A Parallel auxiliary Grid AMG Method

Lu Wang
Penn State University
March 20th, 2013
GTC 2013

Joint work with:
Xiaozhe Hu(PSU), Jinchao Xu(PSU), and Jonathan Cohen(NVIDIA)

OUTLINE

- Introduction
- Auxiliary Grid Based AMG
- Parallel Implementation on GPU
- Numerical Results
- Conclusions and future works

Models⇒Algorithms⇒Algebraic Systems

Models⇒Algorithms⇒Algebraic Systems

Scientific Simulations

Models⇒Algorithms⇒Algebraic Systems

Scientific Simulations

- CFD Simulation
- Electromagnetism
- Continuum mechanics

Models⇒Algorithms⇒Algebraic Systems

Scientific Simulations

Problems	Partial Differential Equations
Poisson	$-\Delta u = f$
Reaction-Diffusion	$-\nabla \cdot (\mu(x)\nabla u) + c(x)u = f$
H(curl)	$\nabla \times \nabla \times u + u = f$
H(div)	$-\nabla\nabla \cdot u + u = f$
Maxwell	$B_t + \nabla \times E = 0,\ \nabla \cdot D = \rho,$ $D_t - \nabla \times H = -J,\ \nabla \cdot B = 0$
Darcy	$-\nabla \cdot u = f,\ u = \mu\nabla p$
Stokes	$-\Delta u + \nabla p = f,\ \nabla \cdot u = 0$
Brinkman	$-\nabla \cdot (\mu(x)\nabla u) + c(x)u + \nabla p = f,\ \nabla \cdot u = g$
Linear Elasticity	$-\Delta u - \frac{\nu}{1-2\nu}\nabla\nabla \cdot u = f$
Biharmonic (plate)	$\Delta^2 u = f$
Reissner-Mindlin	$-\operatorname{div}(\mathcal{C}D(\phi)) = \lambda t^{-2}(\phi - \nabla\omega) = 0,$ $\lambda t^{-2}(-\Delta\omega + \operatorname{div}\phi) = g$
R–C–D	$u_t + v(x) \cdot \nabla u - \nabla \cdot (\mu(x)\nabla u) + c(x)u = f$
Navier–Stokes	$u_t + (u \cdot \nabla)u - \mu\Delta u + \nabla p = f,\ \nabla \cdot u = 0$
Johnson-Seglman	$u_t + (u \cdot \nabla)u - \mu_s\Delta u + \nabla p = \nabla \cdot \tau,\ \nabla \cdot u = 0,$ $\tau + \mathrm{Wi}[u_t + (u \cdot \nabla)u - R\tau - \tau R^T] = \frac{1}{2}\mu_p \mathcal{D}u$ $R = \frac{a+1}{2}\nabla u + \frac{a-1}{2}\nabla u^T$
MHD	$u_t + (u \cdot \nabla)u + \nabla p = \mu\Delta u + \mu_0 J \times B,\ \nabla \cdot u = 0,$ $B_t + \nabla \times E = 0,\ \nabla \times B = \mu_0 J,$ $\nabla \cdot B = 0,\ E + u \times B = \sigma_0^{-1} J$
Black-oil model	

Models⇒Algorithms⇒Algebraic Systems

Scientific Simulations

PDE model

Algorithms

Models⇒Algorithms⇒Algebraic Systems

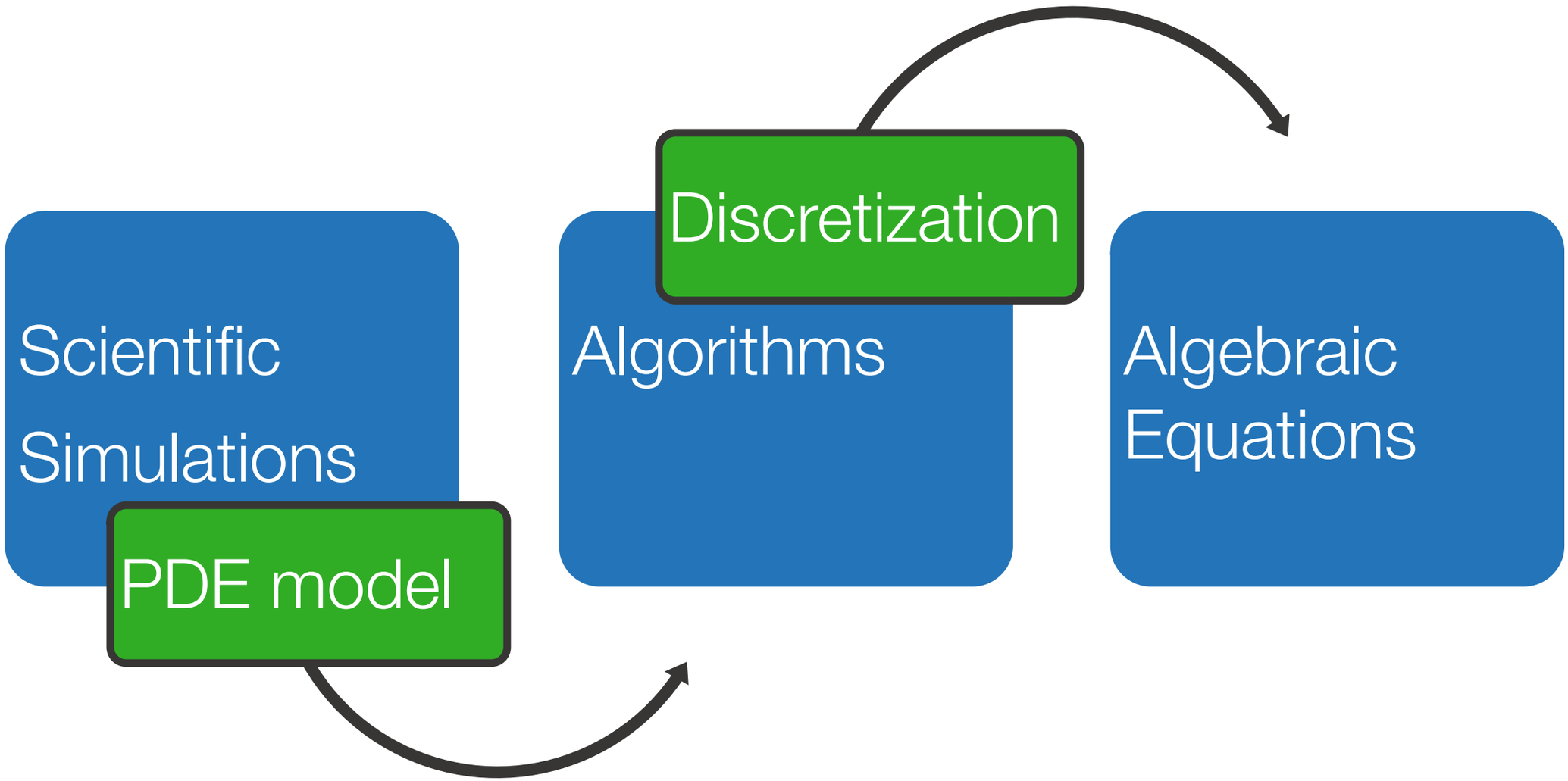
Scientific Simulations

PDE model

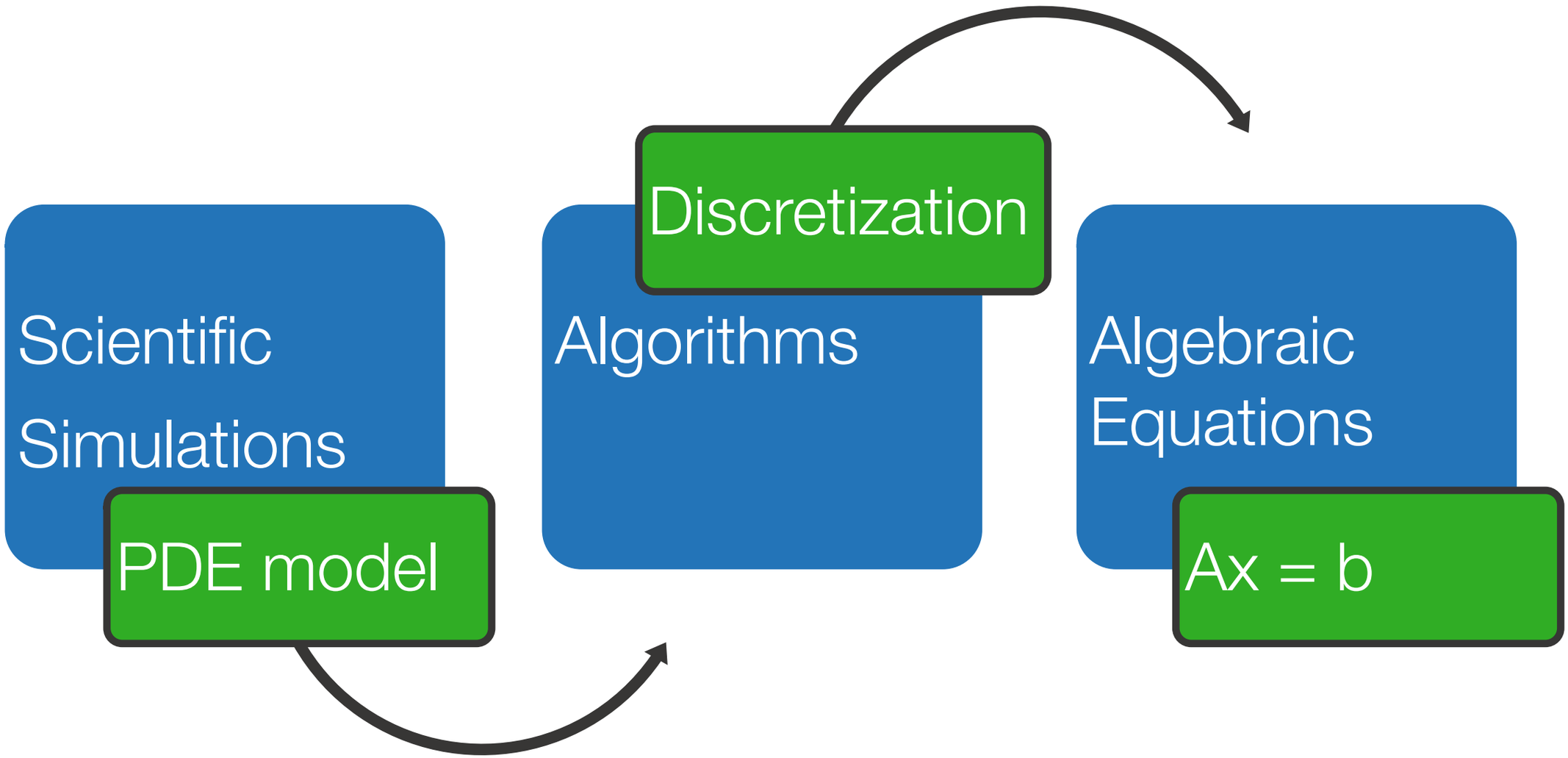
Algorithms

- Space discretization finite difference, finite element method, and finite volume method
- Time discretization
- Linearization / nonlinear solver

Models⇒Algorithms⇒Algebraic Systems



Models⇒Algorithms⇒Algebraic Systems



Model Problems

A fundamental problem in scientific computing:

Given $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$, How to solve $Ax = b$ efficiently?

Model Problems

A fundamental problem in scientific computing:

$$\text{Given } A \in \mathbb{R}^{N\times N} \text{ and } b \in \mathbb{R}^N, \text{How to solve } Ax = b \text{ efficiently?}$$

Solvers: Direct Method or Iterative Method

Model Problems

A fundamental problem in scientific computing:

$$\text{Given } A \in \mathbb{R}^{N \times N} \text{ and } b \in \mathbb{R}^{N}, \text{How to solve } Ax = b \text{ efficiently?}$$

Solvers: Direct Method or Iterative Method

- Direct Method: Gaussian Elimination ($O(N^3)$)

Model Problems

A fundamental problem in scientific computing:

Given $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$, How to solve $Ax = b$ efficiently?

Solvers: Direct Method or Iterative Method

- Direct Method: Gaussian Elimination ($O(N^3)$)

$N=10^6$	$N=10^7$	$N=10^8$	$N=10^9$
18.98s	5.27hr	220 days	602 Years

Model Problems

A fundamental problem in scientific computing:

Given $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$, How to solve $Ax = b$ efficiently?

Solvers: Direct Method or Iterative Method

- Direct Method: Gaussian Elimination ($O(N^3)$)

$N=10^6$	$N=10^7$	$N=10^8$	$N=10^9$
18.98s	5.27hr	220 days	602 Years

- Optimal solvers, e.g. Multigrid (MG) methods ($O(N)$)

Model Problems

A fundamental problem in scientific computing:

Given $A \in \mathbb{R}^{N \times N}$ and $b \in \mathbb{R}^N$, How to solve $Ax = b$ efficiently?

Solvers: Direct Method or Iterative Method

- Direct Method: Gaussian Elimination ($O(N^3)$)

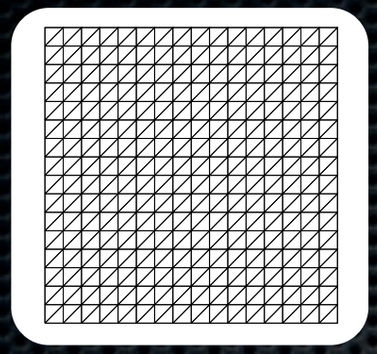
$N=10^6$	$N=10^7$	$N=10^8$	$N=10^9$
18.98s	5.27hr	220 days	602 Years

- Optimal solvers, e.g. Multigrid (MG) methods ($O(N)$)

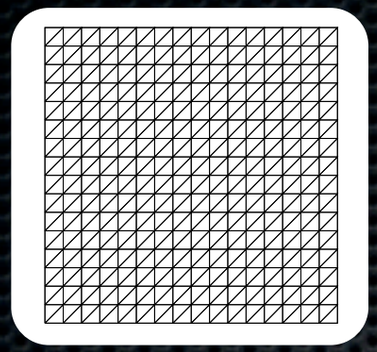
$N=10^6$	$N=10^7$	$N=10^8$	$N=10^9$
0.5s	5s	50s	500s

Geometric Multigrid Method

Geometric Multigrid Method

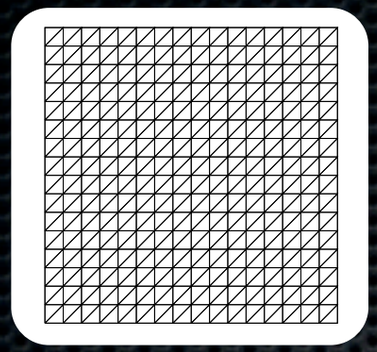


Geometric Multigrid Method



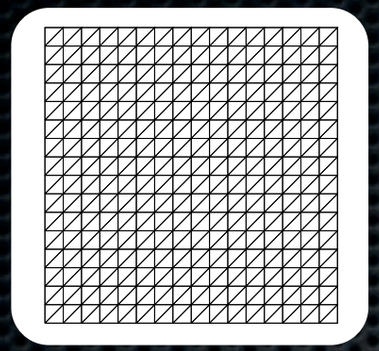
A_h

Geometric Multigrid Method



$A_h \Rightarrow (GS)_h$

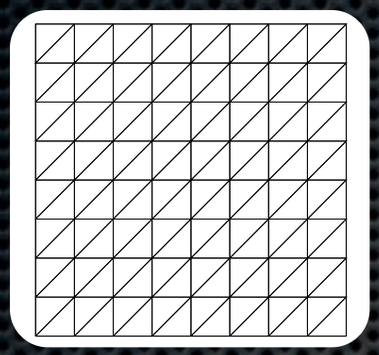
Geometric Multigrid Method



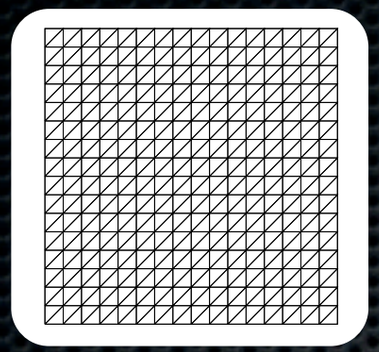
$A_h \Rightarrow (GS)_h$

+

$A_{2h} \Rightarrow (GS)_{2h}$

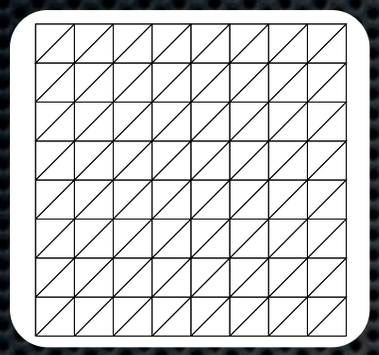


Geometric Multigrid Method



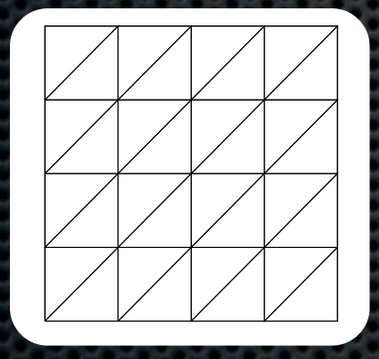
$A_h \Rightarrow (GS)_h$

+



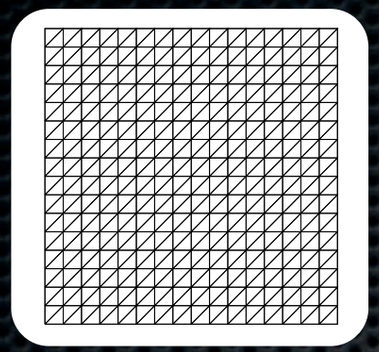
$A_{2h} \Rightarrow (GS)_{2h}$

+



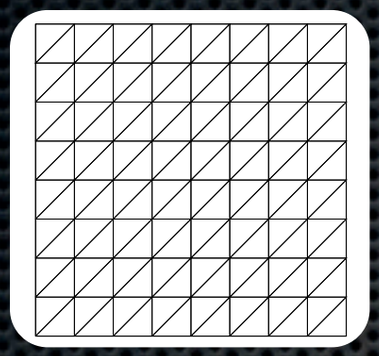
$A_{4h} \Rightarrow (GS)_{4h}$

Geometric Multigrid Method



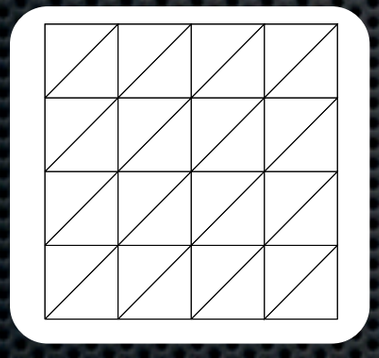
$A_h \Rightarrow (GS)_h$

+



$A_{2h} \Rightarrow (GS)_{2h}$

+

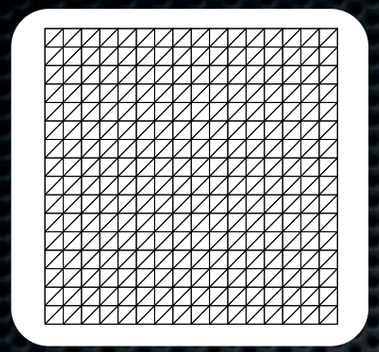


$A_{4h} \Rightarrow (GS)_{4h}$

+

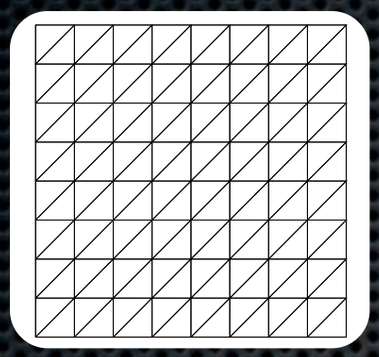
A8h $\Rightarrow$ …

Geometric Multigrid Method



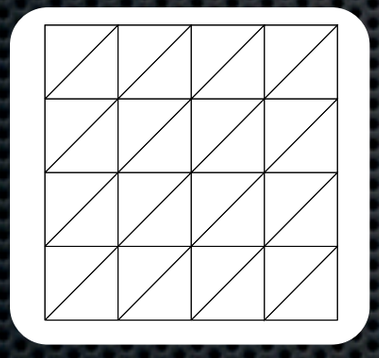
$A_h \Rightarrow (GS)_h$

+



$A_{2h} \Rightarrow (GS)_{2h}$

+



$A_{4h} \Rightarrow (GS)_{4h}$

+

$A8h \Rightarrow \ldots$

$O(N_h) + O(N_{2h}) + O(N_{4h}) + \ldots = O(N_h)$

Algebraic Multigrid Method (AMG)

Algebraic Multigrid Method (AMG)

Setup Phase:

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
- Construct coarse grid matrix: $A_C = RAP = P^TAP$.

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
- Construct coarse grid matrix: $A_C = RAP = P^TAP$.

Solve Phase:

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
- Construct coarse grid matrix: $A_C = RAP = P^TAP$.

Solve Phase:

- Presmoothing: Gauss-Seidel smoother / Jacobi smoother

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
- Construct coarse grid matrix: $A_c = RAP = P^TAP$.

Solve Phase:

- Presmoothing: Gauss-Seidel smoother / Jacobi smoother
- Coarse grid correction: recursively call MG on the coarse level

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
- Construct coarse grid matrix: $A_c = RAP = P^TAP$.

Solve Phase:

- Presmoothing: Gauss-Seidel smoother / Jacobi smoother
- Coarse grid correction: recursively call MG on the coarse level
- Postsmoothing: Gauss-Seidel smoother / Jacobi smoother

Algebraic Multigrid Method (AMG)

Setup Phase:

- Coarsening: Select coarse and find node / form aggregates
- Construct prolongation P and restriction R: standard interpolation / energy minimization / ... ($R = P^T$)
  - Construct coarse grid matrix: $A_c = RAP = P^TAP$.

