科学技術計算のための
マルチコアプログラミング入門
第Ⅰ部：概要，対象アプリケーション，OpenMP

2011年6月30日・7月1日
中島研吾

本セミナーの背景

- マイクロプロセッサのマルチコア化，メニーコア化
  - 低消費電力，様々なプログラミングモデル
- OpenMP
  - 指示行（ディレクティヴ）を挿入するだけで手軽に「並列化」ができるため，広く使用されている
  - 様々な解説書
- データ依存性（data dependency）
  - メモリへの書き込みと参照が同時に発生
  - 並列化を実施するには，適切なデータの並べ替えを施す必要がある
  - このような対策はOpenMP向けの解説書でも詳しく取り上げられることは余りない：とても面倒くさい
- Hybrid 並列プログラミングモデル

謝　辞

- サイバネットシステム株式会社

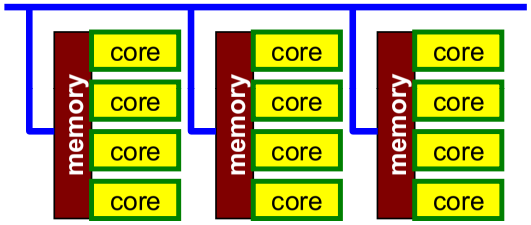
そもそもの動機：地球シミュレータ (ES)

http://www.es.jamstec.go.jp/

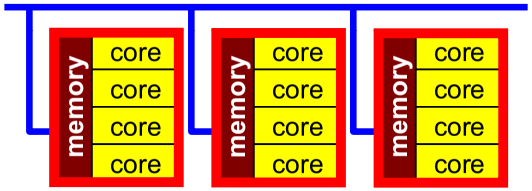
- 640×8= 5,120 Vector Processors
  - SMP Cluster-Type Architecture
  - 8 GFLOPS/PE
  - 64 GFLOPS/Node
  - 40 TFLOPS/ES
- 16 GB Memory/Node, 10 TB/ES
- 640×640 Crossbar Network
  - 12.3 GB/sec×2
- Memory BWTH with 32 GB/sec.
- 35.6 TFLOPS for LINPACK (2002-March)
- 26 TFLOPS for AFES (Climate Simulation)

Flat MPI vs. Hybrid

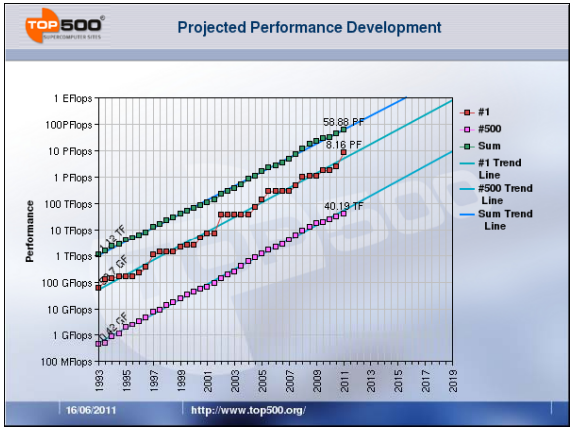
Flat-MPI：Each PE -> Independent



Hybrid：Hierarchal Structure



We are now in Post-Peta-Scale Era



- PFLOPS: Peta (=10^{15}) Floating OPerations per Sec.
- Exa-FLOPS (=10^{18}) will be attained in 2019

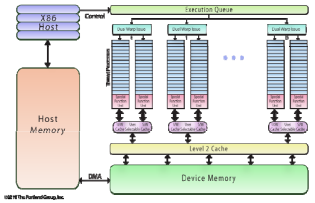
Key-Issues towards Appl./Algorithms on Exa-Scale Systems

Jack Dongarra (ORNL/U. Tennessee) at SIAM/PP10

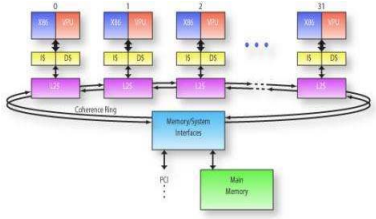
- Hybrid/Heterogeneous Architecture
  - Multicore + GPU
  - Multicore + Manycore (more intelligent)
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Tolerant
- Communication Reducing Algorithms

Heterogeneous Architecture by (CPU+GPU) or (CPU+Manycore) will be general in less than 5 years

NVIDIA Fermi



Intel Knights Ferry



CPU+Accelerator/Co-Processor (GPU, Manycore)

- 高いメモリーバンド幅
- GPU
  - プログラミング環境：CUDA，OpenCL
  - 一部のアプリケーションでは高効率：陽的FDM，BEM
- メニーコア（Manycores）
  - Intel Many Integrated Core Architecture（MIC）
    - GPUより賢い：軽いOS，コンパイラが使える
  - “Intel Knights Ferry”，“Knights Corner”

Hybrid並列プログラミングモデルは必須

- Message Passing
  - MPI
- Multi Threading
  - OpenMP

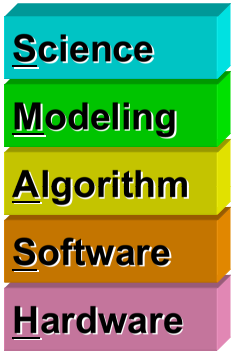
本セミナーの目的

- 「有限体積法から導かれる疎行列を対象としたICCG法」を題材とした，データ配置，reorderingなど，科学技術計算のためのマルチコアプログラミングにおいて重要なアルゴリズムについての講習
- 更に理解を深めるための，T2Kオープンスパコン（東大）を利用した実習

- 単一のアプリケーションに特化した内容であるが，基本的な考え方は様々な分野に適用可能である
  - 実はこの方法は意外に効果的である

本セミナーの目的（続き）

- 単一のアプリケーションに特化した内容であるが，基本的な考え方は様々な分野に適用可能である
  - 実はこの方法は意外に効果的である
- いわゆる「並列化講習会」とはだいぶ趣が異なる
- SMASH：「Science」無き科学技術計算はあり得ない！



スケジュール

- 6月30日（木）
  - 1030～1200　イントロダクション・T2Kオープンスパコン
  - 1300～1600　ICCG法によるポアソン方程式ソルバー
  - 1600～1730　OpenMP「超」入門＋実習
- 7月1日（金）
  - 1000～1200　オーダリング
  - 1300～1400　オーダリング（続き）
  - 1400～1530　並列化実装
  - 1530～1600　TCMallocを使ったCプログラム高速化の工夫
  - 1600～1730　実習，最新の話題

挙手によるアンケート

- 並列プログラミングの経験
  - MPI，OpenMP，その他
- プログラミング言語
  - C，FORTRAN，JAVA，その他
- 分野
  - 数学，数理科学
  - 理学・工学
  - 情報系，計算機科学
- 最後のアンケートにもご協力ください

ファイルの用意

Intelコンパイラを使えるようにする

```
>$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-intel
```

コピー，展開

```
>$ cd
>$ cp /home/t00000/multicore.tar .
>$ tar xvf multicore.tar
>$ cd multicore
```

以下のディレクトリが出来ていることを確認

```
      C    F   omp   stream
```

C,Fの下にはそれぞれ，以下のディレクトリがある：

```
      L1   L2   L3
```

これらを以降　<$L1>，<$L2>，<$L3>　と呼ぶ

また，<$CUR>/multicore/omp を <$omp>
<$CUR>/multicore/stream を <$stream>　と呼ぶ

- 背景
  - 有限体積法
  - 前処理付反復法
- ICCG法によるポアソン方程式法ソルバーについて
  - 実行方法
    - データ構造
  - プログラムの説明
    - 初期化
    - 係数マトリクス生成
    - ICCG法
- OpenMP「超」入門
- T2K（東大）による実習

本セミナーの目的より

- 「有限体積法から導かれる疎行列を対象としたICCG法」を題材とした，データ配置，reorderingなど，科学技術計算のためのマルチコアプログラミングにおいて重要なアルゴリズムについての講習

- 有限体積法
- 疎行列
- ICCG法

対象とするアプリケーションの概要

- 支配方程式：三次元ポアソン方程式

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

- 有限体積法（Finite Volume Method，**FVM**）による空間離散化
  - 任意形状の要素，要素中心で変数を定義。
  - 直接差分法（Direct Finite Difference Method）とも呼ばれる。
- 境界条件
  - ディリクレ，体積フラックス
- 反復法による連立一次方程式解法
  - 共役勾配法（CG）＋前処理

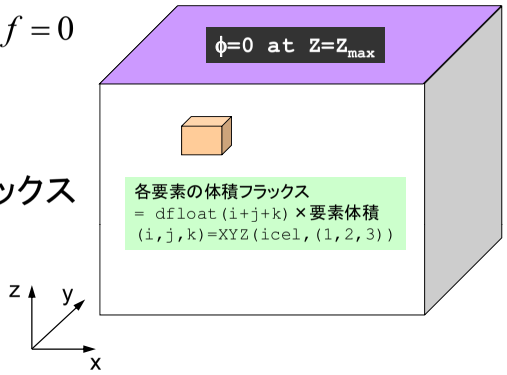
解いている問題：三次元ポアソン方程式
変数：要素中心で定義

ポアソン方程式

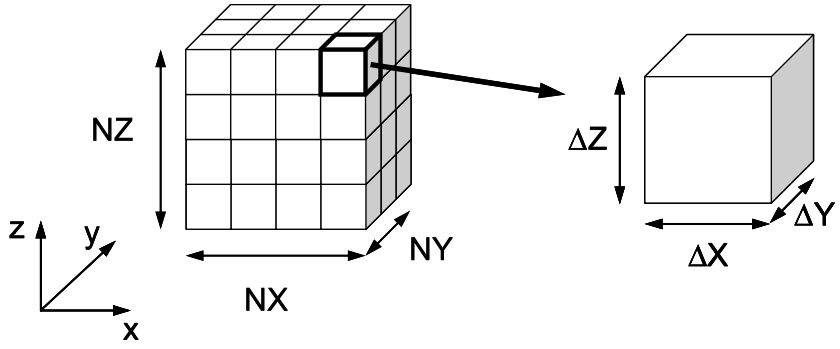
$$\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$$

境界条件

- 各要素で体積フラックス
- Z=Z_{max}面でϕ=0



対象：規則正しい三次元差分格子
半非構造的に扱う

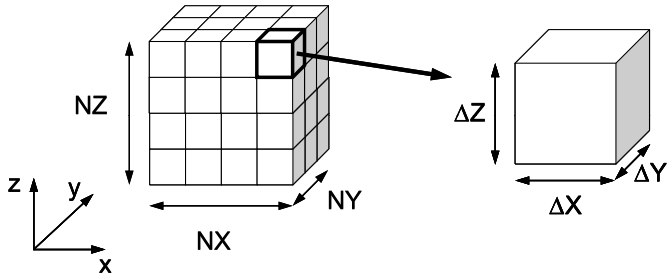


体積フラックスfの内容 $\frac{\partial^2\phi}{\partial x^2}+\frac{\partial^2\phi}{\partial y^2}+\frac{\partial^2\phi}{\partial z^2}+f=0$

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

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

XYZ(icel, k)(k=1,2,3)は
X，Y，Z方向の差分格子のインデックス
各メッシュがX，Y，Z方向の何番目に
あるかを示している。



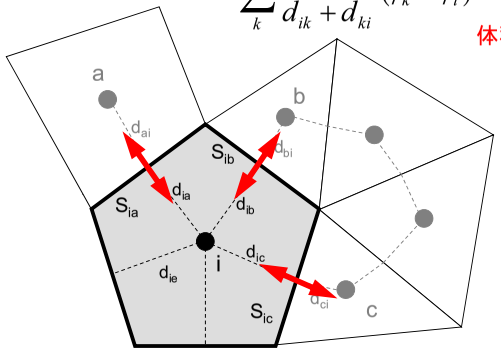
有限体積法
Finite Volume Method（FVM）

面を通過するフラックスの保存に着目

隣接要素との拡散

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

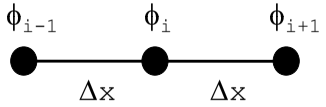
体積フラックス



V_i ：要素体積
S ：表面面積
d_{ij} ：要素中心から表面までの距離
Q ：体積フラックス

一次元ポアソン方程式：中央差分

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



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

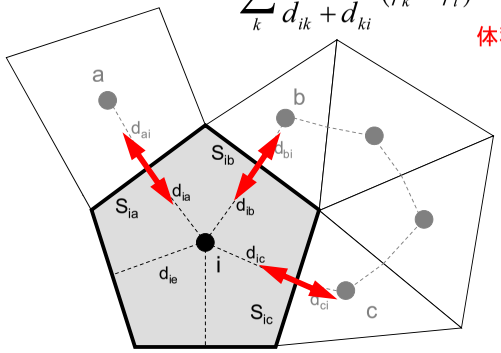
有限体積法
Finite Volume Method（FVM）

面を通過するフラックスの保存に着目

隣接要素との拡散

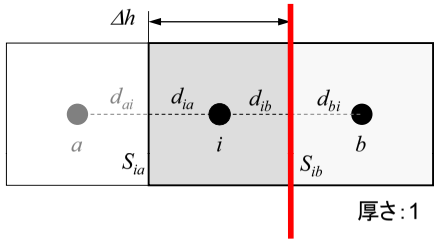
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

体積フラックス



V_i ：要素体積
S ：表面面積
d_{ij} ：要素中心から表面までの距離
Q ：体積フラックス

一次元差分法との比較(1/3)



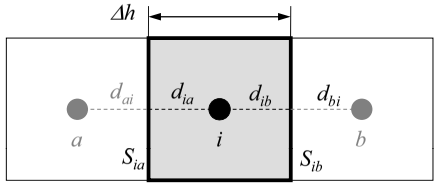
一辺の長さΔhの正方形メッシュ
接触面積: $S_{ik}=\Delta h$
要素体積: $V_i=\Delta h^2$
接触面までの距離: $d_{ij}=\Delta h/2$

この面を通過するフラックス:Qs_{ib}

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

フーリエの法則
面を通過するフラックス
＝ー(ポテンシャル勾配)

一次元差分法との比較(2/3)



一辺の長さΔhの正方形メッシュ
接触面積: $S_{ik}=\Delta h$
要素体積: $V_i=\Delta h^2$
接触面までの距離: $d_{ij}=\Delta h/2$

