

Introduction to Parallel Programming for Multicore/Manycore Clusters

Part B1: FVM Code (ICCG)

Kengo Nakajima
Information Technology Center
The University of Tokyo

Files on PC

Files on WEB:

<http://nkl.cc.u-tokyo.ac.jp/files/multicore-f.tar>

```
>$ tar xvf multicore-f.tar
```

```
>$ cd multicore-f
```

Please confirm that following directories are created:

L1 L2

PC

Odyssey

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

Target of the Class

- Material: ICCG solver for sparse matrices derived from FVM applications (Finite Volume Method).
- Parallelization on a single node of Oakbridge-CX (OBCX) using OpenMP
 - Data Placement
 - Reordering
- Keywords
 - Finite Volume Method (FVM)
 - Sparse Matrices
 - ICCG Method

Target Application

- 3D Poisson Equation/Poisson's Equation

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

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

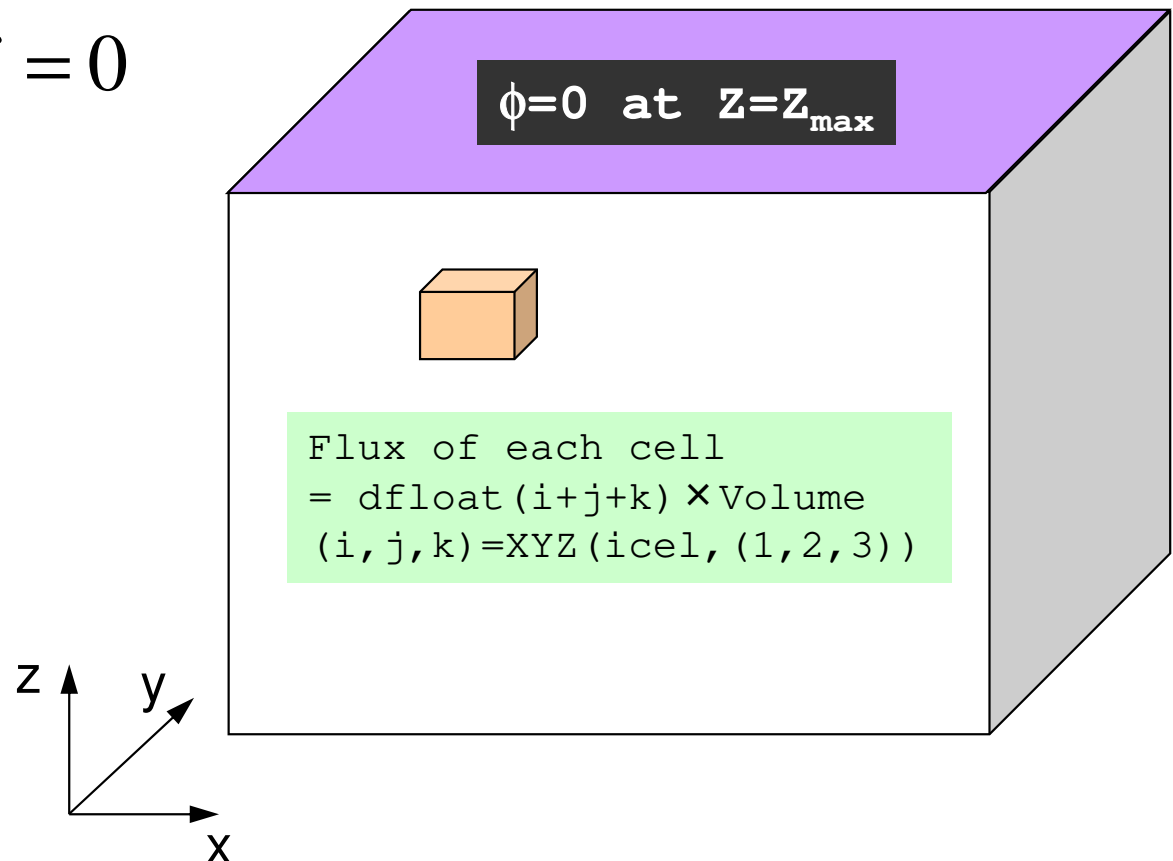
Target Problem: Variables are defined at cell-center's

Poisson Equation/ Poisson's Equation

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

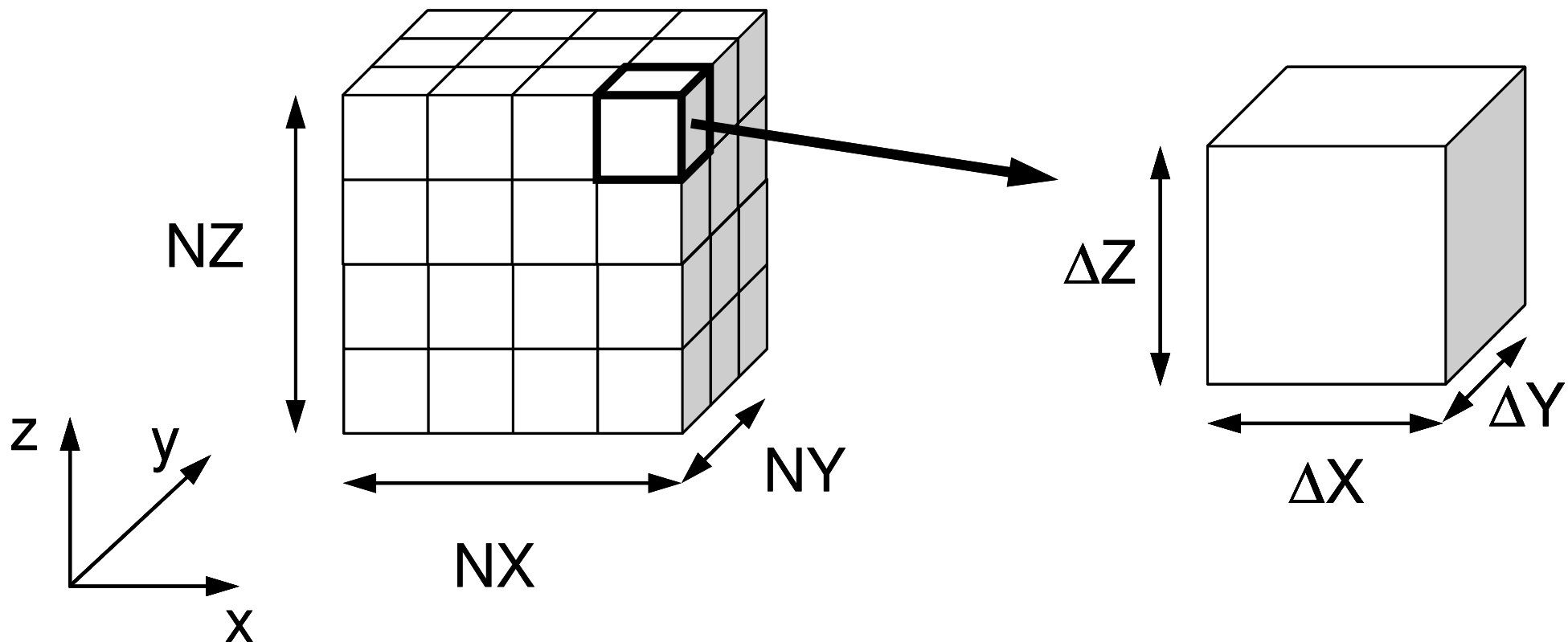
Boundary Conditions (B.C.) etc.

- Volume Flux
- $\phi=0 @ Z=Z_{\max}$



3D Structured Mesh

Internal data structure is “unstructured”



Volume Flux f

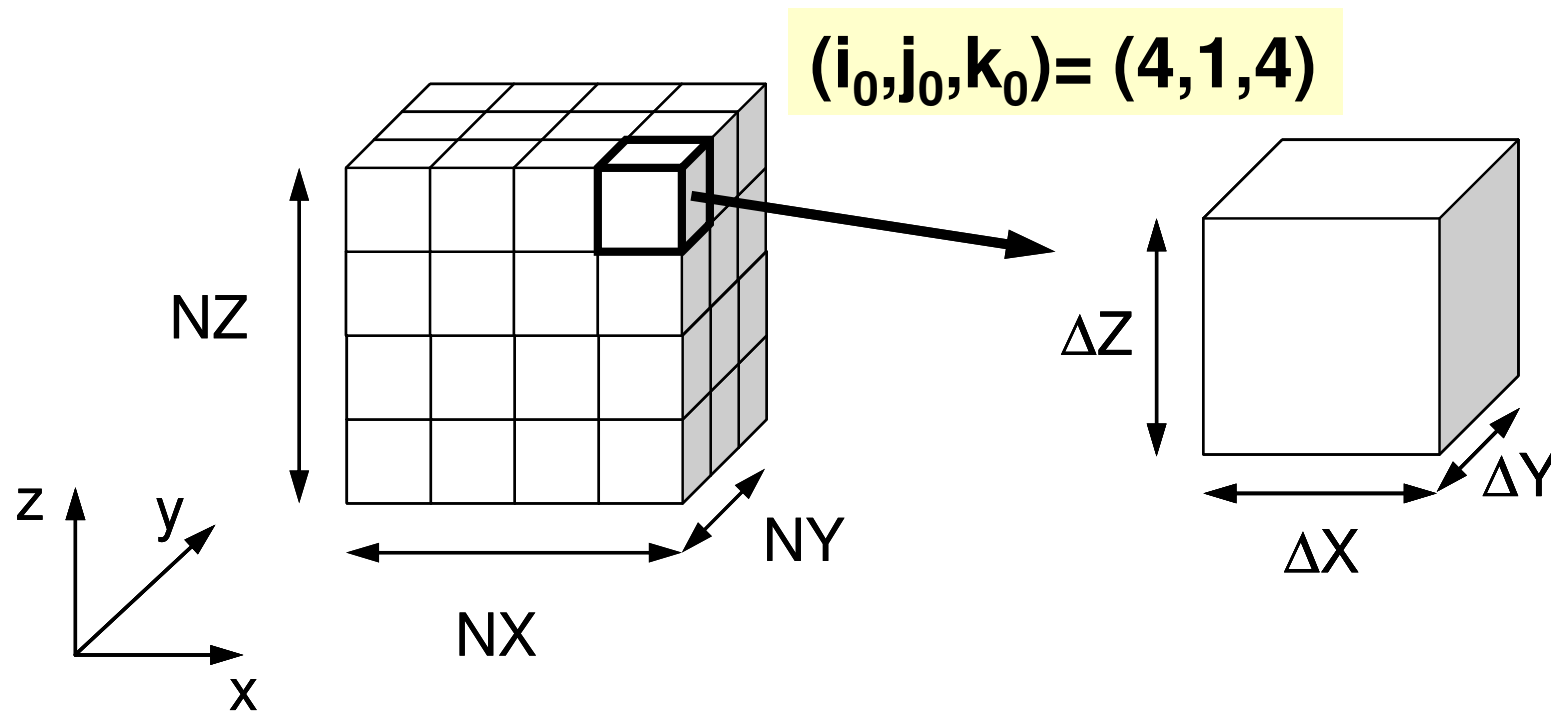
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

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

$$i_0 = XYZ(icel, 1), \quad XYZ(icel, k) (k=1, 2, 3)$$

$$j_0 = XYZ(icel, 2), \quad \text{Index for location of finite-difference}$$

$$k_0 = XYZ(icel, 3) \quad \text{mesh in X-/Y-/Z-axis.}$$

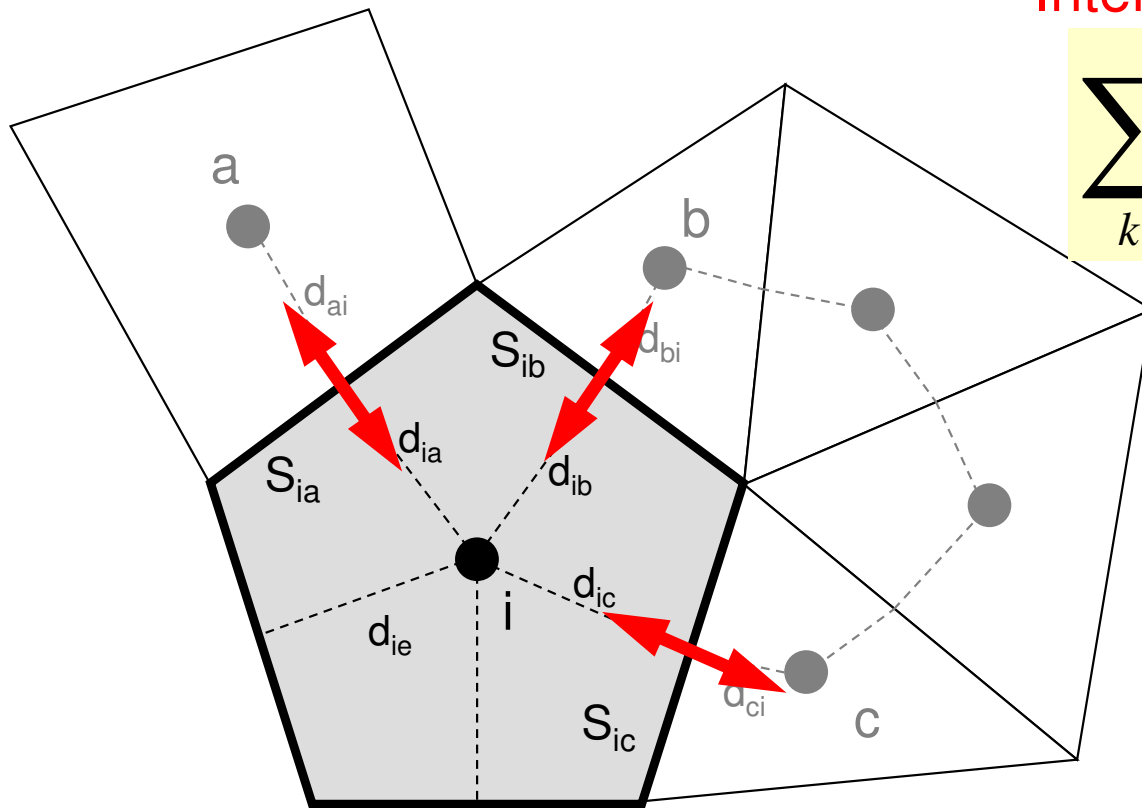


Poisson Equation by Finite Volume Method (FVM)

Conservation of Fluxes through
Surfaces

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

Diffusion:
Interaction with Neighbors



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

Volume Flux

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

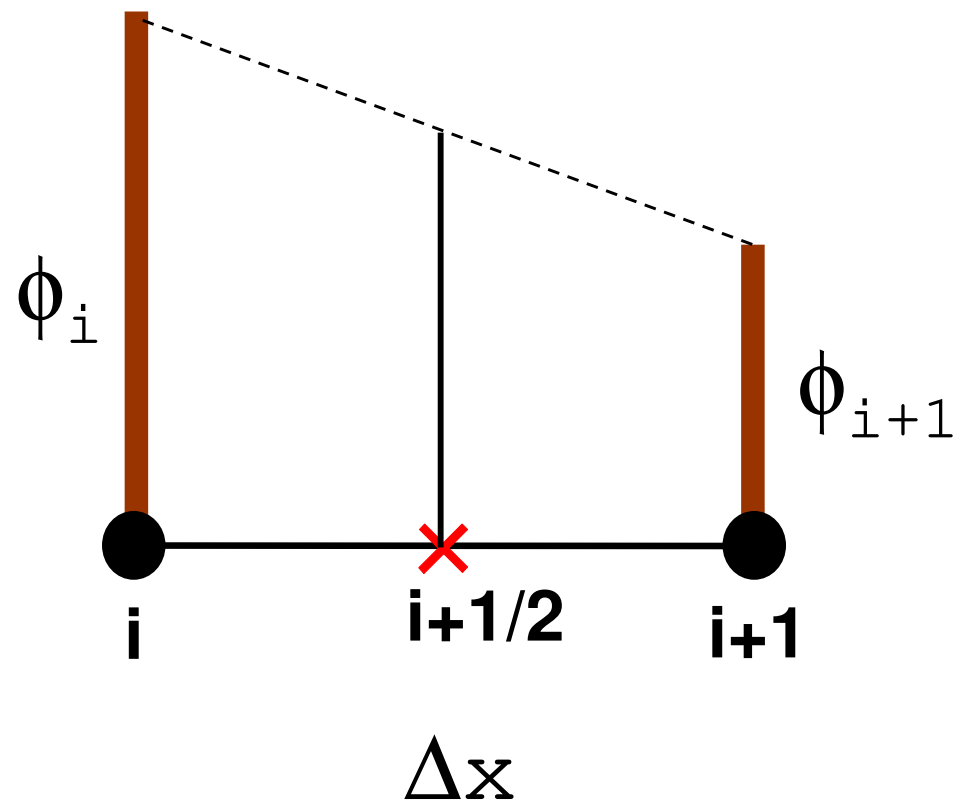
Finite Difference Method (FDM)

(有限)差分法：巨視的微分

macroscopic differentiation

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

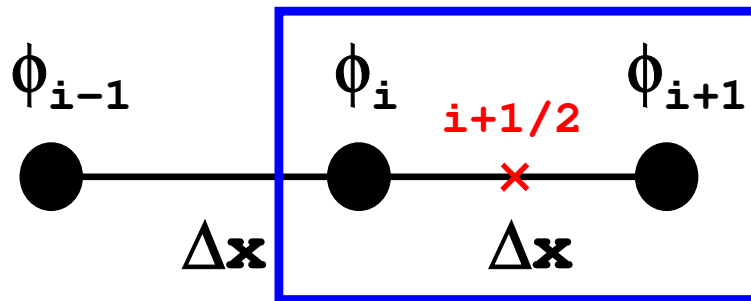
$$\left(\frac{d\phi}{dx}\right)_{i+1/2} = \lim_{\Delta x \rightarrow 0} \frac{\phi_{i+1} - \phi_i}{\Delta x}$$



2nd Order Differentiation in FDM

Taylor Series Expansion

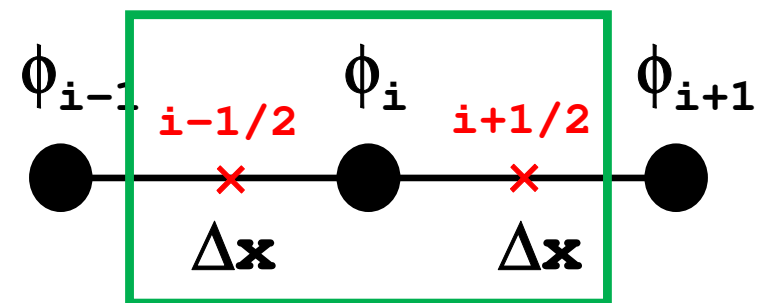
- Approximate Derivative at $\times$ (center of i and $i+1$)



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

$\Delta x \rightarrow 0$: Real Derivative

- 2nd-Order Diff. at i



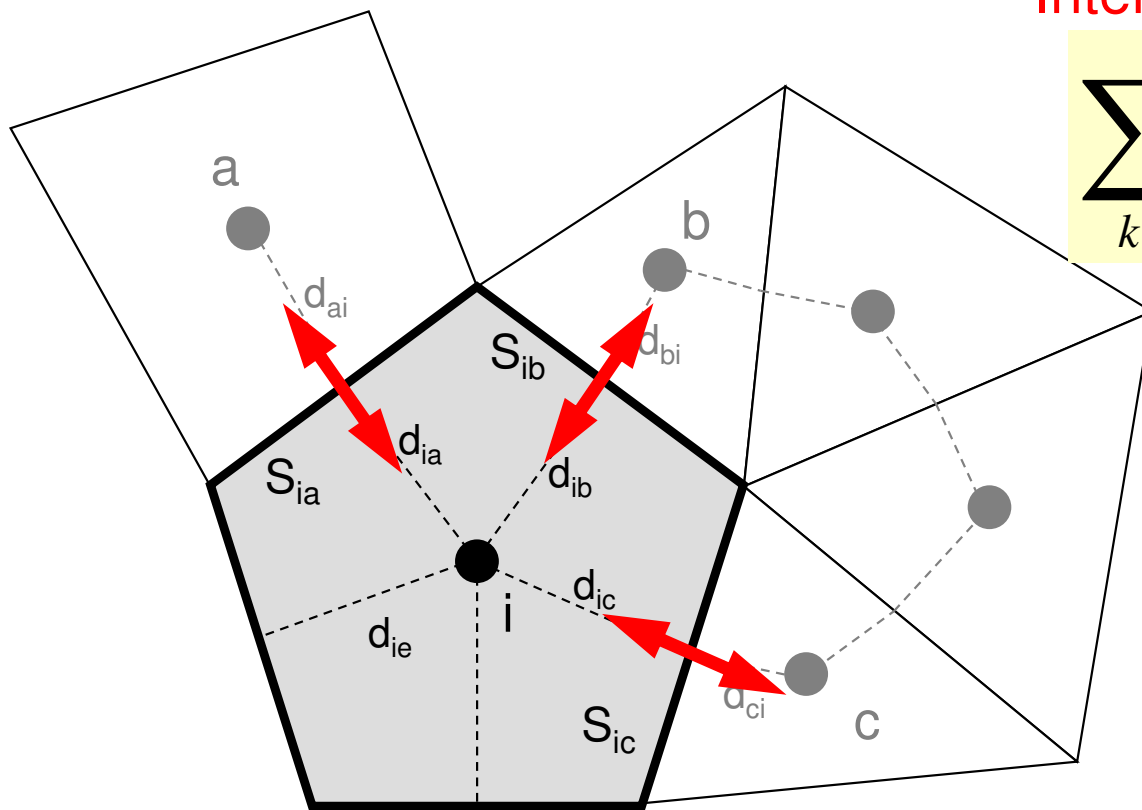
$$\left(\frac{d^2\phi}{dx^2} \right)_i \approx \frac{\left(\frac{d\phi}{dx} \right)_{i+1/2} - \left(\frac{d\phi}{dx} \right)_{i-1/2}}{\Delta x} = \frac{\frac{\phi_{i+1} - \phi_i}{\Delta x} - \frac{\phi_i - \phi_{i-1}}{\Delta x}}{\Delta x} = \frac{\phi_{i+1} - 2\phi_i + \phi_{i-1}}{\Delta x^2}$$

Poisson Equation by Finite Volume Method (FVM)

Conservation of Fluxes through
Surfaces

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

Diffusion:
Interaction with Neighbors

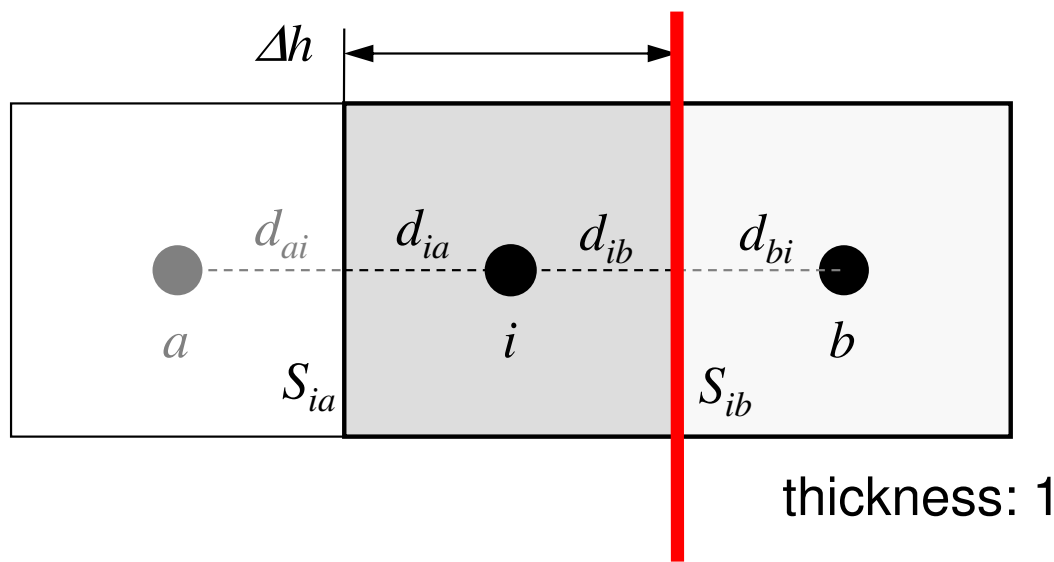


$$\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} (\phi_k - \phi_i) + V_i \dot{Q}_i = 0$$

Volume Flux

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

Comparison with 1D FDM (1/3)



$\Delta h \times \Delta h$ Square Mesh

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

Volume: $V_i = \Delta h^2$

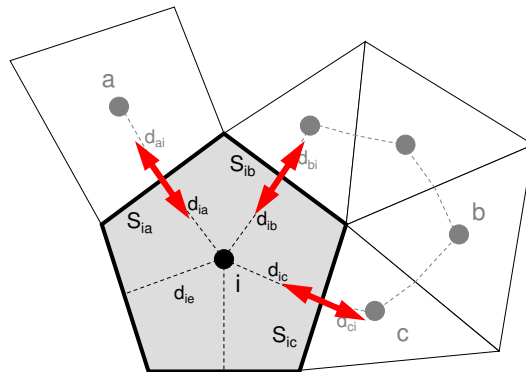
Distance (Ctr.-Suf): $d_{ij} = \Delta h/2$

Flux through this surface: $Q_{S_{ib}}$

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

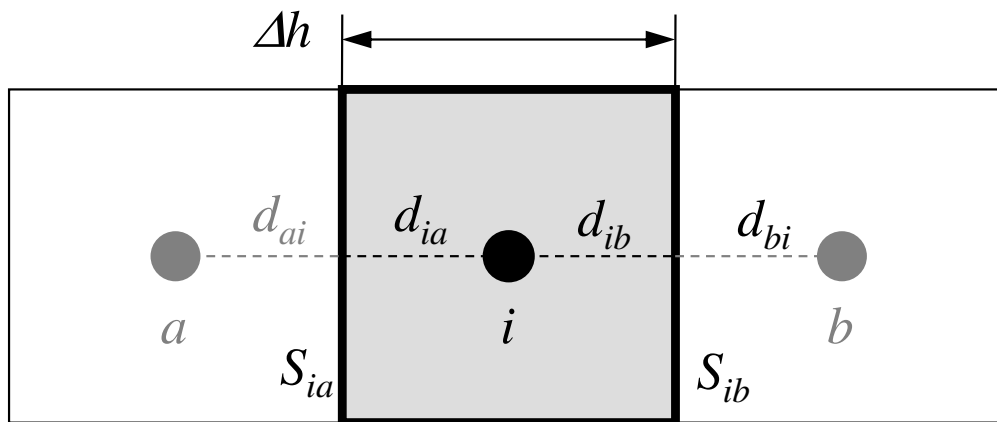
Fourier's Law

Flux through a surface
= - (gradient of potential)



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

Comparison with 1D FDM (2/3)



thickness: 1

$\Delta h \times \Delta h$ Square Mesh

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

Volume: $V_i = \Delta h^2$

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

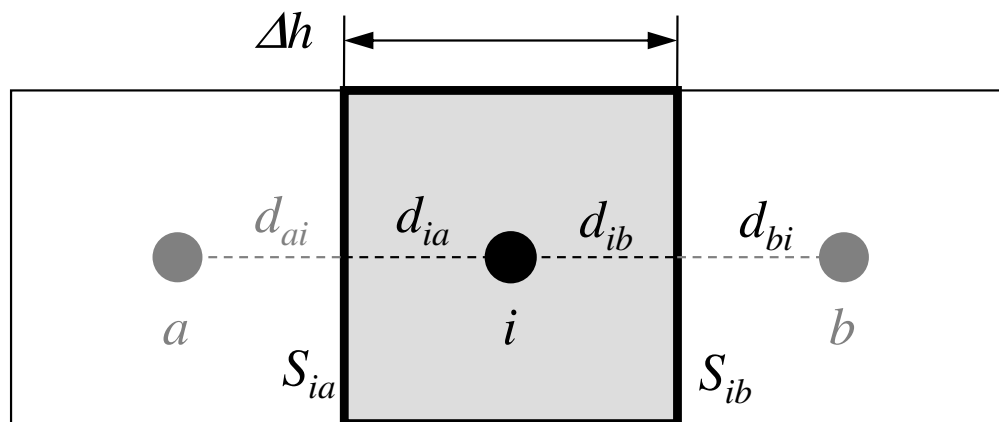
$$\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} (\phi_k - \phi_i) + V_i \dot{Q}_i = 0$$

Divided by V_i :

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

considering this part

Comparison with 1D FDM (3/3)