Solve Phase:

- Presmoothing: Gauss-Seidel smoother / Jacobi smoother
- Coarse grid correction: recursively call MG on the coarse level
- Postsmoothing: Gauss-Seidel smoother / Jacobi smoother

$$I - B^{MG}A = (I - S^TA)(I - PB_c^{MG}RA)(I - SA)$$

Parallel AMG

Parallel AMG

CPU:

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)
- CUSP: smoothed aggregation AMG(Bell, Dalton, and Olsen, 2011)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)
- CUSP: smoothed aggregation AMG(Bell, Dalton, and Olsen, 2011)
- NVAMG (NVIDIA)

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)
- CUSP: smoothed aggregation AMG(Bell, Dalton, and Olsen, 2011)
- NVAMG (NVIDIA)

- Solve phase is ok (x6),setup phase is the **bottleneck (2x)**

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)
- CUSP: smoothed aggregation AMG(Bell, Dalton, and Olsen, 2011)
- NVAMG (NVIDIA)

- **Solve phase is ok (x6),**setup phase is the **bottleneck (2x)**
- We need the setup phase, because we don't have the hierarchical grid to do the MG.

Parallel AMG

CPU:

- BoomerAMG(HYPRE): Classical AMG method(LLNL)
- ML: smoothed aggregation AMG method(Sandia)
- AGMG: unsmoothed aggregation AMG(Y. Notay)

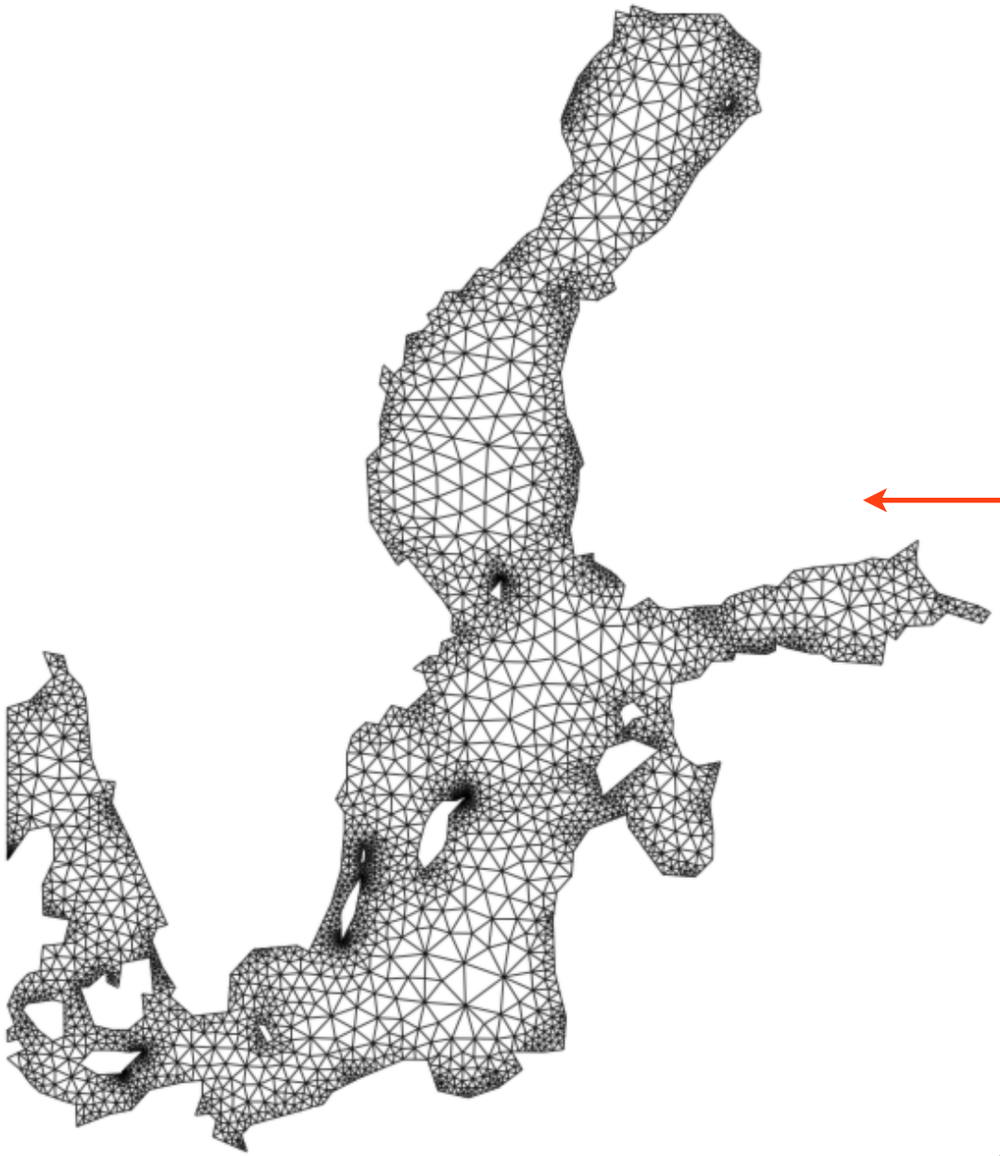
GPU:

- Solve phase (Goddeke et al, 2008;Haase et al, 2010)
- Parallel Toolbox: classical AMG(Liebmann,2011)
- CUSP: smoothed aggregation AMG(Bell, Dalton, and Olsen, 2011)
- NVAMG (NVIDIA)

- **Solve phase is ok (x6),**setup phase is the **bottleneck (2x)**
- We need the setup phase, because we don't have the hierarchical grid to do the MG.
- However, usually we can access the mesh of the problem

