Introduction to Parallel Programming for Multicore/Manycore Clusters

Part I: FVM Code

Kengo Nakajima
Information Technology Center
The University of Tokyo

Files on ECCS2012

Files on WEB:
http://nkl.cc.u-tokyo.ac.jp/files/multicore-c.tar
http://nkl.cc.u-tokyo.ac.jp/files/multicore-f.tar

```
>$ cd <$CUR> :  go to a certain directory
>$ cp /home03/skengon/Documents/multicore/multicore-c.tar .
>$ tar xvf multicore-c.tar

>$ cd multicore
```

Please confirm that following directories are created:

```
L1  L2

<$E-L1>, <$E-L2>
```

ECCS

Oakleaf-FX

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- ICCG Solver for Poisson Equations
  - How to run
    - Data Structure
  - Program
    - Initialization
    - Coefficient Matrices
    - ICCG

Target of this Part

- Material: ICCG solver for sparse matrices derived from FVM applications (Finite Volume Method).
- Parallelization on a single node of Oakleaf-FX (FX10) using OpenMP
  - Data Placement
  - Reordering

- Keywords
  - Finite Volume Method (FVM)
  - Sparse Matrices
  - ICCG Method

Target Application

- 3D Poisson Equations

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

- Finite Volume Method (FVM)
  - Arbitrary Shape Meshes, Cell-Centered
  - “Direct” Finite Difference Method
- Boundary Conditions
  - Dirichlet B.C., Volume Flux
- Preconditioned Iterative Solvers
  - Conjugate Gradient + Preconditioner

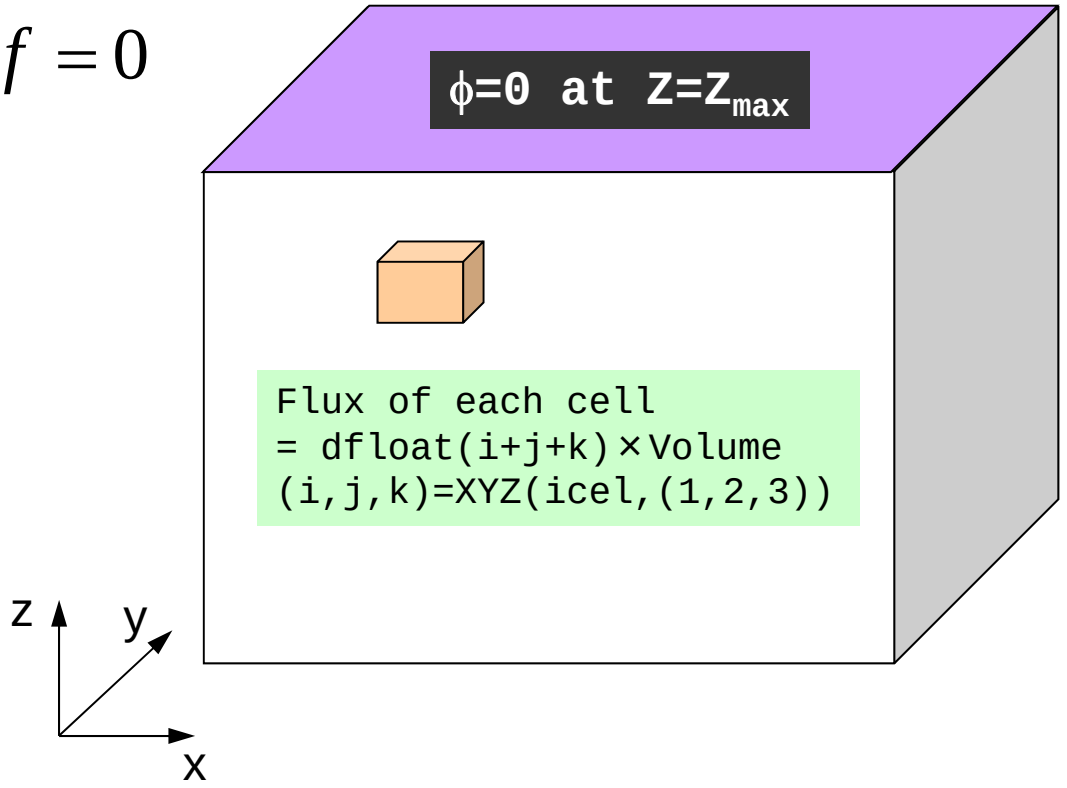
Target Problem: Variables are defined at cell-center's

Poisson Equation

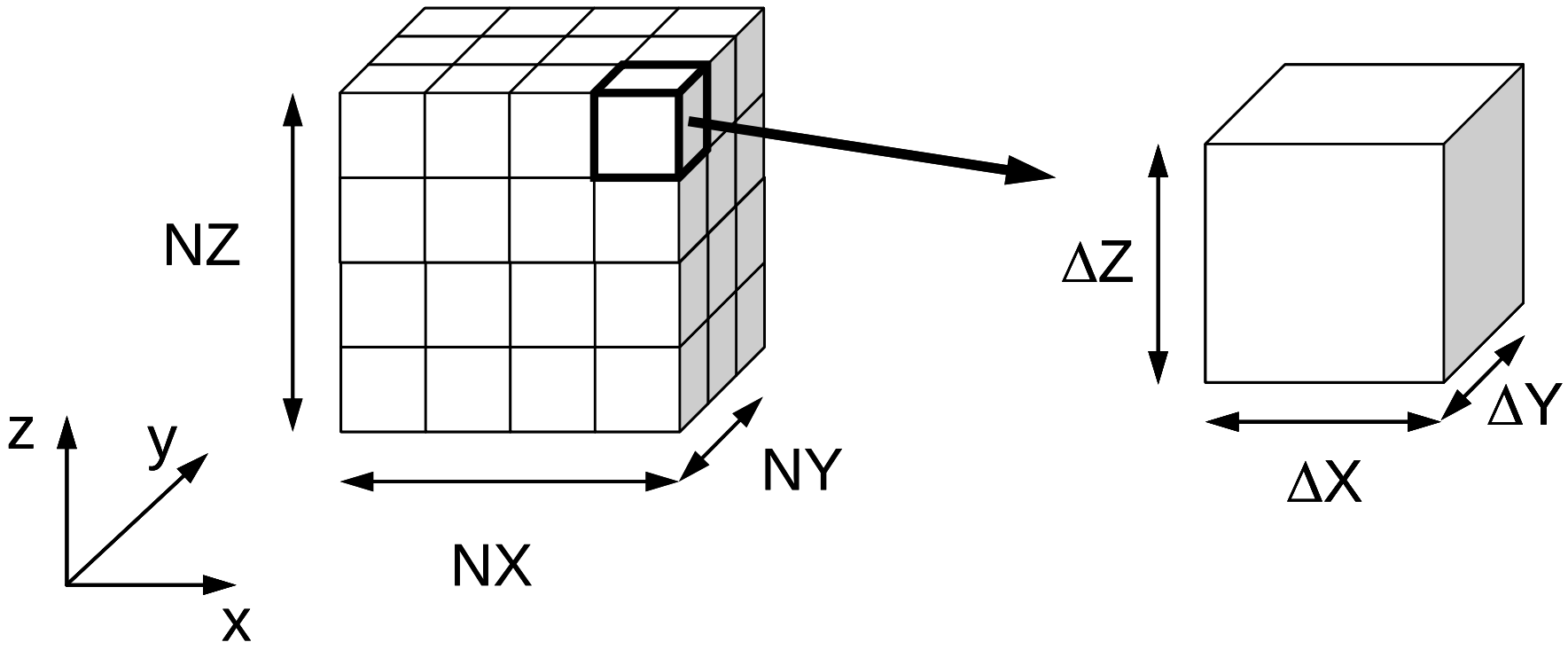
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

B.C.

- Volume Flux
- ϕ=0@Z=Z_{max}



3D Structured Mesh
Internal data structure is “unstructured”



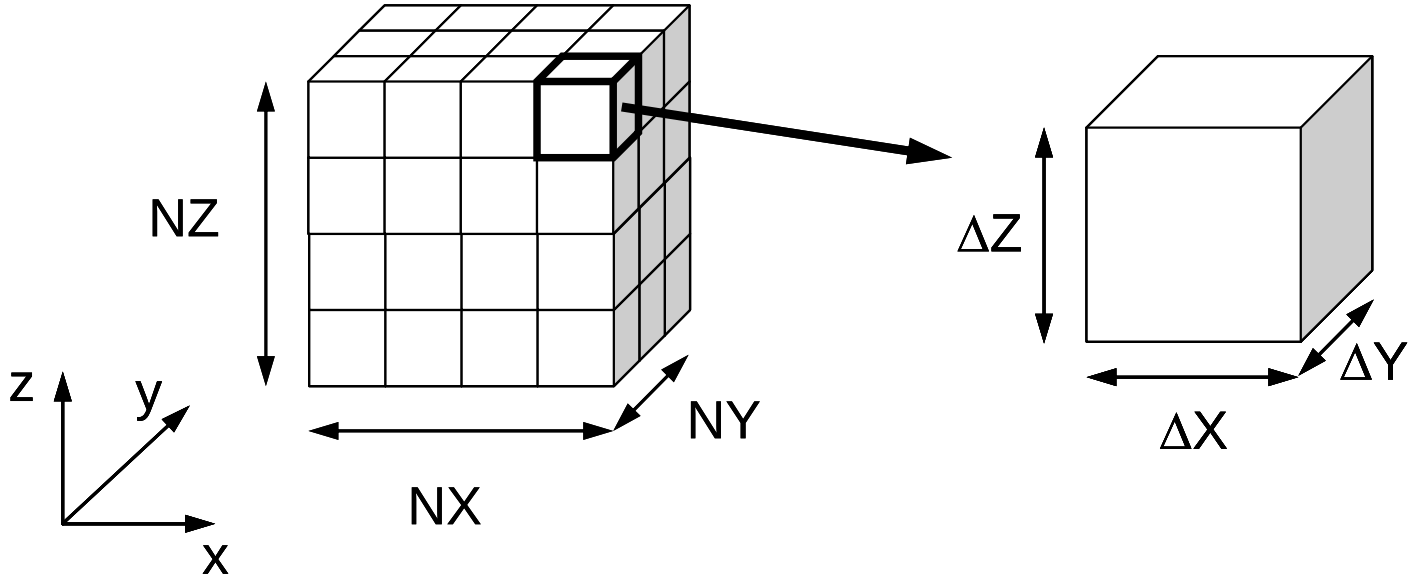
Volume Flux f

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

$$f = dfloat(i_0 + j_0 + k_0)$$

$$i_0 = XYZ(icel,1),$$
$$j_0 = XYZ(icel,2),$$
$$k_0 = XYZ(icel,3)$$

XYZ(icel, k)(k=1,2,3)
Index for location of finite-difference mesh in X-/Y-/Z-axis.



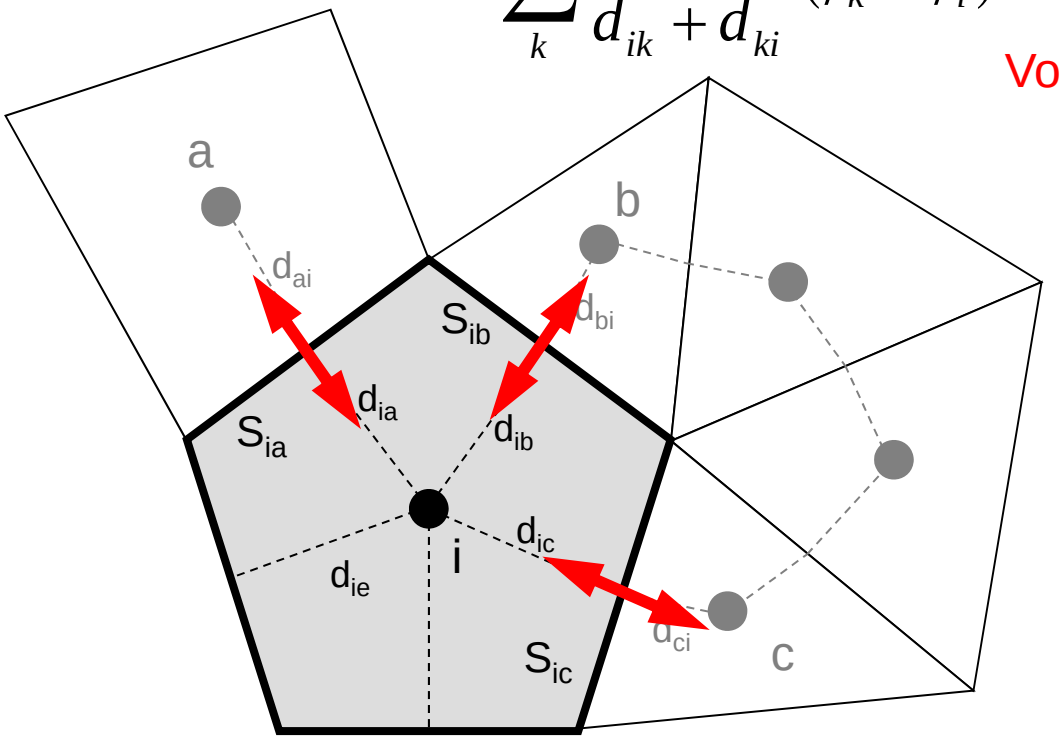
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

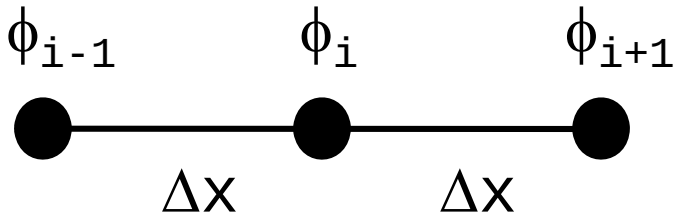
Volume Flux



V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Finite Difference Method (FDM)
Taylor Series Expansion
2nd-Order Central Difference

$$\left(\frac{d^2\phi}{dx^2}\right)_i + Q = 0$$



$$\frac{d}{dx}\left(\frac{d\phi}{dx}\right)_i = \frac{\left(\frac{d\phi}{dx}\right)_{i+1/2} - \left(\frac{d\phi}{dx}\right)_{i-1/2}}{\Delta x} = \frac{\frac{\phi_{i+1}-\phi_i}{\Delta x} - \frac{\phi_i-\phi_{i-1}}{\Delta x}}{\Delta x}$$

$$= \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2}$$

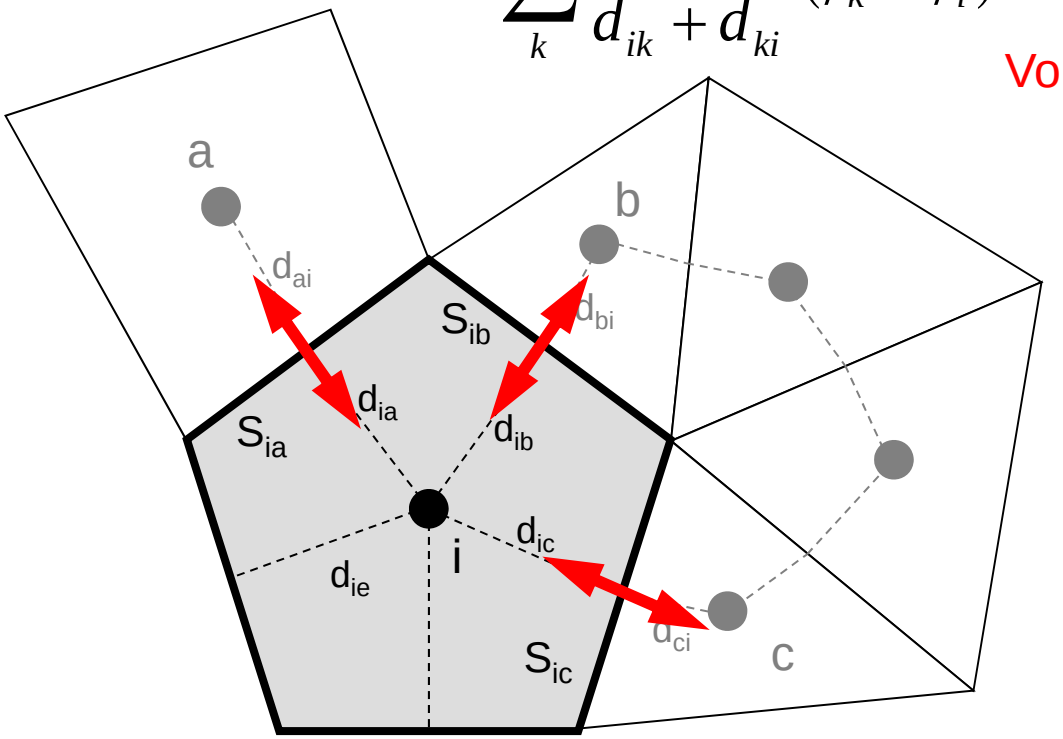
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

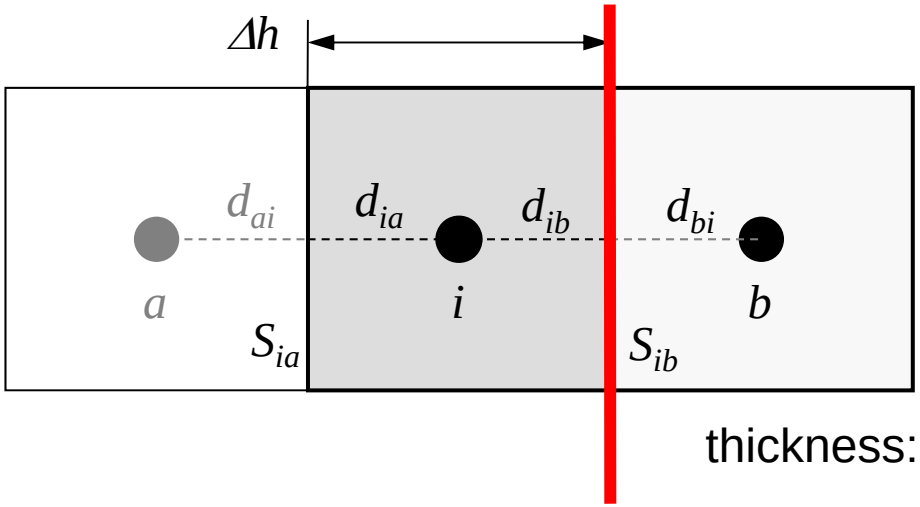
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Comparison with 1D FDM (1/3)



thickness: 1

Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

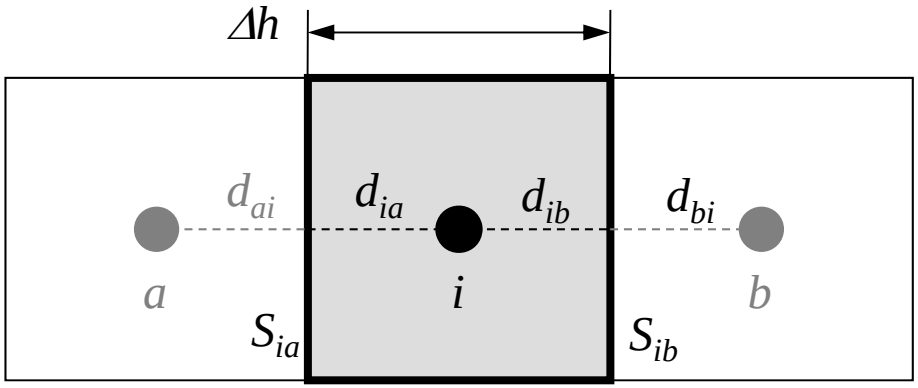
Flux through this surface: Qs_{ib}

$$Qs_{ib} = -\frac{\phi_b - \phi_i}{d_{ib} + d_{bi}} \cdot S_{ib}$$

Fourier's Law

Flux through a surface
= - (gradient of potential)

Comparison with 1D FDM (2/3)



Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

thickness: 1

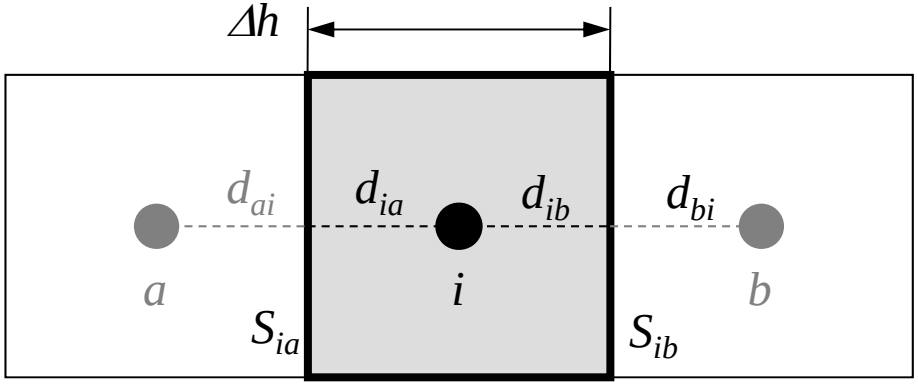
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k - \phi_i\right) + V_i \dot{Q}_i = 0$$

Divided by V_i:

$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k - \phi_i\right) + \dot{Q}_i = 0$$

considering this part

Comparison with 1D FDM (3/3)



Δh x Δh Square Mesh

Surface Area: $S_{ik} = \Delta h$

Volume: $V_i = \Delta h^2$

Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

thickness: 1

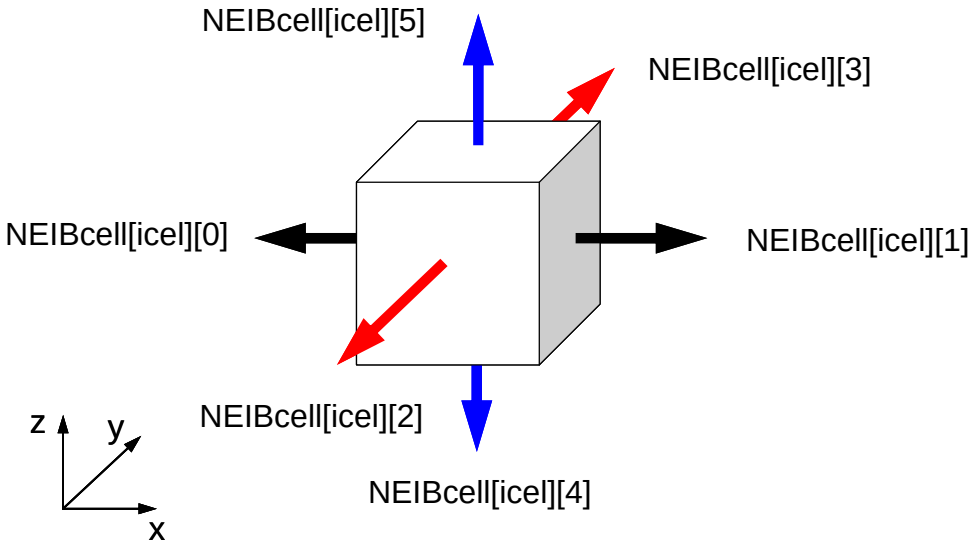
$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i)$$

$$= \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\Delta h}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}(\phi_k-\phi_i)$$

$$= \frac{1}{(\Delta h)^2}(\phi_a-\phi_i)+\frac{1}{(\Delta h)^2}(\phi_b-\phi_i) = \boxed{\frac{\phi_a-2\phi_i+\phi_b}{(\Delta h)^2}}$$

for i-th cell
linear equations

in 3D



$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

Linear Equations

$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

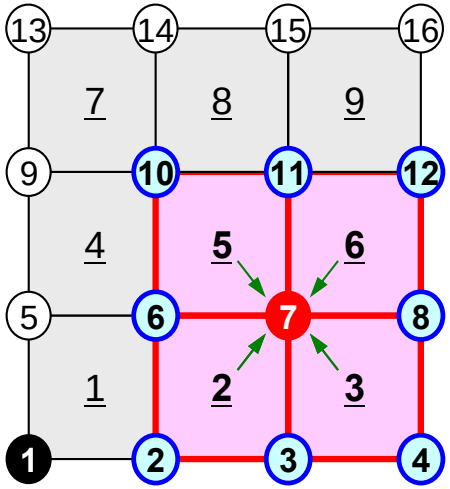
$$\sum_k \frac{S_{icel-k}}{d_{icel-k}}\left(\phi_k-\phi_{icel}\right)=-f_{icel}V_i$$

$$-\left[\sum_k \frac{S_{icel-k}}{d_{icel-k}}\right]\phi_{icel}+\left[\sum_k \frac{S_{icel-k}}{d_{icel-k}}\phi_k\right]=-f_{icel}V_i \quad (icel=1,\ N)$$

Diagonal Off-Diagonal

$$\longrightarrow \quad [A]\{\phi\}=\{f\}$$

Sparse Coef. Matrix in FEM
Many “0’s”



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

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

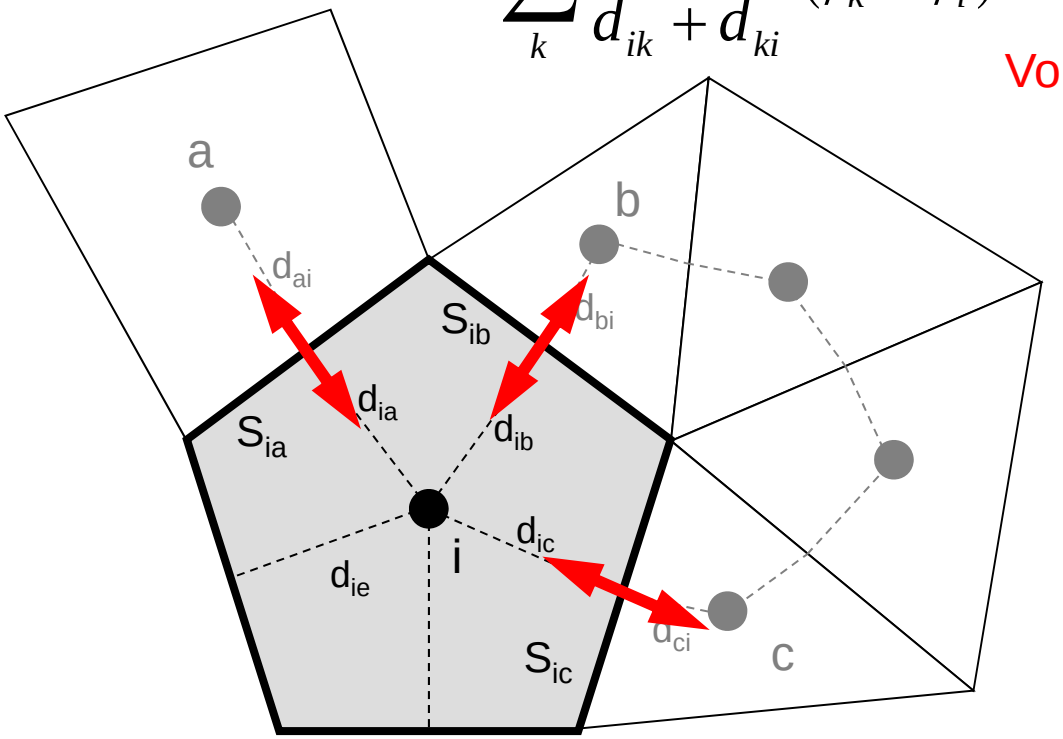
Coef. Matrices for FVM are sparse

Only neighboring cells are considered

Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Sparse Matrix for FVM

- Sparse Matrix
  - Many “0”’s
- Storing all components (e.g. $\mathrm{A}(i,j)$) is not efficient for sparse matrices
  - $\mathrm{A}(i,j)$ is suitable for dense matrices

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

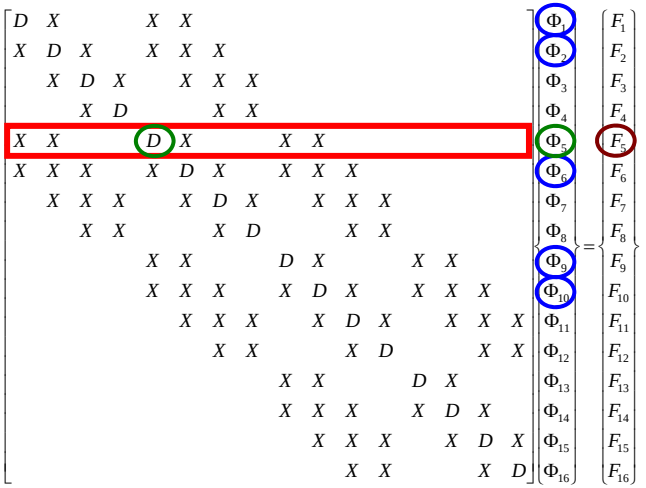
- Number of non-zero off-diagonal components is O(100) in FVM (only 6 in this case)
  - If number of unknowns is 10^8 :
    - $\mathrm{A}(i,j)$: $O(10^{16})$ words
    - Actual Non-zero Components : $O(10^{10})$ words
- Only (really) non-zero off-diag. components should be stored on memory

Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

Diag (i)	Diagonal Components (REAL, i=1~N)
Index(i)	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
Item(k)	Off-Diagonal Components (Corresponding Column ID) （INT, k=1, index(N)）
AMat(k)	Off-Diagonal Components (Value) （ REAL, k=1, index(N) ）

```
{Y}= [A]{X}

do i= 1, N
  Y(i)= Diag(i)*X(i)
  do k= Index(i-1)+1, Index(i)
    Y(i)= Y(i) + Amat(k)*X(Item(k))
  enddo
enddo
```



Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

```
{Q}=[A]{P}

for(i=0;i<N;i++){
    W[Q][i] = Diag[i] * W[P][i];
    for(k=Index[i];k<Index[i+1];k++){
        W[Q][i] += AMat[k]*W[P][Item[k]];
    }
}
```

Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1,N-1} & a_{1,N} \\ a_{21} & a_{22} & & a_{2,N-1} & a_{2,N} \\ \cdots & & \cdots & & \cdots \\ a_{N-1,1} & a_{N-1,2} & & a_{N-1,N-1} & a_{N-1,N} \\ a_{N,1} & a_{N,2} & \cdots & a_{N,N-1} & a_{N,N} \end{bmatrix} \begin{Bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{N-1} \\ x_N \end{Bmatrix} = \begin{Bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{N-1} \\ y_N \end{Bmatrix}$$

```
{Y}= [A]{X}

for(j=0;j<N;i++){
  Y[j]= 0.0;
  for (i=0;i<N;i++){
    Y[j]+= A[j][i]*X[i];
  }
}
```

Compressed Row Storage (CRS)

	1	2	3	4	5	6	7	8
1	1.1	2.4	0	0	3.2	0	0	0
2	4.3	3.6	0	2.5	0	3.7	0	9.1
3	0	0	5.7	0	1.5	0	3.1	0
4	0	4.1	0	9.8	2.5	2.7	0	0
5	3.1	9.5	10.4	0	11.5	0	4.3	0
6	0	0	6.5	0	0	12.4	9.5	0
7	0	6.4	2.5	0	0	1.4	23.1	13.1
8	0	9.5	1.3	9.6	0	3.1	0	51.3

Compressed Row Storage (CRS): C Numbering starts from 0 in program

	0	1	2	3	4	5	6	7
0	1.1 ◎	2.4 ①			3.2 ④			
1	4.3 ◎	3.6 ①		2.5 ③		3.7 ⑤		9.1 ⑦
2			5.7 ②		1.5 ④		3.1 ⑥	
3		4.1 ①		9.8 ③	2.5 ④	2.7 ⑤		
4	3.1 ◎	9.5 ①	10.4 ②		11.5 ④		4.3 ⑥	
5			6.5 ②			12.4 ⑤	9.5 ⑥	
6		6.4 ①	2.5 ②			1.4 ⑤	23.1 ⑥	13.1 ⑦
7		9.5 ①	1.3 ②	9.6 ③		3.1 ⑤		51.3 ⑦

N= 8

Diagonal Components

```
Diag[0]=  1.1
Diag[1]=  3.6
Diag[2]=  5.7
Diag[3]=  9.8
Diag[4]= 11.5
Diag[5]= 12.4
Diag[6]= 23.1
Diag[7]= 51.3
```

Compressed Row Storage (CRS)

	0	1	2	3	4	5	6	7
0	1.1 ◎		2.4 ①			3.2 ④			
1	3.6 ①	4.3 ◎			2.5 ③		3.7 ⑤		9.1 ⑦
2	5.7 ②					1.5 ④		3.1 ⑥	
3	9.8 ③		4.1 ①			2.5 ④	2.7 ⑤		
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②				4.3 ⑥	
5	12.4 ⑤			6.5 ②				9.5 ⑥	
6	23.1 ⑥		6.4 ①	2.5 ②			1.4 ⑤		13.1 ⑦
7	51.3 ⑦		9.5 ①	1.3 ②	9.6 ③		3.1 ⑤		

Compressed Row Storage (CRS)

						# Non-Zero Off-Diag.	
0	1.1 ◎	2.4 ①	3.2 ④			2	Index[0]= 0
1	3.6 ①	4.3 ◎	2.5 ③	3.7 ⑤	9.1 ⑦	4	Index[1]= 2
2	5.7 ②	1.5 ④	3.1 ⑥			2	Index[2]= 6
3	9.8 ③	4.1 ①	2.5 ④	2.7 ⑤		3	Index[3]= 8
4	11.5 ④	3.1 ◎	9.5 ①	10.4 ②	4.3 ⑥	4	Index[4]= 11
5	12.4 ⑤	6.5 ②	9.5 ⑥			2	Index[5]= 15
6	23.1 ⑥	6.4 ①	2.5 ②	1.4 ⑤	13.1 ⑦	4	Index[6]= 17
7	51.3 ⑦	9.5 ①	1.3 ②	9.6 ③	3.1 ⑤	4	Index[7]= 21
							Index[8]= 25

NPLU= 25
(=Index[N])

(Index[i])th~(Index[i+1])th:
Non-Zero Off-Diag. Components corresponding to *i*-th row.

