Parallel Programming for Multicore Processors using OpenMP

Part I: FVM Code, Introduction to OpenMP

Kengo Nakajima
Information Technology Center

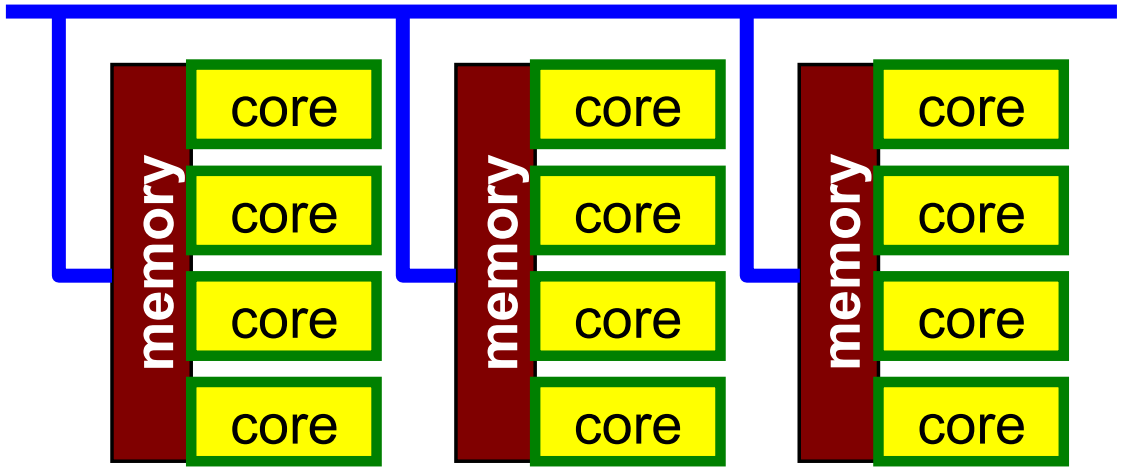
Programming for Parallel Computing (616-2057)
Seminar on Advanced Computing (616-4009)

Background

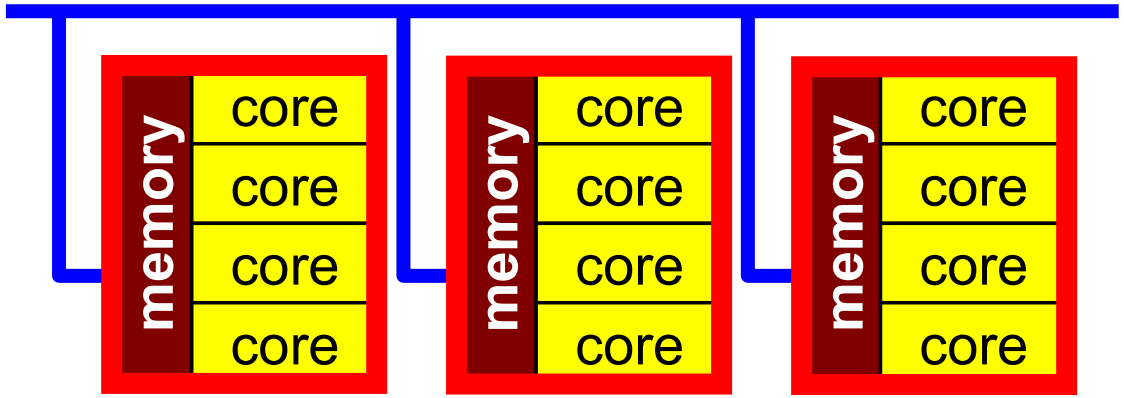
- Multicore/Manycore Processors
  - Low power consumption, Various types of programming models
- OpenMP
  - Directive based, (seems to be) easy
  - Many books
- Data Dependency
  - Conflict of reading from/writing to memory
  - Appropriate reordering of data is needed for “consistent” parallel computing
  - NO detailed information in OpenMP books: very complicated
- OpenMP/MPI Hybrid Parallel Programming Model for Multicore/Manycore Clusters

Flat MPI vs. Hybrid

Flat-MPI：Each PE -> Independent



Hybrid：Hierarchal Structure



Hybrid Parallel Programming Model is essential for Post-Peta/Exascale Computing

- Message Passing (e.g. MPI) + Multi Threading (e.g. OpenMP, CUDA, OpenCL, OpenACC etc.)
- In K computer and FX10, hybrid parallel programming is recommended
  - MPI + Automatic Parallelization by Fujitsu's Compiler
- Expectations for Hybrid
  - Number of MPI processes (and sub-domains) to be reduced
  - $O(10^8\text{-}10^9)$-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

Target of this Part

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

Files on ECCS 2012

```
>$ cd <$E-TOP>

>$ cp /home03/skengon/Documents/class_eps/F/multicore.tar .
>$ cp /home03/skengon/Documents/class_eps/C/multicore.tar .

>$ tar xvf multicore.tar
>$ cd multicore
```

Please confirm that following directories are created:

```
       L1 L2

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

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

Target of this Part

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

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

Target Application

- 3D Poisson Equations

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

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

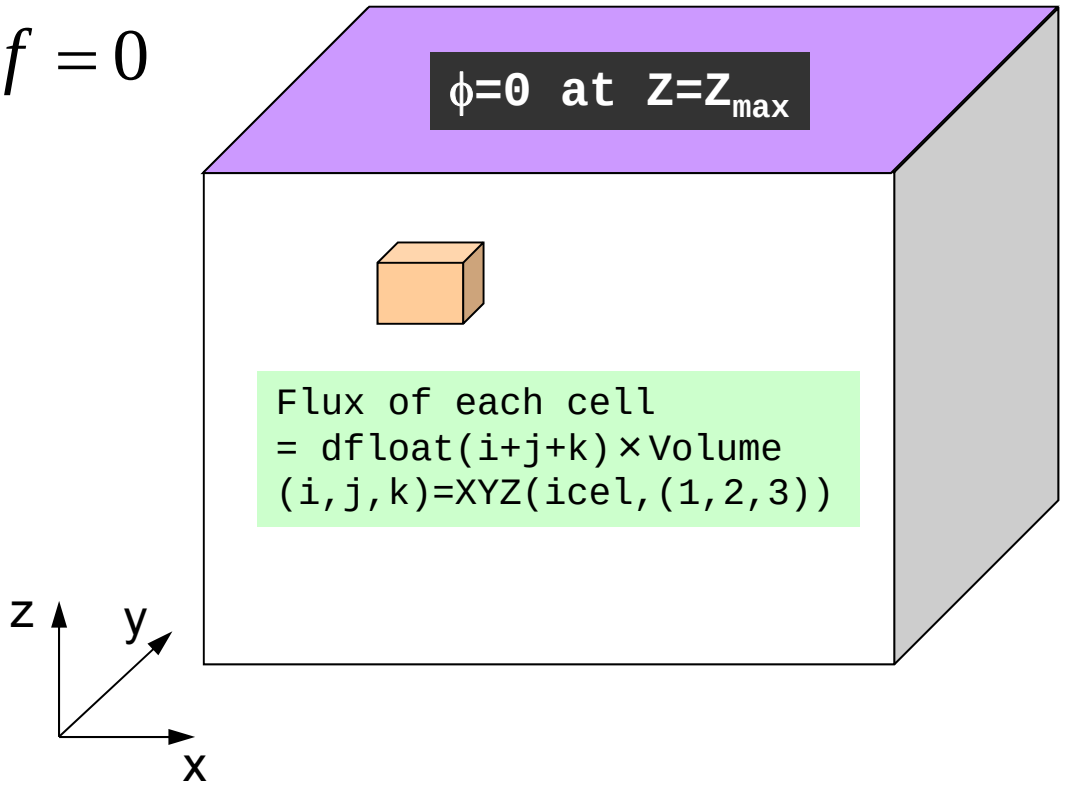
Target Problem: Variables are defined at cell-center'

Poisson Equation

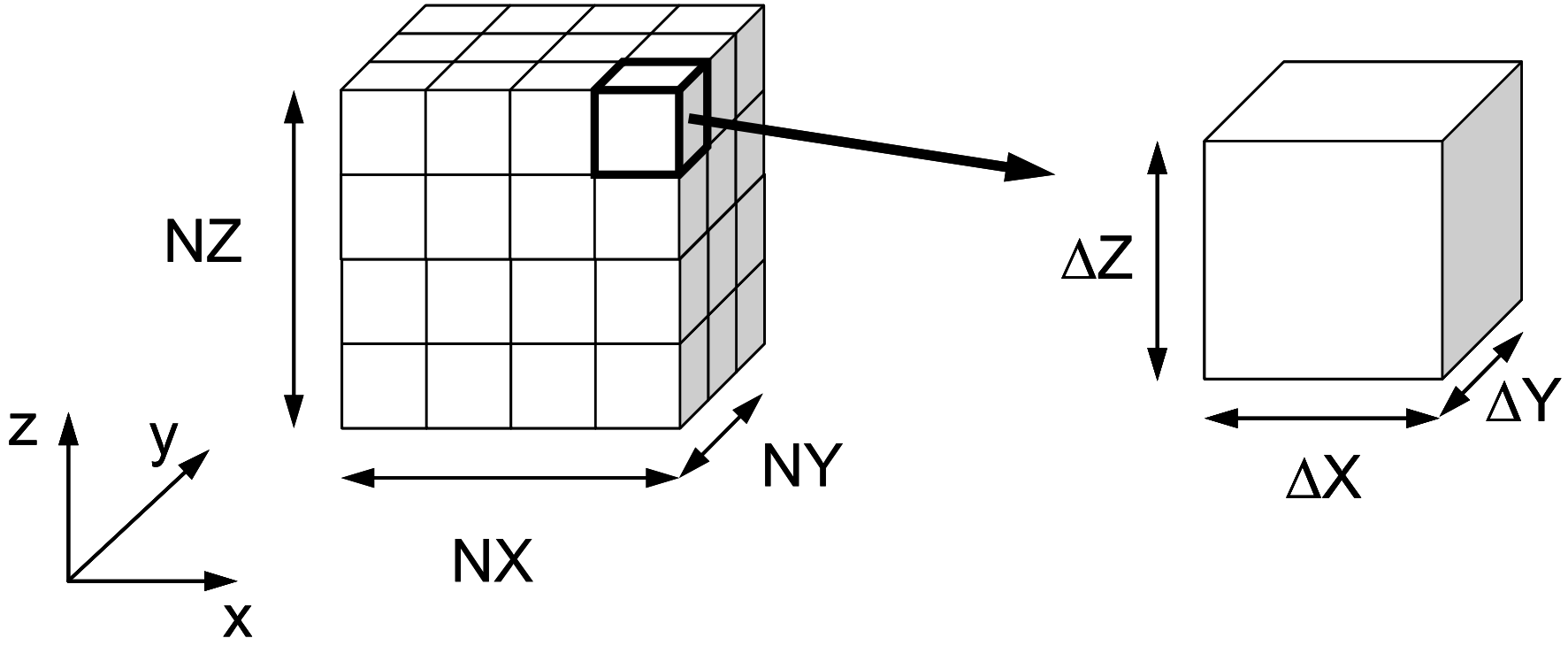
$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} + f = 0$$

B.C.

- Volume Flux
- ϕ=0@Z=Z_{max}



3D Structured Mesh
Internal data structure is “unstructured”



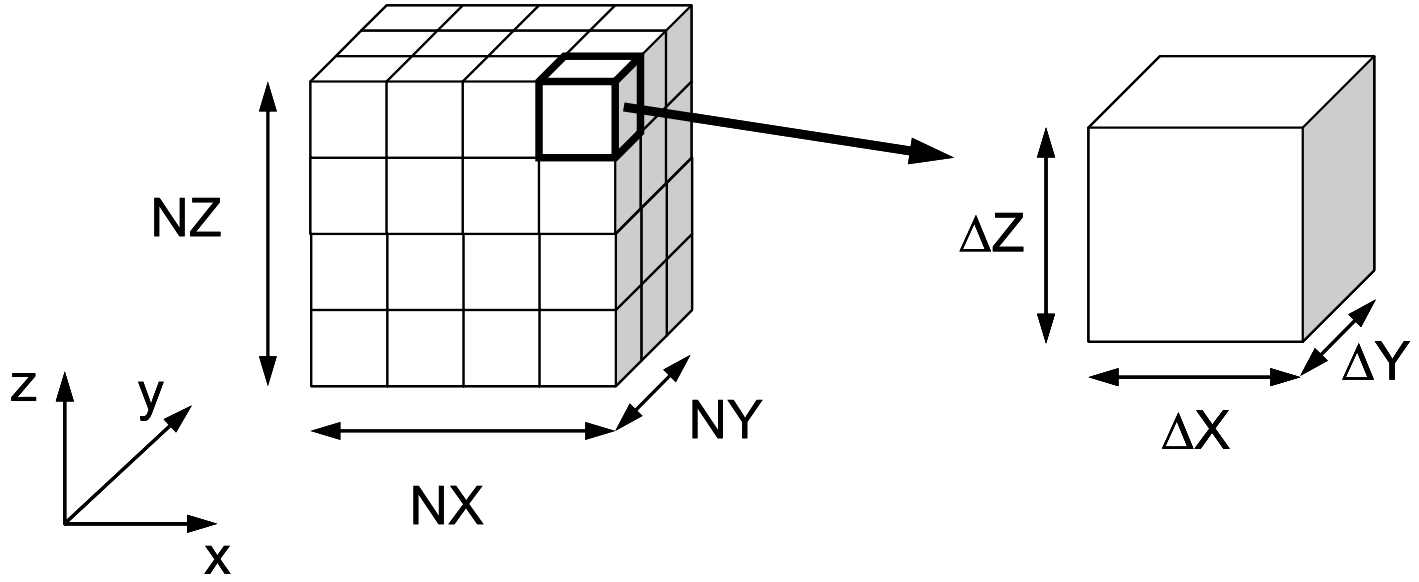
Volume Flux f

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

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

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

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



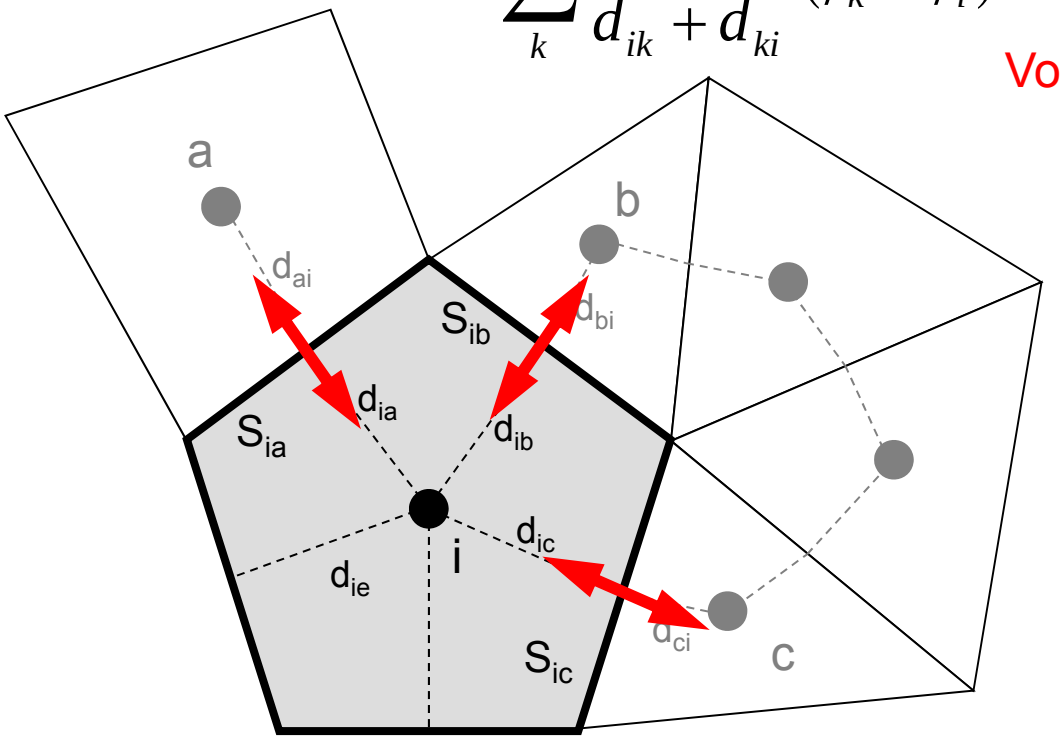
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

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

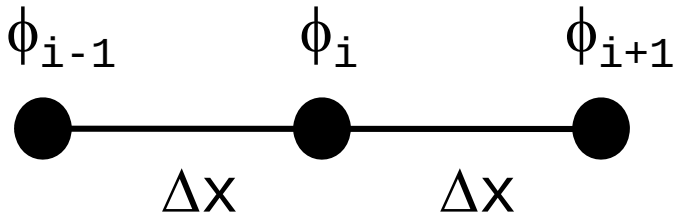
Volume Flux



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

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

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



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

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

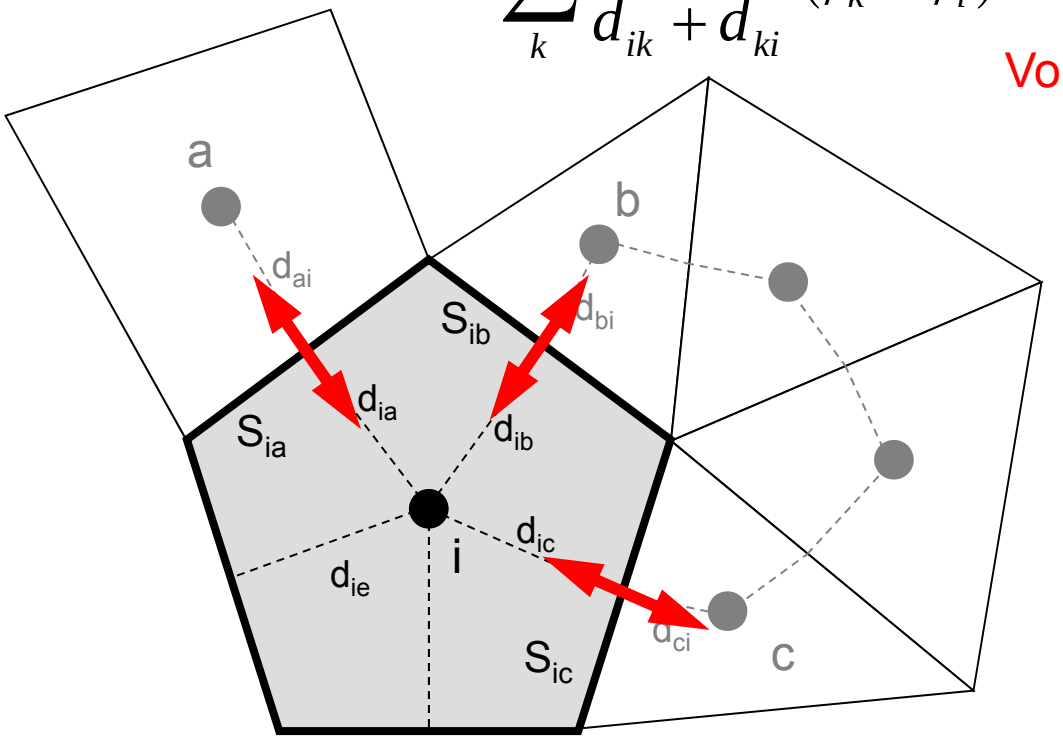
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

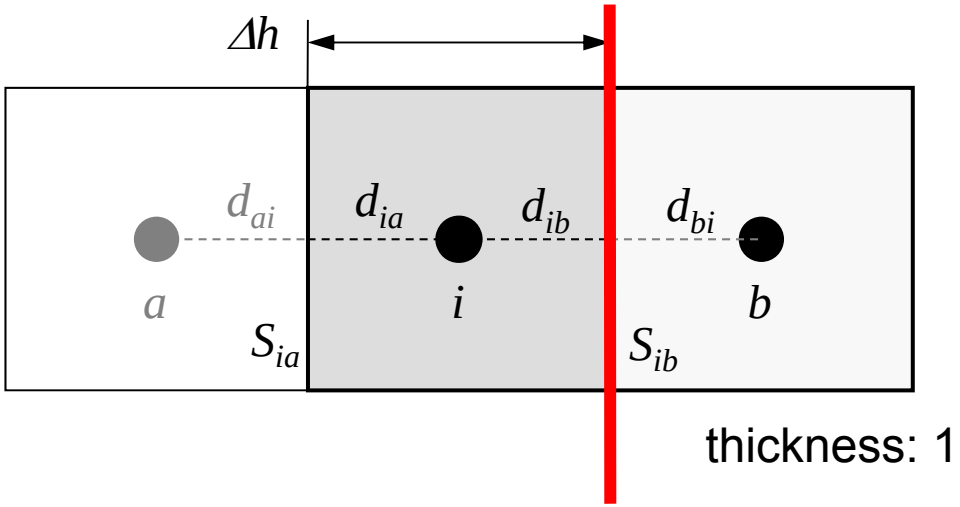
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



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

Comparison with 1D FDM (1/3)



Δh x Δh Square Mesh

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

Volume: $V_i = \Delta h^2$

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

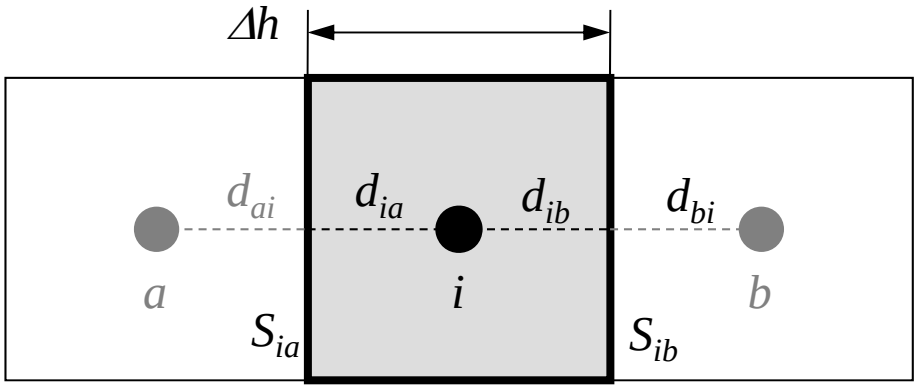
Flux through this surface: Qs_{ib}

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

Fourier's Law

Flux through a surface
= - (gradient of potential)

Comparison with 1D FDM (2/3)



Δh x Δh Square Mesh

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

Volume: $V_i = \Delta h^2$

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

thickness: 1

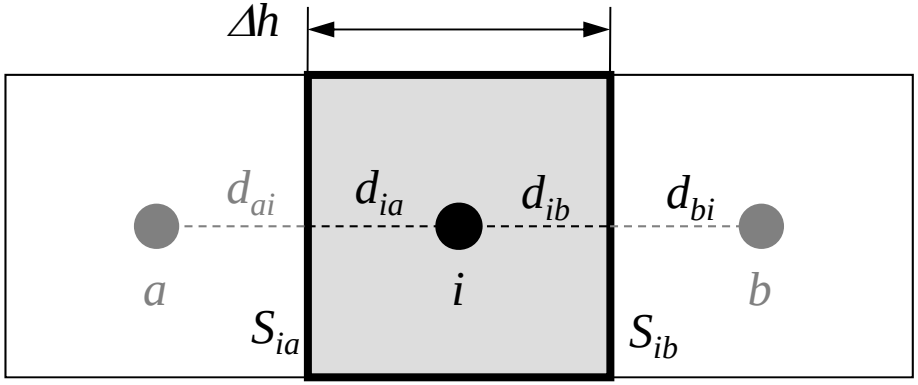
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k - \phi_i\right) + V_i \dot{Q}_i = 0$$

Divided by V_i:

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

considering this part

Comparison with 1D FDM (3/3)



Δh x Δh Square Mesh

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

Volume: $V_i = \Delta h^2$

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

thickness: 1

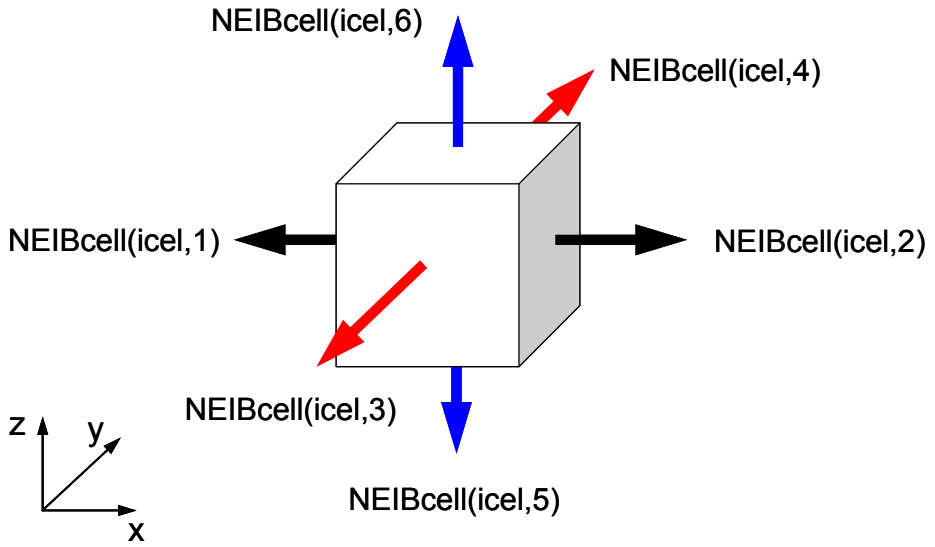
$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i) = \frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i)$$

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

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

for i-th cell
linear equations

in 3D



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

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

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

Linear Equations

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

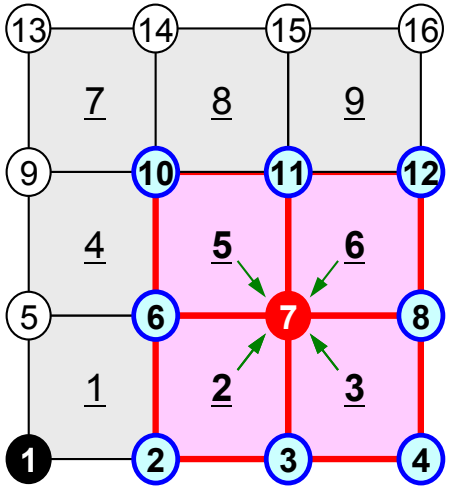
$$\sum_k \frac{S_{icel-k}}{d_{icel-k}}\left(\phi_k-\phi_{icel}\right)=-f_{icel}V_i$$

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

Diagonal Off-Diagonal

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

Sparse Coef. Matrix in FEM



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

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

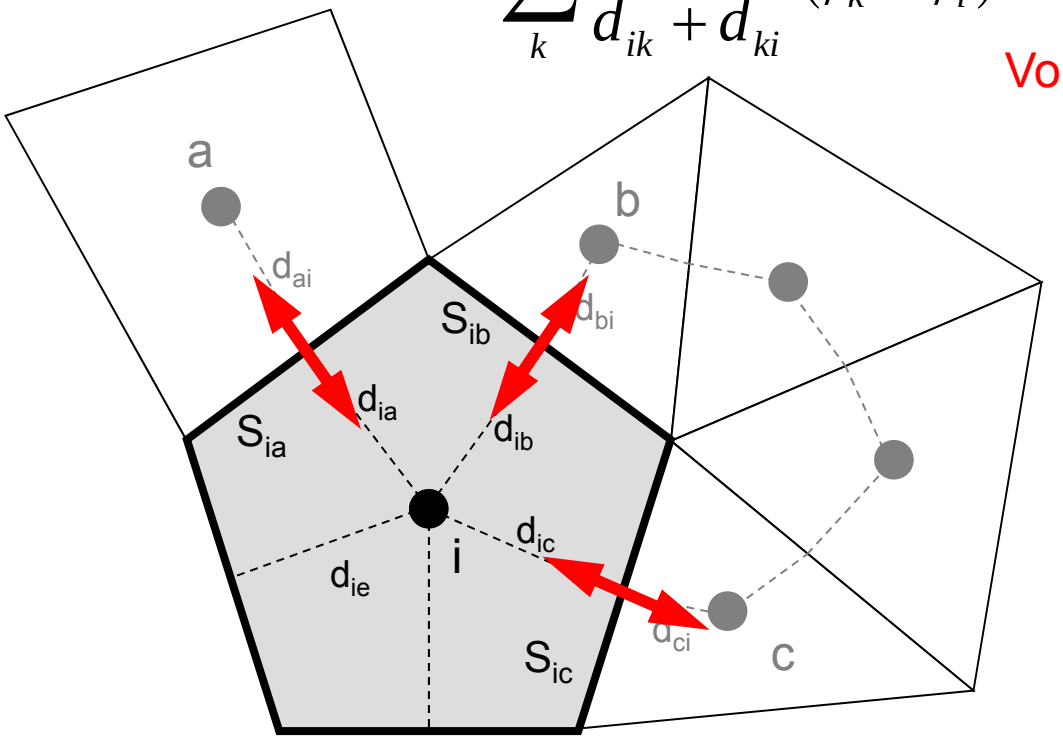
Coef. Matrices for FVM are sparse

Only neighboring cells are considered

Diffusion:
Interaction with Neighbors

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

Volume Flux



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

CRS (Compressed Row Storage)

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

do i= 1, N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    kk= item(k)
    Y(i)= Y(i) + AMAT(k)*X(kk)
  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 **Ax=b** is the most important and **expensive** part of various types of scientific computing.
  - for both linear and nonlinear applications
- Various types of methods proposed & developed.
  - for dense and sparse matrices
  - classified into ***direct*** and ***iterative*** methods
- Dense Matrices:密行列: Globally Coupled Problems
  - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices:疎行列: Locally Defined Problems
  - FEM, FVM, FDM, DEM, MD (solid), BEM w/FMM

Direct Method：直接法

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

Good

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

Bad

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

Iterative Method: 反復法

- Stationary Method
  - SOR, Gauss-Seidel, Jacobi
  - Generally slow, impractical

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

Iterative Method: 反復法 (cont.)

Good

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

Bad

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

Conjugate Gradient Method
共役勾配法

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

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

Procedures of Conjugate Gradient

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

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

$x^{(i)}$: Vector
α_i : Scalar

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.
- A preconditioner **M** (whose properties are similar to those of **A**)transforms the linear system into one with more favorable spectral properties
  - In "ill-conditioned" problems, "condition number" (ratio of max/min eigenvalue if **A** is symmetric) is large.
  - **M** transforms original equation **Ax=b** into **A'x=b'** where $\mathbf{A'=M^{-1}A}$, $\mathbf{b'=M^{-1}b}$
  - If **M~A**, $\mathbf{M^{-1}A}$ is close to identity matrix.
  - If $\mathbf{M^{-1}=A^{-1}}$, this is the best preconditioner (a.k.a. Gaussian Elimination)

Preconditioned Conjugate Gradient Method: PCG

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

What to be done:

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

Inverse Approximation:

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

Real Inverse: Ultimate Precond.

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

Diag. Scaling/Point Jacobi: Simple, Weak

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

Diagonal Scaling, Point-Jacobi

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

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

ILU(0), IC(0)

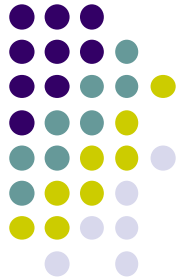
- Widely used Preconditioners for Sparse Matrices
  - Incomplete LU Factorization
  - Incomplete Cholesky Factorization (for Symmetric Matrices)

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

Sample Programs for ILU only Fotran

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

>$ g95 lu1.f -o lu1
>$ g95 lu2.f -o lu2
```

