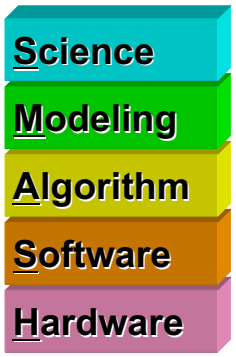
多重格子法
Multigrid Method

2009年3月12日

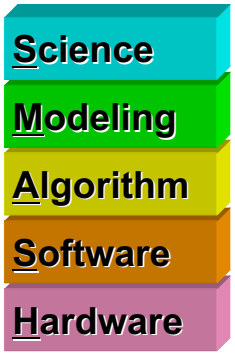
中島研吾（東京大学情報基盤センター）

科学技術計算＝SMASH



- Science
  - 流体力学，材料科学・・・
- Modeling
  - 差分法，有限要素法，分子動力学・・・
- Algorithm
  - 非線形アルゴリズム，線形方程式解法・・・
- Software
  - プログラミング，MPI・・・
- Hardware
  - チューニング，最適化・・・

全てを一人でカバーするのは難しいが・・・



- それぞれの階層についてある程度知っておくことは必要
- このコマでは主に「Algorithm」の話をする
  - SM,SHの話を全くしないわけにも行かない
- 並列計算＝大規模問題
  - 今までとは，
    - 異なったHardware，Software
    - 異なった科学が見えてくるかも知れない
    - 異なったモデルを必要とするかも知れない
    - 異なったアルゴリズムを必要とするかも知れない
    - 今までと同じでもよいかも知れないが・・・

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

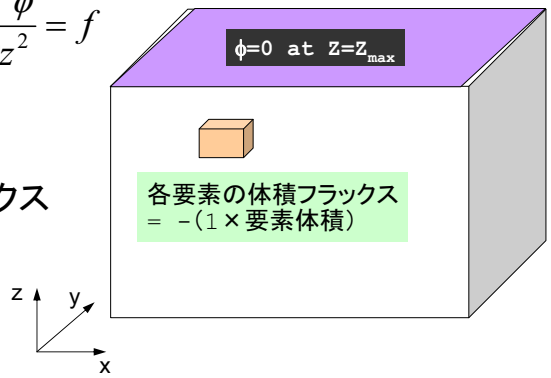
$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f$$

- 有限体積法（Finite Volume Method，**FVM**）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
    - http://nkl.cc.u-tokyo.ac.jp/seminars/0812-JSIAM/2008JSIAMfall-01.pdf
- 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
  - Gauss-Seidel
  - 共役勾配法（CG）＋前処理（IC(0)）⇒ICCG
  - 多重格子法

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

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



境界条件

- 各要素で体積フラックス
- $Z=Z_{max}$面で$\phi=0$

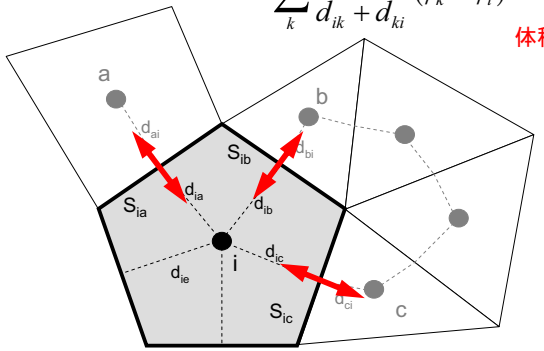
有限体積法
Finite Volume Method（FVM）

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

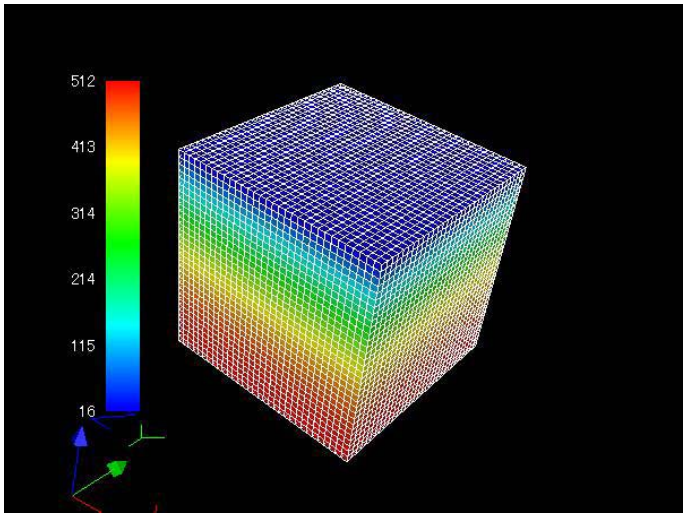
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)+V_i\dot{Q}_i=0$$

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

計算結果（$N=32^3$）



概 要

- 連立一次方程式の解法
  - Gauss-Seidel法
  - CG法（共役勾配法）
- 有限体積法によるアプリケーション
- 多重格子法のアルゴリズム
- Geometrical Multigridの実例
  - 二重球殻内の自然対流，ポアソン方程式
  - 計算例

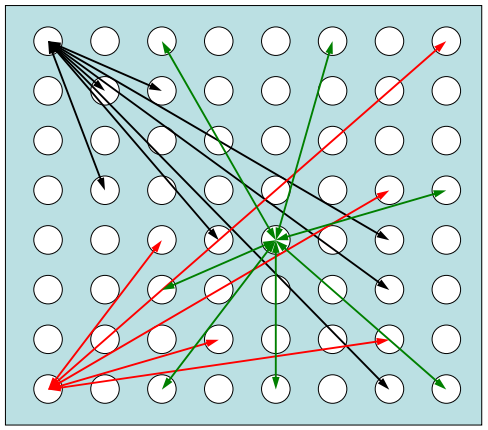
科学技術計算における大規模線形方程式の解法

- 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あり：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

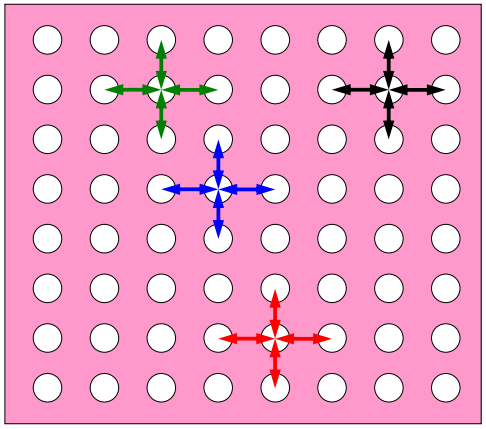
（密行列）

遠隔領域も含め，多数領域との相互作用あり
境界要素法，スペクトル法，MD法



（疎行列）：ここで扱うのはこちら

近接領域のみとの相互作用
差分法，有限要素法



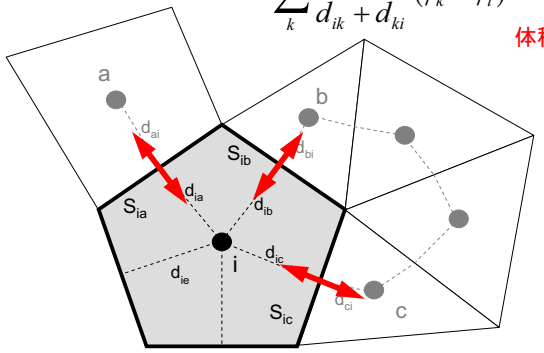
FVMの係数行列：疎行列（0が多い）

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

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

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

有限体積法の係数マトリクス
ゼロが多い：疎行列

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

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

疎行列：非零成分のみ記憶
⇒メモリへの負担大
（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
```


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あり：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。
- 本講習会では反復法を扱う
  - 定常：Gauss-Seidel法
  - 非定常：CG法（共役勾配法）

定常反復法の戦略

連立一次方程式

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

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

初期解

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

適当な初期解 $\mathbf{x}^{(0)}$から始めて，繰り返し計算によって真の解に収束させていく

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

ヤコビ法（Jacobi）

連立一次方程式

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

k回目の反復における解の推定値（k）

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

次のステップの推定値（$k+1$） **右辺の x_i は全て（k）における値**

$$x_1^{(k+1)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} \cdots - a_{1n}x_n^{(k)}\right)$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k)} - a_{23}x_3^{(k)} \cdots - a_{2n}x_n^{(k)}\right)$$

$$\vdots \qquad\qquad \vdots$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}\left(b_n - a_{n1}x_1^{(k)} - a_{n2}x_2^{(k)} \cdots - a_{n,n-1}x_{n-1}^{(k)}\right)$$

$$\rightarrow \quad \mathbf{x}^{(k+1)} = \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix}$$

何をやっているのか？

連立一次方程式：n個の未知数，n個の方程式

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

n個の方程式を一つずつ解いていく
i番目の方程式を解くときは，$x_i^{(k+1)}$ のみが未知数，あとは既知の値として解く

$$a_{i1}x_1 + a_{i2}x_2 \cdots + a_{i\,i-1}x_{i-1} + a_{ii}x_i + a_{i\,i+1}x_{i+1} \cdots + a_{in}x_n = b_i$$

$$a_{i1}x_1^{(k)} + a_{i2}x_2^{(k)} \cdots + a_{i\,i-1}x_{i-1}^{(k)} + a_{ii}x_i^{(k+1)} + a_{i\,i+1}x_{i+1}^{(k)} \cdots + a_{in}x_n^{(k)} = b_i$$

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - a_{i1}x_1^{(k)} - a_{i2}x_2^{(k)} \cdots - a_{i\,i-1}x_{i-1}^{(k)} - a_{i\,i+1}x_{i+1}^{(k)} \cdots - a_{in}x_n^{(k)}\right)$$

ガウス・ザイデル法（Gauss-Seidel）

次のステップの推定値（$k+1$） **右辺の x_i は既に計算されている場合は（$k+1$）における値を使用（最新の値）**

$$x_1^{(k+1)} = \frac{1}{a_{11}}\left(b_1 - a_{12}x_2^{(k)} - a_{13}x_3^{(k)} - a_{14}x_4^{(k)} \cdots - a_{1n}x_n^{(k)}\right)$$

$$x_2^{(k+1)} = \frac{1}{a_{22}}\left(b_2 - a_{21}x_1^{(k+1)} - a_{23}x_3^{(k)} - a_{24}x_4^{(k)} \cdots - a_{2n}x_n^{(k)}\right)$$

$$x_3^{(k+1)} = \frac{1}{a_{33}}\left(b_3 - a_{31}x_1^{(k+1)} - a_{32}x_2^{(k+1)} - a_{34}x_4^{(k)} \cdots - a_{3n}x_n^{(k)}\right)$$

$$\vdots \qquad\qquad\qquad\qquad \vdots$$

$$x_n^{(k+1)} = \frac{1}{a_{nn}}\left(b_n - a_{n1}x_1^{(k+1)} - a_{n2}x_2^{(k+1)} - a_{n2}x_2^{(k+1)} \cdots - a_{n,n-1}x_{n-1}^{(k+1)}\right)$$

通常，ヤコビ法よりも速く収束する（大体2倍くらいの速さと言われている）

$$\rightarrow \quad \mathbf{x}^{(k+1)} = \begin{pmatrix} x_1^{(k+1)} \\ x_2^{(k+1)} \\ \vdots \\ x_n^{(k+1)} \end{pmatrix}$$

ヤコビ法とガウス・ザイデル法

連立一次方程式

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

k回目の反復における解の推定値（k）

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

次のステップの推定値（$k+1$）

ヤコビ法

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1,\,j\neq i}^{n} a_{ij}x_j^{(k)}\right) \quad (1 \le i \le n)$$

ガウス・ザイデル法

$$x_i^{(k+1)} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij}x_j^{(k)}\right) \quad (1 \le i \le n)$$

SOR法

$$x_i^{(k+1)} = x_i^{(k)} + \omega\left(x_{G-S\,i}^{(k+1)} - x_i^{(k)}\right)$$
$$(1 \le i \le n, 1 \le \omega \le 2)$$

反復法の収束判定
（「解を得られた」という判定）

解の推定値 $\mathbf{x}^{(k)}$

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

- 適切な条件のもとで，$k \Rightarrow k+1$のプロセスを繰り返すことによって，$\mathbf{x}^{(k)}$は正しい解に収束していく。
- $[\mathbf{A}]\{\mathbf{x}\}=\{\mathbf{b}\}$という方程式を解いているので，$\|\mathbf{b}-\mathbf{Ax}\| \sim 0$となれば収束したとみなすことができる。
- 通常は$\|\mathbf{b}\|$で無次元化した「残差ノルム」が予め設定した値 ε より小さくなった場合に収束したとみなす。ε の値は要求される精度によって異なる。

残差ノルム

$$\frac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|}{\|\mathbf{b}\|} < \varepsilon$$

$$\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\| = \sqrt{\sum_{i=1}^{n}\left|b_i - \sum_{j=1}^{n} a_{ij} x_j^{(k)}\right|^2}, \quad \|\mathbf{b}\| = \sqrt{\sum_{i=1}^{n}|b_i|^2}$$

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）
  - 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
  - （ガラーキン法）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。
  - 詳細は参考文献参照
    - 例えば：森正武「数値解析（第2版）」（共立出版）

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)
    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（1/5）

yを厳密解（$Ay=b$）とするとき，下式を最小にするxを求める：

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

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

定数

従って，下記 $f(x)$ を最小にするxを求めればよい：

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

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

任意のベクトルh

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

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

任意のベクトルh

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

共役勾配法のアルゴリズム(2/5)

CG法は任意の $x^{(0)}$ から始めて，$f(x)$の最小値を逐次探索する。
今，k番目の近似値 $x^{(k)}$と探索方向 $p^{(k)}$が決まったとすると：

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

$f(x^{(k+1)})$ を最小にするためには：

$$f\left(x^{(k)}+\alpha_k p^{(k)}\right)=\frac{1}{2}\alpha_k{}^2\left(p^{(k)},Ap^{(k)}\right)-\alpha_k\left(p^{(k)},b-Ax^{(k)}\right)+f\left(x^{(k)}\right)$$
$$\frac{\partial f\left(x^{(k)}+\alpha_k p^{(k)}\right)}{\partial\alpha_k}=0\Rightarrow\alpha_k=\frac{\left(p^{(k)},b-Ax^{(k)}\right)}{\left(p^{(k)},Ap^{(k)}\right)}=\frac{\left(p^{(k)},r^{(k)}\right)}{\left(p^{(k)},Ap^{(k)}\right)}$$

$r^{(k)}=b-Ax^{(k)}$ は第k近似に対する残差

共役勾配法のアルゴリズム(3/5)

残差 $r^{(k)}$も以下の式によって計算できる：

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

探索方向を以下の漸化式によって求める：

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

本当のところは下記のように(k+1)回目に厳密解 y が求まれば良いのであるが，解がわかっていない場合は困難・・・

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

共役勾配法のアルゴリズム(4/5)

ところで，下式のような都合の良い直交関係がある：

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

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

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

従って以下が成立する

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

共役勾配法のアルゴリズム(5/5)

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

$\left(p^{(k+1)}, Ap^{(k)}\right)=0$　勾配ベクトル $p^{(k)}$がAに関して共役である

勾配ベクトル $p^{(k)}$, 残差ベクトル $r^{(k)}$ についても以下の関係が成立
(証明略:帰納法)

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

N次元空間で互いに直交で一次独立な残差ベクトル $r^{(k)}$ はN個しか存在しない, 従って共役勾配法は未知数がN個のときにN回以内に収束する
⇒ 実際は丸め誤差の影響がある

$\alpha_k,\ \beta_k$

実際はα_k, β_kはもうちょっと簡単な形に変形できる:

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

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

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
  - 対角優位
- 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。
- 「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。

対角優位性

$$\left|a_{ii}\right| \geq \sum_{j=1, j\neq i}^{n}\left|a_{ij}\right| \quad i=1,2,\cdots,n$$　(対角優位)

第 i 行の、対角項以外の成分の絶対値の和よりも
対角項の絶対値が大きい場合

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 z(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
```