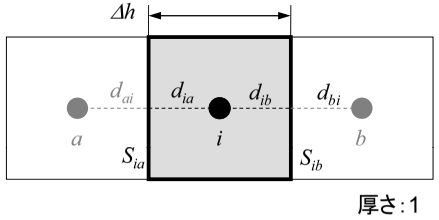
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

両辺をV_iで割る:

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

この部分に注目すると

一次元差分法との比較(3/3)



一辺の長さΔhの正方形メッシュ
接触面積: $S_{ik}=\Delta h$
要素体積: $V_i=\Delta h^2$
接触面までの距離: $d_{ij}=\Delta h/2$

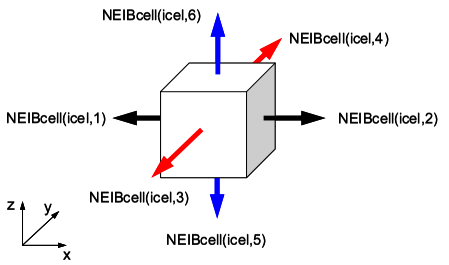
$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)=\frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i)$$

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

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

要素iについて成立
連立一次方程式

三次元では・・・



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

整理すると：連立一次方程式

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

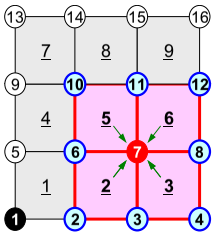
$$\sum_k \frac{S_{icel-k}}{d_{icel-k}}\left(\phi_k-\phi_{icel}\right)=-f_{icel}V_i$$

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

対角項　　非対角項

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

有限要素法の係数マトリクス
ゼロが多い：疎行列



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

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

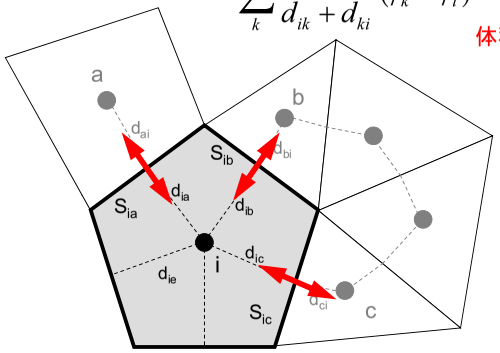
FVMの係数行列も疎行列

面を通過するフラックスの保存に着目
周囲の要素とのみ関係がある

隣接要素との拡散

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

体積フラックス



V_i ：要素体積
S ：表面面積
d_{ij} ：要素中心から表面までの距離
Q ：体積フラックス

疎行列：0が多い

- $A(i,j)$のように正方行列の全成分を記憶することは疎行列では非効率的
  - 「密」行列向け

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

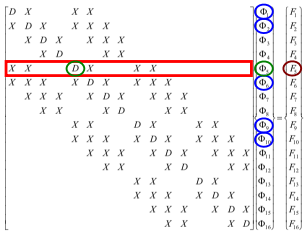
- 有限体積法：非零非対角成分の数は高々「数百」規模
  - 例えば未知数が10^8個あるとすると記憶容量（ワード数）は
    - 正方行列：$O(10^{16})$
    - 非零非対角成分数：$O(10^{10})$
- 非零成分のみ記憶するのが効率的

行列ベクトル積への適用

（非零）非対角成分のみを格納，疎行列向け方法
Compressed Row Storage (CRS)

```
Diag (i)  対角成分(実数, i=1,N)
Index(i)  非対角成分数に関する一次元配列(通し番号)
          (整数, i=0,N)
Item(k)   非対角成分の要素(列)番号
          (整数, k=1, index(N))
AMat(k)   非対角成分
          (実数, 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
```



行列ベクトル積への適用

（非零）非対角成分のみを格納，疎行列向け方法
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]];
      }
}
```


行列ベクトル積：密行列⇒とても簡単

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

```
{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)

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

Compressed Row Storage (CRS)

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

NODE_tot= 8

対角成分
D(1)= 1.1
D(2)= 3.6
D(3)= 5.7
D(4)= 9.8
D(5)= 11.5
D(6)= 12.4
D(7)= 23.1
D(8)= 51.3

Compressed Row Storage (CRS)

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

Compressed Row Storage (CRS)

						非対角成分数	
1	1.1 ①	2.4 ②	3.2 ⑤			2	index(0)= 0 index(1)= 2
2	3.6 ②	4.3 ①	2.5 ④	3.7 ⑥	9.1 ⑧	4	index(2)= 6
3	5.7 ③	1.5 ⑤	3.1 ⑦			2	index(3)= 8
4	9.8 ④	4.1 ②	2.5 ⑤	2.7 ⑥		3	index(4)= 11
5	11.5 ⑤	3.1 ①	9.5 ②	10.4 ③	4.3 ⑦	4	index(5)= 15
6	12.4 ⑥	6.5 ③	9.5 ⑦			2	index(6)= 17
7	23.1 ⑦	6.4 ②	2.5 ③	1.4 ⑥	13.1 ⑧	4	index(7)= 21
8	51.3 ⑧	9.5 ②	1.3 ③	9.6 ④	3.1 ⑥	4	index(8)= 25

NPLU= 25
(=index(N))

index(i-1)+1~index(i)番目がi行目の非対角成分

Compressed Row Storage (CRS)

						非対角成分数	
1	1.1 ①	2.4 ②,1	3.2 ⑤,2			2	index(0)= 0 index(1)= 2
2	3.6 ②	4.3 ①,3	2.5 ④,4	3.7 ⑥,5	9.1 ⑧,6	4	index(2)= 6
3	5.7 ③	1.5 ⑤,7	3.1 ⑦,8			2	index(3)= 8
4	9.8 ④	4.1 ②,9	2.5 ⑤,10	2.7 ⑥,11		3	index(4)= 11
5	11.5 ⑤	3.1 ①,12	9.5 ②,13	10.4 ③,14	4.3 ⑦,15	4	index(5)= 15
6	12.4 ⑥	6.5 ③,16	9.5 ⑦,17			2	index(6)= 17
7	23.1 ⑦	6.4 ②,18	2.5 ③,19	1.4 ⑥,20	13.1 ⑧,21	4	index(7)= 21
8	51.3 ⑧	9.5 ②,22	1.3 ③,23	9.6 ④,24	3.1 ⑥,25	4	index(8)= 25

NPLU= 25
(=index(N))

index(i-1)+1~index(i)番目がi行目の非対角成分

Compressed Row Storage (CRS)

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

例:

```
item( 7)= 5, AMAT( 7)= 1.5
item(19)= 3, AMAT(19)= 2.5
```


Compressed Row Storage (CRS)

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

```
D     (i)  対角成分(実数, i=1,N)
index(i)   非対角成分数に関する一次元配列
           (通し番号)(整数, i=0,N)
item(k)    非対角成分の要素(列)番号
           (整数, k=1, index(N))
AMAT(k)    非対角成分
           (実数, k=1, index(N))
```

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

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


疎行列:非零成分のみ記憶
⇒メモリへの負担大:間接参照
(差分, FEM, FVM)

```
{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
```


行列ベクトル積:密行列⇒とても簡単
メモリへの負担も小さい

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

```
{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
```

- 背景
  - 有限体積法
  - **前処理付反復法**
- ICCG法によるポアソン方程式法ソルバーについて
  - 実行方法
    - データ構造
  - プログラムの説明
    - 初期化
    - 係数マトリクス生成
    - ICCG法
- OpenMP「超」入門
- T2K（東大）による実習

科学技術計算における大規模線形方程式の解法

- 多くの科学技術計算は，最終的に大規模線形方程式 Ax=bを解くことに帰着される。
  - important, expensive
- アプリケーションに応じて様々な手法が提案されている
  - 疎行列（sparse），密行列（dense）
  - 直接法（direct），反復法（iterative）
- 密行列（dense）
  - グローバルな相互作用：BEM，スペクトル法，MO，MD（気液）
- 疎行列（sparse）
  - ローカルな相互作用：FEM，FDM，MD（固），高速多重極展開付BEM

直接法（Direct Method）

- Gaussの消去法，完全LU分解
  - 逆行列A^{-1}を直接求める
- 利点
  - 安定，幅広いアプリケーションに適用可能
    - Partial Pivoting
  - 疎行列，密行列いずれにも適用可能
- 欠点
  - 反復法よりもメモリ，計算時間を必要とする
    - 密行列の場合，$O(N^3)$の計算量
  - 大規模な計算向けではない
    - $O(N^2)$の記憶容量，$O(N^3)$の計算量

反復法（Iterative Method）

- 定常（stationary）法
  - 反復計算中，解ベクトル以外の変数は変化せず
  - SOR，Gauss-Seidel，Jacobiなど
  - 概して遅い
- 非定常（nonstationary）法
  - 拘束，最適化条件が加わる
  - Krylov部分空間（subspace）への写像を基底として使用するため，Krylov部分空間法とも呼ばれる
  - CG（Conjugate Gradient：共役勾配法）
  - BiCGSTAB（Bi-Conjugate Gradient Stabilized）
  - GMRES（Generalized Minimal Residual）

反復法（Iterative Method）（続き）

- 利点
  - 直接法と比較して，メモリ使用量，計算量が少ない。
  - 並列計算には適している。
- 欠点
  - 収束性が，アプリケーション，境界条件の影響を受けやすい。
  - 前処理（preconditioning）が重要。

代表的な「非定常」反復法：共役勾配法

- Conjugate Gradient法，略して「CG」法
  - 最も代表的な「非定常」反復法
- 対称正定値行列（Symmetric Positive Definite：SPD）向け
  - 任意のベクトル$\{x\}$に対して$\{x\}^T[A]\{x\}>0$
  - 全対角成分>0，全固有値>0，全部分行列式>0と同値
  - 熱伝導，弾性，ねじり・・・本コードの場合もSPD
- アルゴリズム
  - 最急降下法（Steepest Descent Method）の変種
  - **$x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}$**
    - **$x^{(i)}$：反復解，$p^{(i)}$：探索ベクトル，α_i：定数）**
  - 厳密解をyとするとき $\{x-y\}^T[A]\{x-y\}$を最小とするような$\{x\}$を求める。
  - 詳細は参考文献〔長谷川ら〕参照

共役勾配法のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$ ：ベクトル
α_i ：スカラー

共役勾配法のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$ ：ベクトル
α_i ：スカラー

共役勾配法のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$:ベクトル
α_i :スカラー

共役勾配法のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$:ベクトル
α_i :スカラー

前処理（preconditioning）とは?

- 反復法の収束は係数行列の固有値分布に依存
  - 固有値分布が少なく，かつ1に近いほど収束が早い（単位行列）
  - 条件数（condition number）（対称正定）＝最大最小固有値比
    - 条件数が1に近いほど収束しやすい
- もとの係数行列 **[A]** に良く似た前処理行列 **[M]** を適用することによって固有値分布を改善する。
  - 前処理行列 **[M]** によって元の方程式 $[A]\{x\}=\{b\}$ を $[A']\{x'\}=\{b'\}$ へと変換する。ここで $[A']=[M]^{-1}[A]$, $\{b'\}=[M]^{-1}\{b\}$ である。
  - $[A']=[M]^{-1}[A]$ が単位行列に近ければ良いということになる。
  - $[A']=[A][M]^{-1}$ のように右からかけることもある。
- 「前処理」は密行列，疎行列ともに使用するが，普通は疎行列を対象にすることが多い。

