有限要素法における高速化
ベクトルプロセッサとスカラープロセッサ

2007年7月18日

中島研吾

並列計算プログラミング（616-2057）・先端計算機演習I（616-4009）

Interface of Sparse Linear Solver Library Optimized for Various Types of Architectures

Kengo Nakajima
The 21st Century Earth Science COE Program
Department of Earth & Planetary Science
The University of Tokyo & CREST/JST

12th SIAM Conference on Parallel Processing for Scientific Computing (PP06)
MS46: Adaptive Tools and Frameworks for High Performance Numerical Computations - Part II of III
February 24th, 2006, San Francisco, CA, USA.

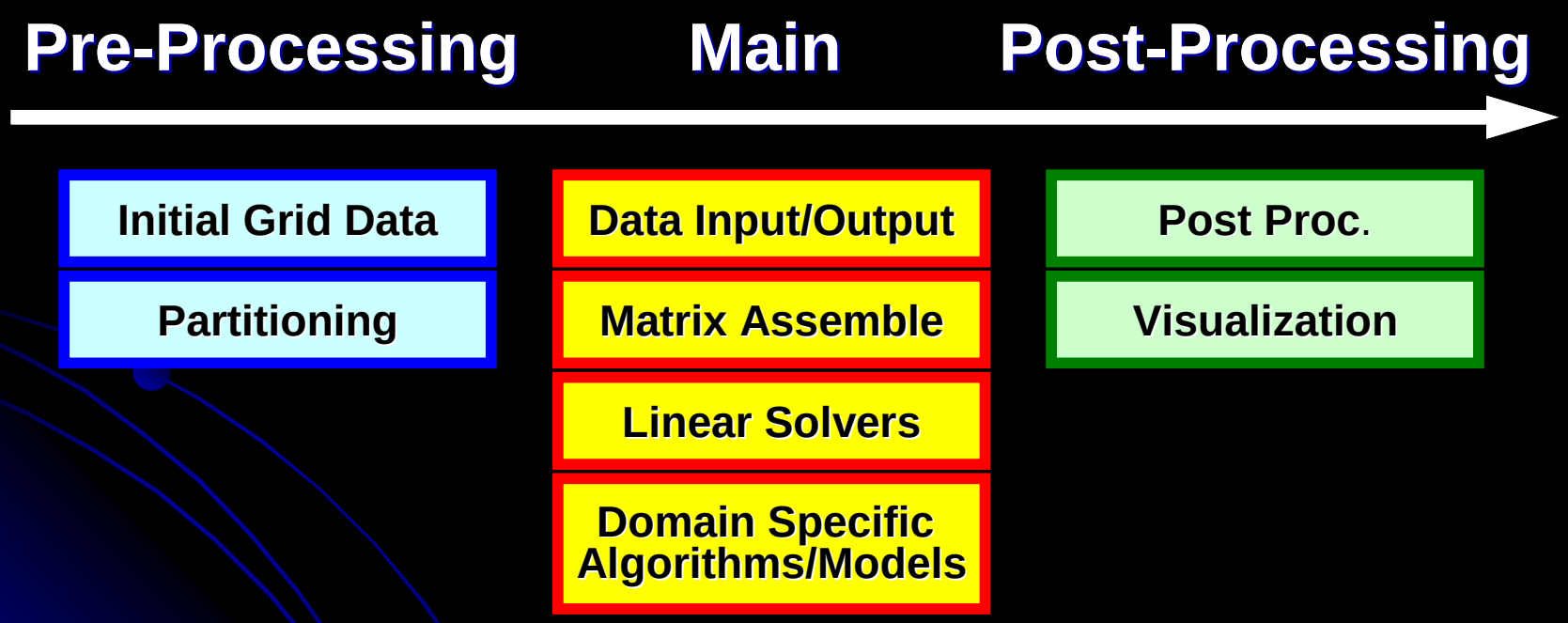
概 要

- 背景
  - HPC-MW (HPC Middleware)
- 有限要素法（FEM）の特徴
  - 線形ソルバーの性能
- 係数行列生成部の最適化
- まとめ

HPC-MW

- FEM（有限要素法），FDM（差分法）などの科学技術計算手法はいくつかの典型的なプロセスから構成されている。

"Parallel" FEM Procedure



HPC-MW

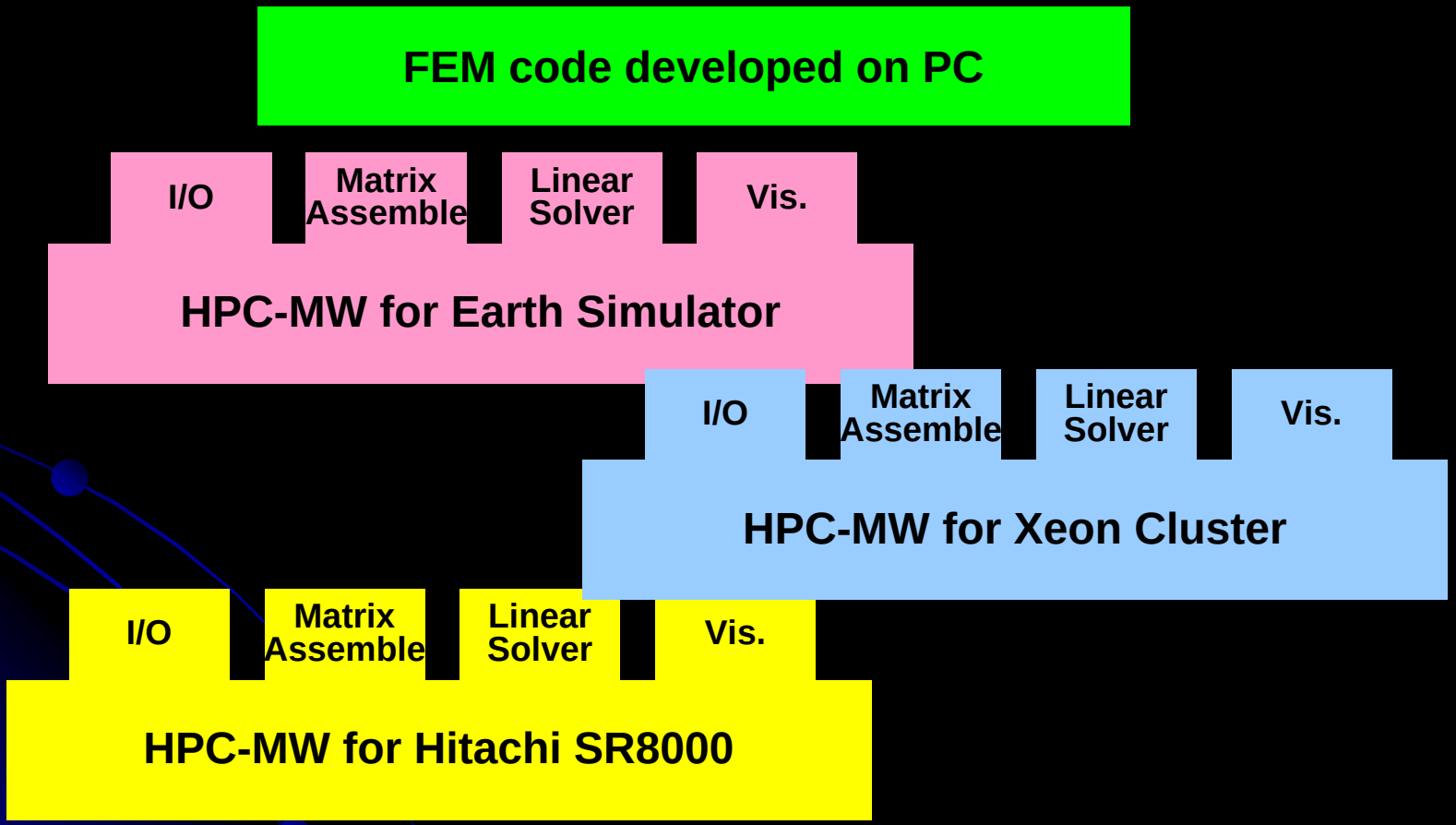
- それぞれのプロセスはH/Wごとに最適化可能。
- HPC-MWは，並列及び単体CPUに対して最適化された有限要素法（FEM）のプログラムの開発を，様々なアーキテクチュア上で支援するためのミドルウェア・ライブラリである。
  - 様々なアーキテクチュア上で単一のインタフェースによって利用できる。