$\Delta h \times \Delta h$ Square Mesh

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

Volume: $V_i = \Delta h^2$

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

thickness: 1

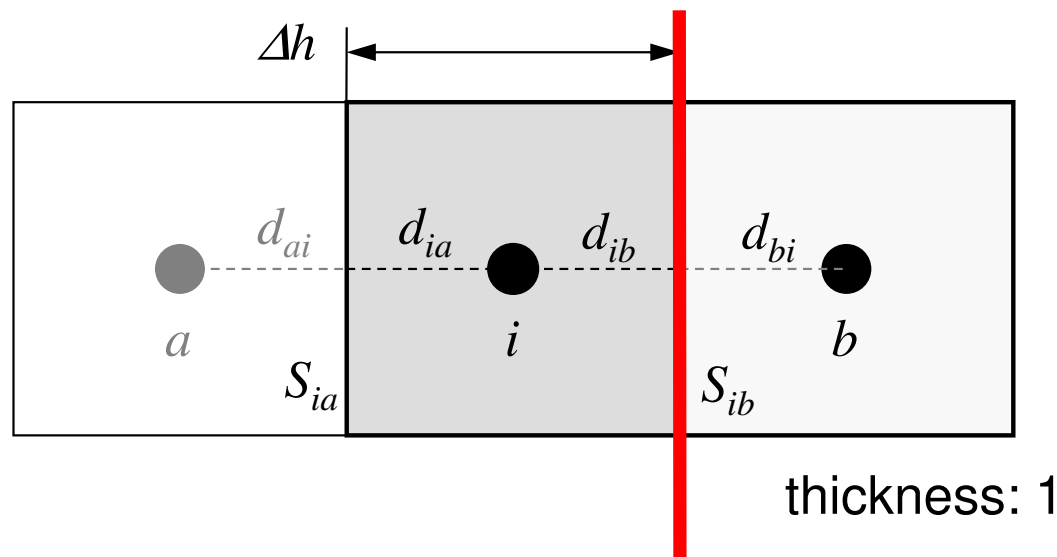
$$\begin{aligned}
 \frac{1}{V_i} \sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} (\phi_k - \phi_i) &= \frac{1}{(\Delta h)^2} \sum_{k=a,b} \frac{\Delta h}{\frac{\Delta h}{2} + \frac{\Delta h}{2}} (\phi_k - \phi_i) \\
 &= \frac{1}{(\Delta h)^2} \sum_{k=a,b} \frac{\Delta h}{\frac{\Delta h}{2} + \frac{\Delta h}{2}}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

for i-th cell
linear equations

Heat Equation (1/3)

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

λ : Thermal Conductivity



$\Delta h \times \Delta h$ Square Mesh

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

Volume: $V_i = \Delta h^2$

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

Heat Flux through this surface: $Q_{S_{ib}}$

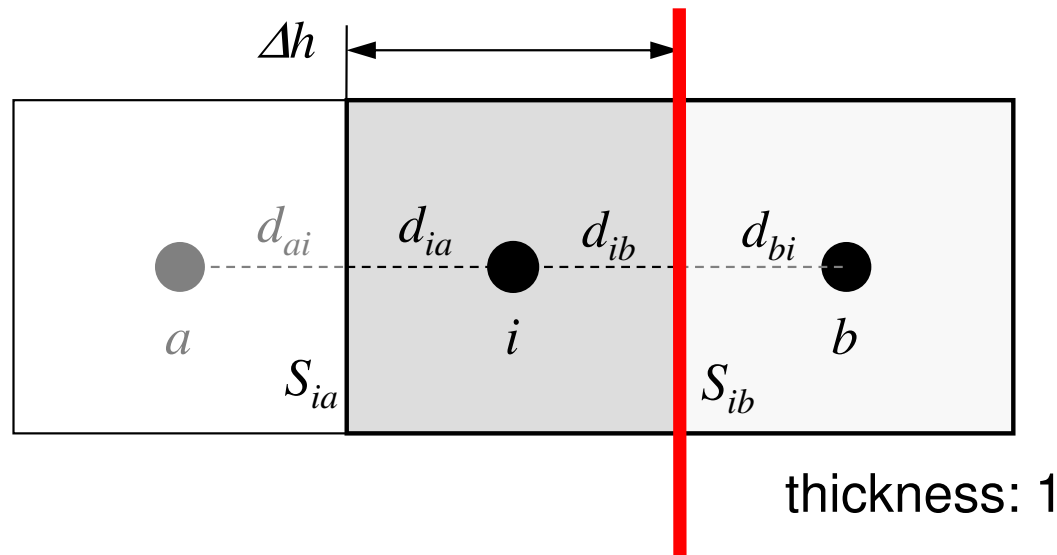
$$Q_{S_{ib}} = -\lambda \frac{T_b - T_i}{\Delta h} \cdot S_{ib}$$

$$\lambda_i = \lambda_b = \lambda$$

Heat Equation (2/3)

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

λ : Thermal Conductivity



$\Delta h \times \Delta h$ Square Mesh

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

Volume: $V_i = \Delta h^2$

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

Heat Flux through this surface: $Q_{S_{ib}}$

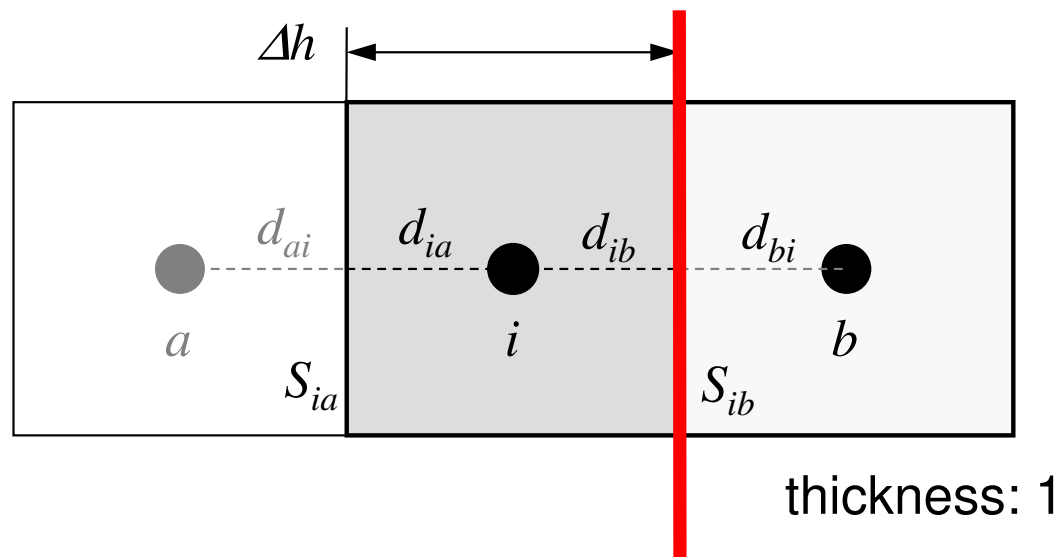
$$Q_{S_{ib}} = -\lambda \frac{T_b - T_i}{\frac{\Delta h}{2} + \frac{\Delta h}{2}} \cdot S_{ib} = -\frac{T_b - T_i}{\left[\left(\frac{\Delta h}{2} \right) / \lambda \right] + \left[\left(\frac{\Delta h}{2} \right) / \lambda \right]} \cdot S_{ib}$$

$$\lambda_i = \lambda_b = \lambda$$

Heat Equation (3/3)

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

λ : Thermal Conductivity



$\Delta h \times \Delta h$ Square Mesh

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

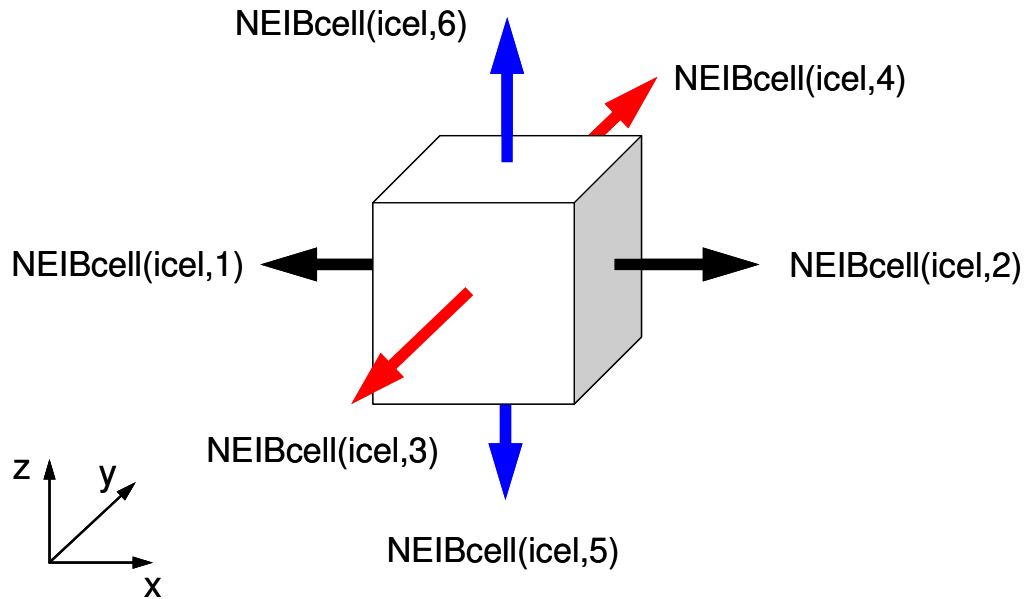
Volume: $V_i = \Delta h^2$

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

Heat Flux through this surface: $Q_{S_{ib}}$

$$Q_{S_{ib}} = - \frac{T_b - T_i}{\left[\left(\frac{\Delta h}{2} \right) / \lambda_i \right] + \left[\left(\frac{\Delta h}{2} \right) / \lambda_b \right]} \cdot S_{ib} \quad \lambda_i \neq \lambda_b$$

in 3D



$$\begin{aligned}
 & \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
 & \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
 & \frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0
 \end{aligned}$$

Linear Equations

$$\begin{aligned} & \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\ & \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\ & \frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0 \end{aligned}$$

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

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

Diagonal

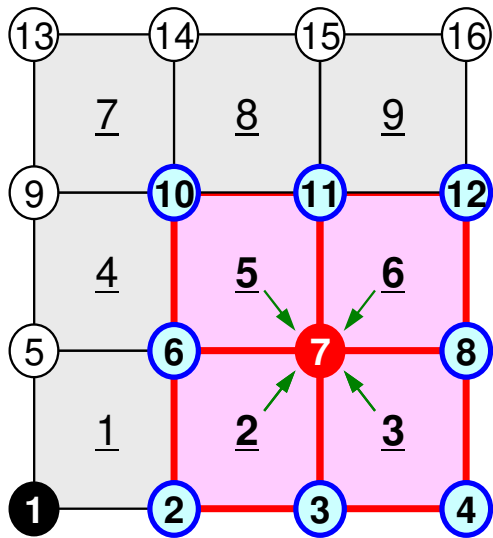
Off-Diagonal



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

Sparse Coef. Matrix in FEM

Many “0’s”



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

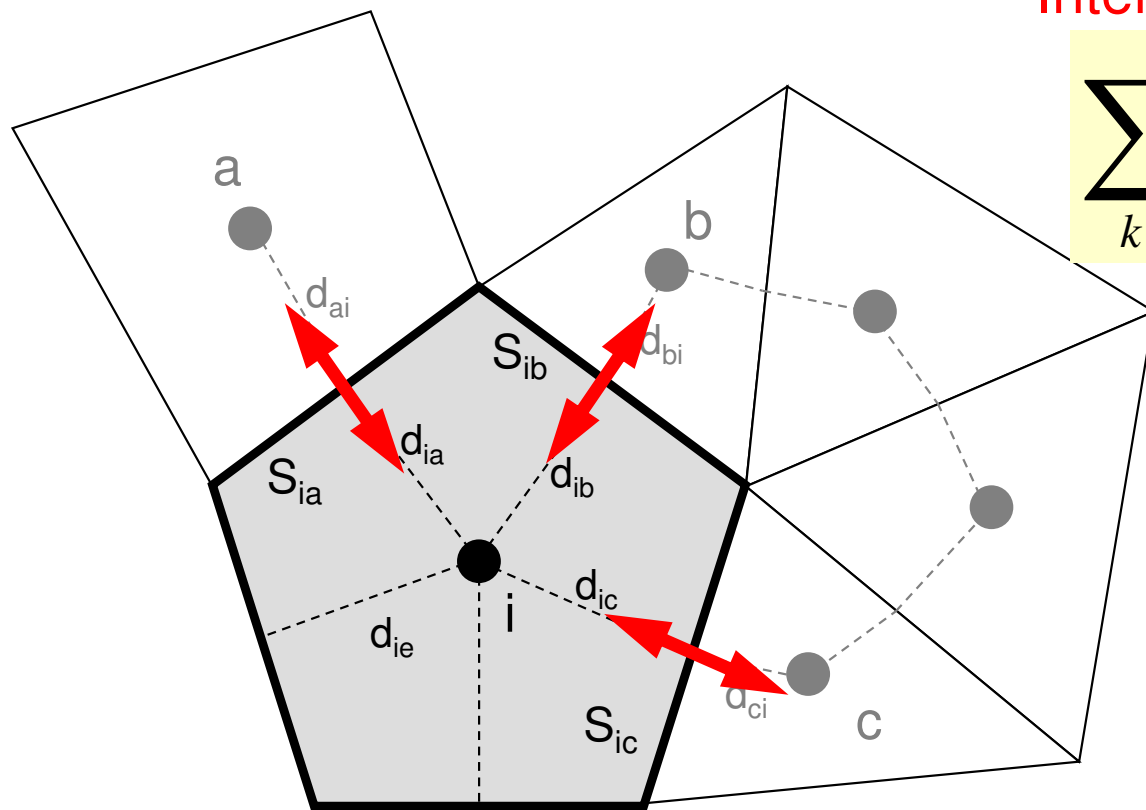
$$\begin{bmatrix} D & X & & & X & X & & & & & & & & & & \\ X & D & X & & X & X & X & & & & & & & & & \\ & X & D & X & & X & X & X & & & & & & & & \\ & & X & D & & & X & X & & & & & & & & \\ X & X & & & D & X & & & X & X & & & & & & \\ X & X & X & & X & D & X & & X & X & X & & & & & \\ X & X & X & X &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

Coefficient Matrices for FVM are sparse

Only neighboring cells are considered

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

Diffusion:
Interaction with Neighbors



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

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. $A(i,j)$) is not efficient for sparse matrices
 - $A(i,j)$ is suitable for dense matrices

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

- Number of non-zero off-diagonal components is $O(100)$ in FVM (only 6 in this case)
 - If number of unknowns is 10^8 :
 - $A(i,j)$: $O(10^{16})$ words $\sim O(10^{17})$ bytes for DP: 100PB (400 x Odyssey)
 - Actual Non-zero Comp.: $O(10^9)$ words: 10GB (Odyssey: 32GB/node)
- Only (really) non-zero off-diag. components should be stored on memory

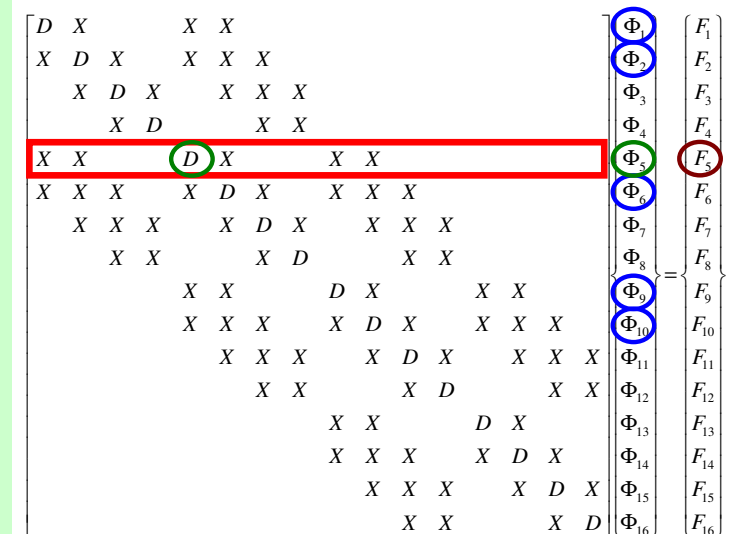
Mat-Vec. Multiplication for Sparse Matrix

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) Non-Zero Off-Diagonal Components (Corresponding Column ID)
 (INT, k=1, index(N))
AMat (k) Non-Zero 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

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

$$\{Y\} = [A] \{X\}$$

```
do j= 1, N
  Y(j)= 0. d0
  do i= 1, N
    Y(j)= Y(j) + A(i, j)*X(i)
  enddo
enddo
```

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): Fortran

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

N= 8

Diagonal Components

Diag (1) = 1.1

Diag (2) = 3.6

Diag (3) = 5.7

Diag (4) = 9.8

Diag (5) = 11.5

Diag (6) = 12.4

Diag (7) = 23.1

Diag (8) = 51.3

Compressed Row Storage (CRS)

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

Compressed Row Storage (CRS)

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

NPLU= 25
(=index(N))

$\text{index}(i-1) + 1^{\text{th}} \sim \text{index}(i)^{\text{th}}$
Non-Zero Off-Diag. Components corresponding to i -th row

Compressed Row Storage (CRS)

		# Non-Zero Off-Diag.	
1	1.1 ①	2	$\text{index}(0) = 0$
2	3.6 ②	4	$\text{index}(1) = 2$
3	5.7 ③	2	$\text{index}(2) = 6$
4	9.8 ④	3	$\text{index}(3) = 8$
5	11.5 ⑤	4	$\text{index}(4) = 11$
6	12.4 ⑥	2	$\text{index}(5) = 15$
7	23.1 ⑦	4	$\text{index}(6) = 17$
8	51.3 ⑧	4	$\text{index}(7) = 21$
			$\text{index}(8) = 25$

NPLU= 25
(=index(N))

$\text{index}(i-1) + 1^{\text{th}} \sim \text{index}(i)^{\text{th}}$
Non-Zero Off-Diag. Components corresponding to i -th row

Compressed Row Storage (CRS)

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

Example:

`item( 7) = 5, AMAT( 7) = 1.5`

`item(19) = 3, AMAT(19) = 2.5`

Compressed Row Storage (CRS)

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

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) Non-Zero Off-Diagonal Components (Corresponding Column ID) (INT, k=1, index(N))
AMat (k) Non-Zero Off-Diagonal Components (Value) (REAL, k=1, index(N))

$$\{Y\} = [A] \{X\}$$

```

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    Y(i)= Y(i) + AMAT(k)*X(item(k))
  enddo
enddo

```

- 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 $\mathbf{Ax}=\mathbf{b}$ 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, FVM, FDM, DEM, MD (solid), BEM w/FMM

Direct Method

直接法

- Gaussian Elimination/LU Factorization
 - compute A^{-1} directly (or equivalent operations)

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

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

A **x** **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 $\mathbf{x}^{(0)}$, iterative method obtains the final converged solutions by iterations

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

Iterative Method

反復法

- Stationary Method

- Only $\mathbf{x}$ (solution vector) changes during iterations.
- SOR, Gauss-Seidel, Jacobi
- Generally slow, impractical

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

- 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 Real-World Applications in Scientific Computing (FEM, FVM, FDM etc)

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

非定常法・クリロフ部分空間法

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

Compute $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_k$ by the following iterative procedures:

$$\mathbf{x}_k = \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}_{k-1}$$

$$= (\mathbf{b} - \mathbf{Ax}_{k-1}) + \mathbf{x}_{k-1}$$

$$= \mathbf{r}_{k-1} + \mathbf{x}_{k-1}$$

where $\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$: residual



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

$$\mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k = \mathbf{b} - \mathbf{A}(\mathbf{r}_{k-1} + \mathbf{x}_{k-1})$$

$$= (\mathbf{b} - \mathbf{Ax}_{k-1}) - \mathbf{Ar}_{k-1} = \mathbf{r}_{k-1} - \mathbf{Ar}_{k-1} = (\mathbf{I} - \mathbf{A})\mathbf{r}_{k-1}$$

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

非定常法・クリロフ部分空間法

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

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



$\mathbf{z}_k$ is a vector which belongs to k^{th} Krylov Subspace (クリロフ部分空間), approximate solution vector $\mathbf{x}_k$ is derived by the Krylov Subspace:

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

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
 - $\{x\}^T[A]\{x\} > 0$ for arbitrary $\{x\}$
 - All of diagonal components, eigenvalues and leading principal minors > 0 (主小行列式・首座行列式)
 - Matrices of Galerkin-based FEM & FVM: heat conduction, Poisson, static linear elastic problems
- Algorithm
 - “Steepest Descent Method”
 - $\mathbf{x}^{(i)} = \mathbf{x}^{(i-1)} + \alpha_i \mathbf{p}^{(i)}$
 - $\mathbf{x}^{(i)}$: solution, $\mathbf{p}^{(i)}$: search direction, α_i : coefficient
 - Solution $\{x\}$ minimizes $\{x-y\}^T[A]\{x-y\}$, 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  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
    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
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    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  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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
 - $\{y\} = a\{x\} + \{y\}$

$x^{(i)}$: Vector

α_i : Scalar

Procedures of Conjugate Gradient

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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 ($Ay=b$)

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

$$\begin{aligned} (x - y)^T [A](x - y) &= (x, Ax) - (y, Ax) - (x, Ay) + (y, Ay) \\ &= (x, Ax) - 2(x, Ay) + (y, Ay) = (x, Ax) - 2(x, b) + \underline{(y, b)} \quad \text{Const.} \end{aligned}$$

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

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

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

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

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

$$\begin{aligned} f(x+h) &= \frac{1}{2}(x+h, A(x+h)) - (x+h, b) \\ &= \frac{1}{2}(x+h, Ax) + \frac{1}{2}(x+h, Ah) - (x, b) - (h, b) \\ &= \frac{1}{2}(x, Ax) + \frac{1}{2}(h, Ax) + \frac{1}{2}(x, Ah) + \frac{1}{2}(h, Ah) - (x, b) - (h, b) \\ &= \frac{1}{2}(x, Ax) - (x, b) + (h, Ax) - (h, b) + \frac{1}{2}(h, Ah) \\ &= f(x) + (h, Ax-b) + \frac{1}{2}(h, Ah) \end{aligned}$$

Derivation of CG Algorithm (2/5)

CG method minimizes $f(x)$ at each iteration.

Start from initial solution $x^{(0)}$ and assume that approximate solution: $x^{(k)}$, and search direction vector $p^{(k)}$ is defined at k -th iter.

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

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

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

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

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

Derivation of CG Algorithm (3/5)

Residual vector at $(k+1)$ -th iteration:

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

$$r^{(k+1)} = b - Ax^{(k+1)}, r^{(k)} = b - Ax^{(k)}$$

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

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

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

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:

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

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

$$\therefore \alpha_k = \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})}$$

Thus, following relation is obtained:

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

Derivation of CG Algorithm (5/5)

$$\begin{aligned} (p^{(k+1)}, Ap^{(k)}) &= (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(k)}) = (r^{(k+1)}, Ap^{(k)}) + \beta_k (p^{(k)}, Ap^{(k)}) = 0 \\ \Rightarrow \beta_k &= \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})} \quad \textcolor{red}{(4)} \end{aligned}$$

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

$p^{(k)}$: search direction vector, “gradient” vector

```

Compute  $p^{(0)} = r^{(0)} = b - [A]x^{(0)}$ 
for i= 1, 2, ...
    calc.  $\alpha_{i-1}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_{i-1}p^{(i-1)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_{i-1}[A]p^{(i-1)}$ 

    check convergence  $|r|$ 
    (if not converged)
    calc.  $\beta_{i-1}$ 
     $p^{(i)} = r^{(i)} + \beta_{i-1}p^{(i-1)}$ 
end
  
```

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

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

Properties of CG Algorithm

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

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

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

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

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

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) \end{aligned}$$

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

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

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

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

Proof (2/3)

Mathematical Induction

数学的帰納法

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) \end{aligned} \quad (*)$$

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

$$\begin{aligned} \text{if } i < k \quad (r^{(k+1)}, r^{(i)}) &= (r^{(i)}, r^{(k+1)}) \stackrel{(2)}{=} (r^{(i)}, r^{(k)} - \alpha_k Ap^{(k)}) \\ &\stackrel{(*)}{=} -\alpha_k (r^{(i)}, Ap^{(k)}) \stackrel{(3)}{=} -\alpha_k (p^{(i)} - \beta_{i-1} p^{(i-1)}, Ap^{(k)}) \\ &= -\alpha_k (p^{(i)}, Ap^{(k)}) + \alpha_k \beta_{i-1} (p^{(i-1)}, Ap^{(k)}) \stackrel{(*)}{=} 0 \end{aligned}$$

$$\begin{aligned} \text{if } i = k \quad (r^{(k+1)}, r^{(k)}) &\stackrel{(2)}{=} (r^{(k)}, r^{(k)}) - (r^{(k)}, \alpha_k Ap^{(k)}) \\ &\stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k Ap^{(k)}) \\ (1) \quad \alpha_k &= \frac{(p^{(k)}, r^{(k)})}{(p^{(k)}, Ap^{(k)})} \quad \stackrel{(*)}{=} (r^{(k)}, r^{(k)}) - \alpha_k (p^{(k)}, Ap^{(k)}) \stackrel{(1)}{=} (r^{(k)}, r^{(k)}) - (p^{(k)}, r^{(k)}) \\ (2) \quad r^{(k+1)} &= r^{(k)} - \alpha_k Ap^{(k)} \quad \stackrel{(3)}{=} (r^{(k)}, r^{(k)}) - (\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)}) \\ (3) \quad p^{(k+1)} &= r^{(k+1)} + \beta_k p^{(k)} \quad = -\beta_{k-1} (p^{(k-1)}, r^{(k)}) \stackrel{(2)}{=} -\beta_{k-1} (p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} Ap^{(k-1)}) \\ (4) \quad \beta_k &= \frac{-(r^{(k+1)}, Ap^{(k)})}{(p^{(k)}, Ap^{(k)})} \quad = -\beta_{k-1} \{ (p^{(k-1)}, r^{(k-1)}) - \alpha_{k-1} (p^{(k-1)}, Ap^{(k-1)}) \} \stackrel{(1)}{=} 0 \end{aligned}$$

Proof (3/3)

Mathematical Induction 数学的帰納法

$$\begin{aligned} (r^{(i)}, r^{(j)}) &= 0 \quad (i \neq j) \\ (p^{(i)}, Ap^{(j)}) &= 0 \quad (i \neq j) \end{aligned} \quad (*)$$

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

$$\begin{aligned} \text{if } i < k \quad (p^{(k+1)}, Ap^{(i)}) &\stackrel{(3)}{=} (r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}) \\ &\stackrel{(*)}{=} (r^{(k+1)}, Ap^{(i)}) \\ &\stackrel{(2)}{=} \frac{1}{\alpha_i} (r^{(k+1)}, r^{(i)} - r^{(i-1)}) = 0 \end{aligned}$$

$$\begin{aligned} \text{if } i = k \quad (p^{(k+1)}, Ap^{(k)}) &\stackrel{(3)}{=} (r^{(k+1)}, Ap^{(k)}) + \beta_k (p^{(k)}, Ap^{(k)}) \\ &\stackrel{(4)}{=} 0 \end{aligned}$$

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

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

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

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

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

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

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

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

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

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

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

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

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

$$\alpha_k, \beta_k$$

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

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

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

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

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

Procedures of Conjugate Gradient

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
     $z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} z^{(i-1)}$ 
    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

```

$\mathbf{x}^{(i)}$: Vector

α_i : Scalar

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

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

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 (前处理)
 - M transforms $Ax=b$ into $A'x=b'$ where $A'=M^{-1}A$, $b'=M^{-1}b$
 - If $M \sim A$, $M^{-1}A$ 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／右前处理)

Preconditioned CG Solver

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    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

```

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

$$[A'] x' = b'$$

$$[A'] = [M_1]^{-1} [A] [M_2]^{-1}$$

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

$$p' \Rightarrow [M_2] p, \quad r' \Rightarrow [M_1]^{-1} r$$

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

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

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

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

$$\beta'_{i-1} = \frac{([M]^{-1} r^{(i-1)}, r^{(i-1)})}{([M]^{-1} r^{(i-2)}, r^{(i-2)})}$$

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

In CG method, preconditioner usually satisfies $[M_2] = [M_1]^T$, such as Incomplete Cholesky/Incomplete Modified Cholesky Factorizations. In this problem, let us define $[M_1]$ and $[M_2]$ as follows:

$$[M_1] = [X]^T, [M_2] = [X], [M] = [M_1][M_2]$$

$$[A']x' = b'$$

$$[A'] = [M_1]^{-1}[A][M_2]^{-1} = [[X]^T]^{-1}[A][X]^{-1} = [X]^{-T}[A][X]^{-1}$$

$$x' = [X]x, \quad b' = [X]^{-T}b, \quad r' = [X]^{-T}r$$

$$\begin{aligned} \alpha'_{i-1} &= \frac{(r'^{(i-1)}, r'^{(i-1)})}{(p'^{(i-1)}, A' p'^{(i-1)})} = \frac{([X]^{-T} r^{(i-1)}, [X]^{-T} r^{(i-1)})}{([X] p^{(i-1)}, [X]^{-T} [A][X]^{-1} [X] p^{(i-1)})} \\ &= \frac{\left(([X]^{-T} r^{(i-1)})^T, [X]^{-T} r^{(i-1)} \right)}{\left(([X] p^{(i-1)})^T, [X]^{-T} [A] p^{(i-1)} \right)} = \frac{\left((r^{(i-1)})^T [X]^{-1}, [X^T]^{-1} r^{(i-1)} \right)}{\left((p^{(i-1)})^T [X]^T, [X]^{-T} [A] p^{(i-1)} \right)} \\ &= \frac{\left(r^{(i-1)}, [[X^T][X]]^{-1} r^{(i-1)} \right)}{(p^{(i-1)}, [A] p^{(i-1)})} = \frac{(r^{(i-1)}, [M]^{-1} r^{(i-1)})}{(p^{(i-1)}, [A] p^{(i-1)})} = \frac{(r^{(i-1)}, z^{(i-1)})}{(p^{(i-1)}, [A] p^{(i-1)})} \end{aligned}$$

$$\begin{aligned}
\beta'_{i-1} &= \frac{\left(r^{(i-1)}, r^{(i-1)} \right)}{\left(r^{(i-2)}, r^{(i-2)} \right)} = \frac{\left([\mathbf{X}]^{-T} r^{(i-1)}, [\mathbf{X}]^{-T} r^{(i-1)} \right)}{\left([\mathbf{X}]^{-T} r^{(i-2)}, [\mathbf{X}]^{-T} r^{(i-2)} \right)} \\
&= \frac{\left(\left([\mathbf{X}]^{-T} r^{(i-1)} \right)^T, [\mathbf{X}]^{-T} r^{(i-1)} \right)}{\left(\left([\mathbf{X}]^{-T} r^{(i-2)} \right)^T, [\mathbf{X}]^{-T} r^{(i-2)} \right)} = \frac{\left(\left(r^{(i-1)} \right)^T [\mathbf{X}]^{-1}, [\mathbf{X}^T]^{-1} r^{(i-1)} \right)}{\left(\left(r^{(i-2)} \right)^T [\mathbf{X}]^{-1}, [\mathbf{X}^T]^{-1} r^{(i-2)} \right)} \\
&= \frac{\left(r^{(i-1)}, \left[[\mathbf{X}^T] [\mathbf{X}] \right]^{-1} r^{(i-1)} \right)}{\left(r^{(i-2)}, \left[[\mathbf{X}^T] [\mathbf{X}] \right]^{-1} r^{(i-2)} \right)} = \frac{\left(r^{(i-1)}, [\mathbf{M}]^{-1} r^{(i-1)} \right)}{\left(r^{(i-2)}, [\mathbf{M}]^{-1} r^{(i-2)} \right)} = \frac{\left(r^{(i-1)}, z^{(i-1)} \right)}{\left(r^{(i-2)}, z^{(i-2)} \right)}
\end{aligned}$$

