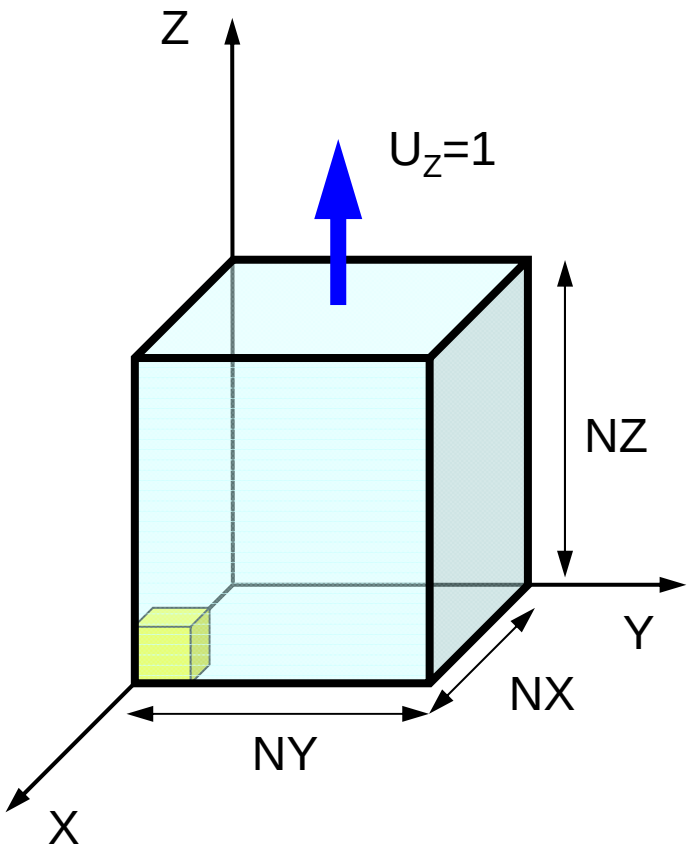
3D Code for Static Linear-Elastic Problems (1/3) Overview

Kengo Nakajima
Information Technology Center

Technical & Scientific Computing I (4820-1027)
Seminar on Computer Science I (4810-1204)

Target Application



- Elastic Material
  - Young's Modulus E
  - Poisson's Ratio ν
- Rectangular Prism
  - 1x1x1 cubes (hexahedra)
  - NX, NY, NZ cubes in each direction
- Boundary Conditions
  - Symmetric B.C.
    - U_X=0@X=0
    - U_Y=0@Y=0
    - U_Z=0@Z=0
  - Dirichlet B.C.
    - U_Z=1@Z=Z_{max}

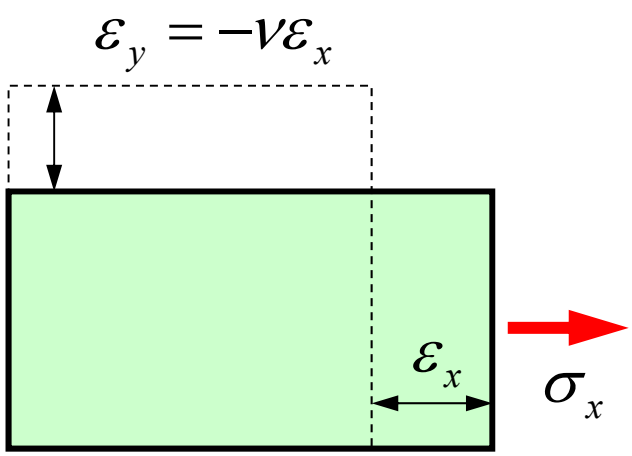
movie

Constitutive Eqn's: Stress-Strain

- Young's Modulus E
  - Stress-Strain: Proportional
  - Proportionality: E (depends on material)

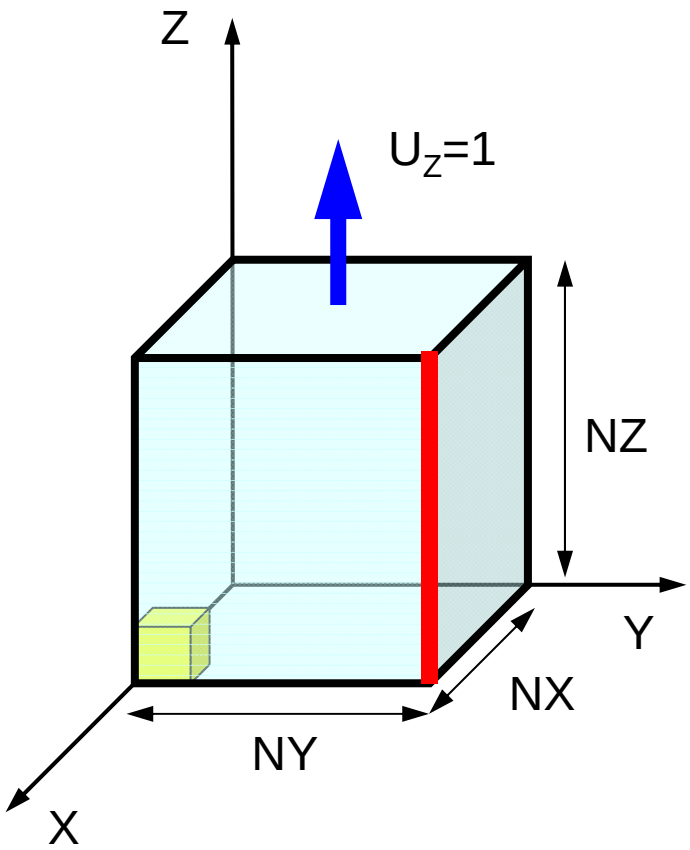
$$\sigma_x = E\varepsilon_x, \quad \varepsilon_x = \frac{\sigma_x}{E}$$

- Poisson's Ratio ν
  - Body deforms in Y- and Z-directions, even if external force is in X-direction.
  - Poisson's ratio is proportionality for this lateral strain.
    - depends on material
      - Metal: 0.30
      - Rubber, Water: 0.50 (incompressible)



$$\varepsilon_y = -\nu\varepsilon_x = -\nu\frac{\sigma_x}{E}$$

NX=NY=NZ=10, ν=0.30



$$\varepsilon_z = \frac{1}{10} = 0.10$$

$$\varepsilon_x = \varepsilon_y = -\nu\varepsilon_z = -0.03$$

$$\therefore u_x\Big|_{X=10,Y=10} = u_y\Big|_{X=10,Y=10} = 10 \times \varepsilon_x = -0.30$$

Finite-Element Procedures

- Governing Equations
- Galerkin Method: Weak Form
- Element-by-Element Integration
  - Element Matrix
- Global Matrix
- Boundary Conditions
- Linear Solver

FEM Procedures: Program

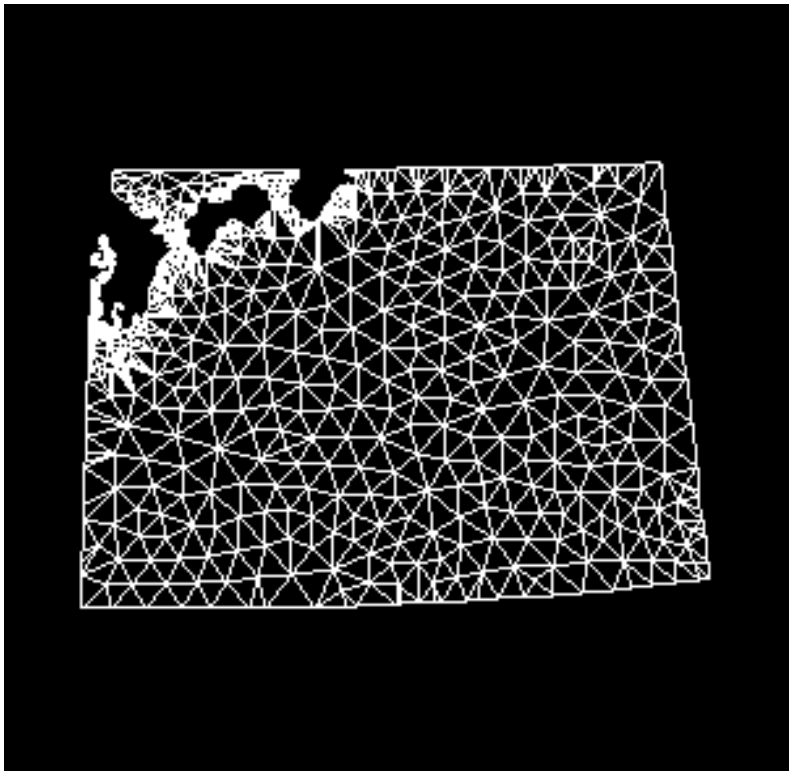
- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method
- Calculation of Stress

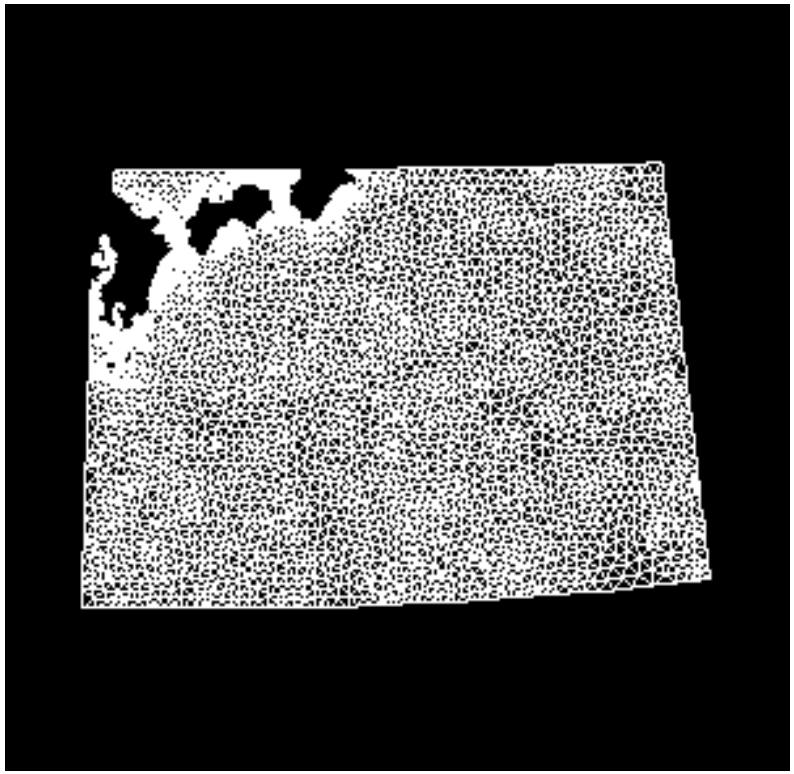
- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- Running the Code
- Data Structure
- Overview of the Program
- Computational Issue
- Visualization by ParaView

Extension to 2D Prob.: Triangles
三角形要素

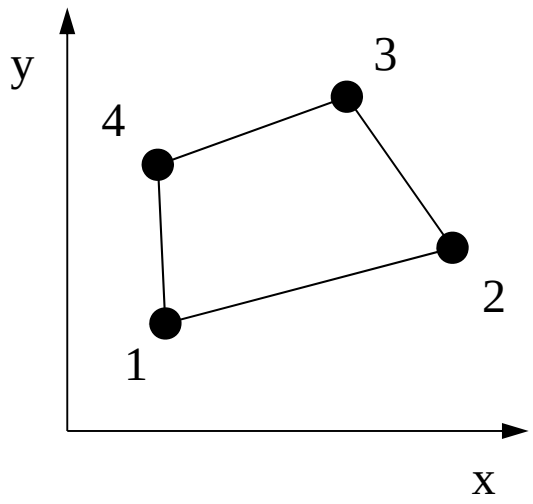
- Triangles can handle arbitrarily shaped object
- “Linear” triangular elements provide low accuracy, therefore they are not used in practical applications.





Extension to 2D Prob.: Quadrilaterals
四角形要素

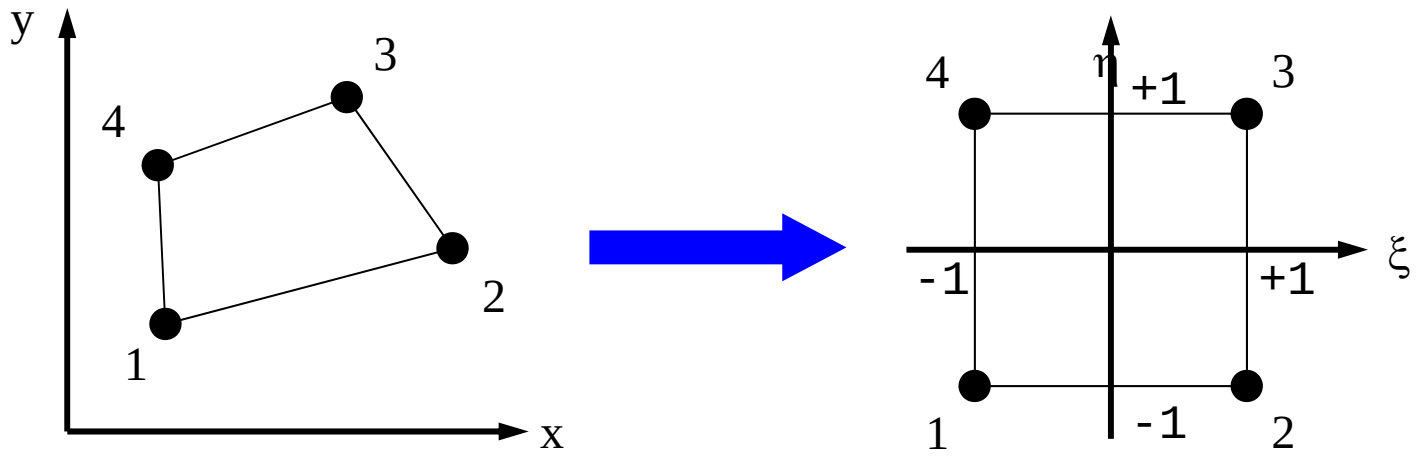
- Formulation of quad. elements is possible if same shape functions in 1D elements are applied along X- and Y- axis.
  - More accurate than triangles
- Each edge must be “parallel” with X- and Y- axis.
  - Similar to FDM



- This type of elements cannot be considered.

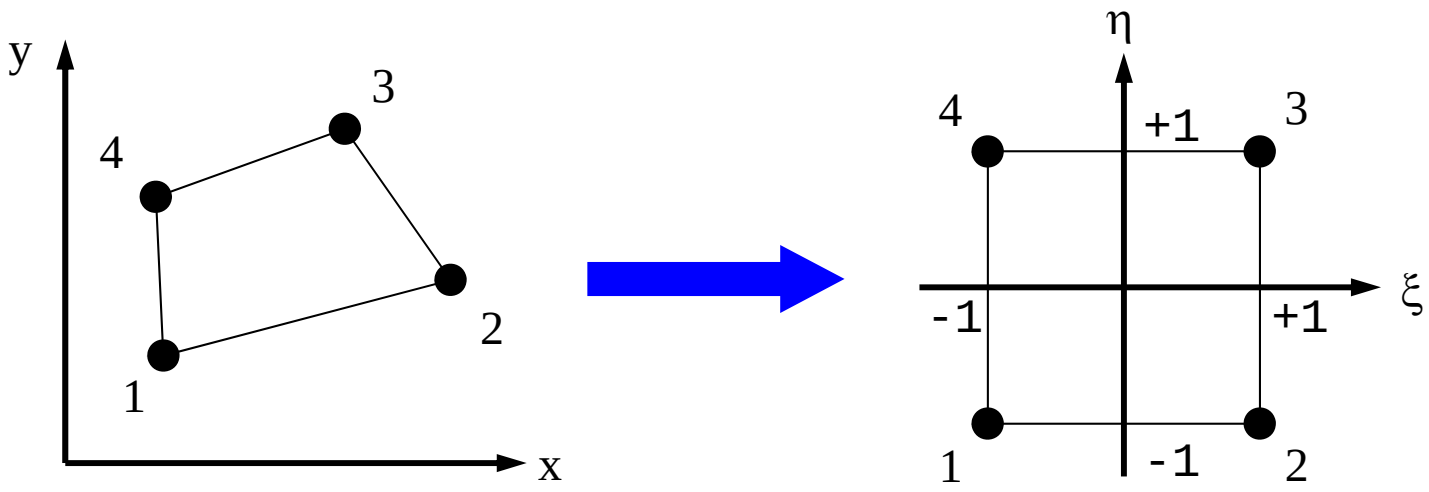
Isoparametric Element (1/3)

- Each element is mapped to square element [±1,±1] on natural/local coordinate (ξ,η)（自然／局所座標系）



- Components of global coordinate system of each node (x,y) for certain kinds of elements are defined by shape functions [N] on natural/local coordinate system, where shape functions [N] are also used for interpolation of dependent variables.

Isoparametric Element (2/3)

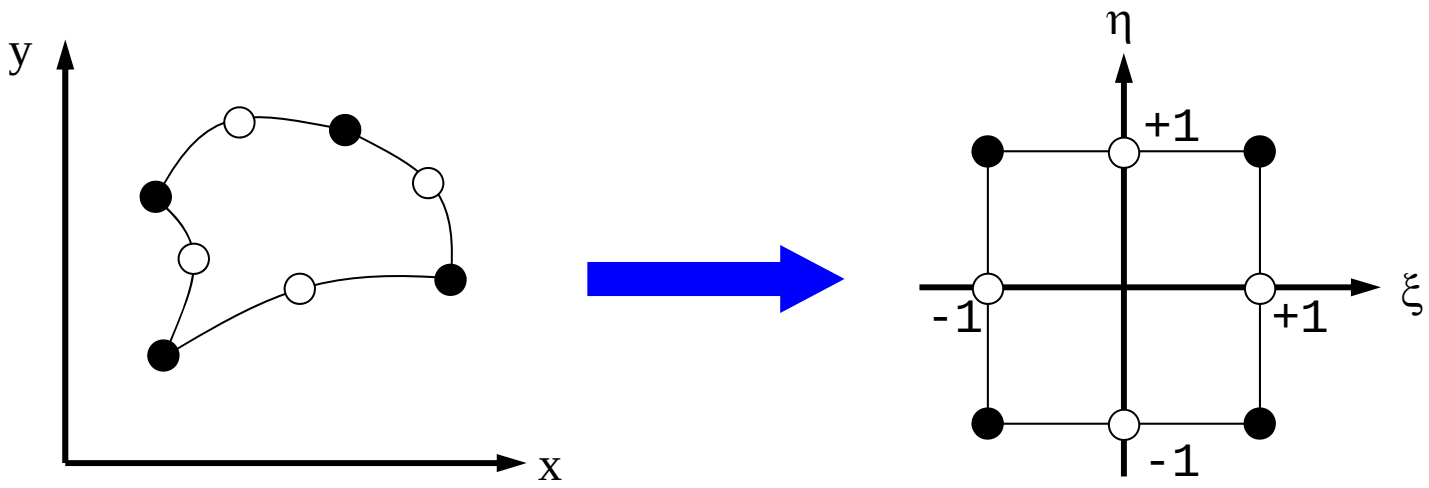


- Coordinate of each node: (x_1,y_1), (x_2,y_2), (x_3,y_3), (x_4,y_4)
- Deformation (in X-direction) of each node: u_1, u_2, u_3, u_4

$$u = \sum_{i=1}^{4} N_i(\xi,\eta)\cdot u_i$$

$$x = \sum_{i=1}^{4} N_i(\xi,\eta)\cdot x_i,\ y = \sum_{i=1}^{4} N_i(\xi,\eta)\cdot y_i$$

Isoparametric Element (3/3)

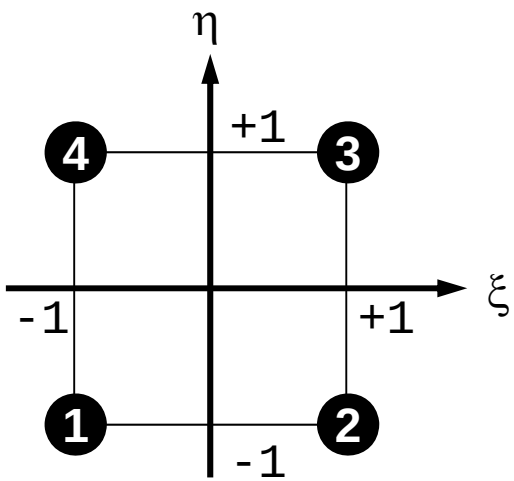


- Higher-order shape function can handle curved lines/surfaces.
- “Natural” coordinate system

Sub-Parametric
Super-Parametric

Shape Fn's on 2D Natural Coord. (1/3)

- Polynomial shape functions on squares of natural coordinate:



$$u = \alpha_1 + \alpha_2 \xi + \alpha_3 \eta + \alpha_4 \xi \eta$$

- Coefficients are calculated as follows:

$$\alpha_1 = \frac{u_1 + u_2 + u_3 + u_4}{4}, \quad \alpha_2 = \frac{-u_1 + u_2 + u_3 - u_4}{4},$$

$$\alpha_3 = \frac{-u_1 - u_2 + u_3 + u_4}{4}, \quad \alpha_4 = \frac{u_1 - u_2 + u_3 - u_4}{4}$$

Shape Fn's on 2D Natural Coord. (2/3)

$$u = \alpha_1 + \alpha_2\xi + \alpha_3\eta + \alpha_4\xi\eta$$

$$= \frac{u_1+u_2+u_3+u_4}{4} + \frac{-u_1+u_2+u_3-u_4}{4}\xi +$$

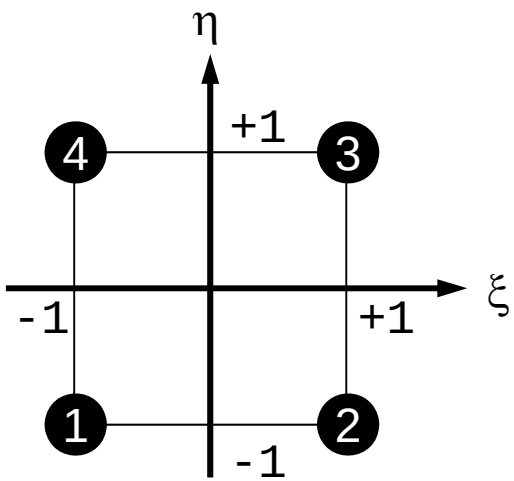
$$\frac{-u_1-u_2+u_3+u_4}{4}\eta + \frac{u_1-u_2+u_3-u_4}{4}\xi\eta$$

$$= \frac{1}{4}(1-\xi-\eta+\xi\eta)u_1 + \frac{1}{4}(1+\xi-\eta-\xi\eta)u_2 +$$

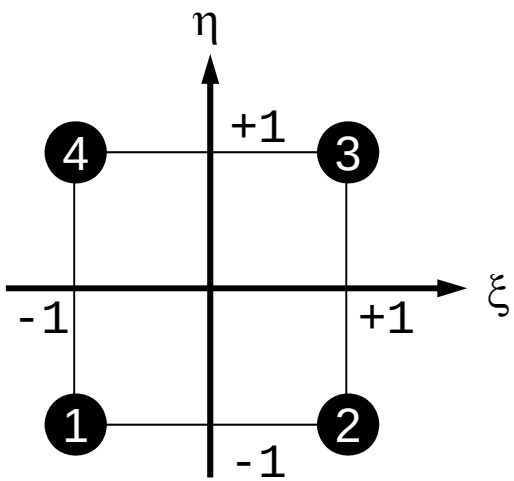
$$\frac{1}{4}(1+\xi+\eta+\xi\eta)u_3 + \frac{1}{4}(1-\xi+\eta-\xi\eta)u_4$$

$$= \frac{1}{4}(1-\xi)(1-\eta)u_1 + \frac{1}{4}(1+\xi)(1-\eta)u_2 +$$

$$\frac{1}{4}(1+\xi)(1+\eta)u_3 + \frac{1}{4}(1-\xi)(1+\eta)u_4$$



Shape Fn's on 2D Natural Coord. (2/3)



$$u = \alpha_1 + \alpha_2\xi + \alpha_3\eta + \alpha_4\xi\eta$$

$$= \frac{u_1+u_2+u_3+u_4}{4} + \frac{-u_1+u_2+u_3-u_4}{4}\xi +$$

$$\frac{-u_1-u_2+u_3+u_4}{4}\eta + \frac{u_1-u_2+u_3-u_4}{4}\xi\eta$$

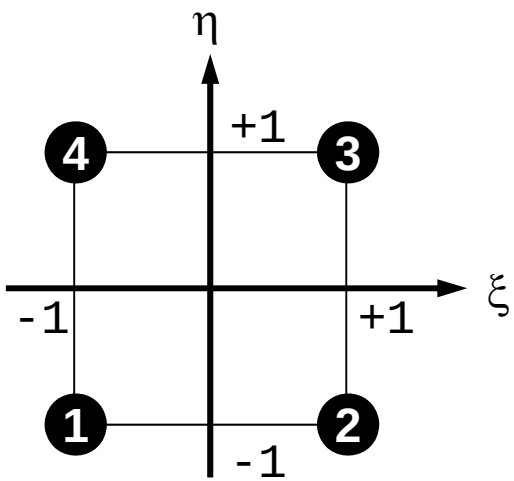
$$= \frac{1}{4}(1-\xi-\eta+\xi\eta)u_1 + \frac{1}{4}(1+\xi-\eta-\xi\eta)u_2 +$$

$$\frac{1}{4}(1+\xi+\eta+\xi\eta)u_3 + \frac{1}{4}(1-\xi+\eta-\xi\eta)u_4$$

$$= \underbrace{\boxed{\frac{1}{4}(1-\xi)(1-\eta)}}_{N_1}u_1 + \underbrace{\boxed{\frac{1}{4}(1+\xi)(1-\eta)}}_{N_2}u_2 +$$

$$\underbrace{\boxed{\frac{1}{4}(1+\xi)(1+\eta)}}_{N_3}u_3 + \underbrace{\boxed{\frac{1}{4}(1-\xi)(1+\eta)}}_{N_4}u_4$$

Shape Fn's on 2D Natural Coord. (3/3)



- u is defined as follows according to u_i :

$$u = N_1 u_1 + N_2 u_2 + N_3 u_3 + N_4 u_4$$

- Shape functions N_i :

$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad N_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta),$$

$$N_3(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad N_4(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

- Also known as "bi-linear" interpolation
- Calculate N_i at each node
- v: deformation in Y-direction

$$v = N_1 v_1 + N_2 v_2 + N_3 v_3 + N_4 v_4$$

Extension to 3D Problems

- Tetrahedron/Tetrahedra（四面体）: Triangles in 2D
  - can handle arbitrary shape objects
  - Linear elements are generally less accurate, not practical
  - Higher-order tetrahedral elements are widely used.