Full LU Factorization (or LU Decomposition)



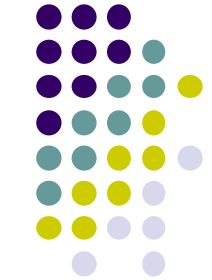
- Direct Method
  - A^{-1} is calculated directly
  - A^{-1} can be saved
  - Fill-in's
- LU factorization

Incomplete LU Factorization (ILU)

- ILU factorization
  - Incomplete LU factorization

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

Solving Linear Equations by LU Factorization



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

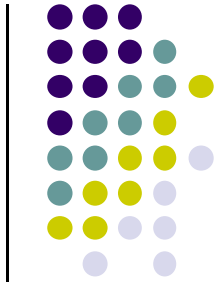
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} \\ a_{31} & a_{32} & a_{33} & \cdots & a_{3n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \cdots & a_{nn} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \\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

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

L : Lower triangular part of matrix A

U : Upper triangular part of matrix A

Linear Equations in Matrix Form



General linear equations with “n” unknowns:

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

Matrix form:

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

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

Solving Ax=b by LU Factorization

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

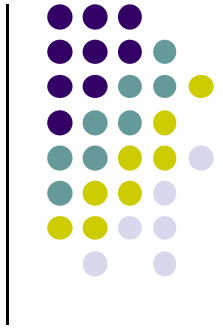
2 $\mathbf{Ly} = \mathbf{b}$ Solve $\mathbf{Ly=b}$

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

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

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

Solving Ly=b: Forward Substitution

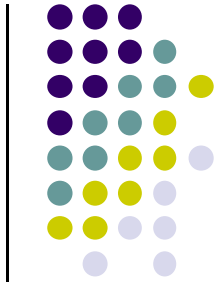


$$\mathbf{Ly} = \mathbf{b} \quad \Longleftrightarrow \quad \begin{pmatrix} 1 & 0 & \cdots & 0 \\ l_{21} & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ l_{n1} & l_{n2} & \cdots & 1 \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots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

$$\Updownarrow$$

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

Solving Ux=y: Backward Substitution



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

$$u_{nn} x_n = y_n$$

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

$$\vdots$$

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

$\Longleftrightarrow$

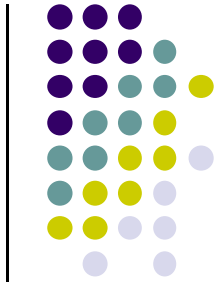
$$x_n = y_n / u_{nn}$$

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

$$\vdots$$

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

How to calculate LU Factorization/Decomposition



①

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

③

② ④

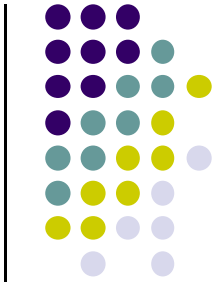
① ➡ $a_{11} = u_{11}, a_{12} = u_{12}, \cdots, a_{1n} = u_{1n} \Rightarrow u_{11}, u_{12}, \cdots, u_{1n}$

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

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

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

Example (1/3)



$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 6 & 7 & 10 \\ 2 & 2 & 8 & 7 \\ 0 & -4 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}$$

1st R ➡ $1 = u_{11}, 2 = u_{12}, 3 = u_{13}, 4 = u_{14}$

1st C ➡ $2 = l_{21}u_{11} \Rightarrow l_{21} = 2/u_{11} = 2, \quad 2 = l_{31}u_{11} \Rightarrow l_{31} = 2/u_{11} = 2$

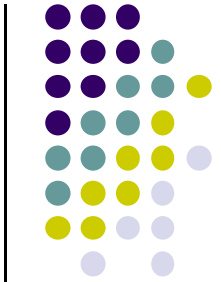
$0 = l_{41}u_{11} \Rightarrow l_{41} = 0/u_{11} = 0$

2nd R ➡ $6 = l_{21}u_{12} + u_{22} \Rightarrow u_{22} = 2, \quad 7 = l_{21}u_{13} + u_{23} \Rightarrow u_{23} = 1$

$10 = l_{21}u_{14} + u_{24} \Rightarrow u_{24} = 2$

2nd C ➡ $2 = l_{31}u_{12} + l_{32}u_{22} \Rightarrow l_{32} = -1, \quad -4 = l_{41}u_{12} + l_{42}u_{22} \Rightarrow l_{42} = -2$

Example (2/3)



$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 6 & 7 & 10 \\ 2 & 2 & 8 & 7 \\ 0 & -4 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{21} & 1 & 0 & 0 \\ l_{31} & l_{32} & 1 & 0 \\ l_{41} & l_{42} & l_{43} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ 0 & u_{22} & u_{23} & u_{24} \\ 0 & 0 & u_{33} & u_{34} \\ 0 & 0 & 0 & u_{44} \end{pmatrix}$$

3rd R ➡ $8 = l_{31}u_{13} + l_{32}u_{23} + u_{33} \Rightarrow u_{33} = 3,$

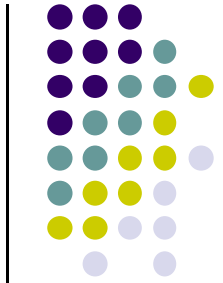
$7 = l_{31}u_{14} + l_{32}u_{24} + u_{34} \Rightarrow u_{34} = 1$

3rd C ➡ $7 = l_{41}u_{13} + l_{42}u_{23} + l_{43}u_{33} \Rightarrow u_{43} = 3$

4th R/C ➡ $1 = l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + u_{44} \Rightarrow u_{44} = 2$

1行、1列、2行、2列、・・・の順に求める式を作っていく.

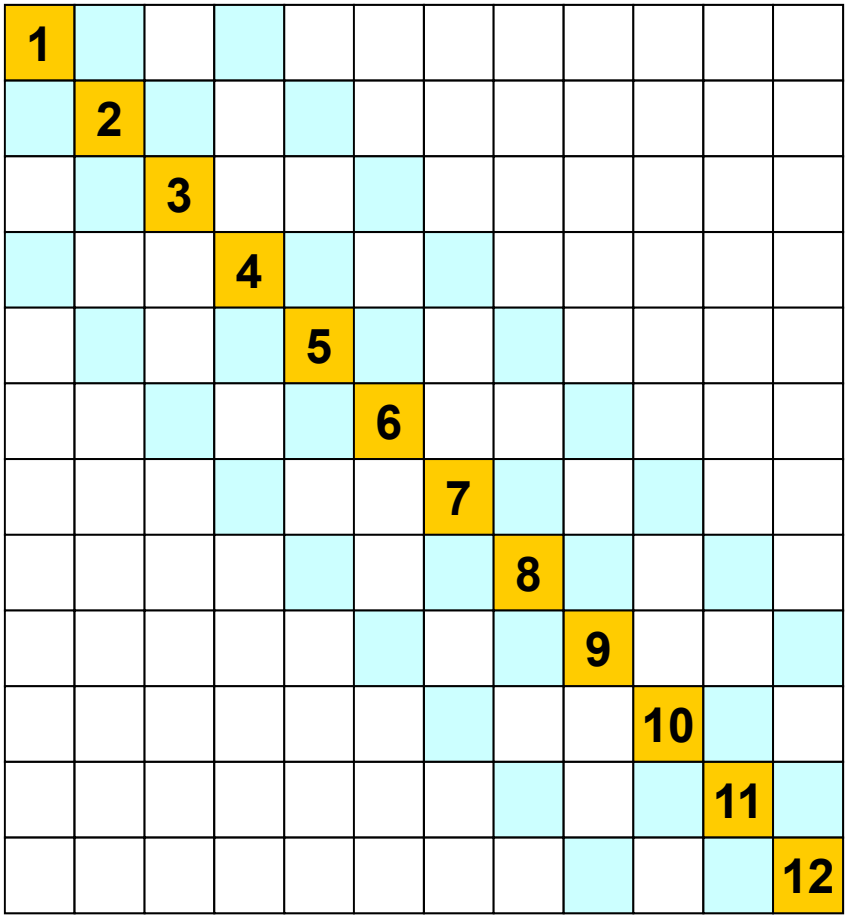
Example (3/3)

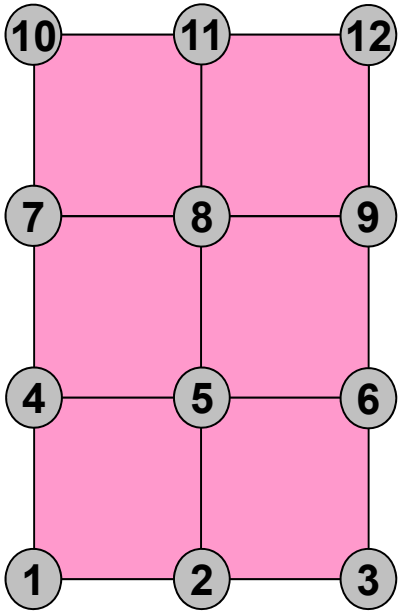


$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 6 & 7 & 10 \\ 2 & 2 & 8 & 7 \\ 0 & -4 & 7 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & -1 & 1 & 0 \\ 0 & -2 & 3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 2 & 1 & 2 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

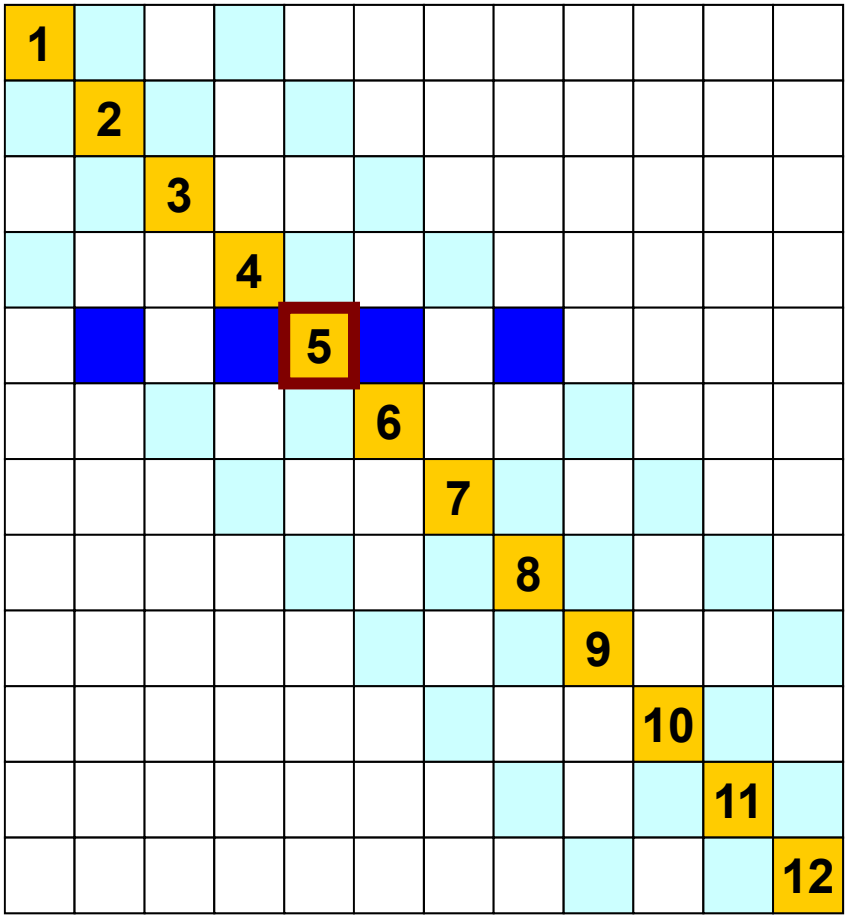
$\mathbf{L}$ $\mathbf{U}$

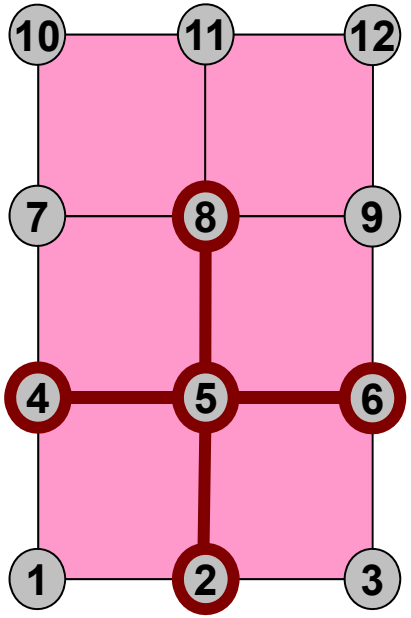
Example: 5-Point Stencil





Example: 5-Point Stencil



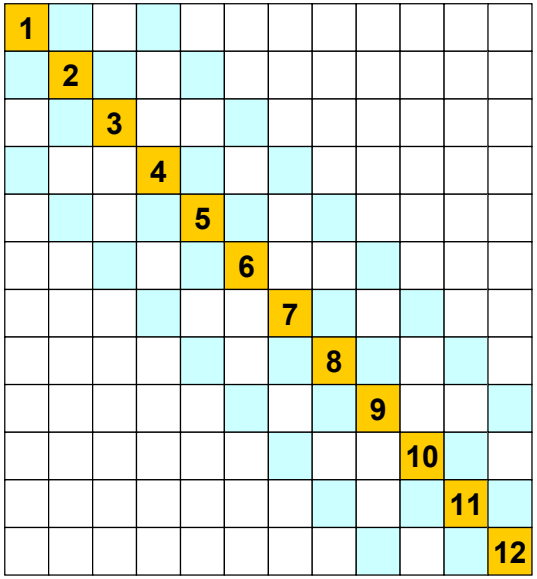


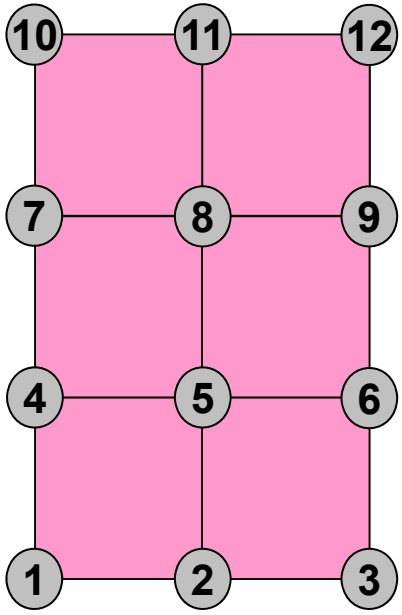
Coef. Matrix

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

X =

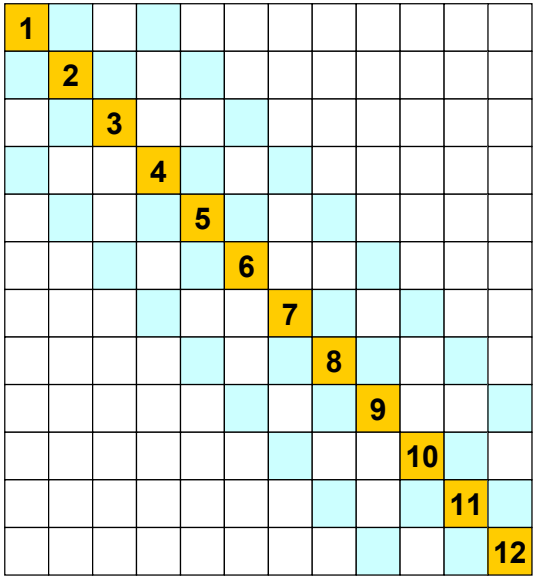
0.00
3.00
10.00
11.00
10.00
19.00
20.00
16.00
28.00
42.00
36.00
52.00

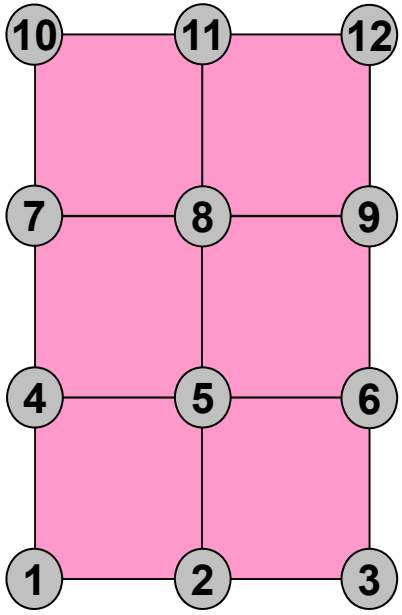




Solution

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





Full LU Factorization

type ./lu1

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)

type ./lu2

ILU Factorization

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

NO fill-in's

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

LU Factorization

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

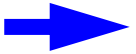
fill-in occurred

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

Solution: A little bit inaccurate ...