MG for Unstructured Grid

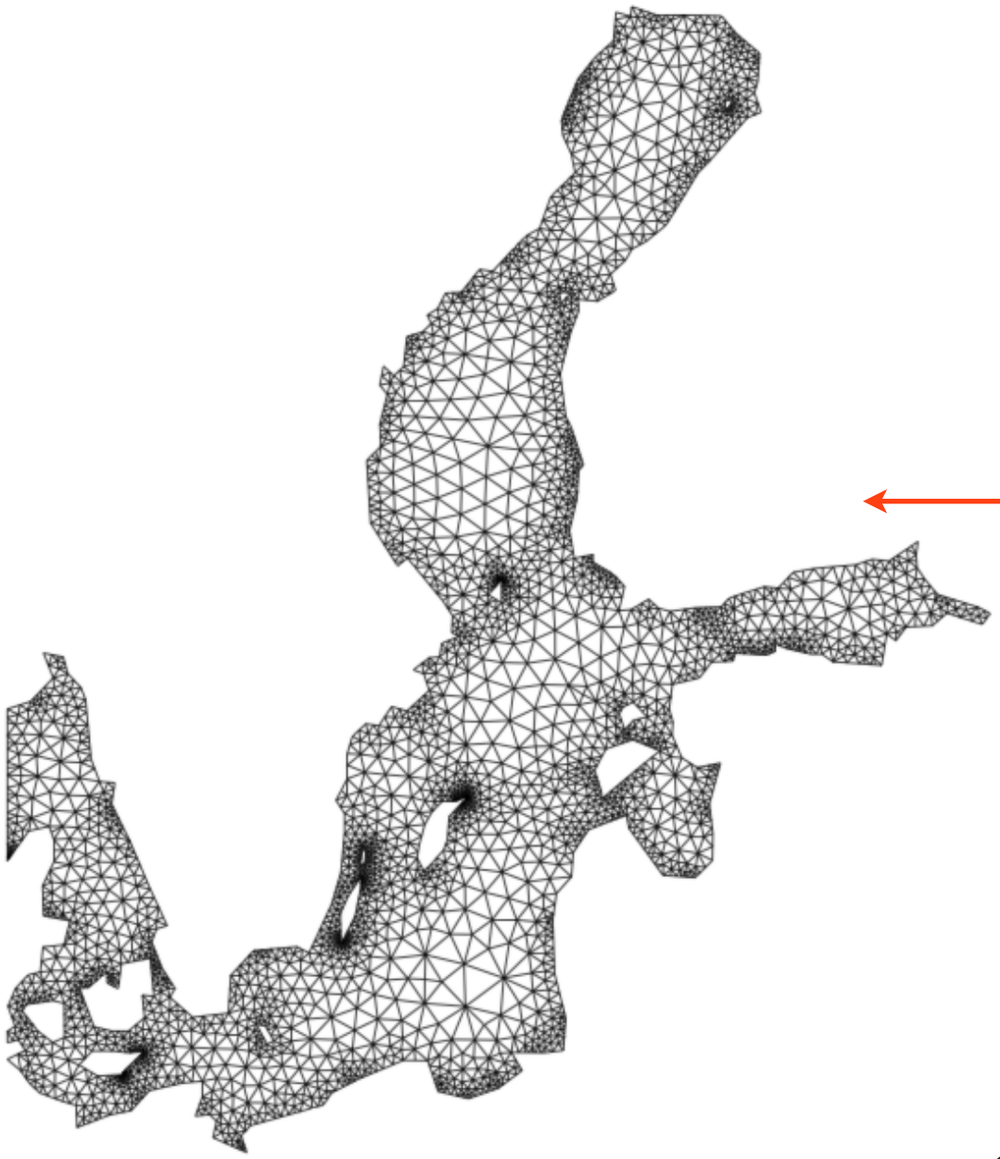
MG for Unstructured Grid



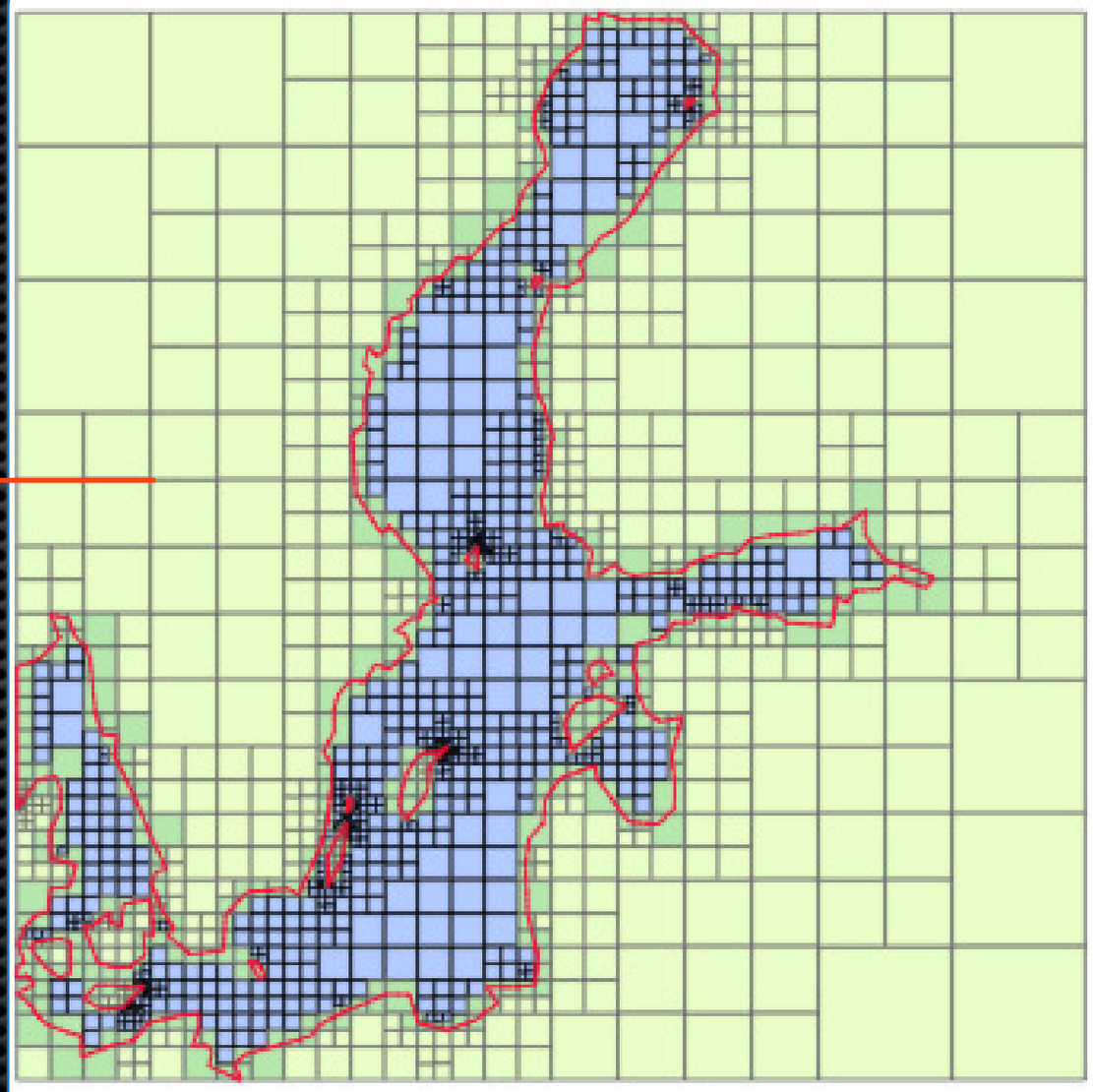
Grid_baltic

Baltic Lake

MG for Unstructured Grid

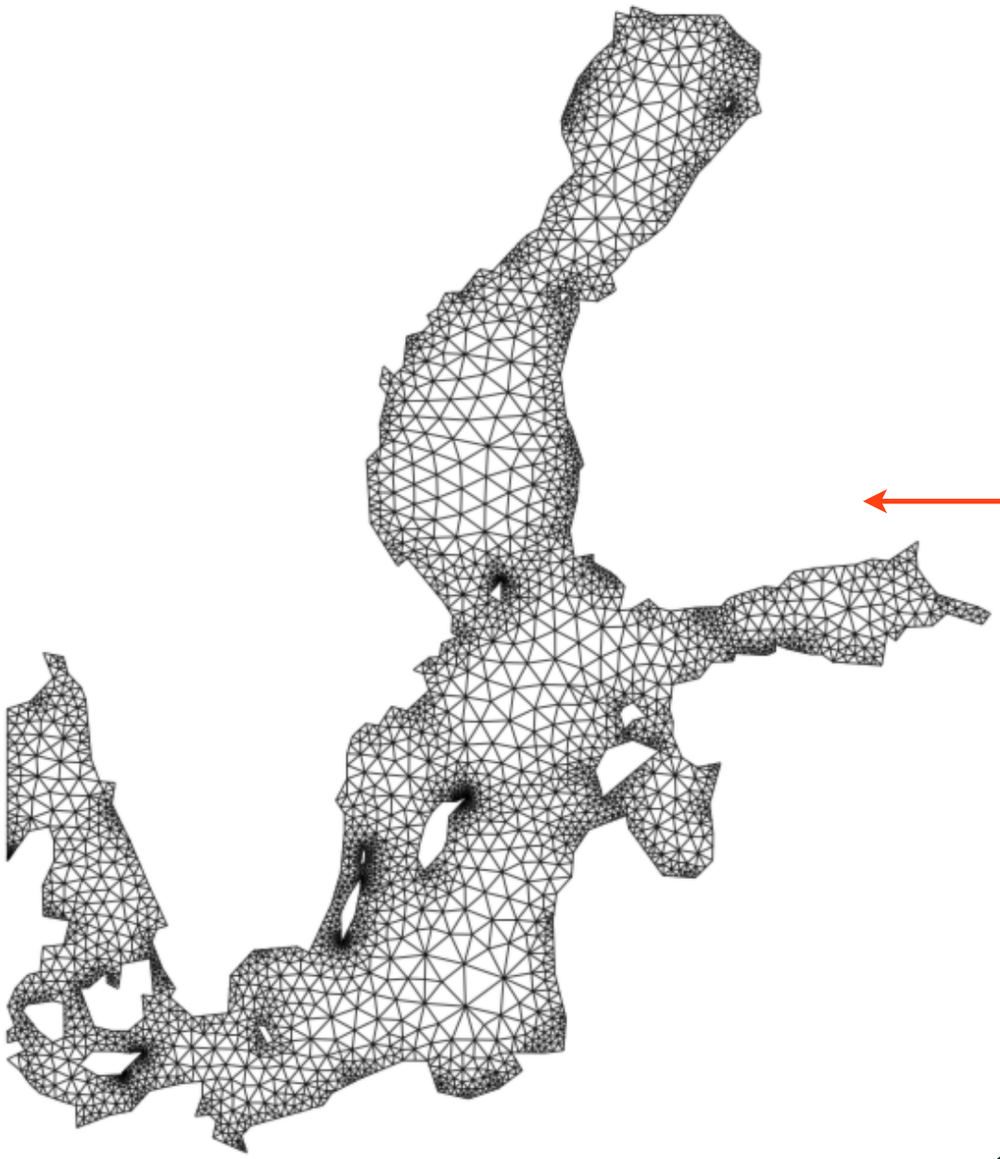


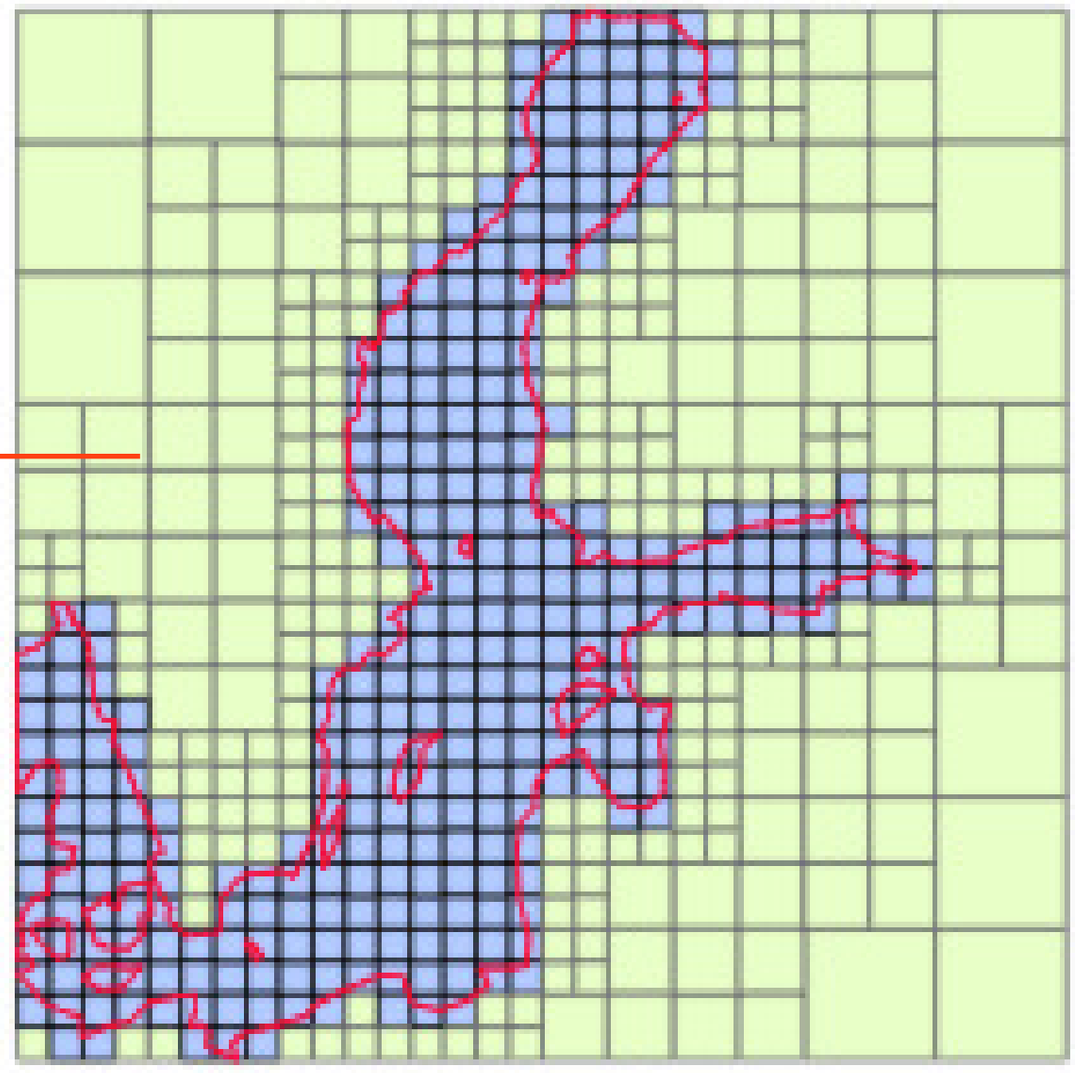
Grid_baltic



Baltic Lake

MG for Unstructured Grid

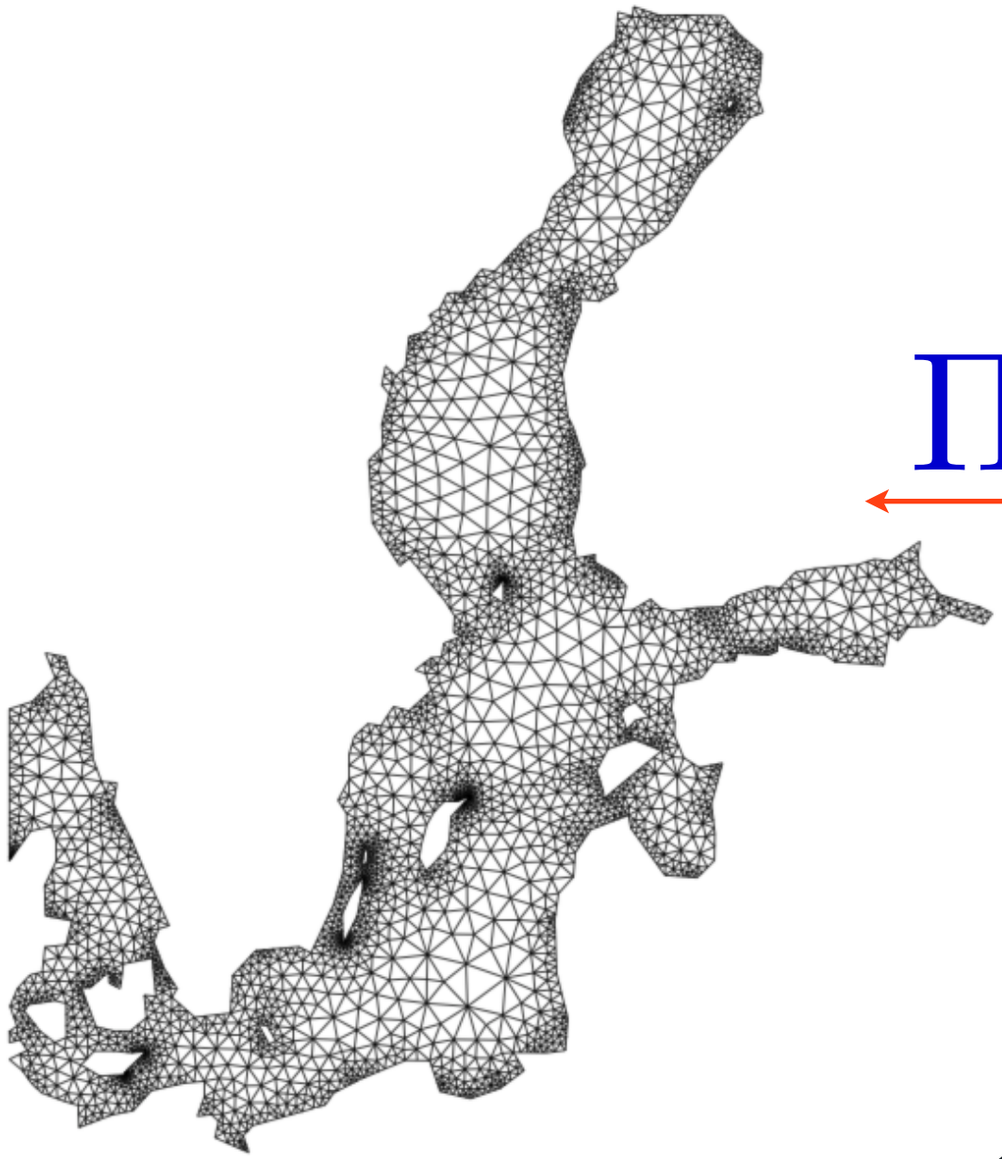




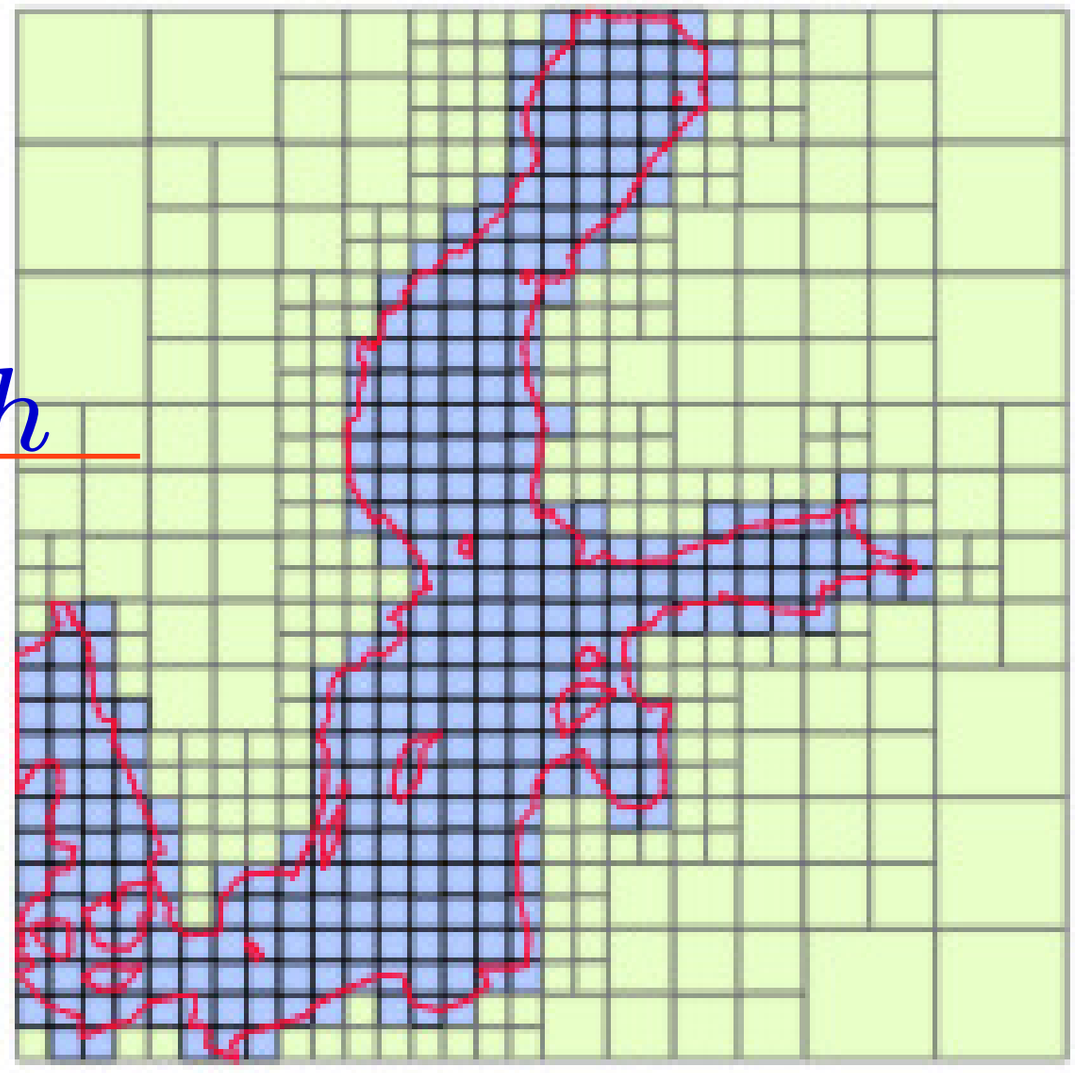
Grid_baltic

Baltic Lake

MG for Unstructured Grid



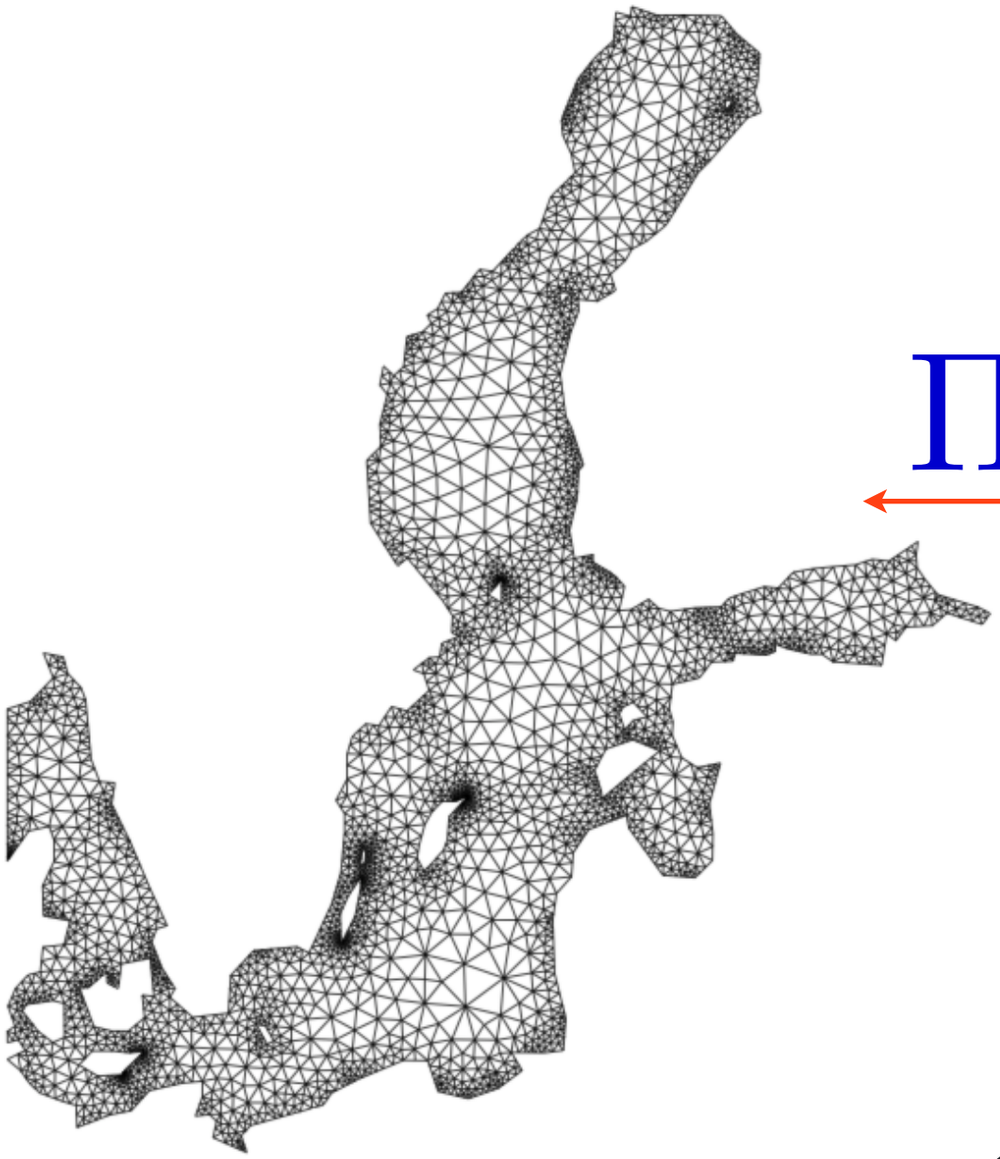
Π_h



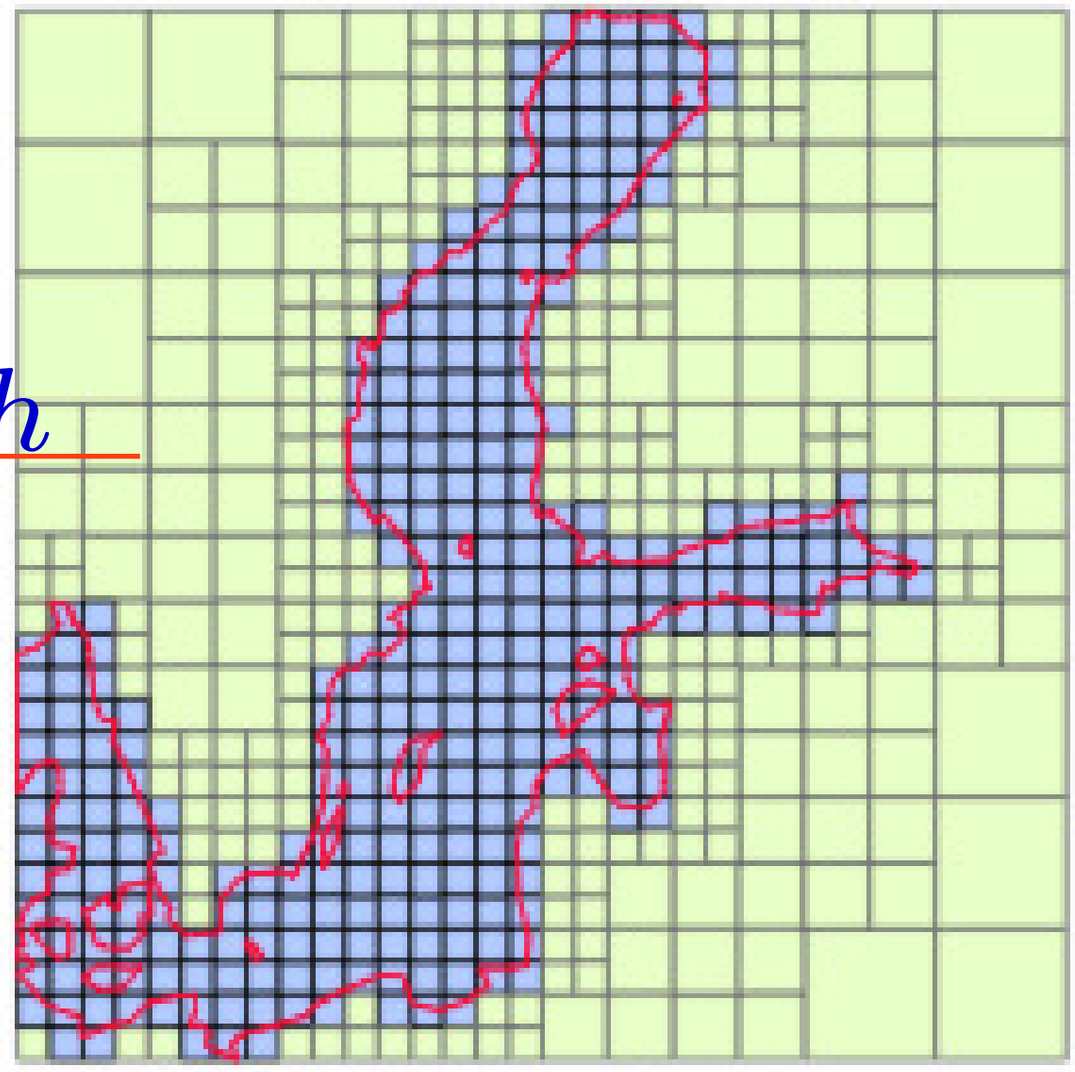
Grid_baltic

Baltic Lake

$$B_h = S_h + \Pi_h \bar{B}_h \Pi_h^T$$



Π_h



Grid_baltic

Baltic Lake

FASP Preconditioner

Problems	Partial Differential Equations	#P
Poisson	$-\Delta u = f$	1
Reaction-Diffusion	$-\nabla \cdot (\mu(x)\nabla u) + c(x)u = f$	1
H(curl)	$\nabla \times \nabla \times u + u = f$	4
H(div)	$-\nabla\nabla \cdot u + u = f$	3–6
Maxwell	$B_t + \nabla \times E = 0,\ \nabla \cdot D = \rho,$ $D_t - \nabla \times H = -J,\ \nabla \cdot B = 0$	4
Darcy	$-\nabla \cdot u = f,\ u = \mu\nabla p$	3–6
Stokes	$-\Delta u + \nabla p = f,\ \nabla \cdot u = 0$	3
Brinkman	$-\nabla \cdot (\mu(x)\nabla u) + c(x)u + \nabla p = f,\ \nabla \cdot u = g$	3
Linear Elasticity	$-\Delta u - \frac{\nu}{1-2\nu}\nabla\nabla \cdot u = f$	3
Biharmonic (plate)	$\Delta^2 u = f$	2
Reissner-Mindlin	$-\operatorname{div}(\mathcal{C}D(\phi)) = \lambda t^{-2}(\phi - \nabla\omega) = 0,$ $\lambda t^{-2}(-\Delta\omega + \operatorname{div}\phi) = g$	4
R–C–D	$u_t + v(x) \cdot \nabla u - \nabla \cdot (\mu(x)\nabla u) + c(x)u = f$	1
Navier–Stokes	$u_t + (u \cdot \nabla)u - \mu\Delta u + \nabla p = f,\ \nabla \cdot u = 0$	4
Johnson-Seglman	$u_t + (u \cdot \nabla)u - \mu_s\Delta u + \nabla p = \nabla \cdot \tau,\ \nabla \cdot u = 0,$ $\tau + \mathrm{Wi}[u_t + (u \cdot \nabla)u - R\tau - \tau R^T] = \frac{1}{2}\mu_p \mathcal{D}u$ $R = \frac{a+1}{2}\nabla u + \frac{a-1}{2}\nabla u^T$	3
MHD	$u_t + (u \cdot \nabla)u + \nabla p = \mu\Delta u + \mu_0 J \times B,\ \nabla \cdot u = 0,$ $B_t + \nabla \times E = 0,\ \nabla \times B = \mu_0 J,$ $\nabla \cdot B = 0,\ E + u \times B = \sigma_0^{-1} J$	7
Black-oil model		

Unsmoothed Aggregation AMG

(Braess, 1995; Kim, Xu, and Ludmil 2003)

Unsmoothed Aggregation AMG

(Braess, 1995; Kim, Xu, and Ludmil 2003)

- Aggregation(coarsening): Find a non-overlapping partition $\{G_j\}$, which are referred as aggregates

Unsmoothed Aggregation AMG

(Braess, 1995; Kim, Xu, and Ludmil 2003)

- Aggregation(coarsening): Find a non-overlapping partition $\{G_j\}$, which are referred as aggregates
- Prolongation and restriction: Boolean Matrix

$$P_{ij} = \begin{cases} 1 & \text{if } i \in G_{ij} \\ 0 & \text{otherwise} \end{cases}$$

Unsmoothed Aggregation AMG

(Braess, 1995; Kim, Xu, and Ludmil 2003)

- Aggregation(coarsening): Find a non-overlapping partition $\{G_j\}$, which are referred as aggregates
- Prolongation and restriction: Boolean Matrix

$$P_{ij} = \begin{cases} 1 & \text{if } i \in G_{ij} \\ 0 & \text{otherwise} \end{cases}$$

- Coarse grid matrix: $A_c = P^t A P$ can be computed simple summations

$$(A_c)_{kl} = \sum_{i \in G_i} \sum_{j \in G_j} a_{ij}$$

Enhanced MG Cycles

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid ⇒Optimal multigrid

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid ⇒Optimal multigrid

- AMLI-cycle

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid ⇒Optimal multigrid
  - AMLI-cycle
  - K-cycle: use preconditioned Krylov iterative methods in grid correction step

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid ⇒Optimal multigrid
  - AMLI-cycle
  - K-cycle: use preconditioned Krylov iterative methods in grid correction step

Theorem(X. Hu, Vassilevski & Xu, SINUM, 2013)

Enhanced MG Cycles

A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid ⇒Optimal multigrid
  - AMLI-cycle
  - K-cycle: use preconditioned Krylov iterative methods in grid correction step

Theorem(X. Hu, Vassilevski & Xu, SINUM, 2013)

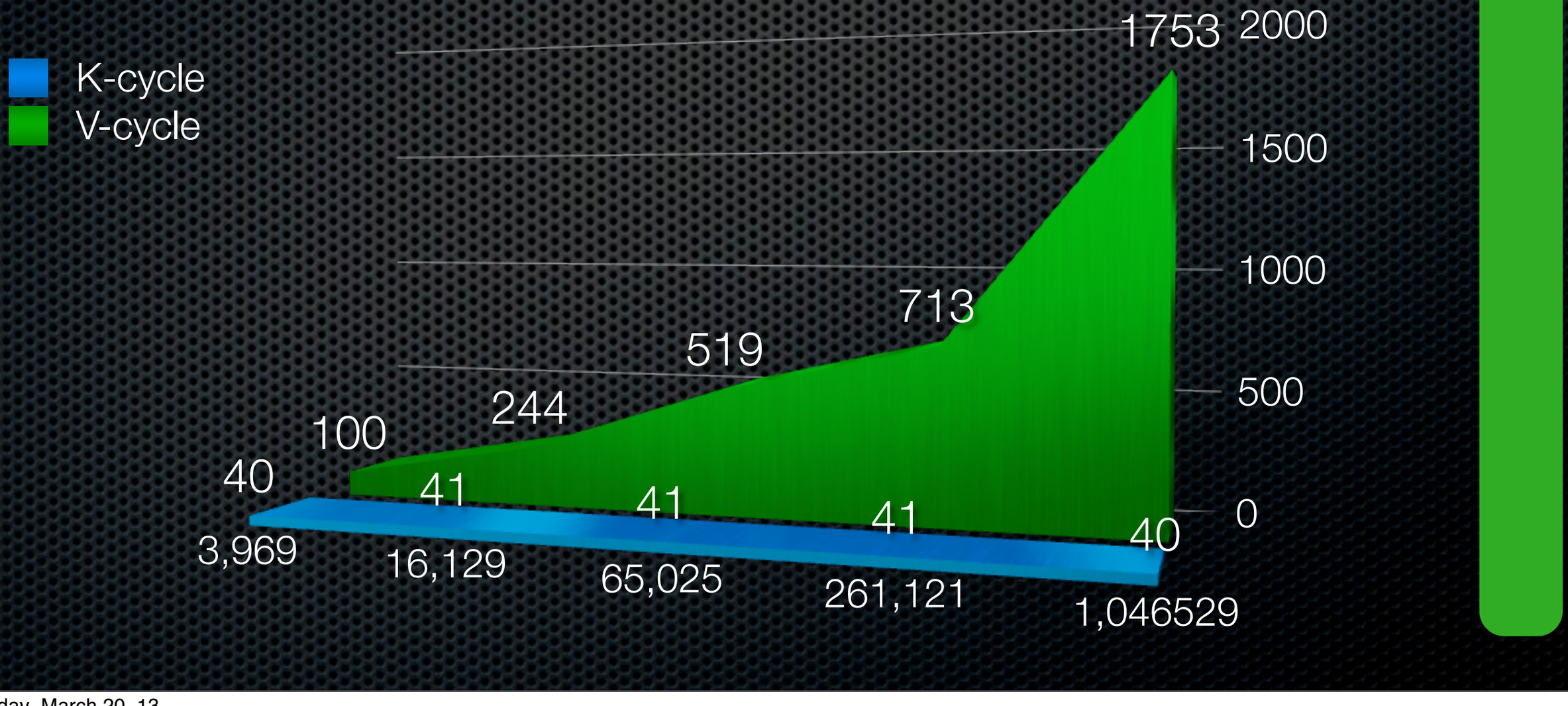
If the two-grid method converges uniformly, then the K-cyle method converges uniformly