前処理付共役勾配法

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
```

実際にやるべき計算は:

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

「近似逆行列」の計算が必要:

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

究極の前処理:本当の逆行列

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

対角スケーリング:簡単＝弱い

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

対角スケーリング，点ヤコビ前処理

- 前処理行列として，もとの行列の対角成分のみを取り出した行列を前処理行列 [M] とする。
  - 対角スケーリング，点ヤコビ（point-Jacobi）前処理

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

- **solve [M]z$^{(i-1)}$= r$^{(i-1)}$**という場合に逆行列を簡単に求めることができる。
- 簡単な問題では収束する。

ILU(0), IC(0)

- 最もよく使用されている前処理（疎行列用）
  - 不完全LU分解
    - Incomplete LU Factorization
  - 不完全コレスキー分解
    - Incomplete Cholesky Factorization（対称行列）
- 不完全な直接法
  - もとの行列が疎でも，逆行列は疎とは限らない。
  - fill-in
  - もとの行列と同じ非ゼロパターン（fill-in無し）を持っているのがILU（0），IC（0）

ファイル類ありか FORTRANだけです

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

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


LU分解法：完全LU分解法

- 直接法の一種
  - 逆行列を直接求める手法
  - 「逆行列」に相当するものを保存しておけるので，右辺が変わったときに計算時間を節約できる
  - 逆行列を求める際にFill-in（もとの行列では0であったところに値が入る）が生じる
- LU factorization

「不」完全LU分解法

- ILU factorization
  - Incomplete LU factorization
- Fill-inの発生を制限して，前処理に使う手法
  - 不完全な逆行列，少し弱い直接法
  - Fill-inを許さないとき：ILU(0)

LU分解による連立一次方程式の解法

Aがn×n行列のとき、Aを次式のように表すことを
（あるいは、そのようなLとUそのものを）AのLU分解という.

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

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

L : Lower triangular part of matrix A
U : Upper triangular part of matrix A

連立一次方程式の行列表現

n元の連立一次方程式の一般形

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

行列表現

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

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

LU分解を用いたAx=bの解法

1 $\mathbf{A} = \mathbf{LU}$ となるAのLU分解LとUを求める.

2 $\mathbf{Ly} = \mathbf{b}$ の解yを求める.（簡単！）

3 $\mathbf{Ux} = \mathbf{y}$ の解xを求める.（簡単！）

このxが $\mathbf{Ax} = \mathbf{b}$ の解となる

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

Ly=bの解法：前進代入

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

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

芋づる式に（one after another）解が求まる.

Ux=yの解法：後退代入

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

$$\begin{aligned} & u_{nn} x_n = y_n \\ & u_{n-1,n-1} x_{n-1} + u_{n-1,n} x_n = y_{n-1} \\ & \vdots \\ & u_{11} x_1 + u_{12} x_2 + \cdots + u_{1n} x_n = y_1 \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} & x_n = y_n / u_{nn} \\ & x_{n-1} = (y_{n-1} - u_{n-1,n} x_n) / u_{n-1,n-1} \\ & \vdots \\ & x_1 = \left(y_1 - \sum_{i=2}^{n} u_{1j} x_j \right) / u_{11} \end{aligned}$$

芋づる式に（one after another）解が求まる.

LU分解の求め方

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

数値例

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

第1行 ➡ $1 = u_{11}, 2 = u_{12}, 3 = u_{13}, 4 = u_{14}$

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

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

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

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

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

数値例(続き)

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

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

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

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

第4行(第4列) ➡ $1=l_{41}u_{14}+l_{42}u_{24}+l_{43}u_{34}+u_{44}\Rightarrow u_{44}=2$

1行、1列、2行、2列、・・・の順に求める式を作っていく.

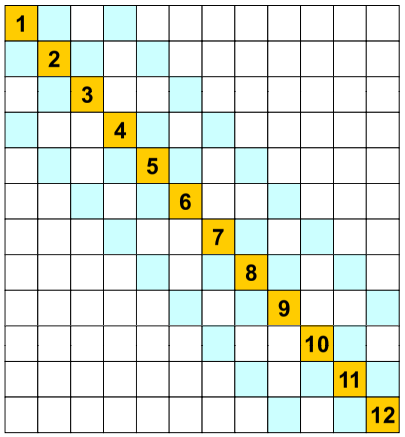
数値例(続き)

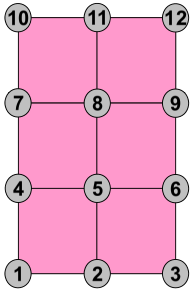
結局

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

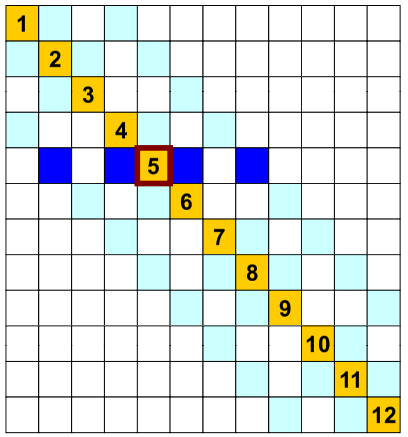
L **U**

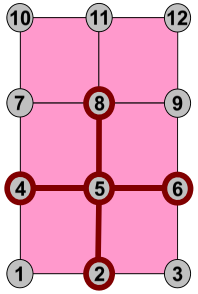
実例:5点差分





実例:5点差分



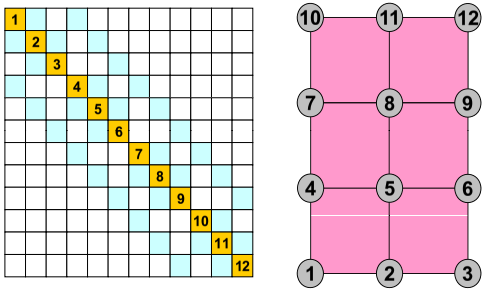


実例：係数マトリクス

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



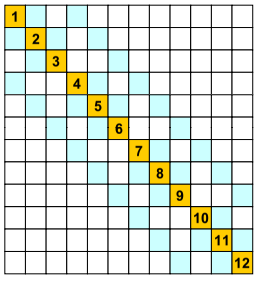
実例：解

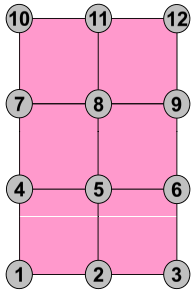
6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	6.00	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00
−1.00	0.00	0.00	6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00
0.00	−1.00	0.00	−1.00	6.00	−1.00	0.00	−1.00	0.00	0.00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

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

=

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





完全LU分解したマトリクス

./lu1 とタイプ

もとのマトリクス

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分解したマトリクス
[L][U]同時に表示
[L]対角成分（=1）省略
（fill-inが生じている。もともと0だった成分が非ゼロになっている）

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

不完全LU分解したマトリクス（fill-in無し）

./lu2 とタイプ

不完全LU分解したマトリクス（fill-in無し）
[L][U]同時に表示
[L]対角成分（=1）省略

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

完全LU分解したマトリクス
[L][U]同時に表示
[L]対角成分（=1）省略
（fill-inが生じている。もともと0だった成分が非ゼロになっている）

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

解の比較：ちょっと違う

不完全LU分解

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.92
−0.17	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.75
0.00	−0.17	5.83	0.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	2.76
−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	3.79
0.00	−0.17	0.00	−0.17	5.66	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	4.46
0.00	0.00	−0.17	0.00	−0.18	5.65	0.00	0.00	−1.00	0.00	0.00	0.00	5.57
0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00	−1.00	0.00	0.00	6.66
0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00	0.00	−1.00	0.00	7.25
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65	0.00	0.00	−1.00	8.46
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	0.00	0.00	5.83	−1.00	0.00	9.66
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.17	5.65	−1.00	10.54
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.65	11.83

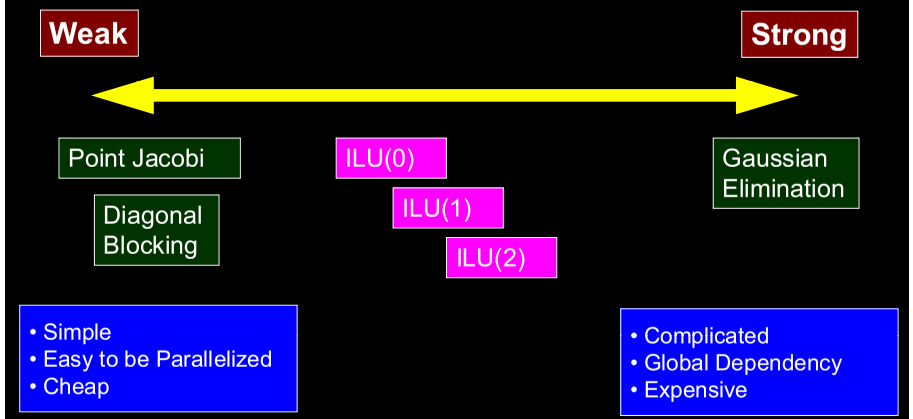
完全LU分解

6.00	−1.00	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	1.00
−0.17	5.83	−1.00	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	2.00
0.00	−0.17	5.83	−0.03	−0.17	−1.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00
−0.17	−0.03	0.00	5.83	−1.03	0.00	−1.00	0.00	0.00	0.00	0.00	0.00	4.00
0.00	−0.17	−0.03	−0.18	5.64	−1.03	−0.18	−1.00	0.00	0.00	0.00	0.00	5.00
0.00	0.00	−0.17	0.00	−0.18	5.64	−0.03	−0.18	−1.00	0.00	0.00	0.00	6.00
0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01	−1.00	0.00	0.00	7.00
0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03	−0.18	−1.00	0.00	8.00
0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63	−0.03	−0.18	−1.00	9.00
0.00	0.00	0.00	0.00	0.00	0.00	−0.17	−0.03	−0.01	5.82	−1.03	−0.01	10.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	−0.03	−0.18	5.63	−1.03	11.00
0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	−0.18	0.00	−0.18	5.63	12.00

ILU(0), IC(0) 前処理

- Fill-inを全く考慮しない「不完全な」分解
  - 記憶容量，計算量削減
- これを解くと「不完全な」解が得られるが，本来の解とそれほどずれているわけではない
  - 問題に依存する

前処理の分類：Trade-off



- 背景
  - 有限体積法
  - 前処理付反復法
- **ICCG法によるポアソン方程式法ソルバーについて**
  - **実行方法**
    - **データ構造**
  - プログラムの説明
    - 初期化
    - 係数マトリクス生成
    - ICCG法
- OpenMP「超」入門
- T2K（東大）による実習

対象とするアプリケーションの概要

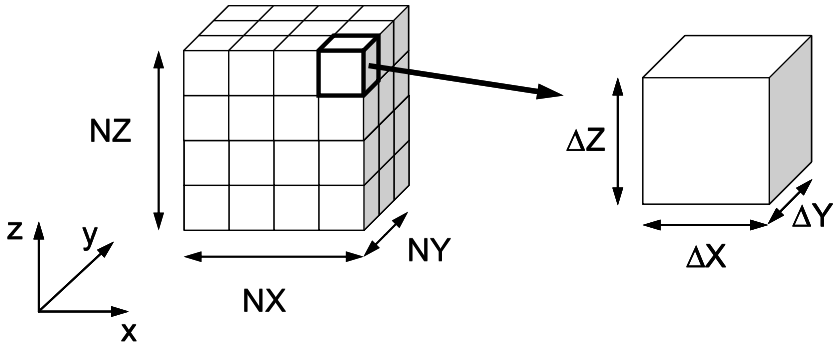
- 支配方程式：三次元ポアソン方程式

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

- 有限体積法（Finite Volume Method，**FVM**）による空間離散化
  - 任意形状の要素，要素中心で変数を定義。
  - 直接差分法（Direct Finite Difference Method）とも呼ばれる。
- 境界条件
  - ディリクレ，体積フラックス
- 反復法による連立一次方程式解法
  - 共役勾配法（CG）＋前処理

対象：規則正しい三次元差分格子

半非構造的に扱う



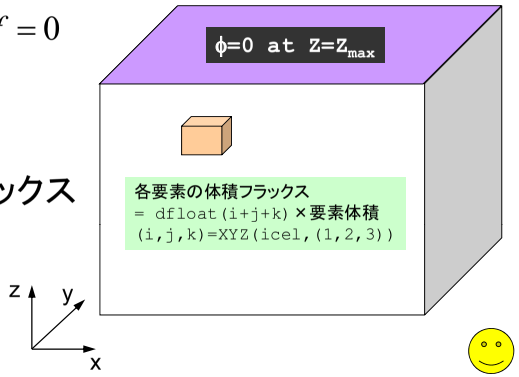
解いている問題：三次元ポアソン方程式
変数：要素中心で定義

ポアソン方程式

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

境界条件

- 各要素で体積フラックス
- Z=Z_{max}面でϕ=0



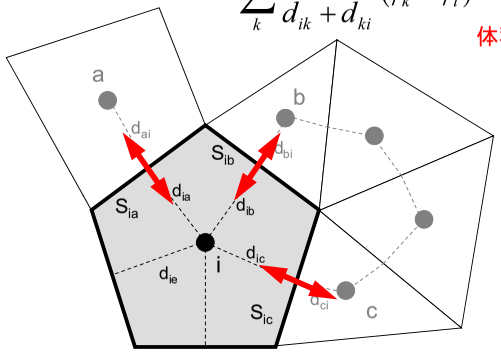
有限体積法
Finite Volume Method（FVM）

面を通過するフラックスの保存に着目

隣接要素との拡散

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

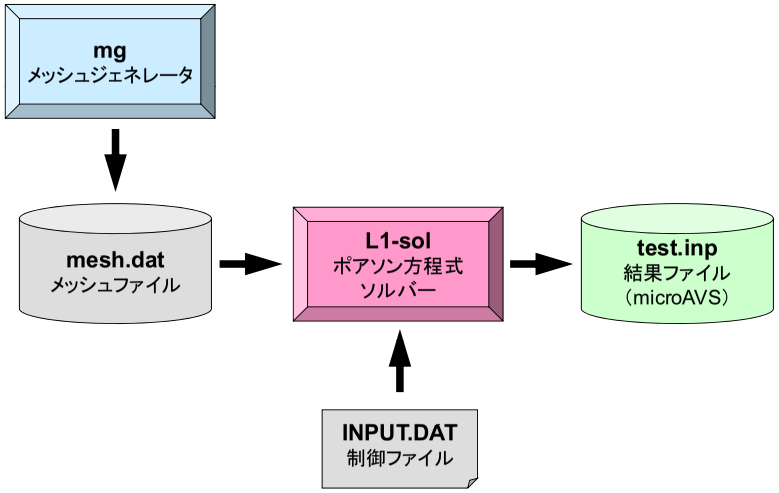
体積フラックス



V_i ：要素体積
S ：表面面積
d_{ij} ：要素中心から表面までの距離
Q ：体積フラックス

プログラムの実行

プログラム，必要なファイル等，実行ディレクトリ：<$L1>/run



プログラムの実行

コンパイル

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

$> ifort -O mg.f -o mg (or icc -O mg.c -o mg)
$> ls mg
      mg

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

メッシュジェネレータ：**mg**

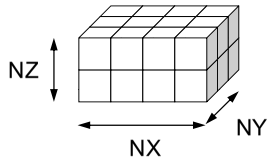
ポアソン方程式ソルバー：**L1-sol**

プログラムの実行

メッシュ生成

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

下図のNX，NY，NZを入力すると，
「mesh.dat」が生成される



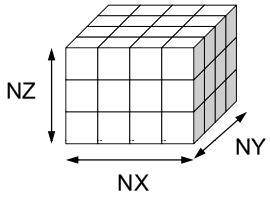
mesh.dat（1/5）

```
  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
```



X,Y,Z方向の要素数

```
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)

```
  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
```

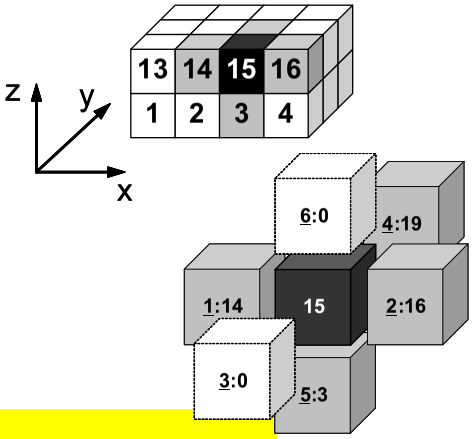
要素数:NX×NY×NZ

```
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
```

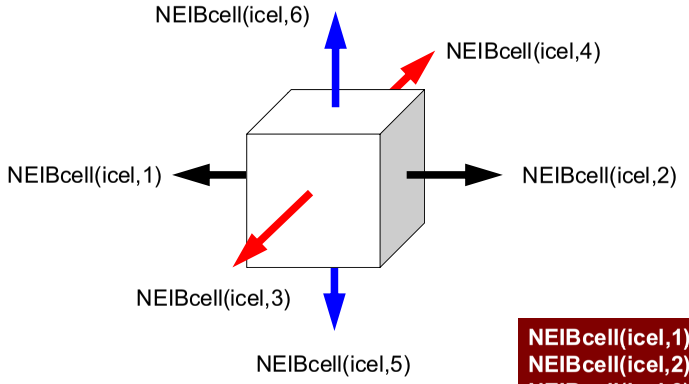
隣接要素:NEIBcell(i,k)




```
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
```

1項目目は通し番号です(読み飛ばし)