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）

- 連立一次方程式の解法
  - Gauss-Seidel法
  - CG法（共役勾配法）
- **有限体積法によるアプリケーション**
- 多重格子法のアルゴリズム
- Geometrical Multigridの実例
  - 二重球殻内の自然対流，ポアソン方程式
  - 計算例

ファイルの用意

コピー，展開，Intelコンパイラ使用

```
>$ cd
>$ source /opt/itc/mpi/mpiswitch.sh mpich-mx-intel
>$ cp /home/h08000/mgclass.tar .
>$ tar xvf mgclass.tar
```

以下のディレクトリが出来ていることを確認

```
mgclass
```

その下には，以下のディレクトリがある：

```
parallel  single
```

これらを以降<$P>，<$S>と呼ぶ

日立コンパイラに戻るとき

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


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

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

- 有限体積法（Finite Volume Method，**FVM**）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
    - http://nkl.cc.u-tokyo.ac.jp/seminars/0812-JSIAM/2008JSIAMfall-01.pdf
- 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
  - Gauss-Seidel
  - 共役勾配法（CG）＋前処理（IC(0)）⇒ICCG
  - 多重格子法

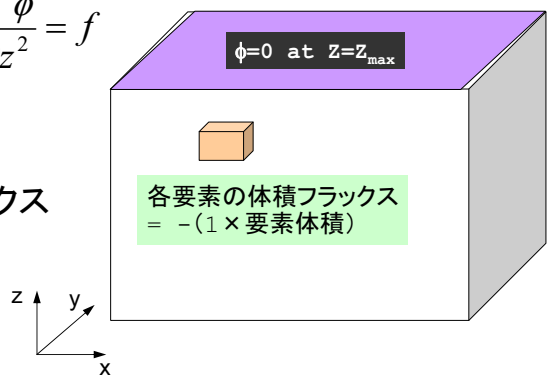
解いている問題：三次元ポアソン方程式
変数：要素中心で定義

ポアソン方程式

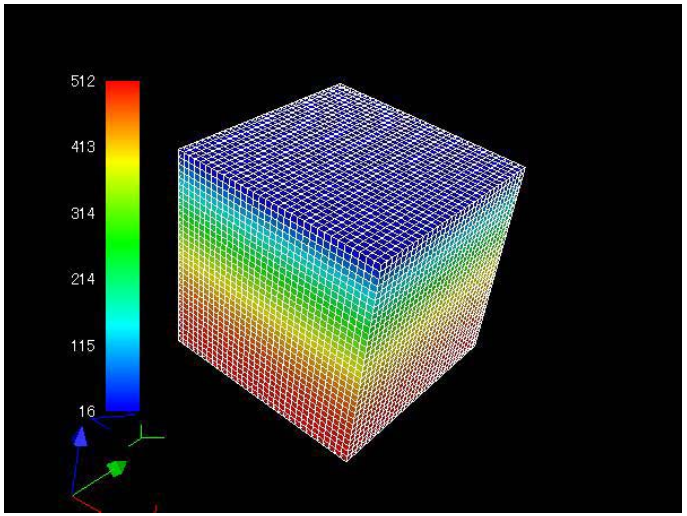
$$\Delta\phi = \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} + \frac{\partial^2\phi}{\partial z^2} = f$$

境界条件

- 各要素で体積フラックス
- $Z=Z_{max}$面で$\phi=0$

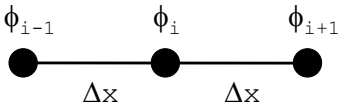


計算結果（$N=32^3$）



一次元ポアソン方程式：中央差分

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



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

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

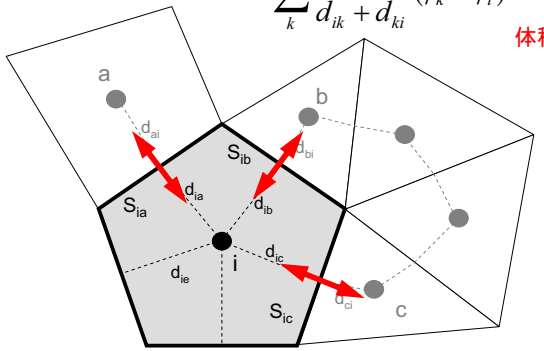
有限体積法
Finite Volume Method（FVM）

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

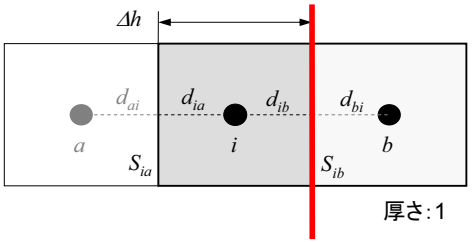
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k - \phi_i) + V_i\dot{Q}_i = 0$$

体積フラックス



Vi ：要素体積
S ：表面面積
dij ：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）



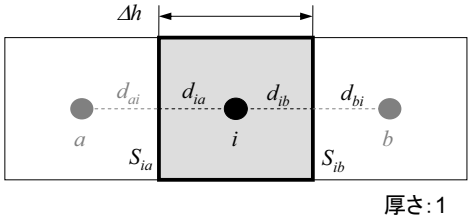
一辺の長さΔhの正方形メッシュ
接触面積：$S_{ik} = \Delta h$
要素体積：$V_i = \Delta h^2$
接触面までの距離：$d_{ij} = \Delta h/2$

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：$d_{ij} = \Delta h/2$

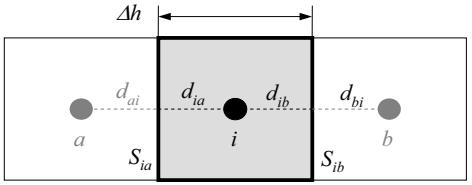
$$\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k - \phi_i) + V_i\dot{Q}_i = 0$$

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)



厚さ:1

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: $d_{ij}=\Delta h/2$

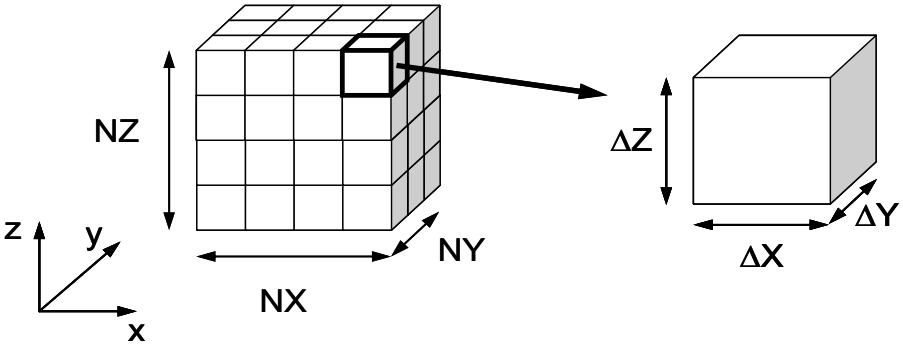
$$\frac{1}{V_i}\sum_k \frac{S_{ik}}{d_{ik}+d_{ki}}(\phi_k-\phi_i)=\frac{1}{(\Delta h)^2}\sum_{k=a,b}\frac{\Delta h}{\frac{\Delta h}{2}+\frac{\Delta h}{2}}(\phi_k-\phi_i)$$

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

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

対象:規則正しい三次元差分格子

半非構造的に扱う



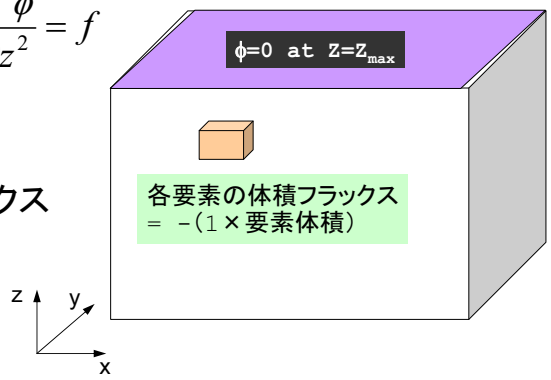
解いている問題:三次元ポアソン方程式
変数:要素中心で定義

ポアソン方程式

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

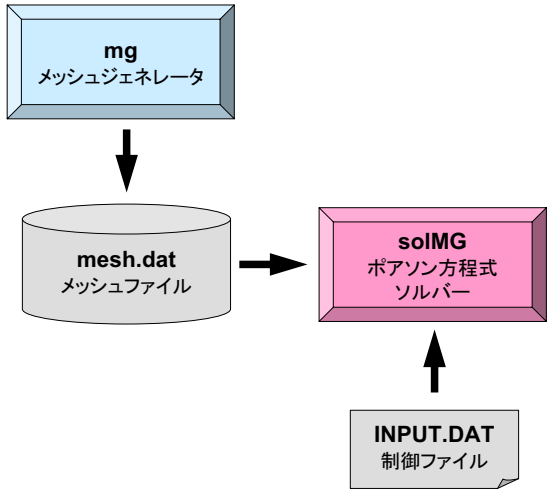
<u>境界条件</u>

- 各要素で体積フラックス
- Z=Z_{max}面でϕ=0



計算の手順

プログラム, 必要なファイル等



プログラムの実行
コンパイル

```
$> cd <$S>/run

$> ls mg
      mg
$> f90 -Oss -noparallel mg.f -o mg

$> cd ../src
$> make
$> ls ../run/solMG
      solMG
```

メッシュジェネレータ: **mg**

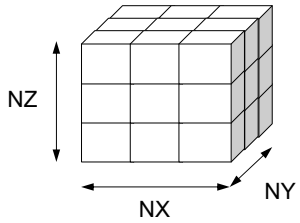
ソースコード:**mg.f, 日立コンパイラ**

ポアソン方程式ソルバー: **solMG**

プログラムの実行
メッシュ生成

```
$> cd <$S>/run
$> $> ./mg
   3
$> ls mesh.dat
     mesh.dat
```