- In this class, “tri-linear” hexahedral elements (isoparametric) are used（六面体要素）
- Displacement-based FEM（変位法）
- DOF: displacement
  - Three components of (u,v,w) are defined on each node.

Shape Fn's: 3D Natural/Local Coord.

$$N_1(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1-\eta)(1-\zeta)$$

$$N_2(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1-\eta)(1-\zeta)$$

$$N_3(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1+\eta)(1-\zeta)$$

$$N_4(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1+\eta)(1-\zeta)$$

$$N_5(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1-\eta)(1+\zeta)$$

$$N_6(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1-\eta)(1+\zeta)$$

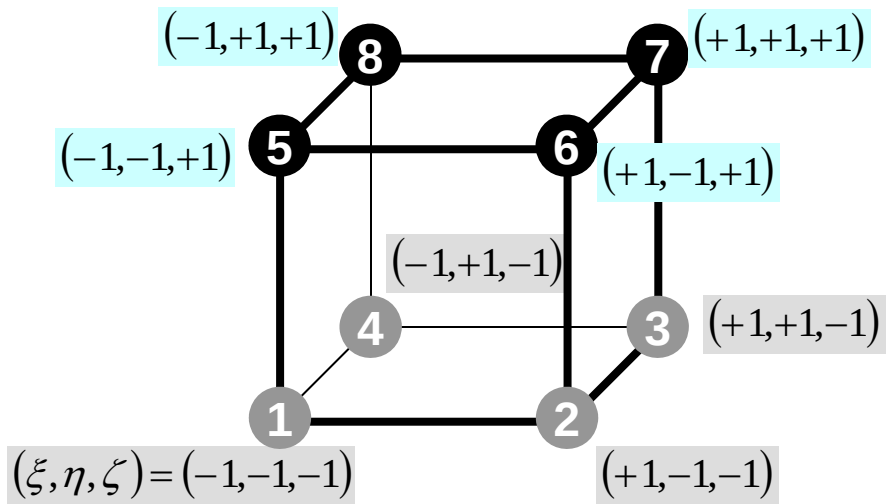
$$N_7(\xi,\eta,\zeta)=\frac{1}{8}(1+\xi)(1+\eta)(1+\zeta)$$

$$N_8(\xi,\eta,\zeta)=\frac{1}{8}(1-\xi)(1+\eta)(1+\zeta)$$

$$u=\sum_{i=1}^{8}N_i(\xi,\eta,\zeta)\cdot u_i$$

$$v=\sum_{i=1}^{8}N_i(\xi,\eta,\zeta)\cdot v_i$$

$$w=\sum_{i=1}^{8}N_i(\xi,\eta,\zeta)\cdot w_i$$



- Formulation of 3D Element
- **Governing Equations of 3D Elastic Problem**
  - **Galerkin Method**
  - **Element Matrices**
- HW

- Running the Code
- Data Structure
- Overview of the Program
- Computational Issue
- Visualization by ParaView

Governing Equations in Theory of Elasticity

- Equilibrium Equations
- Compatibility Conditions: Displacement-Strain
- Constitutive Equations: Stress-Strain

Equilibrium Equations in 3D
6 Independent Stress Components

$$\tau_{xy} = \tau_{yx}$$
$$\tau_{yz} = \tau_{zy}$$
$$\tau_{zx} = \tau_{xz}$$

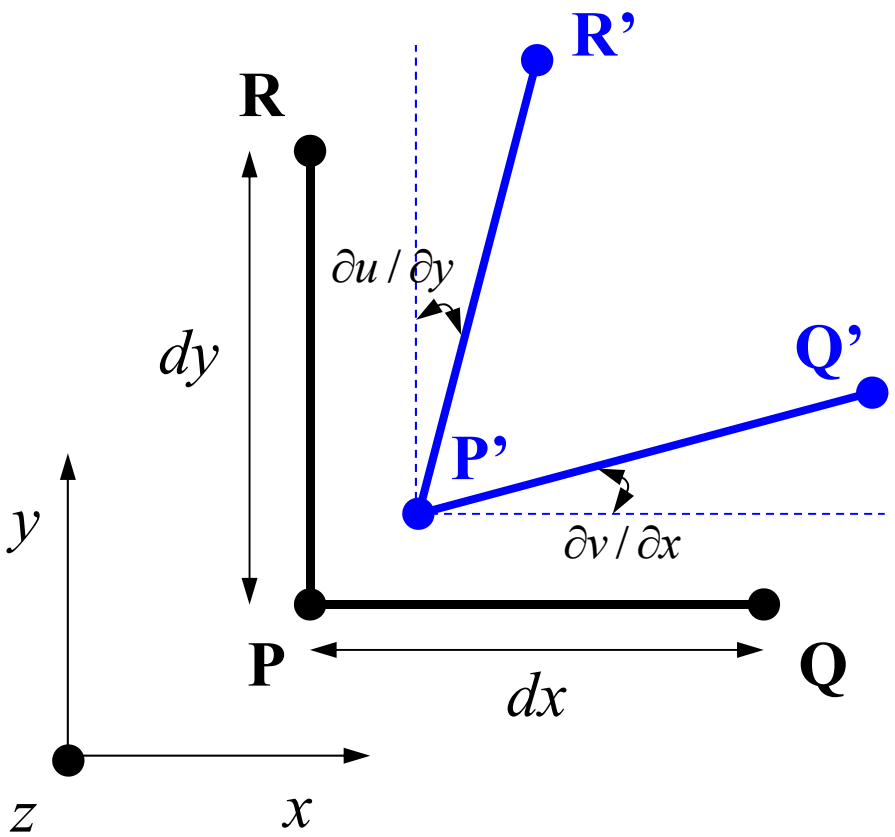
$$\{\boldsymbol{\sigma}\} = \begin{Bmatrix} \sigma_x & \tau_{xy} & \tau_{zx} \\ \tau_{xy} & \sigma_y & \tau_{yz} \\ \tau_{zx} & \tau_{yz} & \sigma_z \end{Bmatrix}$$

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + X = 0$$
$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + Y = 0$$
$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + Z = 0$$

Normal Strain - Displacement

- PQ⇒P'Q'

$$\varepsilon_x = \frac{\left\{\left(x + dx + u + \frac{\partial u}{\partial x} dx\right) - (x + u)\right\} - dx}{dx} = \frac{\partial u}{\partial x}$$

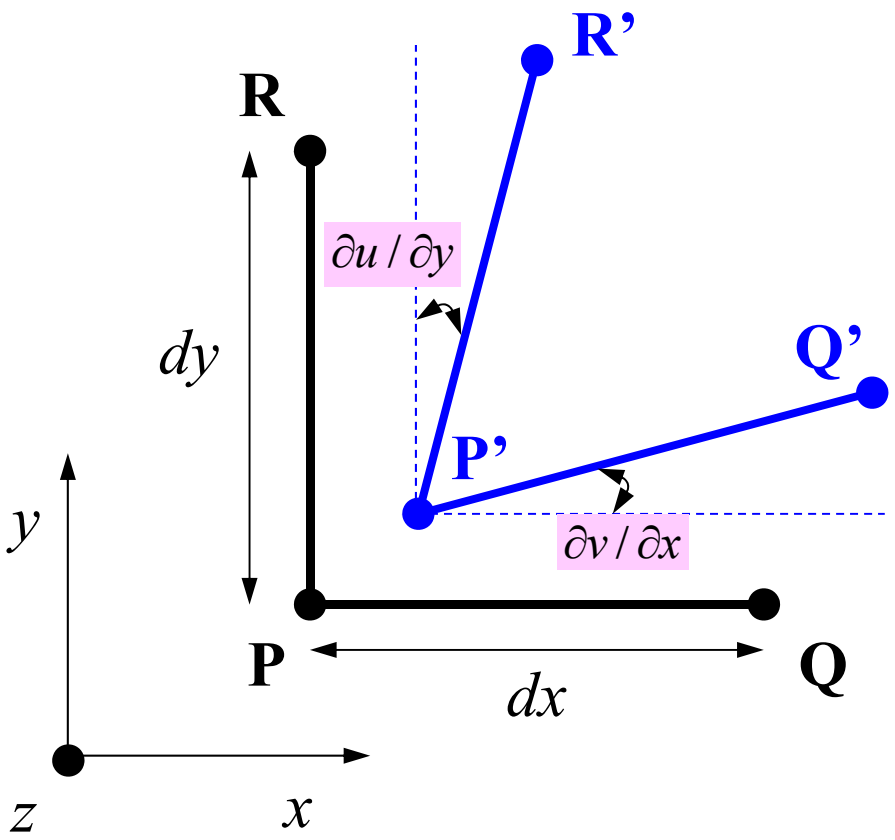


$$\varepsilon_x = \frac{\partial u}{\partial x}$$

$$\varepsilon_y = \frac{\partial v}{\partial y}$$

$$\varepsilon_z = \frac{\partial w}{\partial z}$$

Shear Strain - Displacement



$$\gamma_{xy} = \frac{\partial u}{\partial y} + \frac{\partial v}{\partial x}$$

$$\gamma_{yz} = \frac{\partial v}{\partial z} + \frac{\partial w}{\partial y}$$

$$\gamma_{zx} = \frac{\partial w}{\partial x} + \frac{\partial u}{\partial z}$$

Stress-Strain Relationship

$$\begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix} = \frac{1}{E} \begin{bmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{bmatrix} \begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix}$$

Strain-Stress Relationship

$$\begin{Bmatrix} \sigma_x \\ \sigma_y \\ \sigma_z \\ \tau_{xy} \\ \tau_{yz} \\ \tau_{zx} \end{Bmatrix} = \underbrace{\boxed{\frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & \nu & \nu & 0 & 0 & 0 \\ \nu & 1-\nu & \nu & 0 & 0 & 0 \\ \nu & \nu & 1-\nu & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{2}(1-2\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2}(1-2\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & \frac{1}{2}(1-2\nu) \end{bmatrix}}}_{[D]} \begin{Bmatrix} \varepsilon_x \\ \varepsilon_y \\ \varepsilon_z \\ \gamma_{xy} \\ \gamma_{yz} \\ \gamma_{zx} \end{Bmatrix}$$

$$\{\sigma\} = [D]\{\varepsilon\}$$

- Incompressible Material (ν~0.50)（非圧縮性材料）: Special Treatment Needed

Equilibrium Equations in X-Direction

$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + X = 0$$

$$\sigma_{x,x} + \tau_{xy,y} + \tau_{zx,z} + X = 0$$

Galerkin Method

$$\int [N]^T \left\{ \sigma_{x,x} + \tau_{xy,y} + \tau_{zx,z} + X \right\} dV = 0$$

$$\underline{\int [N]^T \left\{ \sigma_{x,x} \right\} dV} + \int [N]^T \left\{ \tau_{xy,y} \right\} dV + \int [N]^T \left\{ \tau_{zx,z} \right\} dV + \int [N]^T \left\{ X \right\} dV = 0$$

Equilibrium Eqn's in X-dir. (1/3)

$$\int_V [N]^T \{\sigma_{x,x}\} dV = -\int_V [N_{,x}]^T \{\sigma_x\} dV + \int_S [N]^T \{\sigma_x\} n_x dS$$

$$\because \int_V \frac{\partial}{\partial x}\left[[N]^T \{\sigma_x\}\right] dV = \int_S [N]^T \{\sigma_x\} n_x dS$$

1D Gauss

$$\int_V \frac{\partial}{\partial x}\left[[N]^T \{\sigma_x\}\right] dV = \int_V [N]^T \{\sigma_{x,x}\} dV + \int_V [N_{,x}]^T \{\sigma_x\} dV$$

Integration by Parts

Following "weak form" is obtained
(surface integration terms are not considered):

$$\int_V [N]^T \{\sigma_{x,x}\} dV + \int_V [N]^T \{\tau_{xy,y}\} dV + \int_V [N]^T \{\tau_{zx,z}\} dV + \int_V [N]^T \{X\} dV =$$

$$-\int_V [N_{,x}]^T \{\sigma_x\} dV - \int_V [N_{,y}]^T \{\tau_{xy}\} dV - \int_V [N_{,z}]^T \{\tau_{zx}\} dV + \int_V [N]^T \{X\} dV = 0$$

Equilibrium Eqn's in X-dir. (2/3)

$$-\int_V [N_{,x}]^T \{\sigma_x\} dV - \int_V [N_{,y}]^T \{\tau_{xy}\} dV - \int_V [N_{,z}]^T \{\tau_{zx}\} dV + \int_V [N]^T \{X\} dV = 0 \quad (*)$$

$$\sigma_x = \frac{E}{(1+\nu)(1-2\nu)}\left[(1-\nu)\varepsilon_x + \nu\varepsilon_y + \nu\varepsilon_z\right]$$

$$= \frac{E}{(1+\nu)(1-2\nu)}\left[(1-\nu)u_{,x} + \nu v_{,y} + \nu w_{,z}\right]$$

$$\tau_{xy} = \frac{E}{2(1+\nu)}\gamma_{xy} = \frac{E}{2(1+\nu)}\left[u_{,y} + v_{,x}\right]$$

$$\tau_{zx} = \frac{E}{2(1+\nu)}\gamma_{zx} = \frac{E}{2(1+\nu)}\left[u_{,z} + w_{,x}\right]$$

Distribution of Displacement in each Element, Shape Functions

$$u = [N]\{U\}, \quad v = [N]\{V\}, \quad w = [N]\{W\}$$

$$u_{,x} = [N_{,x}]\{U\}, \quad u_{,y} = [N_{,y}]\{U\}, \quad u_{,z} = [N_{,z}]\{U\}$$
$$v_{,x} = [N_{,x}]\{V\}, \quad v_{,y} = [N_{,y}]\{V\}$$
$$w_{,x} = [N_{,x}]\{W\}, \quad w_{,z} = [N_{,z}]\{W\}$$

$$\sigma_x = \frac{E}{(1+\nu)(1-2\nu)}\left((1-\nu)[N_{,x}]\{U\} + \nu[N_{,y}]\{V\} + \nu[N_{,z}]\{W\}\right)$$

$$\tau_{xy} = \frac{E}{2(1+\nu)}\gamma_{xy} = \frac{E}{2(1+\nu)}\left([N_{,y}]\{U\} + [N_{,x}]\{V\}\right)$$

$$\tau_{zx} = \frac{E}{2(1+\nu)}\gamma_{zx} = \frac{E}{2(1+\nu)}\left([N_{,z}]\{U\} + [N_{,x}]\{W\}\right)$$

Equilibrium Eqn's in X-dir. (3/3)

Define: $D = \dfrac{(1-\nu)E}{(1+\nu)(1-2\nu)}, \quad a = \dfrac{\nu E}{(1+\nu)(1-2\nu)}, \quad b = \dfrac{E}{2(1+\nu)}$

$$\sigma_x = D[N_{,x}]\{U\} + a[N_{,y}]\{V\} + a[N_{,z}]\{W\}$$
$$\tau_{xy} = b[N_{,y}]\{U\} + b[N_{,x}]\{V\}$$
$$\tau_{zx} = b[N_{,z}]\{U\} + b[N_{,x}]\{W\}$$

$$(*) = -\int_V [N_{,x}]^T \{\sigma_x\} dV - \int_V [N_{,y}]^T \{\tau_{xy}\} dV - \int_V [N_{,z}]^T \{\tau_{zx}\} dV + \int_V [N]^T \{X\} dV =$$
$$-\int_V \left\{ D[N_{,x}]^T [N_{,x}] + b\left([N_{,y}]^T [N_{,y}] + [N_{,z}]^T [N_{,z}]\right)\right\} dV \{U\}$$
$$-\int_V \left\{ a[N_{,x}]^T [N_{,y}] + b\left([N_{,y}]^T [N_{,x}]\right)\right\} dV \{V\} - \int_V \left\{ a[N_{,x}]^T [N_{,z}] + b[N_{,z}]^T [N_{,x}]\right\} dV \{W\}$$
$$+\int_V [N]^T \{X\} dV = 0$$

Equilibrium Eqn's in Y-dir.

$$-\int_V \left\{D\left[N_{,y}\right]^T\left[N_{,y}\right]+b\left(\left[N_{,z}\right]^T\left[N_{,z}\right]+\left[N_{,x}\right]^T\left[N_{,x}\right]\right)\right\}dV\left\{V\right\}$$

$$-\int_V \left\{a\left[N_{,y}\right]^T\left[N_{,x}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,y}\right]\right)\right\}dV\left\{U\right\}-\int_V \left\{a\left[N_{,y}\right]^T\left[N_{,z}\right]+b\left[N_{,z}\right]^T\left[N_{,y}\right]\right\}dV\left\{W\right\}$$

$$+\int_V \left[N\right]^T\left\{Y\right\}dV=0$$

Equilibrium Eqn's in Z-dir.

$$-\int_V \left\{D\left[N_{,z}\right]^T\left[N_{,z}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,x}\right]+\left[N_{,y}\right]^T\left[N_{,y}\right]\right)\right\}dV\left\{W\right\}$$

$$-\int_V \left\{a\left[N_{,z}\right]^T\left[N_{,x}\right]+b\left(\left[N_{,x}\right]^T\left[N_{,z}\right]\right)\right\}dV\left\{U\right\}-\int_V \left\{a\left[N_{,z}\right]^T\left[N_{,y}\right]+b\left[N_{,y}\right]^T\left[N_{,z}\right]\right\}dV\left\{V\right\}$$

$$+\int_V \left[N\right]^T\left\{Z\right\}dV=0$$

Three Equilibrium Eqn's

- 3 equations for 3 unknowns (disp. components)
- 3 equations are "coupled" (not independent)
- Vector of Shape Function: $[N]$: 8×1 matrix
  - $[N]^T[N]$, $[N_{,x}]^T[N_{,y}]$ etc.: 8×8 matrix

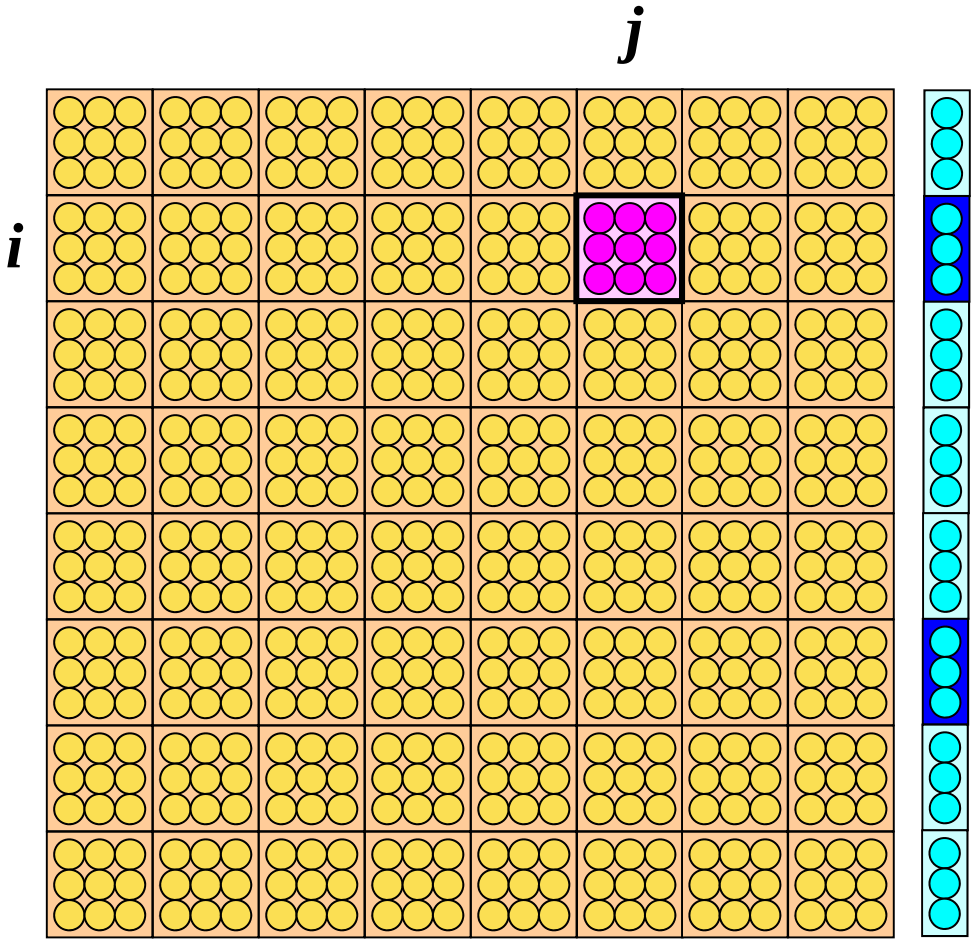
$$\frac{\partial \sigma_x}{\partial x} + \frac{\partial \tau_{xy}}{\partial y} + \frac{\partial \tau_{zx}}{\partial z} + X = 0$$