Example

$$\begin{cases} -\Delta u = f, & \text{in } \Omega \\ u = 0, & \text{on } \partial\Omega \end{cases}$$

Example

$$\begin{cases} -\Delta u = f, & \text{in } \Omega \\ u = 0, & \text{on } \partial\Omega \end{cases}$$



Auxiliary Grid Based UA-AMG

Auxiliary Grid Based UA-AMG

- Construct an auxiliary structured grid

Auxiliary Grid Based UA-AMG

- Construct an auxiliary structured grid
- Use quad-tree to manage the structure grid

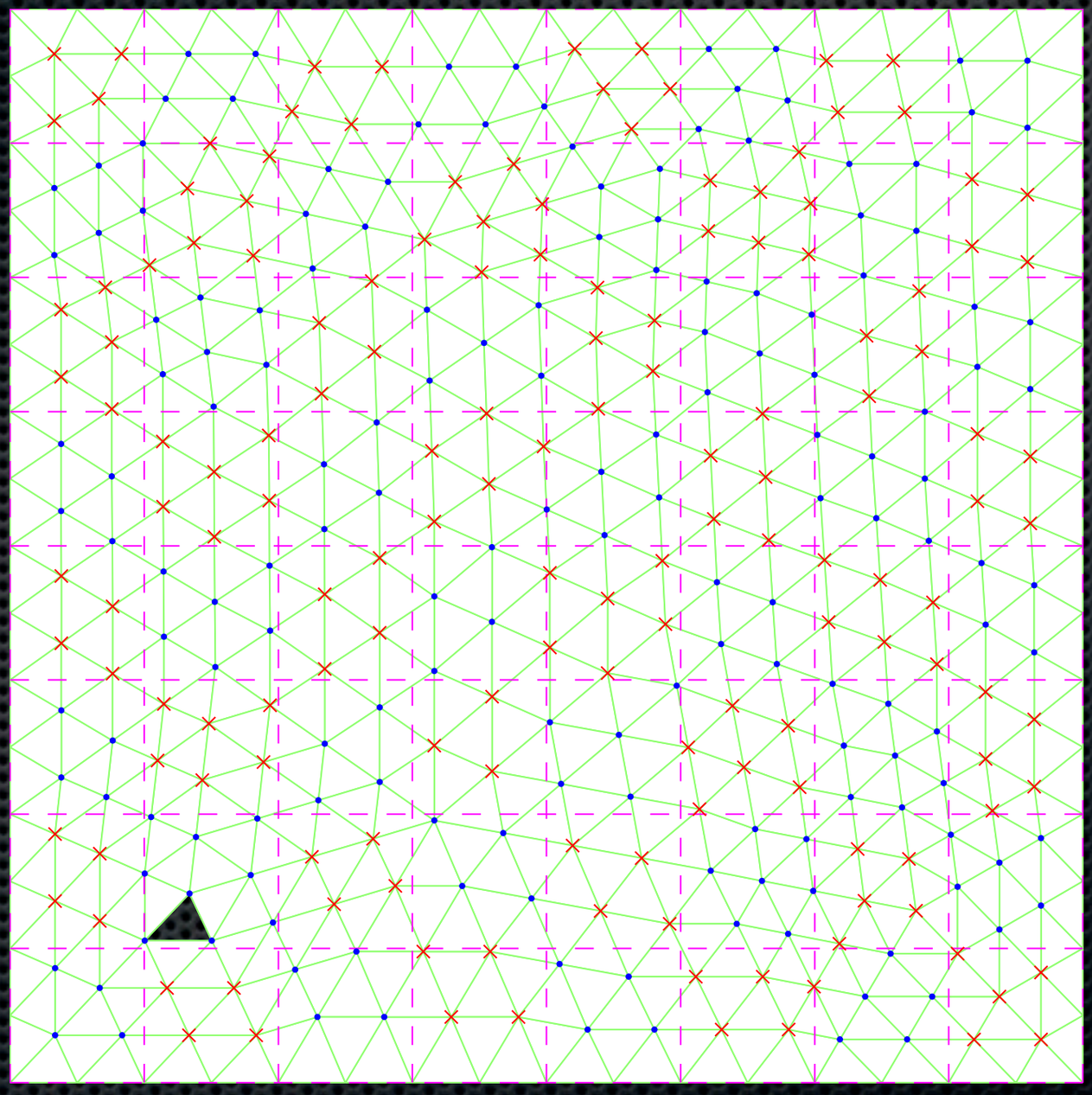
Auxiliary Grid Based UA-AMG

- Construct an auxiliary structured grid
- Use quad-tree to manage the structure grid
- Construct the hierarchical structure using the auxiliary grid

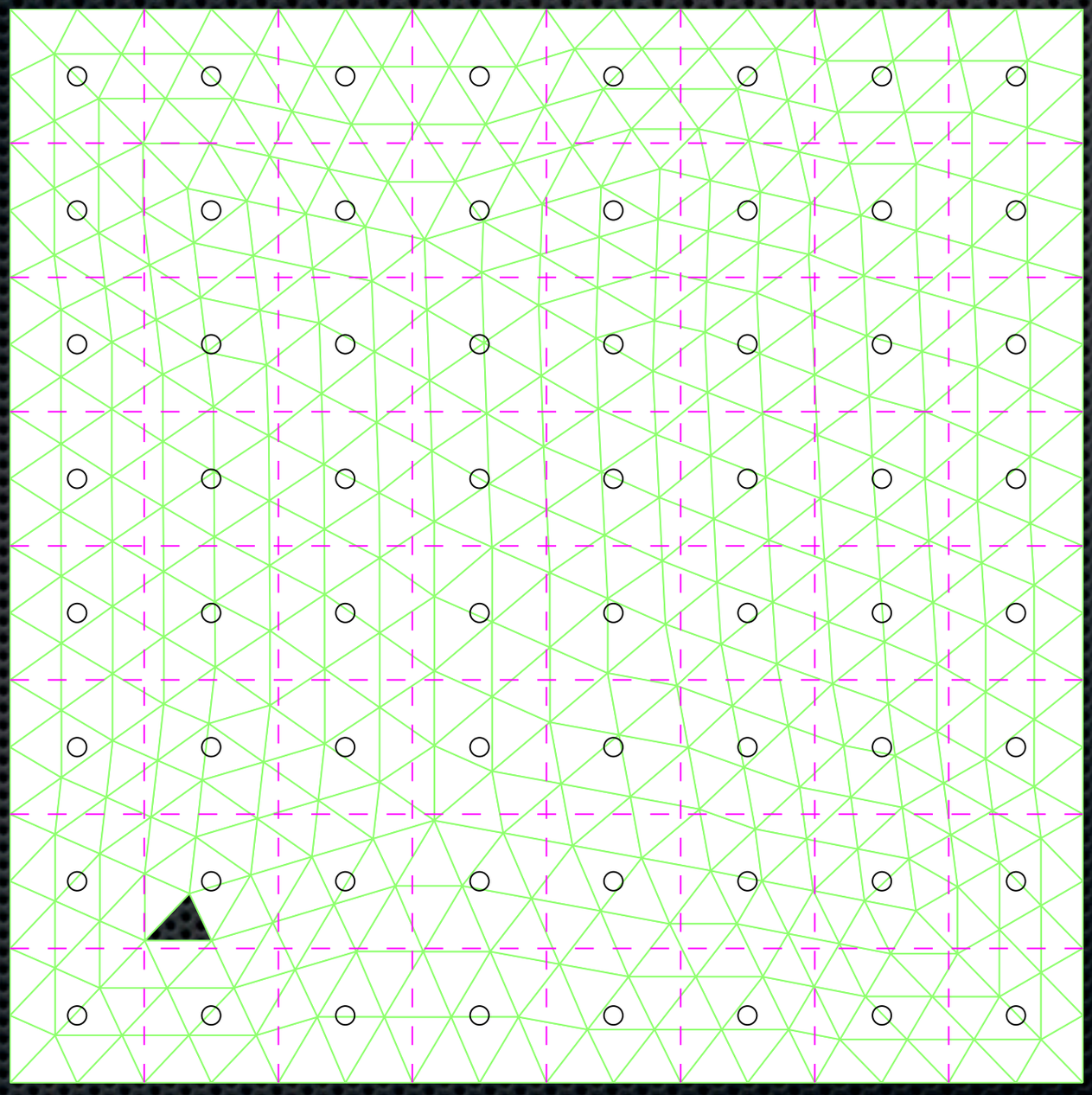
Auxiliary Grid Based UA-AMG

- Construct an auxiliary structured grid
- Use quad-tree to manage the structure grid
- Construct the hierarchical structure using the auxiliary grid
- Apply the UA-AMG method