Preconditioned Conjugate Gradient Method (PCG)

```

Compute  $r^{(0)} = b - [A]x^{(0)}$ 
for  $i = 1, 2, \dots$ 
    solve  $[M]z^{(i-1)} = r^{(i-1)}$ 
     $\rho_{i-1} = r^{(i-1)} \cdot z^{(i-1)}$ 
    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
     $q^{(i)} = [A]p^{(i)}$ 
     $\alpha_i = \rho_{i-1} / p^{(i)} \cdot q^{(i)}$ 
     $x^{(i)} = x^{(i-1)} + \alpha_i p^{(i)}$ 
     $r^{(i)} = r^{(i-1)} - \alpha_i q^{(i)}$ 
    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

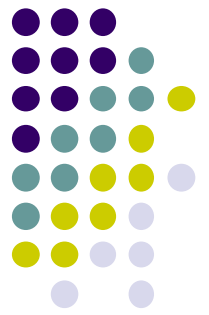
$$[M] = \begin{bmatrix} D_1 & 0 & \dots & 0 & 0 \\ 0 & D_2 & & 0 & 0 \\ \dots & & \dots & & \dots \\ 0 & 0 & & D_{N-1} & 0 \\ 0 & 0 & \dots & 0 & D_N \end{bmatrix}$$

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

ILU(0), IC(0)

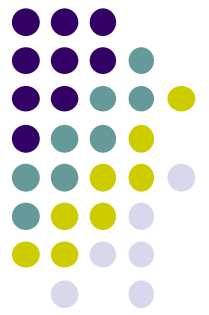
- Widely used Preconditioners for Sparse Matrices
 - Incomplete LU Factorization
 - Incomplete Cholesky Factorization (for Symmetric Matrices)
- Incomplete Direct Method
 - Even if original matrix is sparse, inverse matrix is not necessarily sparse.
 - **fill-in**
 - ILU(0)/IC(0) without fill-in have same non-zero pattern with the original (sparse) matrices

Full LU Factorization (or LU Decomposition)



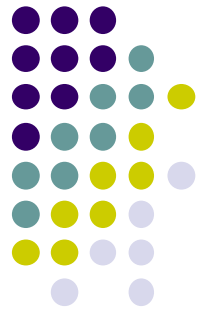
- Direct Method
 - A^{-1} is calculated directly
 - A^{-1} can be saved
 - Fill-in's
- LU factorization

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



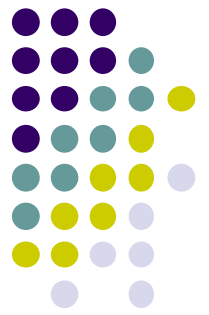
[A] is decomposed to the following form of [L][U]:

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

$$\mathbf{A} = \mathbf{L}\mathbf{U}$$

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:

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$

$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$

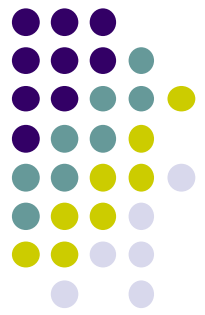
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

Matrix form:

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

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

Solving $Ax=b$ by LU Factorization



1 $A = LU$ Compute $[L]$ and $[U]$

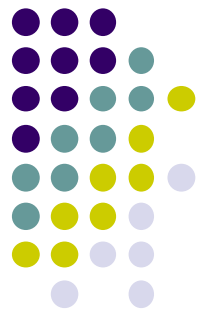
2 $Ly = b$ Solve $Ly=b$

3 $Ux = y$ Solve $Ux=y$

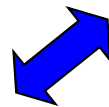
This x is solution of $Ax = b$

$$\therefore Ax = LUx = Ly = b$$

Solving $\mathbf{Ly}=\mathbf{b}$: Forward Substitution



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



$$y_1 = b_1$$

$$l_{21}y_1 + y_2 = b_2$$

$$\vdots$$

$$l_{n1}y_1 + l_{n2}y_2 + \cdots + y_n = b_n$$



$$y_1 = b_1$$

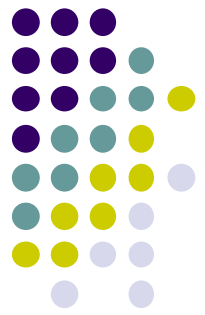
$$y_2 = b_2 - l_{21}y_1$$

$$\vdots$$

$$y_n = b_n - l_{n1}y_1 - l_{n2}y_2 \cdots - l_{n,n-1}y_{n-1}$$

$$= b_n - \sum_{i=1}^{n-1} l_{ni}y_i$$

Solving $Ux=y$: Backward Substitution



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

$$\begin{aligned} u_{nn}x_n &= y_n \\ u_{n-1,n-1}x_{n-1} + u_{n-1,n}x_n &= y_{n-1} \\ \vdots & \\ u_{11}x_1 + u_{12}x_2 + \cdots + u_{1n}x_n &= y_1 \end{aligned}$$

$$\begin{aligned} x_n &= y_n / u_{nn} \\ x_{n-1} &= (y_{n-1} - u_{n-1,n}x_n) / u_{n-1,n-1} \\ \vdots & \\ x_1 &= \left(y_1 - \sum_{j=2}^n u_{1j}x_j \right) / u_{11} \end{aligned}$$

How to calculate LU Factorization/Decomposition



①

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

② ④

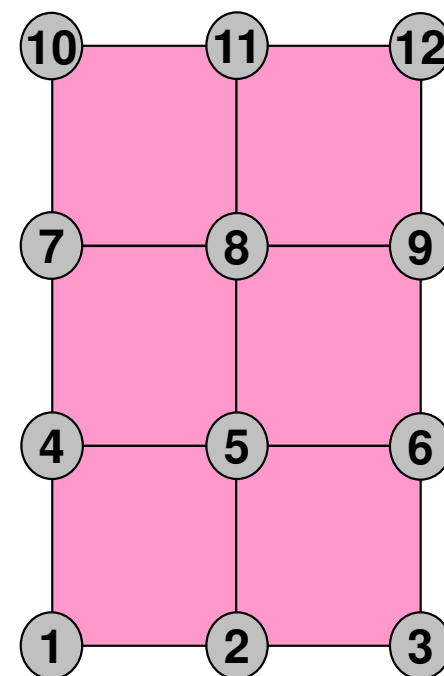
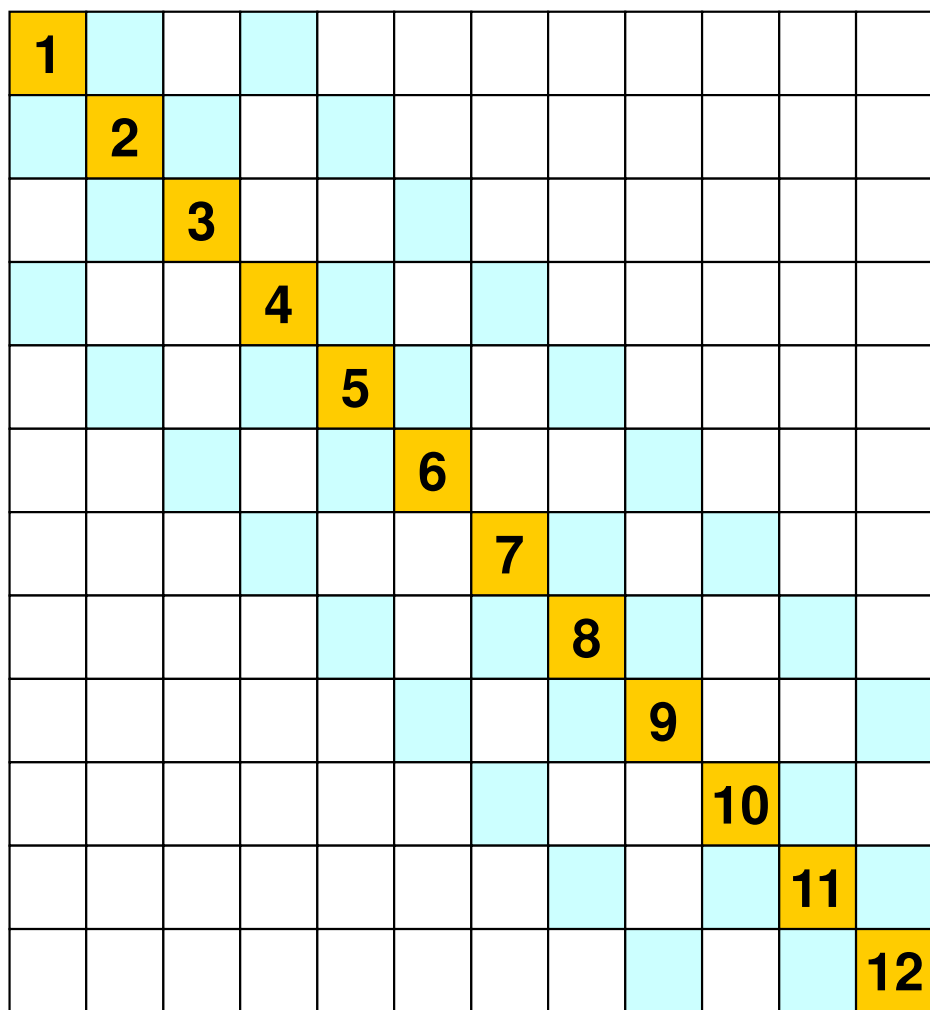
① $\Rightarrow a_{11} = u_{11}, a_{12} = u_{12}, \dots, a_{1n} = u_{1n} \Rightarrow u_{11}, u_{12}, \dots, u_{1n}$

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

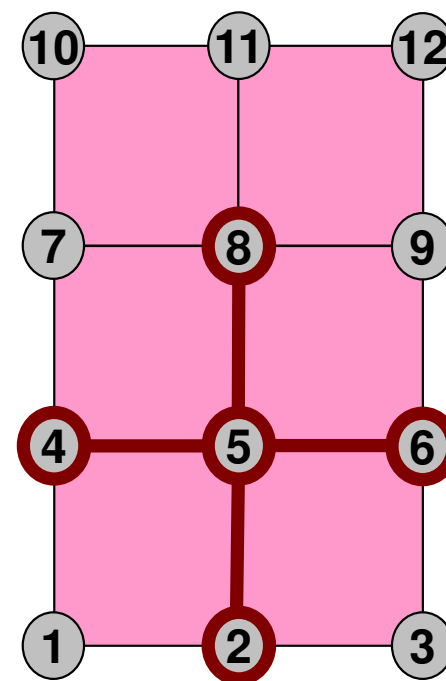
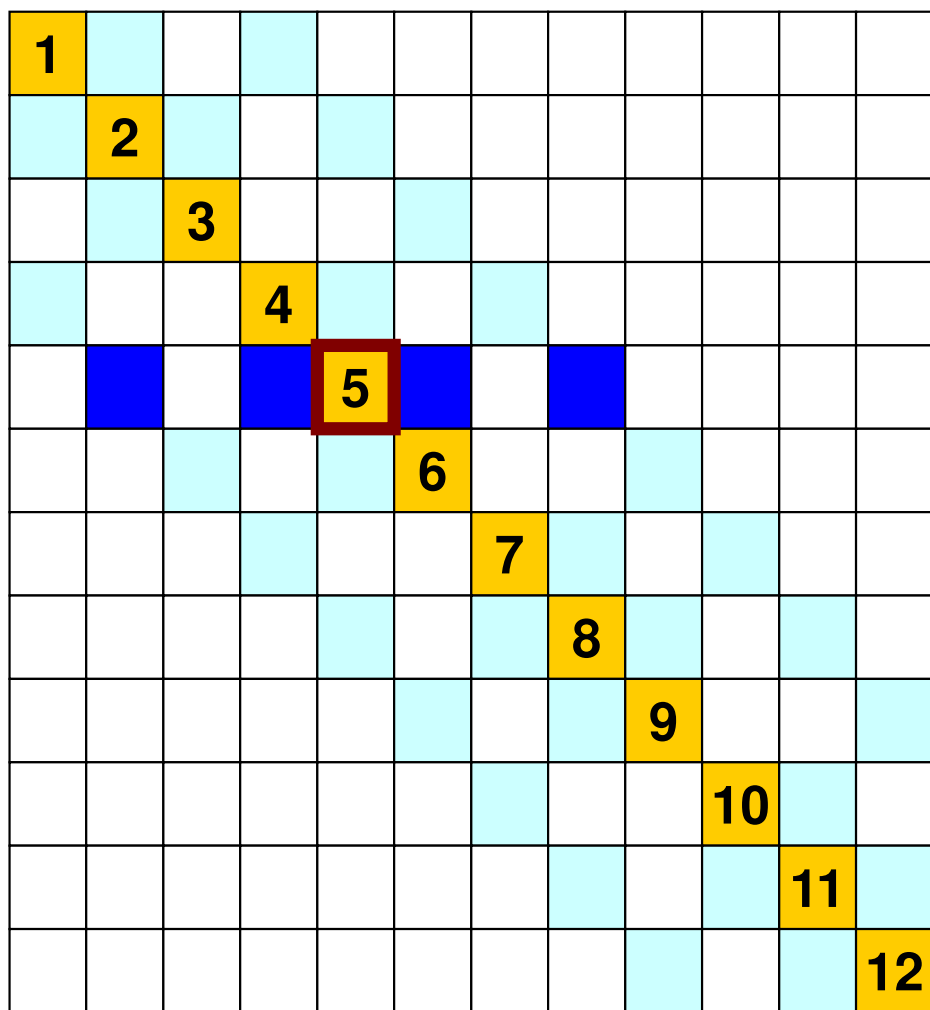
③ $\Rightarrow a_{22} = l_{21}u_{12} + u_{22}, \dots, a_{2n} = l_{21}u_{1n} + u_{2n} \Rightarrow u_{22}, u_{23}, \dots, u_{2n}$

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

Example: 5-Point Stencil

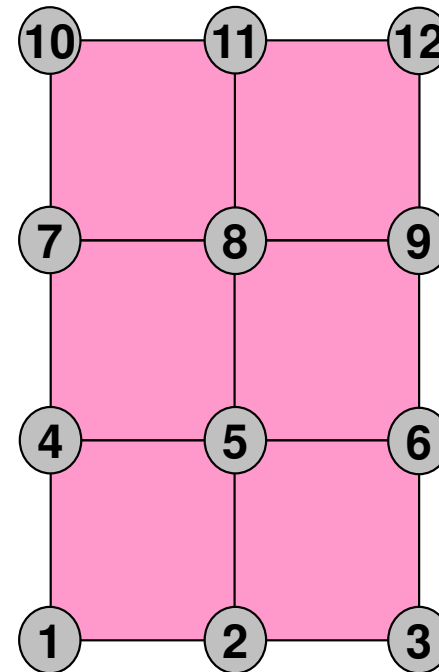
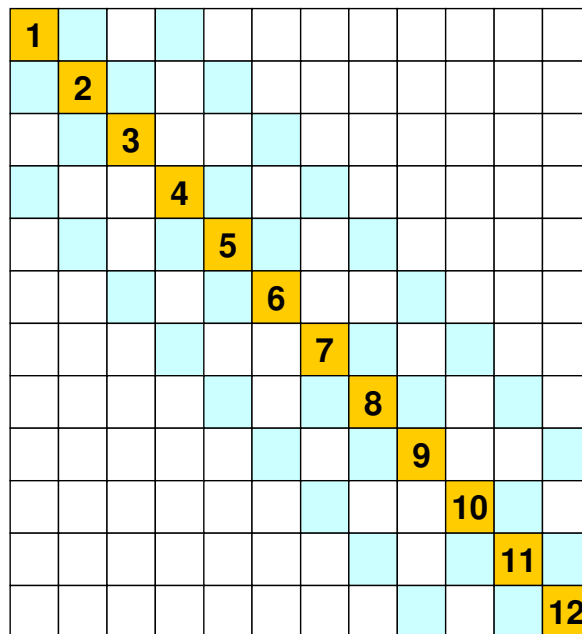


Example: 5-Point Stencil



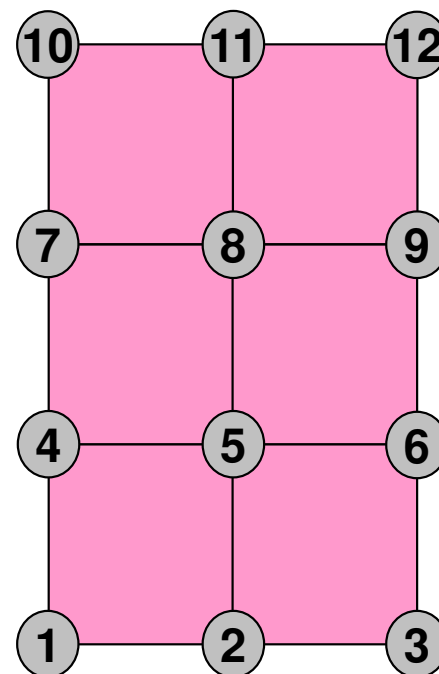
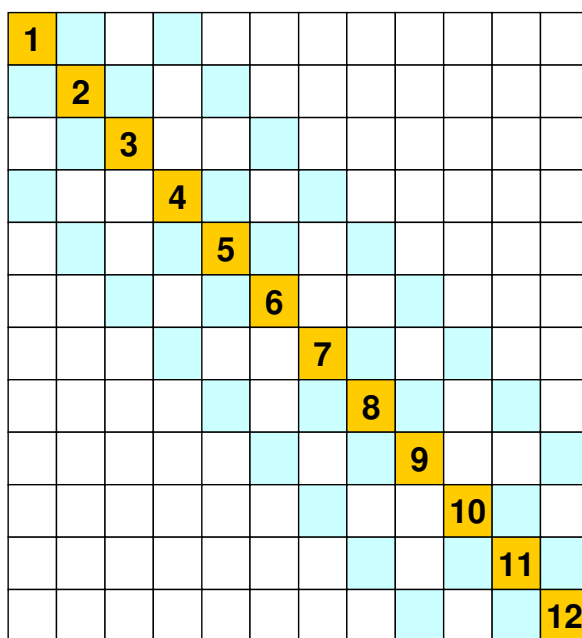
Coef. Matrix

$$\begin{bmatrix}
 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00
 \end{bmatrix}
 \times
 \begin{bmatrix}
 \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\ \\
 \end{bmatrix}
 =
 \begin{bmatrix}
 0.00 \\
 3.00 \\
 10.00 \\
 11.00 \\
 10.00 \\
 19.00 \\
 20.00 \\
 16.00 \\
 28.00 \\
 42.00 \\
 36.00 \\
 52.00
 \end{bmatrix}$$



Solution

$$\begin{bmatrix}
 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 \\
 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00
 \end{bmatrix}
 \begin{bmatrix}
 1.00 \\
 2.00 \\
 3.00 \\
 4.00 \\
 5.00 \\
 6.00 \\
 7.00 \\
 8.00 \\
 9.00 \\
 10.00 \\
 11.00 \\
 12.00
 \end{bmatrix}
 =
 \begin{bmatrix}
 0.00 \\
 3.00 \\
 10.00 \\
 11.00 \\
 10.00 \\
 19.00 \\
 20.00 \\
 16.00 \\
 28.00 \\
 42.00 \\
 36.00 \\
 52.00
 \end{bmatrix}$$



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	0.92
-0.17	5.83	-1.00	0.00	-1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.75
0.00	-0.17	5.83	0.00	0.00	-1.00	0.00	0.00	0.00	0.00	0.00	0.00	2.76
-0.17	0.00	0.00	5.83	-1.00	0.00	-1.00	0.00	0.00	0.00	0.00	0.00	3.79
0.00	-0.17	0.00	-0.17	5.66	-1.00	0.00	-1.00	0.00	0.00	0.00	0.00	4.46
0.00	0.00	-0.17	0.00	-0.18	5.65	0.00	0.00	-1.00	0.00	0.00	0.00	5.57
0.00	0.00	0.00	-0.17	0.00	0.00	5.83	-1.00	0.00	-1.00	0.00	0.00	6.66
0.00	0.00	0.00	0.00	-0.18	0.00	-0.17	5.65	-1.00	0.00	-1.00	0.00	7.25
0.00	0.00	0.00	0.00	0.00	-0.18	0.00	-0.18	5.65	0.00	0.00	-1.00	8.46
0.00	0.00	0.00	0.00	0.00	0.00	-0.17	0.00	0.00	5.83	-1.00	0.00	9.66
0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.18	0.00	-0.17	5.65	-1.00	10.54
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.18	0.00	-0.18	5.65	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	1.00
-0.17	5.83	-1.00	-0.17	-1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00
0.00	-0.17	5.83	-0.03	-0.17	-1.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00
-0.17	-0.03	0.00	5.83	-1.03	0.00	-1.00	0.00	0.00	0.00	0.00	0.00	4.00
0.00	-0.17	-0.03	-0.18	5.64	-1.03	-0.18	-1.00	0.00	0.00	0.00	0.00	5.00
0.00	0.00	-0.17	0.00	-0.18	5.64	-0.03	-0.18	-1.00	0.00	0.00	0.00	6.00
0.00	0.00	0.00	-0.17	-0.03	-0.01	5.82	-1.03	-0.01	-1.00	0.00	0.00	7.00
0.00	0.00	0.00	0.00	-0.18	-0.03	-0.18	5.63	-1.03	-0.18	-1.00	0.00	8.00
0.00	0.00	0.00	0.00	0.00	-0.18	0.00	-0.18	5.63	-0.03	-0.18	-1.00	9.00
0.00	0.00	0.00	0.00	0.00	0.00	-0.17	-0.03	-0.01	5.82	-1.03	-0.01	10.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.18	-0.03	-0.18	5.63	-1.03	11.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	-0.18	0.00	-0.18	5.63	12.00

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

Diagonal
Scaling
(Point
Jacobi)

6.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	6.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	6.00	0.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	0.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	0.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	0.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	0.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00



0.00
0.50
1.67
1.83
1.67
3.17
3.33
2.67
4.67
7.00
6.00
8.67

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

Weak

Strong



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

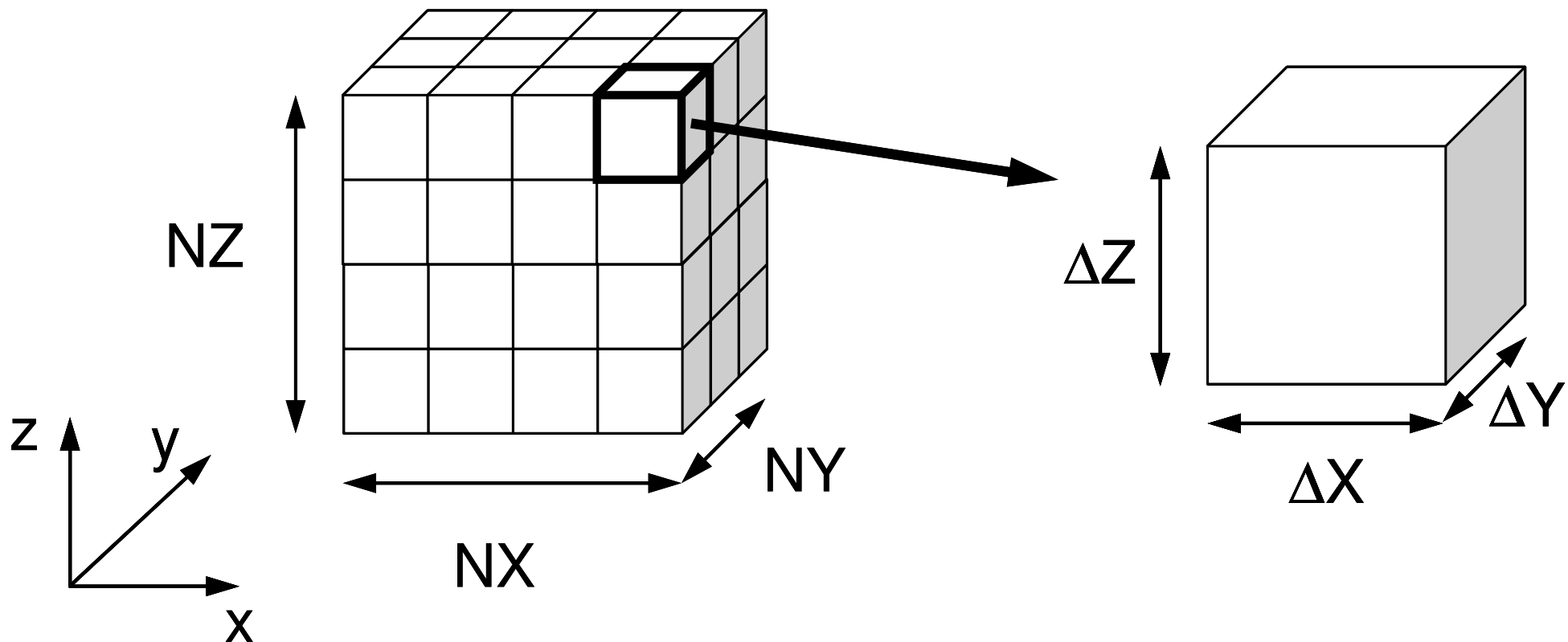
- 3D Poisson Equation/Poisson's Equation

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

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

3D Structured Mesh

Internal data structure is “unstructured”



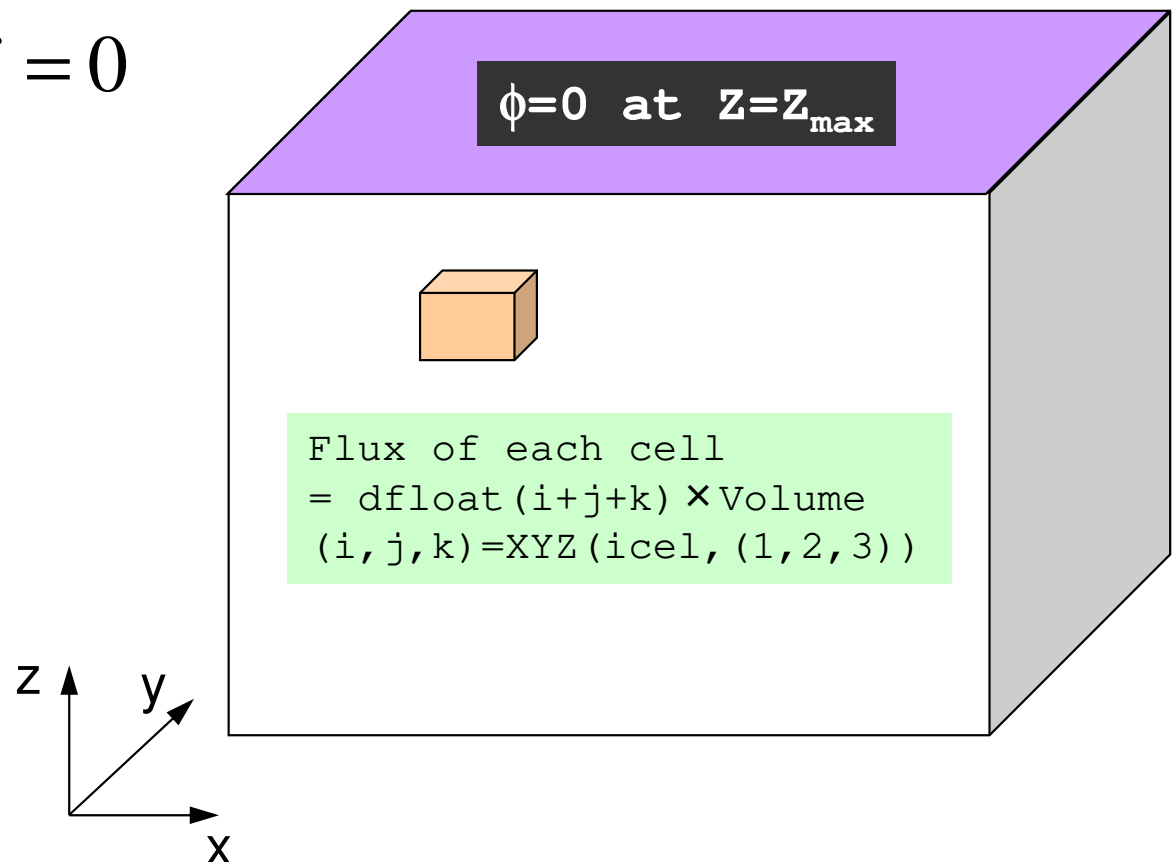
Target Problem: Variables are defined at cell-center's

Poisson Equation/ Poisson's Equation

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

Boundary Conditions (B.C.) etc.

- Volume Flux
- $\phi=0 @ Z=Z_{\max}$

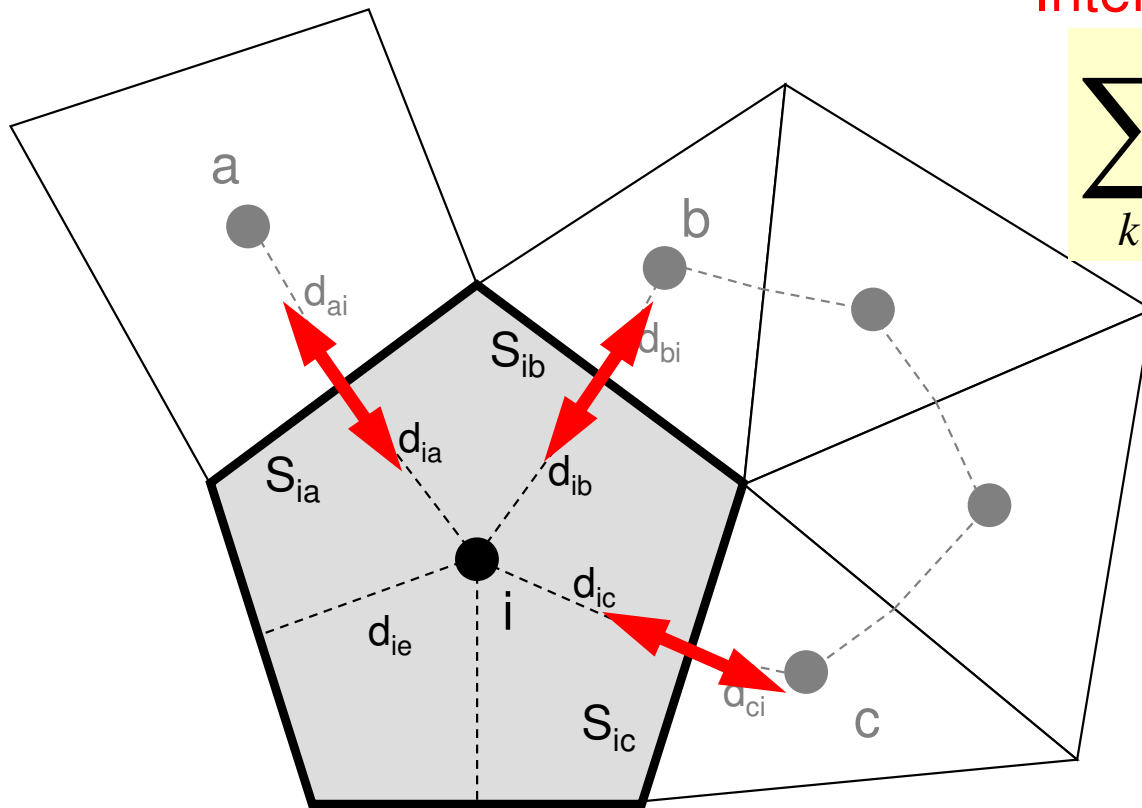


Poisson Equation by Finite Volume Method (FVM)

Conservation of Fluxes through
Surfaces

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

Diffusion:
Interaction with Neighbors

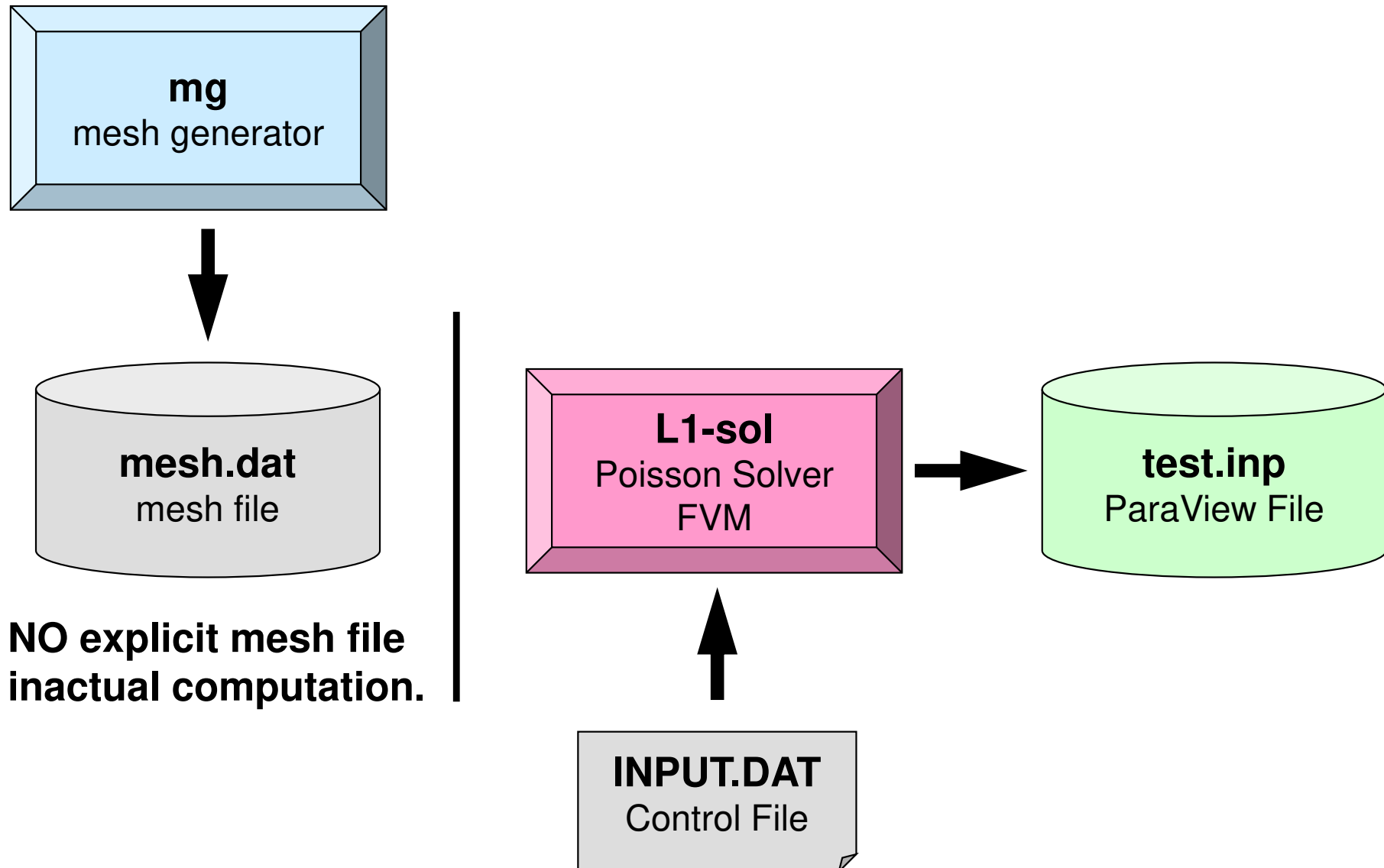


$$\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} (\phi_k - \phi_i) + V_i \dot{Q}_i = 0$$

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 multicore-f/L1/run
```