Goal of HPC Middleware

FEM code developed on PC

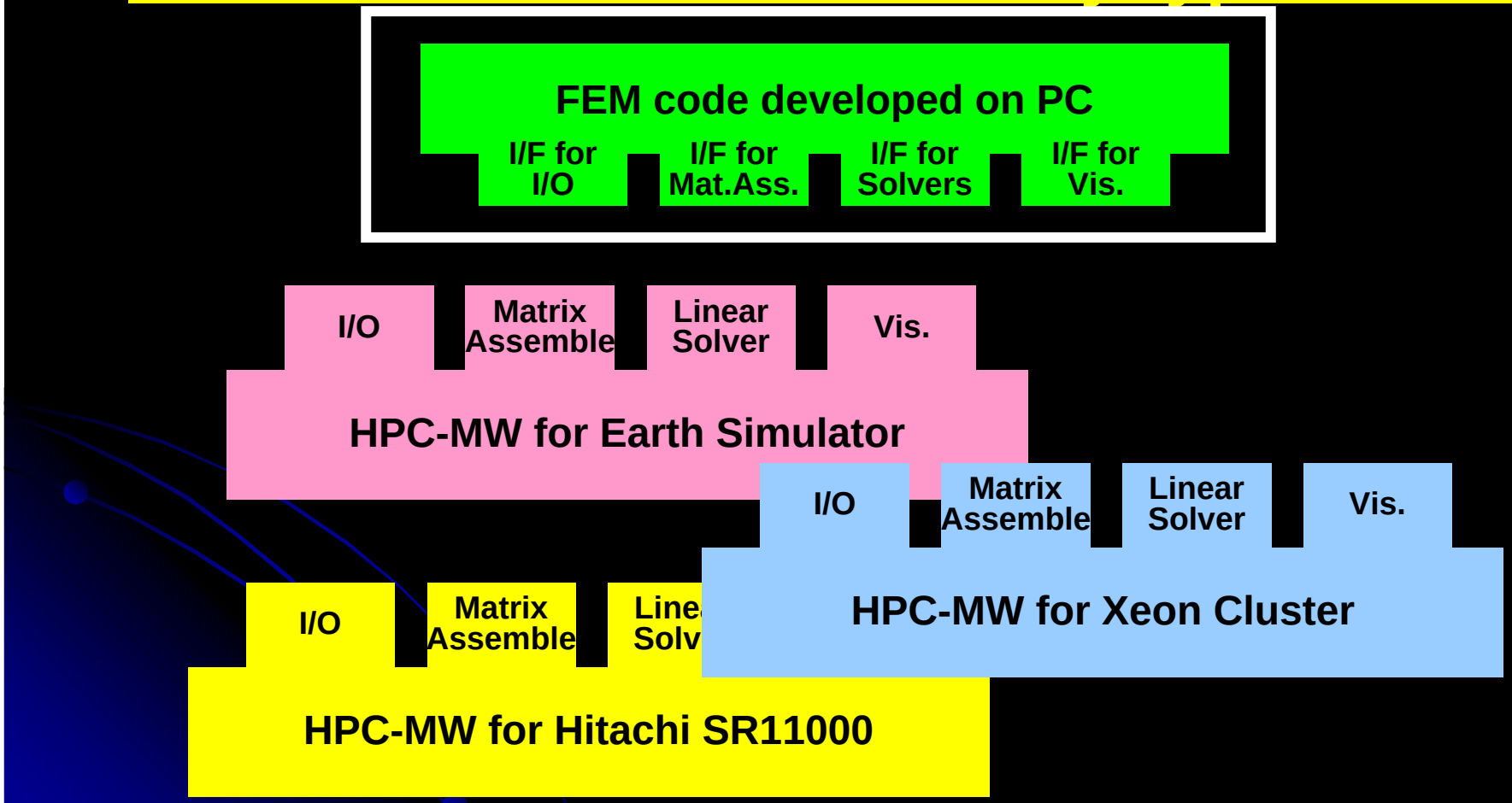
Goal of HPC Middleware

HPC-MW library optimized for each H/W



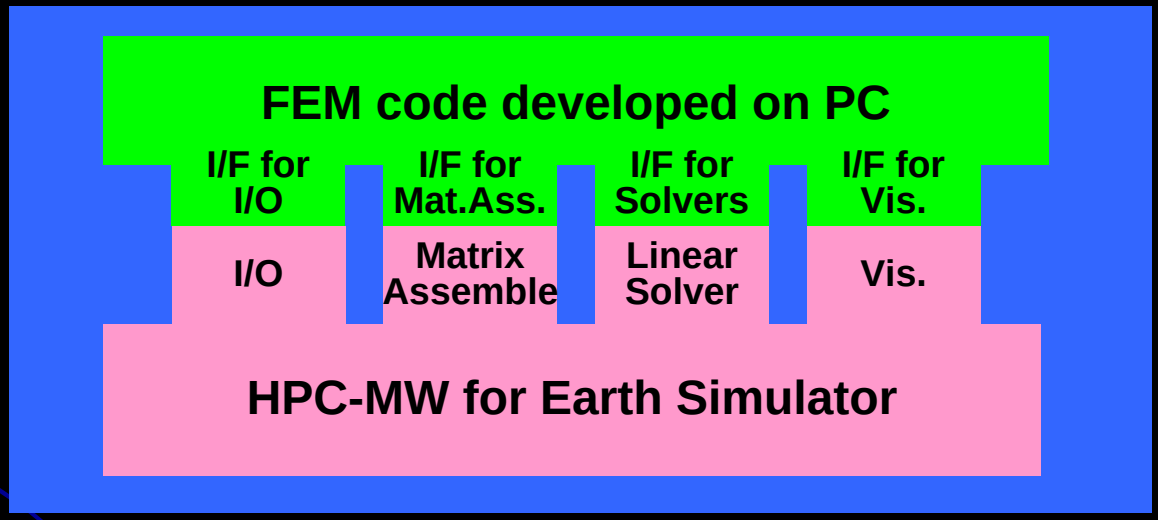
Goal of HPC Middleware

UNIFORM interface for every type of HW



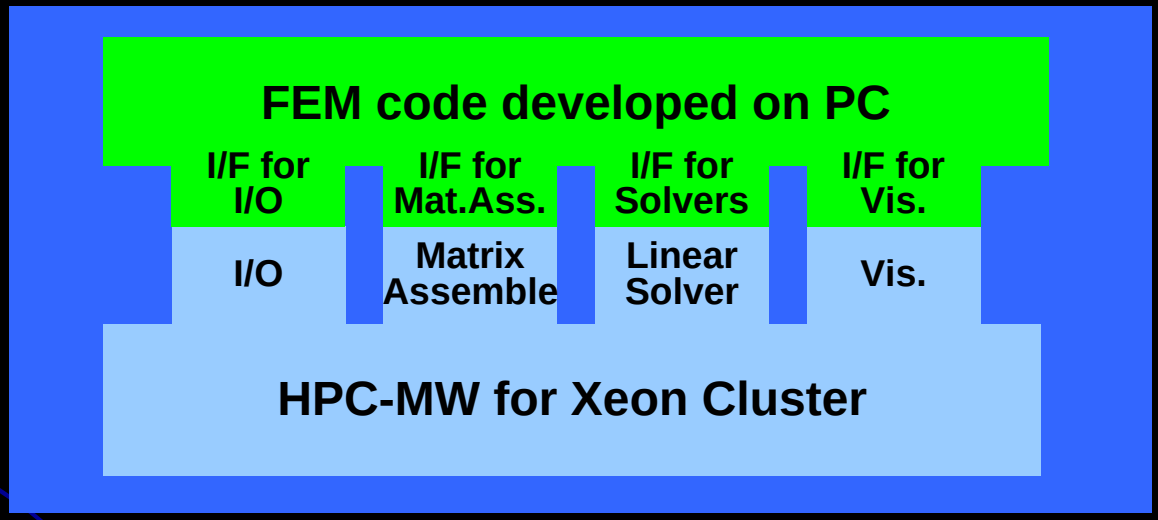
Goal of HPC Middleware

Parallel FEM code optimized for ES



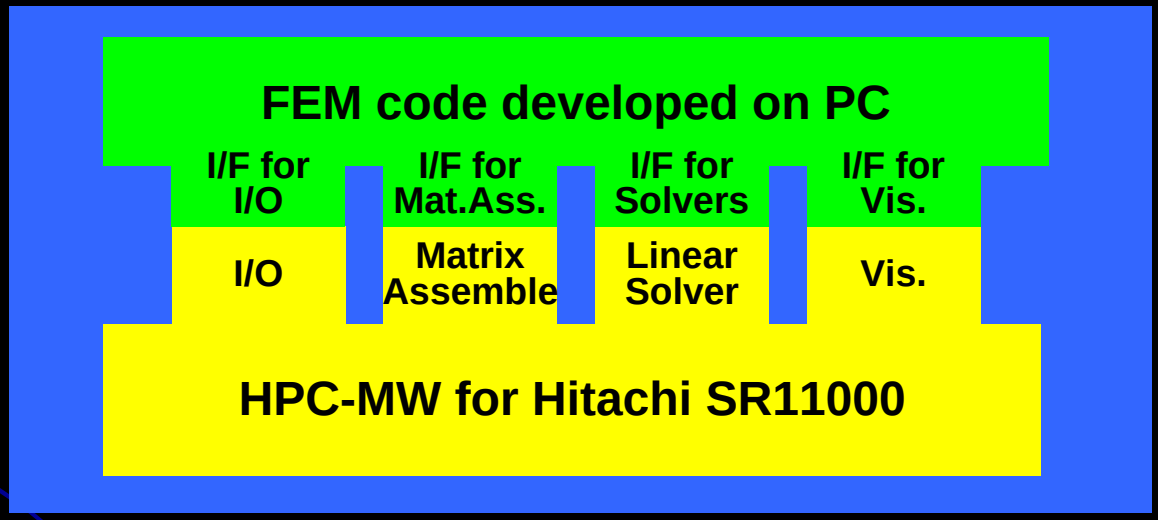
Goal of HPC Middleware

Parallel FEM code optimized for Xeon



Goal of HPC Middleware

Parallel FEM code optimized for SR11000



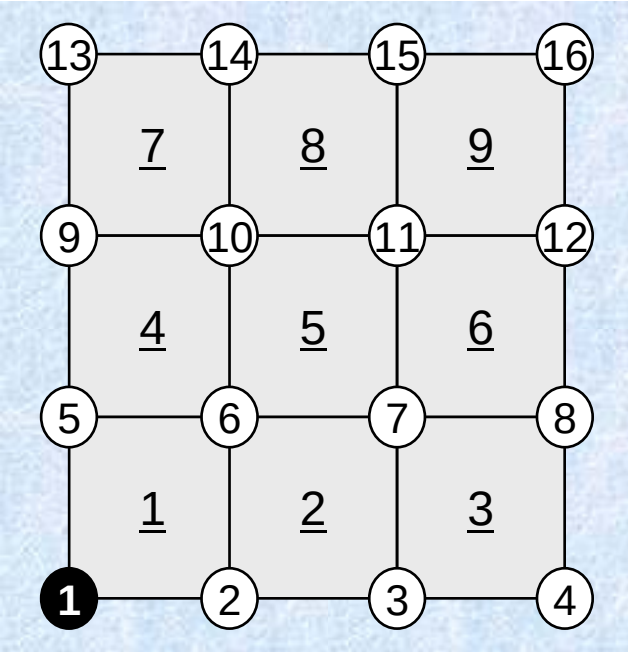
- 背景
  - HPC-MW (HPC Middleware)
- 有限要素法（FEM）の特徴
  - 線形ソルバーの性能
- 係数行列生成部の最適化
- まとめ

Finite-Element Method (FEM)

- **偏微分方程式の解法として広く知られている**
  - elements (meshes，**要素**) & nodes (vertices，**節点**)
- **以下の二次元熱伝導問題を考える**:

$$\lambda\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)+Q=0$$

  - 16**節点**，9**要素（四角形）**
  - **一様な熱伝導率** (λ=1)
  - **一様な体積発熱** (Q=1)
  - **節点**1**で温度固定：**T=0
  - **周囲断熱**



Galerkin FEM procedures

- **各要素にガラーキン法を適用**:

$$\int_V [N]^T \left\{ \lambda \left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q \right\} dV = 0$$

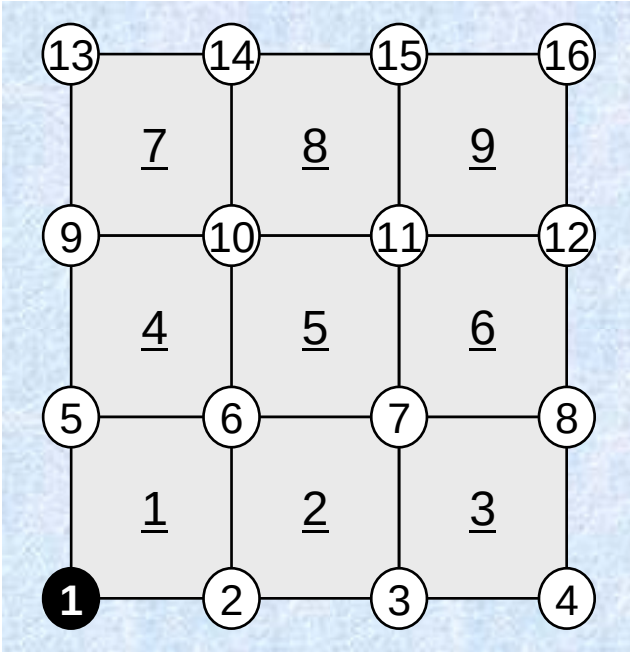
各要素で: $T = [N]\{\phi\}$

$[N]$: 形状関数（内挿関数）

- **偏微分方程式に対して，ガウス・グリーンの定理を適用し，以下の「弱形式」を導く**

$$-\int_V \lambda \left(\frac{\partial [N]^T}{\partial x} \frac{\partial [N]}{\partial x} + \frac{\partial [N]^T}{\partial y} \frac{\partial [N]}{\partial y} \right) dV \cdot \{\phi\}$$

$$+\int_V Q[N]^T dV = 0$$

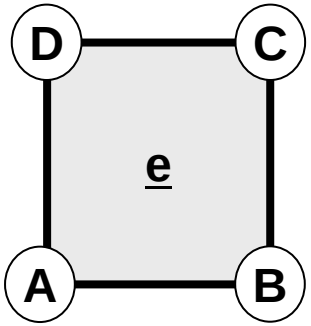


Element Matrix：要素マトリクス

- 各要素において積分を実行し，要素マトリクスを得る

$$-\int_V \lambda\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}+\frac{\partial[N]^T}{\partial y}\frac{\partial[N]}{\partial y}\right)dV\cdot\{\phi\}$$

$$+\int_V Q[N]^T dV=0$$

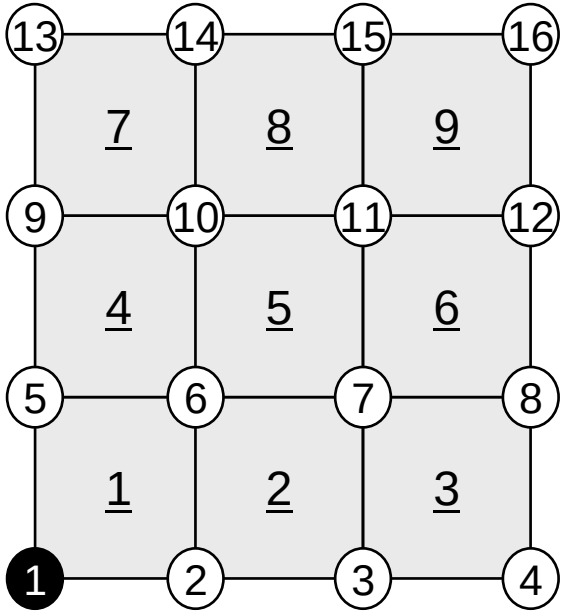


$$[k^{(e)}]\{\phi^{(e)}\}=\{f^{(e)}\}$$

$$\begin{bmatrix} k_{AA}^{(e)} & k_{AB}^{(e)} & k_{AC}^{(e)} & k_{AD}^{(e)} \\ k_{BA}^{(e)} & k_{BB}^{(e)} & k_{BC}^{(e)} & k_{BD}^{(e)} \\ k_{CA}^{(e)} & k_{CB}^{(e)} & k_{CC}^{(e)} & k_{CD}^{(e)} \\ k_{DA}^{(e)} & k_{DB}^{(e)} & k_{DC}^{(e)} & k_{DD}^{(e)} \end{bmatrix} \begin{Bmatrix} \phi_A^{(e)} \\ \phi_B^{(e)} \\ \phi_C^{(e)} \\ \phi_D^{(e)} \end{Bmatrix} = \begin{Bmatrix} f_A^{(e)} \\ f_B^{(e)} \\ f_C^{(e)} \\ f_D^{(e)} \end{Bmatrix}$$

