全国共同利用施設
東京大学情報基盤センター
Information Technology Center, The University of Tokyo

東京大学
THE UNIVERSITY OF TOKYO

Iterative Linear Solvers for Sparse Matrices

Kengo Nakajima

Information Technology Center, The University of Tokyo, Japan

RIKEN AICS Spring School 2014
March 5-7, 2014

TOC

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

Goal

- Introduction to Parallel Iterative Solvers

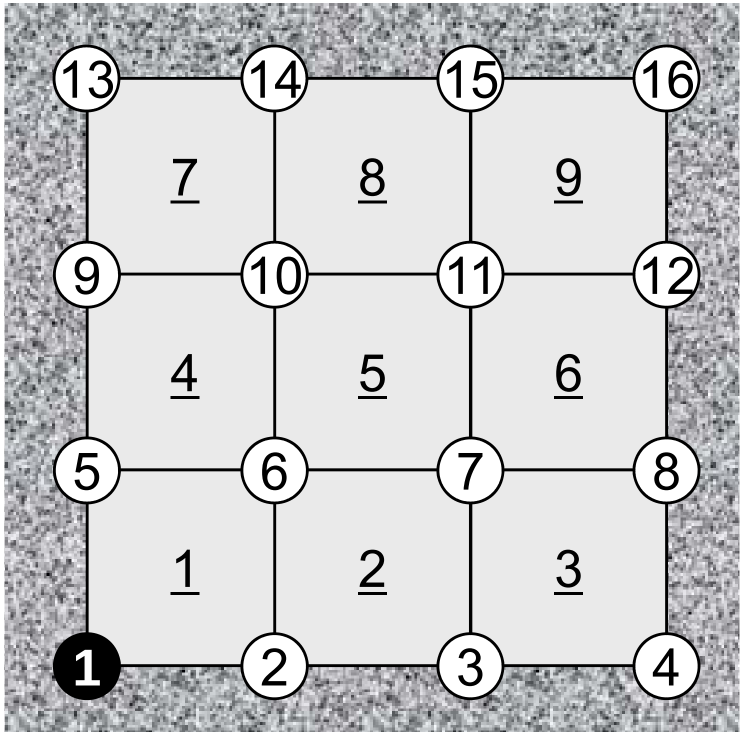
There are a lot of topics and issues all of which I cannot cover. I just try to talk about my experiences in the area of scientific applications and parallel numerical algorithms, with some general introductions.

Finite-Element Method (FEM)

- One of the most popular numerical methods for solving PDE’s.
  - elements (meshes) & nodes (vertices)
- Consider the following 2D heat transfer problem:

$$\lambda\left(\frac{\partial^2 T}{\partial x^2}+\frac{\partial^2 T}{\partial y^2}\right)+Q=0$$

  - 16 nodes, 9 bi-linear elements
  - uniform thermal conductivity (λ=1)
  - uniform volume heat flux (Q=1)
  - T=0 at node 1
  - Insulated boundaries



Galerkin FEM procedures

- Apply Galerkin procedures to each element:

$$\int_V [N]^T \left\{ \lambda\left(\frac{\partial^2 T}{\partial x^2} + \frac{\partial^2 T}{\partial y^2} \right) + Q \right\} dV = 0$$

where $T = [N]\{\phi\}$ in each elem.

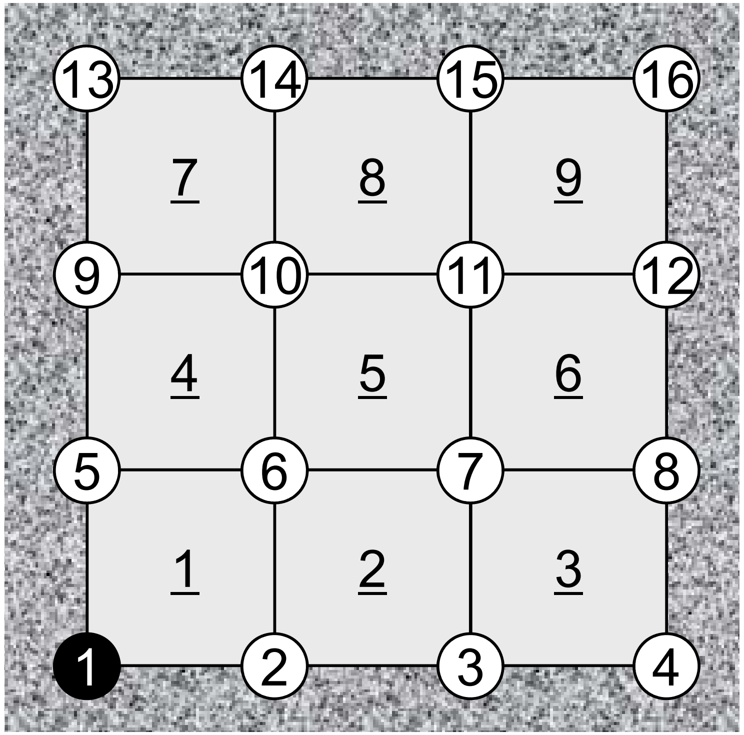
$\{\phi\}$: T at each vertex

$[N]$: Shape function (Interpolation function)

- Introduce the following "weak form" of original PDE using Green's theorem:

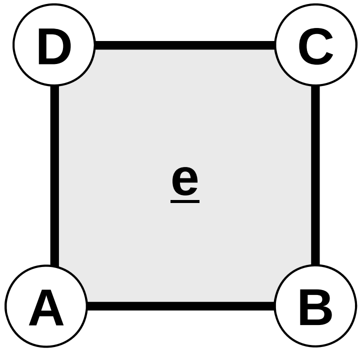
$$-\int_V \lambda \left(\frac{\partial [N]^T}{\partial x} \frac{\partial [N]}{\partial x} + \frac{\partial [N]^T}{\partial y} \frac{\partial [N]}{\partial y} \right) dV \cdot \{\phi\}$$

$$+ \int_V Q[N]^T dV = 0$$



Element Matrix

- Apply the integration to each element and form “element” matrix.



$$-\int_V \lambda\left(\frac{\partial[N]^T}{\partial x}\frac{\partial[N]}{\partial x}+\frac{\partial[N]^T}{\partial y}\frac{\partial[N]}{\partial y}\right)dV\cdot\{\phi\}$$

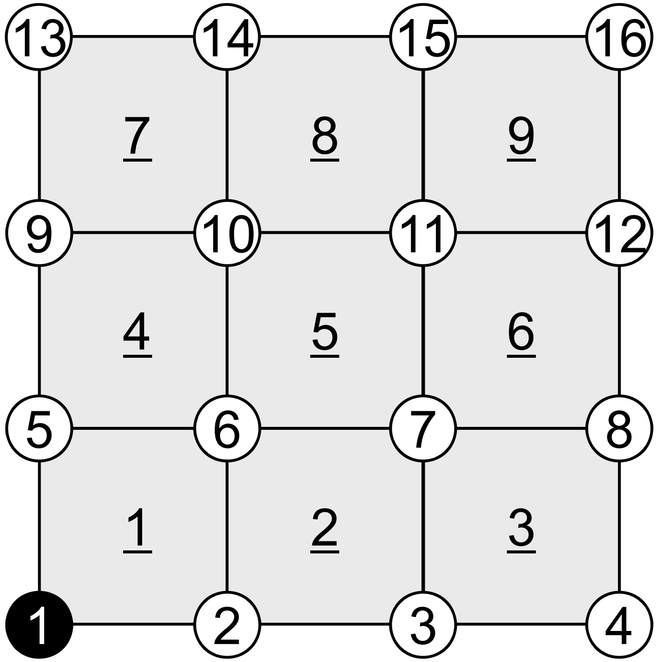
$$+\int_V Q[N]^T dV = 0$$

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

$$\longrightarrow \begin{bmatrix} k_{AA}^{(e)} & k_{AB}^{(e)} & k_{AC}^{(e)} & k_{AD}^{(e)} \\ k_{BA}^{(e)} & k_{BB}^{(e)} & k_{BC}^{(e)} & k_{BD}^{(e)} \\ k_{CA}^{(e)} & k_{CB}^{(e)} & k_{CC}^{(e)} & k_{CD}^{(e)} \\ k_{DA}^{(e)} & k_{DB}^{(e)} & k_{DC}^{(e)} & k_{DD}^{(e)} \end{bmatrix}\begin{Bmatrix} \phi_A^{(e)} \\ \phi_B^{(e)} \\ \phi_C^{(e)} \\ \phi_D^{(e)} \end{Bmatrix}=\begin{Bmatrix} f_A^{(e)} \\ f_B^{(e)} \\ f_C^{(e)} \\ f_D^{(e)} \end{Bmatrix}$$

Global (Overall) Matrix

Accumulate each element matrix to “global” matrix.

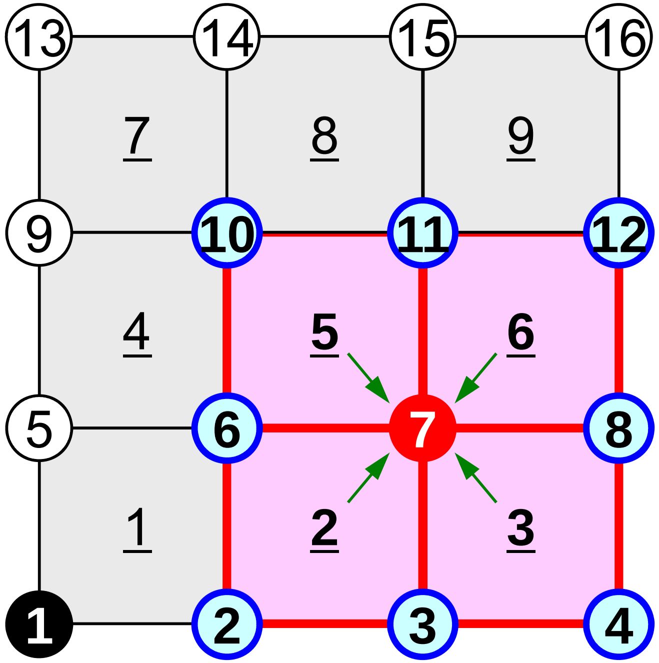


$$[K]\{\Phi\}=\{F\}$$

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

To each node …

Effect of surrounding elem’s/nodes are accumulated.



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

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

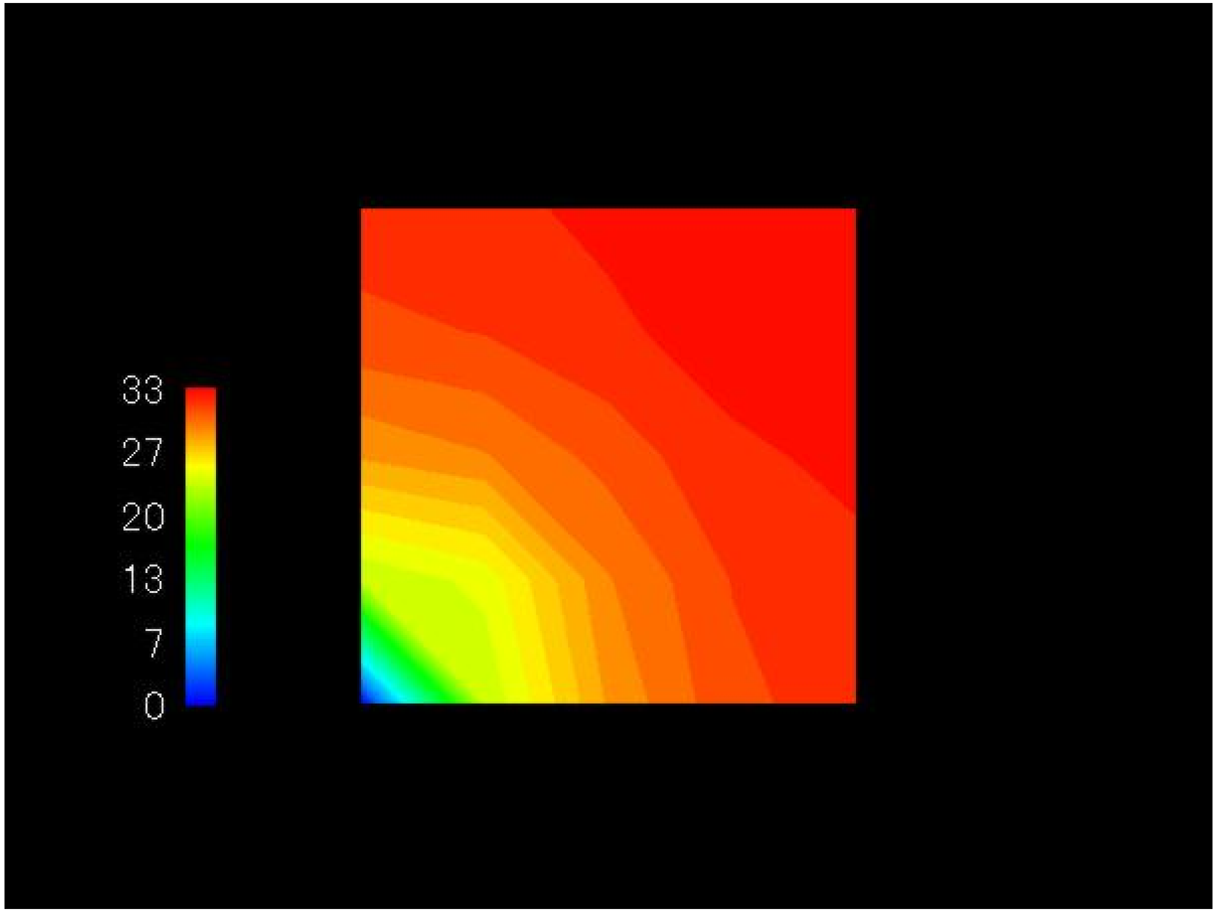
Solve the obtained global eqn’s

under certain boundary conditions

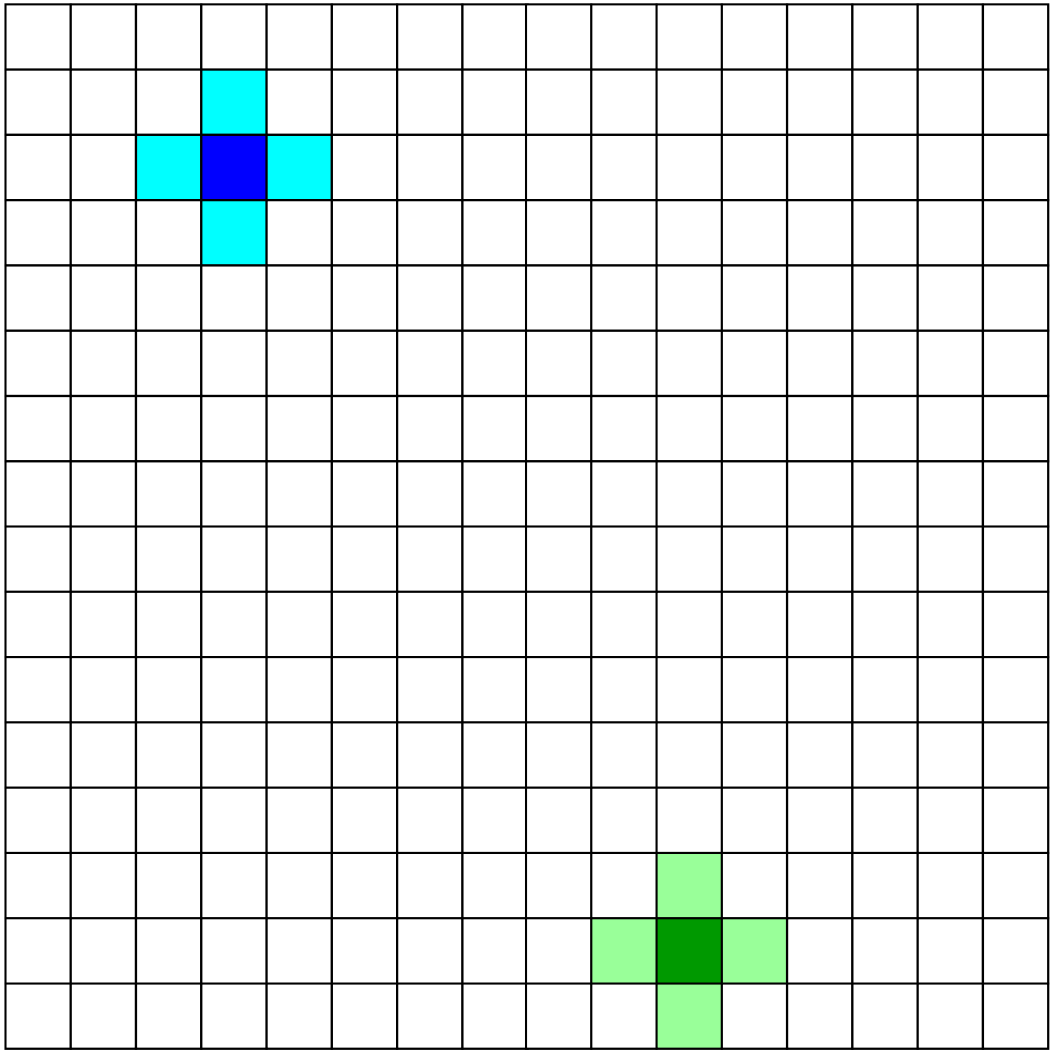
(Φ_1=0 in this case)

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

Result …



2D FDM Mesh (5-point stencil)



Coef. Matrix derived from FDM/FEM

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

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

- Number of non-zero off-diagonal components is O(10^2) in FEM
  - If number of unknowns is 10^8 :
    - A(*i*,*j*): $O(10^{16})$ words
    - Actual Non-zero Components : $O(10^{10})$ words
- Only (really) non-zero off-diag. components should be stored on memory

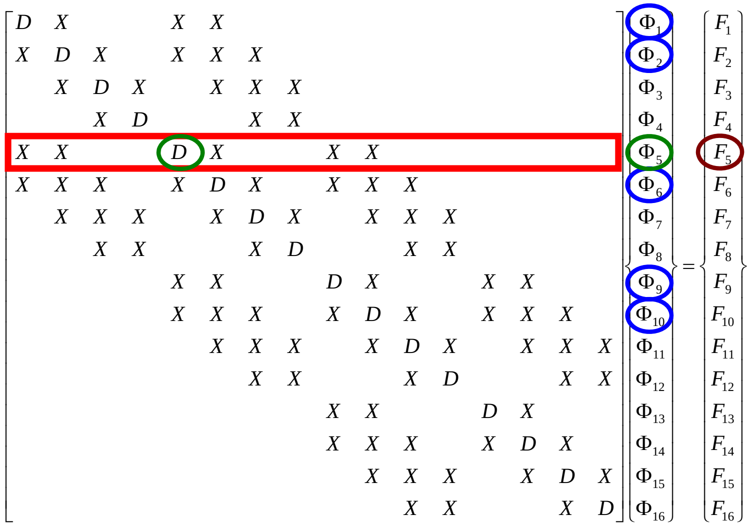
Mat-Vec. Multiplication for Sparse Matrix

Memory-Bound Process

Compressed Row Storage (CRS)

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

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

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



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

CRS or CSR ?
for Compressed Row Storage

- In Japan and USA, “CRS” is very general for abbreviation of “Compressed Row Storage”, but they usually use “CSR” in Europe (especially in France).

- “CRS” in France
  - Compagnie Républicaine de Sécurité
    - Republic Security Company of France
- French scientists may feel uncomfortable when we use “CRS” in technical papers and/or presentations.

Mat-Vec. Multiplication for Dense Matrix

Very Easy, Straightforward

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

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

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

Compressed Row Storage (CRS)

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

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

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

```
N= 8
```

Diagonal Components

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

Compressed Row Storage (CRS)：C

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

Compressed Row Storage (CRS)：C

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

Diag (i) Diagonal Components (REAL, i=1~N)

Index(i) Number of Non-Zero Off-Diagonals at Each ROW (INT, i=0~N)

Item(k) Off-Diagonal Components (Corresponding Column ID) （INT, k=1, index(N)）

AMat(k) Off-Diagonal Components (Value) （ REAL, k=1, index(N) ）

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

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

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

Large-Scale Linear Equations in Scientific Applications

- Solving large-scale linear equations **Ax=b** is the most important and **expensive** part of various types of scientific computing.
  - for both linear and nonlinear applications
- Various types of methods proposed & developed.
  - for dense and sparse matrices
  - classified into ***direct*** and ***iterative*** methods
- Dense Matrices : Globally Coupled Problems
  - BEM, Spectral Methods, MO/MD (gas, liquid)
- Sparse Matrices : Locally Defined Problems
  - FEM, FDM, DEM, MD (solid), BEM w/FMP

直接法（Direct Method）

- Gaussの消去法，完全LU分解他
  - 行列の変形，逆行列に相当するものの計算

- 利点
  - 安定，幅広いアプリケーションに適用可能
    - 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

反復法とは・・・

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(converge)させていく

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

反復法（Iterative Method）

$$\mathbf{Ax} = \mathbf{b} \Rightarrow$$

$$\mathbf{x}^{(k+1)} = \mathbf{Mx}^{(k)} + \mathbf{Nb}$$

- 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。
    - 収束しない（答えが得られない）可能性がある
  - 前処理（preconditioning）が重要。

非定常反復法：クリロフ部分空間法（1/2）
Krylov Subspace Method

$$\mathbf{Ax} = \mathbf{b} \Rightarrow \mathbf{x} = \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}$$

以下の反復式を導入し$\mathbf{x_0}$, $\mathbf{x_1}$, $\mathbf{x_2}$, ..., $\mathbf{x_k}$を求める：

$$\begin{aligned}\mathbf{x}_k &= \mathbf{b} + (\mathbf{I} - \mathbf{A})\mathbf{x}_{k-1} \\ &= (\mathbf{b} - \mathbf{Ax}_{k-1}) + \mathbf{x}_{k-1} \\ &= \mathbf{r}_{k-1} + \mathbf{x}_{k-1}\end{aligned}$$

$where\ \mathbf{r}_k = \mathbf{b} - \mathbf{Ax}_k$:残差ベクトル（residual）

↓

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1} \mathbf{r}_i$$

$$\begin{aligned}\mathbf{r}_k &= \mathbf{b} - \mathbf{Ax}_k = \mathbf{b} - \mathbf{A}(\mathbf{r}_{k-1} + \mathbf{x}_{k-1}) \\ &= (\mathbf{b} - \mathbf{Ax}_{k-1}) - \mathbf{Ar}_{k-1} = \mathbf{r}_{k-1} - \mathbf{Ax}_{k-1}\mathbf{r}_{k-1} = (\mathbf{I} - \mathbf{A})\mathbf{r}_{k-1}\end{aligned}$$

非定常反復法：クリロフ部分空間法（2/2）
Krylov Subspace Method

$$\mathbf{x}_k = \mathbf{x}_0 + \sum_{i=0}^{k-1}\mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=0}^{k-2}(\mathbf{I}-\mathbf{A})\mathbf{r}_i = \mathbf{x}_0 + \mathbf{r}_0 + \sum_{i=1}^{k-1}(\mathbf{I}-\mathbf{A})^i\mathbf{r}_0$$

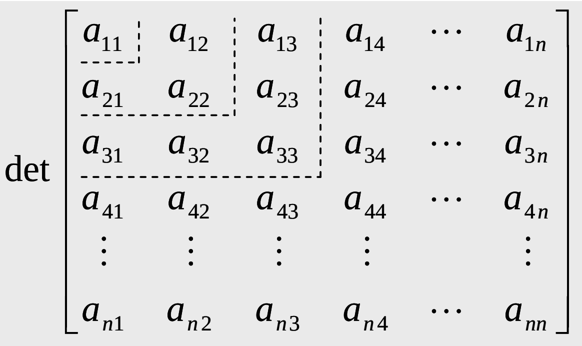
$$\mathbf{z}_k = \mathbf{r}_0 + \sum_{i=1}^{k-1}(\mathbf{I}-\mathbf{A})^i\mathbf{r}_0 = \left[\mathbf{I} + \sum_{i=1}^{k-1}(\mathbf{I}-\mathbf{A})^i\right]\mathbf{r}_0$$

$\mathbf{z}_\mathbf{k}$はk次のクリロフ部分空間（Krylov Subspace）に属するベクトル，問題はクリロフ部分空間からどのようにして解の近似ベクトル$\mathbf{x}_\mathbf{k}$を求めるかにある：

$$\left[\mathbf{r}_0, \mathbf{A}\mathbf{r}_0, \mathbf{A}^2\mathbf{r}_0, \ldots, \mathbf{A}^{k-1}\mathbf{r}_0\right]$$

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（主小行列式・首座行列式）>0と同値
- アルゴリズム
  - 最急降下法（Steepest Descent Method）の変種
  - $\mathbf{x^{(i)}= x^{(i-1)} + \alpha_i p^{(i)}}$
    - $x^{(i)}$：反復解，$p^{(i)}$：探索方向，α_i：定数）
  - 厳密解をyとするとき $\{x\text{-}y\}^T[A]\{x\text{-}y\}$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）



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

共役勾配法（CG法）のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減（DAXPY）

$x^{(i)}$: Vector

α_i : Scalar

共役勾配法（CG法）のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減（DAXPY）

$x^{(i)}$: Vector

α_i : Scalar

共役勾配法（CG法）のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減（DAXPY）

$x^{(i)}$: Vector

α_i : Scalar

共役勾配法（CG法）のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減（DAXPY）
  - Double
  - {y}= a{x} + {y}

$x^{(i)}$: Vector

α_i : Scalar

共役勾配法（CG法）のアルゴリズム

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

$x^{(i)}$: Vector

α_i : Scalar

CG法アルゴリズムの導出(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：

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

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

$$= (x, Ax) - 2(x, Ay) + (y, Ay) = (x, Ax) - 2(x, b) + \underline{(y, b)}$$

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

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

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

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

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

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

CG法アルゴリズムの導出(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:

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

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

$r^{(k)} = b - Ax^{(k)}$ は第k近似に対する残差

CG法アルゴリズムの導出(3/5)

残差 $r^{(k)}$も以下の式によって計算できる：

$$r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$$ **(2)**

$$r^{(k+1)} = b - Ax^{(k+1)},\ r^{(k)} = b - Ax^{(k)}$$

$$r^{(k+1)} - r^{(k)} = Ax^{(k+1)} - Ax^{(k)} = \alpha_k A p^{(k)}$$

探索方向を以下の漸化式によって求める：

$$p^{(k+1)} = r^{(k+1)} + \beta_k p^{(k)},\ r^{(0)} = p^{(0)}$$ **(3)**

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･･･

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

CG法アルゴリズムの導出(4/5)

ところで，下式のような都合の良い直交関係がある：

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

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

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

従って以下が成立する：

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

CG法アルゴリズムの導出(5/5)

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

$$\Rightarrow \beta_k = \frac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)} \qquad \textbf{(4)}$$

$\left(p^{(k+1)}, Ap^{(k)}\right) = 0$ $p^{(k)}$ と $p^{(k+1)}$ が行列Aに関して共役（conjugate）

```
Compute $p^{(0)}=r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
    calc. $\alpha_{i-1}$
    $x^{(i)}= x^{(i-1)} + \alpha_{i-1}p^{(i-1)}$
    $r^{(i)}= r^{(i-1)} - \alpha_{i-1}[A]q^{(i-1)}$

    check convergence |r|
    (if not converged)
    calc. $\beta_{i-1}$
    $p^{(i)}= r^{(i)} + \beta_{i-1} p^{(i-1)}$
end
```

$$\alpha_{i-1} = \frac{\left(p^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

$$\beta_{i-1} = \frac{-\left(r^{(i)}, Ap^{(i-1)}\right)}{\left(p^{(i-1)}, Ap^{(i-1)}\right)}$$

CG法アルゴリズム

任意の(*i,j*)に対して以下の共役関係が得られる:

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

探索方向$p^{(k)}$，残差ベクトル$r^{(k)}$についても以下の関係が成立する:

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

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（条件数が大きい場合）

Top 10 Algorithms in the 20th Century (SIAM)

http://www.siam.org/news/news.php?id=637

モンテカルロ法，シンプレックス法，クリロフ部分空間法，行列分解法，最適化Fortranコンパイラ，QR法，クイックソート，FFT，整数関係アルゴリズム，FMM（高速多重極法）

Features of CG Method

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

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

$\left(p^{(k+1)}, Ap^{(k)}\right) = 0$ $p^{(k)}$ is “conjugate” for matrix A

Following “conjugate” relationship is obtained for arbitrary (i,j):

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

Following relationships are also obtained for $p^{(k)}$ and $r^{(k)}$:

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

In N-dimensional space, only N sets of orthogonal and linearly independent residual vector $r^{(k)}$. This means CG method converges after N iterations if number of unknowns is N.

Actually, round-off error sometimes affects convergence.

