準値以下になるまで繰り返す。

上記は2レベルの場合のプロセスであるが、これを、複数のメッシュ階層にわたり、再帰的に実施することにより、収束の加速が可能となる。その代表的な手法を図 1.10.1 および図 1.10.2 に示す。マルチグリッド法は、ソルバーとして、また、CG 法など、他の反復法ソルバーの前処理として利用することが可能である[17][18]。

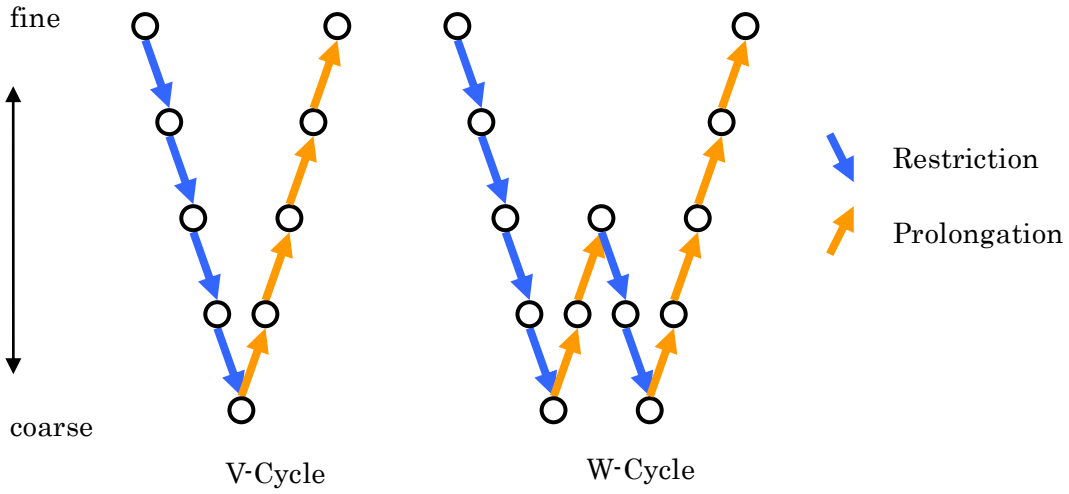


図 1.10.1 Multigrid Method: V-Cycle and W-Cycle

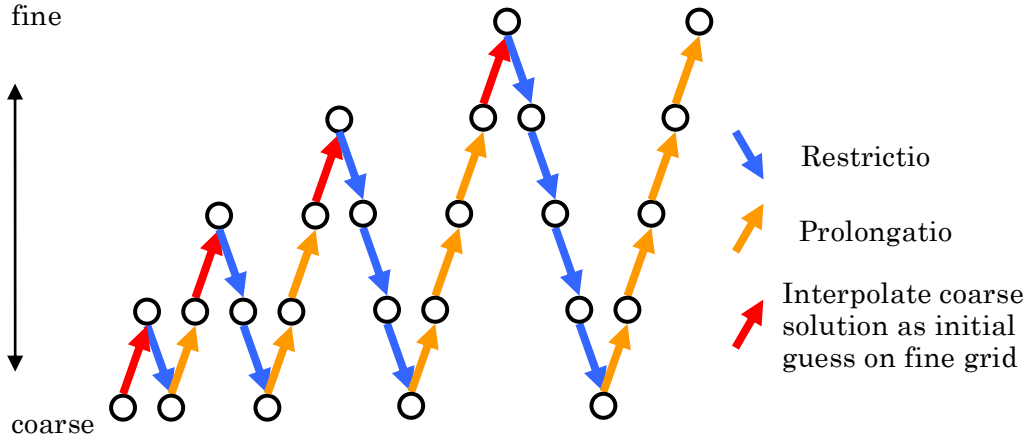


図 1.10.2 Full Multigrid (FMG)

1.10.3 マルチグリッド前処理

マルチグリッド法を CG 法などの反復解法の前処理として利用することも可能である。以下にその手法を図 1.10.3 に示す。なお、図中の MG はマルチグリッド前処理であり、図 1.10.4 にその詳細を示す。

$\mathbf{r}_0 = \mathbf{f} - \mathbf{K}\mathbf{u}_0$
for $k = 0..maxiter$
  $\mathbf{z}^k = \mathrm{MG}\left(\mathbf{K}, \mathbf{r}^k, \mathrm{initial\_}\mathbf{z}^k, \gamma, \mu_1, \mu_2\right)$
  $\rho_k = \left(\mathbf{r}_k, \mathbf{z}_k\right)$
  if $k \equiv 0$
    $\mathbf{p}_0 = \mathbf{r}_0$
  else
    $\beta_k = \dfrac{\rho_k}{\rho_{k-1}}$
    $\mathbf{p}_k = \mathbf{z}_k + \beta_k \mathbf{p}_{k-1}$
  endif
  $\mathbf{q}_k = \mathbf{K}\mathbf{p}_k$
  $\alpha_k = \dfrac{\rho_k}{\left(\mathbf{p}_k, \mathbf{q}_k\right)}$
  $\mathbf{u}_{k+1} = \mathbf{u}_k + \alpha_k \mathbf{p}_k$
  $\mathbf{r}_{k+1} = \mathbf{r}_k - \alpha_k \mathbf{q}_k$
  check convergence; continue if necessary
end

図 1.10.3　Multigrid preconditioned Conjugate Gradient method.

```
MG(K_level, f, u, γ, μ1, μ2)
{
  if level ≡ coarsest_level
    solve K_level u = f
  else
    u = pre_smoothing(K_level, f, u, μ1)
    r = restrict(f − K_level u)
    Δu_C = initial_Δu
    for n = 1.. γ
      Δu_C = MG(K_level−1, r, Δu_C, γ, μ1, μ2)
    end
    u = u + prolongate(Δu_C)
    u = post_smoothing(K_level, f, u, μ2)
  endif
  return u
}
```

図 1.10.4　Multigrid preconditioner

1.11 参考文献

(1) R. Barrett, M. Berry, T.F. Chan, J.W. Demmel, J. Donato, J.J. Dongarra, V. Eijkhout, R. Pozo, C. Romine and H. van der Horst: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, SIAM, (1994).

(2) J.J. Dongarra, I. Duff, D.C. Sorensen and H.A. van der Vorst: Numerical Linear Algebra for High-Performance Computers, SIAM, (1998).

(3) K. Garatani, H. Nakamura, H. Okuda and G. Yagawa: GeoFEM: High Performance Parallel FEM for Solid Earth, Lecture Notes in Computer Science No.1593, pp.132-140, (1999).

(4) D. Hysom and A. Pothen: Efficient Parallel Computaion of ILU(k) Preconditioners, NASA/CR-2000-210210, ICASE Report No.2000-23, (2000).

(5) K. Nakajima, H. Nakamura and T. Tanahashi: Parallel Iterative Solvers with Localized ILU Preconditioning, Lecture Notes in Computer Science 1225, pp.342-350, (1997).

(6) K. Nakajima and H. Okuda: Parallel Iterative Solvers with Localized ILU Preconditioning for Unstructured Grids, IMACS Series in Computational and Applied Mathematics Volume 5: Iterative Methods in Scientific Computation IV, p.85-98, (1999).

(7) K. Nakajima and H. Okuda: Parallel Iterative Solvers with Localized ILU Preconditioning for Unstructured Grids on Workstation Cluster, International Journal for Computational Fluid Dynamics, Vol.12, pp.315-322, (1999).

(8) Nakajima, K.: "OpenMP/MPI Hybrid vs. Flat MPI on the Earth Simulator: Parallel Iterative Solvers for Finite Element Method", International Workshop on OpenMP: Experiences and Implementations (WOMPEI 2003), Tokyo, Japan, Lecture Notes in Computer Science 2858, pp.486-499, Springer, (2003).

(9) K.Nakajima, " Parallel Iterative Solvers of GeoFEM with Selective Blocking Preconditioning for Nonlinear Contact Problems on the Earth Simulator", SC2003 Technical Session, Phoenix, Arizona, (2003).