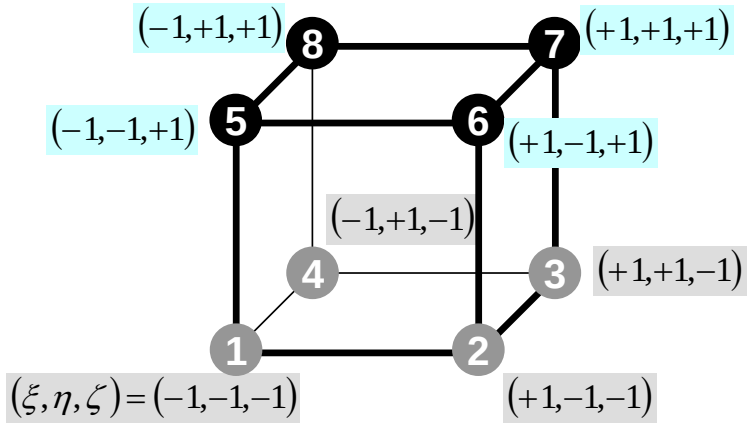
$$\frac{\partial \tau_{xy}}{\partial x} + \frac{\partial \sigma_y}{\partial y} + \frac{\partial \tau_{yz}}{\partial z} + Y = 0$$

$$\frac{\partial \tau_{zx}}{\partial x} + \frac{\partial \tau_{yz}}{\partial y} + \frac{\partial \sigma_z}{\partial z} + Z = 0$$

Element Matrix: 24×24

(u,v,w) components on each node are physically strongly-coupled: these three components are treated in block-wise manner: 8×8 matrix

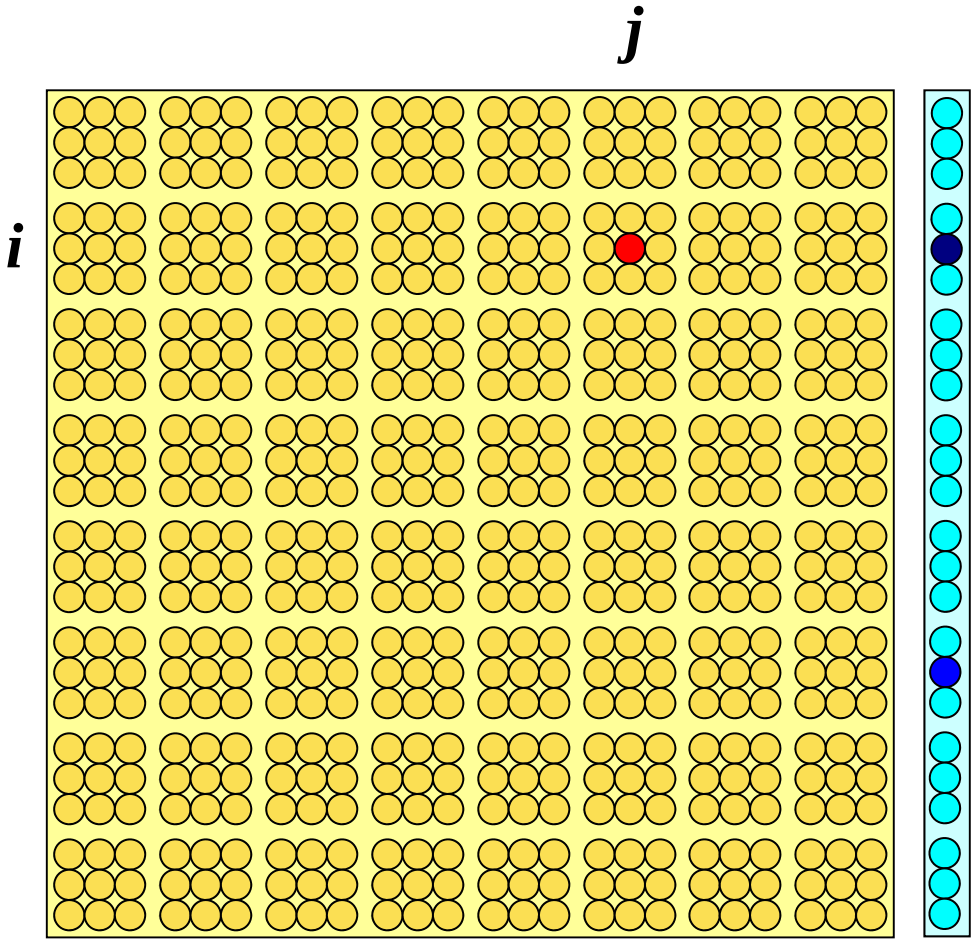


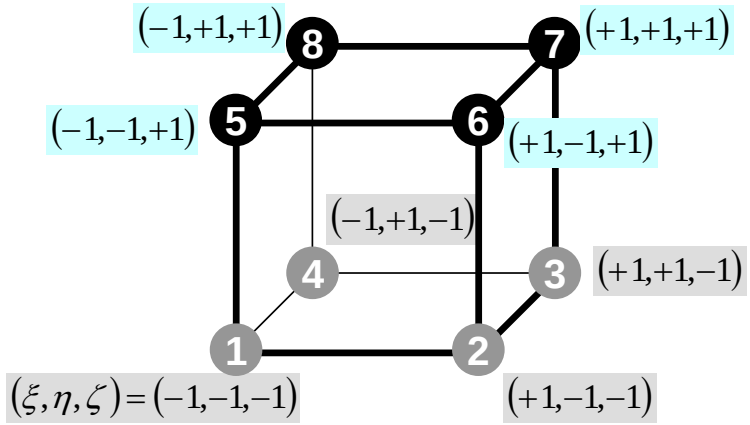


$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

Element Matrix: 24×24

(u,v,w) components on each node can be treated independently, but 8×8 matrix is more robust (next week)

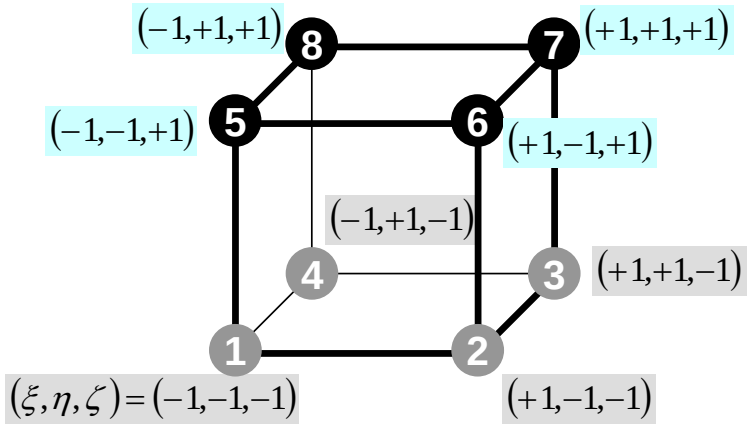




$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

Element Matrix: *i-j*, *X*-dir. (1/3)

$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$



$$-\int_V \left\{ D\left[N_{,x}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,y}\right]^T\left[N_{,y}\right] + \left[N_{,z}\right]^T\left[N_{,z}\right]\right)\right\} dV \{U\}$$

$$-\int_V \left\{ a\left[N_{,x}\right]^T\left[N_{,y}\right] + b\left(\left[N_{,y}\right]^T\left[N_{,x}\right]\right)\right\} dV \{V\}$$

$$-\int_V \left\{ a\left[N_{,x}\right]^T\left[N_{,z}\right] + b\left[N_{,z}\right]^T\left[N_{,x}\right]\right\} dV \{W\}$$

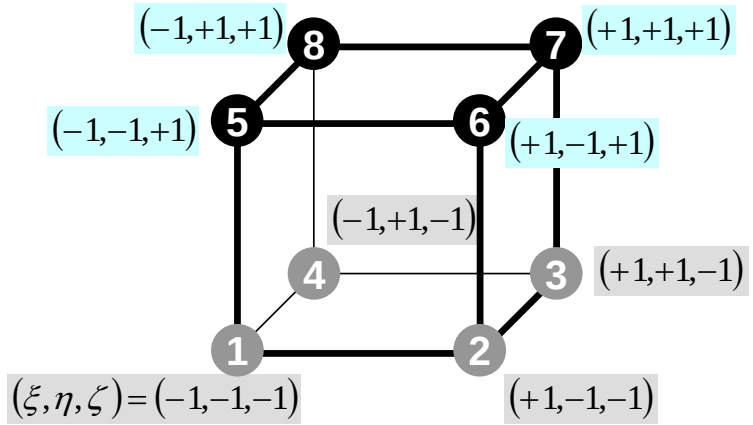
$$k_{ij_{11}} = -\int_V \left\{ D \cdot N_{i,x} \cdot N_{j,x} + b \cdot \left(N_{i,y} \cdot N_{j,y} + N_{i,z} \cdot N_{j,z}\right)\right\} dV$$

$$k_{ij_{12}} = -\int_V \left\{ a \cdot N_{i,x} \cdot N_{j,y} + b \cdot N_{i,y} \cdot N_{j,x}\right\} dV$$

$$k_{ij_{13}} = -\int_V \left\{ a \cdot N_{i,x} \cdot N_{j,z} + b \cdot N_{i,z} \cdot N_{j,x}\right\} dV$$

Element Matrix: *i-j*, *Y*-dir. (2/3)

$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$



$$-\int_V \left\{ D\left[N_{,y}\right]^T\left[N_{,y}\right] + b\left(\left[N_{,z}\right]^T\left[N_{,z}\right] + \left[N_{,x}\right]^T\left[N_{,x}\right]\right)\right\} dV \{V\}$$

$$-\int_V \left\{ a\left[N_{,y}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,y}\right]\right)\right\} dV \{U\}$$

$$-\int_V \left\{ a\left[N_{,y}\right]^T\left[N_{,z}\right] + b\left[N_{,z}\right]^T\left[N_{,y}\right]\right\} dV \{W\}$$

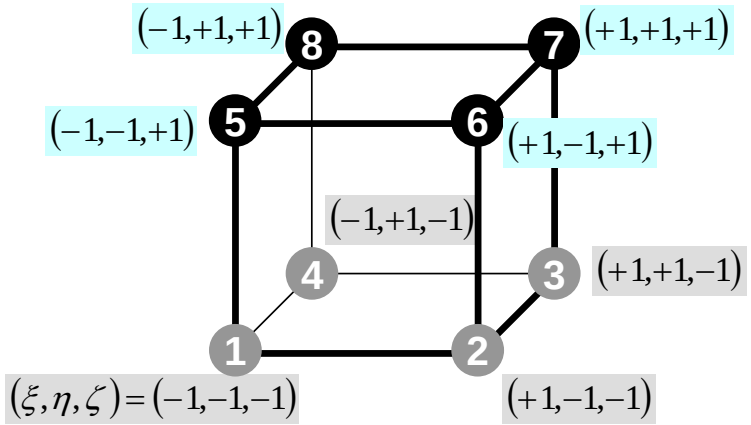
$$k_{ij_{21}} = -\int_V \left\{ a \cdot N_{i,y} \cdot N_{j,x} + b \cdot N_{i,x} \cdot N_{j,y} \right\} dV$$

$$k_{ij_{22}} = -\int_V \left\{ D \cdot N_{i,y} \cdot N_{j,y} + b \cdot \left(N_{i,z} \cdot N_{j,z} + N_{i,x} \cdot N_{j,x} \right) \right\} dV$$

$$k_{ij_{23}} = -\int_V \left\{ a \cdot N_{i,y} \cdot N_{j,z} + b \cdot N_{i,z} \cdot N_{j,y} \right\} dV$$

Element Matrix: *i-j*, *Z*-dir. (3/3)

$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$



$$-\int_V \left\{ D\left[N_{,z}\right]^T\left[N_{,z}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,x}\right] + \left[N_{,y}\right]^T\left[N_{,y}\right]\right)\right\} dV \{W\}$$

$$-\int_V \left\{ a\left[N_{,z}\right]^T\left[N_{,x}\right] + b\left(\left[N_{,x}\right]^T\left[N_{,z}\right]\right)\right\} dV \{U\}$$

$$-\int_V \left\{ a\left[N_{,z}\right]^T\left[N_{,y}\right] + b\left[N_{,y}\right]^T\left[N_{,z}\right]\right\} dV \{V\}$$

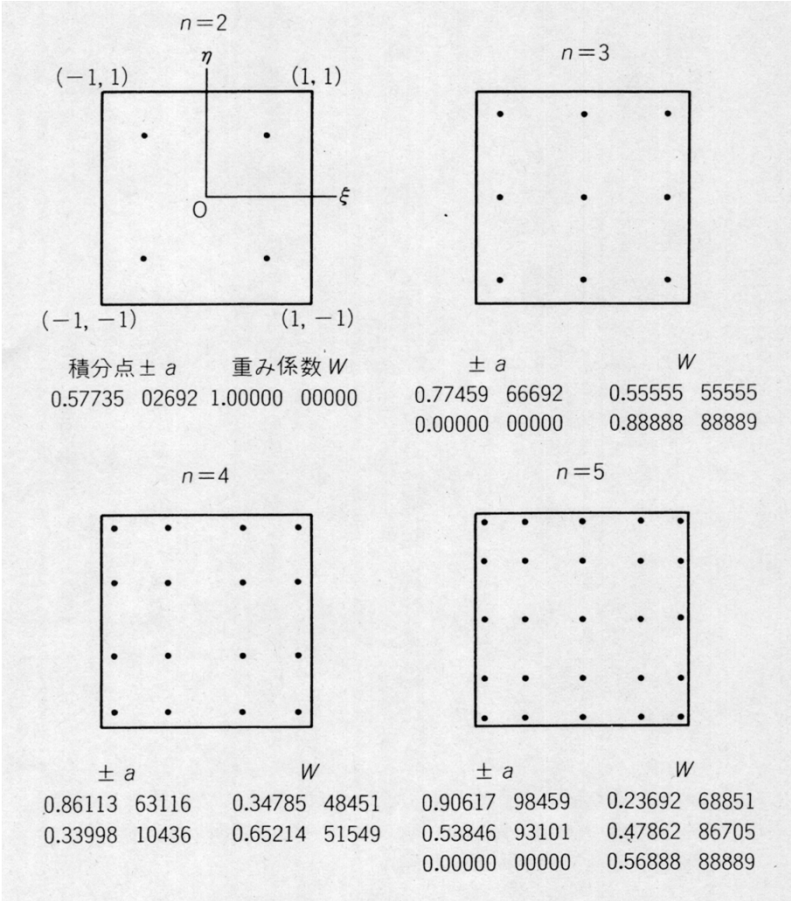
$$k_{ij_{31}} = -\int_V \left\{ a \cdot N_{i,z} \cdot N_{j,x} + b \cdot N_{i,x} \cdot N_{j,z} \right\} dV$$

$$k_{ij_{32}} = -\int_V \left\{ a \cdot N_{i,z} \cdot N_{j,y} + b \cdot N_{i,y} \cdot N_{j,z} \right\} dV$$

$$k_{ij_{33}} = -\int_V \left\{ D \cdot N_{i,z} \cdot N_{j,z} + b \cdot \left(N_{i,x} \cdot N_{j,x} + N_{i,y} \cdot N_{j,y} \right) \right\} dV$$

Next Stage: Integration

- 3D Natural/Local Coordinate (ξ,η,ζ):
  - Gaussian Quadrature



$$I = \int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta,\zeta)\, d\xi d\eta d\zeta$$

$$= \sum_{i=1}^{L}\sum_{j=1}^{M}\sum_{k=1}^{N}\left[W_i \cdot W_j \cdot W_k \cdot f(\xi_i,\eta_j,\zeta_k)\right]$$

L,M,N: number of quadrature points in ξ,η,ζ-direction

(ξ_i,η_j,ζ_k) : Coordinates of Quad's

W_i,W_j,W_k : Weighting Factor

Partial Diff. on Natural Coord. (1/4)

- According to formulae:

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \xi} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \xi} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \xi} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \xi}$$

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \eta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \eta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \eta} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \eta}$$

$$\frac{\partial N_i(\xi,\eta,\zeta)}{\partial \zeta} = \frac{\partial N_i}{\partial x}\frac{\partial x}{\partial \zeta} + \frac{\partial N_i}{\partial y}\frac{\partial y}{\partial \zeta} + \frac{\partial N_i}{\partial z}\frac{\partial z}{\partial \zeta}$$

$\left[\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta}\right]$ can be easily derived according to definitions.

$\left[\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z}\right]$ are required for computations.

Partial Diff. on Natural Coord. (2/4)

- In matrix form:

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = [J] \begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix}$$

$[J]$: Jacobi matrix, Jacobian

Partial Diff. on Natural Coord. (3/4)

- Components of Jacobian:

$$J_{11} = \frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} x_i, \quad J_{12} = \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} y_i,$$

$$J_{13} = \frac{\partial z}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \xi} z_i$$

$$J_{21} = \frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} x_i, \quad J_{22} = \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} y_i,$$

$$J_{23} = \frac{\partial z}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \eta} z_i$$

$$J_{31} = \frac{\partial x}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i x_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} x_i, \quad J_{32} = \frac{\partial y}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i y_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} y_i,$$

$$J_{33} = \frac{\partial z}{\partial \zeta} = \frac{\partial}{\partial \zeta}\left(\sum_{i=1}^{8} N_i z_i\right) = \sum_{i=1}^{8} \frac{\partial N_i}{\partial \zeta} z_i$$

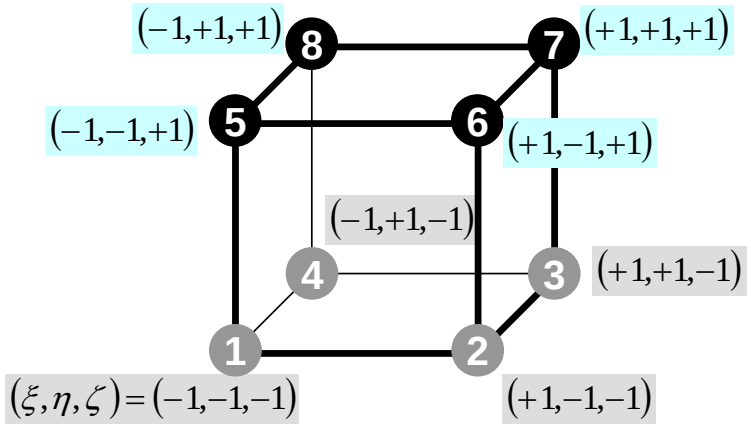
Partial Diff. on Natural Coord. (4/4)

- Partial differentiation on global coordinate system is introduced as follows (with inverse of Jacobian matrix (3 × 3))

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \\ \dfrac{\partial N_i}{\partial z} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} & \dfrac{\partial z}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} & \dfrac{\partial z}{\partial \eta} \\ \dfrac{\partial x}{\partial \zeta} & \dfrac{\partial y}{\partial \zeta} & \dfrac{\partial z}{\partial \zeta} \end{bmatrix}^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \\ \dfrac{\partial N_i}{\partial \zeta} \end{Bmatrix}$$

Integration on Each Element

$$\begin{bmatrix} k_{ij_{11}} & k_{ij_{12}} & k_{ij_{13}} \\ k_{ij_{21}} & k_{ij_{22}} & k_{ij_{23}} \\ k_{ij_{31}} & k_{ij_{32}} & k_{ij_{33}} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$

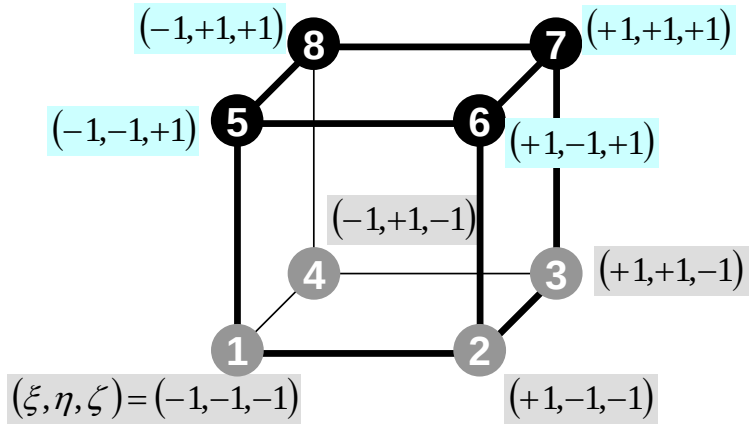


$$k_{ij_{11}} = -\int_V \left\{ D \cdot N_{i,x} \cdot N_{j,x} + b\left(N_{i,y} \cdot N_{j,y} + N_{i,z} \cdot N_{j,z} \right) \right\} dV$$

$$= -\int_V \left\{ D \frac{\partial N_i}{\partial x} \frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y} \frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z} \frac{\partial N_j}{\partial z} \right) \right\} dV$$

Integration on Natural Coord. System

$$\begin{bmatrix} k_{ij11} & k_{ij12} & k_{ij13} \\ k_{ij21} & k_{ij22} & k_{ij23} \\ k_{ij31} & k_{ij32} & k_{ij33} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$



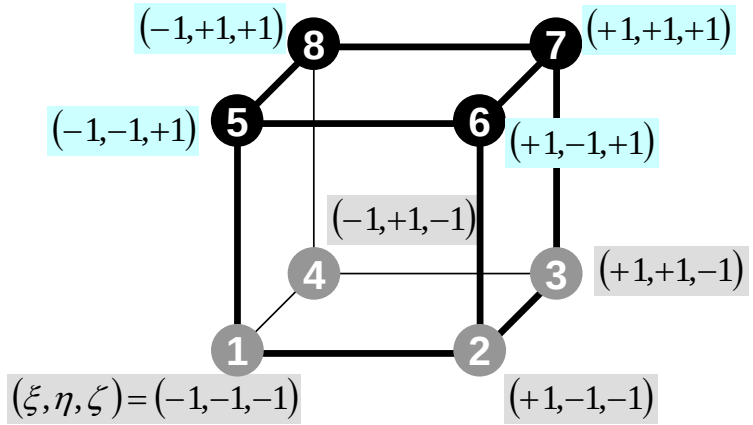
$$-\int_V \left\{ D \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z} \right) \right\} dV =$$

$$-\iiint \left\{ D \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z} \right) \right\} dxdydz =$$

$$-\int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} \left\{ D \frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x} + b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y} + \frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z} \right) \right\} \det|J| \, d\xi d\eta d\zeta$$

Gaussian Quadrature

$$\begin{bmatrix} k_{ij11} & k_{ij12} & k_{ij13} \\ k_{ij21} & k_{ij22} & k_{ij23} \\ k_{ij31} & k_{ij32} & k_{ij33} \end{bmatrix} \quad (i, j = 1 \ldots 8)$$



$$-\int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1}\left\{D\frac{\partial N_i}{\partial x}\frac{\partial N_j}{\partial x}+b\left(\frac{\partial N_i}{\partial y}\frac{\partial N_j}{\partial y}+\frac{\partial N_i}{\partial z}\frac{\partial N_j}{\partial z}\right)\right\}\det|J|\,d\xi d\eta d\zeta$$

$$I=\int_{-1}^{+1}\int_{-1}^{+1}\int_{-1}^{+1} f(\xi,\eta,\zeta)\,d\xi d\eta d\zeta$$

$$=\sum_{i=1}^{L}\sum_{j=1}^{M}\sum_{k=1}^{N}\left[W_i\cdot W_j\cdot W_k\cdot f(\xi_i,\eta_j,\zeta_k)\right]$$

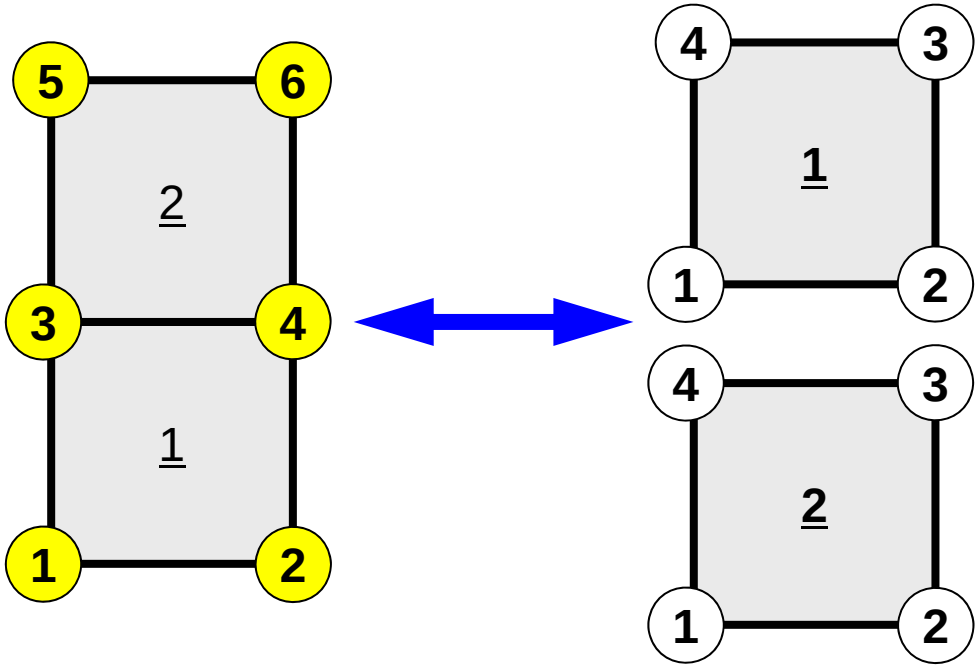
Remaining Procedures

- Element matrices have been formed.