Proof (1/3)

Mathematical Induction

数学的帰納法

$$\left(r^{(i)}, r^{(j)}\right) = 0 \ (i \neq j)$$ 直交性

$$\left(p^{(i)}, Ap^{(j)}\right) = 0 \ (i \neq j)$$ 共役性

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

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

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

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

Proof (2/3)

Mathematical Induction
数学的帰納法

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

(*) is satisfied for $i \le k,\ j \le k$ where $i \neq j$ または $0 \le i < j \le k$

<u>if $i < k$</u>

$$\left(r^{(k+1)}, r^{(i)}\right) = \left(r^{(i)}, r^{(k+1)}\right) \overset{(2)}{=} \left(r^{(i)}, r^{(k)} - \alpha_k A p^{(k)}\right)$$
$$\overset{(*)}{=} -\alpha_k \left(r^{(i)}, Ap^{(k)}\right) \overset{(4)}{=} -\alpha_k \left(p^{(i)} - \beta_{i-1} p^{(i-1)}, Ap^{(k)}\right)$$
$$= -\alpha_k \left(p^{(i)}, Ap^{(k)}\right) + \alpha_k \beta_{i-1} \left(p^{(i-1)}, Ap^{(k)}\right) \overset{(*)}{=} 0$$

<u>if $i = k$</u>

$$\left(r^{(k+1)}, r^{(k)}\right) \overset{(2)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(r^{(k)}, \alpha_k A p^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)} - \beta_{k-1} p^{(k-1)}, \alpha_k A p^{(k)}\right)$$
$$\overset{(*)}{=} \left(r^{(k)}, r^{(k)}\right) - \alpha_k \left(p^{(k)}, Ap^{(k)}\right) \overset{(1)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(p^{(k)}, r^{(k)}\right)$$
$$\overset{(3)}{=} \left(r^{(k)}, r^{(k)}\right) - \left(\beta_{k-1} p^{(k-1)} + r^{(k)}, r^{(k)}\right)$$
$$= -\beta_{k-1} \left(p^{(k-1)}, r^{(k)}\right) \overset{(2)}{=} -\beta_{k-1} \left(p^{(k-1)}, r^{(k-1)} - \alpha_{k-1} A p^{(k-1)}\right)$$
$$= -\beta_{k-1} \left\{ \left(p^{(k-1)}, r^{(k-1)}\right) - \alpha_{k-1} \left(p^{(k-1)}, Ap^{(k-1)}\right) \right\} \overset{(1)}{=} 0$$

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

(2) $r^{(k+1)} = r^{(k)} - \alpha_k A p^{(k)}$

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

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

Proof (3/3)

Mathematical Induction

数学的帰納法

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

$(*)$ is satisfied for $i \leq k,\ j \leq k$ where $i \neq j$ または $0 \leq i < j \leq k$

<u>if $i < k$</u>

$$\left(p^{(k+1)}, Ap^{(i)}\right) \overset{(3)}{=} \left(r^{(k+1)} + \beta_k p^{(k)}, Ap^{(i)}\right)$$
$$\overset{(*)}{=} \left(r^{(k+1)}, Ap^{(i)}\right)$$
$$\overset{(2)}{=} \frac{1}{\alpha_k}\left(r^{(k+1)}, r^{(i)} - r^{(i-1)}\right) = 0$$

<u>if $i = k$</u>

$$\left(p^{(k+1)}, Ap^{(k)}\right) \overset{(3)}{=} \left(r^{(k+1)}, Ap^{(k)}\right) + \beta_k\left(p^{(k)}, Ap^{(k)}\right)$$
$$\overset{(4)}{=} 0$$

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

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

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

(4) $\beta_k = \dfrac{-\left(r^{(k+1)}, Ap^{(k)}\right)}{\left(p^{(k)}, Ap^{(k)}\right)}$

$$\left(p^{(k)},r^{(k)}\right)\overset{(3)}{=}\left(r^{(k)}+\beta_{k-1}p^{(k-1)},r^{(k)}\right)$$

$$=\left(\beta_{k-1}p^{(k-1)},r^{(k)}\right)+\left(r^{(k)},r^{(k)}\right)=\left(r^{(k)},r^{(k)}\right)$$

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

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

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

(4) $\beta_k=\dfrac{-\left(r^{(k+1)},Ap^{(k)}\right)}{\left(p^{(k)},Ap^{(k)}\right)}$

$$\because\left(Ap^{(k)},y-x^{(k+1)}\right)=\left(p^{(k)},Ay-Ax^{(k+1)}\right)$$

$$=\left(p^{(k)},b-Ax^{(k+1)}\right)=\left(p^{(k)},r^{(k+1)}\right)=0$$

$$\left(p^{(i)},r^{(k+1)}\right)=0,\quad i=0,1,\dots,k$$

$$x^{(k+1)}=x^{(i+1)}+\sum_{j=i+1}^{k}\alpha_j p^{(j)}$$

$$r^{(k+1)}=b-Ax^{(k+1)}=b-A\left[x^{(i+1)}+\sum_{j=i+1}^{k}\alpha_j p^{(j)}\right]$$

$$=\left[b-Ax^{(i+1)}\right]-\sum_{j=i+1}^{k}\alpha_j Ap^{(j)}=r^{(i+1)}-\sum_{j=i+1}^{k}\alpha_j Ap^{(j)}$$

$$\left(p^{(i)},r^{(k+1)}\right)=\left(p^{(i)},r^{(i+1)}-\sum_{j=i+1}^{k}\alpha_j Ap^{(j)}\right)$$

$$=\underbrace{\left(p^{(i)},r^{(i+1)}\right)}_{=0}-\underbrace{\left(p^{(i)},\sum_{j=i+1}^{k}\alpha_j Ap^{(j)}\right)}_{=0}=0$$

α_k, β_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:

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

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

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

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

共役勾配法（CG法）のアルゴリズム

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

$x^{(i)}$: Vector

α_i : Scalar

$$\beta_{i-1} = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(r^{(i-2)}, r^{(i-2)}\right)} \qquad \begin{matrix}(= \rho_{i-1})\\ (= \rho_{i-2})\end{matrix}$$

$$\alpha_i = \frac{\left(r^{(i-1)}, r^{(i-1)}\right)}{\left(p^{(i)}, Ap^{(i)}\right)} \qquad (= \rho_{i-1})$$

プログラム例（CG法）　（1/3）

```
do i= 1, N
  R(i)= B(i)
  do j= 1, N
    R(i)= R(i) - AMAT(i,j)*X(j)
  enddo
enddo

BNRM2= 0.0D0
do i= 1, N
  BNRM2= BNRM2  + B(i)  **2
enddo
```

AMAT(i,j): Aのa_{ij}成分
B(i):　bの各成分
X(i):　xの各成分

P(i):　pの各成分
Q(i):　qの各成分
R(i):　rの各成分

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

プログラム例（CG法）　（2/3）

```
  do iter= 1, ITERmax
    RHO= 0.d0
    do i= 1, N
      RHO= RHO + R(i)*R(i)
    enddo

    if ( iter.eq.1 ) then
      do i= 1, N
        P(i)= R(i)
      enddo
     else
      BETA= RHO / RHO1
      do i= 1, N
        P(i)= R(i) + BETA*P(i)
      enddo
    endif

    do i= 1, N
      Q(i)= 0.d0
      do j= 1, N
        Q(i)= Q(i) + AMAT(i,j)*P(j)
      enddo
    enddo

...
  enddo
```

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

プログラム例（CG法）（3/3）

```
  do iter= 1, ITERmax
...
    C1= 0.d0
    do i= 1, N
      C1= C1 + P(i)*Q(i)
    enddo
    ALPHA= RHO / C1

    do i= 1, N
      X(i)= X(i) + ALPHA * P(i)
      R(i)= R(i) - ALPHA * Q(i)
    enddo

    DNRM2 = 0.0
    do i= 1, N
      DNRM2= DNRM2 + R(i)**2
    enddo

    RESID= dsqrt(DNRM2/BNRM2)

    if ( RESID.le.EPS) exit

    RHO1 = RHO

  enddo
```

$$\frac{\left\|\mathbf{b}-\mathbf{A}\mathbf{x}^{(k)}\right\|_2}{\left\|\mathbf{b}\right\|_2} < \varepsilon$$

$\rho_{i-1}=\rho_{i-2}$

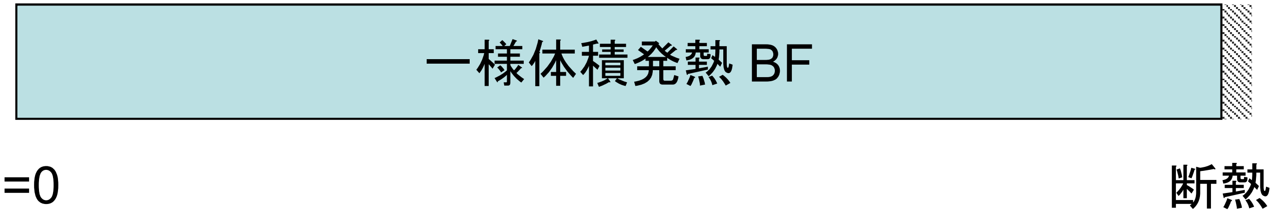
```
Compute $r^{(0)}= b-[A]x^{(0)}$
for i= 1, 2, …
    $\rho_{i-1}= r^{(i-1)} r^{(i-1)}$
    if i=1
      $p^{(1)}= r^{(0)}$
     else
      $\beta_{i-1}= \rho_{i-1}/\rho_{i-2}$
      $p^{(i)}= r^{(i-1)} + \beta_{i-1} p^{(i-1)}$
    endif
    $q^{(i)}= [A]p^{(i)}$
    $\alpha_i = \rho_{i-1}/p^{(i)}q^{(i)}$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

一次元熱伝導方程式

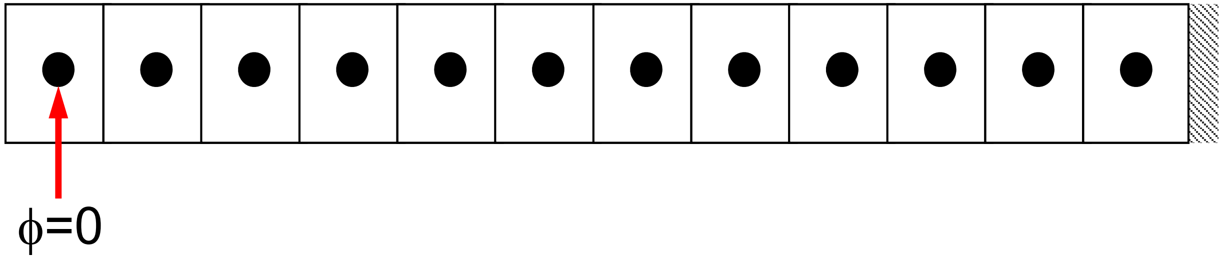
支配方程式：熱伝導率＝1（一様）

$$\frac{d^2\phi}{dx^2}+BF=0,\quad \phi=0@x=0,\frac{d\phi}{dx}=0@x=x_{\max}$$

$$\phi=-\frac{1}{2}BF\,x^2+BF\,x_{\max}\,x$$



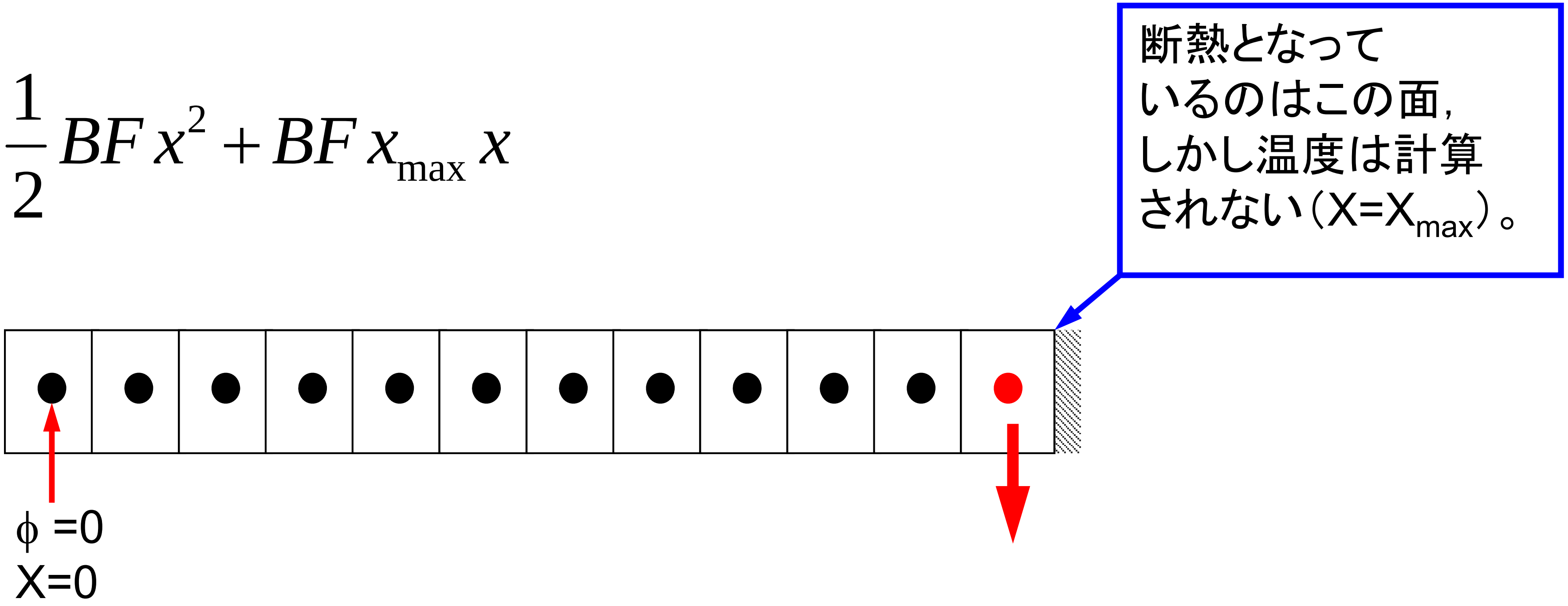
以下のような離散化（要素中心で従属変数を定義）をしている



一次元熱伝導方程式

解析解

$$\phi = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$



断熱となっている
いるのはこの面，
しかし温度は計算
されない（X=X_{max}）。

ϕ =0
X=0

Δx=1.d0，メッシュ数=50，とすると，X_{max}=49.5，
●の点のX座標は49.0となる。BF=1.0d0とすると●での温度は：

$$\phi = -\frac{1}{2} 49^2 + 49.5 \times 49 = -1200.5 + 9850.5 = 1225$$

計算例（N=50）：Jacobi法

```
 1000 iters, RESID=    5.443248E-01 PHI(N)=     4.724513E+02
 2000 iters, RESID=    3.255667E-01 PHI(N)=     7.746137E+02
 3000 iters, RESID=    1.947372E-01 PHI(N)=     9.555996E+02
...
34000 iters, RESID=    2.347113E-08 PHI(N)=     1.225000E+03
35000 iters, RESID=    1.403923E-08 PHI(N)=     1.225000E+03
35661 iters, RESID=    9.999053E-09 PHI(N)=     1.225000E+03

1        0.000000E+00     0.000000E+00
2        4.899999E+01     4.900000E+01
3        9.699999E+01     9.700000E+01
4        1.440000E+02     1.440000E+02
5        1.900000E+02     1.900000E+02
...

41       1.180000E+03     1.180000E+03
42       1.189000E+03     1.189000E+03
43       1.197000E+03     1.197000E+03
44       1.204000E+03     1.204000E+03
45       1.210000E+03     1.210000E+03
46       1.215000E+03     1.215000E+03
47       1.219000E+03     1.219000E+03
48       1.222000E+03     1.222000E+03
49       1.224000E+03     1.224000E+03
50       1.225000E+03     1.225000E+03
```

反復回数
最大残差
ϕ(50)

数値解，解析解

$$\phi = -\frac{1}{2}49^2 + 49.5 \times 49 = -1200.5 + 9850.5 = 1225$$

計算例（N=50）：Gauss-Seidel 法

```
 1000 iters, RESID=    3.303725E-01 PHI(N)=     7.785284E+02
 2000 iters, RESID=    1.182010E-01 PHI(N)=     1.065259E+03
 3000 iters, RESID=    4.229019E-02 PHI(N)=     1.167848E+03
...
16000 iters, RESID=    6.657001E-08 PHI(N)=     1.225000E+03
17000 iters, RESID=    2.381754E-08 PHI(N)=     1.225000E+03
17845 iters, RESID=    9.993196E-09 PHI(N)=     1.225000E+03
```

反復回数
最大残差
φ(50)

```
1        0.000000E+00     0.000000E+00
2        4.899999E+01     4.900000E+01
3        9.699999E+01     9.700000E+01
4        1.440000E+02     1.440000E+02
5        1.900000E+02     1.900000E+02
...

41       1.180000E+03     1.180000E+03
42       1.189000E+03     1.189000E+03
43       1.197000E+03     1.197000E+03
44       1.204000E+03     1.204000E+03
45       1.210000E+03     1.210000E+03
46       1.215000E+03     1.215000E+03
47       1.219000E+03     1.219000E+03
48       1.222000E+03     1.222000E+03
49       1.224000E+03     1.224000E+03
50       1.225000E+03     1.225000E+03
```

数値解，解析解

計算例（N=50）：CG法

```
    49 iters, RESID=    0.000000E-00 PHI(N)=     1.225000E+03

1          0.000000E+00      0.000000E+00
2          4.899999E+01      4.900000E+01
3          9.699999E+01      9.700000E+01
4          1.440000E+02      1.440000E+02
5          1.900000E+02      1.900000E+02
...

41         1.180000E+03      1.180000E+03
42         1.189000E+03      1.189000E+03
43         1.197000E+03      1.197000E+03
44         1.204000E+03      1.204000E+03
45         1.210000E+03      1.210000E+03
46         1.215000E+03      1.215000E+03
47         1.219000E+03      1.219000E+03
48         1.222000E+03      1.222000E+03
49         1.224000E+03      1.224000E+03
50         1.225000E+03      1.225000E+03
```

数値解，解析解

反復回数
最大残差
ϕ(50)

49回目に収束していることに注意（未知数は49個）

$$\phi = -\frac{1}{2}49^2 + 49.5 \times 49 = -1200.5 + 9850.5 = 1225$$

SOR法（Successive Over-Relaxation）

逐次加速緩和法

ヤコビ法

$$\mathbf{M} = -\mathbf{D}^{-1}(\mathbf{L}+\mathbf{U}), \quad \mathbf{N} = \mathbf{D}^{-1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{b} - (\mathbf{L}+\mathbf{U})\mathbf{x}^{(k)}\right]$$

ガウス・ザイデル法

$$\mathbf{M} = -(\mathbf{D}+\mathbf{L})^{-1}\mathbf{U}, \quad \mathbf{N} = (\mathbf{D}+\mathbf{L})^{-1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right]$$

SOR法

$$\mathbf{M} = (\mathbf{D}+\omega\mathbf{L})^{-1}\{(1-\omega)\mathbf{D} - \omega\mathbf{U}\}, \quad \mathbf{N} = \omega(\mathbf{D}+\omega\mathbf{L})^{-1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega\left[\xi^{(k+1)} - \mathbf{x}^{(k)}\right] \quad (0 < \omega < 2)$$

$$\xi^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right]$$ ガウス・ザイデル法の解

ω=1の場合，ガウス・ザイデル法と一致

プログラム例（ガウス・ザイデル法）

```
do iter= 1, N*100
  do i= 1, N
    RESID= B(i)
    do j= 1, N
      if (j.ne.i) then
        RESID= RESID - A(i,j)*X(j)
      endif
    enddo
    X(i)= RESID/A(i,i)
  enddo
enddo
```

$$\mathbf{M} = \left(\mathbf{D} + \mathbf{L}\right)^{-1}\mathbf{U}, \quad \mathbf{N} = \left(\mathbf{D} + \mathbf{L}\right)^{-1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right]$$

A(i,j): **A**のa_{ij}成分

B(i): **b**の各成分

X(i): **x**の各成分

プログラム例（SOR法）

```
do iter= 1, N*100
  do i= 1, N
    RESID= B(i)
    do j= 1, N
      if (j.ne.i) then
        RESID= RESID - A(i,j)*X(j)
      endif
    enddo
    X(i)= OMEGA*(RESID/A(i,i)-X(i)) + X(i)
  enddo
enddo
```

$$\mathbf{M} = (\mathbf{D} + \omega\mathbf{L})^{-1}\{(1-\omega)\mathbf{D} - \omega\mathbf{U}\}$$

$$\mathbf{N} = \omega(\mathbf{D} + \omega\mathbf{L})^{-1}$$

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega\left[\xi^{(k+1)} - \mathbf{x}^{(k)}\right]$$

$$\xi^{(k+1)} = \mathbf{D}^{-1}\left[\mathbf{b} - \mathbf{L}\mathbf{x}^{(k+1)} - \mathbf{U}\mathbf{x}^{(k)}\right]$$

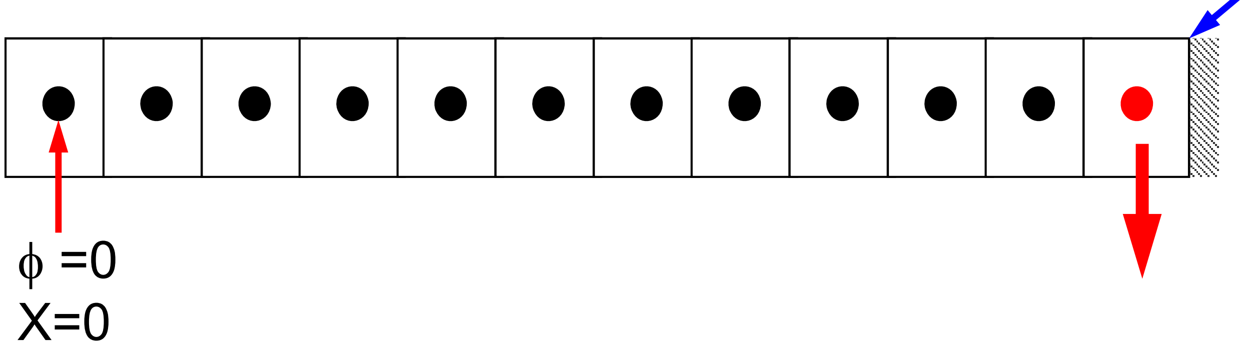
A(i,j): **A**のa_{ij}成分
B(i): **b**の各成分
X(i): **x**の各成分

一次元熱伝導方程式

解析解

$$\phi = -\frac{1}{2} BF\, x^2 + BF\, x_{\max}\, x$$

断熱となって
いるのはこの面，
しかし温度は計算
されない（X=X$_{max}$）。



Δx=1.d0，メッシュ数=50，とすると，X$_{max}$=49.5，
●の点のX座標は49.0となる。BF=1.0d0とすると●での温度は：

$$\phi = -\frac{1}{2} 49^2 + 49.5 \times 49 = -1200.5 + 9850.5 = 1225$$

	反復回数
Jacobi	35,561
Gauss-Seidel	17,845
CG	49
SOR（ω=0.70）	33,131
SOR（ω=0.80）	26,762
SOR（ω=0.90）	21,808
SOR（ω=1.00）	17,845
SOR（ω=1.30）	9,614
SOR（ω=1.50）	5,955
SOR（ω=1.60）	4,469
SOR（ω=1.70）	3,155
SOR（ω=1.80）	1,980
SOR（ω=1.90）	886

	反復回数
SOR（ω=1.91）	773
SOR（ω=1.92）	653
SOR（ω=1.93）	520
SOR（ω=1.94）	342
SOR（ω=1.95）	392
SOR（ω=1.96）	497
SOR（ω=1.97）	682
SOR（ω=1.98）	1,020
SOR（ω=1.99）	2,028
SOR（ω=2.00）	NA

ωの範囲

$$\mathbf{M} = -(\mathbf{D}+\omega\mathbf{L})^{-1}\{(1-\omega)\mathbf{D}-\omega\mathbf{U}\},\quad \mathbf{N} = \omega(\mathbf{D}+\omega\mathbf{L})^{-1}$$

$$\det(\mathbf{M}) = \prod_{i=1}^{N}\lambda_i$$ λ$_\mathrm{i}$:Mの固有値（i=1~N）

$$\begin{aligned}\det(\mathbf{M}) &= \det\left((\mathbf{D}+\omega\mathbf{L})^{-1}\{(1-\omega)\mathbf{D}-\omega\mathbf{U}\}\right)\\ &= \det\left((\mathbf{D}+\omega\mathbf{L})^{-1}\right)\cdot\det\left((1-\omega)\mathbf{D}-\omega\mathbf{U}\right)\\ &= \det\left(\mathbf{D}^{-1}\right)\cdot\det\left((1-\omega)\mathbf{D}-\omega\mathbf{U}\right)\\ &= \det\left((1-\omega)\mathbf{I}-\omega\mathbf{D}^{-1}\mathbf{U}\right) = \det\left((1-\omega)\mathbf{I}\right) = (1-\omega)^N\end{aligned}$$

$$\rho(\mathbf{M}) = \max_i|\lambda_i| \geq \sqrt[N]{|(1-\omega)|^N} = |1-\omega|$$ 再びスペクトル半径

|1-ω|<1となるので，0<ω<2，通常は1<ω<2

反復法（Iterative Method）

- 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。
    - 収束しない（答えが得られない）可能性がある
  - 前処理（preconditioning）が重要。
    - 条件数（condition number）の大きい問題

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

共役勾配法のアルゴリズム

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

- 行列ベクトル積
- ベクトル内積
- ベクトル定数倍の加減

$x^{(i)}$ ：ベクトル

α_i ：スカラー

前処理（preconditioning）とは?

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

前処理付共役勾配法

Preconditioned Conjugate Gradient Method（PCG）

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

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

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

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

ILU(0), IC(0)

- 最もよく使用されている前処理（疎行列用）
  - 不完全LU分解
    - Incomplete LU Factorization
  - 不完全コレスキー分解
    - Incomplete Cholesky Factorization（対称行列）

- 不完全な直接法
  - もとの行列が疎でも，逆行列は疎とは限らない。
  - fill-in
  - もとの行列と同じ非ゼロパターン（fill-in無し）を持っているのがILU（0），IC（0）

ファイル類 on PC FORTRANだけです

コピー，展開

http://nkl.cc.u-tokyo.ac.jp/files/ilu.tar