NEIBcell:隣接している要素番号
境界面の場合は0

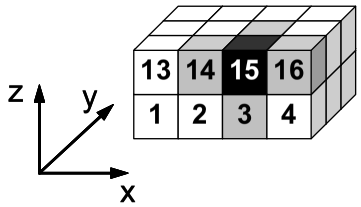


NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

mesh.dat(5/5)

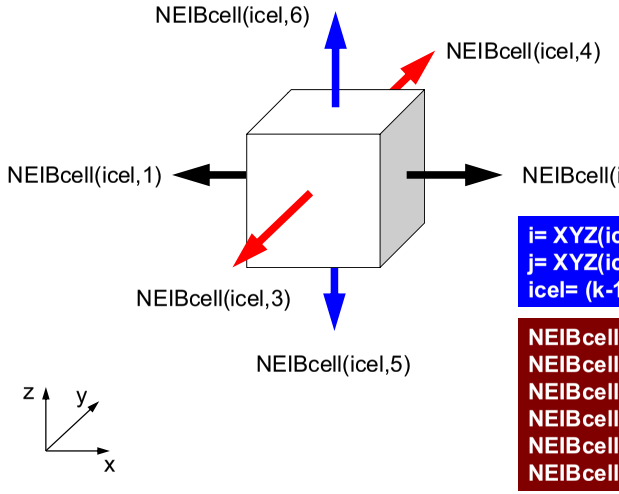
```
  4   3   2
 24
  1   0   2   0   5   0  13   1   1   1
  2   1   3   0   6   0  14   2   1   1
  3   2   4   0   7   0  15   3   1   1
  4   3   0   0   8   0  16   4   1   1
  5   0   6   1   9   0  17   1   2   1
  6   5   7   2  10   0  18   2   2   1
  7   6   8   3  11   0  19   3   2   1
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

X,Y,Z方向の位置:XYZ(i,j)




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

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


NEIBcell:隣接している要素番号
境界面の場合は0



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

プログラムの実行

制御データ「<$L1>/run/INPUT.DAT」の作成

```
1                                   METHOD 1:2
1.00e-00 1.00e-00 1.00e-00          DX/DY/DZ
1.0e-08                             EPSICCG
```

- METHOD
  - 前処理行列の作成方法:次ページ
- DX, DY, DZ
  - 各要素のX,Y,Z方向辺長さ
- EPSICCG
  - ICCG法の収束判定値

プログラムの実行

制御データ「<$L1>/run/INPUT.DAT」の作成

```
1                                   METHOD 1:2
1.00e-00 1.00e-00 1.00e-00          DX/DY/DZ
1.0e-08                             EPSICCG
```

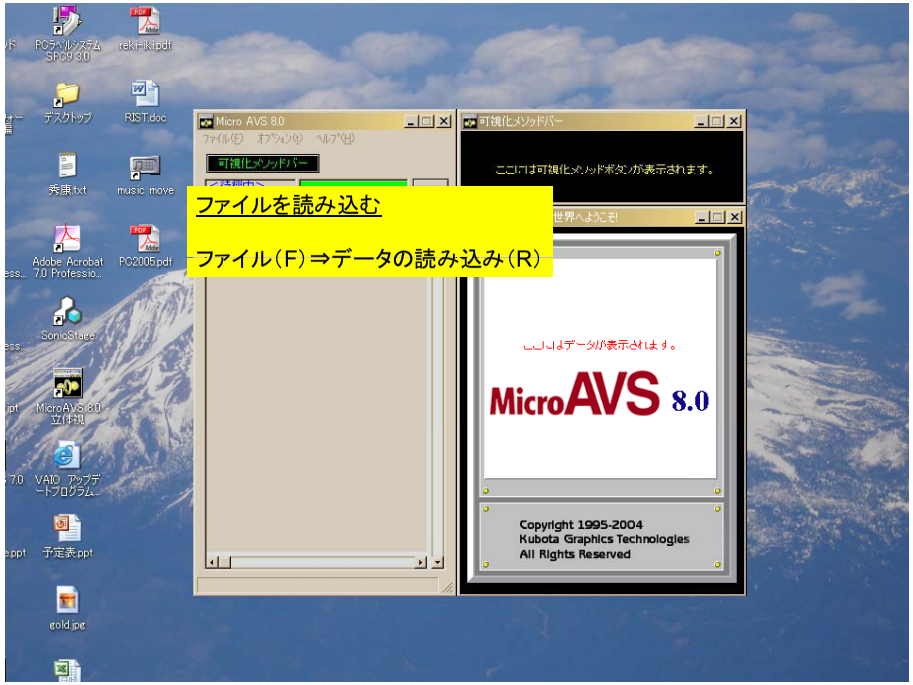
- METHOD
  - 前処理行列の作成方法:次ページ
- DX, DY, DZ
  - 各要素のX,Y,Z方向辺長さ
- EPSICCG
  - ICCG法の収束判定値

前処理手法の選択

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

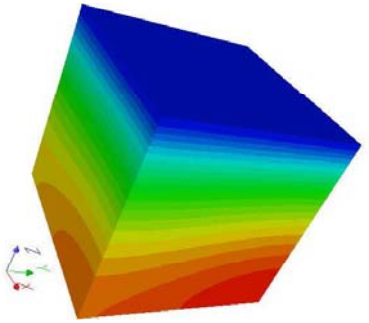
- METHOD=1　不完全修正コレスキー分解（非対角項保存）
- METHOD=2　不完全修正コレスキー分解
- METHOD=3　対角スケーリング（点ヤコビ）

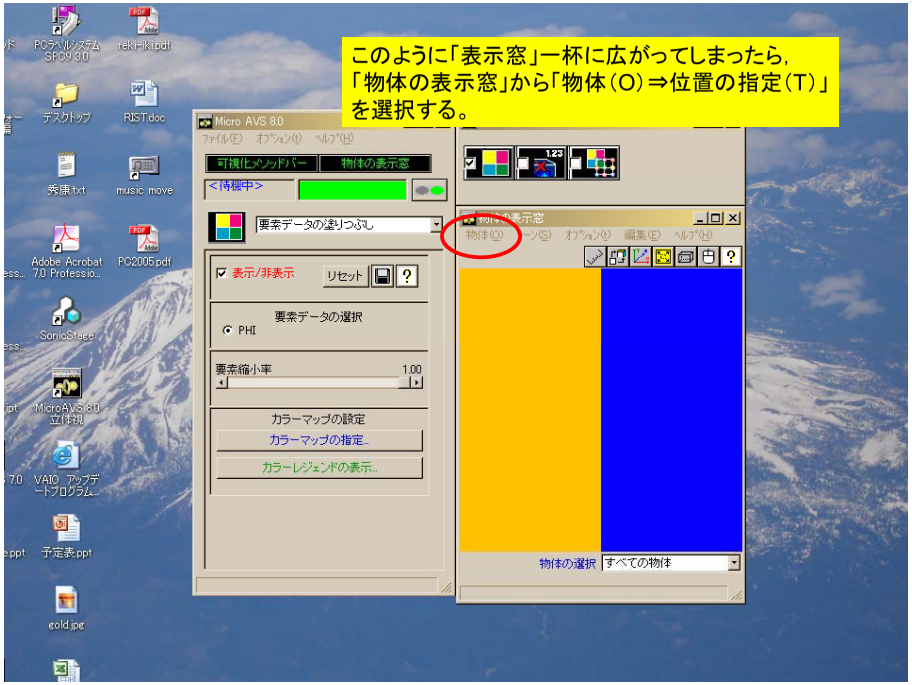
- NX=NY=NZ=32として，METHOD=1,2,3について計算してみよ　！



プログラムの実行
計算実行，ポスト処理

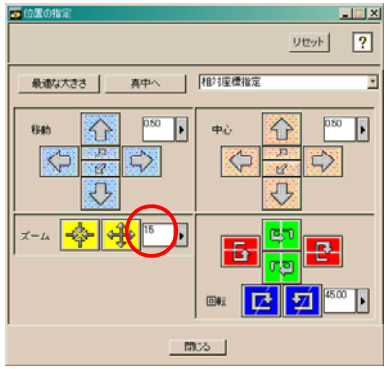
```
$> cd <$L1>/run
$> ./L1-sol

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





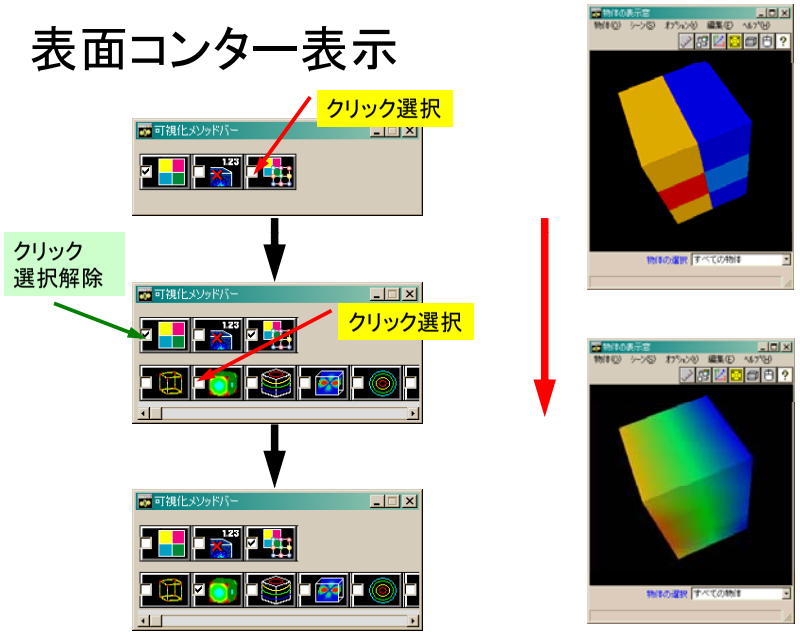
このような「位置の指定」ウィンドウが表示される

「最適な大きさ」,「ズーム（縮小）」などのボタンを使用

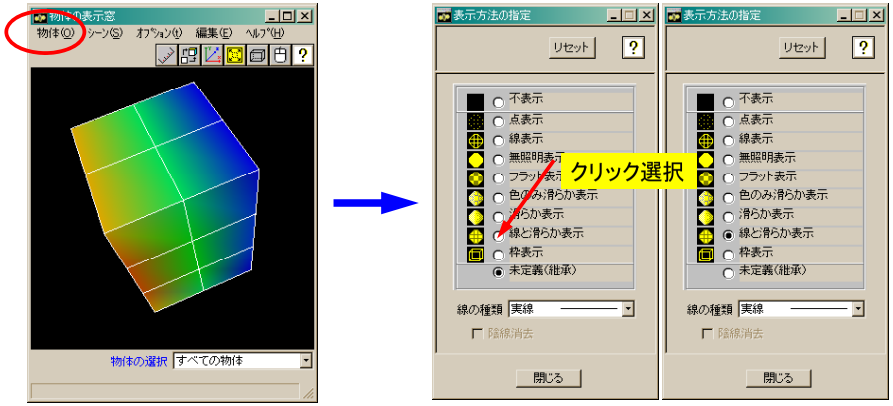


「最適な大きさ」,「ズーム（縮小）」などのボタンを使用

表面コンター表示



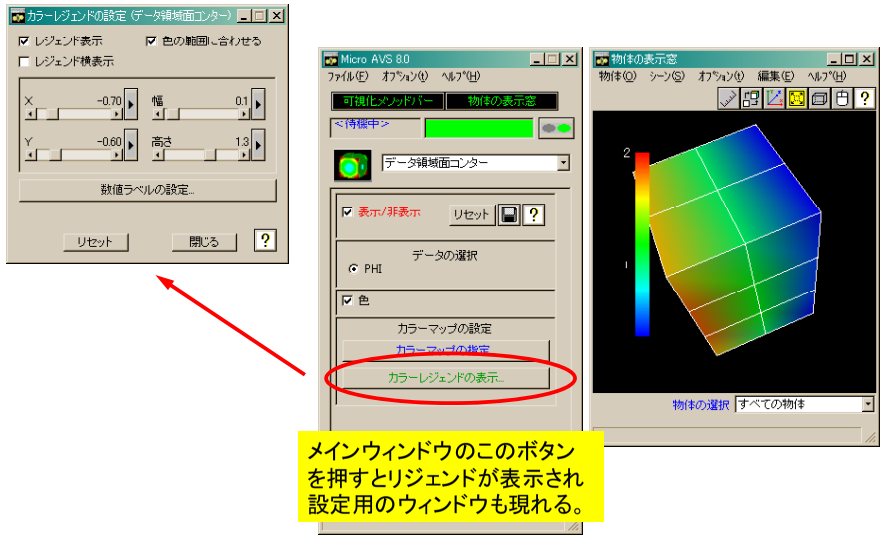
メッシュ表示



「物体の表示窓」から「物体（O）⇒表示方法の指定（M）」を選択する。

出てきたウインドウの「線と滑らか表示」をクリックする。

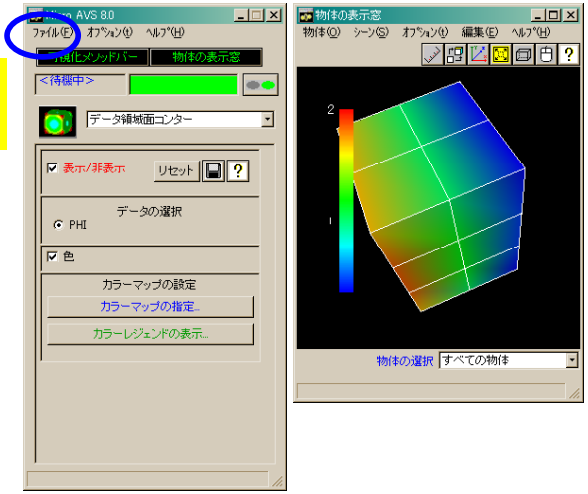
リジェンド表示



メインウィンドウのこのボタンを押すとリジェンドが表示され設定用のウィンドウも現れる。

ファイル保存

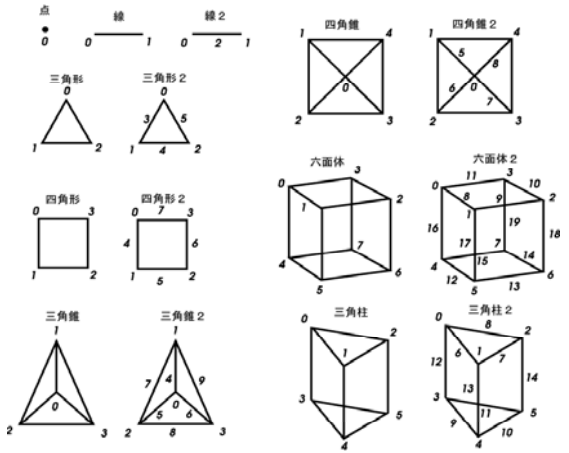
「ファイル(F)⇒データの書き込み(W)」で様々なフォーマットでの保存可能



UCDフォーマットについて(1/3)

