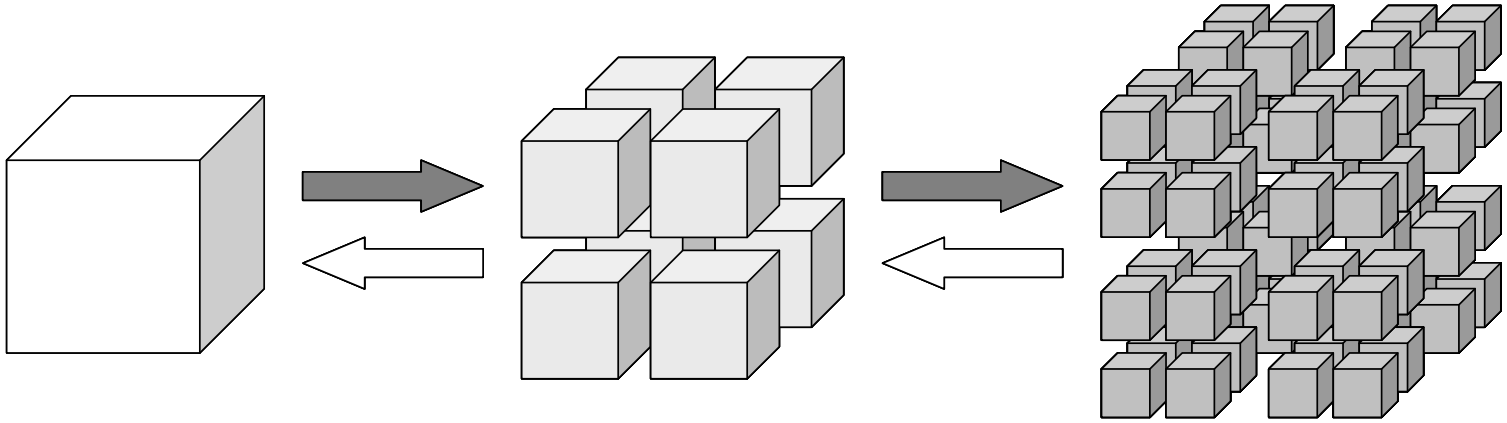
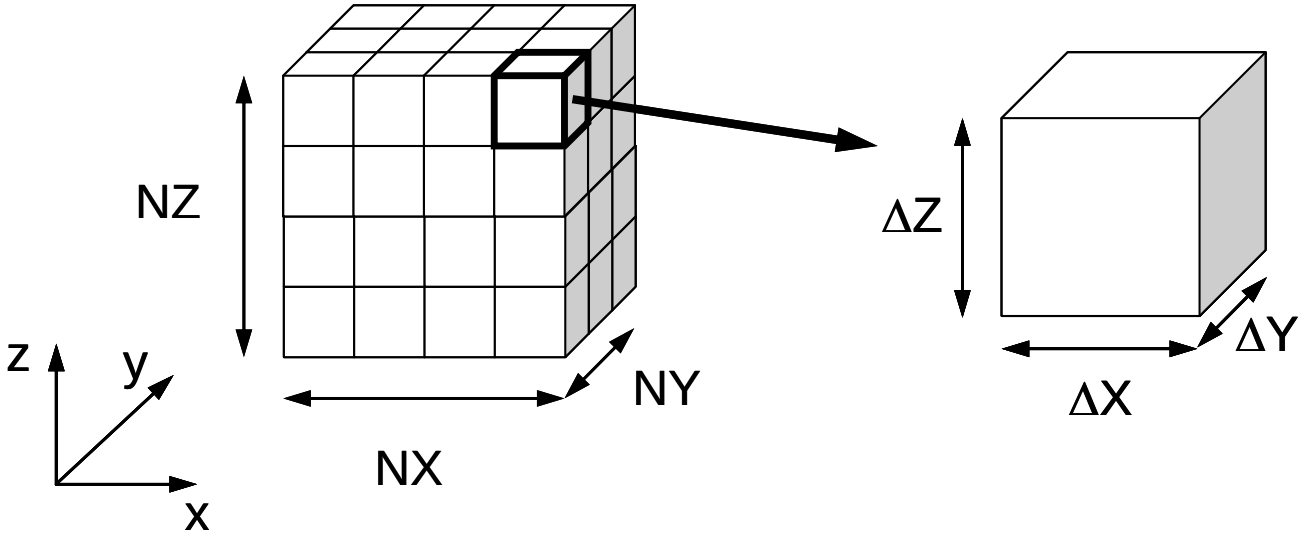
多重格子法について



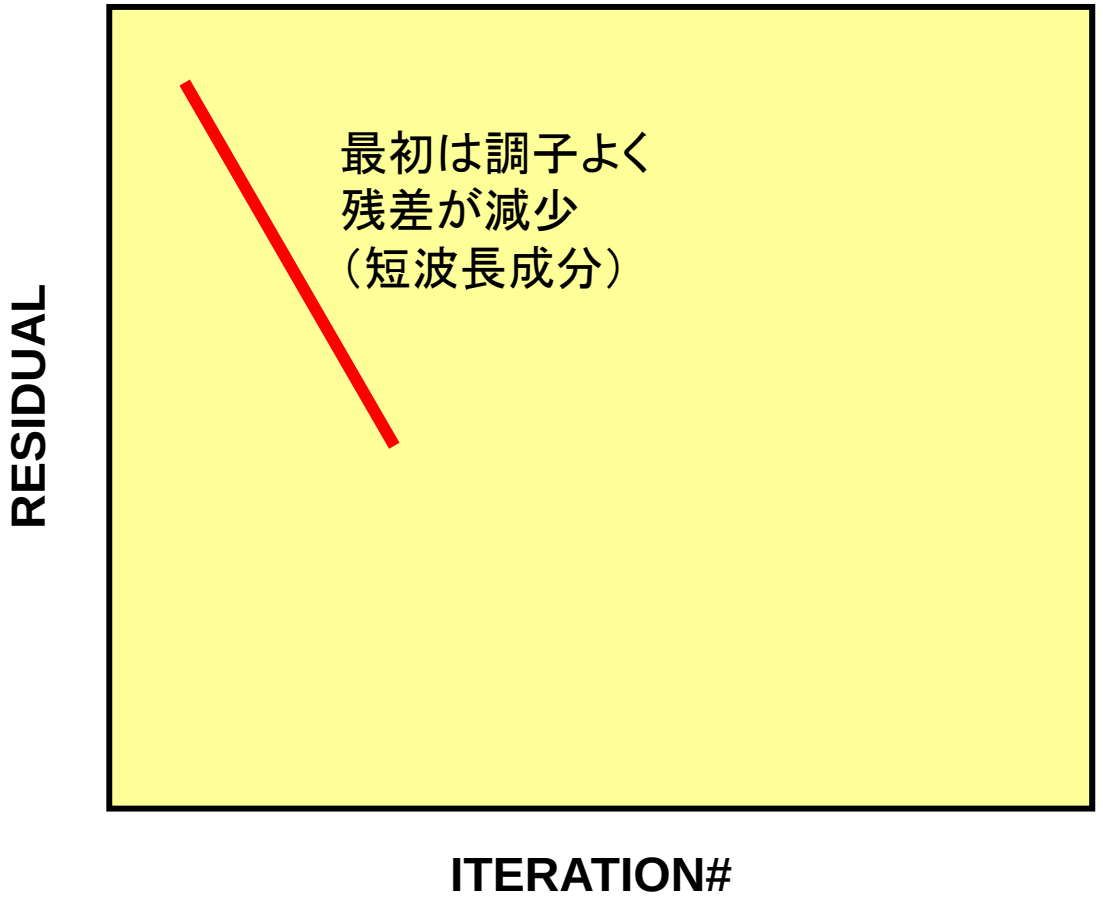
Multigridとは

http://www.llnl.gov/CASC/

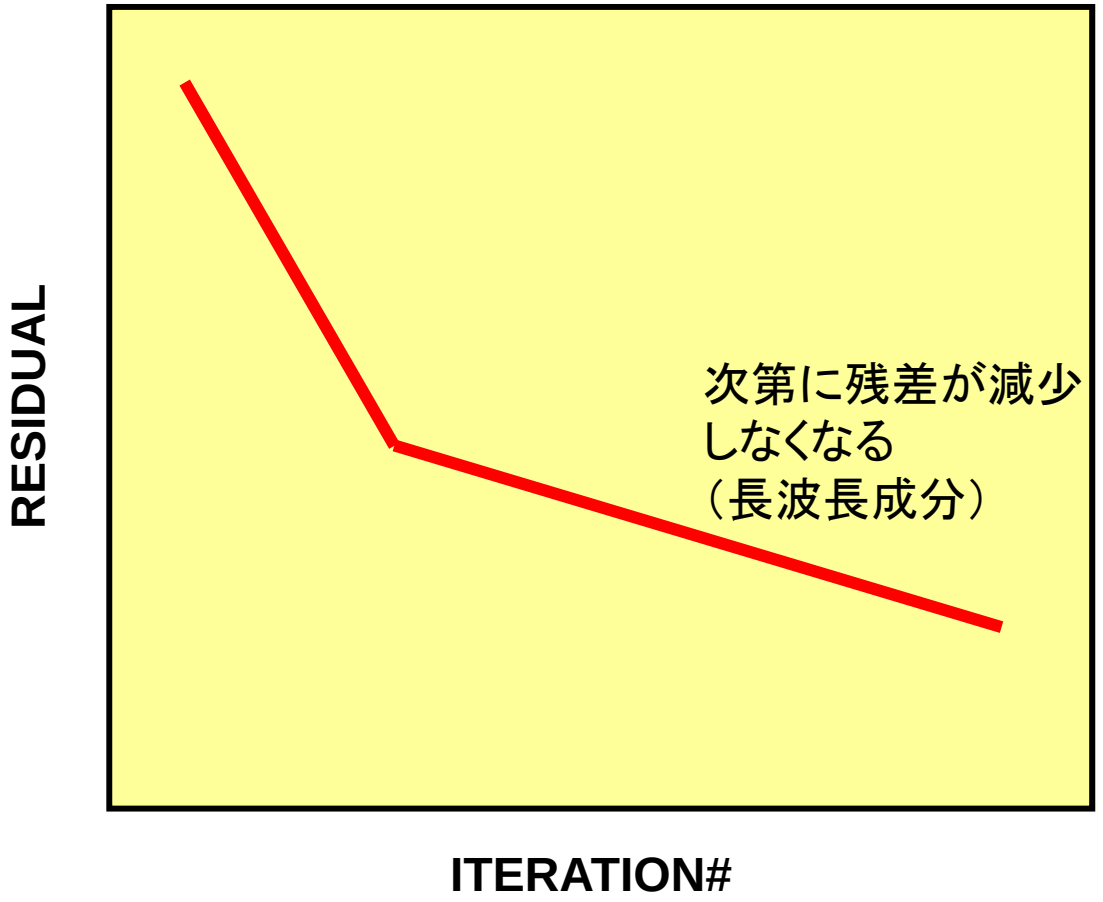
- 格子を使用して離散化された系に対して反復法を適用する場合，基本的に格子サイズと同じ波長を持った誤差が効率よく減衰する。
  - Gauss-Seidel，SOR
- しかしながら，長波長の誤差はなかなか減衰しない。