```
>$ cd <$CUR>

>$ tar xvf ilu.tar
>$ cd ilu

>$ ls

   lu1.f lu2.f
```

LU分解法：完全LU分解法

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

- LU factorization

「不」完全LU分解法

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

LU分解による連立一次方程式の解法

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

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

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

$$\mathrm{L:Lower\ triangular\ part\ of\ \ matrix\ A}$$
$$\mathrm{U:Upper\ triangular\ part\ of\ \ matrix\ A}$$

連立一次方程式の行列表現

n元の連立一次方程式の一般形

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

$$\Updownarrow$$

行列表現

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

LU分解を用いたAx=bの解法

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

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

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

このxが $\mathbf{Ax} = \mathbf{b}$ の解となる

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

Ly=bの解法：前進代入

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

$$\Longleftrightarrow$$

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

芋づる式に（one after another）解が求まる.

Ux=yの解法：後退代入

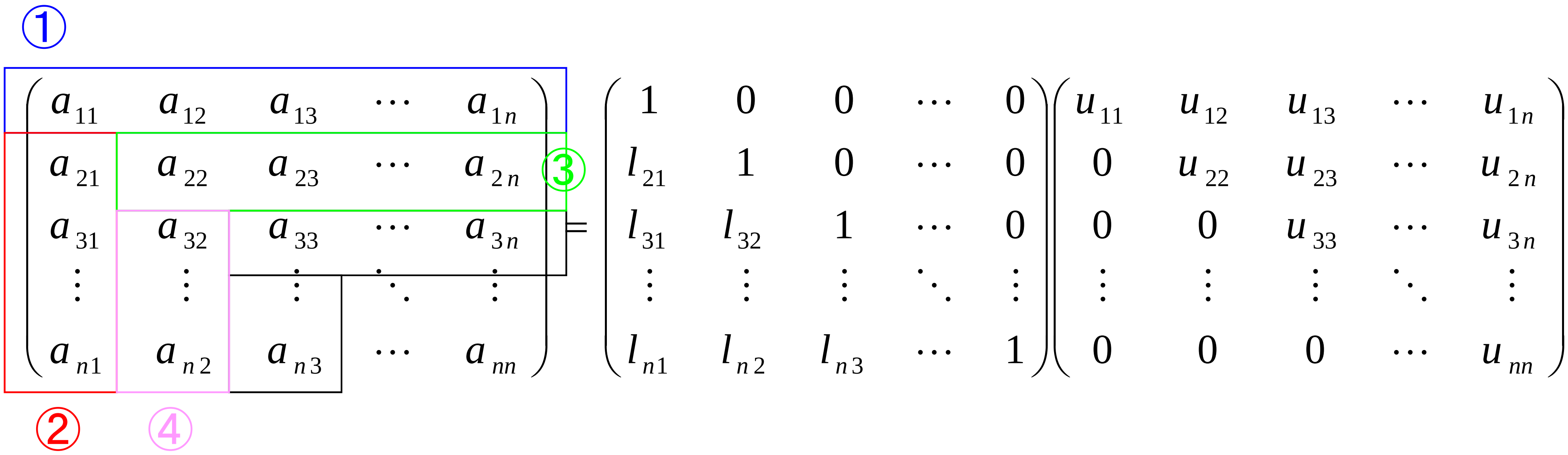
$$\mathbf{Ux} = \mathbf{y} \quad \Longleftrightarrow \quad \begin{pmatrix} u_{11} & u_{12} & \cdots & u_{1n} \\ 0 & u_{22} & \cdots & u_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} y_1 \\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

⟺

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

芋づる式に（one after another）解が求まる.

LU分解の求め方



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

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

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

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

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

数値例

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

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

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

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

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

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

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

数値例(続き）

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

第３行 ➡

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

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

第３列 ➡

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

第４行(第４列） ➡

$$1 = l_{41}u_{14} + l_{42}u_{24} + l_{43}u_{34} + u_{44} \Rightarrow u_{44} = 2$$

１行、１列、２行、２列、・・・の順に求める式を作っていく.

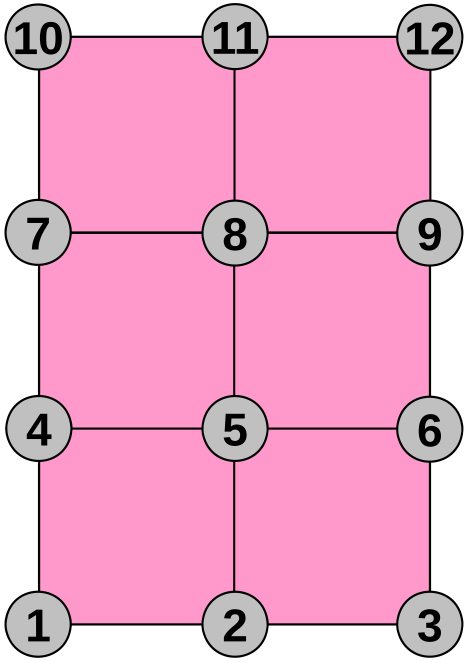
数値例(続き）

結局

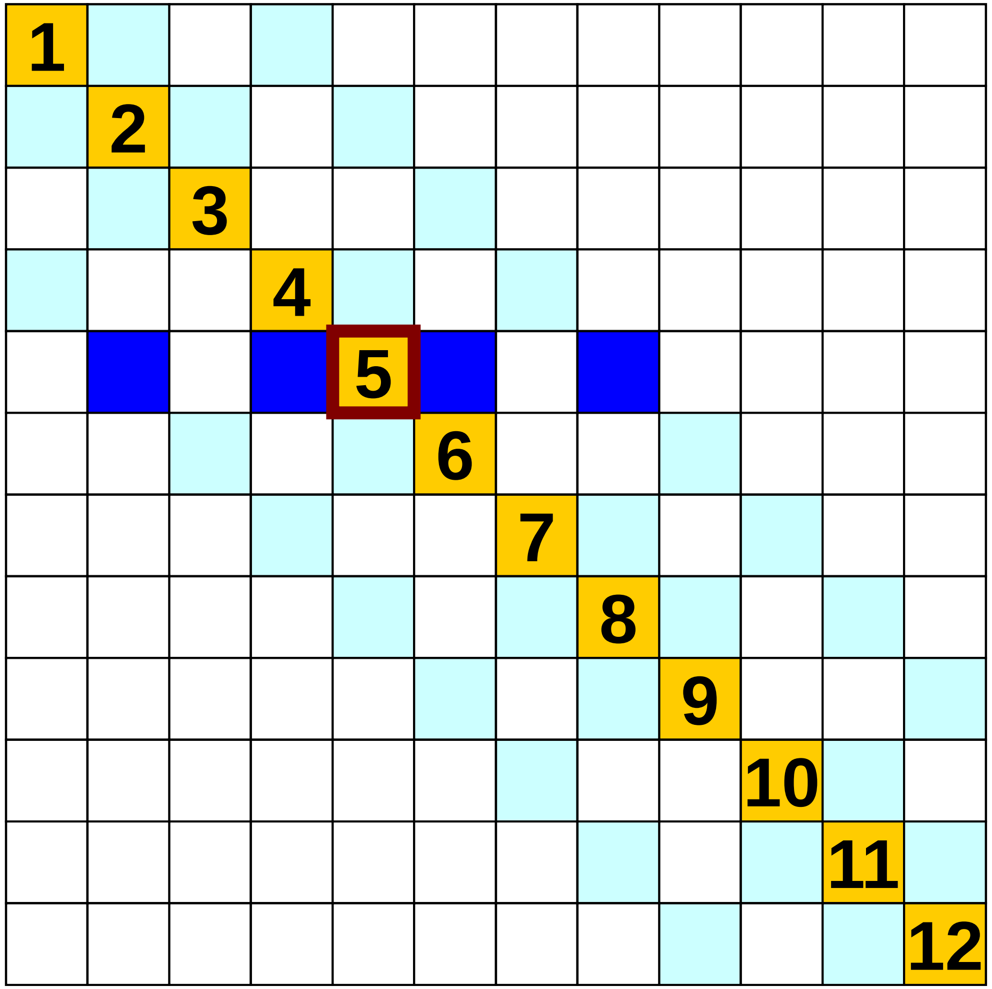
$$\mathbf{A} = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 2 & 6 & 7 & 10 \\ 2 & 2 & 8 & 7 \\ 0 & -4 & 7 & 1 \end{pmatrix} = \underset{\substack{\Big\downarrow \\ \mathbf{L}}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 2 & -1 & 1 & 0 \\ 0 & -2 & 3 & 1 \end{pmatrix}} \underset{\substack{\Big\downarrow \\ \mathbf{U}}}{\begin{pmatrix} 1 & 2 & 3 & 4 \\ 0 & 2 & 1 & 2 \\ 0 & 0 & 3 & 1 \\ 0 & 0 & 0 & 2 \end{pmatrix}}$$

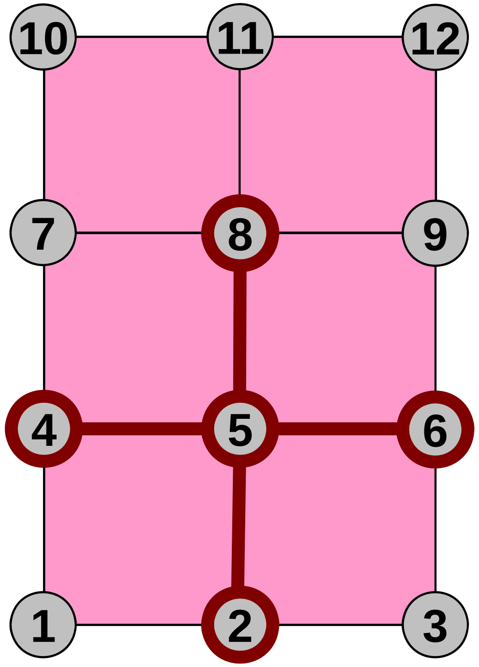
実例：5点差分

1	■		■								
■	2	■		■							
	■	3			■						
■			4	■		■					
	■		■	5	■		■				
		■		■	6			■			
			■			7	■		■		
				■		■	8	■		■	
					■		■	9			■
						■			10	■	
							■		■	11	■
								■		■	12



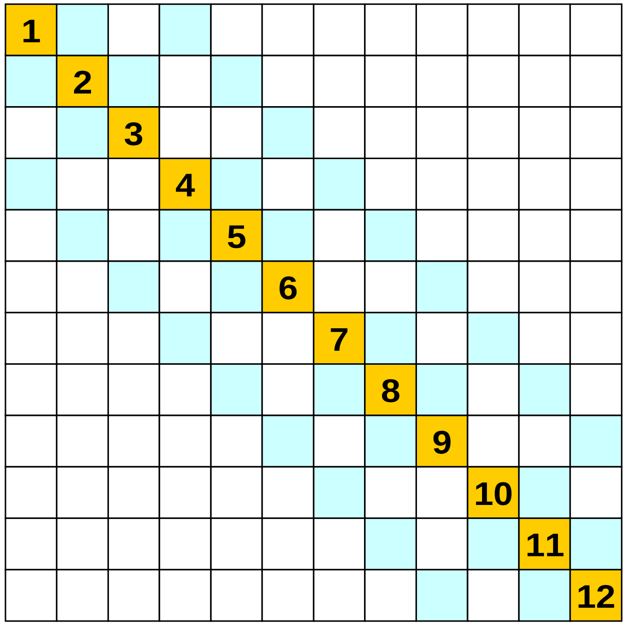
実例：5点差分

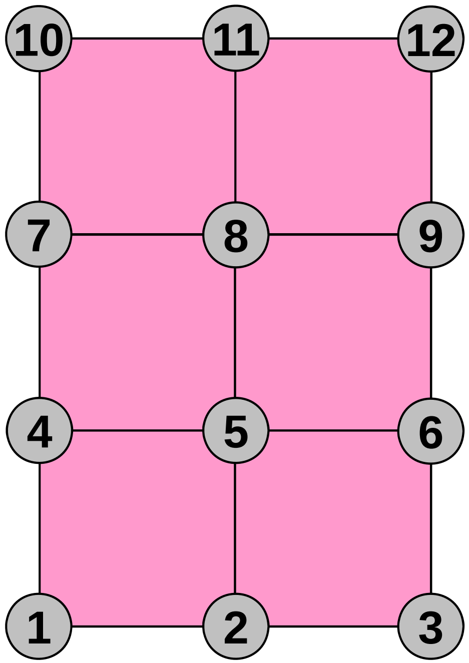




実例：係数マトリクス

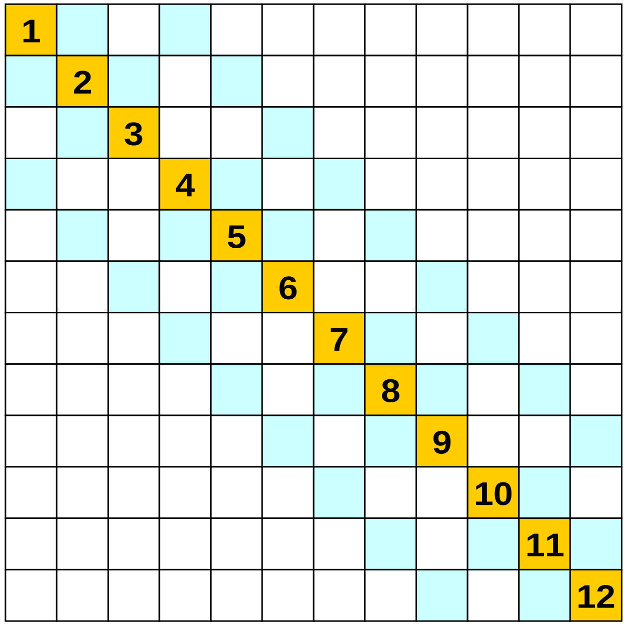
$$\begin{bmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & 0.00 & 0.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 6.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -1.00 & 0.00 & -1.00 & 6.00
\end{bmatrix}
\begin{bmatrix} \\ \\ \\ \\ \\ X \\ \\ \\ \\ \\ \\ \end{bmatrix}
=
\begin{bmatrix}
0.00 \\ 3.00 \\ 10.00 \\ 11.00 \\ 10.00 \\ 19.00 \\ 20.00 \\ 16.00 \\ 28.00 \\ 42.00 \\ 36.00 \\ 52.00
\end{bmatrix}$$

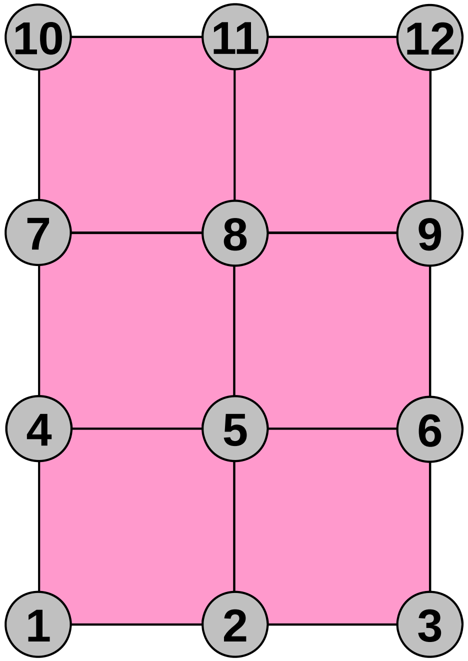




実例：解

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





完全LU分解したマトリクス

lu1.f

もとのマトリクス

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

LU分解したマトリクス

[L][U]同時に表示

[L]対角成分（=1）省略

（fill-inが生じている。もともと0だった成分が非ゼロになっている）

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

不完全LU分解したマトリクス（fill-in無し）

lu2.f

不完全LU分解した
マトリクス（fill-in無し）

[L][U]同時に表示
[L]対角成分（=1）省略

$$\begin{bmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 5.83 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 0.00 & -0.17 & 5.66 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -0.17 & 0.00 & -0.18 & 5.65 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.17 & 5.65 & -1.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.65 & 0.00 & 0.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.17 & 5.65 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.65
\end{bmatrix}$$

完全LU分解した
マトリクス

[L][U]同時に表示
[L]対角成分（=1）省略

（fill-inが生じている。もともと0だった成分が非ゼロになっている）

$$\begin{bmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 5.83 & -1.00 & -0.17 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 5.83 & -0.03 & -0.17 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & -0.03 & 0.00 & 5.83 & -1.03 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & -0.03 & -0.18 & 5.64 & -1.03 & -0.18 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -0.17 & 0.00 & -0.18 & 5.64 & -0.03 & -0.18 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -0.17 & -0.03 & -0.01 & 5.82 & -1.03 & -0.01 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.03 & -0.18 & 5.63 & -1.03 & -0.18 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.63 & -0.03 & -0.18 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.17 & -0.03 & -0.01 & 5.82 & -1.03 & -0.01 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.03 & -0.18 & 5.63 & -1.03 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.63
\end{bmatrix}$$

解の比較：ちょっと違う

不完全LU分解
lu2.f

$$\begin{pmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 5.83 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 0.00 & -0.17 & 5.66 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -0.17 & 0.00 & -0.18 & 5.65 & 0.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.17 & 5.65 & -1.00 & 0.00 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.65 & 0.00 & 0.00 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.17 & 0.00 & 0.00 & 5.83 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.17 & 5.65 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.65
\end{pmatrix}
\rightarrow
\begin{pmatrix}
0.92 \\ 1.75 \\ 2.76 \\ 3.79 \\ 4.46 \\ 5.57 \\ 6.66 \\ 7.25 \\ 8.46 \\ 9.66 \\ 10.54 \\ 11.83
\end{pmatrix}$$

完全LU分解
lu1.f

$$\begin{pmatrix}
6.00 & -1.00 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & 5.83 & -1.00 & -0.17 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & 5.83 & -0.03 & -0.17 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
-0.17 & -0.03 & 0.00 & 5.83 & -1.03 & 0.00 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & -0.17 & -0.03 & -0.18 & 5.64 & -1.03 & -0.18 & -1.00 & 0.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & -0.17 & 0.00 & -0.18 & 5.64 & -0.03 & -0.18 & -1.00 & 0.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & -0.17 & -0.03 & -0.01 & 5.82 & -1.03 & -0.01 & -1.00 & 0.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.03 & -0.18 & 5.63 & -1.03 & -0.18 & -1.00 & 0.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.63 & -0.03 & -0.18 & -1.00 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.17 & -0.03 & -0.01 & 5.82 & -1.03 & -0.01 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & -0.03 & -0.18 & 5.63 & -1.03 \\
0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & 0.00 & -0.18 & 0.00 & -0.18 & 5.63
\end{pmatrix}
\rightarrow
\begin{pmatrix}
1.00 \\ 2.00 \\ 3.00 \\ 4.00 \\ 5.00 \\ 6.00 \\ 7.00 \\ 8.00 \\ 9.00 \\ 10.00 \\ 11.00 \\ 12.00
\end{pmatrix}$$

ILU(0), IC(0) 前処理

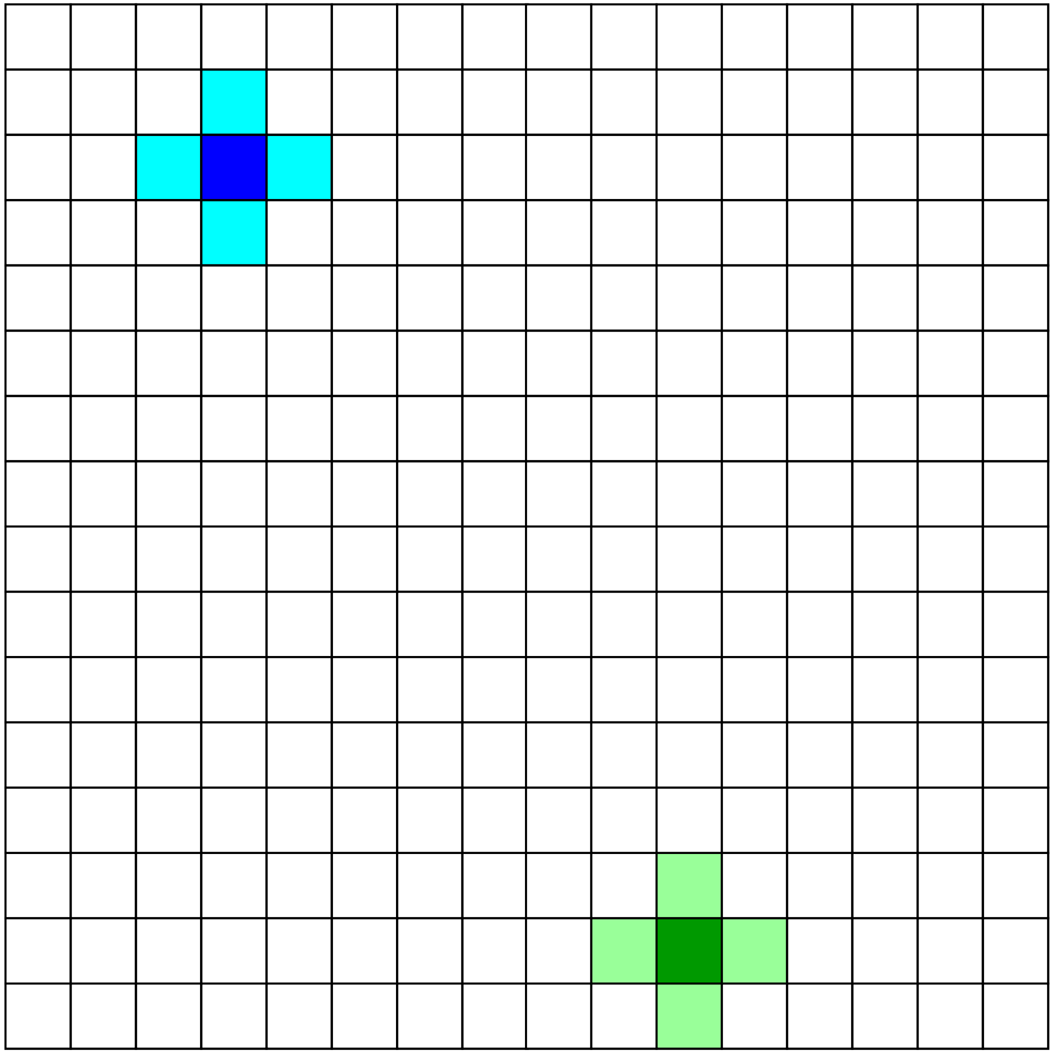
- Fill-inを全く考慮しない「不完全な」分解
  - 記憶容量，計算量削減
- これを解くと「不完全な」解が得られるが，本来の解とそれほどずれているわけではない
  - 問題に依存する

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

Parallel Iterative Solvers

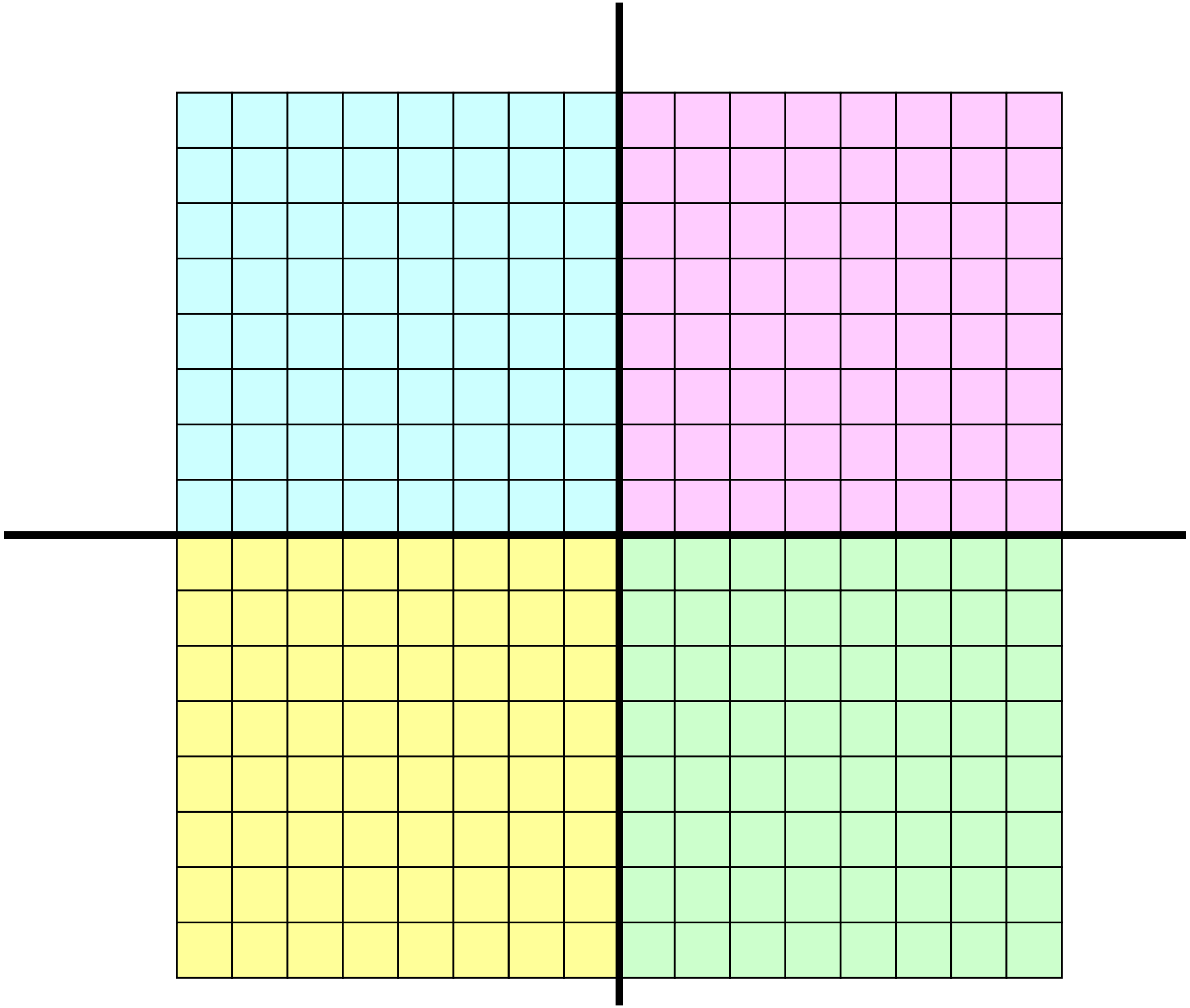
- Both of convergence (robustness) and efficiency (single/parallel) are important
- Global communications needed
  - Mat-Vec (P2P communications, MPI_Isend/Irecv/Waitall): Local Data Structure with HALO
    - ➢ effect of latency
  - Dot-Products (MPI_Allreduce)
  - Preconditioning (up to algorithm)
- Remedy for Robust Parallel ILU Preconditioner
  - Additive Schwartz Domain Decomposition
  - HID (Hierarchical Interface Decomposition, based on global nested dissection) [Henon & Saad 2007], ext. HID [KN 2010]
- Parallel "Direct" Solvers (e.g. SuperLU, MUMPS etc.)

Local Data Structures for Parallel FEM/FDM using Krylov Iterative Solvers Example: 2D FDM Mesh (5-point stencil)



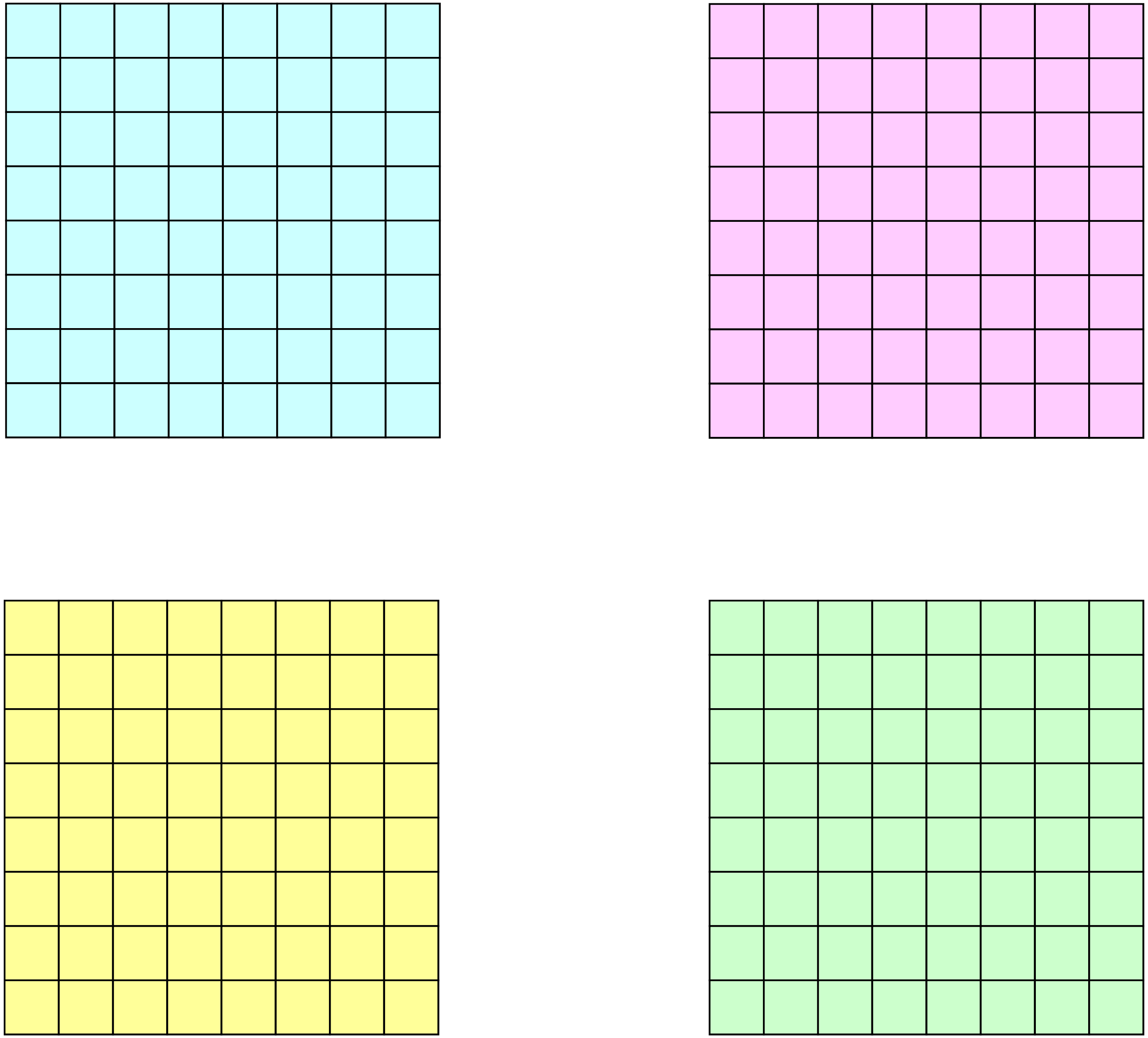
Example: 2D FDM Mesh (5-point stencil)

4-regions/domains



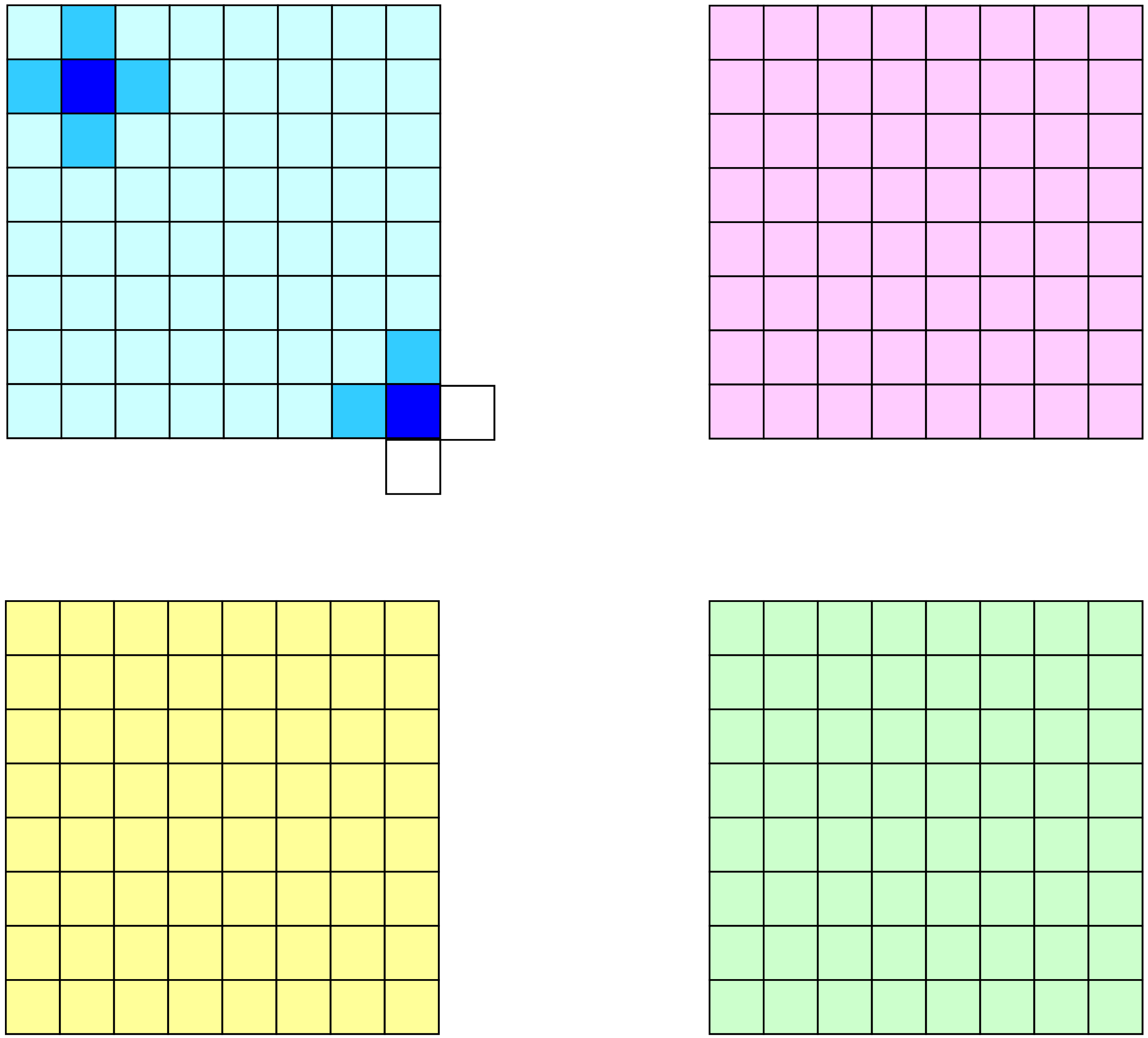
Example: 2D FDM Mesh (5-point stencil)

4-regions/domains



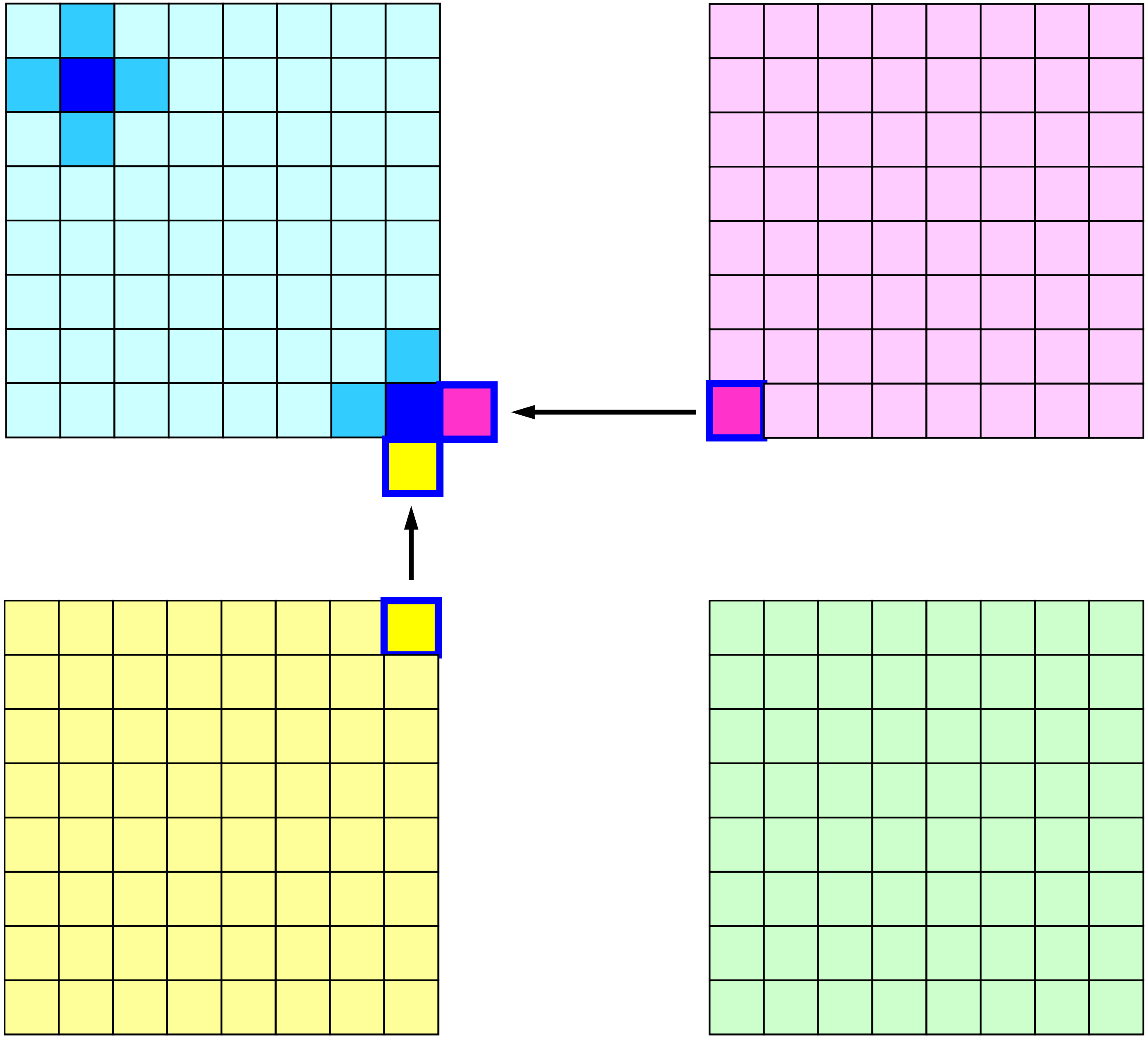
Example: 2D FDM Mesh (5-point stencil)

meshes at domain boundary need info. neighboring domains



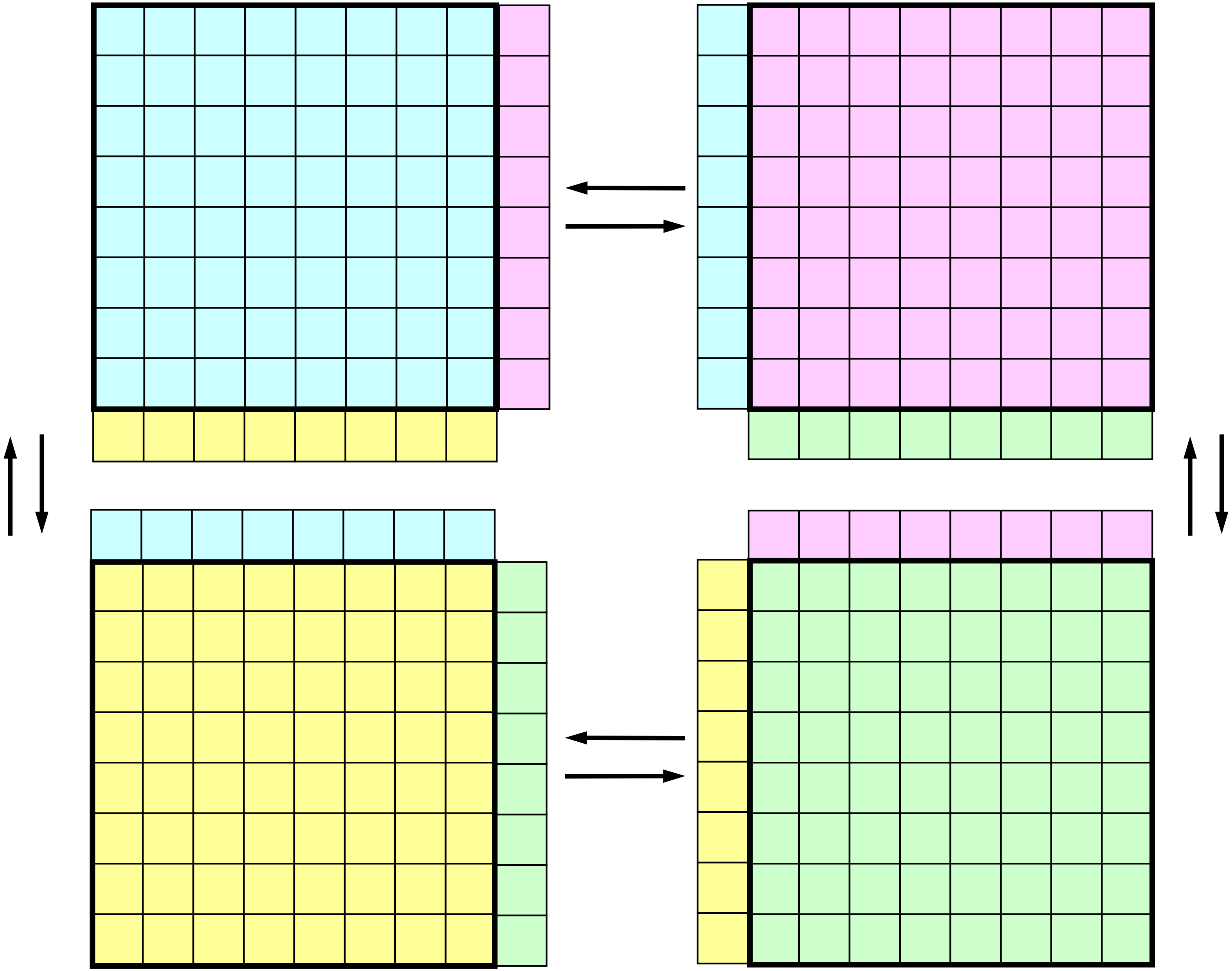
Example: 2D FDM Mesh (5-point stencil)

meshes at domain boundary need info. neighboring domains



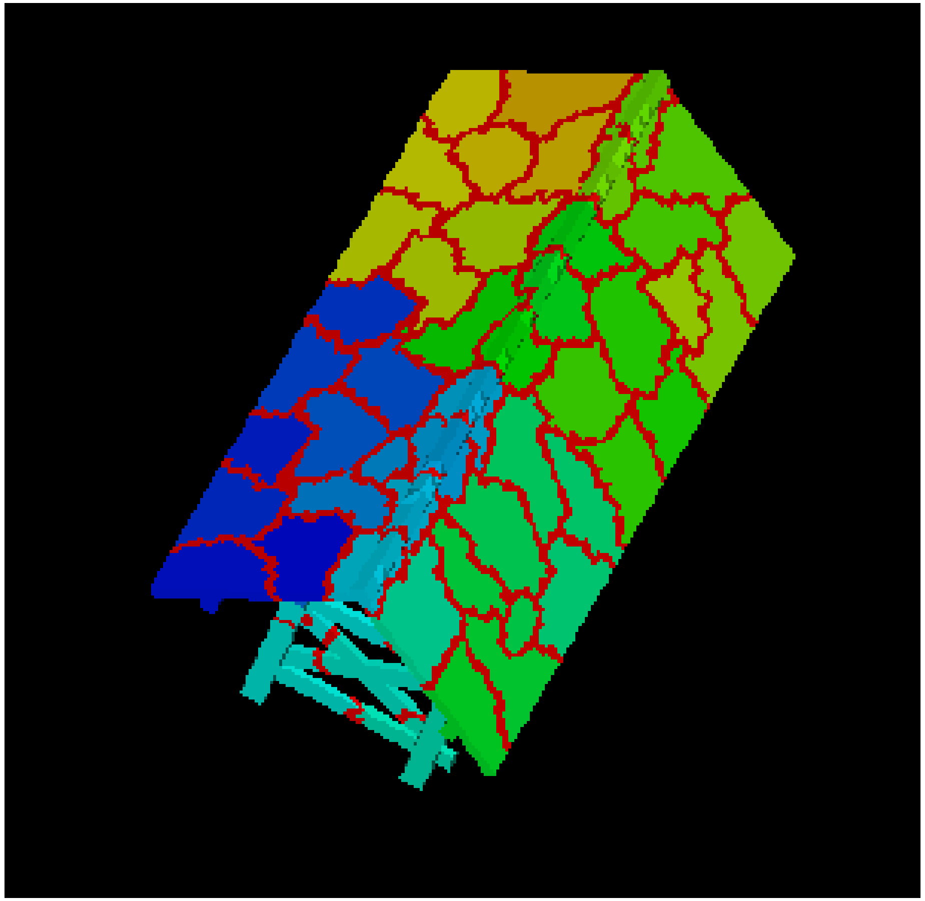
Example: 2D FDM Mesh (5-point stencil)

comm. using "HALO (overlapped meshes)"



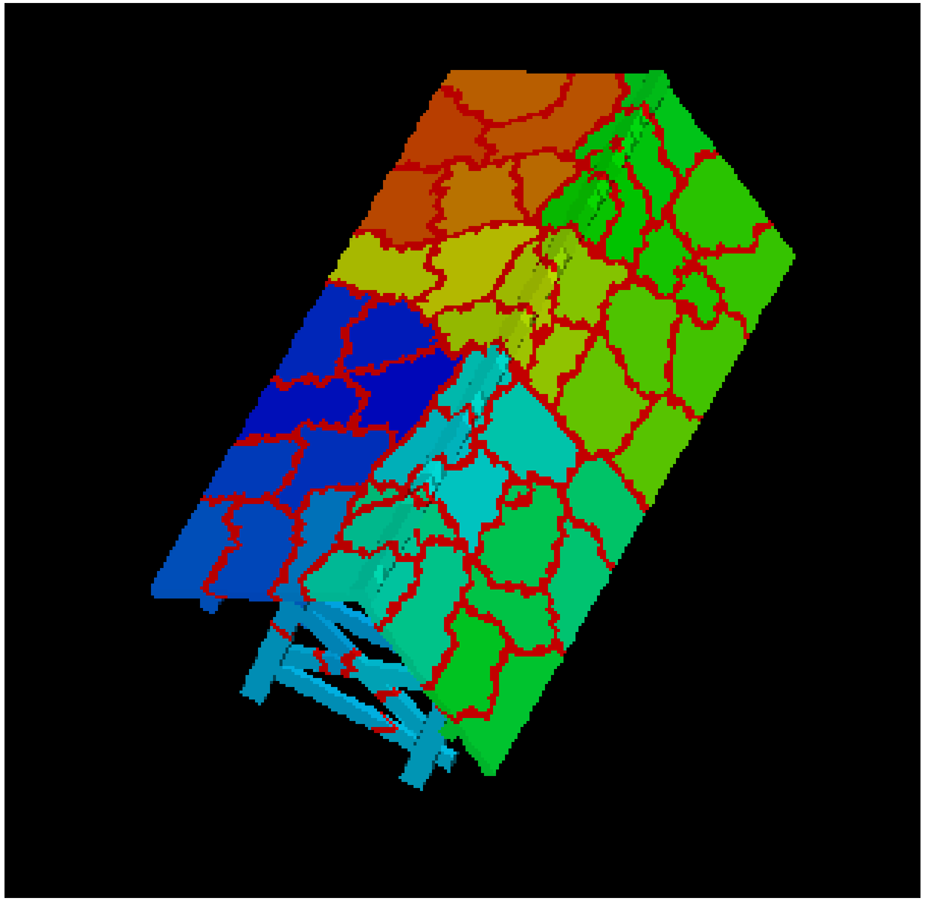
Red Lacquered Gate in 64 PEs

40,624 elements, 54,659 nodes



k-METIS

Load Balance= 1.03
edgecut = 7,563



p-METIS

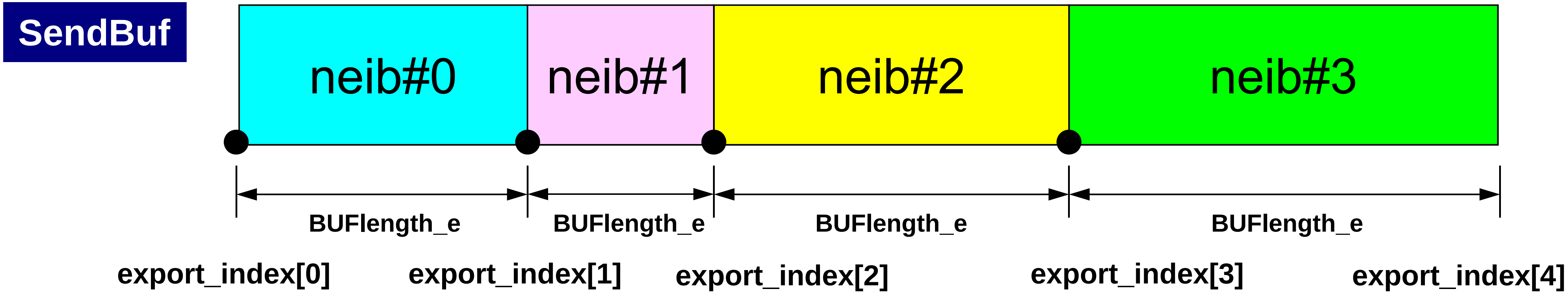
Load Balance= 1.00
edgecut = 7,738

Generalized Comm. Table: Send

- Neighbors
  - NeibPETot, NeibPE[neib]
- Message size for each neighbor
  - export_index[neib], neib= 0, NeibPETot-1
- ID of **boundary** points
  - export_item[k], k= 0, export_index[NeibPETot]-1
- Messages to each neighbor
  - SendBuf[k], k= 0, export_index[NeibPETot]-1

SEND: MPI_Isend/Irecv/Waitall

C



SendBuf

neib#0 neib#1 neib#2 neib#3

BUFlength_e BUFlength_e BUFlength_e BUFlength_e

export_index[0] export_index[1] export_index[2] export_index[3] export_index[4]

export_item (export_index[neib]:export_index[neib+1]-1) are sent to neib-th neighbor