NX, NY, NZを入力すると,「mesh.dat」が生成される, 本プログラムではNX=NY=NZなので, NXのみ入力



X,Y,Z方向の要素数

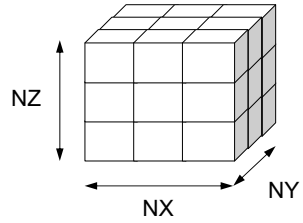
mesh.dat(1/5)

```
  3   3   3
 27
  1   0   2   0   4   0  10   1   1   1
  2   1   3   0   5   0  11   2   1   1
  3   2   0   0   6   0  12   3   1   1
  4   0   5   1   7   0  13   1   2   1
  5   4   6   2   8   0  14   2   2   1
  6   5   0   3   9   0  15   3   2   1
  7   0   8   4   0   0  16   1   3   1
  8   7   9   5   0   0  17   2   3   1
  9   8   0   6   0   0  18   3   3   1
 10   0  11   0  13   1  19   1   1   2
 11  10  12   0  14   2  20   2   1   2
 12  11   0   0  15   3  21   3   1   2
 13   0  14  10  16   4  22   1   2   2
 14  13  15  11  17   5  23   2   2   2
 15  14   0  12  18   6  24   3   2   2
 16   0  17  13   0   7  25   1   3   2
 17  16  18  14   0   8  26   2   3   2
 18  17   0  15   0   9  27   3   3   2
 19   0  20   0  22  10   0   1   1   3
 20  19  21   0  23  11   0   2   1   3
 21  20   0   0  24  12   0   3   1   3
 22   0  23  19  25  13   0   1   2   3
 23  22  24  20  26  14   0   2   2   3
 24  23   0  21  27  15   0   3   2   3
 25   0  26  22   0  16   0   1   3   3
 26  25  27  23   0  17   0   2   3   3
 27  26   0  24   0  18   0   3   3   3
```

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

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


mesh.dat(2/5)

```
  3   3   3
 27
  1   0   2   0   4   0  10   1   1   1
  2   1   3   0   5   0  11   2   1   1
  3   2   0   0   6   0  12   3   1   1
  4   0   5   1   7   0  13   1   2   1
  5   4   6   2   8   0  14   2   2   1
  6   5   0   3   9   0  15   3   2   1
  7   0   8   4   0   0  16   1   3   1
  8   7   9   5   0   0  17   2   3   1
  9   8   0   6   0   0  18   3   3   1
 10   0  11   0  13   1  19   1   1   2
 11  10  12   0  14   2  20   2   1   2
 12  11   0   0  15   3  21   3   1   2
 13   0  14  10  16   4  22   1   2   2
 14  13  15  11  17   5  23   2   2   2
 15  14   0  12  18   6  24   3   2   2
 16   0  17  13   0   7  25   1   3   2
 17  16  18  14   0   8  26   2   3   2
 18  17   0  15   0   9  27   3   3   2
 19   0  20   0  22  10   0   1   1   3
 20  19  21   0  23  11   0   2   1   3
 21  20   0   0  24  12   0   3   1   3
 22   0  23  19  25  13   0   1   2   3
 23  22  24  20  26  14   0   2   2   3
 24  23   0  21  27  15   0   3   2   3
 25   0  26  22   0  16   0   1   3   3
 26  25  27  23   0  17   0   2   3   3
 27  26   0  24   0  18   0   3   3   3
```



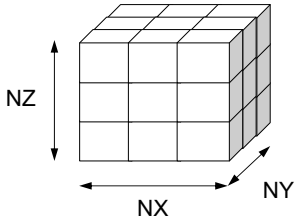
X,Y,Z方向の要素数

```
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)

```
  3   3   3
 27
  1   0   2   0   4   0  10   1   1   1
  2   1   3   0   5   0  11   2   1   1
  3   2   0   0   6   0  12   3   1   1
  4   0   5   1   7   0  13   1   2   1
  5   4   6   2   8   0  14   2   2   1
  6   5   0   3   9   0  15   3   2   1
  7   0   8   4   0   0  16   1   3   1
  8   7   9   5   0   0  17   2   3   1
  9   8   0   6   0   0  18   3   3   1
 10   0  11   0  13   1  19   1   1   2
 11  10  12   0  14   2  20   2   1   2
 12  11   0   0  15   3  21   3   1   2
 13   0  14  10  16   4  22   1   2   2
 14  13  15  11  17   5  23   2   2   2
 15  14   0  12  18   6  24   3   2   2
 16   0  17  13   0   7  25   1   3   2
 17  16  18  14   0   8  26   2   3   2
 18  17   0  15   0   9  27   3   3   2
 19   0  20   0  22  10   0   1   1   3
 20  19  21   0  23  11   0   2   1   3
 21  20   0   0  24  12   0   3   1   3
 22   0  23  19  25  13   0   1   2   3
 23  22  24  20  26  14   0   2   2   3
 24  23   0  21  27  15   0   3   2   3
 25   0  26  22   0  16   0   1   3   3
 26  25  27  23   0  17   0   2   3   3
 27  26   0  24   0  18   0   3   3   3
```

要素数:NX×NY×NZ

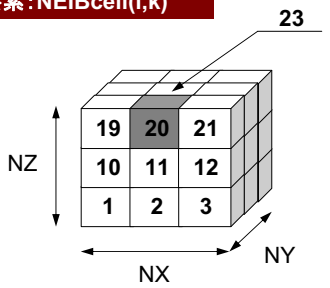



```
read (21,'(10i10)') NX , NY , NZ
read (21,'(10i10)') ICELTOT

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)

```
  3   3   3
 27
  1   0   2   0   4   0  10   1   1   1
  2   1   3   0   5   0  11   2   1   1
  3   2   0   0   6   0  12   3   1   1
  4   0   5   1   7   0  13   1   2   1
  5   4   6   2   8   0  14   2   2   1
  6   5   0   3   9   0  15   3   2   1
  7   0   8   4   0   0  16   1   3   1
  8   7   9   5   0   0  17   2   3   1
  9   8   0   6   0   0  18   3   3   1
 10   0  11   0  13   1  19   1   1   2
 11  10  12   0  14   2  20   2   1   2
 12  11   0   0  15   3  21   3   1   2
 13   0  14  10  16   4  22   1   2   2
 14  13  15  11  17   5  23   2   2   2
 15  14   0  12  18   6  24   3   2   2
 16   0  17  13   0   7  25   1   3   2
 17  16  18  14   0   8  26   2   3   2
 18  17   0  15   0   9  27   3   3   2
 19   0  20   0  22  10   0   1   1   3
 20  19  21   0  23  11   0   2   1   3
 21  20   0   0  24  12   0   3   1   3
 22   0  23  19  25  13   0   1   2   3
 23  22  24  20  26  14   0   2   2   3
 24  23   0  21  27  15   0   3   2   3
 25   0  26  22   0  16   0   1   3   3
 26  25  27  23   0  17   0   2   3   3
 27  26   0  24   0  18   0   3   3   3
```

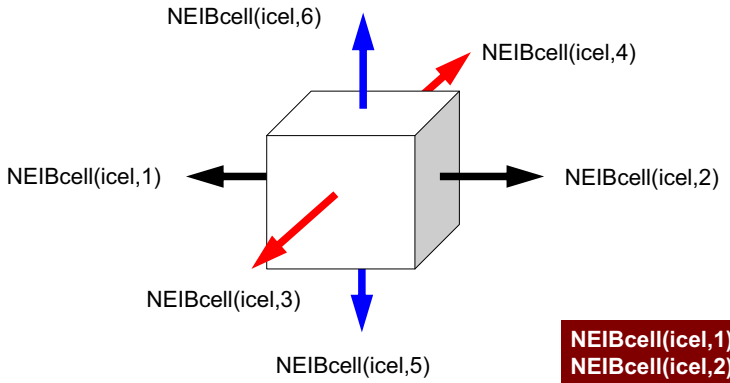
隣接要素:NEIBcell(i,k)




```
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

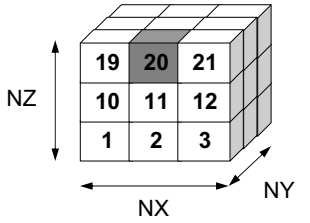


NEIBcell(icel,1)= icel – 1
NEIBcell(icel,2)= icel + 1
NEIBcell(icel,3)= icel – NX
NEIBcell(icel,4)= icel + NX
NEIBcell(icel,5)= icel – NX*NY
NEIBcell(icel,6)= icel + NX*NY

mesh.dat(5/5)

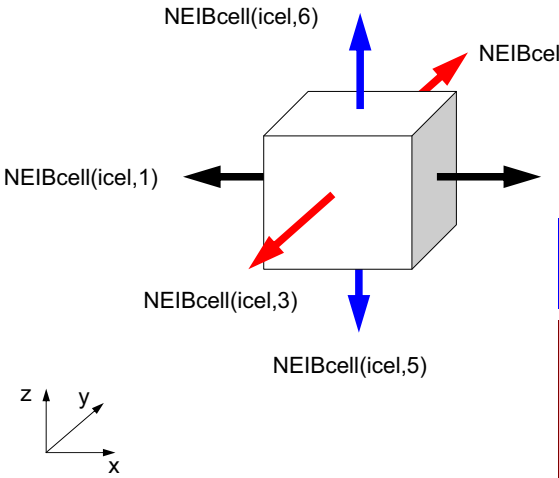
```
  3   3   3
 27
  1   0   2   0   4   0  10   1   1   1
  2   1   3   0   5   0  11   2   1   1
  3   2   0   0   6   0  12   3   1   1
  4   0   5   1   7   0  13   1   2   1
  5   4   6   2   8   0  14   2   2   1
  6   5   0   3   9   0  15   3   2   1
  7   0   8   4   0   0  16   1   3   1
  8   7   9   5   0   0  17   2   3   1
  9   8   0   6   0   0  18   3   3   1
 10   0  11   0  13   1  19   1   1   2
 11  10  12   0  14   2  20   2   1   2
 12  11   0   0  15   3  21   3   1   2
 13   0  14  10  16   4  22   1   2   2
 14  13  15  11  17   5  23   2   2   2
 15  14   0  12  18   6  24   3   2   2
 16   0  17  13   0   7  25   1   3   2
 17  16  18  14   0   8  26   2   3   2
 18  17   0  15   0   9  27   3   3   2
 19   0  20   0  22  10   0   1   1   3
 20  19  21   0  23  11   0   2   1   3
 21  20   0   0  24  12   0   3   1   3
 22   0  23  19  25  13   0   1   2   3
 23  22  24  20  26  14   0   2   2   3
 24  23   0  21  27  15   0   3   2   3
 25   0  26  22   0  16   0   1   3   3
 26  25  27  23   0  17   0   2   3   3
 27  26   0  24   0  18   0   3   3   3
```

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

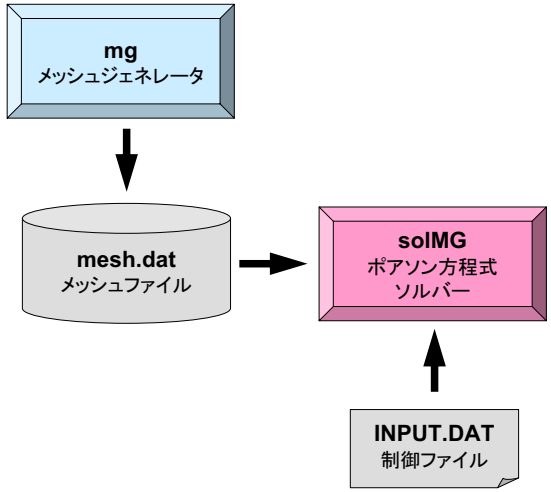


i= XYZ(icel,1)
j= XYZ(icel,2), k= XYZ(icel,3)
icel= (k-1)*NX*NY + (j-1)*NX + i

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

プログラム，必要なファイル等



プログラムの実行

制御データ「INPUT.DAT」の作成（1/2）

```
1.00e-0 1.00e-0 1.00e-0      DX/DY/DZ
1.00      1.0e-08            OMEGA, EPSICCG
1                            SOLVER  1:GS,2:ICCG,3:MG
100  100                     ITERtotCr/p
0.95                         CHANGElevel
```

- **DX, DY, DZ**
  - 各要素のX,Y,Z方向辺長さ
- **OMEGA**
  - 1.0に固定
- **EPSICCG**
  - 収束判定値

$$\frac{\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\|}{\|\mathbf{b}\|}<\varepsilon$$

プログラムの実行

制御データ「INPUT.DAT」の作成（2/2）

```
1.00e-0 1.00e-0 1.00e-0      DX/DY/DZ
1.00      1.0e-08            OMEGA, EPSICCG
1                            SOLVER  1:GS,2:ICCG,3:MG
100  100                     ITERtotCr/p
0.95                         CHANGElevel
```

- **SOLVER**
  - 1:GS（Gauss-Seidel），2：ICCG，3：MG
- **ITERtotCr，ITERtotCp**
  - **ITERtotCr**：Restriction（細⇒粗）の場合の反復回数
  - **ITERtotCp**：Prolongation（粗⇒細）の場合の反復回数
- **CHANGElevel**
  - 前の反復の残差との比を取って，この値より大きければ次のレベル（細，粗）へ行く

実 行