Gauss-Seidel法，SOR法の収束

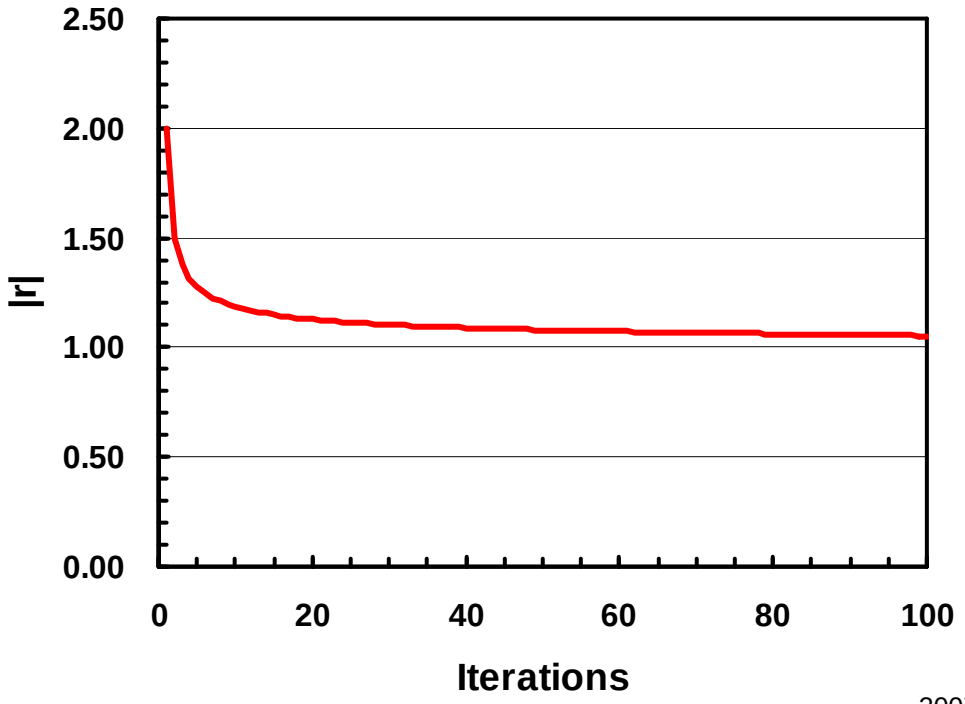


Gauss-Seidel法，SOR法の収束



Multigridとは
http://www.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



Gauss-Seidel法
収束履歴の例

Multigridとは
http://www.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（ポアソン方程式）:

反復回数（$\varepsilon=10^{-8}$）

N	IC(0)	Point Jacobi
$8^3=$ 512	19	48
$16^3=$ 4,096	38	102
$32^3=$ 32,768	75	208
$64^3=$ 262,144	146	413
$128^3=$2,097,152	290	826

- 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

Multigridとは（続き）

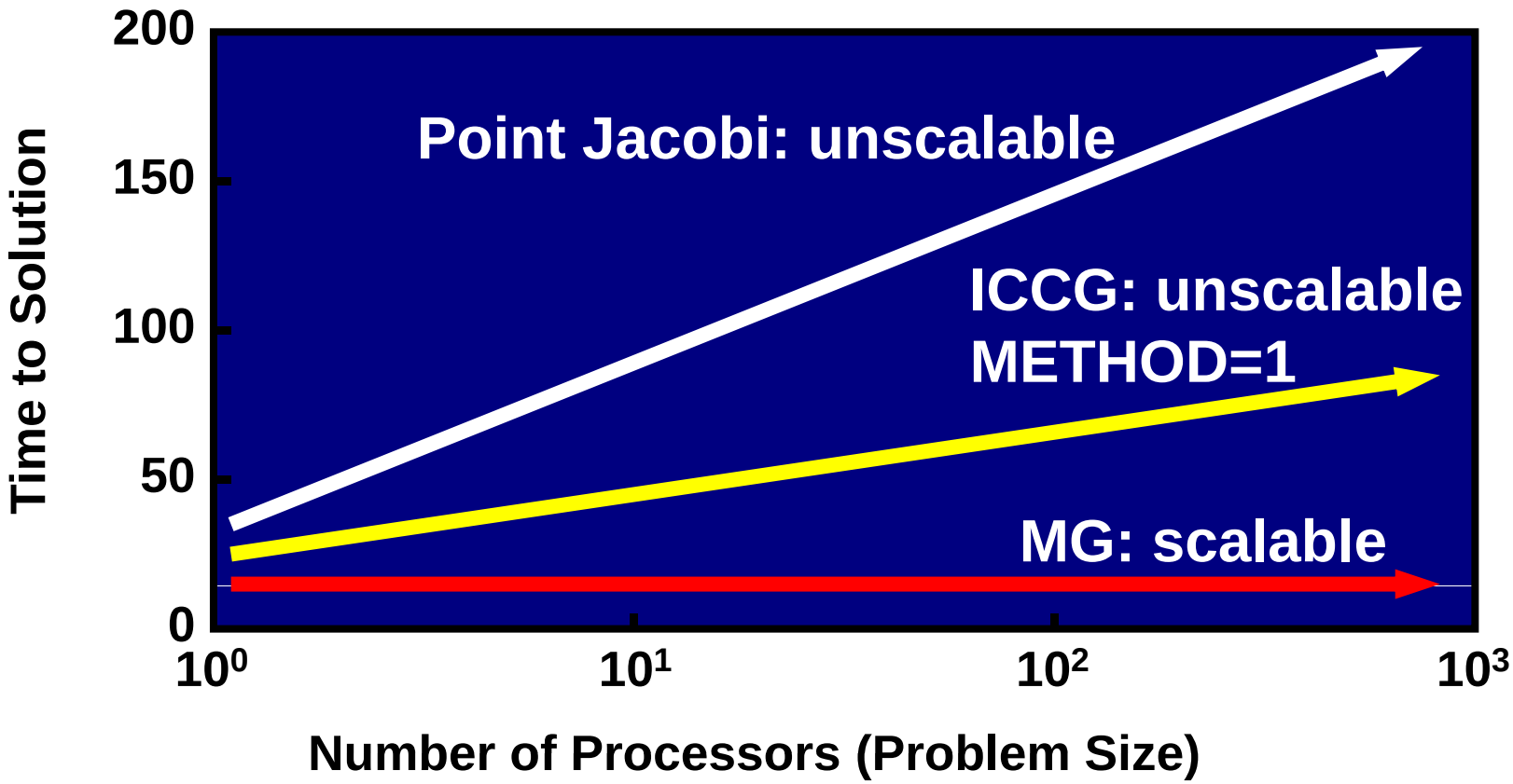
http://www.llnl.gov/CASC/

- 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

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

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：$r_C = R_{F\Rightarrow C}\, r_F$

$$A_C u_C = r_C$$

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。

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

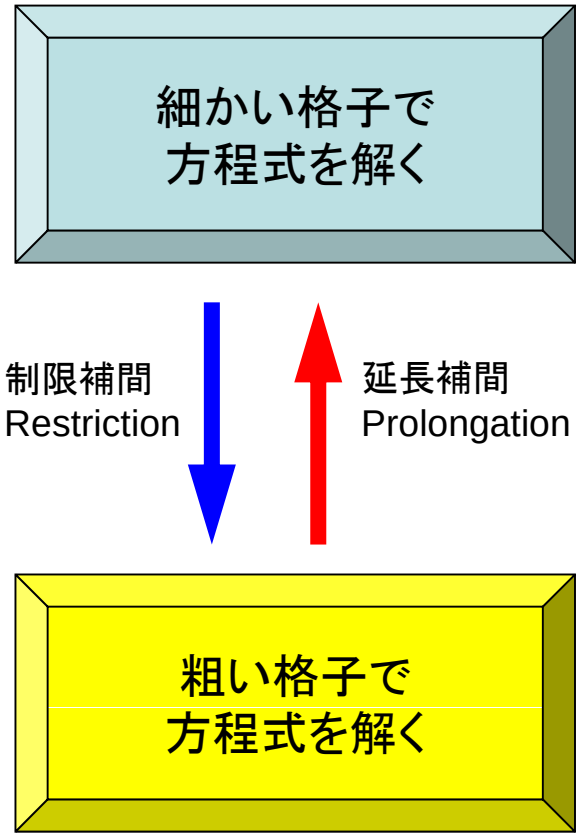
5. 粗い格子上での解 u_C から，延長補間オペレータ $R_{C\Rightarrow F}$ によって，細かい格子における修正量 $\Delta u_F^{(i)} = R_{C\Rightarrow F}\, u_C$ を求める。