```
for (neib=0; neib<NeibPETot;neib++){
   for (k=export_index[neib];k<export_index[neib+1];k++){
      kk= export_item[k];
      SendBuf[k]= VAL[kk];
   }
}
```

Copied to sending buffers

```
for (neib=0; neib<NeibPETot; neib++){
  tag= 0;
  iS_e= export_index[neib];
  iE_e= export_index[neib+1];
  BUFlength_e= iE_e - iS_e

  ierr= MPI_Isend
         (&SendBuf[iS_e], BUFlength_e, MPI_DOUBLE, NeibPE[neib], 0,
          MPI_COMM_WORLD, &ReqSend[neib])
}

MPI_Waitall(NeibPETot, ReqSend, StatSend);
```

C

MPI_Isend

- Begins a non-blocking send
  - Send the contents of sending buffer (starting from **`sendbuf`**, number of messages: **`count`**) to **`dest`** with **`tag`** .
  - Contents of sending buffer cannot be modified before calling corresponding **`MPI_Waitall`**.

- **`MPI_Isend (sendbuf,count,datatype,dest,tag,comm,request)`**

– **`sendbuf`**	choice	`I`	starting address of sending buffer
– **`count`**	int	`I`	number of elements in sending buffer
– **`datatype`**	MPI_Datatype	`I`	datatype of each sending buffer element
– **`dest`**	int	`I`	rank of destination
– **`tag`**	int	`I`	message tag This integer can be used by the application to distinguish messages. Communication occurs if `tag's` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be "0" (in this class),
– **`comm`**	MPI_Comm	`I`	communicator
– **`request`**	MPI_Request	`O`	communication request array used in `MPI_Waitall`

MPI_Waitall

C

- **MPI_Waitall** blocks until all comm’s, associated with **request** in the array, complete. It is used for synchronizing **MPI_Isend** and **MPI_Irecv** in this class.
- At sending phase, contents of sending buffer cannot be modified before calling corresponding **MPI_Waitall**. At receiving phase, contents of receiving buffer cannot be used before calling corresponding **MPI_Waitall**.
- **MPI_Isend** and **MPI_Irecv** can be synchronized simultaneously with a single **MPI_Waitall** if it is consitent.
  - Same **request** should be used in **MPI_Isend** and **MPI_Irecv**.
- Its operation is similar to that of **MPI_Barrier** but, **MPI_Waitall** can not be replaced by **MPI_Barrier**.
  - Possible troubles using **MPI_Barrier** instead of **MPI_Waitall**: Contents of **request** and **status** are not updated properly, very slow operations etc.

- **MPI_Waitall (count,request,status)**

– **count**	int	I	number of processes to be synchronized
– **request**	MPI_Request	I/O	comm. request used in MPI_Waitall (array size: count)
– **status**	MPI_Status	O	array of status objects MPI_STATUS_SIZE: defined in ‘mpif.h’, ‘mpi.h’

Generalized Comm. Table: Receive

- Neighbors
  - NeibPETot , NeibPE[neib]
- Message size for each neighbor
  - import_index[neib], neib= 0, NeibPETot-1
- ID of **external** points
  - import_item[k], k= 0, import_index[NeibPETot]-1
- Messages from each neighbor
  - RecvBuf[k], k= 0, import_index[NeibPETot]-1

C

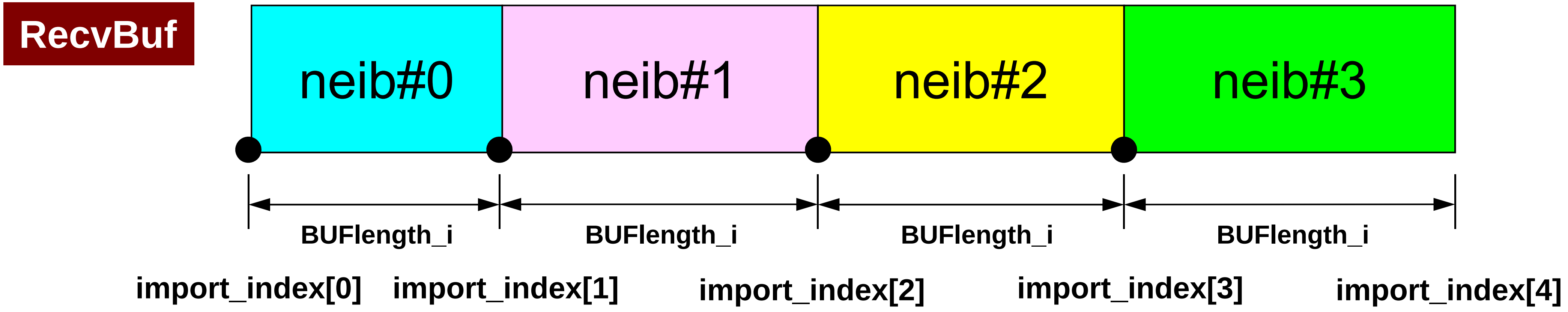
RECV: MPI_Isend/Irecv/Waitall

C

```
for (neib=0; neib<NeibPETot; neib++){
   tag= 0;
   iS_i= import_index[neib];
   iE_i= import_index[neib+1];
   BUFlength_i= iE_i - iS_i

   ierr= MPI_Irecv
          (&RecvBuf[iS_i], BUFlength_i, MPI_DOUBLE, NeibPE[neib], 0,
           MPI_COMM_WORLD, &ReqRecv[neib])
}

MPI_Waitall(NeibPETot, ReqRecv, StatRecv);