```
>$ cd <$S>/run
>$ qsub go.sh
```

go.sh

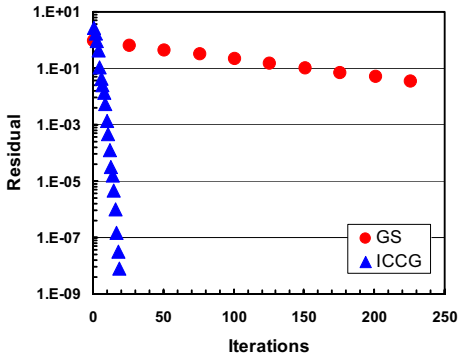
```
#@$-r mg-single
#@$-q lecture1
#@$-N 1
#@$-J T1
#@$-e err
#@$-o 064_cg.lst
#@$-lM 28GB
#@$-lT 00:15:00
#@$

cd $PBS_O_WORKDIR

./solMG
exit
```


計算例（反復回数，計算時間，T2K）

N	Gauss-Seidel	ICCG
8^3= 512	1,251 (0.03 sec)	19 (<0.01)
16^3= 4,096	5,350 (1.22 sec)	38 (0.01)
32^3= 32,768	22,100 (55.7)	73 (0.28)
64^3= 262,144		141 (6.59)
128^3=2,097,152		273 (105.4)

- ICCGの反復回数：概ね$N^{1/3}$に比例（8倍の問題規模で2倍）
  - 境界条件にもよるが
- 問題規模が1000倍：計算時間$1000^{4/3}=10^4$倍

計算例，収束履歴（N=8^3=512）



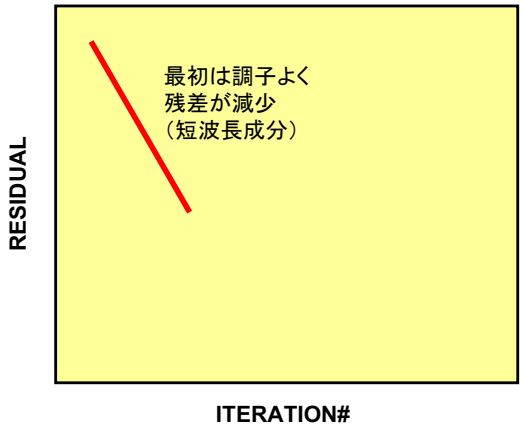
- 連立一次方程式の解法
  - Gauss-Seidel法
  - CG法（共役勾配法）
- 有限体積法によるアプリケーション
- **多重格子法のアルゴリズム**
- Geometrical Multigridの実例
  - 二重球殻内の自然対流，ポアソン方程式
  - 計算例

Multigridとは

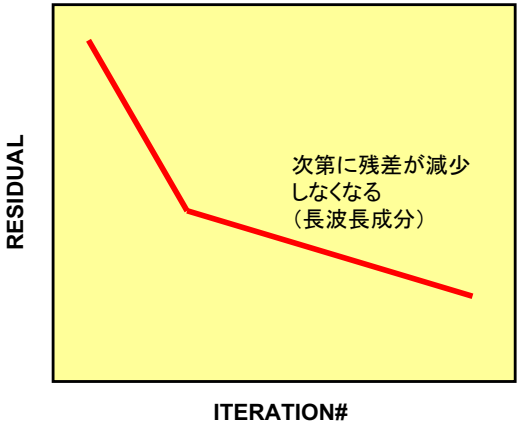
https://computation.llnl.gov/casc/

- 格子を使用して離散化された系に対して反復法を適用する場合，基本的に格子サイズと同じ波長を持った誤差が効率よく減衰する。
  - Gauss-Seidel，SOR
- しかしながら，長波長の誤差はなかなか減衰しない。

Gauss-Seidel法，SOR法の収束



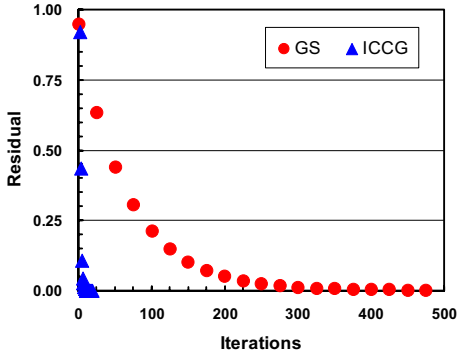
Gauss-Seidel法，SOR法の収束



Multigridとは

https://computation.llnl.gov/casc/

- 格子を使用して離散化された系に対して反復法を適用する場合，基本的に格子サイズと同じ波長を持った誤差が効率よく減衰する。
  - Gauss-Seidel，SOR



Multigridとは

https://computation.llnl.gov/casc/

- 格子を使用して離散化された系に対して反復法を適用する場合，基本的に格子サイズと同じ波長を持った誤差が効率よく減衰する。
  - Gauss-Seidel，SOR
- しかしながら，長波長の誤差はなかなか減衰しない。
- 通常，メッシュ数（問題規模）が多くなればそれだけ，収束までの反復回数が増加する。
  - 計算時間は「問題サイズ×反復回数増大効果」に比例するため，「Scalable（計算時間が問題サイズにのみ比例する）」とは言えない。
  - ICCG法の例

Multigridとは（続き）

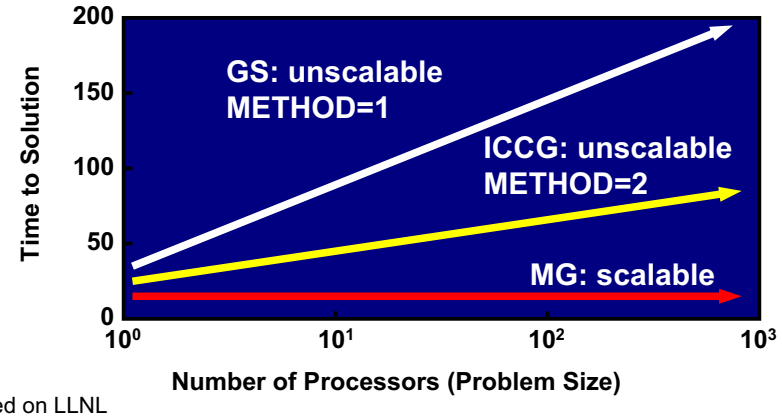
https://computation.llnl.gov/casc/

- Multigrid（多重格子）法とは，細かい格子と粗い格子を使用して，各波長の誤差を効率的に減衰させる手法である。粗い格子を使用すれば，長波長の誤差を減衰させることができる。
- Multigrid法では，各波長の誤差を一様に減衰させることが可能なため：
  - 収束が速く，かつscalable（反復までの収束回数が問題サイズに関わらず一様，したがって計算時間が問題サイズにのみ比例）
  - 大規模問題向け解法として注目されている
- 粗い格子で求めた答えを細かい格子に代入する・・・ということではない（そういう解法もあるが）。

Multigrid

Multigrid is scalable !!!

プロセッサ当たりの問題規模固定：Weak Scaling



Based on LLNL

Multigridの概要（1/5）

- Gauss-Seidel 法などの古典的反復法はメッシュサイズに相当する波長をもった誤差成分の減衰には適しているが，誤差の成分のうち，長い波長の成分は緩和を繰り返しても中々収束しない場合が多い。
- マルチグリッド法はこのような，長い波長の成分が粗い格子上で効率的に減衰するという考え方に基づいている。これについて 2 レベルの例を使用して説明する。

Multigridの概要(2/5)

細かい格子レベル（fine grid level）において以下の線形方程式を解くことを考える：

$$A_F\, u_F = f \qquad (1)$$

ここで A_C を粗い格子レベル（coarse grid level）における係数マトリクスとすると，粗い格子レベルにおける補正は以下のように記述される：

$$u_F^{(i+1)} = u_F^{(i)} + R_{C\Rightarrow F}\, A_C^{-1}\, R_{F\Rightarrow C}\,(f - A_F\, u_F^{(i)}) \qquad (2)$$

Multigridの概要(3/5)

ここで：

$R_{F\Rightarrow C}$ 細かい格子から粗い格子への補正オペレータ
（restriction operator，制限補間）

$R_{C\Rightarrow F}$ 粗い格子から細かい格子への補正オペレータ
（prolongation operator，延長補間）

このような補正計算によって，細かい格子で残差を計算し，それを粗い格子で補正し，その結果を細かい格子に補間して誤差を補正するというプロセスを構築できる。

$$R_{C\Rightarrow F} = c\left(R_{F\Rightarrow C}\right)^T, \quad c \in \mathbf{R}$$

cは「実数」

Multigridの概要(4/5)

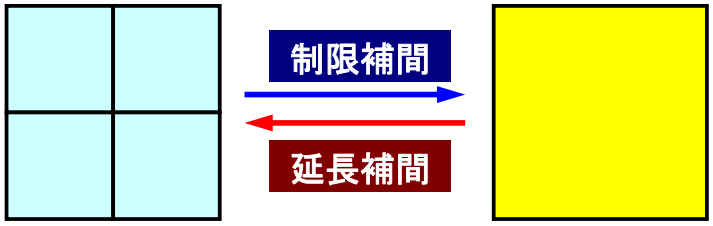
1. 線形化された方程式 $A_F\, u_F = f$ を細かい格子上で緩和し，結果を $u_F^{(i)} = S_F(A_F, f)$ とする。演算子 S_F (例えば Gauss-Seidel) は緩和演算子（*smoothing operator*）と呼ばれる。
2. 細かい格子上で残差 $r_F = f - A_F\, u_F^{(i)}$ を求める。
3. 制限補間オペレータ $R_{F\Rightarrow C}$ によって，細かい格子上での残差を粗い格子に補間する：$r_C = R_{F\Rightarrow C}\, r_F$
4. 方程式 $A_C\, u_C = r_C$ （修正量に関する方程式なので「修正方程式」と呼ぶ）を粗い格子で解く。
5. 粗い格子上での解 u_C から，延長補間オペレータ $R_{C\Rightarrow F}$ によって，細かい格子における修正量 $\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C$ を求める。
6. 細かい格子での解を修正量によって更新する：$u_F^{(i+1)} = u_F^{(i)} + \Delta u_F^{(i)}$
7. 残差が基準値以下になるまで，以上のプロセスを繰り返す。

制限補間・延長補間演算子とは？

$R_{F\Rightarrow C}$ 細かい格子から粗い格子への補正オペレータ
（restriction operator，制限補間）

$R_{C\Rightarrow F}$ 粗い格子から細かい格子への補正オペレータ
（prolongation operator，延長補間）

通常は $R_{F\Rightarrow C}=[1,1,1,\ldots.]^T$ のようなオペレータを使用



$$u_F^{(i)} = S_F\,(A_F, f)$$

1. 線形化された方程式 $A_F u_F = f$ を細かい格子上で緩和し，結果を $u_F^{(i)} = S_F(A_F, f)$ とする。演算子 S_F (例えば Gauss-Seidel) は緩和演算子（*smoothing operator*）と呼ばれる。

$$r_F = f - A_F u_F^{(i)}$$

2. 細かい格子上で残差 $r_F = f - A_F u_F^{(i)}$ を求める。

$$r_C = R_{F\Rightarrow C}\, r_F = R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

3. 制限補間オペレータ $R_{F\Rightarrow C}$ によって，細かい格子上での残差を粗い格子に補間する：$r_C = R_{F\Rightarrow C}\, r_F$

4. 方程式 $A_C u_C = r_C$ （修正量に関する方程式なので「修正方程式」と呼ぶ）を粗い格子で解く。

$$A_C u_C = r_C$$

$$u_C = A_C^{-1} r_C = A_C^{-1} R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

5. 粗い格子上での解 u_C から，延長補間オペレータ $R_{C\Rightarrow F}$ によって，細かい格子における修正量 $\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C$ を求める。

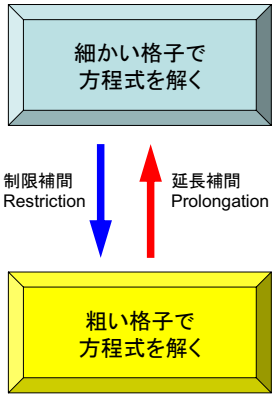
$$\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C = R_{C\Rightarrow F} A_C^{-1} R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