Global/overall Matrix：全体マトリクス

各要素マトリクスを全体マトリクスに足しこむ

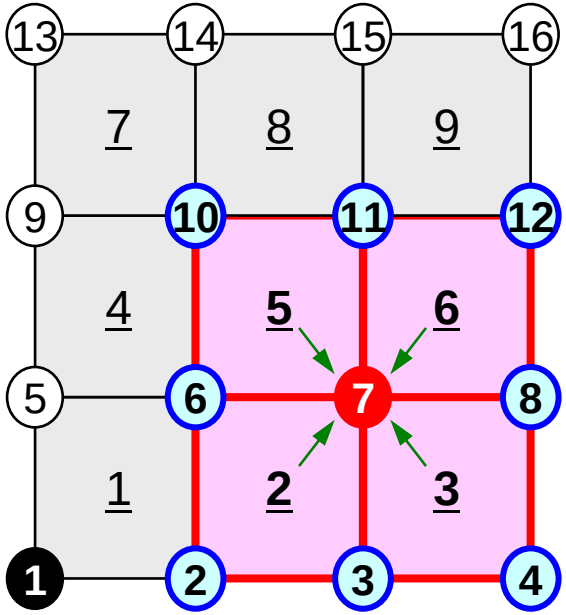


$$[K]\{\Phi\}=\{F\}$$

$$\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}\Phi_1\\ \Phi_2\\ \Phi_3\\ \Phi_4\\ \Phi_5\\ \Phi_6\\ \Phi_7\\ \Phi_8\\ \Phi_9\\ \Phi_{10}\\ \Phi_{11}\\ \Phi_{12}\\ \Phi_{13}\\ \Phi_{14}\\ \Phi_{15}\\ \Phi_{16}\end{Bmatrix}
=
\begin{Bmatrix}F_1\\ F_2\\ F_3\\ F_4\\ F_5\\ F_6\\ F_7\\ F_8\\ F_9\\ F_{10}\\ F_{11}\\ F_{12}\\ F_{13}\\ F_{14}\\ F_{15}\\ F_{16}\end{Bmatrix}$$

Global/overall Matrix：全体マトリクス

各要素マトリクスを全体マトリクスに足しこむ



$$[K]\{\Phi\}=\{F\}$$

$$\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}\Phi_1\\ \Phi_2\\ \Phi_3\\ \Phi_4\\ \Phi_5\\ \Phi_6\\ \Phi_7\\ \Phi_8\\ \Phi_9\\ \Phi_{10}\\ \Phi_{11}\\ \Phi_{12}\\ \Phi_{13}\\ \Phi_{14}\\ \Phi_{15}\\ \Phi_{16}\end{Bmatrix}
=
\begin{Bmatrix}F_1\\ F_2\\ F_3\\ F_4\\ F_5\\ F_6\\ F_7\\ F_8\\ F_9\\ F_{10}\\ F_{11}\\ F_{12}\\ F_{13}\\ F_{14}\\ F_{15}\\ F_{16}\end{Bmatrix}$$

得られた大規模連立一次方程式を解く

ある適切な境界条件 (ここでは$\Phi_1=0$)を適用

「疎」な係数行列

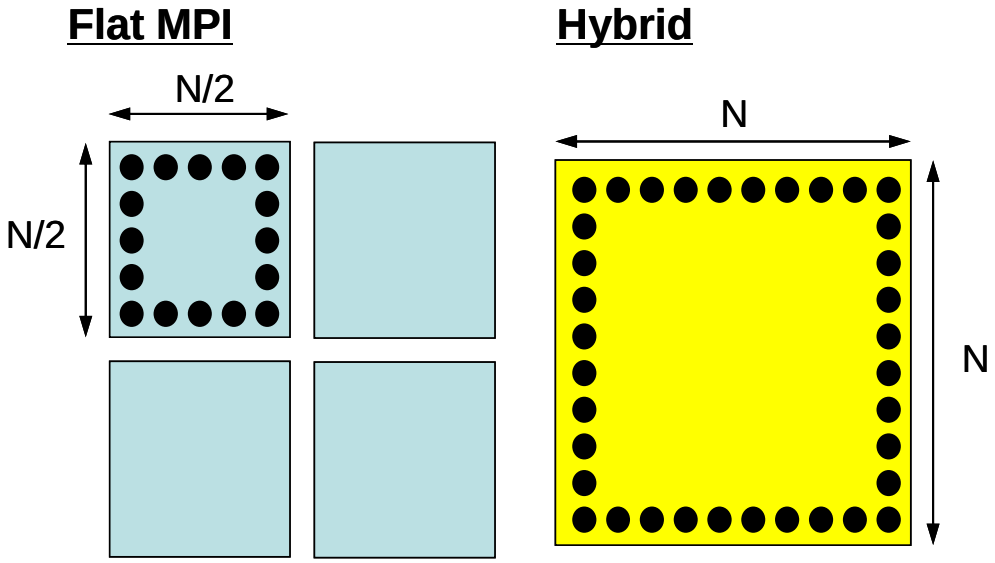
$$
\begin{bmatrix}
D & X & & & X & X & & & & & & & & & & \\
X & D & X & & X & X & X & & & & & & & & & \\
 & X & D & X & & X & X & X & & & & & & & & \\
 & & X & D & & & X & X & & & & & & & & \\
X & X & & & D & X & & & X & X & & & & & & \\
X & X & X & & X & D & X & & X & X & X & & & & & \\
 & X & X & X & & X & D & X & & X & X & X & & & & \\
 & & X & X & & & X & D & & & X & X & & & & \\
 & & & & X & X & & & D & X & & & X & X & & \\
 & & & & X & X & X & & X & D & X & & X & X & X & \\
 & & & & & X & X & X & & X & D & X & & X & X & X \\
 & & & & & & X & X & & & X & D & & & X & X \\
 & & & & & & & & X & X & & & D & X & & \\
 & & & & & & & & X & X & X & & X & D & X & \\
 & & & & & & & & & X & X & X & & X & D & X \\
 & & & & & & & & & & X & X & & & X & D
\end{bmatrix}
\begin{Bmatrix}
\Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \\ \Phi_7 \\ \Phi_8 \\ \Phi_9 \\ \Phi_{10} \\ \Phi_{11} \\ \Phi_{12} \\ \Phi_{13} \\ \Phi_{14} \\ \Phi_{15} \\ \Phi_{16}
\end{Bmatrix}
=
\begin{Bmatrix}
F_1 \\ F_2 \\ F_3 \\ F_4 \\ F_5 \\ F_6 \\ F_7 \\ F_8 \\ F_9 \\ F_{10} \\ F_{11} \\ F_{12} \\ F_{13} \\ F_{14} \\ F_{15} \\ F_{16}
\end{Bmatrix}
$$

FEMアプリケーションの特徴（1/2）

- 要素単位の局所的な処理
  - 係数行列：疎行列
  - 基本的に並列計算向きである
- 間接参照の多さ
  - memory intensive
  - 疎行列
- 有限体積法と似ているがもっと複雑
  - 要素と節点
- ほとんどの計算時間は，係数行列生成（要素・全体マトリクス）と線形方程式の求解に費やされる
  - poi_gen と solver
  - 線形方程式の最適化については研究が進んでいる。

FEMアプリケーションの特徴 （2/2）

- **並列**FEM
  - **内積などをのぞくと，通信は近隣プロセッサ間とのみ発生**
  - **通信量もそれほど大きくない（領域境界の値のみ交換）**
  - communication (MPI) latency **が「効く（**critical**）」**



ICCG型ソルバーの特徴

- IC (Incomplete Cholesky)/ILU (Incomplete LU) **分解は線形方程式の反復解法の前処理手法として広く使用されている。**
- **IC/ILU における前進後退代入のプロセスはベクトル，並列計算機向けではないが・・・**
  - **色々な対策が既に考えられている**
  - **ベクトル，SMP並列（OpenMP）**
    - Mulricolor Ordering
  - **分散メモリ並列（MPI）**
    - **ブロックヤコビ型局所前処理手法**
  - **適切な係数行列格納法**
    - DJDS，CRS

後期に詳しくやります

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

$$x_k = \tilde{d}_k \left(y_k - \sum_{j=k+1}^{N} u_{kj} y_j \right) \quad (k = N, N-1, \cdots, 1)$$

疎行列：メモリに負担がかかる
間接参照：キャッシュに乗りにくい

疎行列：差分法，有限要素法，有限体積法

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

飛び飛びになる可能性

密行列：スペクトル法，分子動力学，境界要素法

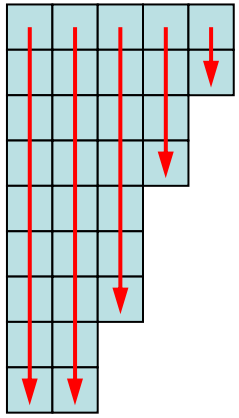
```
do i= 1, N
  q(i)= 0.d0
  do j= 1, N
    q(i)= q(i) + AMATd(i,j)*p(j)
  enddo
enddo
```

連続したアドレス

係数行列格納法，ループの構造

- DJDS (Descending order Jagged Diagonal Storage) with long innermost loops is suitable for vector processors.
  - ベクトル計算機向け
  - 最内ループ長が大きい

DJDS

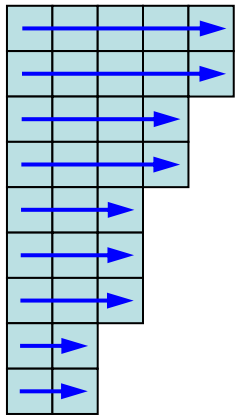


```
do iv= 1, NVECT
  iv0= STACKmc(iv-1)
  do j= 1, NLhyp(iv)
    iS= index_L(NL*(iv-1)+ j-1)
    iE= index_L(NL*(iv-1)+ j  )
    do i= iv0+1, iv0+iE-iS
       k= i+iS - iv0
      kk= item_L(k)
      Z(i)= Z(i) - AL(k)*Z(kk)
    enddo

    iS= STACKmc(iv-1) + 1
    iE= STACKmc(iv  )
    do i= iS, iE
      Z(i)= Z(i)/DD(i)
    enddo
  enddo
enddo
```

- Reduction type loop of DCRS is more suitable for cache-based scalar processor because of its localized operation.
  - スカラー計算機向け
  - 右辺はともかく左辺（SW）は固定
  - DJDSでは左辺（Z(i)）も変化