(10) Y. Saad: Iterative Methods for Sparse Linear Systems, PWS Publishing Company, (1996).

(11) H.D. Simon: Partitioning of unstructured problems for parallel processing, Computing Systems in Engineering, Vol.2, pp.135-148, (1991).

(12) B. Smith, P. Bj ▪ rstad and W. Gropp: Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations, Cambridge University Press, (1996).

(13) LAPACK User's Guide

(14) METIS Web Site: http://www-users.cs.umn.edu/~karypis/metis/

(15) MPI (Message Passing Interface) Forum Web Site: http://www.mpi-forum.org/
(16) W. L. Briggs, V. E. Henson and S. F. McCormick, A Multigrid Tutorial, second edition, Society for Industrial and Applied Mathematics, 2000.
(17) U. Trottenberg, C. Oosterlee and A. Schuller, Multigrid, Academic Press, 2000.
(18) Osamu Tatebe and Yoshio Oyanagi: Efficient Implementation of the Multigrid Preconditioned Conjugate Gradient Method on Distributed Memory Machines, Proceedings of Supercomputing '94, IEEE Computer Society, pp.194-203, (1994).

2. 並列可視化機能

2.1 はじめに

数値シミュレーションシステムは、通常以下の 3 つの構成要素からなる。

① メッシュ生成
② 基礎方程式の求解
③ 結果の可視化

最近のコンピュータハードウェアやソフトウェアの急速な発展により、詳細な解析を目的として②の求解部で大規模な計算が可能になってきている。それゆえグラフィクスによる結果表示なしでは現象の解析は非常に困難である。そこで強力な可視化プログラムが求められている。

現状の商用可視化プログラムは、比較的小規模なデータには有効に働くが、残念ながら大きなデータについては、メモリ量の制限や並列化が不十分などの理由により十分機能しないことがある。そこで今回、HEC-MW[1]プロジェクトにおいて可視化プログラムを重要な要素と位置づけ、並列可視化ライブラリの開発を行うことになった。これは大規模な解析計算の結果を見やすく効果的に図化し、それにより研究者や技術者が解析結果に潜む物理現象の把握の一助となることを目的とするものである。

本プロジェクトで開発された並列可視化ライブラリは以下の 6 つの特徴を有する。

(1)同一の並列計算機内で解析計算と可視化処理を同時並行で行える。
(2)グラフィックボードなど表示システムに依存しない。
(3)スカラ、ベクトル及びテンソルなどのデータが持つ意味を詳細に表現するため、種々のテクニックを駆使している。
(4)可視化自体も高速処理が行えるよう、さまざまな並列計算機（PC クラスタから地球シミュレータまで）に対して並列化を含む最適化を行っている。
(5)すべてのモジュールは、複雑で大規模な非構造格子に対応している。
(6)図の質的向上をねらった手法を取り入れている。

2.2　解析計算との並行処理

並列計算機を用いる場合は通常データ規模は非常に大きく数ペタバイトにまで及ぶことがありうる。そうした大規模なデータを他のコンピュータに転送したり、ディスク間に保存したりするのは効率が悪いものである。更に、可視化のためにも大きなメモリが必要となりクライアントマシンでそれを行うのも困難である。そこで計算結果を即座に同じ並列計算機上で可視化処理してしまい、比較的小規模になった可視化データとして保持するようにした。この時下記の２つの実行モードが可能である。

(1)サーバー・クライアントモード

計算サーバーにてユーザーは可視化の内容（等値線、流線やそのパラメーター）を指定する。それに応じた幾何情報データをファイル出力し、実際の可視化はクライアントマシンにて行う。その際ユーザーは AVS などのプログラムを用いて、その段階でシェーディングや視点などのパラメータを指定できる。

この方法をとることにより当初巨大なデータもかなり縮小されるのでメモリ制限が緩和できる。クライアントマシンにて自由性のある可視化ができるなどのメリットがある。

(2)サーバーモード

このモードは、①と違って可視化処理をすべて計算サーバーで行ってしまうものである。時系列の解析結果データを描画したりアニメーションしたりした可視化データを直接生成し保存する。この可視化データは計算データサイズには依存せず一定である。このモードで生成された可視化データは一切の幾何情報を含んでいないため、単に視点の変更を行うためだけでも再計算が必要である。

極めて大規模なデータを扱うときにこのモードを用いると便利である。

図 2.2.1 に HEC-MW 可視化ライブラリの構成を示す。計算モジュールはメッシュデータと制御パラメータ(計算および可視化両方の)を読み込む。計算を始めから１タイムステップ終了した段階で可視化モジュールがスタートする。おのおののタイムステップ毎にユーザーは、等値線、流線、ボリュームレンダリングなどの描画内容を指定できる。更に描画のパラメータのそれぞれ指定できる。例えば、毎回視点を変えたボリュームレンダリングや異なる値での等値線図などを作成できる。多くのメッシュ点数のデータについて多くののタイムステップがあるデータについて数ヶ月にも及ぶ計算が必要なことがある。その際、計算結果のデータも巨大でそのままでは保存できない場合など、可視化データとして小規模なデータとして保存できることは便利である。更に、計算の途中で解析の様子を描画図で検証し、もし不具合（可視化のパラメータが好ましくないなどを含む）があれば計算を中断しパラメータ変更の後リスタートすることができる。こうして徒に計算資源を浪費することを防げる。

サーバー・クライアントモードにて小規模な試計算を行った後、サーバーモードで本来の目的の計算を行うなどの方法は有効である。

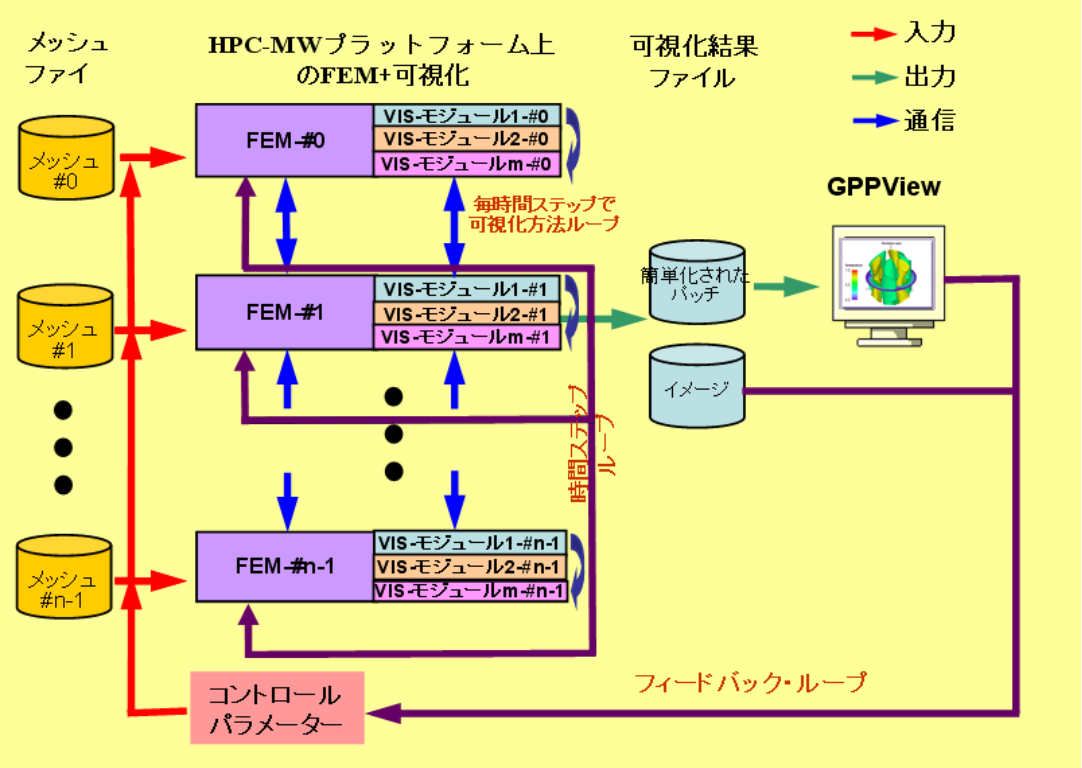


図 2.2.1　HEC-MW の並列可視化ライブラリの構成

2.3　並列可視化技術について

HEC-MW 並列可視化ライブラリでは、スカラ、ベクトル、テンソル表示のために様々なテクニックの導入を予定している。今回のリリース版では、スカラデータの表示部が主なものになっている。以下では、初めにリリース版の中の並列ボリュームレンダリングと並列サーフェスレンダリングを紹介する。続いて現在開発中のモジュールについて述べる。