6. 細かい格子での解を修正量によって更新する

$$u_F^{(i+1)} = u_F^{(i)} + \Delta u_F^{(i)} = u_F^{(i)} + R_{C\Rightarrow F} A_C^{-1} R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

3ページ前の（2）

Multigridのアルゴリズム：2レベルの例

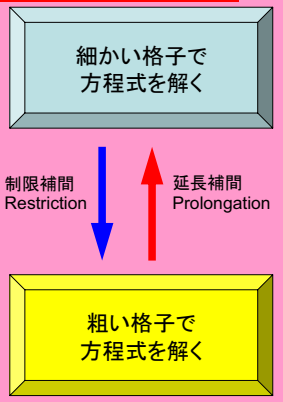


1. 線形化された方程式 $A_F u_F = f$ を細かい格子上で緩和し，結果を $u_F^{(i)} = S_F(A_F, f)$ とする。演算子 S_F (例えば Gauss-Seidel) は緩和演算子（*smoothing operator*）と呼ばれる。
2. 細かい格子上で残差 $r_F = f - A_F u_F^{(i)}$ を求める。
3. 制限補間オペレータ $R_{F\Rightarrow C}$ によって，細かい格子上での残差を粗い格子に補間する：$r_C = R_{F\Rightarrow C}\, r_F$
4. 方程式 $A_C u_C = r_C$ （修正量に関する方程式なので「修正方程式」と呼ぶ）を粗い格子で解く。
5. 粗い格子上での解 u_C から，延長補間オペレータ $R_{C\Rightarrow F}$ によって，細かい格子における修正量 $\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C$ を求める。
6. 細かい格子での解を修正量によって更新する：$u_F^{(i+1)} = u_F^{(i)} + \Delta u_F^{(i)}$
7. 残差が基準値以下になるまで，以上のプロセスを繰り返す。

$$u_F^{(i+1)} = u_F^{(i)} + R_{C\Rightarrow F} A_C^{-1} R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

Multigridのアルゴリズム：2レベルの例

do iter= 1, ITERmax

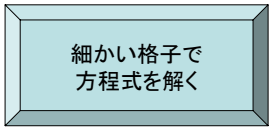


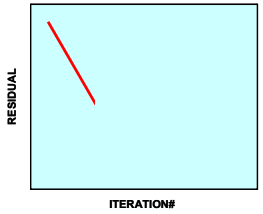
enddo

1. 線形化された方程式 $A_F u_F = f$ を細かい格子上で緩和し，結果を $u_F^{(i)} = S_F(A_F, f)$ とする。演算子 S_F (例えば Gauss-Seidel) は緩和演算子（*smoothing operator*）と呼ばれる。
2. 細かい格子上で残差 $r_F = f - A_F u_F^{(i)}$ を求める。
3. 制限補間オペレータ $R_{F\Rightarrow C}$ によって，細かい格子上での残差を粗い格子に補間する：$r_C = R_{F\Rightarrow C}\, r_F$
4. 方程式 $A_C u_C = r_C$ （修正量に関する方程式なので「修正方程式」と呼ぶ）を粗い格子で解く。
5. 粗い格子上での解 u_C から，延長補間オペレータ $R_{C\Rightarrow F}$ によって，細かい格子における修正量 $\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C$ を求める。
6. 細かい格子での解を修正量によって更新する：$u_F^{(i+1)} = u_F^{(i)} + \Delta u_F^{(i)}$
7. 残差が基準値以下になるまで，以上のプロセスを繰り返す。

$$u_F^{(i+1)} = u_F^{(i)} + R_{C\Rightarrow F} A_C^{-1} R_{F\Rightarrow C}\,(f - A_F u_F^{(i)})$$

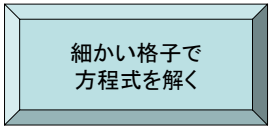
細かい格子で厳密に方程式を解く必要は無い

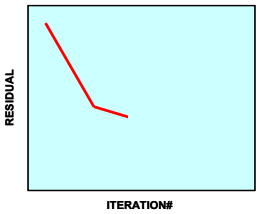




- 何回か反復する

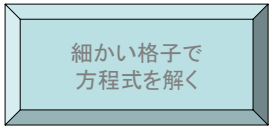
細かい格子で厳密に方程式を解く必要は無い

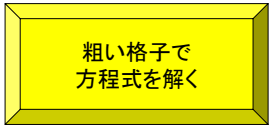




- 長波長成分が支配的になってきたら

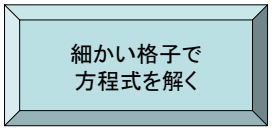
細かい格子で厳密に方程式を解く必要は無い

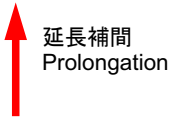


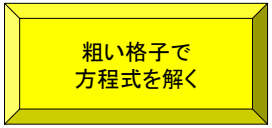


- 粗い格子に移る

細かい格子で厳密に方程式を解く必要は無い

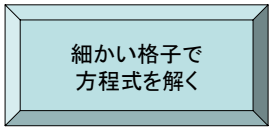


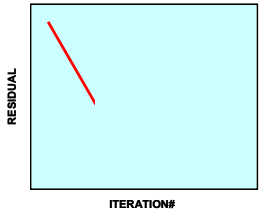




- 細かい格子の解を修正，補正して

細かい格子で厳密に方程式を解く必要は無い



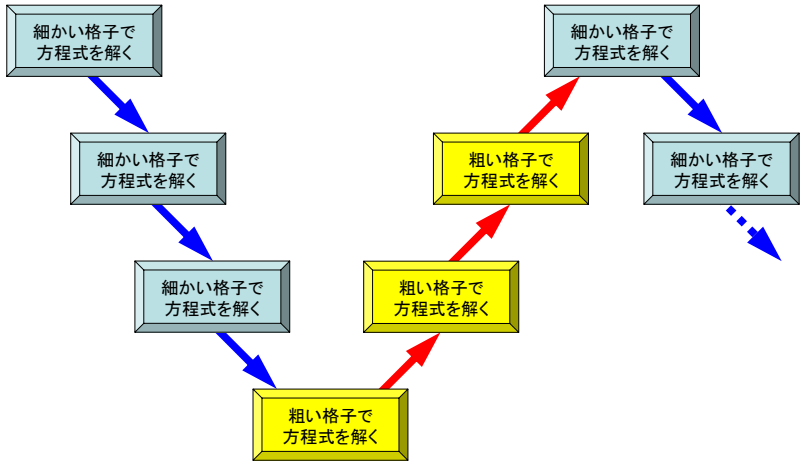


- また，細かい格子で何回か反復する

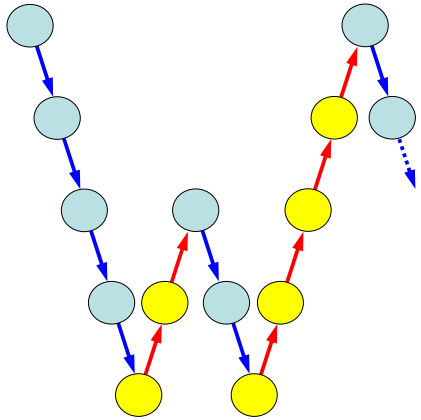
Multigridの概要(5/5)

- この 2 レベルのプロセスを再帰的に多段階に適用することによって，マルチグリッドの「V-Cycle」が構築可能である。
- 「V-Cycle」の各レベルの計算が適切に実施されれば，誤差のあらゆる長さの波長をもった成分を一様に減衰させることができる。
- 計算時間が問題規模に比例するいわゆる「scalable」な手法の実現が可能である。

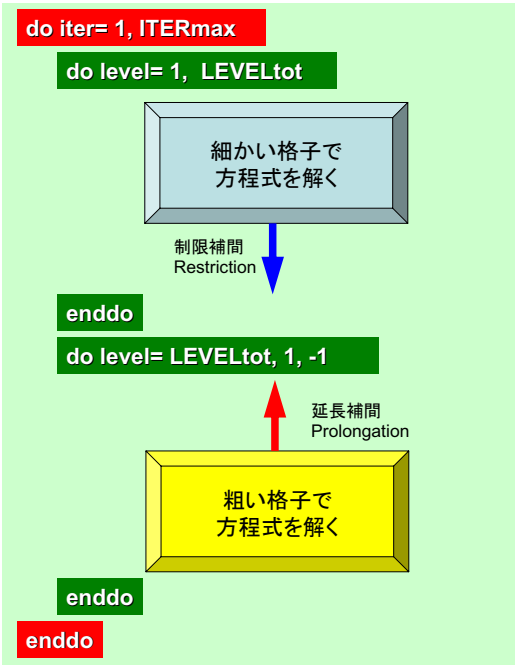
これを多段階で再帰的に実施するのがMultigrid(V-Cylcleの例)



これを多段階で再帰的に実施するのがMultigrid(W-Cylcleの例)



V-Cylcleの例



計算例（反復回数，計算時間，T2K）

N	Gauss-Seidel	ICCG	MG
8^3= 512	1,251 （0.03 sec）	19 （<0.01）	4 （<0.01）
16^3= 4,096	5,350 （1.22 sec）	38 （0.01）	6 （0.02）
32^3= 32,768	22,100 （55.7）	73 （0.28）	9 （0.24）
64^3= 262,144		141 （6.59）	12 （2.42）
128^3=2,097,152		273 （105.4）	15 （23.8）

Geometric/Algebraic Multigrid

- 「粗い格子」をどうやって作るかによって，GeometricとAlgebraicの二通りの手法がある。
- Geometricは直感的に理解しやすい。
  - メッシュの階層構造を使用：差分，Adaptive Mesh
- Algebraicは，マトリクスのコネクティビティに基づいているため，とっつきにくい感じがするが，はるかにフレキシビリティに富んでいる。

Multigridの特徴，問題点

- Gauss-Seidelのような簡単な手法の組み合わせでICCG法より速く計算ができる場合がある
  - 詳しくはプログラムをご覧ください（FORTRANだけど・・・）
- 問題に依存：どんな問題でも解けるわけではない
  - ポアソン方程式（拡散型，1節点1自由度）が得意
  - パラメータが多い（場合が多い）：後述
- 様々な拡張
  - 非線形問題：Newton-Krylov
  - Krylov系反復解法の前処理：MGCG
    - 共役勾配法の前処理手法として，MGを用いる
  - Semi-Coarsening: 不均質，異方性

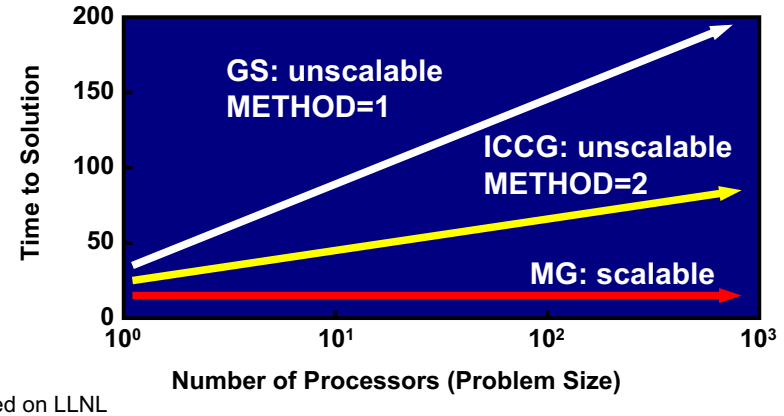
キーワード

- 「Scalable」とは
  - 計算時間が問題規模のみに比例
    - 反復法：問題規模が大きくなっても反復回数が変わらない
    - 問題規模に比例した数のプロセッサを使用すれば計算時間不変
  - 全体問題規模を固定して，m個のプロセッサを使用して，m分の1で計算できる・・・というときも「scalable」と言う：strong scaling
- 誤差の減衰
  - 短波長，長波長
  - メッシュ長さに対応した誤差が効率良く減衰する
- 緩和演算子：Smoother
- 制限補間（Restriction），延長補間（Prolongation）
- 修正方程式
- Vサイクル，Wサイクル

Multigrid

Multigrid is scalable !!!

プロセッサ当たりの問題規模固定：Weak Scaling



Based on LLNL

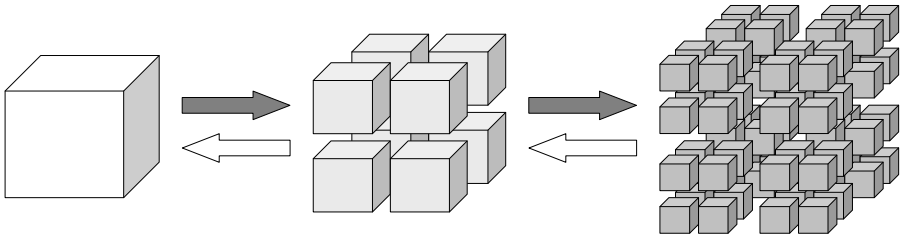
プログラムの概要

http://nkl.cc.u-tokyo.ac.jp/07w/CW08/2007-11-21-CW08.pdf
http://nkl.cc.u-tokyo.ac.jp/07w/CW09/2007-11-28-CW09.pdf

- 三次元差分格子
- ポアソン方程式

- Gauss-Seidel
- ICCG
- MG
  - 緩和演算子：Gauss-Seidel法
  - Vサイクル

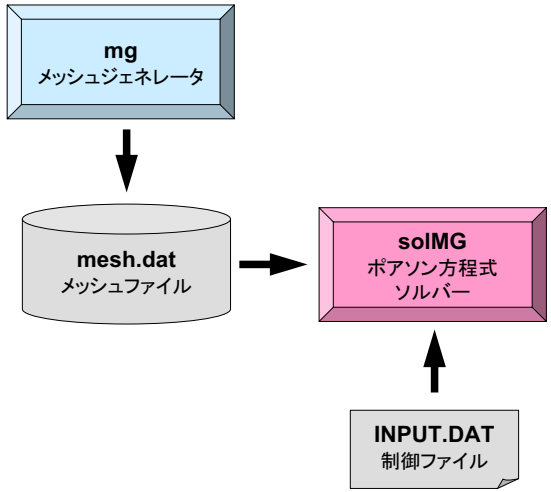
8（2^3）個の「子」メッシュが集まって1個の「親」メッシュを形成する



最も粗いメッシュは1×1×1：
全体が1メッシュ

プログラムの実行

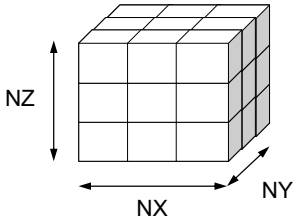
プログラム，必要なファイル等



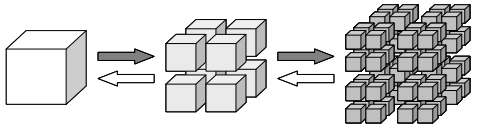
プログラムの実行
メッシュ生成