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

```
$> ls mg  
mg
```

Mesh Generator: **mg**

```
$> cd ../src
```

```
$> make
```

```
$> ls ../run/L1-sol  
L1-sol
```

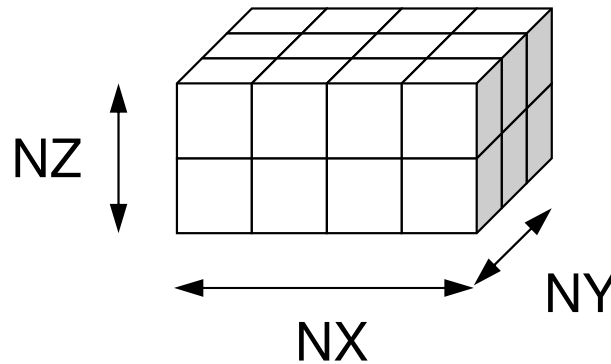
Poisson Solver (FVM): **L1-sol**

Running the Program

Mesh Generation

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

NX, NY, NZ



mesh.dat (1/5)

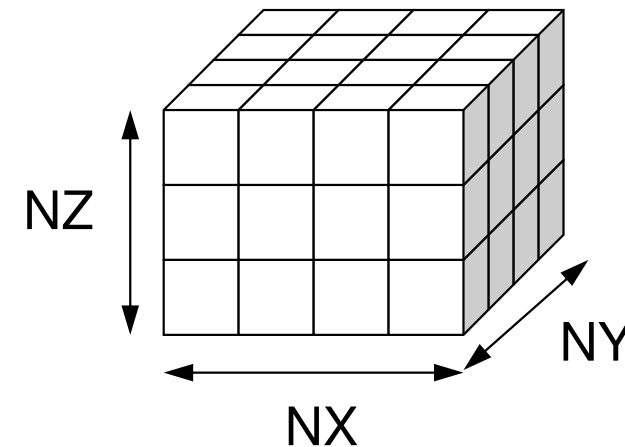
4	3	2							
24									
1	0	2	0	5	0	13	1	1	1
2	1	3	0	6	0	14	2	1	1
3	2	4	0	7	0	15	3	1	1
4	3	0	0	8	0	16	4	1	1
5	0	6	1	9	0	17	1	2	1
6	5	7	2	10	0	18	2	2	1
7	6	8	3	11	0	19	3	2	1
8	7	0	4	12	0	20	4	2	1
9	0	10	5	0	0	21	1	3	1
10	9	11	6	0	0	22	2	3	1
11	10	12	7	0	0	23	3	3	1
12	11	0	8	0	0	24	4	3	1
13	0	14	0	17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

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

```
read (21,'(10i10)') ICELTOT
```

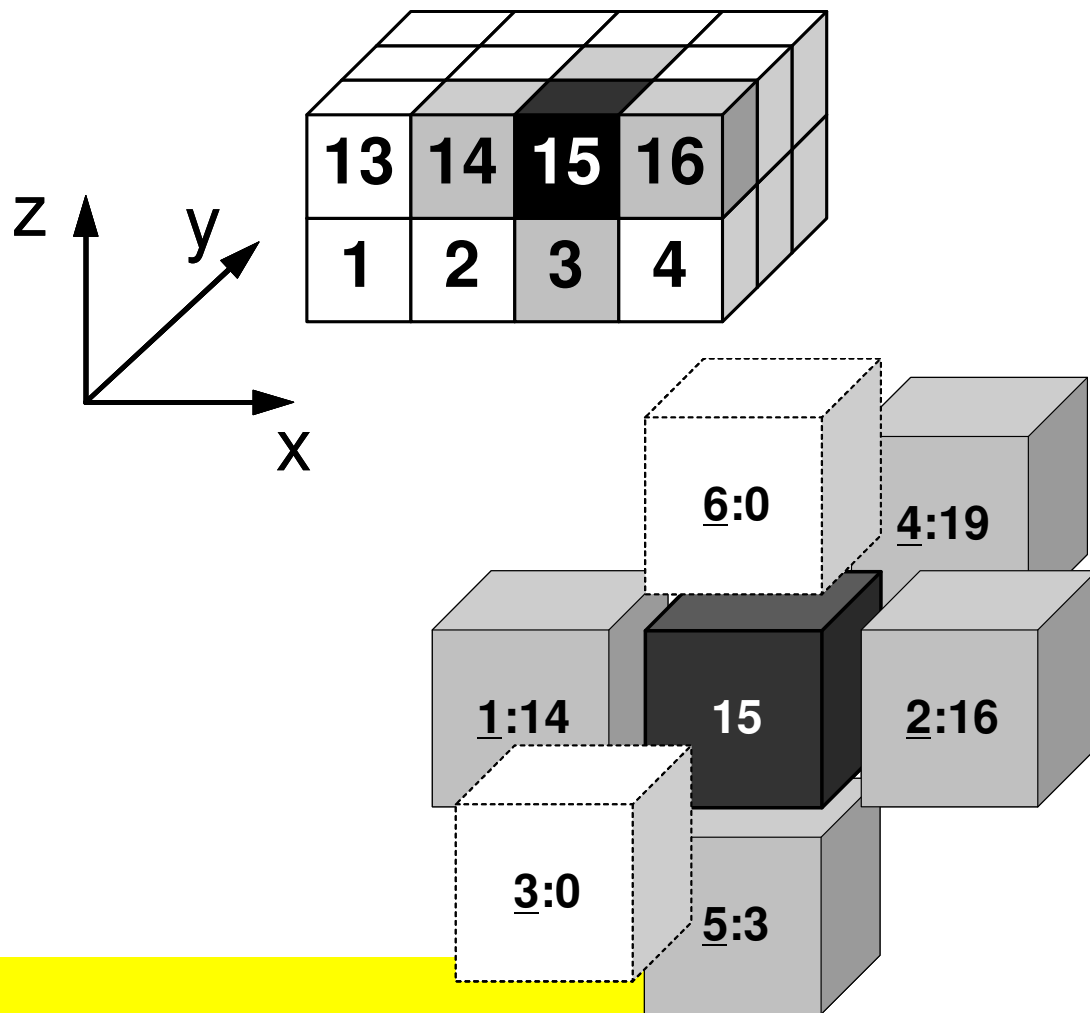
```
do i= 1, ICELTOT
```

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

```
enddo
```


mesh.dat (4/5)

Neighboring Cells: NEIBcell(i,k)



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

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

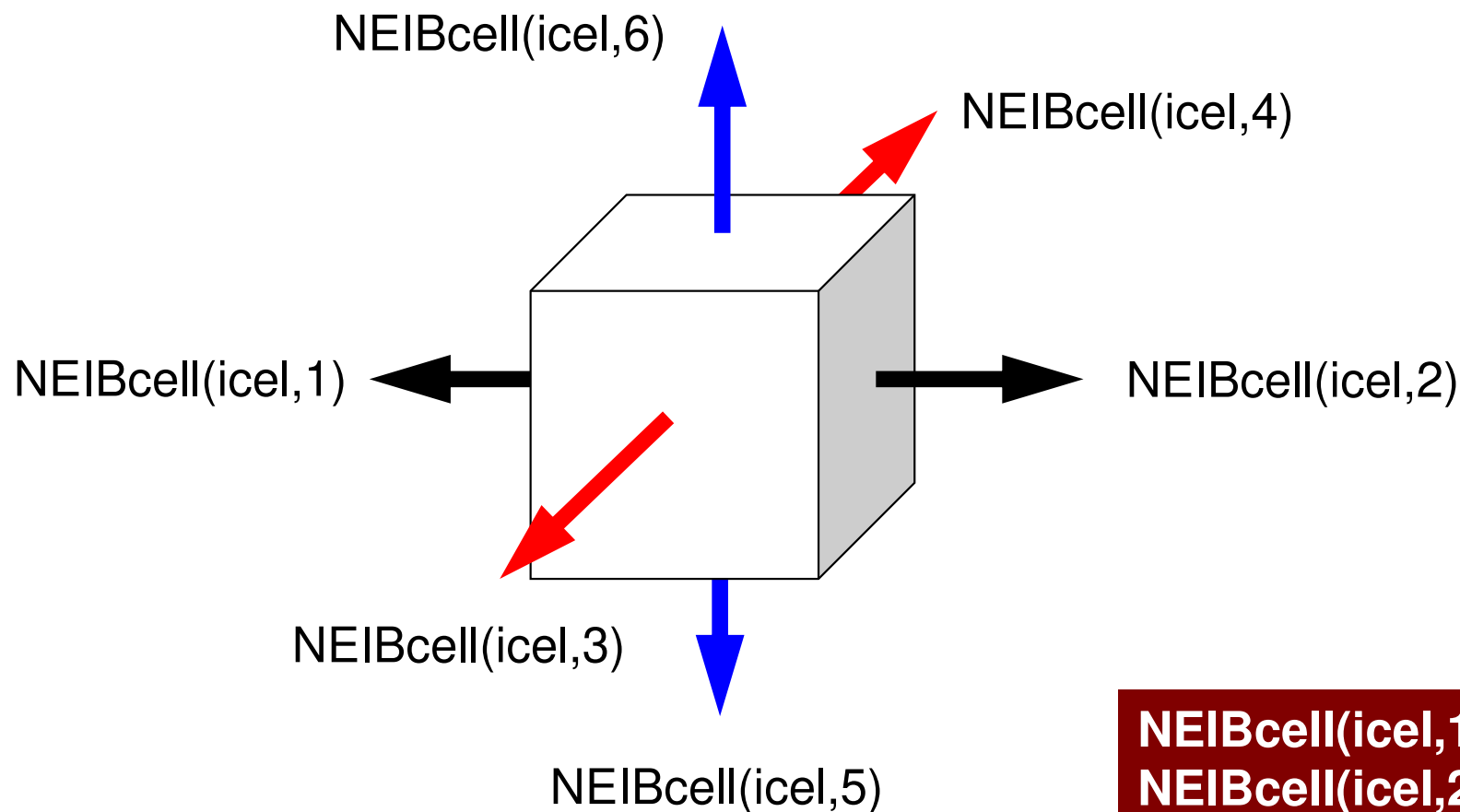
```
do i= 1, ICELTOT
```

```
  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

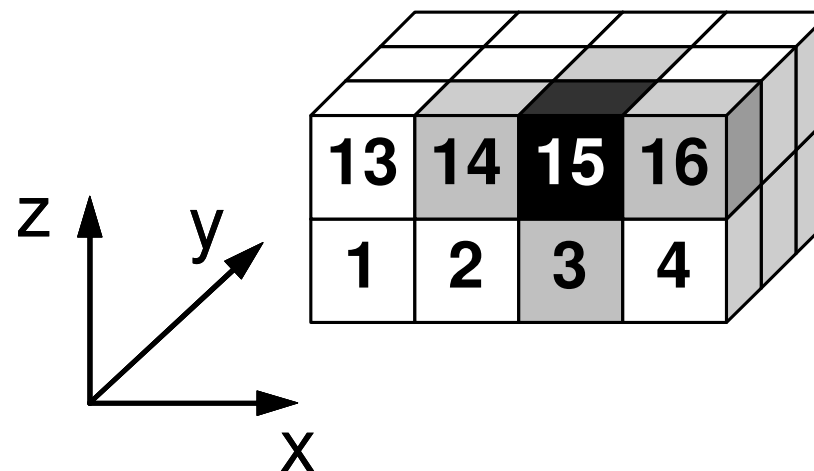


NEIBcell(icel,1)= icel - 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel - NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel - NX*NY
NEIBcell(icel,6)= icel + NX*NY

mesh.dat (5/5)

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

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

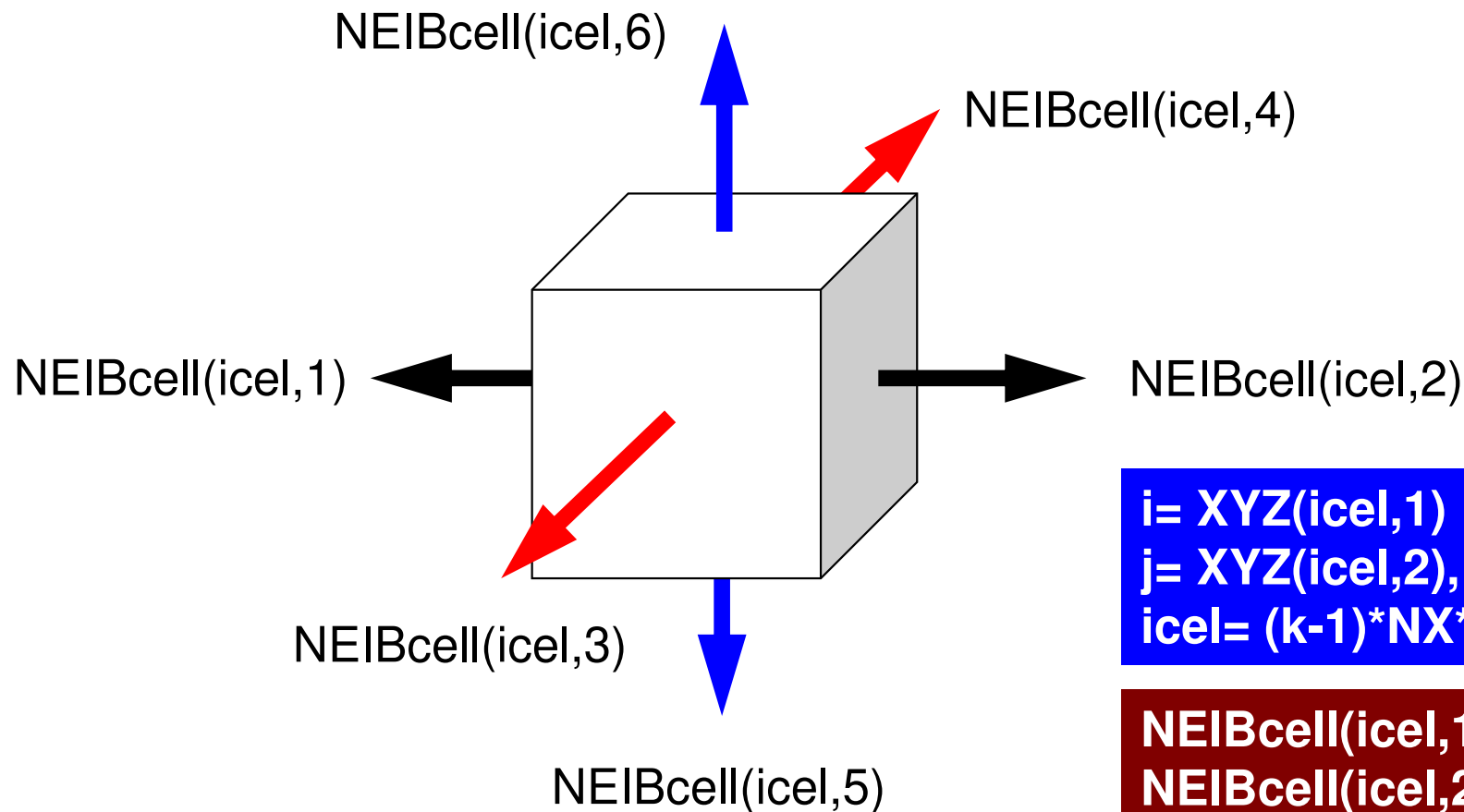


```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT
```

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

NEIBcell: ID of Neighboring Mesh/Cell

=0: for Boundary Surface



$i = \text{XYZ}(\text{icel}, 1)$
 $j = \text{XYZ}(\text{icel}, 2), k = \text{XYZ}(\text{icel}, 3)$
 $\text{icel} = (k-1) \cdot \text{NX} \cdot \text{NY} + (j-1) \cdot \text{NX} + i$

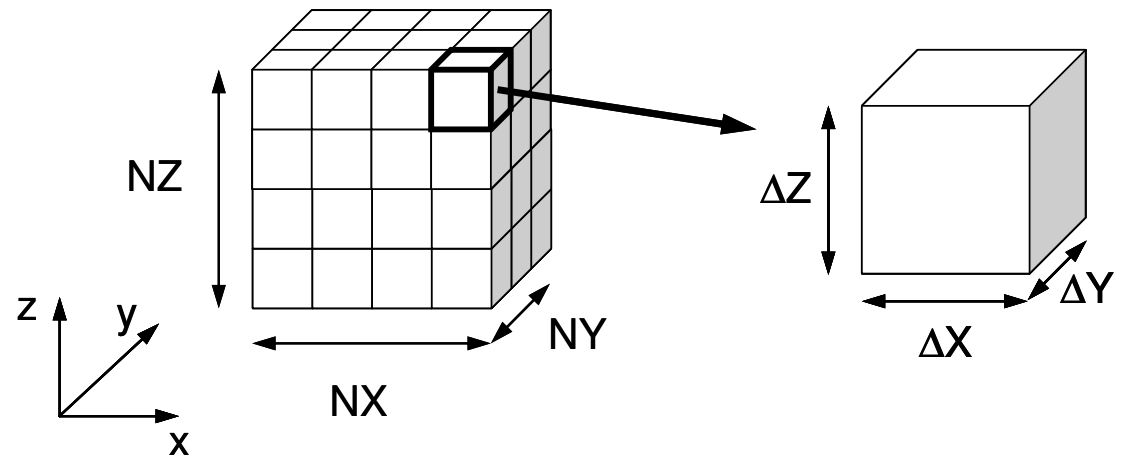
$\text{NEIBcell}(\text{icel}, 1) = \text{icel} - 1$
 $\text{NEIBcell}(\text{icel}, 2) = \text{icel} + 1$
 $\text{NEIBcell}(\text{icel}, 3) = \text{icel} - \text{NX}$
 $\text{NEIBcell}(\text{icel}, 4) = \text{icel} + \text{NX}$
 $\text{NEIBcell}(\text{icel}, 5) = \text{icel} - \text{NX} \cdot \text{NY}$
 $\text{NEIBcell}(\text{icel}, 6) = \text{icel} + \text{NX} \cdot \text{NY}$

Running the Program

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

32 32 32	NX/NY/NZ
1	METHOD 1:2:3
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	METHOD 1:2:3
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

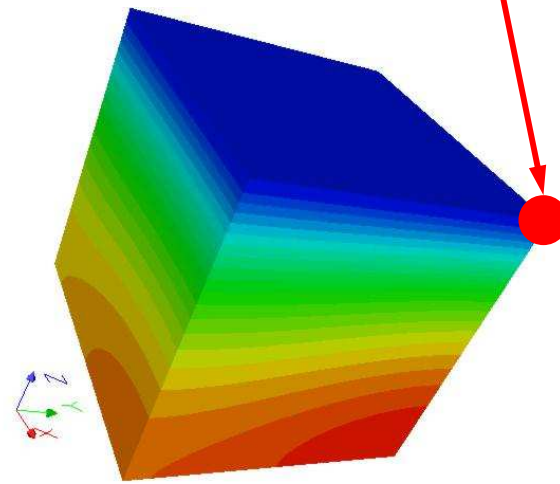
Running, Post Processing by ParaView

<http://nkl.cc.u-tokyo.ac.jp/21s/ParaView.pdf>

```
$> cd
$> cd multicore-f/L1/run
$> ./L1-sol
      1      4.504513E+00      Residual at the 1st Iteration
     75      8.377861E-09      Residual at convergence (<10-8)

##ANSWER      32768      9.297409E+02      Result at ●-point

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



UCD Format (1/2)

Unstructured Cell Data

要素の種類

点

線

三角形

四角形

四面体

角錐

三角柱

六面体

二次要素

線2

三角形2

四角形2

四面体2

角錐2

三角柱2

六面体2

キーワード

pt

line

tri

quad

tet

pyr

prism

hex

line2

tri2

quad2

tet2

pyr2

prism2

hex2

点

0

線

1

線2

2

1

三角形

0

1

2

三角形2

0

3

5

1

2

4

四角錐

1

4

2

3

四角錐2

1

4

2

3

四角形

0

3

1

2

四角形2

0

3

4

6

1

2

5

六面体

0

3

1

2

4

6

5

六面体2

0

11

3

10

8

9

2

16

17

19

18

4

12

15

14

5

13

6

三角錐

1

2

3

三角錐2

1

7

4

9

5

6

2

8

3

三角柱

0

2

3

5

4

三角柱2

0

8

2

12

6

1

7

3

13

11

10

9

4

14

5

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

```
program MAIN
```

```
use STRUCT
use PCG
use solver_ICCG
use solver_ICCG2
use solver_PCG
```