DCRS

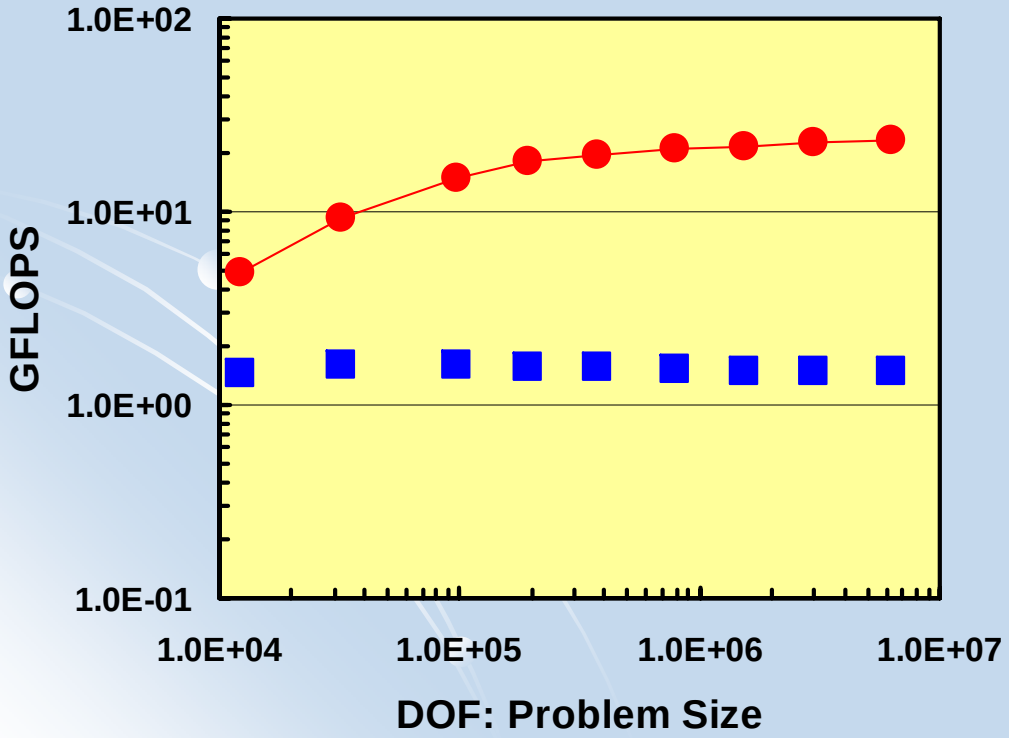


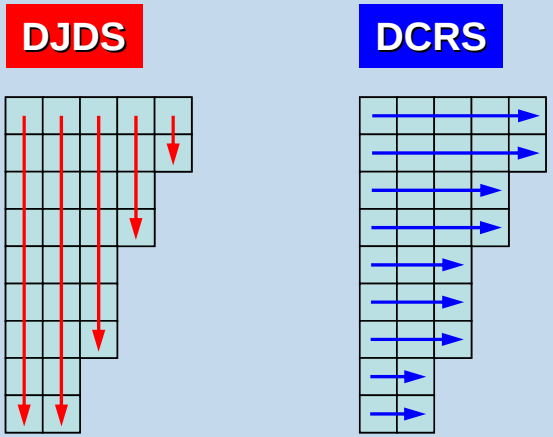
```
do i= 1, N
  SW= WW(i,Z)
  isL= index_L(i-1)+1
  ieL= index_L(i)
  do j= isL, ieL
    k = item_L(j)
    SW= SW - AL(j)*Z(k)
  enddo

  Z(i)= SW/DD(i)
enddo
```

Impact of Reordering on 3D Elastic Simulation (FEM) Problem Size~GFLOPS

Earth Simulator, 8 PE's, Flat-MPI

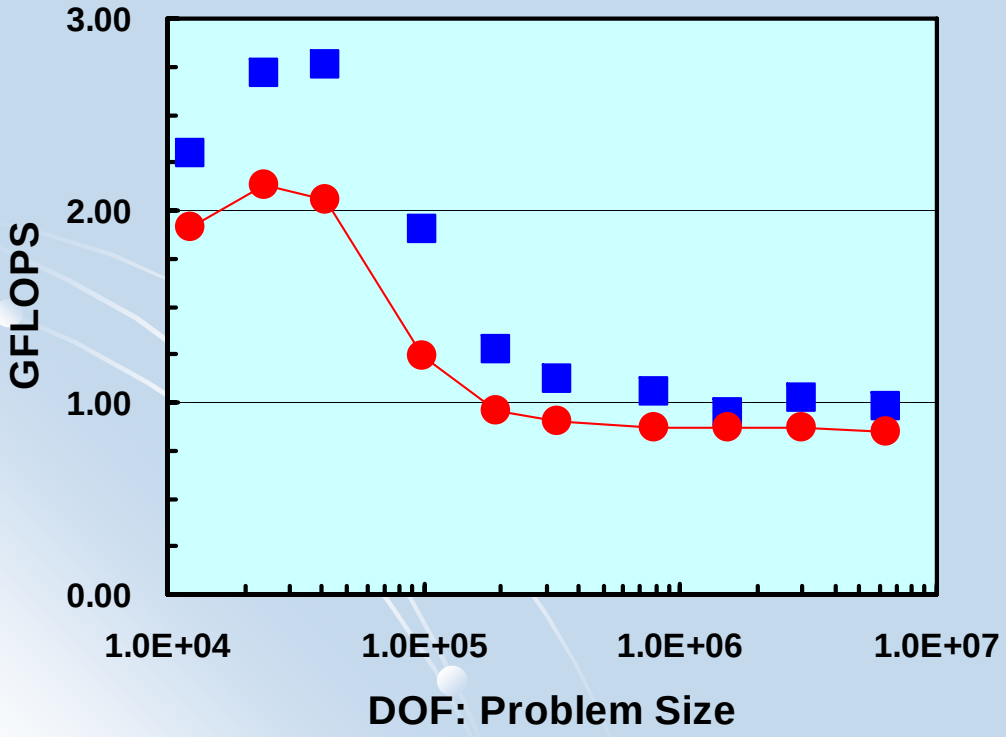


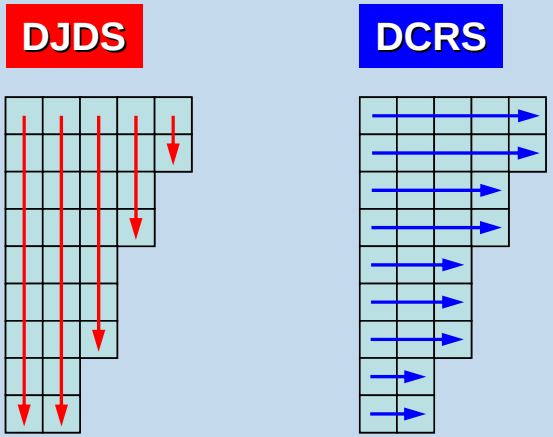


DJDS (Descending order Jagged Diagonal Storage) with long innermost loops is suitable for vector processors.

Impact of Reordering on 3D Elastic Simulation (FEM) Problem Size~GFLOPS

IBM SP-3 (Seaborg@NERSC), 8 PE's, Flat-MPI





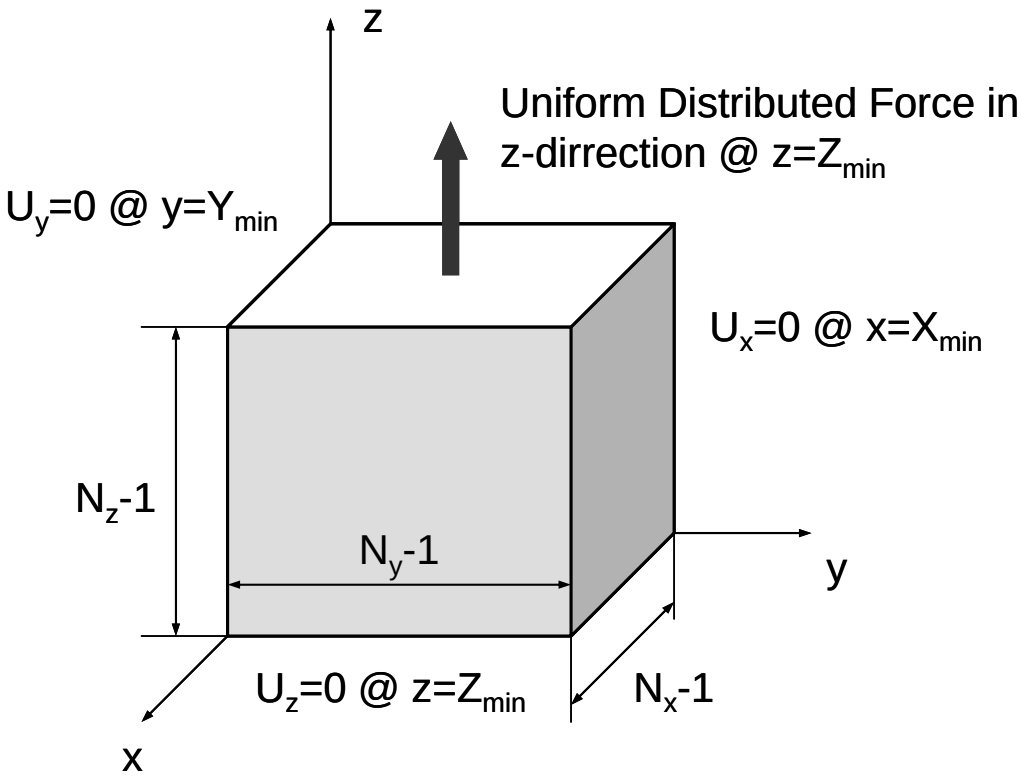
Reduction type loop of DCRS is more suitable for cache-based scalar processor because of its localized operation.

- 背景
  - HPC-MW (HPC Middleware)
- 有限要素法（FEM）の特徴
  - 線形ソルバーの性能
- 係数行列生成部の最適化
- まとめ

“CUBE” Benchmark

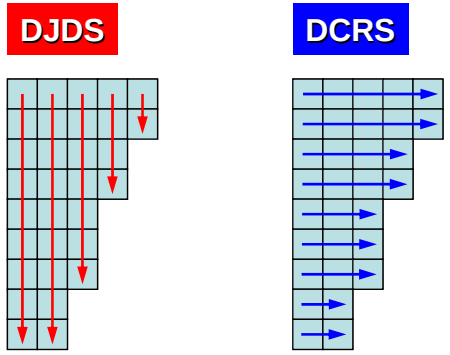
- **単純な三次元弾性解析**

- Hardware：Single CPU
  - Earth Simulator
  - AMD Opteron (1.8GHz)