for (neib=0; neib<NeibPETot;neib++){
    for (k=import_index[neib];k<import_index[neib+1];k++){
       kk= import_item[k];
       VAL[kk]= RecvBuf[k];
    }
}
```

Copied from receiving buffer

import_item (import_index[neib]:import_index[neib+1]-1) are received from neib-th neighbor



C

MPI_Irecv

- Begins a non-blocking receive
  - Receiving the contents of receiving buffer (starting from **`recvbuf`**, number of messages: **`count`**) from **`source`** with **`tag`** .
  - Contents of receiving buffer cannot be used before calling corresponding **`MPI_Waitall`**.

- **`MPI_Irecv (recvbuf,count,datatype,source,tag,comm,request)`**

– **`recvbuf`**	choice	`I`	starting address of receiving buffer
– **`count`**	int	`I`	number of elements in receiving buffer
– **`datatype`**	MPI_Datatype	`I`	datatype of each receiving buffer element
– **`source`**	int	`I`	rank of source
– **`tag`**	int	`I`	message tag This integer can be used by the application to distinguish messages. Communication occurs if `tag’s` of `MPI_Isend` and `MPI_Irecv` are matched. Usually tag is set to be “0” (in this class),
– **`comm`**	MPI_Comm	`I`	communicator
– **`request`**	MPI_Request	`O`	communication request array used in `MPI_Waitall`

References: Libraries (mainly for flat MPI)

- Talk by the Next Speaker (Tony Drummond)

- Trillinos
  - http://trilinos.sandia.gov/
- PETSc
  - http://www.mcs.anl.gov/petsc/
- GeoFEM
  - http://geofem.tokyo.rist.or.jp/
- ppOpen-HPC
  - http://ppopenhpc.cc.u-tokyo.ac.jp/

Preconditioning for Iterative Solvers

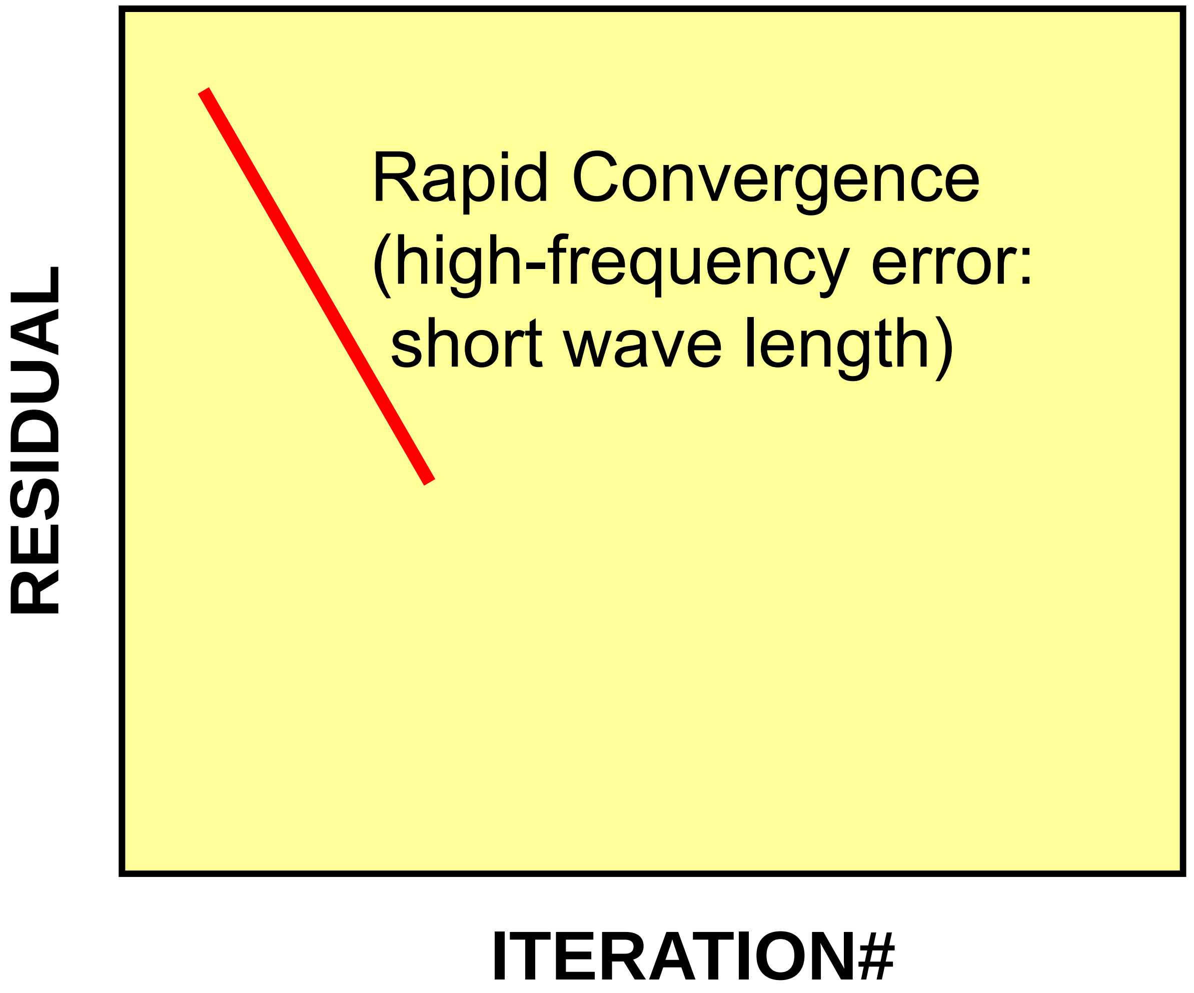
- A critical issue for both of robustness and efficiency
- Libraries (e.g. PETSc, Trillinos, ppOpen-HPC) cover only general ones (e.g. ILU(p))
- Selection of preconditioner strongly depends on:
  - numerical property of matrix
  - features of physics, PDE, boundary conditions, mat. property, size of FEM mesh etc.
    - sometimes, problem specific preconditioning needed
- “Parallel” preconditioning is really an exciting research area, important for practical computing.
- All of computational scientists, computer scientists, and mathematicians must work hard for that under intensive collaboration

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG
- ppOpen-HPC

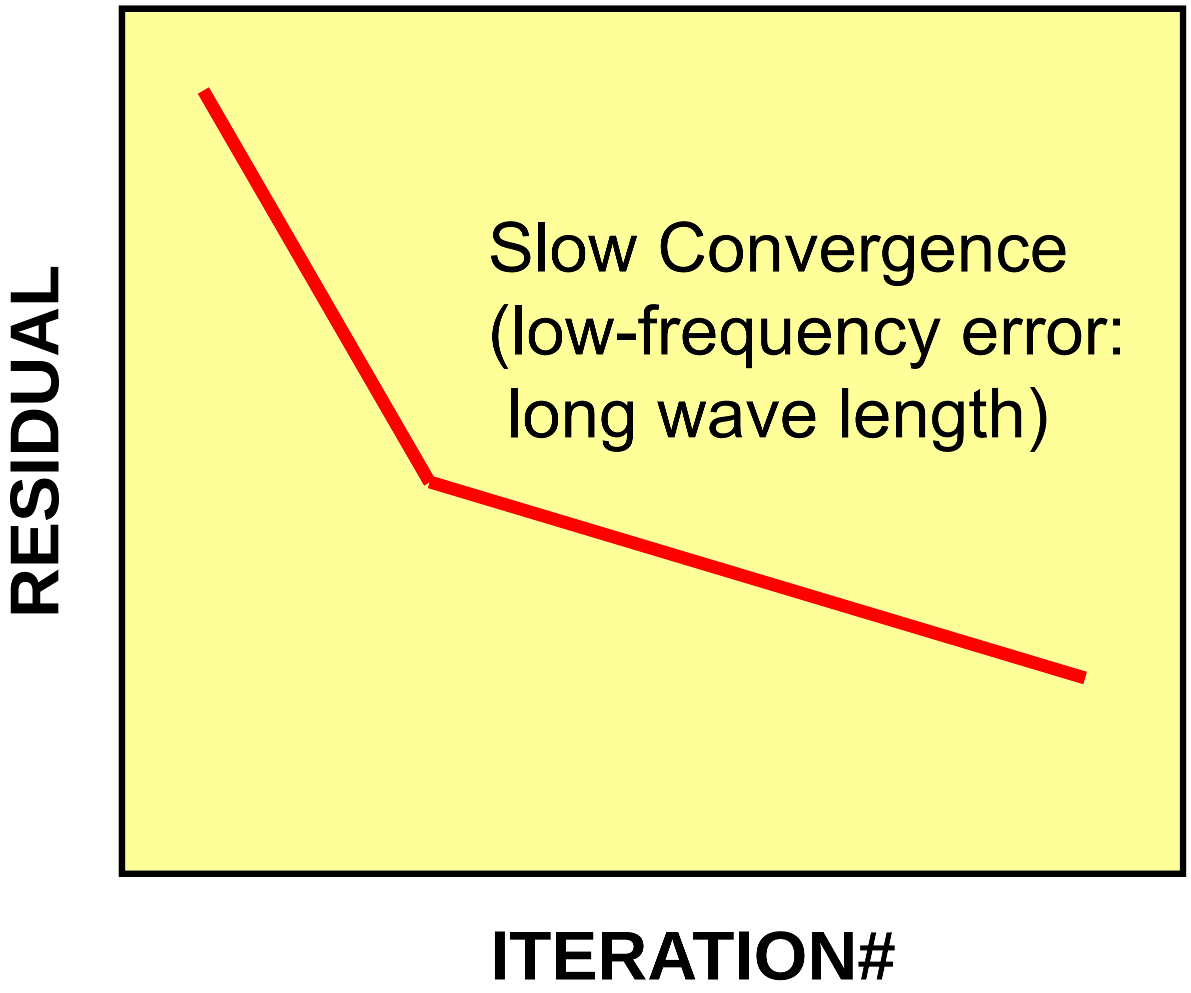
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.

Convergence of Gauss-Seidel & SOR



Convergence of Gauss-Seidel & SOR

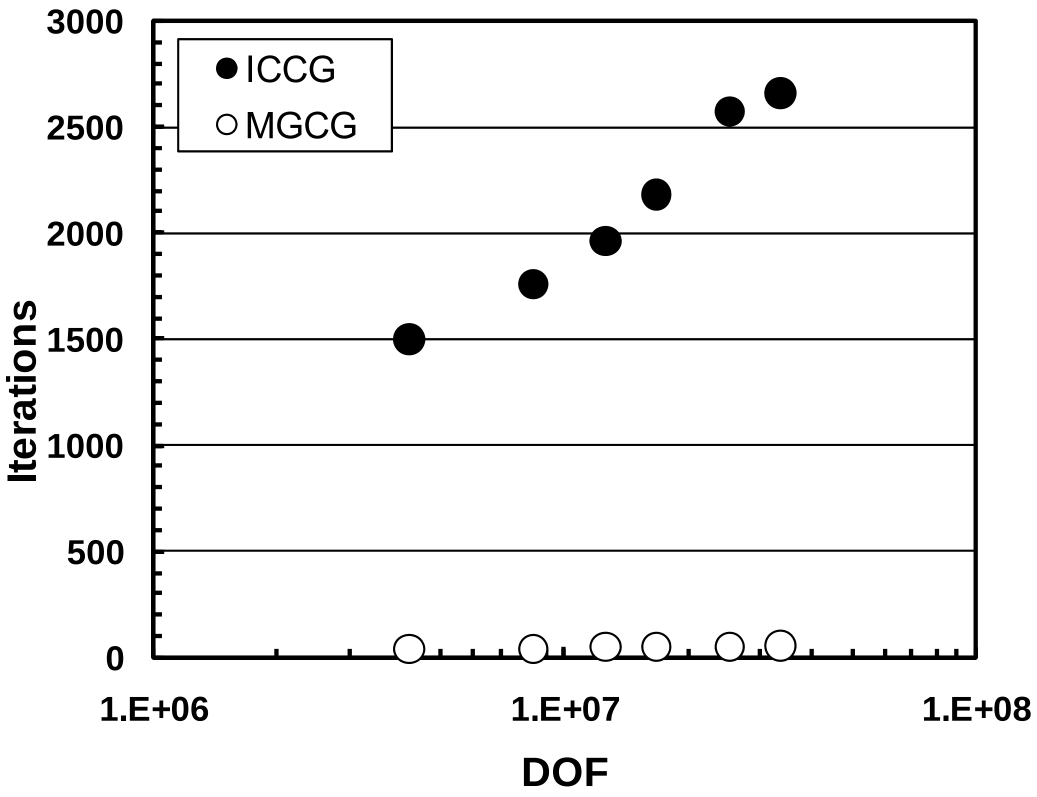


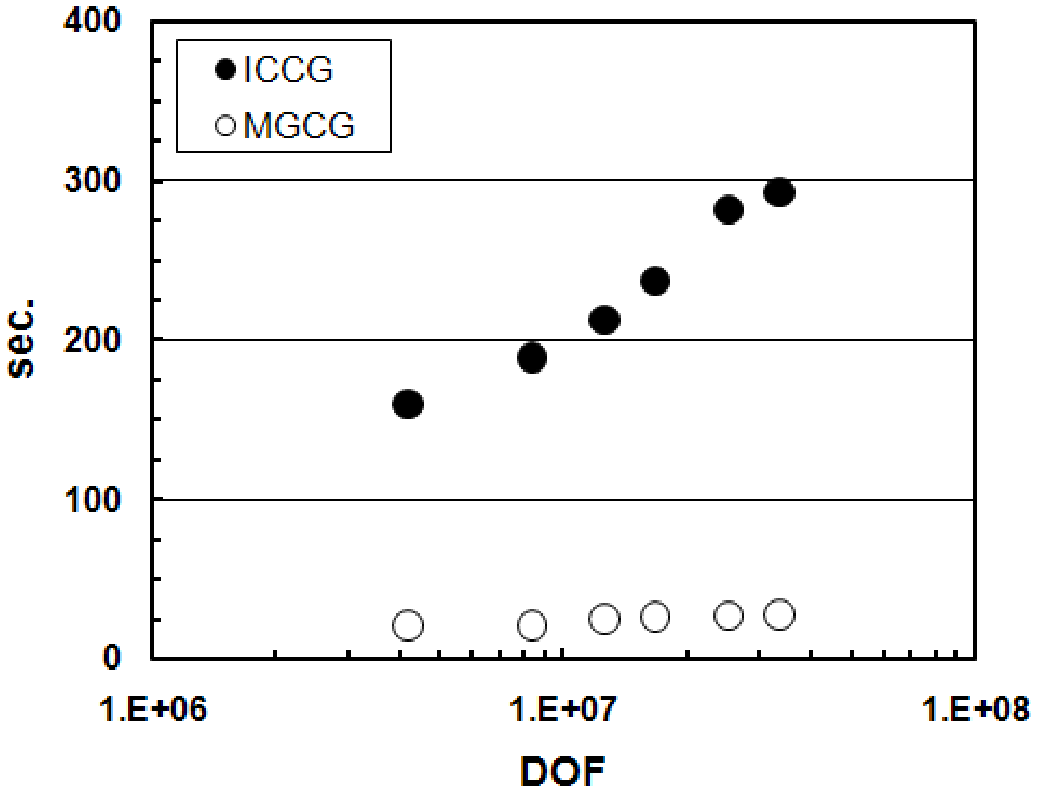
Around the multigrid in a single slide

- Multigrid is a scalable method for solving linear equations.
- Relaxation methods (smoother/smoothing operator in MG world) such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error.
- The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid.
- Multigrid method uniformly damps all frequencies of error components with a computational cost that depends only linearly on the problem size (=scalable).
  - Good for large-scale computations
- Multigrid is also a good preconditioning algorithm for Krylov iterative solvers.

Multigrid is scalable
Weak Scaling: Problem Size/Core Fixed
for 3D Poisson Eqn’s ($\Delta\phi=q$)

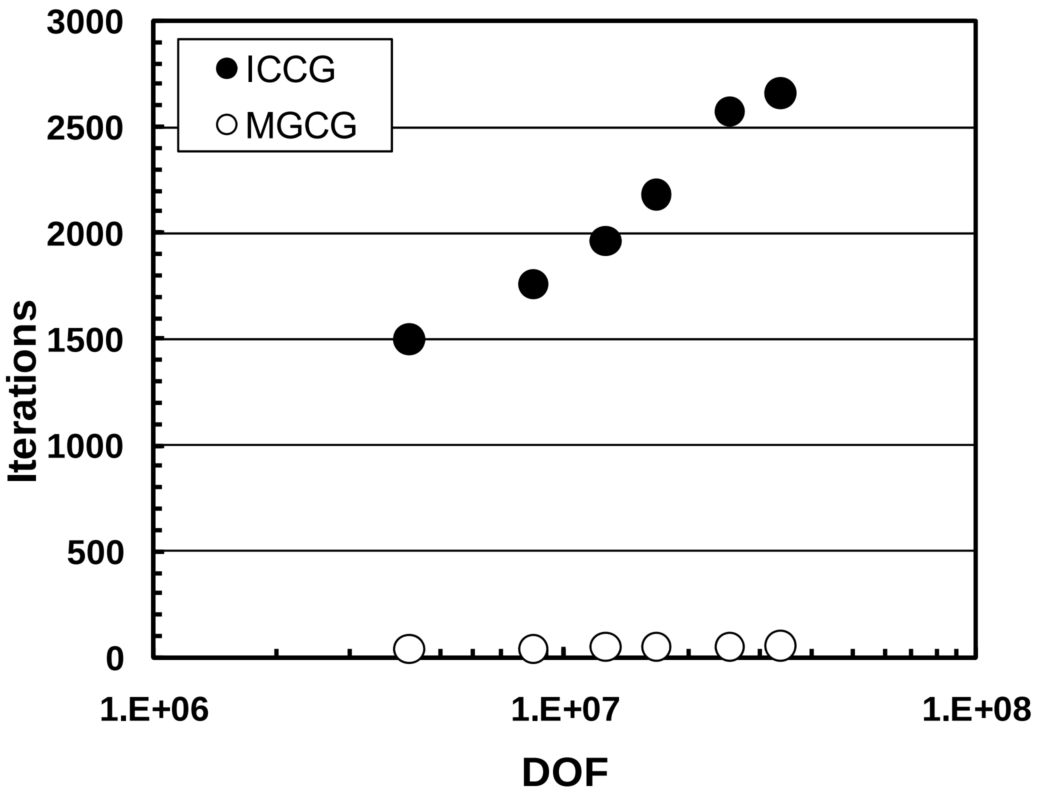
MGCG= Conjugate Gradient with Multigrid Preconditioning

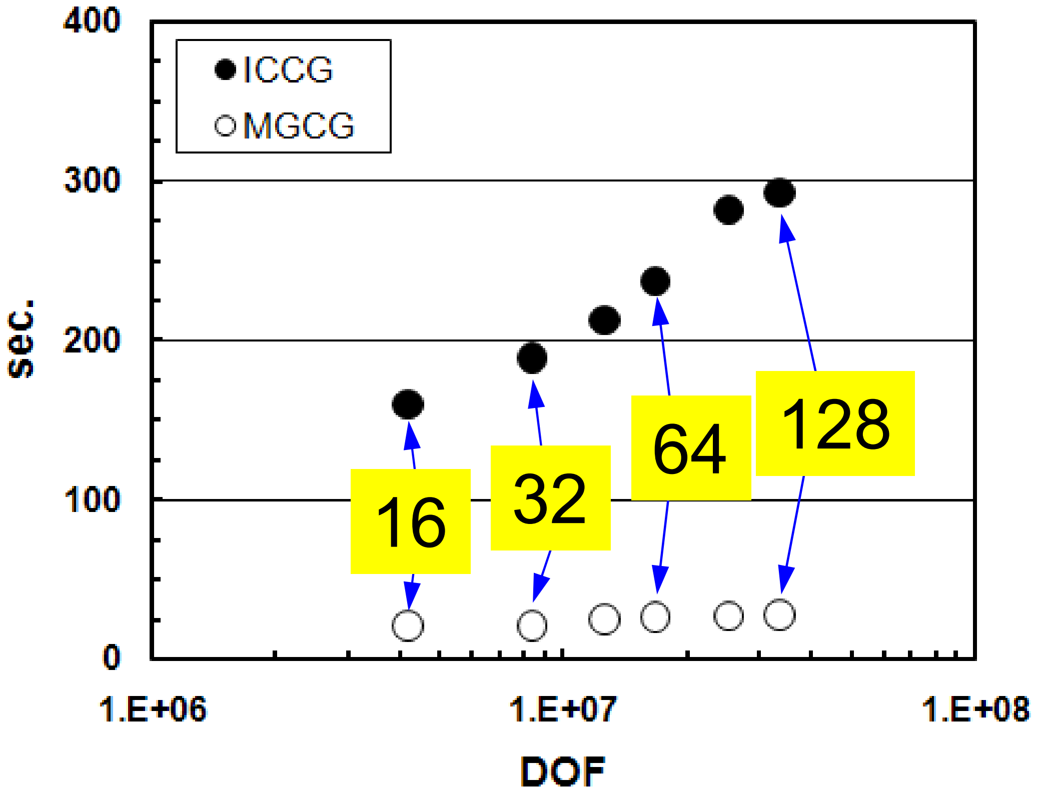




Multigrid is scalable
Weak Scaling: Problem Size/Core Fixed

Comp. time of MGCG for weak scaling is constant: => scalable





Procedure of Multigrid (1/3)

Multigrid is a scalable method for solving linear equations. Relaxation methods such as Gauss-Seidel efficiently damp high-frequency error but do not eliminate low-frequency error. The multigrid approach was developed in recognition that this low-frequency error can be accurately and efficiently solved on a coarser grid. This concept is explained here in the following simple 2-level method. If we have obtained the following linear system on a fine grid :

$$A_F \, u_F = f$$

and A_C as the discrete form of the operator on the coarse grid, a simple coarse grid correction can be given by :

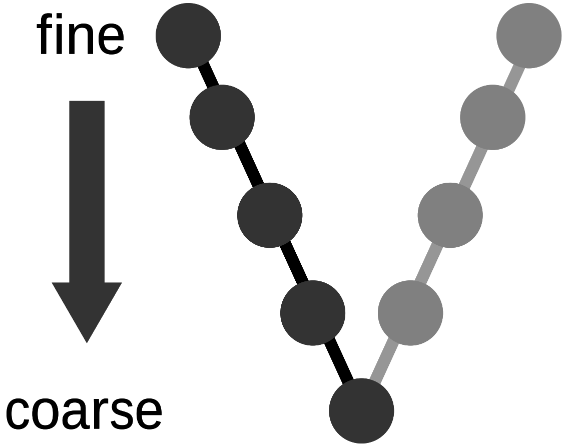
$$u_F^{(i+1)} = u_F^{(i)} + R^T A_C^{-1} R \, (f - A_F \, u_F^{(i)})$$

where R^T is the matrix representation of linear interpolation from the coarse grid to the fine grid (*prolongation* operator) and *R* is called the restriction operator. Thus, it is possible to calculate the residual on the fine grid, solve the coarse grid problem, and interpolate the coarse grid solution on the fine grid.

Procedure of Multigrid (2/3)

This process can be described as follows :

1. Relax the equations on the fine grid and obtain the result $u_F^{(i)}$ = S_F (A_F, f). This operator S_F (e.g., Gauss-Seidel) is called the *smoothing operator (or)*.
2. Calculate the residual term on the fine grid by $r_F = f - A_F\, u_F^{(i)}$.
3. *Restric*t the residual term on to the coarse grid by $r_C = R\, r_F$.
4. Solve the equation $A_C\, u_C = r_C$ on the coarse grid ; the accuracy of the solution on the coarse grid affects the convergence of the entire multigrid system.
5. Interpolate (or *prolong*) the coarse grid correction on the fine grid by D$u_C^{(i)}$ = $R^T\, u_C$.
6. Update the solution on the fine grid by $u_F^{(i+1)}$ = $u_F^{(i)}$ + D$u_C^{(i)}$



$L^k\ W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k\ w_1{}^k$

$v^k = W^k - w_1{}^k,\ L^k\ v^k = R^k$

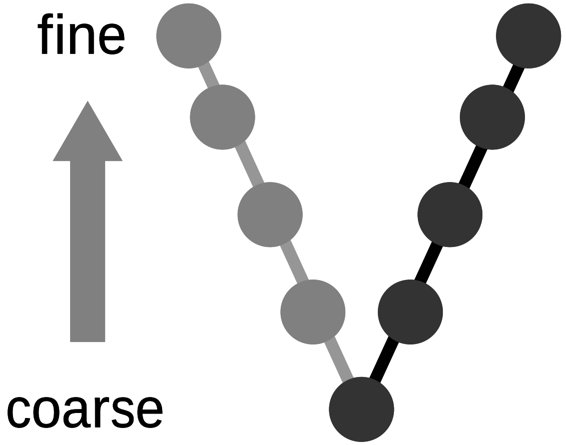
$R^{k-1} = I_k{}^{k-1}\ R^k$

$L^{k-1}\ v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

$v^k = I_{k-1}{}^k\ v^{k-1}$

$w_2{}^k = w_1{}^k + v^k$

$w_1{}^k$: Approx. Solution
v^k : Correction
$I_k{}^{k-1}$: Restriction Operator



$L^k\ W^k = F^k$ **(Linear Equation: Fine Level)**

$R^k = F^k - L^k\ w_1{}^k$

$v^k = W^k - w_1{}^k,\ L^k\ v^k = R^k$

$R^{k-1} = I_k{}^{k-1}\ R^k$

$L^{k-1}\ v^{k-1} = R^{k-1}$ **(Linear Equation: Coarse Level)**

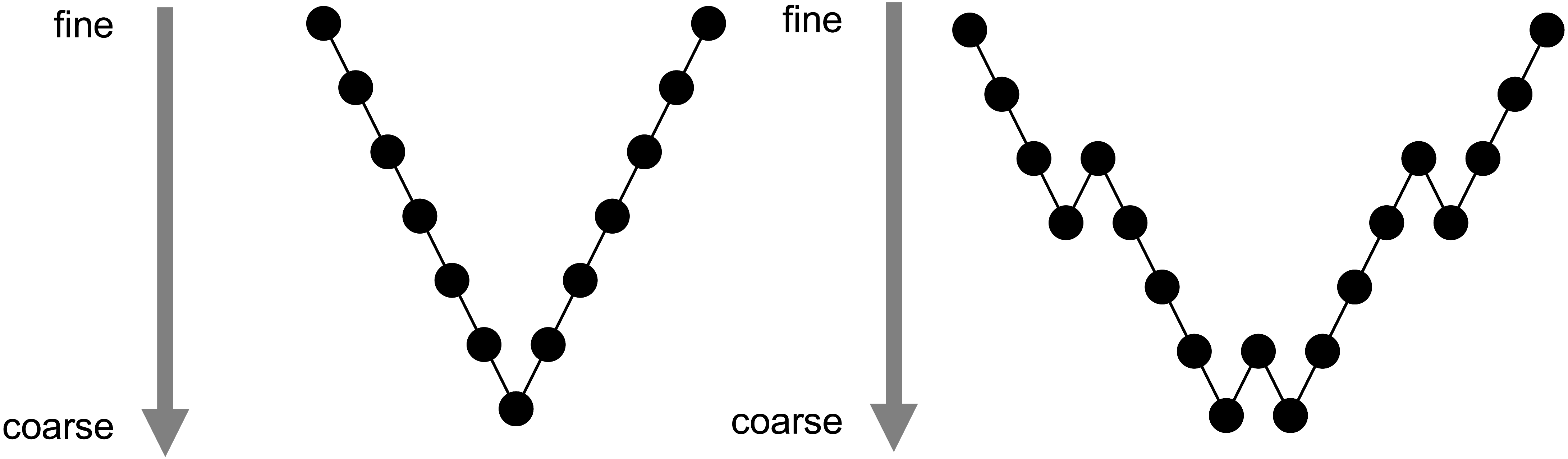
$v^k = I_{k-1}{}^k\ v^{k-1}$

$w_2{}^k = w_1{}^k + v^k$

$I_{k-1}{}^k$: Prolongation Operator
$w_2{}^k$: Approx. Solution by Multigrid

Procedure of Multigrid (3/3)

- Recursive application of this algorithm for 2-level procedure to consecutive systems of coarse-grid equations gives a multigrid V-cycle. If the components of the V-cycle are defined appropriately, the result is a method that uniformly damps all frequencies of error with a computational cost that depends only linearly on the problem size.
  - In other words, multigrid algorithms are *scalable*.
- In the V-cycle, starting with the finest grid, all subsequent coarser grids are visited only once.
  - In the down-cycle, smoothers damp oscillatory error components at different grid scales.
  - In the up-cycle, the smooth error components remaining on each grid level are corrected using the error approximations on the coarser grids.
- Alternatively, in a W-cycle, the coarser grids are solved more rigorously in order to reduce residuals as much as possible before going back to the more expensive finer grids.



(a) V-Cycle

(b) W-Cycle

Multigrid as a Preconditioner

- Multigrid algorithms tend to be problem-specific solutions and less robust than preconditioned Krylov iterative methods such as the IC/ILU methods.
- Fortunately, it is easy to combine the best features of multigrid and Krylov iterative methods into one algorithm
  - multigrid-preconditioned Krylov iterative methods.
- The resulting algorithm is robust, efficient and scalable.

- Mutigrid solvers and Krylov iterative solvers preconditioned by multigrid are intrinsically suitable for parallel computing.

Geometric and Algebraic Multigrid

- One of the most important issues in multigrid is the construction of the coarse grids.
- There are 2 basic multigrid approaches
  - geometric and algebraic
- In geometric multigrid, the geometry of the problem is used to define the various multigrid components.
- In contrast, algebraic multigrid methods use only the information available in the linear system of equations, such as matrix connectivity.
- Algebraic multigrid method (AMG) is suitable for applications with unstructured grids.
- Many tools for both geometric and algebraic methods on unstructured grids have been developed.

“Dark Side” of Multigrid Method

- Its performance is excellent for well-conditioned simple problems, such as homogeneous Poisson equations.
- But convergence could be worse for ill-conditioned problems.
- Extension of applicability of multigrid method is an active research area.

References

- 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

- https://computation.llnl.gov/casc/
- Hypre (AMG Library)
  - https://computation.llnl.gov/casc/linear_solvers/sls_hypre.html

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

Key-Issues for Appl's/Algorithms towards Post-Peta & Exa Computing

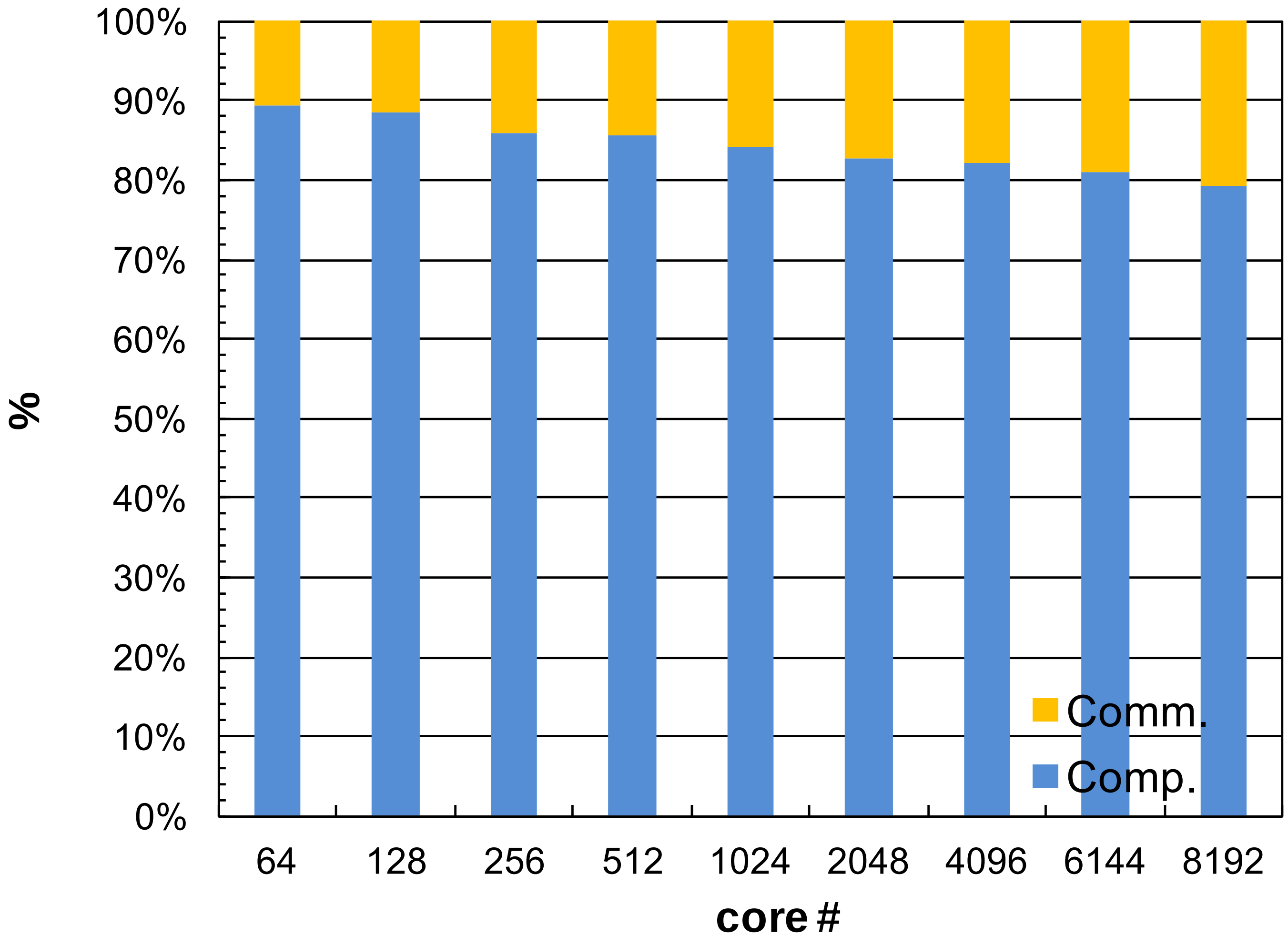
Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

- Hybrid/Heterogeneous Architecture
  - Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    - Data Movement, Hierarchy of Memory
- Communication/Synchronization Reducing Algorithms
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Resilient Algorithms
- Reproducibility of Results