2.3.1 リリース版

(1) 並列サーフェスレンダリング(Parallel Surface Rendering:PSR)

サーフェスレンダリングは 3 次元可視化において非常に重要な技術である。これにより立体表面のデータ分布を詳細かつ明確に表示することができる。以下に 3 つの例を示す。

a. 境界表面の表示

境界表面の表示は自動的に行われる。しかし複数個の境界がある時は、それぞれに名前をつけることにより、以降名前指定で表示できる。図 2.3.1 に境界表面の例を示す。

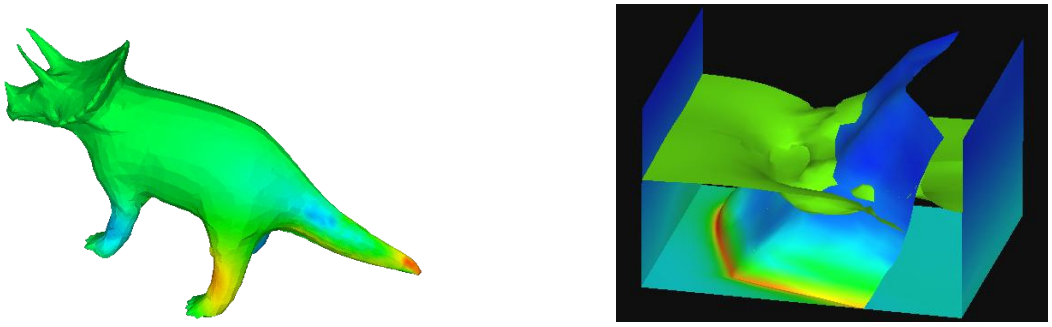


図 2.3.1　境界表面の表示例 (データ提供：東京大学　奥田氏，RIST　飯塚氏)

b. 等値面

データの値の分布を見るにはこの等値面表示を用いるのが便利である。本プログラムでは複数の等値面を並列に処理できる。図 2.3.2 は 6 つの等値面と平面の切り口でのデータ分布を示している。

c. 任意曲面の切り口

複数の任意曲面の切り口でのデータ分布を表示する。曲面は方程式で与えられる。図 2.3.3 は西南日本の地震解析の例である。

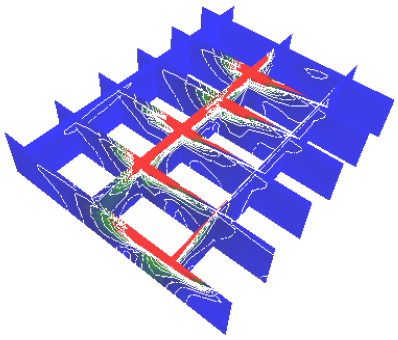


図 2.3.2　6 つの等値面と平面切り口でのデータ分布（データ提供：松井氏）

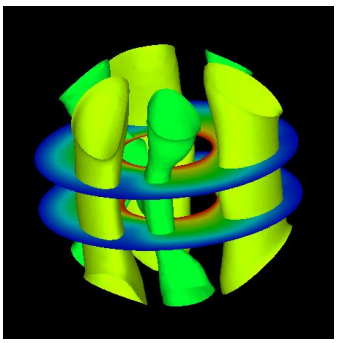


図 2.3.3　西南日本の地震解析，応力分布図 (データ提供：RIST 飯塚氏)

(2) 並列ボリュームレンダリング (Parallel Volume Rendering:PVR)

ボリュームレンダリングは 3 次元可視化において非常に強力な手法である。これにより 3 次元立体の内部までの状況が観測できる[2]。ボリュームレンダリングは、時間的にもデータ容量的にも高コストであるため並列化の導入が極めて重要である。並列ボリュームレンダリング(PVR)は、並列マシンのタイプ、メッシュタイプ、射影の方法などによって分類できる[3]。効率アップが実現している例は参考文献[4-7]。いずれにしてもボリュームレンダリングのプログラムを設計する時には、複雑で大規模なデータを想定しなければならない。PVR ではスーパーボクセルによる領域分割を行いそれぞれで処理を行った後、描画データを統合して 1 つの描画データとしている。

スーパーボクセルによる領域分割

HEC-MW で扱うメッシュは、複雑で規模が大きいことが多い。しかも扱う要素は非構造格子で、四面体、六面体、プリズムなどが混在し、しかも多層構造になることもある。こうした複雑なメッシュ構成は描画データの作成処理にとおて非常にやっかなもので、レイトレーシングによるボリュームレンダリングなどでは処理時間が大きくなってしまう。これの対策として並列 BSP という手法が研究開発されている[5][8]。しかしこの手法は中間データを用いるので大きなメモリやディスクスペースが必要となる。そこで極めて大きなデータに対しては、並列処理が効果的となる（高速化の意味だけでなくメモリ、ディスクスペースを大きく取れるため)。この時領域バランスよく分割するためにスーパーボクセルの手法を取り入れている。図 2.3.4 参照。

スーパーボクセルの定義はデータの座標値から自動生成できるが、ユーザー指定により削除や細分も可能である。

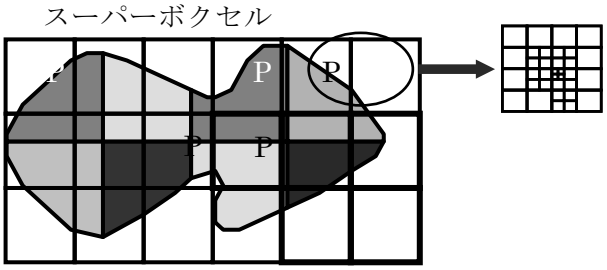


図 2.3.4　Supervoxel-based partition in the PVR module

- 分割した描画データの生成

スーパーボクセルによる領域分割後、各プロセスには均等にスーパーボクセルが割り振られる。それぞれのプロセスにおいて通常のレイトレーシング処理を行う。すなわち対象の描画対象となるピクセルを求める。次に視点から各ピクセルに光を当て、その光線とボクセルとの交点を求め、その点での対象データの値を求めカラーマップへの変換を行う。この処理を順次奥の方に進める。こうしてこのスーパーボクセルに対する分割描画データを作成する。

- 分割描画データの統合

それぞれのスーパーボクセルについてピクセル毎の処理をすべて行いその不透明度も合わせて記録する。こうして各プロセスにある分割描画データをその位置関係を考慮しながら統合する。統合に際してはやはり手前から奥のほうにレイトレーシングを進め書くピクセルでのカラーマップを求めていく。そして最終的に統合した描画データを得る。

図 2.3.5 に PVR によるボリュームレンダリングの例を示す。

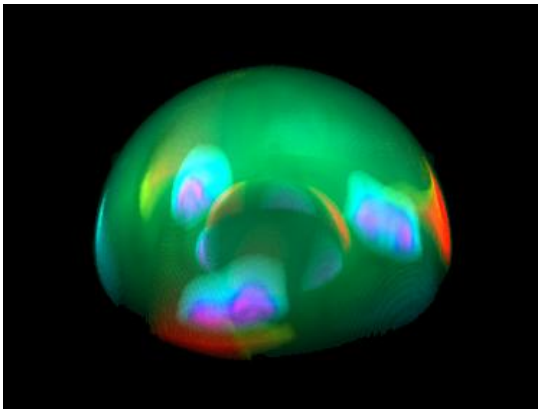


図 2.3.5　半球の熱流体シミュレーションの PVR による表示（データ提供：松井氏）

2.3.2 開発中のモジュール

並列粒子追跡、及び並列ストリームラインモジュールを HEC-MW にて開発中する。これにより大規模データのベクトル場などの表示が可能となる。並列粒子追跡法によって粒子の挙動を様々なスタイルで表示できる。粒子の初期状態、指定の時刻の状態、または特定の平面からの流れ出しの様子などを表示できる。図 2.3.6 は、地下水流れの様子を粒子追跡法にて表示したものである。ストリームラインは 3 つの表示法がある。