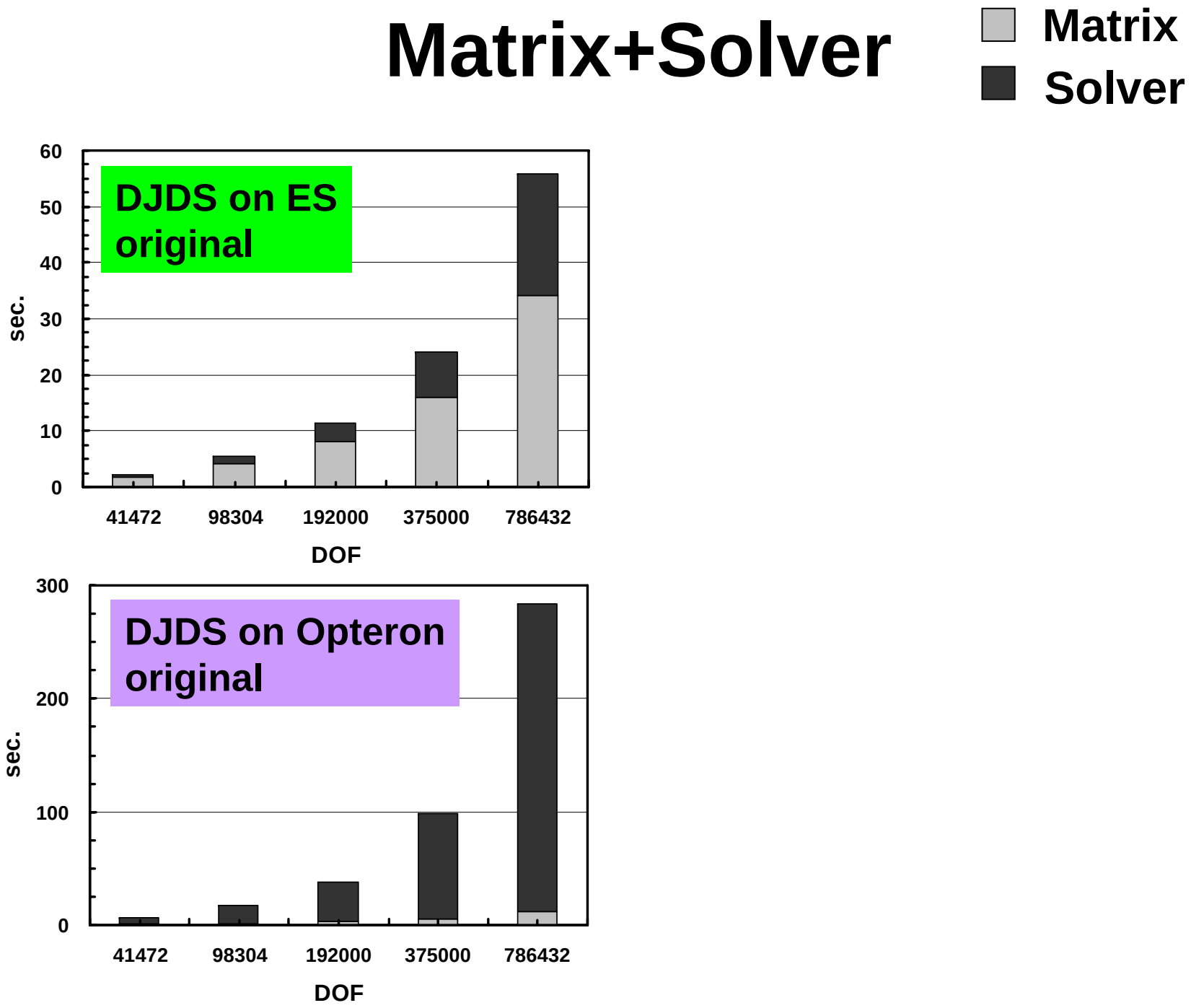


Time for $3 \times 64^3 = 786{,}432$ DOF

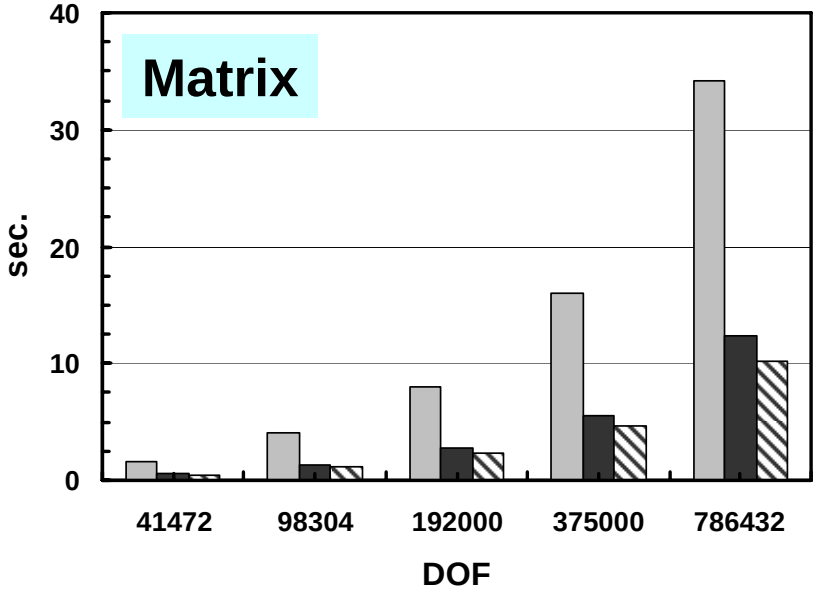
		DCRS sec. (MFLOPS)	DJDS original sec. (MFLOPS)
ES **8.0 GFLOPS**	Matrix	28.6 (291)	**34.2 (240)**
	Solver	360 (171)	**21.7 (3246)**
	Total	389	55.9
Opteron 1.8GHz **3.6 GFLOPS**	Matrix	**10.2 (818)**	12.4 (663)
	Solver	**225 (275)**	271 (260)
	Total	235	283

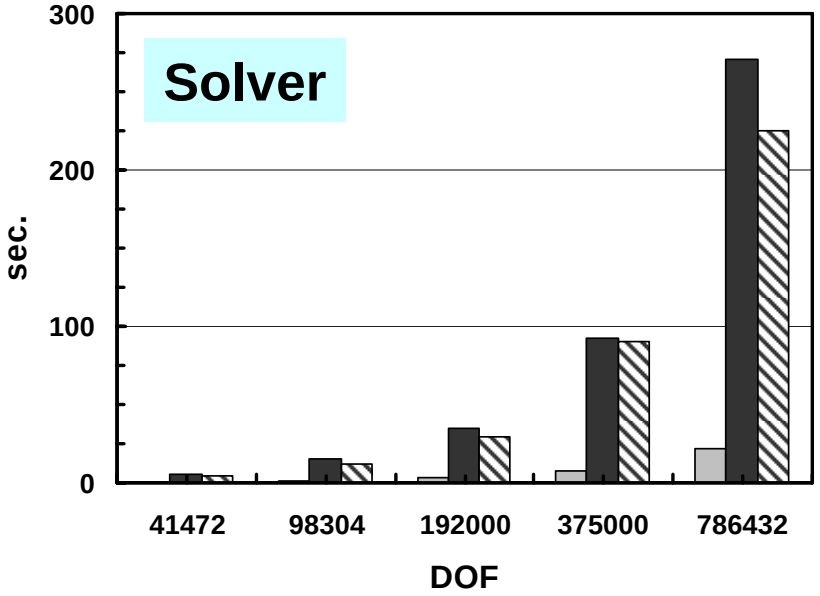


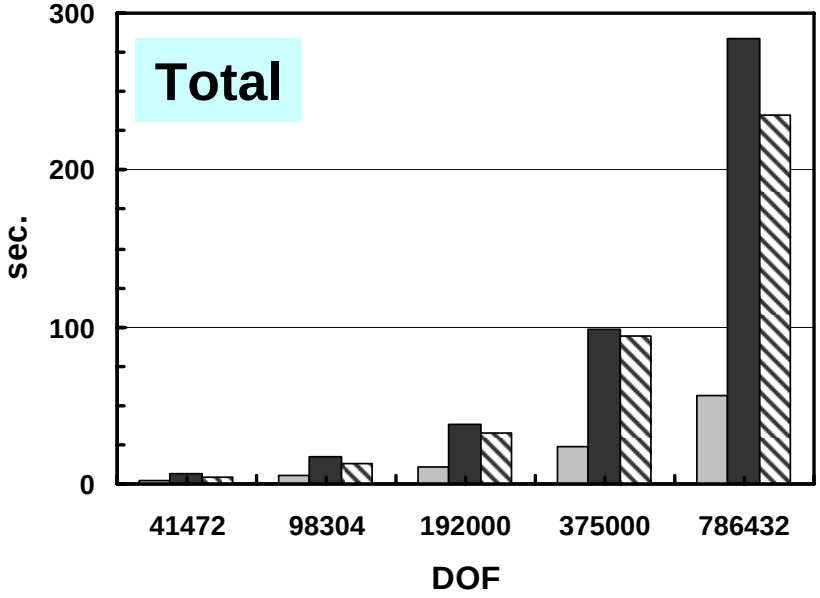
Matrix+Solver



Computation Time vs. Problem Size







ES (DJDS original)
Opteron (DJDS original)
Opteron (DCRS)

地球シミュレータ：係数行列生成部に時間がかかっている

- この部分も最適化が必要
- 非線形解析の場合には，反復ごとに係数行列の作り直しが必要になる場合もあるため，重要である。

- 「係数行列生成部」はアプリケーションに対する依存性が強い（弱形式の積分を実施する部分）
  - 線形ソルバーなどと違ってライブラリ化が難しい
  - ベクトル化しにくい複雑なプロセスが含まれている
    - ベクトル機のスカラー処理性能は概して低い

FEMにおける係数行列作成のプロセス

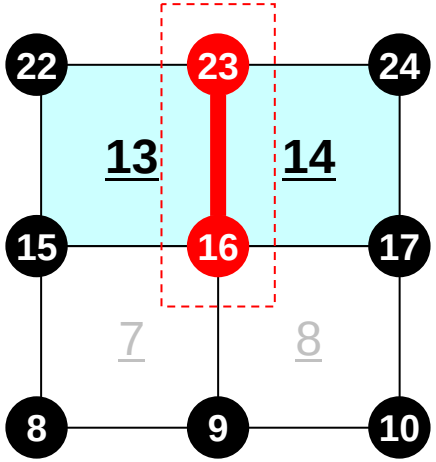
- 前処理
  - 要素⇒全体行列のマッピング情報の生成
    - 分岐多い
    - ベクトル計算向けでは無い
- 計算部分
  - 要素ごとに弱形式の積分を計算し，「要素マトリクス」を計算する。
  - 前処理で求めた関係を使用して，「要素マトリクス」を「全体マトリクス」に重ね合わせる。
    - 現状ではこの部分にもかなり分岐が多く，かつ最内ループ長が短い
    - ベクトル計算機向けではない
- ここではこの「計算部分」の最適化について考える

Element-by-Element Operations

- Integration over each **element** => element-matrix
- Element matrices are accumulated to each **node** => global-matrix
- Linear equations for each **node**

$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & & & & & \\ a_{2,1} & a_{2,2} & & & & & & & \\ & & a_{3,3} & & & & & & \\ & & & \dots & & & & & \\ & & & & a_{33,33} & & & & \\ & & & & & & a_{34,34} & a_{34,35} & \\ & & & & & & a_{35,34} & a_{35,35} & \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{33} \\ u_{34} \\ u_{35} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_{33} \\ f_{34} \\ f_{35} \end{Bmatrix}$$

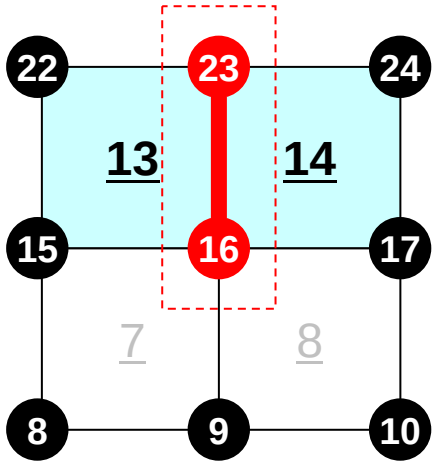
Element-by-Element Operations

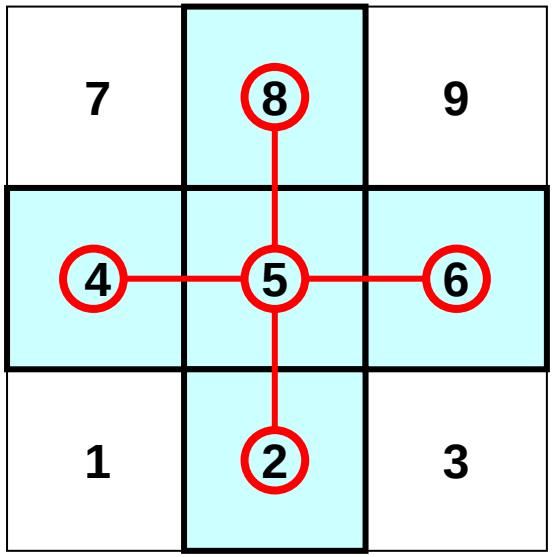


$$\begin{bmatrix} a_{1,1} & a_{1,2} & & & & & & \\ a_{2,1} & a_{2,2} & & & & & & \\ & & a_{3,3} & & & & & \\ & & & \ddots & & & & \\ & & & & a_{33,33} & & & \\ & & & & & a_{34,34} & a_{34,35} & \\ & & & & & a_{35,34} & a_{35,35} & \end{bmatrix} \begin{Bmatrix} u_1 \\ u_2 \\ u_3 \\ \dots \\ u_{33} \\ u_{34} \\ u_{35} \end{Bmatrix} = \begin{Bmatrix} f_1 \\ f_2 \\ f_3 \\ \dots \\ f_{33} \\ f_{34} \\ f_{35} \end{Bmatrix}$$

- If you calculate $a_{23,16}$ and $a_{16,23}$, you have to consider contribution by both of 13th and 14th elements.