ILU

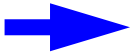
6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	5.83	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−0.17	0.00	−0.17	5.66	−1.00	0.00	−1.00	0.00	0.00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



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

Full LU

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



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

ILU(0), IC(0)

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

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

Classification of Preconditioning Methods: Trade-off

Point Jacobi

Diagonal Blocking

ILU(0)

ILU(1)

ILU(2)

Gaussian Elimination

- Simple
- Easy to be Parallelized
- Cheap

- Complicated
- Global Dependency
- Expensive

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

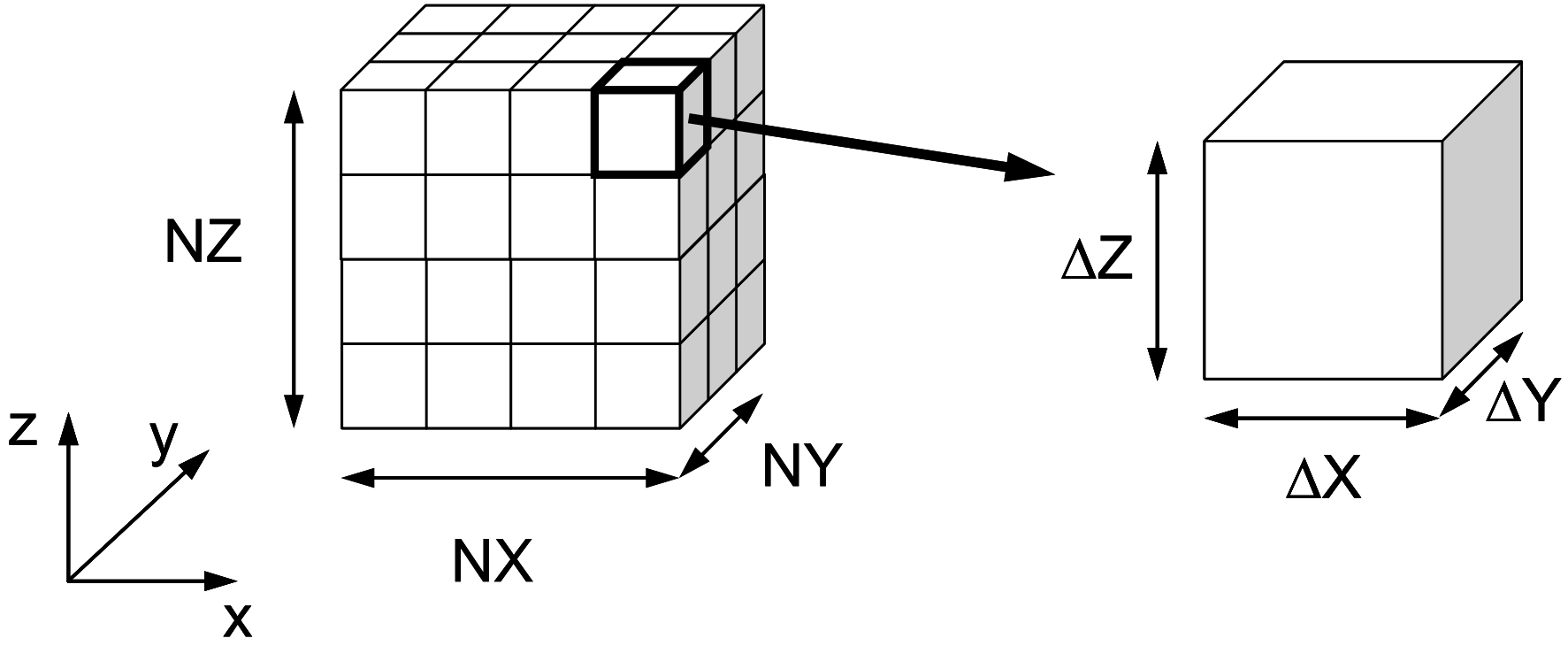
Target Application

- 3D Poisson Equations

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

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

3D Structured Mesh
Internal data structure is “unstructured”



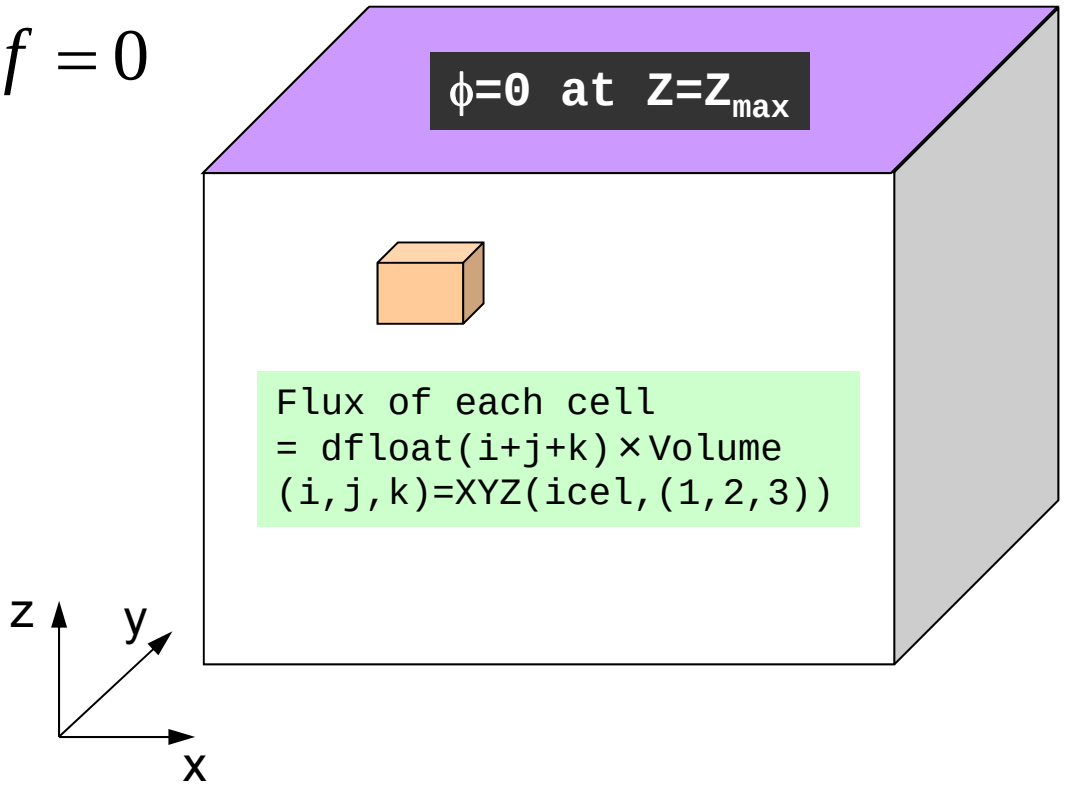
Target Problem: Variables are defined at cell-center'

Poisson Equation

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

B.C.

- Volume Flux
- ϕ=0@Z=Z_{max}



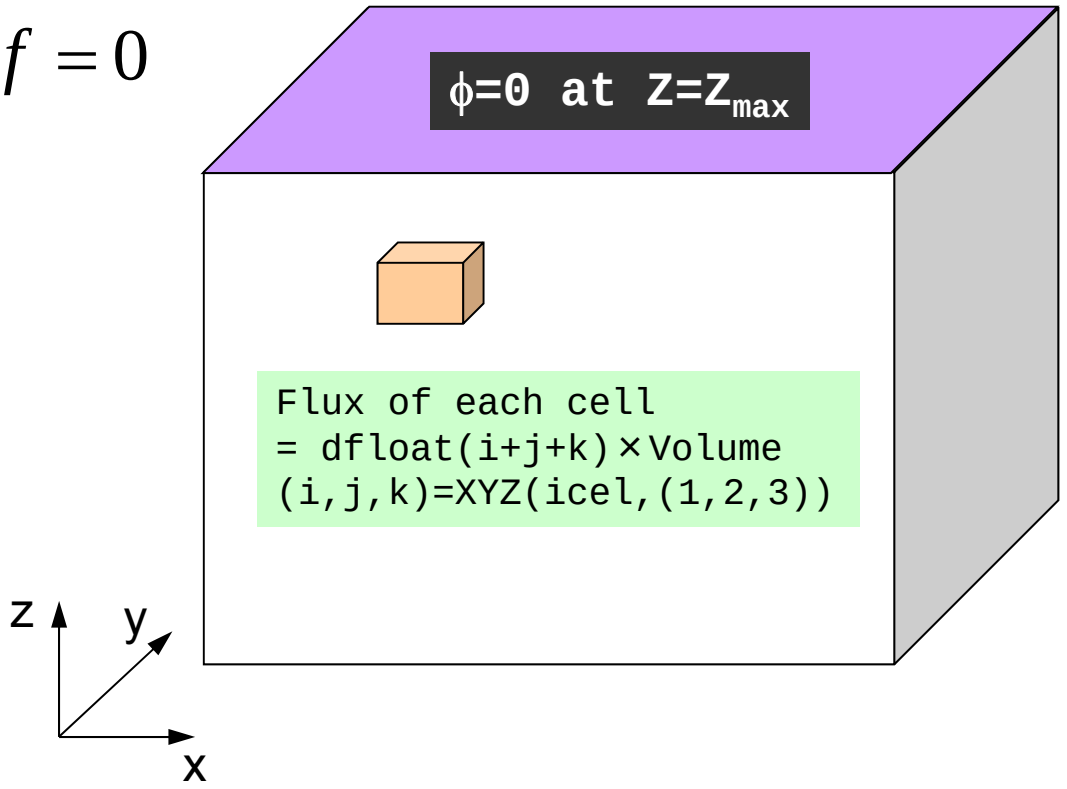
Target Problem: Variables are defined at cell-center'

Poisson Equation

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

B.C.

- Volume Flux
- ϕ=0@Z=Z_{max}



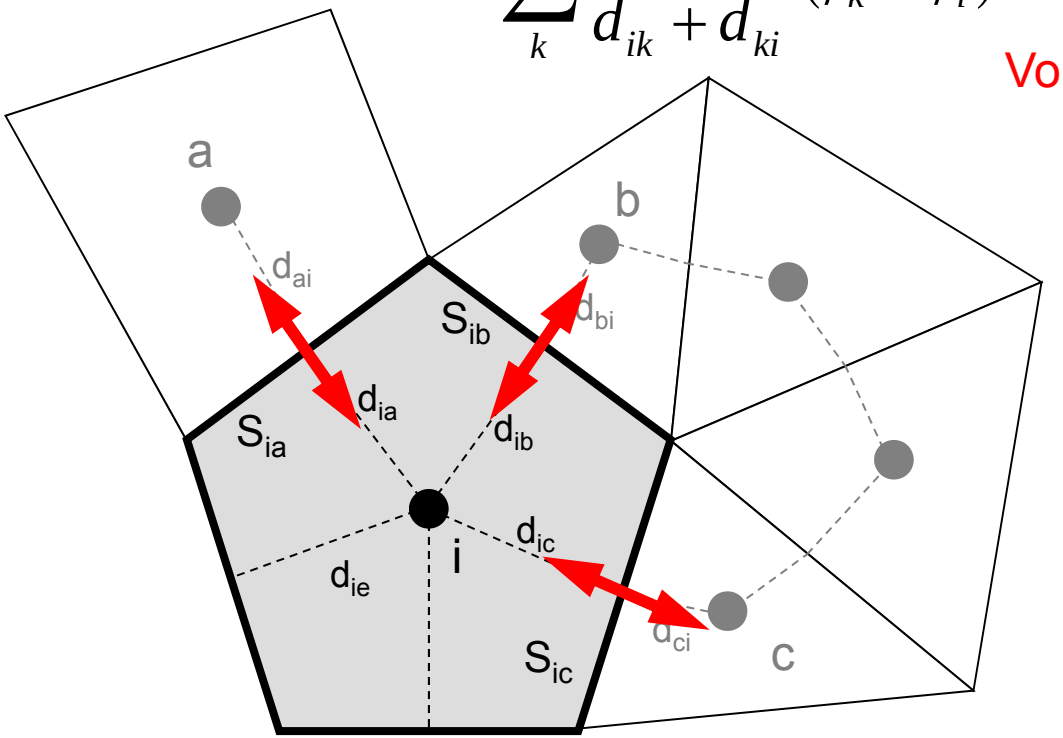
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

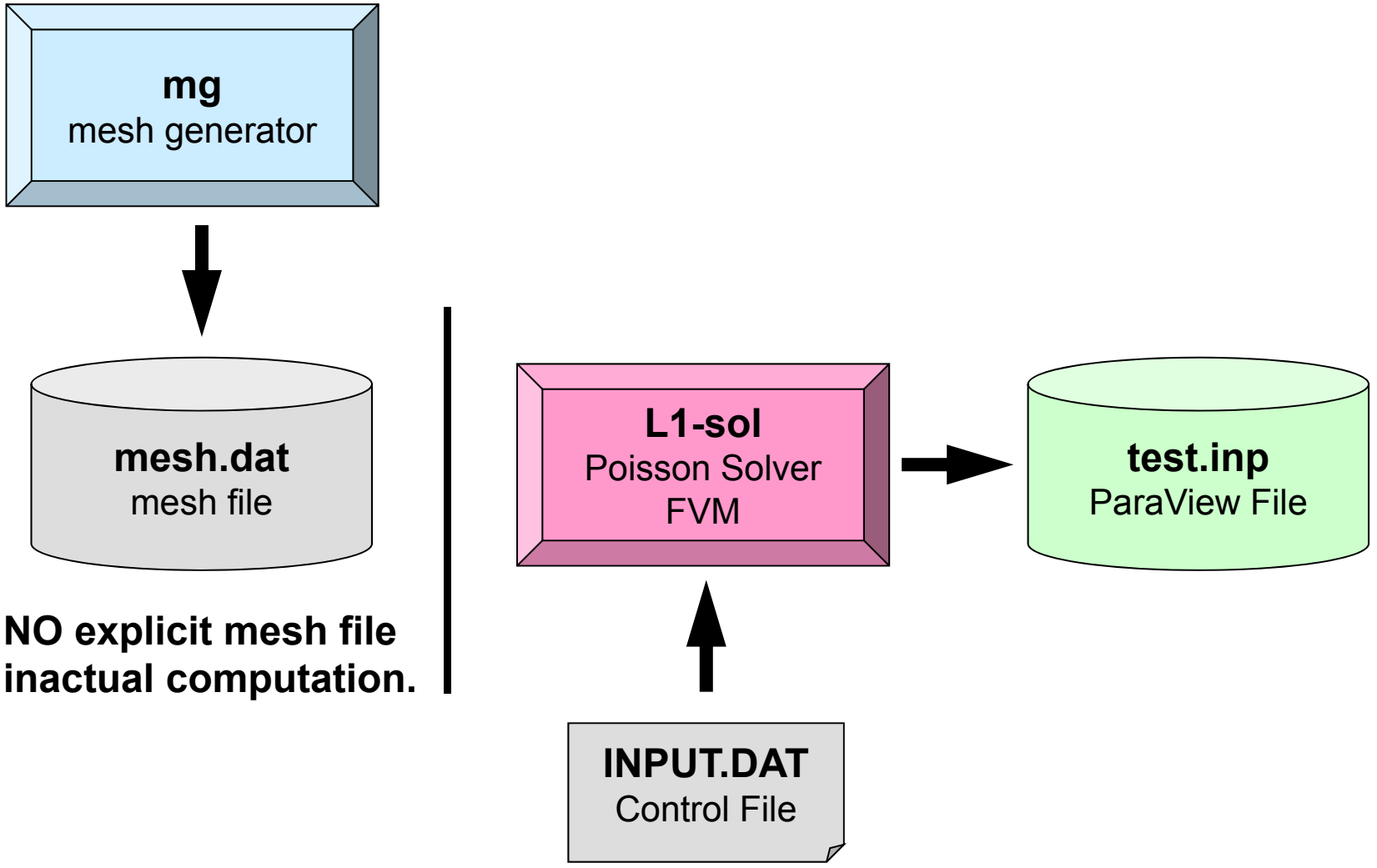
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

Volume Flux



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

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



Running the Program

Compiling

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

$> g95 -O mg.f -o mg (or cc –O mg.c –o mg)
$> ls mg
      mg

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

Mesh Generator: **mg**

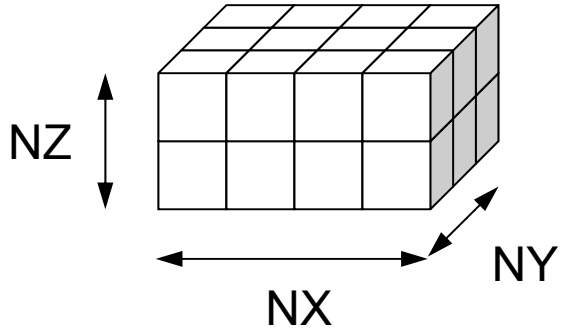
Poisson Solver (FVM): **L1-sol**

Running the Program

Mesh Generation

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

NX, NY, NZ



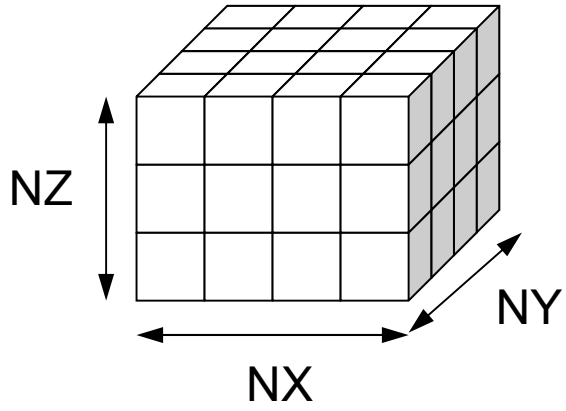
mesh.dat (1/5)

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

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

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

mesh.dat (2/5)

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



Number of meshes in X/Y/Z directions

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

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

mesh.dat (3/5)