従来の方法
太さをもったストリームラインレンダリングをしたもの[9]。図 2.3.7 に例を示す
ストリームラインの太さでその量を表現したもの

テクスチャベース手法はベクトル場の表示には強力な手法である[10]. しかし時として 3 次元の密なベクトル場に適用するとうまく動作しないことがある。そこでこの方法に重要度マップという概念を入れ改良を加えた[11]。ベクトル場での可視化では、渦のような流れの特徴となる領域を調べることがしばしば重要となりので、重要度マップにより選択的に強調描画することが効果的な可視化と考えられる。重要度マップの算定においては流体位相解析技法を用いている。更にこの位相解析技法により、切り口面での等値線の強調表示や不連続のある物理量の表示などが行える。現状 Zockler, et al. [9]のストリームライン照明モデルを用いている。これにより 3 次元流線畳み込み法の適用の場を増やしている。

図 2.3.8 に竜巻の解析結果の例を示すが、これにより多くの解析情報を得ることができる。

図 2.3.6　並列粒子探索法による地下水流の表示
(データ提供：RIST 中島氏)

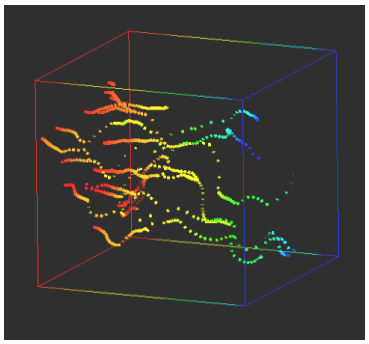


図 2.3.7　照明つきストリームラインによる立方体内の流れ表示
(データ提供：松井氏)

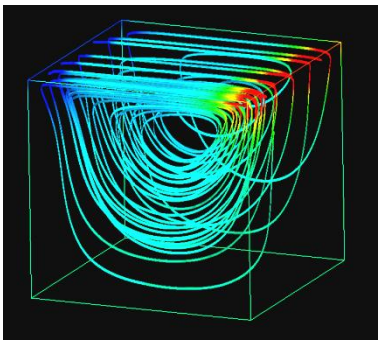


図 2.3.8　重要度マップを考慮した流線畳み込み法による竜巻解析の表示
(データ提供： オハイオ大 Roger Crawfis 氏）.

テンソル場の表示にはハイパーストリームライン技法を用いている[12]。これにより 3 次元空間の 2 次テンソル場をパスに沿って表示することができる。それと同時に 9 つのコンポーネント（3 つの固有ベクトル）も表示できる。あるスタートポイントから、最大主値に属する固有ベクトルに沿ったベクトルのトラジェクトリを描く。各ポイントでは楕円形の切り口であるが、その長径と短径が他の 2 つ主値の大きさと方向を示している。また色は最大主値の大きさを示している。図 2.3.9

に応力テンソルの例を示す。

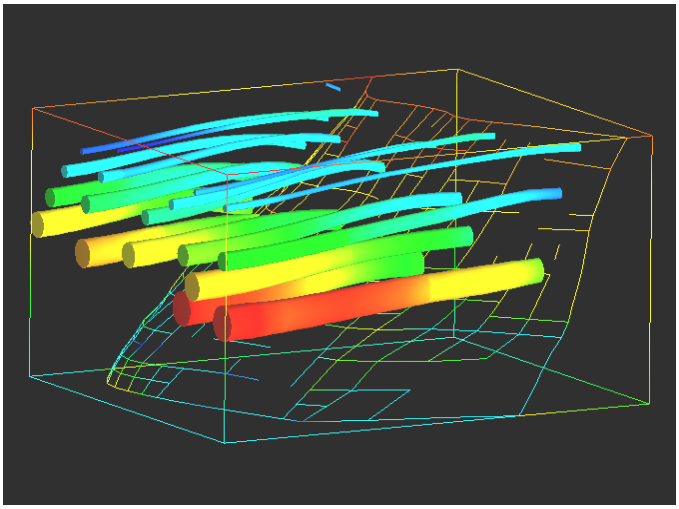


図 2.3.9　並列ハイパーストリーム手法による日本列島周辺の地震解析の表示

2.4　計算機依存の最適化

一口に並列計算機といってもそのハードウェアアーキテクチャーは、多種多様である。可視化においても高速化の手法は並列計算機のタイプに依存して手法を選択する必要がある。ある並列計算機にて効果的であっても他のタイプの計算機で効果的であるとは限らない。そこで HEC-MW プロジェクトでは可視化ライブラリについても計算機のタイプ毎の最適化コードの開発を行っている。以下に PVR を例に挙げて、各種のマシンにおける最適化手法を述べる。

2.4.1 スカラ計算機とベクトル計算機の違い

ベクトル計算機は通常巣から計算機より高速である。しかしベクトル化率が性能に大きく影響する。コンパイラによる自動ベクトル化機能だけでは不十分で、往々にしてベクトル化に注意を払ってコード作成をしなければならない。

(1) データ構造 - 8 分木と一次元配列の比較
複雑なメッシュデータに対して PVR 処理を高速で行うのは困難ものである。その理由は、

① 隣接情報を事前に作成しておく必要がある
② 階層構造でデータを扱うのは効果的であるが、非構造格子について階層構造を作ることは難しい。
③ 不規則な形状では PVR の合成処理が複雑になる
④ 点の同定が構造化メッシュに比べて非常に困難である

そこで効率よい処理のために直交メッシュへの変換が必要である。しかしながら、同等の精度を維持してこの処理を行うと随分大きなメモリスペースが必要となる。またボリュームレンダリングの処理に時間がかかってしまう。そこで通常は 8 分木手法による階層型のデータ構造を採用する。これにより、トータルのボクセルの数を減らしても、内部情報の詳しさは維持できる(図 2.4.1(a))。

高速化のために更に Branch-on-need Octree 手法を採用している（BONO）[13]。従来型の 8 分木法は基本的にバイセクション法である。そのため空のボクセルが多数生成されてしまうことがある。反面、BONO では分割が必要になった段階で行うので、徒に分割がなされることはない。加えて BONO では空のボクセルや透明なボクセルを効率的にスキップしながら必要な点での最大値と最小値を求めることができる。この方法においてはまずそれぞれのスーパーボクセルを直交メッシュの中で再同定を行う。続けて隣接するボクセルでの値の差を見て、一定以上大きければボクセルを 2 分割する作業を再帰的に行う。スカラ計算機においてはこの手法が効果的である。

一方、ベクトル計算機では BONO はベクトル化されないため効率的ではない。そのため従来型の直交メッシュへの分割を行い 1 次元配列での処理となるようにしている。これによりベクトル化が実現できる（図 2.4.1(b) ）。

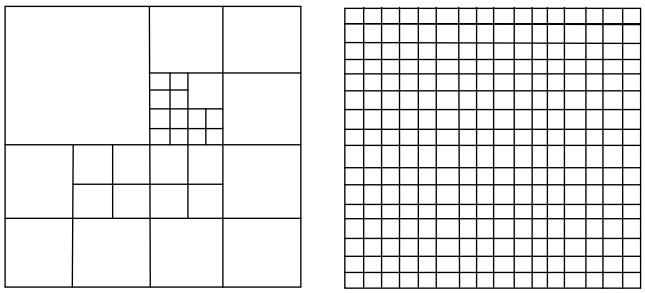


図 2.4.1　スカラ計算機、ベクトル計算機での違い

(a)スカラ計算機における 8 分木探索
(b)ベクトル計算機における独立探索

2.5 参考文献

(1) http://www.fsis.iis.u-tokyo.ac.jp/

(2) Levoy, M.: "Display of surfaces from volume data", *IEEE CG&A*, Vol. 8, No. 3, pp.29-37, (1988).