- Accumulation to Global Matrix
- Implementation of Boundary Conditions
- Solving Linear Equations

- Details of implementation will be discussed in classes later than next week through explanation of programs

Accumulation: Local -> Global Matrices



$$[k^{(2)}]\{\phi^{(2)}\} = \{f^{(2)}\}$$

$$\begin{bmatrix} k_{11}^{(2)} & k_{12}^{(2)} & k_{13}^{(2)} & k_{14}^{(2)} \\ k_{21}^{(2)} & k_{22}^{(2)} & k_{23}^{(2)} & k_{24}^{(2)} \\ k_{31}^{(2)} & k_{32}^{(2)} & k_{33}^{(2)} & k_{34}^{(2)} \\ k_{41}^{(2)} & k_{42}^{(2)} & k_{43}^{(2)} & k_{44}^{(2)} \end{bmatrix} \begin{Bmatrix} \phi_1^{(2)} \\ \phi_2^{(2)} \\ \phi_3^{(2)} \\ \phi_4^{(2)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(2)} \\ f_2^{(2)} \\ f_3^{(2)} \\ f_4^{(2)} \end{Bmatrix}$$

$$[k^{(1)}]\{\phi^{(1)}\} = \{f^{(1)}\}$$

$$\begin{bmatrix} k_{11}^{(1)} & k_{12}^{(1)} & k_{13}^{(1)} & k_{14}^{(1)} \\ k_{21}^{(1)} & k_{22}^{(1)} & k_{23}^{(1)} & k_{24}^{(1)} \\ k_{31}^{(1)} & k_{32}^{(1)} & k_{33}^{(1)} & k_{34}^{(1)} \\ k_{41}^{(1)} & k_{42}^{(1)} & k_{43}^{(1)} & k_{44}^{(1)} \end{bmatrix} \begin{Bmatrix} \phi_1^{(1)} \\ \phi_2^{(1)} \\ \phi_3^{(1)} \\ \phi_4^{(1)} \end{Bmatrix} = \begin{Bmatrix} f_1^{(1)} \\ f_2^{(1)} \\ f_3^{(1)} \\ f_4^{(1)} \end{Bmatrix}$$

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

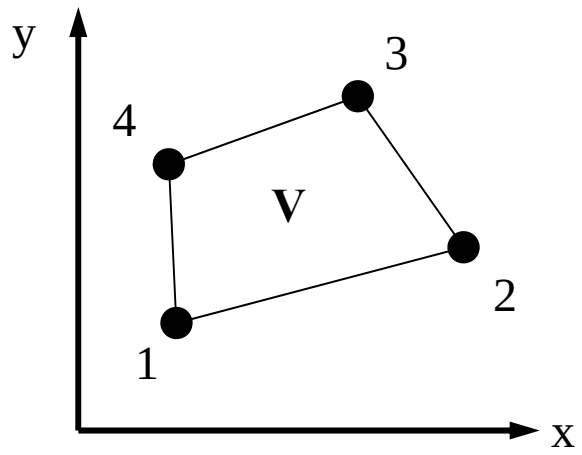
$$\begin{bmatrix} D_1 & X & X & X & & \\ X & D_2 & X & X & & \\ X & X & D_3 & X & X & X \\ X & X & X & D_4 & X & X \\ & & X & X & D_5 & X \\ & & X & X & X & D_6 \end{bmatrix} \begin{Bmatrix} \Phi_1 \\ \Phi_2 \\ \Phi_3 \\ \Phi_4 \\ \Phi_5 \\ \Phi_6 \end{Bmatrix} = \begin{Bmatrix} B_1 \\ B_2 \\ B_3 \\ B_4 \\ B_5 \\ B_6 \end{Bmatrix}$$

- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- **HW**

- Running the Code
- Data Structure
- Overview of the Program
- Computational Issue
- Visualization by ParaView

Homework

- Develop a program and calculate area of the following quadrilateral using Gaussian Quadrature.



1:(1.0, 1.0)
2:(4.0, 2.0)
3:(3.0, 5.0)
4:(2.0, 4.0)

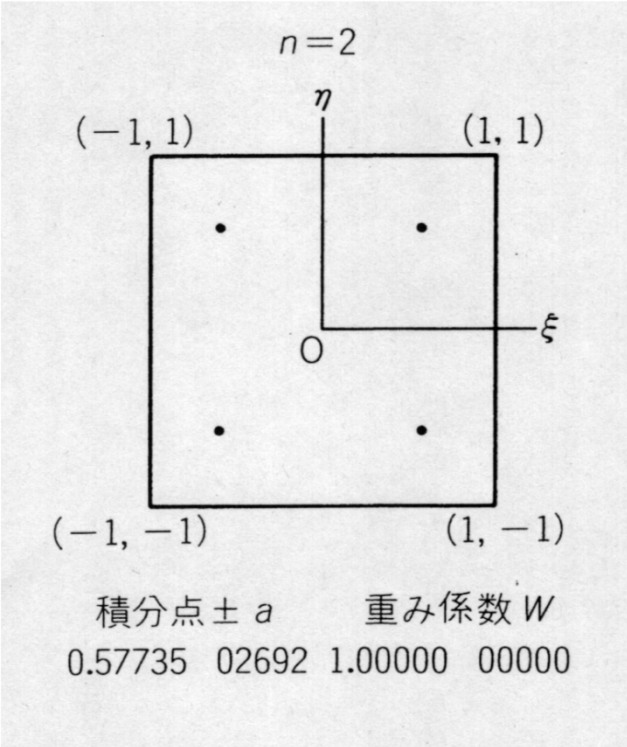
$$I = \int_V dV = \int_{-1}^{+1} \int_{-1}^{+1} \det|J| \, d\xi d\zeta$$

Tips (1/2)

- Calculate Jacobian
- Apply Gaussian Quadrature (n=2)

$$I = \int_{-1}^{+1} \int_{-1}^{+1} f(\xi,\eta)\, d\xi d\eta = \sum_{i=1}^{m} \sum_{j=1}^{n} \left[W_i \cdot W_j \cdot f(\xi_i, \eta_j) \right]$$

```
implicit REAL*8 (A-H,O-Z)
real*8 W(2)
real*8 POI(2)

W(1)= 1.0d0
W(2)= 1.0d0
POI(1)= -0.5773502692d0
POI(2)= +0.5773502692d0

SUM= 0.d0
do jp= 1, 2
do ip= 1, 2
  FC = F(POI(ip),POI(jp))
  SUM= SUM + W (ip)*W (jp)*FC
enddo
enddo
```



Tips (2/2)

$$[J]=\begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix}, \quad \det|J| = \frac{\partial x}{\partial \xi}\cdot\frac{\partial y}{\partial \eta} - \frac{\partial y}{\partial \xi}\cdot\frac{\partial x}{\partial \eta}$$

$$\frac{\partial x}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{4} N_i x_i\right) = \sum_{i=1}^{4}\frac{\partial N_i}{\partial \xi}x_i, \quad \frac{\partial y}{\partial \xi} = \frac{\partial}{\partial \xi}\left(\sum_{i=1}^{4} N_i y_i\right) = \sum_{i=1}^{4}\frac{\partial N_i}{\partial \xi}y_i,$$

$$\frac{\partial x}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{4} N_i x_i\right) = \sum_{i=1}^{4}\frac{\partial N_i}{\partial \eta}x_i, \quad \frac{\partial y}{\partial \eta} = \frac{\partial}{\partial \eta}\left(\sum_{i=1}^{4} N_i y_i\right) = \sum_{i=1}^{4}\frac{\partial N_i}{\partial \eta}y_i$$

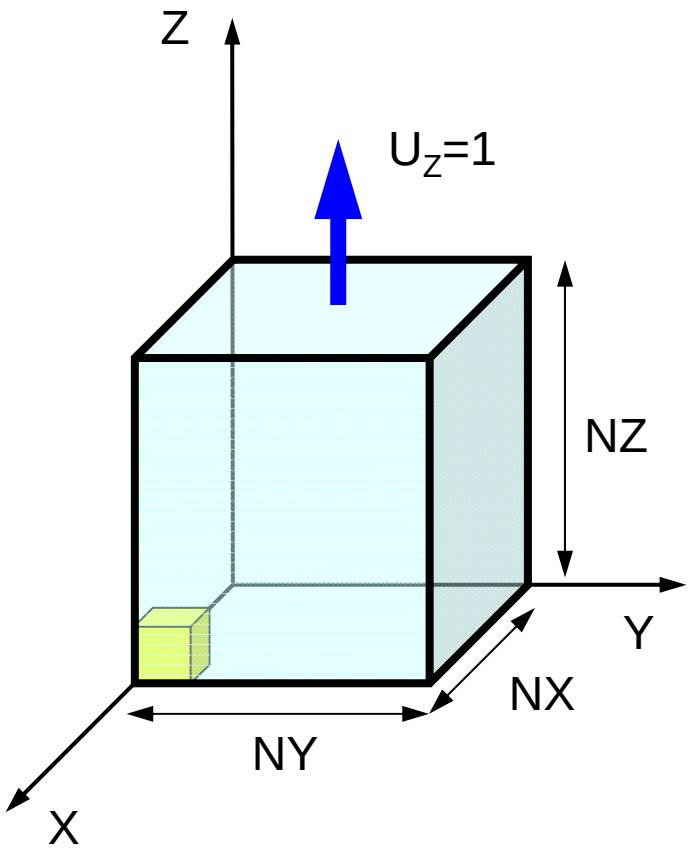
$$N_1(\xi,\eta) = \frac{1}{4}(1-\xi)(1-\eta), \quad N_2(\xi,\eta) = \frac{1}{4}(1+\xi)(1-\eta),$$

$$N_3(\xi,\eta) = \frac{1}{4}(1+\xi)(1+\eta), \quad N_4(\xi,\eta) = \frac{1}{4}(1-\xi)(1+\eta)$$

- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- **Running the Code**
- Data Structure
- Overview of the Program
- Computational Issue
- Visualization by ParaView

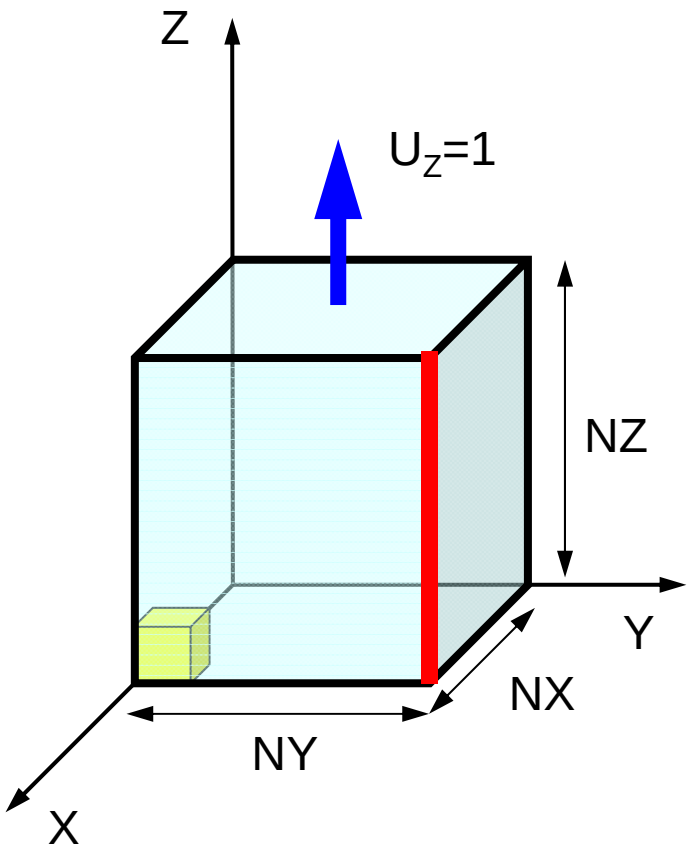
Target Application



- Elastic Material
  - Young's Modulus E
  - Poisson's Ratio ν
- Rectangular Prism
  - 1x1x1 cubes (hexahedra)
  - NX, NY, NZ cubes in each direction
- Boundary Conditions
  - Symmetric B.C.
    - U_X=0@X=0
    - U_Y=0@Y=0
    - U_Z=0@Z=0
  - Dirichlet B.C.
    - U_Z=1@Z=Z_{max}

movie

NX=NY=NZ=10, ν=0.30



$$\varepsilon_z = \frac{1}{10} = 0.10$$

$$\varepsilon_x = \varepsilon_y = -\nu\varepsilon_z = -0.03$$

$$\therefore u_x\Big|_{X=10,Y=10} = u_y\Big|_{X=10,Y=10} = 10 \times \varepsilon_x = -0.30$$

Copy/Installation (1/2)

```
3D Code

  >$ cd <$fem1>
  >$ cp /home03/skengon/Documents/class/fem1/fem3d.tar .
  >$ tar xvf fem3d.tar
  >$ cd fem3d
  >$ ls
     c   f  run
```

```
FEM Code in C

  >$ cd <$fem1>/fem3d/c
  >$ make
  >$ ls ../run/sol
     sol
```

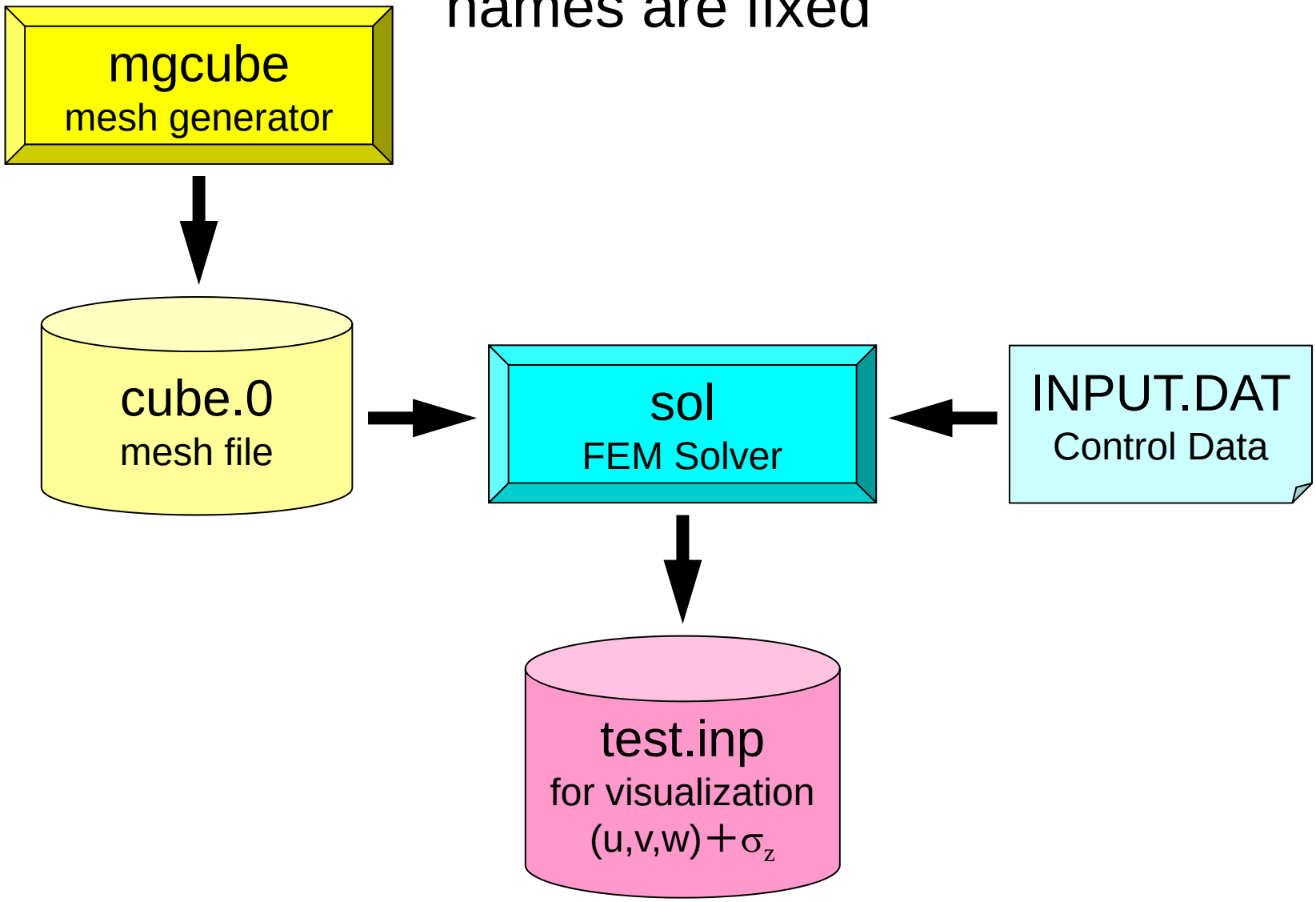
```
FEM Code in FORTRAN

  >$ cd <$fem1>/fem3d/f
  >$ make
  >$ ls ../run/sol
     sol
```

Copy/Installation (2/2)

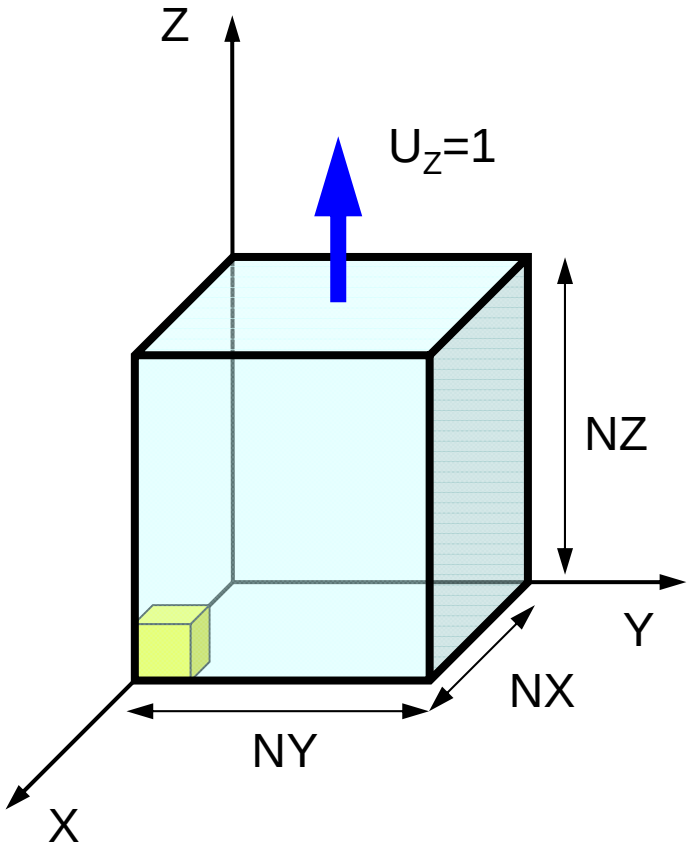
```
Mesh Generator

  >$ cd <$fem1>/fem3d/run
  >$ g95 -O3 mgcube.f -o mgcube
```

Operations

Starting from Grid Generation to Computation, File-names are fixed



Mesh Generation




```
>$ cd <$fem1>/fem3d/run
>$ ./mgcube

  NX, NY, NZ          ←  Number of
                         Elem's in each
                         direction
  10,10,10            ←  example


>$ ls cube.0             confirmation

   cube.0
```

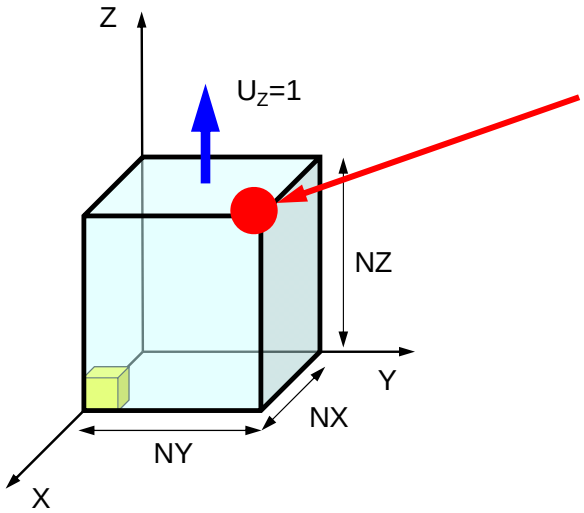
Control File: INPUT.DAT

```
INPUT.DAT

cube.0          fname
1  0            METHOD, PRECOND
1               iterPREmax (not in use)
2000            ITER
1.0 0.3         ELAST, POISSON
```

- **fname** : Name of Mesh File
- **METHOD** : Iterative Method (fixed as 1)
- **PRECOND** : Preconditioning Method
  - 0 : Block-LU-GS, 1 : Block-Diagonal-Scaling
- **iterPREmax:** (not in use)
- **ITER** : Maximum Iterations for CG
- **ELAST** : Young's Modulus
- **POISSON** : Poisson's Ratio
  - Try cases with Poisson's Ratio=0.4999

Running

```
  >$ cd <$fem1>/fem3d/run
  >$ ./sol

 (History of Iterations)
   33    2.218867E-08
   34    1.325902E-08
   35    7.384341E-09

### DISPLACEMENT at (Xmax,Ymax,Zmax))
    1331   -3.000000E-01   -3.000002E-01    1.000000E+00

  >$ ls test.inp        Confirm that file is created
     test.inp
```



Displacement at this point
1331th Node（=11^3）

- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- Running the Code
- **Data Structure**
- Overview of the Program
- Computational Issue
- Visualization by ParaView

Overview of Mesh File: cube.0

numbering starts from “1”

- Nodes
  - Node # (How many nodes ?)
  - Node ID, Coordinates
- Elements
  - Element #
  - Element Type
  - Element ID, Material ID, Connectivity
- Node Groups
  - Group #
  - Node # in each group
  - Group Name
  - Nodes in each group

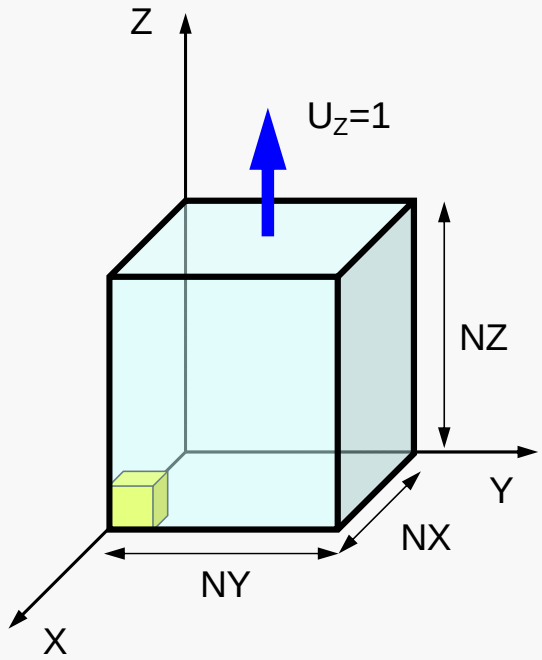
Mesh Generator: mgcube.f (1/5)

```
      implicit REAL*8 (A-H,O-Z)
      real(kind=8), dimension(:,:), allocatable :: X , Y
      real(kind=8), dimension(:,:), allocatable :: X0, Y0
      character(len=80) :: GRIDFILE, HHH
      integer , dimension(:,:), allocatable :: IW
!C
!C +-------+
!C | INIT. |
!C +-------+
!C===
      write (*,*) 'NX,NY,NZ'
      read  (*,*)  NX,NY,NZ

      NXP1= NX + 1                              Node # in X-direction
      NYP1= NY + 1                              Node # in Y-direction
      NZP1= NZ + 1                              Node # in Z-direction

      DX= 1.d0

      INODTOT=  NXP1*NYP1*NZP1                  Node #
      ICELTOT=  NX  *NY  *NZ                    Element #
      IBNODTOT= NXP1*NYP1                       Node # on XY-surface

      allocate (IW(INODTOT,4))
      IW= 0
```