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

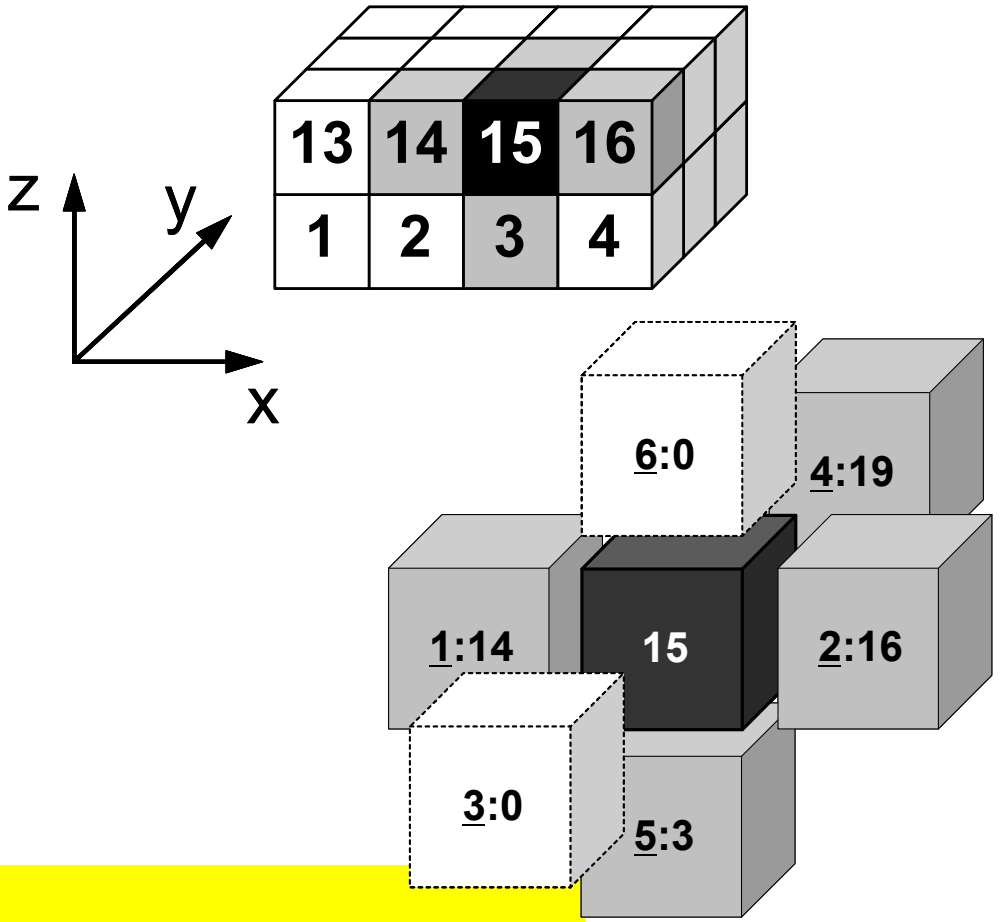
```
 4   3   2
24
 1   0   2   0   5   0  13   1   1   1
 2   1   3   0   6   0  14   2   1   1
 3   2   4   0   7   0  15   3   1   1
 4   3   0   0   8   0  16   4   1   1
 5   0   6   1   9   0  17   1   2   1
 6   5   7   2  10   0  18   2   2   1
 7   6   8   3  11   0  19   3   2   1
 8   7   0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

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

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

mesh.dat (4/5)

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

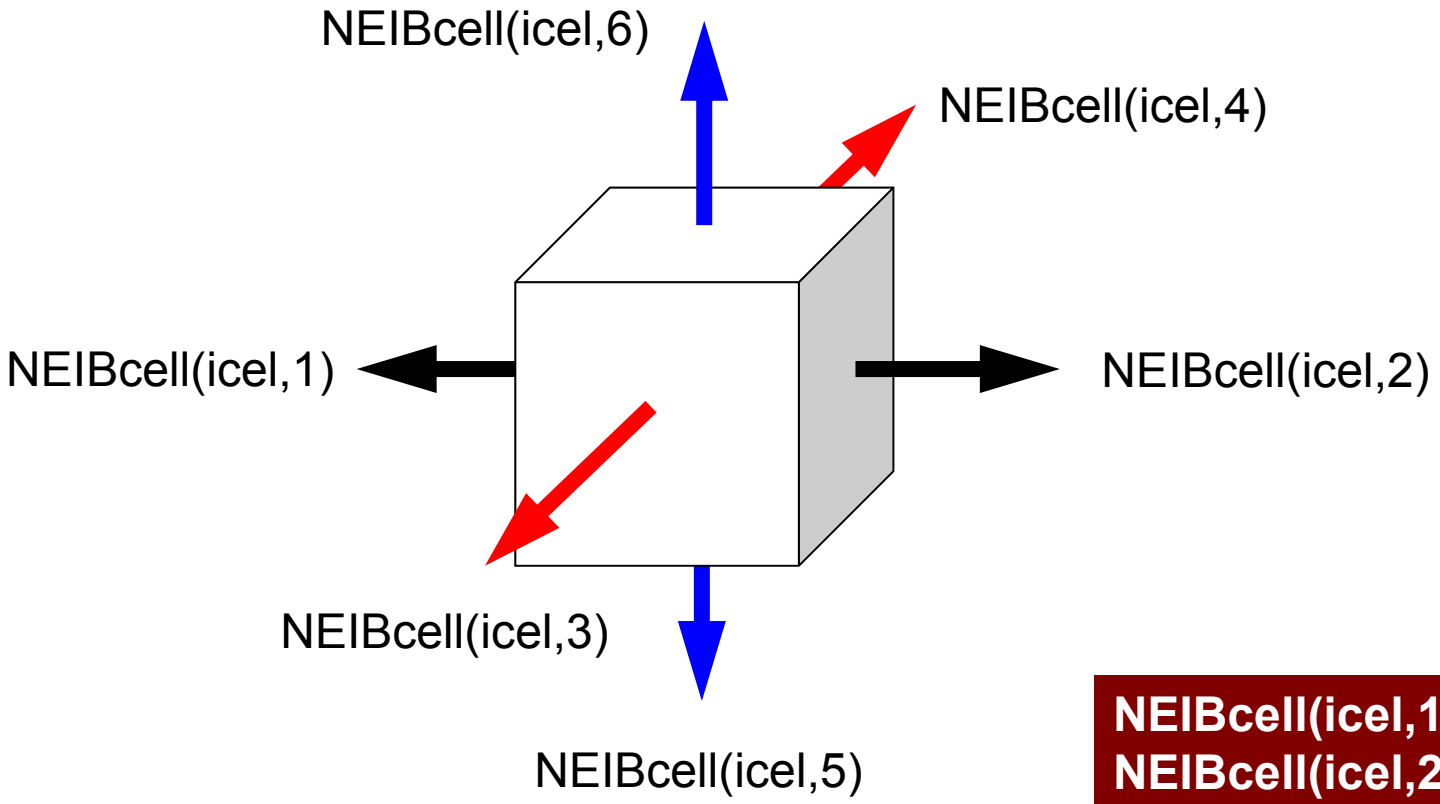
Neighboring Cells: NEIBcell(i,k)




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

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

1st Col.: Global ID of the Cell

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

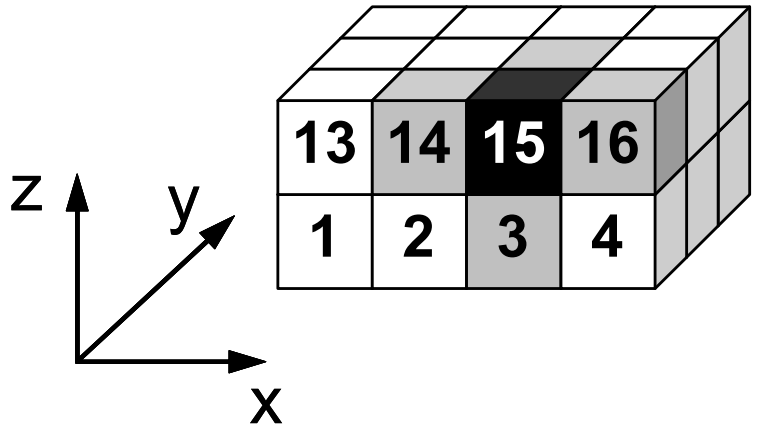


NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

mesh.dat (5/5)

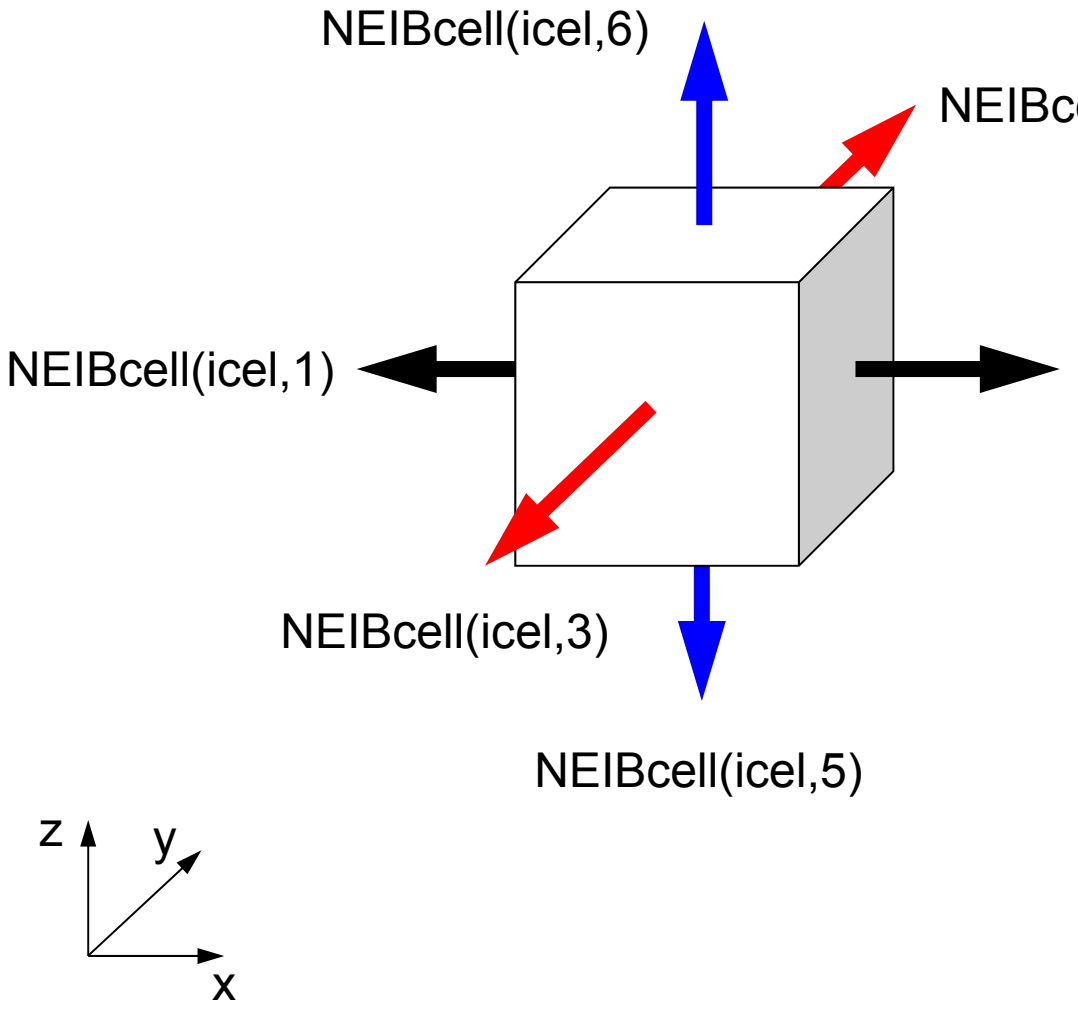
Location in X,Y,Z-directions: XYZ(i,j)

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

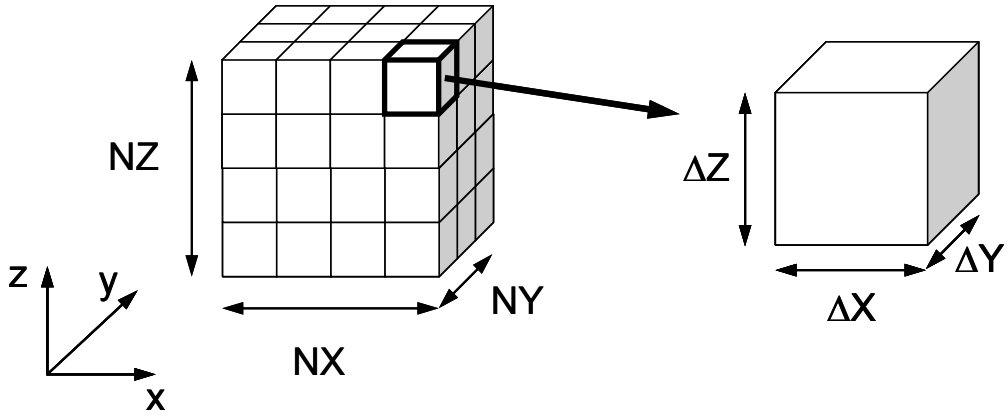
NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

Running the Program

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

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

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



Preconditioning Method

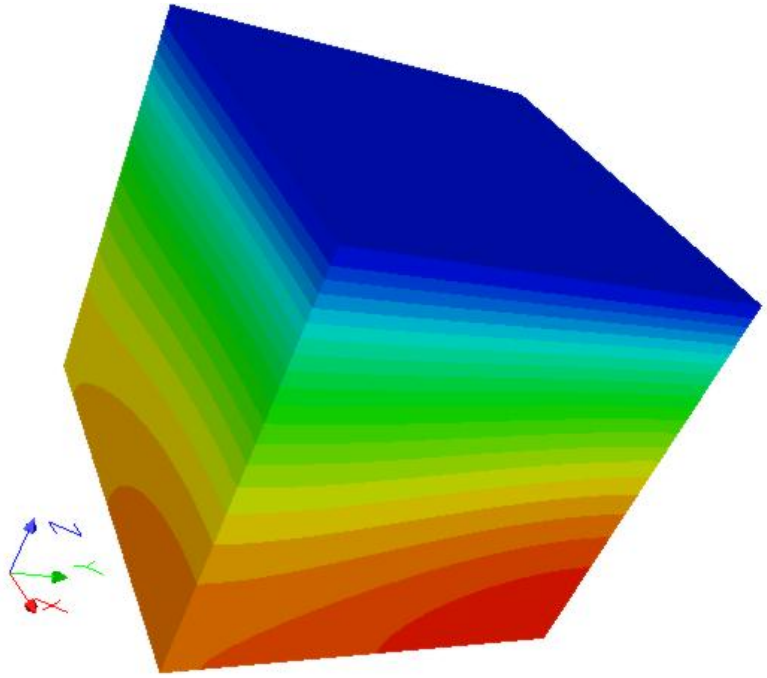
```
32 32 32                                NX/NY/NZ
1                                       MEHOD 1:2
1.00e-00 1.00e-00 1.00e-00              DX/DY/DZ
1.0e-08                                 EPSICCG
```

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

Running the Program

Running, Post Processing by ParaView

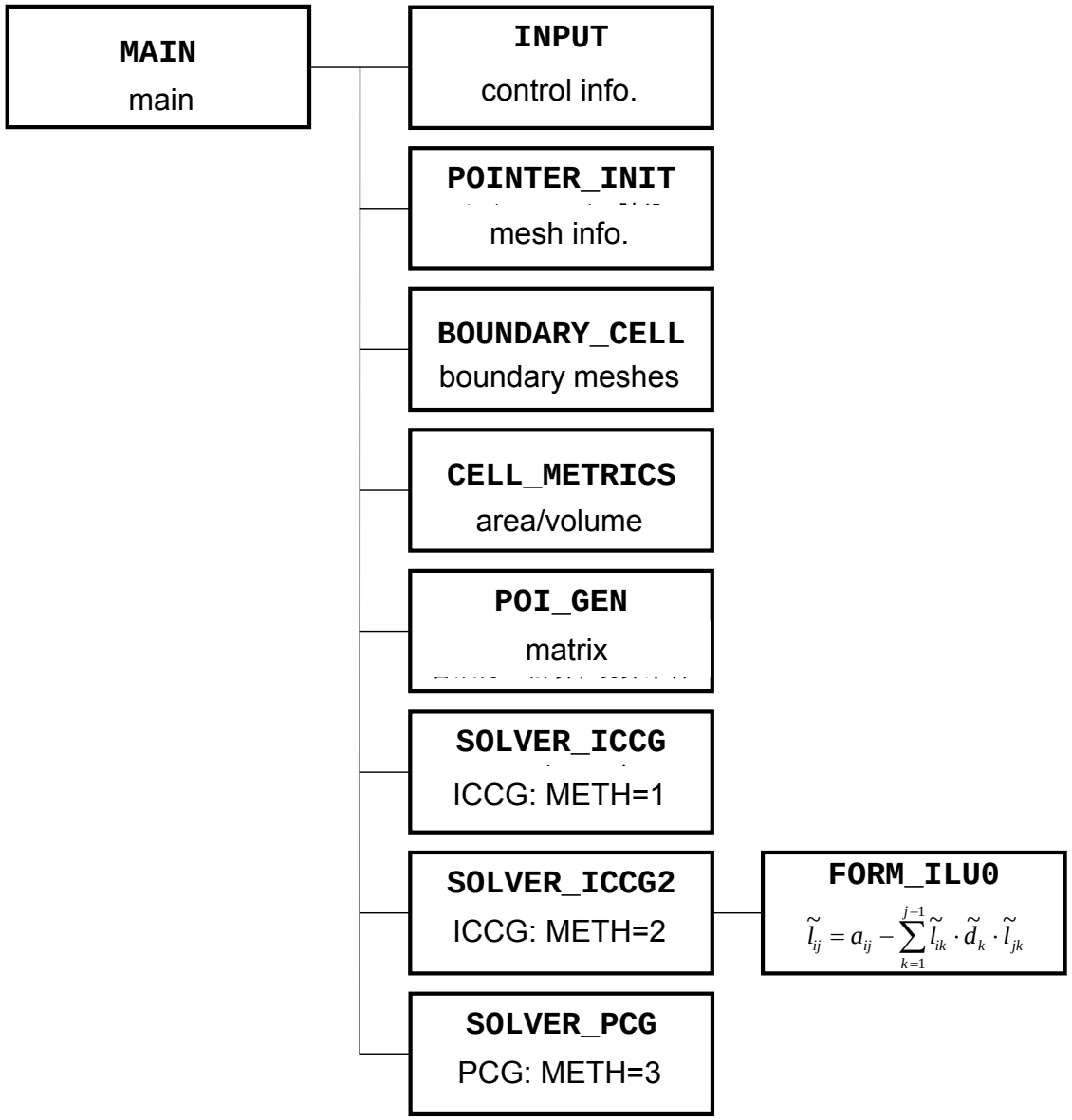
```
$> cd <$E-L1>/run
$> ./L1-sol

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



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

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 modes

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

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

IBNODTOT:
= NXP1 x NYP1

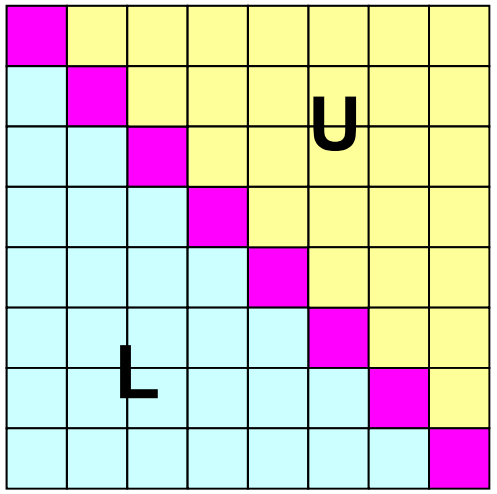
XYZ(ICELTOT,3):
Location of meshes

NEIBcell(ICELTOT,6):
Neighboring meshes

module PCG (1/5)

```
module PCG

integer, parameter :: N2= 256
integer :: NUmax, NLmax, NCOLORtot, NCOLORk, NU, NL, METHOD

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



module PCG (2/5)

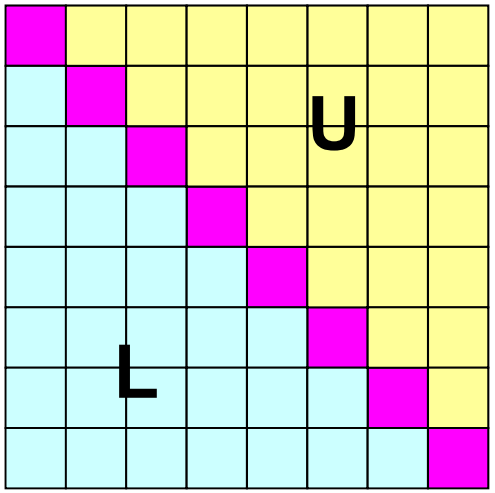
```
module PCG

integer, parameter :: N2= 256
integer :: NUmax, 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 (3/5)

```
module PCG

integer, parameter :: N2= 256
integer :: NUmax, 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`

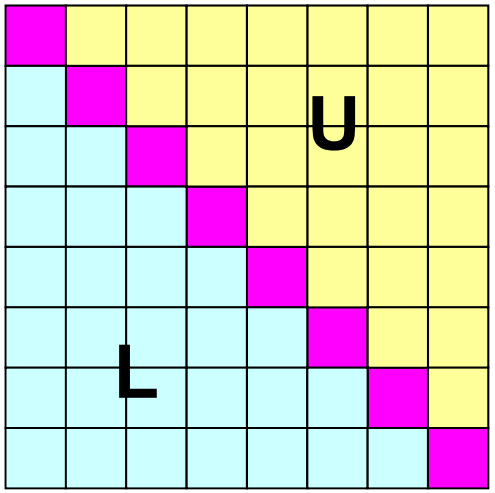
`INL(i)`: Number@each row

Upper Part (Column ID)

`IAU(icou,i) > i`

`INU(i)`: Number@each row

`INL(ICELTOT)`	**# 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 (4/5)

```
      module PCG

      integer, parameter :: N2= 256
      integer :: NUmax, 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`

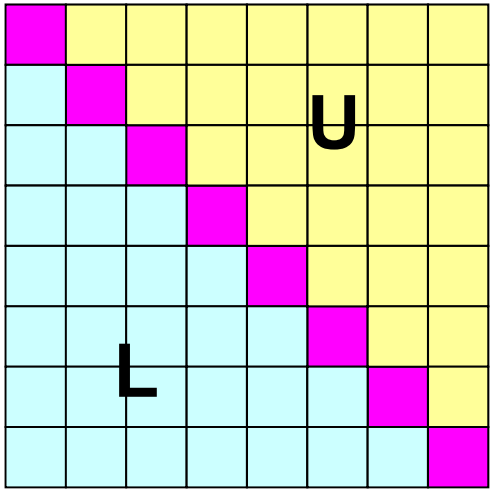
`INL(i)`: Number@each row

Upper Part (Column ID)

`IAU(icou,i) > i`

`INU(i)`: Number@each row

`INL(ICELTOT)`	# 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 (5/5)

```
      module PCG

      integer, parameter :: N2= 256
      integer :: NUmax, 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)

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.

Why Auxiliary Arrays ?

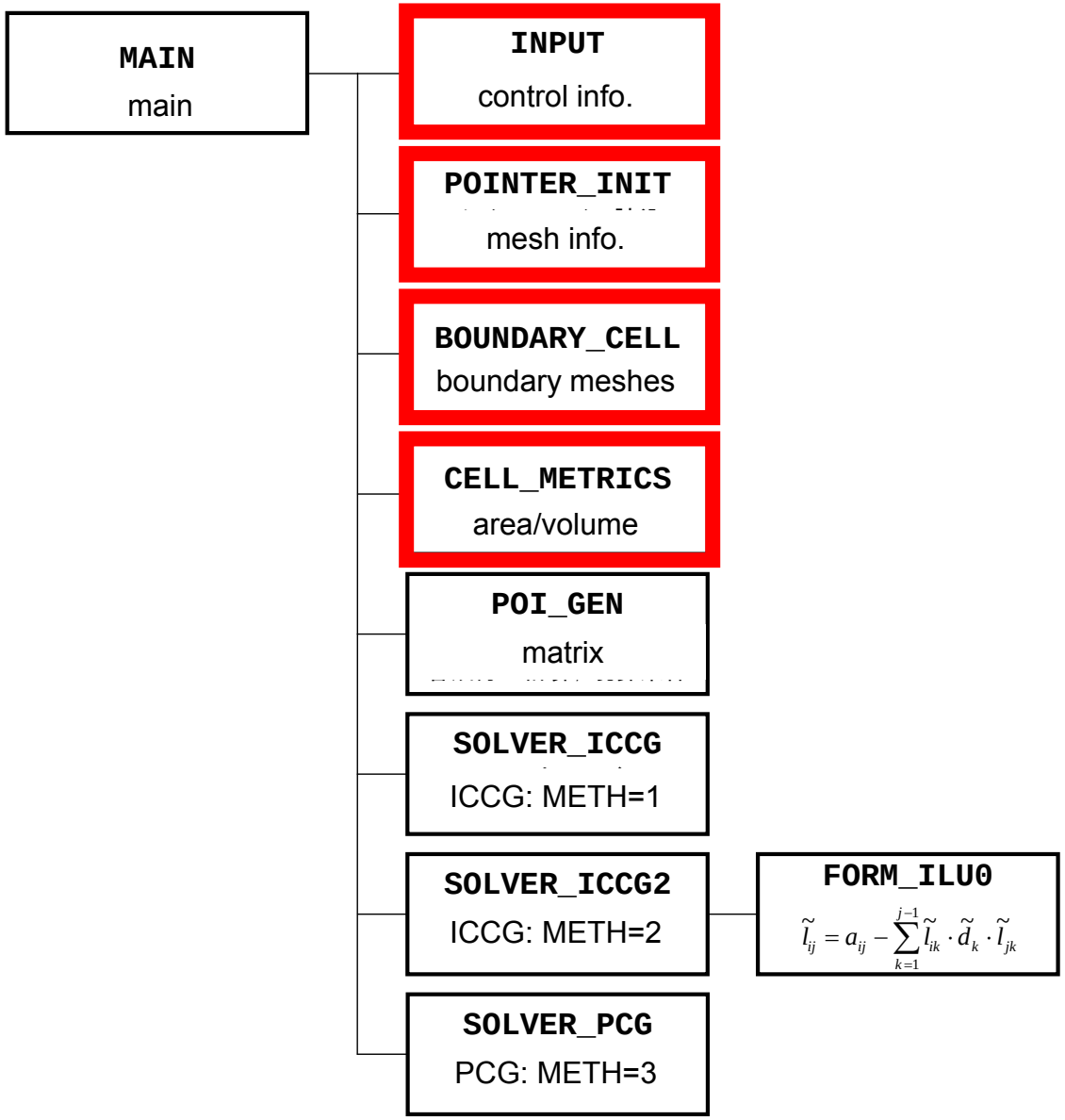
① NPL and NPU are unknown before computation.
② CRS is not suitable for reordering.

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

pointer_init (1/3): “mesh.dat”

```
!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)

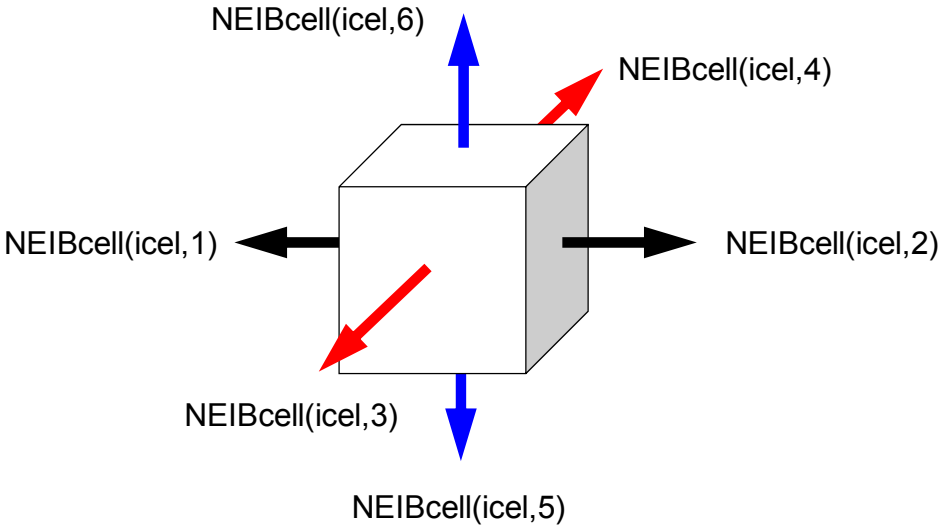
ICELTOT:
Number of meshes (NX x NY x NZ)