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

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

6. 細かい格子での解を修正量によって更新する

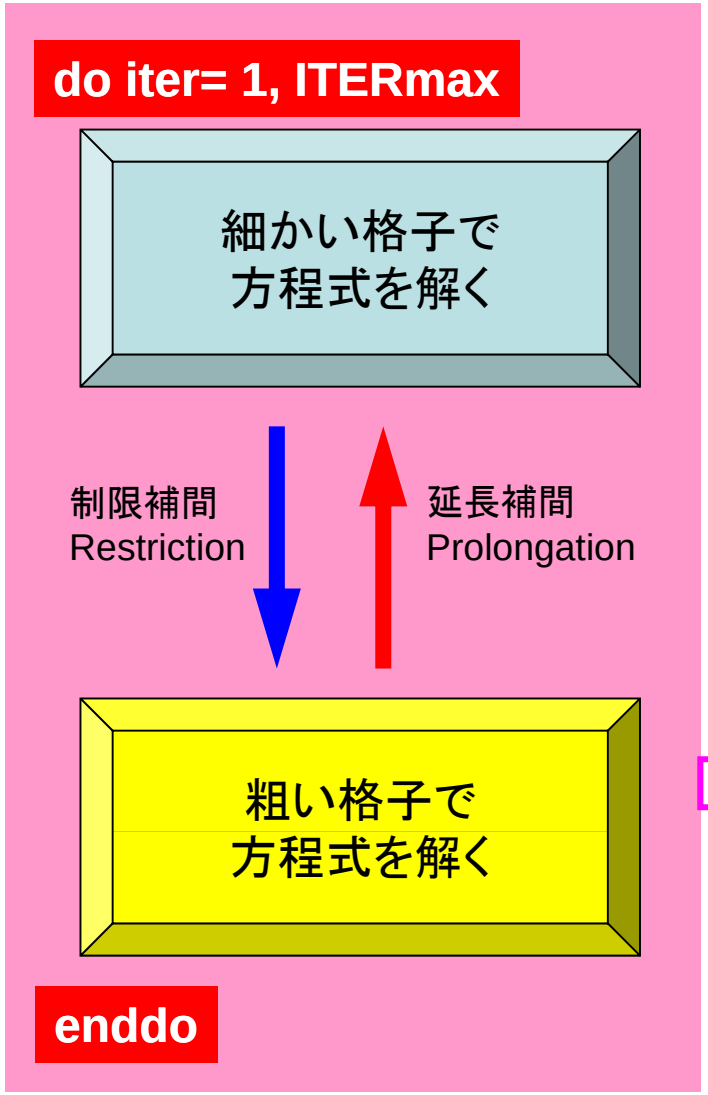
Multigridのアルゴリズム:2レベルの例



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。

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

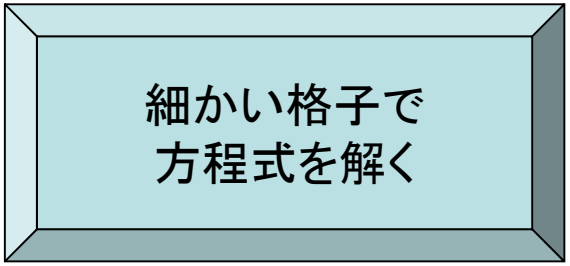
Multigridのアルゴリズム：2レベルの例

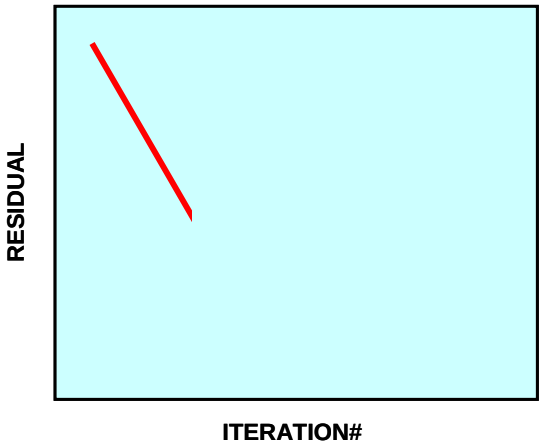


1. 線形化された方程式 $A_F\, u_F = f$ を細かい格子上で緩和し，結果を $u_F^{(i)} = S_F\,(A_F, f)$ とする。演算子 S_F (例えば Gauss-Seidel) は緩和演算子（*smoothing operator*）と呼ばれる。
2. 細かい格子上で残差 $r_F = f - A_F\, u_F^{(i)}$ を求める。
3. 制限補間オペレータ $R_{F \Rightarrow C}$ によって，細かい格子上での残差を粗い格子に補間する：$r_C = R_{F \Rightarrow C}\, r_F$
4. 方程式 $A_C\, u_C = r_C$ （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。