```
$> cd <$cur>/W3/mg/run
$> ./mg
   4
$> 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



- ただし・・・
  - NX，NY，NZは2のべき乗でなければならない。メッシュ生成時に特にエラーは出ないがマルチグリッドの計算ができない。



プログラムの実行
制御データ「INPUT.DAT」の作成（1/2）

```
1.00e-0 1.00e-0 1.00e-0      DX/DY/DZ
1.00      1.0e-08            OMEGA, EPSICCG
1                            SOLVER  1:GS,2:ICCG,3:MG
100  100                     ITERtotCr/p
0.95                         CHANGElevel
```

- **DX, DY, DZ**
  - 各要素のX,Y,Z方向辺長さ
- **OMEGA**
  - =1.0：Gauss-Seidel，変えることは可能
- **EPSICCG**
  - 収束判定値

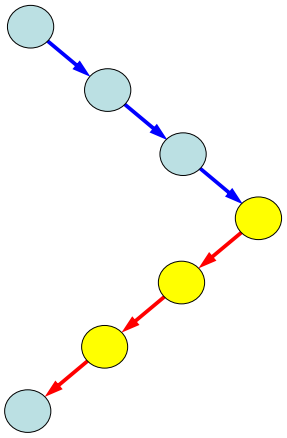
プログラムの実行
制御データ「INPUT.DAT」の作成（2/2）

```
1.00e-0 1.00e-0 1.00e-0      DX/DY/DZ
1.00      1.0e-08            OMEGA, EPSICCG
1                            SOLVER  1:GS,2:ICCG,3:MG
100  100                     ITERtotCr/p
0.95                         CHANGElevel
```

- **SOLVER**
  - 1:GS（Gauss-Seidel），2：ICCG，3：MG
- **ITERtotCr，ITERtotCp**
  - **ITERtotCr**：Restriction（細⇒粗）の場合の反復回数
  - **ITERtotCp**：Prolongation（粗⇒細）の場合の反復回数
- **CHANGElevel**
  - 前の反復の残差との比を取って，この値より大きければ次のレベル（細，粗）へ行く

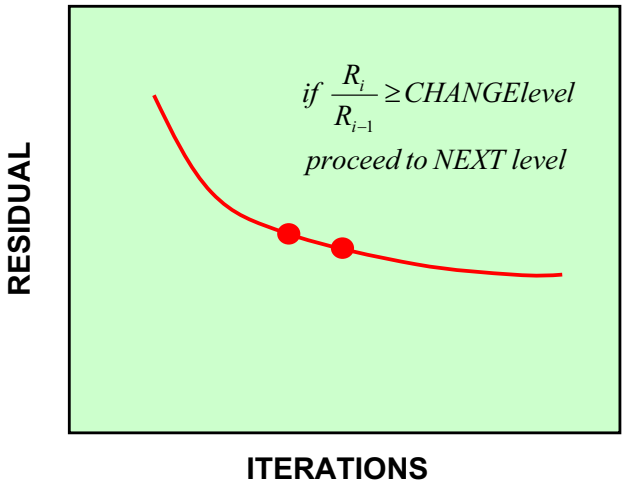
ITERtotCr，ITERtotCp

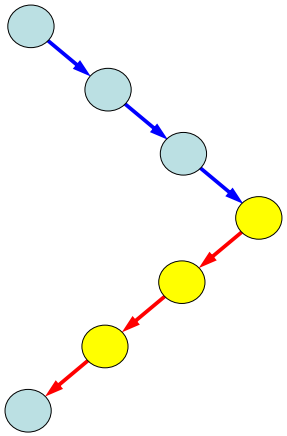
```
do iter_out= 1, ITERmax

  Restriction
  do iterk= 1, ITERtotCr

    (RELAX+RESTRICTION)

  enddo

  Prolongation
  do iterk= 1, ITERtotCp
      (RELAX+PROLONGATION)

  enddo

enddo
```



CHANGElevel

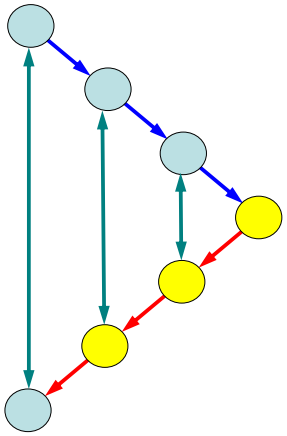
長波長成分が支配的な場合の指標





理論的な問題点

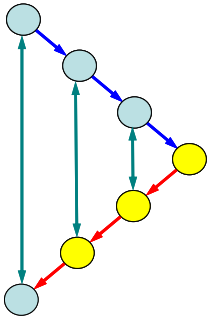
- 問題が正定となるには，RestrictionとProlongationで同じ回数の反復が必要
  - 緩和計算の順番も逆にしなければならない
- 実際はそのようにはなっていないが，正定性を保った場合よりも速く収束する場合がある



緩和計算の順番

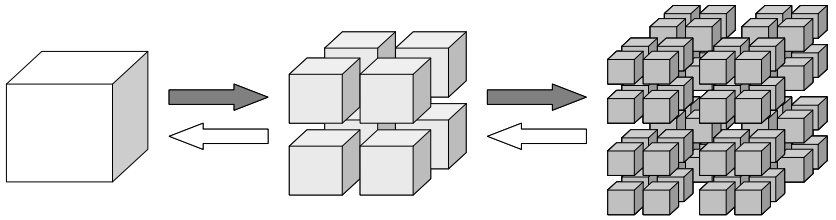
idir=1：制限補間，idir=-1：延長補間

```
 if (idir.eq.1) then
   do icel= levMGcelINDEX(lev-1)+1, levMGcelINDEX(lev)
     XPREV= XMG(icel)
     RF= BMG(icel) - WAcoefMG(1,icel)*XMG(NEIBcellMG(1,icel))      &
&                  - WAcoefMG(2,icel)*XMG(NEIBcellMG(2,icel))      &
&                  - WAcoefMG(3,icel)*XMG(NEIBcellMG(3,icel))      &
&                  - WAcoefMG(4,icel)*XMG(NEIBcellMG(4,icel))      &
&                  - WAcoefMG(5,icel)*XMG(NEIBcellMG(5,icel))      &
&                  - WAcoefMG(6,icel)*XMG(NEIBcellMG(6,icel))
     XMG(icel)= (RF*DDMG(icel)-XPREV)*OMEGA + XPREV
   enddo
  else
   do icel= levMGcelINDEX(lev), levMGcelINDEX(lev-1)+1, -1
     XPREV= XMG(icel)
     RF= BMG(icel) - WAcoefMG(1,icel)*XMG(NEIBcellMG(1,icel))      &
&                  - WAcoefMG(2,icel)*XMG(NEIBcellMG(2,icel))      &
&                  - WAcoefMG(3,icel)*XMG(NEIBcellMG(3,icel))      &
&                  - WAcoefMG(4,icel)*XMG(NEIBcellMG(4,icel))      &
&                  - WAcoefMG(5,icel)*XMG(NEIBcellMG(5,icel))      &
&                  - WAcoefMG(6,icel)*XMG(NEIBcellMG(6,icel))
     XMG(icel)= (RF*DDMG(icel)-XPREV)*OMEGA + XPREV
   enddo
 endif
```



多重格子法のプログラム

- 準備が大変

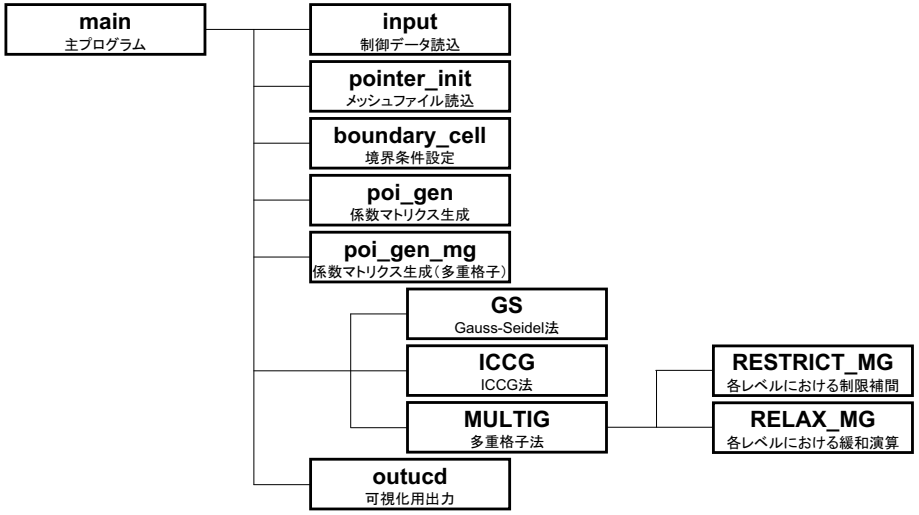


$$r_C = R_{F \Rightarrow C}\, r_F$$

$$A_C\, u_C = r_C$$

$$\Delta u_F^{(i)} = R_{C \Rightarrow F}\, u_C$$

多重格子法のプログラム <$S>/src



計算例（反復回数，計算時間，T2K）

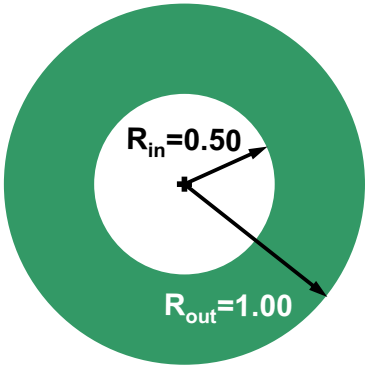
N	Gauss-Seidel	ICCG	MG
8^3= 512	1,251 （0.03 sec）	19 （<0.01）	4 （<0.01）
16^3= 4,096	5,350 （1.22 sec）	38 （0.01）	6 （0.02）
32^3= 32,768	22,100 （55.7）	73 （0.28）	9 （0.24）
64^3= 262,144		141 （6.59）	12 （2.42）
128^3=2,097,152		273 （105.4）	15 （23.8）

参考文献

- 荒川（1994）「数値流体工学」，東京大学出版会
  - 「高速解法」の一例として紹介されている
- Briggs, W.L., Henson, V.E. and McCormick, S.F. (2000) A Multigrid Tutorial Second Edition, SIAM
- Trottemberg, U., Oosterlee, C. and Schüller, A. (2001) Multigrid, Academic Press

- 連立一次方程式の解法
  - Gauss-Seidel法
  - CG法（共役勾配法）
- 有限体積法によるアプリケーション
- 多重格子法のアルゴリズム
- **Geometrical Multigridの実例**
  - **二重球殻内の自然対流，ポアソン方程式**
  - **計算例**

計算対象：二重球殻内領域における自然対流問題



- 地球科学分野ではよく出てくる形状
  - マントル，コアのダイナミックス
  - 大気，海洋モデル
- 境界条件
  - r= R_{in}
    - u=v=w=0, T=1
  - r= R_{out}
    - u=v=w=0, T=0
  - 体積発熱

半陰解法による圧力補正スキーム

- 非圧縮性流体
  - 連続の式，質量保存則
  - 運動量保存則（Navier-Stokes方程式）
  - エネルギー保存則
- 連続の式から圧力補正量に関するポアソン方程式が得られる
  - 計算時間がかかる
- 並列MGCG法によるポアソン方程式解法
  - マルチグリッド（MG）前処理付きCG法
  - Vサイクル

非圧縮性流れ：支配方程式

- 最大速度（音速で無次元化）＜0.30 の流れ
  - 密度変化を無視できる。
  - 空気：約100m/sec=360km/h
- エネルギー方程式と運動量，質量保存式が直接カップリングしなくなる・・・
  - 状態方程式も無くなる
- 簡単になったようだが，アルゴリズム上はそうではない。
  - 圧力が運動量方程式の勾配項としてしか現われなくなる。
  - 質量保存則は速度のみの関数となる。
  - 三次元の場合，未知数4つ（u,v,w,P），方程式は4つだがPが陽な未知数ではない。

非圧縮性流れ：支配方程式

- 連続の式，質量保存式

$$\nabla \cdot \mathbf{u} = 0$$

- 運動量保存式，Navier-Stokes方程式

$$\frac{\partial \mathbf{u}}{\partial t} + \nabla \cdot (\mathbf{uu}) + \nabla p - \frac{1}{\mathrm{Re}} \Delta \mathbf{u} = 0$$

- 自然対流の場合はこれにエネルギー保存則が入るがここでは省略

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$\mathbf{u}^{(n+1)}$が求めるべき速度場
質量保存則を満たす

$$\nabla\cdot\mathbf{u}^{(n+1)}=0$$

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$$\frac{\mathbf{u}'-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

中間的な速度場 **u'**
既知の（**(n)**時間）速度場，
圧力場に関して陽的に計算可能
質量保存則を満たすとは限らない

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$$\frac{\mathbf{u}'-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

中間的な速度場 **u'**
既知の（**(n)**時間）速度場，
圧力場に関して陽的に計算可能
質量保存則を満たすとは限らない

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$$\frac{\mathbf{u}'-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