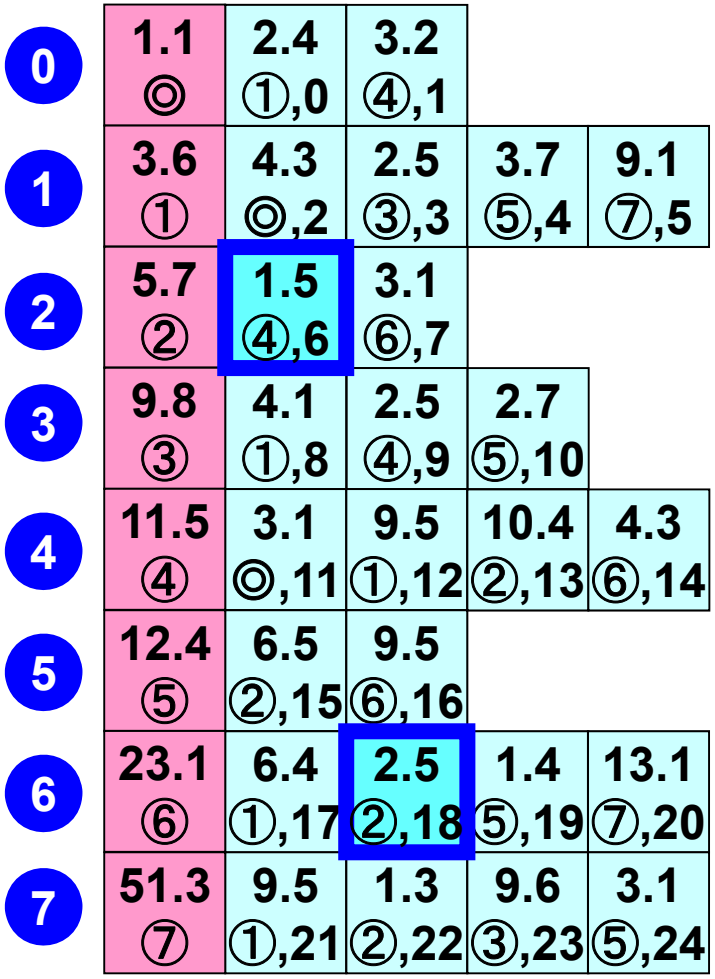
Compressed Row Storage (CRS)

						# Non-Zero Off-Diag.	
							Index[0]= 0
0	1.1 ◎	2.4 ①,0	3.2 ④,1			2	Index[1]= 2
1	3.6 ①	4.3 ◎,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5	4	Index[2]= 6
2	5.7 ②	1.5 ④,6	3.1 ⑥,7			2	Index[3]= 8
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10		3	Index[4]= 11
4	11.5 ④	3.1 ◎,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14	4	Index[5]= 15
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16			2	Index[6]= 17
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20	4	Index[7]= 21
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24	4	Index[8]= 25

NPLU= 25
(=Index[N])

(Index[i])th~(Index[i+1])th:
Non-Zero Off-Diag. Components corresponding to *i*-th row.

Compressed Row Storage (CRS)




```
Item[ 6]= 4, AMat[ 6]= 1.5
Item[18]= 2, AMat[18]= 2.5
```

Compressed Row Storage (CRS)

0	1.1 ◎	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ◎,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ◎,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

`Diag [i]`	Diagonal Components (REAL, i=0~N-1)
`Index[i[`	Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)
`Item[k]`	Off-Diagonal Components (Corresponding Column ID) (INT, k=0, index[N])
`Amat[k]`	Off-Diagonal Components (Value) (REAL, k=0, index[N])

```
{Y}=[A]{X}

for(i=0;i<N;i++){
    Y[i] = Diag[i] * X[i];
    for(k=Index[i];k<Index[i+1];k++){
        Y[i] += AMat[k]*X[Item[k]];
      }
}
```

- Background
  - Finite Volume Method
  - **Preconditioned Iterative Solvers**
- ICCG Solver for Poisson Equations
  - How to run
    - Data Structure
  - Program
    - Initialization
    - Coefficient Matrices
    - ICCG
- OpenMP

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations **Ax=b** is the most important and **expensive** part of various types of scientific computing.
  - for both linear and nonlinear applications
- Various types of methods proposed & developed.
  - for dense and sparse matrices
  - classified into ***direct*** and ***iterative*** methods
- Dense Matrices:密行列: Globally Coupled Problems
  - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices:疎行列: Locally Defined Problems
  - FEM, FDM, DEM, MD (solid), BEM w/FMM

Direct Method
直接法

- Gaussian Elimination/LU Factorization
  - compute A^{-1} directly.

Good

- Robust for wide range of applications.
- Good for both dense and sparse matrices

Bad

- More expensive than iterative methods (memory, CPU)
  - not scalable

What is Iterative Method ?
反復法

Linear Equations
連立一次方程式

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\mathbf{A} \qquad \mathbf{x} \qquad \mathbf{b}$$

Initial Solution
初期解

$$\mathbf{x}^{(0)} = \begin{pmatrix} x_1^{(0)} \\ x_2^{(0)} \\ \vdots \\ x_n^{(0)} \end{pmatrix}$$

Starting from a initial vector $\mathrm{x}^{(0)}$, iterative method obtains the final converged solutions by iterations

$$\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \cdots$$

Iterative Method
反復法

- Stationary Method
  - Only x (solution vector) changes during iterations.
  - SOR, Gauss-Seidel, Jacobi
  - Generally slow, impractical

$$\mathbf{Ax} = \mathbf{b} \Rightarrow$$
$$\mathbf{x}^{(k+1)} = \mathbf{Mx}^{(k)} + \mathbf{Nb}$$

- Non-Stationary Method
  - With restriction/optimization conditions
  - Krylov-Subspace
  - CG: Conjugate Gradient
  - BiCGSTAB: Bi-Conjugate Gradient Stabilized
  - GMRES: Generalized Minimal Residual

Iterative Method (cont.)

Good

- Less expensive than direct methods, especially in memory.
- Suitable for parallel and vector computing.

Bad

- Convergence strongly depends on problems, boundary conditions (condition number etc.)
- Preconditioning is required : Key Technology for Parallel FEM

Non-Stationary/Krylov Subspace Method (1/2)

非定常法・クリロフ部分空間法

$$\mathbf{Ax}=\mathbf{b}\Rightarrow\mathbf{x}=\mathbf{b}+\left(\mathbf{I}-\mathbf{A}\right)\mathbf{x}$$

Compute $\mathbf{x_0}$, $\mathbf{x_1}$, $\mathbf{x_2}$, ..., $\mathbf{x_k}$ by the following iterative procedures:

$$\begin{aligned}\mathbf{x}_k &= \mathbf{b}+\left(\mathbf{I}-\mathbf{A}\right)\mathbf{x}_{k-1}\\ &= \left(\mathbf{b}-\mathbf{A}\mathbf{x}_{k-1}\right)+\mathbf{x}_{k-1}\\ &= \mathbf{r}_{k-1}+\mathbf{x}_{k-1}\end{aligned}$$

where $\mathbf{r}_k=\mathbf{b}-\mathbf{A}\mathbf{x}_k$ **: residual**

$$\mathbf{x}_k=\mathbf{x}_0+\sum_{i=0}^{k-1}\mathbf{r}_i$$

$$\begin{aligned}\mathbf{r}_k &= \mathbf{b}-\mathbf{A}\mathbf{x}_k=\mathbf{b}-\mathbf{A}\left(\mathbf{r}_{k-1}+\mathbf{x}_{k-1}\right)\\ &= \left(\mathbf{b}-\mathbf{A}\mathbf{x}_{k-1}\right)-\mathbf{A}\mathbf{r}_{k-1}=\mathbf{r}_{k-1}-\mathbf{A}\mathbf{r}_{k-1}=\left(\mathbf{I}-\mathbf{A}\right)\mathbf{r}_{k-1}\end{aligned}$$

Non-Stationary/Krylov Subspace Method (2/2)

非定常法・クリロフ部分空間法

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=0}^{k-2} \left(\mathbf{I} - \mathbf{A}\right)\mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=1}^{k-1} \left(\mathbf{I} - \mathbf{A}\right)^i \mathbf{r}_0$$

$$\mathbf{z}_k = \mathbf{r}_0 + \sum_{i=1}^{k-1} \left(\mathbf{I} - \mathbf{A}\right)^i \mathbf{r}_0 = \left[\mathbf{I} + \sum_{i=1}^{k-1} \left(\mathbf{I} - \mathbf{A}\right)^i\right]\mathbf{r}_0$$

$\mathbf{z_k}$ is a vector which belongs to kth Krylov Subspace（クリロフ部分空間）, approximate solution vector $\mathbf{x_k}$ is derived by the Krylov Subspace:

$$\left[\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\right]$$

Conjugate Gradient Method
共役勾配法

- Conjugate Gradient: CG
  - Most popular “non-stationary” iterative method
- for Symmetric Positive Definite (SPD) Matrices
  - 対称正定
  - $\{x\}^{\mathrm{T}}[A]\{x\}>0$ for arbitrary $\{x\}$
  - All of diagonal components, eigenvaules and leading principal minors > 0（主小行列式・首座行列式）
  - Matrices of Galerkin-based FEM: heat conduction, Poisson, static linear elastic problems
- Algorithm
  - “Steepest Descent Method”
  - $\boldsymbol{x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}}$
    - $x^{(i)}$: solution, $p^{(i)}$: search direction, α_i: coefficient
  - Solution $\{x\}$ minimizes $\boldsymbol{\{x-y\}^{\mathrm{T}}[A]\{x-y\}}$, where $\{y\}$ is exact solution.

$$\det\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & a_{24} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & a_{34} & \cdots & a_{3n} \\ a_{41} & a_{42} & a_{43} & a_{44} & \cdots & a_{4n} \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} & \cdots & a_{nn} \end{bmatrix}$$

Procedures of Conjugate Gradient

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

- Mat-Vec. Multiplication
- Dot Products
- DAXPY (Double Precision: a{X} + {Y})

$x^{(i)}$: Vector
α_i : Scalar

Procedures of Conjugate Gradient

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector
α_i : Scalar

Procedures of Conjugate Gradient

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

- Mat-Vec. Multiplication
- Dot Products
- DAXPY

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

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

- Mat-Vec. Multiplication
- Dot Products
- DAXPY
  - Double
  - {y}= a{x} + {y}

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  $z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\, z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\, p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

$x^{(i)}$: Vector
α_i : Scalar

Derivation of CG Algorithm (1/5)

Solution x minimizes the following equation if y is the exact solution ($Ay=b$)

$$(x-y)^T[A](x-y)$$

$$(x-y)^T[A](x-y)=(x,Ax)-(y,Ax)-(x,Ay)+(y,Ay)$$
$$=(x,Ax)-2(x,Ay)+(y,Ay)=(x,Ax)-2(x,b)+(y,b)$$

Const.

Therefore, the solution x minimizes the following $f(x)$:

$$f(x)=\frac{1}{2}(x,Ax)-(x,b)$$

$$f(x+h)=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$

Arbitrary vector h

$$f(x)=\frac{1}{2}(x,Ax)-(x,b)$$

$$f(x+h)=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$

Arbitrary vector h

$$f(x+h)=\frac{1}{2}(x+h,A(x+h))-(x+h,b)$$

$$=\frac{1}{2}(x+h,Ax)+\frac{1}{2}(x+h,Ah)-(x,b)-(h,b)$$

$$=\frac{1}{2}(x,Ax)+\frac{1}{2}(h,Ax)+\frac{1}{2}(x,Ah)+\frac{1}{2}(h,Ah)-(x,b)-(h,b)$$

$$=\frac{1}{2}(x,Ax)-(x,b)+(h,Ax)-(h,b)+\frac{1}{2}(h,Ah)$$

$$=f(x)+(h,Ax-b)+\frac{1}{2}(h,Ah)$$

Derivation of CG Algorithm (2/5)

CG method minimizes $f(x)$ at each iteration.
Assume that approximate solution: $x^{(0)}$, and
search direction vector $p^{(k)}$ is defined at k-th iteration.

$$x^{(k+1)} = x^{(k)} + \alpha_k p^{(k)}$$

Minimization of $f(x^{(k+1)})$ is done as follows:

$$f\left(x^{(k)} + \alpha_k p^{(k)}\right) = \frac{1}{2}\alpha_k{}^2\left(p^{(k)}, Ap^{(k)}\right) - \alpha_k\left(p^{(k)}, b - Ax^{(k)}\right) + f\left(x^{(k)}\right)$$

$$\frac{\partial f\left(x^{(k)} + \alpha_k p^{(k)}\right)}{\partial \alpha_k} = 0 \Rightarrow \alpha_k = \frac{\left(p^{(k)}, b - Ax^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} \quad \text{(1)}$$

$r^{(k)} = b - Ax^{(k)}$ residual vector

Derivation of CG Algorithm (3/5)

Residual vector at $(k+1)$-th iteration: $r^{(k+1)} = b - Ax^{(k+1)}, r^{(k)} = b - Ax^{(k)}$

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)} \quad \textbf{(2)}$$

$$r^{(k+1)} - r^{(k)} = Ax^{(k+1)} - Ax^{(k)} = \alpha_k A p^{(k)}$$

Search direction vector p is defined by the following recurrence formula:

$$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}, r^{(0)} = p^{(0)} \quad \textbf{(3)}$$

It's lucky if we can get exact solution y at $(k+1)$-th iteration:

$$y = x^{(k+1)} + \alpha_{k+1} p^{(k+1)}$$

Derivation of CG Algorithm (4/5)

BTW, we have the following (convenient) orthogonality relation:

$$\left(Ap^{(k)}, y - x^{(k+1)}\right) = 0$$

$$\begin{aligned}
&\left(Ap^{(k)}, y - x^{(k+1)}\right) = \left(p^{(k)}, Ay - Ax^{(k+1)}\right) = \left(p^{(k)}, b - Ax^{(k+1)}\right) \\
&= \left(p^{(k)}, b - A\left[x^{(k)} + \alpha_k p^{(k)}\right]\right) = \left(p^{(k)}, b - Ax^{(k)} - \alpha_k Ap^{(k)}\right) \\
&= \left(p^{(k)}, r^{(k)} - \alpha_k Ap^{(k)}\right) = \left(p^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, Ap^{(k)}\right) = 0
\end{aligned}$$

$$\because \alpha_k = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

Thus, following relation is obtained:

$$\left(Ap^{(k)}, y - x^{(k+1)}\right) = \left(Ap^{(k)}, \alpha_{k+1} p^{(k+1)}\right) = 0 \Rightarrow \left(p^{(k+1)}, Ap^{(k)}\right) = 0$$

Derivation of CG Algorithm (5/5)

$$\left(p^{(k+1)}, Ap^{(k)}\right) = \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(k)}\right) = \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k \left(p^{(k)}, Ap^{(k)}\right) = 0$$

$$\Rightarrow \beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} \quad \text{(4)}$$

$\left(p^{(k+1)}, Ap^{(k)}\right) = 0$ $p^{(k)}$ and $p^{(k+1)}$ are “conjugate（共役）” for matrix A

```
Compute p(0)=r(0)= b-[A]x(0)
for i= 1, 2, …
    calc. α(i-1)
    x(i)= x(i-1) + α(i-1)p(i-1)
    r(i)= r(i-1) – α(i-1)[A]p(i-1)

    check convergence |r|
    (if not converged)
    calc. β(i-1)
    p(i)= r(i) + β(i-1) p(i-1)
end
```

$$\alpha_{i-1} = \frac{\left(p^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

$$\beta_{i-1} = \frac{-\left(r^{(i)}, Ap^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

Properties of CG Algorithm

Following “conjugate（共役）” relationship is obtained for arbitrary (i,j):

$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \left(i \neq j\right)$$

Following relationships are also obtained for $p^{(k)}$ and $r^{(k)}$:

$$\left(r^{(i)}, r^{(j)}\right) = 0 \left(i \neq j\right), \quad \left(p^{(k)}, r^{(k)}\right) = \left(r^{(k)}, r^{(k)}\right)$$

In N-dimensional space, only N sets of orthogonal and linearly independent residual vector $r^{(k)}$. This means CG method converges after N iterations if number of unknowns is N. Actually, round-off error sometimes affects convergence.

Proof (1/3)

Mathematical Induction

数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \left(i \neq j\right)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \left(i \neq j\right)$$

(1) $$\alpha_k = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

(2) $$r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$$

(3) $$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}, \; r^{(0)} = p^{(0)}$$

(4) $$\beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

Proof (2/3)

Mathematical Induction
数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \ (i \neq j)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \ (i \neq j) \qquad (*)$$

$(*)$ is satisfied for $i \leq k,\ j \leq k$ where $i \neq j$

if $i < k$

$$\left(r^{(k+1)}, r^{(i)}\right) = \left(r^{(i)}, r^{(k+1)}\right) \overset{(2)}{=} \left(r^{(i)}, r^{(k)} - \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} -\alpha_k \left(r^{(i)}, Ap^{(k)}\right) \overset{(4)}{=} -\alpha_k \left(p^{(i)} - \beta_{i-1} p^{(i-1)}, Ap^{(k)}\right)$$
$$= -\alpha_k \left(p^{(i)}, Ap^{(k)}\right) + \alpha_k \beta_{i-1} \left(p^{(i-1)}, Ap^{(k)}\right) \overset{(*)}{=} 0$$

if $i = k$

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k Ap^{(k)}\right)$$
$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, Ap^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)}\right)$$
$$= -\beta_{k-1} \left(p^{(k-1)}, r^{(k)}\right) \overset{(2)}{=} -\beta_{k-1} \left(p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} Ap^{(k-1)}\right)$$
$$= -\beta_{k-1} \left\{\left(p^{(k-1)}, r^{(k-1)}\right) - \alpha_{k-1} \left(p^{(k-1)}, Ap^{(k-1)}\right)\right\} \overset{(1)}{=} 0$$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

Proof (3/3)

Mathematical Induction

数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \ (i \neq j)$$
$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \ (i \neq j) \qquad (*)$$

$(*)$ is satisfied for $i \leq k,\ j \leq k$ where $i \neq j$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k Ap^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

if $i < k$

$$\left(p^{(k+1)}, Ap^{(i)}\right) \overset{(3)}{=} \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}\right)$$
$$\overset{(*)}{=} \left(r^{(k+1)}, Ap^{(i)}\right)$$
$$\overset{(2)}{=} \frac{1}{\alpha_k}\left(r^{(k+1)}, r^{(i)} - r^{(i-1)}\right) = 0$$

if $i = k$

$$\left(p^{(k+1)}, Ap^{(k)}\right) \overset{(3)}{=} \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k \left(p^{(k)}, Ap^{(k)}\right)$$
$$\overset{(4)}{=} 0$$

$$\left(r^{(k+1)}, r^{(k)}\right) = 0$$

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k A p^{(k)}\right)$$

$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k A p^{(k)}\right)$$

$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, A p^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right) = 0$$

$$\therefore \left(r^{(k)}, r^{(k)}\right) = \left(p^{(k)}, r^{(k)}\right)$$

(1) $\alpha_k = \dfrac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, A p^{(k)}\right)}$

(2) $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$

(3) $p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)}$

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, A p^{(k)}\right)}{\left(p^{(k)}, A p^{(k)}\right)}$

α_k, β_k

Usually, we use simpler definitions of α_k, β_k as follows:

$$\alpha_k = \frac{\left(p^{(k)}, b - Ax^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(p^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(r^{(k)}, r^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$$

$$\because \left(p^{(k)}, r^{(k)}\right) = \left(r^{(k)}, r^{(k)}\right)$$

$$\beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} = \frac{\left(r^{(k+1)}, r^{(k+1)}\right)}{\left(r^{(k)}, r^{(k)}\right)}$$

$$\because \left(r^{(k+1)}, Ap^{(k)}\right) = \frac{\left(r^{(k+1)}, r^{(k)} - r^{(k+1)}\right)}{\alpha_k} = -\frac{\left(r^{(k+1)}, r^{(k+1)}\right)}{\alpha_k}$$

Procedures of Conjugate Gradient

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

$x^{(i)}$: Vector

α_i : Scalar

$$\beta_{i-1} = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(r^{(i-2)}, r^{(i-2)}\right)} \quad \left(= \rho_{i-1}\right) \quad \left(= \rho_{i-2}\right)$$

$$\alpha_i = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i)}, Ap^{(i)}\right)} \quad \left(= \rho_{i-1}\right)$$

Preconditioning for Iterative Solvers

- Convergence rate of iterative solvers strongly depends on the spectral properties (eigenvalue distribution) of the coefficient matrix A.
  - Eigenvalue distribution is small, eigenvalues are close to 1
  - In “ill-conditioned” problems, “condition number” (ratio of max/min eigenvalue if A is symmetric) is large（条件数）.
- A preconditioner M (whose properties are similar to those of A) transforms the linear system into one with more favorable spectral properties （前処理）
  - M transforms Ax=b into A'x=b' where $A'=M^{-1}A$, $b'=M^{-1}b$
  - If M~A, $M^{-1}A$ is close to identity matrix.
  - If $M^{-1}=A^{-1}$, this is the best preconditioner (Gaussian Elim.)
  - Generally, A'x’=b’ where $A'=M_L^{-1}AM_R^{-1}$, $b'=M_L^{-1}b$, $x'=M_Rx$
  - M_L/M_R: Left/Right Preconditioning（左／右前処理）

Preconditioned CG Solver

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\, z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\, p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

$[M]= [M_1][M_2]$

$[A']x'=b'$
$[A']=[M_1]^{-1}[A][M_2]^{-1}$
$x'=[M_2]x,\ b'=[M_1]^{-1}b$

$p'=>[M_2]p,\ r'=>[M_1]^{-1}r$

$p'^{(i)}= r'^{(i-1)} + \beta'_{i-1}\, p'^{(i-1)}$

$[M_2]p^{(i)}= [M_1]^{-1}r^{(i-1)} + \beta'_{i-1}\, [M_2]p^{(i-1)}$

$p^{(i)}= [M_2]^{-1}[M_1]^{-1}r^{(i-1)} + \beta'_{i-1}\, p^{(i-1)}$
$p^{(i)}= [M]^{-1}r^{(i-1)} + \beta'_{i-1}\, p^{(i-1)}$

$\beta'_{i-1}= ([M]^{-1}r^{(i-1)}, r^{(i-1)})/$
$\quad ([M]^{-1}r^{(i-2)}, r^{(i-2)})$
$\alpha'_{i-1}= ([M]^{-1}r^{(i-1)}, r^{(i-1)})/$
$\quad (\ p^{(i-1)}, [A]p^{(i-1)})$

Preconditioned Conjugate Gradient Method（PCG）

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

Solving the following equation:

$$\{z\} = [M]^{-1}\{r\}$$

“Approximate Inverse Matrix”

$$[M]^{-1} \approx [A]^{-1}, \quad [M] \approx [A]$$

Ultimate Preconditioning:
Inverse Matrix

$$[M]^{-1} = [A]^{-1}, \quad [M] = [A]$$

Diagonal Scaling: Simple but weak

$$[M]^{-1} = [D]^{-1}, \quad [M] = [D]$$

Diagonal Scaling, Point-Jacobi

$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

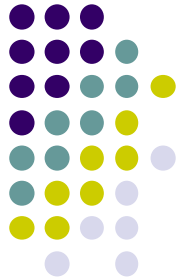
- **solve $[M]z^{(i-1)} = r^{(i-1)}$** is very easy.
- Provides fast convergence for simple problems.