MicroAVSで扱うことのできる要素形状

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フォーマットについて(2/3)

書式の概要

- ファイルの拡張子
  - データファイルの拡張子は “.inp”
- 書式
  - アスキーファイル
  - 時系列(複数ステップ)に対応したものが標準フォーマット
  - MicroAVS Ver.6.0(現在はVer.10.0)まで使用していた旧フォーマット(単一ステップデータのための書式)も読み込むことは可能
    - 実は一部これを使用している

UCDフォーマットについて(3/3)

書式の概要

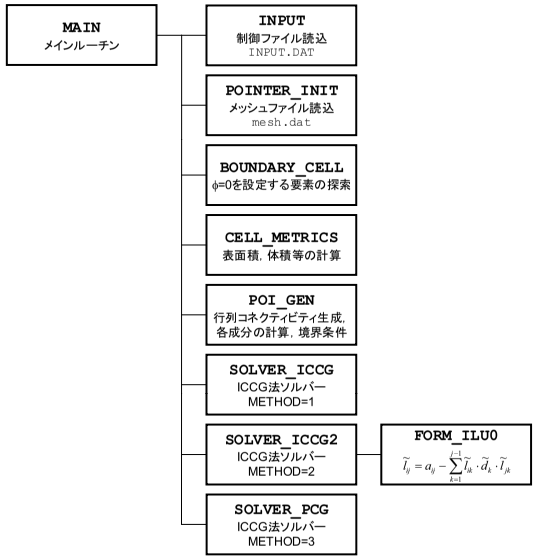
(コメント行)
(ステップ数)
(データの繰り返しタイプ)
(ステップ番号1) (コメント)
(全節点数) (全要素数)
(節点番号1) (X座標) (Y座標) (Z座標)
(節点番号2) (X座標) (Y座標) (Z座標)
.
.
.
(要素番号1) (材料番号) (要素の種類) (要素を構成する節点のつながり)
(要素番号2) (材料番号) (要素の種類) (要素を構成する節点のつながり)
.
.
.
(各節点のデータ数) (各要素のデータ数)
(節点のデータ成分数) (成分1の構成数) (成分2の構成数) ・・・(各成分の構成数)
(節点データ成分1のラベル), (単位)
(節点データ成分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) ・・・・・
.
.
.
(ステップ番号2) (コメント) (全節点数) (全要素数)
.
.
.

- 背景
  - 有限体積法
  - 前処理付反復法
- **ICCG法によるポアソン方程式法ソルバーについて**
  - 実行方法
    - データ構造
  - **プログラムの説明**
    - **初期化**
    - **係数マトリクス生成**
    - ICCG法
- OpenMP「超」入門
- T2K(東大)による実習

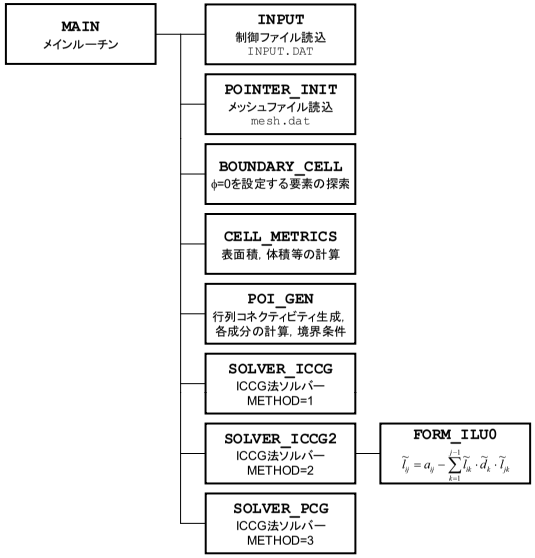
プログラムの構成

```
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
```



プログラムの構成

```
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： 要素数 (NX×NY×NZ)
N： 節点数

NX,NY,NZ： x,y,z方向要素数
NXP1,NYP1,NZP1：
x,y,z方向節点数

IBNODTOT： NXP1×NYP1

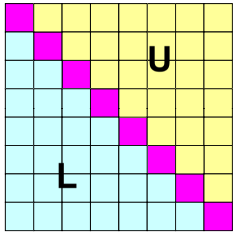
XYZ(ICELTOT,3)： 要素座標
NEIBcell(ICELTOT,6)：
隣接要素

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
```

扱う行列：疎行列
（自分の周辺のみと接続）
⇒ 非ゼロ成分のみを記憶する

上下三角成分を別々に記憶



module PCG(2/5)

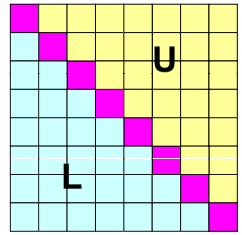
```
      module PCG

      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
```

補助配列

下三角成分(列番号):
非対角成分で自分より要素番号が小さい。
IAL(icou,i) < i

上三角成分(列番号):
非対角成分で自分より要素番号が大きい。
IAU(icou,i) > i

INL(ICELTOT)	非零下三角成分の数
IAL(NL,ICELTOT)	**非零下三角成分（列番号）**
INU(ICELTOT)	非零上三角成分の数
IAU(NU,ICELTOT)	**非零上三角成分（列番号）**
NU,NL	非零上下三角成分の最大数（ここでは6）
indexL(0:ICELTOT)	各行の非零下三角成分数（CRS）
indexU(0:ICELTOT)	各行の非零上三角成分数（CRS）
NPL, NPU	非零上下三角成分数の合計（CRS）
itemL(NPL),itemU(NPU)	比零上下三角成分（列番号）（CRS）



module PCG(3/5)

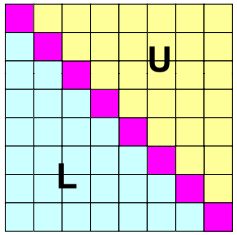
```
      module PCG

      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
```

補助配列

下三角成分(列番号):
IAL(icou,i) < i
その個数が INL(i)

上三角成分(列番号):
IAU(icou,i) > i
その個数がINU(i)

INL(ICELTOT)	**非零下三角成分の数**
IAL(NL,ICELTOT)	非零下三角成分（列番号）
INU(ICELTOT)	**非零上三角成分の数**
IAU(NU,ICELTOT)	非零上三角成分（列番号）
NU,NL	非零上下三角成分の最大数（ここでは6）
indexL(0:ICELTOT)	各行の非零下三角成分数（CRS）
indexU(0:ICELTOT)	各行の非零上三角成分数（CRS）
NPL, NPU	非零上下三角成分数の合計（CRS）
itemL(NPL),itemU(NPU)	比零上下三角成分（列番号）（CRS）



module PCG(4/5)

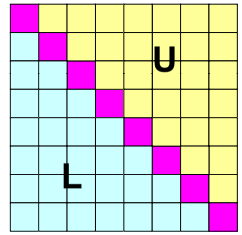
```
      module PCG

      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
```

補助配列

下三角成分(列番号):
IAL(icou,i) < i
その個数が INL(i)

上三角成分(列番号):
IAU(icou,i) > i
その個数がINU(i)

INL(ICELTOT)	非零下三角成分の数
IAL(NL,ICELTOT)	非零下三角成分（列番号）
INU(ICELTOT)	非零上三角成分の数
IAU(NU,ICELTOT)	非零上三角成分（列番号）
NU,NL	非零上下三角成分の最大数（ここでは6）
indexL(0:ICELTOT)	**各行の非零下三角成分数（CRS）**
indexU(0:ICELTOT)	**各行の非零上三角成分数（CRS）**
NPL, NPU	**非零上下三角成分数の合計（CRS）**
itemL(NPL),itemU(NPU)	**比零上下三角成分（列番号）（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	前処理手法（=1,=2,=3）
EPSICCG	収束打切残差
D　(ICELTOT)	係数行列の対角成分
PHI　(ICLETOT)	従属変数
BFORCE(ICELTOT)	連立一次方程式の右辺ベクトル
AL(NPL), AU(NPU)	係数行列の比零上下三角成分（CRS）

行列関係変数：まとめ

配列・変数名	型	内　容
D(N)	**R**	対角成分，(N:全メッシュ数)
BFORCE(N)	**R**	右辺ベクトル
PHI(N)	**R**	未知数ベクトル
indexL(0:N)	**I**	各行の非零下三角成分数(CRS)
indexU(0:N)	**I**	各行の非零上三角成分数(CRS)
NPL	**I**	非零下三角成分総数(CRS)
NPU	**I**	非零上三角成分総数(CRS)
itemL(NPL)	**I**	非零下三角成分(列番号)(CRS)
itemU(NPU)	**I**	非零下三角成分(列番号)(CRS)
AL(NPL)	**R**	非零下三角成分(係数)(CRS)
AU(NPL)	**R**	非零上三角成分(係数)(CRS)

行列関係変数：まとめ（補助配列）

配列・変数名	型	内　容
NL,NU	**I**	各行の非零上下三角成分の最大数（ここでは6）
INL(N)	**I**	各行の非零下三角成分数
INU(N)	**I**	各行の非零上三角成分数
IAL(NL,N)	**I**	各行の非零下三角成分に対応する列番号
IAU(NU,N)	**I**	各行の非零上三角成分に対応する列番号

補助配列を使う理由

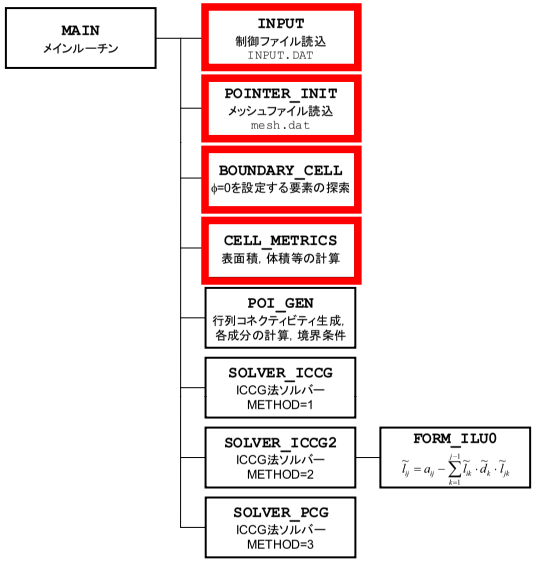
① NPL，NPUの値が前以てわからない
② 後掲の並び替え（ordering，reordering）のとき
CRS形式ではやりにくい

行列ベクトル積：{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
```

プログラムの構成

```
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:「input.dat」の読み込み

```
!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,*) METHOD
        read (11,*) DX, DY, DZ
        read (11,*) EPSICCG
      close (11)
!C===

      return
      end
```

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


pointer_init:「mesh.dat」の読み込み

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

      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
      character*9 fname

      open (21, file='mesh.dat', status='unknown', form='formatted')

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

      allocate (NEIBcell(ICELTOT,6), XYZ(ICELTOT,3))

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

      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

      return
      end
```

NX,NY,NZ：
　x,y,z方向要素数

ICELTOT：
　要素数（NX×NY×NZ）

NEIBcell(ICELTOT,6)：
　隣接要素

XYZ(ICELTOT,3)：
　要素座標

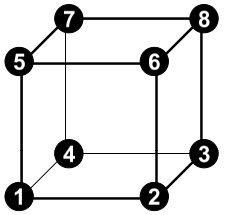
pointer_init

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

      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
      character*9 fname

      open (21, file='mesh.dat' , status='unknown', form='formatted')

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

      allocate (NEIBcell(ICELTOT,6), XYZ(ICELTOT,3))

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

      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

      return
      end
```

DX=0.0となっていた場合のみ，DX,DY,DZをこのように指定

pointer_init：節点⇒可視化用

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

      use STRUCT
      use PCG
      implicit REAL*8 (A-H,O-Z)
      character*9 fname

      write(fname,'(a,i4.4)') 'in.p_', my_rank

      open (21, file= fname, status='unknown', form='formatted')

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

      allocate (NEIBcell(ICELTOT,6), XYZ(ICELTOT,3))

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

      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

      return
      end
```



NXP1, NYP1, NZP1：
　x, y, z方向節点数

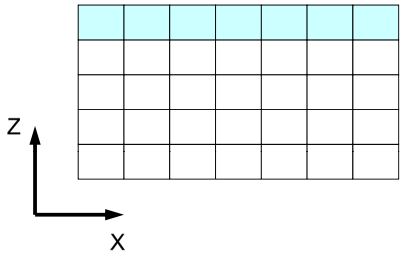
IBNODTOT：
　NXP1×NYP1

N：
　節点数

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
```

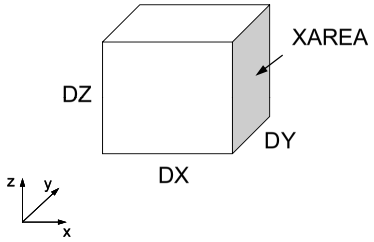
Z=Z_{max}の要素の定義
　総数：　　ZmaxCELtot
　要素番号：　ZmaxCEL(:)



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)

       VO= DX * DY * DZ
      RVO= 1.d0/VO

      VOLCEL=  VO
      RVC   = RVO

      return
      end
```

計算に必要な諸パラメータ



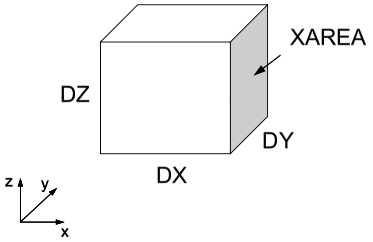
$$XAREA = \Delta Y \times \Delta Z, \quad YAREA = \Delta Z \times \Delta X, \quad ZAREA = \Delta X \times \Delta Y$$

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

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)

       VO= DX * DY * DZ
      RVO= 1.d0/VO

      VOLCEL=  VO
      RVC   = RVO

      return
      end
```

計算に必要な諸パラメータ



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

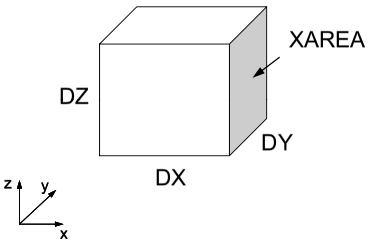
$$R2DX = \frac{1}{0.5 \times \Delta X}, \quad R2DY = \frac{1}{0.5 \times \Delta Y},$$