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

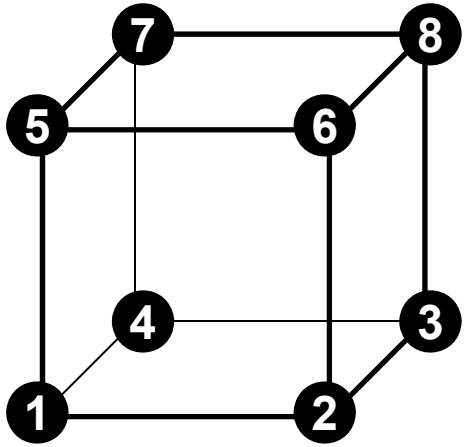
pointer_init (3/3): “mesh.dat”

```
!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)

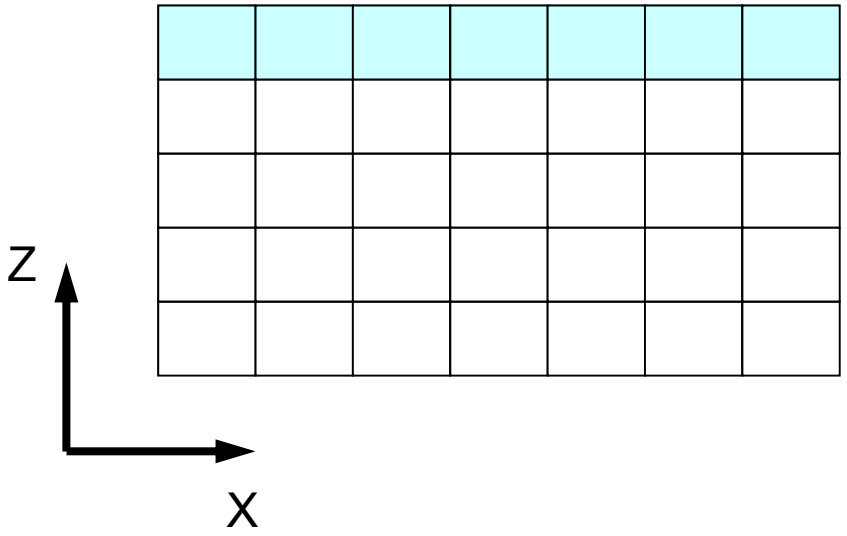
boundary_cell

```
!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: ZmaxCELtot

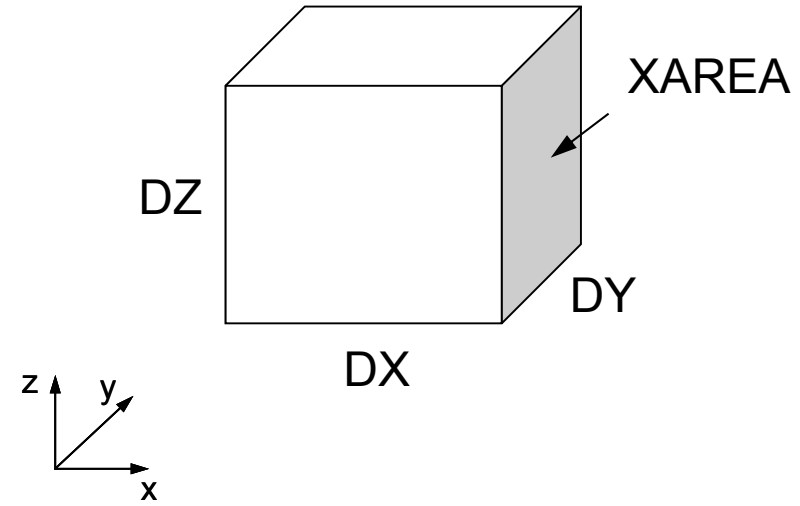
Mesh ID: ZmaxCEL(:)



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



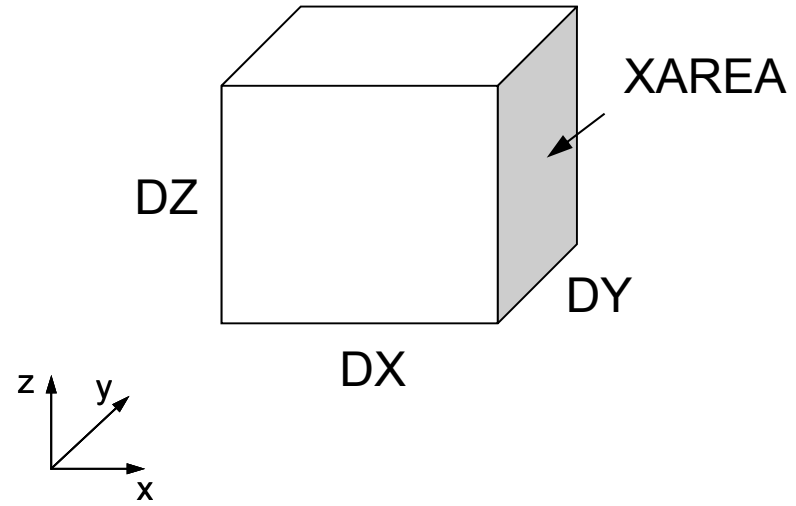
$$XAREA = \Delta Y \times \Delta Z, \quad YAREA = \Delta Z \times \Delta X,$$
$$ZAREA = \Delta X \times \Delta Y$$

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

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



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

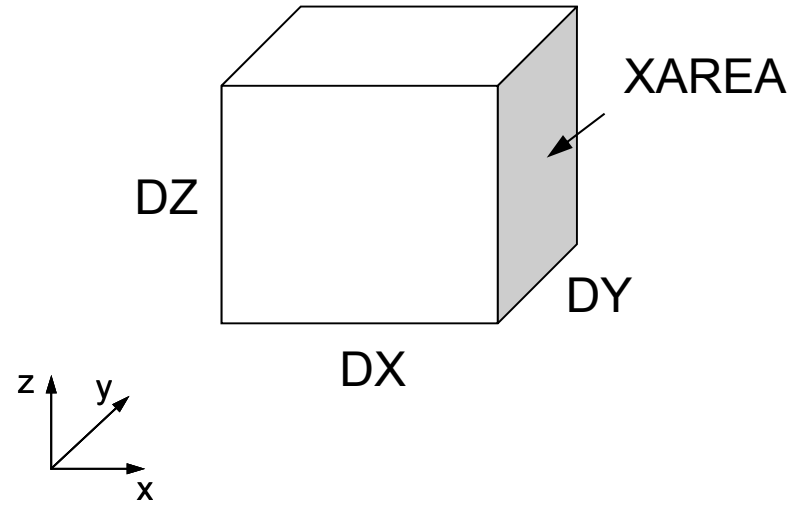
$$R2DX = \frac{1}{0.5 \times \Delta X}, \quad R2DY = \frac{1}{0.5 \times \Delta Y},$$

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

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

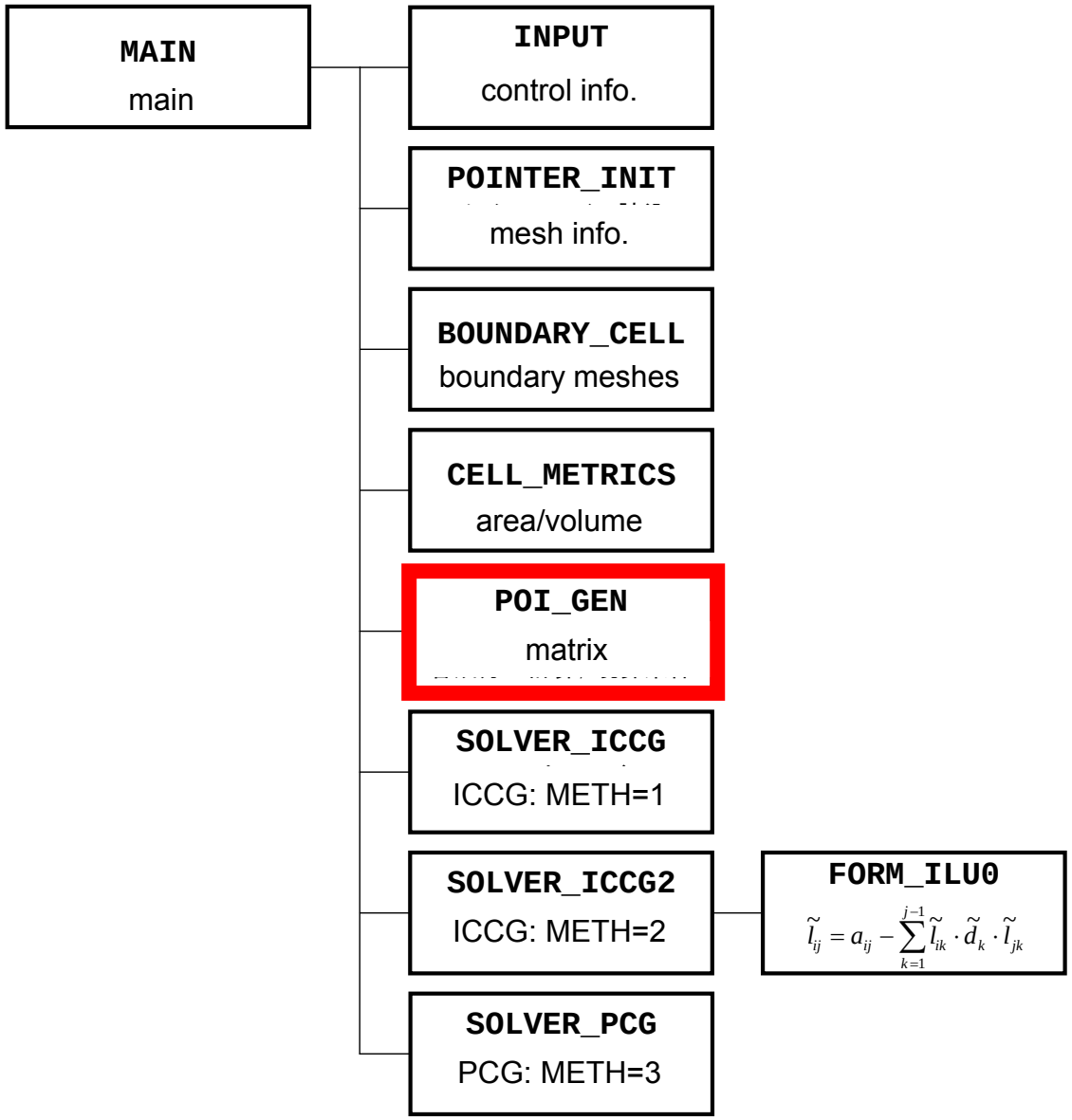


$$VOLCEL = V0 = \Delta X \times \Delta Y \times \Delta Z$$

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

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))
      allocate (                   AL(NL,nn),  AU(NU,nn))

      PHI= 0.d0
        D= 0.d0

      INL= 0
      INU= 0
      IAL= 0
      IAU= 0
       AL= 0.d0
       AU= 0.d0
```

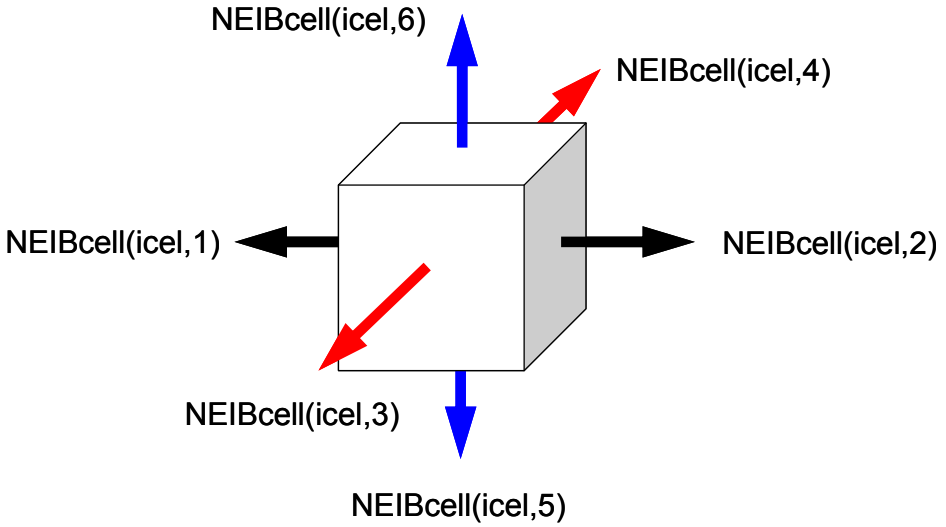
Variables/Arrays for Matrix

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
```

poi_gen (2/7)

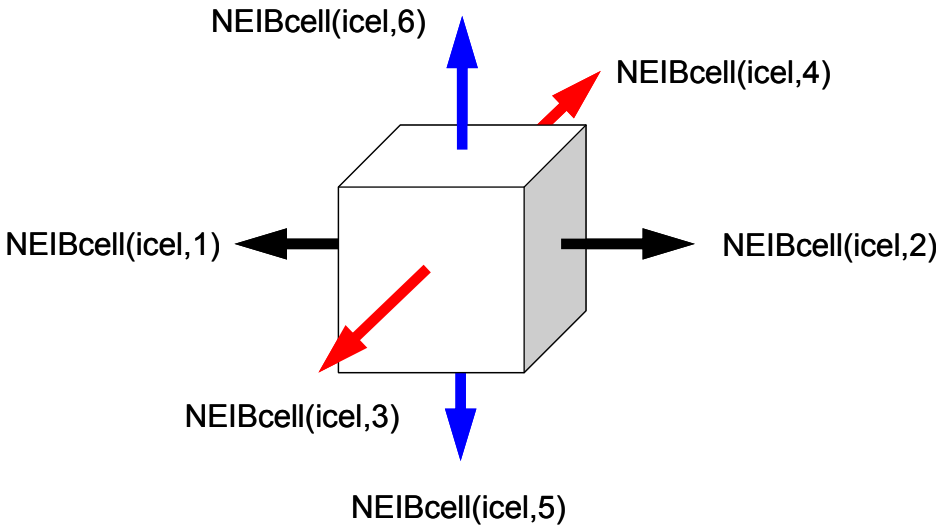


Lower Triangular Part

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

poi_gen (4/7)

```
!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===
```

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
```

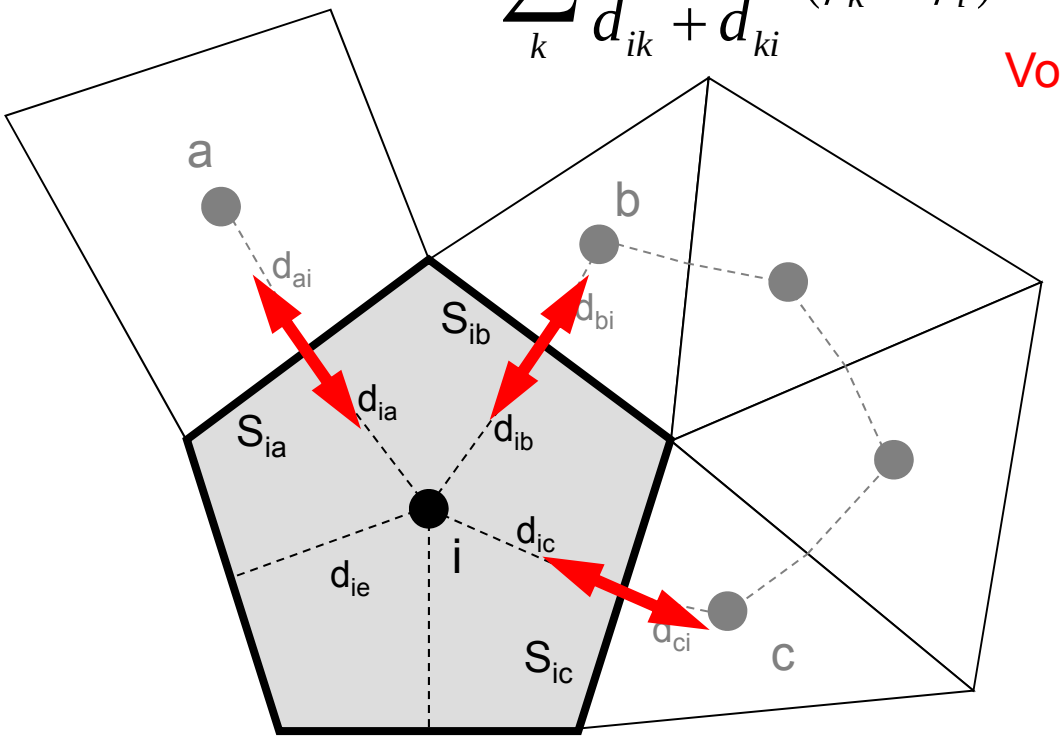
Finite Volume Method（FVM）

Conservation of Fluxes through Surfaces

Diffusion:
Interaction with Neighbors

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

Volume Flux



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

Constructing Coefficient Matrix

Conservation for i-th mesh

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

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

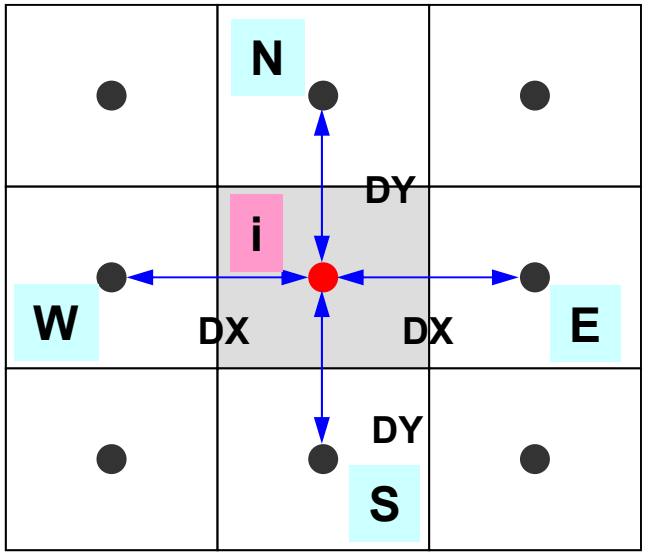
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

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

```
!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
```

poi_gen (5/7)

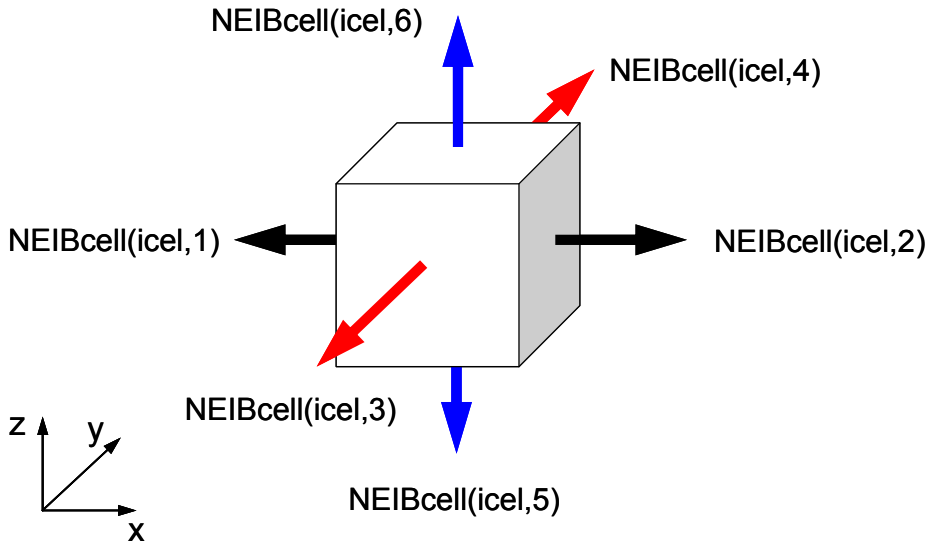
Calculation of Coefficients



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

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

in 3D ...




```
if (icN5.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
```

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

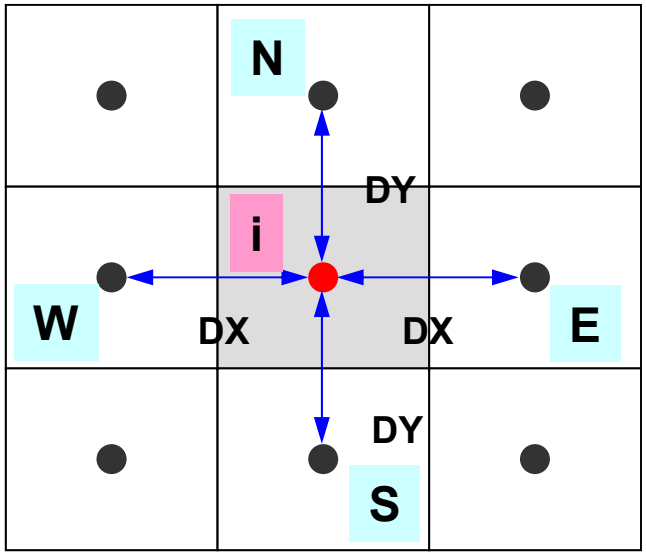
$$\frac{\phi_{neib(icel,3)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+\frac{\phi_{neib(icel,4)}-\phi_{icel}}{\Delta y}\Delta z\Delta x+$$

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

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===
```

Calculation of Coefficients



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

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

poi_gen (6/7)

```
        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===
```

Volume Flux

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

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

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

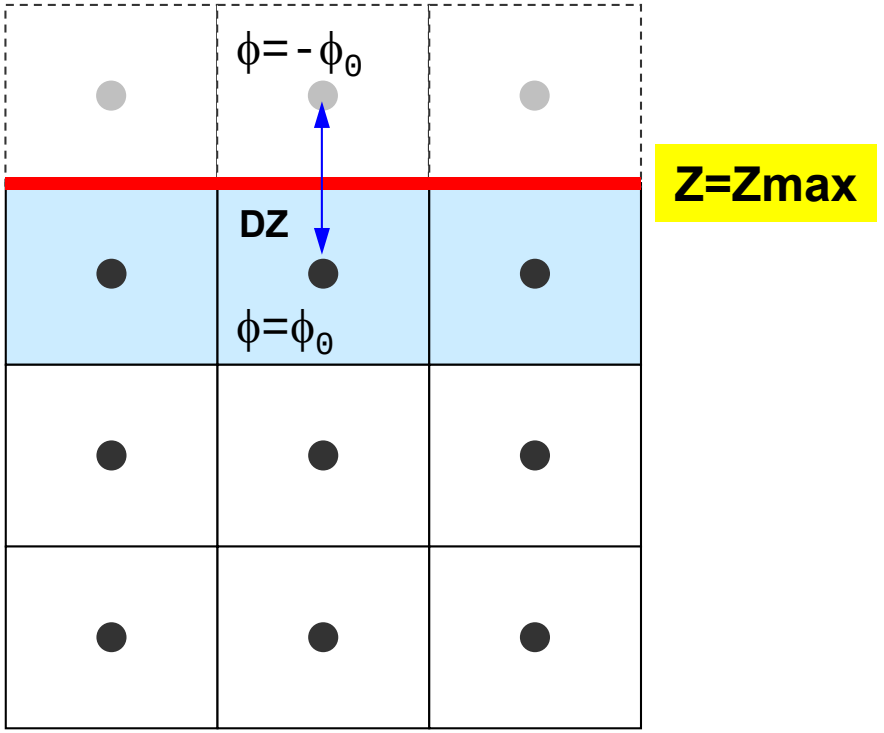
$$k_0 = XYZ(icel,3)$$

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

```
!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
```

poi_gen (7/7)

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



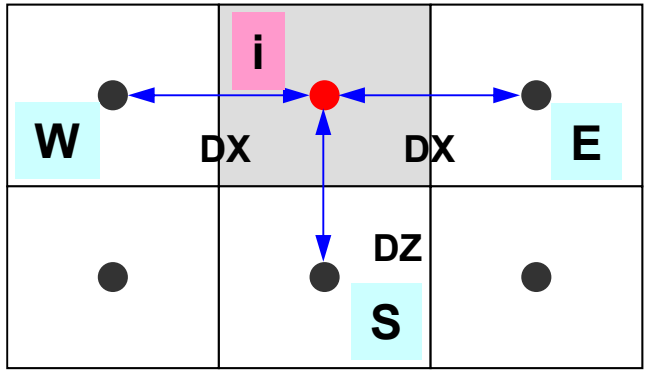
1st Order Approximation:

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

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

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

Dirichlet B.C.