Mesh Generator: mgcube.f (2/5)

```
      icou= 0
      ib  = 1
      do k= 1, NZP1
      do j= 1, NYP1
        i= 1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 2
      do k= 1, NZP1
      j= 1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 3
      k= 1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 4
      k= NZP1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo
!C===
```

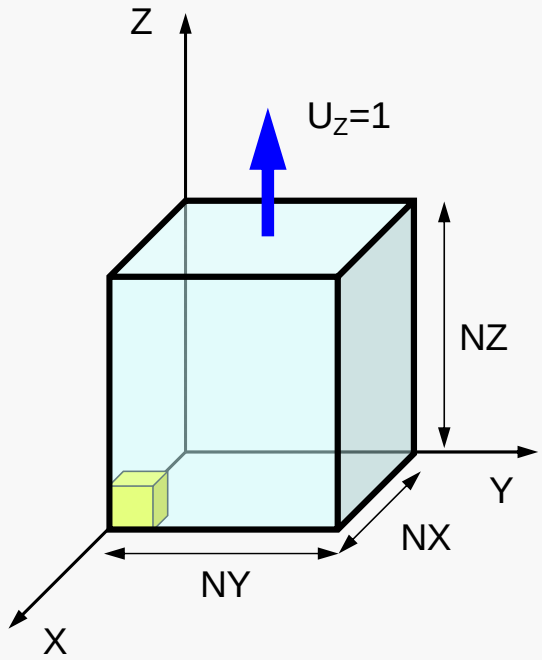
Nodes on X=Xmin surface are stored in IW(ib,1) where ib= 1, NYP1*NZP1



Mesh Generator: mgcube.f (2/5)

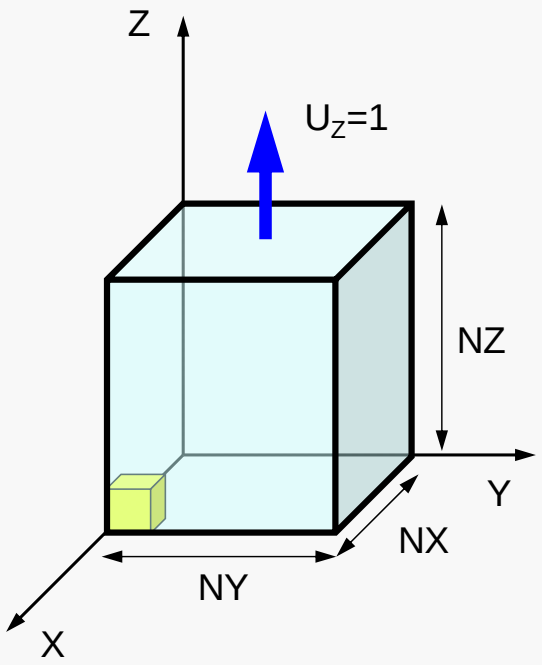
```
      icou= 0
      ib  = 1
      do k= 1, NZP1
      do j= 1, NYP1
        i= 1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 2
      do k= 1, NZP1
      j= 1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 3
      k= 1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 4
      k= NZP1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo
!C===
```

Nodes on Y=Ymin surface are stored in IW(ib,2) where ib= 1, NXP1*NZP1



Mesh Generator: mgcube.f (2/5)

```
      icou= 0
      ib  = 1
      do k= 1, NZP1
      do j= 1, NYP1
        i= 1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 2
      do k= 1, NZP1
      j= 1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 3
      k= 1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo

      icou= 0
      ib  = 4
      k= NZP1
      do j= 1, NYP1
      do i= 1, NXP1
        icou= icou + 1
        ii  = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        IW(icou,ib)= ii
      enddo
      enddo
!C===
```



Nodes on Z=Zmin surface are stored in IW(ib,3) where ib= 1, NXP1*NYP1

Nodes on Z=Zmax surface are stored in IW(ib,4) where ib= 1, NXP1*NYP1

Mesh Generator: mgcube.f (3/5)

```
!C
!C +-------------+
!C | GeoFEM data |
!C +-------------+
!C===
      NN= 0
      write (*,*)       'GeoFEM gridfile name ?'
      GRIDFILE= 'cube.0'

      open  (12, file= GRIDFILE, status='unknown',form='formatted')
          write(12,'(10i10)') INODTOT

          icou= 0
          do k= 1, NZP1
          do j= 1, NYP1
          do i= 1, NXP1
            XX= dfloat(i-1)*DX
            YY= dfloat(j-1)*DX
            ZZ= dfloat(k-1)*DX

            icou= icou + 1
            write (12,'(i10,3(1pe16.6))') icou, XX, YY, ZZ
          enddo
          enddo
          enddo

          write(12,'(i10)') ICELTOT

          IELMTYPL= 361
          write(12,'(10i10)') (IELMTYPL, i=1,ICELTOT)
```

Node #
Node ID, Coordinates

Element #
Element Type (not in use)

Example of “cube.0” (NX=NY=NZ=4) Node, Element-Type

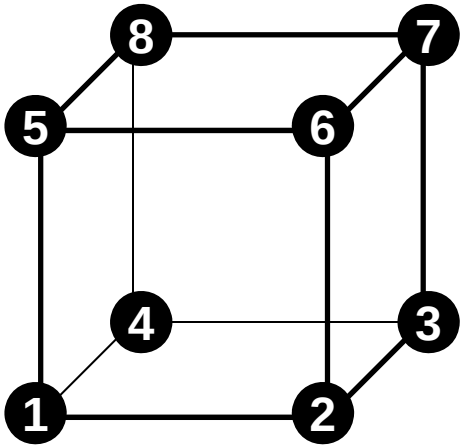
```
       125                                                                  =5*5*5
         1    0.000000E+00    0.000000E+00    0.000000E+00
         2    1.000000E+00    0.000000E+00    0.000000E+00
         3    2.000000E+00    0.000000E+00    0.000000E+00
         4    3.000000E+00    0.000000E+00    0.000000E+00
         5    4.000000E+00    0.000000E+00    0.000000E+00
         6    0.000000E+00    1.000000E+00    0.000000E+00
         7    1.000000E+00    1.000000E+00    0.000000E+00
         8    2.000000E+00    1.000000E+00    0.000000E+00
         9    3.000000E+00    1.000000E+00    0.000000E+00

(…)

       121    0.000000E+00    4.000000E+00    4.000000E+00
       122    1.000000E+00    4.000000E+00    4.000000E+00
       123    2.000000E+00    4.000000E+00    4.000000E+00
       124    3.000000E+00    4.000000E+00    4.000000E+00
       125    4.000000E+00    4.000000E+00    4.000000E+00
        64                                                                  =4*4*4
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361       361       361       361       361       361       361
       361       361       361       361
```

movie

Mesh Generator: mgcube.f (4/5)

```
      icou= 0
      imat= 1
      do k= 1, NZ
      do j= 1, NY
      do i= 1, NX
        icou= icou + 1
        in1 = (k-1)*IBNODTOT + (j-1)*NXP1 + i
        in2 = in1 + 1
        in3 = in2 + NXP1
        in4 = in3 - 1
        in5 = in1 + IBNODTOT
        in6 = in2 + IBNODTOT
        in7 = in3 + IBNODTOT
        in8 = in4 + IBNODTOT
        write (12,'(10i10)') icou, imat, in1, in2, in3, in4,        &
     &                                   in5, in6, in7, in8
      enddo
      enddo
      enddo
```

imat: Material ID (=1)



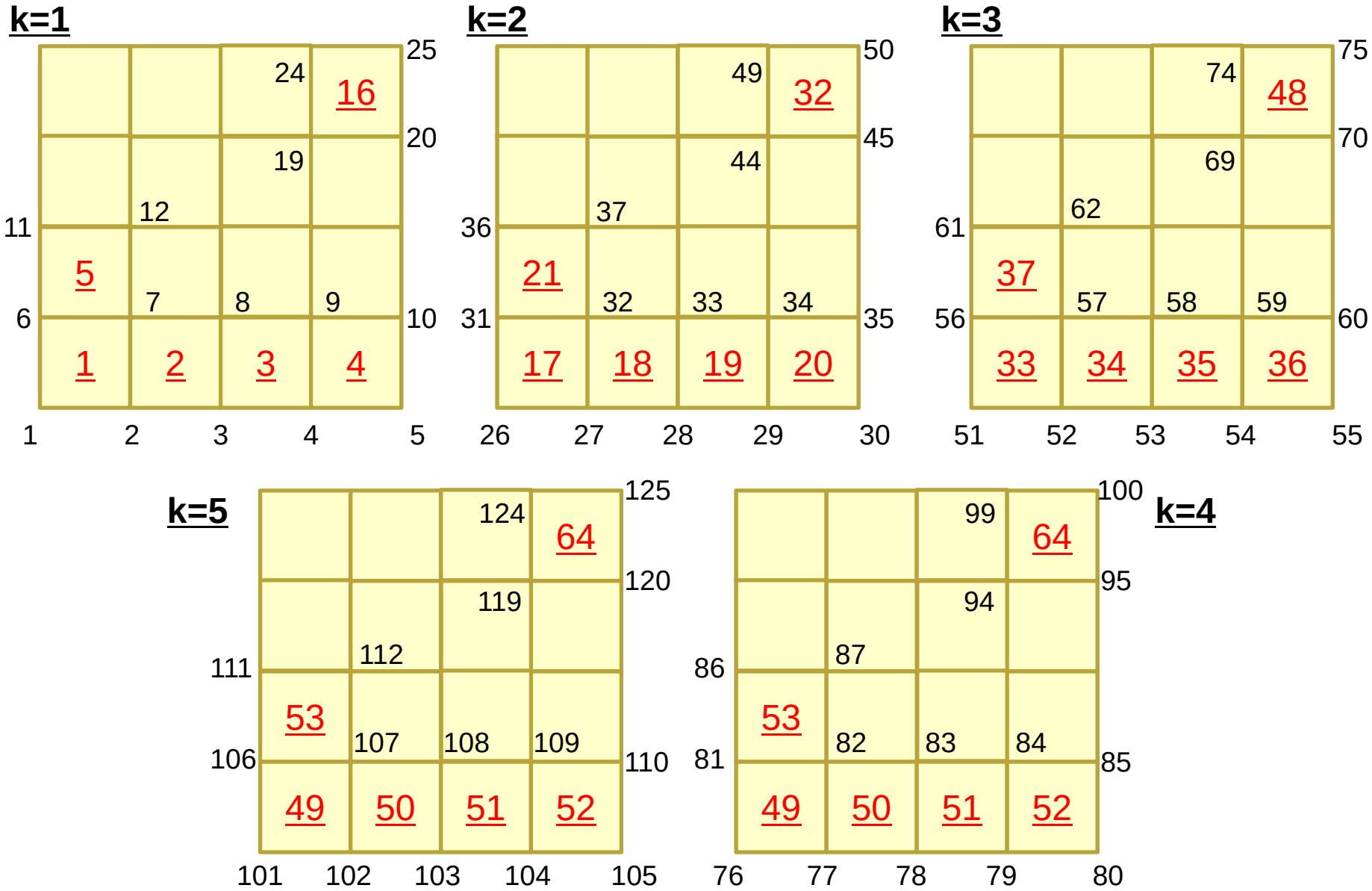
Example of “cube.0” (NX=NY=NZ=4) Element Connectivity

1	1	1	2	7	6	26	27	32	31
2	1	2	3	8	7	27	28	33	32
3	1	3	4	9	8	28	29	34	33
4	1	4	5	10	9	29	30	35	34
5	1	6	7	12	11	31	32	37	36
6	1	7	8	13	12	32	33	38	37
7	1	8	9	14	13	33	34	39	38
8	1	9	10	15	14	34	35	40	39
9	1	11	12	17	16	36	37	42	41
10	1	12	13	18	17	37	38	43	42
11	1	13	14	19	18	38	39	44	43
12	1	14	15	20	19	39	40	45	44
13	1	16	17	22	21	41	42	47	46

(…)

42	1	62	63	68	67	87	88	93	92
43	1	63	64	69	68	88	89	94	93
44	1	64	65	70	69	89	90	95	94
45	1	66	67	72	71	91	92	97	96
46	1	67	68	73	72	92	93	98	97
47	1	68	69	74	73	93	94	99	98
48	1	69	70	75	74	94	95	100	99
49	1	76	77	82	81	101	102	107	106
50	1	77	78	83	82	102	103	108	107
51	1	78	79	84	83	103	104	109	108
52	1	79	80	85	84	104	105	110	109
53	1	81	82	87	86	106	107	112	111
54	1	82	83	88	87	107	108	113	112
55	1	83	84	89	88	108	109	114	113
56	1	84	85	90	89	109	110	115	114
57	1	86	87	92	91	111	112	117	116
58	1	87	88	93	92	112	113	118	117
59	1	88	89	94	93	113	114	119	118
60	1	89	90	95	94	114	115	120	119
61	1	91	92	97	96	116	117	122	121
62	1	92	93	98	97	117	118	123	122
63	1	93	94	99	98	118	119	124	123
64	1	94	95	100	99	119	120	125	124

NX=NY=NZ=4, NXP1=NYP1=NZP1=5
ICELTOT= 64, INODTOT= 125, IBNODTOT= 25



Mesh Generator: mgcube.f (5/5)

```
          IGTOT= 4

          IBT1= NYP1*NZP1
          IBT2= NXP1*NZP1 + IBT1
          IBT3= NXP1*NYP1 + IBT2
          IBT4= NXP1*NYP1 + IBT3


          write (12,'(10i10)') IGTOT
          write (12,'(10i10)') IBT1, IBT2, IBT3, IBT4

          HHH= 'Xmin'
          write (12,'(a80)')  HHH
          write (12,'(10i10)') (IW(ii,1), ii=1,NYP1*NZP1)
          HHH= 'Ymin'
          write (12,'(a80)')  HHH
          write (12,'(10i10)') (IW(ii,2), ii=1,NXP1*NZP1)
          HHH= 'Zmin'
          write (12,'(a80)')  HHH
          write (12,'(10i10)') (IW(ii,3), ii=1,NXP1*NYP1)
          HHH= 'Zmax'
          write (12,'(a80)')  HHH
          write (12,'(10i10)') (IW(ii,4), ii=1,NXP1*NYP1)


          deallocate (IW)
(以下略)
          close (12)
!C===
      stop
      end
```

IGTOT	Group # (Xmin,Ymin,Zmin,Zmax)
IBTx	Accumulated #

Example of “cube.0” (NX=NY=NZ=4) Group Info.

```
         4
        25        50        75       100
Xmin
         1         6        11        16        21        26        31        36        41        46
        51        56        61        66        71        76        81        86        91        96
       101       106       111       116       121
Ymin
         1         2         3         4         5        26        27        28        29        30
        51        52        53        54        55        76        77        78        79        80
       101       102       103       104       105
Zmin
         1         2         3         4         5         6         7         8         9        10
        11        12        13        14        15        16        17        18        19        20
        21        22        23        24        25
Zmax
       101       102       103       104       105       106       107       108       109       110
       111       112       113       114       115       116       117       118       119       120
       121       122       123       124       125

(not in use after this point)
```

Mesh Generation

- Big Technical & Research Issue
  - Complicated Geometry
  - Large Scale
- Parallelization is difficult

- Commercial Mesh Generator
  - FEMAP
    - Interface to CAD Data Format

movie

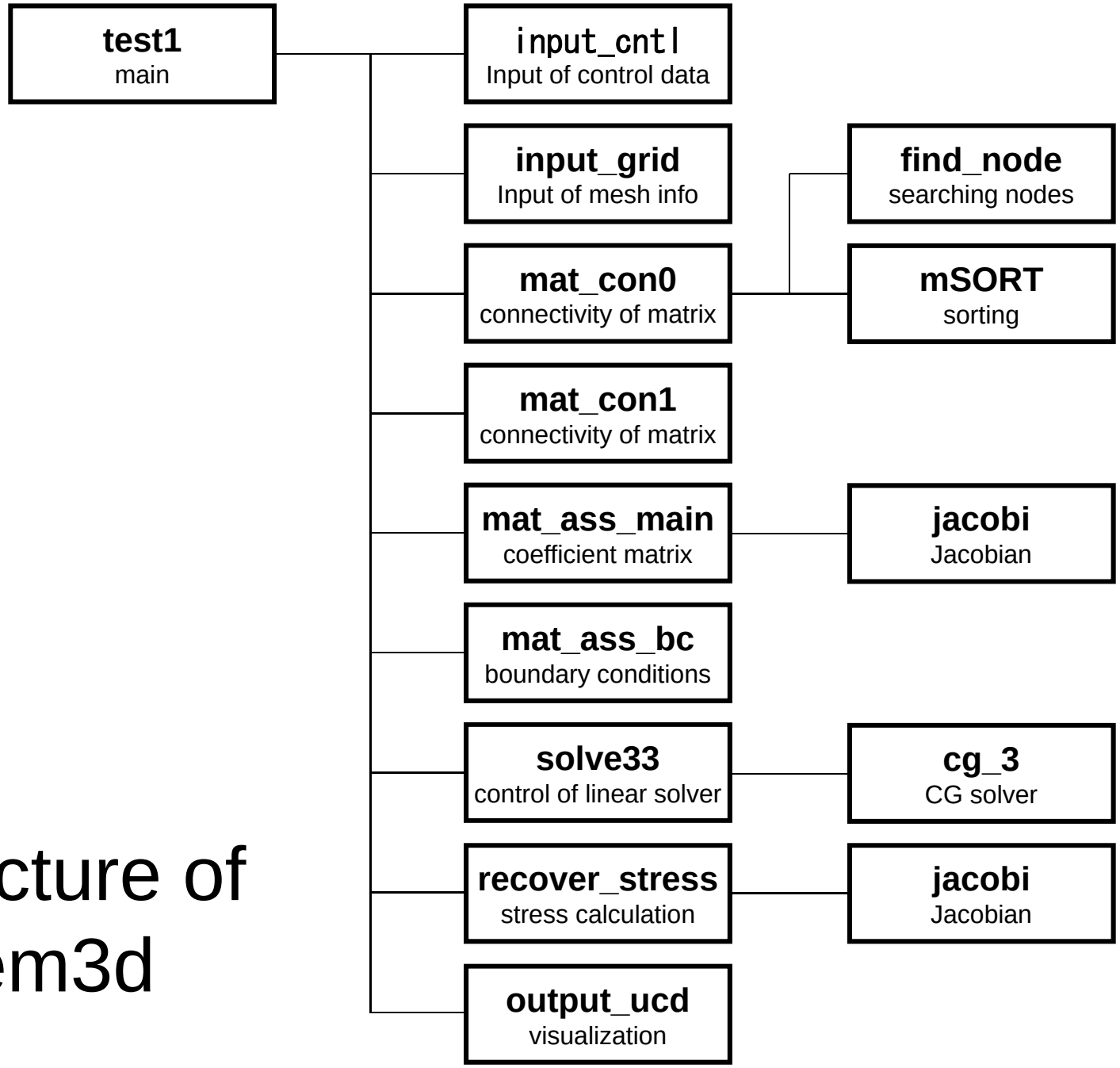
- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- Running the Code
- Data Structure
- **Overview of the Program**
- Computational Issue
- Visualization by ParaView

FEM Procedures: Program

- Initialization
  - Control Data
  - Node, Connectivity of Elements (N: Node#, NE: Elem#)
  - Initialization of Arrays (Global/Element Matrices)
  - Element-Global Matrix Mapping (Index, Item)
- Generation of Matrix
  - Element-by-Element Operations (do icel= 1, NE)
    - Element matrices
    - Accumulation to global matrix
  - Boundary Conditions
- Linear Solver
  - Conjugate Gradient Method
- Calculation of Stress

Structure of fem3d



Main Part

```
#include <stdio.h>
#include <stdlib.h>
FILE* fp_log;
#define GLOBAL_VALUE_DEFINE
#include "pfem_util.h"

extern void INPUT_CNTL();
extern void INPUT_GRID();
extern void MAT_CON0();
extern void MAT_CON1();
extern void MAT_ASS_MAIN();
extern void MAT_ASS_BC();
extern void SOLVE33();
extern void RECOVER_STRESS();
extern void OUTPUT_UCD();
int main()
{

/** Logfile for debug **/
        if( (fp_log=fopen("log.log","w")) == NULL){
                fprintf(stdout,"input file cannot be opened!¥n");
                exit(1);
        }

        INPUT_CNTL();
        INPUT_GRID();


        MAT_CON0();
        MAT_CON1();

        MAT_ASS_MAIN();
        MAT_ASS_BC()  ;

        SOLVE33();

        RECOVER_STRESS();
        OUTPUT_UCD()    ;
}
```

Global Variables: pfem_util.h (1/3)

Name	Type	Size	I/O	Definition
fname	C	[80]	I	Name of mesh file
N,NP	I		I	# Node
ICELTOT	I		I	# Element
NODGRPtot	I		I	# Node Group
XYZ	R	[N][3]	I	Node Coordinates
ICELNOD	I	[ICELTOT][8]	I	Element Connectivity
NODGRP_INDEX	I	[NODGRPtot+1]	I	# Node in each Node Group
NODGRP_ITEM	I	[NODGRP_INDEX[NODGRPTOT+1]]	I	Node ID in each Node Group
NODGRP_NAME	C80	[NODGRP_INDEX[NODGRPTOT+1]]	I	Name of NodeGroup
NL, NU	I		O	# Upper/Lower Triangular Non-Zero Off-Diagonals at each node
NPL, NPU	I		O	# Upper/Lower Triangular Non-Zero Off-Diagonals
D	R	[9*N]	O	Diagonal Block of Global Matrix
B,X	R	[3*N]	O	RHS, Unknown Vector

Global Variables: pfem_util.h (2/3)

Name	Type	Size	I/O	Definition
ALUG	R	[9*N]	O	Full LU factorization of Diagonal Blocks D
AL,AU	R	[9*NPL],[9*NPU]	O	Upper/Lower Triangular Components of Global Matrix
indexL, indexU	I	[N+1]	O	# Non-Zero Off-Diagonal Blocks
itemL, itemU	I	[NPL], [NPU]	O	Column ID of Non-Zero Off-Diagonal Blocks
INL, INU	I	[N]	O	Number of Off-Diagonal Blocks at Each Node
IAL, IAU	I	[N][NL],[N][NU]	O	Off-Diagonal Blocks at Each Node
IWKX	I	[N][2]	O	Work Arrays
METHOD	I		I	Iterative Method (fixed as 1)
PRECOND	I		I	Preconditioning Method (0: SSOR, 1: Block Diagonal Scaling)
ITER, ITERactual	I		I	Number of CG Iterations (MAX, Actual)
RESID	R		I	Convergence Criteria (fixed as 1.e-8)
SIGMA_DIAG	R		I	Coefficient for LU Factorization (fixed as 1.00)
pfemIarray	I	[100]	O	Integer Parameter Array
pfemRarray	R	[100]	O	Real Parameter Array

Global Variables: pfem_util.h (3/3)

Name	Type	Size	I/O	Definition
O8th	R		I	= 0.125
PNQ, PNE, PNT	R	[2][2][8]	O	$\frac{\partial N_i}{\partial \xi}, \frac{\partial N_i}{\partial \eta}, \frac{\partial N_i}{\partial \zeta} (i = 1 \sim 8)$ at each Gaussian Quad. Point
POS, WEI	R	[2]	O	Coordinates, Weighting Factor at each Gaussian Quad. Point
NCOL1, NCOL2	I	[100]	O	Work arrays for sorting
SHAPE	R	[2][2][2][8]	O	N_i (i=1~8) at each Gaussian Quad Point
PNX, PNY, PNZ	R	[2][2][2][8]	O	$\frac{\partial N_i}{\partial x}, \frac{\partial N_i}{\partial y}, \frac{\partial N_i}{\partial z} (i = 1 \sim 8)$ at each Gaussian Quad. Point
DETJ	R	[2][2][2]	O	Determinant of Jacobian Matrix at each Gaussian Quad. Point
ELAST, POISSON	R		I	Young's Modulus, Poisson's Ratio
SIGMA_N, TAU_N	R	[N][3]	O	Normal/Shear Stress at each Node