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

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)

       VO= DX * DY * DZ
      RVO= 1.d0/VO

      VOLCEL=  VO
      RVC   = RVO

      return
      end
```

計算に必要な諸パラメータ

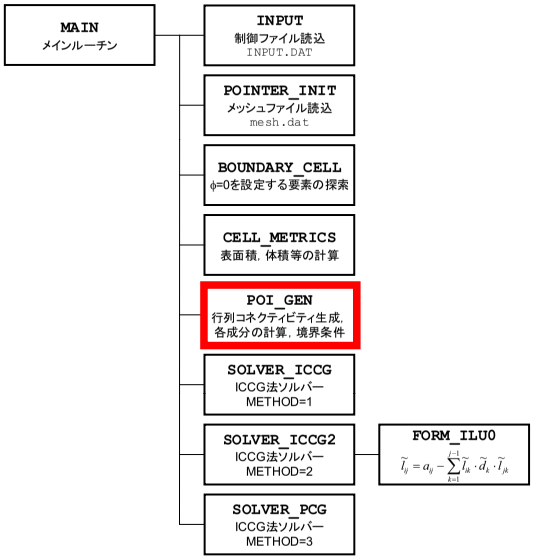


$$VOLCEL = V0 = \Delta X \times \Delta Y \times \Delta Z$$

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

プログラムの構成

```
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（1/6）

```
      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
```

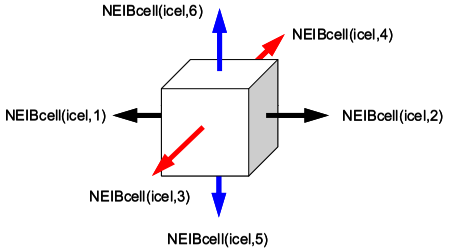
配列の宣言

配列・変数名	型	内 容
D(N)	R	対角成分，(N:全メッシュ数)
BFORCE(N)	R	右辺ベクトル
PHI(N)	R	未知数ベクトル
indexL(0:N)	I	各行の非零下三角成分数(CRS)
indexU(0:N)	I	各行の非零上三角成分数(CRS)
NPL	I	非零下三角成分総数(CRS)
NPU	I	非零上三角成分総数(CRS)
itemL(NPL)	I	非零下三角成分(列番号)(CRS)
itemU(NPU)	I	非零下三角成分(列番号)(CRS)
AL(NPL)	R	非零下三角成分(係数)(CRS)
AU(NPL)	R	非零上三角成分(係数)(CRS)

配列・変数名	型	内 容
NL,NU	I	各行の非零上下三角成分の最大数（ここでは6）
INL(N)	I	各行の非零下三角成分数
INU(N)	I	各行の非零上三角成分数
IAL(NL,N)	I	各行の非零下三角成分に対応する列番号
IAU(NU,N)	I	各行の非零上三角成分に対応する列番号


```
!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

        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===
```

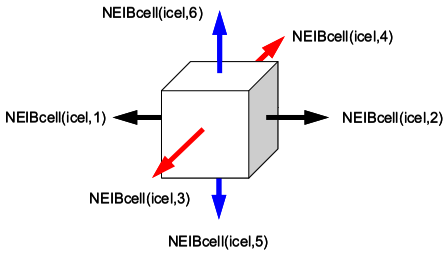
poi_gen（2/6）



下三角成分
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,1)= icel – 1

```
!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

        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（3/6）



上三角成分
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,6)= icel + NX*NY


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

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

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

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

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

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

      itemL= 0
      itemU= 0

        AL= 0.d0
        AU= 0.d0
!C===
```

poi_gen（4/6）

配列の宣言

配列・変数名	型	内 容
D(N)	R	対角成分，(N:全メッシュ数)
BFORCE(N)	R	右辺ベクトル
PHI(N)	R	未知数ベクトル
indexL(0:N)	I	各行の非零下三角成分数(CRS)
indexU(0:N)	I	各行の非零上三角成分数(CRS)
NPL	I	非零下三角成分総数(CRS)
NPU	I	非零上三角成分総数(CRS)
itemL(NPL)	I	非零下三角成分(列番号)(CRS)
itemU(NPU)	I	非零下三角成分(列番号)(CRS)
AL(NPL)	R	非零下三角成分(係数)(CRS)
AU(NPL)	R	非零上三角成分(係数)(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
```

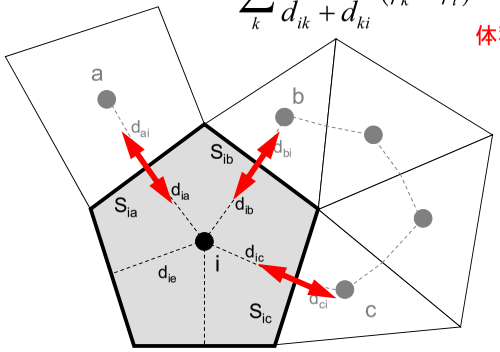

有限体積法
Finite Volume Method（FVM）

面を通過するフラックスの保存に着目

隣接要素との拡散

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

体積フラックス



V_i ：要素体積
S ：表面面積
d_{ij} ：要素中心から表面までの距離
Q ：体積フラックス

全体マトリクスの生成

要素iに関する釣り合い

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

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

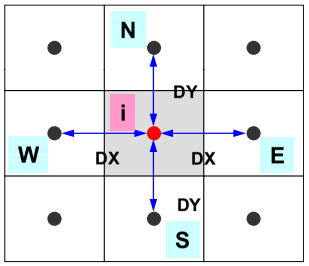
$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=+V_i\dot{Q}_i$$

D（対角成分）　**AL, AU（非対角成分）**　**BFORCE（右辺）**

poi_gen（5/6）

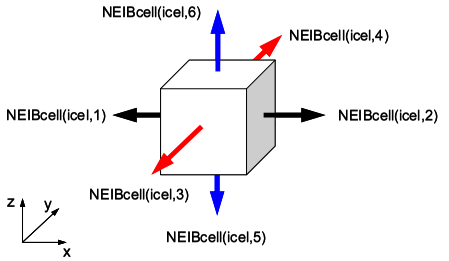
```
!C
!C +-----------------------------+
!C | INTERIOR & NEUMANN BOUNDARY CELLs |
!C +-----------------------------+
!C===
      icouG= 0
      do icel= 1, ICELTOT
        icN1= NEIBcell(icel,1)
        icN2= NEIBcell(icel,2)
        icN3= NEIBcell(icel,3)
        icN4= NEIBcell(icel,4)
        icN5= NEIBcell(icel,5)
        icN6= NEIBcell(icel,6)

        VOLO= 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
```

係数の計算（境界面以外）



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

三次元では・・・




```
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（6/6）

```
        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===

!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
```

体積発熱

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

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

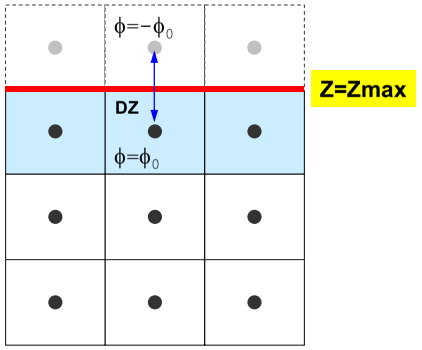
XYZ(icel, k)(k=1,2,3)はX，Y，Z方向の差分格子のインデックス

各メッシュがX，Y，Z方向の何番目にあるかを示している。

poi_gen（6/6）

```
        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===

!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
```

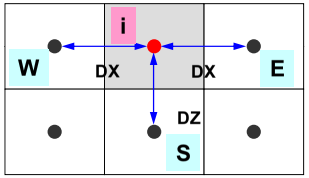
係数の計算（境界面）



境界面の外側に，大きさが同じで，値が$\phi=-\phi_0$となるような要素があると仮定（境界面で丁度$\phi=0$となる）：一次近似

ディリクレ条件

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

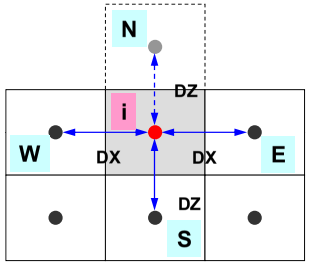


$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=+V_i\dot{Q}_i$$

D（対角成分）　AL, AU（非対角成分）　BFORCE（右辺）

ディリクレ条件

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



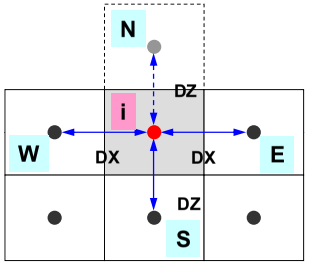
$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=+V_i\dot{Q}_i$$

D（対角成分）　AL, AU（非対角成分）　BFORCE（右辺）

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

ディリクレ条件

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



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

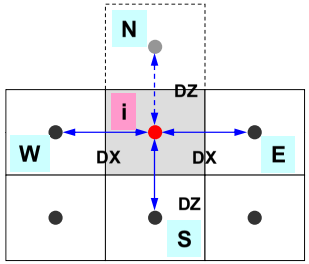
D（対角成分）　AL, AU（非対角成分）　BFORCE（右辺）

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

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

ディリクレ条件

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

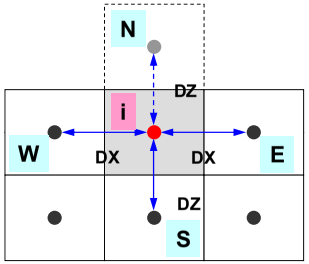


$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=+V_i\dot{Q}_i$$

D（対角成分）　AL, AU（非対角成分）　BFORCE（右辺）

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

ディリクレ条件



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

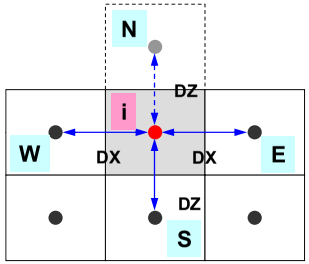
$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]=+V_i\dot{Q}_i$$

D(対角成分)　AL, AU（非対角成分）　BFORCE（右辺）

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

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

ディリクレ条件



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

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

D(対角成分)　AL, AU（非対角成分）　BFORCE（右辺）

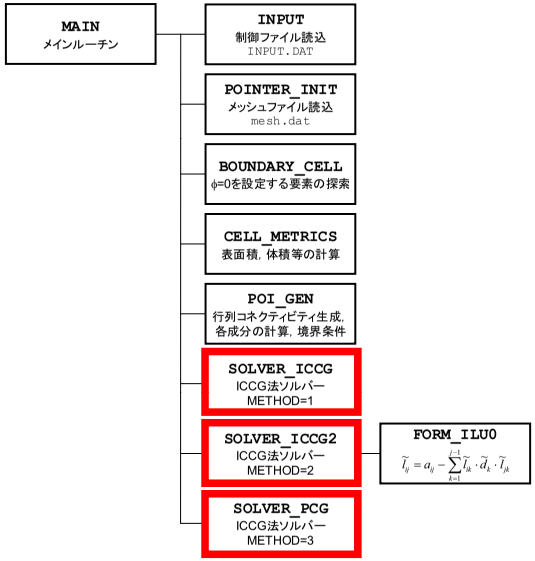
$$\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\right]\phi_i-\left[\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}\phi_k\right]+$$

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

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

プログラムの構成

```
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
```



- 背景
  - 有限体積法
  - 前処理付反復法
- **ICCG法によるポアソン方程式法ソルバーについて**
  - 実行方法
    - データ構造
  - **プログラムの説明**
    - 初期化
    - 係数マトリクス生成
    - **ICCG法**
- OpenMP「超」入門
- T2K（東大）による実習

あとは線形方程式を解けば良い

- 共役勾配法（Conjugate Gradient，CG）
- 前処理
  - 不完全コレスキー分解（Incomplete Cholesky Factorization，IC）
  - 実は不完全「修正」コレスキー分解
- ICCG

修正コレスキー分解

- 対称行列AのLU分解
- 対称行列Aは，対角行列Dを利用して，$[A]=[L][D][L]^T$ のような形に分解することができる。
  - この分解をLDL^T分解または修正コレスキー分解（modified Cholesky decomposition）と呼ぶ。
  - $[A]=[L][L]^T$とするような分解法もある（コレスキー分解）

N=5の場合の例

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

修正コレスキー分解

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

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

ここで $l_{ii} \cdot d_i = 1$ とすると以下が導かれる

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

不完全修正コレスキー分解

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

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

ここで $l_{ii} \cdot d_i = 1$ とすると以下が導かれる

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

実際には，「不完全」な分解を実施し，このような形を用いることが多い

プログラムの実行

制御データ「<$L1>/run/INPUT.DAT」の作成

```
1                                   MEHOD 1:2
1.00e-00 1.00e-00 1.00e-00          DX/DY/DZ
0.10       1.0e-08                  OMEGA, EPSICCG
```

- METHOD
  - 前処理行列の作成方法：不完全修正コレスキー分解
    - METHOD=1　対角成分のみ変更
    - METHOD=2　非対角成分変更（Fill-inは無し：a_{ij}≠0の場合のみ）
    - METHOD=3　対角スケーリング（点ヤコビ）

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

不完全修正コレスキー分解

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

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

ここで $l_{ii}\cdot d_i = 1$　とすると以下が導かれる

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

対角成分のみがもとの行列と変わる

不完全修正コレスキー分解を使用した前進後退代入

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

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

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

不完全修正コレスキー分解を使用した前進後退代入

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

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

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

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

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

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

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

不完全修正コレスキー分解を使用した前進後退代入

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

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

```
!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,Y)
        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,Y) - W(i,DD) * SW
      enddo
!C===
```

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

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

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
```

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===
```

不完全修正コレスキー分解
W(i,DD)= d_i

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

対角成分のみがもとの行列と変わる

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===
```

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==
```

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

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

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

前進代入
Forward Substitution

後退代入
Backward Substitution

solve_ICCG(5/7):METHOD=1

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

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


solve_ICCG2(1/3): METHOD=2

```
!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 :: ALUU0, AUUU0
      real(kind=8), dimension(:), allocatable :: DUU0
```

```
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
```


solve_ICCG2(2/3): METHOD=2

```
!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==
```