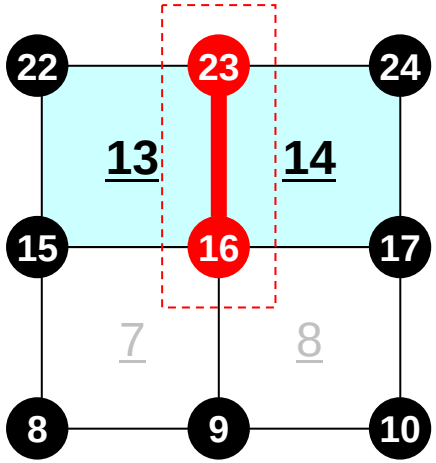
これがFEMがちょっと難しいところ
要素と節点がある





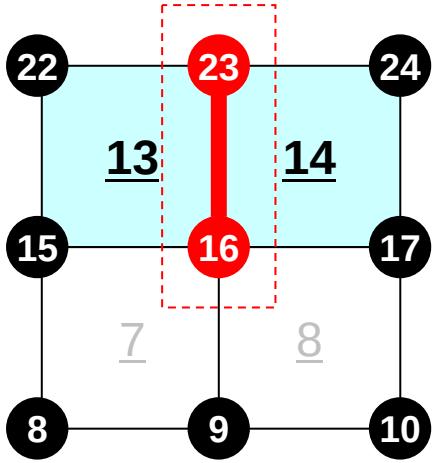
有限体積法:
変数が要素中心で定義されるため,
要素～要素の関係しか存在しない

Current Approach



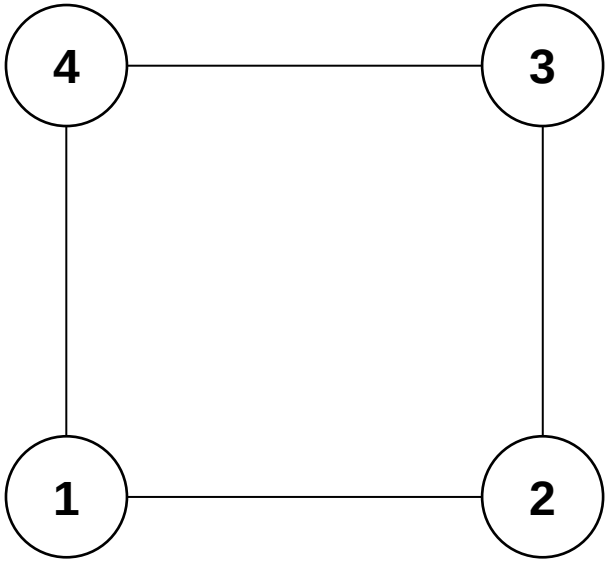

```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulated element-matrix to global-matrix
  enddo
  enddo
enddo
```

Current Approach



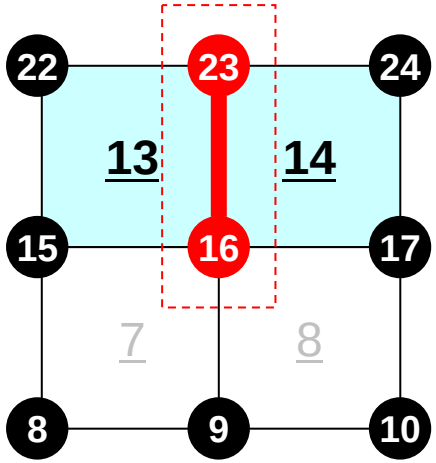

```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulat element-matrix to global-matrix
  enddo
  enddo
enddo
```

Local Node ID



Local Node ID
for each bi-linear 4-node element

Current Approach




```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulat element-matrix to global-matrix
  enddo
  enddo
enddo
```

Local Node ID

- Nice for cache reuse because of localized operations
- Not suitable for vector processors
  - $a_{16,23}$ and $a_{23,16}$ might not be calculated properly.
  - Short innermost loops
  - There are many “if-then-else” s

		DCRS sec. (MFLOPS)	DJDS original sec. (MFLOPS)
ES 8.0 GFLOPS	Matrix	28.6 (291)	**34.2 (240)**
	Solver	360 (171)	**21.7 (3246)**
	Total	389	55.9
Opteron 1.8GHz 3.6 GFLOPS	Matrix	**10.2 (818)**	12.4 (663)
	Solver	**225 (275)**	271 (260)
	Total	235	283

実はこの内側に更にこのようなガウス積分のプロセスがある

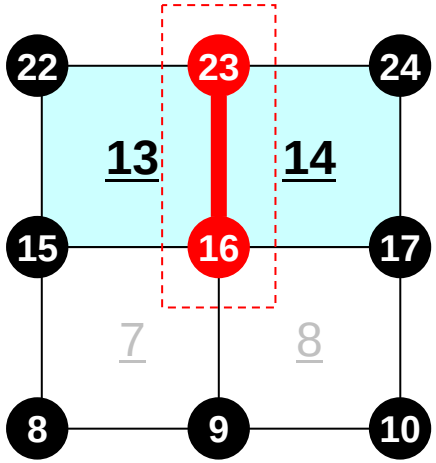
```
do jpn= 1, 2
do ipn= 1, 2
  coef= dabs(DETJ(ipn,jpn))*WEI(ipn)*WEI(jpn)

  PNXi= PNX(ipn,jpn,ie)
  PNYi= PNY(ipn,jpn,ie)

  PNXj= PNX(ipn,jpn,je)
  PNYj= PNY(ipn,jpn,je)

  a11= a11 + (valX*PNXi*PNXj + valB*PNYi*PNYj)*coef
  a22= a22 + (valX*PNYi*PNYj + valB*PNXi*PNXj)*coef
  a12= a12 + (valA*PNXi*PNYj + valB*PNXj*PNYi)*coef
  a21= a21 + (valA*PNYi*PNXj + valB*PNYj*PNXi)*coef
enddo
enddo
```

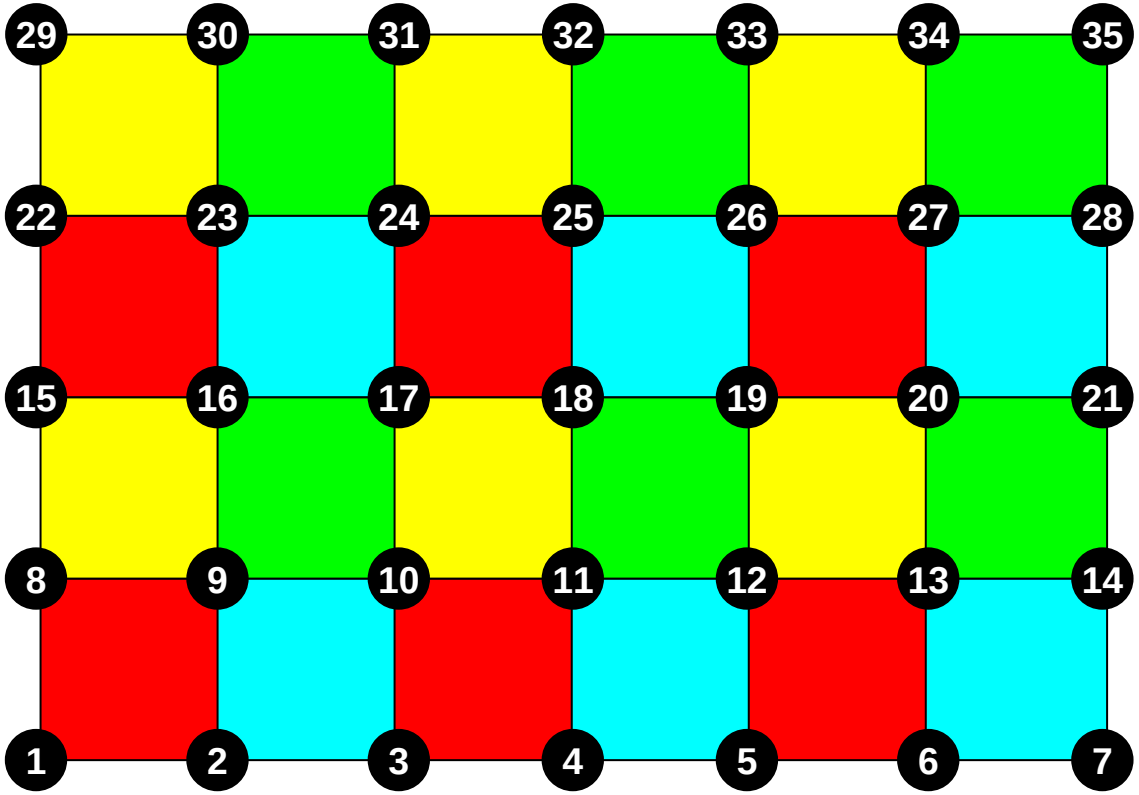
Remedy




```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulat element-matrix to global-matrix
  enddo
  enddo
enddo
```

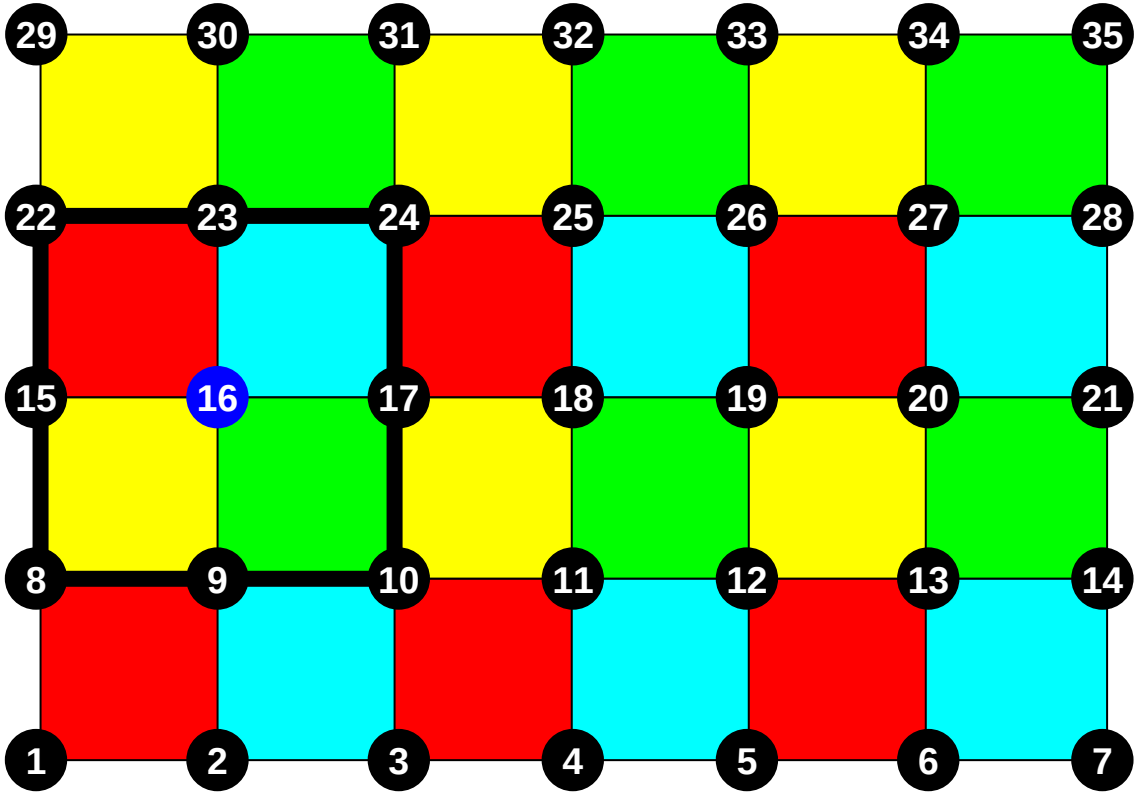
- $a_{16,23}$ and $a_{23,16}$ might not be calculated properly.
  - **要素13と要素14からの寄与がある。**
  - **ベクトルプロセッサでは，ベクトル化されると要素13と要素14の計算が同時に実施される可能性があり，要素マトリクスを全体マトリクスに足しこむ際に $a_{16,23}$ と $a_{23,16}$ が正しく計算されない可能性がある。**
  - coloring the elements: elements which do not share any nodes are in same color.

Coloring of Element



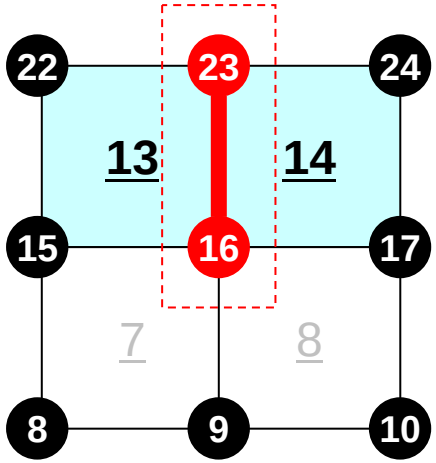
Coloring of Element

Elements sharing the 16th node are assigned to different colors
同じ「色」に属する要素は同時に計算することを許す。



後期に詳しくやります

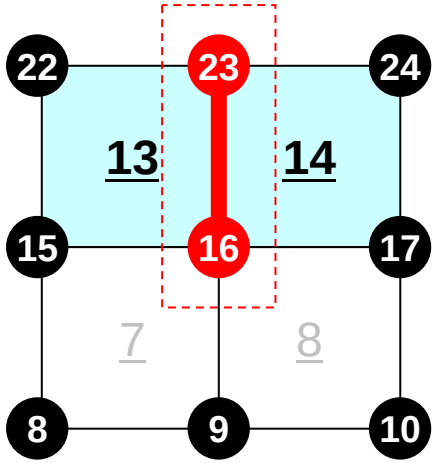
Remedy




```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulat element-matrix to global-matrix
  enddo
  enddo
enddo
```

- $a_{16,23}$ and $a_{23,16}$ might not be calculated properly.
  - coloring the elements: elements which do not share any nodes are in same color.
- Short innermost loops
  - loop exchange