INPUT_CNTL: Control Data

```c
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
/** **/
void INPUT_CNTL()
{
        FILE *fp;
        if( (fp=fopen("INPUT.DAT","r")) == NULL){
                fprintf(stdout,"input file cannot be opened!¥n");
                exit(1);
        }
        fscanf(fp,"%s",fname);
        fscanf(fp,"%d %d",&METHOD,&PRECOND);
        fscanf(fp,"%d",&iterPREmax);
        fscanf(fp,"%d",&ITER);
        fscanf(fp, "%lf %lf", &ELAST, &POISSON);
        fclose(fp);

        if( ( iterPREmax < 1 ) ){
                iterPREmax= 1;
        }
        if( ( iterPREmax > 4 ) ){
                iterPREmax= 4;
        }

        SIGMA_DIAG= 1.0;
        SIGMA     = 0.0;
        RESID     = 1.e-8;
        NSET      = 0;

        pfemRarray[0]= RESID;
        pfemRarray[1]= SIGMA_DIAG;
        pfemRarray[2]= SIGMA;

        pfemIarray[0]= ITER;
        pfemIarray[1]= METHOD;
        pfemIarray[2]= PRECOND;
        pfemIarray[3]= NSET;
        pfemIarray[4]= iterPREmax;
}
```

INPUT.DAT

```
cube.0          fname
1  0            METHOD, PRECOND
1               iterPREmax (not in use)
2000            ITER
1.0 0.3         ELAST, POISSON
```

INPUT_GRID (1/2)

```
#include <stdio.h>
#include <stdlib.h>
#include "pfem_util.h"
#include "allocate.h"
void INPUT_GRID()
{
        FILE *fp;
        int i,j,k,ii,kk,nn,icel,iS,iE;
        int NTYPE,IMAT;

        if( (fp=fopen(fname,"r")) == NULL){
                fprintf(stdout,"input file cannot be opened!¥n");
                exit(1);
        }
/**
        NODE
**/
        fscanf(fp,"%d",&N);

        NP=N;
        XYZ=(KREAL**)allocate_matrix(sizeof(KREAL),N,3);

        for(i=0;i<N;i++){
                for(j=0;j<3;j++){
                        XYZ[i][j]=0.0;
                }
        }

        for(i=0;i<N;i++){
                fscanf(fp,"%d %lf %lf %lf",&ii,&XYZ[i][0],&XYZ[i][1],&XYZ[i][2]);
        }
```

XYZ[N][3]

allocate, deallocate

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

void deallocate_vector(void *a)
{
  free( a );
}

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

void deallocate_matrix(void **aa)
{
        free( aa );
}
```

Same interface with FORTRAN

INPUT_GRID (1/2)

```
/**
        ELEMENT
**/
        fscanf(fp,"%d",&ICELTOT);

        ICELNOD=(KINT**)allocate_matrix(sizeof(KINT),ICELTOT,8);
        for(i=0;i<ICELTOT;i++) fscanf(fp,"%d",&NTYPE);


        for(icel=0;icel<ICELTOT;icel++){
                fscanf(fp,"%d %d %d %d %d %d %d %d %d %d",&ii,&IMAT,
                &ICELNOD[icel][0],&ICELNOD[icel][1],&ICELNOD[icel][2],&ICELNOD[icel][3],
                &ICELNOD[icel][4],&ICELNOD[icel][5],&ICELNOD[icel][6],&ICELNOD[icel][7]);
        }
/**
        NODE grp.  info.
**/
        fscanf(fp,"%d",&NODGRPtot);

        NODGRP_INDEX=(KINT*  )allocate_vector(sizeof(KINT),NODGRPtot+1);
        NODGRP_NAME =(CHAR80*)allocate_vector(sizeof(CHAR80),NODGRPtot);
        for(i=0;i<NODGRPtot+1;i++) NODGRP_INDEX[i]=0;

        for(i=0;i<NODGRPtot;i++) fscanf(fp,"%d",&NODGRP_INDEX[i+1]);
        nn=NODGRP_INDEX[NODGRPtot];
        NODGRP_ITEM=(KINT*)allocate_vector(sizeof(KINT),nn);

        for(k=0;k<NODGRPtot;k++){
                iS= NODGRP_INDEX[k];
                iE= NODGRP_INDEX[k+1];
                fscanf(fp,"%s",NODGRP_NAME[k].name);
                nn= iE - iS;
                if( nn != 0 ){
                        for(kk=iS;kk<iE;kk++) fscanf(fp,"%d",&NODGRP_ITEM[kk]);
                }
        }

        fclose(fp);
}
```

ICELNOD[i][j]:
Node ID starting from “1”.
Element ID starts from “0”.

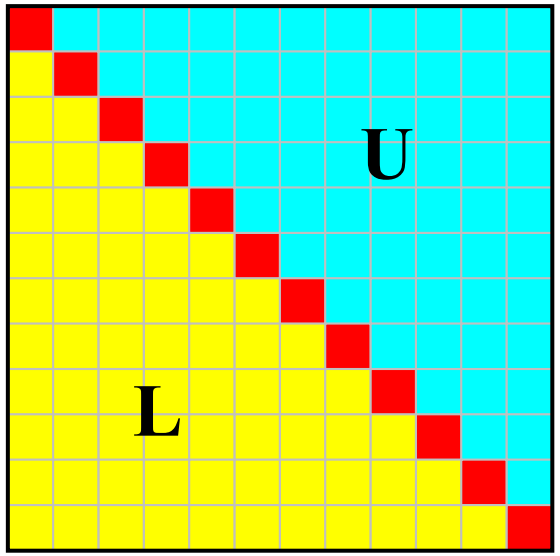
INPUT_GRID (2/2)

```
/**
        ELEMENT
**/
        fscanf(fp,"%d",&ICELTOT);

        ICELNOD=(KINT**)allocate_matrix(sizeof(KINT),ICELTOT,8);
        for(i=0;i<ICELTOT;i++) fscanf(fp,"%d",&NTYPE);


        for(icel=0;icel<ICELTOT;icel++){
                fscanf(fp,"%d %d %d %d %d %d %d %d %d %d",&ii,&IMAT,
                &ICELNOD[icel][0],&ICELNOD[icel][1],&ICELNOD[icel][2],&ICELNOD[icel][3],
                &ICELNOD[icel][4],&ICELNOD[icel][5],&ICELNOD[icel][6],&ICELNOD[icel][7]);
        }
/**
        NODE grp. info.
**/
        fscanf(fp,"%d",&NODGRPtot);

        NODGRP_INDEX=(KINT*  )allocate_vector(sizeof(KINT),NODGRPtot+1);
        NODGRP_NAME =(CHAR80*)allocate_vector(sizeof(CHAR80),NODGRPtot);
        for(i=0;i<NODGRPtot+1;i++) NODGRP_INDEX[i]=0;

        for(i=0;i<NODGRPtot;i++) fscanf(fp,"%d",&NODGRP_INDEX[i+1]);
        nn=NODGRP_INDEX[NODGRPtot];
        NODGRP_ITEM=(KINT*)allocate_vector(sizeof(KINT),nn);

        for(k=0;k<NODGRPtot;k++){
                iS= NODGRP_INDEX[k];
                iE= NODGRP_INDEX[k+1];
                fscanf(fp,"%s",NODGRP_NAME[k].name);
                nn= iE - iS;
                if( nn != 0 ){
                        for(kk=iS;kk<iE;kk++) fscanf(fp,"%d",&NODGRP_ITEM[kk]);
                }
        }

        fclose(fp);
}
```

Node Group:
Node ID’s start from “1”

Some Features of “fem3d”

- Non-Zero Off-Diagonals
  - Upper/Lower triangular components are stored separately（上三角成分，下三角成分）.



- Stored as Block
  - Vector: 3-components per node
  - Matrix: 9-components per block
  - Processed as “block” based on 3 variables on each node (not each component of matrix)

Coef. Matrix derived from FEM

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

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

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

Variables/Arrays in 1d.f, 1d.c related to coefficient matrix

name	type	size	description
N	I	-	# Unknowns
NPLU	I	-	# Non-Zero Off-Diagonal Components
Diag(:)	R	N	Diagonal Components
U(:)	R	N	Unknown Vector
Rhs(:)	R	N	RHS Vector
Index(:)	I	0:N N+1	Off-Diagonal Components (Number of Non-Zero Off-Diagonals at Each ROW)
Item(:)	I	NPLU	Off-Diagonal Components (Corresponding Column ID)
AMat(:)	R	NPLU	Off-Diagonal Components (Value)

Only non-zero components are stored according to “Compressed Row Storage”.

Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

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

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

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

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

Mat-Vec. Multiplication for Sparse Matrix

Compressed Row Storage (CRS)

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

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

Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

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

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

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

Compressed Row Storage (CRS)

$$
\begin{array}{c|cccccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
1 & 1.1 & 2.4 & 0 & 0 & 3.2 & 0 & 0 & 0 \\
2 & 4.3 & 3.6 & 0 & 2.5 & 0 & 3.7 & 0 & 9.1 \\
3 & 0 & 0 & 5.7 & 0 & 1.5 & 0 & 3.1 & 0 \\
4 & 0 & 4.1 & 0 & 9.8 & 2.5 & 2.7 & 0 & 0 \\
5 & 3.1 & 9.5 & 10.4 & 0 & 11.5 & 0 & 4.3 & 0 \\
6 & 0 & 0 & 6.5 & 0 & 0 & 12.4 & 9.5 & 0 \\
7 & 0 & 6.4 & 2.5 & 0 & 0 & 1.4 & 23.1 & 13.1 \\
8 & 0 & 9.5 & 1.3 & 9.6 & 0 & 3.1 & 0 & 51.3
\end{array}
$$

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

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

N= 8

Diagonal Components

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

Compressed Row Storage (CRS)：C

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

Compressed Row Storage (CRS) : C

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

NPLU= 25 (=Index[N])

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

Compressed Row Storage (CRS) : C

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

NPLU= 25 (=Index[N])

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

Compressed Row Storage (CRS)：C

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

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

Compressed Row Storage (CRS)：C

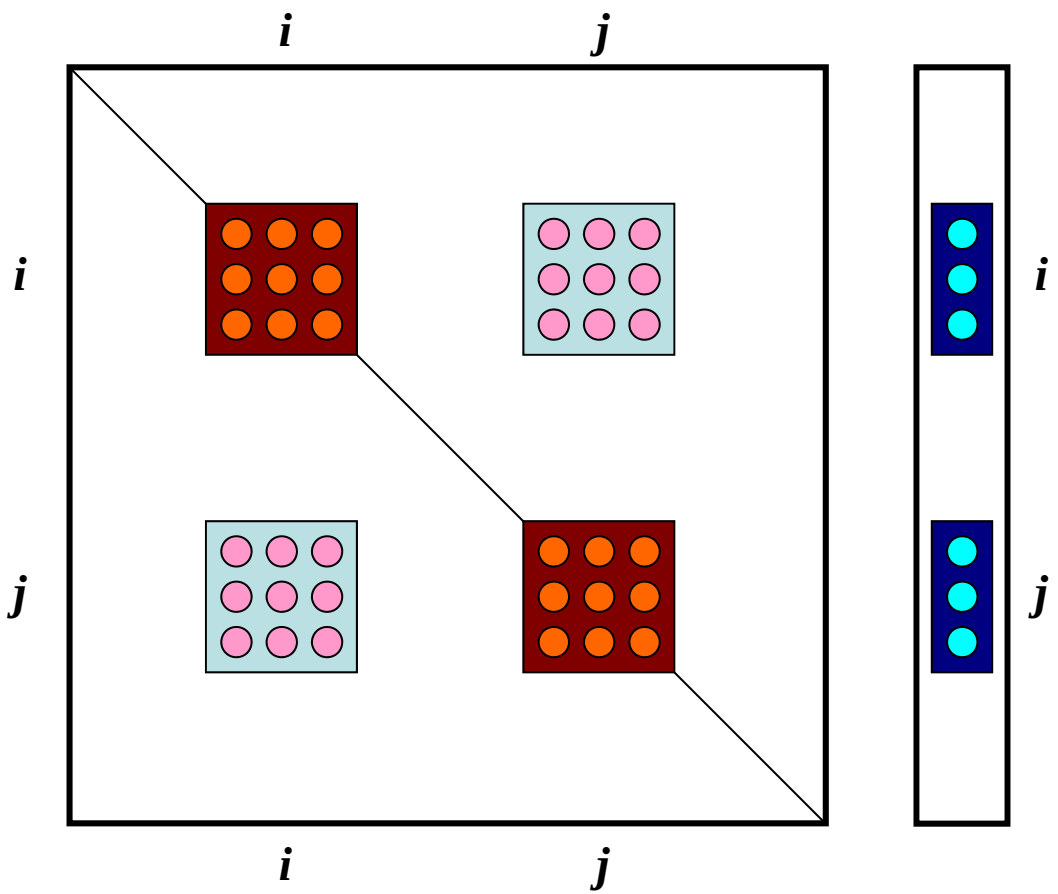
0	1.1 ◎	2.4 ①,0	3.2 ④,1		
1	3.6 ①	4.3 ◎,2	2.5 ③,3	3.7 ⑤,4	9.1 ⑦,5
2	5.7 ②	1.5 ④,6	3.1 ⑥,7		
3	9.8 ③	4.1 ①,8	2.5 ④,9	2.7 ⑤,10	
4	11.5 ④	3.1 ◎,11	9.5 ①,12	10.4 ②,13	4.3 ⑥,14
5	12.4 ⑤	6.5 ②,15	9.5 ⑥,16		
6	23.1 ⑥	6.4 ①,17	2.5 ②,18	1.4 ⑤,19	13.1 ⑦,20
7	51.3 ⑦	9.5 ①,21	1.3 ②,22	9.6 ③,23	3.1 ⑤,24

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

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

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

Storing 3x3 Block (1/3)

- Less memory requirement
  - Index, Item



Storing 3x3 Block (2/3)

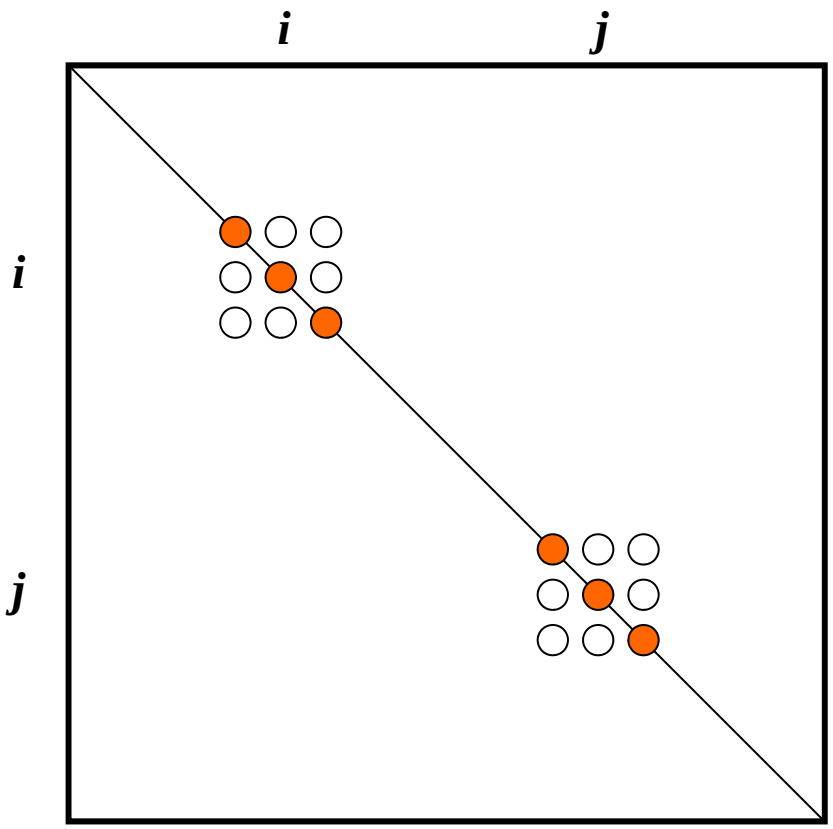
- Computational Efficiency
  - Ratio of (Comptation/Indirect Memory Access) is larger
  - >2x speed-up both for vector/scalar processors
    - Contiguous memory access, Cache Utilization, Larger Flop/Byte

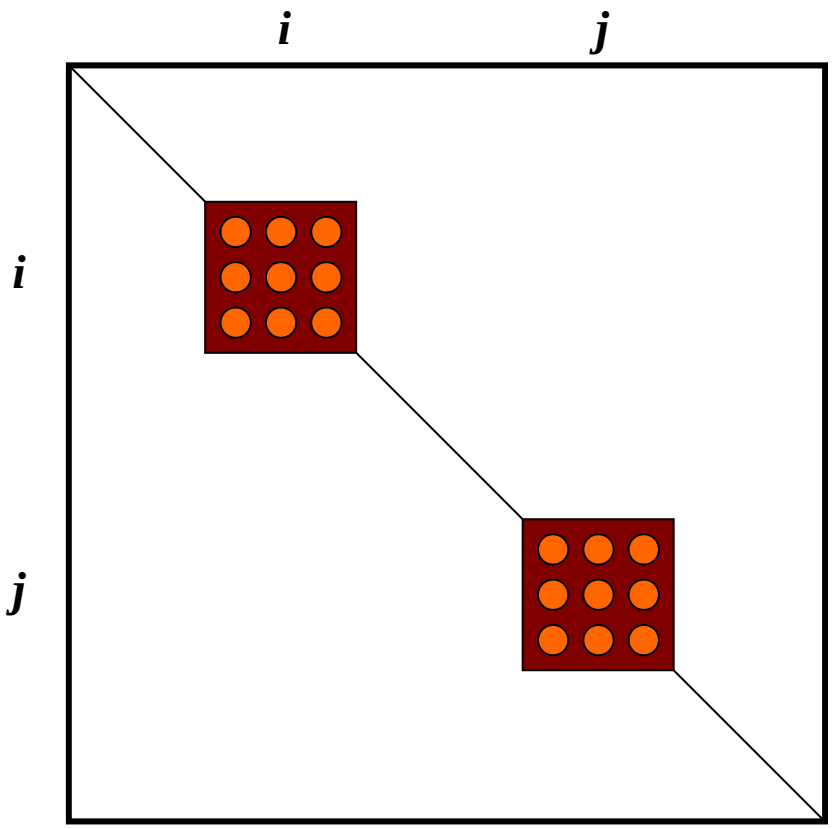
```
do i= 1, 3*N
  Y(i)= D(i)*X(i)
  do k= index(i-1)+1, index(i)
    kk= item(k)
    Y(i)= Y(i) + AMAT(k)*X(kk)
  enddo
enddo
```