DIu0, ALIu0, AUIu0にはILU(0)分解された対角，下三角，上三角成分が入る(行列[M])。

FORM_ILU0(1/2)

不完全修正コレスキー分解：正確には不完全修正LU分解

solver_ICCG2.flに附属

```
 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 (ALUU0(NPL), AUUU0(NPU))
      allocate (DUU0(N))

      do i= 1, N
        DUU0(i)= D(i)
        do k= 1, INL(i)
          ALUU0(k,i)= AL(k,i)
        enddo

        do k= 1, INU(i)
          AUUU0(k,i)= AU(k,i)
        enddo
      enddo
!C===
```

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

DIu0, ALIu0, AUIu0にはILU(0)分解された対角，下三角，上三角成分が入る(行列[M])。

「DIu0,ALIu0,AUIu0」初期値として，「D,AL,AU」の値を代入する。

FORM_ILU0 (2/2)

```
!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= 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

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

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

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

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


FORM_ILU0 (2/2)

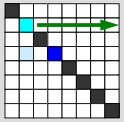
```
!C
!C +--------------------+
!C | ILU(0) factorization |
!C +--------------------+
!C===
      allocate (IW1(N) , IW2(N))
      IW1=0
      IW2= 0

      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= 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

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

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

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

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


FORM_ILU0 (2/2)

```
!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= 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

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



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

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

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


FORM_ILU0 (2/2)

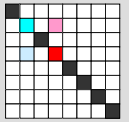
$$i = 1,2,\cdots,n$$

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

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

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

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

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

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



j=i

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

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

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

FORM_ILU0（2/2）

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

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

    do icon= indexL(i-1)+1, indexL(i)
        k= itemL(icon)
      D11= DIU0(k)

      DkINV= 1.d0/D11
        Aik= ALIU0(icon)

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

      if (j.eq.i) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj
        DIU0(i)= DIU0(i) - RHS_Aij
      endif

      if (j.lt.i .and. IW1(j).ne.0) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj

                ijO = IW1(j)
        ALIU0(ijO)= ALIU0(ijO) - RHS_Aij
      endif
```

j<i

```
          if (j.gt.i .and. IW2(j).ne.0) then
                Akj= AUIU0(kcon)
            RHS_Aij= Aik * DkINV * Akj

                    ijO = IW2(j)
              AUIU0(ijO)= AUIU0(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
        DIU0(i)= 1.d0 / DIU0(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）

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

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

    do icon= indexL(i-1)+1, indexL(i)
        k= itemL(icon)
      D11= DIU0(k)

      DkINV= 1.d0/D11
        Aik= ALIU0(icon)

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

      if (j.eq.i) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj
        DIU0(i)= DIU0(i) - RHS_Aij
      endif

      if (j.lt.i .and. IW1(j).ne.0) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj

                ijO = IW1(j)
        ALIU0(ijO)= ALIU0(ijO) - RHS_Aij
      endif
```

```
          if (j.gt.i .and. IW2(j).ne.0) then
                Akj= AUIU0(kcon)
            RHS_Aij= Aik * DkINV * Akj

                    ijO = IW2(j)
              AUIU0(ijO)= AUIU0(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
        DIU0(i)= 1.d0 / DIU0(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

j>i

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


FORM_ILU0（2/2）

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

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

    do icon= indexL(i-1)+1, indexL(i)
        k= itemL(icon)
      D11= DIU0(k)

      DkINV= 1.d0/D11
        Aik= ALIU0(icon)

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

      if (j.eq.i) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj
        DIU0(i)= DIU0(i) - RHS_Aij
      endif

      if (j.lt.i .and. IW1(j).ne.0) then
            Akj= AUIU0(kcon)
        RHS_Aij= Aik * DkINV * Akj

                ijO = IW1(j)
        ALIU0(ijO)= ALIU0(ijO) - RHS_Aij
      endif
```

j<i

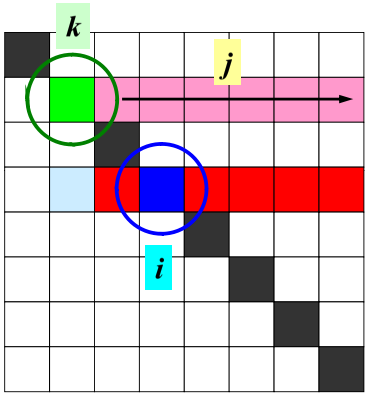
```
          if (j.gt.i .and. IW2(j).ne.0) then
                Akj= AUIU0(kcon)
            RHS_Aij= Aik * DkINV * Akj

                    ijO = IW2(j)
              AUIU0(ijO)= AUIU0(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
        DIU0(i)= 1.d0 / DIU0(i)
      enddo
      deallocate (IW1, IW2)
!C===
      end subroutine FORM_ILU0
```

j>i

実はこのIF文の中は通らない（理由は後述）したがって，

AL|u0= AL
AU|u0= AU

j=i, j<i, j>i（1/3）

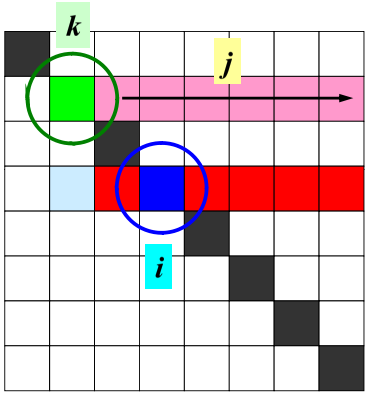


ある要素「i(○)」に接続する下三角成分「k(■○)」の上三角成分「j(■)」が：

(1) **j=i**
「i」自身である場合，Dlu(■)更新

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

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

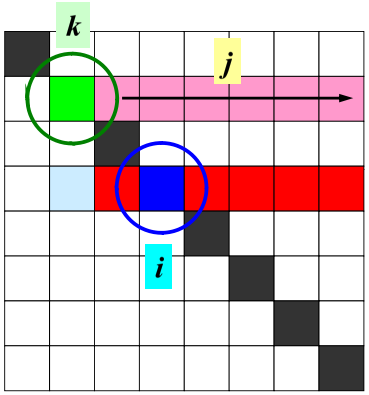


ある要素「i(○)」に接続する下三角成分「k(■○)」の上三角成分「j(■)」が：

(2) **j<i**
「i」の下三角成分である場合
ALlu0(i-j)(■)更新

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

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



ある要素「i(○)」に接続する下三角成分「k(■○)」の上三角成分「j(■)」が：

(3) **j>i**
「i」の上三角成分である場合
AUlu0(i-j)(■)更新

実際は(2)，(3)に該当する場合は無し

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

solve_ICCG2(3/3): METHOD=2

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

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

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

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

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

これ以下の処理は「solve_ICCG」と全く同じ

solve_ICCG2(3/3): METHOD=2

```
!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==
```

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

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

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

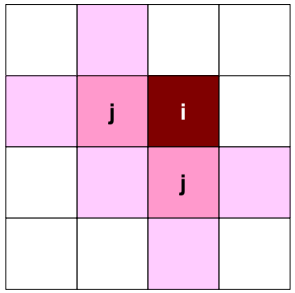
前進代入
Forward Substitution

後退代入
Backward Substitution

実は，ALlu0，AUlu0の値はAL，AUと全く同じである。
METHOD=1，METHOD=2の答え(反復回数)は同じ

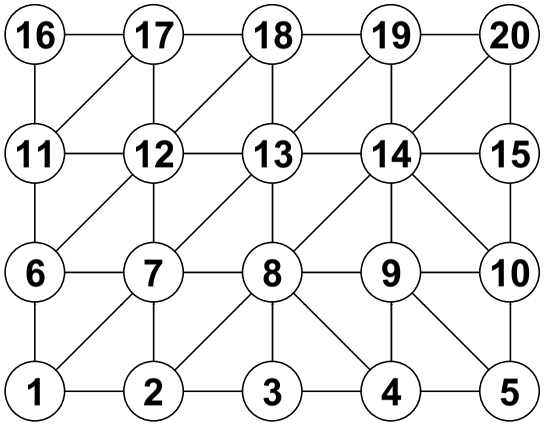
不完全修正コレスキー分解 現在のようなメッシュの場合

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



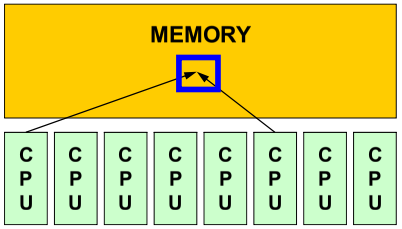
□を満たすような(i-j-k)(i,jの両方に接続するk)が無い
従って，$l_{ij} = a_{ij}$

こういう場合はAUlu0, ALlu0が更新される可能性あり



- 背景
  - 有限体積法
  - 前処理付反復法
- ICCG法によるポアソン方程式法ソルバーについて
  - 実行方法
    - データ構造
  - プログラムの説明
    - 初期化
    - 係数マトリクス生成
    - ICCG法
- **OpenMP「超」入門**
- T2K(東大)による実習

共有メモリ型計算機



- SMP
  - Symmetric Multi Processors
  - 複数のCPU(コア)で同じメモリ空間を共有するアーキテクチュア

OpenMPとは

- 共有メモリ型並列計算機用のDirectiveの統一規格
  - この考え方が出てきたのは MPIやHPFに比べると遅く1996年であるという。
  - 現在 Ver.3.0
- 背景
  - CrayとSGIの合併
  - ASCI計画の開始
- 主な計算機ベンダーが集まって OpenMP ARBを結成し，1997年にはもう規格案ができていたそうである
  - SC98ではすでにOpenMPのチュートリアルがあったし，すでにSGI Origin2000でOpenMP-MPIハイブリッドのシミュレーションをやっている例もあった。

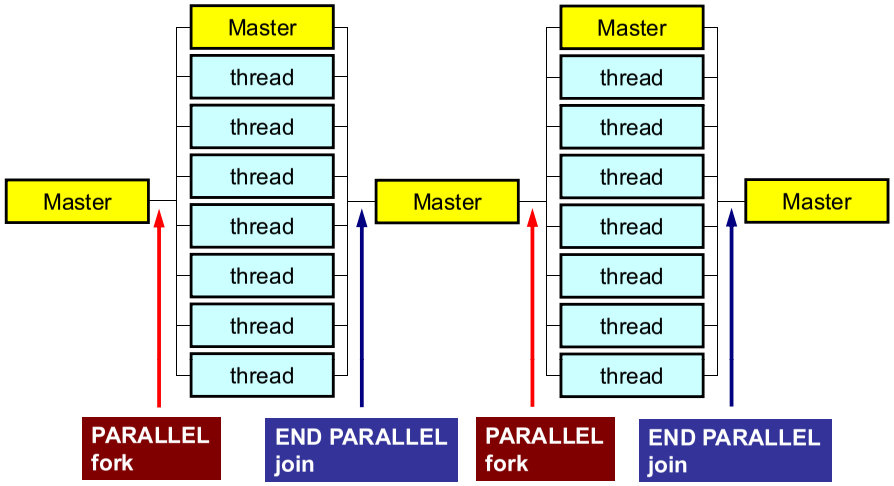
OpenMPとは（続き）

- OpenMPはFortan版とC/C++版の規格が全く別々に進められてきた。
  - Ver.2.5で言語間の仕様を統一
  - 現在特に力が注がれているのが，性能の向上とデバッグツールの整備だということで，データ分割に関しては非常に大変なので今のところ予定には入っていないそうである。
    - 利用者の責任でやらなければならない

OpenMPの概要

- 基本的仕様
  - プログラムを並列に実行するための動作をユーザーが明示
  - OpenMP実行環境は，依存関係，衝突，デッドロック，競合条件，結果としてプログラムが誤った実行につながるような問題に関するチェックは要求されていない。
  - プログラムが正しく実行されるよう構成するのはユーザーの責任である。
- 実行モデル
  - fork-join型並列モデル
    - 当初はマスタスレッドと呼ばれる単一プログラムとして実行を開始し，「PARALLEL」，「END PARALLEL」ディレクティヴの対で並列構造を構成する。並列構造が現れるとマスタスレッドはスレッドのチームを生成し，そのチームのマスタとなる。
  - いわゆる「入れ子構造」も可能であるが，ここでは扱わない

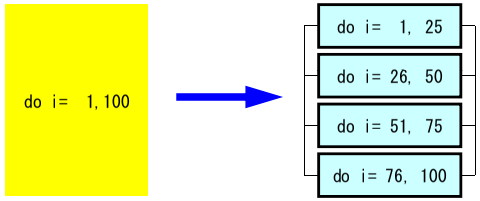
Fork-Join 型並列モデル



スレッド数

- 環境変数 **OMP_NUM_THREADS**
  - 値の変え方
    - bash(.bashrc)　　`export OMP_NUM_THREADS=8`
    - csh(.cshrc)　　`setenv OMP_NUM_THREADS 8`

- **たとえば，OMP_NUM_THREADS=4とすると，以下のようにi=1~100のループが4分割され，同時に実行される。**



OpenMPに関連する情報

- OpenMP Architecture Review Board (ARB)
  - http://www.openmp.org
- 参考文献
  - 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!
- 富士通他による翻訳：（OpenMP 3.0）必携！
  - http://www.openmp.org/mp-documents/OpenMP30spec-ja.pdf

OpenMPに関する国際会議

- WOMPEI(International Workshop on OpenMP: Experiences and Implementations)
  - 日本(1年半に一回)
- WOMPAT(アメリカ)，EWOMP(欧州)

- 2005年からこれらが統合されて「IWOMP」となる，毎年開催。
  - International Workshop on OpenMP
  - http://www.nic.uoregon.edu/iwomp2005/
  - Eugene, Oregon, USA