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

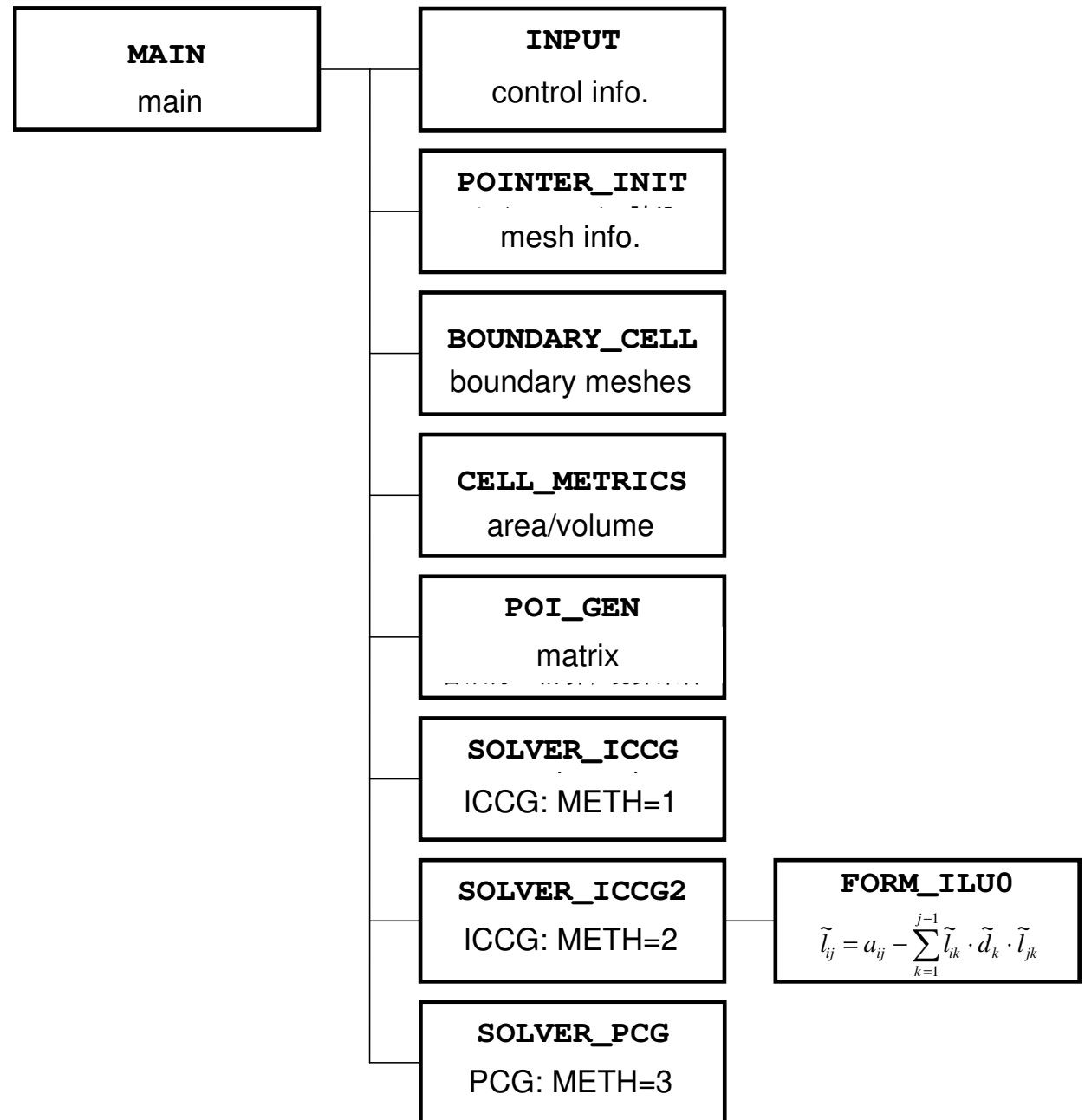
```
call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
```

```
PHI= 0.d0
```

```
if (METHOD.eq.1) call solve_ICCG (...)
if (METHOD.eq.2) call solve_ICCG2 (...)
if (METHOD.eq.3) call solve_PCG (...)
```

```
call OUTUCD
```

```
stop
end
```



module STRUCT

```

module STRUCT

  include 'precision.inc'

!C
!C-- METRICs & FLUX
  integer (kind=kint) :: ICELTOT, ICELTOTp, N
  integer (kind=kint) :: NX, NY, NZ, NXP1, NYP1, NZP1, IBNODTOT
  integer (kind=kint) :: NXc, NYc, NZc

  real (kind=kreal) ::
&    DX, DY, DZ, XAREA, YAREA, ZAREA, RDX, RDY, RDZ,
&    RDX2, RDY2, RDZ2, R2DX, R2DY, R2DZ

  real (kind=kreal), dimension(:), allocatable ::
&    VOLCEL, VOLNOD, RVC, RVN

  integer (kind=kint), dimension(:,:), allocatable ::
&    XYZ, NEIBcell

!C
!C-- BOUNDARYs
  integer (kind=kint) :: ZmaxCEltot
  integer (kind=kint), dimension(:), allocatable :: BC_INDEX, BC_NOD
  integer (kind=kint), dimension(:), allocatable :: ZmaxCEL

!C
!C-- WORK
  integer (kind=kint), dimension(:,:), allocatable :: IWKX
  real (kind=kreal), dimension(:,:), allocatable :: FCV

end module STRUCT

```

ICELTOT :

Number of meshes (NX x NY x NZ)

N :

Number of nodes

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

module PCG (1/5)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmex, NLmax, NCOLORTot, NCOLORK, NU, NL, METHOD
integer :: NPL, NPU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

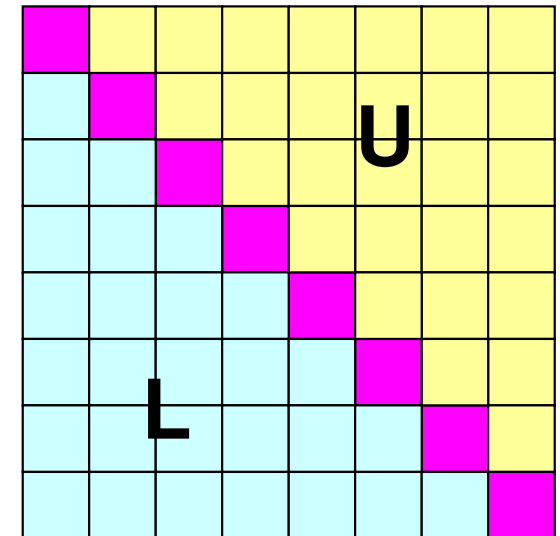
integer, dimension(:,:), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU

end module PCG

```

- Sparse Matrix
- Only non-zero off-diagonal components (CRS)
- Diagonal/Lower/Upper components are stored separately
- Although the matrix is symmetric, all the components are stored for efficient computing.



module PCG (2/5)

```

module PCG

integer, parameter :: N2= 256
integer :: NUm, NLmax, NCOLORTot, NCOLORK, NU, NL, METHOD
integer :: NPL, NPU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

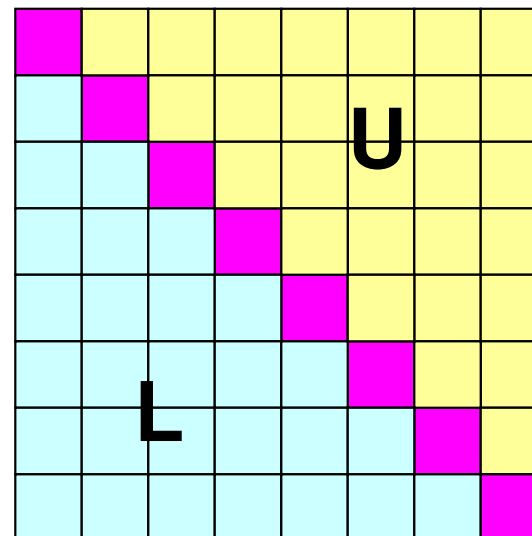
integer, dimension(:, :), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU

end module PCG

```

INL (ICELTOT)	# Non-zero off-diag. components (lower)
IAL (NL, ICELTOT)	Col. ID: non-zero off-diag. comp. (lower)
INU (ICELTOT)	# Non-zero off-diag. components (upper)
IAU (NU, ICELTOT)	Col. ID: non-zero off-diag. comp. (upper)
NU, NL	Max # of L/U non-zero off-diag. comp. (=6)
indexL (0:ICELTOT)	# Non-zero off-diag. comp. (lower, CRS)
indexU (0:ICELTOT)	# 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)



module PCG (3/5)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmex, NLmax, NCOLORtot, NCOLORk, NU, NL, METHOD
integer :: NPL, NPU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:,:), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU

end module 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)

module PCG (4/5)

```

module PCG

integer, parameter :: N2= 256
integer :: NUmex, NLmax, NCOLORTot, NCOLORK, NU, NL, METHOD
integer :: NPL, NPU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU

end module PCG

```

Auxiliary Arrays

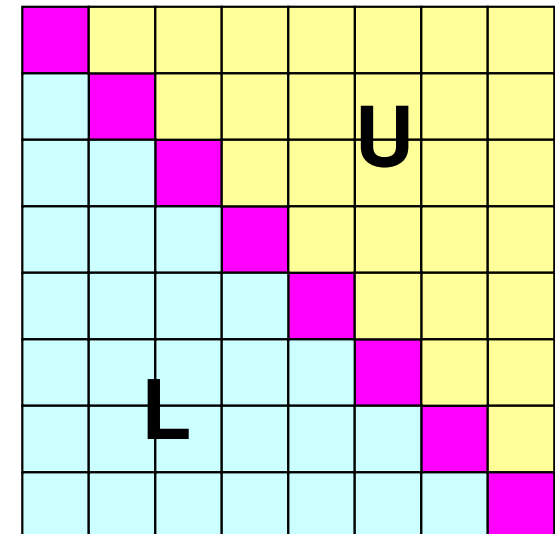
Lower Part (Column ID)

IAL(icou, i) < i

Upper Part (Column ID)

IAU(icou, i) > i

INL(ICELTOT)	# Non-zero off-diag. components (lower)
IAL(NL, ICELTOT)	Col. ID: non-zero off-diag. comp. (lower)
INU(ICELTOT)	# Non-zero off-diag. components (upper)
IAU(NU, ICELTOT)	Col. ID: non-zero off-diag. comp. (upper)
NU, NL	Max # of L/U non-zero off-diag. comp.s (=6)
indexL(0:ICELTOT)	# Non-zero off-diag. comp. (lower, CRS)
indexU(0:ICELTOT)	# 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)



module PCG (5/5)

```

module PCG

integer, parameter :: N2= 256
integer :: NUm, NLmax, NCOLORTot, NCOLORK, NU, NL, METHOD
integer :: NPL, NPU

real(kind=8) :: EPSICCG

real(kind=8), dimension(:), allocatable :: D, PHI, BFORCE
real(kind=8), dimension(:), allocatable :: AL, AU

integer, dimension(:), allocatable :: INL, INU, COLORindex
integer, dimension(:), allocatable :: OLDtoNEW, NEWtoOLD

integer, dimension(:, :), allocatable :: IAL, IAU

integer, dimension(:), allocatable :: indexL, itemL
integer, dimension(:), allocatable :: indexU, itemU

end module PCG

```

Auxiliary Arrays

Lower Part (Column ID)

IAL(icou, i) < i

INL(i) : Number@each row

Upper Part (Column ID)

IAU(icou, i) > i

INU(i) : Number@each row

INL(ICELTOT)

IAL(NL, ICELTOT)

INU(ICELTOT)

IAU(NU, ICELTOT)

NU, NL

indexL(0:ICELTOT)

indexU(0:ICELTOT)

NPL, NPU

itemL(NPL), itemU(NPU)

Non-zero off-diag. components (lower)

Col. ID: non-zero off-diag. comp. (lower)

Non-zero off-diag. components (upper)

Col. ID: non-zero off-diag. comp. (upper)

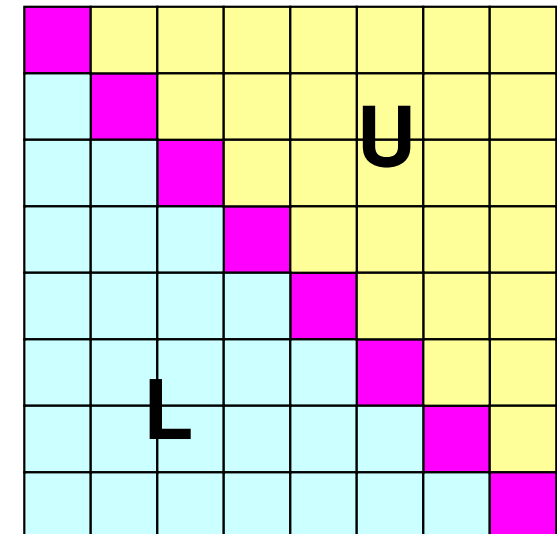
Max # of L/U non-zero off-diag. comp.s (=6)

Non-zero off-diag. comp. (lower, CRS)

Non-zero off-diag. comp. (upper, CRS)

Total # of L/U non-zero off-diag. comp.

Col. ID: non-zero off-diag. comp. (L/U, 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 (0:N)	I	Number of Lower non-zero off-diag. comp. (CRS)
indexU (0:N)	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 (NL, N)	I	Column ID of Lower non-zero off-diag. comp.
IAU (NU, N)	I	Column ID of Uppwer non-zero off-diag. comp.

- INL, INU $\Rightarrow$ indexL, indexU
- IAL, IAU $\Rightarrow$ itemL, itemU

Why Auxiliary Arrays ?

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

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

```
do i= 1, N
```

```
    VAL= D(i)*p(i)
```

```
    do k= indexL(i-1)+1, indexL(i)
```

```
        VAL= VAL + AL(k)*p(itemL(k))
```

```
    enddo
```

```
    do k= indexU(i-1)+1, indexU(i)
```

```
        VAL= VAL + AU(k)*p(itemU(k))
```

```
    enddo
```

```
    q(i)= VAL
```

```
enddo
```

Structure of the Program

```
program MAIN
```

```
use STRUCT
use PCG
use solver_ICCG
use solver_ICCG2
use solver_PCG
```

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

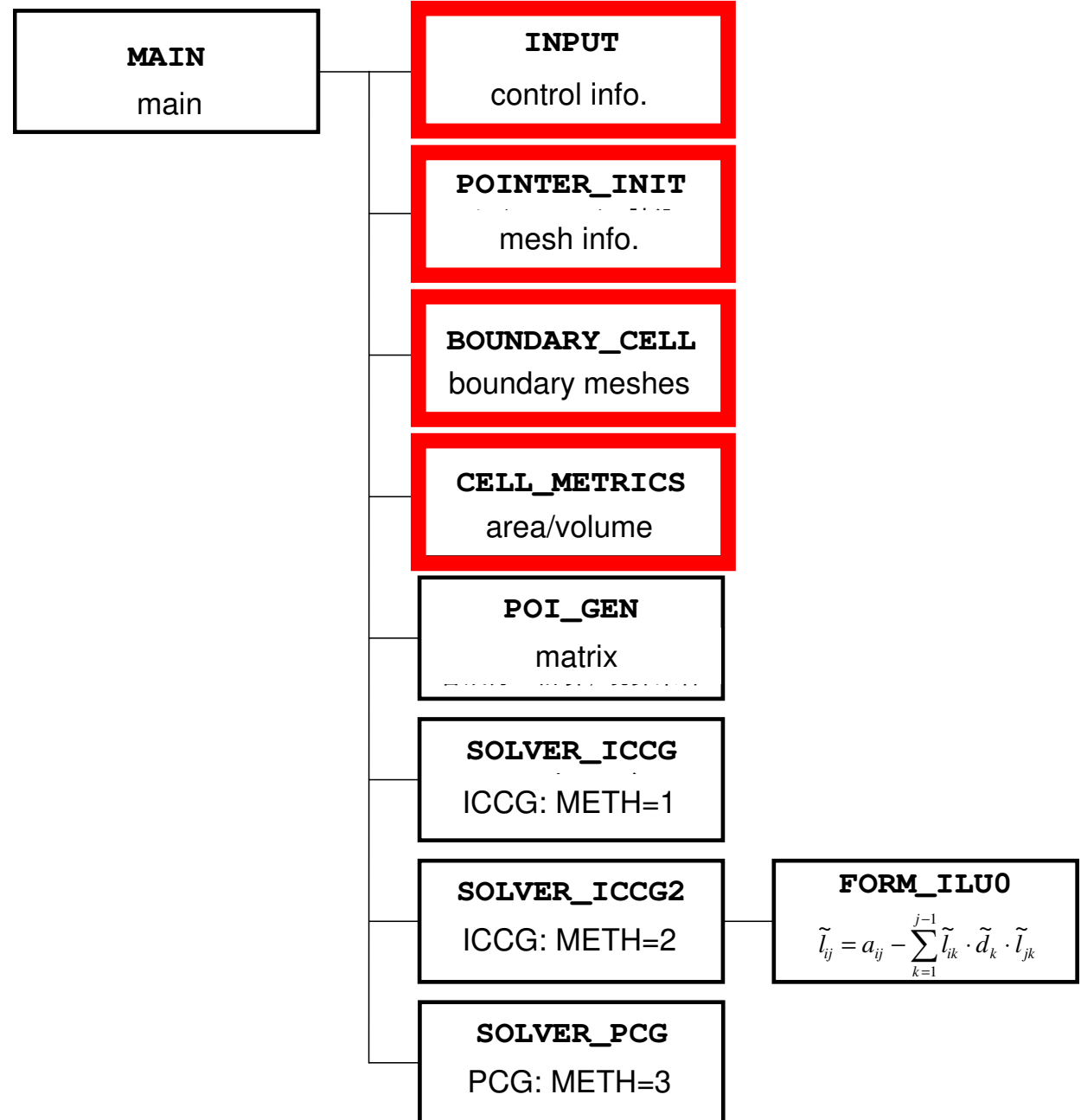
```
call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
```

```
PHI= 0. d0
```

```
if (METHOD.eq.1) call solve_ICCG (...)
if (METHOD.eq.2) call solve_ICCG2(...)
if (METHOD.eq.3) call solve_PCG (...)
```

```
call OUTUCD
```

```
stop
end
```



input: reading “INPUT.DAT”

```

!C
!C***
!C*** INPUT
!C***
!C
!C INPUT CONTROL DATA
!C
      subroutine INPUT
      use STRUCT
      use PCG

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

      character*80 CNTFIL

!C
!C-- CNTL. file
      open (11, file=' INPUT.DAT', status='unknown')
      read (11,*) NX, NY, NZ
      read (11,*) METHOD
      read (11,*) DX, DY, DZ
      read (11,*) EPSICCG
      close (11)
!C===

      return
      end

```

32 32 32

1

1.00e-00 1.00e-00 1.00e-00

1.0e-08

NX/NY/NZ

MEHOD 1-3

DX/DY/DZ

EPSICCG

pointer_init (1/3): “mesh.dat”

```
!C
!C***
!C*** POINTER_INIT
!C***
!C
      subroutine POINTER_INIT

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

!C
!C +-----+
!C | Generating MESH info. |
!C +-----+
!C===
      ICELTOT= NX  * NY  * NZ

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

      allocate (NEIBcell (ICELTOT, 6), XYZ (ICELTOT, 3))
      NEIBcell= 0
```

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

pointer_init (2/3): “mesh.dat”

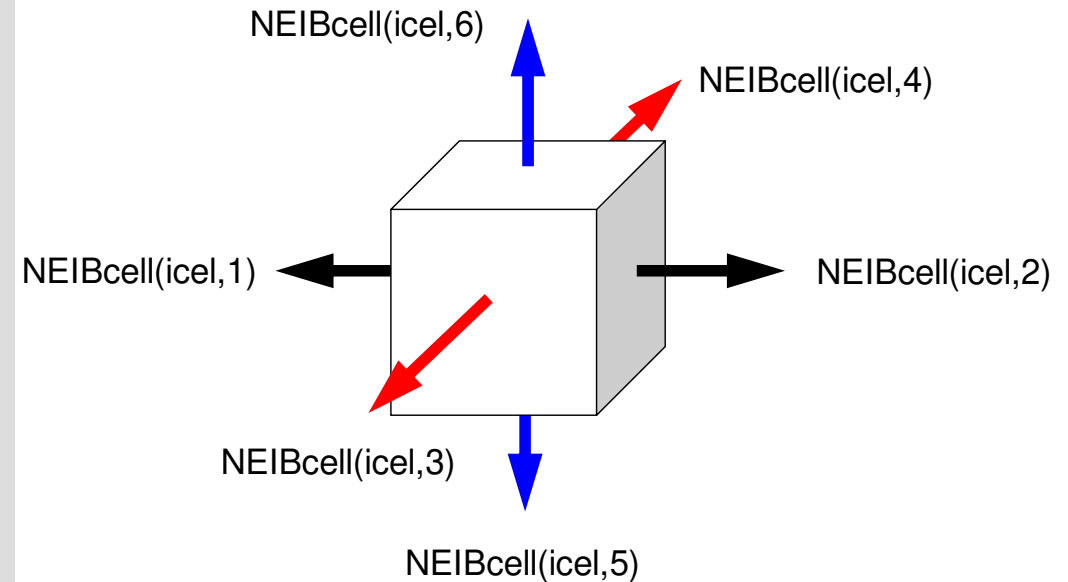
```

do k= 1, NZ
  do j= 1, NY
    do i= 1, NX
      icel= (k-1)*NX*NY + (j-1)*NX + i
      NEIBcell(icel,1)= icel - 1
      NEIBcell(icel,2)= icel + 1
      NEIBcell(icel,3)= icel - NX
      NEIBcell(icel,4)= icel + NX
      NEIBcell(icel,5)= icel - NX*NY
      NEIBcell(icel,6)= icel + NX*NY
      if (i.eq. 1) NEIBcell(icel,1)= 0
      if (i.eq. NX) NEIBcell(icel,2)= 0
      if (j.eq. 1) NEIBcell(icel,3)= 0
      if (j.eq. NY) NEIBcell(icel,4)= 0
      if (k.eq. 1) NEIBcell(icel,5)= 0
      if (k.eq. NZ) NEIBcell(icel,6)= 0

      XYZ(icel,1)= i
      XYZ(icel,2)= j
      XYZ(icel,3)= k

    enddo
  enddo
enddo
!C===

```



i= XYZ(icel,1)
j= XYZ(icel,2), k= XYZ(icel,3)
icel= (k-1)*NX*NY + (j-1)*NX + i

NEIBcell(icel,1)= icel - 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel - NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel - NX*NY
NEIBcell(icel,6)= icel + NX*NY

pointer_init (3/3): “mesh.dat”

```
!C
!C +-----+
!C | Parameters |
!C +-----+
!C===
      if (DX.le.0.0e0) then
        DX= 1.d0 / dfloat(NX)
        DY= 1.d0 / dfloat(NY)
        DZ= 1.d0 / dfloat(NZ)
      endif

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

      IBNODTOT= NXP1 * NYP1
      N        = NXP1 * NYP1 * NZP1
!C===
      return
      end
```

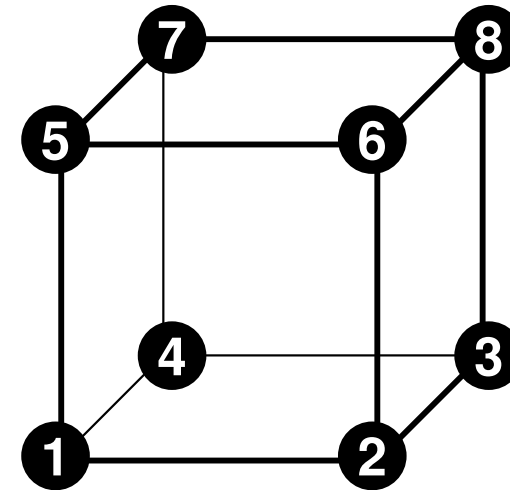
if DX is no larger than 0.0

pointer_init (3/3): “mesh.dat”

```
!C
!C +-----+
!C | Parameters |
!C +-----+
!C===
      if (DX. le. 0. 0e0) then
        DX= 1. d0 / dfloat(NX)
        DY= 1. d0 / dfloat(NY)
        DZ= 1. d0 / dfloat(NZ)
      endif

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

      IBNODTOT= NXP1 * NYP1
      N        = NXP1 * NYP1 * NZP1
!C===
      return
      end
```



NXP1, NYP1, NZP1 :

Number of nodes in x/y/z directions

IBNODTOT :

= NXP1 x NYP1

N :

Number of modes
meshes (for visualization)

boundary_cell

```

!C
!C***
!C*** BOUNDARY_CELL
!C***
!C
      subroutine BOUNDARY_CELL
      use STRUCT

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

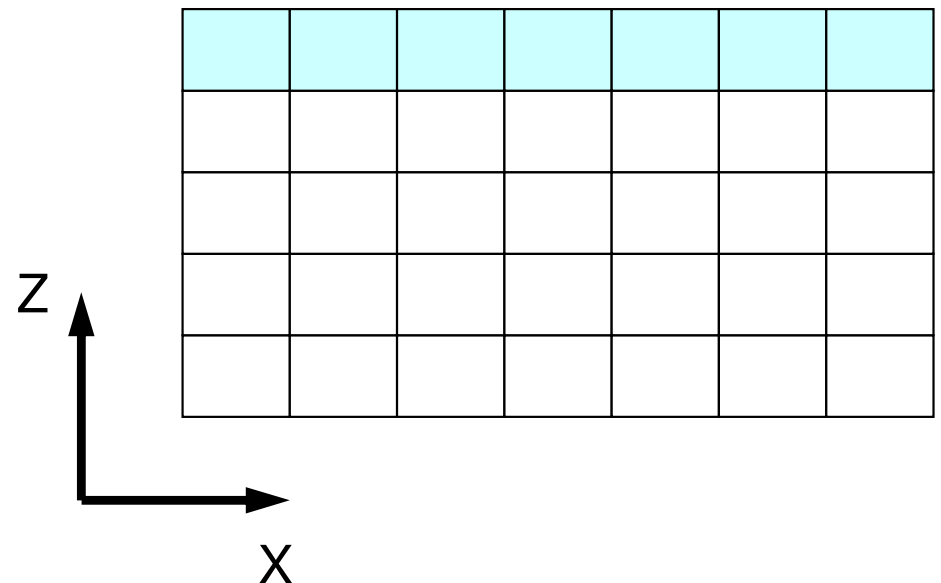
!C
!C +-----+
!C | Zmax |
!C +-----+
!C===
      IFACTOT= NX * NY
      ZmaxCEltot= IFACTOT

      allocate (ZmaxCEL (ZmaxCEltot))

      icou= 0
      k    = NZ
      do j= 1, NY
      do i= 1, NX
          icel= (k-1)*IFACTOT + (j-1)*NX + i
          icou= icou + 1
          ZmaxCEL(icou)= icel
      enddo
      enddo
!C===
      return
      end

```

Meshes @ $Z=Z_{\max}$
 Number: $Z_{\max}CEL_{\text{tot}}$
 Mesh ID: $Z_{\max}CEL(:)$



cell_metrics

```

!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H, O-Z)

!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))

!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1. d0 / DX
      RDY= 1. d0 / DY
      RDZ= 1. d0 / DZ

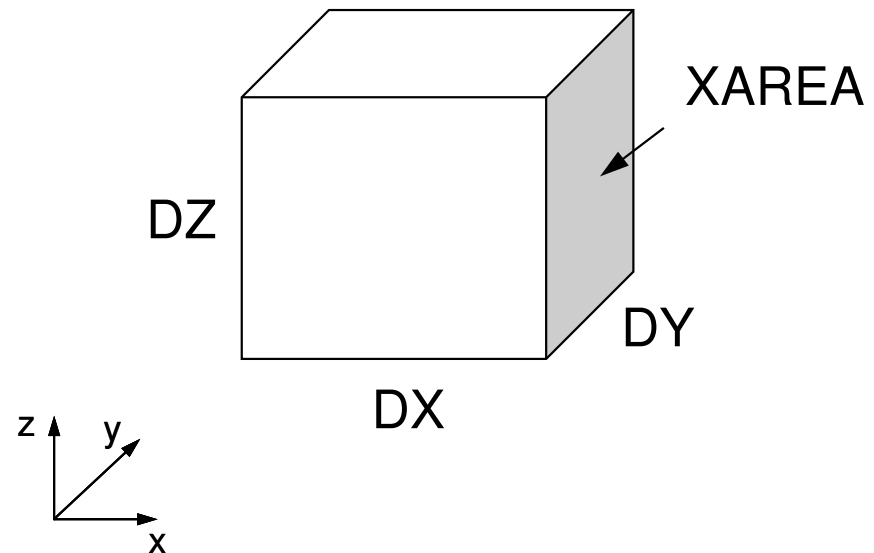
      RDX2= 1. d0 / (DX**2)
      RDY2= 1. d0 / (DY**2)
      RDZ2= 1. d0 / (DZ**2)
      R2DX= 1. d0 / (0. 50d0*DX)
      R2DY= 1. d0 / (0. 50d0*DY)
      R2DZ= 1. d0 / (0. 50d0*DZ)

      V0= DX * DY * DZ
      RV0= 1. d0/V0
      VOLCEL= V0
      RVC   = RV0

      return
      end

```

Parameters for Computations



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

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

cell_metrics

```

!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H, O-Z)

!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))

!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1.d0 / DX
      RDY= 1.d0 / DY
      RDZ= 1.d0 / DZ

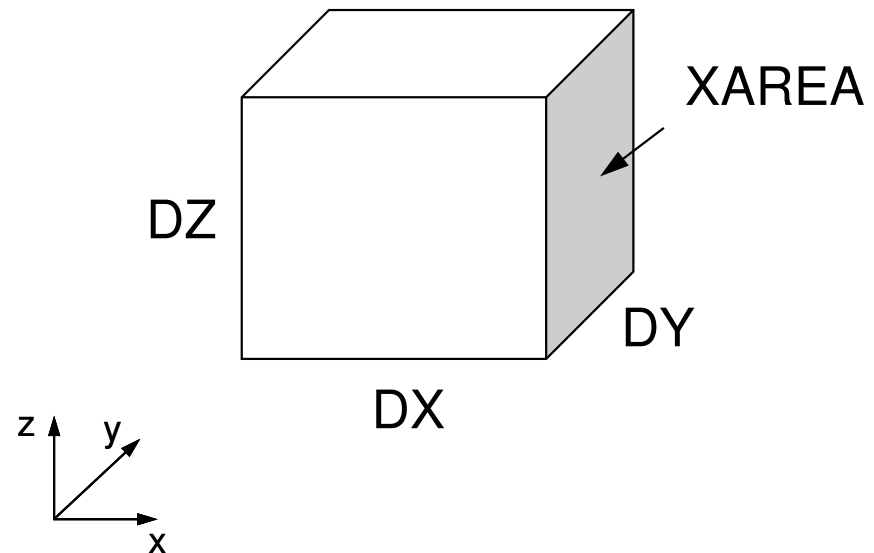
      RDX2= 1.d0 / (DX**2)
      RDY2= 1.d0 / (DY**2)
      RDZ2= 1.d0 / (DZ**2)
      R2DX= 1.d0 / (0.50d0*DX)
      R2DY= 1.d0 / (0.50d0*DY)
      R2DZ= 1.d0 / (0.50d0*DZ)