(3) Wittenbrink, C. M.: "Survey of parallel volume rendering algorithms", *Proceedings of International Conference on Parallel distributed Processing Techniques and Applications*, Las Vegas, Nevada, pp. 1329-1336, (1998).

(4) Yagel, R.: "Towards real time volume rendering", In *Proceedings of GRAPHICON*, Saint-Petersburg, Russia, volume 1, pp. 230-241, (1996).

(5) Ma, K. L., Painter, J., Hansen, C. D. and M. F. Krogh: "Parallel volume rendering using binary-swap compositing", *IEEE CG&A*, vol. 14, No. 4, pp. 59-67, (1994).

(6) Wittenbrink, C. M. and Somani, A. K.: "Time and space optimal parallel volume rendering using permutation warping", *Journal of Parallel and Distributed Computing*, vol. 46, No. 2, 148-164, (1997).

(7) Silva, C.: Parallel volume rendering of irregular Grids, *Ph.D. thesis*, State University of New York at Stony Brook, (1996).

(8) Ramakrishnan, C.R. and Silva, C: "Optimal processor allocation for sort-last compositing under BSP-tree ordering", *SPIE Electronic Imaging*, pp. 73-80, (1999).

(9) Zoeckler, M., Stalling, D., and Hege, H.-C.: "Interactive visualization of 3D vector fields using illuminated streamlines", In *Proceedings of IEEE Visualization'96*, pp. 107-113, (1996).

(10) Cabral, B. and Leedom, C.: "Image vector field using line integral convolution", *Computer Graphics Proceedings*, ACM SIGGRAPH, pp. 263-272, (1993).

(11) Chen, L., Fujishiro, I. and Suzuki, Y.: "Comprehensible volume LIC rendering based on 3D significance map", *Proceedings of SPIE Conference on Visualization and Data Analysis 2002* (San Jose) pp. 142-153, (2002).

(12) Delmarcelle, T. and L. Hesselink: "Visualizing second-order tensor fields with hyper-streamlines", *IEEE CG&A*, 13(4), 25-33, (1993).

(13) Wilhelms, J. and Gelder, von A.: "Octree for faster isosurface generation", *ACM Trans. on Graphics*, Vol. 11, No. 3, pp. 201-227, (1992).

(14) Rabenseifner, R.: "Communication Bandwidth of Parallel Programming Models on Hybrid Architectures", Lecture Notes in Computer Science 2327, pp.401-412, (2002)

(15) OpenMP Web Site: http//www.openmp.org

(16) Nakajima, K. and Okuda, H.: "Parallel iterative solvers for unstructured grids using directive/MPI hybrid programming model for GeoFEM platform on SMP cluster architectures", *Journal of Concurrency and Computation: Practice and Experience*, Vol.14, No.6-7, pp.411-430, (2002).

3. 領域分割ユーティリティ

3.1 はじめに

本ソフトウェアは、単一領域のメッシュデータを部分領域に領域分割し、並列有限要素法の計算に使用する分散メッシュデータを作成するユーティリティソフトウェアである。データ入力にHEC-MW ライブラリのデータ入力機能を用いており、同機能で読み込みが可能な単一領域のメッシュデータ「NASTRAN，ABAQUS，FEMAP Neutral または GeoFEM フォーマットメッシュデータ」「単一領域メッシュデータ」を任意の数の部分領域に領域分割する。また、HEC-MW ライブラリのデータ出力機能を用いて「分散メッシュデータ」の出力を行っており、HEC-MW を用いて開発したソフトウェアにおいては、本ソフトウェアにより作成した分散メッシュデータを用いて解析を行うことが可能である。

本ソフトウェアでは、単一領域メッシュの領域分割の実施の仕方として、節点単位で領域分割を行う「節点ベース分割」、および、要素単位で領域分割を行う「要素ベース分割」の 2 種類の領域分割タイプを実装している。領域分割手法としては、座標値を基準として部分領域への分割を行う RCB 法を用いた領域分割手法を実装しており、また、グラフ理論を領域分割に応用した METIS のライブラリへのインターフェイスを実装することで、同ライブラリにおける kMETIS と pMETIS を用いた領域分割も可能にしている。さらに、節点ベース分割においては、部分領域間のオーバーラップ深さを任意に設定することが可能であり、また、MicroAVS などで利用可能な UCD ファイルフォーマットで領域分割イメージを表示するためのファイルを出力機能も実装されている。

以下、各機能について述べる。

3.2 領域分割手法

本ソフトウェアにおいては、以下の領域分割手法が利用可能である。

- RCB

Recursive Coordinate Bisection の略であり、座標値の大小を基準に領域分割を行う方法である。高速で安定した手法であるが、部分領域数が 2^n に限定される。単純な形状においては、有効な手法である。

- MeTiS

高速、かつ安定した手法であり、複雑形状においても良好な領域分割が得られることから、世界中で広く利用されているオープンソースのフリーソフトウェアである。本ソフトウェアにおいては、この MeTiS のライブラリへのインターフェイスを備えており、pMeTiS と kMeTiS を直接利用して領域分割を行うことが可能である。但し、MeTiS が予め利用環境にインストールされていることが前提となる。MeTiS は以下の URL よりダウンロード可能である。

http://www-users.cs.umn.edu/~karypis/metis/index.html

3.3 領域分割タイプ

本ソフトウェアにおいては、単一領域メッシュの部分領域への分割の仕方として、以下の 2 種類の領域分割タイプに対応している。

- 節点単位での領域分割（節点ベース分割）

図 3.3.1 に示すように、節点単位で領域分割を行う方法であり、この場合、全ての節点に対して唯一の所属部分領域が決定し、隣接する部分領域間でオーバーラップする要素が生じる。ゆえに、節点ベース分割においては、各部分領域は、以下に示す節点および要素の情報を所持する（図 3.3.2）。

- その部分領域に属している節点（内部節点）
- 内部節点を含む要素
- 内部節点を含む要素を構成する節点

また、隣接する部分領域間での通信に関する情報（通信テーブル）として、以下に示す情報を所持する。

- 輸入節点：部分領域内の節点のうち、他の部分領域に属する節点
- 輸出節点：他の部分領域の輸入節点となっている内部節点
- 共有要素：他の部分領域と共有する要素

図 3.3.1 の第 2 番の部分領域における輸入節点、輸出節点、および共有要素を図 3.3.3、図 3.3.4、図 3.3.5 に示す。

- 要素単位での領域分割（要素ベース分割）

図 3.3.6 に示すように、要素単位で領域分割を行う方法であり、この場合、全ての要素に対して唯一の所属部分領域が決定し、隣接する部分領域間でオーバーラップする節点が生じる。ゆえに、要素ベース分割においては、各部分領域は、以下に示す節点および要素の情報を所持する（図 3.3.7）。

- その部分領域に属している要素（内部要素）
- 内部要素を構成する節点
- 内部要素を構成する節点を含む要素

また、隣接する部分領域間での通信に関する情報（通信テーブル）として、以下に示す情報を所持する。

- 輸入要素：部分領域内の要素のうち、他の部分領域に属する要素
- 輸出要素：他の部分領域の輸入要素となっている内部要素

➢　共有節点：他の部分領域と共有する節点

図 3.3.7 の第 2 番の部分領域における輸入要素、輸出要素、および共有節点を図 3.3.8、図 3.3.9、図 3.3.10 に示す。

どちらの領域分割タイプにおいても、その通信テーブルは領域分割ユーティリティが自動的に作成し、分散メッシュデータに書き込むため、ユーザーは通信に関して考慮する必要はない。