ILU(0), IC(0)

- Widely used Preconditioners for Sparse Matrices
  - Incomplete LU Factorization
  - Incomplete Cholesky Factorization (for Symmetric Matrices)

- Incomplete Direct Method
  - Even if original matrix is sparse, inverse matrix is not necessarily sparse.
  - **fill-in**
  - ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices

Full LU Factorization (or LU Decomposition)

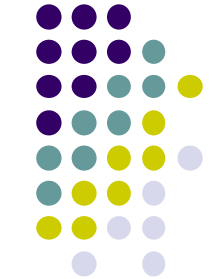


- Direct Method
  - A^{-1} is calculated directly
  - A^{-1} can be saved
  - Fill-in's
- LU factorization

Incomplete LU Factorization (ILU)

- ILU factorization
  - Incomplete LU factorization
- Generation of fill-in's is controlled
  - Preconditioning method
  - Incomplete Inverse Matrix, "Weaker" Direct Method
  - ILU(0): NO fill-in's

Solving Linear Equations by LU Factorization



[A] is decomposed to the following form of [L][U]:

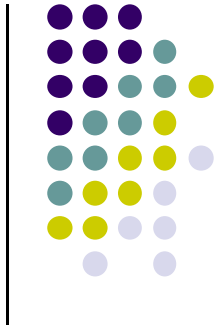
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\ l_{21} & 1 & 0 & \cdots & 0 \\ l_{31} & l_{32} & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & l_{n3} & \cdots & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ 0 & u_{22} & u_{23} & \cdots & u_{2n} \\ 0 & 0 & u_{33} & \cdots & u_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & u_{nn} \end{pmatrix}$$

$$\mathbf{A} = \mathbf{LU}$$

L : Lower triangular part of matrix A

U : Upper triangular part of matrix A

Linear Equations in Matrix Form



General linear equations with “n” unknowns:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

Matrix form:

$$\begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix} \Leftrightarrow \mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} \qquad \mathbf{x} \qquad \mathbf{b}$$

Solving Ax=b by LU Factorization

1 $\mathbf{A} = \mathbf{LU}$ Compute [L] and [U]

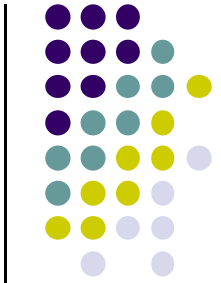
2 $\mathbf{Ly} = \mathbf{b}$ Solve $\mathbf{Ly=b}$

3 $\mathbf{Ux} = \mathbf{y}$ Solve $\mathbf{Ux=y}$

This $\mathbf{x}$ is solution of $\mathbf{Ax} = \mathbf{b}$

$$\because \mathbf{Ax} = \mathbf{LUx} = \mathbf{Ly} = \mathbf{b}$$

Solving Ly=b: Forward Substitution

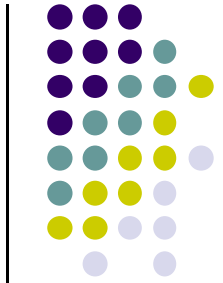


$$\mathbf{Ly} = \mathbf{b} \quad \Longleftrightarrow \quad \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$$

$$\Updownarrow$$

$$\begin{aligned} y_1 &= b_1 \\ l_{21} y_1 + y_2 &= b_2 \\ &\vdots \\ l_{n1} y_1 + l_{n2} y_2 + \cdots + y_n &= b_n \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - l_{21} y_1 \\ &\vdots \\ y_n &= b_n - l_{n1} y_1 - l_{n2} y_2 = b_n - \sum_{i=1}^{n-1} l_{ni} y_i \end{aligned}$$

Solving Ux=y: Backward Substitution



$$\mathbf{Ux} = \mathbf{y} \quad \Longleftrightarrow \quad \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}$$

$$\Longleftrightarrow$$

$$u_{nn} x_n = y_n$$

$$u_{n-1,n-1} x_{n-1} + u_{n-1,n} x_n = y_{n-1}$$

$$\vdots$$

$$u_{11} x_1 + u_{12} x_2 + \cdots + u_{1n} x_n = y_1$$

$$\Longleftrightarrow$$

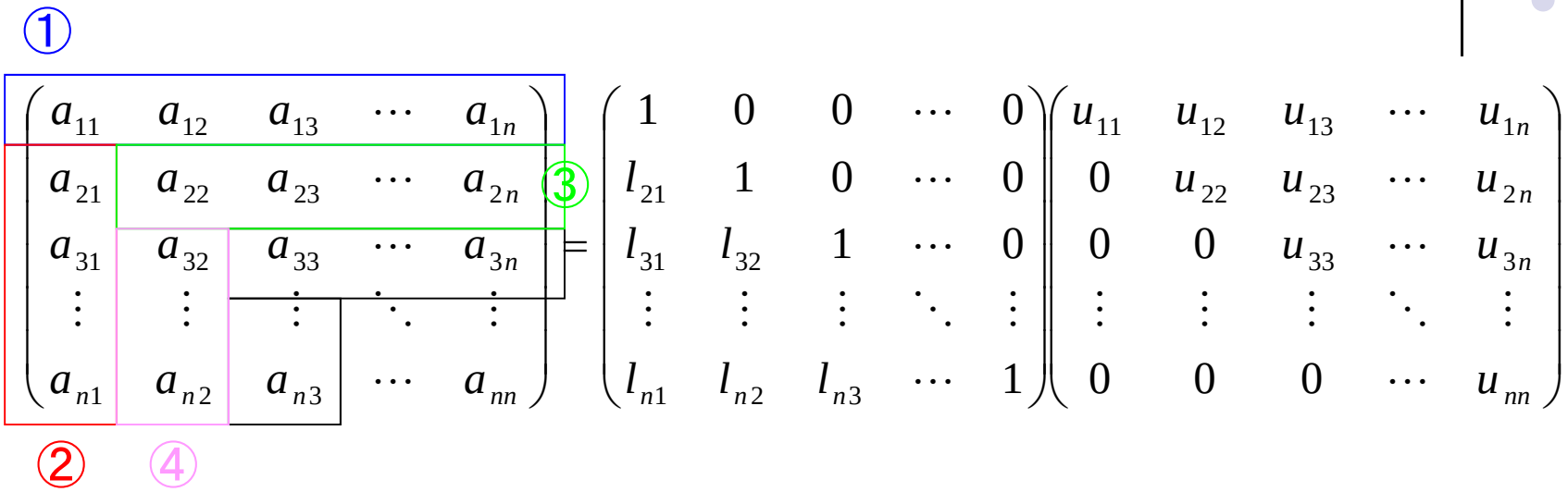
$$x_n = y_n / u_{nn}$$

$$x_{n-1} = (y_{n-1} - u_{n-1,n} x_n) / u_{n-1,n-1}$$

$$\vdots$$

$$x_1 = \left(y_1 - \sum_{i=2}^{n} u_{1j} x_j \right) / u_{11}$$

How to calculate LU Factorization/Decomposition



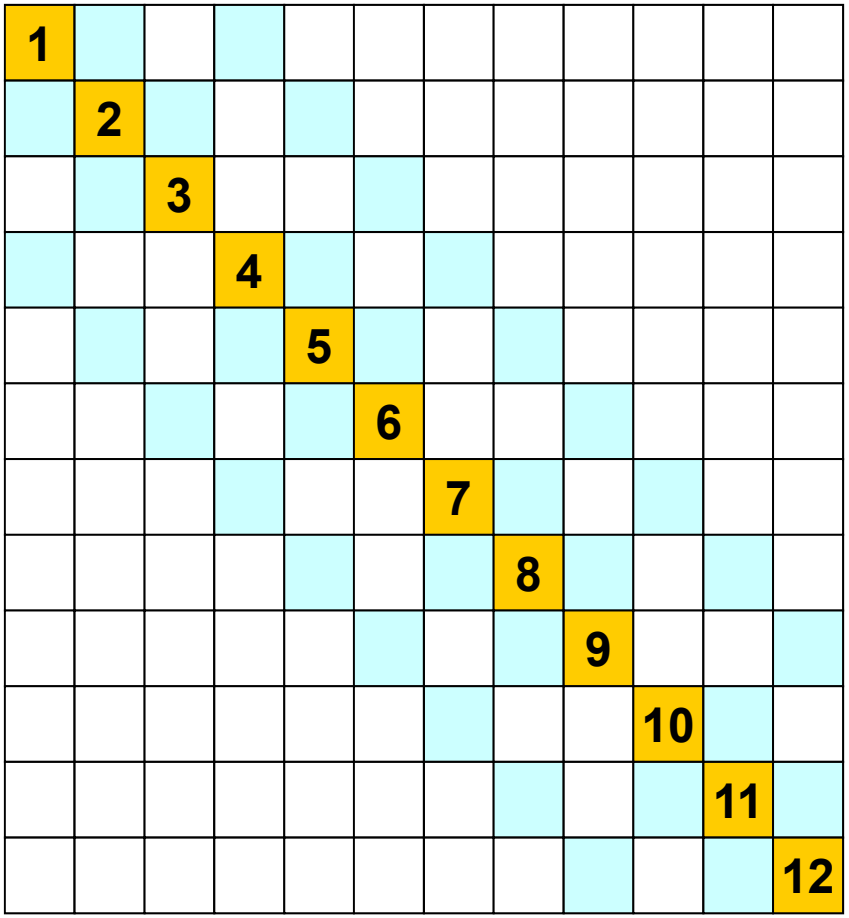
① ➡ $a_{11} = u_{11}, a_{12} = u_{12}, \cdots, a_{1n} = u_{1n} \Rightarrow u_{11}, u_{12}, \cdots, u_{1n}$

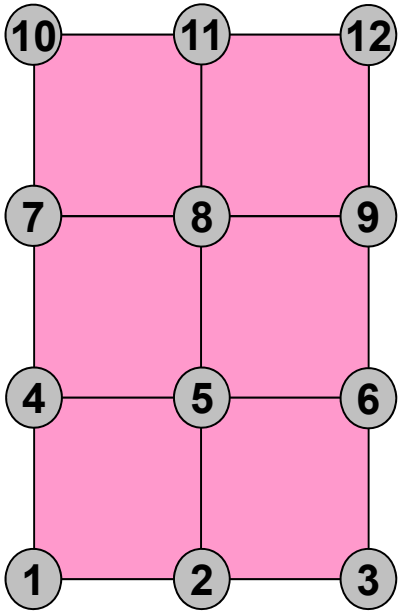
② ➡ $a_{21} = l_{21}u_{11}, a_{31} = l_{31}u_{11}, \cdots, a_{n1} = l_{n1}u_{11} \Rightarrow l_{21}, l_{31}, \cdots, l_{n1}$

③ ➡ $a_{22} = l_{21}u_{12} + u_{22}, \cdots, a_{2n} = l_{21}u_{1n} + u_{2n} \Rightarrow u_{22}, u_{23}, \cdots, u_{2n}$

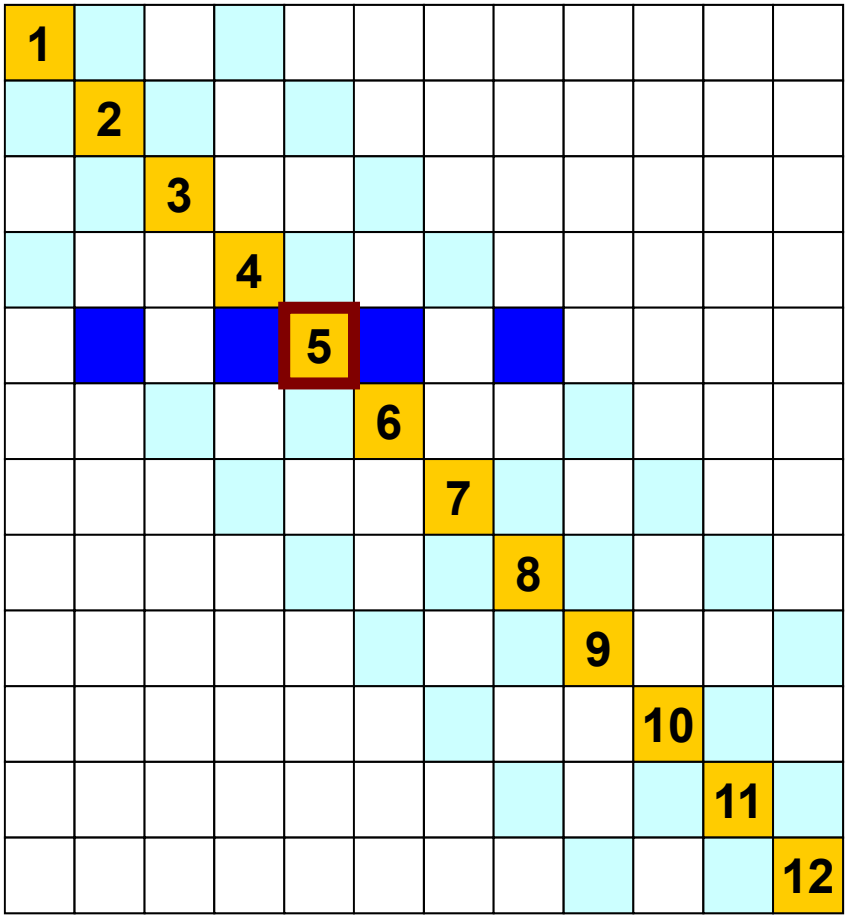
④ ➡ $a_{32} = l_{31}u_{12} + l_{32}u_{22}, \cdots \Rightarrow l_{32}, l_{42}, \cdots, l_{n2}$

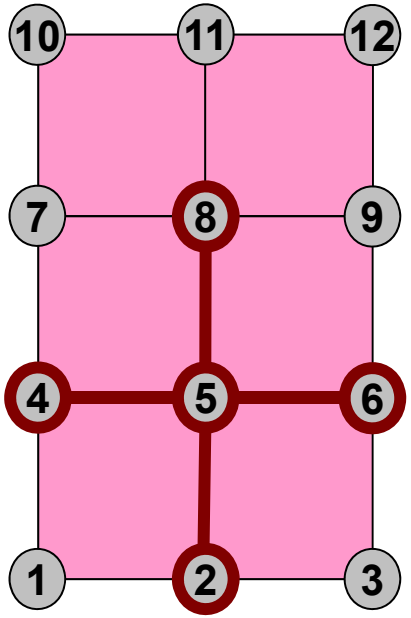
Example: 5-Point Stencil





Example: 5-Point Stencil



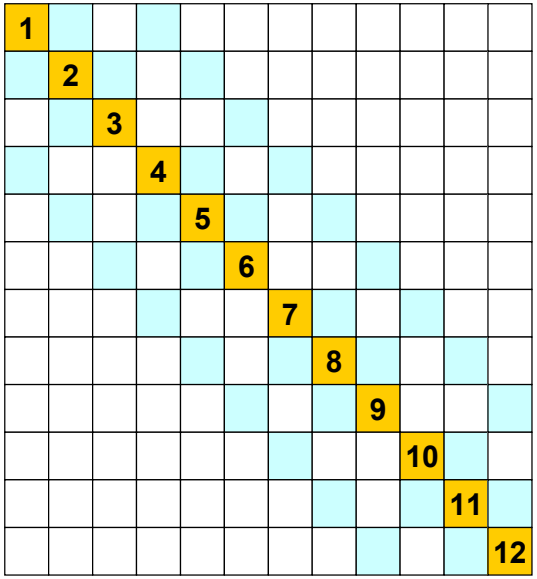


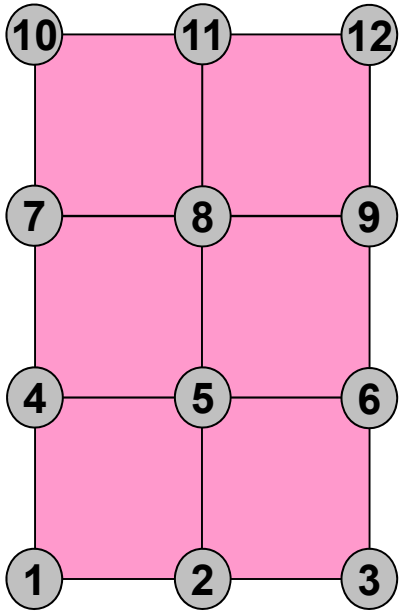
Coef. Matrix

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00

X =

0.00
3.00
10.00
11.00
10.00
19.00
20.00
16.00
28.00
42.00
36.00
52.00

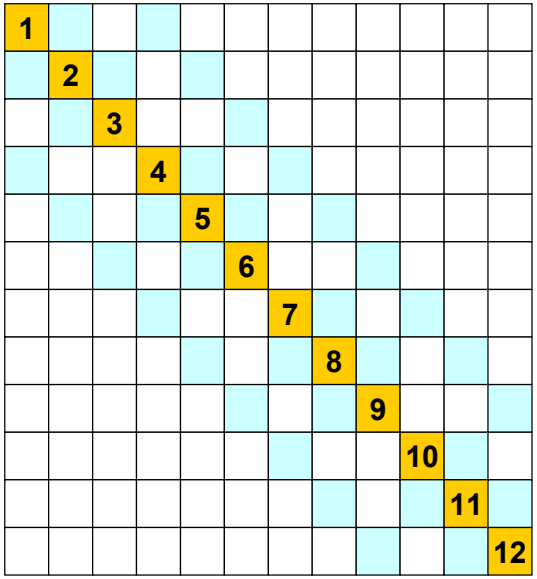


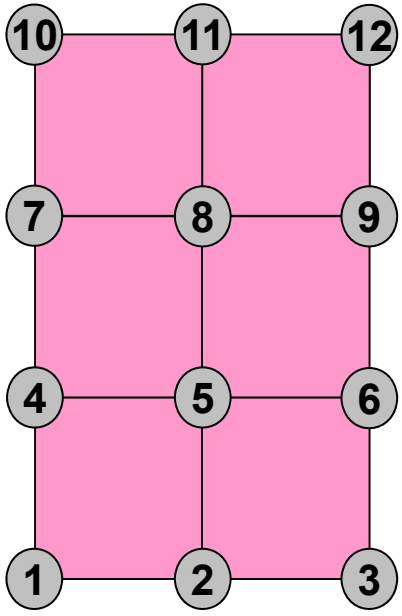


Solution

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00

		=	
1.00			0.00
2.00			3.00
3.00			10.00
4.00			11.00
5.00			10.00
6.00			19.00
7.00			20.00
8.00			16.00
9.00			28.00
10.00			42.00
11.00			36.00
12.00			52.00





Full LU Factorization

Original Matrix

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	0.00	0.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	0.00	6.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−1.00	0.00	−1.00	6.00

LU Factorization

[L][U]
Diagonal Components of
[L] (=1) are not displayed

fill-in occurred

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	−0.03	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	−0.03	0.00	5.83	−1.03	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	−0.03	−0.18	5.64	−1.03	−0.18	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−0.17	0.00	−0.18	5.64	−0.03	−0.18	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03	−0.18	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63	−0.03	−0.18	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63

ILU Factorization (without Fill-in)

ILU Factorization

[L][U]
Diagonal Components of
[L] (=1) are not displayed

NO fill-in's

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	0.00	−0.17	5.66	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−0.17	0.00	−0.18	5.65	0.00	0.00	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00	0.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65	0.00	0.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65

LU Factorization

[L][U]
Diagonal Components of
[L] (=1) are not displayed

fill-in occurred

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	−0.03	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	−0.03	0.00	5.83	−1.03	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	−0.03	−0.18	5.64	−1.03	−0.18	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−0.17	0.00	−0.18	5.64	−0.03	−0.18	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03	−0.18	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63	−0.03	−0.18	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63

Solution: A little bit inaccurate ...

ILU

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	0.00	−0.17	5.66	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−0.17	0.00	−0.18	5.65	0.00	0.00	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00	0.00	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65	0.00	0.00	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65

→

0.92
1.75
2.76
3.79
4.46
5.57
6.66
7.25
8.46
9.66
10.54
11.83

Full LU

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	−0.03	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	−0.03	0.00	5.83	−1.03	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	−0.03	−0.18	5.64	−1.03	−0.18	−1.00	0.00	0.00	0.00	0.00
0.00	0.00	−0.17	0.00	−0.18	5.64	−0.03	−0.18	−1.00	0.00	0.00	0.00
0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01	−1.00	0.00	0.00
0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03	−0.18	−1.00	0.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63	−0.03	−0.18	−1.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63

→

1.00
2.00
3.00
4.00
5.00
6.00
7.00
8.00
9.00
10.00
11.00
12.00

ILU(0), IC(0)

- Incomplete Factorization without Fill-in's
  - Saving Memory, Smaller Computations

- **If we solve equations by this incomplete factorization, we can get "incomplete" solutions.**
  - **But those are not far from accurate ones.**
  - **"Accuracy"/"Inaccuracy" depends on property of matrices**

Classification of Preconditioning Methods: Trade-off

Point Jacobi

Diagonal Blocking

ILU(0)

ILU(1)

ILU(2)

Gaussian Elimination

- Simple
- Easy to be Parallelized
- Cheap

- Complicated
- Global Dependency
- Expensive

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **ICCG Solver for Poisson Equations**
  - **How to run**
    - **Data Structure**
  - Program
    - Initialization
    - Coefficient Matrices
    - ICCG

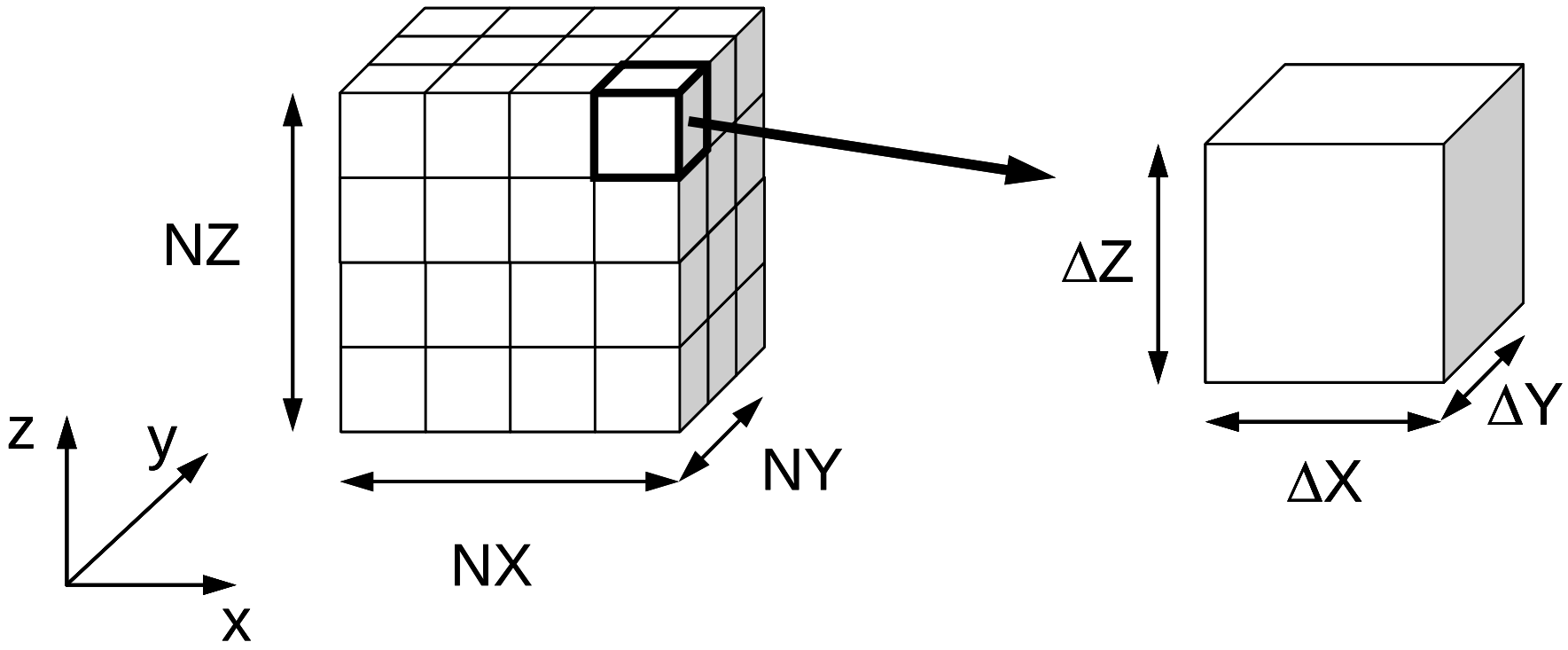
Target Application

- 3D Poisson Equations

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

- Finite Volume Method (FVM)
  - Arbitrary Shape Elements, Cell-Centered
  - “Direct” Finite Difference Method
- Boundary Conditions
  - Dirichlet B.C., Volume Flux
- Preconditioned Iterative Solvers
  - Conjugate Gradient + Preconditioner

3D Structured Mesh
Internal data structure is “unstructured”



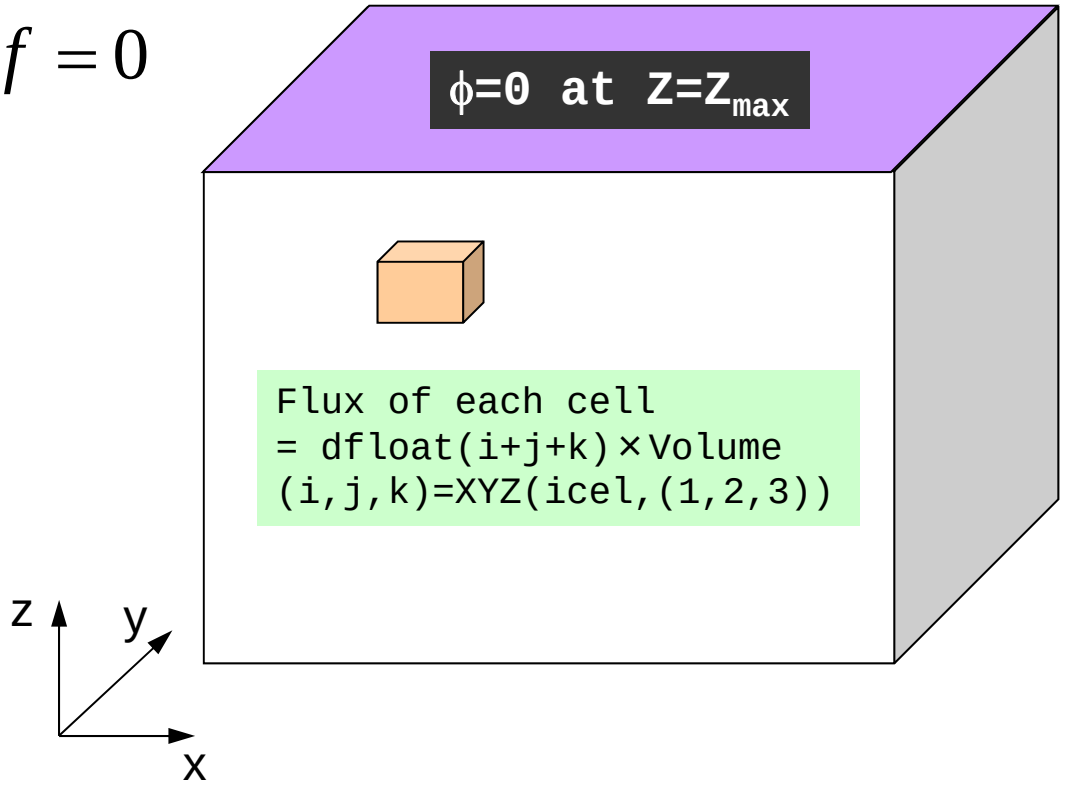
Target Problem: Variables are defined at cell-center'

Poisson Equation

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

B.C.

- Volume Flux
- ϕ=0@Z=Z_{max}



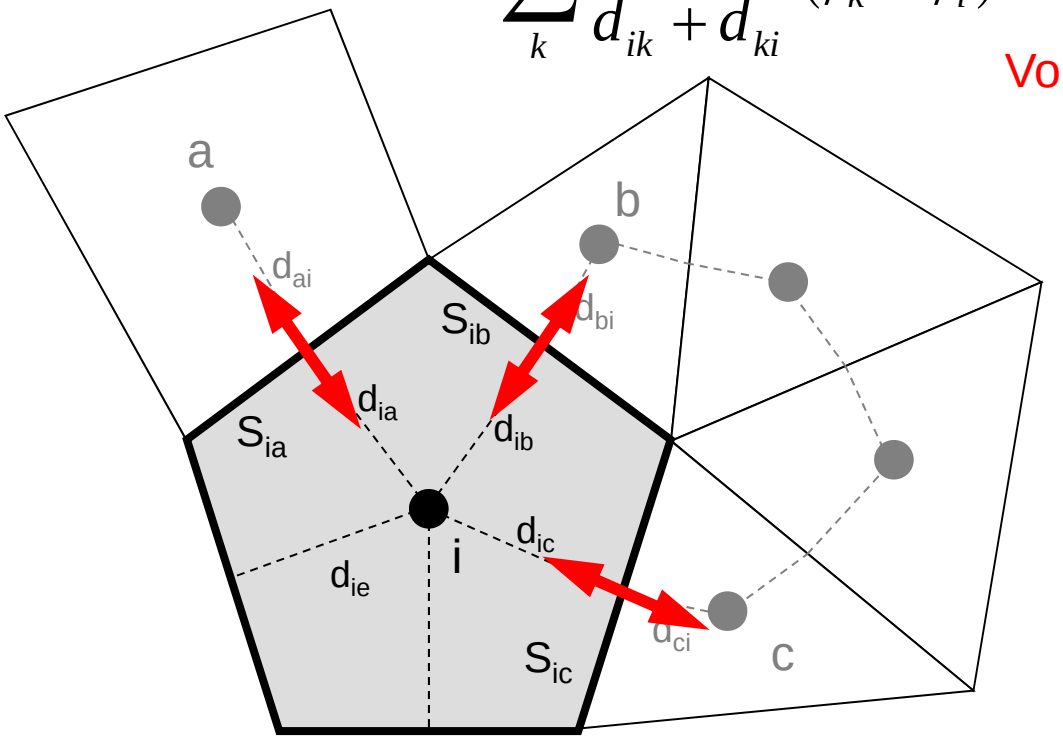
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

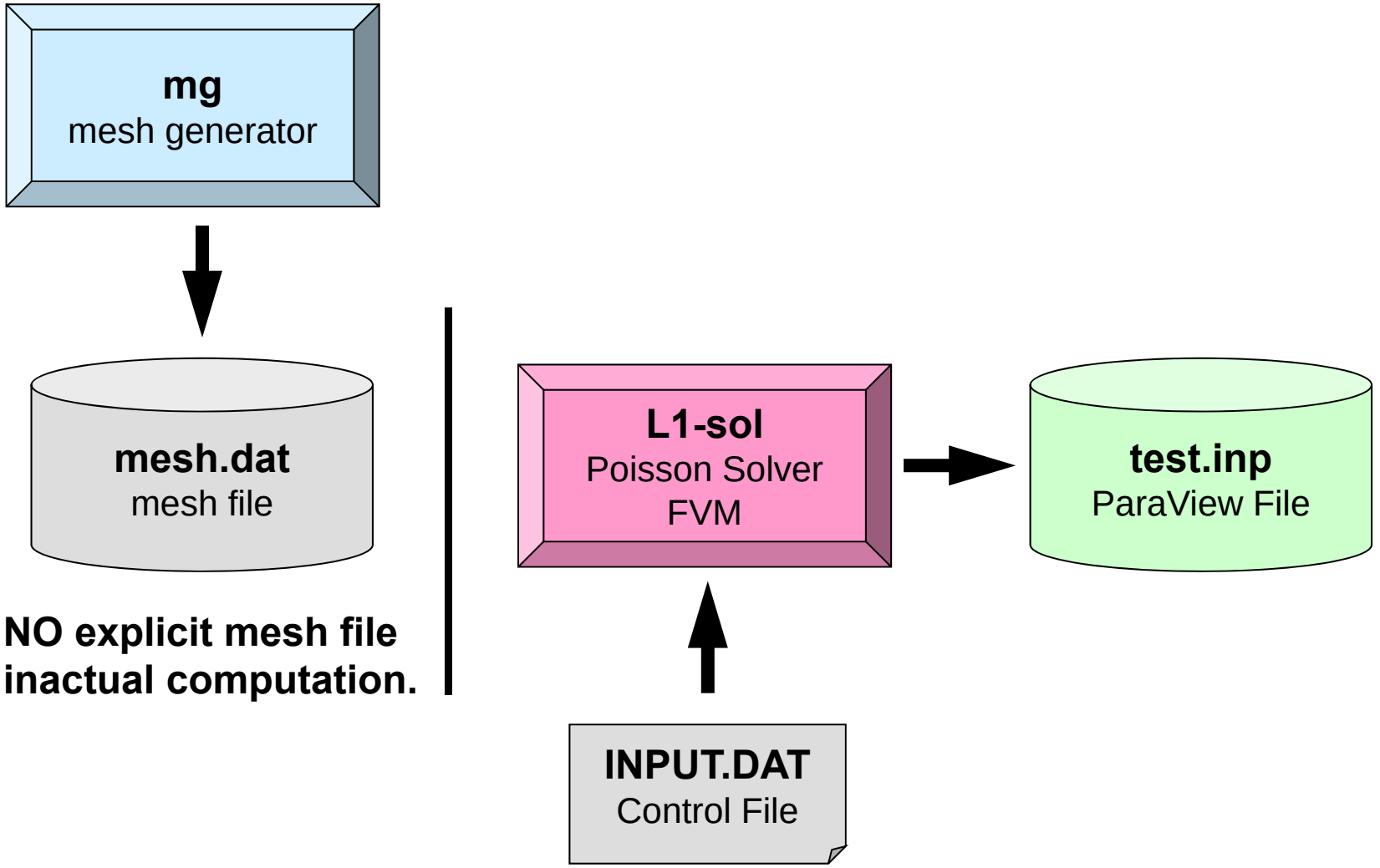
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Running the Program: <$E-L1>/run