      V0= DX * DY * DZ
      RV0= 1.d0/V0
      VOLCEL= V0
      RVC   = RV0

      return
      end

```

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

```

!C
!C***
!C*** CELL_METRICS
!C***
!C
      subroutine CELL_METRICS
      use STRUCT
      use PCG
      implicit REAL*8 (A-H, O-Z)

!C
!C-- ALLOCATE
      allocate (VOLCEL(ICELTOT))
      allocate (   RVC(ICELTOT))

!C
!C-- VOLUME, AREA, PROJECTION etc.
      XAREA= DY * DZ
      YAREA= DX * DZ
      ZAREA= DX * DY

      RDX= 1.d0 / DX
      RDY= 1.d0 / DY
      RDZ= 1.d0 / DZ

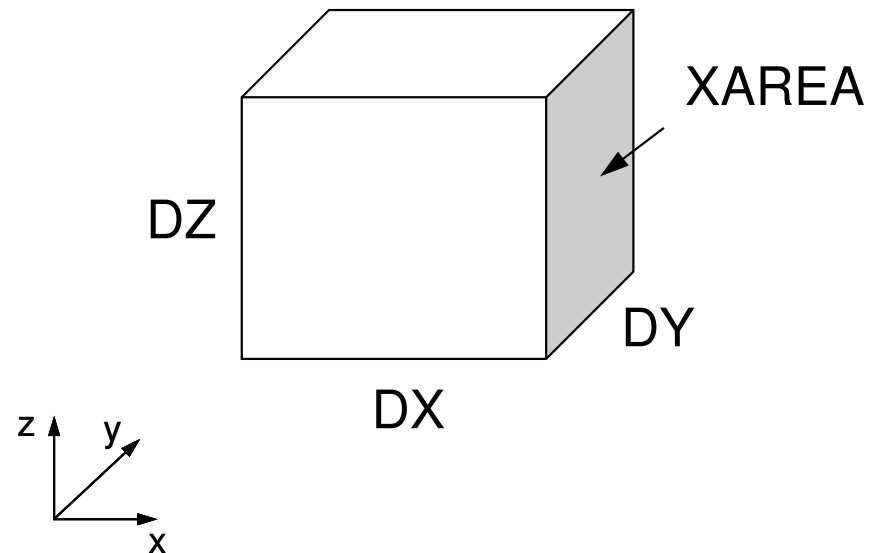
      RDX2= 1.d0 / (DX**2)
      RDY2= 1.d0 / (DY**2)
      RDZ2= 1.d0 / (DZ**2)
      R2DX= 1.d0 / (0.50d0*DX)
      R2DY= 1.d0 / (0.50d0*DY)
      R2DZ= 1.d0 / (0.50d0*DZ)

      V0= DX * DY * DZ
      RV0= 1.d0/V0
      VOLCEL= V0
      RVC   = RV0

      return
      end

```

Parameters for Computations



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

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

Structure of the Program

```
program MAIN
```

```
use STRUCT
use PCG
use solver_ICCG
use solver_ICCG2
use solver_PCG
```

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

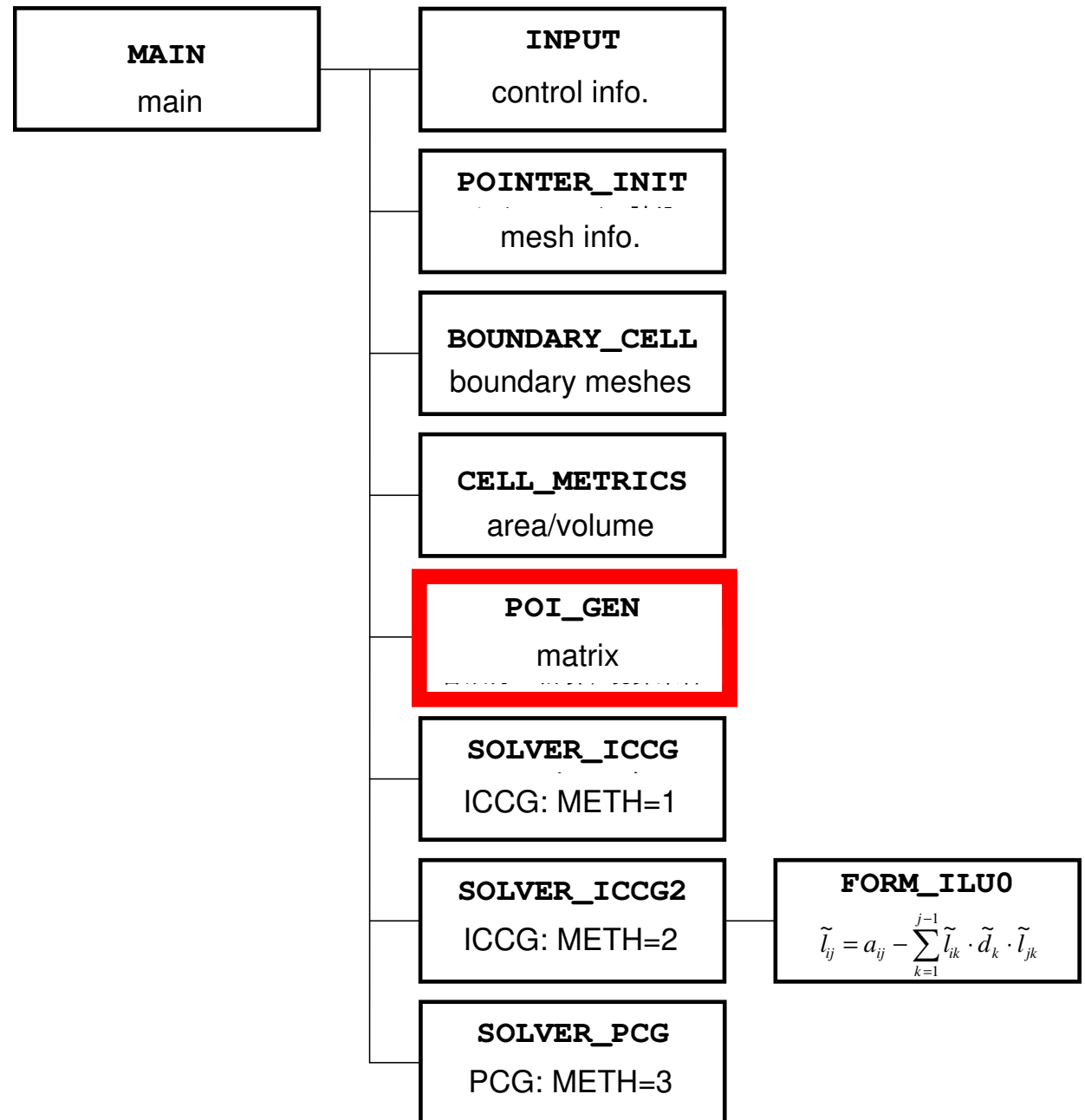
```
call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
```

```
PHI= 0. d0
```

```
if (METHOD.eq.1) call solve_ICCG (...)
if (METHOD.eq.2) call solve_ICCG2(...)
if (METHOD.eq.3) call solve_PCG (...)
```

```
call OUTUCD
```

```
stop
end
```



poi_gen (1/7)

```
subroutine POI_GEN

  use STRUCT
  use PCG

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

!C
!C-- INIT.
  nn = ICELTOT

  NU= 6
  NL= 6

  allocate (BFORCE(nn), D(nn), PHI(nn))
  allocate (INL(nn), INU(nn), IAL(NL, nn), IAU(NU, nn))

  PHI= 0. d0
  D= 0. d0

  INL= 0
  INU= 0
  IAL= 0
  IAU= 0
```


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 (0:N) , indexU (0:N)	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 (NL, N) , IAU (NU, N)	I	Column ID of L/U non-zero off-diag. comp.

```

!C
!C +-----+
!C | CONNECTIVITY |
!C +-----+
!C===
      do icel= 1, ICELTOT
        icN1= NEIBcell(icel,1)
        icN2= NEIBcell(icel,2)
        icN3= NEIBcell(icel,3)
        icN4= NEIBcell(icel,4)
        icN5= NEIBcell(icel,5)
        icN6= NEIBcell(icel,6)

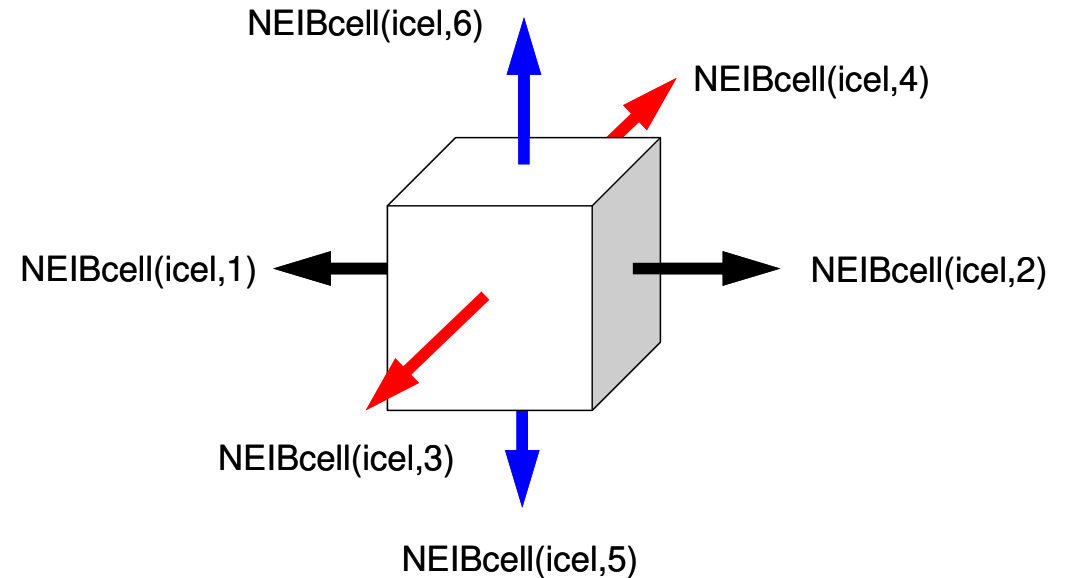
        icouG= 0
        if (icN5.ne.0) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN5
          INL(      icel)= icou
        endif

        if (icN3.ne.0) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN3
          INL(      icel)= icou
        endif

        if (icN1.ne.0) then
          icou= INL(icel) + 1
          IAL(icou, icel)= icN1
          INL(      icel)= icou
        endif
      enddo

```

poi_gen (2/7)



Lower Triangular Part

$NEIBcell(icel,5) = icel - NX * NY$
 $NEIBcell(icel,3) = icel - NX$
 $NEIBcell(icel,1) = icel - 1$

```

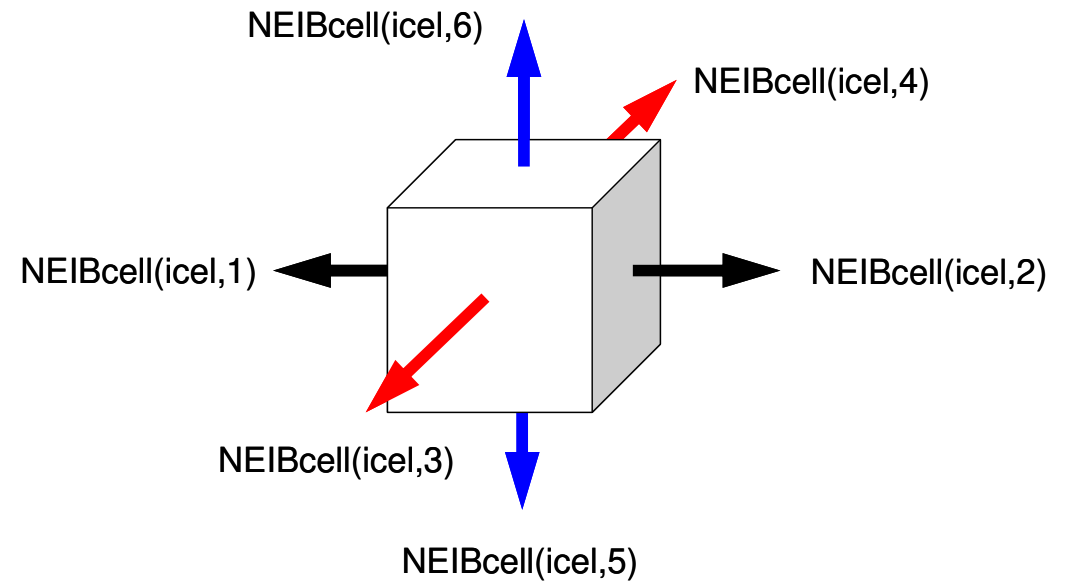
if (icN2.ne.0) then
  icou= INU(icel) + 1
  IAU(icou, icel)= icN2
  INU(      icel)= icou
endif

if (icN4.ne.0) then
  icou= INU(icel) + 1
  IAU(icou, icel)= icN4
  INU(      icel)= icou
endif

if (icN6.ne.0) then
  icou= INU(icel) + 1
  IAU(icou, icel)= icN6
  INU(      icel)= icou
endif
enddo
!C===

```

poi_gen (3/7)



Upper Triangular Part

```

NEIBcell(icel,2)= icel + 1
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,6)= icel + NX*NY

```

```
!C
!C-- 1D array
```

```
allocate (indexL(0:nn), indexU(0:nn))
indexL= 0
indexU= 0
```

```
do icel= 1, ICELTOT
  indexL(icel)= INL(icel)
  indexU(icel)= INU(icel)
enddo
```

```
do icel= 1, ICELTOT
  indexL(icel)= indexL(icel) + indexL(icel-1)
  indexU(icel)= indexU(icel) + indexU(icel-1)
enddo
```

```
NPL= indexL(ICELTOT)
NPU= indexU(ICELTOT)
```

```
allocate (itemL(NPL), AL(NPL))
allocate (itemU(NPU), AU(NPU))
```

```
itemL= 0
itemU= 0
```

```
AL= 0. d0
AU= 0. d0
```

```
!C===
```

poi_gen (4/7)

Name	Type	Content
D (N)	R	Diagonal components of the matrix (N= ICELTOT)
BFORCE (N)	R	RHS vector
PHI (N)	R	Unknown vector
indexL(0:N) , indexU(0:N)	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)

```
do i= 1, N
```

```
  VAL= D(i)*p(i)
```

```
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL + AL(k)*p(itemL(k))
  enddo
```

```
  do k= indexU(i-1)+1, indexU(i)
    VAL= VAL + AU(k)*p(itemU(k))
  enddo
```

```
  q(i)= VAL
```

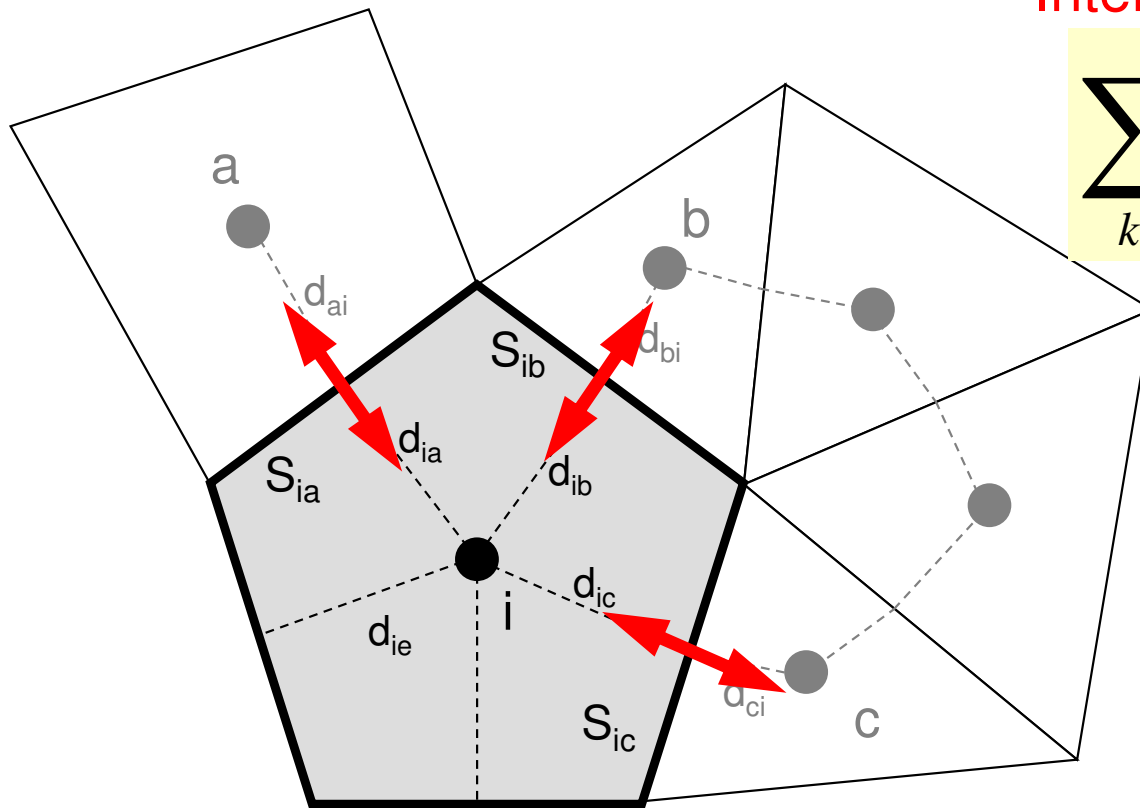
```
enddo
```

Poisson Equation by Finite Volume Method (FVM)

Conservation of Fluxes through
Surfaces

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

Diffusion:
Interaction with Neighbors



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

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

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

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

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

D (diagonal)

**AL, AU
(off-diag.)**

**BFORCE
(RHS)**

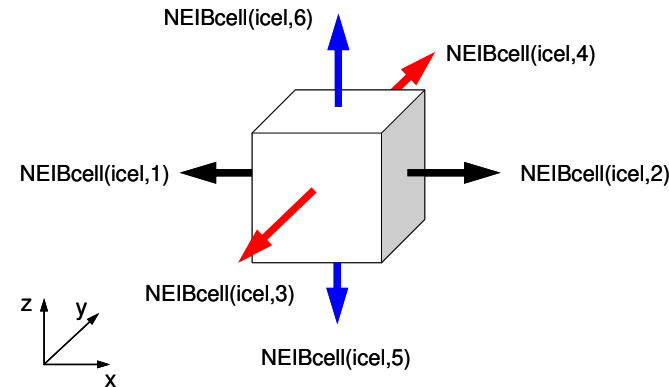
poi_gen (5/7)

```
!C
!C +-----+
!C | INTERIOR & NEUMANN BOUNDARY CELLS |
!C +-----+
!C==
```

```
icouG= 0
do icel= 1, ICELTOT
  icN1= NEIBcell(icel,1)
  icN2= NEIBcell(icel,2)
  icN3= NEIBcell(icel,3)
  icN4= NEIBcell(icel,4)
  icN5= NEIBcell(icel,5)
  icN6= NEIBcell(icel,6)
  VOL0= VOLCEL(icel)
  icou= 0
  if (icN5.ne.0) then
    coef =RDZ * ZAREA
    D(icel)= D(icel) - coef
    icou= icou + 1
    k = icou + indexL(icel-1)
    itemL(k)= icN5
    AL(k)= coef
  endif

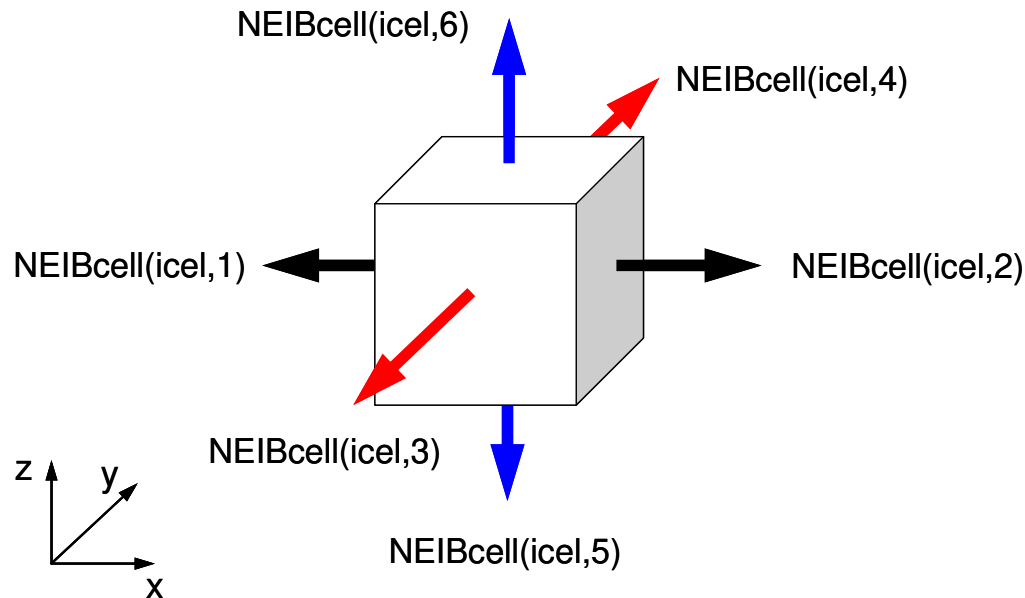
  if (icN3.ne.0) then
    coef =RDY * YAREA
    D(icel)= D(icel) - coef
    icou= icou + 1
    k = icou + indexL(icel-1)
    itemL(k)= icN3
    AL(k)= coef
  endif

  if (icN1.ne.0) then
    coef =RDX * XAREA
    D(icel)= D(icel) - coef
    icou= icou + 1
    k = icou + indexL(icel-1)
    itemL(k)= icN1
    AL(k)= coef
  endif
enddo
```



$$\begin{aligned}
 & \frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \\
 & \frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \\
 & \boxed{\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y} - \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0
 \end{aligned}$$

Calculations of Coefficients



```

if (icN5.ne.0) then
  coef = RDZ * ZAREA
  D(icel)= D(icel) - coef

  icou= icou + 1
  k= icou + indexL(icel-1)

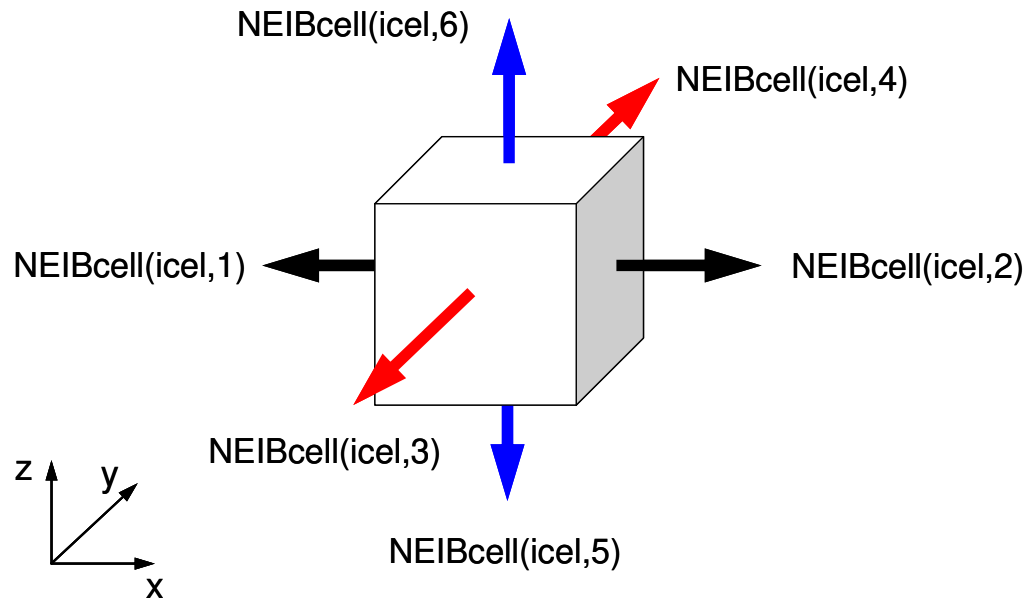
  itemL(k)= icN5
  AL(k)= coef
endif
  
```

$$\frac{\phi_{neib(icel,1)} - \phi_{icel}}{\Delta x} \Delta y \Delta z + \frac{\phi_{neib(icel,2)} - \phi_{icel}}{\Delta x} \Delta y \Delta z +$$

$$\frac{\phi_{neib(icel,3)} - \phi_{icel}}{\Delta y} \Delta z \Delta x + \frac{\phi_{neib(icel,4)} - \phi_{icel}}{\Delta y} \Delta z \Delta x +$$

$$\frac{\phi_{neib(icel,5)} - \phi_{icel}}{\Delta z} \Delta x \Delta y - \frac{\phi_{neib(icel,6)} - \phi_{icel}}{\Delta z} \Delta x \Delta y + f_{icel} \Delta x \Delta y \Delta z = 0$$

Calculations of Coefficients



```

if (icN5.ne.0) then
  coef = RDZ * ZAREA
  D(icel)= D(icel) - coef

  icou= icou + 1
  k= icou + indexL(icel-1)

  itemL(k)= icN5
  AL(k)= coef
endif
  
```

$$\frac{\Delta y \Delta z}{\Delta x} (\phi_{neib(icel,1)} - \phi_{icel}) + \frac{\Delta y \Delta z}{\Delta x} (\phi_{neib(icel,2)} - \phi_{icel}) +$$

$$\frac{\Delta z \Delta x}{\Delta y} (\phi_{neib(icel,3)} - \phi_{icel}) + \frac{\Delta z \Delta x}{\Delta y} (\phi_{neib(icel,4)} - \phi_{icel}) +$$

ZAREA

RDZ

$$\frac{\Delta x \Delta y}{\Delta z} (\phi_{neib(icel,5)} - \phi_{icel}) + \frac{\Delta x \Delta y}{\Delta z} (\phi_{neib(icel,6)} - \phi_{icel}) + f_{icel} \Delta x \Delta y \Delta z = 0$$

poi_gen (6/7)

```

icou= 0
if (icN2.ne.0) then
  coef = RDX * XAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN2
  AU(k)= coef
endif

```

```

if (icN4.ne.0) then
  coef = RDY * YAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN4
  AU(k)= coef
endif

```

```

if (icN6.ne.0) then
  coef = RDZ * ZAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN6
  AU(k)= coef
endif

```

```

ii= XYZ(icel,1)
jj= XYZ(icel,2)
kk= XYZ(icel,3)

```

```

BFORCE(icel)= -dfloat(ii+jj+kk) * VOL0
enddo

```

```
!C===
```

poi_gen (6/7)

Volume Flux

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

$$i_0 = XYZ(icel, 1),$$

$$j_0 = XYZ(icel, 2),$$

$$k_0 = XYZ(icel, 3)$$

$$XYZ(icel, k) (k=1,2,3)$$

Index for location of finite-difference mesh in X-/Y-/Z-axis.