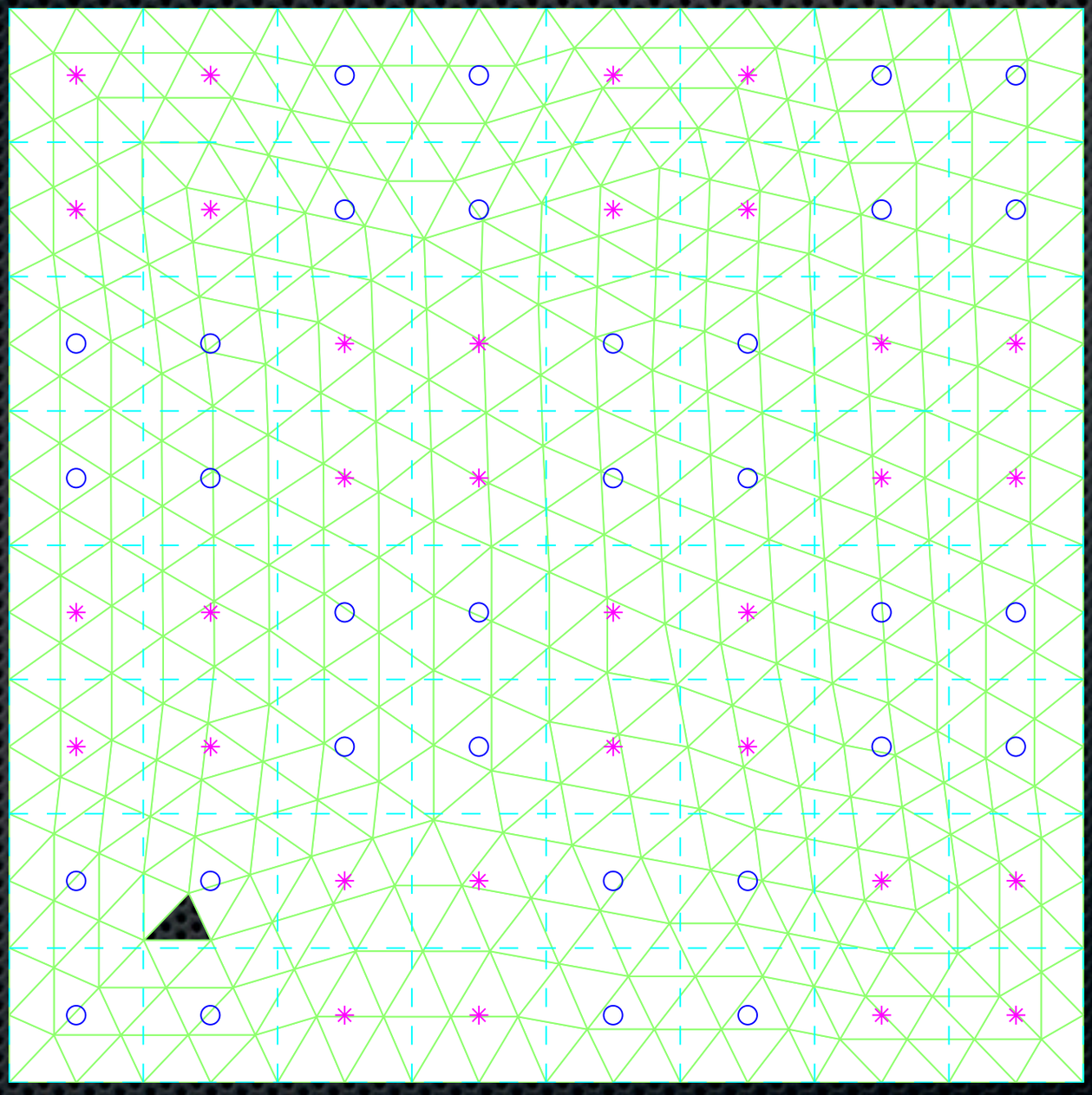
Aggregation



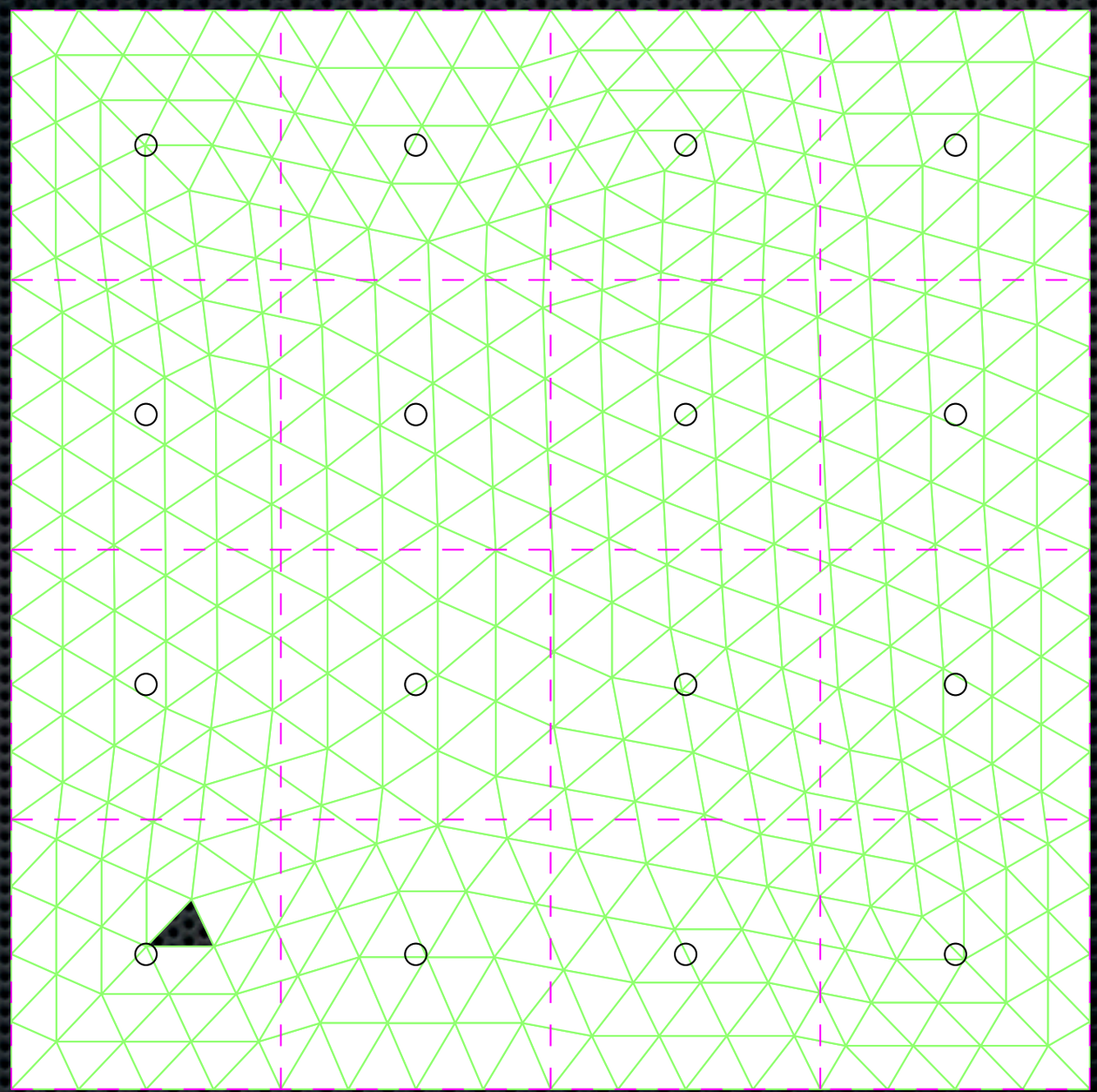
Aggregation



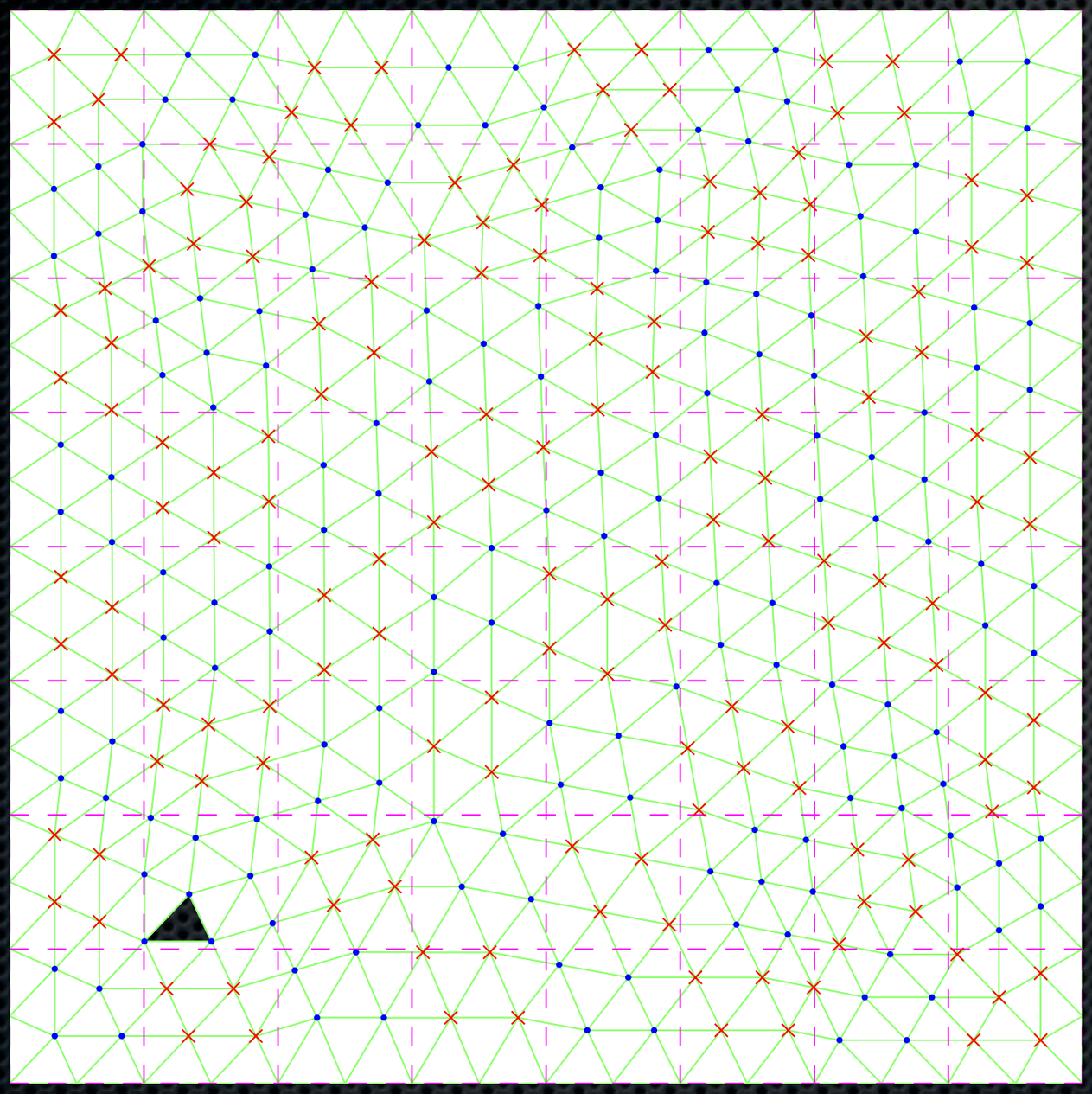
Aggregation

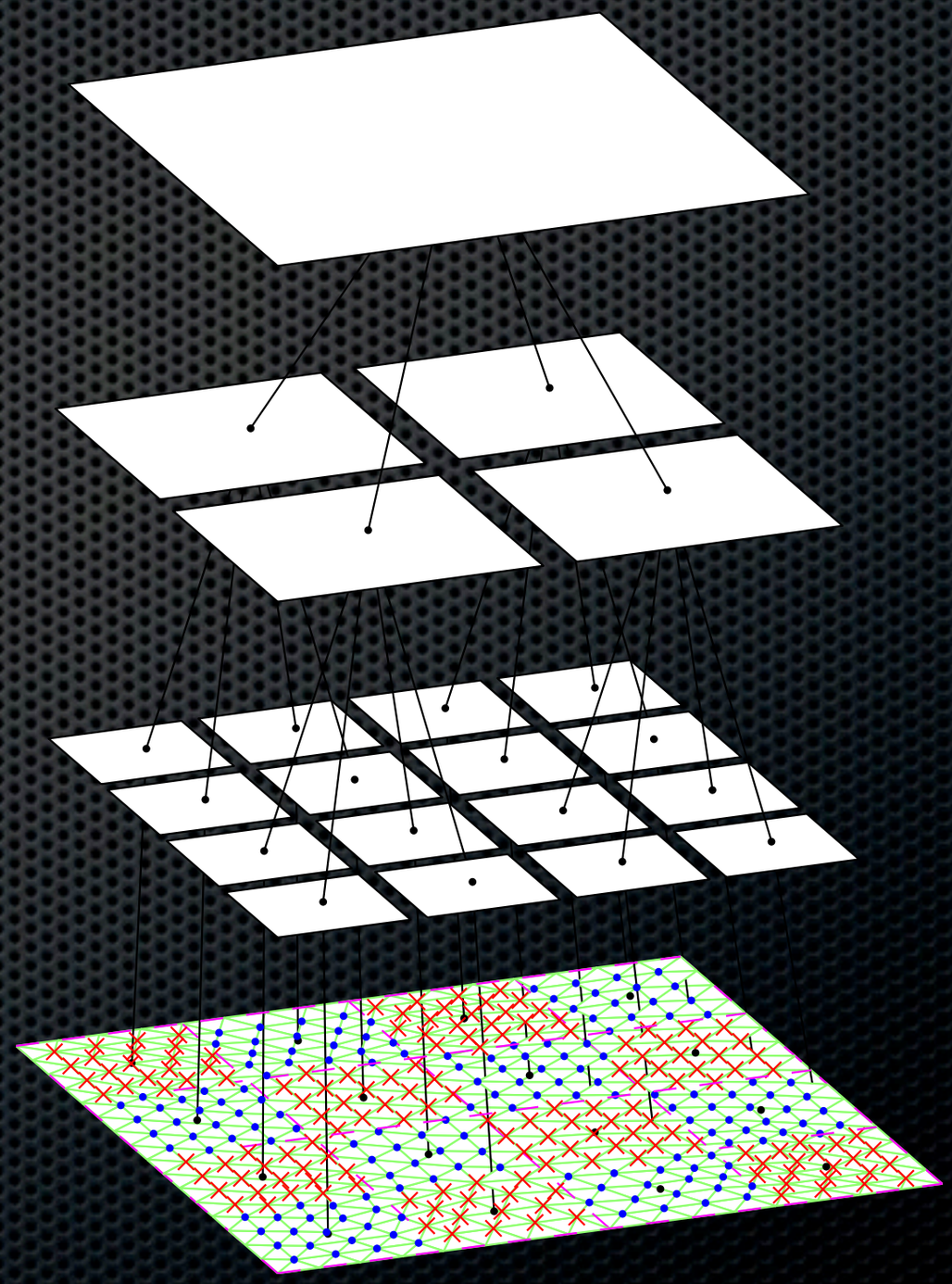


Aggregation



Aggregation





Prolongation & Restriction

Prolongation & Restriction

Prolongation

Prolongation & Restriction

Prolongation

- Value of a leaf node
  = Value of its parent node

Prolongation & Restriction

Prolongation

- Value of a leaf node
  = Value of its parent node

For $i \in G_j$,

$$v = Pv^c \Rightarrow v_i = v^c_j,$$

Prolongation & Restriction

Prolongation

- Value of a leaf node
  = Value of its parent node

For $i \in G_j$,

$$v = Pv^c \Rightarrow v_i = v^c_j,$$

Restriction

Prolongation & Restriction

Prolongation

- Value of a leaf node
  = Value of its parent node

For $i \in G_j$,

$v = Pv^c \Rightarrow v_i = v^c_j$,

Restriction

- value of parent node
  = sum of its children nodes

Prolongation & Restriction

Prolongation

- Value of a leaf node
  = Value of its parent node

For $i \in G_j$,

$$v = Pv^c \Rightarrow v_i = v^c_j,$$

Restriction

- value of parent node
  = sum of its children nodes

$$v^c = P^T v$$

$$\Rightarrow v^c_i = \sum_{j \in G_i} v_j,$$

Main Components on GPU

Main Components on GPU

- Sparse matrix storage format

Main Components on GPU

- Sparse matrix storage format
- Setup Phase

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

- Form aggregates

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

- Form aggregates
- Construct prolongation & restriction

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

- Form aggregates
- Construct prolongation & restriction
- Construct coarse grid matrix

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

- Form aggregates
- Construct prolongation & restriction
- Construct coarse grid matrix

Solve Phase

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

- Form aggregates
- Construct prolongation & restriction
- Construct coarse grid matrix

Solve Phase

- Sparse matrix and vector multiplications (SpMV)

Main Components on GPU

- Sparse matrix storage format
- Setup Phase
- Solve Phase

Setup Phase

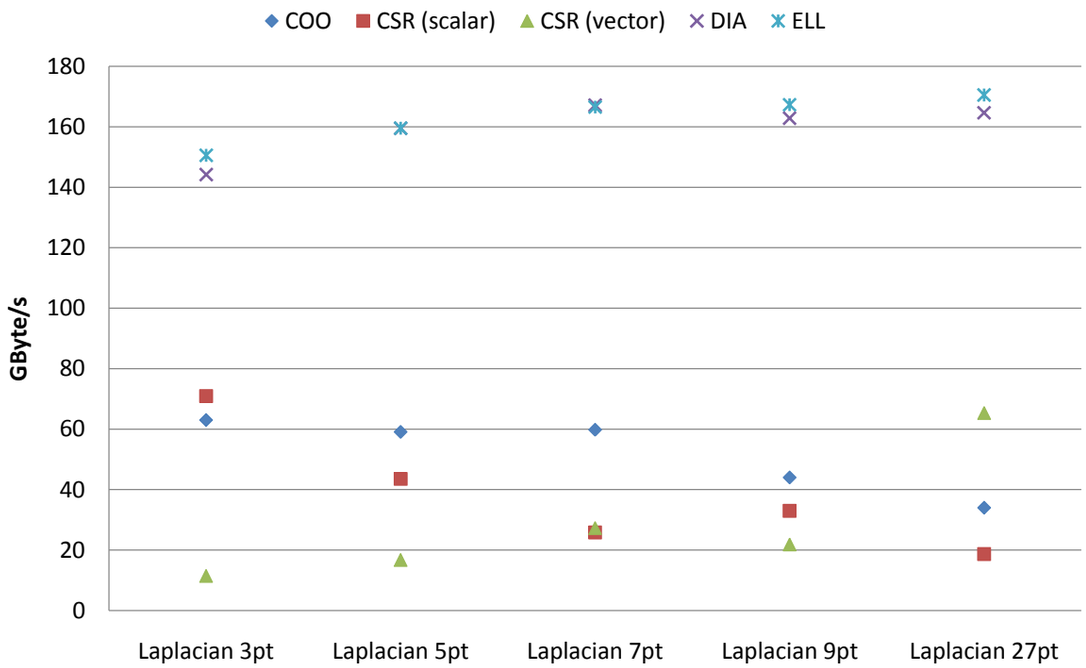
- Form aggregates
- Construct prolongation & restriction
- Construct coarse grid matrix

Solve Phase

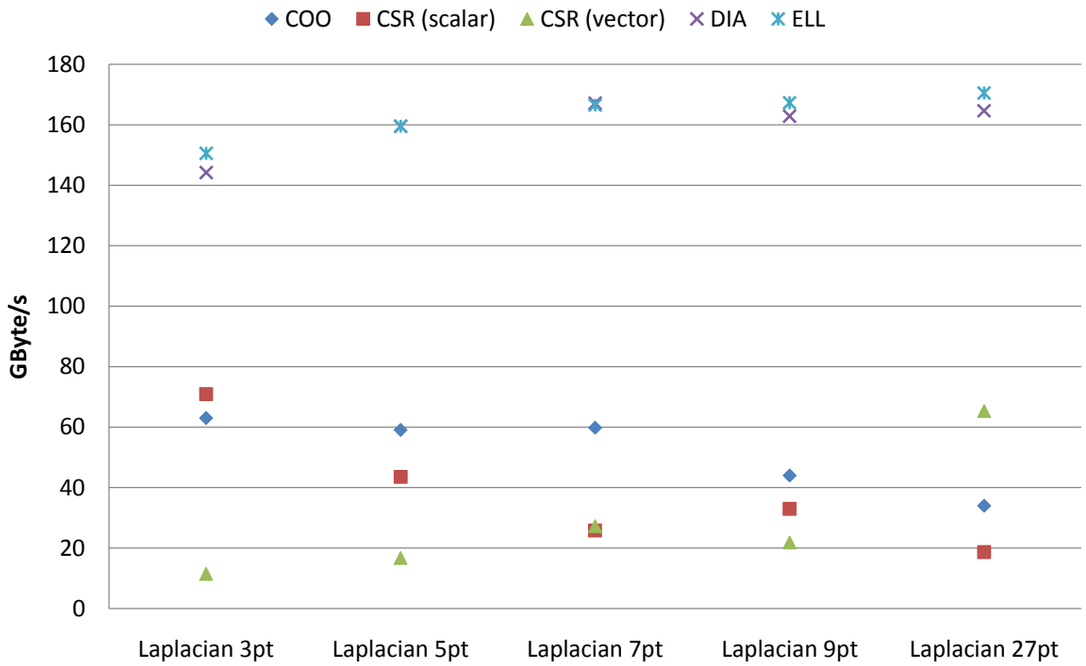
- Sparse matrix and vector multiplications (SpMV)
- Parallel smoothers

Performance(N. Bell and M.Garland, 2009)

Performance(N. Bell and M.Garland, 2009)

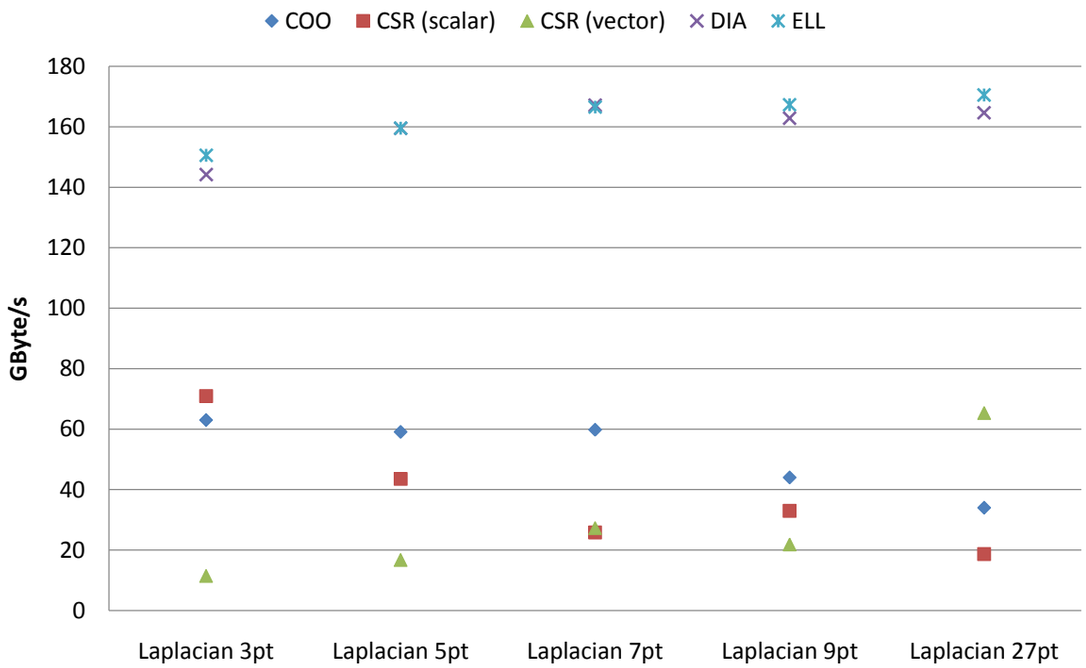


Performance(N. Bell and M.Garland, 2009)



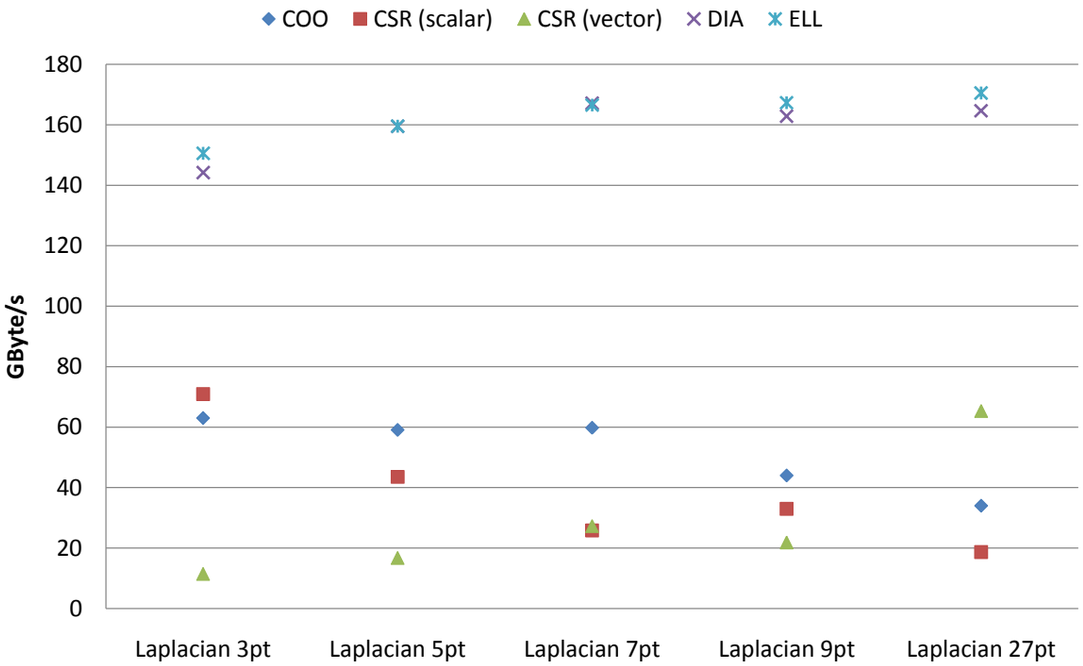
- CSR is commonly used but has low performance

Performance (N. Bell and M.Garland, 2009)



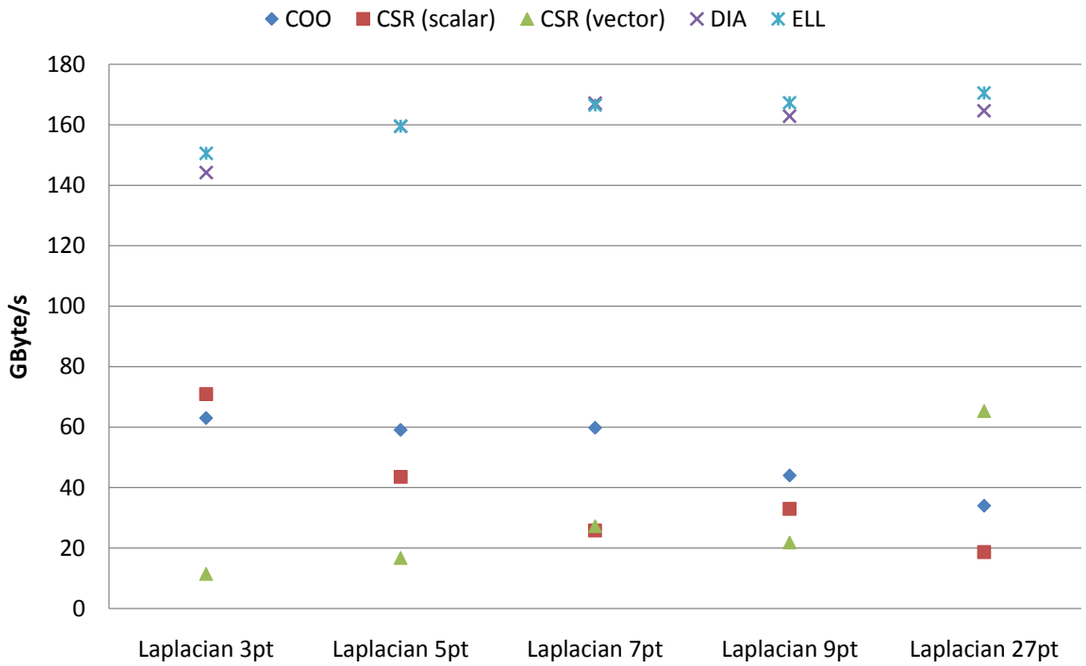
- CSR is commonly used but has low performance
- COO requires more memory and slow

Performance (N. Bell and M.Garland, 2009)



- CSR is commonly used but has low performance
- COO requires more memory and slow
- DIA has good performance, but it requires special sparsity pattern

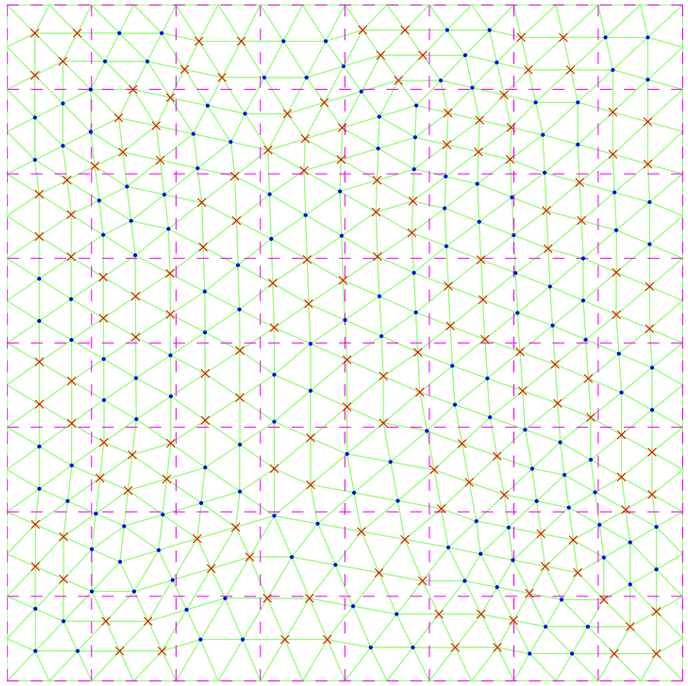
Performance(N. Bell and M.Garland, 2009)

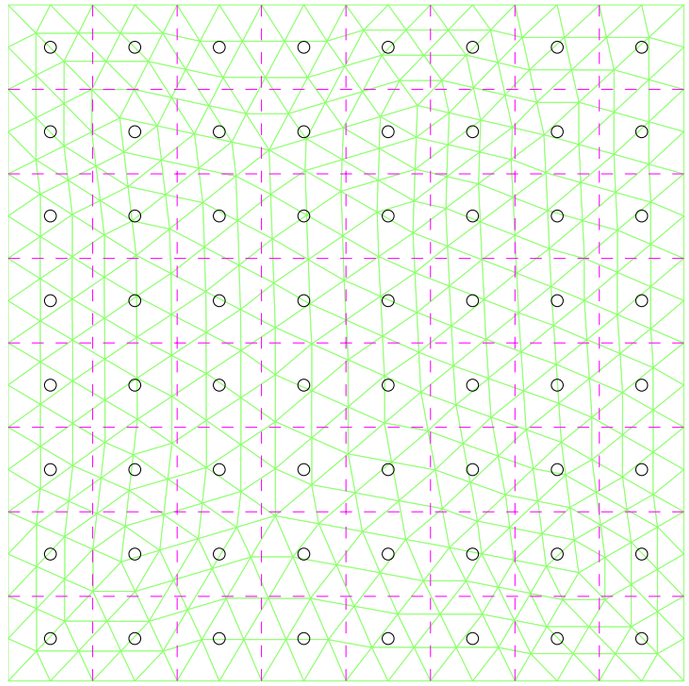


- CSR is commonly used but has low performance
- COO requires more memory and slow
- DIA has good performance, but it requires special sparsity pattern
- **ELL: our choice**

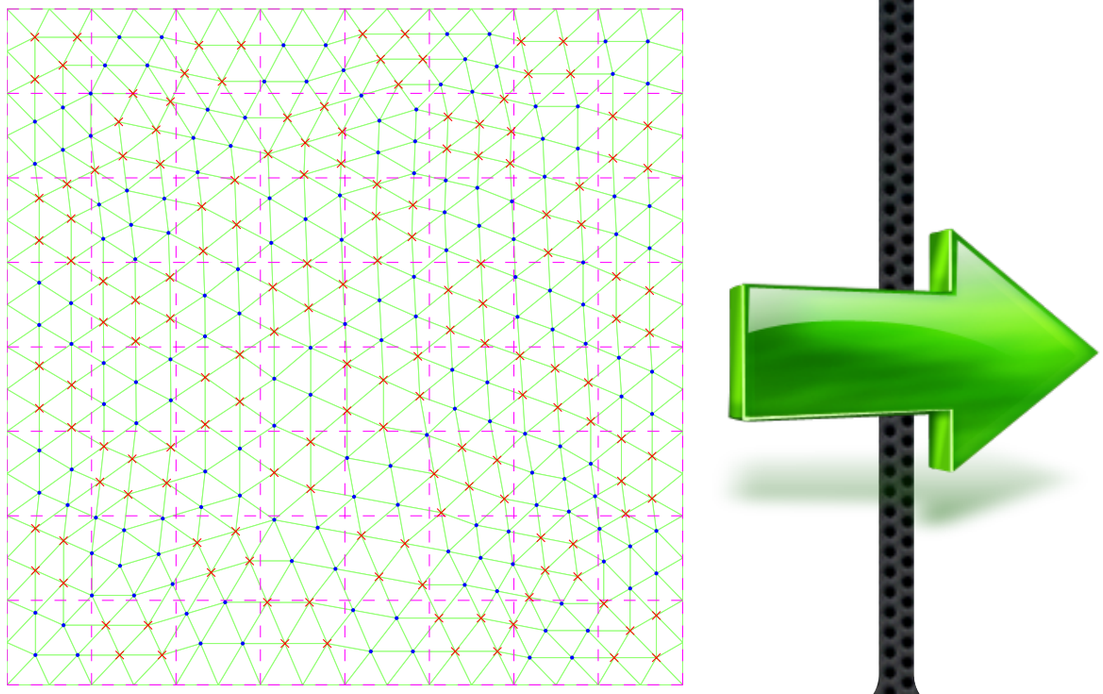
Parallel Aggregation on the finest level