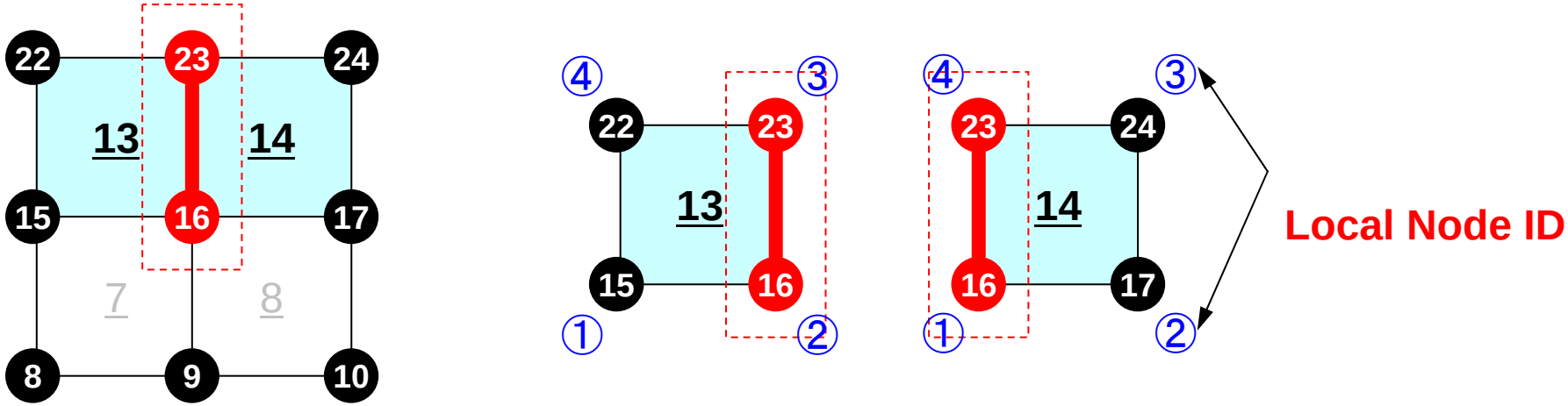
Remedy




```
do icel= 1, ICELTOT
  do ie= 1, 4
  do je= 1, 4
    - assemble element-matrix
    - accumulat element-matrix to global-matrix
  enddo
  enddo
enddo
```

- $a_{16,23}$ and $a_{23,16}$ might not be calculated properly.
  - coloring the elements: elements which do not share any nodes are in same color.
- Short innermost loops
  - loop exchange
- There are many “if-then-else” s
  - define ELEMENT-to-MATRIX array

Define ELEMENT-to-MATRIX array




```
if
   kkU= index_U(16-1+k) and
   item_U(kkU)= 23      then

   ELEMmat(13,2,3)= +kkU
   ELEMmat(14,1,4)= +kkU
endif
```

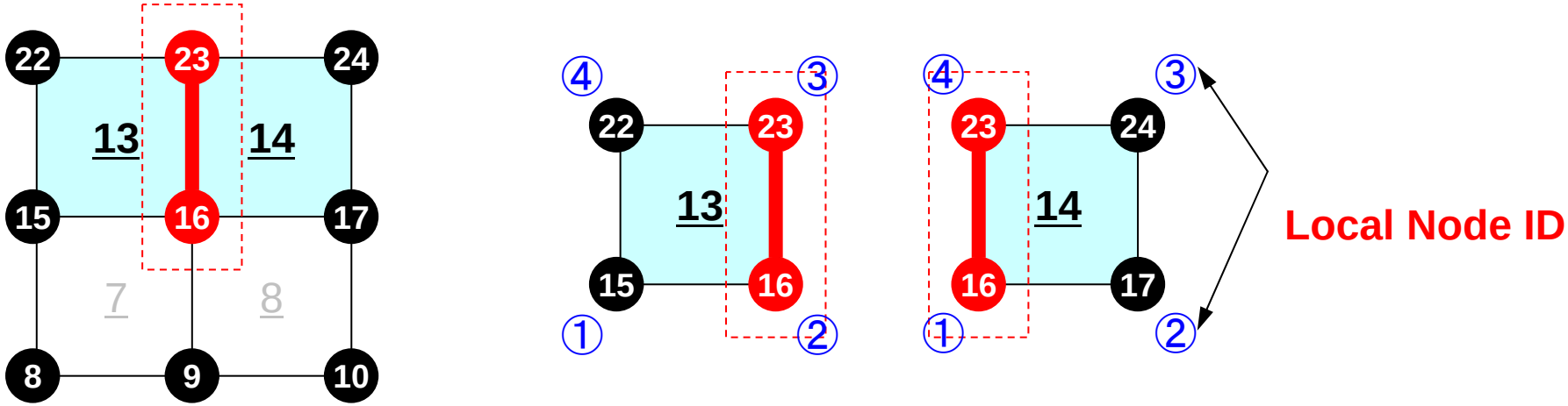
```
if
   kkL= index_L(23-1+k) and
   item_L(kkL)= 16      then

   ELEMmat(13,3,2)= -kkL
   ELEMmat(14,4,1)= -kkL
endif
```

ELEMmat(icel, ie, je)

Element ID Local Node ID

Define ELEMENT-to-MATRIX array




```
if
   kkU= index_U(16-1+k) and
   item_U(kkU)= 23      then

   ELEMmat(13,2,3)= +kkU
   ELEMmat(14,1,4)= +kkU
endif
```

```
if
   kkL= index_L(23-1+k) and
   item_L(kkL)= 16      then

   ELEMmat(13,3,2)= -kkL
   ELEMmat(14,4,1)= -kkL
endif
```

“ELEMmat” specifies relationship between node pairs of each node of each element and address of global coefficient matrix.

ベクトルプロセッサ用高速化：まとめ

- **要素の色分け（Coloring）**
  - **同じ節点を共有する要素は別別に色分け，各色ごとに「全体マトリクス」への重ね合わせを実施する。**
  - **「計算順序」を保つ。**
- **ループの交換**
  - **最内ループ長**
- **「if-then-else（ベクトル化の阻害要因）」の解消**
  - **配列の導入（ELEMmat）**
- **最終的にプログラムを処理別に分割する必要があった（次ページ以降）**

Optimized Procedure

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - define “ELEMmat” array
    enddo
  enddo
  enddo
enddo
```

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - assemble element-matrix
    enddo
  enddo
  enddo

  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - accumulate element-matrix to global-matrix
    enddo
  enddo
  enddo
enddo
```

Extra Computation for
- ELEMmat

Extra Storage for
- ELEMmat array
- element-matrix components for elements in each color
- < 10% increase

Optimized Procedure

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - define “ELEMmat” array
    enddo
  enddo
  enddo
enddo
```

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - assemble element-matrix
    enddo
  enddo
  enddo

  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - accumulate element-matrix to global-matrix
    enddo
  enddo
  enddo
enddo
```

PART I
“Integer” operations for “ELEMmat”
In nonlinear cases, this part should be done **just once** (before initial iteration), as long as mesh connectivity does not change.

この部分は「if-then-else」が多い

Optimized Procedure

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - define “ELEMmat” array
    enddo
  enddo
  enddo
enddo
```

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - assemble element-matrix
    enddo
  enddo
  enddo

  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - accumulate element-matrix to global-matrix
    enddo
  enddo
  enddo
enddo
```

PART I
“Integer” operations for “ELEMmat”
In nonlinear cases, this part should be done **just once** (before initial iteration), as long as mesh connectivity does not change.

この部分は「if-then-else」が多い

PART II
“Floating” operations for matrix assembling/accumulation.

In nonlinear cases, this part is repeated for every nonlinear iteration.
「if-then-else」はほとんど無い

Time for $3 \times 64^3 = 786{,}432$ DOF

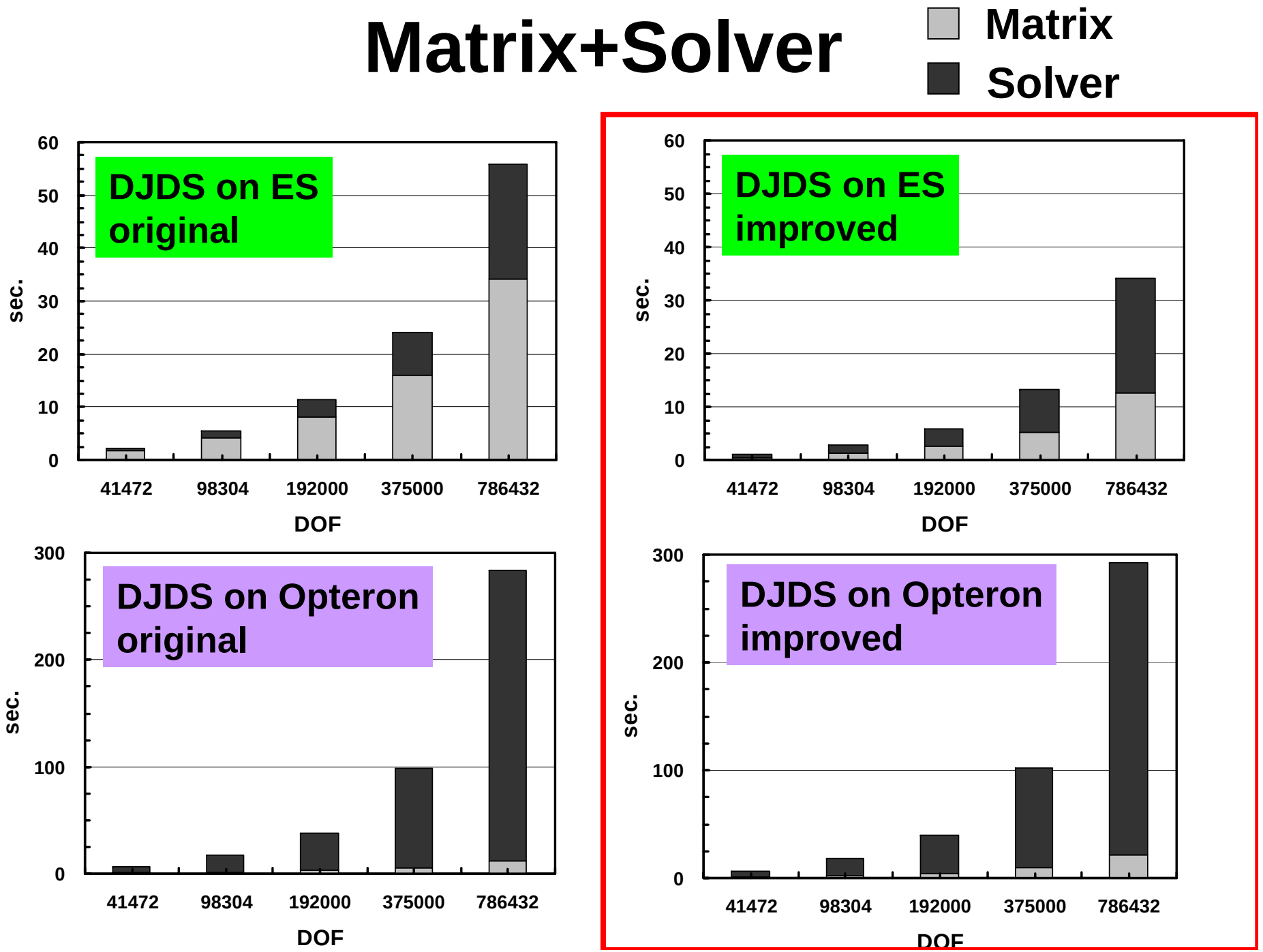
		DCRS sec. (MFLOPS)	**DJDS original** sec. (MFLOPS)	**DJDS improved** sec. (MFLOPS)
ES **8.0 GFLOPS**	Matrix	28.6 (291)	**34.2 (240)**	**12.5 (643)**
	Solver	360 (171)	21.7 (3246)	21.7 (3246)
	Total	389	55.9	34.2
Opteron 1.8GHz **3.6 GFLOPS**	Matrix	**10.2 (818)**	12.4 (663)	21.2 (381)
	Solver	225 (275)	271 (260)	271 (260)
	Total	235	283	292