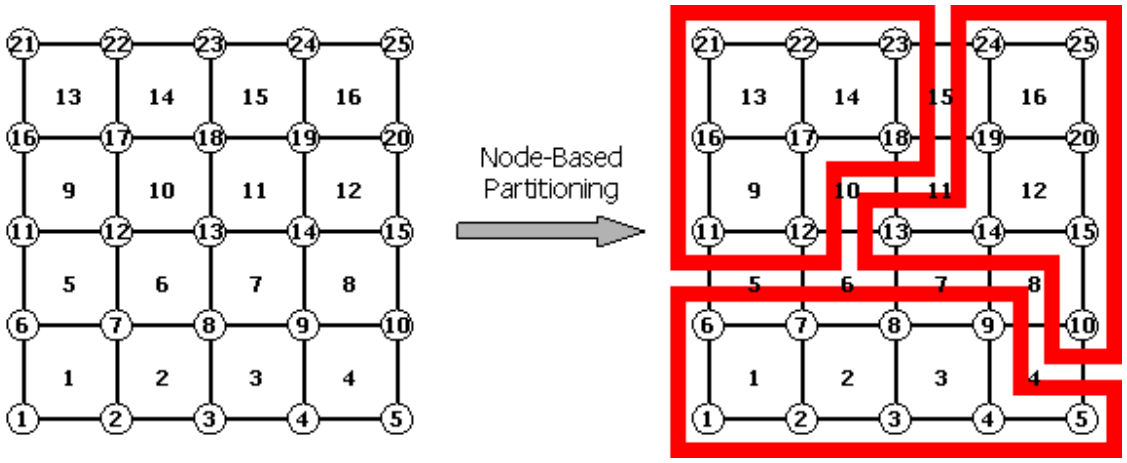


図 3.3.1　節点単位での領域分割

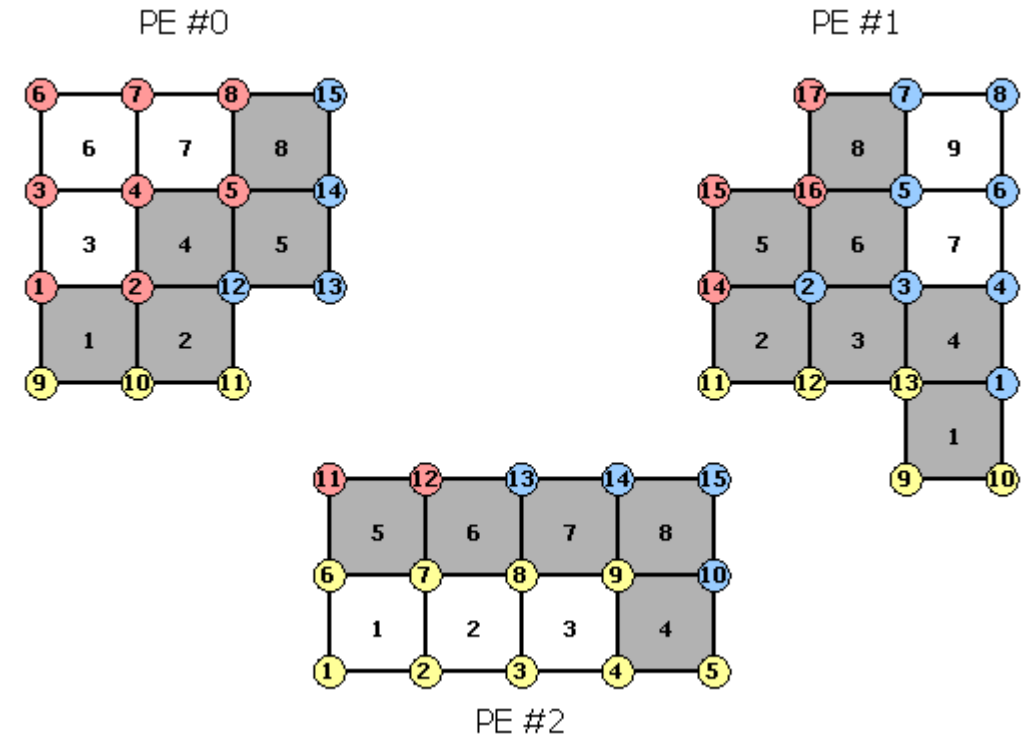


図 3.3.2　各部分領域が保持する節点および要素（節点ベース分割）

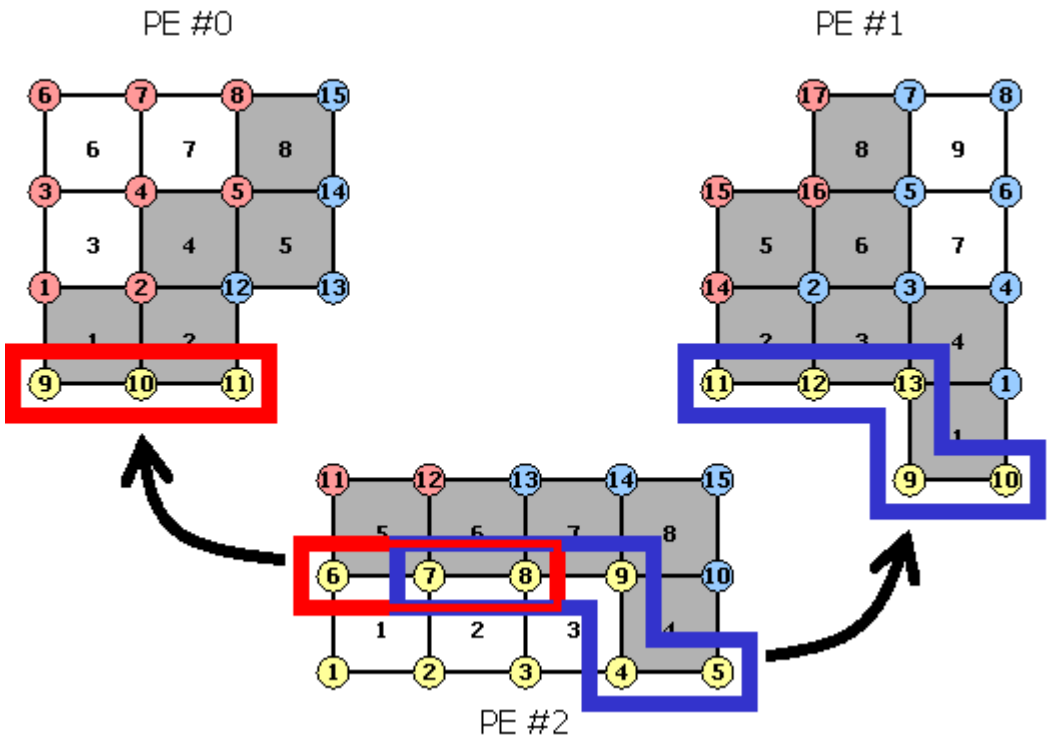


図 3.3.3　第 2 番の部分領域における輸入節点

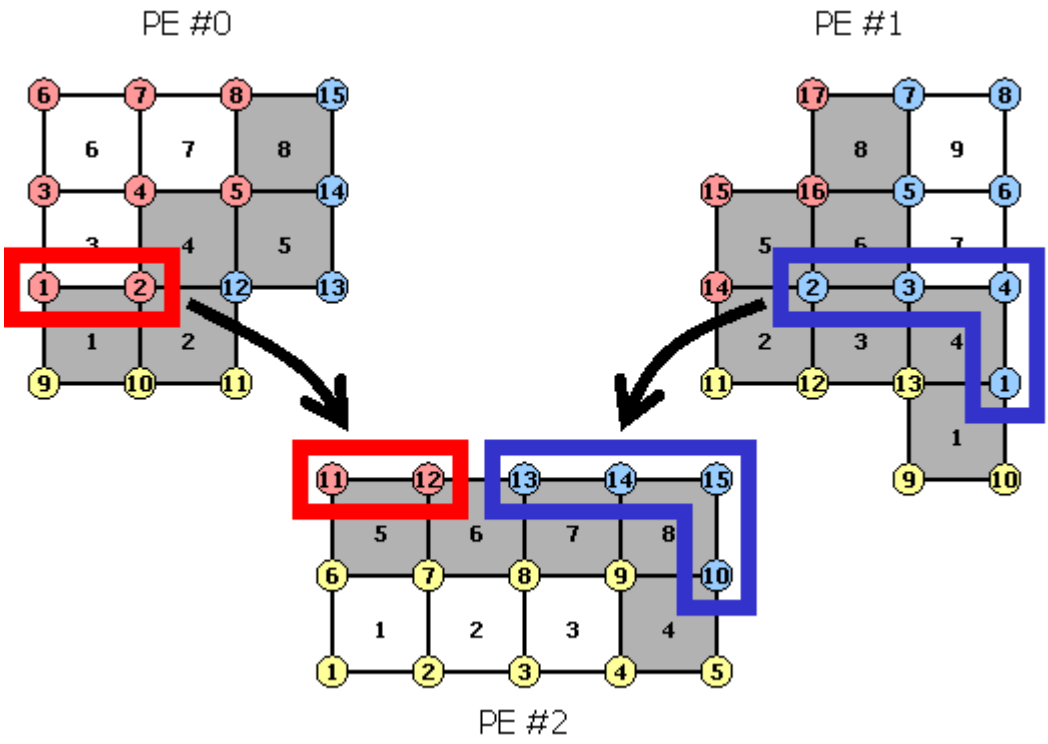