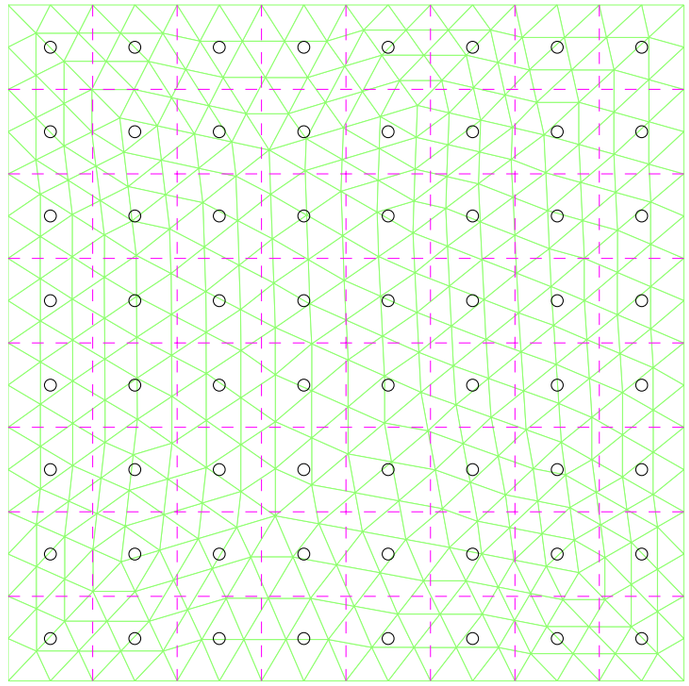
Parallel Aggregation on the finest level





Parallel Aggregation on the finest level

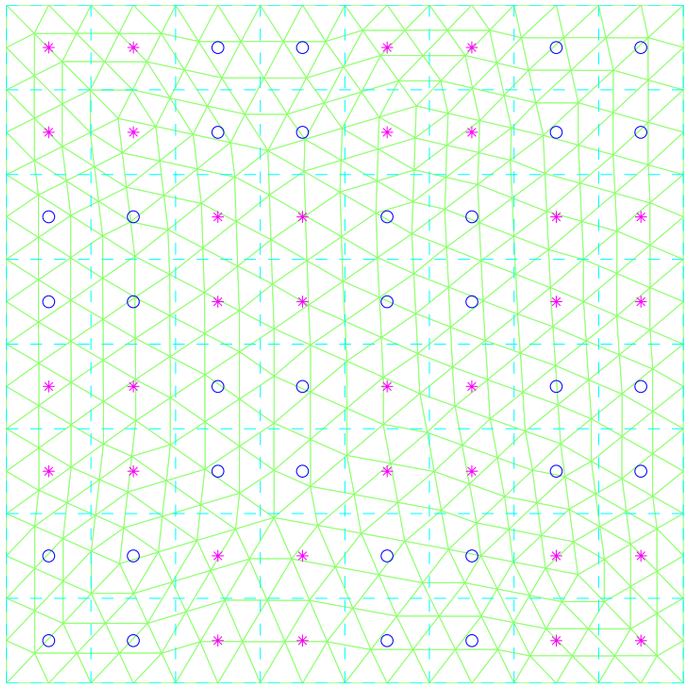


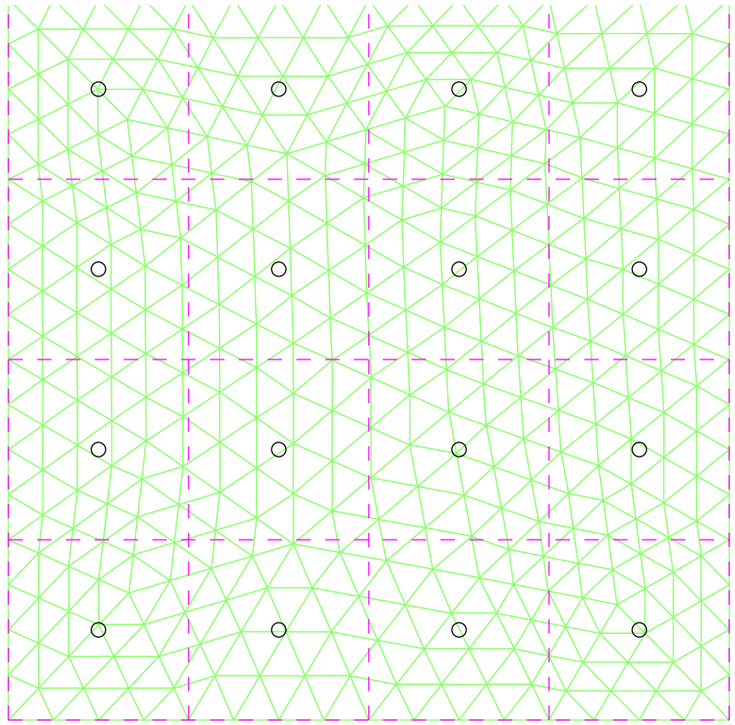


```
__global__ void aggregation fine(int* aggregation, double* x, double* y, double xmax, double ymax, double xmin, double ymin, int L)
{

    const unsigned int idx = blockIdx.x*blockDim.x+threadIdx.x; % get thread ID
    const unsigned int powL = (int)pow(2.0, L);% check x coordinate
    const unsigned int xIdx = (int)((x - xmin)/(xmax-xmin)*powL); % check y coordinate
    const unsigned int yIdx = (int)((y - ymin)/(ymax-ymin)*powL); % label the aggregate
    aggregation[idx] = yIdx*powL + xIdx;

}
```

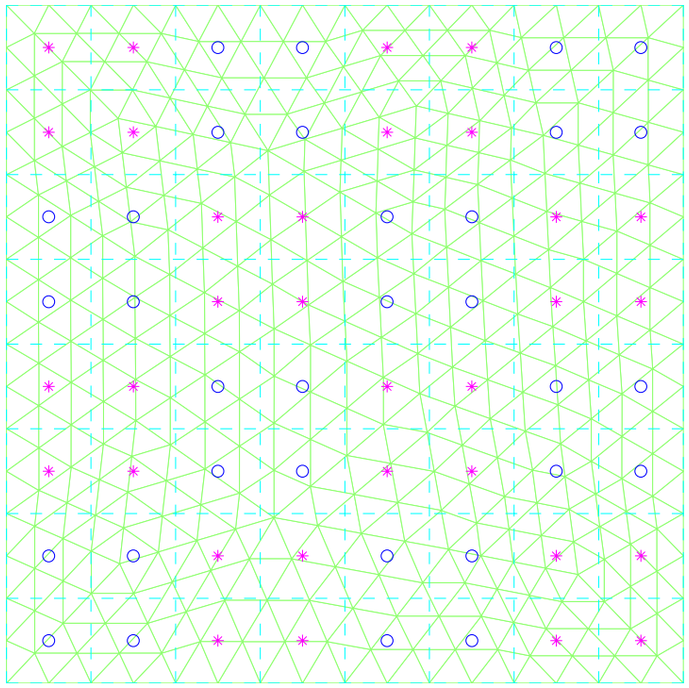
Parallel Aggregation on the finest level

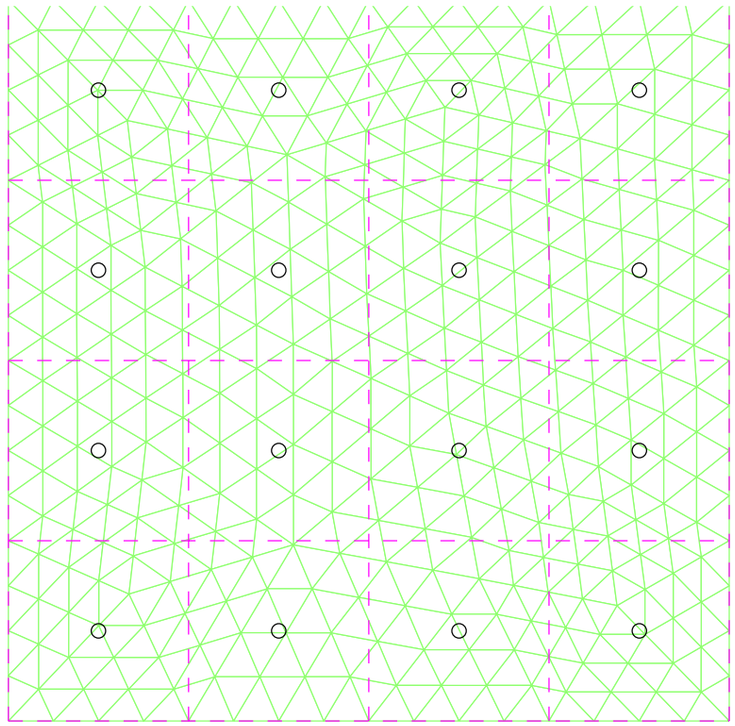
Parallel Aggregation on the finest level





Parallel Aggregation on the finest level





```
__global__ void aggregation coarse(int* aggregation, int level)
{
    const unsigned int idx = blockIdx.x*blockDim.x+threadIdx.x; % get thread ID
    const unsigned int powL = (int)pow(2.0, level);
    const unsigned int xIdx = (int)((idx%powL)/2); % check x index
    const unsigned int yIdx = (int)((idx/powL)/2); % check y index
    aggregation[idx] = yIdx*(powL/2) + xIdx; % label the aggregate
}
```

Prolongation& Restriction

Prolongation& Restriction

Prolongation $v = Pv^c \Rightarrow v_i = v_j^c, \text{ if } i \in G_j.$

```
__global__ void prolongation(int* aggregation, double* vc, double* v)
{
    const unsigned int idx = blockIdx.x*blockDim.x+threadIdx.x; % get thread ID
    v[idx] = vc[aggregation[idx]]; % assign the value
}
```

Prolongation& Restriction

Prolongation $v = Pv^c \Rightarrow v_i = v_j^c,$ if $i \in G_j$.

```
__global__ void prolongation(int* aggregation, double* vc, double* v)
{
    const unsigned int idx = blockIdx.x*blockDim.x+threadIdx.x; % get thread ID
    v[idx] = vc[aggregation[idx]]; % assign the value
}
```

Restriction $v^c = P^T v \Rightarrow v_i^c = \sum_{j \in G_i} v_j$

```
__global__ void restriction(int* aggregation, double* v, double* vc)
{
const unsigned int idx = blockIdx.x*blockDim.x+threadIdx.x; % get thread ID
vc[aggregation[idx]] = atomicAdd(&vc[aggregation[idx]], v[idx]); % assign the value
}
```

Coarse Grid Matrix $A_c = P^T A P$

Coarse Grid Matrix $A_c = P^T AP$

Triple matrices multiplication

In general, triple matrices multiplication consists two steps:

1. Determine the sparsity pattern of A_c
2. Compute the entries of A_c

In CUSP, triple matrices multiplication is done by COO format. **(1.65x)**

Coarse Grid Matrix $A_c = P^T A P$

Triple matrices multiplication

In general, triple matrices multiplication consists two steps:

1. Determine the sparsity pattern of A_c
2. Compute the entries of A_c

In CUSP, triple matrices multiplication is done by COO format. **(1.65x)**

Compute A_c using the auxiliary grid

Sparsity pattern is controlled for all the coarse matrix

Column index could be predefined

Parallel Smoother

Parallel Smoother

Some choice

Parallel Smoother

Some choice

- Damped Jacobi smoother

Parallel Smoother

Some choice

- Damped Jacobi smoother
- Polynomial smoother

Parallel Smoother

Some choice

- Damped Jacobi smoother
- Polynomial smoother
- Hybrid Gauss-Seidel smoother

Parallel Smoother

Some choice

- Damped Jacobi smoother
- Polynomial smoother
- Hybrid Gauss-Seidel smoother
- Colored Gauss-Seidel smoother:

Parallel Smoother

Some choice

- Damped Jacobi smoother
- Polynomial smoother
- Hybrid Gauss-Seidel smoother
- Colored Gauss-Seidel smoother:
  - has good smoothing property

Parallel Smoother

Some choice

- Damped Jacobi smoother
- Polynomial smoother
- Hybrid Gauss-Seidel smoother
- Colored Gauss-Seidel smoother:
  - has good smoothing property
  - has good parallel property

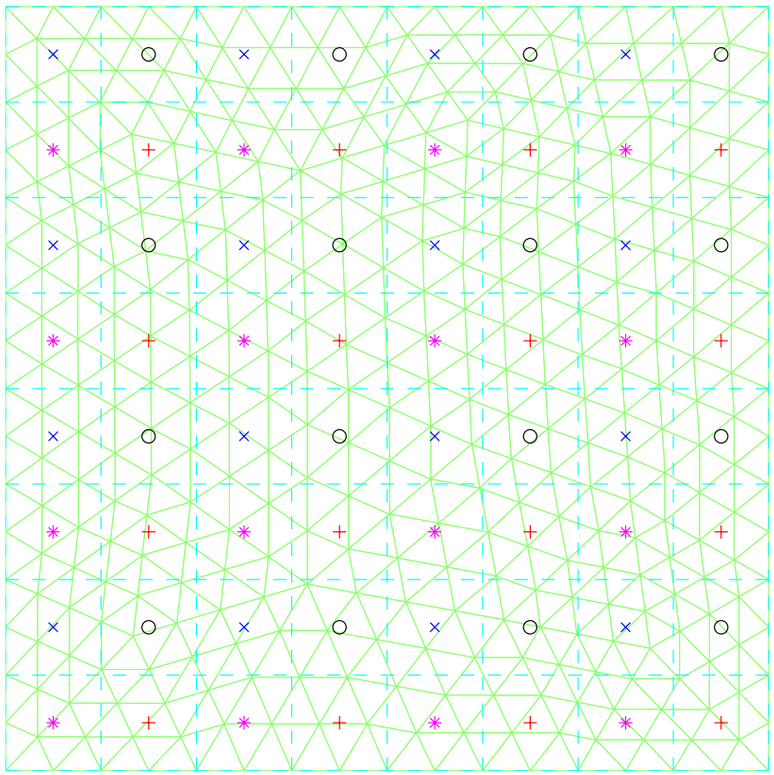
Parallel Smoother

Some choice

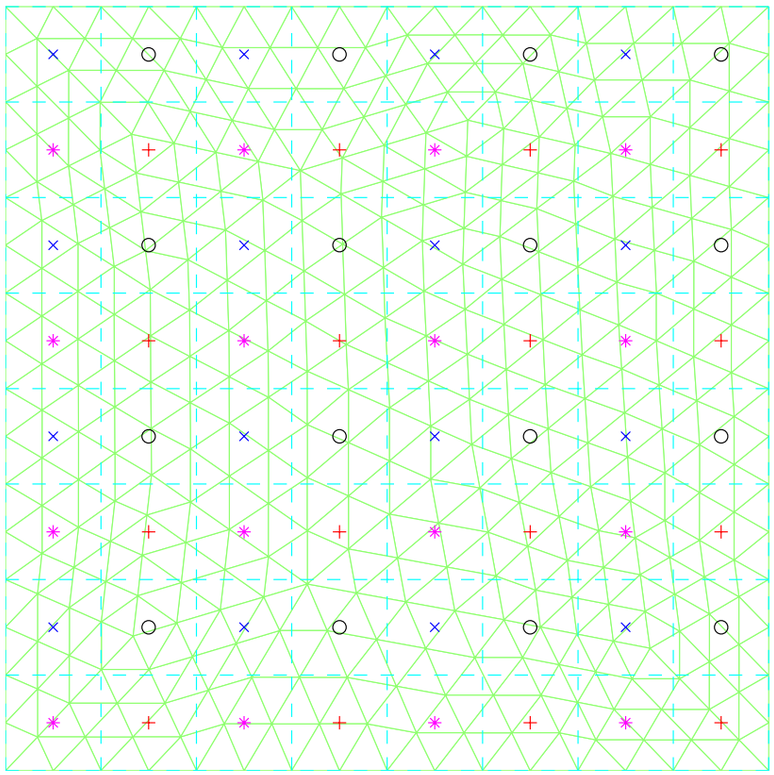
- Damped Jacobi smoother
- Polynomial smoother
- Hybrid Gauss-Seidel smoother
- Colored Gauss-Seidel smoother:
  - has good smoothing property
  - has good parallel property
  - difficult to apply to unstructured grid

4-color Gauss-Seidel

4-color Gauss-Seidel

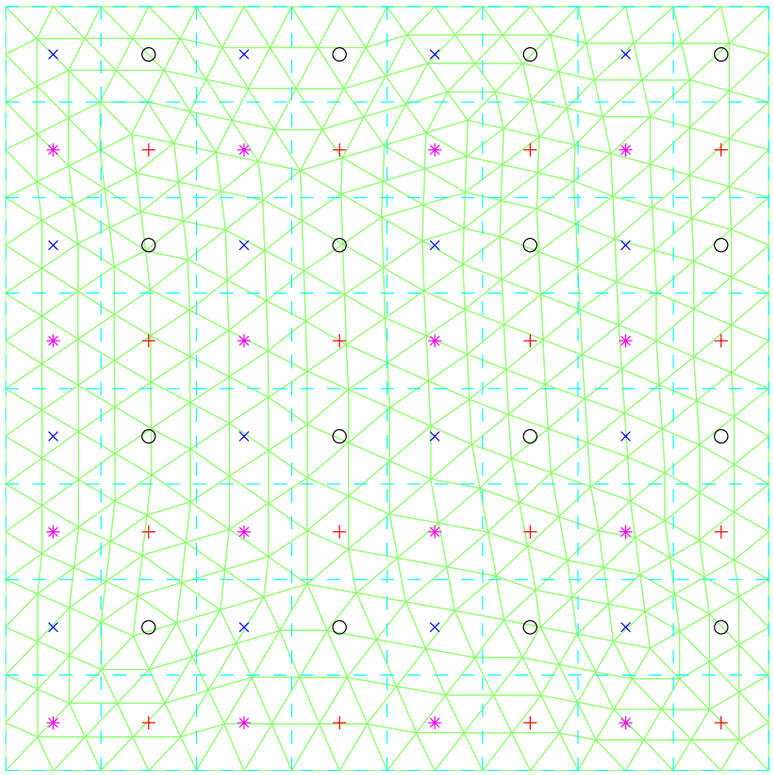


4-color Gauss-Seidel



- 4-color block Gauss-Seidel on the finest level

4-color Gauss-Seidel



- 4-color block Gauss-Seidel on the finest level
- 4-color block Gauss-Seidel on the coarse levels

Benefits of the Auxiliary Space AMG

Benefits of the Auxiliary Space AMG

Benefits of the Auxiliary Space AMG

- Setup Phase

Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices

Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices
  - Prolongation&restriction: no matrix need

Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices
  - Prolongation&restriction: no matrix need
  - Coarse-grid matrix: avoid triple matrices multiplication

Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices
  - Prolongation&restriction: no matrix need
  - Coarse-grid matrix: avoid triple matrices multiplication
- Solve Phase

Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices
  - Prolongation&restriction: no matrix need
  - Coarse-grid matrix: avoid triple matrices multiplication
- Solve Phase
  - SpMV: bounded bandwidth makes ELL format feasible

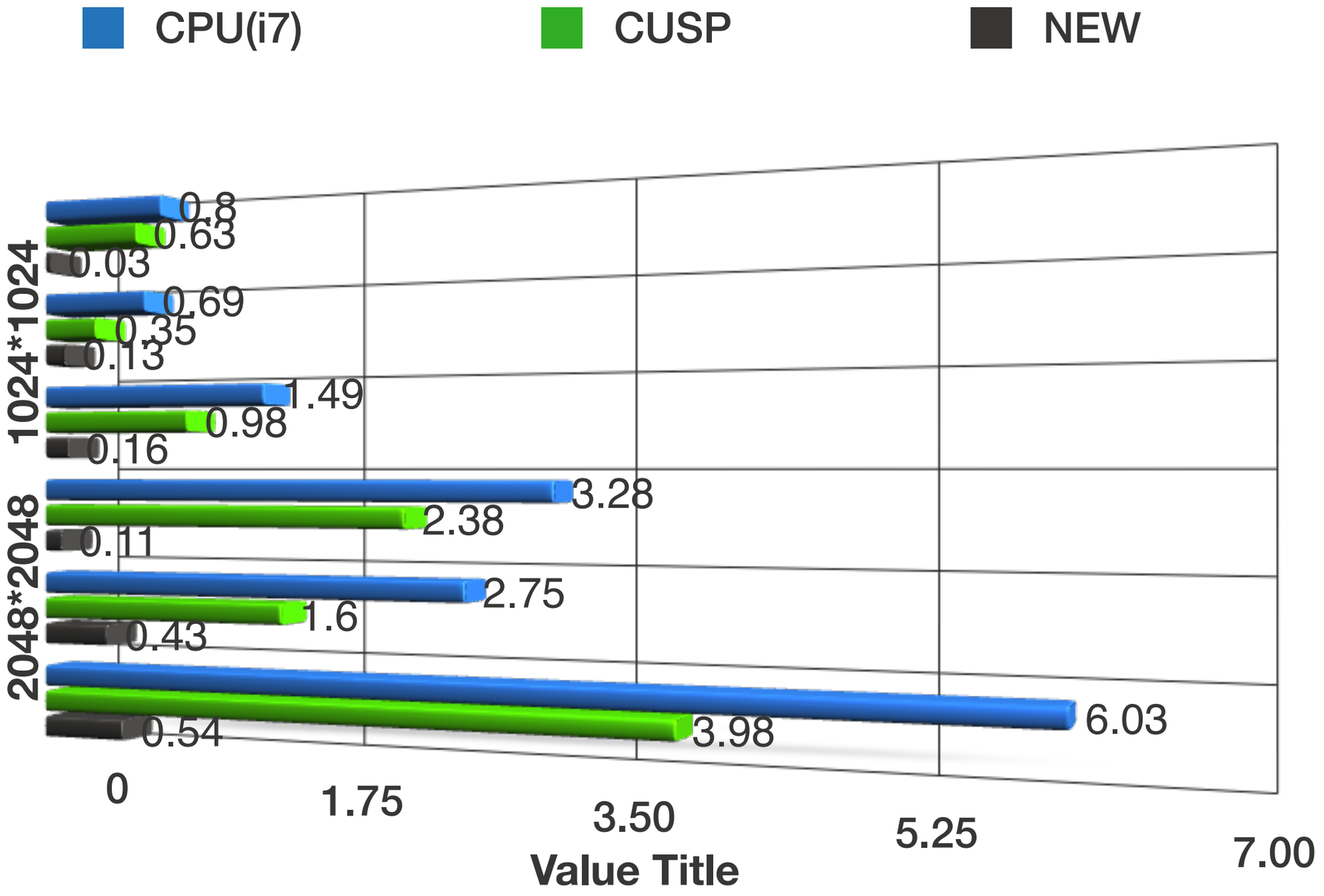
Benefits of the Auxiliary Space AMG

- Setup Phase
  - Form aggregates: only need to check coordinates or indices
  - Prolongation&restriction: no matrix need
  - Coarse-grid matrix: avoid triple matrices multiplication
- Solve Phase
  - SpMV: bounded bandwidth makes ELL format feasible
  - Parallel smoother: colored GS available