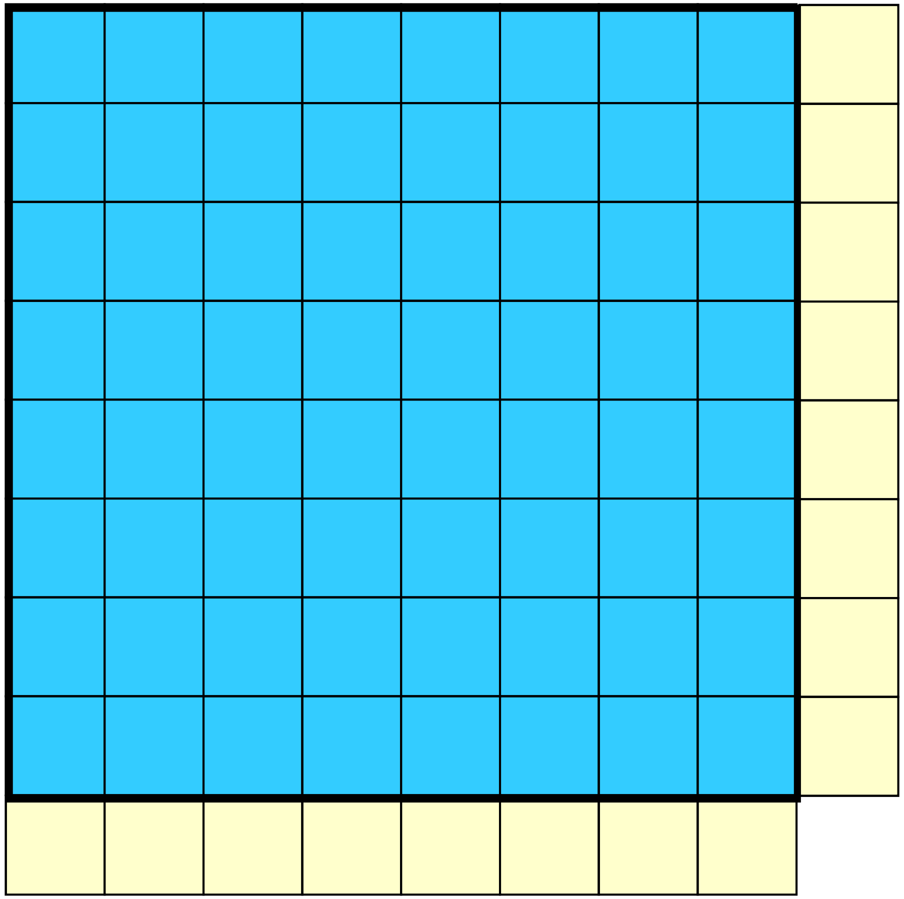
Recent Technical Issues in Parallel Iterative Solvers

- Communication overhead becomes significant
- Communication-Computation Overlap
  - Not so effective for Mat-Vec operations
- Communication Avoiding/Reducing Algorithms
- OpenMP/MPI Hybrid Parallel Programming Model
  - (Next section)

Communication overhead becomes larger as node/core number increases Weak Scaling: MGCG on T2K Tokyo



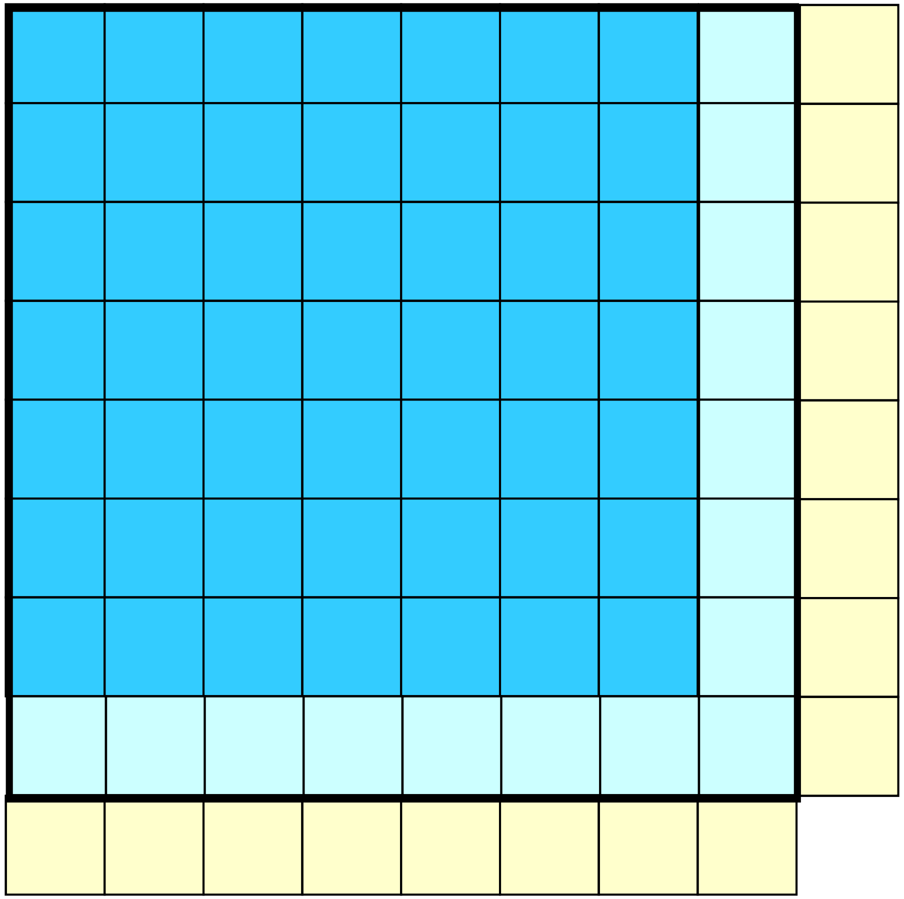
Comm.-Comp. Overlapping



🟦 Internal Meshes

🟨 External (HALO) Meshes

Comm.-Comp. Overlapping



🟦 Internal Meshes

🟨 External (HALO) Meshes

⬜ Internal Meshes on Boundary’s

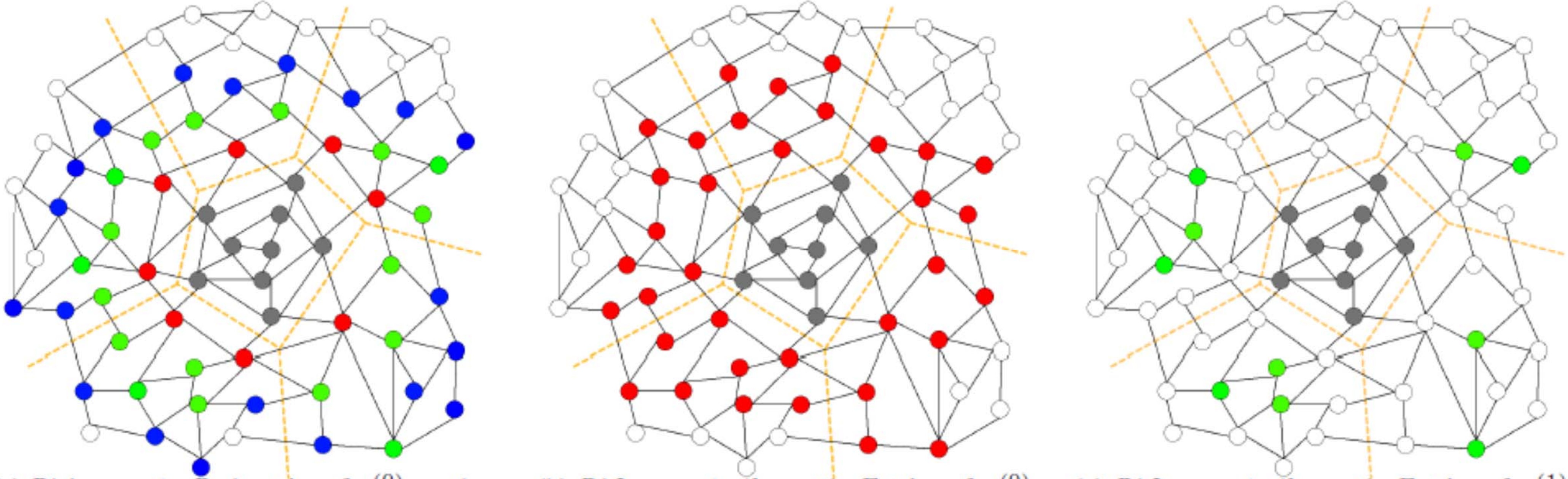
Mat-Vec operations

- Overlapping of computations of internal meshes, and importing external meshes.
- Then computation of international meshes on boundary’s
- Difficult for IC/ILU on Hybrid

Communication Avoiding/Reducing Algorithms for Sparse Linear Solvers

- Krylov Iterative Method without Preconditioning
  - Demmel, Hoemmen, Mohiyuddin etc. (UC Berkeley)
- *s-step* method
  - Just one P2P communication for each Mat-Vec during *s* iterations. Convergence becomes unstable for large *s*.
  - matrix powers kernel: Ax, A^2x, A^3x ...
    - additional computations needed
- Communication Avoiding ILU0 (CA-ILU0) [Moufawad & Grigori, 2013]
  - First attempt to CA preconditioning
  - Nested dissection reordering for limited geometries (2D FDM)

Comm. Avoiding Krylov Iterative Methods using "Matrix Powers Kernel"



(a) PA1 example: Red entries of $x^{(0)}$ are the ones needed when $k = 1$, green are the additional ones needed when $k = 2$ and blue are the additional ones needed when $k = 3$.

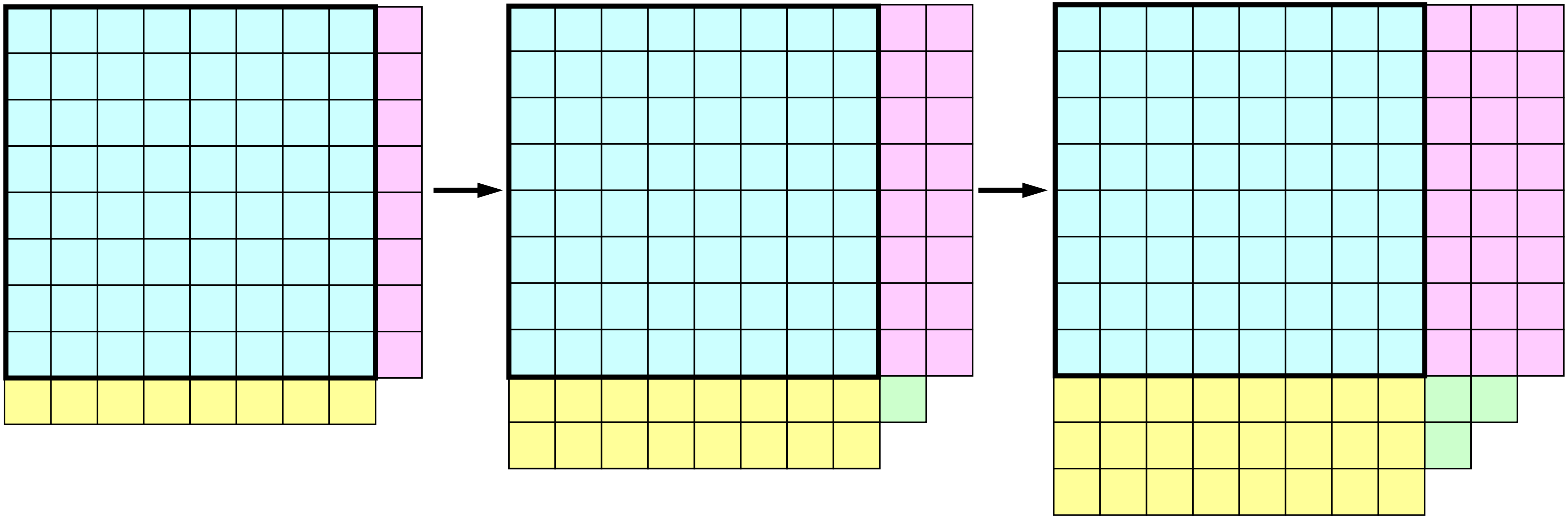
(b) PA2 example ($k = 3$): Entries of $x^{(0)}$ which need to be fetched are colored red.

(c) PA2 example ($k = 3$): Entries of $x^{(1)}$ which need to be fetched are colored green.

Figure 2. Example for PA1 and PA2. The dotted lines define the different blocks. Each block resides on a different processor. The example shows from the perspective of the processor holding the central block.

Avoiding Communication in Sparse Matrix Computations. James Demmel, Mark Hoemmen, Marghoob Mohiyuddin, and Katherine Yelick. , 2008 IPDPS

Required Information of Local Meshes for s-step CA computations (2D 5pt.)



s=1
(original)

s=2

s=3

- Sparse Matrices
- Iterative Linear Solvers
  - Preconditioning
  - Parallel Iterative Linear Solvers
  - Multigrid Method
  - Recent Technical Issues
- Example of Parallel MGCG

Key-Issues for Appl’s/Algorithms towards Post-Peta & Exa Computing

Jack Dongarra (ORNL/U. Tennessee) at ISC 2013

- Hybrid/Heterogeneous Architecture
  - Multicore + GPU/Manycores (Intel MIC/Xeon Phi)
    - Data Movement, Hierarchy of Memory
- Communication/Synchronization Reducing Algorithms
- Mixed Precision Computation
- Auto-Tuning/Self-Adapting
- Fault Resilient Algorithms
- Reproducibility of Results

Background

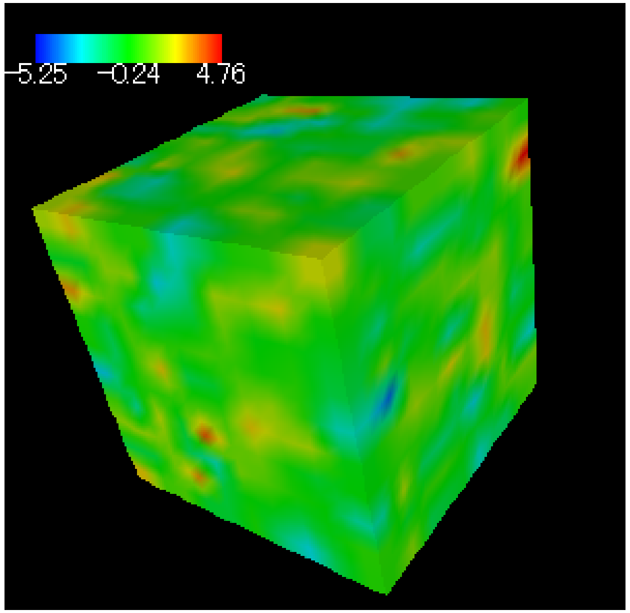
- Large-scale 3D Groundwater Flow
  - Poisson equations
  - Heterogeneous porous media
- Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)
- Flat MPI vs. Hybrid (OpenMP+MPI)
- Expectations for Hybrid Parallel Programming Model
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson’s equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda\left(x, y, z\right)\nabla \phi\right) = q,\ \phi = 0\ at\ z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh
- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3



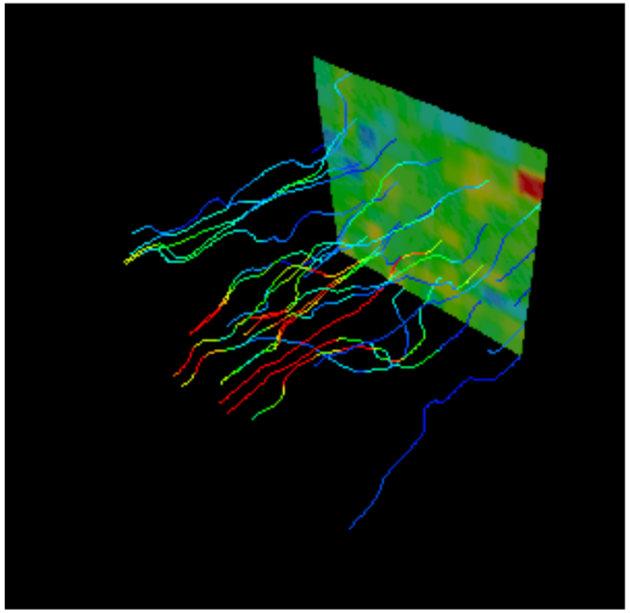
Target Application: *pGW3D-FVM*

- 3D Groundwater Flow via. Heterogeneous Porous Media
  - Poisson’s equation
  - Randomly distributed water conductivity

$$\nabla \cdot \left(\lambda(x, y, z)\nabla \phi\right) = q, \phi = 0\ at\ z = z_{\max}$$

  - Distribution of water conductivity is defined through methods in geostatistics〔Deutsch & Journel, 1998〕
- Finite-Volume Method on Cubic Voxel Mesh

- Distribution of Water Conductivity
  - 10^{-5}-10^{+5}, Condition Number ~ 10^{+10}
  - Average: 1.0
- Cyclic Distribution: 128^3



Background

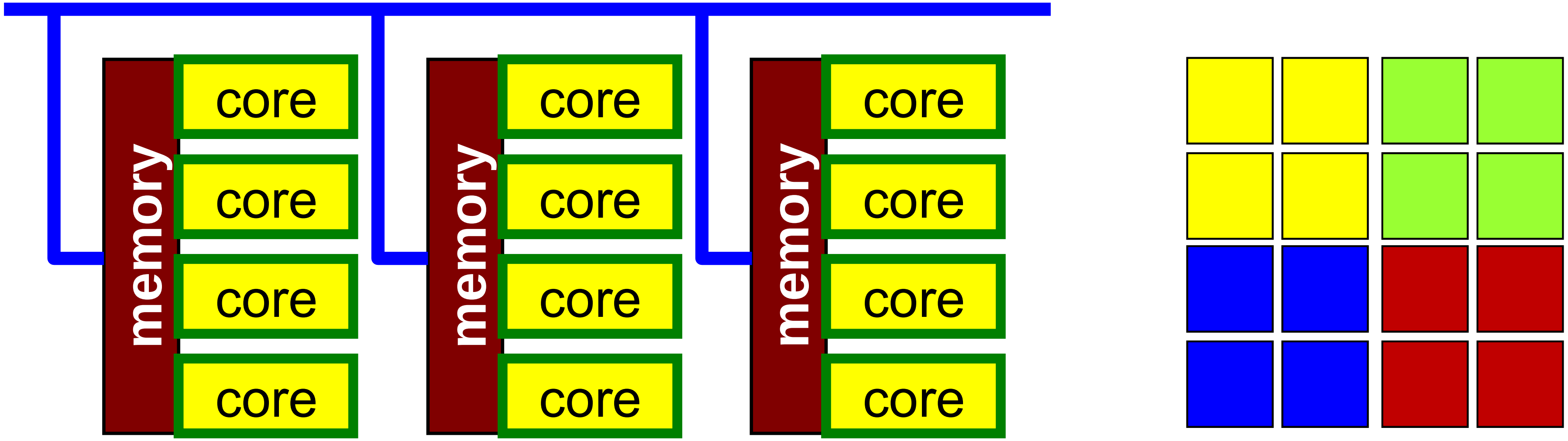
- Large-scale 3D Groundwater Flow
  - Poisson equations
  - Heterogeneous porous media
- Parallel (Geometric) Multigrid Solvers for FVM-type appl. on Fujitsu PRIMEHPC FX10 at University of Tokyo (Oakleaf-FX)
- Flat MPI vs. Hybrid (OpenMP+MPI)
- Expectations for Hybrid Parallel Programming Model
  - Number of MPI processes (and sub-domains) to be reduced
  - O(10^8-10^9)-way MPI might not scale in Exascale Systems
  - Easily extended to Heterogeneous Architectures
    - CPU+GPU, CPU+Manycores (e.g. Intel MIC/Xeon Phi)
    - MPI+X: OpenMP, OpenACC, CUDA, OpenCL

Keywords

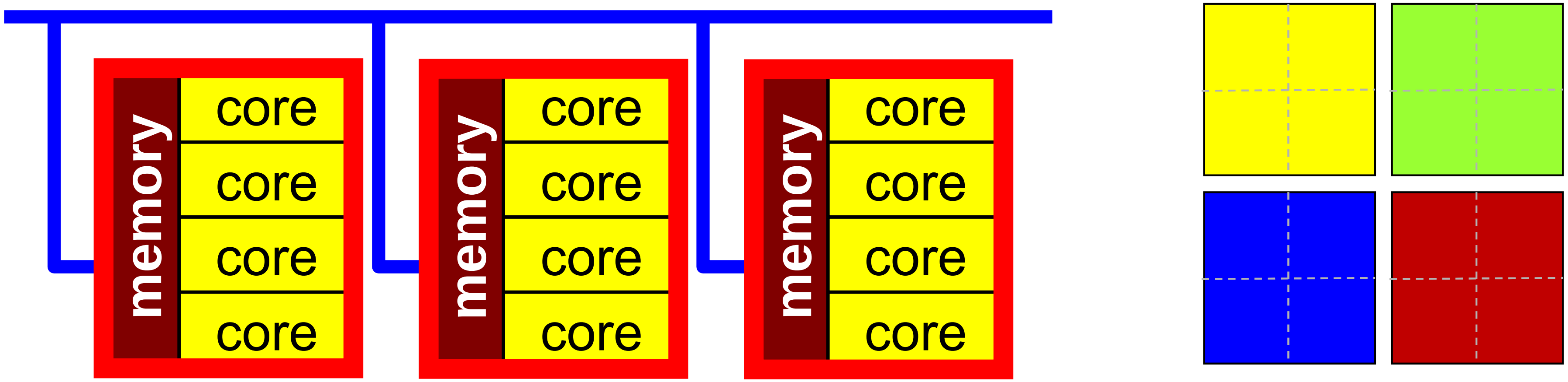
- Parallel Geometric Multigrid
- OpenMP/MPI Hybrid Parallel Programming Model
- Localized Block Jacobi Preconditioning
  - Overlapped Additive Schwartz Domain Decomposition (ASDD)
- OpenMP Parallelization with Coloring
- Coarse Grid Aggregation (CGA), Hierarchical CGA

Flat MPI vs. Hybrid

Flat-MPI：Each Core -> Independent

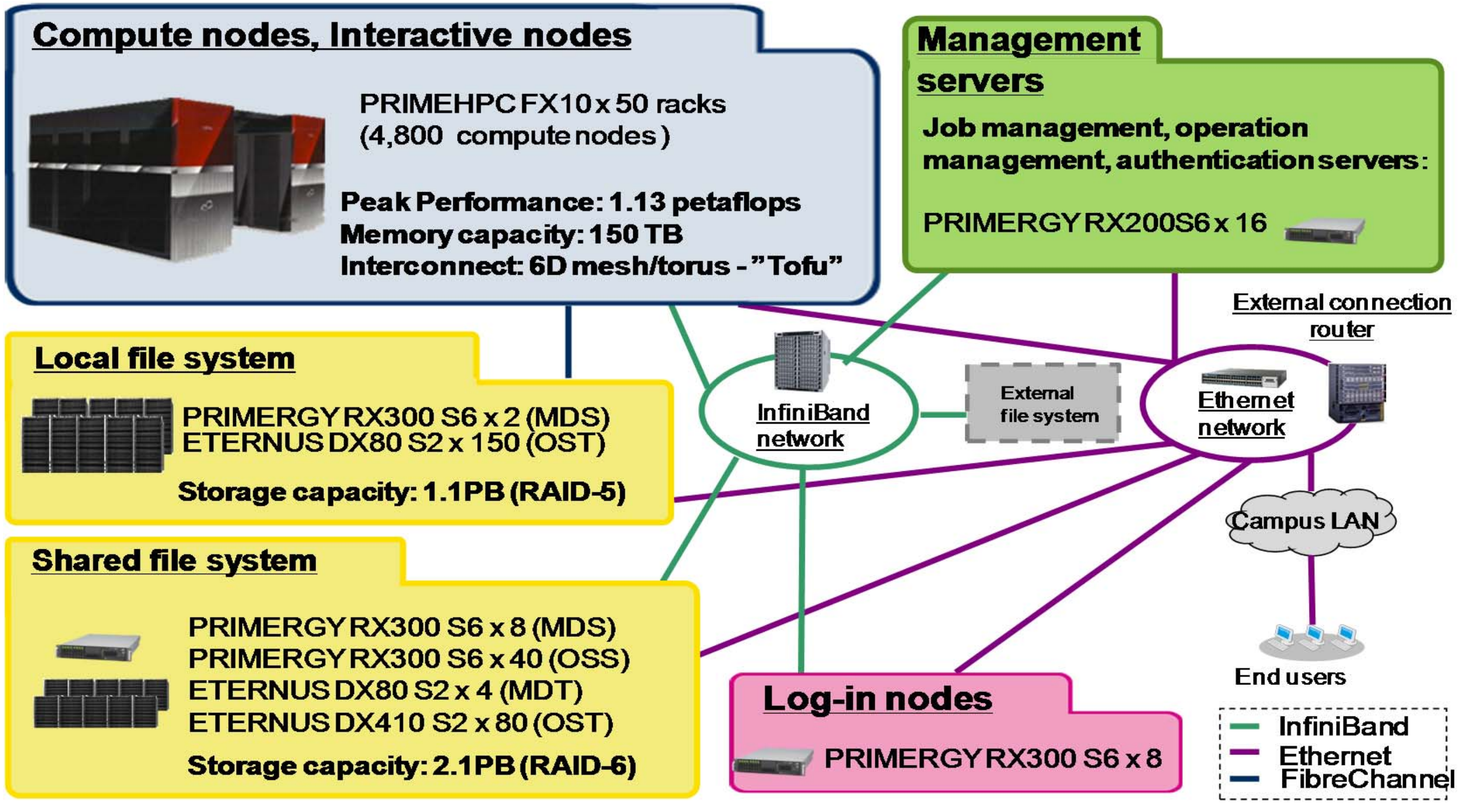


Hybrid：Hierarchal Structure



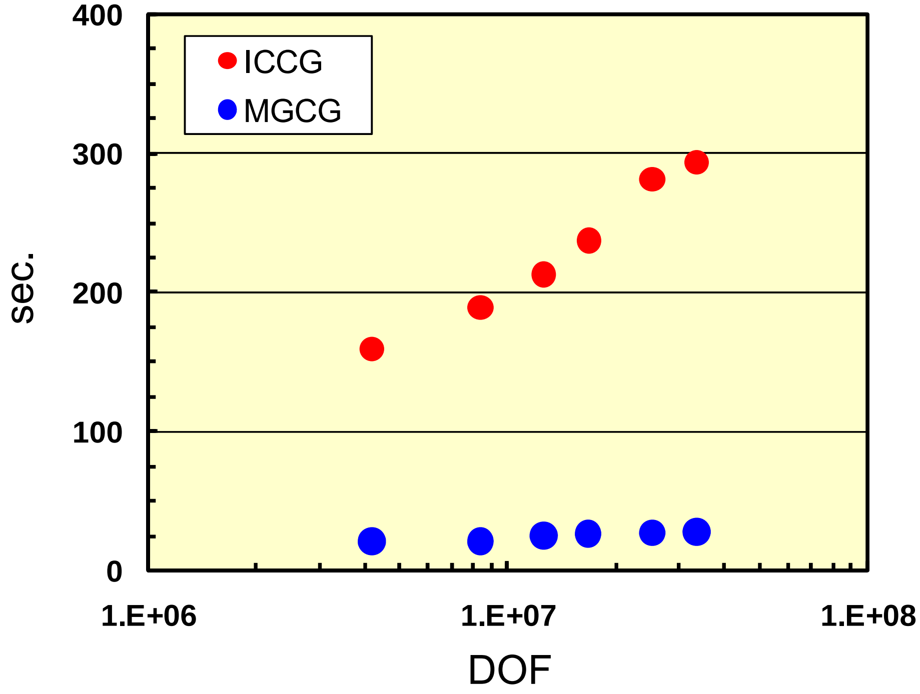
Fujitsu PRIMEHPC FX10 (Oakleaf-FX) at the U. Tokyo

- SPARC64 Ixfx (4,800 nodes, 76,800 cores)
- Commercial version of K computerx
- Peak: 1.13 PFLOPS (1.043 PF, 26th, 41th TOP 500 in 2013 June.)
- Memory BWTH 398 TB/sec.


Compute nodes, Interactive nodes
PRIMEHPC FX10 x 50 racks
(4,800 compute nodes)
Peak Performance: 1.13 petaflops
Memory capacity: 150 TB
Interconnect: 6D mesh/torus - "Tofu"
Management servers
Job management, operation management, authentication servers:
PRIMERGY RX200S6 x 16
Local file system
PRIMERGY RX300 S6 x 2 (MDS)
ETERNUS DX80 S2 x 150 (OST)
Storage capacity: 1.1PB (RAID-5)
Shared file system
PRIMERGY RX300 S6 x 8 (MDS)
PRIMERGY RX300 S6 x 40 (OSS)
ETERNUS DX80 S2 x 4 (MDT)
ETERNUS DX410 S2 x 80 (OST)
Storage capacity: 2.1PB (RAID-6)
InfiniBand network
External file system
Ethernet network
External connection router
Campus LAN
End users
Log-in nodes
PRIMERGY RX300 S6 x 8
InfiniBand
Ethernet
FibreChannel

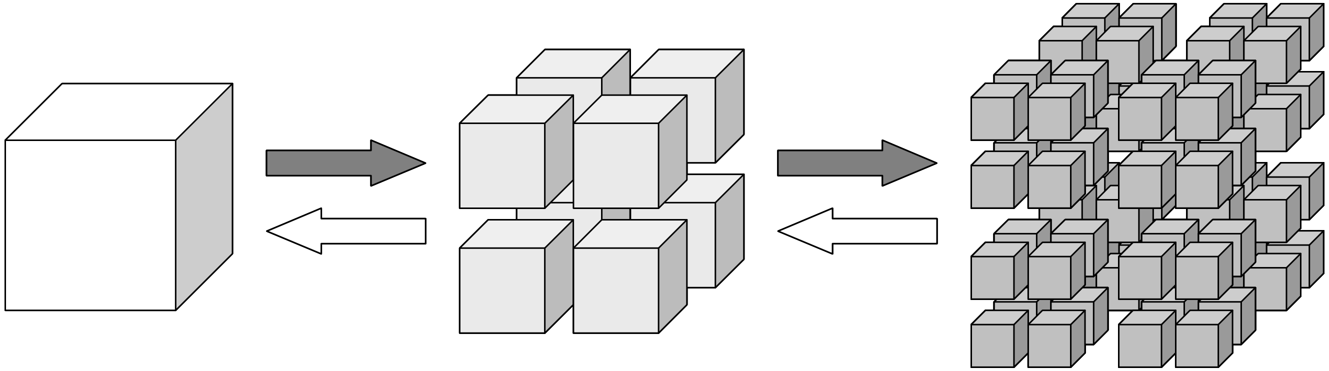
Multigrid

- Scalable Multi-Level Method using Multilevel Grid for Solving Linear Eqn's
  - Computation Time ~ O(N) (N: # unknowns)
  - Good for large-scale problems
- Preconditioner for Krylov Iterative Linear Solvers
  - MGCG



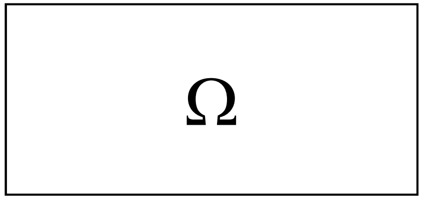
Linear Solvers

- Preconditioned CG Method
  - Multigrid Preconditioning (MGCG)
  - IC(0) for Smoothing Operator (Smoother): good for ill-conditioned problems
- Parallel Geometric Multigrid Method
  - 8 fine meshes (children) form 1 coarse mesh (parent) in isotropic manner (octree)
  - V-cycle
  - Domain-Decomposition-based: Localized Block-Jacobi, Overlapped Additive Schwartz Domain Decomposition (ASDD)
  - Operations using a single core at the coarsest level (redundant)



Overlapped Additive Schwartz Domain Decomposition Method

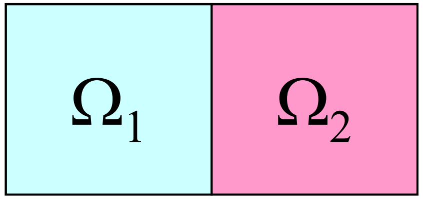
ASDD: Localized Block-Jacobi Precond. is stabilized



Global Operation

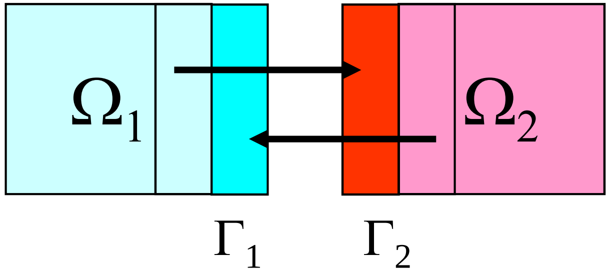
$$Mz = r$$

Ω$_\mathrm{i}$: Internal （i≦N）
Γ$_\mathrm{i}$: External（i＞N）



Local Operation

$$z_{\Omega_1} = M_{\Omega_1}^{-1} r_{\Omega_1}, \quad z_{\Omega_2} = M_{\Omega_2}^{-1} r_{\Omega_2}$$



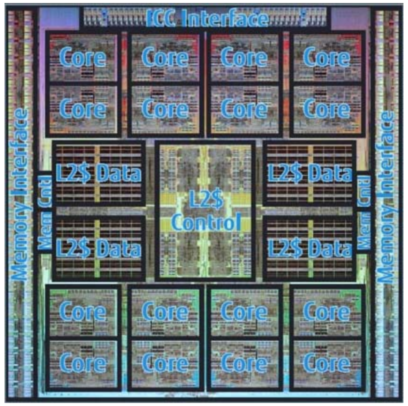
Global Nesting Correction

$$z_{\Omega_1}^{n} = z_{\Omega_1}^{n-1} + M_{\Omega_1}^{-1}\left(r_{\Omega_1} - M_{\Omega_1} z_{\Omega_1}^{n-1} - M_{\Gamma_1} z_{\Gamma_1}^{n-1}\right)$$

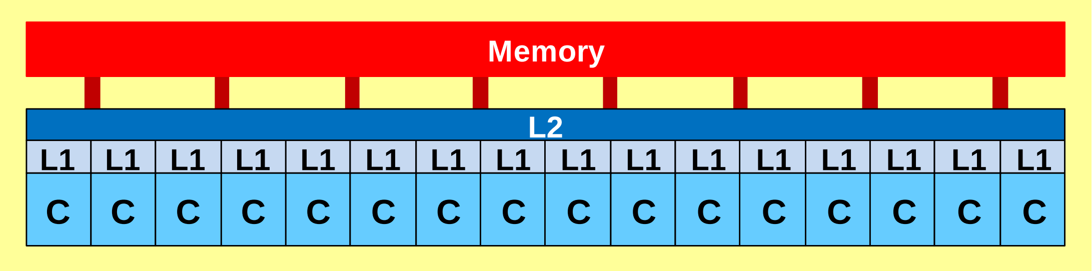
$$z_{\Omega_2}^{n} = z_{\Omega_2}^{n-1} + M_{\Omega_2}^{-1}\left(r_{\Omega_2} - M_{\Omega_2} z_{\Omega_2}^{n-1} - M_{\Gamma_2} z_{\Gamma_2}^{n-1}\right)$$

Computations on Fujitsu FX10

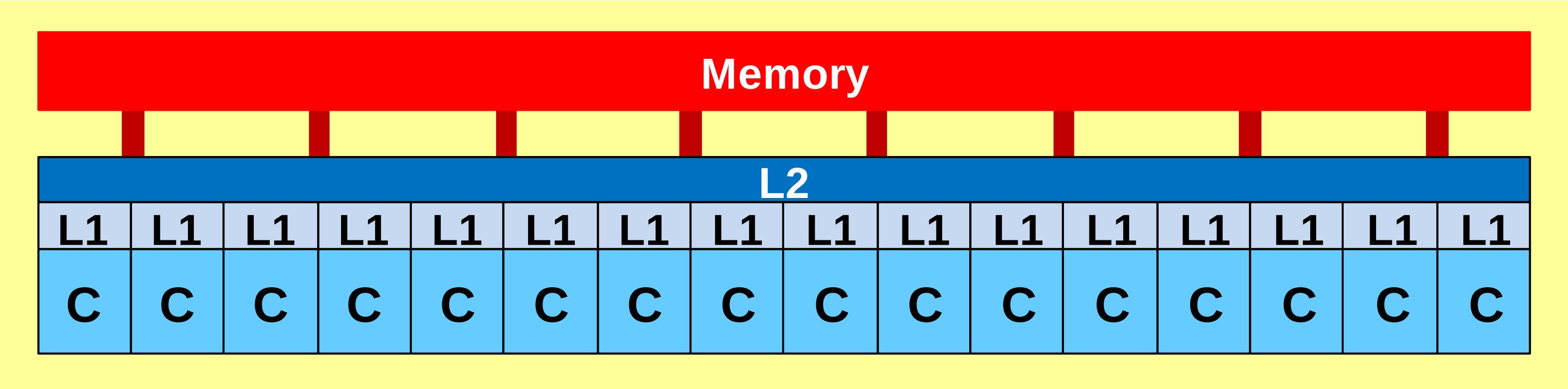
- Fujitsu PRIMEHPC FX10 at U.Tokyo (Oakleaf-FX)
  - 16 cores/node, flat/uniform access to memory
- Up to 4,096 nodes (65,536 cores) (Large-Scale HPC Challenge)
  - Max 17,179,869,184 unknowns
  - Flat MPI, HB 4x4, HB 8x2, HB 16x1
    - HB MxN: M-threads x N-MPI-processes on each node



- Weak Scaling
  - 64^3 cells/core


Memory
L2
L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1
C C C C C C C C C C C C C C C C

- Strong Scaling
  - $128^3 \times 8$= 16,777,216 unknowns, from 8 to 4,096 nodes
- Network Topology is not specified
  - 1D


Memory
L2
L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1 L1
C C C C C C C C C C C C C C C C

HB M x N

Number of OpenMP threads per a single MPI process

Number of MPI process per a single node

Reordering for extracting parallelism in each domain (= MPI Process)

- Krylov Iterative Solvers
  - Dot Products
  - SMVP
  - DAXPY
  - Preconditioning
- IC/ILU Factorization, Forward/Backward Substitution
  - Global Data Dependency
  - Reordering needed for parallelism ([KN 2003] on the Earth Simulator, KN@CMCIM-2002)
  - Multicoloring, RCM, CM-RCM

Parallerization of ICCG

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

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

(Global) Data Dependency:

Writing/reading may occur simultaneously, hard to parallelize

IC Factorization