図 3.3.4　第 2 番の部分領域における輸出節点

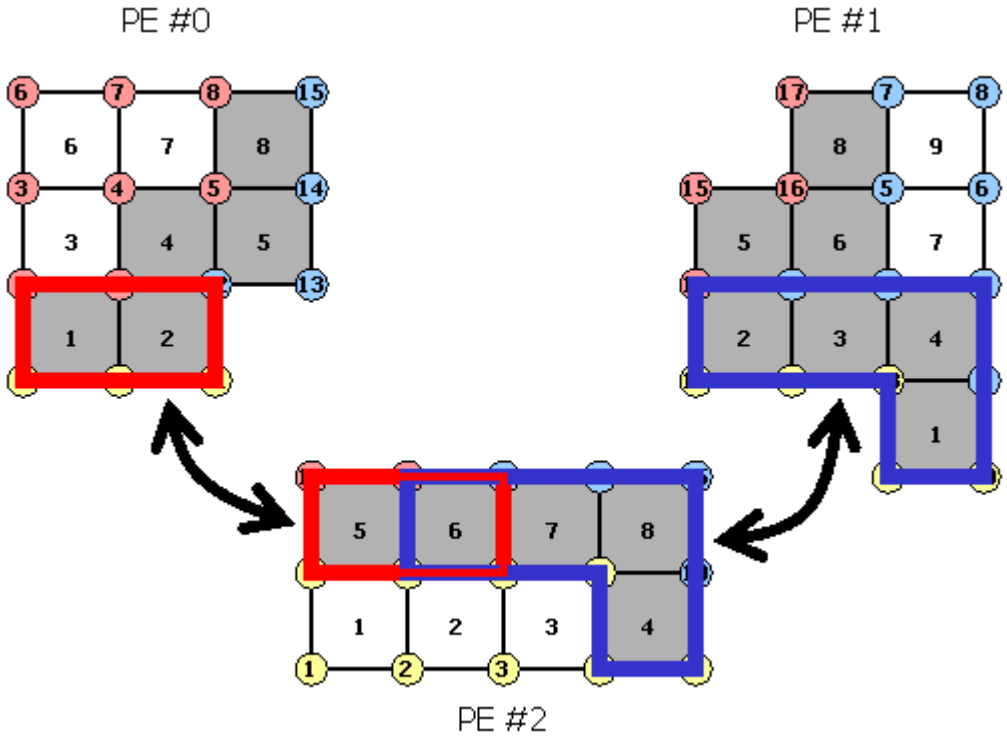


図 3.3.5　第 2 番の部分領域における共有要素

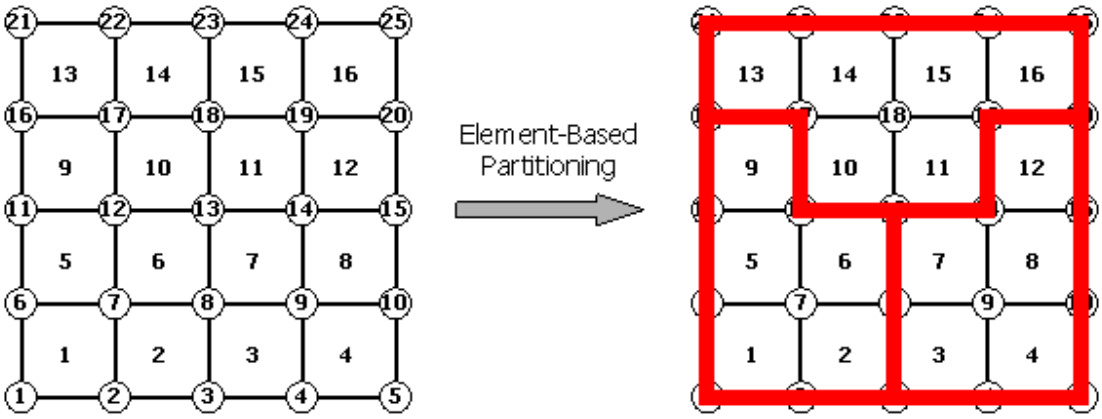


図 3.3.6　要素単位での領域分割

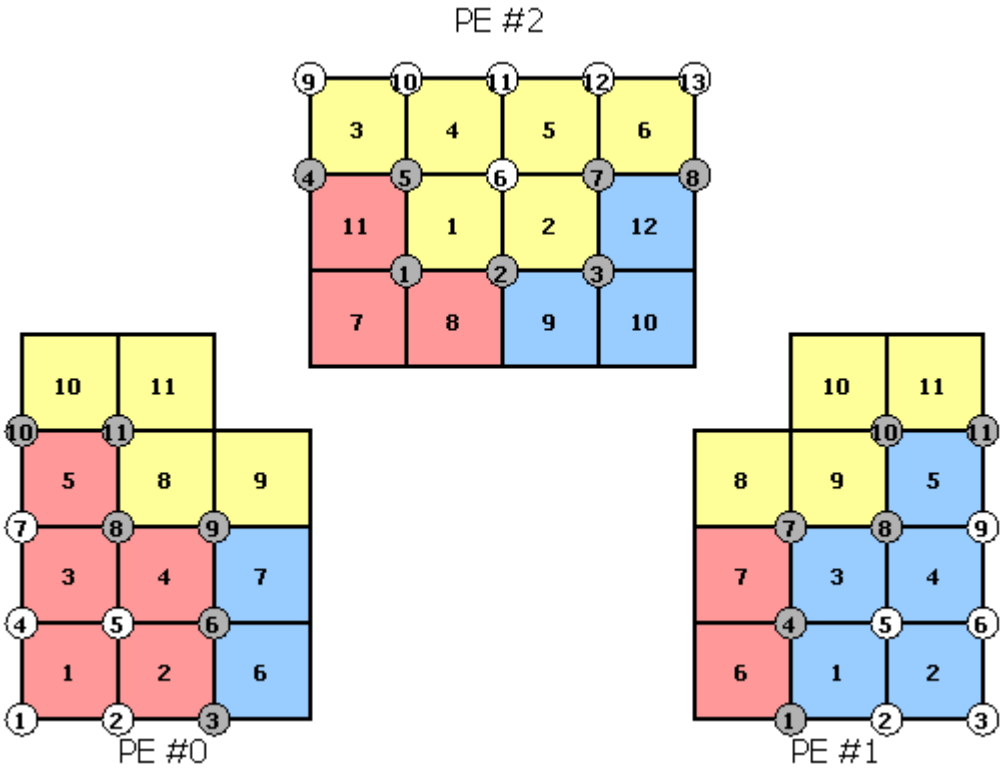