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

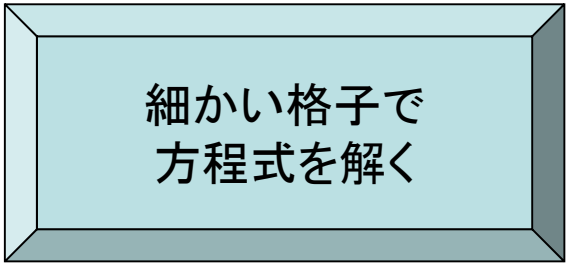
細かい格子で厳密に方程式を解く必要は無い

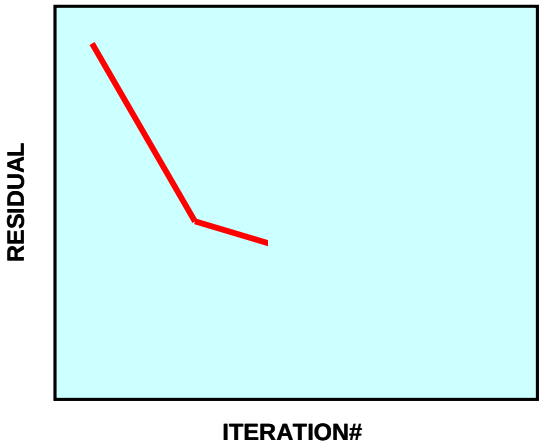




- 何回か反復する

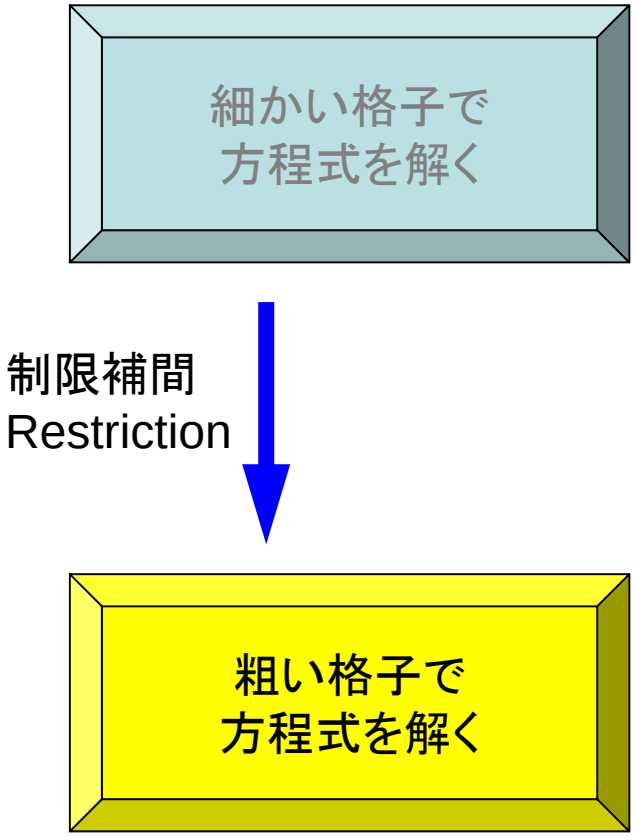
細かい格子で厳密に方程式を解く必要は無い





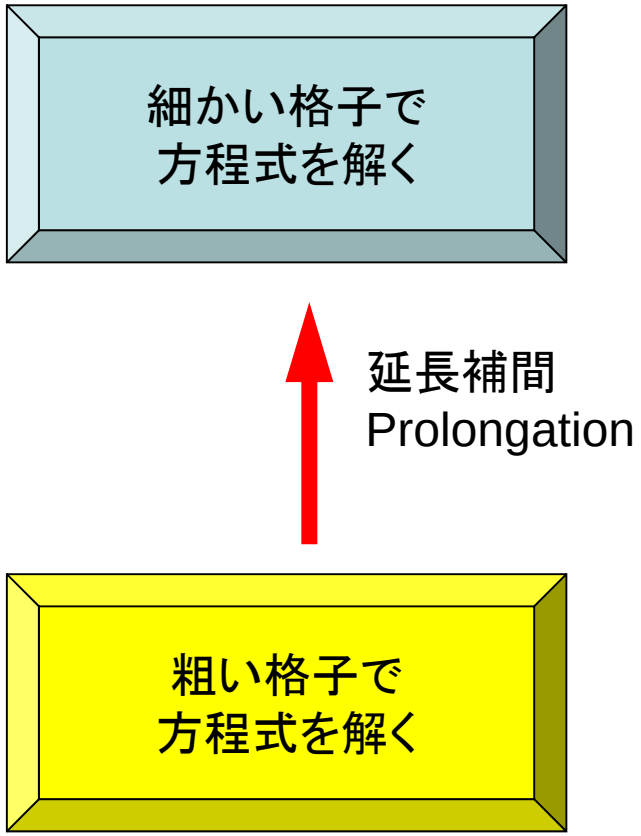
- 長波長成分が支配的になってきたら

細かい格子で厳密に方程式を解く必要は無い



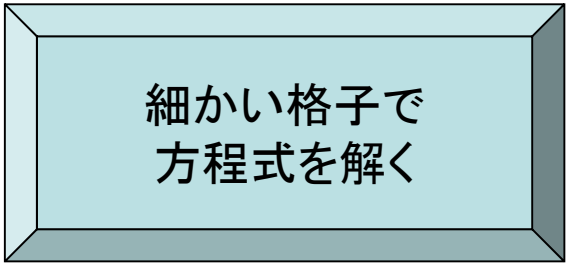
- 粗い格子に移る

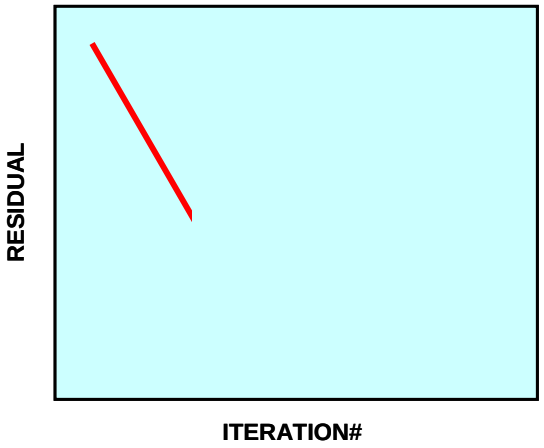
細かい格子で厳密に方程式を解く必要は無い



- 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

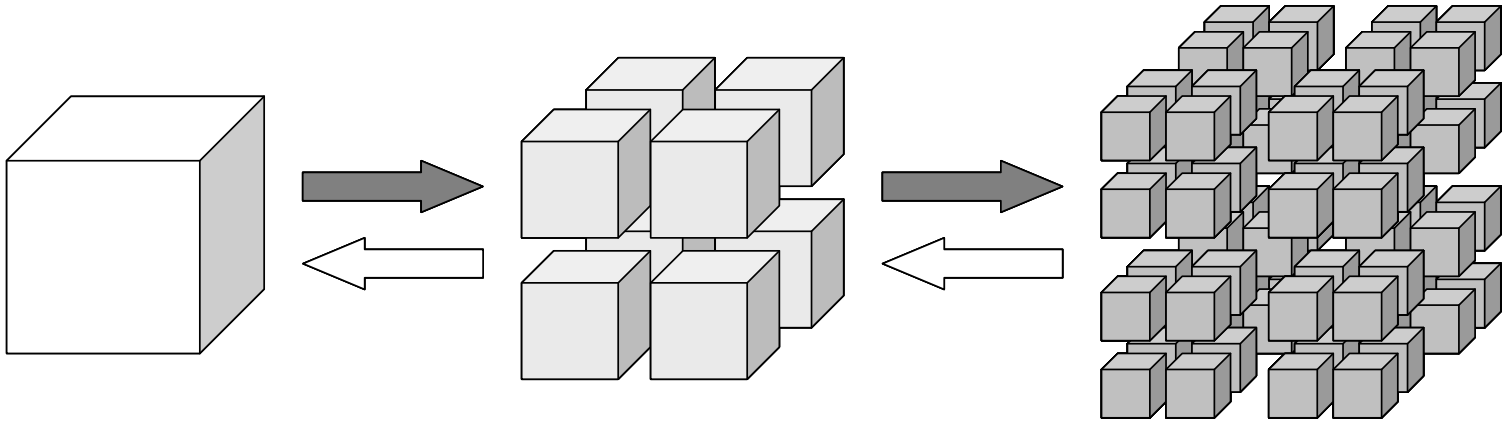




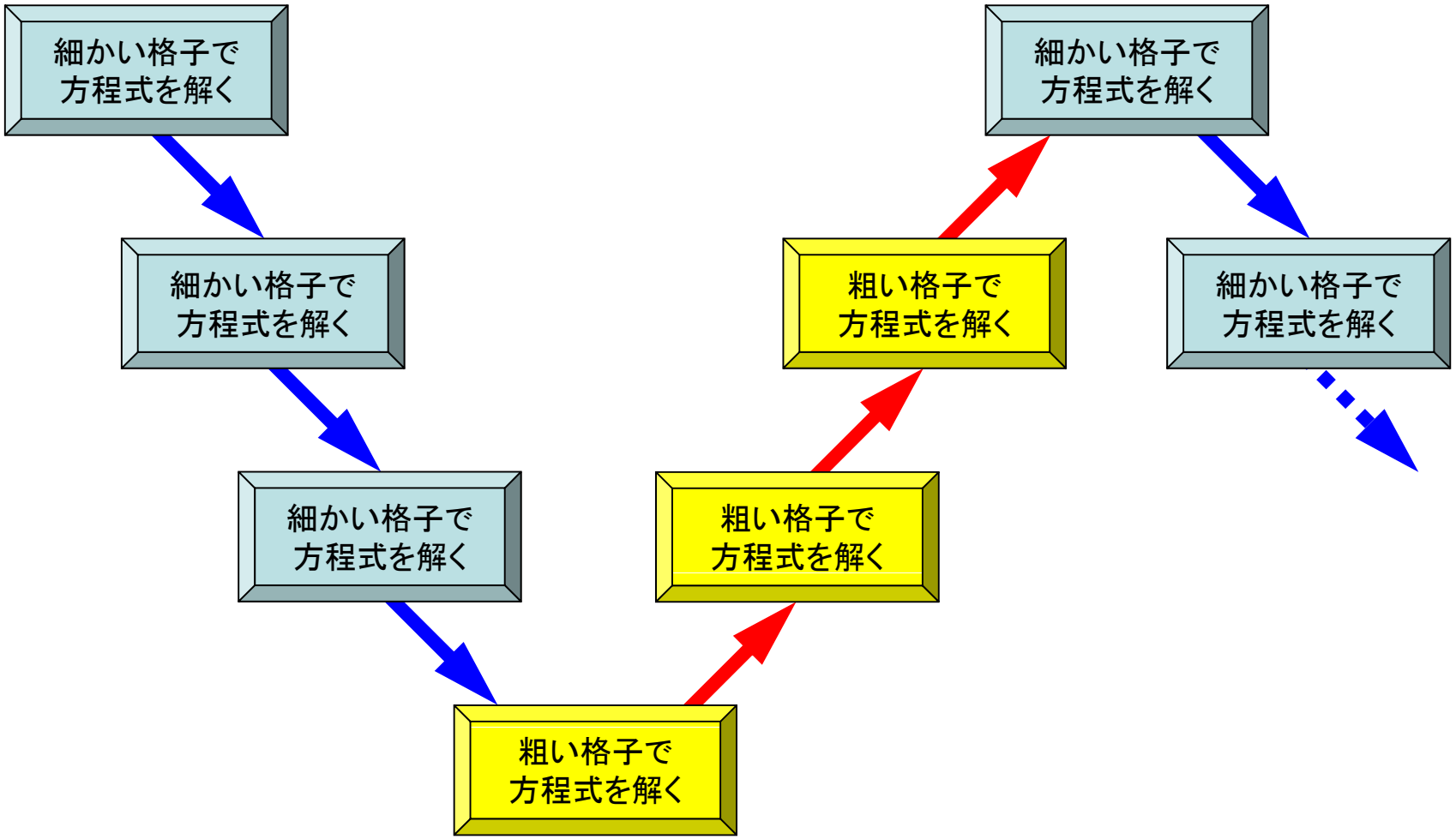
- また，細かい格子で何回か反復する

Multigridの概要

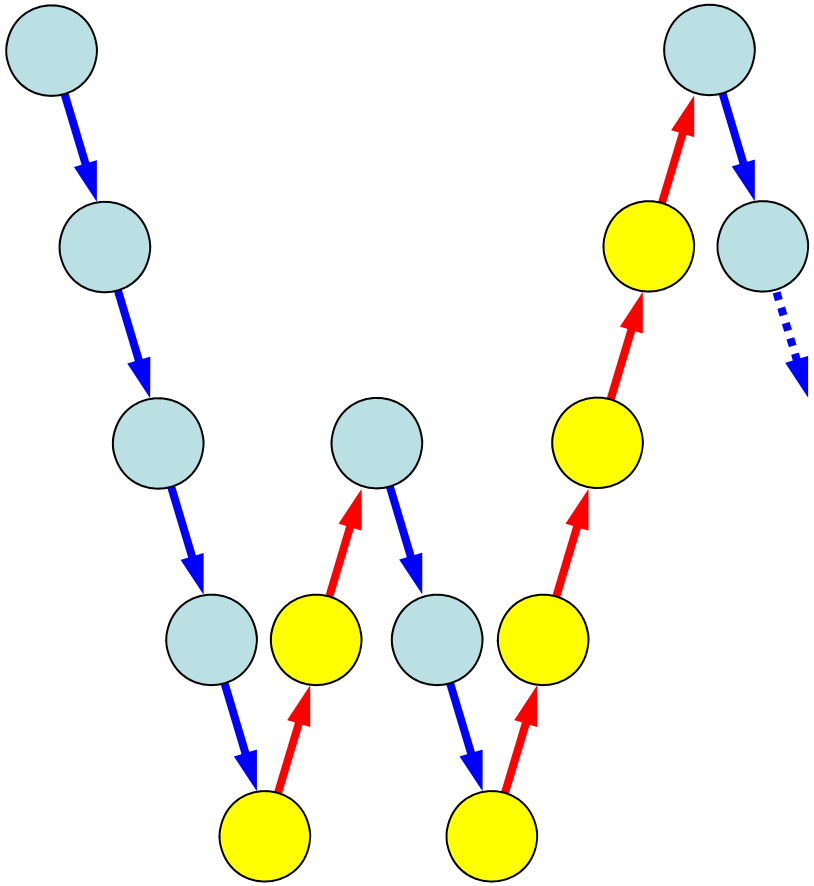
- この 2 レベルのプロセスを再帰的に多段階に適用することによって，マルチグリッドの「V-Cycle」が構築可能である。
- 「V-Cycle」の各レベルの計算が適切に実施されれば，誤差のあらゆる長さの波長をもった成分を一様に減衰させることができる。
- 計算時間が問題規模に比例するいわゆる「scalable」な手法の実現が可能である。



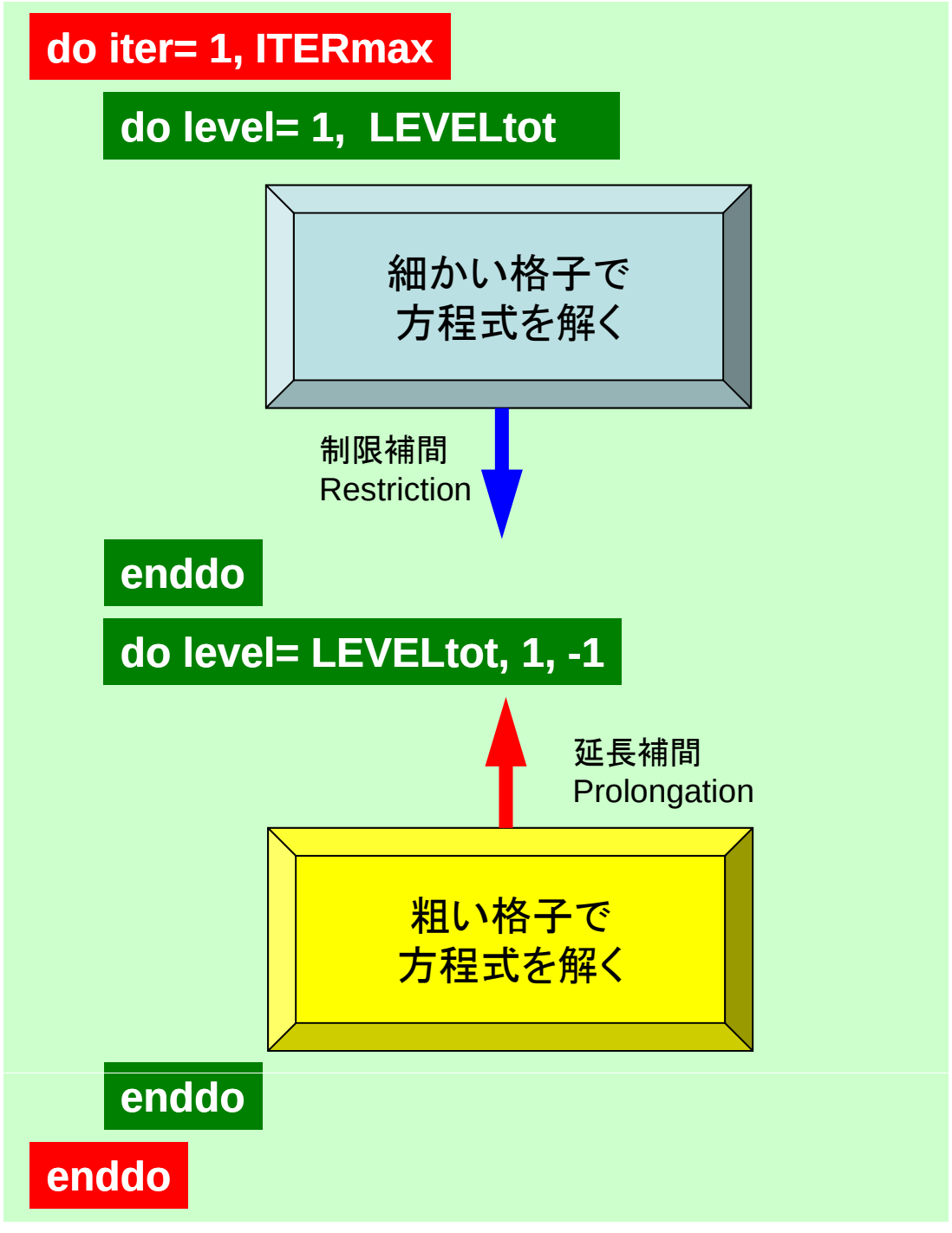
これを多段階で再帰的に実施するのがMultigrid（V-Cylcleの例）



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


do iter= 1, ITERmax
do level= 1, LEVELtot
細かい格子で
方程式を解く
制限補間
Restriction
enddo
do level= LEVELtot, 1, -1
延長補間
Prolongation
粗い格子で
方程式を解く
enddo
enddo

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

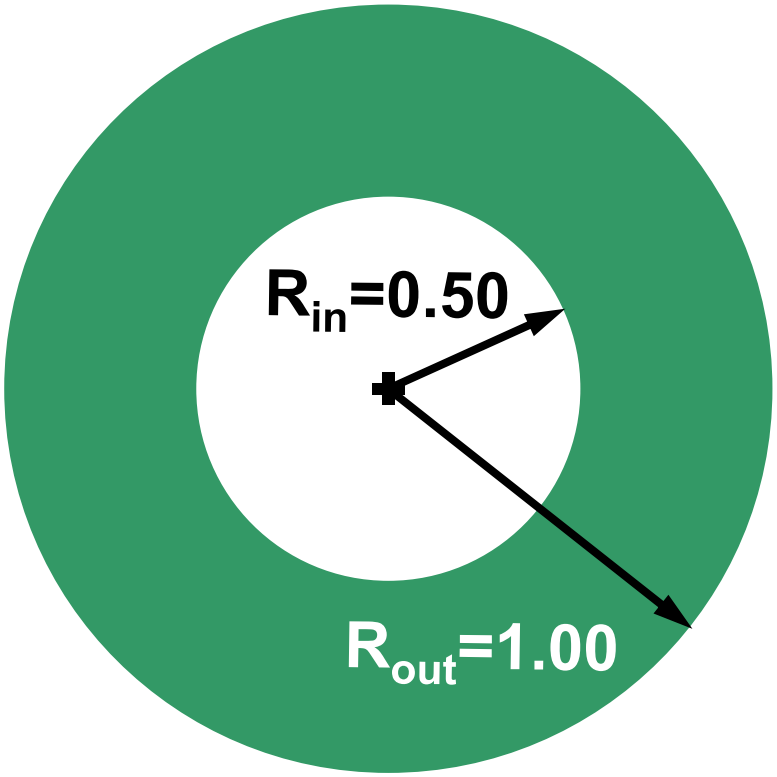
- Algebraicは，マトリクスのコネクティビティに基づいているため，とっつきにくい感じがするが，はるかにフレキシビリティに富んでいる。