```
do i= 1, N
  VAL= D(i)
  do k= indexL(i-1)+1, indexL(i)
    VAL= VAL - (AL(k)**2) * W(itemL(k),DD)
  enddo
  W(i,DD)= 1.d0/VAL
enddo
```

Forward Substitution

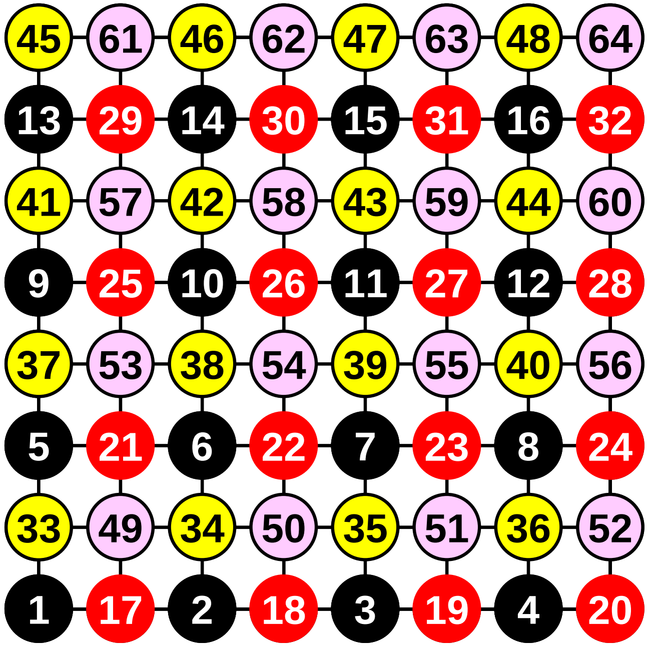
```
do i= 1, N
  WVAL= W(i,Z)
  do k= indexL(i-1)+1, indexL(i)
    WVAL= WVAL -  AL(k) * W(itemL(k),Z)
  enddo
  W(i,Z)= WVAL * W(i,DD)
enddo
```

OpenMP for SpMV: Straightforward NO data dependency

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

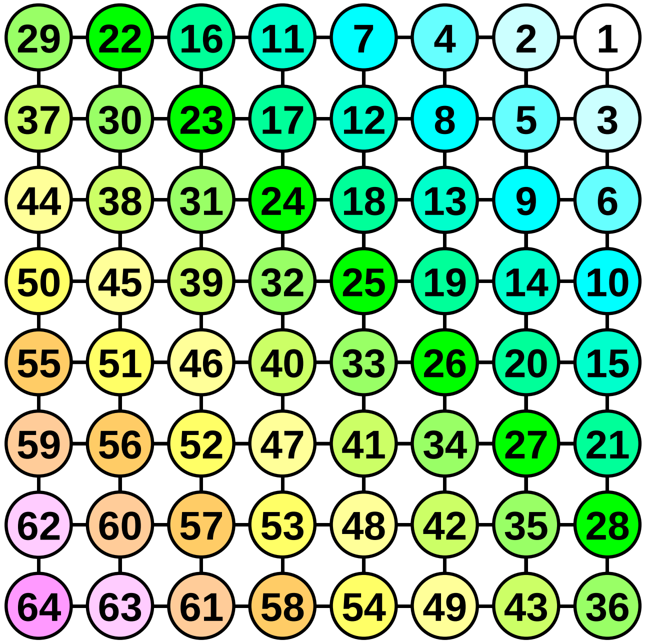
Ordering Methods

Elements in “same color” are independent: to be parallelized



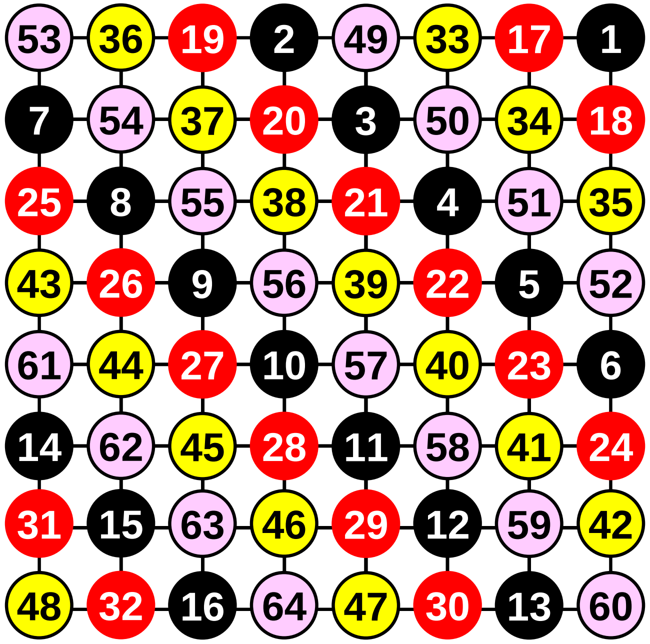
45	61	46	62	47	63	48	64
13	29	14	30	15	31	16	32
41	57	42	58	43	59	44	60
9	25	10	26	11	27	12	28
37	53	38	54	39	55	40	56
5	21	6	22	7	23	8	24
33	49	34	50	35	51	36	52
1	17	2	18	3	19	4	20

MC (Color#=4)
Multicoloring



29	22	16	11	7	4	2	1
37	30	23	17	12	8	5	3
44	38	31	24	18	13	9	6
50	45	39	32	25	19	14	10
55	51	46	40	33	26	20	15
59	56	52	47	41	34	27	21
62	60	57	53	48	42	35	28
64	63	61	58	54	49	43	36

RCM
Reverse Cuthill-Mckee

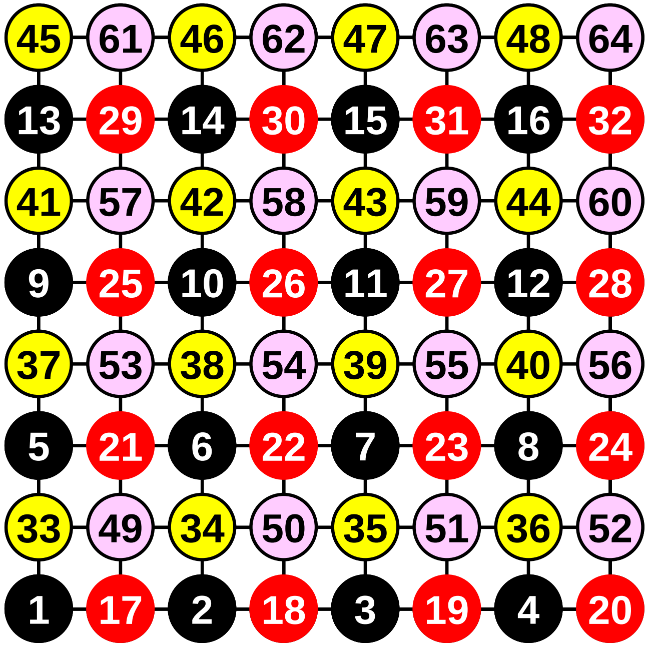


53	36	19	2	49	33	17	1
7	54	37	20	3	50	34	18
25	8	55	38	21	4	51	35
43	26	9	56	39	22	5	52
61	44	27	10	57	40	23	6
14	62	45	28	11	58	41	24
31	15	63	46	29	12	59	42
48	32	16	64	47	30	13	60

CM-RCM (Color#=4)
Cyclic MC + RCM

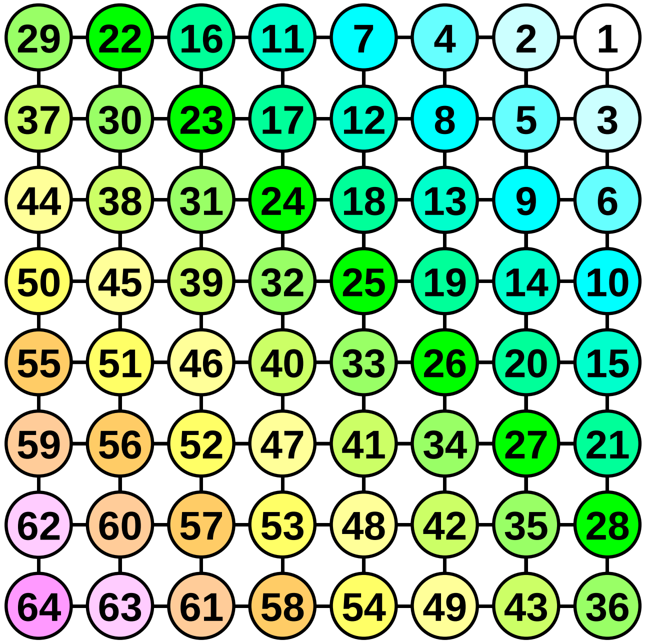
Ordering Methods

Elements in “same color” are independent: to be parallelized



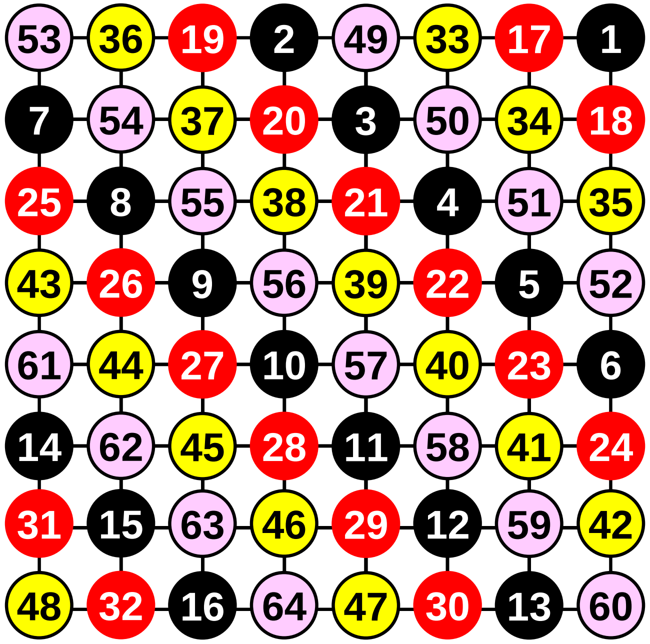
45	61	46	62	47	63	48	64
13	29	14	30	15	31	16	32
41	57	42	58	43	59	44	60
9	25	10	26	11	27	12	28
37	53	38	54	39	55	40	56
5	21	6	22	7	23	8	24
33	49	34	50	35	51	36	52
1	17	2	18	3	19	4	20

MC (Color#=4)
Multicoloring



29	22	16	11	7	4	2	1
37	30	23	17	12	8	5	3
44	38	31	24	18	13	9	6
50	45	39	32	25	19	14	10
55	51	46	40	33	26	20	15
59	56	52	47	41	34	27	21
62	60	57	53	48	42	35	28
64	63	61	58	54	49	43	36

RCM
Reverse Cuthill-Mckee



53	36	19	2	49	33	17	1
7	54	37	20	3	50	34	18
25	8	55	38	21	4	51	35
43	26	9	56	39	22	5	52
61	44	27	10	57	40	23	6
14	62	45	28	11	58	41	24
31	15	63	46	29	12	59	42
48	32	16	64	47	30	13	60

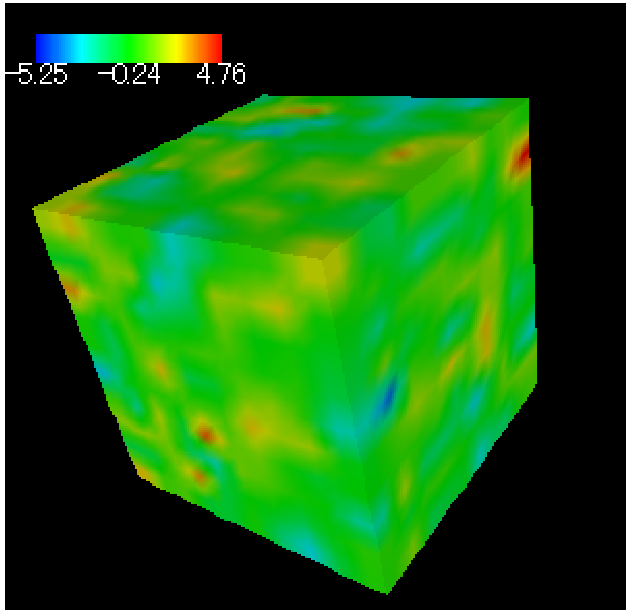
CM-RCM (Color#=4)
Cyclic MC + RCM

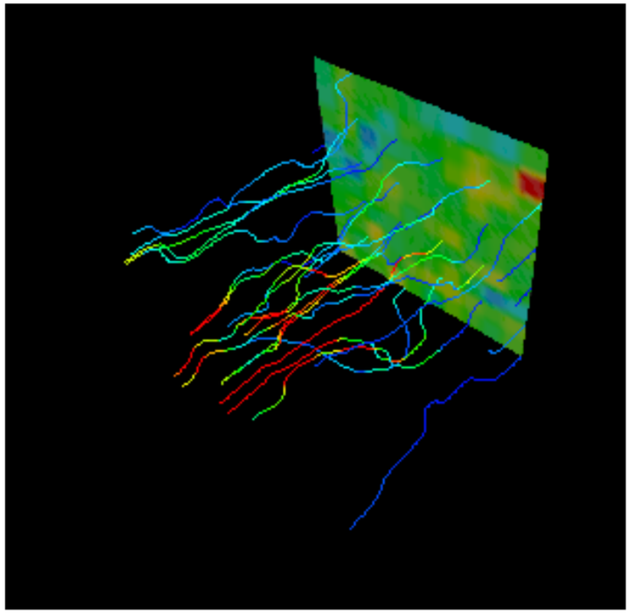
Motivation

- Optimization of Parallel MGCG
  - Conjugate Gradient Solver with Multigrid Preconditioning
  - OpenMP/MPI Hybrid Parallel Programming Model
  - Efficiency & Convergence
- Communications are expensive
  - Serial Communications
    - ➢ Data Transfer through Hierarchical Memory
  - Parallel Communications
    - ➢ Message Passing through Network
- There are a lot of talks related to Multigrid and CA in SIAM PP14.
  - “Coarse Grid Solver” is important
    - ➢ Efficiency & Convergence

Parallel MG Solvers: pGW3D-FVM

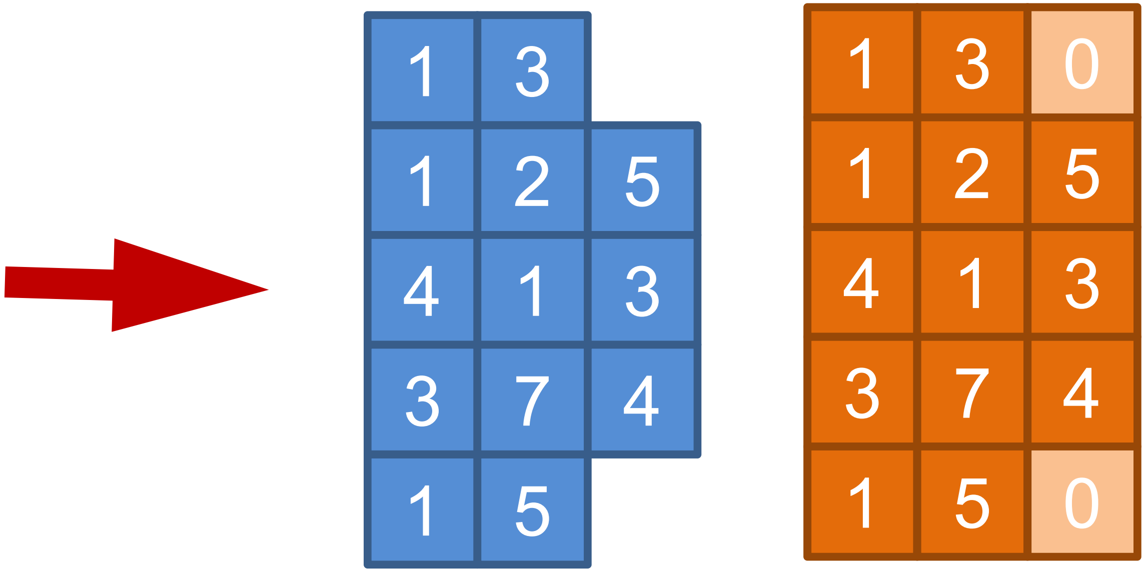
- Storage format of coefficient matrices (Serial Comm.)
  - CRS (Compressed Row Storage)
  - ELL (Ellpack-Itpack)
- Comm. /Sych. Reducing MG (Parallel Comm.)
  - Coarse Grid Aggregation (CGA)
  - Hierarchical CGA: Communication Reducing CGA





ELL: Fixed Loop-length, Nice for Pre-fetching

$$\begin{bmatrix} 1 & 3 & 0 & 0 & 0 \\ 1 & 2 & 5 & 0 & 0 \\ 4 & 1 & 3 & 0 & 0 \\ 0 & 3 & 7 & 4 & 0 \\ 1 & 0 & 0 & 0 & 5 \end{bmatrix}$$



1	3	
1	2	5
4	1	3
3	7	4
1	5	

(a) CRS

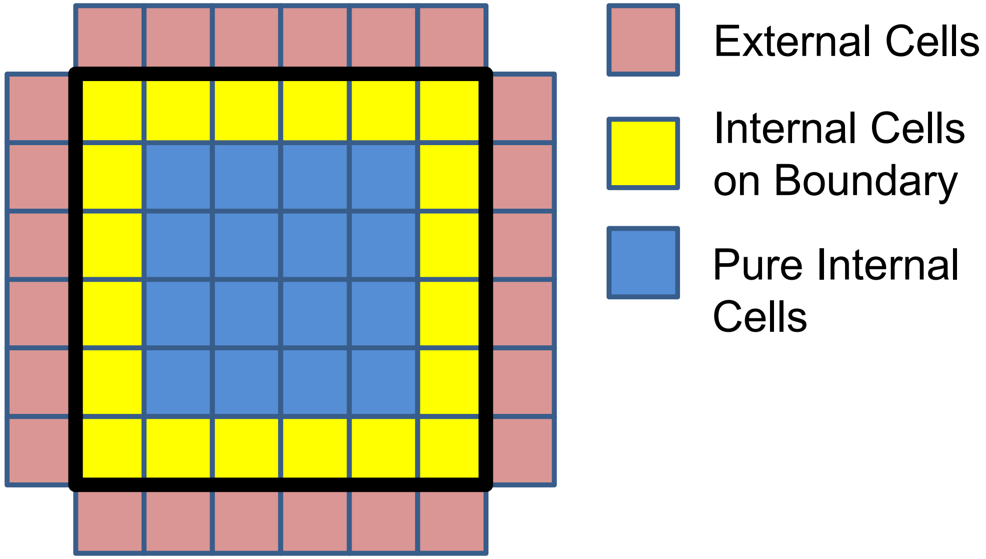
1	3	0
1	2	5
4	1	3
3	7	4
1	5	0

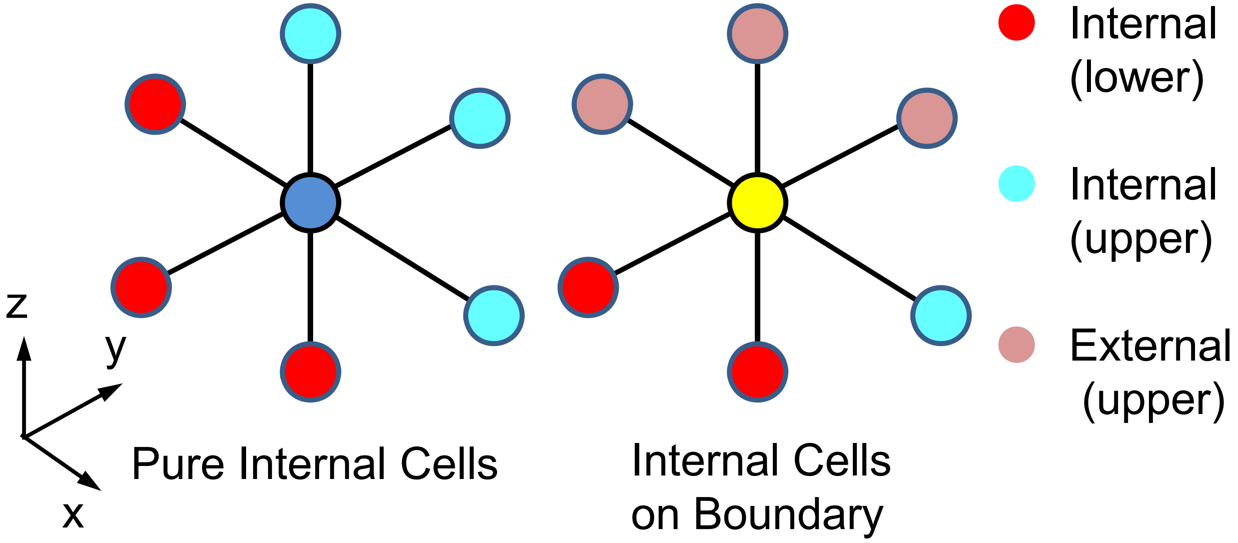
(b) ELL

Special Treatment for “Boundary” Cells connected to “Halo”

- Distribution of Lower/Upper Non-Zero Off-Diagonal Components

- Pure Internal Cells
  - L: ~3, U: ~3
- Boundary Cells
  - L: ~3, U: ~6





Original ELL: Backward Subst.

Cache is not well-utilized: IAUnew(6,N), Aunew(6,N)