差をとる

$$\mathbf{u}^{(n+1)}-\mathbf{u}'=-\left[\nabla\left(p^{(n+1)}-p^{(n)}\right)\right]\Delta t$$

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$$\frac{\mathbf{u}'-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

↓ 差をとる

$$\mathbf{u}^{(n+1)}-\mathbf{u}'=-\left[\nabla\left(p^{(n+1)}-p^{(n)}\right)\right]\Delta t$$

↓

両辺を微分する，ただし：$p^{(n+1)}-p^{(n)}=\frac{1}{\Delta t}\phi$

$$\nabla\cdot\mathbf{u}^{(n+1)}=0$$

非圧縮性流れ：解法

- 速度を陽的に，圧力を陰的に解く
  - MAC，SMAC法，圧力補正法などとして知られている方法

$$\frac{\mathbf{u}^{(n+1)}-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n+1)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

$$\frac{\mathbf{u}'-\mathbf{u}^{(n)}}{\Delta t}+\nabla\cdot\left(\mathbf{u}^{(n)}\mathbf{u}^{(n)}\right)+\nabla p^{(n)}-\frac{1}{\mathrm{Re}}\Delta\mathbf{u}^{(n)}=0$$

↓ 差をとる

$$\mathbf{u}^{(n+1)}-\mathbf{u}'=-\left[\nabla\left(p^{(n+1)}-p^{(n)}\right)\right]\Delta t$$

↓

両辺を微分する，ただし：$p^{(n+1)}-p^{(n)}=\frac{1}{\Delta t}\phi$

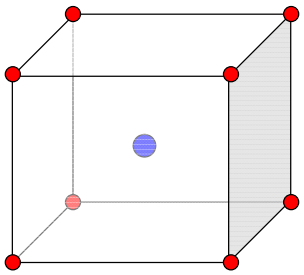
$$\nabla\cdot\mathbf{u}^{(n+1)}=0$$

$\Delta\phi=\nabla\cdot\mathbf{u}'$ 圧力補正量 ϕ に関するポアソン方程式

スタガード格子 （Staggered Grid）

速度と圧力を異なった点で定義する
離散化が広く使用されている

- 頂点
  - 速度成分
  - 温度
- 要素中心
  - 圧力
  - 圧力補正項
  - 材料物性



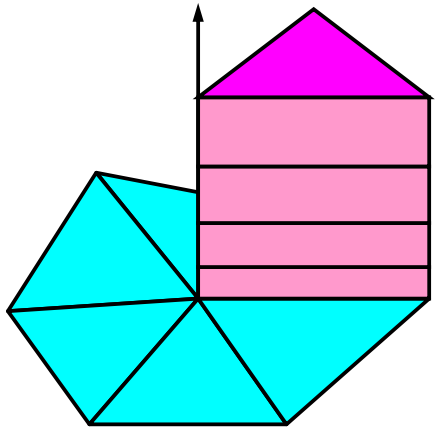
- ガウス＝グリーン型有限体積法

ポアソン方程式に対する並列マルチグリッド法

- 前処理付きCG法
- Vサイクル
- ガウス=ザイデル法によるスムージング：緩和演算子
- 多段階通信テーブル
- FORTRAN90＋MPI

半構造的三角柱メッシュ

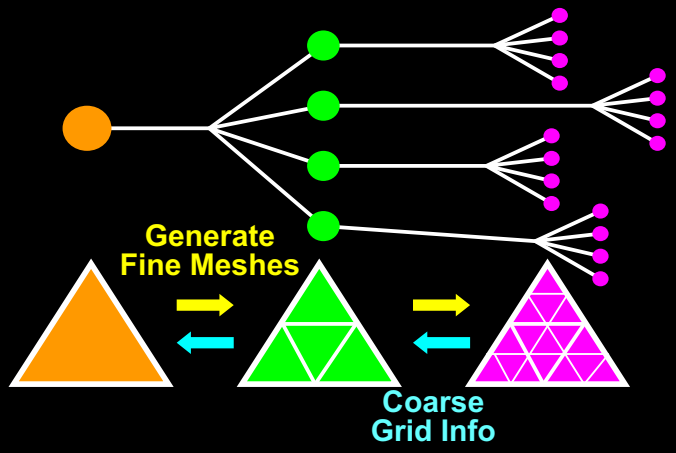
- 球面表面に発生させた三角形を底面として発生。
- 境界近く（境界層）に細かいメッシュを発生させることも可能。
- 三角形メッシュ
  - flexible



20面体を細分化することによってメッシュを生成する

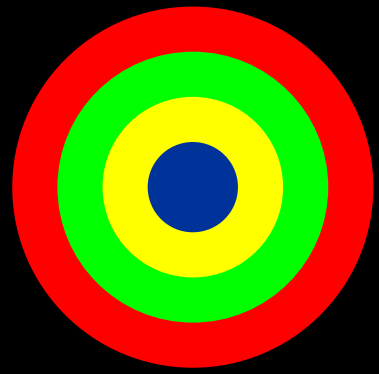
Level 0	Level 1	Level 2	Level 3	Level 4
12 nodes	42 nodes	162 nodes	642 nodes	2,562 nodes
20 tri's	80 tri's	320 tri's	1,280 tri's	5,120 tri's

メッシュ生成時の親子関係をそのままマルチグリッドに利用する



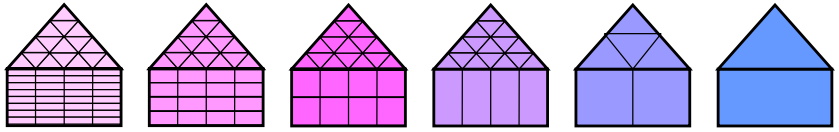
領域分割

- 半径方向のみに分割
  - 半構造的なメッシュ体系を保持



Semi-Coarsening

- 制限補間を実施する際には，まず，半径方向についてメッシュを粗くしていき，半径方向のメッシュ数が1になったら，球表面の三角形を正20面体まで粗くしていく。
  - 半径方向メッシュ数＝1，三角形数＝20，合計メッシュ数＝20となったところでシリアル計算を実施
- 延長補間についてはこの逆の順序で実施する。



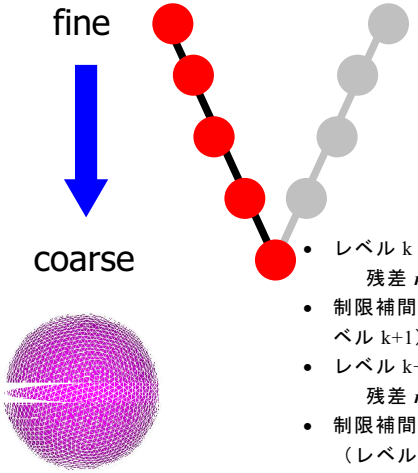
MG前処理付きCGソルバー

```
Compute {r}= {b} - [A]{x} for initial guess
for i= 1, 2, ...
    solve [M]{z}= {r}
    ρ_{i-1}= {r}{z}
    if i= 1
      {p}= {z}
     else
      β_{i-1}= ρ_{i-1} / ρ_{i-2}
      {p}= {z} + β_{i-1}{p}
    endif
    {q}= [A]{p}
    α_i= ρ_{i-1} / {p}{q}
    {x}= {x} + α_i {p}
    {r}= {r} - α_i {p}
    check convergence; continue if necessary
end
```

Multigrid Preconditioning

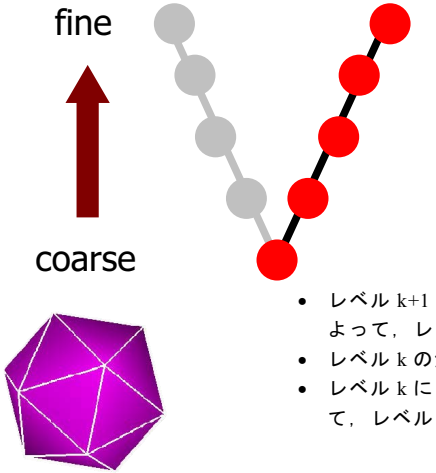
この部分をMultigridを使って解く

Vサイクルマルチグリッド Restriction（制限補間）



- レベル k において方程式 $A_k\ u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\ u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\ r_k$
- レベル k+1 において方程式 $A_{k+1}\ u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\ u_{k+1}$ を求める。
- 制限補間オペレータ $R_{k+1 \Rightarrow k+2}$ によって，r_{k+1} を一段階粗い格子（レベル k+2）に補間する：$f_{k+2} = R_{k+1 \Rightarrow k+2}\ r_{k+1}$

Vサイクルマルチグリッド Prolongation（延長補間）



- レベル k+1 における解 u_{k+1} から，延長補間オペレータ $R_{k+1 \Rightarrow k}$ によって，レベル k における修正量 $\Delta u_k = R_{k+1 \Rightarrow k}\ u_{k+1}$ を求める。
- レベル k の解を修正量によって更新する：$u_k = u_k + \Delta u_k$
- レベル k における解 u_k から，延長補間オペレータ $R_{k \Rightarrow k-1}$ によって，レベル k-1 における修正量 $\Delta u_{k-1} = R_{k \Rightarrow k-1}\ u_k$ を求める。

制限補間・延長補間演算子

$$R_{C\Rightarrow F} = c\left(R_{F\Rightarrow C}\right)^T, \quad c \in \mathbf{R}$$

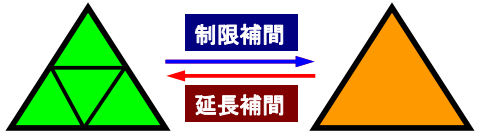
c は「実数」

制限補間 $\{r_C\} = [R_{F\Rightarrow C}]\{r_F\}$

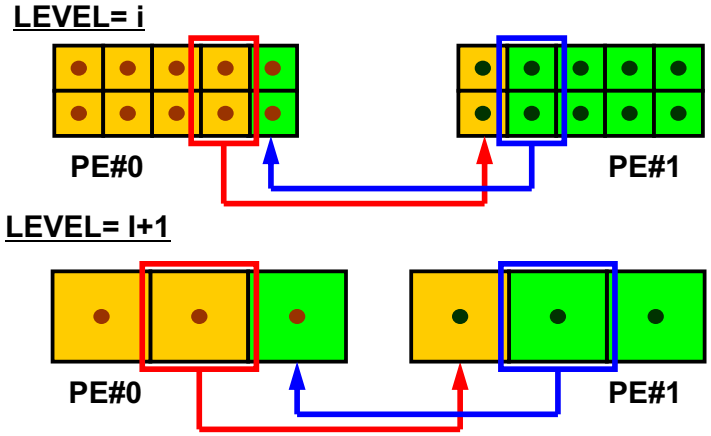
$$\{r_C\} = [1 \quad 1 \quad 1 \quad 1]\{r_{f1} \quad r_{f2} \quad r_{f3} \quad r_{f4}\}^T$$

延長補間 $\{\Delta u_F\} = [R_{C\Rightarrow F}]\{u_C\}$ $c = 1$

$$\{\Delta u_{f1} \quad \Delta u_{f2} \quad \Delta u_{f3} \quad \Delta u_{f4}\} = [1 \quad 1 \quad 1 \quad 1]^T\{u_C\}$$

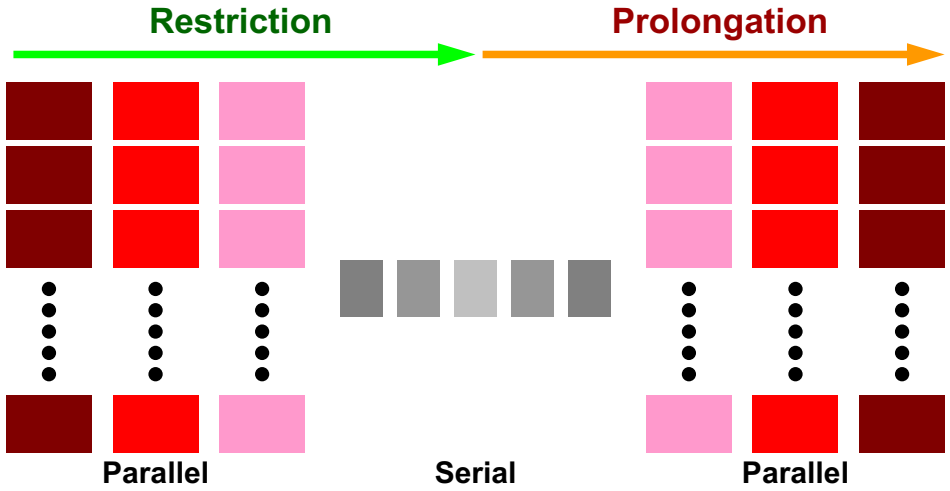


多段階通信テーブル



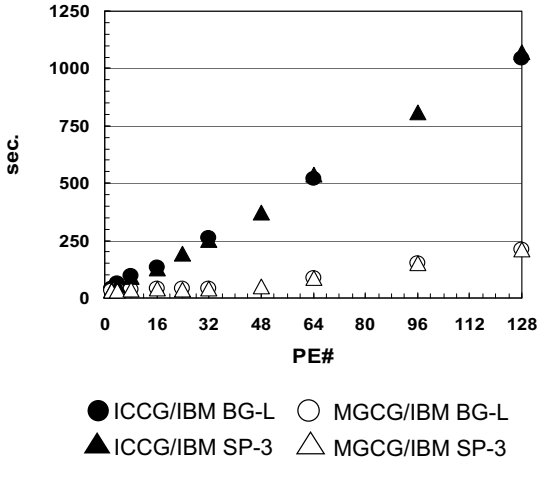
並列，シリアル計算の切り替え

粗いレベルではシリアル計算に切り替える
本プログラムでは最も粗いレベルのみシリアル計算



計算結果

1280x340=435,000 cells/PE
up to 56M DOF on 128 PE's



- 1PEあたりの問題サイズ固定
- 「Scalable」であれば，PE数が増加，すなわち問題サイズが大きくなっても，計算時間が変わらないはず。
- ICCGでは，全体問題サイズが大きくなると，反復回数が増加するため，計算時間も増加する。
- MGCGでは反復回数が変わらないため，計算時間はほとんど変わらない。
  - BG-L：IBM BG/L
  - SP-3：IBM SP3