```

icou= 0
if (icN2.ne.0) then
  coef = RDX * XAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN2
  AU(k)= coef
endif

```

```

if (icN4.ne.0) then
  coef = RDY * YAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN4
  AU(k)= coef
endif

```

```

if (icN6.ne.0) then
  coef = RDZ * ZAREA
  D(icel)= D(icel) - coef
  icou= icou + 1
  k = icou + indexU(icel-1)
  itemU(k)= icN6
  AU(k)= coef
endif

```

```

ii= XYZ(icel, 1)
jj= XYZ(icel, 2)
kk= XYZ(icel, 3)

```

```

BFORCE(icel)= -dfloat(ii+jj+kk) * VOLO
enddo

```

```

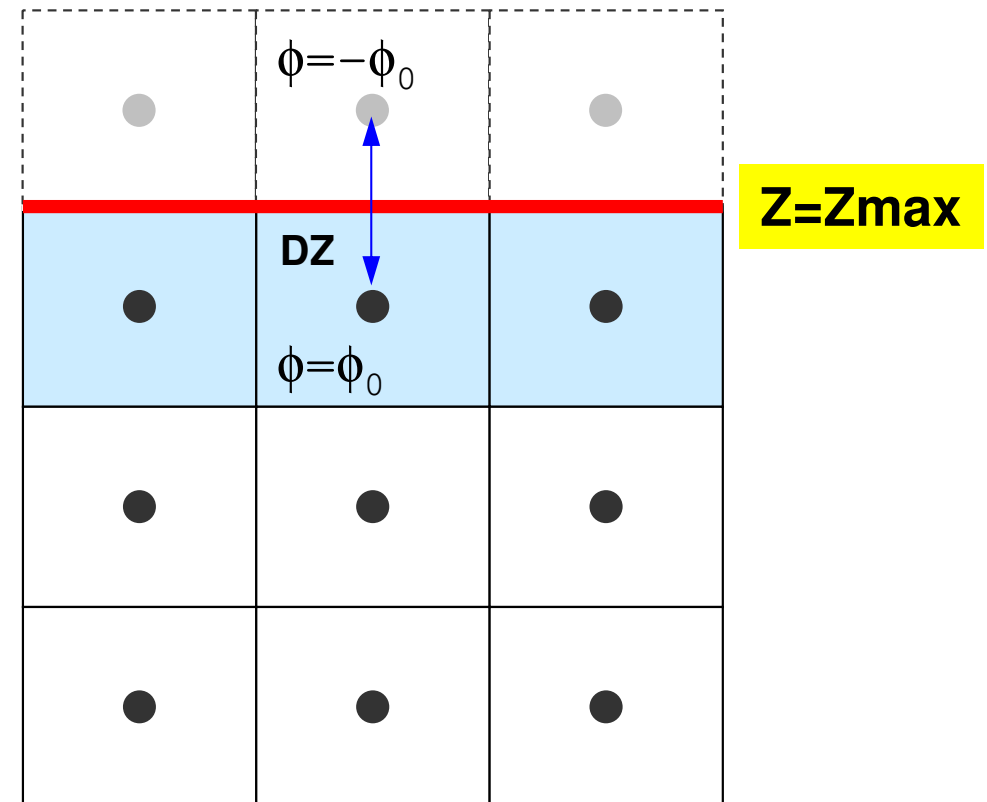
!C===

```

poi_gen (7/7)

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

```
!C
!C +-----+
!C | DIRICHLET BOUNDARY CELLS |
!C +-----+
!C TOP SURFACE
!C==
      do ib= 1, ZmaxCELtot
        icel= ZmaxCEL(ib)
        coef= 2. d0 * RDZ * ZAREA
        D(icel)= D(icel) - coef
      enddo
!C==
      return
      end
```



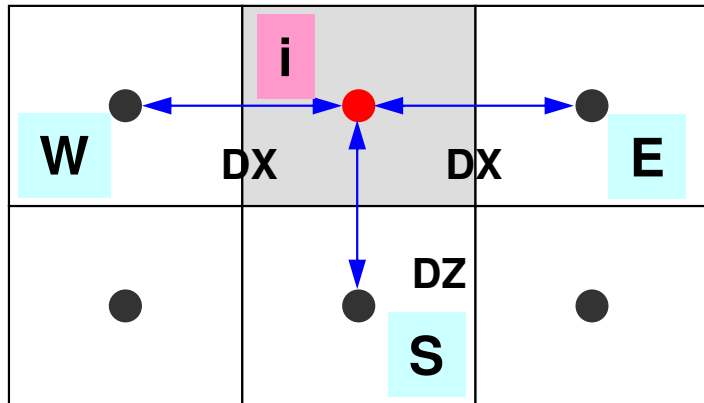
1st Order Approximation:

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.



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

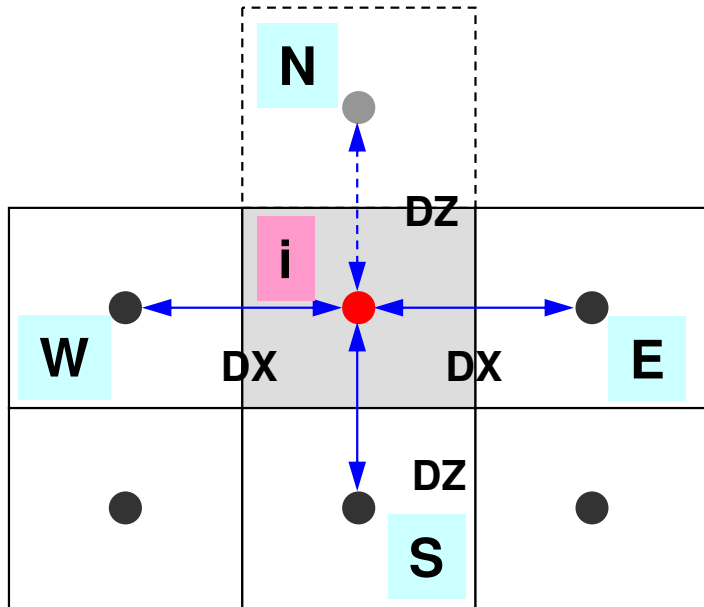
$$-\left[\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} \right] \phi_i + \left[\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} \phi_k \right] = -V_i \dot{Q}_i$$

D (diagonal)

**AL, AU
(off-diag.)**

**BFORCE
(RHS)**

Dirichlet B.C.



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

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

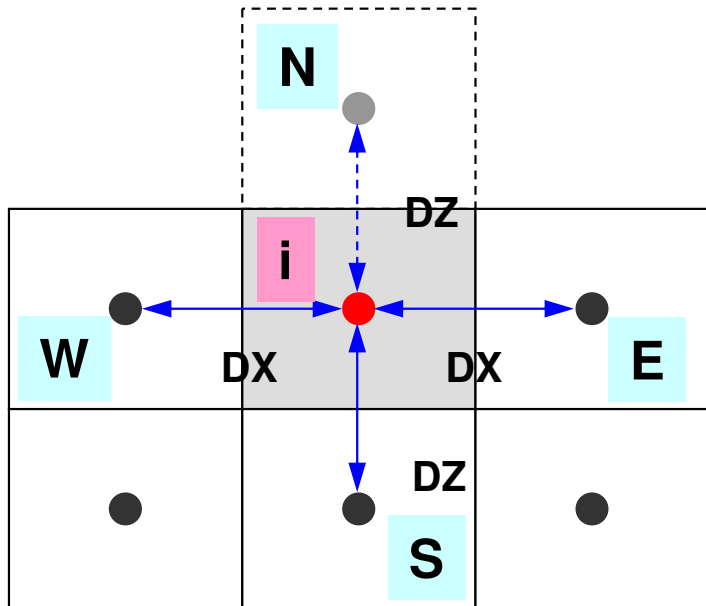
D (diagonal)

**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.



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

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

D (diagonal)

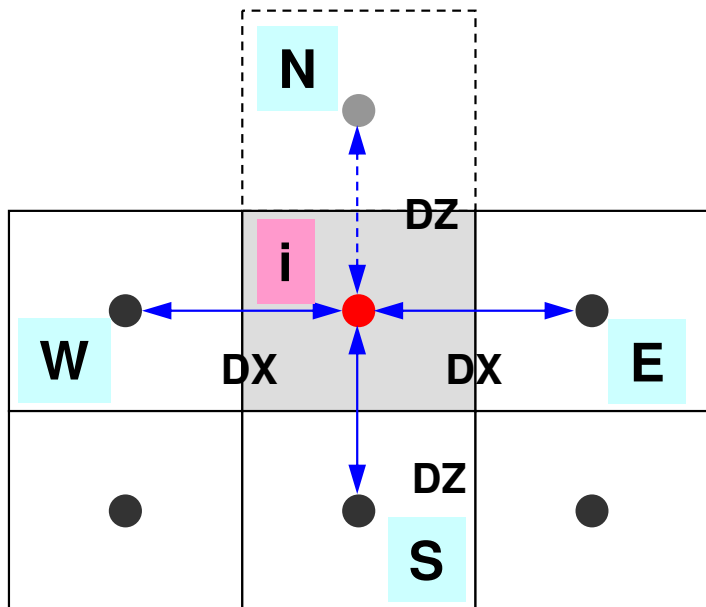
**AL, AU
(off-diag.)**

**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.



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

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

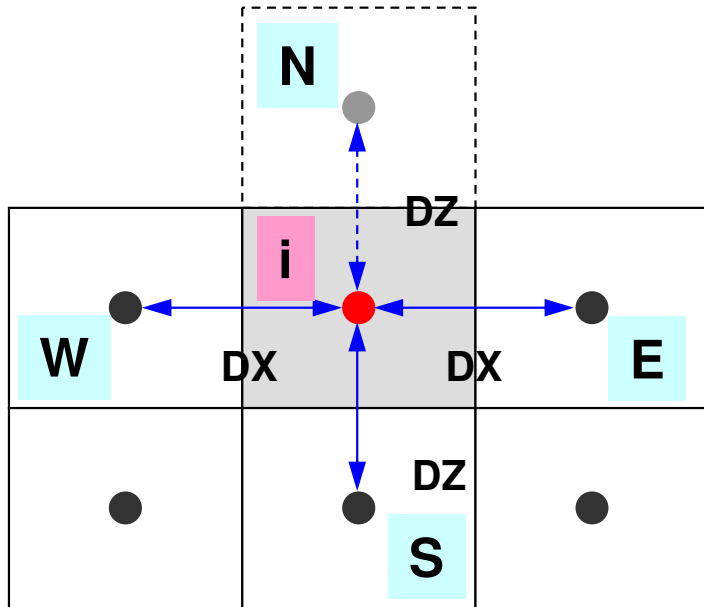
D (diagonal)

**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.



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

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

D (diagonal)

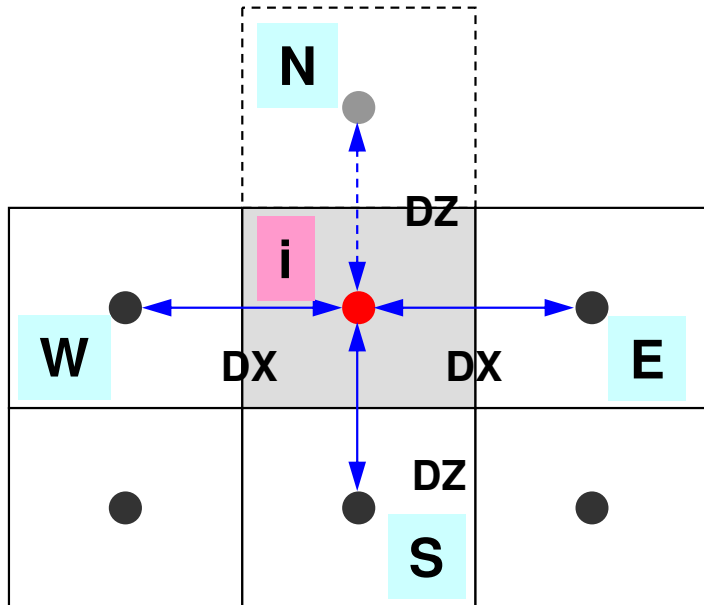
**AL, AU
(off-diag.)**

**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.



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

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

D (diagonal)

AL, AU
(off-diag.)

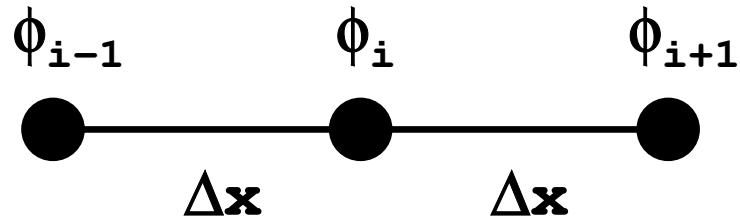
BFORCE
(RHS)

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

```
do ib= 1, ZmaxCELtot
  icel= ZmaxCEL(ib)
  coef= 2. d0 * RDZ * ZAREA
  D(icel)= D(icel) - coef
enddo
```

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

Taylor Series Expansion



$$\phi_{i+1} = \phi_i + \Delta x \left(\frac{\partial \phi}{\partial x} \right)_i + \frac{(\Delta x)^2}{2!} \left(\frac{\partial^2 \phi}{\partial x^2} \right)_i + \frac{(\Delta x)^3}{3!} \left(\frac{\partial^3 \phi}{\partial x^3} \right)_i \dots$$

$$\phi_{i-1} = \phi_i - \Delta x \left(\frac{\partial \phi}{\partial x} \right)_i + \frac{(\Delta x)^2}{2!} \left(\frac{\partial^2 \phi}{\partial x^2} \right)_i - \frac{(\Delta x)^3}{3!} \left(\frac{\partial^3 \phi}{\partial x^3} \right)_i \dots$$

$$\phi_{i-1} + \phi_{i+1} = 2\phi_i + 2 \times \frac{(\Delta x)^2}{2!} \left(\frac{\partial^2 \phi}{\partial x^2} \right)_i + 2 \times \frac{(\Delta x)^4}{4!} \left(\frac{\partial^4 \phi}{\partial x^4} \right)_i \dots$$

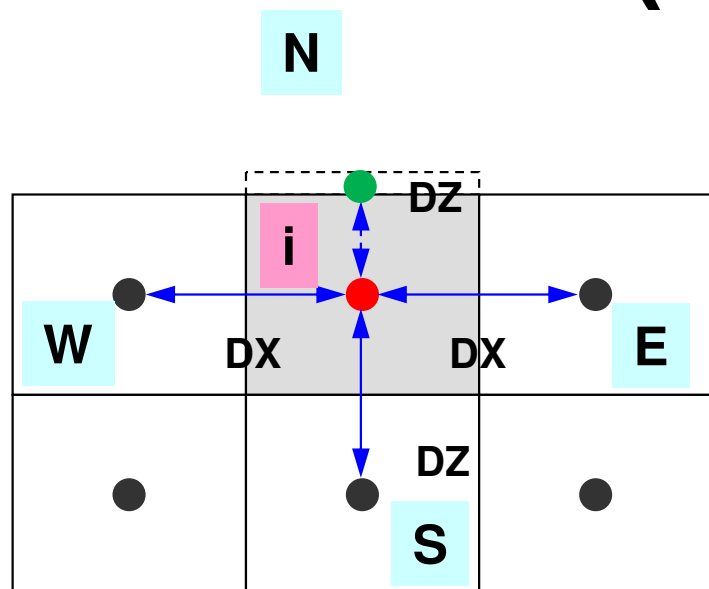
$$\frac{\phi_{i-1} - 2\phi_i + \phi_{i+1}}{(\Delta x)^2} = \left(\frac{\partial^2 \phi}{\partial x^2} \right)_i + \frac{(\Delta x)^2}{12} \left(\frac{\partial^4 \phi}{\partial x^4} \right)_i \dots$$

**Truncation Err.: 2nd Order
2nd Order Accuracy**

**If Δx is not uniform: 1st or
Lower Order Accuracy**

Dirichlet B.C. “N” is very thin ($=\varepsilon$)

1st order (or lower) Accuracy



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

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

D (diagonal)

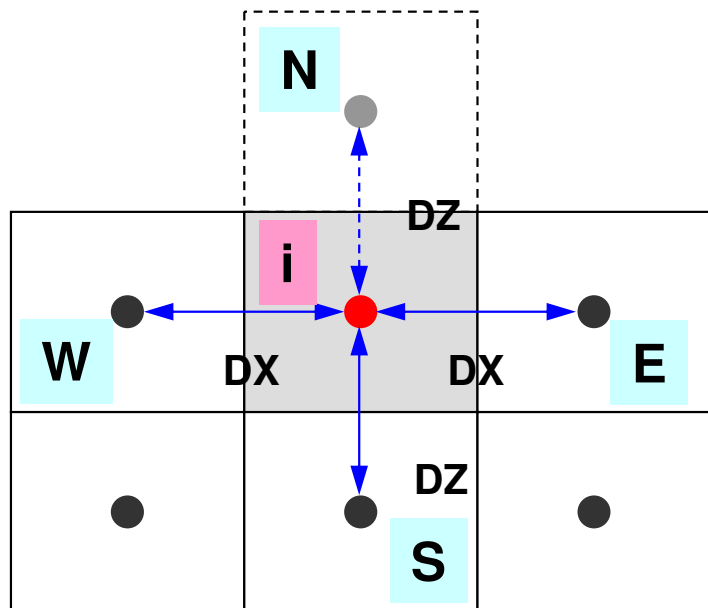
**AL, AU
(off-diag.)**

**BFORCE
(RHS)**

$$-\left[\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} \right] \phi_i + \left[\sum_k \frac{S_{ik}}{d_{ik} + d_{ki}} \phi_k \right] + \frac{\phi_N - \phi_i}{\left(\frac{\Delta z}{2} + \frac{\varepsilon}{2} \right)} \Delta x \Delta y = -V_i \dot{Q}_i, \quad \phi_N = 0, \quad \varepsilon \sim 0$$

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

Dirichlet B.C. using Mirror Image



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

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

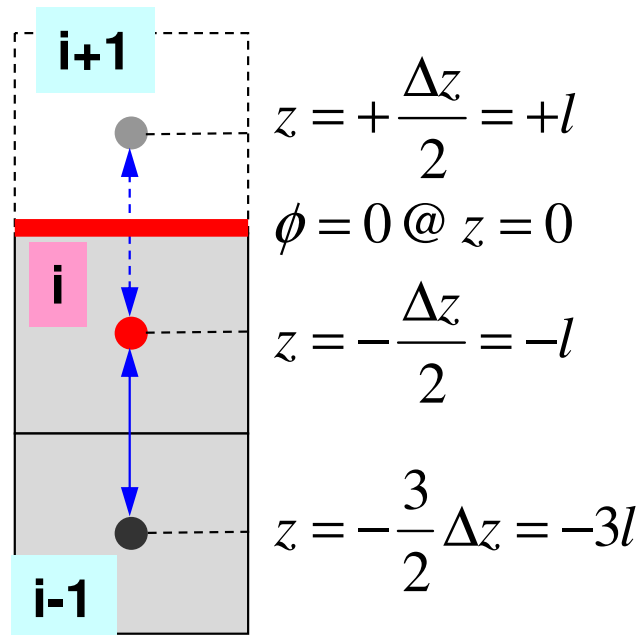
D (diagonal)

**AL, AU
(off-diag.)**

**BFORCE
(RHS)**

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

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



Higher Order Approximation for Dirichlet B.C. in 1D Problem

more complicated in 2D/3D cases

$$\phi = az^2 + bz + c$$

$$\phi(z = 0) = c = 0$$

$$\phi_i = al^2 - bl + c = al^2 - bl, \quad \phi_{i-1} = 9al^2 - 3bl + c = 9al^2 - 3bl$$

$$a = \frac{\phi_{i-1} - 3\phi_i}{6l^2}, \quad b = \frac{\phi_{i-1} - 9\phi_i}{6l} \Rightarrow \phi_{i+1} = al^2 + bl = \frac{1}{3}\phi_{i-1} - 2\phi_i$$

Structure of the Program

```
program MAIN
```

```
use STRUCT
use PCG
use solver_ICCG
use solver_ICCG2
use solver_PCG
```

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

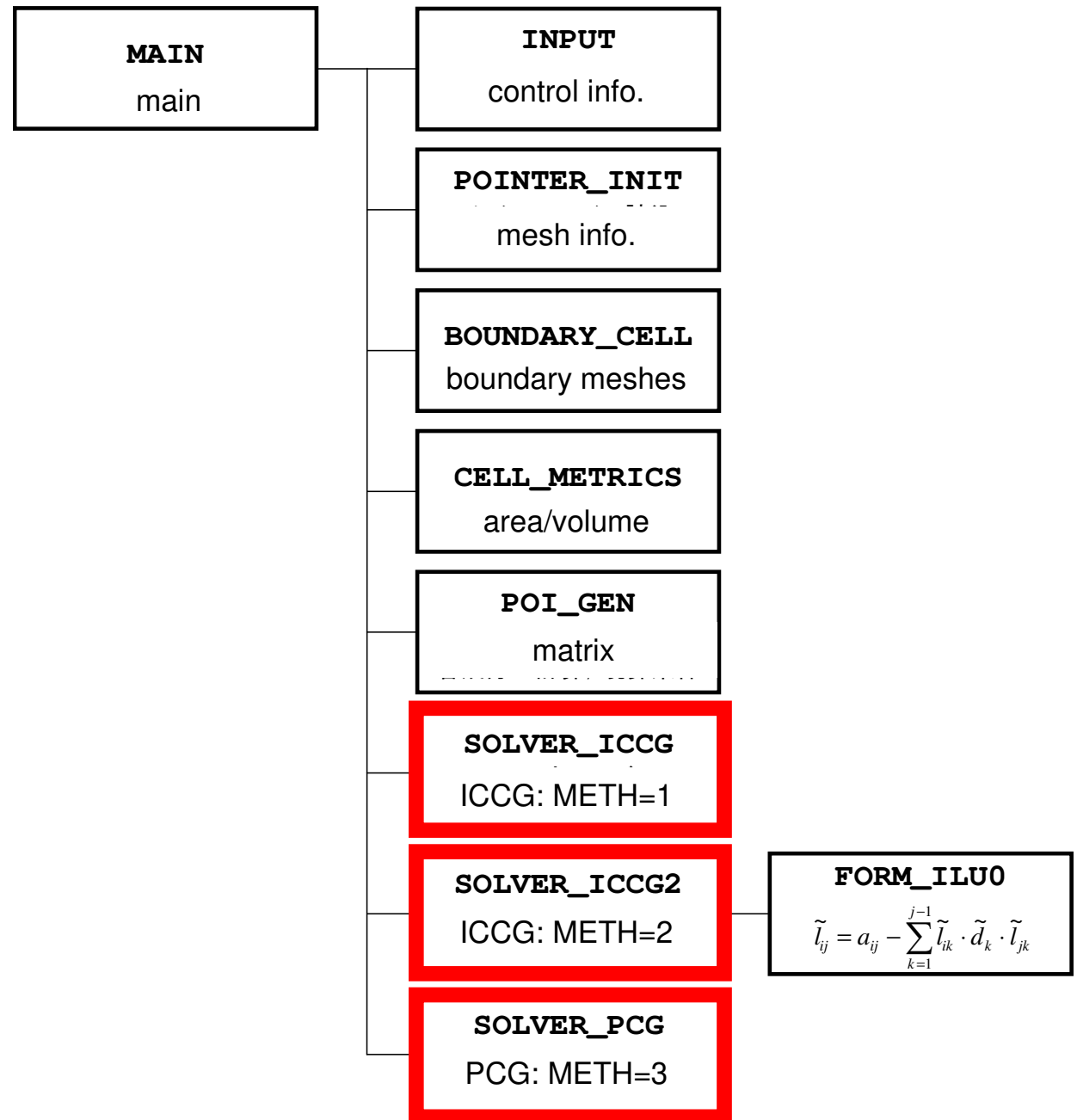
```
call INPUT
call POINTER_INIT
call BOUNDARY_CELL
call CELL_METRICS
call POI_GEN
```

```
PHI= 0. d0
```

```
if (METHOD.eq.1) call solve_ICCG (...)
if (METHOD.eq.2) call solve_ICCG2(...)
if (METHOD.eq.3) call solve_PCG  (...)
```

```
call OUTUCD
```

```
stop
end
```



- 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
 - $[A] = [L][L]^T \Rightarrow$ 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)

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

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

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

$$l_{ii} \cdot d_i = 1$$

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

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

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

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

$$\begin{aligned} & l_{ij} \cdot d_j \cdot l_{jj} + \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\ &= l_{ij} + \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} = a_{ij} \end{aligned}$$

$$\begin{aligned}
& \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}
\end{aligned}$$

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

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

$$\begin{aligned}
& a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk} \\
& = l_{ij} \cdot d_j \cdot l_{jj} = l_{ij}
\end{aligned}$$

Incomplete “Modified” Cholesky Factorization (NO Fill-in)

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

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

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

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

Actually, more “incomplete” factorization is applied in practical use.

Running the Program

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

32 32 32

1

1.00e-00 1.00e-00 1.00e-00

0.10 1.0e-08

NX/NY/NZ

MEHOD 1:2:3

DX/DY/DZ

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

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

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

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

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

Incomplete “Modified” Cholesky Factorization (NO Fill-in)

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

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

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

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

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

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

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

Only diagonal components are changed

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

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

$$\{z\} = (LDL^T)^{-1}\{r\} \longrightarrow \begin{array}{l} (L)\{y\} = \{r\} \\ (DL^T)\{z\} = \{y\} \end{array}$$

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

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

```

!C
!C +-----+
!C | {z}= [Minv]{r} |
!C +-----+
!C===

      do i= 1, N
        W(i, Y)= W(i, R)
      enddo

      do i= 1, N
        WVAL= W(i, Y)
        do k= indexL(i-1)+1, indexL(i)
          WVAL= WVAL - AL(k) * W(itemL(k), Y)
        enddo
        W(i, Y)= WVAL * W(i, DD)
      enddo

      do i= N, 1, -1
        SW = 0.0d0
        do k= indexU(i-1)+1, indexU(i)
          SW= SW + AU(k) * W(itemU(k), Z)
        enddo
        W(i, Z)= W(i, Y) - W(i, DD) * SW
      enddo

!C===

```

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

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

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

Forward Substitution for Incomplete Modified Cholesky Fact.

```

do i= 1, N
  W(i, Y)= W(i, R)
enddo

do i= 1, N
  WVAL= W(i, Y)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - AL(k) * W(itemL(k), Y)
  enddo
  W(i, Y)= WVAL * W(i, DD)
enddo

```

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

$$l_{11}y_1 = r_1$$

$$l_{21}y_1 + l_{22}y_2 = r_2$$

$$\vdots$$

$$l_{n1}y_1 + l_{n2}y_2 + \cdots + l_{nn}y_n = r_n$$

$$y_1 = r_1 / l_{11}$$

$$\vdots$$

$$y_2 = (r_2 - l_{21}y_1) / l_{22}$$



$$y_n = \left(r_n - l_{n1}y_1 - l_{n2}y_2 \cdots = r_n - \sum_{j=1}^{n-1} l_{nj}y_j \right) / l_{nn}$$

Forward Substitution for Incomplete Modified Cholesky Fact.

```

do i= 1, N
  W(i, Y)= W(i, R)
enddo

do i= 1, N
  WVAL= W(i, Y)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - AL(k) * W(itemL(k), Y)
  enddo
  W(i, Y)= WVAL * W(i, DD)
enddo

```

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

$$y_i = \left(r_i - \sum_{j=1}^{i-1} l_{ij} y_j \right) / l_{ii}$$

WVAL

Backward Substitution for Incomplete Modified Cholesky Fact.

```

do i= N, 1, -1
  SW = 0.0d0
  do k= indexU(i-1)+1, indexU(i)
    SW= SW + AU(k) * W(itemU(k),Z)
  enddo
  W(i,Z)= W(i,Y) - W(i,DD) * SW
enddo

```

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

$$z_n = y_n$$

$$z_{n-1} + (l_{n-1,n} / l_{n-1,n-1}) z_n = y_{n-1}$$

$$\vdots$$

$$z_1 + (l_{21} / l_{11}) z_2 + \cdots + (l_{n1} / l_{11}) z_n = y_1$$

$$z_n = y_n$$

$$z_{n-1} = y_{n-1} - (l_{n-1,n} z_n) / l_{n-1,n-1}$$

$$\vdots$$

$$z_1 = y_1 - \left(\sum_{j=2}^n l_{j1} z_j \right) / l_{11}$$



Backward Substitution for Incomplete Modified Cholesky Fact.