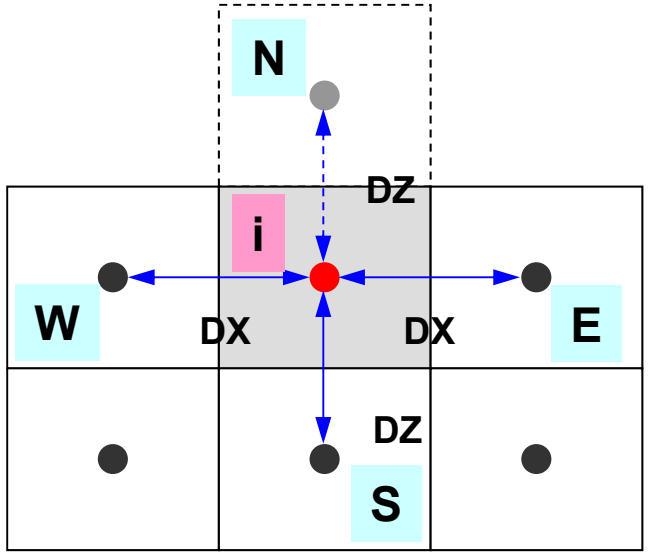


$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

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

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

Dirichlet B.C.



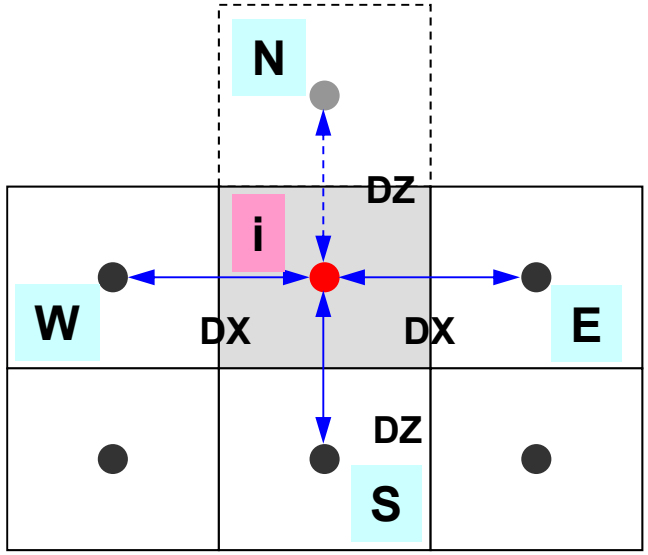
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

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

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

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

Dirichlet B.C.



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

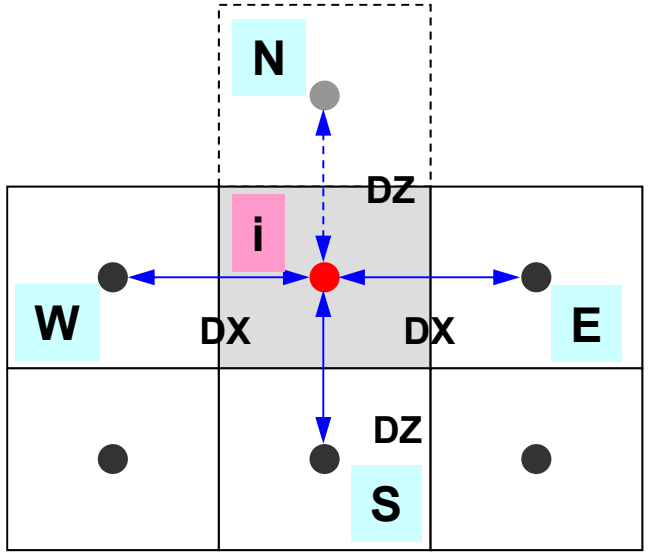
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

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

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

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

Dirichlet B.C.



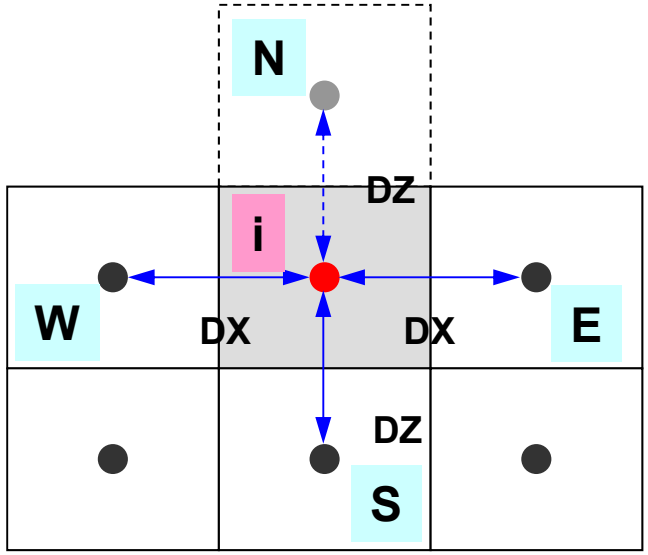
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\left(\phi_k-\phi_i\right)+V_i\dot{Q}_i=0$$

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

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

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

Dirichlet B.C.



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

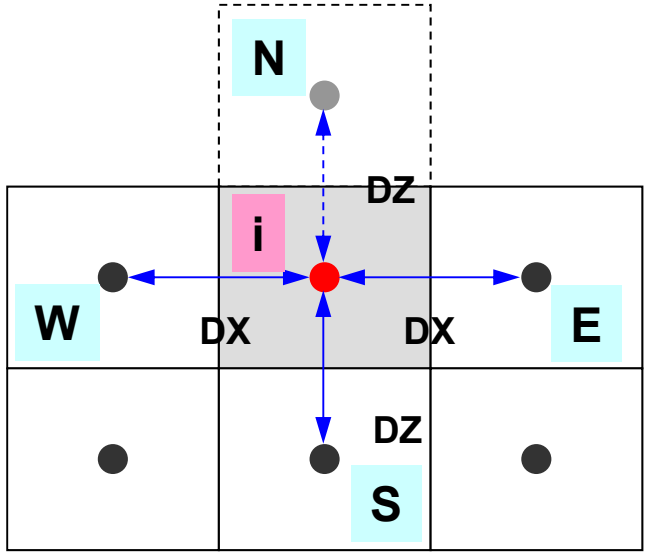
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=-V_i\dot{Q}_i$$

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

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

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

Dirichlet B.C.



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

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

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

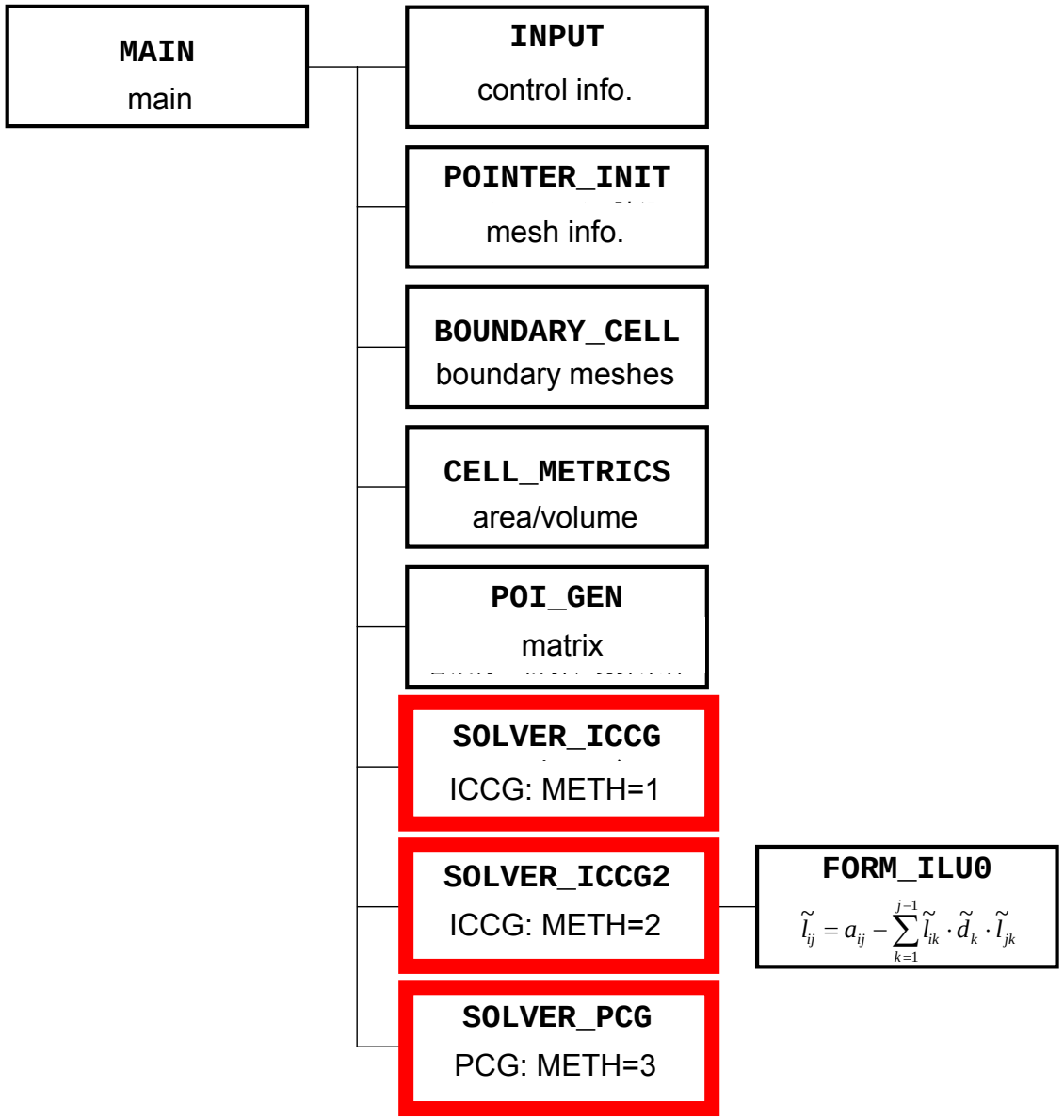
$$-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i+\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]$$

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

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

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**
- OpenMP

Solving Linear Equations

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

“Modified” Cholesky Factorization

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

N=5

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

$$l_{ij} = a_{ij}$$

Running the Program

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

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

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

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

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

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

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

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

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

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

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

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

Only diagonal components are changed

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

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

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

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

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

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

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

```
!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===
```

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

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

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

Forward/Backward Substitution for Incomplete Modified Cholesky Fact.

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

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

```
!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===
```

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

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

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

solve_ICCG (1/7): METHOD= 1

```
!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)

      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) ⇒ {r}
W(i,2)= W(i,Z) ⇒ {z}
W(i,2)= W(i,Q) ⇒ {q}
W(i,3)= W(i,P) ⇒ {p}

W(i,4)= W(i,DD) ⇒{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

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

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

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

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

Only diagonal components are changed

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===
```

BNRM2=|b|^2
Convergence criteria

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

solve_ICCG (4/7): METHOD= 1

```
!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
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  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

      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, ind
          SW= SW + AU(k) * W(itemU(k),Z)
        enddo
        W(i,Z)= W(i,Z) - W(i,DD)*SW
      enddo
!C===
```

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

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

Forward Substitution

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

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
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  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 / RHO1  otherwise |
!C +----------------------------+
!C===
      if ( L.eq.1 ) then

        do i= 1, N
          W(i,P)= W(i,Z)
        enddo
       else
         BETA= RHO / RHO1
         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
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  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
          RHO1 = 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
    $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
    $p^{(i)}= z^{(i-1)} + \beta_{i-1}\ p^{(i)}$
  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
          RHO1 = RHO
        endif
      enddo
      IER = 1
```

r= b-[A]x
DNRM2=|r|²
BNRM2=|b|²

ERR= |r|/|b|

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

solve_ICCG2 (1/3): METHOD= 2

Fortran ONLY

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

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

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

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

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

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

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

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

solve_ICCG2 (2/3): METHOD= 2

Fortran ONLY

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

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

      call FORM_ILU0
!C===
```

Dlu0, ALlu0, AUlu0:

Factorized Matrix Components

FORM_ILU0 (1/2): Fortran only

Incomplete Modified LU Factorization in solve_ICCG2

```
 contains

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

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

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

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

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

Dlu0, ALlu0, AUlu0:

Factorized Matrix Components

Initial Conditions

Dlu0 = D

ALlu0= AL

AUlu0= AU

FORM_ILU0 (2/2): Fortran only

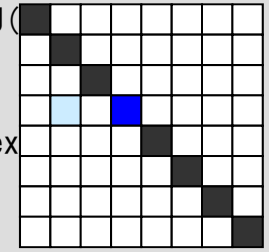
```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
      allocate (IW1(N) , IW2(N))
      IW1=0
      IW2= 0

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

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

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

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

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

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

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

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

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

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

          enddo
        enddo

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

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

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

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

FORM_ILU0 (2/2): Fortran only

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

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

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

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

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

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

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

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

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

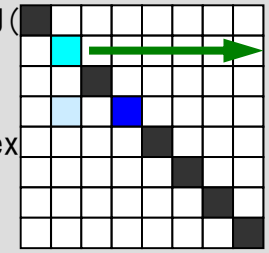


```
            if (j.gt.i .and. IW2(j).ne.0) then
                  Akj= AULU0(kcon)
              RHS_Aij= Aik * DkINV * Akj

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

          enddo
        enddo

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

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

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

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

FORM_ILU0 (2/2): Fortran only

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

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

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

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

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

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

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

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

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



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

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

          enddo
        enddo

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

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

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

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

FORM_ILU0 (2/2): Fortran only

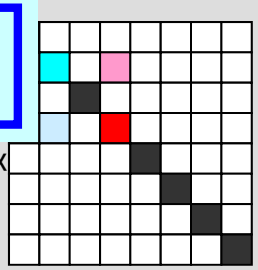
```
!C
!C +---------------------+
!C | ILU(0) factorization |
!C +---------------------+
!C===
```

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

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

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

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



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

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

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

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

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

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

j=i

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

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

        enddo
      enddo

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

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

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

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

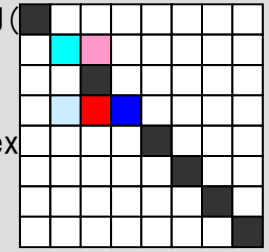
FORM_ILU0 (2/2): Fortran only

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

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

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

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



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

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

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

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

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

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

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

j<i

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

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

        enddo
      enddo

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

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

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

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

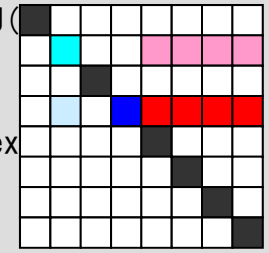
FORM_ILU0 (2/2): Fortran only

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

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

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

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



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

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

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

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

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

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

                   ijU = IW1(j)
            ALIuU(ijU)= ALIuU(ijU) - RHS_Aij
        endif
```

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

                   ijU = IW2(j)
            AUIuU(ijU)= AUIuU(ijU) - 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
    DIuU(i)= 1.d0 / DIuU(i)
  enddo
  deallocate (IW1, IW2)
!C===
  end subroutine FORM_ILU0
```

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

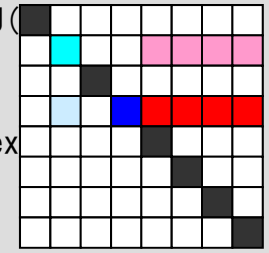
FORM_ILU0 (2/2): Fortran only

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

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

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

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



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

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

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

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

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

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

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

j<i

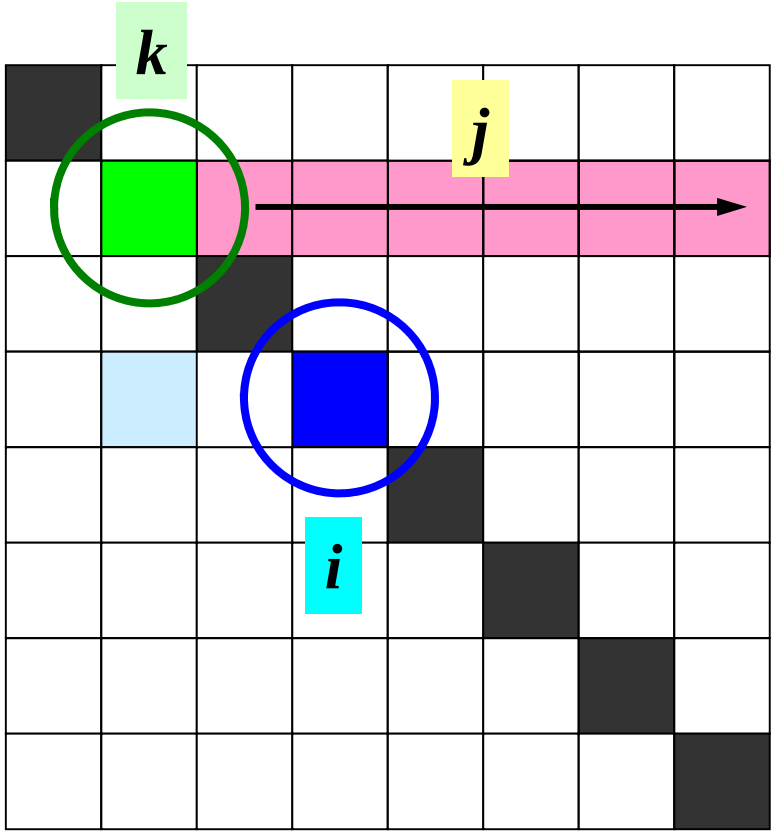
```
        if (j.gt.i .and. IW2(j).ne.0) then
              Akj= AUIuO(kcon)
          RHS_Aij= Aik * DkINV * Akj

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

      enddo
    enddo

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

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

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

j>i

These if –then clauses never applied, therefore:

ALIuO= AL
AUIuO= AU

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



○: Mesh i

○: Mesh k (Lower Triangular Component of i)

■: Mesh j (Upper Triangular Component of k)

if *j=i* Dlu(■) is updated

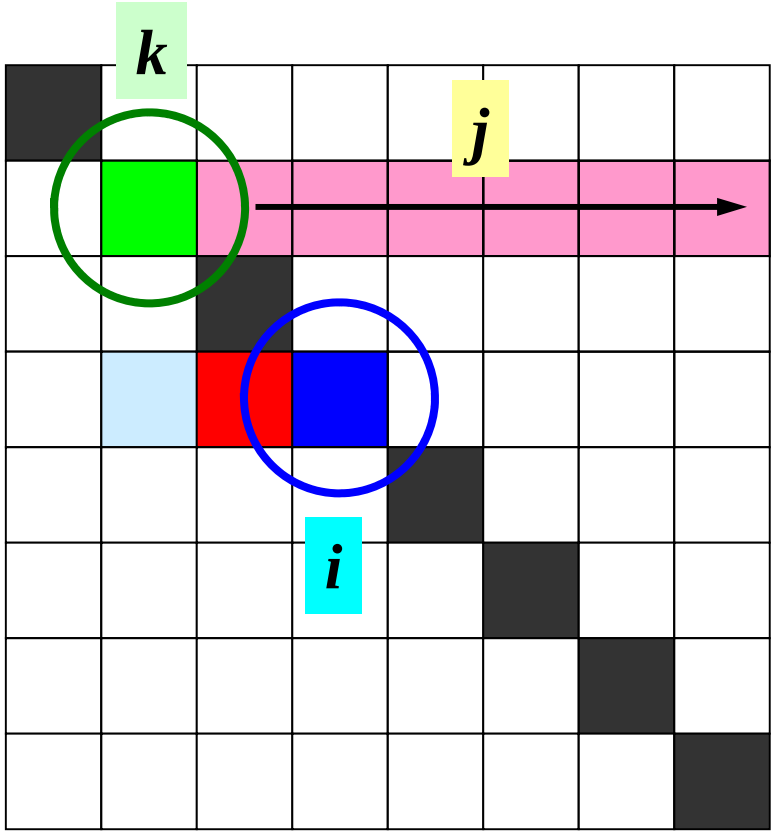
$$i = 1,2,\cdots,n$$

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

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

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

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



○: Mesh i

○: Mesh k (Lower Triangular Component of i)