Running the Program

Compiling

```
$> cd <$E-L1>/run

$> gfortran -O mg.f -o mg (or cc –O mg.c –o mg)
$> ls mg
      mg

$> cd ../src
$> make
$> ls ../run/L1-sol
      L1-sol
```

Mesh Generator: **mg**

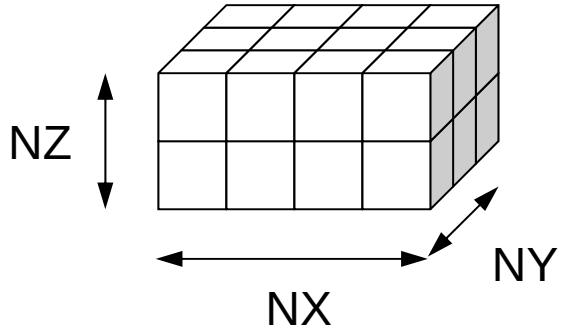
Poisson Solver (FVM): **L1-sol**

Running the Program

Mesh Generation

```
$> cd ../run
$> ./mg
   4 3 2
$> ls mesh.dat
     mesh.dat
```

NX, NY, NZ



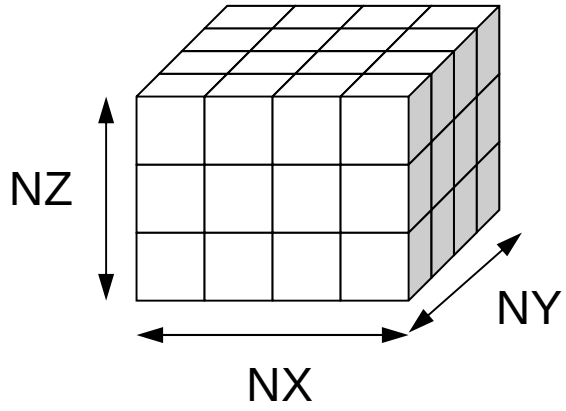
mesh.dat (1/5)

```
  4   3   2
 24
  1   0   2   0   5   0  13   1   1   1
  2   1   3   0   6   0  14   2   1   1
  3   2   4   0   7   0  15   3   1   1
  4   3   0   0   8   0  16   4   1   1
  5   0   6   1   9   0  17   1   2   1
  6   5   7   2  10   0  18   2   2   1
  7   6   8   3  11   0  19   3   2   1
  8   7   0   4  12   0  20   4   2   1
  9   0  10   5   0   0  21   1   3   1
 10   9  11   6   0   0  22   2   3   1
 11  10  12   7   0   0  23   3   3   1
 12  11   0   8   0   0  24   4   3   1
 13   0  14   0  17   1   0   1   1   2
 14  13  15   0  18   2   0   2   1   2
 15  14  16   0  19   3   0   3   1   2
 16  15   0   0  20   4   0   4   1   2
 17   0  18  13  21   5   0   1   2   2
 18  17  19  14  22   6   0   2   2   2
 19  18  20  15  23   7   0   3   2   2
 20  19   0  16  24   8   0   4   2   2
 21   0  22  17   0   9   0   1   3   2
 22  21  23  18   0  10   0   2   3   2
 23  22  24  19   0  11   0   3   3   2
 24  23   0  20   0  12   0   4   3   2
```

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (2/5)

```
 4   3   2
24
 1   0   2   0   5   0  13   1   1   1
 2   1   3   0   6   0  14   2   1   1
 3   2   4   0   7   0  15   3   1   1
 4   3   0   0   8   0  16   4   1   1
 5   0   6   1   9   0  17   1   2   1
 6   5   7   2  10   0  18   2   2   1
 7   6   8   3  11   0  19   3   2   1
 8   7   0   4  12   0  20   4   2   1
 9   0  10   5   0   0  21   1   3   1
10   9  11   6   0   0  22   2   3   1
11  10  12   7   0   0  23   3   3   1
12  11   0   8   0   0  24   4   3   1
13   0  14   0  17   1   0   1   1   2
14  13  15   0  18   2   0   2   1   2
15  14  16   0  19   3   0   3   1   2
16  15   0   0  20   4   0   4   1   2
17   0  18  13  21   5   0   1   2   2
18  17  19  14  22   6   0   2   2   2
19  18  20  15  23   7   0   3   2   2
20  19   0  16  24   8   0   4   2   2
21   0  22  17   0   9   0   1   3   2
22  21  23  18   0  10   0   2   3   2
23  22  24  19   0  11   0   3   3   2
24  23   0  20   0  12   0   4   3   2
```



Number of meshes in X/Y/Z directions

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (3/5)

Number of Meshes (Cells)
= NX x NY x NZ

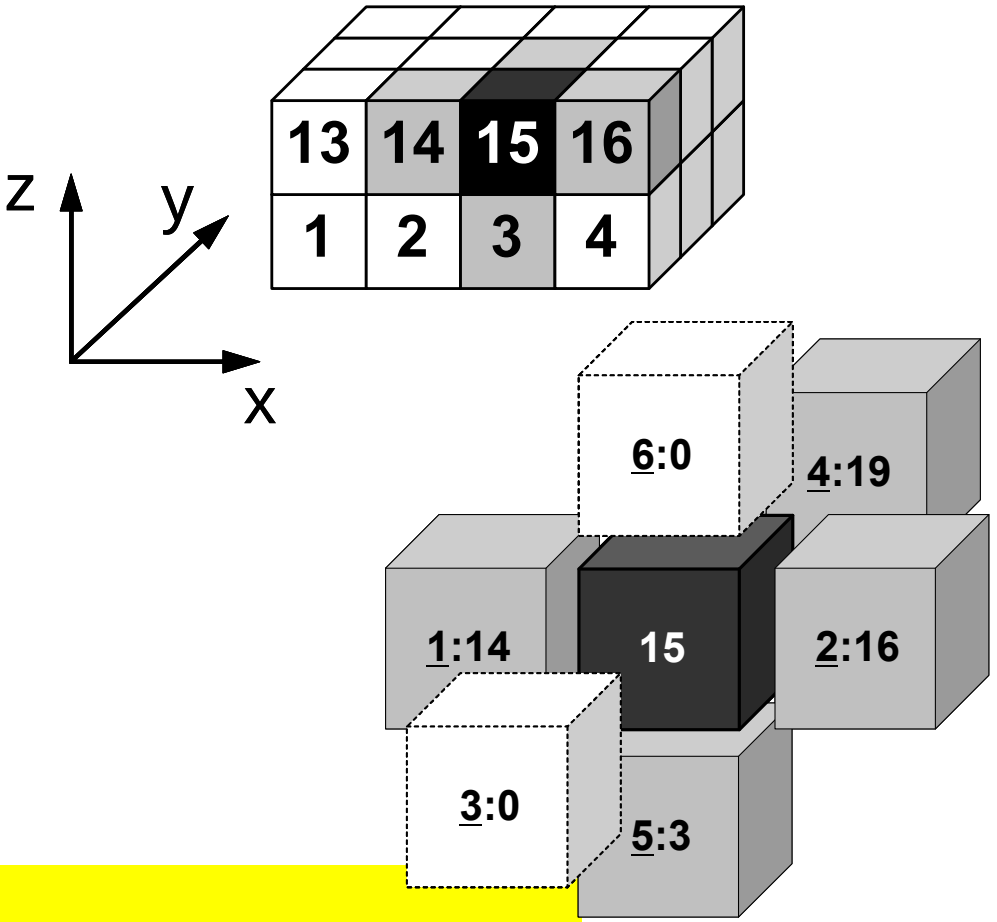
```
  4   3   2
24
  1   0   2   0   5   0  13   1   1   1
  2   1   3   0   6   0  14   2   1   1
  3   2   4   0   7   0  15   3   1   1
  4   3   0   0   8   0  16   4   1   1
  5   0   6   1   9   0  17   1   2   1
  6   5   7   2  10   0  18   2   2   1
  7   6   8   3  11   0  19   3   2   1
  8   7   0   4  12   0  20   4   2   1
  9   0  10   5   0   0  21   1   3   1
 10   9  11   6   0   0  22   2   3   1
 11  10  12   7   0   0  23   3   3   1
 12  11   0   8   0   0  24   4   3   1
 13   0  14   0  17   1   0   1   1   2
 14  13  15   0  18   2   0   2   1   2
 15  14  16   0  19   3   0   3   1   2
 16  15   0   0  20   4   0   4   1   2
 17   0  18  13  21   5   0   1   2   2
 18  17  19  14  22   6   0   2   2   2
 19  18  20  15  23   7   0   3   2   2
 20  19   0  16  24   8   0   4   2   2
 21   0  22  17   0   9   0   1   3   2
 22  21  23  18   0  10   0   2   3   2
 23  22  24  19   0  11   0   3   3   2
 24  23   0  20   0  12   0   4   3   2
```

```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

mesh.dat (4/5)

```
  4    3    2
 24
  1    0    2    0    5    0   13    1    1    1
  2    1    3    0    6    0   14    2    1    1
  3    2    4    0    7    0   15    3    1    1
  4    3    0    0    8    0   16    4    1    1
  5    0    6    1    9    0   17    1    2    1
  6    5    7    2   10    0   18    2    2    1
  7    6    8    3   11    0   19    3    2    1
  8    7    0    4   12    0   20    4    2    1
  9    0   10    5    0    0   21    1    3    1
 10    9   11    6    0    0   22    2    3    1
 11   10   12    7    0    0   23    3    3    1
 12   11    0    8    0    0   24    4    3    1
 13    0   14    0   17    1    0    1    1    2
 14   13   15    0   18    2    0    2    1    2
 15   14   16    0   19    3    0    3    1    2
 16   15    0    0   20    4    0    4    1    2
 17    0   18   13   21    5    0    1    2    2
 18   17   19   14   22    6    0    2    2    2
 19   18   20   15   23    7    0    3    2    2
 20   19    0   16   24    8    0    4    2    2
 21    0   22   17    0    9    0    1    3    2
 22   21   23   18    0   10    0    2    3    2
 23   22   24   19    0   11    0    3    3    2
 24   23    0   20    0   12    0    4    3    2
```

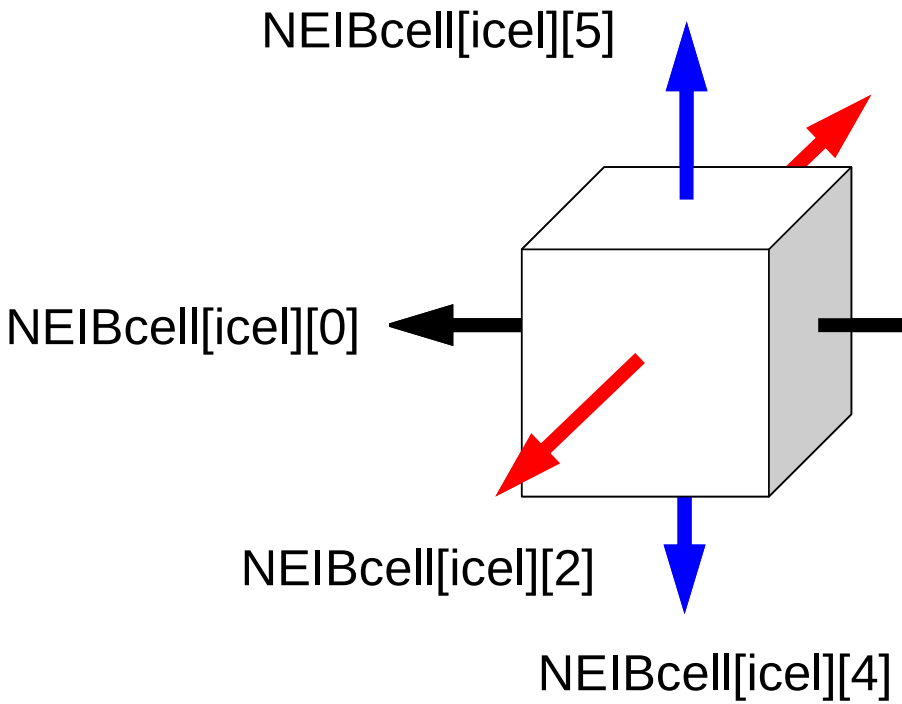
Neighboring Cells: NEIBcell(i,k)




```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

1st Col.: Global ID of the Cell

NEIBcell: ID of Neighboring Mesh/Cell =0: for Boundary Surface



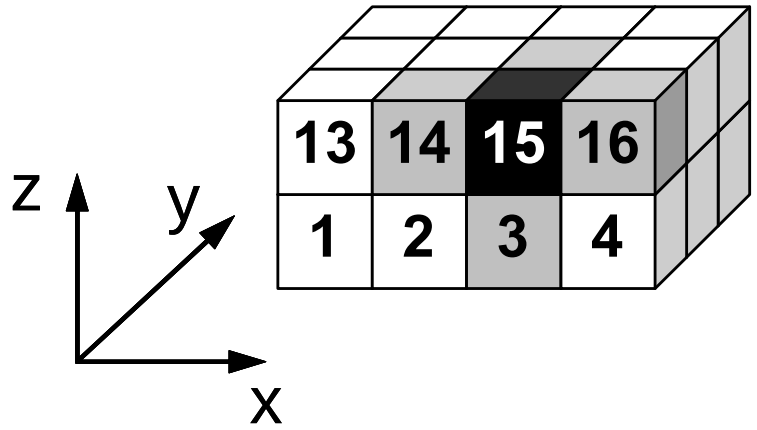
i= XYZ[icel][0]
j= XYZ[icel][1], k= XYZ[icel][2]
icel= k*NX*NY + j*NX + i

NEIBcell[icel][0]= icel – 1 + 1
NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][5]= icel + NX*NY + 1

mesh.dat (5/5)

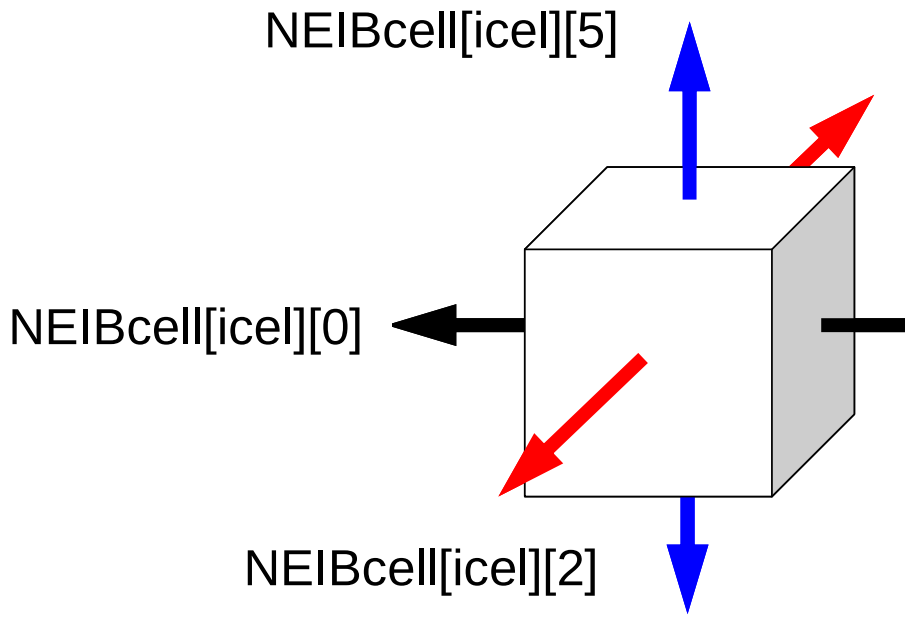
```
  4    3    2
 24
  1    0    2    0    5    0   13    1    1    1
  2    1    3    0    6    0   14    2    1    1
  3    2    4    0    7    0   15    3    1    1
  4    3    0    0    8    0   16    4    1    1
  5    0    6    1    9    0   17    1    2    1
  6    5    7    2   10    0   18    2    2    1
  7    6    8    3   11    0   19    3    2    1
  8    7    0    4   12    0   20    4    2    1
  9    0   10    5    0    0   21    1    3    1
 10    9   11    6    0    0   22    2    3    1
 11   10   12    7    0    0   23    3    3    1
 12   11    0    8    0    0   24    4    3    1
 13    0   14    0   17    1    0    1    1    2
 14   13   15    0   18    2    0    2    1    2
 15   14   16    0   19    3    0    3    1    2
 16   15    0    0   20    4    0    4    1    2
 17    0   18   13   21    5    0    1    2    2
 18   17   19   14   22    6    0    2    2    2
 19   18   20   15   23    7    0    3    2    2
 20   19    0   16   24    8    0    4    2    2
 21    0   22   17    0    9    0    1    3    2
 22   21   23   18    0   10    0    2    3    2
 23   22   24   19    0   11    0    3    3    2
 24   23    0   20    0   12    0    4    3    2
```

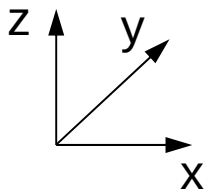
Location in X,Y,Z-directions: XYZ(i,j)




```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

do i= 1, ICELTOT
  read (21, '(10i10)' ) ii, (NEIBcell(i,k), k= 1, 6), (XYZ(i,j), j= 1, 3)
enddo
```

NEIBcell: ID of Neighboring Mesh/Cell =0: for Boundary Surface





i= XYZ[icel][0]
j= XYZ[icel][1], k= XYZ[icel][2]
icel= k*NX*NY + j*NX + i

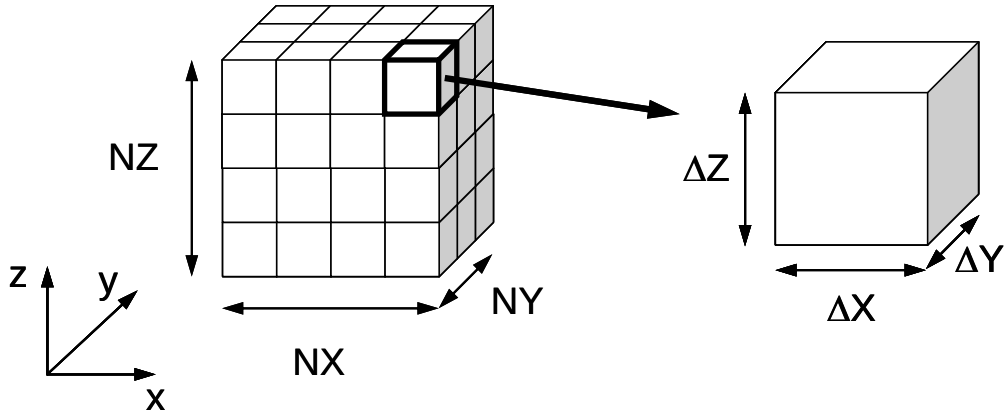
NEIBcell[icel][0]= icel – 1 + 1
NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][5]= icel + NX*NY + 1

Running the Program

Control Data: <$E-L1>/run/INPUT.DAT

```
32 32 32                                NX/NY/NZ
1                                       MEHOD 1:2
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

- NX, NY, NZ
  - Number of meshes in X/Y/Z dir.
- METHOD
  - Preconditioner
- DX, DY, DZ
  - Size of meshes
- EPSICCG
  - Convergence Criteria for ICCG



Preconditioning Method

```
32 32 32                                NX/NY/NZ
1                                       MEHOD 1:2
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

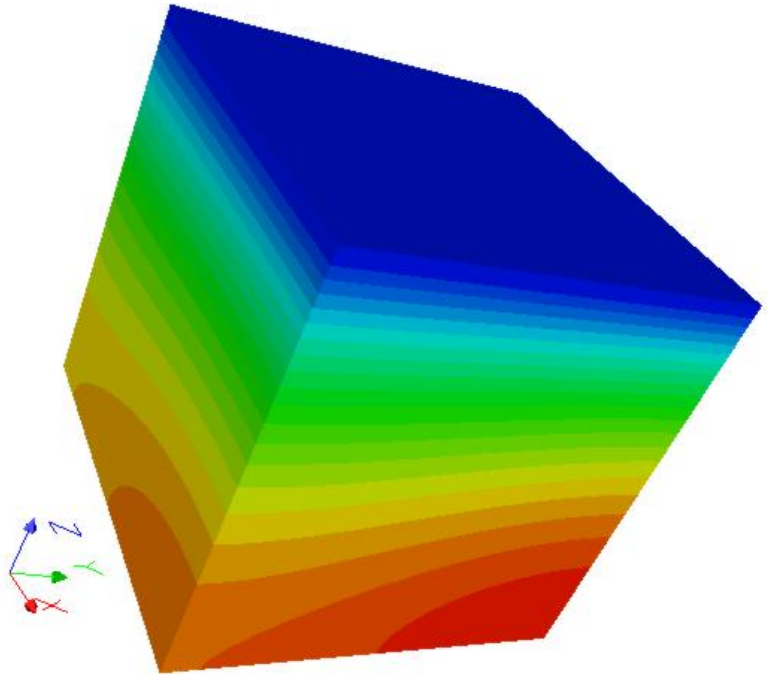
- METHOD=1 Incomplete Modified Cholesky Fact. (Off-Diagonal Components unchanged)
- METHOD=2 Incomplete Modified Cholesky Fact. (Fortran ONLY)
- METHOD=3 Diagonal Scaling/Point Jacobi

Running the Program

Running, Post Processing by ParaView

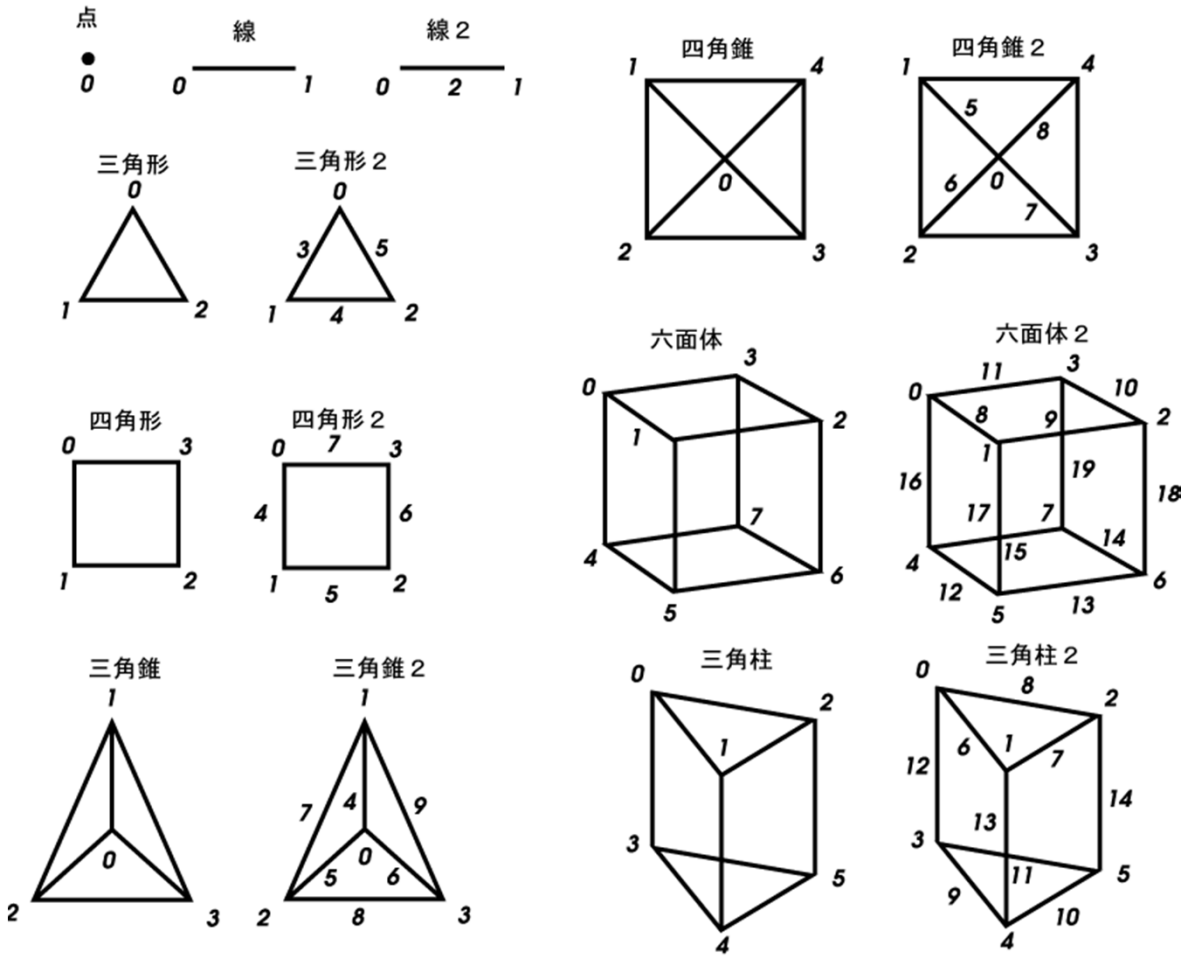
http://nkl.cc.u-tokyo.ac.jp/class/HowtouseParaViewE.pdf

```
$> cd <$P-L1>/run
$> ./L1-sol

$> ls test.inp
   test.inp
```



UCD Format (1/2)
Unstructured Cell Data

要素の種類	キーワード
点	pt
線	line
三角形	tri
四角形	quad
四面体	tet
角錐	pyr
三角柱	prism
六面体	hex
二次要素	
線2	line2
三角形2	tri2
四角形2	quad2
四面体2	tet2
角錐2	pyr2
三角柱2	prism2
六面体2	hex2



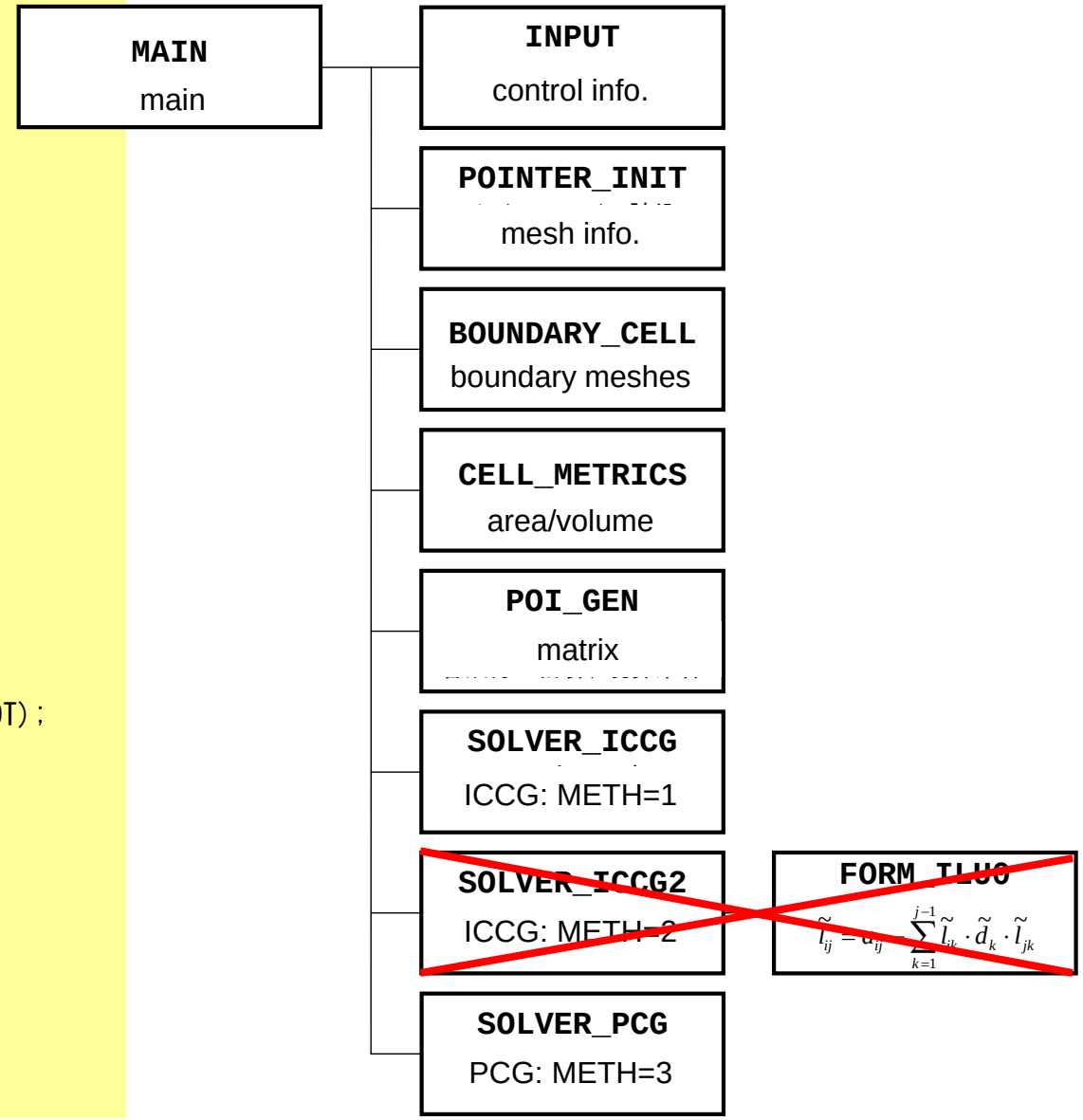
UCD Format (2/2)

- Originally for AVS, microAVS
- Extension of the UCD file is “inp”
- There are two types of formats. Only old type can be read by ParaView.

- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **ICCG Solver for Poisson Equations**
  - How to run
    - Data Structure
  - **Program**
    - **Initialization**
    - **Coefficient Matrices**
    - ICCG

Structure of the Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
#include "input.h " ...

int
main()
{
  double *WK;
  int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
  double xN, xL, xU; Stime, Etime;

  if(INPUT()) goto error;
  if(POINTER_INIT()) goto error;
  if(BOUNDARY_CELL()) goto error;
  if(CELL_METRICS()) goto error;
  if(POI_GEN()) goto error;

  memset(PHI, 0.0, sizeof(double)*ICELTOT);
  ISET = 0;
  WK = (double *)malloc(sizeof(double)*ICELTOT);

  if(METHOD==1){
     if(solve_ICCG(...)) goto error;
     } else if(METHOD==3){
     if(solve_PCG(...)) goto error;}

  if(OUTUCD()) goto error;
  return 0;
error:
  return -1;
}
```



struct.h

```
#ifndef __H_STRUCT
#define __H_STRUCT

#include <omp.h>

int ICELTOT, ICELTOTp, N;
int NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT;
int NXc, NYc, NZc;

double DX, DY, DZ, XAREA, YAREA, ZAREA;
double RDX, RDY, RDZ, RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ;
double *VOLCEL, *VOLNOD, *RVC, *RVN;

int **XYZ, **NEIBcell;

int ZmaxCELtot;
int *BC_INDEX, *BC_NOD;
int *ZmaxCEL;

int **IWKX;
double **FCV;

int my_rank, PETOT, PEsmpTOT;

#endif /* __H_STRUCT */
```

ICELTOT:
Number of meshes (NX x NY x NZ)

N:
Number of modes

NX,NY,NZ:
Number of meshes in x/y/z directions

NXP1,NYP1,NZP1:
Number of nodes in x/y/z directions

IBNODTOT:
= NXP1 x NYP1

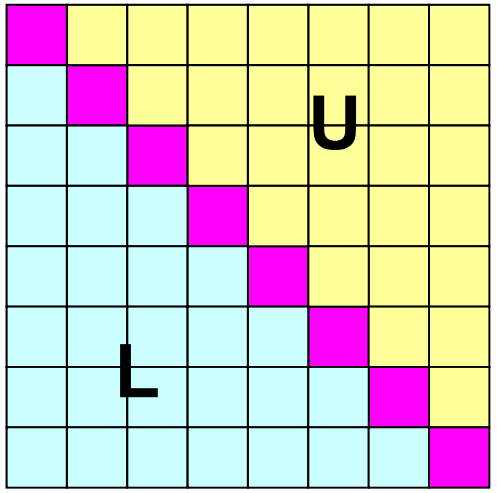
XYZ[ICELTOT][3]:
Location of meshes

NEIBcell[ICELTOT][6]:
Neighboring meshes

pcg.h (1/5)

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

- Sparse Matrix
- Only non-zero off-diagonal components (CRS)
- Diagonal/Lower/Upper components are stored separately



pcg.h (2/5)

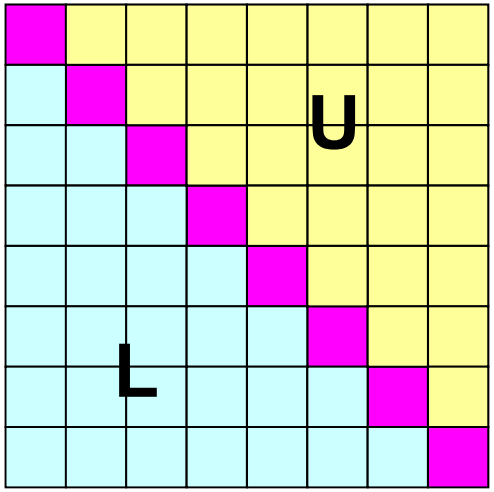
```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

Auxiliary Arrays

Lower Part (Column ID)
`IAL[i][icou] < i`

Upper Part (Column ID)
`IAU[i][icou] > i`

`INL[ICELTOT]`	# Non-zero off-diag. components (lower)
`IAL[ICELTOT][NL]`	**Col. ID: non-zero off-diag. comp. (lower)**
`INU[ICELTOT]`	# Non-zero off-diag. components (upper)
`IAU[ICELTOT][NU]`	**Col. ID: non-zero off-diag. comp. (upper)**
`NU,NL`	Max # of L/U non-zero off-diag. comp.s (=6)
`indexL[ICELTOT+1]`	# Non-zero off-diag. comp. (lower, CRS)
`indexU[ICELTOT+1]`	# Non-zero off-diag. comp. (upper, CRS)
`NPL, NPU`	Total # of L/U non-zero off-diag. comp.
`itemL[NPL],itemU[NPU]`	Col. ID: non-zero off-diag. comp. (L/U, CRS)



pcg.h (3/5)

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

Auxiliary Arrays

Lower Part (Column ID)

`IAL[i][icou] < i`

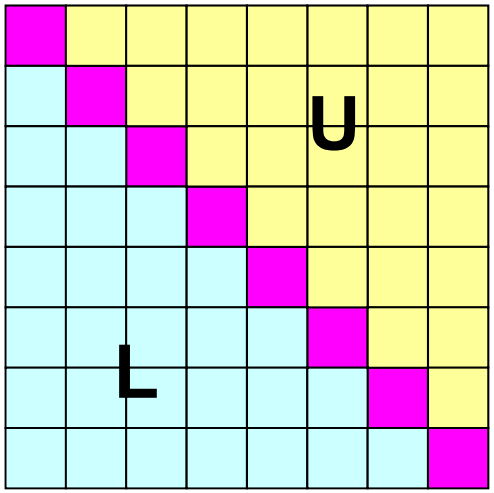
`INL[i]`: Number@each row

Upper Part (Column ID)

`IAU[i][icou] > i`

`INU[i]`: Number@each row

`INL[ICELTOT]`	**# Non-zero off-diag. components (lower)**
`IAL[ICELTOT][NL]`	Col. ID: non-zero off-diag. comp. (lower)
`INU[ICELTOT]`	**# Non-zero off-diag. components (upper)**
`IAU[ICELTOT][NU]`	Col. ID: non-zero off-diag. comp. (upper)
`NU,NL`	Max # of L/U non-zero off-diag. comp.s (=6)
`indexL[ICELTOT+1]`	# Non-zero off-diag. comp. (lower, CRS)
`indexU[ICELTOT+1]`	# Non-zero off-diag. comp. (upper, CRS)
`NPL, NPU`	Total # of L/U non-zero off-diag. comp.
`itemL[NPL],itemU[NPU]`	Col. ID: non-zero off-diag. comp. (L/U, CRS)



pcg.h (4/5)

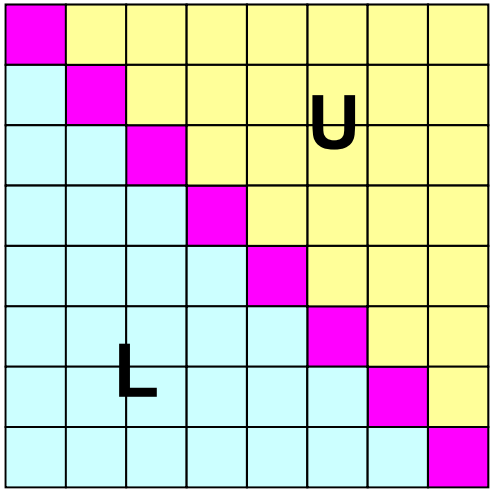
```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

Auxiliary Arrays

Lower Part (Column ID)
`IAL[i][icou] < i`
`INL[i]`: Number@each row

Upper Part (Column ID)
`IAU[i][icou] > i`
`INU[i]`: Number@each row

`INL[ICELTOT]`	# Non-zero off-diag. components (lower)
`IAL[ICELTOT][NL]`	Col. ID: non-zero off-diag. comp. (lower)
`INU[ICELTOT]`	# Non-zero off-diag. components (upper)
`IAU[ICELTOT][NU]`	Col. ID: non-zero off-diag. comp. (upper)
`NU,NL`	Max # of L/U non-zero off-diag. comp.s (=6)
`indexL[ICELTOT+1]`	**# Non-zero off-diag. comp. (lower, CRS)**
`indexU[ICELTOT+1]`	**# Non-zero off-diag. comp. (upper, CRS)**
`NPL, NPU`	**Total # of L/U non-zero off-diag. comp.**
`itemL[NPL],itemU[NPU]`	**Col. ID: non-zero off-diag. comp. (L/U, CRS)**



pcg.h (5/5)

```
#ifndef __H_PCG
#define __H_PCG
        static int N2 = 256;
        int NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL;
        int METHOD, ORDER_METHOD;
        double EPSICCG;

        double *D, *PHI, *BFORCE;
        double *AL, *AU;

        int *INL, *INU, *COLORindex;
        int *indexL, *indexU;
        int *OLDtoNEW, *NEWtoOLD;
        int **IAL, **IAU;
        int *itemL, *itemU;
        int NPL, NPU;
#endif /* __H_PCG */
```

METHOD	Preconditioning method (=1, =2, =3)
EPSICCG	Convergence criteria for ICCG
D [ICELTOT]	Diagonal components of the matrix
PHI [ICLETOT]	Unknown vector
BFORCE[ICELTOT]	RHS vector
AL[NPL], AU[NPU]	Non-zero off-diagonal L/U components of the matrix (CRS)

Variables/Arrays for Matrix

Name	Type	Content
D[N]	**R**	Diagonal components of the matrix (N= ICELTOT)
BFORCE[N]	**R**	RHS vector
PHI[N]	**R**	Unknown vector
indexL[N+1]	**I**	Number of Lower non-zero off-diag. comp. (CRS)
indexU[N+1]	**I**	Number of Upper non-zero off-diag. comp. (CRS)
NPL	**I**	Total number of Lower non-zero off-diag. comp. (CRS)
NPU	**I**	Total number of Upper non-zero off-diag. comp. (CRS)
itemL[NPL]	**I**	Column ID of Lower non-zero off-diag. comp. (CRS)
itemU[NPU]	**I**	Column ID of Upper non-zero off-diag. comp. (CRS)
AL[NPL]	**R**	Lower non-zero off-diag. comp. (CRS)
AU[NPU]	**R**	Upper non-zero off-diag. comp. (CRS)

Variables/Arrays for Matrix Auxiliary Arrays

Name	Type	Content
NL,NU	**I**	MAX. number of Lower/Upper non-zero off-diag. comp. for each mesh (=6 in this case)
INL[N]	**I**	Number of Lower non-zero off-diag. comp.
INU[N]	**I**	Number of Upper non-zero off-diag. comp.
IAL[N][NL]	**I**	Column ID of Lower non-zero off-diag. comp.
IAU[N][NU]	**I**	Column ID of Uppwer non-zero off-diag. comp.

Why Auxiliary Arrays ?

① NPL and NPU are unknown before computation.
② CRS is not suitable for reordering.

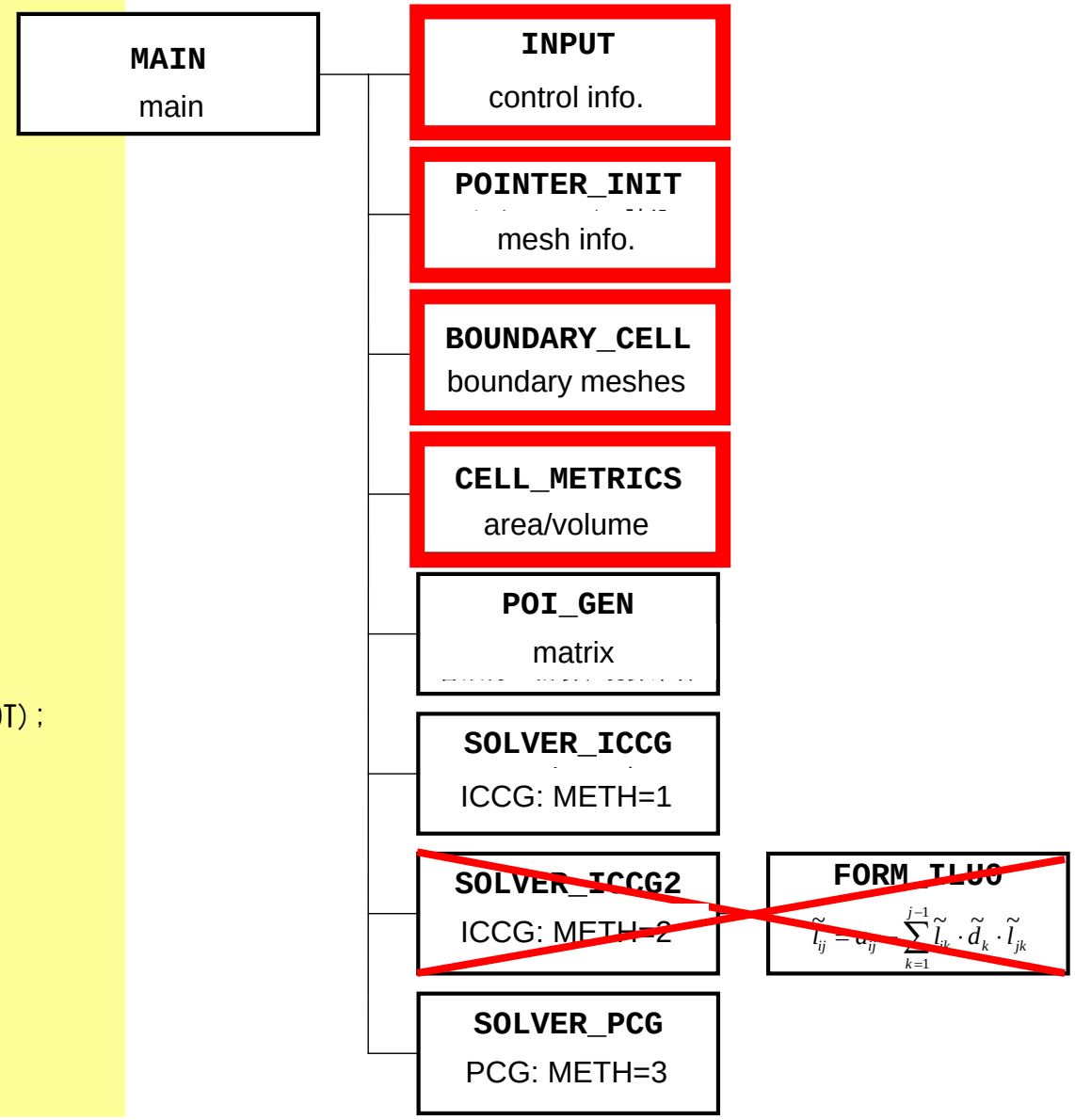
Mat-Vec Multiplication: {q}=[A]{p}

```
for (i=0; i<N; i++) {
      q[i]= D[i] * p[i];
      for (j=indexL[i]; j<indexL[i+1]; j++) {
           q[i] += AL[j] * p[itemL[j]-1];
       }
       for (j=indexU[i]; j<indexU[i+1]; j++) {
            q[i] += AU[j] * p[itemU[j]-1];
        }
}
```

In this program numbering in itemL, itemU starts from “1” (not 0)

Structure of the Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
#include "input.h " ...

int
main()
{
  double *WK;
  int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
  double xN, xL, xU; Stime, Etime;

  if(INPUT()) goto error;
  if(POINTER_INIT()) goto error;
  if(BOUNDARY_CELL()) goto error;
  if(CELL_METRICS()) goto error;
  if(POI_GEN()) goto error;

  memset(PHI, 0.0, sizeof(double)*ICELTOT);
  ISET = 0;
  WK = (double *)malloc(sizeof(double)*ICELTOT);

  if(METHOD==1){
     if(solve_ICCG(...)) goto error;
     } else if(METHOD==3){
     if(solve_PCG(...)) goto error;}

  if(OUTUCD()) goto error;
  return 0;
error:
  return -1;
}
```



input: reading “INPUT.DAT”

```
#include <stdio.h>; <stdlib.h>; <string.h>; <errno.h>
#include "struct_ext.h “; "pcg_ext.h “; "input.h"

extern int
INPUT(void)
{
#define BUF_SIZE 1024
        char line[BUF_SIZE];
        char CNTFIL[81];
        double OMEGA;
        FILE *fp11;

        if((fp11 = fopen("INPUT.DAT", "r")) == NULL) {
            fprintf(stderr, "Error: %s¥n", strerror(errno));
            return -1;
        }
        fgets(line, BUF_SIZE, fp11); sscanf(line, "%d%d%d", &NX, &NY, &NZ);
        fgets(line, BUF_SIZE, fp11); sscanf(line, "%d", &METHOD);
        fgets(line, BUF_SIZE, fp11); sscanf(line, "%le%le%le", &DX, &DY, &DZ);
        fgets(line, BUF_SIZE, fp11); sscanf(line, "%le", &EPSICCG);
        fgets(line, BUF_SIZE, fp11);

        fclose(fp11);
        return 0;
}
```

```
32 32 32                                NX/NY/NZ
1                                       MEHOD 1-3
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

pointer_init (1/3): “mesh.dat”

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct_ext.h"
#include "pcg_ext.h"
#include "pointer_init.h"
#include "allocate.h"

extern int
POINTER_INIT(void)
{

        int icel, ipe, i, j, k;

/*
 * INIT.
 */

  ICELTOT = NX * NY * NZ;

  NXP1 = NX + 1;
  NYP1 = NY + 1;
  NZP1 = NZ + 1;


  NEIBcell =
    (int **)allocate_matrix(sizeof(int),ICELTOT,6);

  XYZ      =
    (int **)allocate_matrix(sizeof(int),ICELTOT,3);
```

NX,NY,NZ:
Number of meshes in x/y/z directions

NXP1,NYP1,NZP1:
Number of nodes in x/y/z directions (for visualization)

ICELTOT:
Number of meshes (NX x NY x NZ)

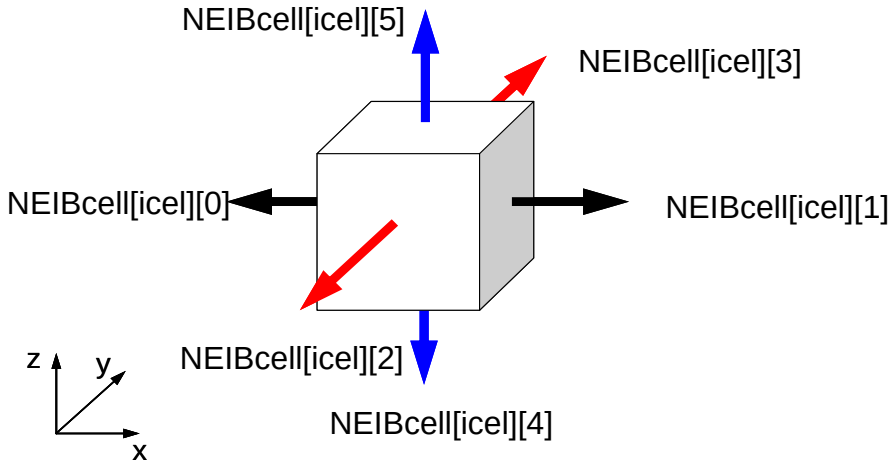
XYZ[ICELTOT][3]:
Location of meshes

NEIBcell[ICELTOT][6]:
Neighboring meshesc

allocate/deallocate

pointer_init (2/3): “mesh.dat”

```
for(k=0; k<NZ; k++) {
    for(j=0; j<NY; j++) {
        for(i=0; i<NX; i++) {
            icel = k * NX * NY + j * NX + i;
            NEIBcell[icel][0] = icel - 1       + 1;
            NEIBcell[icel][1] = icel + 1       + 1;
            NEIBcell[icel][2] = icel - NX      + 1;
            NEIBcell[icel][3] = icel + NX      + 1;
            NEIBcell[icel][4] = icel - NX * NY + 1;
            NEIBcell[icel][5] = icel + NX * NY + 1;
            if(i == 0)    NEIBcell[icel][0] = 0;
            if(i == NX-1) NEIBcell[icel][1] = 0;
            if(j == 0)    NEIBcell[icel][2] = 0;
            if(j == NY-1) NEIBcell[icel][3] = 0;
            if(k == 0)    NEIBcell[icel][4] = 0;
            if(k == NZ-1) NEIBcell[icel][5] = 0;

            XYZ[icel][0] = i + 1;
            XYZ[icel][1] = j + 1;
            XYZ[icel][2] = k + 1;
        }
    }
}
```



i= XYZ[icel][0]
j= XYZ[icel][1], k= XYZ[icel][2]
icel= k*NX*NY + j*NX + i

“icel” starts at 0

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“NEIBcell” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

NEIBcell[icel][0]= icel – 1 + 1
NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][5]= icel + NX*NY + 1

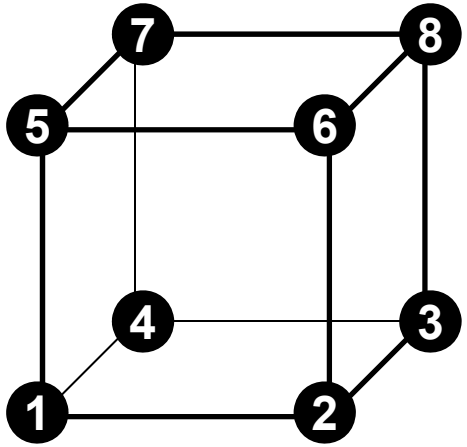
pointer_init (3/3): “mesh.dat”

```
        if(DX <= 0.0) {
                DX = 1.0 / (double)NX;
                DY = 1.0 / (double)NY;
                DZ = 1.0 / (double)NZ;
        }

        NXP1 = NX + 1;
        NYP1 = NY + 1;
        NZP1 = NZ + 1;

        IBNODTOT = NXP1 * NYP1;
        N        = NXP1 * NYP1 * NZP1;

        return 0;
}
```

if DX is no larger than 0.0

pointer_init (3/3): “mesh.dat”

```
        if(DX <= 0.0) {
                DX = 1.0 / (double)NX;
                DY = 1.0 / (double)NY;
                DZ = 1.0 / (double)NZ;
        }

        NXP1 = NX + 1;
        NYP1 = NY + 1;
        NZP1 = NZ + 1;

        IBNODTOT = NXP1 * NYP1;
        N        = NXP1 * NYP1 * NZP1;

        return 0;
}
```



NXP1,NYP1,NZP1:
Number of nodes in x/y/z directions

IBNODTOT:
= NXP1 x NYP1

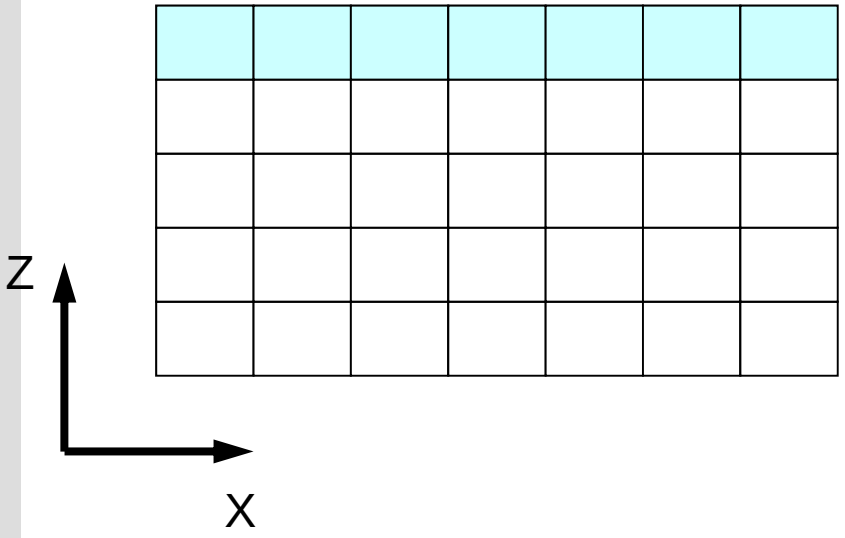
N:
Number of modes
meshes (for visualization)

boundary_cell

Meshes @ $Z=Z_{max}$

Number: ZmaxCELtot

Mesh ID: ZmaxCEL[:]




```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct_ext.h"
#include "boundary_cell.h"
#include "allocate.h"

extern int
BOUNDARY_CELL(void)
{
 int IFACTOT;
 int icou, icel, i, j, k;

 IFACTOT = NX * NY;
 ZmaxCELtot = IFACTOT;

 ZmaxCEL =
    (int *)allocate_vector(sizeof(int),ZmaxCELtot);

 icou = 0;
 k    = NZ - 1;

 for(j=0; j<NY; j++) {
     for(i=0; i<NX; i++) {
         icel = k*IFACTOT + j*NX + i+1;
         ZmaxCEL[icou] = icel;
         icou++;
     }
 }

 return 0;
}
```

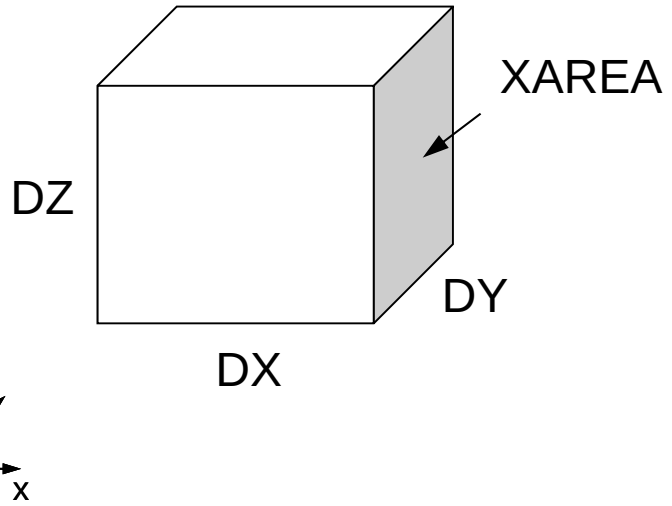
allocate.c

```c
/**************************
  allocate vector
**************************/

void* allocate_vector(int size,int m)
{
  void *a;
  if ( ( a=(void * )malloc( m * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in vector ¥n");
     exit(1);
  }
  return a;
}
```

cell_metrics

Parameters for Computations

```
#include <stdio.h> ...

extern int
CELL_METRICS(void)
{       double V0, RV0;
        int i;
VOLCEL =
(double*)allocate_vector(sizeof(double),ICELTOT);
RVC    =
(double*)allocate_vector(sizeof(double),ICELTOT);

XAREA = DY * DZ;
YAREA = DZ * DX;
ZAREA = DX * DY;

RDX = 1.0 / DX;
RDY = 1.0 / DY;
RDZ = 1.0 / DZ;

RDX2 = 1.0 / (pow(DX, 2.0));
RDY2 = 1.0 / (pow(DY, 2.0));
RDZ2 = 1.0 / (pow(DZ, 2.0));
R2DX = 1.0 / (0.5 * DX);
R2DY = 1.0 / (0.5 * DY);
R2DZ = 1.0 / (0.5 * DZ);

V0  = DX * DY * DZ;
RV0 = 1.0 / V0;

for(i=0; i<ICELTOT; i++) {
    VOLCEL[i] = V0;
    RVC[i]    = RV0;
}
return 0; }
```



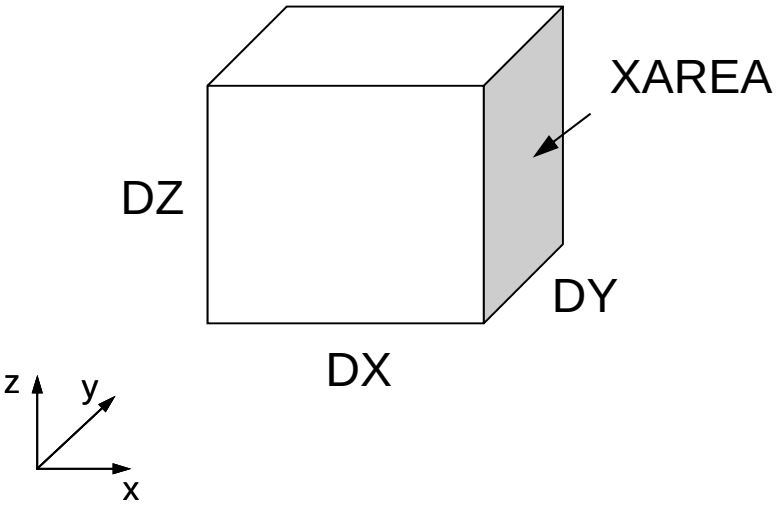
$$XAREA = \Delta Y \times \Delta Z, \quad YAREA = \Delta Z \times \Delta X,$$
$$ZAREA = \Delta X \times \Delta Y$$

$$RDX = \frac{1}{\Delta X}, \quad RDY = \frac{1}{\Delta Y}, \quad RDZ = \frac{1}{\Delta Z}$$

cell_metrics

```
#include <stdio.h> ...

extern int
CELL_METRICS(void)
{       double V0, RV0;
        int i;
VOLCEL =
(double*)allocate_vector(sizeof(double),ICELTOT);
RVC    =
(double*)allocate_vector(sizeof(double),ICELTOT);

XAREA = DY * DZ;
YAREA = DZ * DX;
ZAREA = DX * DY;

RDX = 1.0 / DX;
RDY = 1.0 / DY;
RDZ = 1.0 / DZ;

RDX2 = 1.0 / (pow(DX, 2.0));
RDY2 = 1.0 / (pow(DY, 2.0));
RDZ2 = 1.0 / (pow(DZ, 2.0));
R2DX = 1.0 / (0.5 * DX);
R2DY = 1.0 / (0.5 * DY);
R2DZ = 1.0 / (0.5 * DZ);

V0  = DX * DY * DZ;
RV0 = 1.0 / V0;

for(i=0; i<ICELTOT; i++) {
    VOLCEL[i] = V0;
    RVC[i]    = RV0;
}
return 0; }
```