図 3.3.7　各部分領域が保持する節点および要素（要素ベース分割）

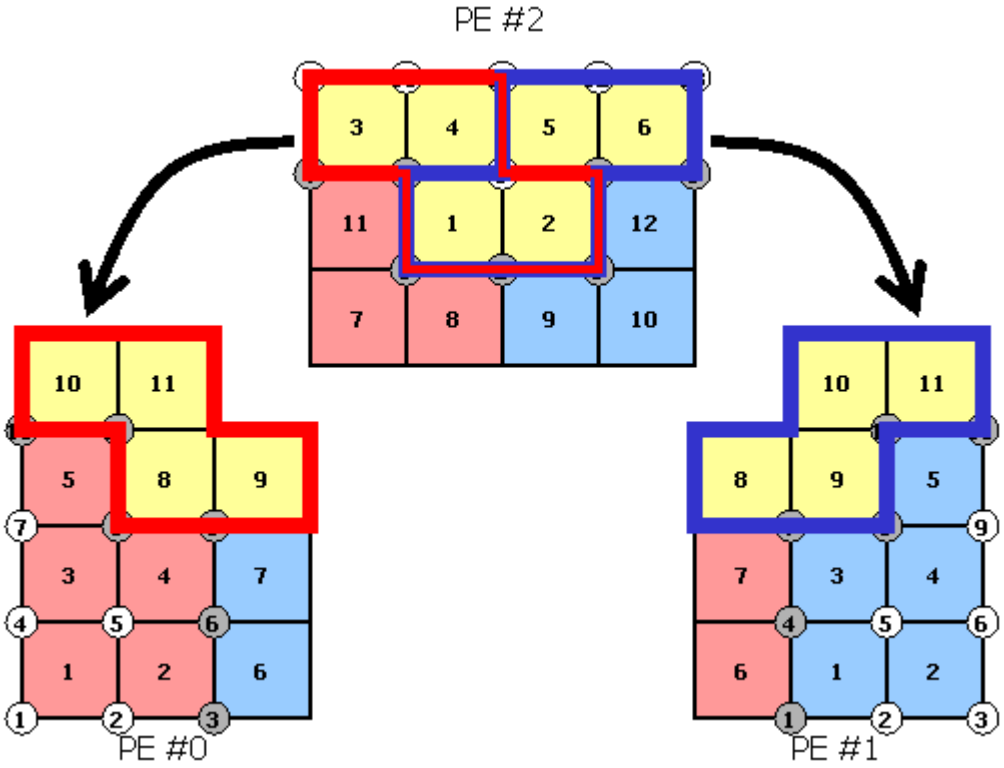


図 3.3.8　第 2 番の部分領域における輸入要素

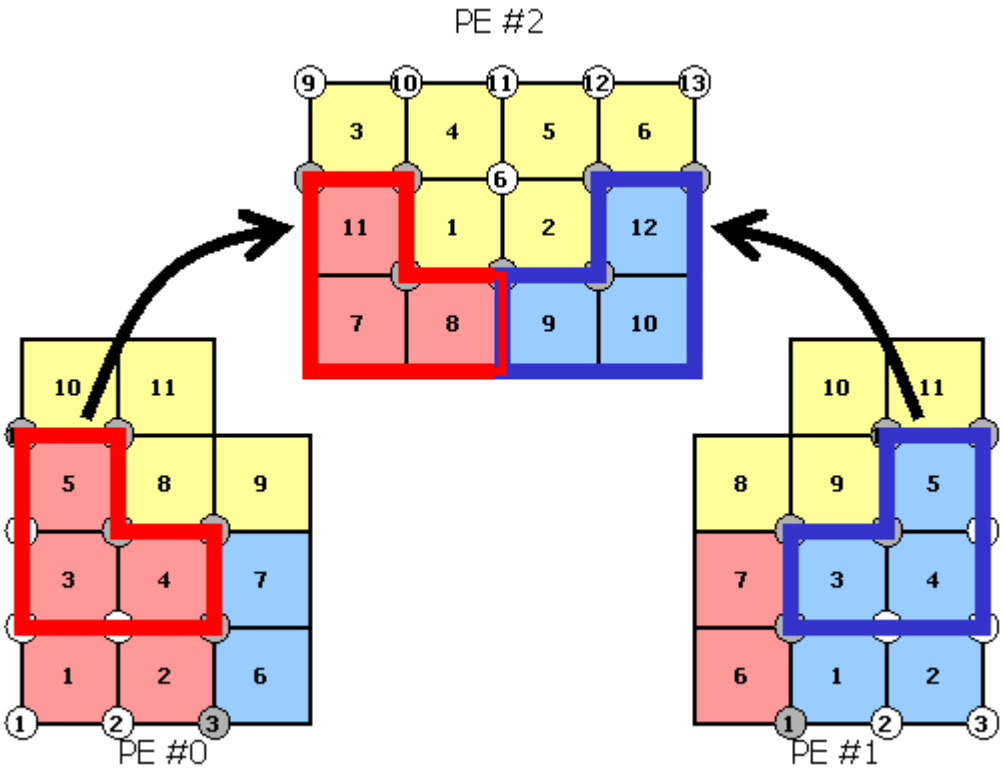


図 3.3.9　第 2 番の部分領域における輸出要素

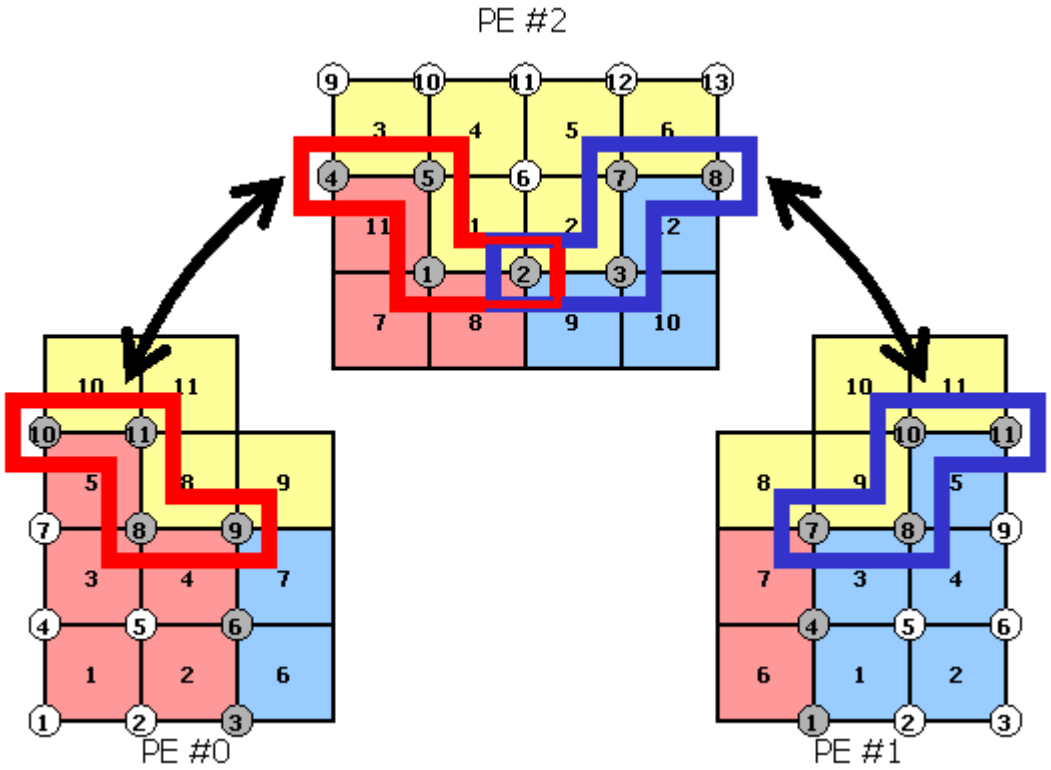


図 3.3.10　第 2 番の部分領域における共有節点