Multigridの問題点等

- 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

計算例: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：$p^{(n+1)}-p^{(n)}=\frac{1}{\Delta t}\phi$

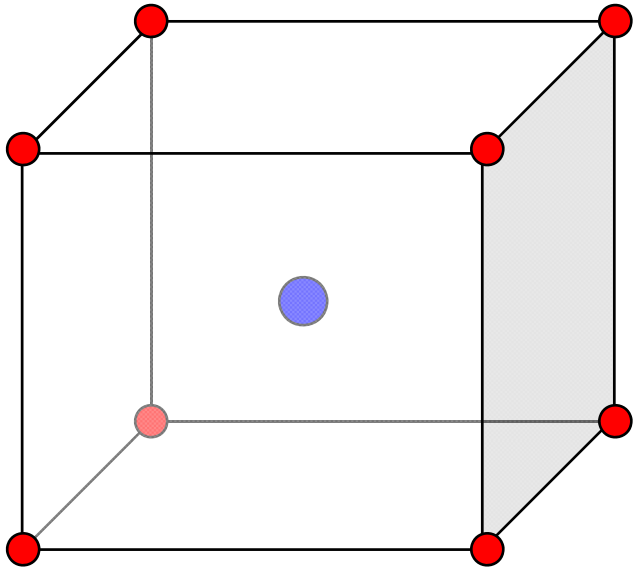
$$\nabla\cdot\mathbf{u}^{(n+1)}=0$$

$$\Delta\phi=\nabla\cdot\mathbf{u}'$$ 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



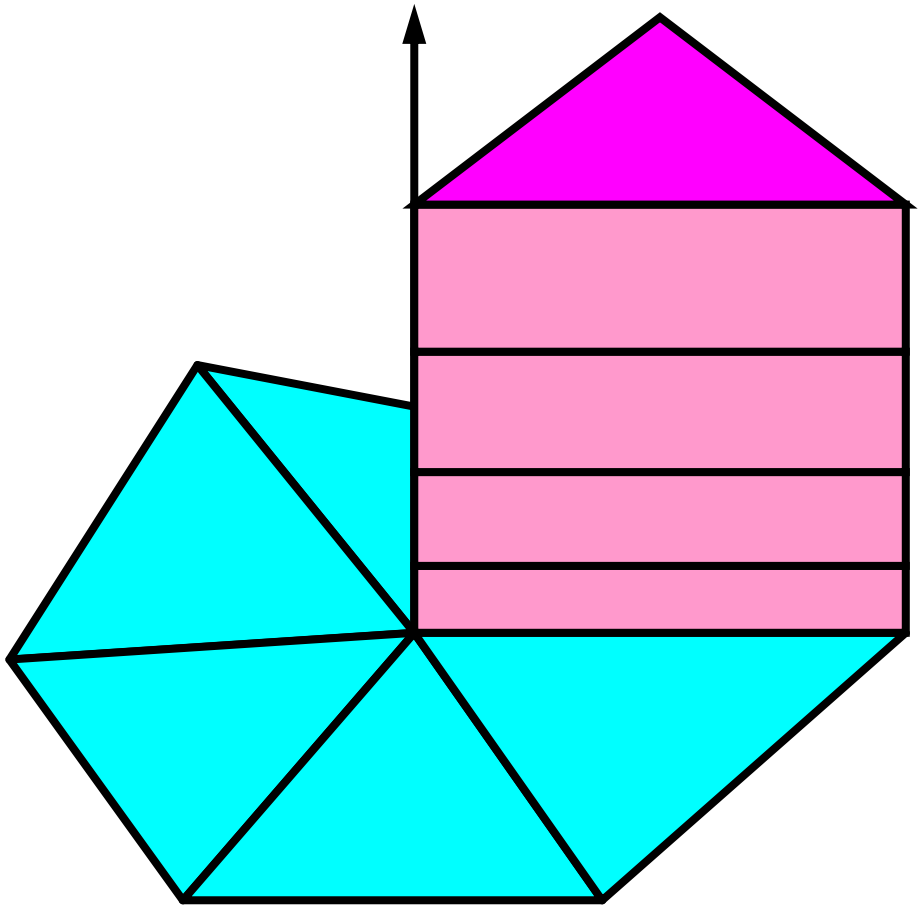
- ガウス＝グリーン型有限体積法

ポアソン方程式に対する並列マルチグリッド法

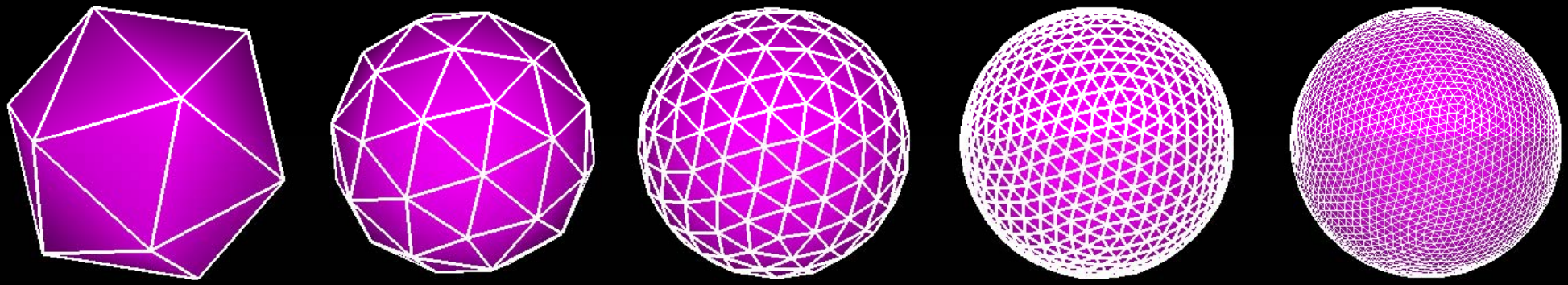
- 前処理付きCG法
- Vサイクル
- ガウス=ザイデル法によるスムージング：緩和演算子
- 多段階通信テーブル
- FORTRAN90＋MPI