OpenMPの特徴

- ディレクティヴ(指示行)の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる

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
```

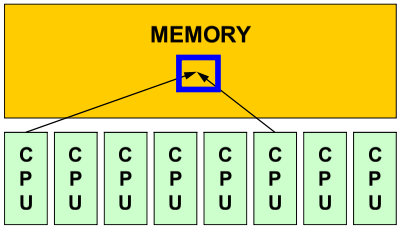

OpenMP/Direceives Matrix/Vector Products

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


OpenMPの特徴

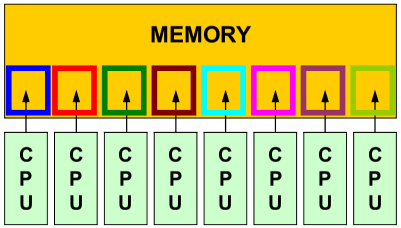
- ディレクティヴ（指示行）の形で利用
  - 挿入直後のループが並列化される
  - コンパイラがサポートしていなければ，コメントとみなされる
- 何も指定しなければ，何もしない
  - 「自動並列化」，「自動ベクトル化」とは異なる。
  - 下手なことをするとおかしな結果になる：ベクトル化と同じ
  - データ分散等（Ordering）は利用者の責任
- 共有メモリユニット内のプロセッサ数に応じて，「Thread」が立ち上がる
  - 「Thread」：MPIでいう「プロセス」に相当する。
  - 普通は「Thread数＝共有メモリユニット内プロセッサ数，コア数」

メモリ競合



- 複雑な処理をしている場合，複数のスレッドがメモリ上の同じアドレスにあるデータを同時に更新する可能性がある。
  - 複数のCPUが配列の同じ成分を更新しようとする。
  - メモリを複数のコアで共有しているためこのようなことが起こりうる。
  - 場合によっては答えが変わる

メモリ競合（続き）



- 本演習で扱っている例は，そのようなことが生じないよう，各スレッドが同時に同じ成分を更新するようなことはないようにする。
  - これはユーザーの責任でやること，である。
- ただ多くのコア数（スレッド数）が増えるほど，メモリへの負担が増えて，処理速度は低下する。

OpenMPの特徴（続き）

- 基本は「!omp parallel do」～「!omp end parallel do」
- 変数について，グローバルな変数と，Thread内でローカルな「private」な変数に分けられる。
  - デフォルトは「global」
  - 内積を求める場合は「reduction」を使う

```
!$omp parallel do private(iS,iE,i)
!$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 などはグローバル変数

FORTRANと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
```

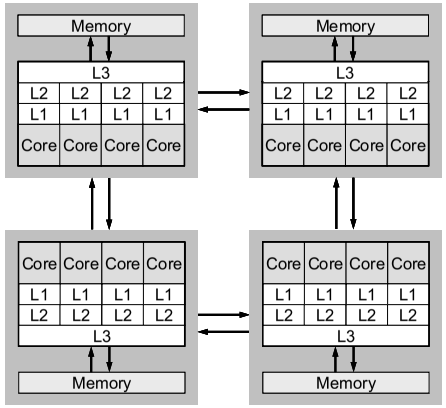
```
#include <omp.h>
...
{
  #pragma omp parallel for default(none) shared(n,x,y) private(i)

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


First Touch Rule

- NUMA（Non-Uniform Access）アーキテクチュアでは，「最初にそのバッファにアクセスしたプロセッサ」のメモリ上にバッファの内容がアサインされる。

- 初期化等の工夫が必要
  - Hitachi SR シリーズ，IBM SP，地球シミュレータ等では問題にはならない
  - T2Kでは結構効く（あとで実習してみよう）
    - ローカルなメモリ上のデータをアクセスするような工夫が必要

HA80000-tc/RS425：ノードの構成



本セミナーにおける方針

- OpenMPは多様な機能を持っているが，それらの全てを逐一教えることはしない。
  - 講演者も全てを把握，理解しているわけではない。
- 数値解析に必要な最低限の機能のみ学習する。
  - 具体的には，講義で扱っているICCG法によるポアソン方程式ソルバーを動かすために必要な機能のみについて学習する
  - それ以外の機能については，自習，質問のこと（全てに答えられるとは限らない）。

最初にやること

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

- 様々な環境変数，インタフェースの定義（OpenMP3.0以降でサポート）

OpenMPディレクィヴ（FORTRAN）

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

- 大文字小文字は区別されない。
- sentinel
  - 接頭辞
  - FORTRANでは「!$OMP」，「C$OMP」，「*$OMP」，但し自由ソース形式では「!$OMP」のみ。
  - 継続行にはFORTRANと同じルールが適用される。以下はいずれも「!$OMP PARALLEL DO SHARED(A,B,C)」

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

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

OpenMPディレクィヴ（C）

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

- 継続行は「＼」
- 小文字を使用（変数名以外）

```
#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)
```

- 多重スレッドによって実行される領域を定義し，DOループの並列化を実施する。
- 並び項目（clause）：よく利用するもの
  - PRIVATE（list）
  - SHARED（list）
  - DEFAULT（PRIVATE|SHARED|NONE）
  - REDUCTION（{operation|intrinsic}: list）

REDUCTION

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

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

- 「MPI_REDUCE」のようなものと思えばよい
- Operator
  - ＋，*，-，.AND.，.OR.，.EQV.，.NEQV.
- Intrinsic
  - MAX，MIN，IAND，IOR，IEQR

実例A1：簡単なループ

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

- ループの繰り返し変数（ここでは「i」）はデフォルトでprivateなので，明示的に宣言は不要。
- 「END PARALLEL DO」は省略可能。
  - C言語ではそもそも存在しない

実例A2：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」は省略可能。

OpenMPを適用するには？（内積）

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


OpenMPを適用するには？（内積）

```
    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
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

OpenMPを適用するには？（内積）

```
    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
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

↓

```
    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
```

多重ループの導入
PEsmpTOT：スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施
（別に効率がよくなるわけでは無い）

OpenMPを適用するには？(内積)

```
      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
```

OpenMPディレクティヴの挿入
これでも並列計算は可能

```
      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
```

多重ループの導入
PEsmpTOT:スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，
並列に実行

OpenMPを適用するには？(内積)

```
      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
```

多重ループの導入
PEsmpTOT:スレッド数
あらかじめ「INDEX(:)」を用意しておく
より確実に並列計算実施

PEsmpTOT個のスレッドが立ち上がり，
並列に実行

各要素が計算されるスレッドを
指定できる

例えば，N=100，PEsmpTOT=4とすると：

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

実例：FORTRAN，C共通

```
>$ cd <$omp>


>$ ifort -O4 -openmp test.f
>$ icc -O3 -openmp test.c
```


test.fの内容

- DAXPY
  - ベクトルとその定数倍の加算
- DOT
  - 内積
- OpenMPディレクティブ挿入の効果

test.f(1/3) : 初期化

```
      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===
      call MPI_INIT (ierr)
      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===
```

時間計測のためにMPI使用

問題サイズ，オプション指定

nopt＝0　First Touchなし
nopt≠0　First Touchあり

test.f(1/3) : 初期化

```
      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===
      call MPI_INIT (ierr)
      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　First Touchなし
並列化せずに初期化

test.f(1/3) : 初期化

```
      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===
      call MPI_INIT (ierr)
      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　First Touchあり

計算をするときと同じように
OpenMPを使って並列化

これで計算をするコアのローカル
メモリにデータが保存される

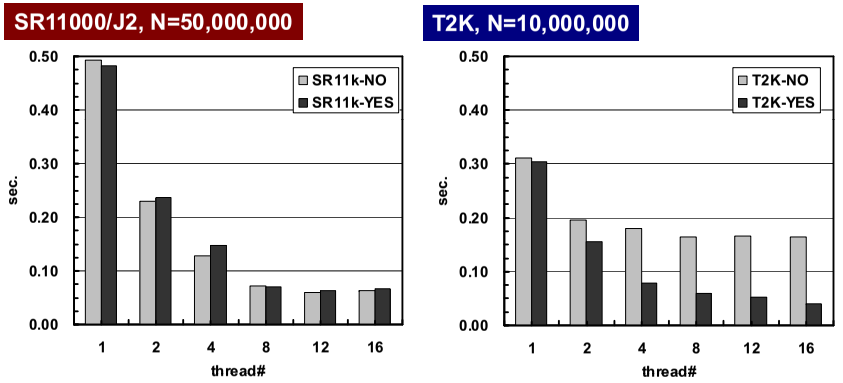
test.f(2/3) : DAXPY

```
!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):内積

```
!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===
      call MPI_FINALIZE (ierr)
      stop
      end
```


DAXPY:First Touchの効果



- T2K:First Touch(後述)の有無の効果が大
- コア数を増やしても性能が上がらない
  - オーバーヘッド，メモリ競合

ICCG法の並列化

- 内積:OK
- DAXPY:OK
- 行列ベクトル積
- 前処理

行列ベクトル積

```
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
```

行列ベクトル積

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


行列ベクトル積：これでもOK

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


ICCG法の並列化

- 内積：OK
- DAXPY：OK
- 行列ベクトル積：OK
- 前処理

前処理はどうするか？
対角スケーリングなら簡単：でも遅い

```
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
```

前処理はどうするか?

不完全修正コレスキー分解

```
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
```

前進代入

```
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
```


データ依存性:メモリの読み込みと書き出しが同時に発生し，並列化困難

不完全修正コレスキー分解

```
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
```

前進代入

```
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
```


前進代入 4スレッドによる並列化を試みる

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

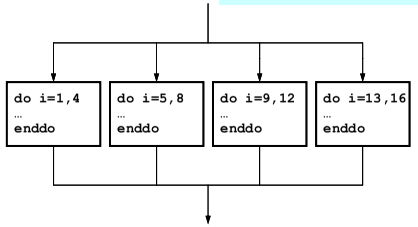
```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```


前進代入 4スレッドによる並列化を試みる

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
```



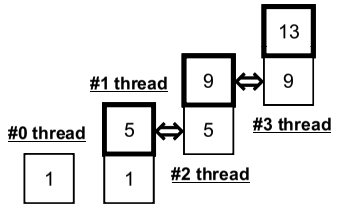
このような4スレッドが同時に実施される・・・

データ依存性：メモリへの書き出し，読み込みが同時に発生

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
```



⇔の部分に
データ依存性発生

ICCG法の並列化

- 内積：OK
- DAXPY：OK
- 行列ベクトル積：OK
- 前処理：なんとかしなければならない
  - 単純にOpenMPなどの指示行（directive）を挿入しただけでは「並列化」できない。

- 背景
  - 有限体積法
  - 前処理付反復法
- ICCG法によるポアソン方程式法ソルバーについて
  - 実行方法
    - データ構造
  - プログラムの説明
    - 初期化
    - 係数マトリクス生成
    - ICCG法
- OpenMP「超」入門
- **T2K（東大）による実習**

T2K（東大）：バッチジョブ実行

- 実行手順
  - ジョブスクリプトを書きます
  - ジョブを投入します
  - ジョブの状態を確認します
  - 結果を確認します
- その他
  - 実行時には1ノード（16コア）が占有されます
  - 他のユーザーのジョブに使われることはありません

ジョブスクリプト

- <$omp>/go.sh
- スケジューラへの指令 ＋ シェルスクリプト

```
#@$-r S1-3                  実行ジョブ名（qstatで表示）
#@$-q tutorial              実行キュー名
#@$-N 1                     使用ノード数（固定）
#@$-J T1                    ノードあたりMPIプロセス数（固定）
#@$-e err                   標準エラー出力ファイル名
#@$-o a016.lst              標準出力ファイル名
#@$-lM 28GB                 1ノードあたりメモリ使用量（固定）
#@$-lT 00:05:00             実行時間（上限10分，この場合は5分）
#@$

export OMP_NUM_THREADS=16   スレッド数（利用可能コア数）指定
cd $PBS_O_WORKDIR           実行ディレクトリ移動
./a.out                     計算実行
```


ジョブ投入

```
>$ cd <$omp>
>$ qsub go.sh
```


利用可能なキュー

```
#@$-r S1-3                  実行ジョブ名（qstatで表示）
#@$-q tutorial              実行キュー名
#@$-N 1                     使用ノード数
#@$-J T16                   ノードあたりMPIプロセス数（T1~T16）
```

- tutorial
  - 1ノード（16コア），15分
  - 本講習会の受講者のみ利用可能
  - 有効期間
    - 6月30日・7月1日の講義時間中
- lecture
  - 1ノード（16コア），15分
  - 他の教育利用者と共用
  - 有効期間
    - 7月8日（金）16:59 それ以降はログイン不可

ジョブ投入，確認等

- ジョブの投入　`qsub スクリプト名`
- ジョブの確認　`qstat`
- キューの状態の確認　`qstat -b`
- ジョブの取り消し・強制終了　`qdel ジョブID`

```
[t19000@ha8000-2 omp]$ qstat -b
2008/12/01 (Mon) 22:12:17:    BATCH QUEUES on HA8000 cluster
NQS schedule stop time : 2008/12/19 (Fri)  9:00:00 (Remain: 418h 47m 43s)
QUEUE NAME      STATUS    TOTAL   RUNNING  RUNLIMIT  QUEUED  HELD    IN-TRANSIT
debug           AVAILBL     0       0        4         0       0       0  これは使わないこと
lecture9        AVAILBL     0       0        4         0       0       0
[t19000@ha8000-2 omp]$ qsub go.sh
Request 122345.batch1 submitted to queue: lecture9.
[t19000@ha8000-2 omp]$ qstat
2008/12/01 (Mon) 22:12:24:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2008/12/19 (Fri)  9:00:00 (Remain: 418h 47m 36s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
122345.batch1   go0       t19000     lecture9  63  0    unlimit    27GB  QUEUED
[t19000@ha8000-2 omp]$ qstat
2008/12/01 (Mon) 22:12:26:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2008/12/19 (Fri)  9:00:00 (Remain: 418h 47m 34s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
122345.batch1   go0       t19000     lecture9  63  0    unlimit    27GB  RUNNING
[t19000@ha8000-2 omp]$ qdel 122345
deleting request 122347.batch1.
[t19000@ha8000-2 omp]$ qstat
2008/12/01 (Mon) 22:12:28:    REQUESTS on HA8000 cluster
NQS schedule stop time : 2008/12/19 (Fri)  9:00:00 (Remain: 418h 47m 32s)
   REQUEST      NAME      OWNER      QUEUE     PRI NICE   CPU    MEM    STATE
No requests.
```

結果確認

- ジョブが終了するとメールがきます
  - ジョブスクリプトに -mu オプションを書けば任意のメールアドレスに送信できます
  - ~/.forward を設定しておけばオプションを書かなくても自分のメールアドレスに送信できます

```
[t19000@ha8000-2 omp]$ mail
Mail version 8.1 6/6/93.  Type ? for help.
"/var/spool/mail/t19000": 2 messages 2 new
>N  1 root@ha8000.cc.u-tok  Mon Dec  1 22:12  24/1061  "NQS Initiator Report: 122345.ba"
 N  2 root@ha8000.cc.u-tok  Mon Dec  1 22:12  31/1279  "NQS Terminator Report: 122345.b"
&
```

- 結果の確認
  - 標準出力:
  - 標準エラー出力