実際に計算してみよう：ICCG法との比較

```
$> cd <$P>/src-iccg
$> make
$> ls ../run/soliccg
```

```
$> cd <$P>/src-mgcg
$> make
$> ls ../run/solmgcg
```

```
$> cd <$P>/run
$> qsub go.sh
```

- 問題
  - R_{in}= 0.50，Δr= 0.01（play.p_XXXX）
  - 320面体，1領域あたり1000層（320,000メッシュ）
  - PE数を変えると，1領域あたりのメッシュ数を320,000に固定して計算を実施可能（R_{out}は変化）

制御データ（input.dat）（赤字の部分が可変）

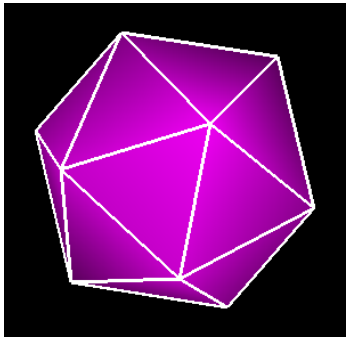
```
test.grid.320
r0.case
T                       : COARSEgrid
4                       : ILEVcoarse
APPROX                  : COARSEgrid METHOD
10                      : iterSSOR
1                       : SOLFLAG= 1 (1:ICCG, 2:MCCG)
0                       : NCOLORtot
10000                     : DEPTHsgl
1.d0                      : SIGMA= 1.d0
1.d-6                     : EPS  = 1.d-8
1000                        : ITERmax=
2                         : iSLEV_MG_type
uniform                   : BOUNDARY conditions
0.80d0, 0.80d0            : CMG1, CMG2
1.0d-24                  : EPSMG
1.0d0                     : EPScoarse
2                         : COARSEsolver_flag
10000, 5, 5               : ITERc/r/p/MAX
1                         : PEsmpTOT
```


制御データ（input.dat）（BC）

```
test.grid.320
r0.case
T                       : COARSEgrid
4                       : ILEVcoarse
APPROX                  : COARSEgrid METHOD
10                      : iterSSOR
1                       : SOLFLAG= 1 (1:ICCG, 2:MCCG)
0                       : NCOLORtot
10000                     : DEPTHsgl
1.d0                      : SIGMA= 1.d0
1.d-6                     : EPS  = 1.d-8
1000                        : ITERmax=
2                         : iSLEV_MG_type
uniform                   : BOUNDARY conditions
0.80d0, 0.80d0            : CMG1, CMG2
1.0d-24                  : EPSMG
1.0d0                     : EPScoarse
2                         : COARSEsolver_flag
10000, 5, 5               : ITERc/r/p/MAX
1                         : PEsmpTOT
```

境界条件
- uniform
- Xmin
- Xmax
- Ymin
- Ymax
- Zmin
- Zmax

境界条件の指定方法
$R=R_{max}$での固定境界条件



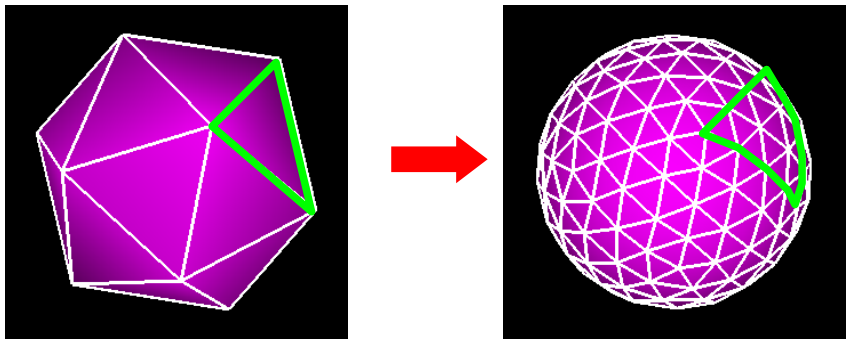
一様にϕ=0と指定
- uniform

正20面体の1パッチでϕ=0と指定
（この三角形から生成される小三角形も同様）
- Xmin，Xmax
- Ymin，Ymax
- Zmin，Zmax

「uniform」と比較して他は収束悪い（特にICCG）

境界条件の指定方法
$R=R_{max}$での固定境界条件（続き）

正20面体の1パッチでϕ=0と指定
（この三角形から生成される小三角形も同様）



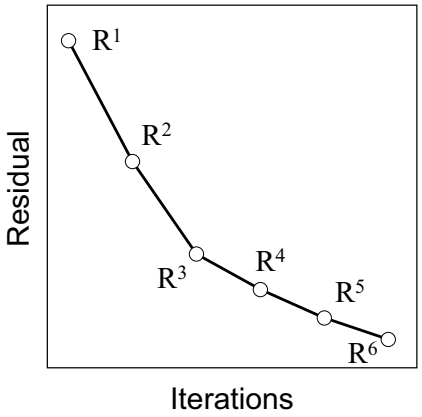
制御データ（input.dat）（CMG1,CMG2）

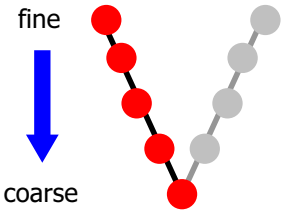
```
test.grid.320
r0.case
T                       : COARSEgrid
4                       : ILEVcoarse
APPROX                  : COARSEgrid METHOD
10                      : iterSSOR
1                       : SOLFLAG= 1 (1:ICCG, 2:MCCG)
0                       : NCOLORtot
10000                     : DEPTHsgl
1.d0                      : SIGMA= 1.d0
1.d-6                     : EPS  = 1.d-8
1000                        : ITERmax=
2                         : iSLEV_MG_type
uniform                   : BOUNDARY conditions
0.80d0, 0.80d0            : CMG1, CMG2
1.0d-24                  : EPSMG
1.0d0                     : EPScoarse
2                         : COARSEsolver_flag
10000, 5, 5               : ITERc/r/p/MAX
1                         : PEsmpTOT
```

反復の切替パラメータ

「CMG1」とは？

```
0.80d0, 0.80d0            : CMG1, CMG2
```





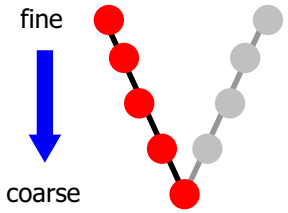
「fine⇒coarse」となっている「Restriction（制限補間）」の各レベルにおける反復において：

$\mathbf{R^{(i)} > CMG1 \times R^{(i-1)}}$
$\mathbf{(0<CMG1<1)}$

という状況になったらこれ以上の反復は無駄として，次の「レベル」（より粗いレベル）に移動する。

「CMG2」とは？

```
0.80d0, 0.80d0            : CMG1, CMG2
```



「fine⇒coarse」となっている「Restriction（制限補間）」の各レベルにおける反復において，あるレベルkにおける残差を$\mathbf{ERR_k}$とするとき

$\mathbf{ERR_k < CMG2 \times ERR_{k-1}}$
$\mathbf{(0<CMG2<1)}$

という状況になったら，次の「レベル」（より粗いレベル）に移動する。

制御データ（input.dat）（EPSMG）

```
test.grid.320
r0.case
T                       : COARSEgrid
4                       : ILEVcoarse
APPROX                  : COARSEgrid METHOD
10                      : iterSSOR
1                       : SOLFLAG= 1 (1:ICCG, 2:MCCG)
0                       : NCOLORtot
10000                     : DEPTHsgl
1.d0                      : SIGMA= 1.d0
1.d-6                     : EPS  = 1.d-8
1000                        : ITERmax=
2                         : iSLEV_MG_type
uniform                   : BOUNDARY conditions
0.80d0, 0.80d0            : CMG1, CMG2
1.0d-24                  : EPSMG
1.0d0                     : EPScoarse
2                         : COARSEsolver_flag
10000, 5, 5               : ITERc/r/p/MAX
1                         : PEsmpTOT
```

最も粗い格子での反復打切誤差
（最も粗い格子での計算をどこまで正確にやるか）

制御データ（input.dat）（ITERc/r/p/MAX）

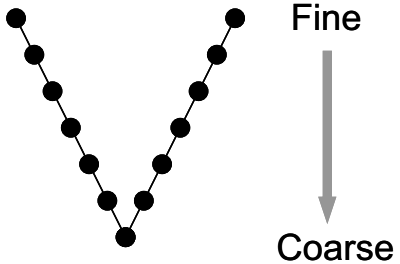
```
test.grid.320
r0.case
T                       : COARSEgrid
4                       : ILEVcoarse
APPROX                  : COARSEgrid METHOD
10                      : iterSSOR
1                       : SOLFLAG= 1 (1:ICCG, 2:MCCG)
0                       : NCOLORtot
10000                     : DEPTHsgl
1.d0                      : SIGMA= 1.d0
1.d-6                     : EPS  = 1.d-8
1000                        : ITERmax=
2                         : iSLEV_MG_type
uniform                   : BOUNDARY conditions
0.80d0, 0.80d0            : CMG1, CMG2
1.0d-24                   : EPSMG
1.0d0                     : EPScoarse
2                         : COARSEsolver_flag
10000, 5, 5               : ITERc/r/p/MAX
1                         : PEsmpTOT
```

各レベルでのGauss-Seidelの反復回数

制御データ（input.dat）（ITERc/r/p/MAX）

Gauss-Seidelの反復回数

```
10000, 5, 5               : ITERc/r/p/MAX
```



ITERcMAX：
最も粗い格子

ITERrMAX：
Restriction（制限補間，細⇒粗）

ITERpMAX：
Prolongataion（延長補間，粗⇒細）

これらの値を大きくすると，全体の反復回数は減少するが一反復あたりの計算時間は増加する⇒トレードオフ
前述の正定性の問題もあり

go.sh

```
#@$-r mg
#@$-q lecture1
#@$-N 4
#@$-J T16
#@$-e err
#@$-o mgcg-064.lst
#@$-lM 28GB
#@$-lT 00:15:00
#@$

cd $PBS_O_WORKDIR

mpirun n2.sh ./solmgcg
exit
```

```
$> cd <$P>/run
$> qsub go.sh
```

soliccg：ICCG法
solmgcg：MGCG法

コア数（領域数）

- 4
  - #@$-N 1
  - #@$-J T4
- 8
  - #@$-N 1
  - #@$-J T8
- 16
  - #@$-N 1
  - #@$-J T16
- 32
  - #@$-N 2
  - #@$-J T16
- 64
  - #@$-N 4
  - #@$-J T16

コア数	ICCG	MGCG
4	181回（13.9 sec）	6回（7.14 sec）
8	328回（25.3 sec）	6回（7.18 sec）
16	610回（54.0 sec）	7回（9.50 sec）
32	1195回（107.0 sec）	9回（12.1 sec）
64	2362回（215.7 sec）	19回（26.0 sec）

計算結果（コア数=32, 10.24×10^6 cell's）

soliccg (ICCG)

```
…
 1141    3.491998E-06
 1161    2.085699E-06
 1181    1.418004E-06
 1195    8.236873E-07
### solver tot time    1.069786E+02 sec.   PE=    0
### comm.      time    4.377219E+00 sec.   PE=    0
    9.590832E+01
       0    1000    1.660144E+04    1.658377E+04
```

solmg (MGCG)

```
    1    1.189154E+00
    2    6.495532E-02
    3    1.383950E-02
    4    5.833822E-04
    5    2.290085E-04
    6    1.387011E-05
    7    8.196302E-06
    8    1.108853E-06
    9    3.970644E-07
### solver tot time    1.212685E+01 sec.   PE=    0
### MGCG               6.109128E-01 sec.   PE=    0
### MG                 1.151593E+01 sec.   PE=    0
### relax              1.097669E+01 sec.   PE=    0
### restriction        4.404089E-01 sec.   PE=    0
### prolongation       8.220959E-02 sec.   PE=    0
### comm.      time    1.852987E+00 sec.   PE=    0
    8.471996E+01
       0    1000    1.660144E+04    1.658377E+04
```


並列計算における問題点

- 領域境界において，必ずしも最新の値を使えない場合
  - 収束の悪化
- 様々な対策が試みられている
  - 領域間オーバーラップ
  - HID（Hierarchical Interface Decomposition）