Parameters for Computations



$$RDX2 = \frac{1}{\Delta X^2}, \quad RDY2 = \frac{1}{\Delta Y^2}, \quad RDZ2 = \frac{1}{\Delta Z^2}$$

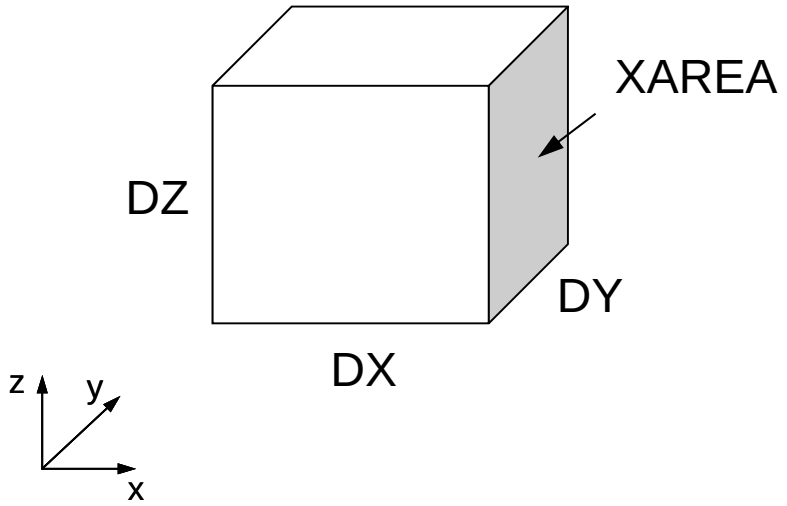
$$R2DX = \frac{1}{0.5 \times \Delta X}, \quad R2DY = \frac{1}{0.5 \times \Delta Y},$$

$$R2DZ = \frac{1}{0.5 \times \Delta Z}$$

cell_metrics

Parameters for Computations

```
#include <stdio.h> ...

extern int
CELL_METRICS(void)
{       double V0, RV0;
        int i;
VOLCEL =
(double*)allocate_vector(sizeof(double),ICELTOT);
RVC    =
(double*)allocate_vector(sizeof(double),ICELTOT);

XAREA = DY * DZ;
YAREA = DZ * DX;
ZAREA = DX * DY;

RDX = 1.0 / DX;
RDY = 1.0 / DY;
RDZ = 1.0 / DZ;

RDX2 = 1.0 / (pow(DX, 2.0));
RDY2 = 1.0 / (pow(DY, 2.0));
RDZ2 = 1.0 / (pow(DZ, 2.0));
R2DX = 1.0 / (0.5 * DX);
R2DY = 1.0 / (0.5 * DY);
R2DZ = 1.0 / (0.5 * DZ);

V0  = DX * DY * DZ;
RV0 = 1.0 / V0;

for(i=0; i<ICELTOT; i++) {
    VOLCEL[i] = V0;
    RVC[i]    = RV0;
}
return 0; }
```

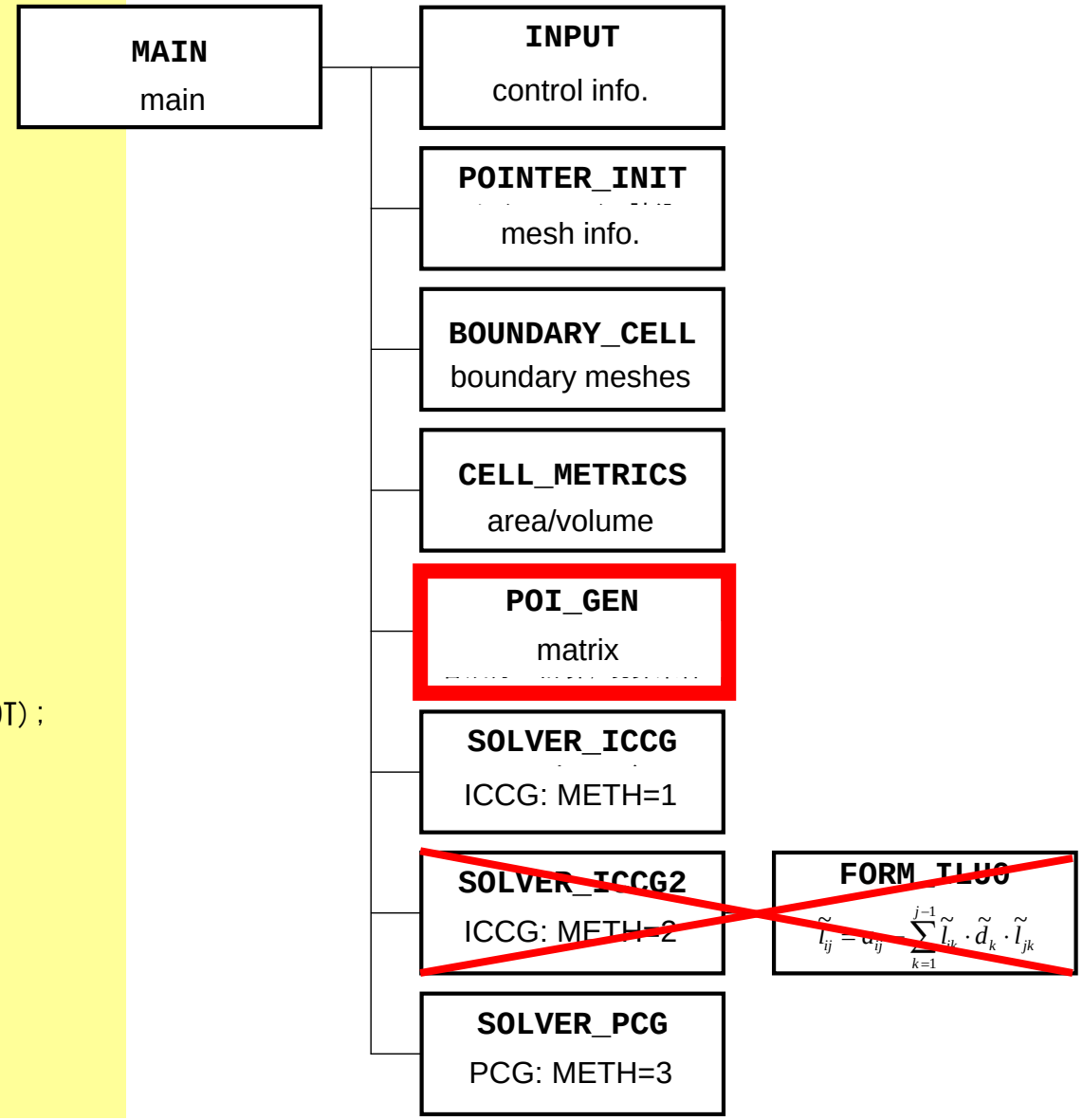


$$VOLCEL = V0 = \Delta X \times \Delta Y \times \Delta Z$$

$$RV0 = RVC = \frac{1}{VOLCEL}$$

Structure of the Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
#include "input.h " ...

int
main()
{
  double *WK;
  int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
  double xN, xL, xU; Stime, Etime;

  if(INPUT()) goto error;
  if(POINTER_INIT()) goto error;
  if(BOUNDARY_CELL()) goto error;
  if(CELL_METRICS()) goto error;
  if(POI_GEN()) goto error;

  memset(PHI, 0.0, sizeof(double)*ICELTOT);
  ISET = 0;
  WK = (double *)malloc(sizeof(double)*ICELTOT);

  if(METHOD==1){
     if(solve_ICCG(...)) goto error;
     } else if(METHOD==3){
     if(solve_PCG(...)) goto error;}

  if(OUTUCD()) goto error;
  return 0;
error:
  return -1;
}
```



poi_gen (1/7)

```
#include "allocate.h"
extern int
POI_GEN(void)
{ int nn;
  int icO, icN1, icN2, icN3, icN4, icN5, icN6;
  int i, j, k, ib, ic, ip, icel, icou, icol, icouG;
  int ii, jj, kk, nn1, num, nr, jO, j1;
  double coef, VOLO, S1t, E1t;
  int isL, ieL, isU, ieU;
  NL=6; NU= 6;
  IAL = (int
**)allocate_matrix(sizeof(int),ICELTOT,NL);
  IAU = (int
**)allocate_matrix(sizeof(int),ICELTOT,NU);
  BFORCE = (double
*)allocate_vector(sizeof(double),ICELTOT);
  D      = (double
*)allocate_vector(sizeof(double),ICELTOT);
  PHI    = (double
*)allocate_vector(sizeof(double),ICELTOT);
  INL = (int *)allocate_vector(sizeof(int),ICELTOT);
  INU = (int *)allocate_vector(sizeof(int),ICELTOT);

  for (i = 0; i <ICELTOT ; i++) {
       BFORCE[i]=0.0;
       D[i]  =0.0; PHI[i]=0.0;
       INL[i] = 0; INU[i] = 0;
       for(j=0;j<6;j++){
           IAL[i][j]=0; IAU[i][j]=0;
       }
  }
  for (i = 0; i <=ICELTOT ; i++) {
       indexL[i] = 0; indexU[i] = 0;
  }
```

```
/*****************************
  allocate matrix                          allocate.c
*****************************/
void** allocate_matrix(int size,int m,int n)
{
  void **aa;
  int i;
  if ( ( aa=(void ** )malloc( m * sizeof(void*) ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! aa in matrix ¥n");
     exit(1);
  }
  if ( ( aa[0]=(void * )malloc( m * n * size ) ) == NULL ) {
     fprintf(stdout,"Error:Memory does not enough! in matrix ¥n");
     exit(1);
  }
        for(i=1;i<m;i++) aa[i]=(char*)aa[i-1]+size*n;
  return aa;
}
```

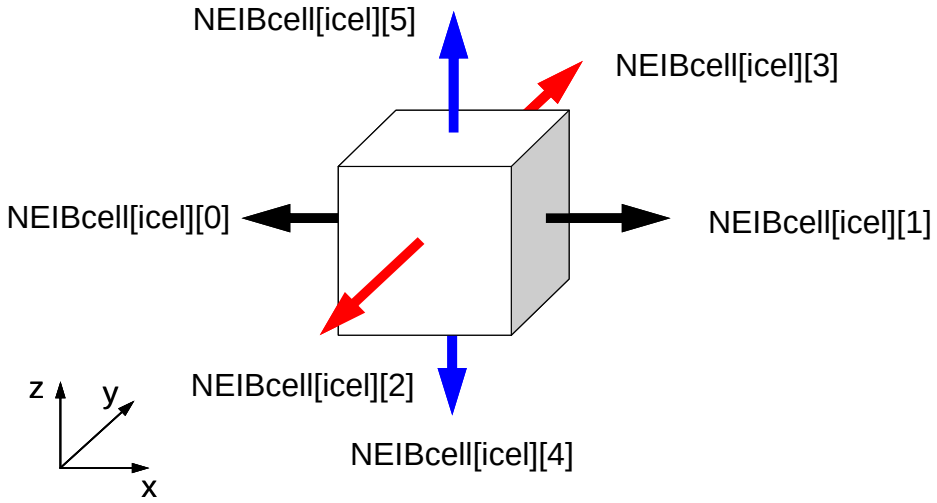
Variables/Arrays for Matrix

Name	Type	Content
D[N]	**R**	Diagonal components of the matrix (N= ICELTOT)
BFORCE[N]	**R**	RHS vector
PHI[N]	**R**	Unknown vector
indexL[N+1] **indexU[N+1]**	**I**	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	**I**	Total # of L/U non-zero off-diag. comp. (CRS)
itemL[NPL] **itemU[NPU]**	**I**	Column ID of L/U non-zero off-diag. comp. (CRS)
AL[NPL] **AU[NPU]**	**R**	L/U non-zero off-diag. comp. (CRS)

Name	Type	Content
NL,NU	**I**	MAX. # of L/U non-zero off-diag. comp. for each mesh (=6)
INL[N] **INU[N]**	**I**	# of L/U non-zero off-diag. comp.
IAL[N][NL] **IAU[N][NU]**	**I**	Column ID of L/U non-zero off-diag. comp.

poi_gen (2/7)

```
for(icel=0; icel<ICELTOT; icel++) {

     icN1 = NEIBcell[icel][0];
     icN2 = NEIBcell[icel][1];
     icN3 = NEIBcell[icel][2];
     icN4 = NEIBcell[icel][3];
     icN5 = NEIBcell[icel][4];
     icN6 = NEIBcell[icel][5];

     if(icN5 != 0) {
       icou = INL[icel] + 1;
       IAL[icel][icou-1] = icN5;
       INL[icel]         = icou;
     }

     if(icN3 != 0) {
       icou = INL[icel] + 1;
       IAL[icel][icou-1] = icN3;
       INL[icel]         = icou;
     }

     if(icN1 != 0) {
       icou = INL[icel] + 1;
       IAL[icel][icou-1] = icN1;
       INL[icel]         = icou;
     }
```



Lower Triangular Part

NEIBcell[icel][4]= icel – NX*NY + 1
NEIBcell[icel][2]= icel – NX + 1
NEIBcell[icel][0]= icel – 1 + 1

“icel” starts at 0

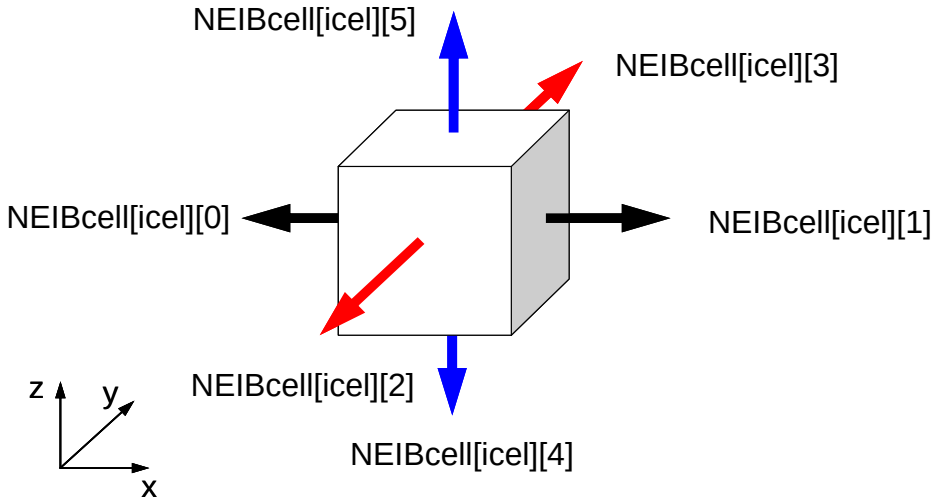
12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“IAL” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

poi_gen (3/7)

```
 for(icel=0; icel<ICELTOT; icel++) {

      icN1 = NEIBcell[icel][0];
      icN2 = NEIBcell[icel][1];
      icN3 = NEIBcell[icel][2];
      icN4 = NEIBcell[icel][3];
      icN5 = NEIBcell[icel][4];
      icN6 = NEIBcell[icel][5];
....
      if(icN2 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN2;
        INU[icel]         = icou;
      }

      if(icN4 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN4;
        INU[icel]         = icou;
      }

      if(icN6 != 0) {
        icou = INU[icel] + 1;
        IAU[icel][icou-1] = icN6;
        INU[icel]         = icou;
      }
}
```



Upper Triangular Part

NEIBcell[icel][1]= icel + 1 + 1
NEIBcell[icel][3]= icel + NX + 1
NEIBcell[icel][5]= icel + NX*NY + 1

“icel” starts at 0

12	13	14	15
8	9	10	11
4	5	6	7
0	1	2	3

“IAU” starts at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

poi_gen (4/7)

```
indexL =
   (int *)allocate_vector(sizeof(int),ICELTOT+1);
indexU =
   (int *)allocate_vector(sizeof(int),ICELTOT+1);

for(i=0; i<ICELTOT; i++){
    indexL[i+1]=indexL[i]+INL[i];
    indexU[i+1]=indexU[i]+INU[i];
}
NPL = indexL[ICELTOT];
NPU = indexU[ICELTOT];

itemL = (int *)allocate_vector(sizeof(int),NPL);
itemU = (int *)allocate_vector(sizeof(int),NPU);
AL =
(double *)allocate_vector(sizeof(double),NPL);
AU =
(double *)allocate_vector(sizeof(double),NPU);

memset(itemL, 0, sizeof(int)*NPL);
memset(itemU, 0, sizeof(int)*NPU);
memset(AL, 0.0, sizeof(double)*NPL);
memset(AU, 0.0, sizeof(double)*NPU);

for(i=0; i<ICELTOT; i++){
    for(k=0;k<INL[i];k++){
        kk= k + indexL[i];
        itemL[kk]= IAL[i][k];
    }
    for(k=0;k<INU[i];k++){
        kk= k + indexU[i];
        itemU[kk]= IAU[i][k];
    }
}

free(INL); free(INU);
free(IAL); free(IAU);
```

Name	Type	Content
D[N]	**R**	Diagonal components of the matrix (N= ICELTOT)
BFORCE[N]	**R**	RHS vector
PHI[N]	**R**	Unknown vector
indexL[N+1] **indexU[N+1]**	**I**	# of L/U non-zero off-diag. comp. (CRS)
NPL, NPU	**I**	Total # of L/U non-zero off-diag. comp. (CRS)
itemL[NPL] **itemU[NPU]**	**I**	Column ID of L/U non-zero off-diag. comp. (CRS)
AL[NPL] **AU[NPU]**	**R**	L/U non-zero off-diag. comp. (CRS)

“itemL” /”itemU” start at 1

13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

```
for (i=0; i<N; i++) {
    q[i]= D[i] * p[i];
    for (j=indexL[i]; j<indexL[i+1]; j++) {
        q[i] += AL[j] * p[itemL[j]-1];
    }
    for (j=indexU[i]; j<indexU[i+1]; j++) {
        q[i] += AU[j] * p[itemU[j]-1];
    }
}
```

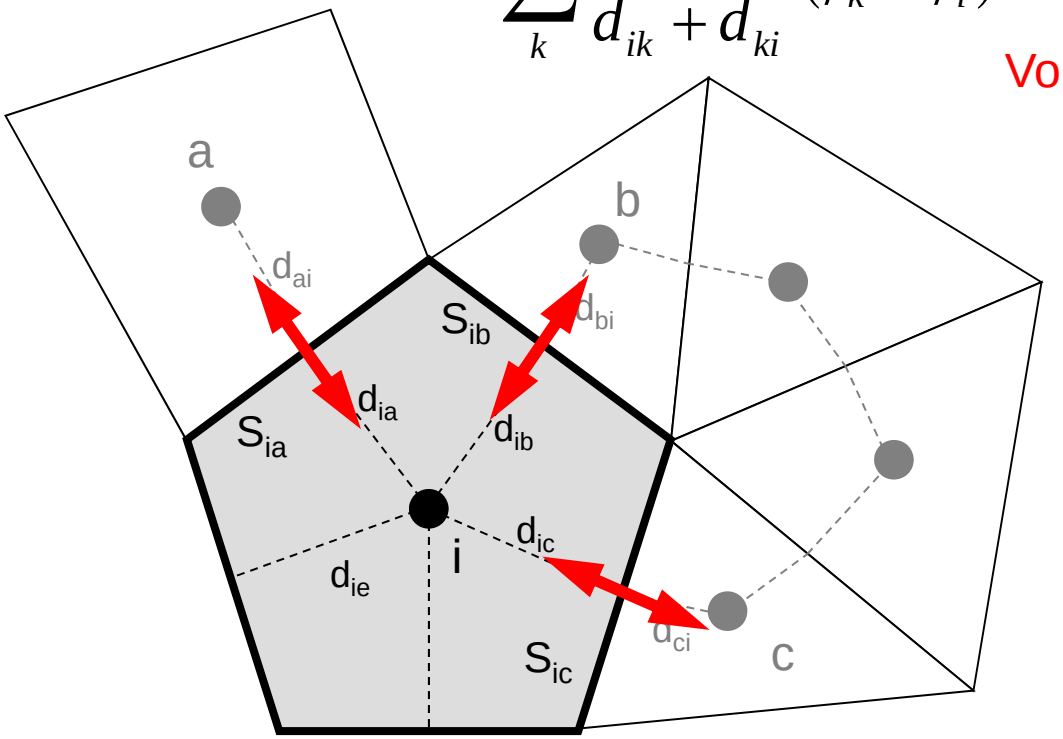
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



V_i : Volume
S : Surface Area
d_{ij} : Distance between Cell-Center & Surface
Q : Volume Flux

Constructing Coefficient Matrix

Conservation for i-th mesh

$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

$$+\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_i=-V_i\dot{Q}_i$$

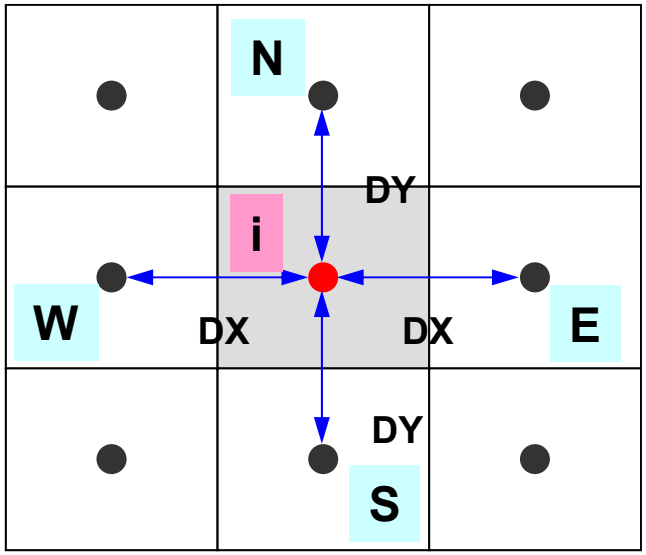
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) **AL, AU (off-diag.)** **BFORCE (RHS)**

poi_gen (5/7)

```
for(icel=0; icel<ICELTOT; icel++) {
    icN1 = NEIBcell[icel][0];
    icN2 = NEIBcell[icel][1];
    icN3 = NEIBcell[icel][2];
    icN4 = NEIBcell[icel][3];
    icN5 = NEIBcell[icel][4];
    icN6 = NEIBcell[icel][5];
   VOL0 = VOLCEL[icel];
    isL =indexL[icel  ];    ieL =indexL[icel+1];
    isU =indexU[icel  ];    ieU =indexU[icel+1];

    if(icN5 != 0) {
      coef = RDZ * ZAREA;
      D[icel] -= coef;
      for(j=isL; j<ieL; j++) {
          if(itemL[j] == icN5) {
            AL[j] = coef;
            break;}
      }
    }
    if(icN3 != 0) {
      coef = RDY * YAREA;
      D[icel] -= coef;
      for(j=isL; j<ieL; j++) {
          if(itemL[j] == icN3) {
            AL[j] = coef;
            break;}
      }
    }
    if(icN1 != 0) {
      coef = RDX * XAREA;
      D[icel] -= coef;
      for(j=isL; j<ieL; j++) {
          if(itemL[j] == icN1) {
            AL[j] = coef;
            break;}
      }
    }
```

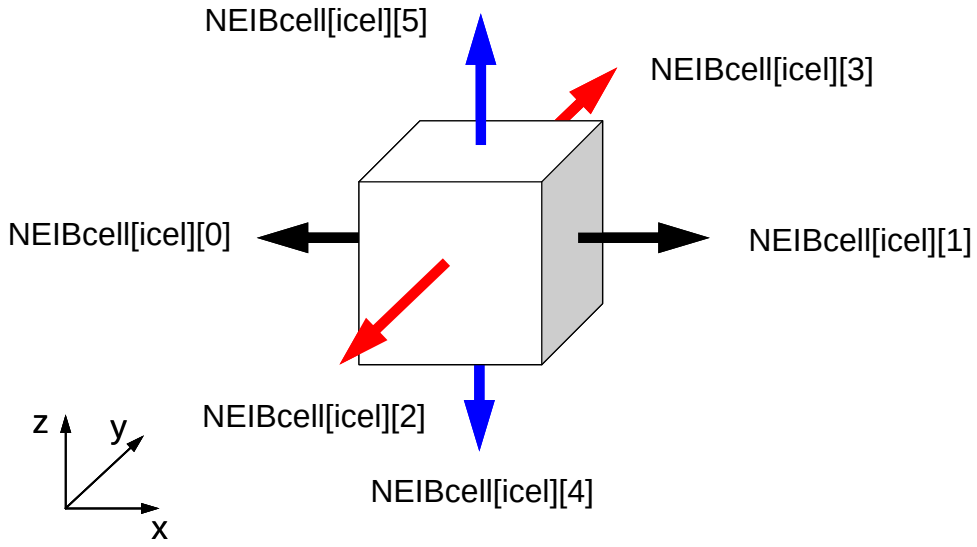
Calculation of Coefficients



$$\frac{\phi_E-\phi_i}{\Delta x}\Delta y+\frac{\phi_W-\phi_i}{\Delta x}\Delta y+$$

$$\frac{\phi_N-\phi_i}{\Delta y}\Delta x+\frac{\phi_S-\phi_i}{\Delta y}\Delta x+f_c\Delta x\Delta y=0$$

in 3D




```
if(icN5 != 0) {
   coef = RDZ * ZAREA;
   D[icel] -= coef;
   for(j=isL; j<ieL; j++) {
      if(itemL[j] == icN5) {
         AL[j] = coef;
         break;
      }
   }
}
```

$$\frac{\phi_{neib[icel][0]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+\frac{\phi_{neib[icel][1]}-\phi_{icel}}{\Delta x}\Delta y\Delta z+$$

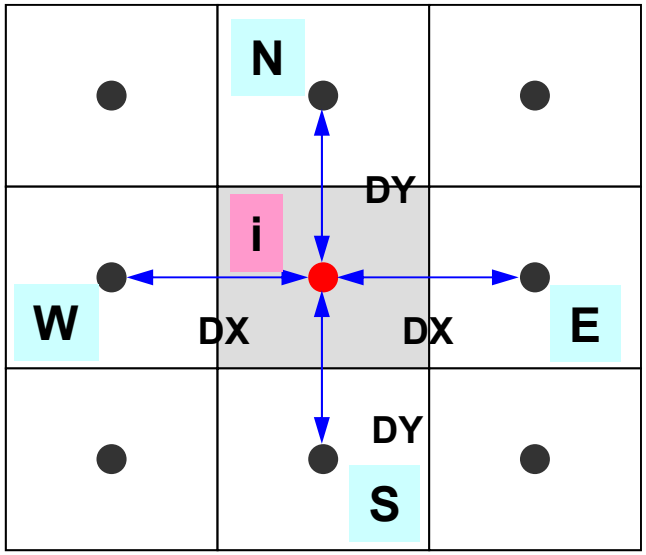
$$\frac{\phi_{neib[icel][2]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib[icel][3]}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

$$\frac{\phi_{neib[icel][4]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+\frac{\phi_{neib[icel][5]}-\phi_{icel}}{\Delta z}\Delta x\Delta y+f_{icel}\Delta x\Delta y\Delta z=0$$

poi_gen (6/7)

```
      if(icN2 != 0) {
        coef = RDX * XAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN2) {
               AU[j] = coef;
               break;}
        }
      }

      if(icN4 != 0) {
        coef = RDY * YAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN4) {
               AU[j] = coef;
               break;}
        }
      }

      if(icN6 != 0) {
        coef = RDZ * ZAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN6) {
               AU[j] = coef;
               break;}
        }
      }

      ii = XYZ[icel][0];
      jj = XYZ[icel][1];
      kk = XYZ[icel][2];

      BFORCE[icel]= -(double)(ii+jj+kk) *
                      VOLCEL[icel];
}
```

Calculation of Coefficients



$$\frac{\phi_E - \phi_i}{\Delta x}\Delta y + \frac{\phi_W - \phi_i}{\Delta x}\Delta y +$$

$$\frac{\phi_N - \phi_i}{\Delta y}\Delta x + \frac{\phi_S - \phi_i}{\Delta y}\Delta x + f_c \Delta x \Delta y = 0$$

poi_gen (6/7)

```
      if(icN2 != 0) {
        coef = RDX * XAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN2) {
               AU[j] = coef;
               break;}
        }
      }

      if(icN4 != 0) {
        coef = RDY * YAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN4) {
               AU[j] = coef;
               break;}
        }
      }

      if(icN6 != 0) {
        coef = RDZ * ZAREA;
        D[icel] -= coef;
        for(j=isU; j<ieU; j++) {
            if(itemU[j] == icN6) {
               AU[j] = coef;
               break;}
        }
      }

      ii = XYZ[icel][0];
      jj = XYZ[icel][1];
      kk = XYZ[icel][2];

      BFORCE[icel]= -(double)(ii+jj+kk) *
                     VOLCEL[icel];
}
```

Volume Flux

$$f = dfloat(i_0 + j_0 + k_0)$$

$$i_0 = XYZ[icel][0],$$

$$j_0 = XYZ[icel][1],$$

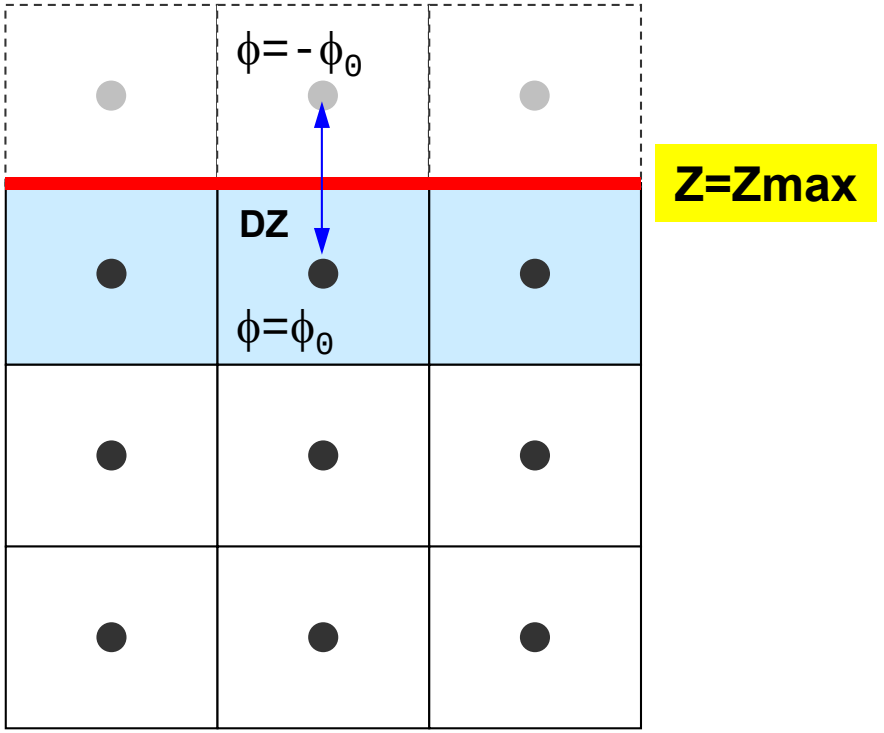
$$k_0 = XYZ[icel][2]$$

$XYZ[icel][k]$ (k=0,1,2)
Index for location of finite-difference mesh in X-/Y-/Z-axis.