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
  - 佐藤准教授（AORI）：NICAM



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
12 nodes
20 tri's

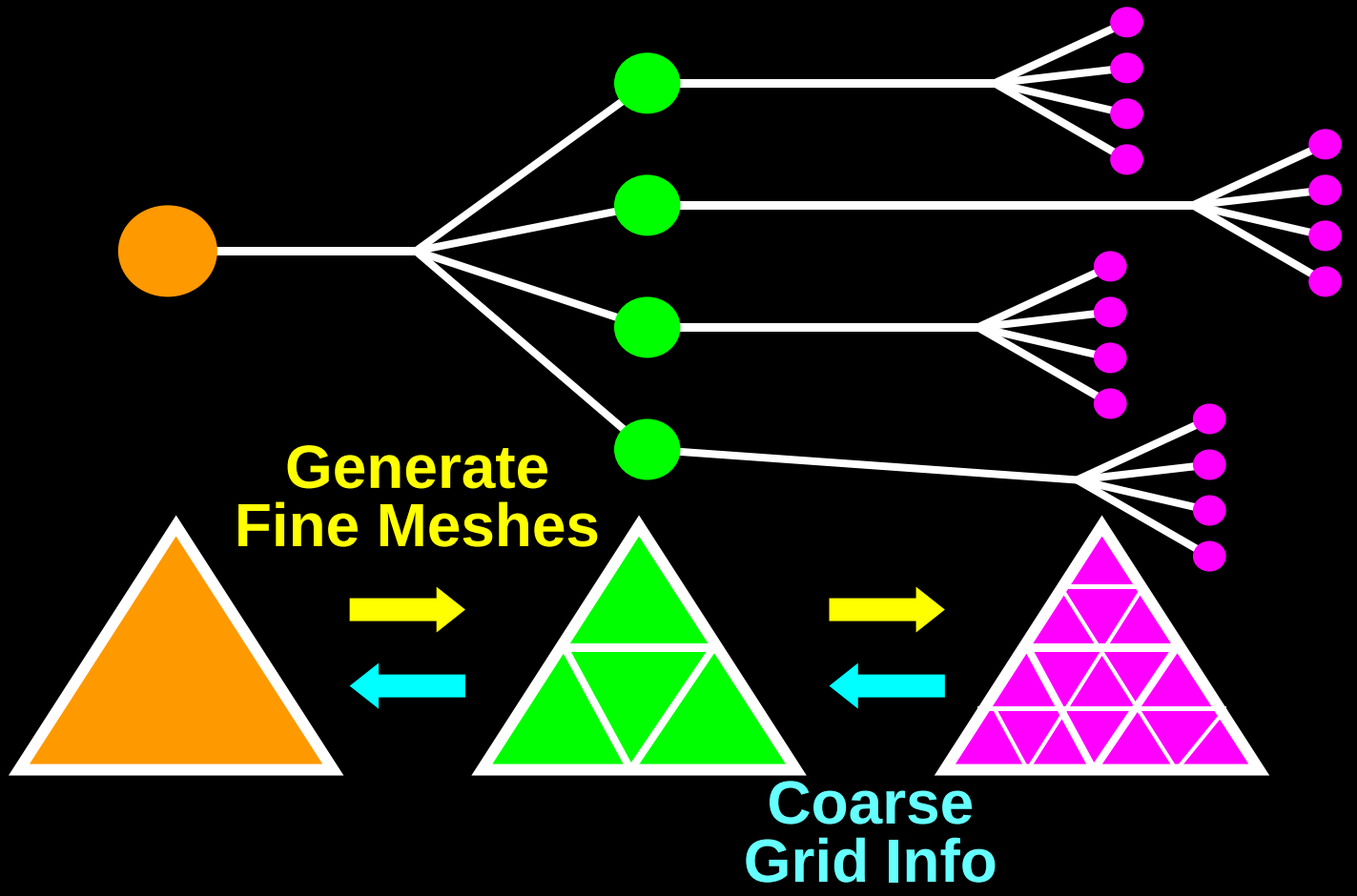
Level 1
42 nodes
80 tri's

Level 2
162 nodes
320 tri's

Level 3
642 nodes
1,280 tri's

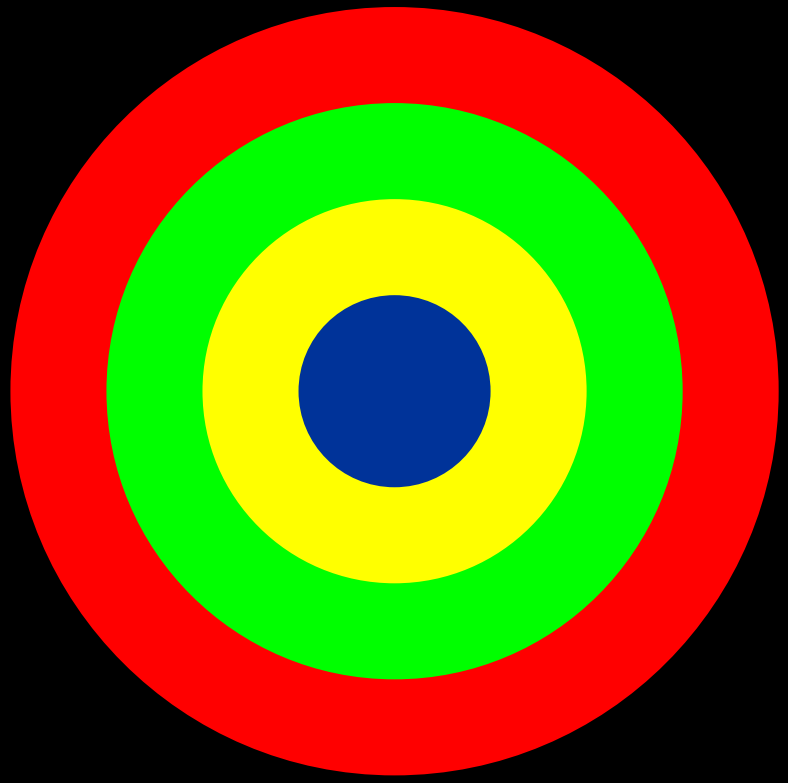
Level 4
2,562 nodes
5,120 tri's

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



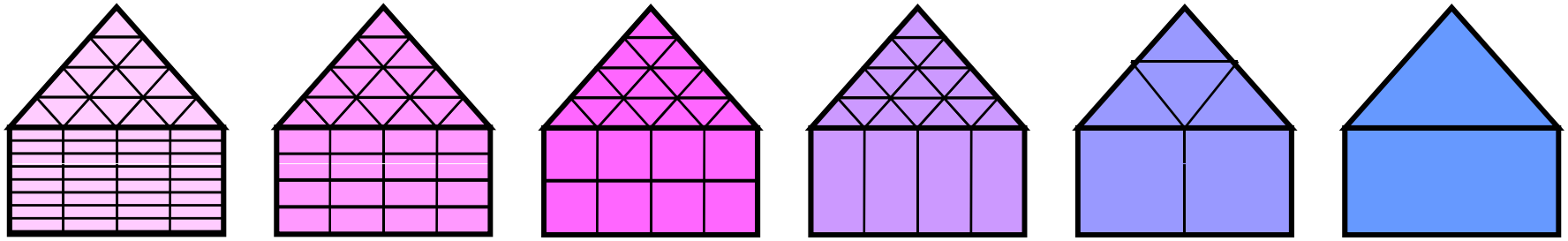
領域分割

- 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。



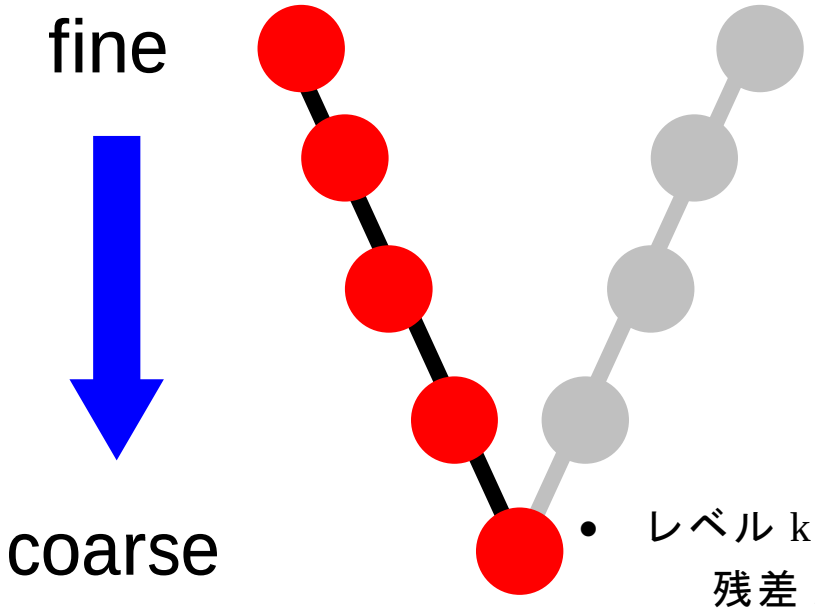
MG前処理付きCGソルバー

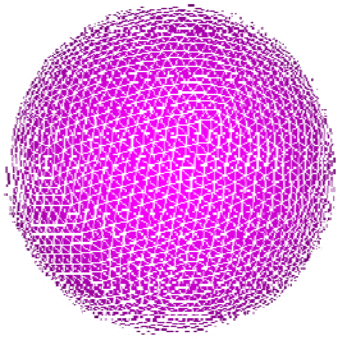
```
Compute {r}= {b} - [A]{x} for initial guess
for i= 1, 2, ...
    solve [M]{z}= {r}
    ρ_{i-1}= {r}{z}
    if i= 1
      {p}= {z}
     else
      β_{i-1}= ρ_{i-1} / ρ_{i-2}
      {p}= {z} + β_{i-1}{p}
    endif
    {q}= [A]{p}
    α_i= ρ_{i-1} / {p}{q}
    {x}= {x} + α_i {p}
    {r}= {r} - α_i {p}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

Multigrid Preconditioning

この部分をMultigridを使って解く

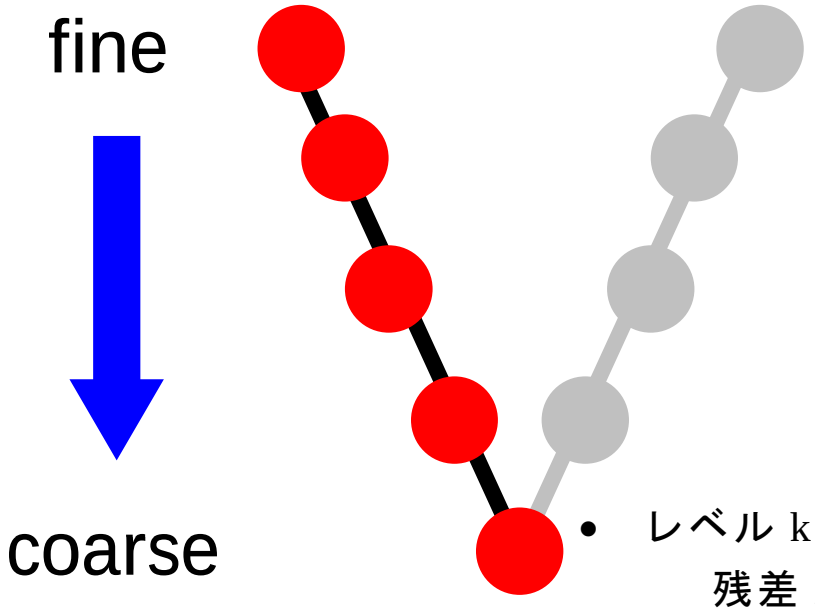
Vサイクルマルチグリッド Restriction（制限補間）

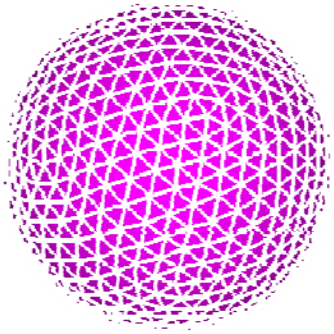




- レベル k において方程式 $A_k\, u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\, u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\, r_k$
- レベル k+1 において方程式 $A_{k+1}\, u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\, u_{k+1}$ を求める。
- 制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

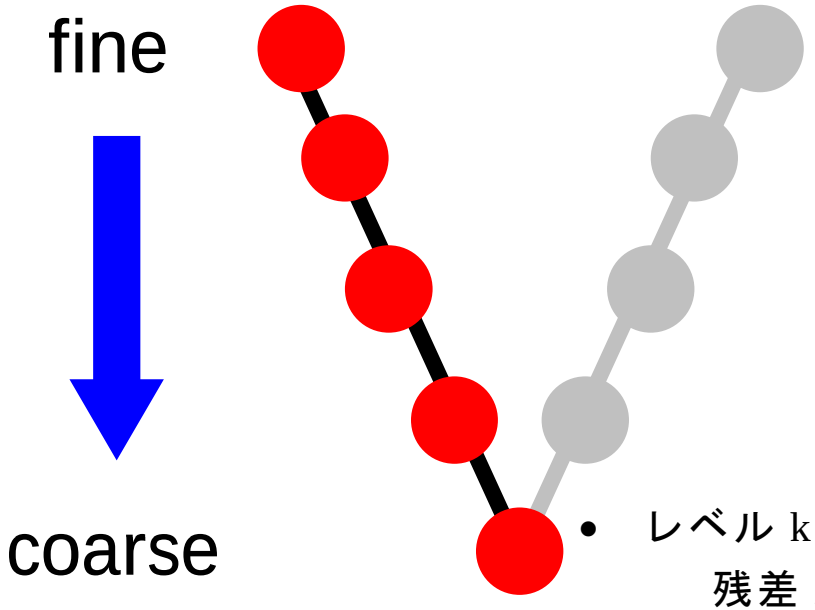
Vサイクルマルチグリッド Restriction（制限補間）

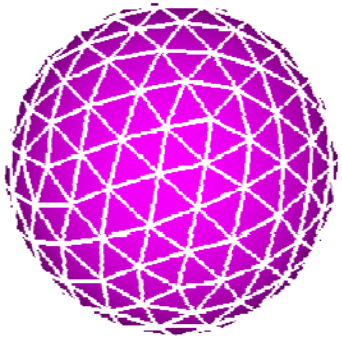




- レベル k において方程式 $A_k\, u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\, u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\, r_k$
- レベル k+1 において方程式 $A_{k+1}\, u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\, u_{k+1}$ を求める。