■: Mesh j (Upper Triangular Component of k)

if $j<i$ ALlu0(i-j)(■) is updated

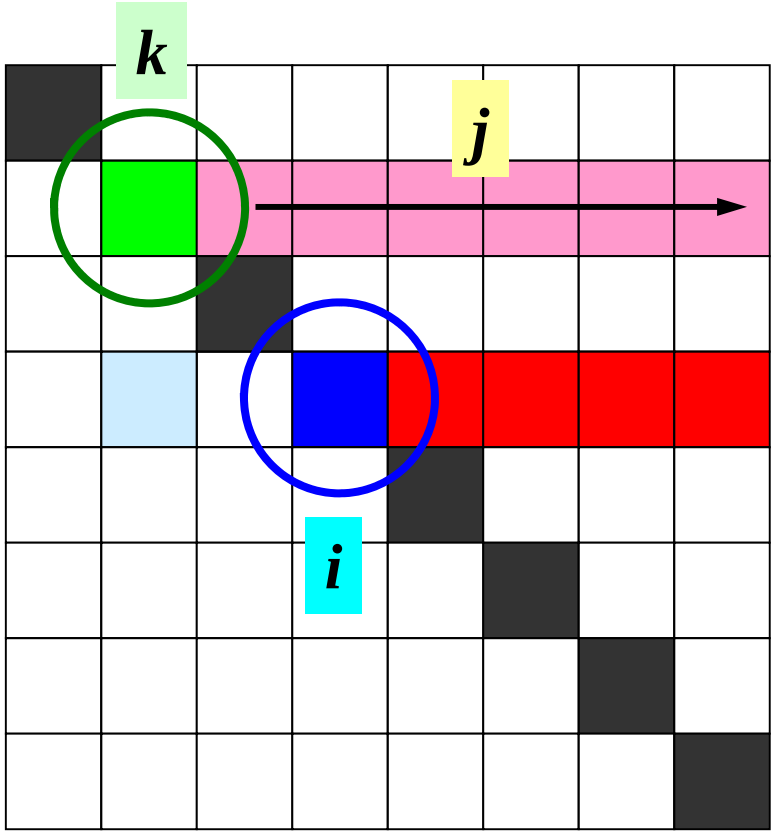
$$i = 1,2,\cdots,n$$

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

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

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

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

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

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

solve_ICCG2 (3/3): METHOD= 2

Fortran ONLY

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

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

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

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

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

Other parts are as same as those in “solve_ICCG”

solve_ICCG2 (3/3): METHOD= 2

Fortran ONLY

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

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

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

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

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

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

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

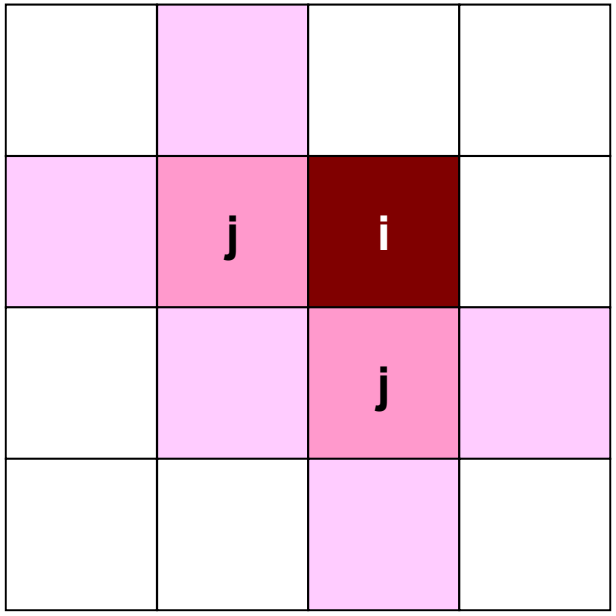
Forward Substitution

Backward Substitution

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

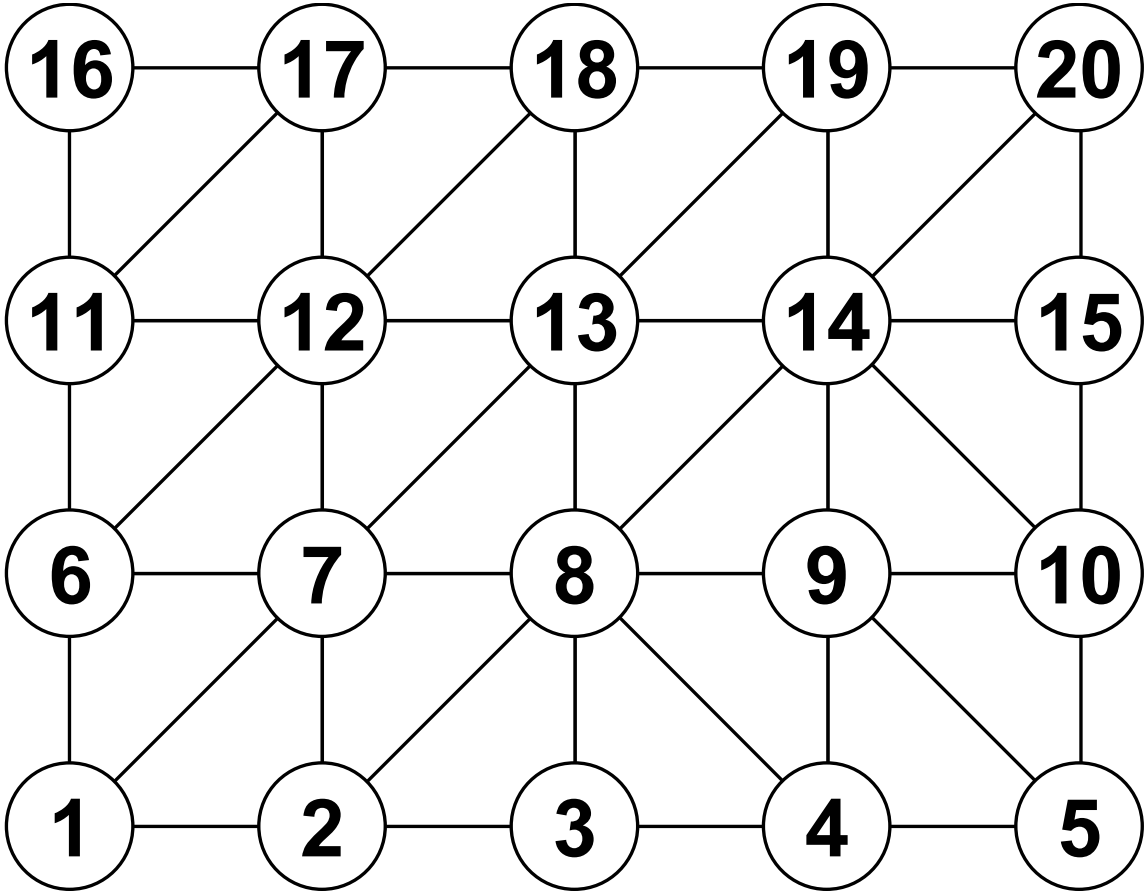
Incomplete Modified Cholesky Fact. Structured Meshes

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



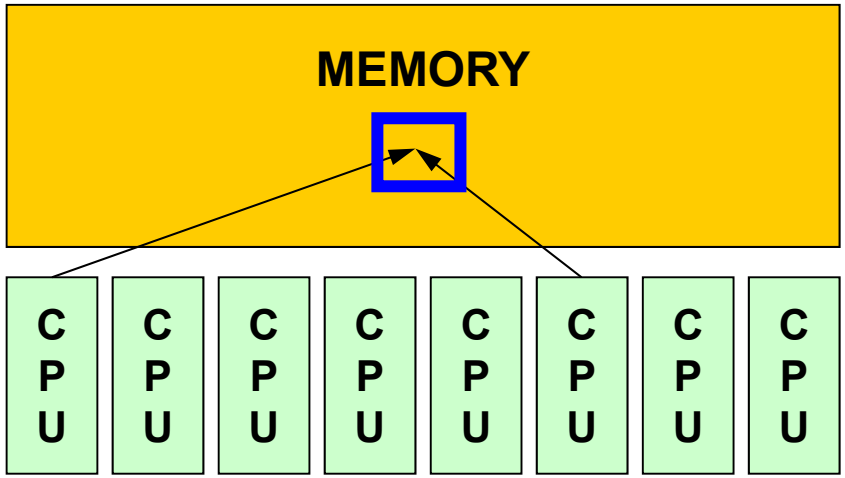
There are no k-mesh which is adjacent to both of i and j simultaneously. Therefore, $l_{ij} = a_{ij}$.

In this case, ALlu0/AUlu0 could be changed



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

SMP



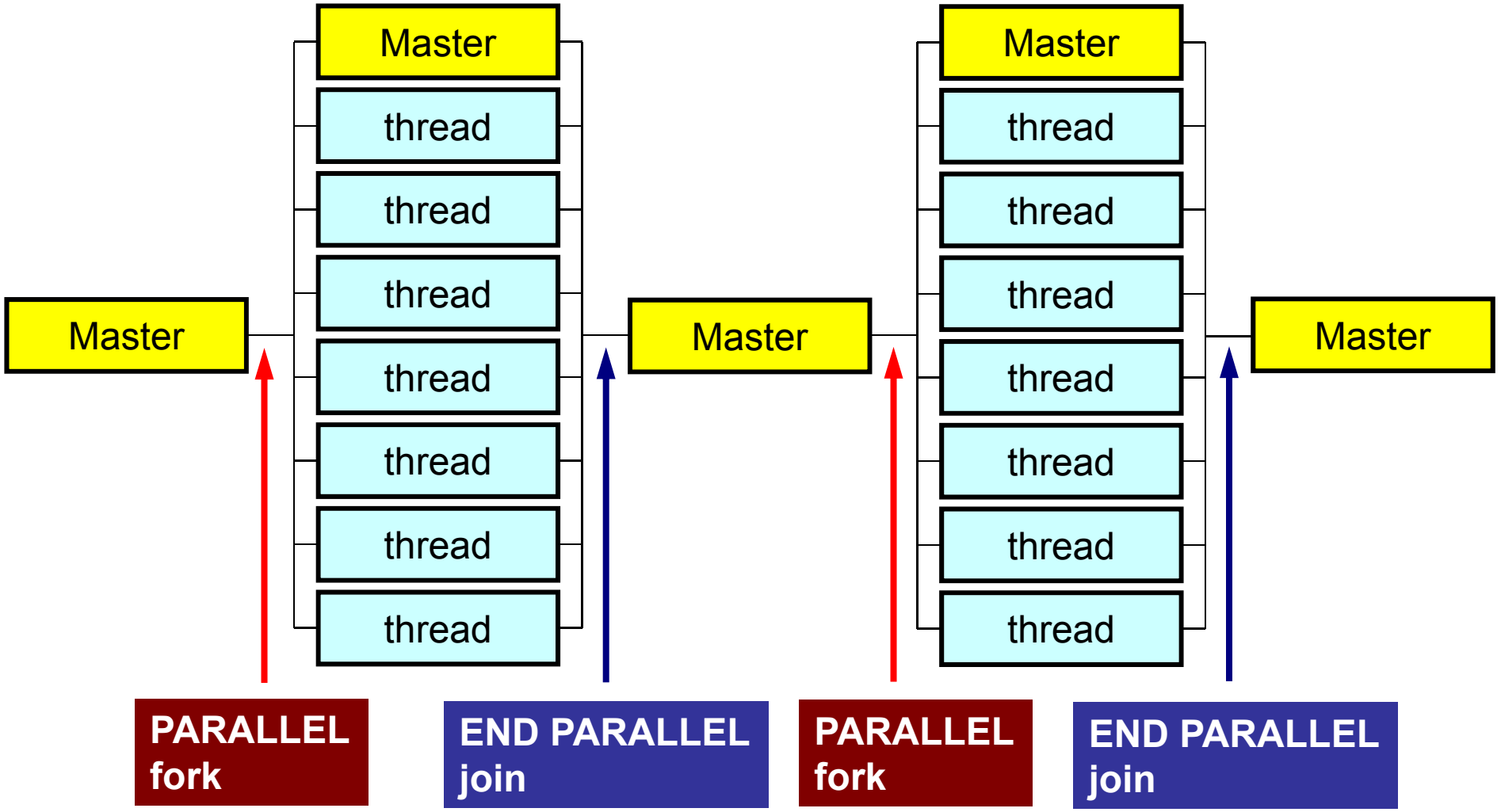
- SMP
  - Symmetric Multi Processors
  - Multiple CPU's (cores) share a single memory space

What is OpenMP ?

http://www.openmp.org

- An API for multi-platform shared-memory parallel programming in C/C++ and Fortran
  - Current version: 3.X: 4.0 comes soon
- Background
  - Merger of Cray and SGI in 1996
  - ASCI project (DOE) started
- C/C++ version and Fortran version have been separately developed until ver.2.5.
- Fork-Join Parallel Execution Model
- Users have to specify everything by directives.
  - Nothing happen, if there are no directives

Fork-Join Parallel Execution Model



Number of Threads

- **OMP_NUM_THREADS**
  - How to change ?
    - bash(.bashrc) **export OMP_NUM_THREADS=8**
    - csh(.cshrc) **setenv OMP_NUM_THREADS 8**

Information about OpenMP

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- References
  - Chandra, R. et al.「Parallel Programming in OpenMP」（Morgan Kaufmann）
  - Quinn, M.J.「Parallel Programming in C with MPI and OpenMP」（McGrawHill）
  - Mattson, T.G. et al.「Patterns for Parallel Programming」（Addison Wesley）
  - 牛島「OpenMPによる並列プログラミングと数値計算法」（丸善）
  - Chapman, B. et al.「Using OpenMP」（MIT Press）最新!
- Japanese Version of OpenMP 3.0 Spec. (Fujitsu etc.)
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

Features of OpenMP

- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.

OpenMP/Directives Array Operations

Simple Substitution

```
!$omp parallel do
      do i= 1, NP
        W(i,1)= 0.d0
        W(i,2)= 0.d0
      enddo
!$omp end parallel do
```

DAXPY

```
!$omp parallel do
      do i= 1, NP
        Y(i)= ALPHA*X(i) + Y(i)
      enddo
!$omp end parallel do
```

Dot Products

```
!$omp parallel do private(iS,iE,i)
!$omp&             reduction(+:RHO)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          RHO= RHO + W(i,R)*W(i,Z)
        enddo
      enddo
!$omp end parallel do
```

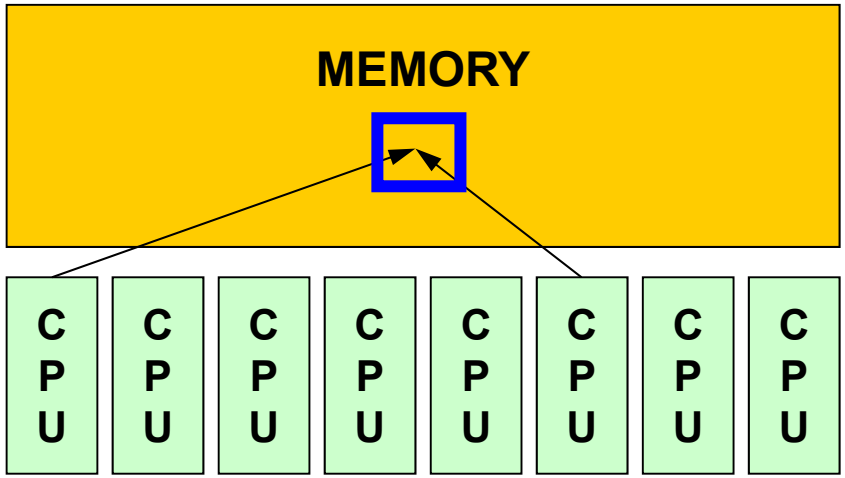
OpenMP/Direceives Matrix/Vector Products

```
!$omp parallel do private(ip,iS,iE,i,j)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          W(i,Q)= D(i)*W(i,P)
          do j= 1, INL(i)
            W(i,Q)= W(i,Q) + W(IAL(j,i),P)
          enddo
          do j= 1, INU(i)
            W(i,Q)= W(i,Q) + W(IAU(j,i),P)
          enddo
        enddo
      enddo
!$omp end parallel do
```

Features of OpenMP

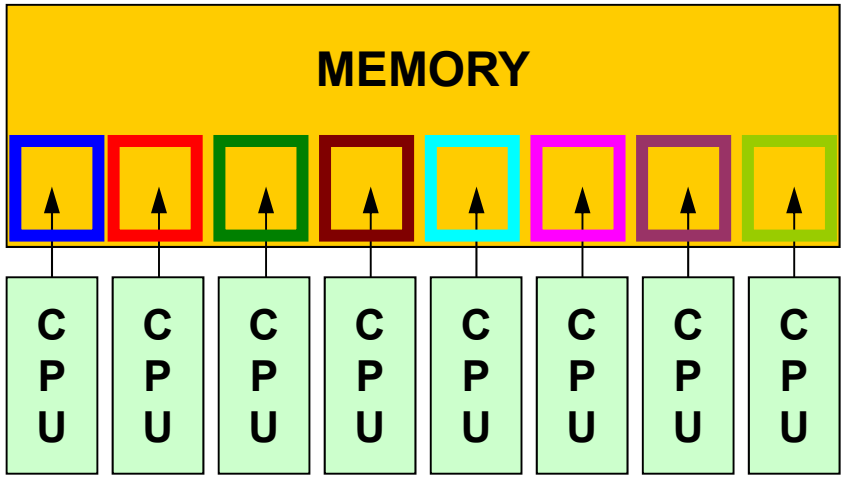
- Directives
  - Loops right after the directives are parallelized.
  - If the compiler does not support OpenMP, directives are considered as just comments.
- Nothing happen without explicit directives
  - Different from “automatic parallelization/vectorization”
  - Something wrong may happen by un-proper way of usage
  - Data configuration, ordering etc. are done under users’ responsibility
- “Threads” are created according to the number of cores on the node
  - Thread: “Process” in MPI
  - Generally, “# threads = # cores”: Xeon Phi supports 4 threads per core (Hyper Multithreading)

Memory Contention: メモリ競合



- During a complicated process, multiple threads may simultaneously try to update the data in same address on the memory.
  - e.g.: Multiple cores update a single component of an array.
  - This situation is possible.
  - Answers may change compared to serial cases with a single core (thread).

Memory Contention (cont.)



- In this lecture, no such case does not happen by reordering etc.
  - In OpenMP, users are responsible for such issues (e.g. proper data configuration, reordering etc.)
- Generally speaking, performance per core reduces as number of used cores (thread number) increases.
  - Memory access performance: STREAM

Features of OpenMP (cont.)

- “!omp parallel do”-”!omp end parallel do”
- Global (Shared) Variables, Private Variables
  - Default: Global (Shared)
  - Dot Products: reduction

```
!$omp parallel do private(iS,iE,i)
!$omp&            reduction(+:RHO)
      do ip= 1, PEsmpTOT
        iS= STACKmcG(ip-1) + 1
        iE= STACKmcG(ip  )
        do i= iS, iE
          RHO= RHO + W(i,R)*W(i,Z)
        enddo
      enddo
!$omp end parallel do
```

W(:,:), R, Z, PEsmpTOT
global (shared)

FORTRAN & C

```
use omp_lib
...
!$omp parallel do shared(n,x,y) private(i)
      do i= 1, n
        x(i)= x(i) + y(i)
      enddo
!$ omp end parallel do
```

```
#include <omp.h>
...
{
  #pragma omp parallel for default(none) shared(n,x,y) private(i)

    for (i=0; i<n; i++)
        x[i] += y[i];
}
```

In this class ...

- There are many capabilities of OpenMP.
- In this class, only several functions are shown for parallelization of ICCG solver.

First things to be done

- use omp_lib Fortran
- #include <omp.h> C

OpenMP Directives (Fortran)

```
sentinel directive_name [clause[[,] clause]…]
```

- NO distinctions between upper and lower cases.
- sentinel
  - Fortran: !OMP, COMP, *$OMP
    - !$OMP only for free format
  - Continuation Lines (Same rule as that of Fortran compiler is applied)
    - Example for **`!$OMP PARALLEL DO SHARED(A,B,C)`**

```
!$OMP PARALLEL DO
!$OMP+SHARED (A,B,C)
```

```
!$OMP PARALLEL DO &
!$OMP SHARED (A,B,C)
```

OpenMP Directives (C)

```
#pragma omp directive_name [clause[[,] clause]…]
```

- “＼” for continuation lines
- Only lower case (except names of variables)

```
#pragma omp parallel for shared (a,b,c)
```

PARALLEL DO

```
!$OMP PARALLEL DO[clause[[,] clause] … ]
  (do_loop)
!$OMP END PARALLEL DO
```

```
#pragma parallel for [clause[[,] clause] … ]
  (for_loop)
```

- Parallerize DO/for Loops
- Examples of “clause”
  - PRIVATE（list）
  - SHARED（list）
  - DEFAULT（PRIVATE|SHARED|NONE）
  - REDUCTION（{operation|intrinsic}: list）

REDUCTION

```
REDUCTION ({operator|instinsic}: list)
```

```
reduction ({operator|instinsic}: list)
```

- Similar to “MPI_Reduce”
- Operator
  - +, *, -, .AND., .OR., .EQV., .NEQV.
- Intrinsic
  - MAX, MIN, IAND, IOR, IEQR

Example-1: A Simple Loop

```
!$OMP PARALLEL DO
      do i= 1, N
        B(i)= (A(i) + B(i)) * 0.50
      enddo
!$OMP END PARALLEL DO
```

- Default status of loop variables (“i” in this case) is private. Therefore, explicit declaration is not needed.
- “END PARALLEL DO” is not required
  - In C, there are no definitions of “end parallel do”

Example-1: REDUCTION

```
!$OMP PARALLEL DO DEFAULT(PRIVATE) REDUCTION(+:A,B)
      do i= 1, N
        call WORK (Alocal, Blocal)
        A= A + Alocal
        B= B + Blocal
      enddo
!$OMP END PARALLEL DO
```

- “END PARALLEL DO” is not required

OpenMP for Dot Products

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

OpenMP for Dot Products

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

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

Directives are just inserted.

OpenMP for Dot Products

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

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(i) REDUCTION(+:VAL)
      do i= 1, N
        VAL= VAL + W(i,R) * W(i,Z)
      enddo
!$OMP END PARALLEL DO
```

Directives are just inserted.

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
PEsmpTOT: Number of threads

Additional array **INDEX(:)** is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.

OpenMP for Dot Products

```
      VAL= 0.d0
!$OMP PARALLEL DO PRIVATE(ip,i) REDUCTION(+:VAL)
      do ip= 1, PEsmpTOT
        do i= index(ip-1)+1, index(ip)
          VAL= VAL + W(i,R) * W(i,Z)
        enddo
      enddo
!$OMP END PARALLEL DO
```

Multiple Loop
PEsmpTOT: Number of threads

Additional array **INDEX(:)** is needed.
Efficiency is not necessarily good, but users can specify thread for each component of data.

e.g.: N=100, PEsmpTOT=4

INDEX(0)= 0
INDEX(1)= 25
INDEX(2)= 50
INDEX(3)= 75
INDEX(4)= 100

Files on Oakleaf-FX

```
>$ cd <$O-TOP>

>$ cp /home/z30088/class_eps/F/multicore.tar .
>$ cp /home/z30088/class_eps/C/multicore.tar .