```
/* TOP SURFACE */

        for(ib=0; ib<ZmaxCELtot; ib++) {
            icel  = ZmaxCEL[ib];
            coef = 2.0 * RDZ * ZAREA;
            D[icel-1] -= coef;
        }


        return 0;
}
```

poi_gen (7/7)

Calculation of Coefficients on Boundary Surface @ $Z=Z_{max}$



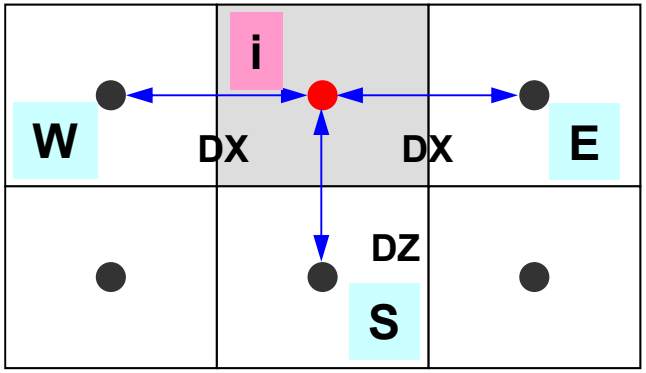
1st Order Approximation:

Mirror Image according to $Z=Z_{max}$ surface.

$\phi=-\phi_0$ at the center of the (virtual) mesh

$\phi=0$@$Z=Z_{max}$ surface

Dirichlet B.C.

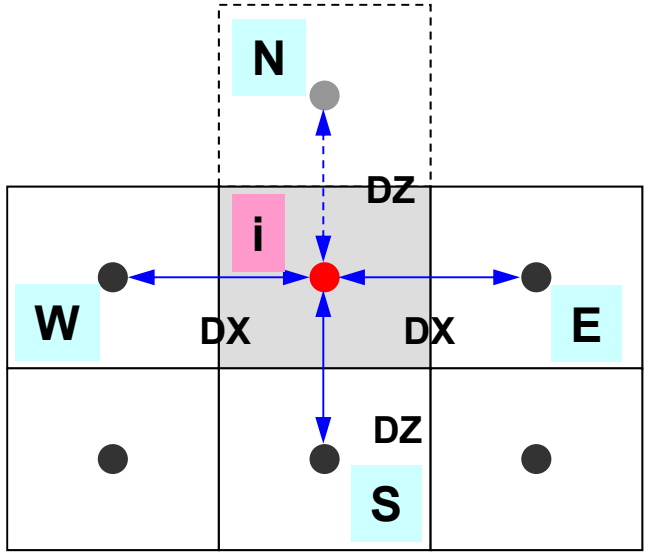


$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

Dirichlet B.C.



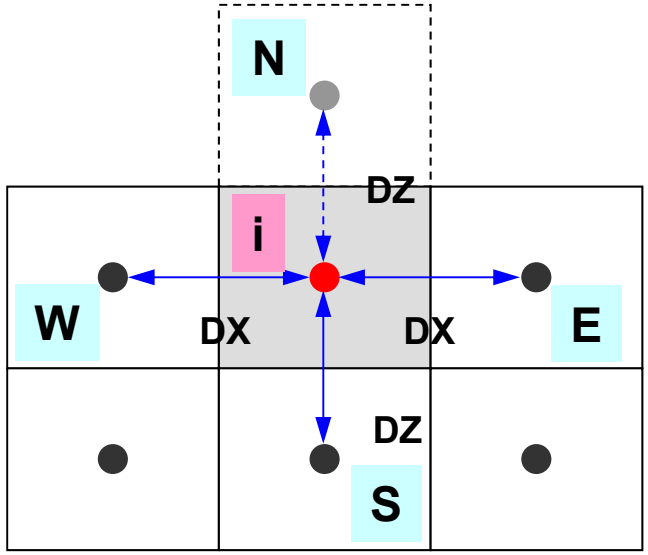
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{\phi_N-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i,\quad \phi_N=-\phi_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

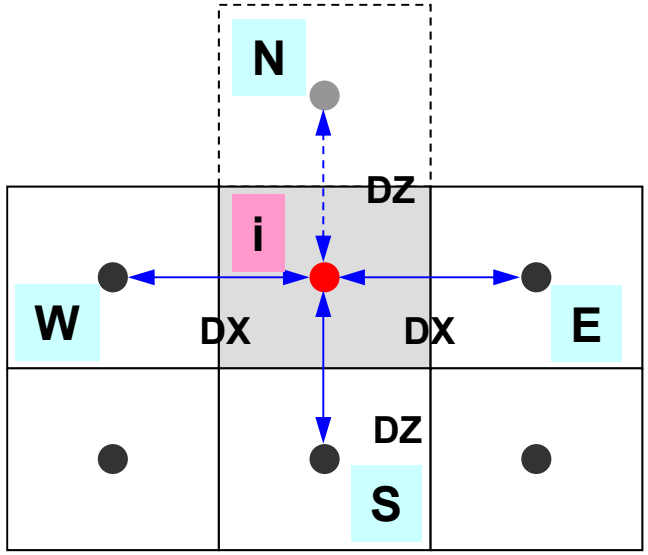
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{\phi_N-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i,\quad \phi_N=-\phi_i$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-\phi_i-\phi_i}{\Delta z}\Delta x\Delta y=-V_i\dot{Q}_i$$

Dirichlet B.C.



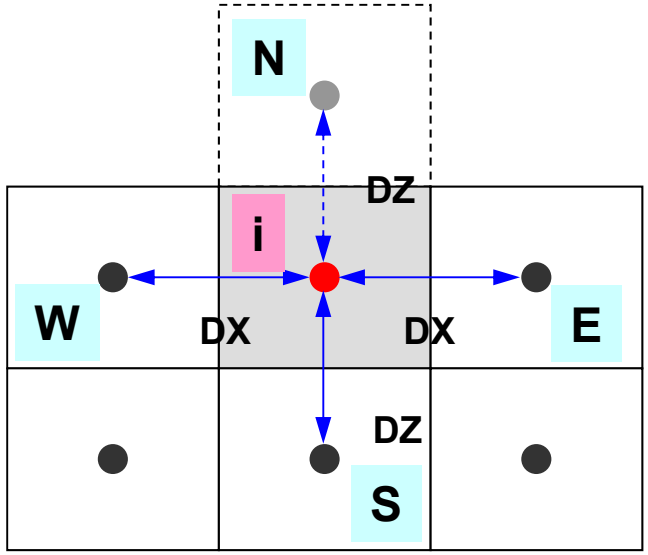
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-2\phi_i}{\Delta z}\Delta x\Delta y=+V_i\dot{Q}_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

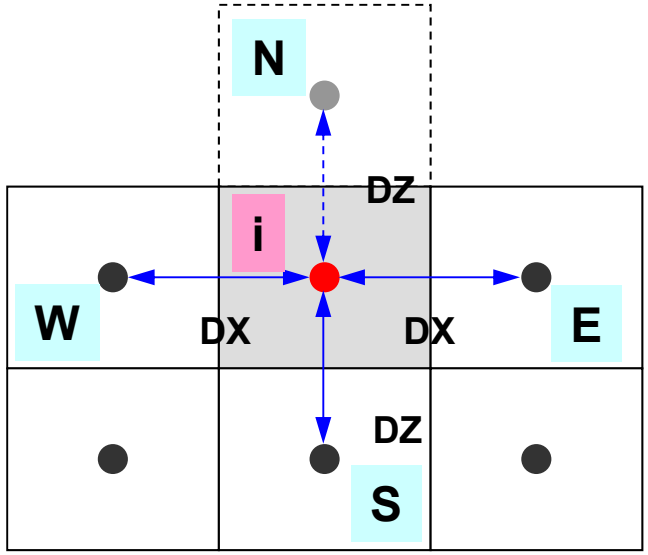
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+\frac{-2\phi_i}{\Delta z}\Delta x\Delta y=+V_i\dot{Q}_i$$

$$\left[-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}-\frac{2}{\Delta z}\Delta x\Delta y\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+=-V_i\dot{Q}_i$$

Dirichlet B.C.



$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

D (diagonal) | **AL, AU (off-diag.)** | **BFORCE (RHS)**

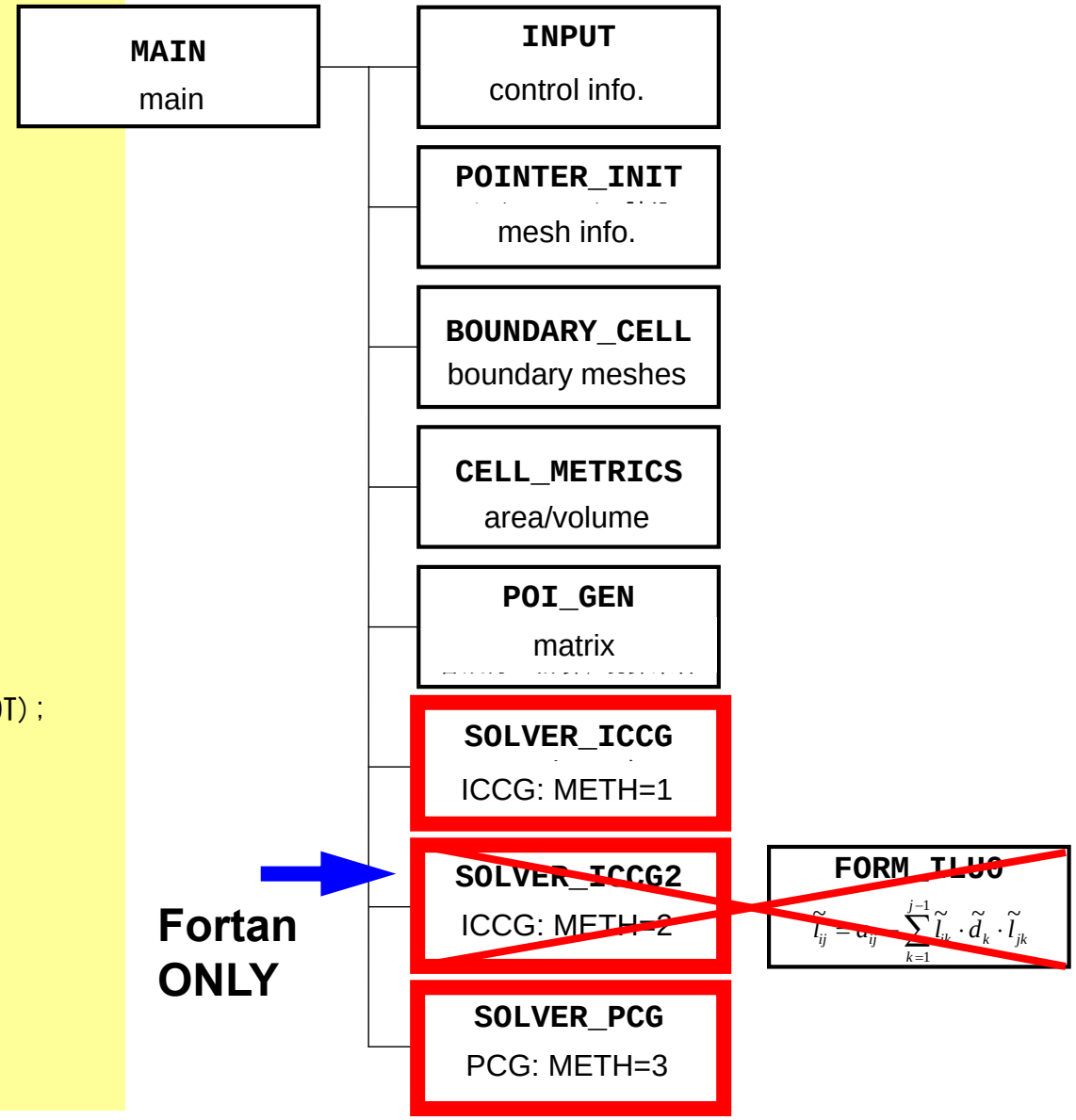
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]$$

```
for(ib=0; ib<ZmaxCELtot; ib++) {
    icel  = ZmaxCEL[ib];
    coef = 2.0 * RDZ * ZAREA;
    D[icel-1] -= coef;
}
```

$$\left[-\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}-\frac{2}{\Delta z}\Delta x\Delta y\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+=-V_i\dot{Q}_i$$

Structure of the Program

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>

#include "struct.h"
#include "pcg.h"
#include "input.h " ...

int
main()
{
  double *WK;
  int NPL, NPU; ISET, ITR, IER; icel, ic0, i;
  double xN, xL, xU; Stime, Etime;

  if(INPUT()) goto error;
  if(POINTER_INIT()) goto error;
  if(BOUNDARY_CELL()) goto error;
  if(CELL_METRICS()) goto error;
  if(POI_GEN()) goto error;

  memset(PHI, 0.0, sizeof(double)*ICELTOT);
  ISET = 0;
  WK = (double *)malloc(sizeof(double)*ICELTOT);

  if(METHOD==1){
     if(solve_ICCG(...)) goto error;
     } else if(METHOD==3){
     if(solve_PCG(...)) goto error;}

  if(OUTUCD()) goto error;
  return 0;
error:
  return -1;
}
```



- Background
  - Finite Volume Method
  - Preconditioned Iterative Solvers
- **ICCG Solver for Poisson Equations**
  - How to run
    - Data Structure
  - **Program**
    - Initialization
    - Coefficient Matrices
    - **ICCG**

Solving Linear Equations

- Conjugate Gradient, CG
- Preconditioner: Incomplete Cholesky Factorization, IC
  - Incomplete “Modified” Cholesky Factorization, more precisely

- ICCG

“Modified” Cholesky Factorization

- LU factorization of symmetric matrices
- Symmetric matrix [A] can be factorized into the form of $[A]= [L][D][L]^T$
  - LDL^T Factorization, Modified Cholesky Factorization
  - $[A]= [L][L]^T$ => Cholesky Factorization

N=5

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ 0 & l_{22} & l_{32} & l_{42} & l_{52} \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{bmatrix}$$

Incomplete "Modified" Cholesky Factorization (NO Fill-in)

$$\sum_{k=1}^{j} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \quad (j = 1,2,\cdots,i-1)$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{ik} = a_{ii}$$

if $l_{ii} \cdot d_i = 1$, following relationship is obtained:

$$\left[\begin{array}{l} i = 1,2,\cdots,n \\ \quad \left[\begin{array}{l} j = 1,2,\cdots,i-1 \\ \quad l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\ \quad d_i = \left(a_{ii} - \sum_{k=1}^{i-1} {l_{ik}}^2 \cdot d_k \right)^{-1} = {l_{ii}}^{-1} \end{array} \right. \end{array} \right.$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix} = \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ 0 & l_{22} & l_{32} & l_{42} & l_{52} \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{bmatrix}$$

$$= \begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix} \begin{bmatrix} d_1 \cdot l_{11} & d_1 \cdot l_{21} & d_1 \cdot l_{31} & d_1 \cdot l_{41} & d_1 \cdot l_{51} \\ 0 & d_2 \cdot l_{22} & d_2 \cdot l_{32} & d_2 \cdot l_{42} & d_2 \cdot l_{52} \\ 0 & 0 & d_3 \cdot l_{33} & d_3 \cdot l_{43} & d_3 \cdot l_{53} \\ 0 & 0 & 0 & d_4 \cdot l_{44} & d_4 \cdot l_{54} \\ 0 & 0 & 0 & 0 & d_5 \cdot l_{55} \end{bmatrix}$$

$$= \begin{bmatrix} l_{11} \cdot d_1 \cdot l_{11} & l_{11} \cdot d_1 \cdot l_{21} & l_{11} \cdot d_1 \cdot l_{31} & l_{11} \cdot d_1 \cdot l_{41} & l_{11} \cdot d_1 \cdot l_{51} \\ l_{21} \cdot d_1 \cdot l_{11} & l_{21} \cdot d_1 \cdot l_{21} + l_{22} \cdot d_2 \cdot l_{22} & l_{21} \cdot d_1 \cdot l_{31} + l_{22} \cdot d_2 \cdot l_{32} & l_{21} \cdot d_1 \cdot l_{41} + l_{22} \cdot d_2 \cdot l_{42} & l_{21} \cdot d_1 \cdot l_{51} + l_{22} \cdot d_2 \cdot l_{52} \\ l_{31} \cdot d_1 \cdot l_{11} & l_{31} \cdot d_1 \cdot l_{21} + l_{32} \cdot d_2 \cdot l_{22} & l_{31} \cdot d_1 \cdot l_{31} + l_{32} \cdot d_2 \cdot l_{32} + l_{33} \cdot d_3 \cdot l_{33} & l_{31} \cdot d_1 \cdot l_{41} + l_{32} \cdot d_2 \cdot l_{42} + l_{33} \cdot d_3 \cdot l_{43} & l_{31} \cdot d_1 \cdot l_{51} + l_{32} \cdot d_2 \cdot l_{52} + l_{33} \cdot d_3 \cdot l_{53} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{bmatrix}$$

$$\sum_{k=1}^{j} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \quad (j = 1,2,\cdots,i-1)$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{ik} = a_{ii}$$

Incomplete "Modified" Cholesky Factorization (NO Fill-in)

$$\sum_{k=1}^{j} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \quad (j = 1,2,\cdots,i-1)$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{ik} = a_{ii}$$

if $l_{ii} \cdot d_i = 1$, following relationship is obtained:

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \rightarrow \boxed{l_{ij} = a_{ij}}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} {l_{ik}}^2 \cdot d_k \right)^{-1} = {l_{ii}}^{-1}$$

Actually, more "incomplete" factorization is applied in practical use.

Running the Program

<$E-L1>/run/INPUT.DAT

```
32 32 32                                  NX/NY/NZ
1                                         MEHOD 1:2:3
1.00e-00 1.00e-00 1.00e-00                DX/DY/DZ
0.10      1.0e-08                         OMEGA, EPSICCG
```

- METHOD: Preconditioning Method
  1. Incomplete Modified Cholesky Fact. (Off-Diagonal Components unchanged)
  2. Incomplete Modified Cholesky Fact.(Fortran ONLY)
  3. Diagonal Scaling/Point Jacobi

$$
\begin{aligned}
&i = 1,2,\cdots,n \\
&\quad j = 1,2,\cdots,i-1 \\
&\qquad l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\
&\qquad d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\,2} \cdot d_k \right)^{-1} = l_{ii}^{\,-1}
\end{aligned}
$$

Incomplete "Modified" Cholesky Factorization (NO Fill-in)

$$\sum_{k=1}^{j} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \quad (j = 1,2,\cdots,i-1)$$

$$\sum_{k=1}^{i} l_{ik} \cdot d_k \cdot l_{ik} = a_{ii}$$

if $l_{ii} \cdot d_i = 1$, following relationship is obtained:

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \rightarrow \boxed{l_{ij} = a_{ij}}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\,2} \cdot d_k \right)^{-1} = l_{ii}^{\,-1}$$

Only diagonal components are changed

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

$$(M)\{z\} = \left(LDL^T\right)\{z\} = \{r\}$$

$$\{z\} = \left(LDL^T\right)^{-1}\{r\} \longrightarrow \begin{aligned} (L)\{y\} &= \{r\} \\ \left(DL^T\right)\{z\} &= \{y\} \end{aligned}$$

$$\begin{bmatrix} d_1 & 0 & 0 & 0 & 0 \\ 0 & d_2 & 0 & 0 & 0 \\ 0 & 0 & d_3 & 0 & 0 \\ 0 & 0 & 0 & d_4 & 0 \\ 0 & 0 & 0 & 0 & d_5 \end{bmatrix} \begin{bmatrix} l_{11} & l_{21} & l_{31} & l_{41} & l_{51} \\ 0 & l_{22} & l_{32} & l_{42} & l_{52} \\ 0 & 0 & l_{33} & l_{43} & l_{53} \\ 0 & 0 & 0 & l_{44} & l_{54} \\ 0 & 0 & 0 & 0 & l_{55} \end{bmatrix} = \begin{bmatrix} 1 & l_{21}/l_{11} & l_{31}/l_{11} & l_{41}/l_{11} & l_{51}/l_{11} \\ 0 & 1 & l_{32}/l_{22} & l_{42}/l_{22} & l_{52}/l_{22} \\ 0 & 0 & 1 & l_{43}/l_{33} & l_{53}/l_{33} \\ 0 & 0 & 0 & 1 & l_{54}/l_{44} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

$$(L)\{y\} = \{r\}$$

$$\left(DL^T\right)\{z\} = \{y\}$$

```
for(i=0; i<N; i++) {
    W[Y][i] = W[R][i];
}

for(i=0; i<N; i++) {
    WVAL = W[Y][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        WVAL -= AL[j] * W[Y][itemL[j]-1];
    }
    W[Y][i] = WVAL * W[DD][i];
}

for(i=N-1; i>=0; i--) {
    SW = 0.0;
    for(j=indexU[i]; j<indexU[i+1]; j++) {
    SW += AU[j] * W[Z][itemU[j]-1];
    }
    W[Z][i]= W[Y][i]  - W[DD][i] * SW;
}
```

$$W[DD][i] = 1/l_{ii} = d_{ii}$$

$$\begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix}$$

$$\begin{bmatrix} 1 & l_{21}/l_{11} & l_{31}/l_{11} & l_{41}/l_{11} & l_{51}/l_{11} \\ 0 & 1 & l_{32}/l_{22} & l_{42}/l_{22} & l_{52}/l_{22} \\ 0 & 0 & 1 & l_{43}/l_{33} & l_{53}/l_{33} \\ 0 & 0 & 0 & 1 & l_{54}/l_{44} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

$$(L)\{z\} = \{z\}$$

$$(DL^T)\{z\} = \{z\}$$

```
for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}

for(i=0; i<N; i++) {
    WVAL = W[Z][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        WVAL -= AL[j] * W[Z][itemL[j]-1];
    }
    W[Z][i] = WVAL * W[DD][i];
}

for(i=N-1; i>=0; i--) {
    SW = 0.0;
    for(j=indexU[i]; j<indexU[i+1]; j++) {
    SW += AU[j] * W[Z][itemU[j]-1];
    }
    W[Z][i]= W[Z][i]  - W[DD][i] * SW;
}
```

$$W[DD][i] = 1/l_{ii} = d_{ii}$$

$$\begin{bmatrix} l_{11} & 0 & 0 & 0 & 0 \\ l_{21} & l_{22} & 0 & 0 & 0 \\ l_{31} & l_{32} & l_{33} & 0 & 0 \\ l_{41} & l_{42} & l_{43} & l_{44} & 0 \\ l_{51} & l_{52} & l_{53} & l_{54} & l_{55} \end{bmatrix}$$

$$\begin{bmatrix} 1 & l_{21}/l_{11} & l_{31}/l_{11} & l_{41}/l_{11} & l_{51}/l_{11} \\ 0 & 1 & l_{32}/l_{22} & l_{42}/l_{22} & l_{52}/l_{22} \\ 0 & 0 & 1 & l_{43}/l_{33} & l_{53}/l_{33} \\ 0 & 0 & 0 & 1 & l_{54}/l_{44} \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

solve_ICCG (1/7): METHOD= 1

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <math.h> etc.

#include "solver_ICCG.h"

extern int
solve_ICCG (int N, int NL, int NU, int *indexL, int *itemL,
            int *indexU, int *itemU,
            double *D, double *B, double *X, double *AL,
            double *AU,
            double EPS, int *ITR, int *IER)
{
  double **W;
  double VAL, BNRM2, WVAL, SW, RHO, BETA, RHO1, C1, DNRM2;
  doble  ALPHA, ERR;

        int i, j, ic, ip, L, ip1;
        int R  = 0;
        int Z  = 1;
        int Q  = 1;
        int P  = 2;
        int DD = 3;
```

ICELTOT → N
BFORCE → B
PHI → X
EPSICCG→ EPS

W[0][i]= W[R][i] ⇒ {r}
W[1][i]= W[Z][i] ⇒ {z}
W[1][i]= W[Q][i] ⇒ {q}
W[2][i]= W[P][i] ⇒ {p}

W[3][i]= W[DD][i] ⇒{d}

solve_ICCG (2/7): METHOD= 1

```
W = (double **)malloc(sizeof(double *)*4);
if(W == NULL) {
   fprintf(stderr, "Error: %s¥n", strerror(errno));
   return -1;
}

for(i=0; i<4; i++) {
    W[i] = (double *)malloc(sizeof(double)*N);
    if(W[i] == NULL) {
       fprintf(stderr, "Error: %s¥n",
strerror(errno));
       return -1;
    }
}

for(i=0; i<N; i++) {
    X[i] = 0.0;
    W[1][i] = 0.0;
    W[2][i] = 0.0;
    W[3][i] = 0.0;
}



for(i=0; i<N; i++) {
    VAL = D[i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL = VAL - AL[j]*AL[j]*W[DD][itemL[j] - 1];
    }
    W[DD][i] = 1.0 / VAL;
}
```

W[DD][i]= d_i
in incomplete modified Cholesky factorization

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \rightarrow l_{ij} = a_{ij}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$

Only diagonal components are changed

W[DD][i]: d_i
D[i]: a_{ii}
itemL[j]: k
AL[j]: a_{ik}

solve_ICCG (3/7): METHOD= 1

```
for(i=0; i<N; i++) {
    VAL = D[i] * X[i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * X[itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * X[itemU[j]-1];
    }
    W[R][i] = B[i] - VAL;
}

BNRM2 = 0.0;
for(i=0; i<N; i++) {
    BNRM2 += B[i]*B[i];
}
```

BNRM2=|b|^2
Convergence criteria

Compute $\mathbf{r^{(0)}= b-[A]x^{(0)}}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)} z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1} p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG (4/7): METHOD= 1

```
*ITR = N;
for(L=0; L<(*ITR); L++) {

    for(i=0; i<N; i++) {
        W[Z][i] = W[R][i];
    }

    for(i=0; i<N; i++) {
        WVAL = W[Z][i];
        for(j=indexL[i]; j<indexL[i+1]; j++) {
            WVAL -= AL[j] * W[Z][itemL[j]-1];
        }
        W[Z][i] = WVAL * W[DD][i];
    }

    for(i=N-1; i>=0; i--) {
        SW = 0.0;
        for(j=indexU[i]; j<indexU[i+1]; j++) {
        SW += AU[j] * W[Z][itemU[j]-1];
        }
        W[Z][i]= W[Z][i]  - W[DD][i] * SW;
    }
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG (4/7): METHOD= 1

$$(M)\{z\} = (LDL^T)\{z\} = \{r\}$$

```
for(i=0; i<N; i++) {
    W[Z][i] = W[R][i];
}
for(i=0; i<N; i++) {
    WVAL = W[Z][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        WVAL -= AL[j] * W[Z][itemL[j]-1];
    }
    W[Z][i] = WVAL * W[DD][i];
}
```

$(L)\{z\} = \{r\}$

前進代入
Forward Substitution

```
for(i=N-1; i>=0; i--) {
    SW = 0.0;
    for(j=indexU[i]; j<indexU[i+1]; j++) {
    SW += AU[j] * W[Z][itemU[j]-1];
    }
    W[Z][i]= W[Z][i]  - W[DD][i] * SW;
}
```

$(DL^T)\{z\} = \{z\}$

後退代入
Backward Substitution

solve_ICCG (5/7): METHOD= 1

```
/****************
 * RHO = {r}{z} *
 ****************/

RHO = 0.0;
for(i=0; i<N; i++) {
    RHO += W[R][i] * W[Z][i];
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG (6/7): METHOD= 1

```
/*********************************
 * {p}  = {z} if      ITER=0    *
 * BETA = RHO / RHO1  otherwise *
 *********************************/

if(L == 0) {
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i];
   }
   } else {
   BETA = RHO / RHO1;
   for(i=0; i<N; i++) {
       W[P][i] = W[Z][i] + BETA * W[P][i];
   }
}