```
do i= 1, N
  X1= X(3*i-2)
  X2= X(3*i-1)
  X3= X(3*i)
  Y(3*i-2)= D(9*i-8)*X1+D(9*i-7)*X2+D(9*i-6)*X3
  Y(3*i-1)= D(9*i-5)*X1+D(9*i-4)*X2+D(9*i-3)*X3
  Y(3*I  )= D(9*i-2)*X1+D(9*i-1)*X2+D(9*I  )*X3
  do k= index(i-1)+1, index(i)
    kk= item(k)
    X1= X(3*kk-2)
    X2= X(3*kk-1)
    X3= X(3*kk)
    Y(3*i-2)= Y(3*i-2)+AMAT(9*k-8)*X1+AMAT(9*k-7)*X2 &
                                     +AMAT(9*k-6)*X3
    Y(3*i-1)= Y(3*i-1)+AMAT(9*k-5)*X1+AMAT(9*k-4)*X2 &
                                     +AMAT(9*k-3)*X3
    Y(3*I  )= Y(3*I  )+AMAT(9*k-2)*X1+AMAT(9*k-1)*X2 &
                                     +AMAT(9*k  )*X3
  enddo
enddo
```

Storing 3x3 Block (3/3)

- Stabilization of Computation（計算の安定化）
  - Instead of division by diagonal components, full LU factorization of 3x3 Diagonal Block is applied.
  - Effective for ill-conditioned problems

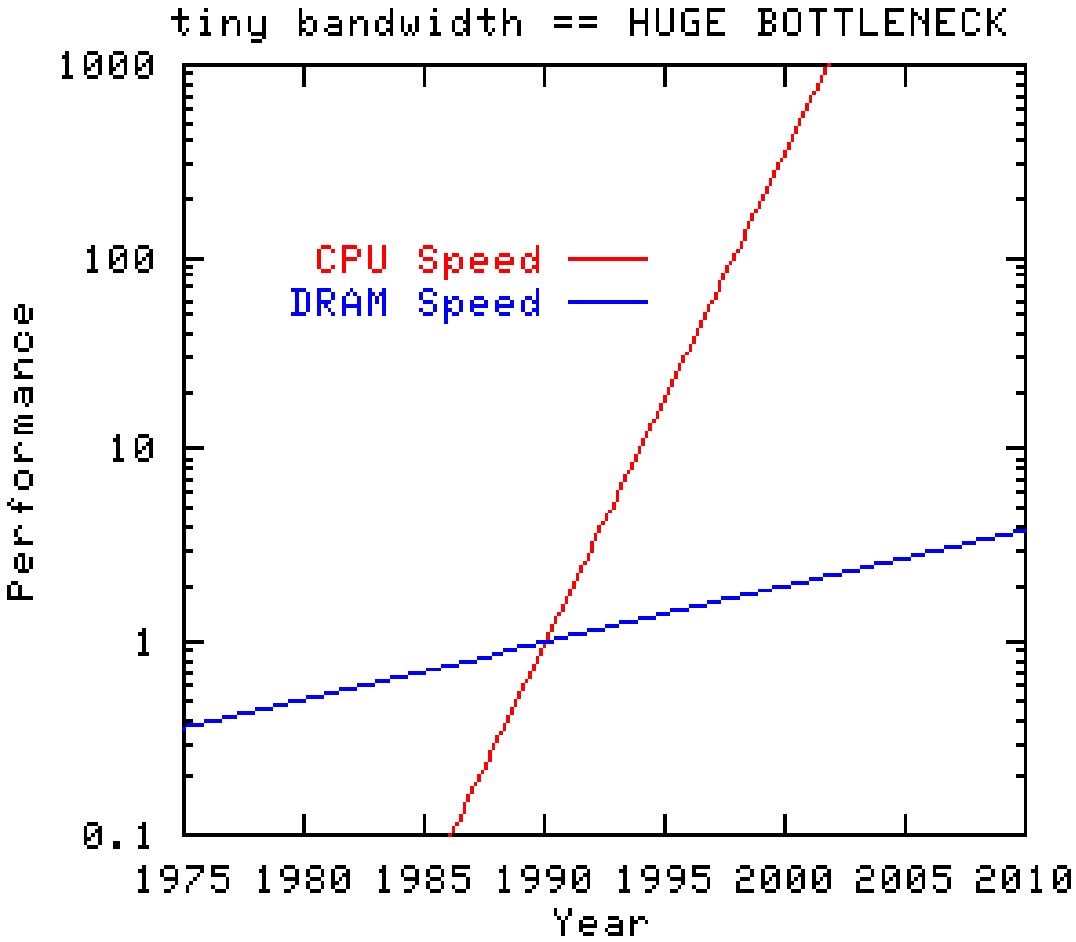




- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- Running the Code
- Data Structure
- Overview of the Program
- **Computational Issue**
- Visualization by ParaView

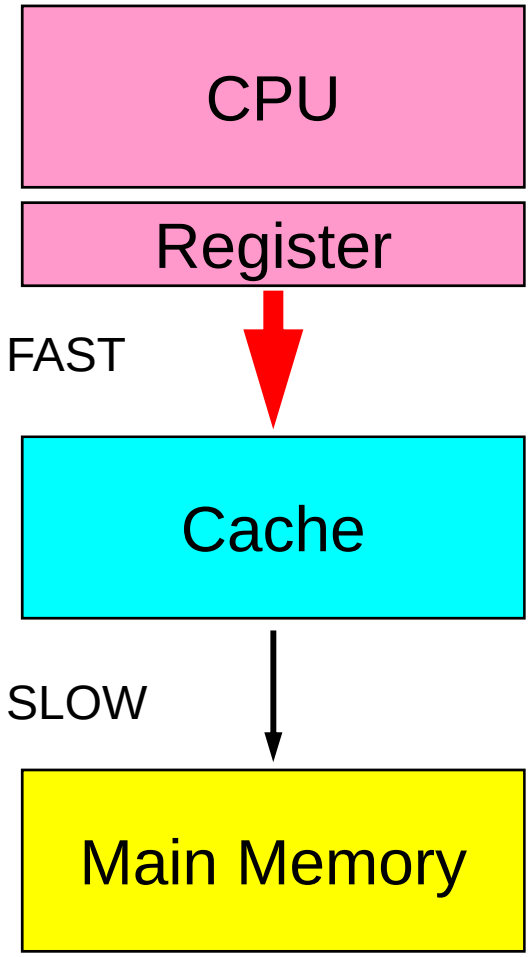
Performance Gap between CPU and Memory



http://www.streambench.org/

Scalar Processor

Hierarchical Structure CPU-Cache-Memory

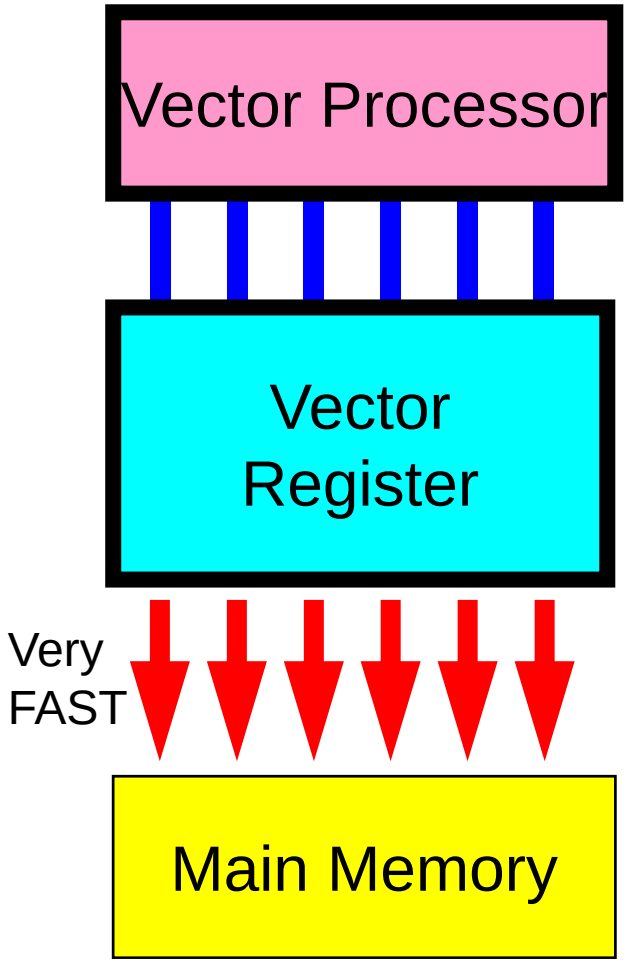


Small Capacity (MB)
Expensive
Large (O(10^9) Transistors)

Large Capacity (GB)
Cheap

Vector Processor

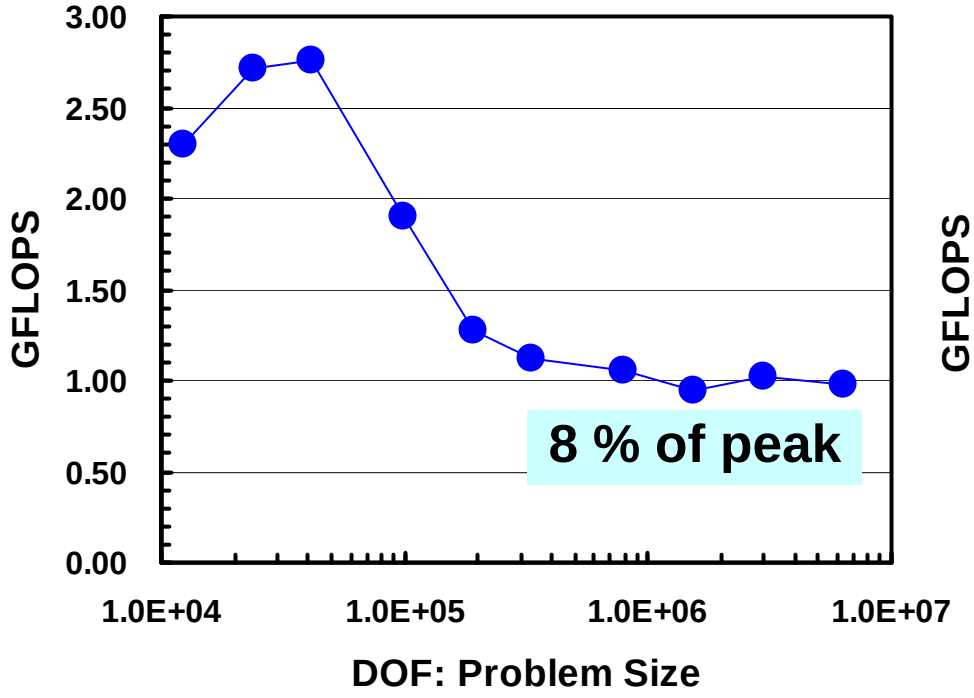
Vector Register/Fast Memory

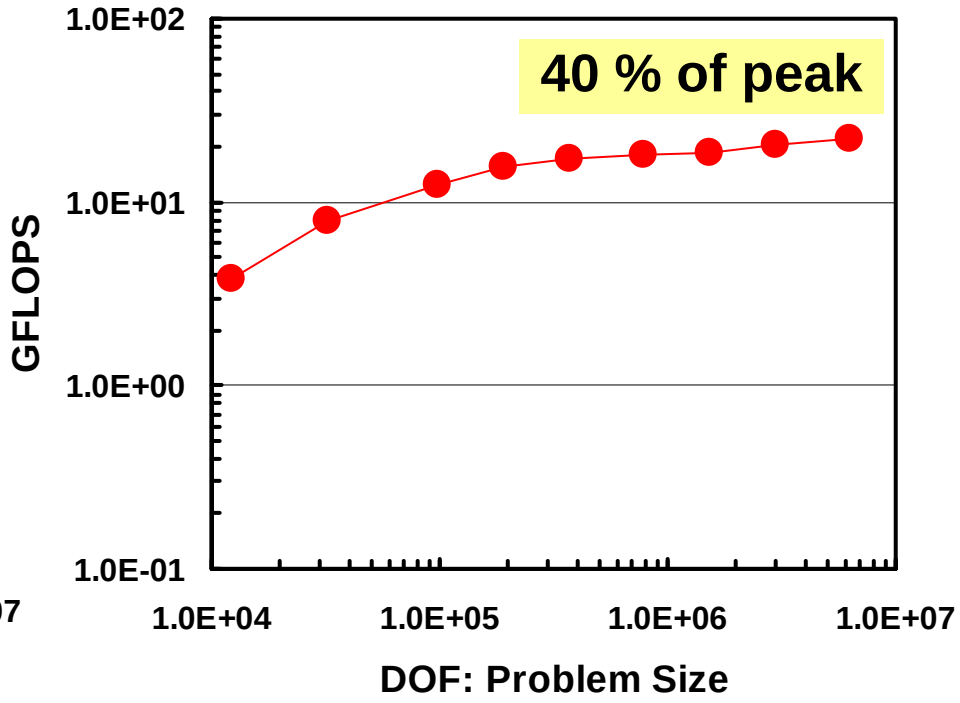


- Good for Parallel Processing of Simple Do Loops
- Suitable for Simple/Huge Calculations

```
do i= 1, N
  A(i)= B(i) + C(i)
enddo
```

Typical Behavior: CG for Sparse Matrices by FEM



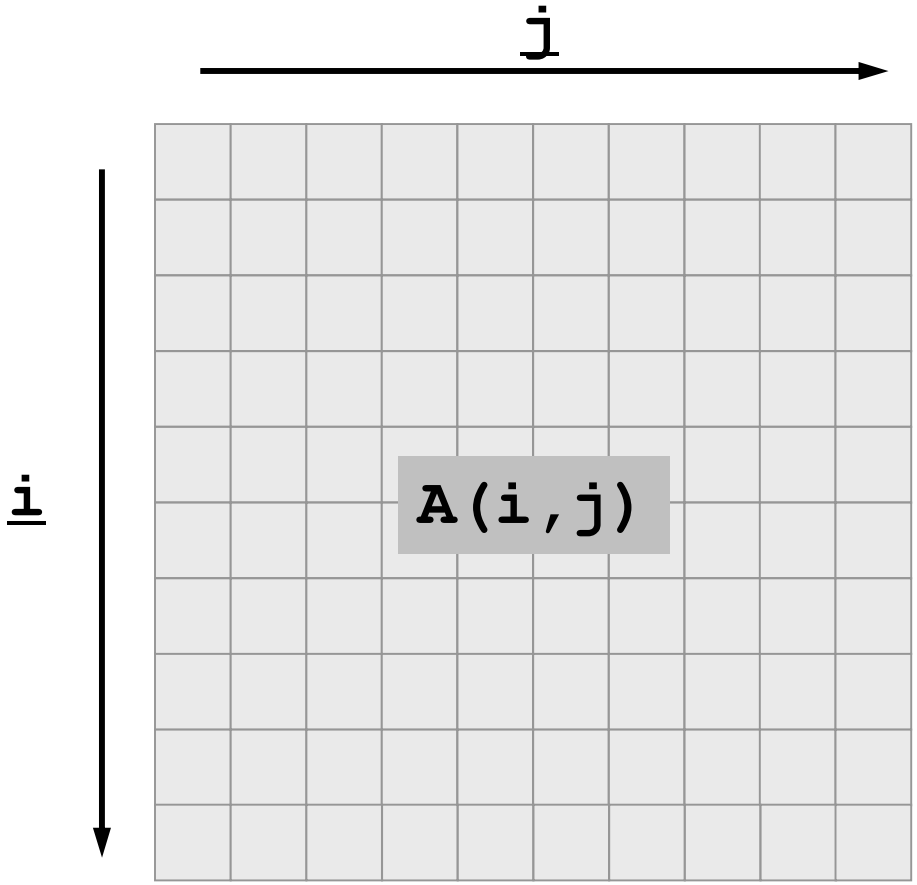


IBM-SP3:
Performance is better for smaller problems, because of cache effect.

Earth Simulator:
Performance is better for larger problems, because of vector length.

Tuning according to Processor

- Optimum Memory Access !!



Tuning according to Processor (cont.)

- Vector Processor
  - Effect of Loop Length
- Scalar Processor
  - Best use of cache: small segments of data
  - Reduction of memory latency, increase of memory bandwidth, but multi-/many-cores compensate these improvement.
- Common Issues (Vector/Scalar)
  - Contiguous Memory Access
  - Locality
  - Changing order of computation may affect restuls.

Typical Tuning for Scalar Processors

- Loop Unrolling
  - Reduction of Loop Overhead
  - Reduction of Load/Store Operations
- Blocking
  - Reduction of Cache Misses

Scalar

Loop Unrolling

Reduction of Load/Store Operations (1/4)

- Ratio of computations compared to loop operation increases by loop unrolling.

```
N= 10000

do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
  enddo
enddo

do j= 1, N-1, 2
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
  enddo
enddo

do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
    A(i)= A(i) + B(i)*C(i,j+1)
    A(i)= A(i) + B(i)*C(i,j+2)
    A(i)= A(i) + B(i)*C(i,j+3)
  enddo
enddo
```

T2K Tokyo (sec.)

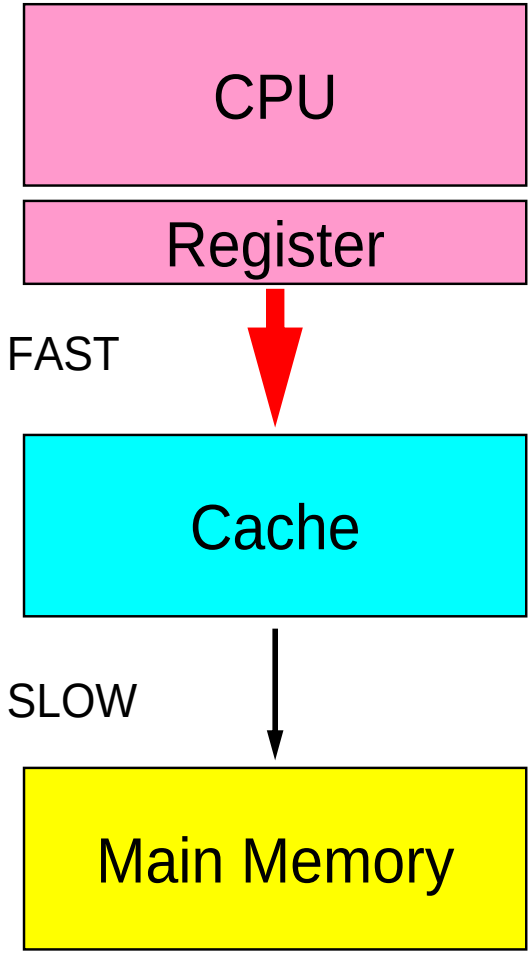
```
4.023438E-01
3.085938E-01
2.617188E-01
```

Scalar

Loop Unrolling

Reduction of Load/Store Operations (2/4)

- Load: Memory~Cache~Register
- Store: Register~Cache~Memory
- Efficient if Load/Store operations are few



Scalar

Loop Unrolling

Reduction of Load/Store Operations (3/4)

```
do j= 1, N
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
   Store  Load   Load Load
  enddo
enddo
```

- In each loop, load/store operations occurs on arrays (A, B, C)

Scalar

Loop Unrolling

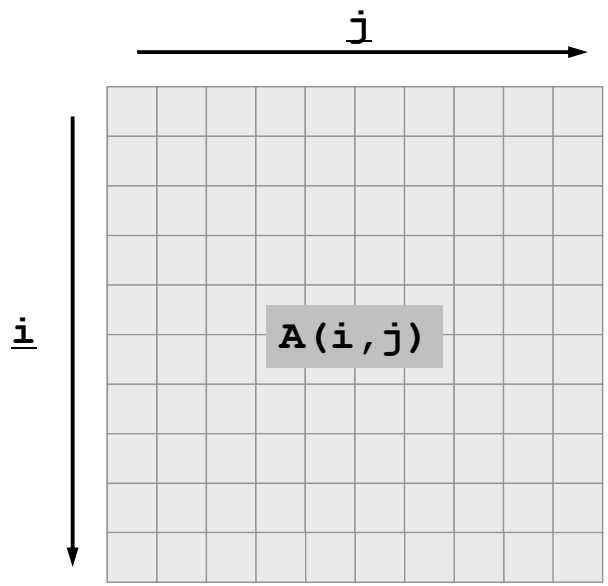
Reduction of Load/Store Operations (4/4)

```
do j= 1, N-3, 4
  do i= 1, N
    A(i)= A(i) + B(i)*C(i,j)
          load   load load
    A(i)= A(i) + B(i)*C(i,j+1)
    A(i)= A(i) + B(i)*C(i,j+2)
    A(i)= A(i) + B(i)*C(i,j+3)
    store
  enddo
enddo
```

- Load occurs at the beginning of the loop, and store only occurs at the end of the loop. During the loop, data is saved on register.
- Be careful about the sequence of computation.

Effect of Loop Exchange (1/2)

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```



- FORTRAN: Address of A(i,j) component is in the sequence of A(1,1), A(2,1), A(3,1),…, A(N,1), A(1,2), A(2,2),…, A(1,N), A(2,N),…, A(N,N)
  - C: A[0][0], A[0][1], A[0][2], …, A[N-1][0], A[N-1][1],…,A[N-1][N-1]
- Each component should be accessed according to this sequence.

Scalar

Effect of Loop Exchange (2/2)

```
TYPE-A
do i= 1, N
  do j= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo

TYPE-B
do j= 1, N
  do i= 1, N
    A(i,j)= A(i,j) + B(i,j)
  enddo
enddo
```

```
TYPE-A
for (j=0; j<N; j++){
  for (i=0; i<N; i++){
    A[i][j]= A[i][j] + B[i][j];
  }
}

TYPE-B
for (i=0; i<N; i++){
  for (j=0; j<N; j++){
    A[i][j]= A[i][j] + B[i][j];
  }
}
```

T2K Tokyo (sec.) by FORTRAN

```
### N ###     512
 A    3.125000E-02
 B    3.906250E-03
### N ###    1024
 A    2.343750E-01
 B    7.812500E-03
### N ###    1536
 A    3.476563E-01
 B    1.562500E-02
### N ###    2048
 A    9.296875E-01
 B    3.125000E-02
### N ###    2560
 A    9.687500E-01
 B    4.687500E-02
### N ###    3072
 A    2.152344E+00
 B    7.421875E-02
### N ###    3584
 A    1.921875E+00
 B    9.765625E-02
### N ###    4096
 A    3.804688E+00
 B    1.250000E-01
```

Scalar

Effect of Blocking on Reduction of Cache Miss (1/7)

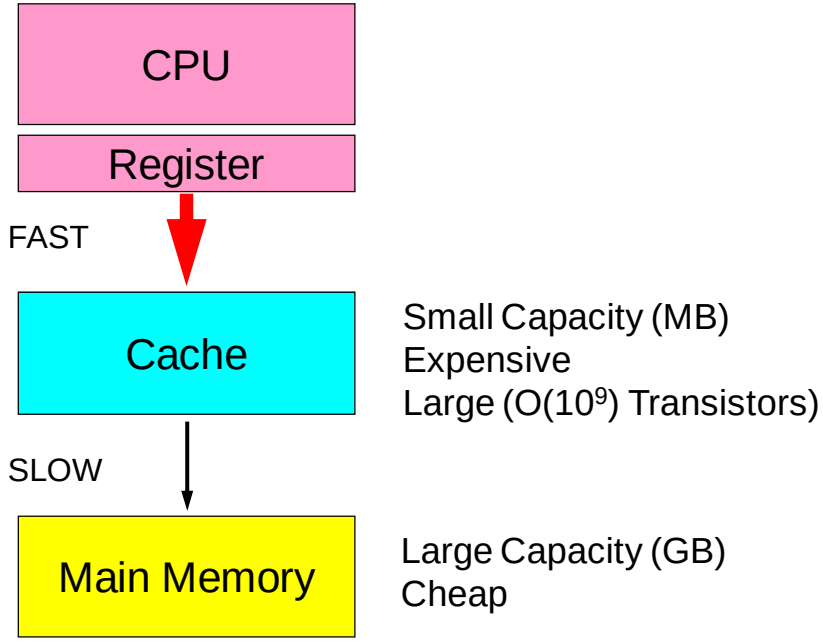
```
do i= 1, NN
  do j= 1, NN
    A(j,i)= A(j,i) + B(i,j)
  enddo
enddo
```