- 制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

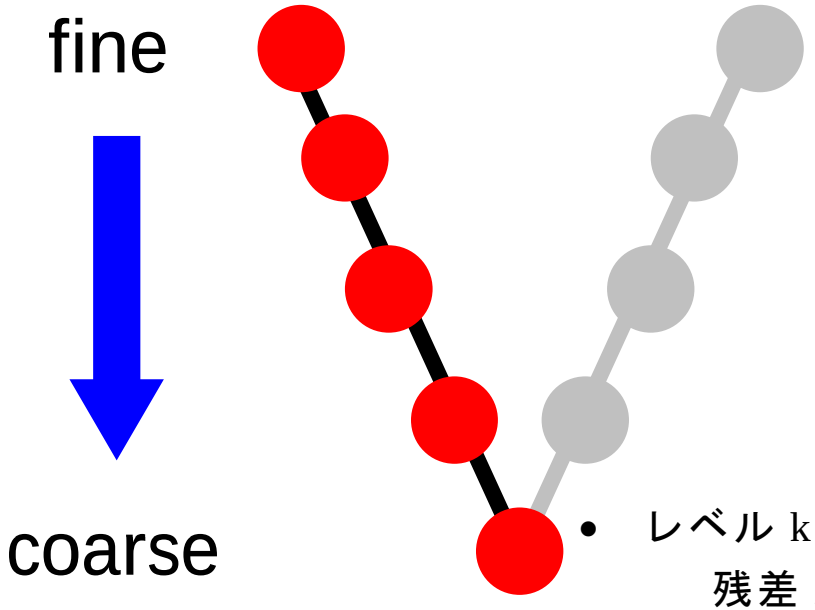
Vサイクルマルチグリッド Restriction（制限補間）

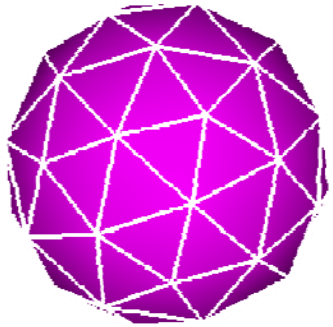




- レベル k において方程式 $A_k\, u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\, u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\, r_k$
- レベル k+1 において方程式 $A_{k+1}\, u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\, u_{k+1}$ を求める。
- 制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

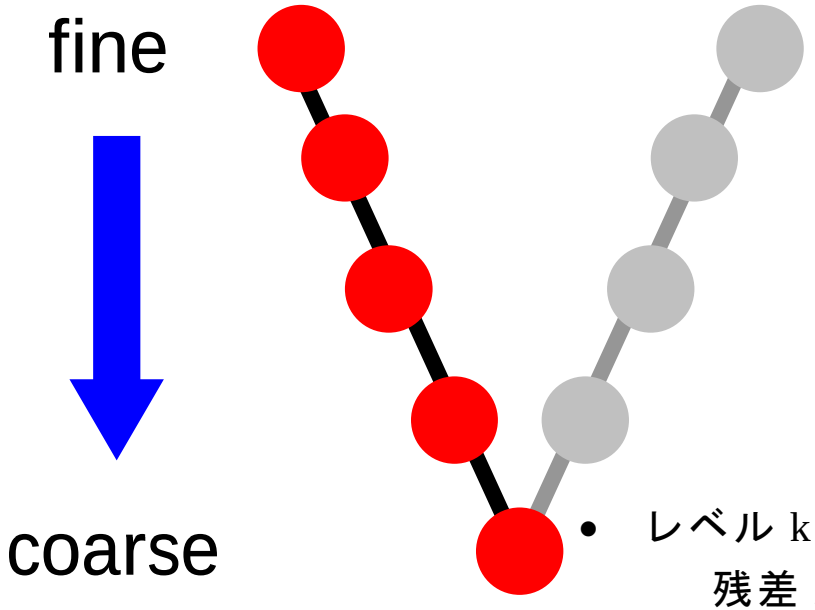
Vサイクルマルチグリッド Restriction（制限補間）

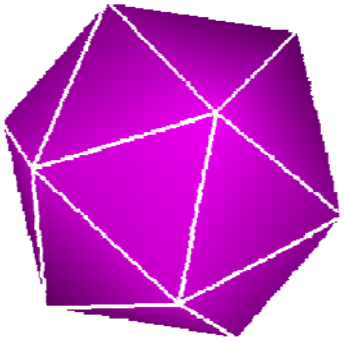




- レベル k において方程式 $A_k\, u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\, u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\, r_k$
- レベル k+1 において方程式 $A_{k+1}\, u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\, u_{k+1}$ を求める。
- 制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

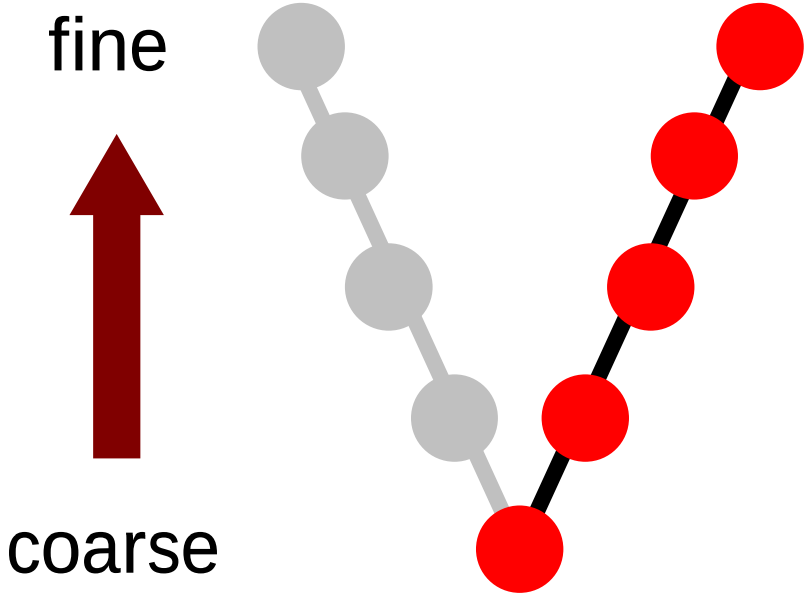
Vサイクルマルチグリッド Restriction（制限補間）

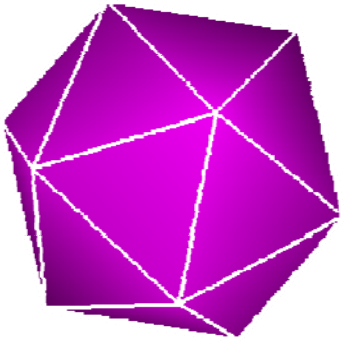




- レベル k において方程式 $A_k\, u_k = f_k$ を解き，結果を u_k とする
  残差 $r_k = f_k - A_k\, u_k$ を求める。
- 制限補間オペレータ $R_{k \Rightarrow k+1}$ によって，r_k を一段階粗い格子（レベル k+1）に補間する：$f_{k+1} = R_{k \Rightarrow k+1}\, r_k$
- レベル k+1 において方程式 $A_{k+1}\, u_{k+1} = f_{k+1}$ を解く。
  残差 $r_{k+1} = f_{k+1} - A_{k+1}\, u_{k+1}$ を求める。
- 制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

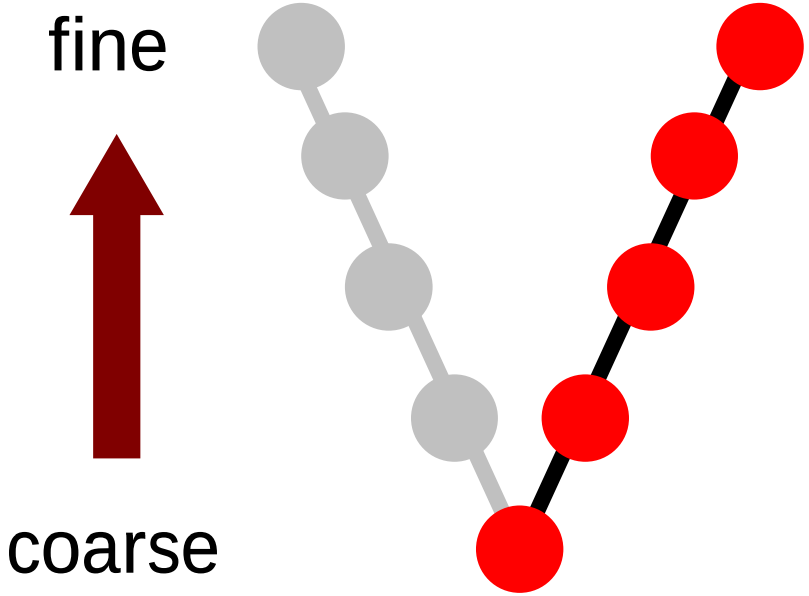
Vサイクルマルチグリッド Prolongation（延長補間）





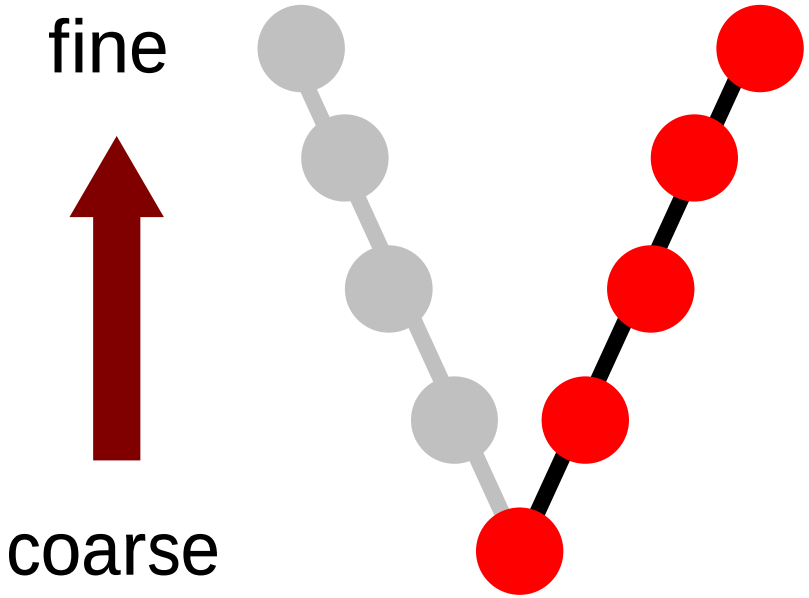
- レベル k+1 における解 u_{k+1} から，延長補間オペレータ $R_{k+1 \Rightarrow k}$ によって，レベル k における修正量 $\Delta u_k = R_{k+1 \Rightarrow k}\ u_{k+1}$ を求める。
- レベル k の解を修正量によって更新する：$u_k = u_k + \Delta u_k$
- レベル k における解 u_k から，延長補間オペレータ $R_{k \Rightarrow k-1}$ によって，レベル k-1 における修正量 $\Delta u_{k-1} = R_{k \Rightarrow k-1}\ u_k$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