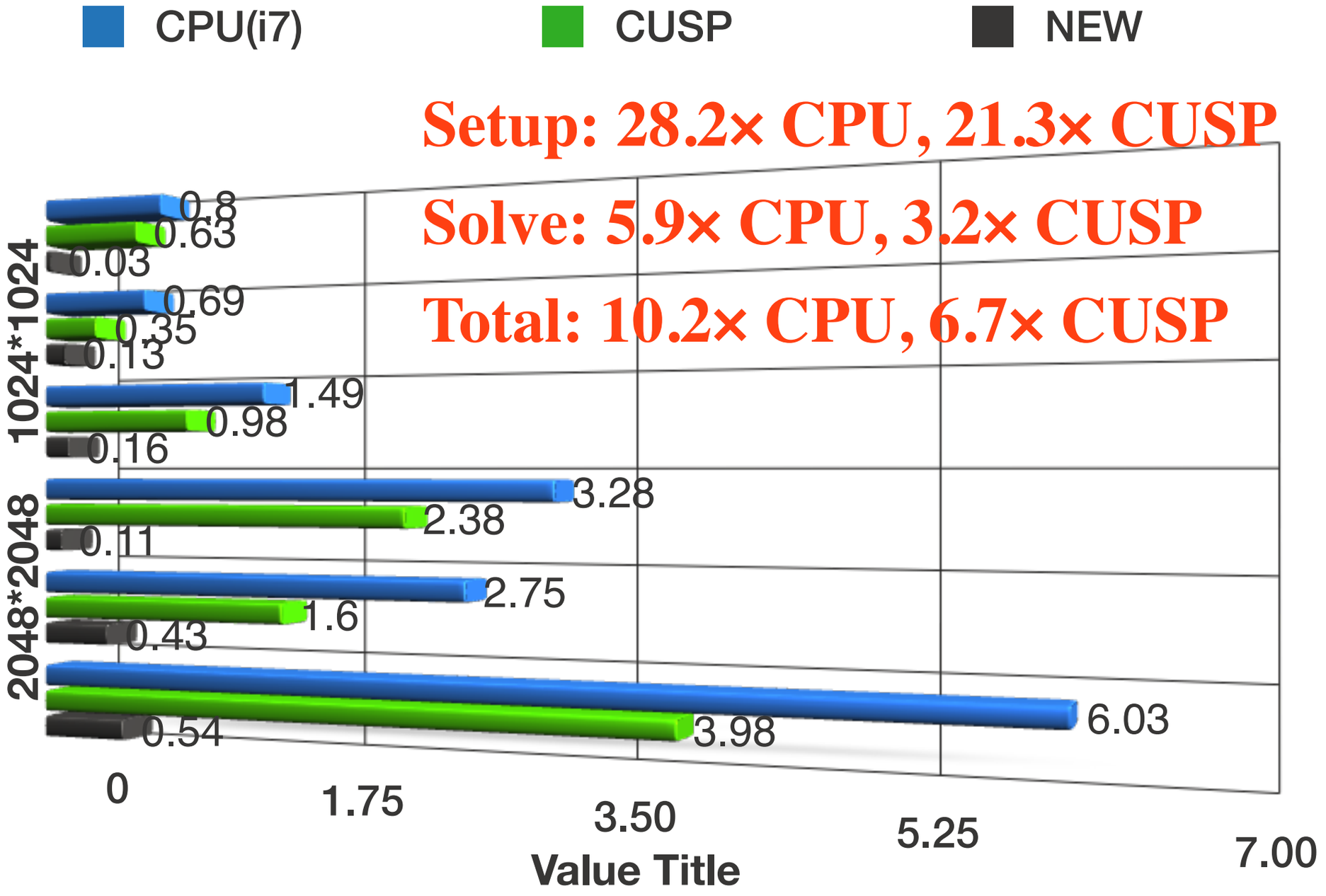
Poisson Problem on Uniform Grid

CPU(i7) CUSP NEW

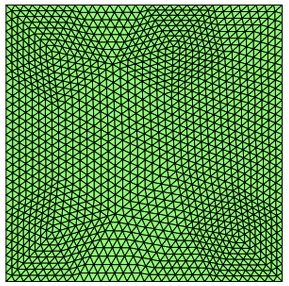
Poisson Problem on Uniform Grid



Poisson Problem on Uniform Grid



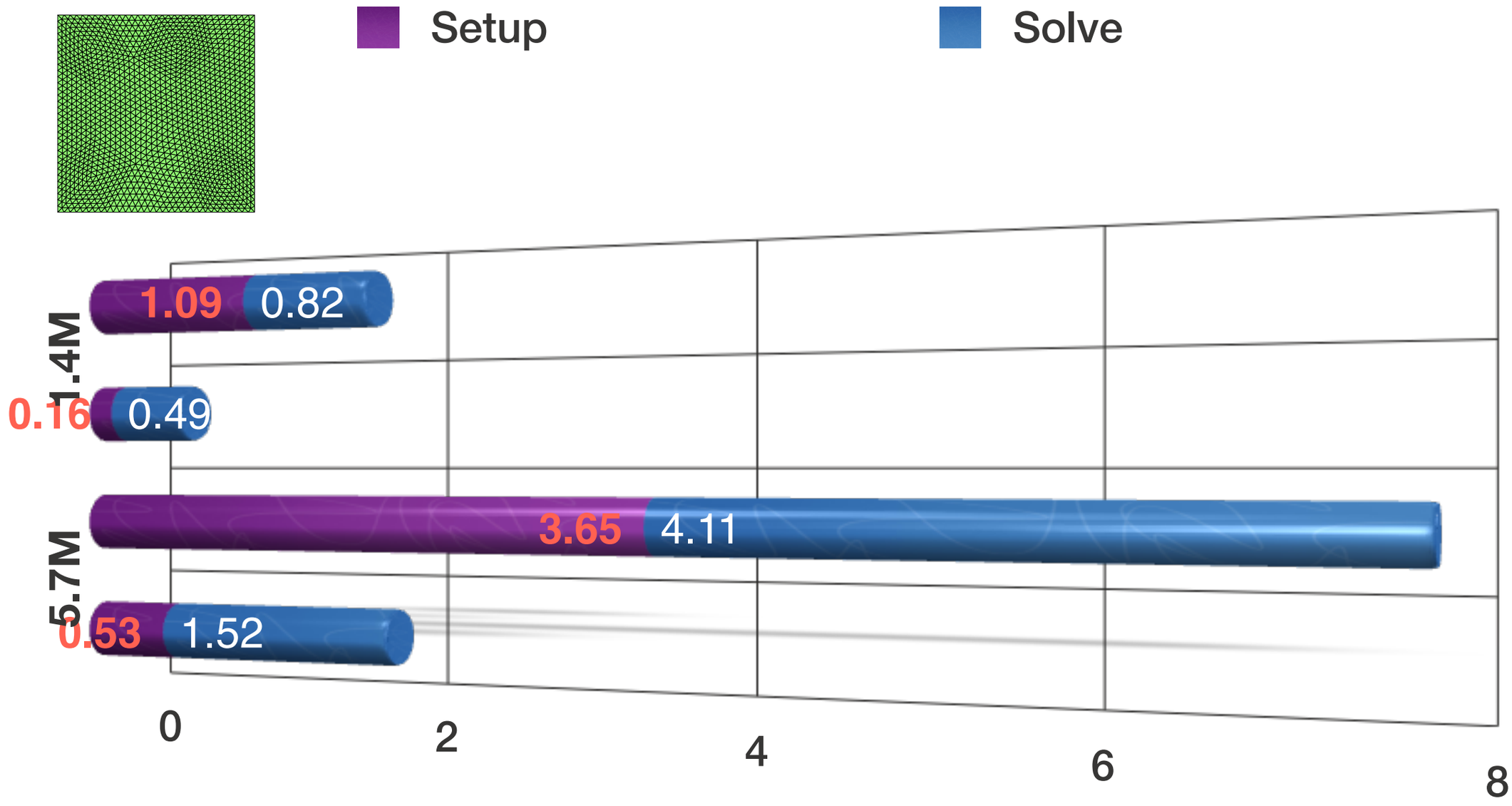
Poisson Problem on Quasi-uniform Grid



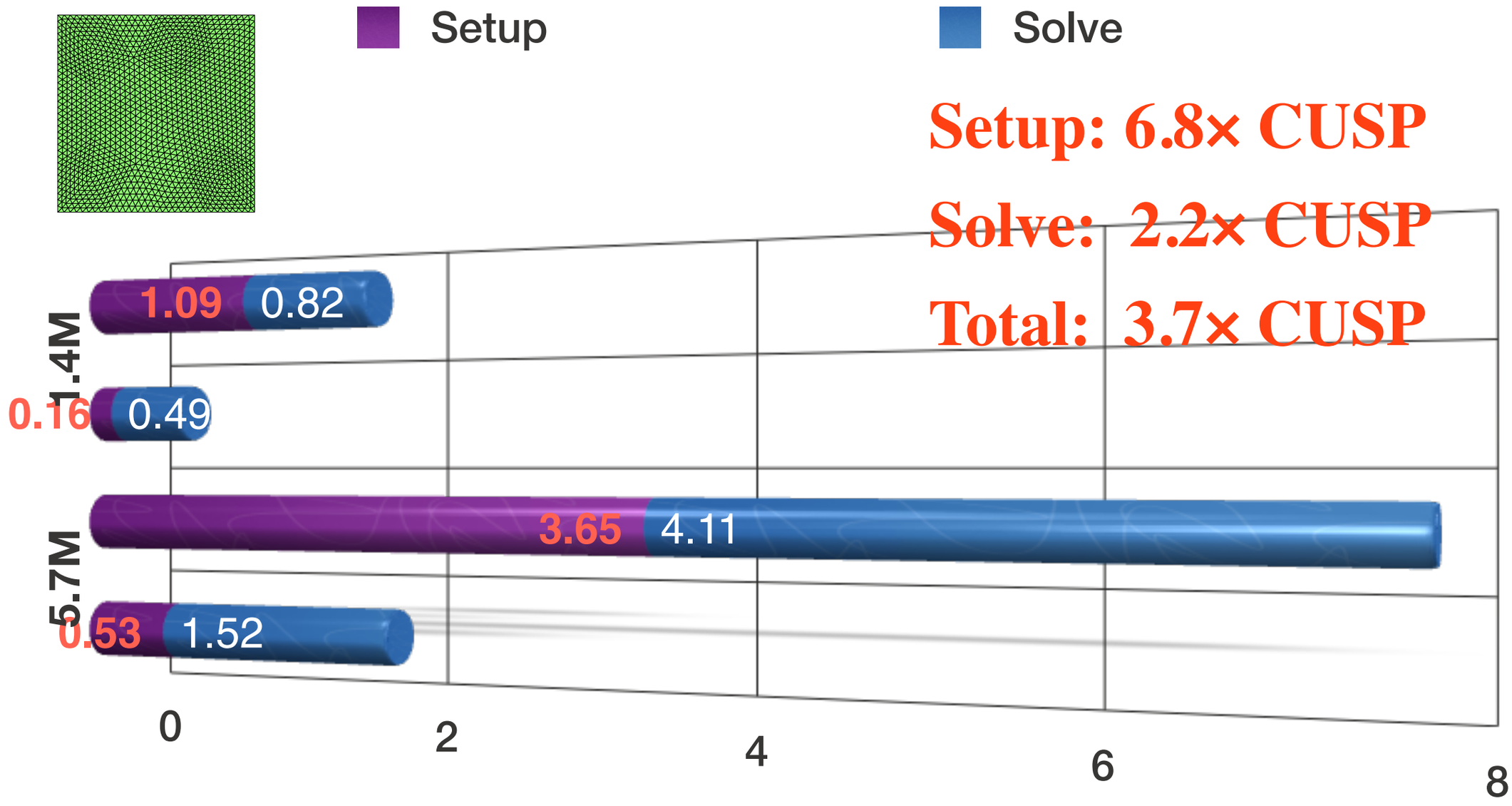
Setup

Solve

Poisson Problem on Quasi-uniform Grid



Poisson Problem on Quasi-uniform Grid

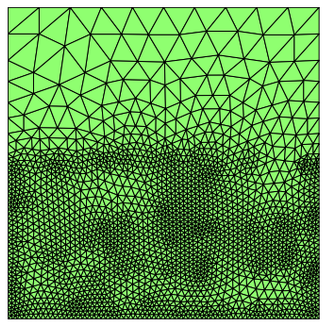


Setup: 6.8× CUSP

Solve: 2.2× CUSP

Total: 3.7× CUSP

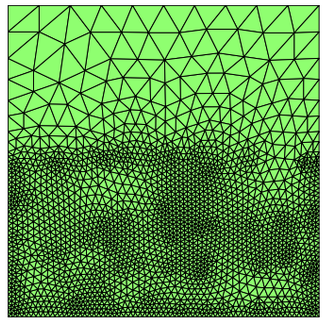
Poisson Problem on Shape-regular Grid

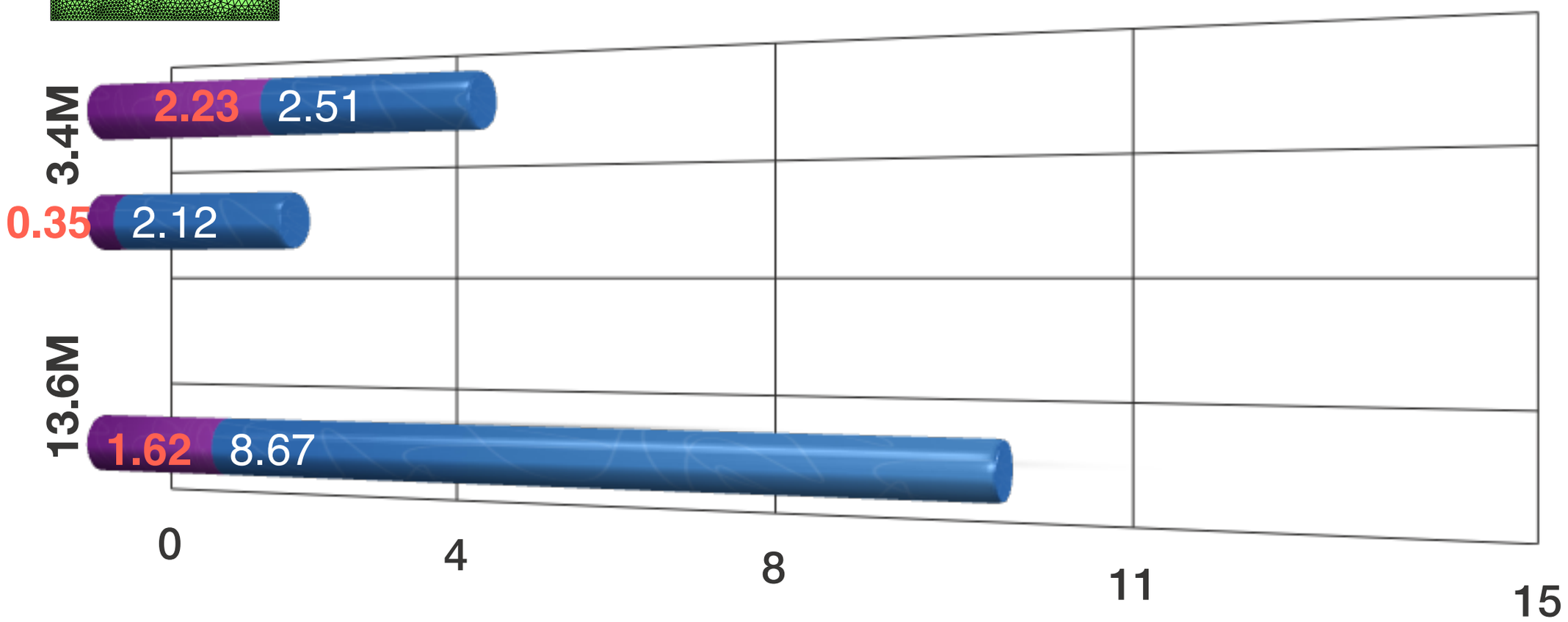


Setup

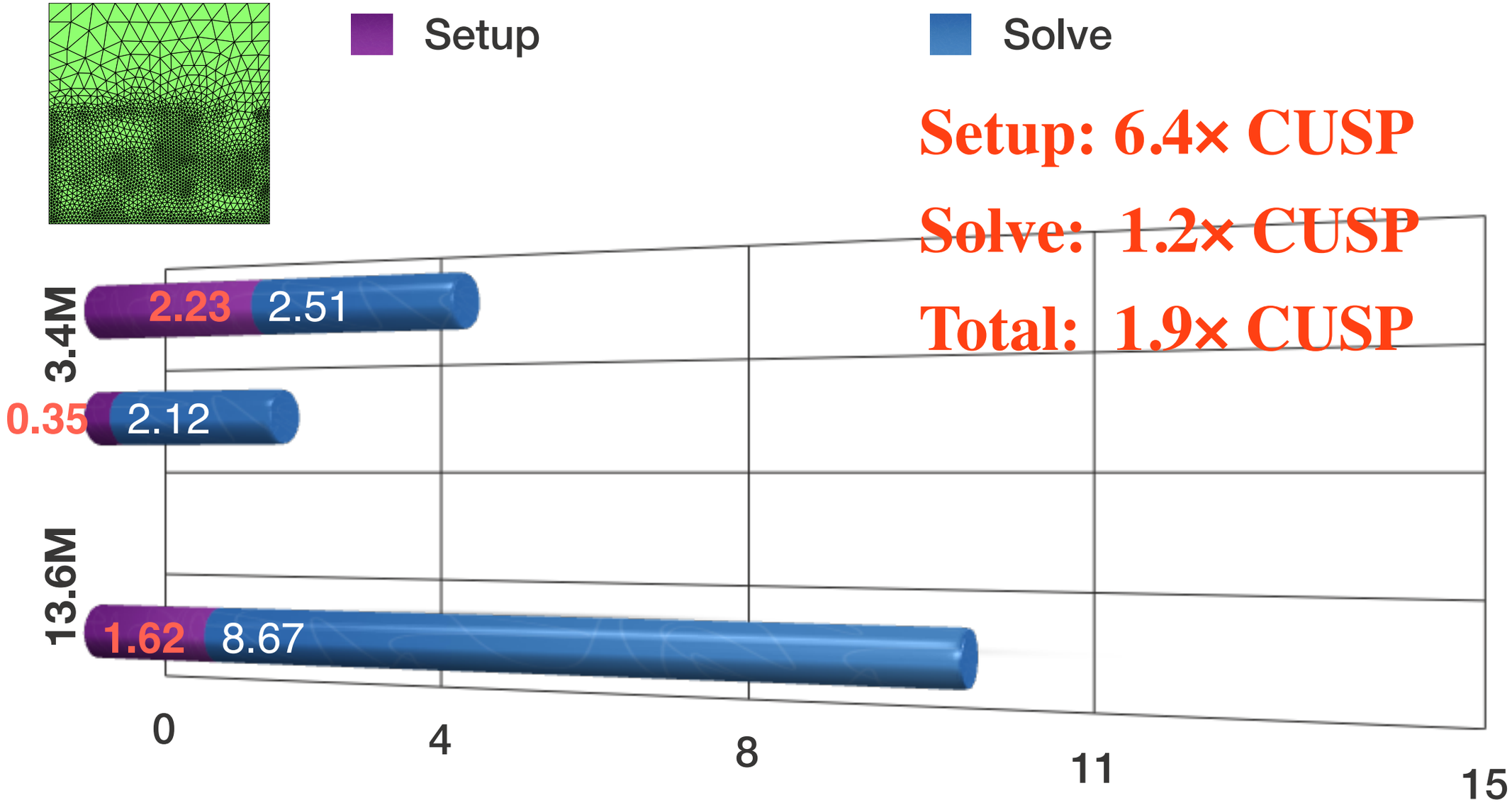
Solve

Poisson Problem on Shape-regular Grid





Poisson Problem on Shape-regular Grid

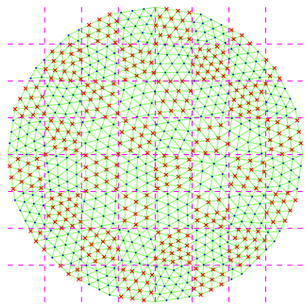


Setup: 6.4× CUSP