Time for $3 \times 64^3 = 786{,}432$ DOF

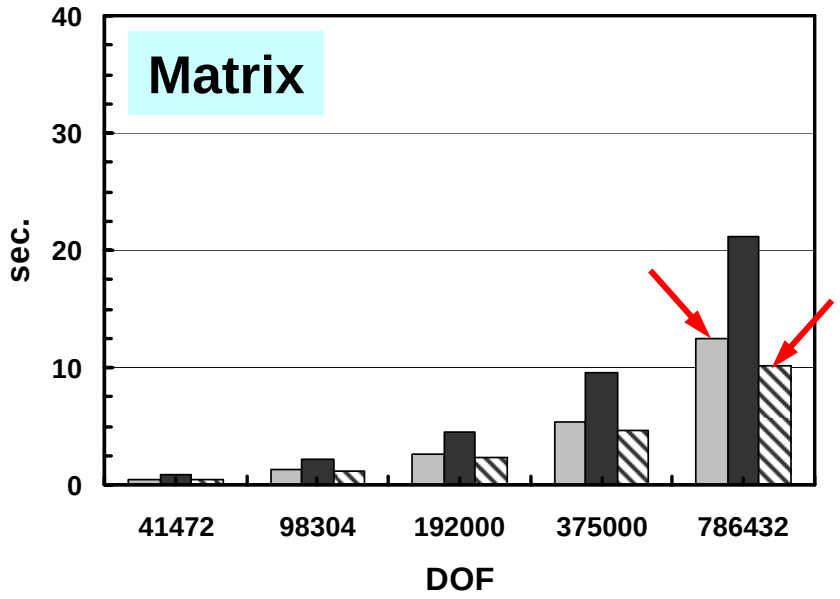
		DCRS sec. (MFLOPS)	DJDS original sec. (MFLOPS)	DJDS improved sec. (MFLOPS)
ES **8.0 GFLOPS**	Matrix	28.6 (291)	**34.2 (240)**	**12.5 (643)**
	Solver	360 (171)	21.7 (3246)	21.7 (3246)
	Total	389	55.9	34.2
Opteron 1.8GHz **3.6 GFLOPS**	Matrix	**10.2 (818)**	12.4 (663)	**21.2 (381)**
	Solver	225 (275)	(	
	Total	235		

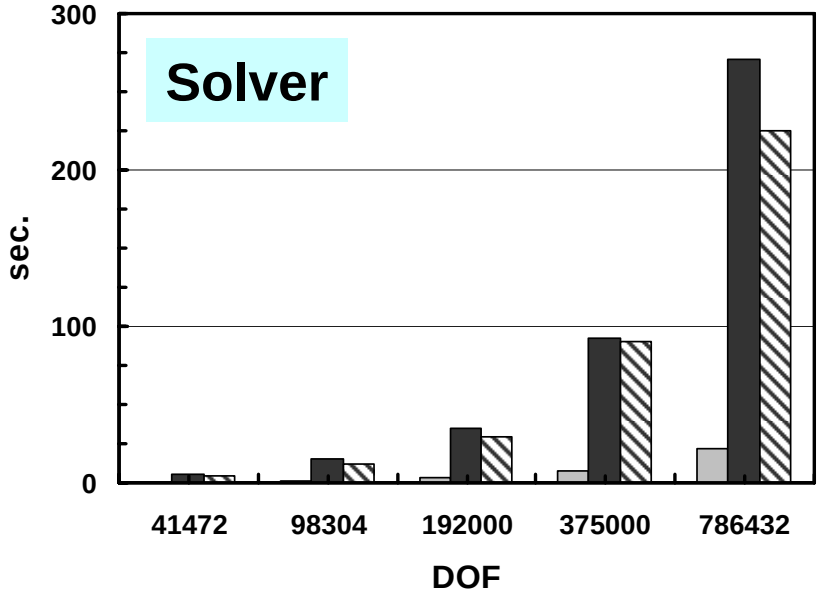
Slower than original because of long innermost loops (data locality has been lost)

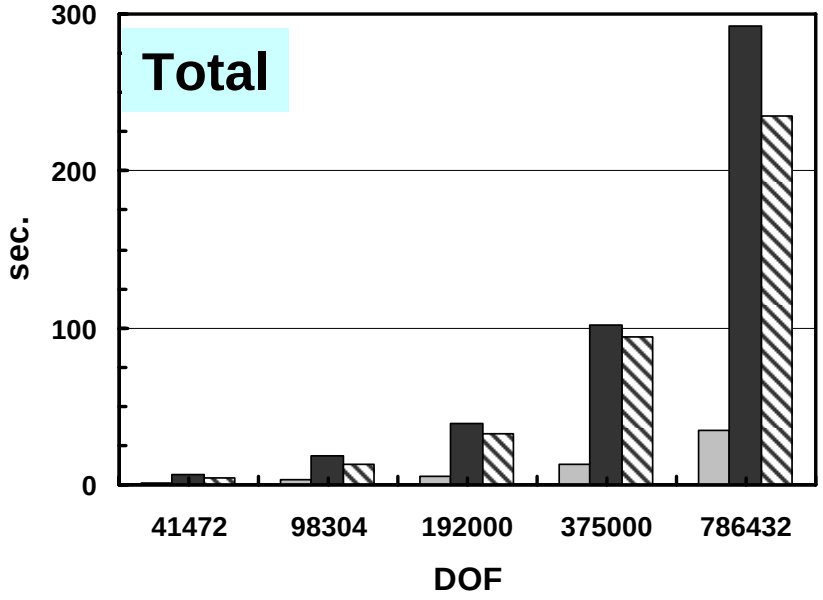
Matrix+Solver



Computation Time vs. Problem Size







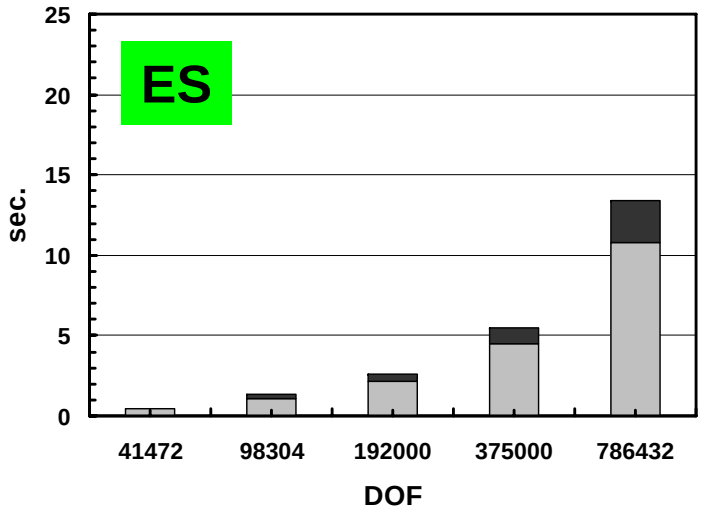
ES (DJDS improved)

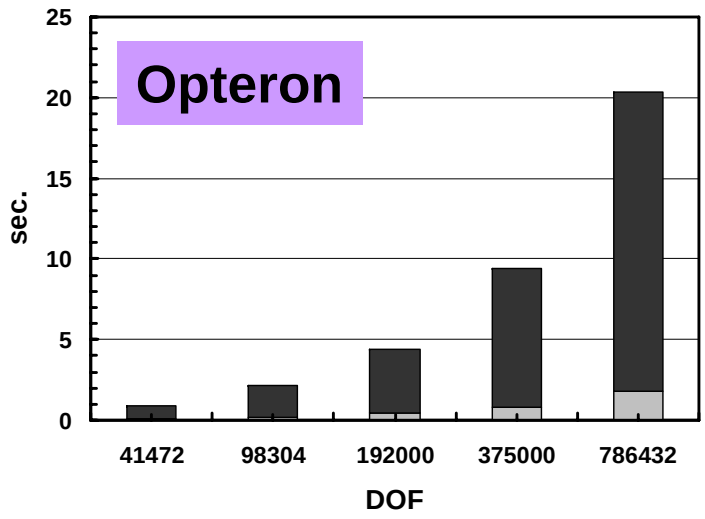
Opteron (DJDS improved)

Opteron (DCRS)

“Matrix” computation time for improved version of DJDS

Integer
Floating





INTEGER/FLOATING parts

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - define “ELEMmat” array
    enddo
  enddo
  enddo
enddo
```

INTEGER

```
do icol= 1, NCOLOR_E_tot
  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - assemble element-matrix
    enddo
  enddo
  enddo

  do ie= 1, 4
  do je= 1, 4
    do ic0= index_COL(icol-1)+1, indexCOL_(icol)
      icel= item_COL(ic0)
      - accumulate element-matrix to global-matrix
    enddo
  enddo
  enddo
enddo
```

FLOATING

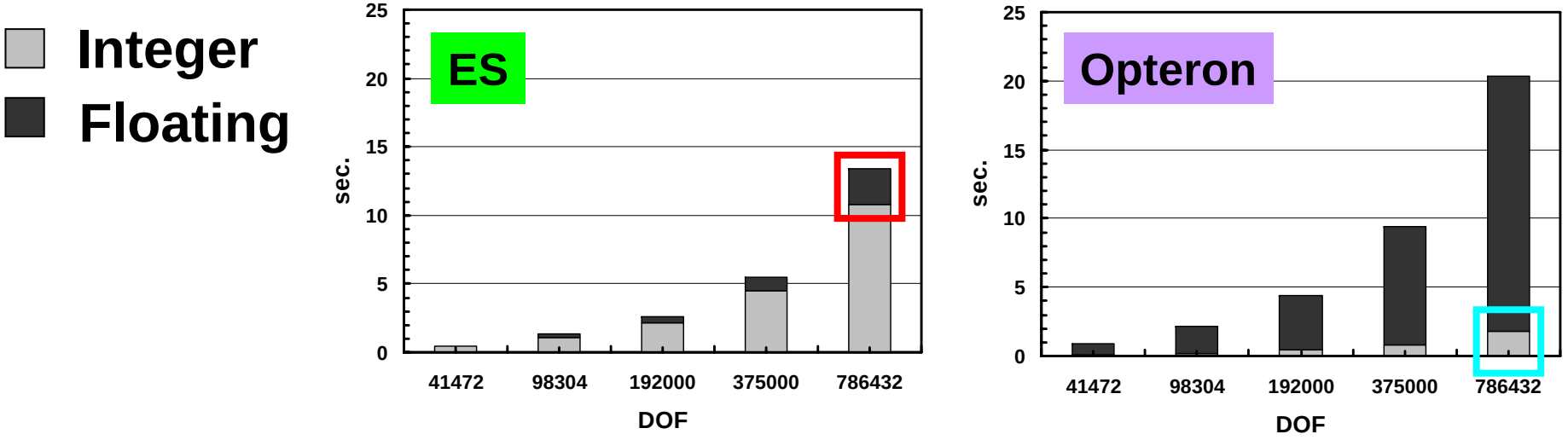
ベクトルプロセッサ用高速化：効果

- **要素の色分け**（Coloring）
- **ループの交換**
- **「**if-then-else**」の解消**
- **プログラム分割**

- ES**で**3**倍近く速くなった**（34.2 sec⇒12.5 sec.）
  - **それでも**Opteron**の最速値**（10.2 sec.）**より遅い**
  - **計算の大部分は「**INTEGER**」の部分にとられている**
    - **分岐が多い：ベクトル化がしにくい**
    - Opteron**ではこの部分は高速**

「理想の（都合の良い）」計算機

- On scalar processor
  - “Integer” operation part
- On vector processor
  - “floating” operation part
  - linear solvers
- Scalar performance of ES (500MHz) is smaller than that of Pentium III



Time for $3 \times 64^3 = 786{,}432$ DOF

		DCRS sec. (MFLOPS)	DJDS improved sec. (MFLOPS)	DJDS virtual sec. (MFLOPS)
ES **8.0 GFLOPS**	Matrix	28.6 (291)	**12.5 (643)**	**1.88 (4431)**
	Solver	360 (171)	21.7 (3246)	21.7 (3246)
	Total	389	34.2	23.6
Opteron 1.8GHz **3.6 GFLOPS**	Matrix	**10.2 (818)**	21.2 (381)	
	Solver	225 (275)	271 (260)	
	Total	235	292	

- 背景
  - HPC-MW (HPC Middleware)
- 有限要素法（FEM）の特徴
  - 線形ソルバーの性能
- 係数行列生成部の最適化
- まとめ

まとめ

- 最適化
  - いろいろな工夫が考えられる
  - 特にベクトルプロセッサの場合「計算量を増やして計算時間を少なくする」アプローチも必要
- ベクトルプロセッサ
  - 「ベクトル化」しないと性能は出ない
  - スカラー処理性能が低いのがネック（～クロック数）
  - 消費電力，温度等の技術的問題点のため，クロック数が上げられないという話もある
- 理想のアーキテクチュアとは？
  - いつも「計算機ありき」，「ハードウェアありき」で，それに対してどのように最適化するか？　ということが主たる話題になってしまう。
  - 専用ハードウェア，アプリケーション側からの提言