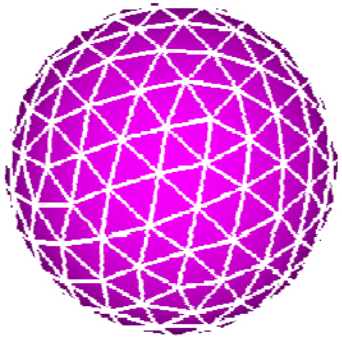
Vサイクルマルチグリッド Prolongation（延長補間）



- 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。

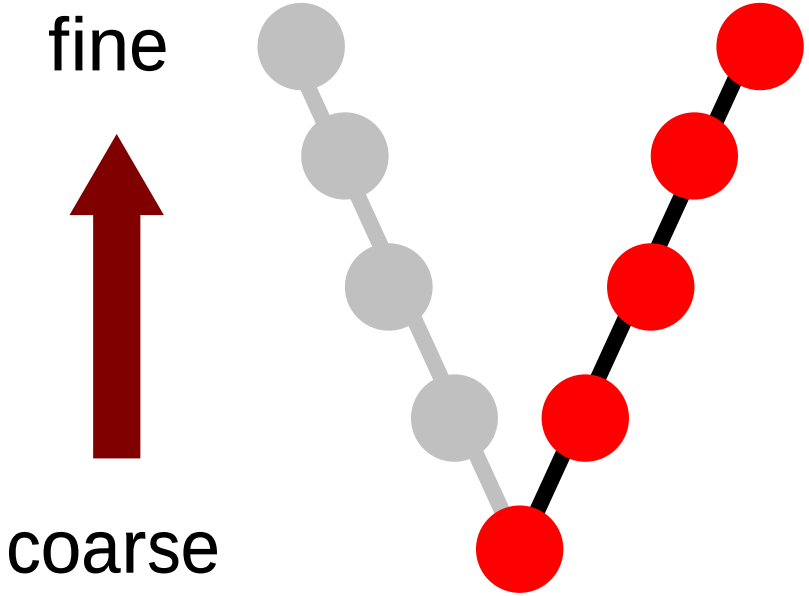
Vサイクルマルチグリッド Prolongation（延長補間）

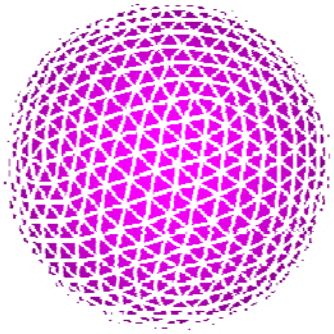




- レベル k+1 における解 u_{k+1} から，延長補間オペレータ $R_{k+1 \Rightarrow k}$ によって，レベル k における修正量 $\Delta u_k = R_{k+1 \Rightarrow k}\ u_{k+1}$ を求める。
- レベル k の解を修正量によって更新する：$u_k = u_k + \Delta u_k$
- レベル k における解 u_k から，延長補間オペレータ $R_{k \Rightarrow k-1}$ によって，レベル k-1 における修正量 $\Delta u_{k-1} = R_{k \Rightarrow k-1}\ u_k$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

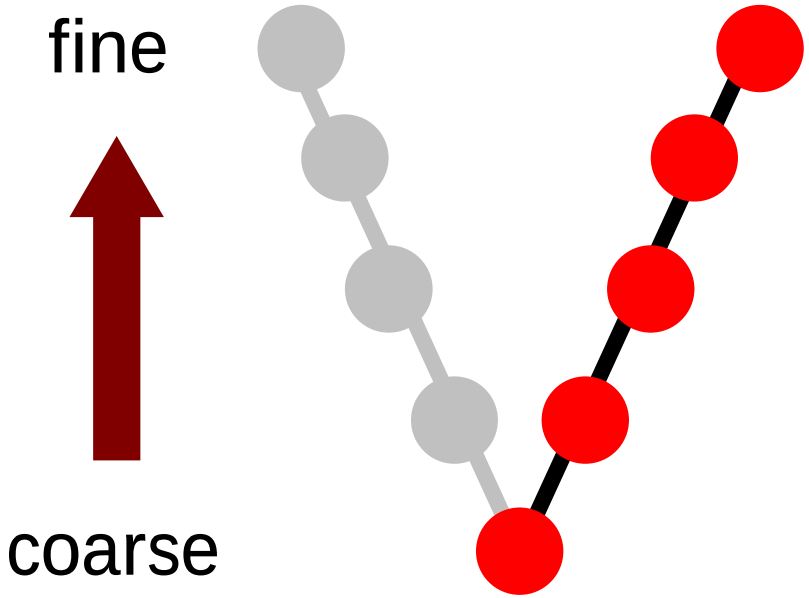
Vサイクルマルチグリッド Prolongation（延長補間）

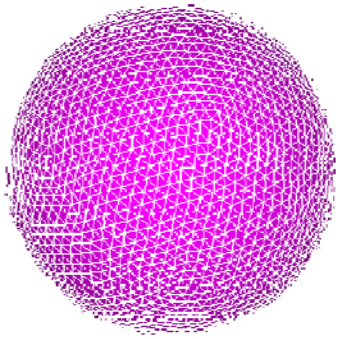




- レベル k+1 における解 u_{k+1} から，延長補間オペレータ $R_{k+1 \Rightarrow k}$ によって，レベル k における修正量 $\Delta u_k = R_{k+1 \Rightarrow k}\ u_{k+1}$ を求める。
- レベル k の解を修正量によって更新する：$u_k = u_k + \Delta u_k$
- レベル k における解 u_k から，延長補間オペレータ $R_{k \Rightarrow k-1}$ によって，レベル k-1 における修正量 $\Delta u_{k-1} = R_{k \Rightarrow k-1}\ u_k$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

Vサイクルマルチグリッド
Prolongation（延長補間）

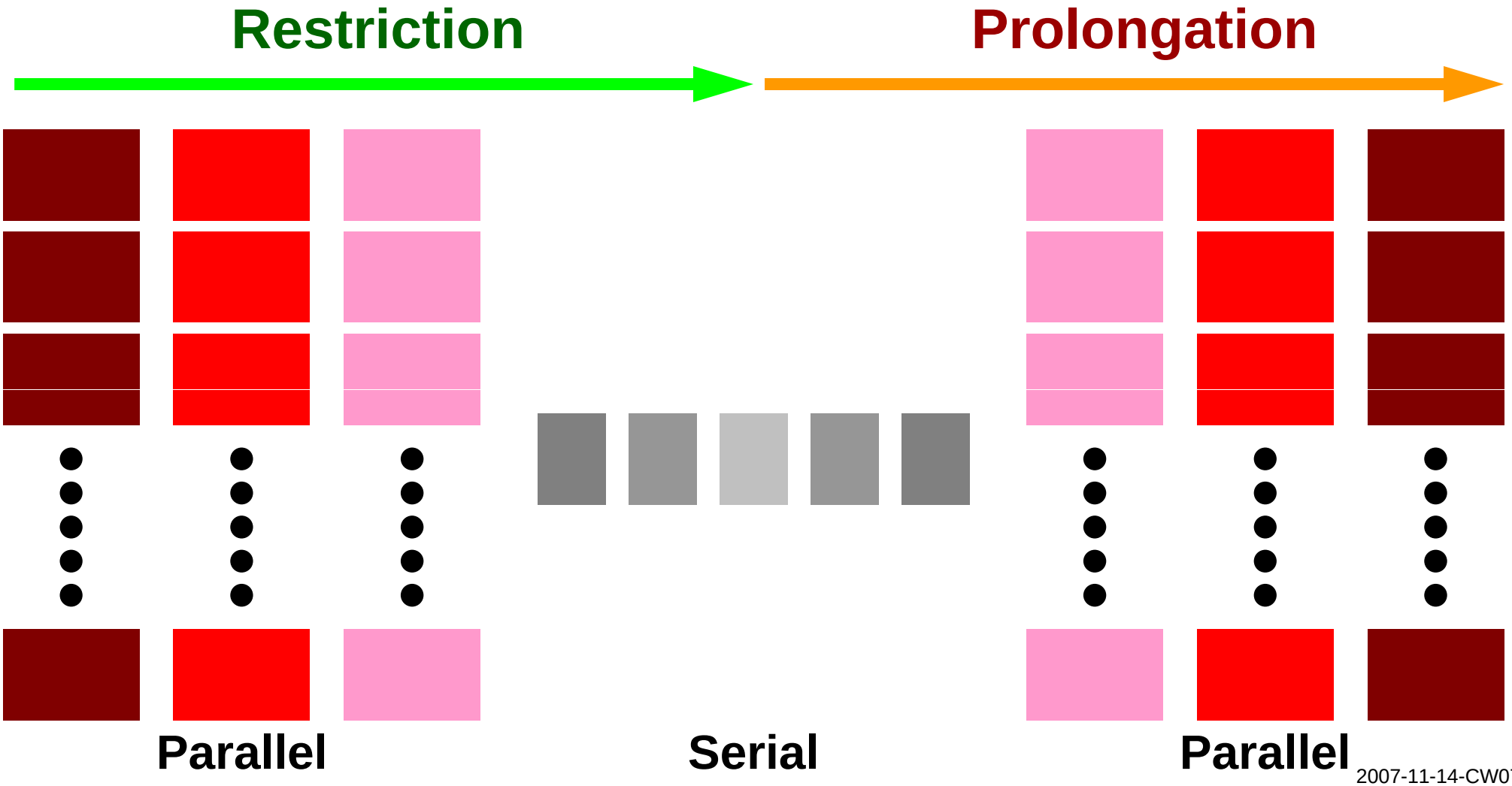




- レベル k+1 における解 u_{k+1} から，延長補間オペレータ $R_{k+1 \Rightarrow k}$ によって，レベル k における修正量 $\Delta u_k = R_{k+1 \Rightarrow k}\ u_{k+1}$ を求める。
- レベル k の解を修正量によって更新する：$u_k = u_k + \Delta u_k$
- レベル k における解 u_k から，延長補間オペレータ $R_{k \Rightarrow k-1}$ によって，レベル k-1 における修正量 $\Delta u_{k-1} = R_{k \Rightarrow k-1}\ u_k$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

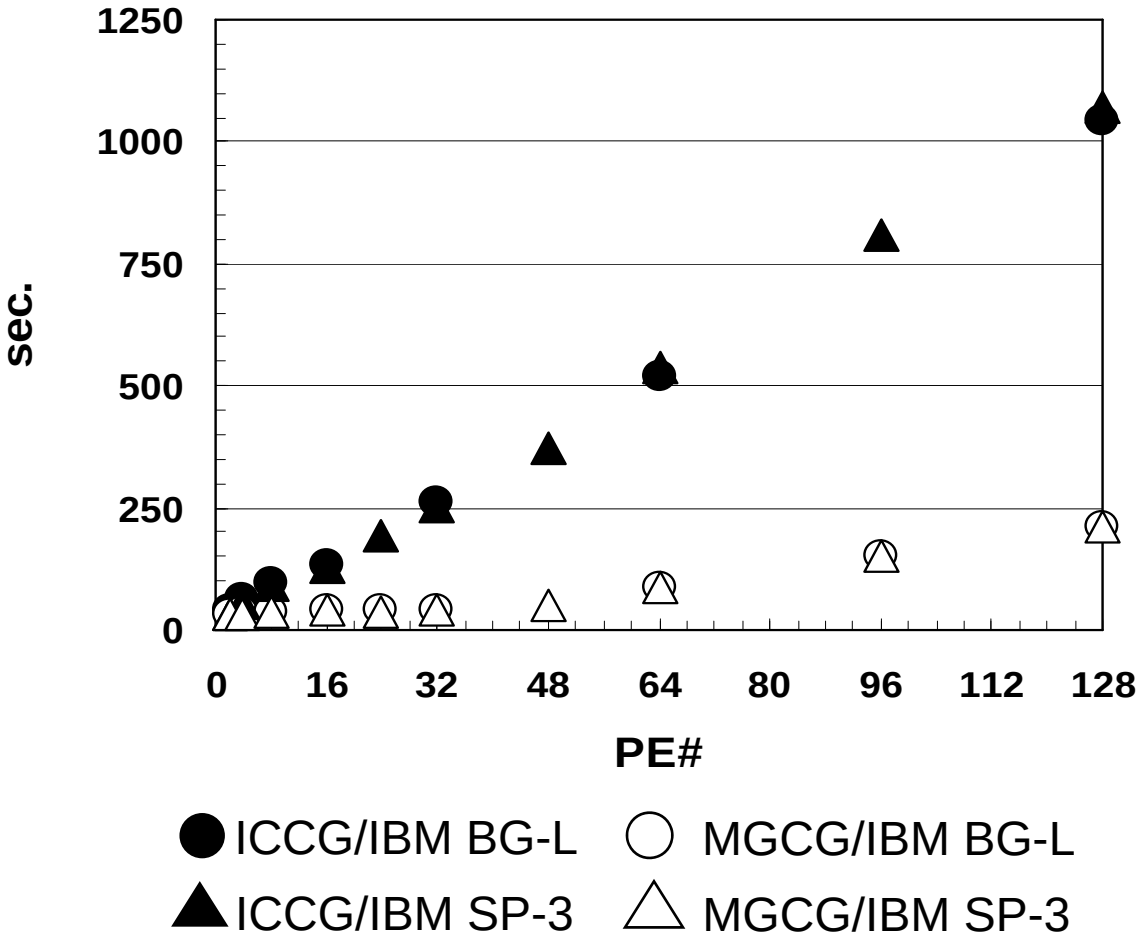
並列，シリアル計算の切り替え

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



並列計算結果（128プロセッサ）

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。