```
      do icol= NHYP(lev), 1, -1
        if (mod(icol,2).eq.0) then
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 3
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
         else
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 6
              SW= SW + AUnew(j,icel)*Rmg(IAUnew(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        endif
      enddo
```

for Pure Internal Cells

for Boundary Cells

IAUnew (6,N), AUnew (6,N)

Improved ELL: Backward Subst.

Cache is well-utilized, separated: AUnew3/AUnew6

```
      do icol= NHYP(lev), 1, -1
        if (mod(icol,2).eq.0) then
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 3
              SW= SW + AUnew3(j,icel)*Rmg(IAUnew3(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
         else
!$omp parallel do private (ip,icel,j,SW)
          do ip= 1, PEsmpTOT
          do icel= SMPindex(icol-1,ip,lev)+1, SMPindex(icol,ip,lev)
            SW= 0.0d0
            do j= 1, 6
              SW= SW + AUnew6(j,icel)*Rmg(IAUnew6(j,icel))
            enddo
            Rmg(icel)= Rmg(icel) - SW*DDmg(icel)
          enddo
          enddo
        endif
      enddo
```

for Pure Internal Cells

for Boundary Cells

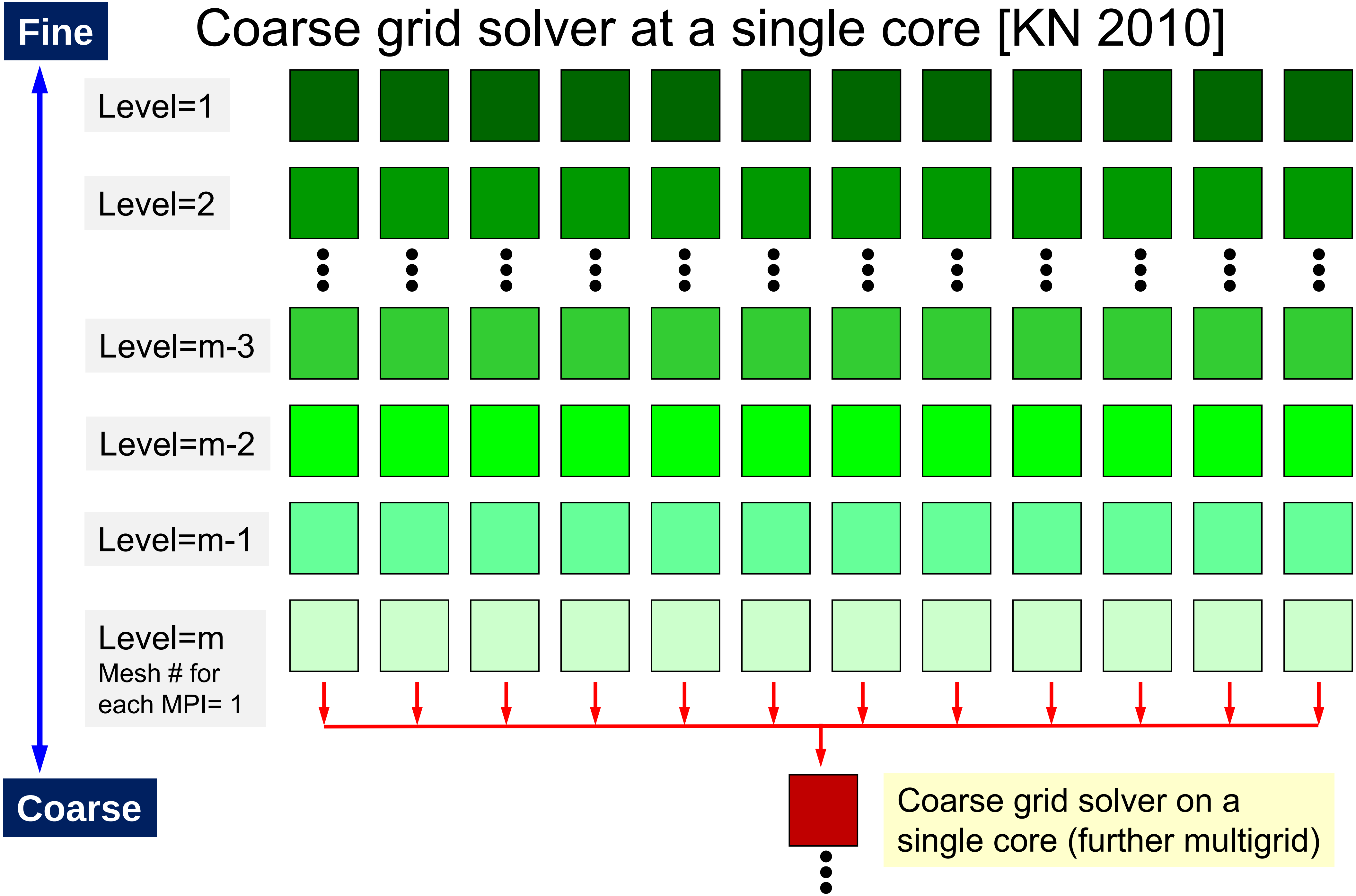
```
IAUnew3(3,N), AUnew3(3,N)
IAUnew6(6,N), AUnew6(6,N)
```

Analyses by Detailed Profiler of Fujitsu FX10, single node, Flat MPI, RCM (Multigrid Part), 64^3cells/core, 1-node

	Instruction	L1D miss	L2 miss	SIMD Op. Ratio	GFLOPS
CRS	1.53×10^9	2.32×10^7	1.67×10^7	30.14%	6.05
Original ELL	4.91×10^8	1.67×10^7	1.27×10^7	93.88%	6.99
Improved ELL	4.91×10^8	1.67×10^7	9.14×10^6	93.88%	8.56

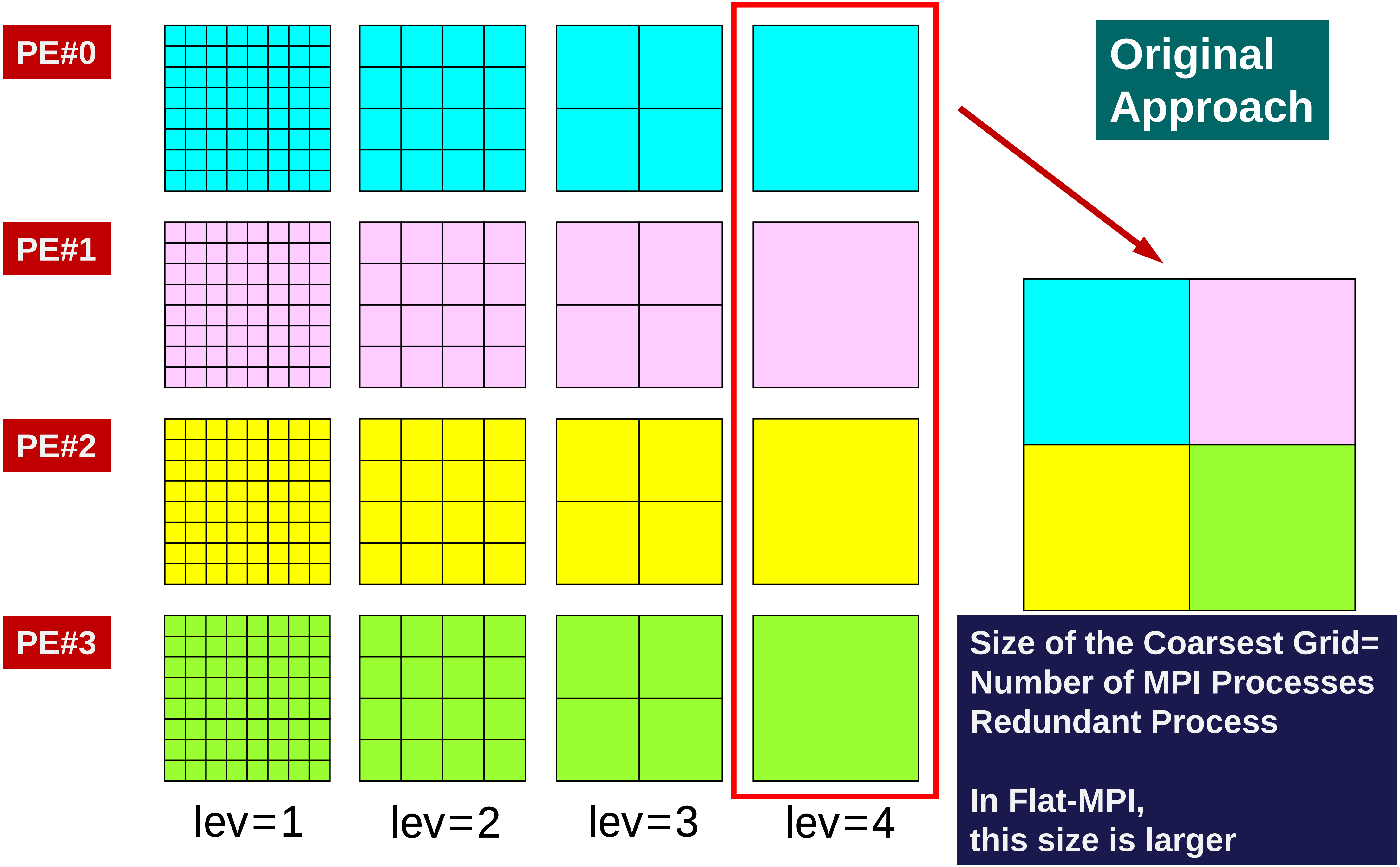
Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]



Coarse grid solver on a single core (further multigrid)

Coarse Grid Solver on a Single Core



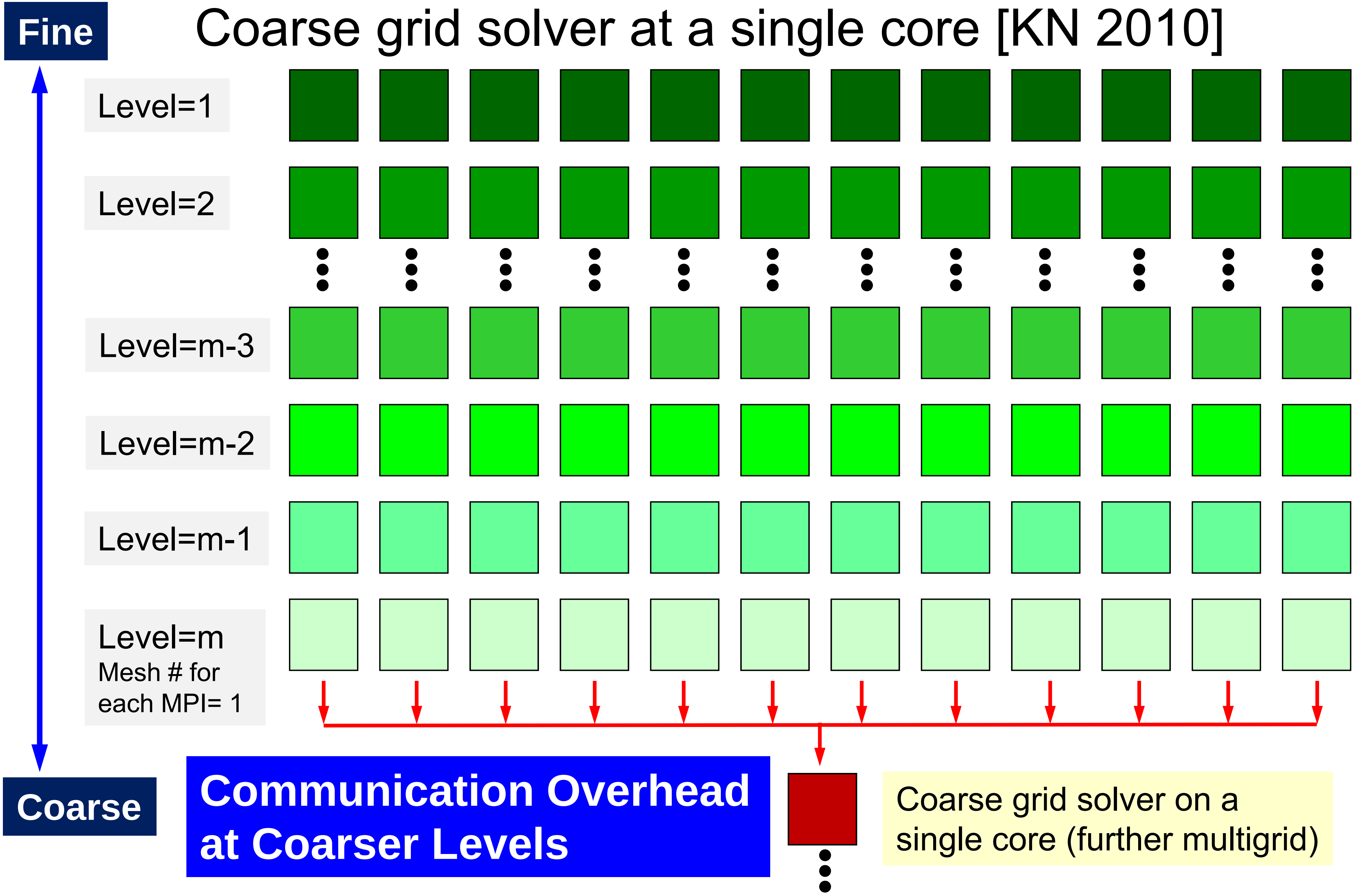
Original Approach

Size of the Coarsest Grid= Number of MPI Processes Redundant Process

In Flat-MPI, this size is larger

Original Approach (restriction)

Coarse grid solver at a single core [KN 2010]



Fine

Level=1

Level=2

Level=m-3

Level=m-2

Level=m-1

Level=m
Mesh # for
each MPI= 1

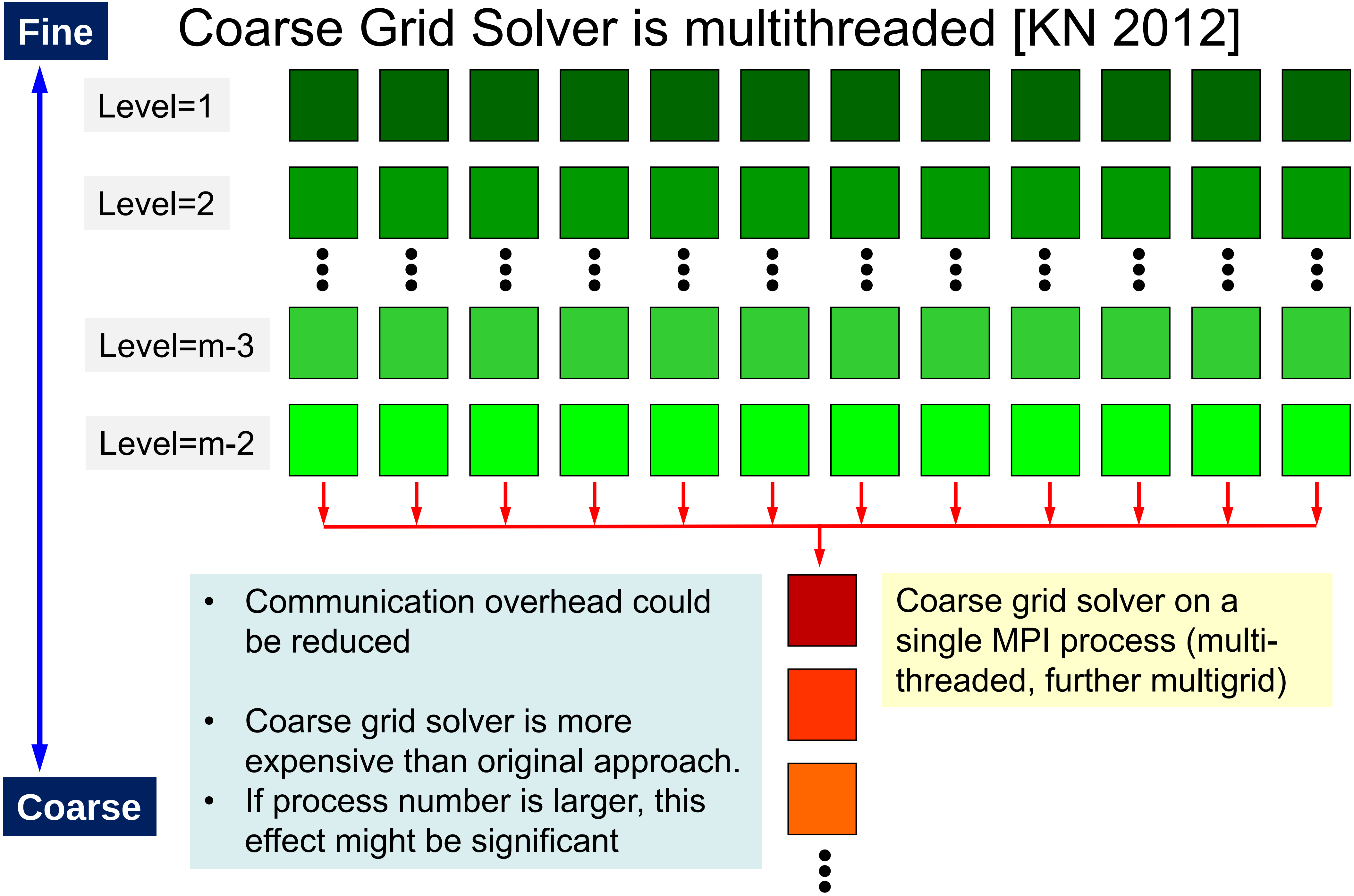
Coarse

Communication Overhead at Coarser Levels

Coarse grid solver on a single core (further multigrid)

Coarse Grid Aggregation (CGA)

Coarse Grid Solver is multithreaded [KN 2012]



- Communication overhead could be reduced
- Coarse grid solver is more expensive than original approach.
- If process number is larger, this effect might be significant

Coarse grid solver on a single MPI process (multi-threaded, further multigrid)

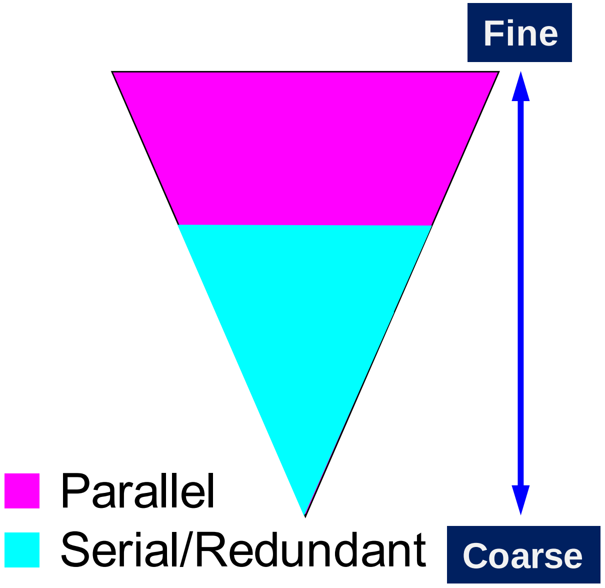
Results

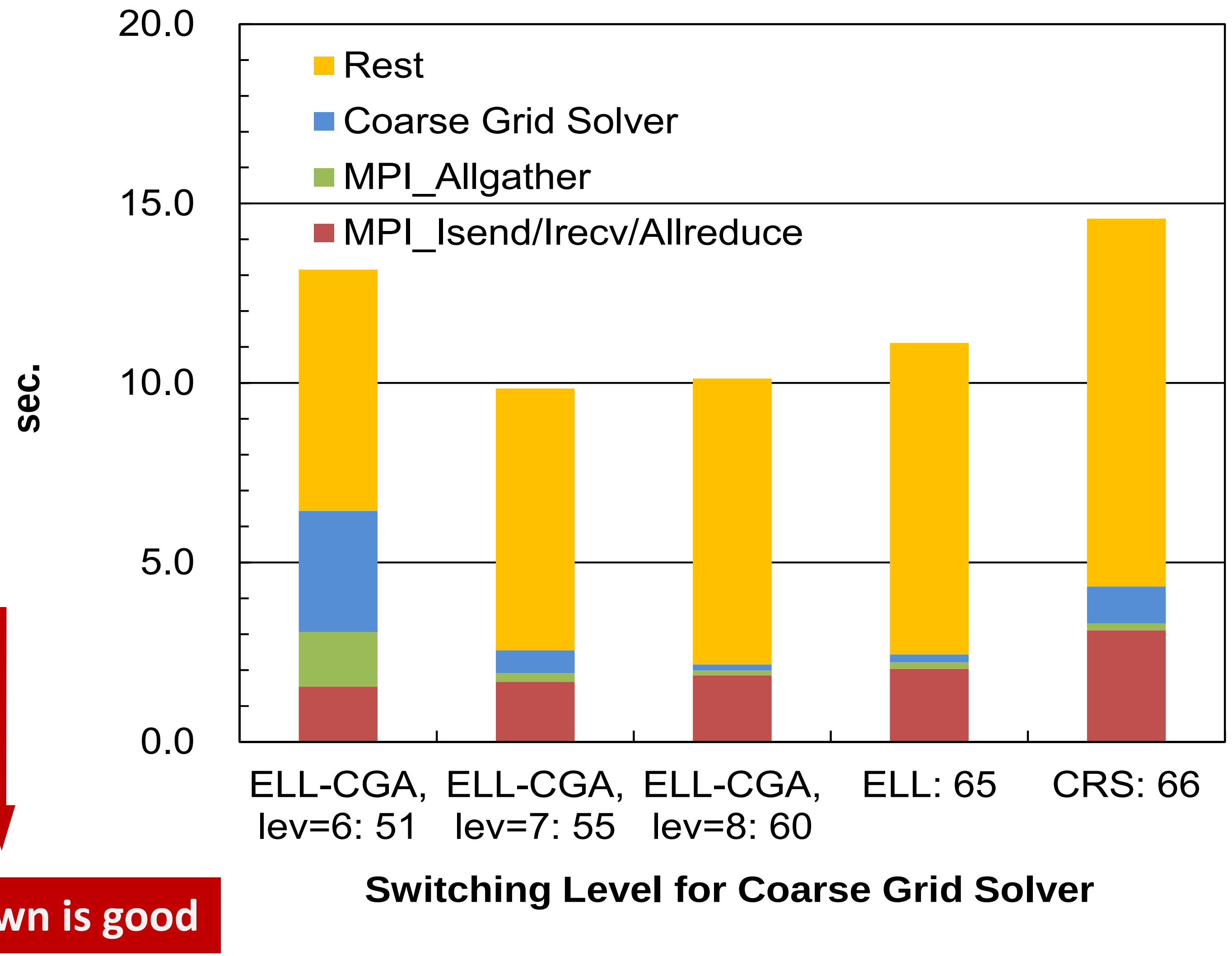
CASE	Matrix	Coarse Grid	Reordering
C0	CRS	Single Core	CM-RCM
CX	ELL	Single Core	RCM
C1	CRS	CGA	CM-RCM
C2	ELL (original)	CGA	RCM
C3	ELL (new)	CGA	RCM

Class	Size	
Weak Scaling	64^3 cells/core	262,144
Strong Scaling	256^3 cells	16,777,216

Results at 4,096 nodes (1.72×10^{10} DOF) (Fujitsu FX10: Oakleaf-FX): HB 8x2

lev: switching level to “coarse grid solver”, Opt. Level= 7

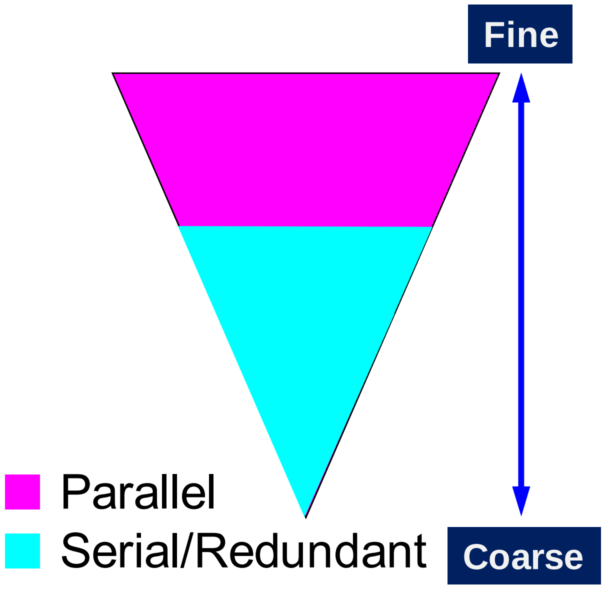


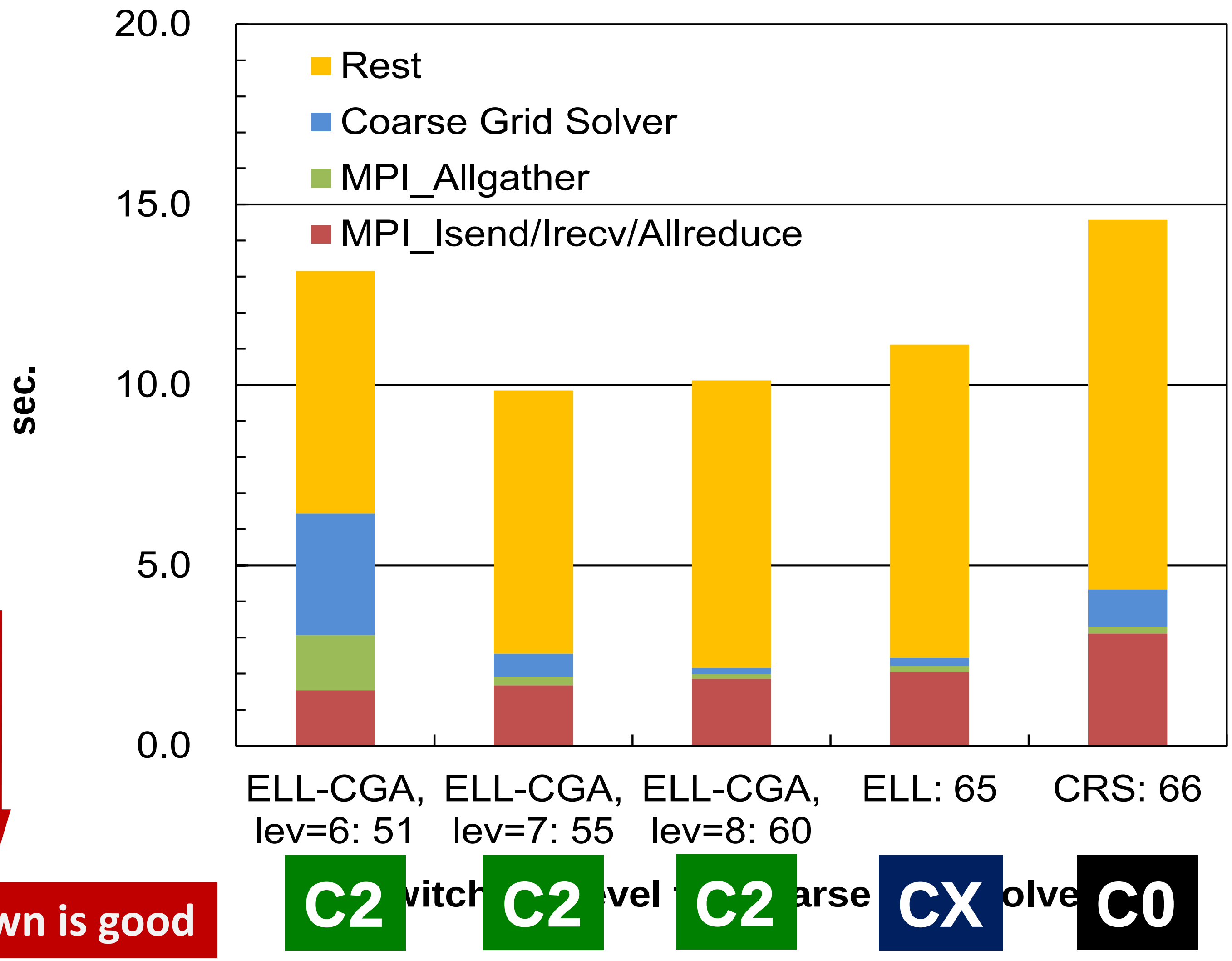


Down is good

Results at 4,096 nodes (1.72×10^{10} DOF) (Fujitsu FX10: Oakleaf-FX): HB 8x2

lev: switching level to “coarse grid solver”, Opt. Level= 7



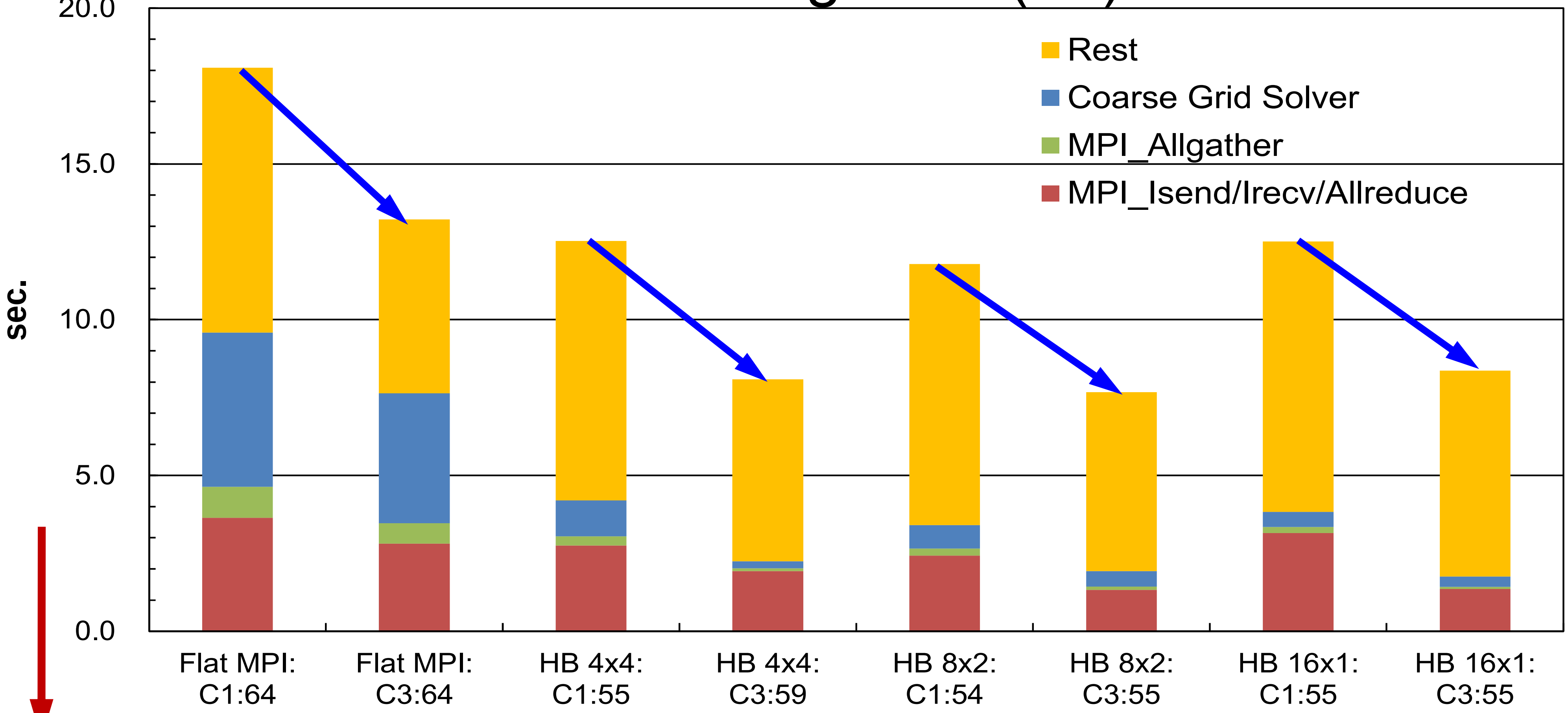


Down is good

Weak Scaling at 4,096 nodes C1 (CGA+CRS) -> C3 (CGA+ELL-new)

17,179,869,184 meshes (64^3 meshes/core)

best switching level (=7)



Flat MPI

HB 4x4

HB 8x2

HB 16x1

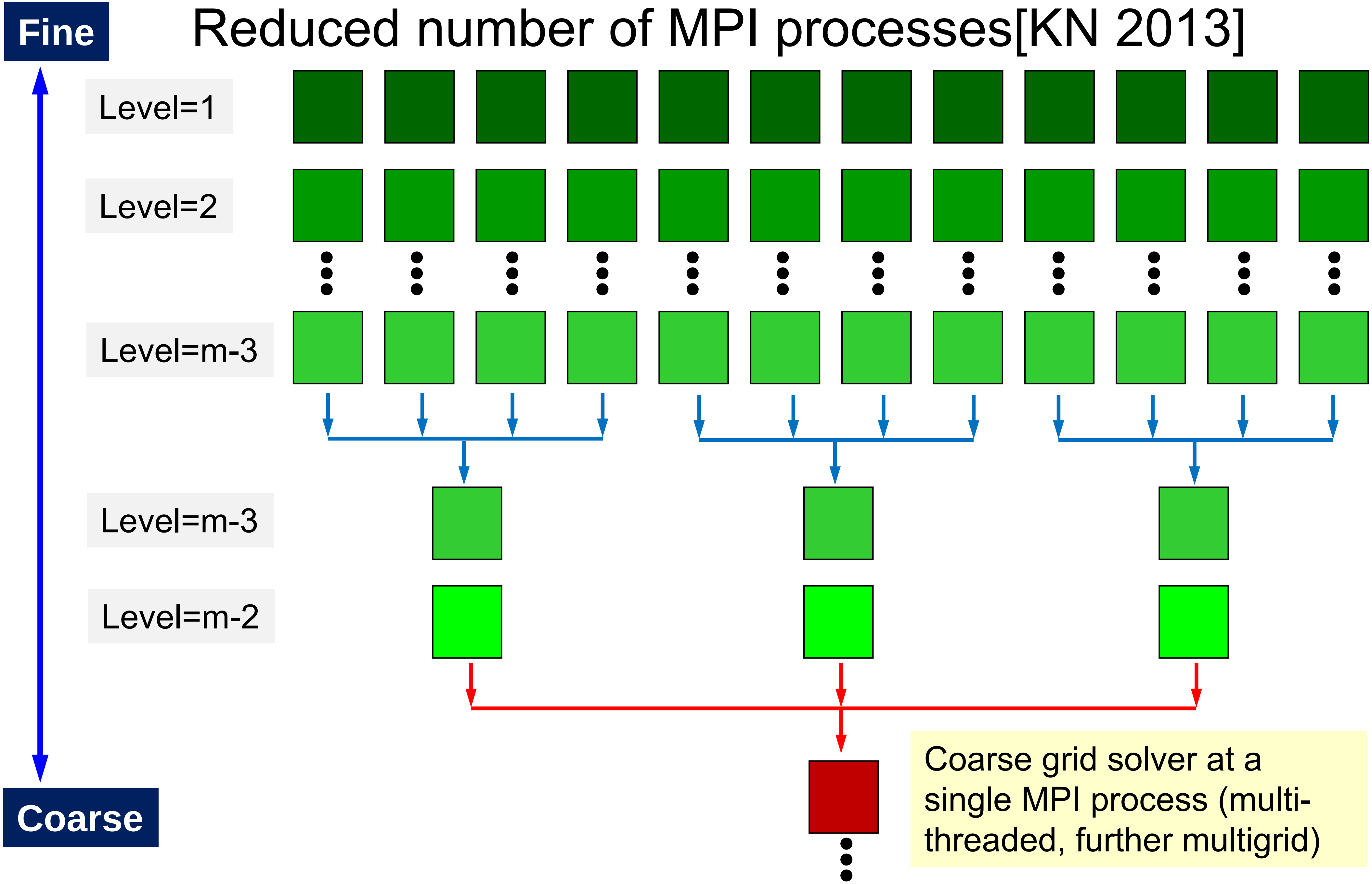
Down is good

Summary so far ...

- “Coarse Grid Aggregation (CGA)” is effective for stabilization of convergence at O(10^4) cores for MGCG
  - Smaller number of parallel domains
  - HB 8x2 is the best at 4,096 nodes
  - Flat MPI, HB 4x4
    - Coarse grid solvers are more expensive, because their number of MPI processes are more than those of HB 8x2 and HB 16x1.
- ELL format is effective !
  - C1 (CRS) -> C2 (ELL-org.): +20-30%
  - C2 -> C3(ELL-new) : +20-30%
- Coarse Grid Solver
  - Very expensive for cases with more than O(10^5) cores
  - Memory of a single node is not enough
  - Multiple nodes should be utilized for coarse grid solver

Hierarchical CGA: Comm. Reducing MG

Reduced number of MPI processes[KN 2013]



Coarse grid solver at a single MPI process (multi-threaded, further multigrid)

Results: *h*CGA

CASE	Matrix	Coarse Grid	Reordering
C3	ELL (new)	CGA	RCM
C4	ELL (new)	hCGA	RCM

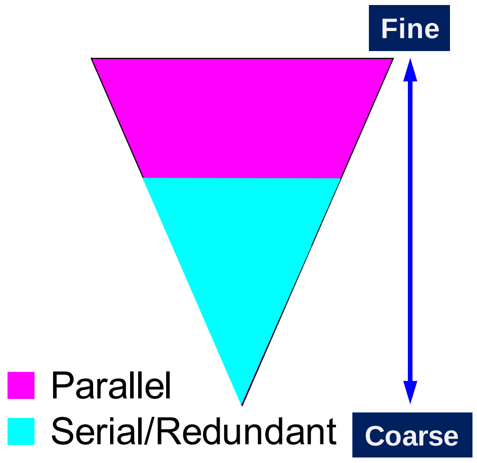
Class	Size	
Small	16^3 cells/core	4,096
Medium	32^3 cells/core	32,768
Large	64^3 cells/core	262,144
X-Large	128^3 cells/core	2,097,152

Results at 4,096 nodes (1.72x10^{10} DOF)

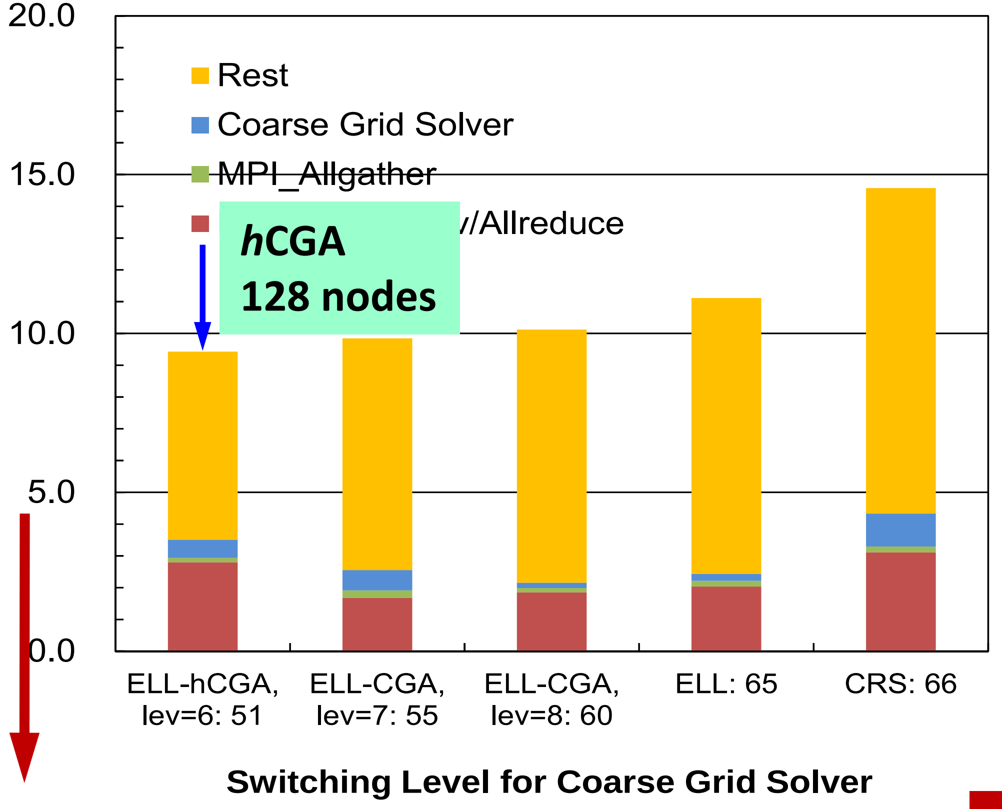
lev: switching level to “coarse grid solver”

Opt. Level= 7, HB 8x8 is the best

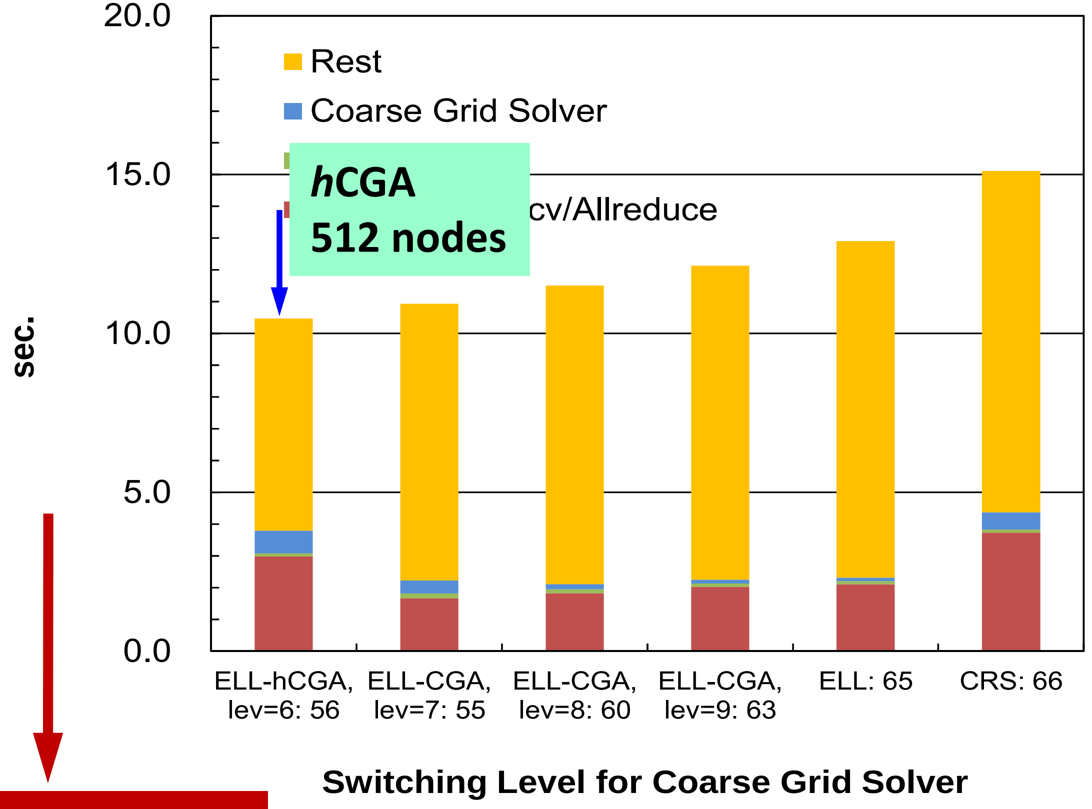
”C2 (ELL org.)” is used, DOWN is GOOD



HB 8x2



HB 16x1



Down is good

Summary

- *h*CGA is also effective, but not so significant.
  - Generally speaking, convergence becomes worse than CGA.
  - Effective for cases with
    - smaller problem size per core (Small or Large)
    - larger number of nodes (MPI processes) (64 nodes or 512 nodes)
    - larger number of MPI process per node (Flat MPI or HB 16x1)
- Future/On-Going Works and Open Problems
  - Algorithms
    - CA-Multigrid (for coarser levels), CA-SPAI
  - Strategy for Automatic Selection
    - switching level, number of processes for *h*CGA, optimum color #
  - More Flexible ELL for Unstructured Grids
  - Optimized MPI (co-design)
    - e.g. MPI on Fujitsu FX10 utilizing RDMA with persistent communications
    - MPI_Allreduce: 0.80 sec. of 2.50 sec (comm.) is spent for dot products @ 4,096 nodes
  - Xeon Phi Clusters
    - Hybrid 240(T)x1(P) is not the only choice

Reference:

Kengo Nakajima
“Large-scale Simulations of 3D Groundwater
Flow using Parallel Geometric Multigrid Method”

Procedia Computer Science 18, 1265-1274, Proceedings of IHPCES 2013 (Third International Workshop on Advances in High-Performance Computational Earth Sciences: Applications and Frameworks) in conjunction with ICCS 2013, Barcelona, Spain