```
for (j=0;j<NN;j++){
  for (i=0;i<NN;i++){
    A[j][i]= A[j][i] + B[i][j];
  }
}
```

- This type of computation could happen.

Effective Use of Cache

- Cache
  - Consists of “cache lines” where size of each cache line is 64-128bytes.
  - Cache line is the unit for requesting data to main memory

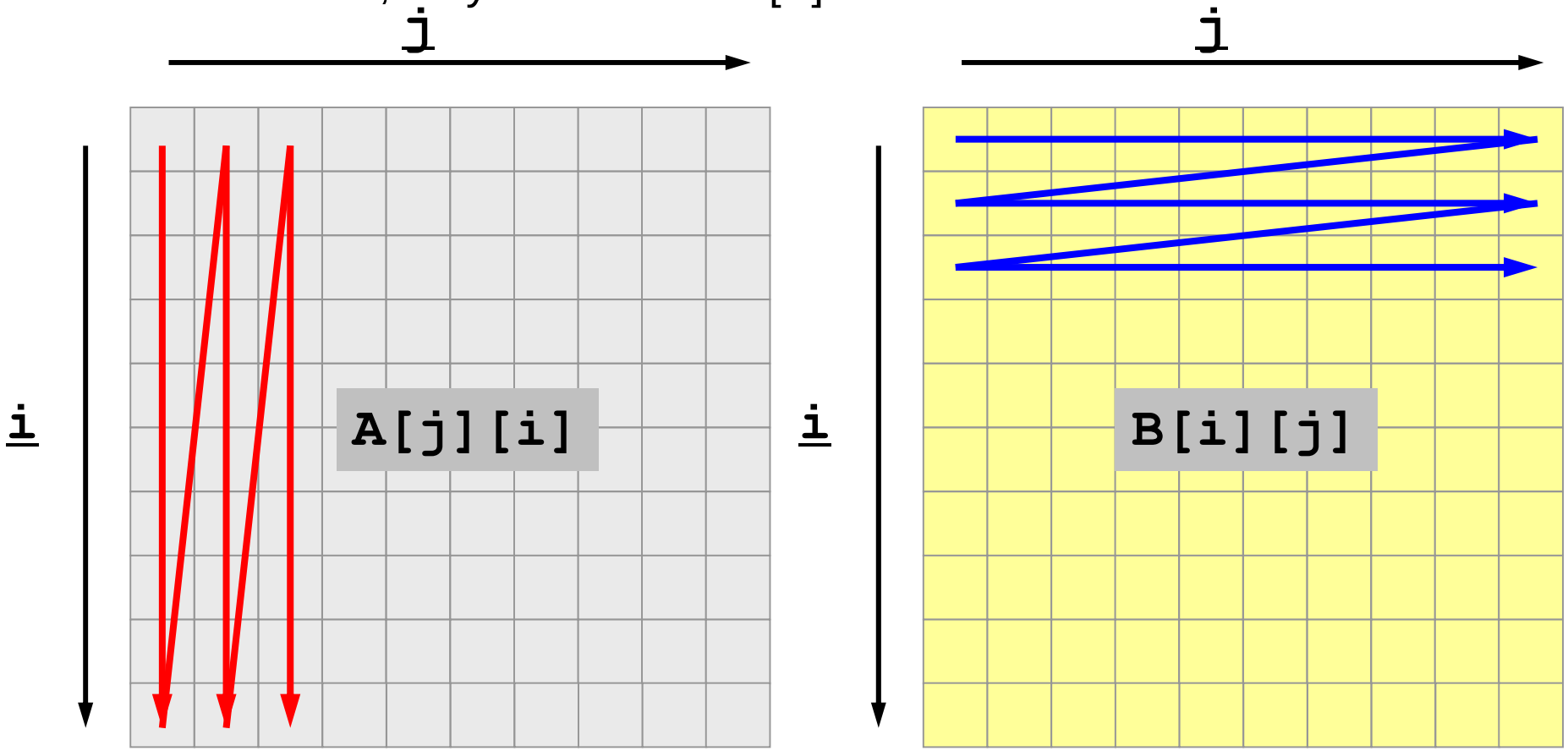


- TLB (Translate Lookaside Buffer)
  - Buffer for Address Translation（アドレス変換バッファ）
    - Translation: Virtual Buffer – Real Buffer
  - Cache for TLB
    - e.g. 128×8 kbytes: can be changed at linkage
  - If cache is well-utilized, TLB-miss does not happen.

Scalar

Effect of Blocking (2/7)

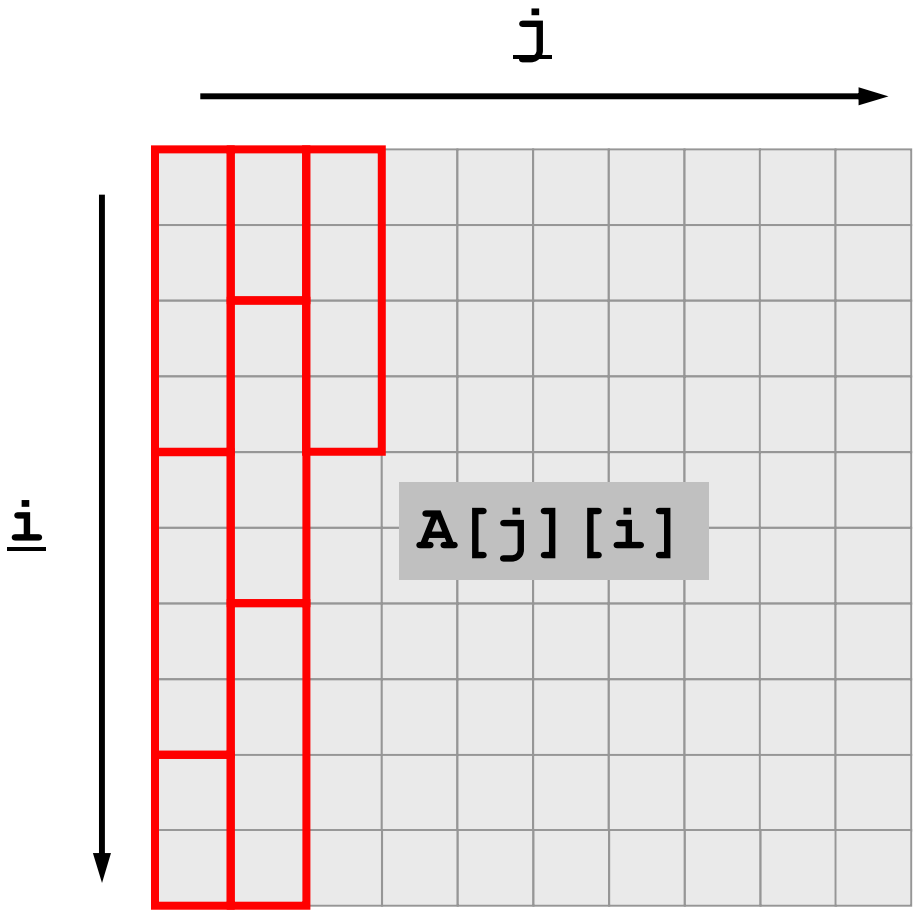
- Memory access patterns of [A] and [B] are different:
  - In this case, very inefficient for [B]



Scalar

Effect of Blocking (3/7)

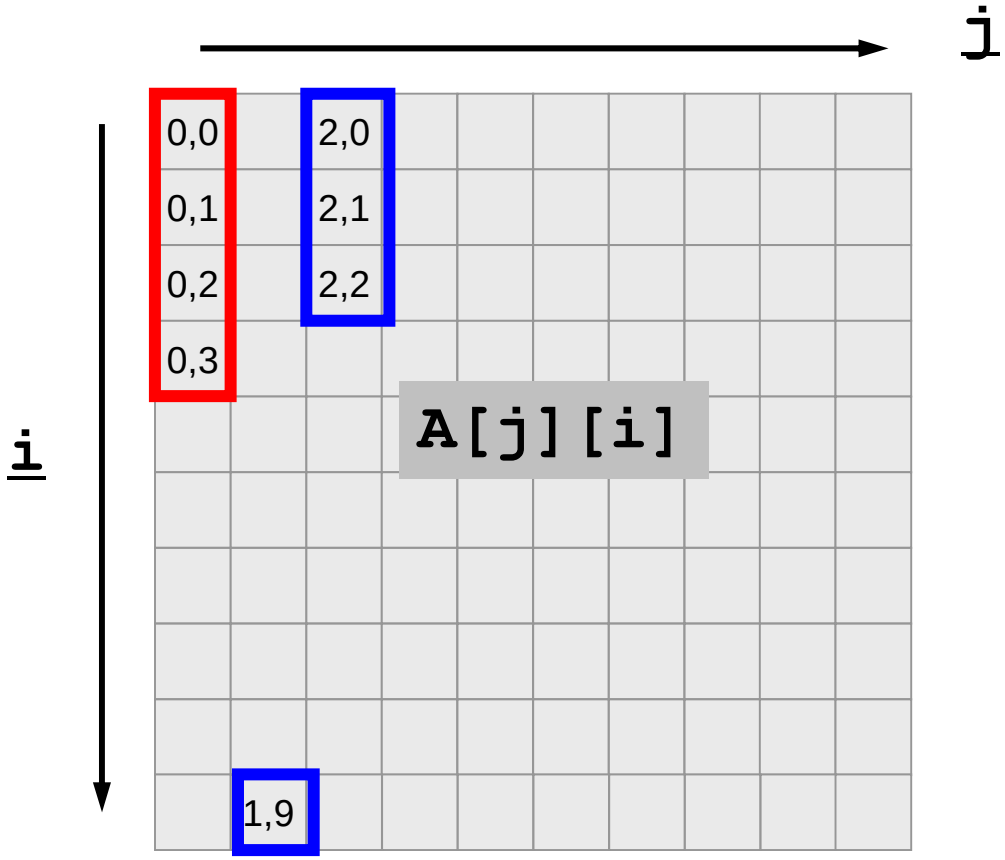
- If the size of cache line is 4-words, values of array [A] is transferred to cache as follows:



Scalar

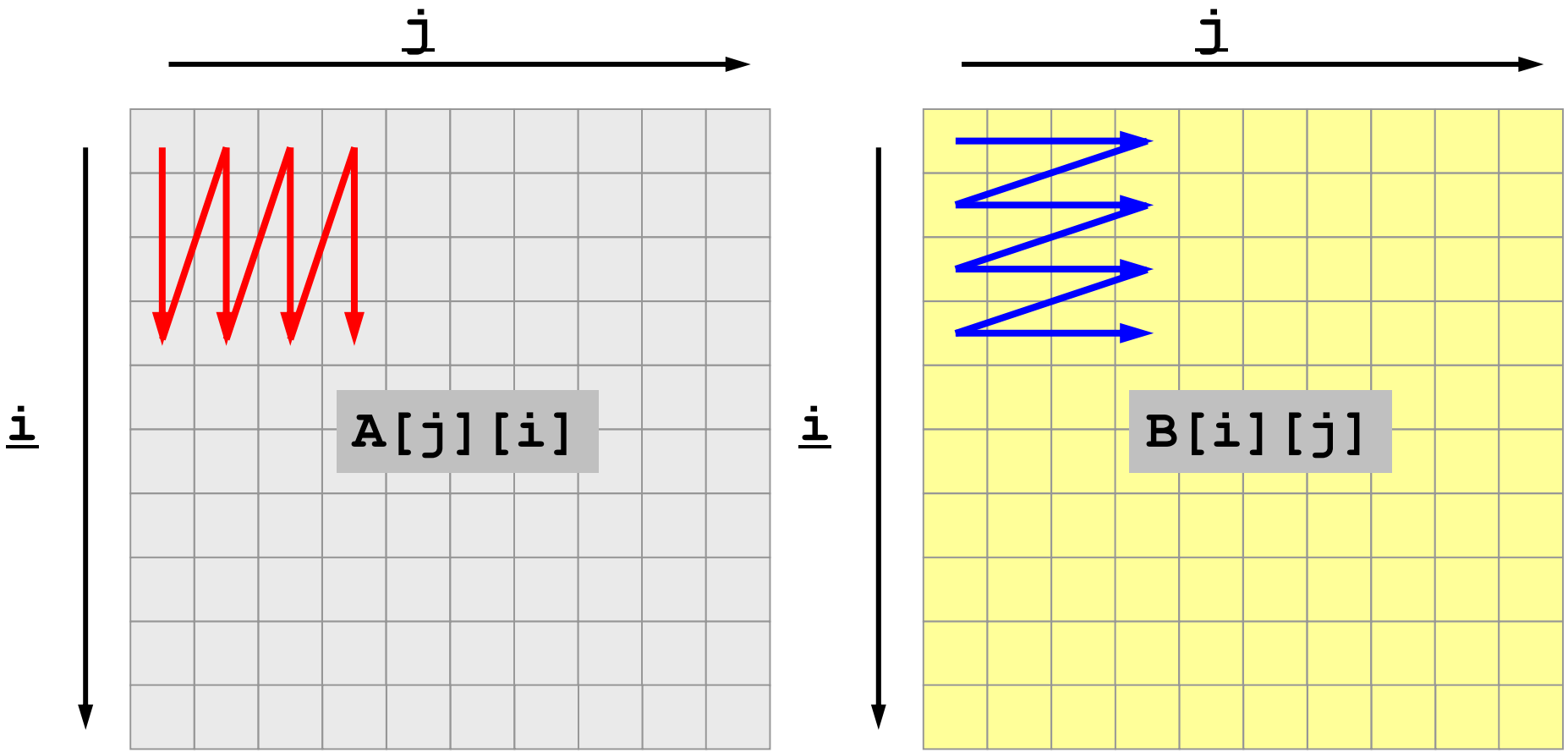
Effect of Blocking (4/7)

- Therefore, if A[0][0] is touched, A[0][0], A[0][1], A[0][2], A[0][3]are on cache.
- If A[1][9] is accessed, A[1][9], A[2][0], A[2][1], A[2][2] are also on cache,



Effect of Blocking (5/7)

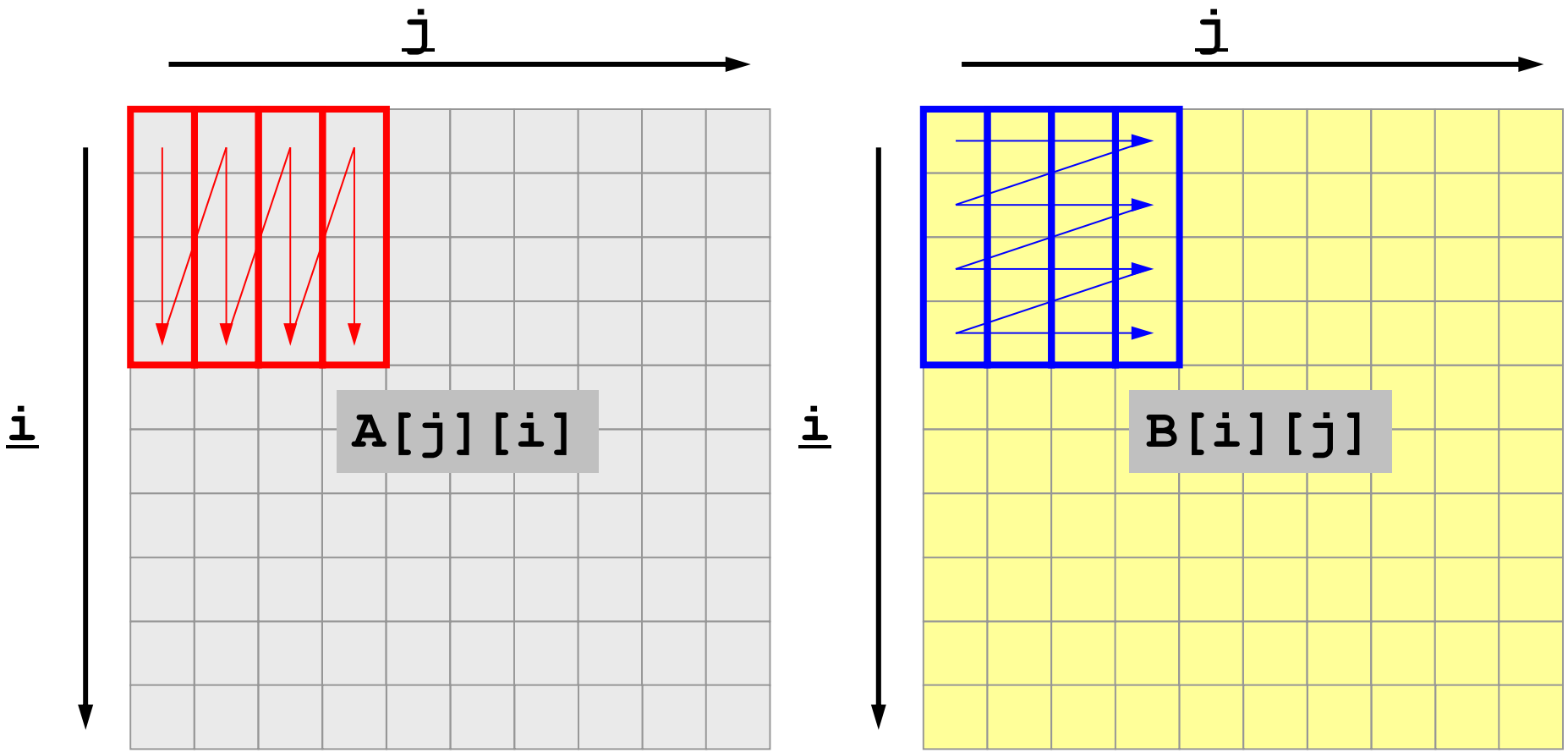
- Therefore, following “block-wise” access pattern is efficient.



Scalar

Effect of Blocking (6/7)

- □, □ : Components are on cache.



Scalar

Effect of Blocking (7/7)

- 2×2 Block

```
for (j=0;j<NN;j=++){
  for (i=0;i<NN;i=++){
    A[j][i]= A[j][i] + B[i][j];
  }
}


for (j=0;j<NN-1;j=+2){
  for (i=0;i<NN-1;i=+2){
    A[j  ][i  ]= A[j  ][i  ] + B[i  ][j  ];
    A[j  ][i+1]= A[j  ][i+1] + B[i  ][j+1];
    A[j+1][i  ]= A[j+1][i  ] + B[i+1][j  ];
    A[j+1][i+1]= A[j+1][i+1] + B[i+1][j+1];
  }
}
```

Fujitsu FX10 (sec.)
FORTRAN
(-O1)

```
### N ###     500
BASIC    2.838309E-03
2x2      1.835505E-03

### N ###    1000
BASIC    1.167853E-02
2x2      9.495229E-03

...

### N ###    4000
BASIC    2.081746E-01
2x2      1.294938E-01

### N ###    4500
BASIC    3.203001E-01
2x2      2.335151E-01

### N ###    5000
BASIC    3.517879E-01
2x2      2.478577E-01
```

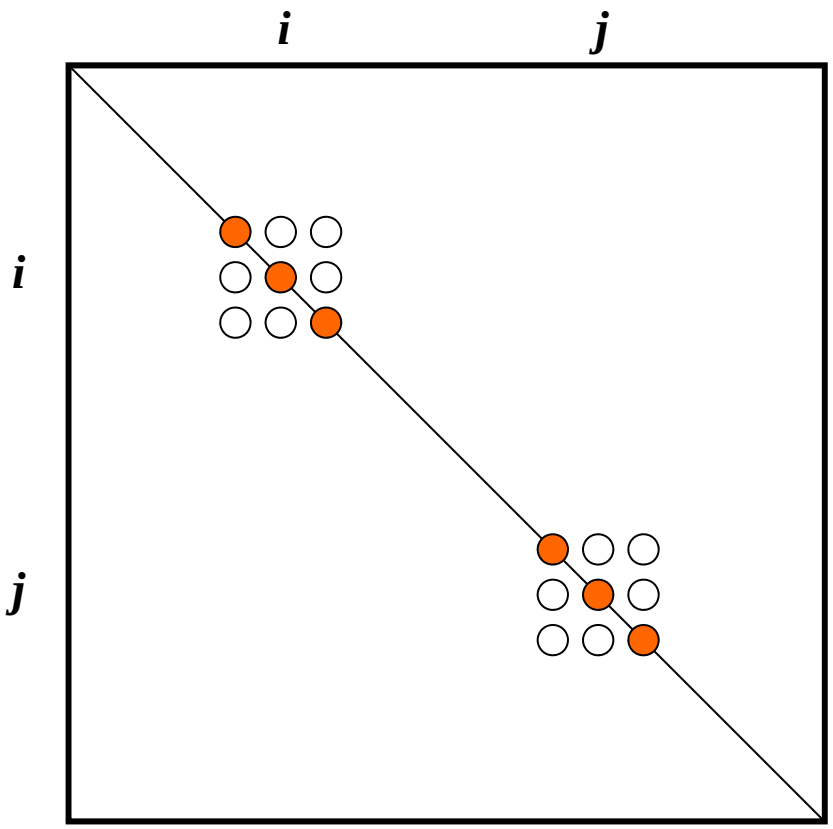
Summary: Tuning

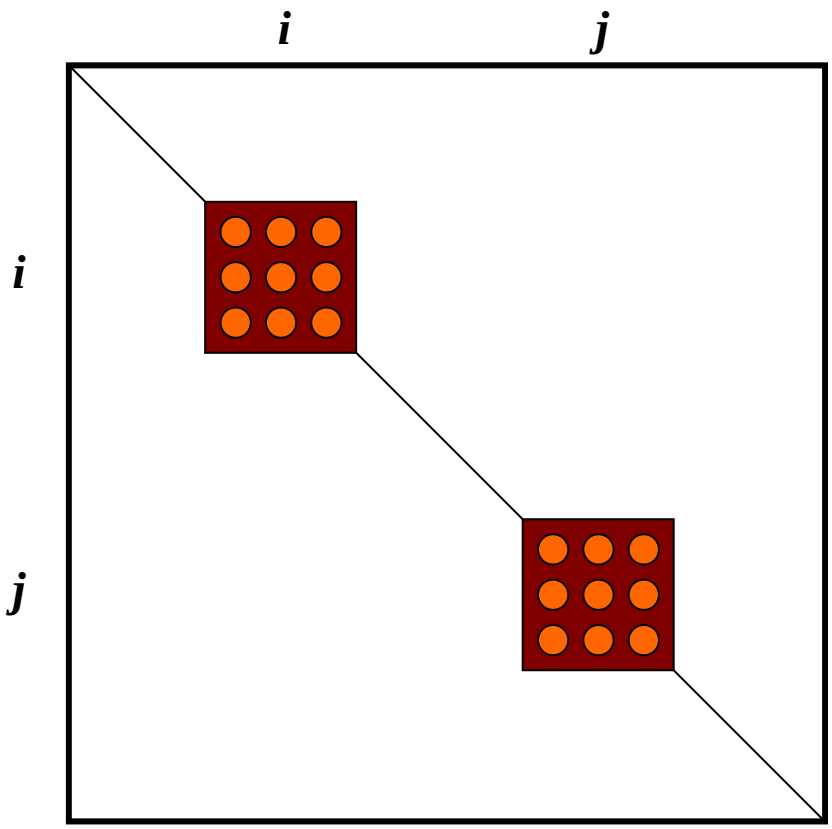
- Scalar Processor
- Dense Matrices

- Tuning of Sparse Matrix Operation is more difficult (on-going research topic)
  - Fundamental idea is same
  - Effective memory access is everything !

Storing 3x3 Block (3/3)

- Stabilization of Computation（計算の安定化）
  - Instead of division by diagonal components, full LU factorization of 3x3 Diagonal Block is applied.
  - Effective for ill-conditioned problems





- Formulation of 3D Element
- Governing Equations of 3D Elastic Problem
  - Galerkin Method
  - Element Matrices
- HW

- Running the Code
- Data Structure
- Overview of the Program
- Computational Issue
- **Visualization by ParaView**

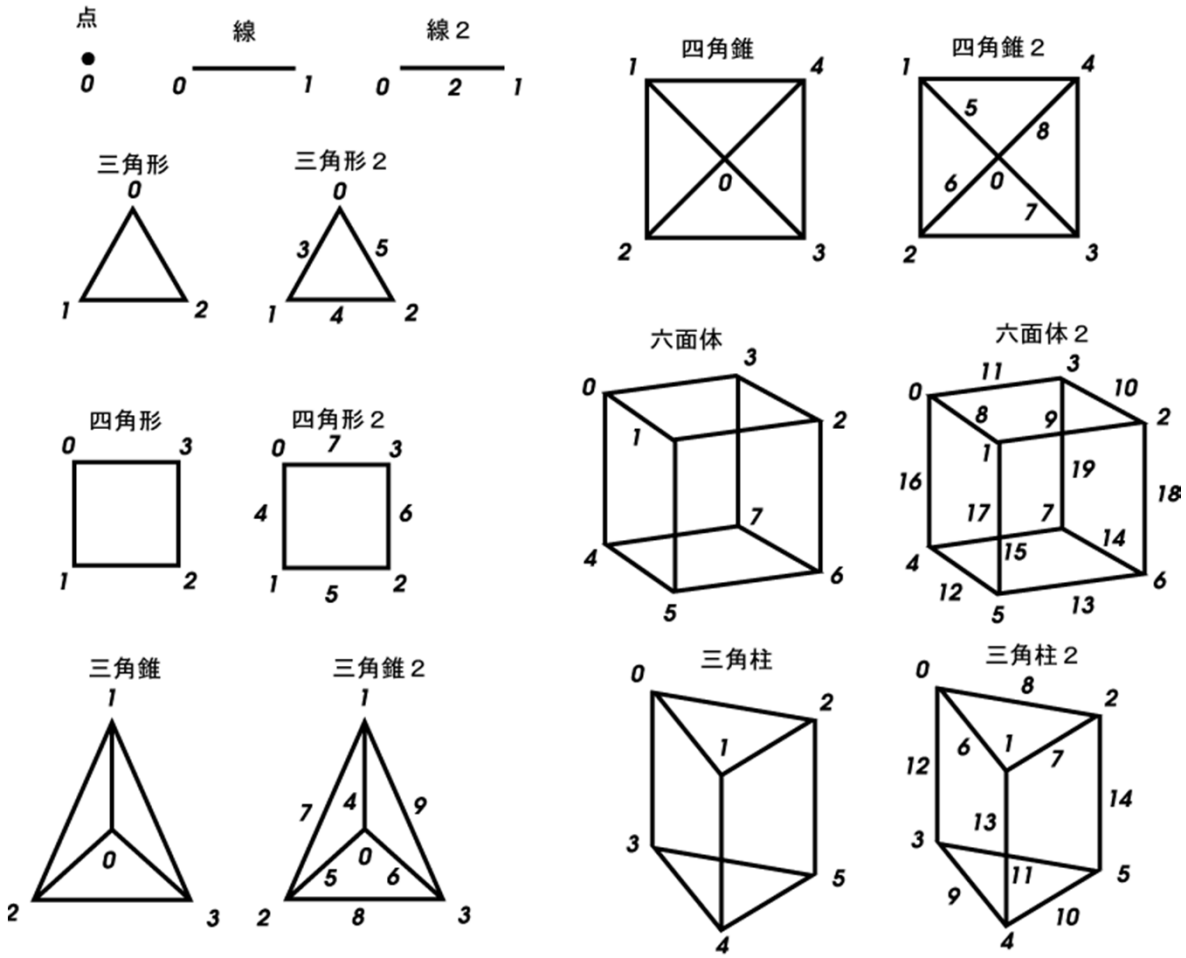
ParaView

- Opening files
- Displaying figures
- Saving image files

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

UCD Format (1/3)
Unstructured Cell Data

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



UCD Format (2/3)

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

UCD Format (3/3): Old Format

(全節点数) (全要素数) (各節点のデータ数) (各要素のデータ数) (モデルのデータ数)
(節点番号1) (X座標) (Y座標) (Z座標)
(節点番号2) (X座標) (Y座標) (Z座標)

.
.
.

(要素番号1) (材料番号) (要素の種類) (要素を構成する節点のつながり)
(要素番号2) (材料番号) (要素の種類) (要素を構成する節点のつながり)

.
.
.

(節点のデータ成分数) (成分1の構成数) (成分2の構成数) ･･･(各成分の構成数)
(節点データ成分1のラベル), (単位)
(節点データ成分2のラベル), (単位)

.
.
.

(各節点データ成分のラベル), (単位)
(節点番号1) (節点データ1) (節点データ2) ･････
(節点番号2) (節点データ1) (節点データ2) ･････

.
.
.

(要素のデータ成分数) (成分1の構成数) (成分2の構成数) ･･･(各成分の構成数)
(要素データ成分1のラベル), (単位)
(要素データ成分2のラベル), (単位)

.
.
.

(各要素データ成分のラベル), (単位)
(要素番号1) (要素データ1) (要素データ2) ･････
(要素番号2) (要素データ1) (要素データ2) ･････

.
.
.