/*****************
 * {q} = [A]{p} *
 *****************/

for(i=0; i<N; i++) {
    VAL = D[i] * W[P][i];
    for(j=indexL[i]; j<indexL[i+1]; j++) {
        VAL += AL[j] * W[P][itemL[j]-1];
    }
    for(j=indexU[i]; j<indexU[i+1]; j++) {
        VAL += AU[j] * W[P][itemU[j]-1];
    }
    W[Q][i] = VAL;
}
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG (7/7): METHOD= 1

```
/*************************
 * ALPHA = RHO / {p}{q} *
 *************************/
C1 = 0.0;
for(i=0; i<N; i++) {
    C1 += W[P][i] * W[Q][i];
}
ALPHA = RHO / C1;

/*****************************
 * {x} = {x} + ALPHA * {p} *
 * {r} = {r} - ALPHA * {q} *
 *****************************/
for(i=0; i<N; i++) {
    X[i]    += ALPHA * W[P][i];
    W[R][i] -= ALPHA * W[Q][i];
}

DNRM2 = 0.0;
for(i=0; i<N; i++) {
    DNRM2 += W[R][i]*W[R][i];
}

ERR = sqrt(DNRM2/BNRM2);
if((L+1)%100 ==1) {
   fprintf(stderr, "%5d%16.6e¥n", L+1, ERR);
}
if(ERR < EPS) {
   *IER = 0; goto N900;
  } else {
   RHO1 = RHO;
  }
}
*IER = 1;
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG (7/7): METHOD= 1

```
/*************************
 * ALPHA = RHO / {p}{q} *
 *************************/
C1 = 0.0;
for(i=0; i<N; i++) {
    C1 += W[P][i] * W[Q][i];
}
ALPHA = RHO / C1;

/*****************************
 * {x} = {x} + ALPHA * {p} *
 * {r} = {r} - ALPHA * {q} *
 *****************************/
for(i=0; i<N; i++) {
    X[i]    += ALPHA * W[P][i];
    W[R][i] -= ALPHA * W[Q][i];
}

DNRM2 = 0.0;
for(i=0; i<N; i++) {
    DNRM2 += W[R][i]*W[R][i];
}

ERR = sqrt(DNRM2/BNRM2);
if((L+1)%100 ==1) {
   fprintf(stderr, "%5d%16.
}
if(ERR < EPS) {
   *IER = 0; goto N900;
  } else {
   RHO1 = RHO;
  }
}
*IER = 1;
```

r= b-[A]x
DNRM2=|r|²
BNRM2=|b|²

ERR= |r|/|b|

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG2 (1/3): METHOD= 2

Fortran ONLY

```
!C
!C***
!C*** module solver_ICCG2
!C***
!
      module solver_ICCG2
      contains
!C
!C*** solve_ICCG2
!C
      subroutine solve_ICCG2                                          &
     &  ( N, NPL, NPU, indexL, itemL, indexU, itemU, D, B, X,         &
     &    AL, AU, EPS, ITR, IER)

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

      real(kind=8), dimension(N)   :: D
      real(kind=8), dimension(N)   :: B
      real(kind=8), dimension(N)   :: X
      real(kind=8), dimension(NPL) :: AL
      real(kind=8), dimension(NPU) :: AU

      integer, dimension(0:N)   :: indexL, indexU
      integer, dimension(NPL):: itemL
      integer, dimension(NPU):: itemU

      real(kind=8), dimension(:,:), allocatable :: W

      integer, parameter ::  R= 1
      integer, parameter ::  Z= 2
      integer, parameter ::  Q= 2
      integer, parameter ::  P= 3
      integer, parameter :: DD= 4

      real(kind=8), dimension(:), allocatable :: ALlu0, AUlu0
      real(kind=8), dimension(:), allocatable :: Dlu0
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

solve_ICCG2 (2/3): METHOD= 2

Fortran ONLY

```
!C
!C +------+
!C | INIT |
!C +------+
!C===
      allocate (W(N,4))

      do i= 1, N
        X(i)   = 0.d0
        W(i,1)= 0.0D0
        W(i,2)= 0.0D0
        W(i,3)= 0.0D0
        W(i,4)= 0.0D0
      enddo

      call FORM_ILU0
!C===
```

Dlu0, ALlu0, AUlu0:

Factorized Matrix Components

FORM_ILU0 (1/2): Fortran only

Incomplete Modified LU Factorization in solve_ICCG2

```
 contains

!C
!C***
!C*** FORM_ILU0
!C***
!C
!C    form ILU(0) matrix
!C
      subroutine FORM_ILU0
      implicit REAL*8 (A-H,O-Z)
      integer, dimension(:), allocatable :: IW1 , IW2
      integer, dimension(:), allocatable :: IWsL, IWsU
      real (kind=8):: RHS_Aij, DkINV, Aik, Akj

!C
!C +-------+
!C | INIT. |
!C +-------+
!C===
      allocate (ALlu0(NPL), AUlu0(NPU))
      allocate (Dlu0(N))

      do i= 1, N
        Dlu0(i)= D(i)
        do k= 1, INL(i)
          ALlu0(k,i)= AL(k,i)
        enddo

        do k= 1, INU(i)
          AUlu0(k,i)= AU(k,i)
        enddo
      enddo
!C===
```

$$
\left[
\begin{array}{l}
i = 1,2,\cdots,n \\
\quad \left[
\begin{array}{l}
j = 1,2,\cdots,i-1 \\
\qquad l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\
\qquad d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}
\end{array}
\right.
\end{array}
\right.
$$

Dlu0, ALlu0, AUlu0:

Factorized Matrix Components

Initial Conditions

Dlu0 = D

ALlu0= AL

AUlu0= AU

FORM_ILU0 (2/2): Fortran only

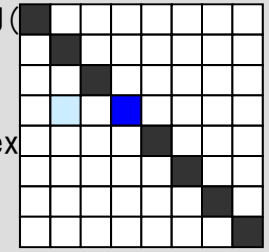
```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
      allocate (IW1(N) , IW2(N))
      IW1=0
      IW2= 0

      do i= 1, N
        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= k0
        enddo

        do k0= indexU(i-1)+1, indexU(i)
          IW2(itemU(k0))= k0
        enddo

        do icon= indexL(i-1)+1, indexL(i)
            k= itemL(icon)
          D11= DluO(k)

          DkINV= 1.d0/D11
            Aik= ALluO(icon)

          do kcon= indexU(k-1)+1, indexU(k)
            j= itemU(kcon)

            if (j.eq.i) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj
              DluO(i)= DluO(i) - RHS_Aij
            endif

            if (j.lt.i .and. IW1(j).ne.0) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj

                       ijO = IW1(j)
                ALluO(ijO)= ALluO(ijO) - RHS_Aij
            endif
```

```
            if (j.gt.i .and. IW2(j).ne.0) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj

                       ijO = IW2(j)
              AUluO(ijO)= AUluO(ijO) - RHS_Aij
            endif

          enddo
        enddo

        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= 0
        enddo

        do k0= indexU(i-1)+1, indexU(i)
          IW2(itemU(k0))= 0
        enddo
      enddo

      do i= 1, N
        DluO(i)= 1.d0 / DluO(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)=  A(i,j)                          &
                  -A(i,k)*(A(k,k))-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

FORM_ILU0 (2/2): Fortran only

```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
      allocate (IW1(N) , IW2(N))
      IW1=0
      IW2= 0

      do i= 1, N
        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= k0
        enddo

        do k0= indexU(i-1)+1, indexU(
          IW2(itemU(k0))= k0
        enddo

        do icon= indexL(i-1)+1, index
           k= itemL(icon)
          D11= DLU0(k)

          DkINV= 1.d0/D11
            Aik= ALLU0(icon)

          do kcon= indexU(k-1)+1, indexU(k)
            j= itemU(kcon)

            if (j.eq.i) then
                  Akj= AULU0(kcon)
              RHS_Aij= Aik * DkINV * Akj
              DLU0(i)= DLU0(i) - RHS_Aij
            endif

            if (j.lt.i .and. IW1(j).ne.0) then
                  Akj= AULU0(kcon)
              RHS_Aij= Aik * DkINV * Akj

                     ij0 = IW1(j)
                ALLU0(ij0)= ALLU0(ij0) - RHS_Aij
            endif
```

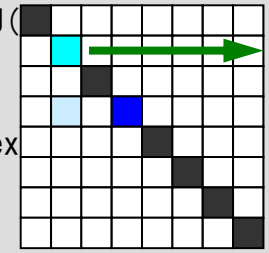


```
            if (j.gt.i .and. IW2(j).ne.0) then
                  Akj= AULU0(kcon)
              RHS_Aij= Aik * DkINV * Akj

                     ij0 = IW2(j)
              AULU0(ij0)= AULU0(ij0) - RHS_Aij
             endif

          enddo
        enddo

        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= 0
        enddo

        do k0= indexU(i-1)+1, indexU(i)
          IW2(itemU(k0))= 0
        enddo
      enddo

      do i= 1, N
        DLU0(i)= 1.d0 / DLU0(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)=  A(i,j)                              &
                  -A(i,k)*(A(k,k))-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

FORM_ILU0 (2/2): Fortran only

```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
      allocate (IW1(N) , IW2(N))
      IW1=0
      IW2= 0

      do i= 1, N
        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= k0
        enddo

        do k0= indexU(i-1)+1, indexU(
          IW2(itemU(k0))= k0
        enddo

        do icon= indexL(i-1)+1, index
            k= itemL(icon)
          D11= DluO(k)

          DkINV= 1.d0/D11
            Aik= ALluO(icon)

          do kcon= indexU(k-1)+1, indexU(k)
            j= itemU(kcon)

            if (j.eq.i) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj
              DluO(i)= DluO(i) - RHS_Aij
            endif

            if (j.lt.i .and. IW1(j).ne.0) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj

                     ijO = IW1(j)
                ALluO(ijO)= ALluO(ijO) - RHS_Aij
            endif
```



```
            if (j.gt.i .and. IW2(j).ne.0) then
                  Akj= AUluO(kcon)
              RHS_Aij= Aik * DkINV * Akj

                     ijO = IW2(j)
                AUluO(ijO)= AUluO(ijO) - RHS_Aij
            endif

          enddo
        enddo

        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= 0
        enddo

        do k0= indexU(i-1)+1, indexU(i)
          IW2(itemU(k0))= 0
        enddo
      enddo

      do i= 1, N
        DluO(i)= 1.d0 / DluO(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)=  A(i,j)                          &
                  -A(i,k)*(A(k,k))-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

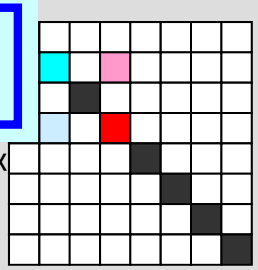
FORM_ILU0 (2/2): Fortran only

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$



```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
      do icon= indexL(i-1)+1, index
        k= itemL(icon)
      D11= DluO(k)

      DkINV= 1.dO/D11
        Aik= ALluO(icon)

      do kcon= indexU(k-1)+1, indexU(k)
        j= itemU(kcon)

        if (j.eq.i) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj
          DluO(i)= DluO(i) - RHS_Aij
        endif

        if (j.lt.i .and. IW1(j).ne.0) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj

                 ijO = IW1(j)
            ALluO(ijO)= ALluO(ijO) - RHS_Aij
        endif
```

j=i

```
        if (j.gt.i .and. IW2(j).ne.0) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj

                 ijO = IW2(j)
            AUluO(ijO)= AUluO(ijO) - RHS_Aij
        endif

      enddo
    enddo

    do kO= indexL(i-1)+1, indexL(i)
      IW1(itemL(kO))= 0
    enddo

    do kO= indexU(i-1)+1, indexU(i)
      IW2(itemU(kO))= 0
    enddo
  enddo

  do i= 1, N
    DluO(i)= 1.dO / DluO(i)
  enddo
  deallocate (IW1, IW2)
!C===
  end subroutine FORM_ILUO
```

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)=  A(i,j)                          &
                  -A(i,k)*(A(k,k))-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

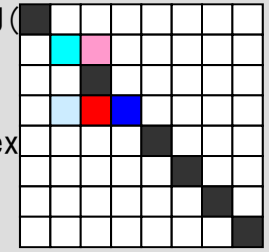
FORM_ILU0 (2/2): Fortran only

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$



```
do k0= indexU(i-1)+1, indexU(
  IW2(itemU(k0))= k0
enddo

do icon= indexL(i-1)+1, index
    k= itemL(icon)
  D11= DIuO(k)

  DkINV= 1.d0/D11
    Aik= ALIuO(icon)

do kcon= indexU(k-1)+1, indexU(k)
    j= itemU(kcon)

    if (j.eq.i) then
          Akj= AUIuO(kcon)
      RHS_Aij= Aik * DkINV * Akj
      DIuO(i)= DIuO(i) - RHS_Aij
    endif

    if (j.lt.i .and. IW1(j).ne.0) then
          Akj= AUIuO(kcon)
      RHS_Aij= Aik * DkINV * Akj

               ij0 = IW1(j)
        ALIuO(ij0)= ALIuO(ij0) - RHS_Aij
    endif
```

j<i

```
            if (j.gt.i .and. IW2(j).ne.0) then
                  Akj= AUIuO(kcon)
              RHS_Aij= Aik * DkINV * Akj

                     ij0 = IW2(j)
              AUIuO(ij0)= AUIuO(ij0) - RHS_Aij
            endif

          enddo
        enddo

        do k0= indexL(i-1)+1, indexL(i)
          IW1(itemL(k0))= 0
        enddo

        do k0= indexU(i-1)+1, indexU(i)
          IW2(itemU(k0))= 0
        enddo
      enddo

      do i= 1, N
        DIuO(i)= 1.d0 / DIuO(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)= A(i,j)                          &
                  -A(i,k)*(A(k,k))-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

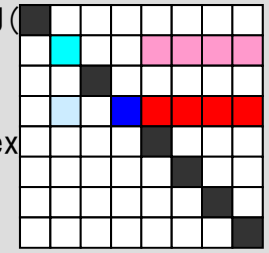
FORM_ILU0 (2/2): Fortran only

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$



```
      do k0= indexU(i-1)+1, indexU(
        IW2(itemU(k0))= k0
      enddo

      do icon= indexL(i-1)+1, index
          k= itemL(icon)
        D11= DluO(k)

        DkINV= 1.d0/D11
          Aik= ALluO(icon)

      do kcon= indexU(k-1)+1, indexU(k)
          j= itemU(kcon)

        if (j.eq.i) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj
          DluO(i)= DluO(i) - RHS_Aij
        endif

        if (j.lt.i .and. IW1(j).ne.0) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj

                   ijO = IW1(j)
            ALluO(ijO)= ALluO(ijO) - RHS_Aij
        endif
```

```
        if (j.gt.i .and. IW2(j).ne.0) then
              Akj= AUluO(kcon)
          RHS_Aij= Aik * DkINV * Akj

                   ijO = IW2(j)
            AUluO(ijO)= AUluO(ijO) - RHS_Aij
        endif

        enddo
      enddo

      do k0= indexL(i-1)+1, indexL(i)
        IW1(itemL(k0))= 0
      enddo

      do k0= indexU(i-1)+1, indexU(i)
        IW2(itemU(k0))= 0
      enddo
    enddo

    do i= 1, N
      DluO(i)= 1.d0 / DluO(i)
    enddo
    deallocate (IW1, IW2)
!C===
    end subroutine FORM_ILU0
```

j>i

```
do i= 1, N
  do k= 1, i-1
    if (A(i,k) is non-zero) then
      do j= k+1, N
        if (A(i,j) is non-zero) then
          A(i,j)=  A(i,j)                          &
                  -A(i,k)*(A(k,k))^-1*A(k,j)
        endif
      enddo
    endif
  enddo
enddo
```

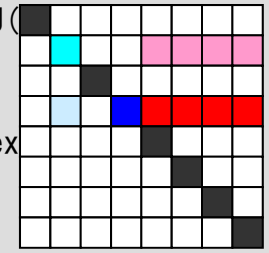
FORM_ILU0 (2/2): Fortran only

$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}$$



```
      do k0= indexU(i-1)+1, indexU(
        IW2(itemU(k0))= k0
      enddo

      do icon= indexL(i-1)+1, index
          k= itemL(icon)
        D11= DIuO(k)

        DkINV= 1.d0/D11
          Aik= ALIuO(icon)

      do kcon= indexU(k-1)+1, indexU(k)
          j= itemU(kcon)

          if (j.eq.i) then
                Akj= AUIuO(kcon)
            RHS_Aij= Aik * DkINV * Akj
            DIuO(i)= DIuO(i) - RHS_Aij
          endif

          if (j.lt.i .and. IW1(j).ne.0) then
                Akj= AUIuO(kcon)
            RHS_Aij= Aik * DkINV * Akj

                   ij0 = IW1(j)
              ALIuO(ij0)= ALIuO(ij0) - RHS_Aij
          endif
```

j<i

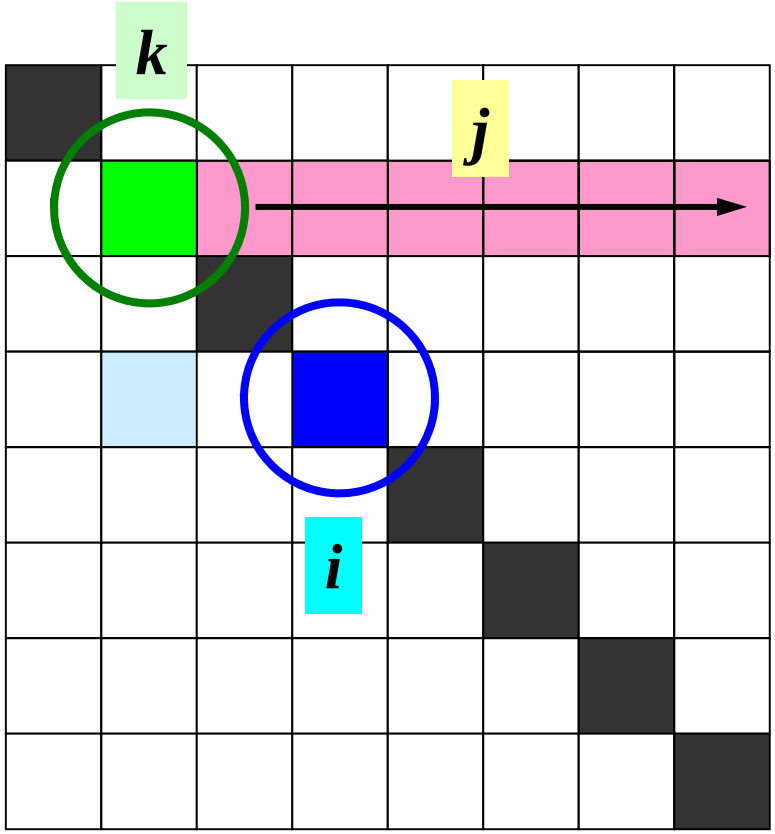
```
          if (j.gt.i .and. IW2(j).ne.0) then
                Akj= AUIuO(kcon)
            RHS_Aij= Aik * DkINV * Akj

                   ij0 = IW2(j)
              AUIuO(ij0)= AUIuO(ij0) - RHS_Aij
          endif

        enddo
      enddo

      do k0= indexL(i-1)+1, indexL(i)
        IW1(itemL(k0))= 0
      enddo

      do k0= indexU(i-1)+1, indexU(i)
        IW2(itemU(k0))= 0
      enddo
    enddo

    do i= 1, N
      DIuO(i)= 1.d0 / DIuO(i)
    enddo
    deallocate (IW1, IW2)
!C===
    end subroutine FORM_ILU0
```

j>i

These if –then clauses never applied, therefore:

ALIuO= AL
AUIuO= AU

j=*i*, *j*<*i*, *j*>*i* (1/3)



○: Mesh i

○: Mesh k (Lower Triangular Component of i)

■: Mesh j (Upper Triangular Component of k)

if *j*=*i* Dlu(■) is updated

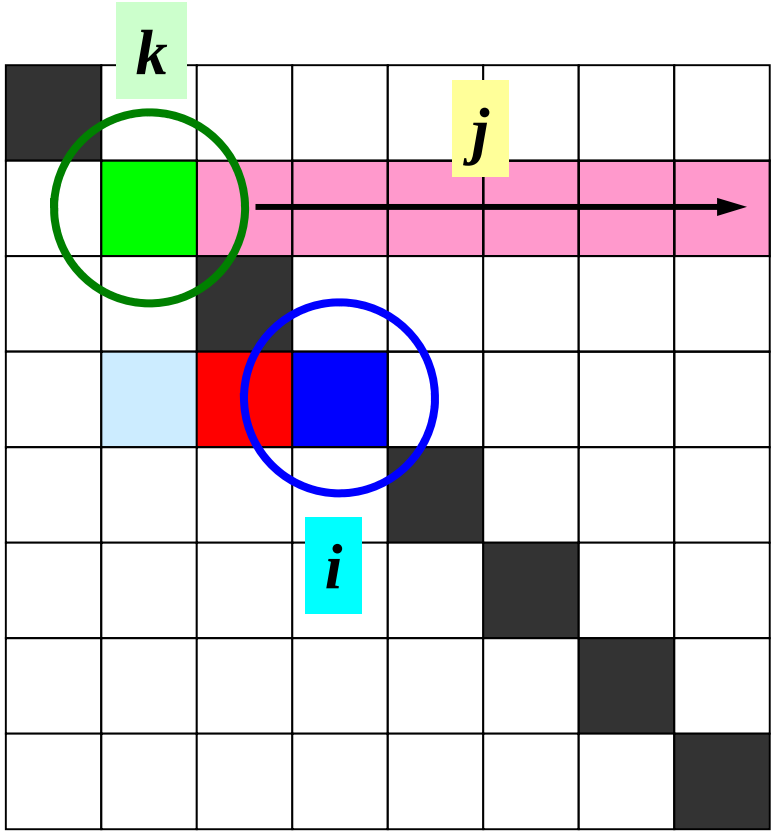
$$i = 1,2,\cdots,n$$

$$j = 1,2,\cdots,i-1$$

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

$$d_i = \left(a_{ii} - \sum_{k=1}^{i-1} {l_{ik}}^2 \cdot d_k \right)^{-1} = {l_{ii}}^{-1}$$

j=*i*, *j*<*i*, *j*>*i* (2/3)



○: Mesh i

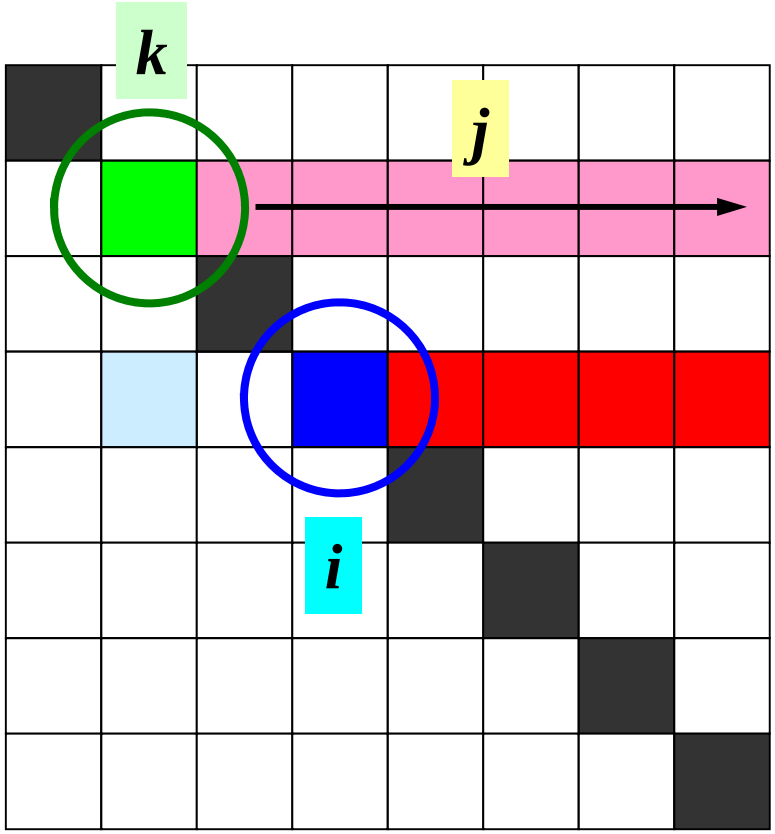
○: Mesh k (Lower Triangular Component of i)

■: Mesh j (Upper Triangular Component of k)

if *j*<*i* ALlu0(i-j)(■) is updated

$$
\left[
\begin{array}{l}
i = 1,2,\cdots,n \\
\quad \left[
\begin{array}{l}
j = 1,2,\cdots,i-1 \\
\quad \boxed{l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}} \\
\quad d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}{}^2 \cdot d_k \right)^{-1} = l_{ii}{}^{-1}
\end{array}
\right.
\end{array}
\right.
$$

$j=i$, $j<i$, $j>i$ (3/3)



○: Mesh i

○: Mesh k (Lower Triangular Component of i)

■: Mesh j (Upper Triangular Component of k)

if $j>i$ AUlu0(i-j)(■) is updated

Actually, there are no cases for:

- **$j<i$**
- **$j>i$**

$$
\left[
\begin{array}{l}
i = 1,2,\cdots,n \\
\quad \left[
\begin{array}{l}
j = 1,2,\cdots,i-1 \\
\quad \boxed{l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}} \\
d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}^{\ 2} \cdot d_k \right)^{-1} = l_{ii}^{\ -1}
\end{array}
\right.
\end{array}
\right.
$$

solve_ICCG2 (3/3): METHOD= 2

Fortran ONLY

```
!C
!C*****************************************************  ITERATION
      ITR= N

      do L= 1, ITR
!C
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===
      do i= 1, N
        W(i,Z)= W(i,R)
      enddo

      do i= 1, N
        WVAL= W(i,Z)
        do k= indexL(i-1)+1, indexL(i)
          WVAL= WVAL -  ALluO(k) * W(itemL(k),Z)
        enddo
        W(i,Z)= WVAL * DluO(i)
      enddo

      do i= N, 1, -1
        SW  = 0.0d0
        do k= indexU(i-1)+1, indexU(i)
          SW= SW + AUluO(k) * W(itemU(k),Z)
        enddo
        W(i,Z)= W(i,Z) - DluO(i)*SW
      enddo
!C===
```

Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
  solve $[M]z^{(i-1)}= r^{(i-1)}$
  $\rho_{i-1}= r^{(i-1)}\ z^{(i-1)}$
  if i=1
    $p^{(1)}= z^{(0)}$
  else
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i-1)}$
  endif
  $q^{(i)}= [A]p^{(i)}$
  $\alpha_i\ = \rho_{i-1}/p^{(i)}q^{(i)}$
  $x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$
  $r^{(i)}= r^{(i-1)} - \alpha_i q^{(i)}$
  check convergence |r|
end

Other parts are as same as those in “solve_ICCG”

solve_ICCG2 (3/3): METHOD= 2

Fortran ONLY

```
!C
!C*****************************************************  ITERATION
      ITR= N

      do L= 1, ITR
!C
!C +---------------+
!C | {z}= [Minv]{r} |
!C +---------------+
!C===
      do i= 1, N
        W(i,Z)= W(i,R)
      enddo

       do i= 1, N
        WVAL= W(i,Z)
        do k= indexL(i-1)+1, indexL(i)
          WVAL= WVAL -  ALlu0(k) * W(itemL(k),Z)
        enddo
        W(i,Z)= WVAL * Dlu0(i)
      enddo

      do i= N, 1, -1
        SW  = 0.0d0
        do k= indexU(i-1)+1, ind
          SW= SW + AUlu0(k) * W(itemU(k),Z)
        enddo
        W(i,Z)= W(i,Z) - Dlu0(i)*SW
      enddo
!C===
```

$$(M)\{z\} = (LDL^T)\{z\} = \{r\}$$

$$(L)\{z\} = \{r\}$$

Forward Substitution

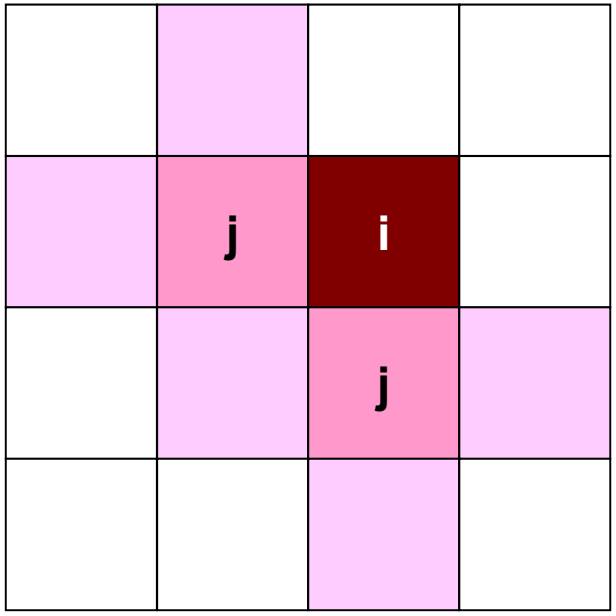
$$(DL^T)\{z\} = \{z\}$$

Backward Substitution

ALlu0=AL, AUlu0=AU: Therefore, iterations for convergence for METHOD=1, and those for METHOD=2 are same.

Incomplete Modified Cholesky Fact. Structured Meshes

$$
\begin{aligned}
&i = 1,2,\cdots,n \\
&\quad j = 1,2,\cdots,i-1 \\
&\qquad l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\
&\qquad d_i = \left(a_{ii} - \sum_{k=1}^{i-1} l_{ik}{}^2 \cdot d_k \right)^{-1} = l_{ii}{}^{-1}
\end{aligned}
$$



There are no k-mesh which is adjacent to both of i and j simultaneously. Therefore, $l_{ij} = a_{ij}$.

In this case, ALlu0/AUlu0 could be changed