```

do i= N, 1, -1
  SW = 0.0d0
  do k= indexU(i-1)+1, indexU(i)
    SW= SW + AU(k) * W(itemU(k),Z)
  enddo
  W(i,Z)= W(i,Y) - W(i,DD) * SW
enddo

```

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

$$z_i = y_i - \left(\sum_{j=i+1}^n l_{ij} z_j \right) / l_{ii}$$

SW

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

```

!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 - AL(k) * W(itemL(k), Z)
        enddo
        W(i, Z)= WVAL * W(i, DD)
      enddo

      do i= N, 1, -1
        SW = 0.0d0
        do k= indexU(i-1)+1, indexU(i)
          SW= SW + AU(k) * W(itemU(k), Z)
        enddo
        W(i, Z)= W(i, Z) - W(i, DD) * SW
      enddo

!C===

```

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

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

$$W(i, DD) = 1/l_{ii} = d_i$$

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

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

solve_ICCG (1/7): METHOD= 1

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

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

$W(i, 1) = W(i, R) \Rightarrow \{r\}$

$W(i, 2) = W(i, Z) \Rightarrow \{z\}$

$W(i, 2) = W(i, Q) \Rightarrow \{q\}$

$W(i, 3) = W(i, P) \Rightarrow \{p\}$

$W(i, 4) = W(i, DD) \Rightarrow \{d\}$

solve_ICCG (2/7): METHOD= 1

```

!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

      do i= 1, N
        VAL= D(i)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
        enddo
        W(i, DD)= 1. d0/VAL
      enddo
!C===

```

$W(i, DD) = d_i$
 in incomplete modified
 Cholesky factorization

$W(i, DD):$ d_i
 $D(i):$ a_{ii}

 $itemL(j):$ k
 $AL(j):$ a_{ik}

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

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

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

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

Only diagonal
components are
changed

solve_ICCG (3/7): METHOD= 1

```

!C
!C +-----+
!C | {r0}= {b} - [A]{xini} |
!C +-----+
!C===
      do i= 1, N
        VAL= D(i)*X(i)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL + AL(k)*X(itemL(k))
        enddo
        do k= indexU(i-1)+1, indexU(i)
          VAL= VAL + AU(k)*X(itemU(k))
        enddo
        W(i,R)= B(i) - VAL
      enddo

      BNRM2= 0.0D0
      do i= 1, N
        BNRM2 = BNRM2 + B(i)  **2
      enddo
!C===

```

$$\text{BNRM2} = |\mathbf{b}|^2$$

Convergence criteria

Compute $\mathbf{r}^{(0)} = \mathbf{b} - [\mathbf{A}]\mathbf{x}^{(0)}$

```

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

```

solve_ICCG (4/7): METHOD= 1

```

!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 - AL(k) * W(itemL(k),Z)
          enddo
          W(i,Z)= WVAL * W(i,DD)
        enddo

        do i= N, 1, -1
          SW = 0.0d0
          do k= indexU(i-1)+1, indexU(i)
            SW= SW + AU(k) * W(itemU(k),Z)
          enddo
          W(i,Z)= W(i,Z) - W(i,DD)*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

```

solve_ICCG (4/7): METHOD= 1

```

!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
      (L){z}= {r}

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

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

```

前進代入
Forward Substitution

後退代入
Backward Substitution

solve_ICCG (5/7): METHOD= 1

```
!C
!C +-----+
!C | RHO= {r} {z} |
!C +-----+
!C===
      RHO= 0. d0
      do i= 1, N
        RHO= RHO + W(i, R)*W(i, Z)
      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
```

solve_ICCG (6/7): METHOD= 1

```

!C
!C +-----+
!C | {p} = {z} if      ITER=1 |
!C | BETA= RHO / RH01 otherwise |
!C +-----+
!C===
      if ( L.eq.1 ) then

        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
      else
        BETA= RHO / RH01
        do i= 1, N
          W(i,P)= W(i,Z) + BETA*W(i,P)
        enddo
      endif
!C===

!C
!C +-----+
!C | {q}= [A] {p} |
!C +-----+
!C===
      do i= 1, N
        VAL= D(i)*W(i,P)
        do k= indexL(i-1)+1, indexL(i)
          VAL= VAL + AL(k)*W(itemL(k),P)
        enddo
        do k= indexU(i-1)+1, indexU(i)
          VAL= VAL + AU(k)*W(itemU(k),P)
        enddo
        W(i,Q)= VAL
      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

```

solve_ICCG (7/7): METHOD= 1

```

!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C===
      C1= 0. d0
      do i= 1, N
        C1= C1 + W(i, P)*W(i, Q)
      enddo
      ALPHA= RHO / C1
!C===
!C
!C +-----+
!C | {x}= {x} + ALPHA*{p} |
!C | {r}= {r} - ALPHA*{q} |
!C +-----+
!C===
      do i= 1, N
        X(i) = X(i) + ALPHA * W(i, P)
        W(i, R)= W(i, R) - ALPHA * W(i, Q)
      enddo
      DNRM2= 0. d0
      do i= 1, N
        DNRM2= DNRM2 + W(i, R)**2
      enddo
!C===
      ERR = dsqrt(DNRM2/BNRM2)
      if (ERR .lt. EPS) then
        IER = 0
        goto 900
      else
        RH01 = RHO
      endif
    enddo
    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
     $\beta_{i-1} = \rho_{i-1} / \rho_{i-2}$ 
     $p^{(i)} = z^{(i-1)} + \beta_{i-1} p^{(i-1)}$ 
  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 (7/7): METHOD= 1

```
!C
!C +-----+
!C | ALPHA= RHO / {p} {q} |
!C +-----+
!C===
      C1= 0. d0
      do i= 1, N
        C1= C1 + W(i, P)*W(i, Q)
      enddo
      ALPHA= RHO / C1
```

```
!C===
!C
!C +-----+
!C | {x}= {x} + ALPH * {r} |
!C | {r}= {r} - ALPH * {A} {x} |
!C +-----+
!C===
```

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

```
      enddo
      DNRM2= 0. d0
      do i= 1, N
        DNRM2= DNRM2 + W(i, R)**2
      enddo
```

```
!C===
      ERR = dsqrt(DNRM2/BNRM2)
      if (ERR .lt. EPS) then
        IER = 0
        goto 900
      else
        RHO1 = RHO
      endif
    enddo
    IER = 1
```

$|r|, |b| : 2 / L2 / Euclidean - norm \quad (\|r\|_2, \|b\|_2)$

$$r = b - [A]x$$

$$DNRM2 = |r|^2$$

$$BNRM2 = |b|^2$$

$$ERR = |r| / |b|$$

$$ERR = \sqrt{\frac{DNorm2}{BNorm2}} = \frac{|r|}{|b|} = \frac{|b - Ax|}{|b|} \leq Eps$$

Compute $r^{(0)} = b - [A]x^{(0)}$

for $i = 1, 2, \dots$

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

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

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

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

$$Ax = b \Rightarrow \alpha Ax = \alpha b$$

$$r = b - Ax \Rightarrow R = \alpha b - \alpha Ax = \alpha r$$

solve_ICCG2 (1/3): METHOD= 2

Fortran ONLY

```
!C
!C***
!C*** module solver_ICCG2
!C***
!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)
!C
      implicit REAL*8 (A-H, O-Z)
!C
      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
!C
      integer, dimension(0:N)         :: indexL, indexU
      integer, dimension(NPL)         :: itemL
      integer, dimension(NPU)         :: itemU
!C
      real(kind=8), dimension(:, :), allocatable :: W
!C
      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===
```

D1u0, AL1u0, AU1u0:

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
!C  subroutine FORM_ILU0
!C    implicit REAL*8 (A-H, O-Z)
!C    integer, dimension(:), allocatable :: IW1 , IW2
!C    integer, dimension(:), allocatable :: IWsL, IWsU
!C    real (kind=8):: RHS_Aij, DkINV, Aik, Akj
!C
!C  +-----+
!C  |  INIT.  |
!C  +-----+
!C===
!C    allocate (ALlu0(NPL), AUlu0(NPU))
!C    allocate (Dlu0(N))
!C
!C    do i= 1, N
!C      Dlu0(i)= D(i)
!C      do k= 1, INL(i)
!C        ALlu0(k, i)= AL(k, i)
!C      enddo
!C
!C      do k= 1, INU(i)
!C        AUlu0(k, i)= AU(k, i)
!C      enddo
!C    enddo
!C===
```

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

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= 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
          enddo
        enddo
      enddo
!C===
```

```
      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(i)
          IW2(itemU(k0))= k0
        enddo

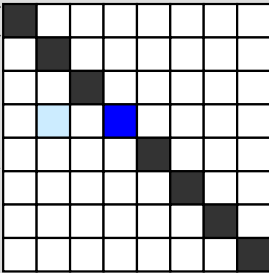
        do icon= indexL(i-1)+1, indexL(i)
          k= itemL(icon)
          D11= Dlu0(k)
          DkINV= 1. d0/D11
          Aik= ALlu0(icon)

          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
          enddo
        enddo
      enddo
```



```
      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

  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===
```

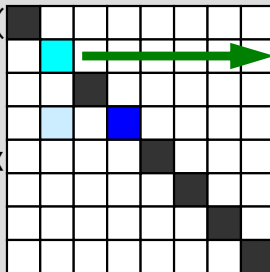
```
allocate (IW1(N) , IW2(N))
IW1=0
IW2= 0
```

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

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

```
do icon= indexL(i-1)+1, indexL(i)
  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

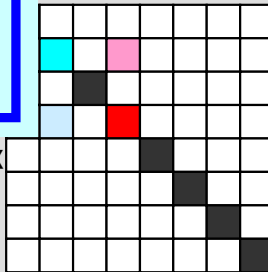
```
!C
!C +-----+
!C | ILU(0) factorization |
!C +-----+
!C===
```

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

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

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

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



```
do icon= indexL(i-1)+1, indexL(i)
  k= itemL(icon)
  D11= Dlu0(k)
```

```
DkINV= 1. d0/D11
Aik= ALu0(icon)
```

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

```
if (j.eq.i) then
  Akj= Aul0(kcon)
  RHS_Aij= Aik * DkINV * Akj
  Dlu0(i)= Dlu0(i) - RHS_Aij
endif
```

j=i

```
if (j.lt.i .and. IW1(j).ne.0) then
  Akj= Aul0(kcon)
  RHS_Aij= Aik * DkINV * Akj
```

```
  ij0 = IW1(j)
  ALu0(ij0)= ALu0(ij0) - RHS_Aij
endif
```

```
if (j.gt.i .and. IW2(j).ne.0) then
  Akj= Aul0(kcon)
  RHS_Aij= Aik * DkINV * Akj

  ij0 = IW2(j)
  Aul0(ij0)= Aul0(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
  endif
enddo
```

FORM_ILU0 (2/2): Fortran only

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

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

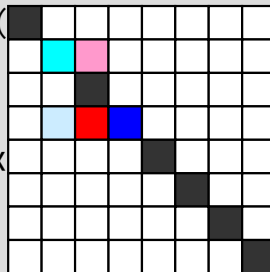
$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

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

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

```
do icon= indexL(i-1)+1, indexL(i)
  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
```

j < i

```
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

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

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

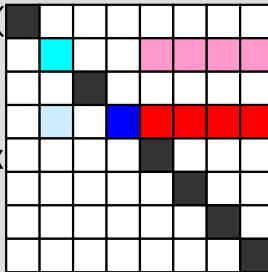
$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

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

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

```
do icon= indexL(i-1)+1, indexL(i)
  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
```

j>i

```
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

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

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

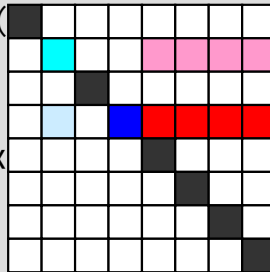
$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} \cdot d_k \cdot l_{jk}$$

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

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

```
do icon= indexL(i-1)+1, indexL(i)
  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
```

j < i

```
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
```

j > i

```
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 / D11(i)
enddo
deallocate (IW1, IW2)
```

!C==

```
end subroutine FORM_ILU0
```

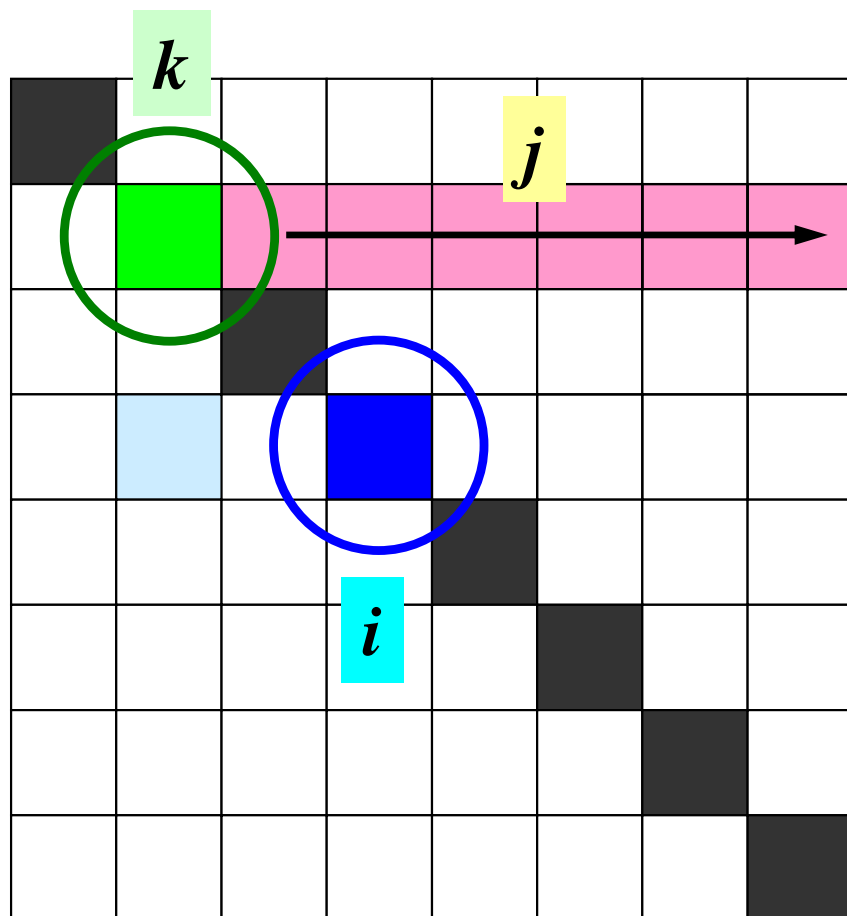
These if –then clauses never applied, therefore:

ALlu0= AL

AUlu0= AU

Dlu0 = W(i, DD)

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



- : Mesh i
- ○: Mesh k (Lower Triangular Component of i)
- : Mesh j (Upper Triangular Component of k)

if $j=i$ Dlu(■) is updated

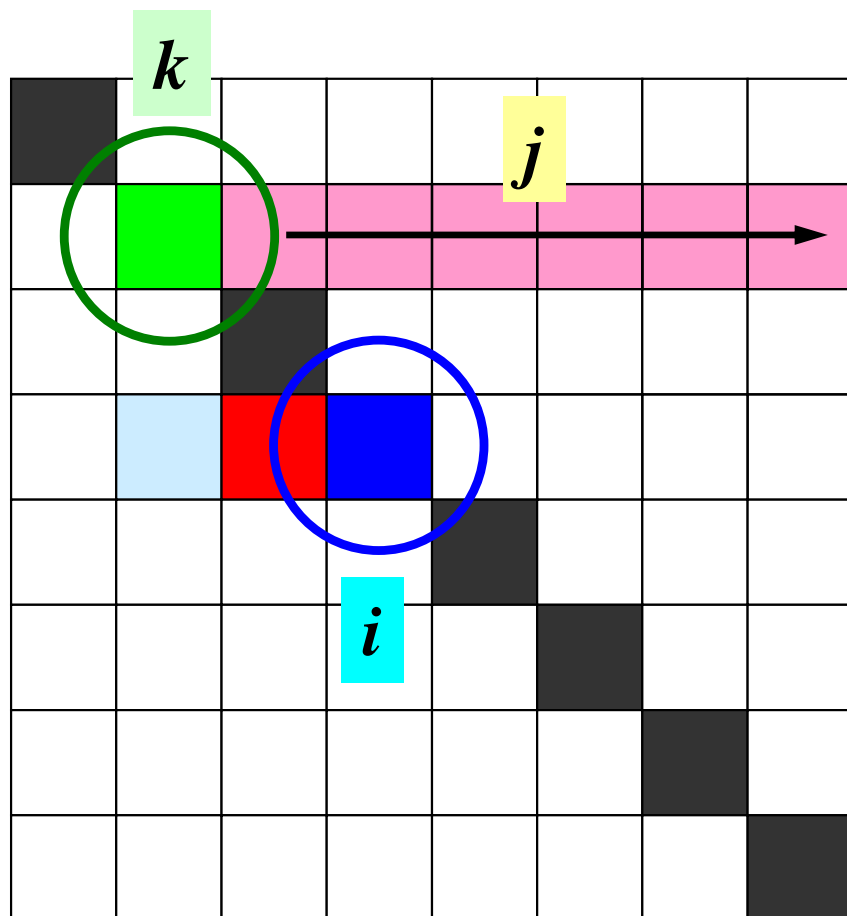
$$i = 1, 2, \dots, n$$

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

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

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

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



- : Mesh i
- ○: Mesh k (Lower Triangular Component of i)
- : Mesh j (Upper Triangular Component of k)

if $j < i$ ALu0($i-j$)(■) is updated

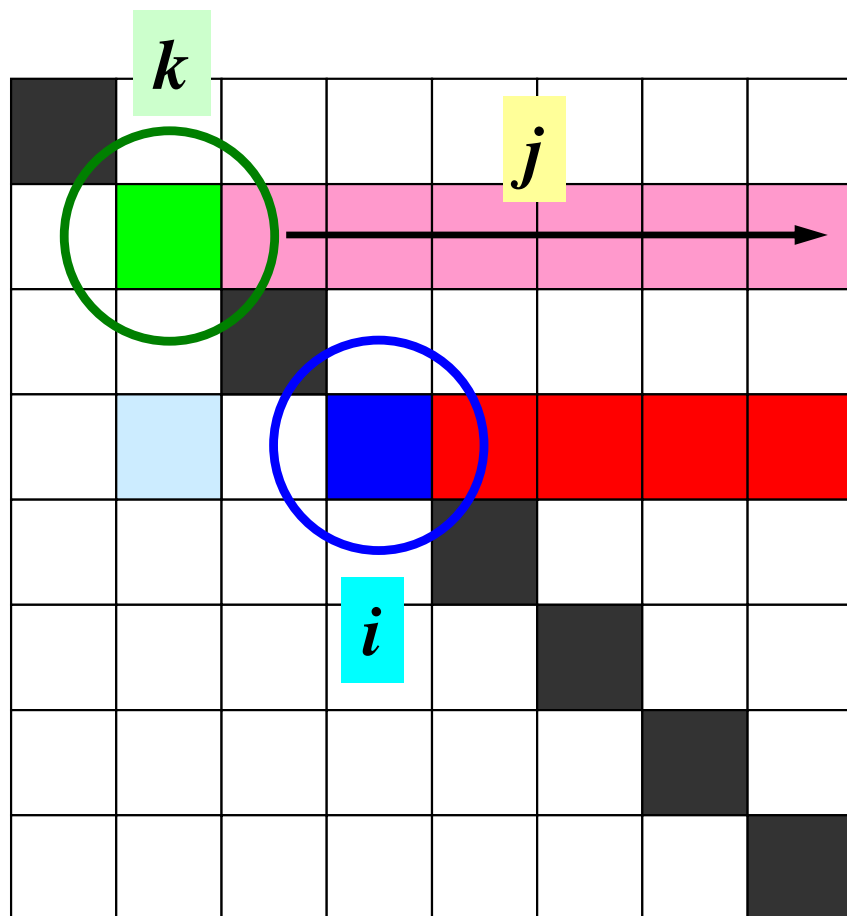
$$i = 1, 2, \dots, n$$

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

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

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

$j=i, j<i, j>i$ (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:

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

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

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

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

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

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, indexU(i)
            SW= SW + AUlu0(k) * W(itemU(k),Z)
          enddo
          W(i,Z)= W(i,Z) - Dlu0(i)*SW
        enddo
!C===
```

Compute $r^{(0)} = b - [A]x^{(0)}$
 for $i = 1, 2, \dots$
 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, indexU(i)
          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

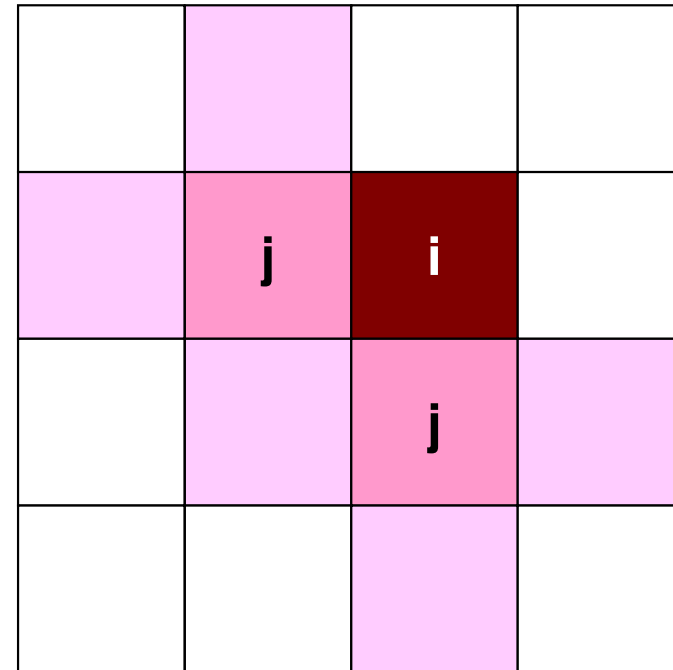
$$(DL^T)\{z\} = \{z\}$$

Backward Substitution

ALlu0=AL, AUlu0=AU, Dlu0=W(i,DD): Therefore, iterations for convergence for METHOD=1, and those for METHOD=2 are same.

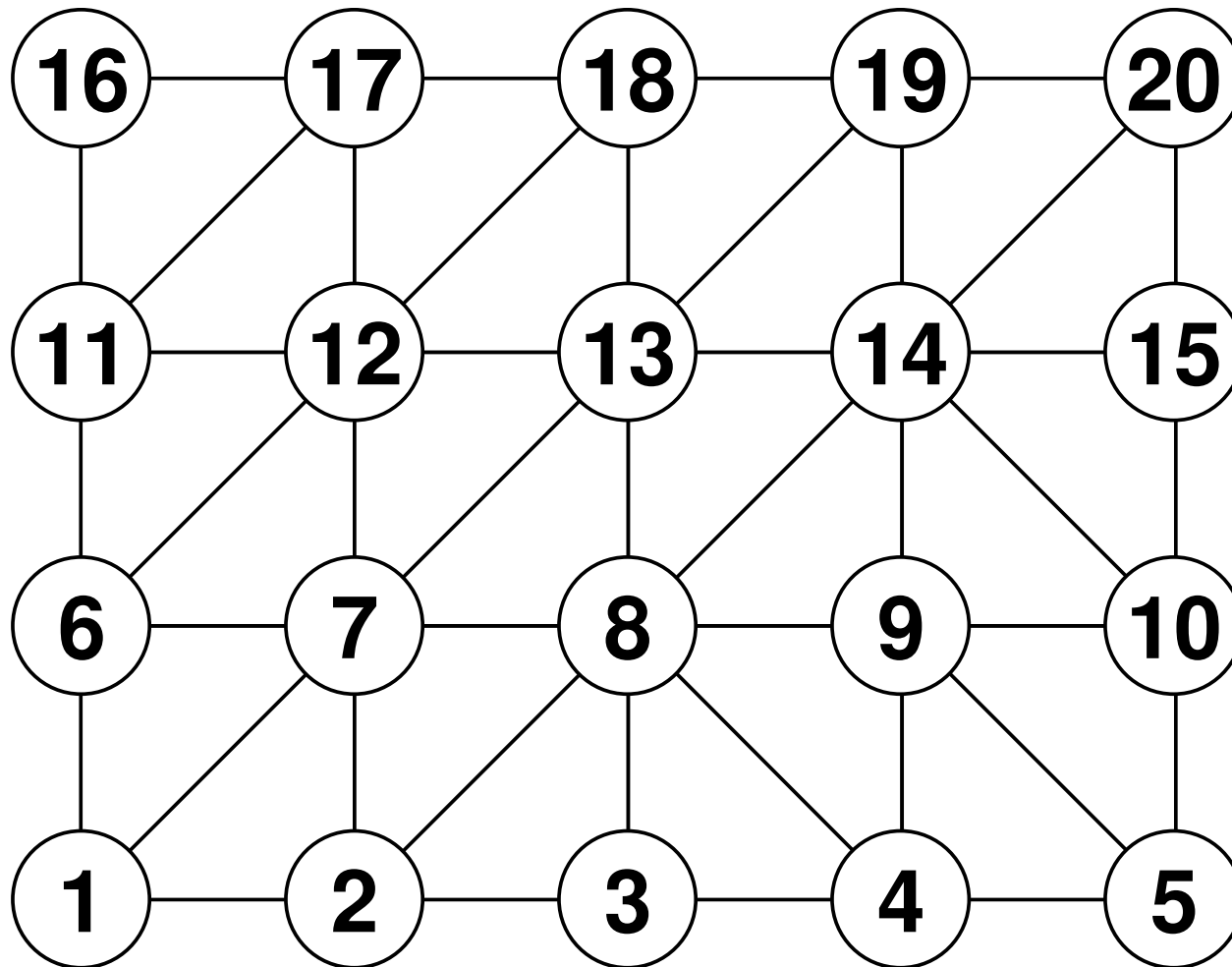
Incomplete Modified Cholesky Fact. Structured Meshes

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



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

In this case, ALU0/AU0 could be changed



Parallelize ICCG Method

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning

How about the preconditioning ?

Point Jacobi is easy: but slow

```
do i= 1, N
  W(i, Z) = W(i, R) * W(i, DD)
enddo
```

```
!$omp parallel do private(i)
  do i = 1, N
    W(i, Z) = W(i, R) * W(i, DD)
  enddo
!$omp end parallel do
```

```
!$omp parallel do private(ip, i)
  do ip= 1, PEsmptOT
    do i = INDEX(ip-1)+1, INDEX(ip)
      W(i, Z) = W(i, R) * W(i, DD)
    enddo
  enddo
!$omp end parallel do
```

64*64*64

METHOD= 1

1	6.543963E+00
101	1.748392E-05
146	9.731945E-09

real 0m14.662s

METHOD= 3

1	6.299987E+00
101	1.298539E+00
201	2.725948E-02
301	3.664216E-05
401	2.146428E-08
413	9.621688E-09

real 0m19.660s

How about the preconditioning ?

IC Factorization

```

do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
  enddo
  W(i, DD)= 1. d0/VAL
enddo

```

Forward Substitution

```

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

```

Data Dependency

Conflict of reading from/writing to memory
Difficult to be parallelized

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k), DD)
  enddo
  W(i, DD)= 1. d0/VAL
enddo
```

Forward Substitution

```
do i= 1, N
  WVAL= W(i, Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - AL(k) * W(itemL(k), Z)
  enddo
  W(i, Z)= WVAL * W(i, DD)
enddo
```

Matrix-Vector Multiply: Easy to be Parallelized $\{q\}=[A]\{p\}$

$\{q\}$: LHS: Updated

$\{p\}$: RHS: No change

```

!$omp parallel do private(i, VAL, k)
do i = 1, N
    VAL= D(i)*W(i, P)
    do k= indexL(i-1)+1, indexL(i)
        VAL= VAL + AL(k)*W(itemL(k), P)
    enddo
    do k= indexU(i-1)+1, indexU(i)
        VAL= VAL + AU(k)*W(itemU(k), P)
    enddo
    W(i, Q)= VAL
enddo
!$omp end parallel do

```

IC Factorization

Four Thread Parallel Operation

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

```
do i= 1, N
  WVAL= W(i, Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL - AL(k) * W(itemL(k), Z)
  enddo
  W(i, Z)= WVAL * W(i, DD)
enddo
```

IC Factorization

Four Thread Parallel Operation

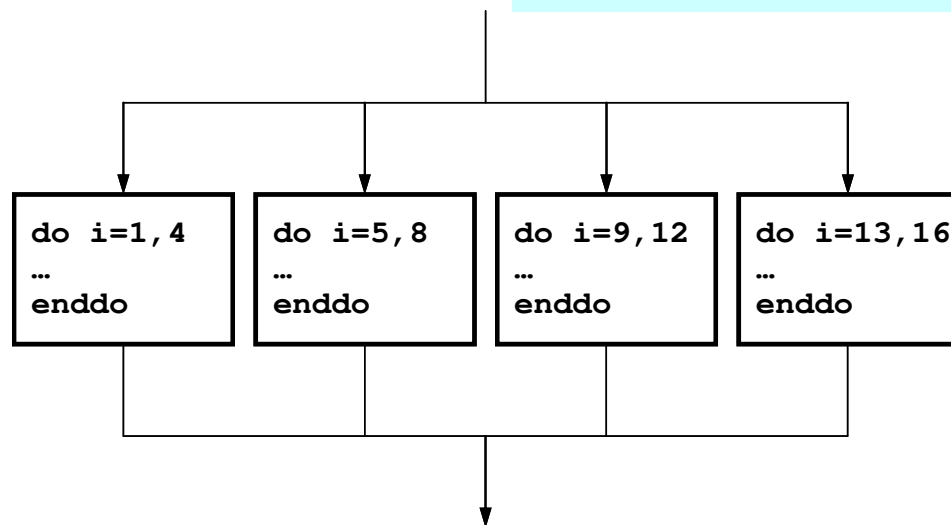
13	14	15	16
9	10	11	12
5	6	7	8
1	2	3	4

```

!$omp parallel do private (ip,i,k,VAL)
  do ip= 1, 4
    do i= INDEX(ip-1)+1, INDEX(ip)
      WVAL= W(i,Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k),Z)
      enddo
      W(i,Z)= WVAL * W(i,DD)
    enddo
  enddo
!$omp parallel enddo
  
```

```

INDEX(0)= 0
INDEX(1)= 4
INDEX(2)= 8
INDEX(3)=12
INDEX(4)=16
  
```



These four threads are executed simultaneously.

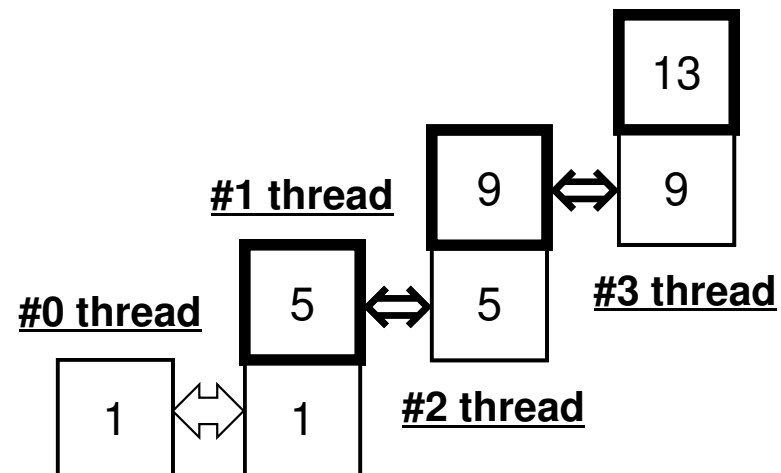
Data Dependency: Conflict of reading from/writing to memory

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

```

!$omp parallel do private (ip, l, k, VAL)
  do ip= 1, 4
    do i= INDEX(ip-1)+1, INDEX(ip)
      WVAL= W(i, Z)
      do k= indexL(i-1)+1, indexL(i)
        WVAL= WVAL - AL(k) * W(itemL(k), Z)
      enddo
      W(i, Z)= WVAL * W(i, DD)
    enddo
  enddo
!$omp parallel enddo
  
```

INDEX(0)= 0
 INDEX(1)= 4
 INDEX(2)= 8
 INDEX(3)=12
 INDEX(4)=16



$\Leftrightarrow$:
 Data Dependency

Parallelize ICCG Method

- Dot Product: OK
- DAXPY: OK
- Matrix-Vector Multiply: OK
- Preconditioning: **Something special needed !**
 - Just inserting OpenMP directive is not enough