>$ tar xvf multicore.tar

>$ cd multicore
```

Confirm Directories:

```
      L3 omp
```

<$O-L3>, <$O-omp>

Files on Oakleaf-FX

```
>$ cd <$O-omp>

>$ frtpx –Kfast,openmp test.f
>$ fccpx –Kfast,openmp test.c

>$ pjsub go.sh
```

Running the Job

go.sh

```
#!/bin/sh
#PJM -L "node=1"
#PJM -L "elapse=00:10:00"
#PJM -L "rscgrp=lecture"
#PJM -g "gt71"
#PJM -j
#PJM -o "t0-08.lst"

export OMP_NUM_THREADS=8      Number of Threads/process

./a.out < INPUT.DAT
```

INPUT.DAT

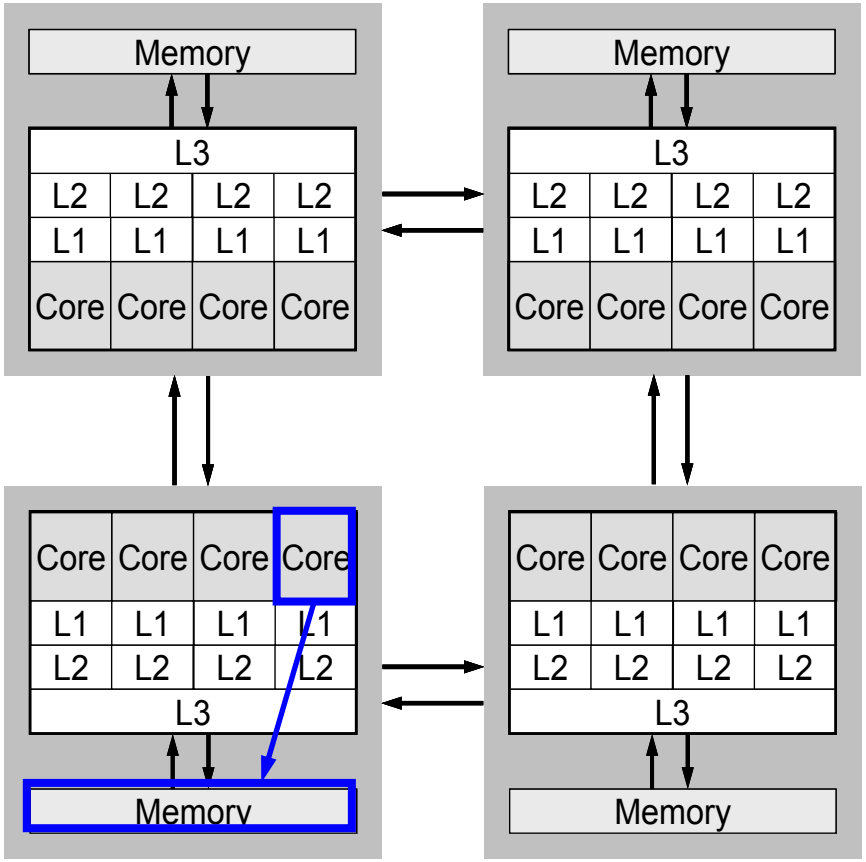
```
N  nopt

N:    Problem Size (Vector Length)
nopt: First-touch  (0:No, 1:Yes)
```

test.f/test.c

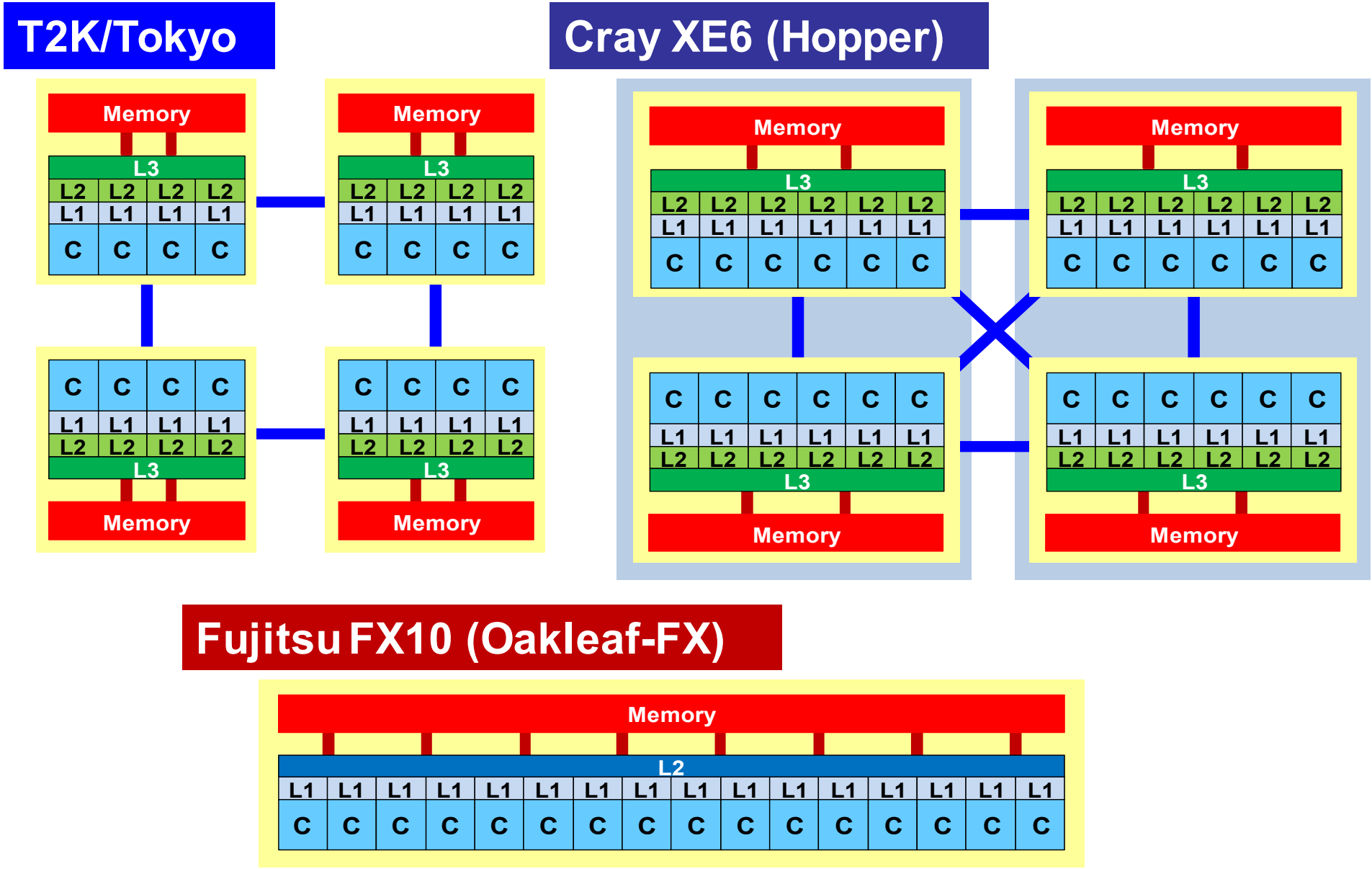
- DAXPY
- Dot Products

- Effect of OpenMP Directives
- Effect of First Touch Data Placement

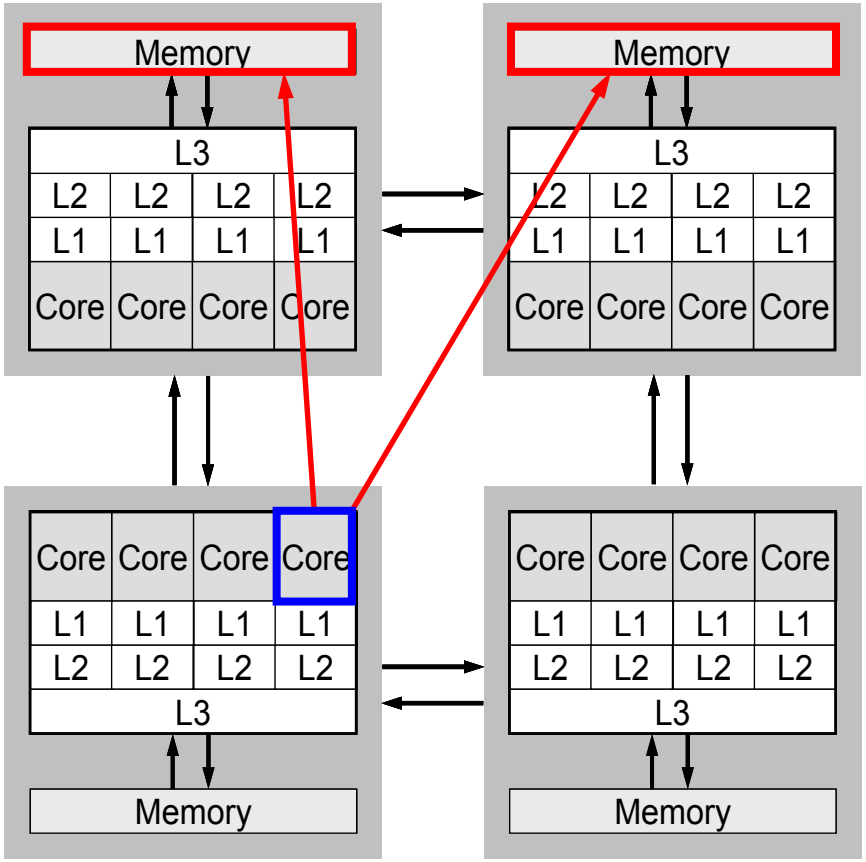
NUMA Architecture
Non-Uniform Memory Access



- Data should be on the local memory
  - Local memory: Memory of CPU socket where the core locates.

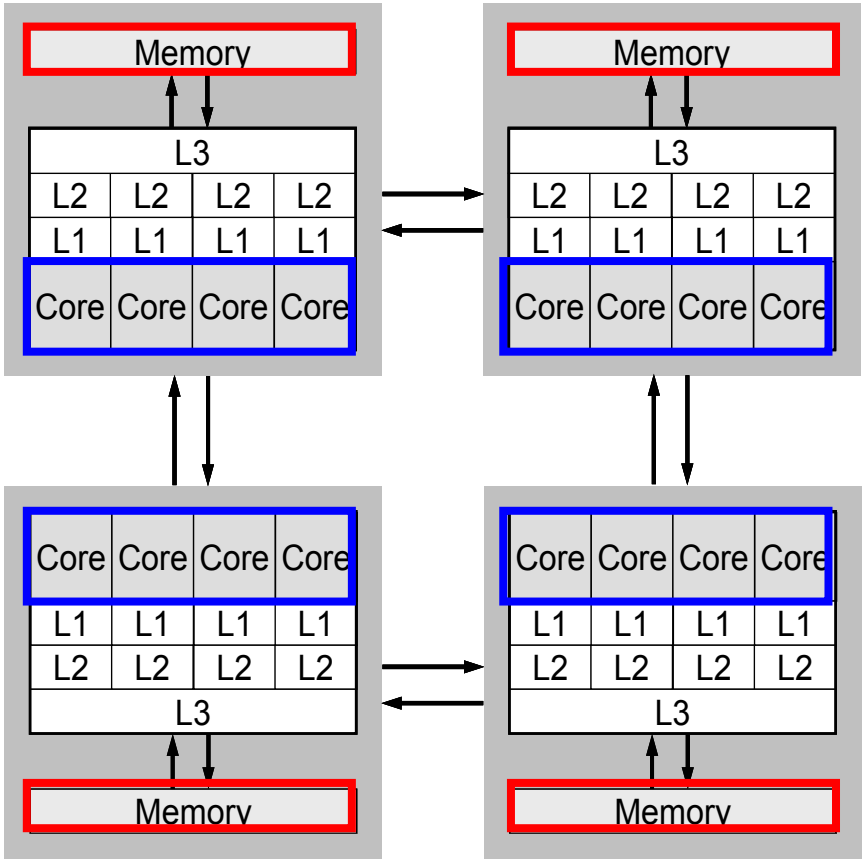

T2K/Tokyo
Memory
L3
L2
L1
C
Cray XE6 (Hopper)
Fujitsu FX10 (Oakleaf-FX)

NUMA Architecture
Non-Uniform Memory Access



- If data are on remote memory, it is not efficient.

NUMA Architecture
Non-Uniform Memory Access



- "First touch data placement" can keep data on local memory.
- "Parallel initialization" of arrays is effective.

First Touch Data Placement

"Patterns for Parallel Programming" Mattson, T.G. et al.

To reduce memory traffic in the system, it is important to keep the data close to the PEs that will work with the data (e.g. NUMA control).

On NUMA computers, this corresponds to making sure the pages of memory are allocated and "owned" by the PEs that will be working with the data contained in the page.

The most common NUMA page-placement algorithm is the "first touch" algorithm, in which the PE first referencing a region of memory will have the page holding that memory assigned to it.

A very common technique in OpenMP program is to initialize data in parallel using the same loop schedule as will be used later in the computations.

test.f (1/3): Initialization

```
      use omp_lib
      implicit REAL*8 (A-H,O-Z)
      real(kind=8), dimension(:), allocatable :: X, Y
      real(kind=8), dimension(:), allocatable :: Z1, Z2
      real(kind=8), dimension(:), allocatable :: Z3, Z4, Z5
      integer, dimension(0:2) :: INDEX
!C
!C +------+
!C | INIT |
!C +------+
!C===
      write (*,*) 'N, nopt ?'
      read  (*,*)  N, nopt

      allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
      if (nopt.eq.0) then
        X = 1.d0
        Y = 1.d0
        Z1= 0.d0
        Z2= 0.d0
        Z3= 0.d0
        Z4= 0.d0
        Z5= 0.d0
       else
!$omp parallel do private (i)
        do i= 1, N
          X (i)= 0.d0
          Y (i)= 0.d0
          Z1(i)= 0.d0
          Z2(i)= 0.d0
          Z3(i)= 0.d0
          Z4(i)= 0.d0
          Z5(i)= 0.d0
        enddo
!$omp end parallel do
      endif

      ALPHA= 1.d0
!C===
```

Problem Size, Option

nopt＝0 NO First Touch
nopt≠0 with First Touch

test.f (1/3): Initialization

```
      use omp_lib
      implicit REAL*8 (A-H,O-Z)
      real(kind=8), dimension(:), allocatable :: X, Y
      real(kind=8), dimension(:), allocatable :: Z1, Z2
      real(kind=8), dimension(:), allocatable :: Z3, Z4, Z5
      integer, dimension(0:2) :: INDEX
!C
!C +------+
!C | INIT |
!C +------+
!C===
      write (*,*) 'N, nopt ?'
      read  (*,*)  N, nopt

      allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
      if (nopt.eq.0) then
        X = 1.d0
        Y = 1.d0
        Z1= 0.d0
        Z2= 0.d0
        Z3= 0.d0
        Z4= 0.d0
        Z5= 0.d0
       else
!$omp parallel do private (i)
        do i= 1, N
          X (i)= 0.d0
          Y (i)= 0.d0
          Z1(i)= 0.d0
          Z2(i)= 0.d0
          Z3(i)= 0.d0
          Z4(i)= 0.d0
          Z5(i)= 0.d0
        enddo
!$omp end parallel do
      endif

      ALPHA= 1.d0
!C===
```

nopt＝0
NO First Touch

Initialization is not parallelized

test.f (1/3): Initialization

```
      use omp_lib
      implicit REAL*8 (A-H,O-Z)
      real(kind=8), dimension(:), allocatable :: X, Y
      real(kind=8), dimension(:), allocatable :: Z1, Z2
      real(kind=8), dimension(:), allocatable :: Z3, Z4, Z5
      integer, dimension(0:2) :: INDEX
!C
!C +------+
!C | INIT |
!C +------+
!C===
      write (*,*) 'N, nopt ?'
      read  (*,*)  N, nopt

      allocate (X(N), Y(N), Z1(N), Z2(N), Z3(N), Z4(N), Z5(N))
      if (nopt.eq.0) then
        X = 1.d0
        Y = 1.d0
        Z1= 0.d0
        Z2= 0.d0
        Z3= 0.d0
        Z4= 0.d0
        Z5= 0.d0
       else
!$omp parallel do private (i)
        do i= 1, N
          X (i)= 0.d0
          Y (i)= 0.d0
          Z1(i)= 0.d0
          Z2(i)= 0.d0
          Z3(i)= 0.d0
          Z4(i)= 0.d0
          Z5(i)= 0.d0
        enddo
!$omp end parallel do
      endif

      ALPHA= 1.d0
!C===
```

nopt≠0
with First Touch

Initialization is in parallel manner.

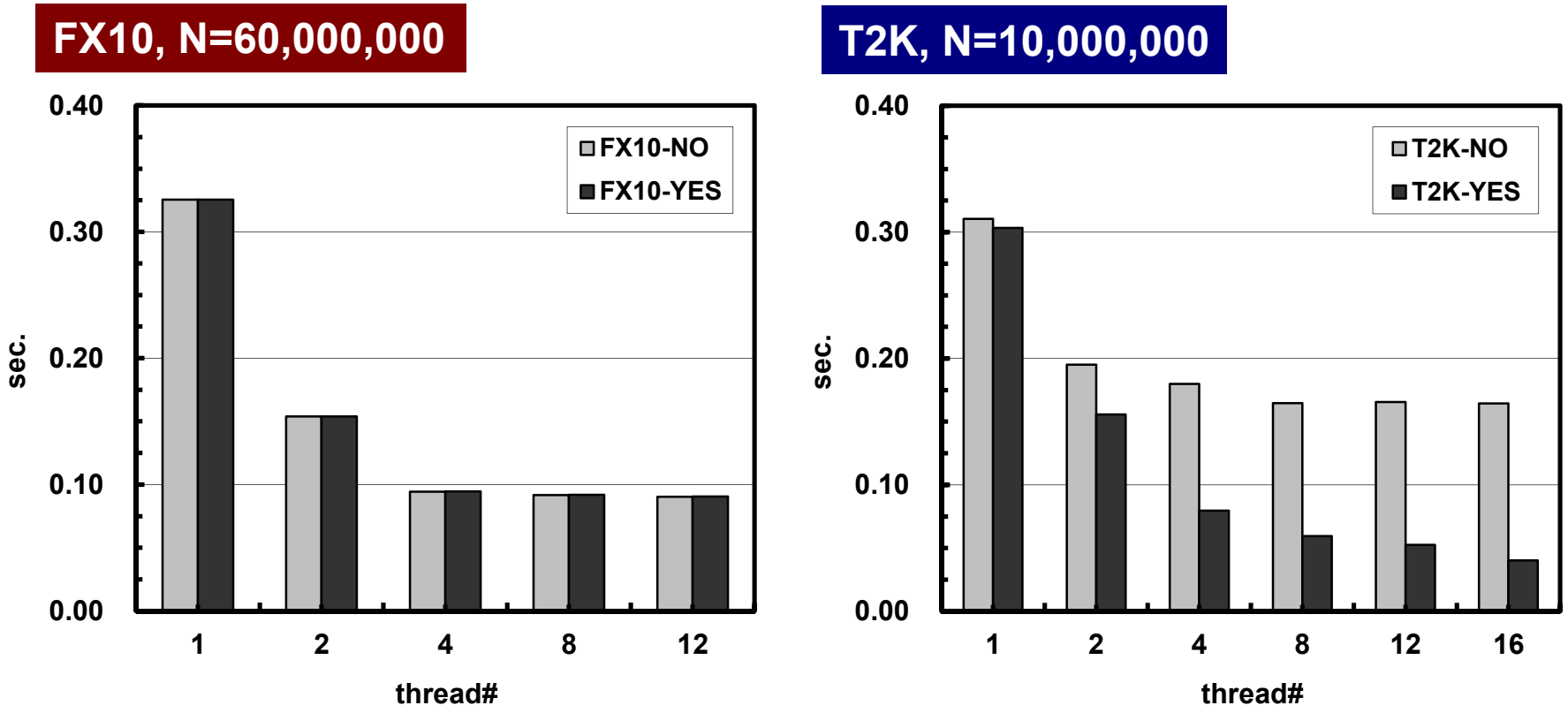
test.f (2/3): DAXPY

```
!C
!C +-------+
!C | DAXPY |
!C +-------+
!C===
      S2time= omp_get_wtime()
!$omp parallel do private (i)
        do i= 1, N
          Z1(i)= ALPHA*X(i) + Y(i)
          Z2(i)= ALPHA*X(i) + Y(i)
          Z3(i)= ALPHA*X(i) + Y(i)
          Z4(i)= ALPHA*X(i) + Y(i)
          Z5(i)= ALPHA*X(i) + Y(i)
        enddo
!$omp end parallel do
      E2time= omp_get_wtime()

      write (*,'(/a)')           '# DAXPY'
      write (*,'( a, 1pe16.6)') '  omp-1 ', E2time - S2time
!C===
```

test.f (3/3): Dot Products

```
!C
!C +-----+
!C | DOT |
!C +-----+
!C===
      V1= 0.d0
      V2= 0.d0
      V3= 0.d0
      V4= 0.d0
      V5= 0.d0
      S2time= omp_get_wtime()
!$omp parallel do private(i) reduction (+:V1,V2,V3,V4,V5)
        do i= 1, N
          V1= V1 + X(i)*(Y(i)+1.d0)
          V2= V2 + X(i)*(Y(i)+2.d0)
          V3= V3 + X(i)*(Y(i)+3.d0)
          V4= V4 + X(i)*(Y(i)+4.d0)
          V5= V5 + X(i)*(Y(i)+5.d0)
        enddo
!$omp end parallel do
      E2time= omp_get_wtime()

      write (*,'(/a)')          '# DOT'
      write (*,'( a, 1pe16.6)') '  omp-1 ', E2time - S2time
!C===
      stop
      end
```

DAXPY: Effect of First Touch



- T2K: Effect of first touch is large
- NOT scalable: memory contention, synchronization overhead

test.c: Dummy Pragma needed

```
#pragma omp parallel
  {
}

        if (nopt==0) {for(i=0; i<N; i++) {
                X[i]  = 1.0;
                Y[i]  = 1.0;
                Z1[i] = 0.0;
                Z2[i] = 0.0;
                Z3[i] = 0.0;
                Z4[i] = 0.0;
                Z5[i] = 0.0;}
        }else{
#pragma omp parallel for private(i)
                for(i=0; i<N; i++) {
                        X[i]  = 1.0;
                        Y[i]  = 1.0;
                        Z1[i] = 0.0;
                        Z2[i] = 0.0;
                        Z3[i] = 0.0;
                        Z4[i] = 0.0;
                        Z5[i] = 0.0;
                }
        }
```

Dummy Pragma needed before actual computation (for Fujitsu C compiler). This problem may be fixed. Threads for OpenMP are generated during the first parallel loop, and this procedure is expensive. In Fortran, this dummy loop is implicitly inserted by compiler.

Parallelize ICCG Method

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

Matrix-Vector Multiply

```
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
```

Matrix-Vector Multiply

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

Matrix-Vector Multiply: Other Approach

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

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
```

Forward Substitution
Four Thread Parallel Operation

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

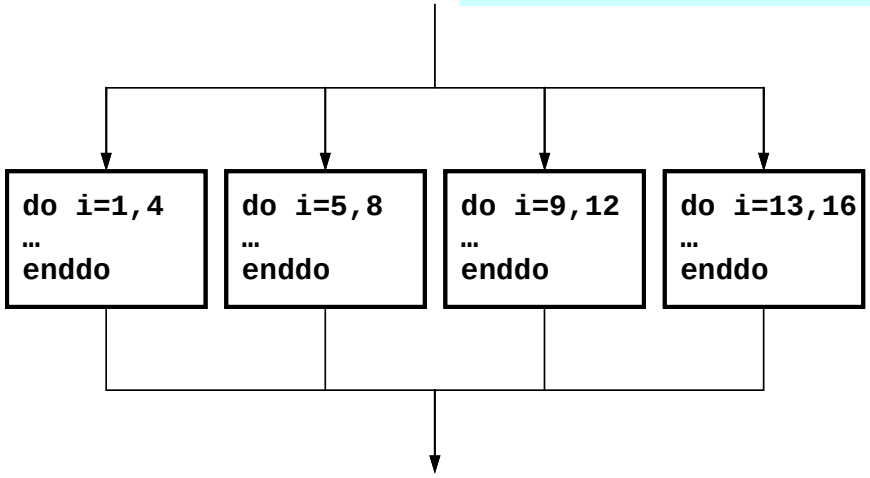
```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

Forward Substitution
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
```



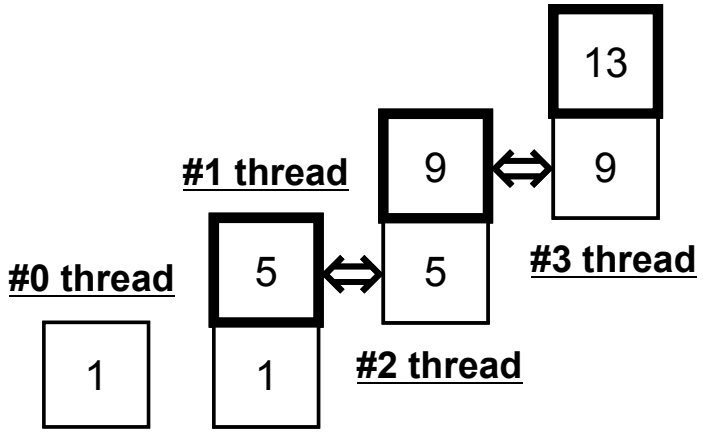
These four threads are executed simultaneously.

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

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

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



⇔:
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