Solve: 1.2× CUSP

Total: 1.9× CUSP

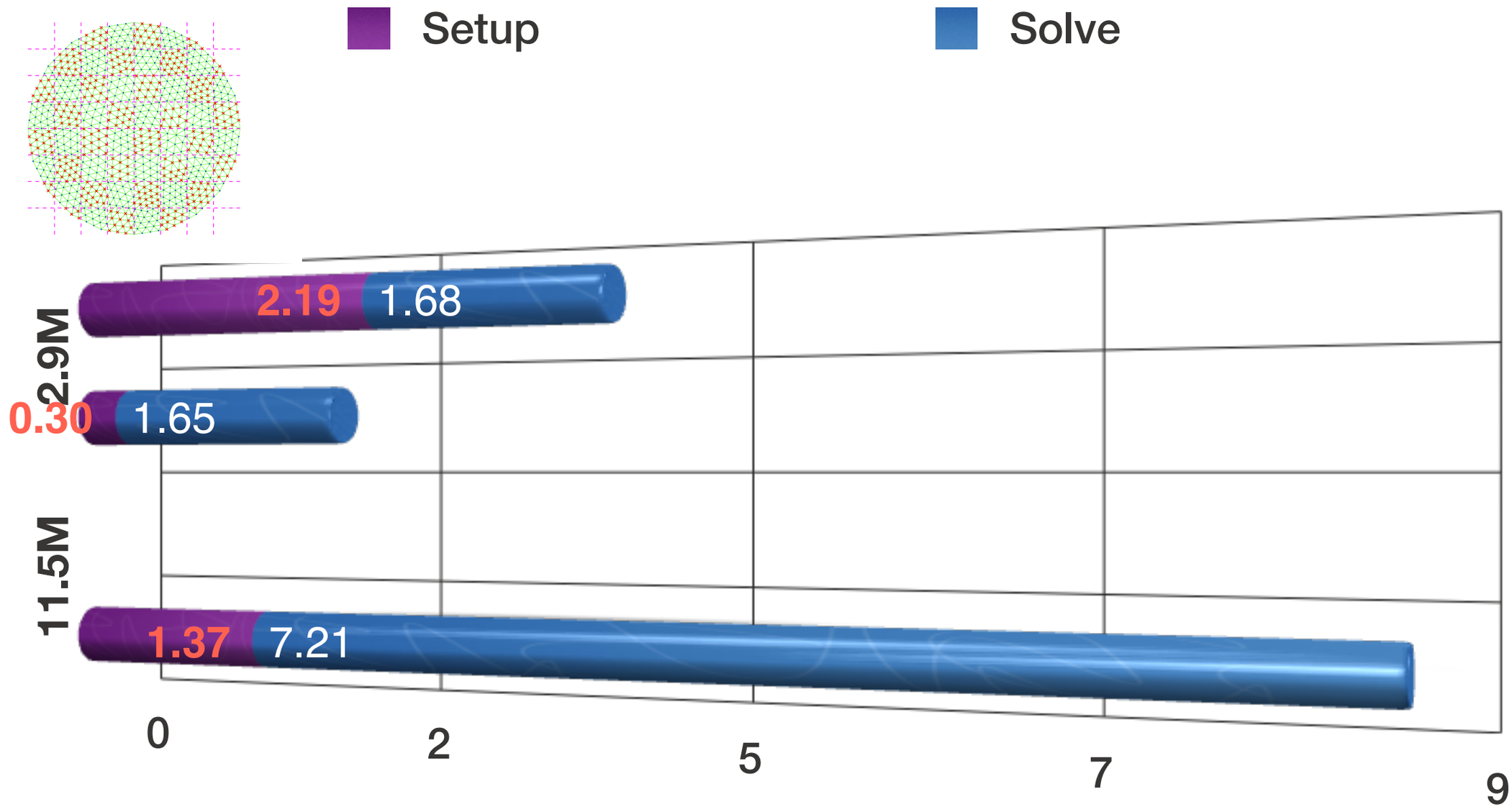
Poisson Problem on Disk Grid



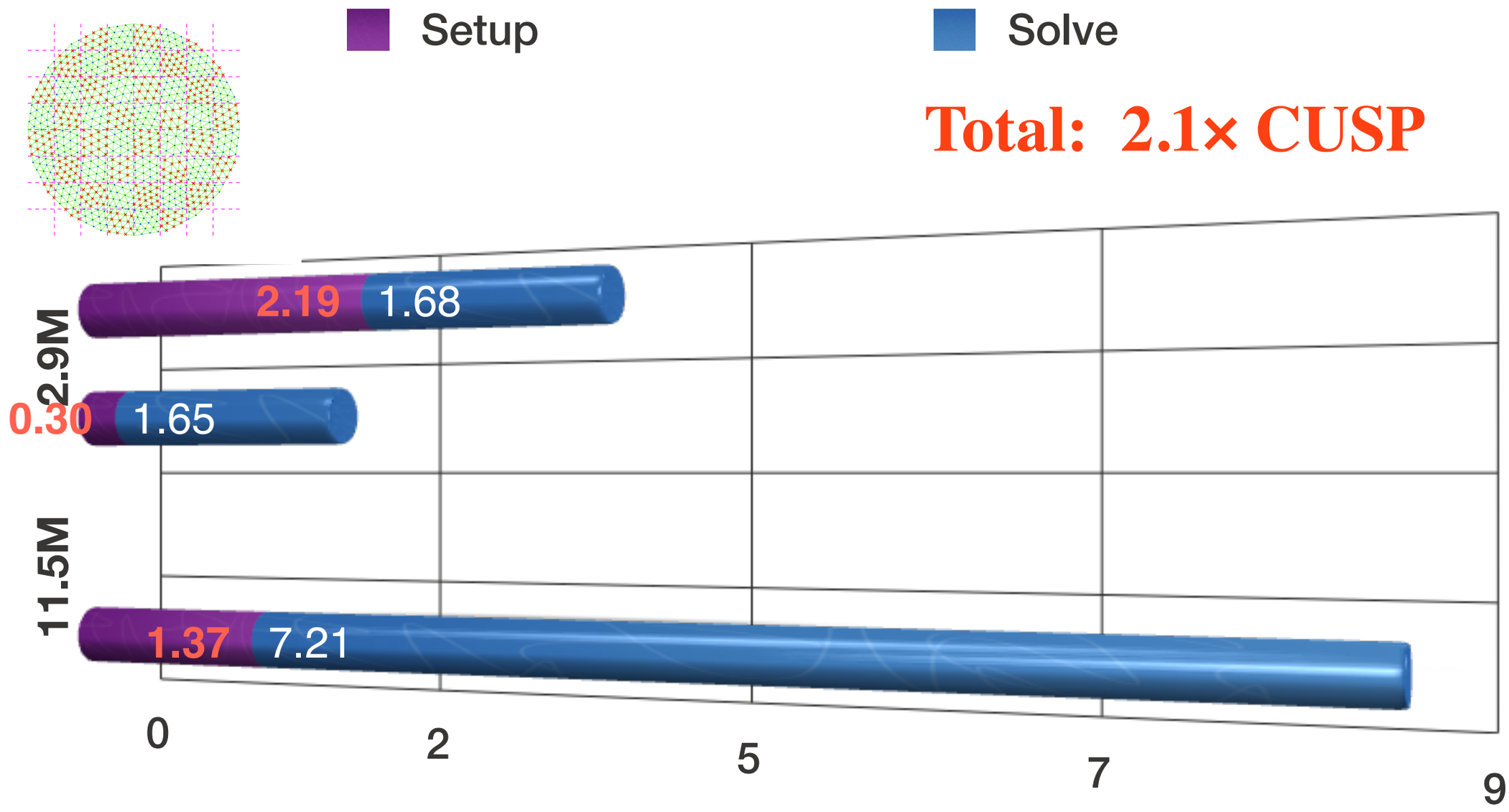
Setup

Solve

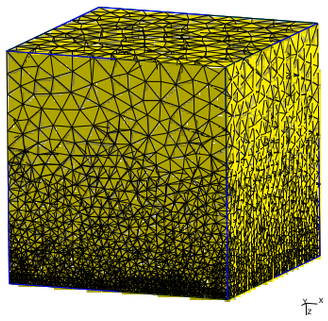
Poisson Problem on Disk Grid



Poisson Problem on Disk Grid



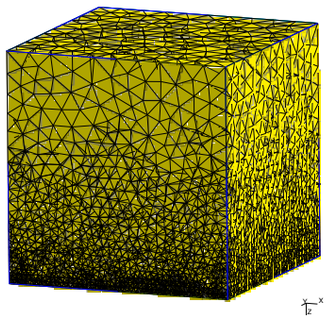
Heat Transfer Problem Problem on a cube

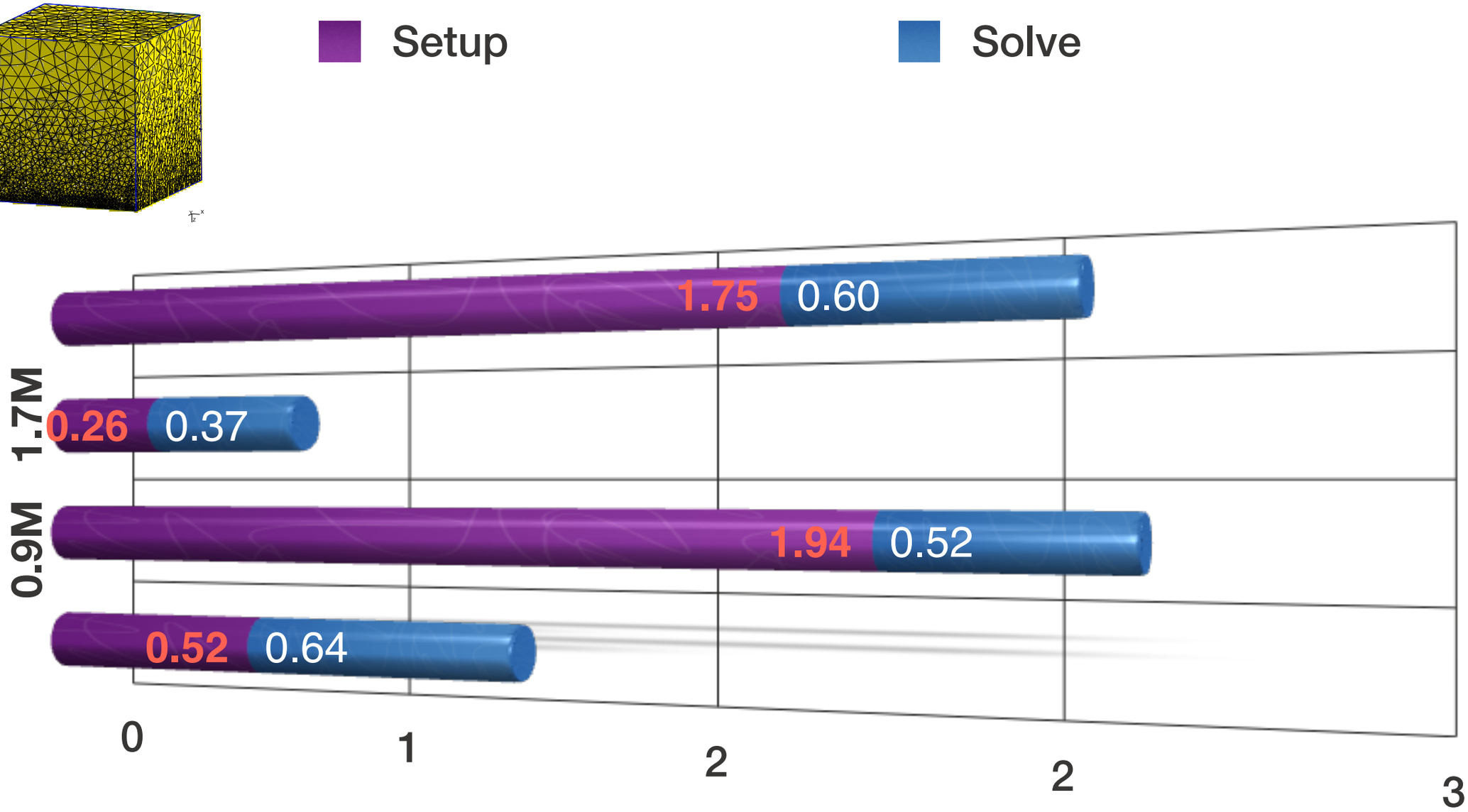


Setup

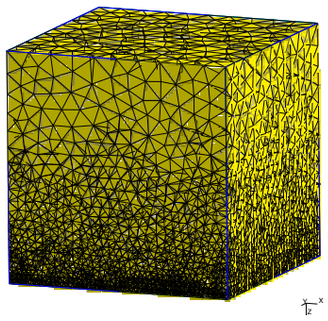
Solve

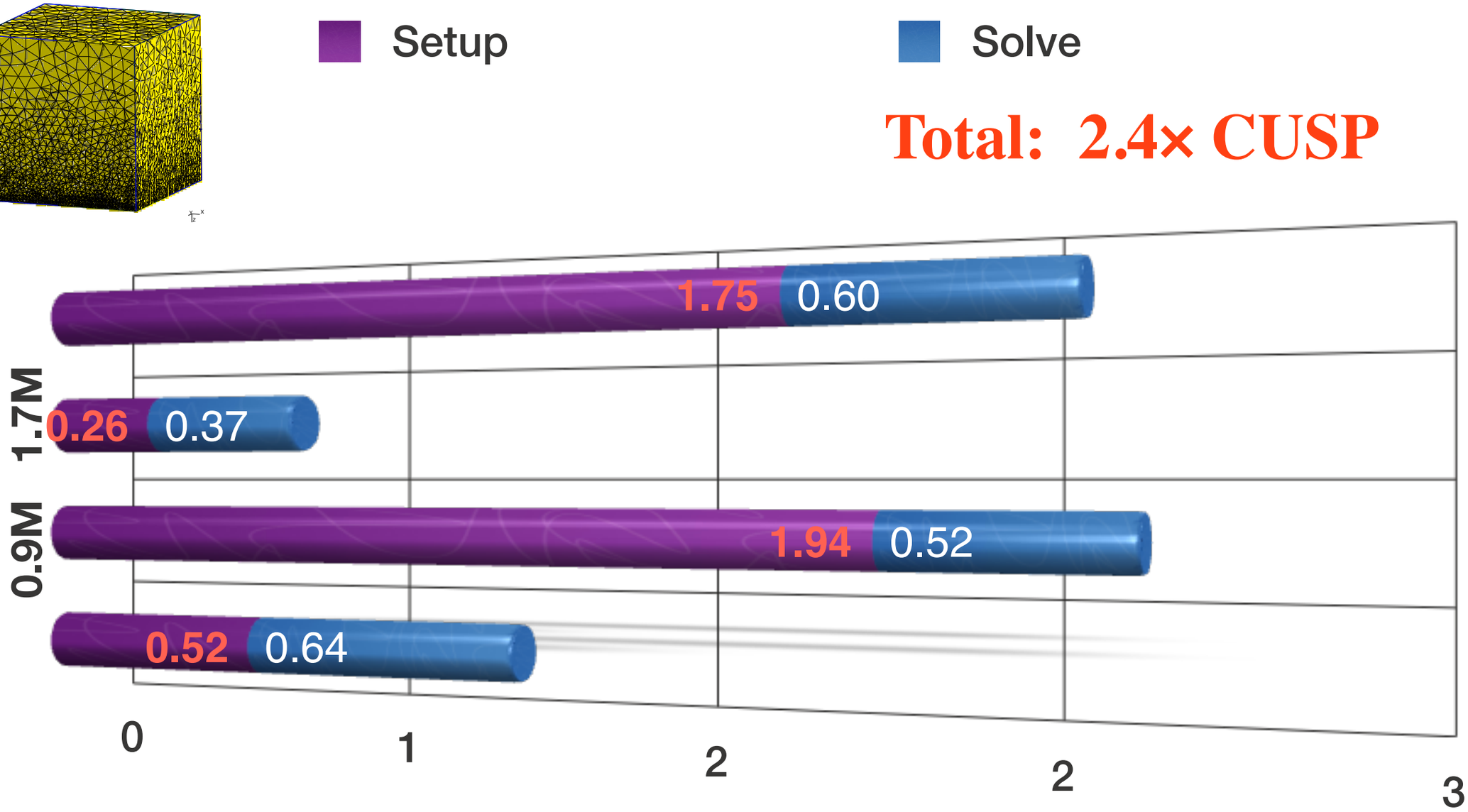
Heat Transfer Problem Problem on a cube



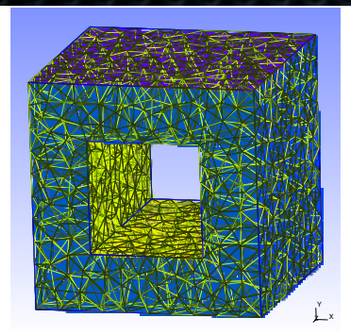


Heat Transfer Problem Problem on a cube





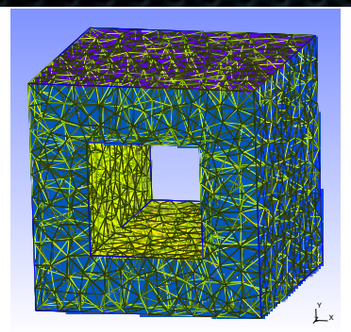
Heat Transfer Problem on Quasi-uniform Grid

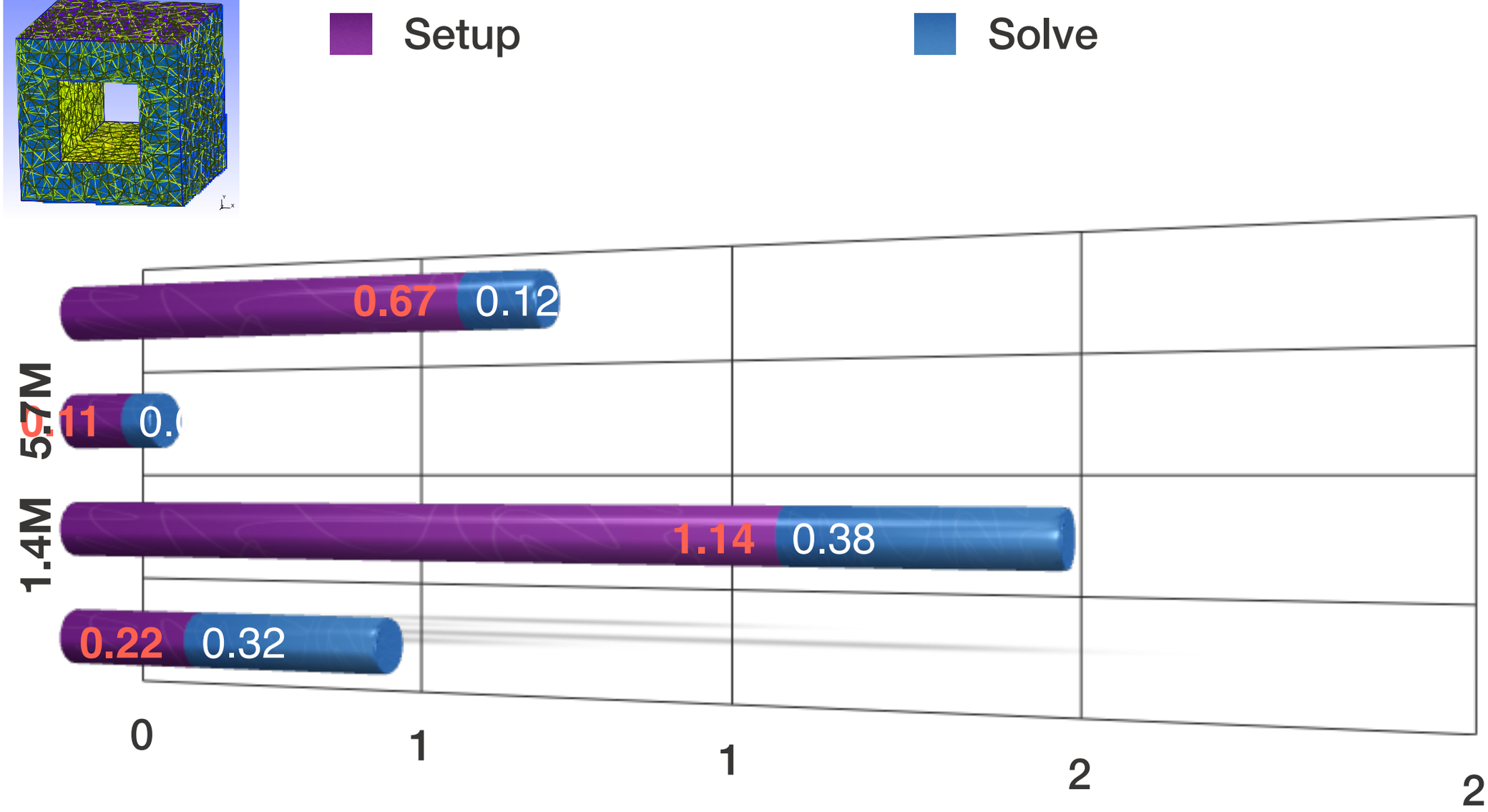


Setup

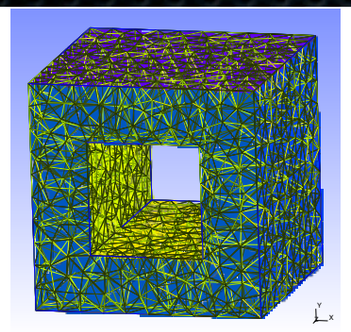
Solve

Heat Transfer Problem on Quasi-uniform Grid





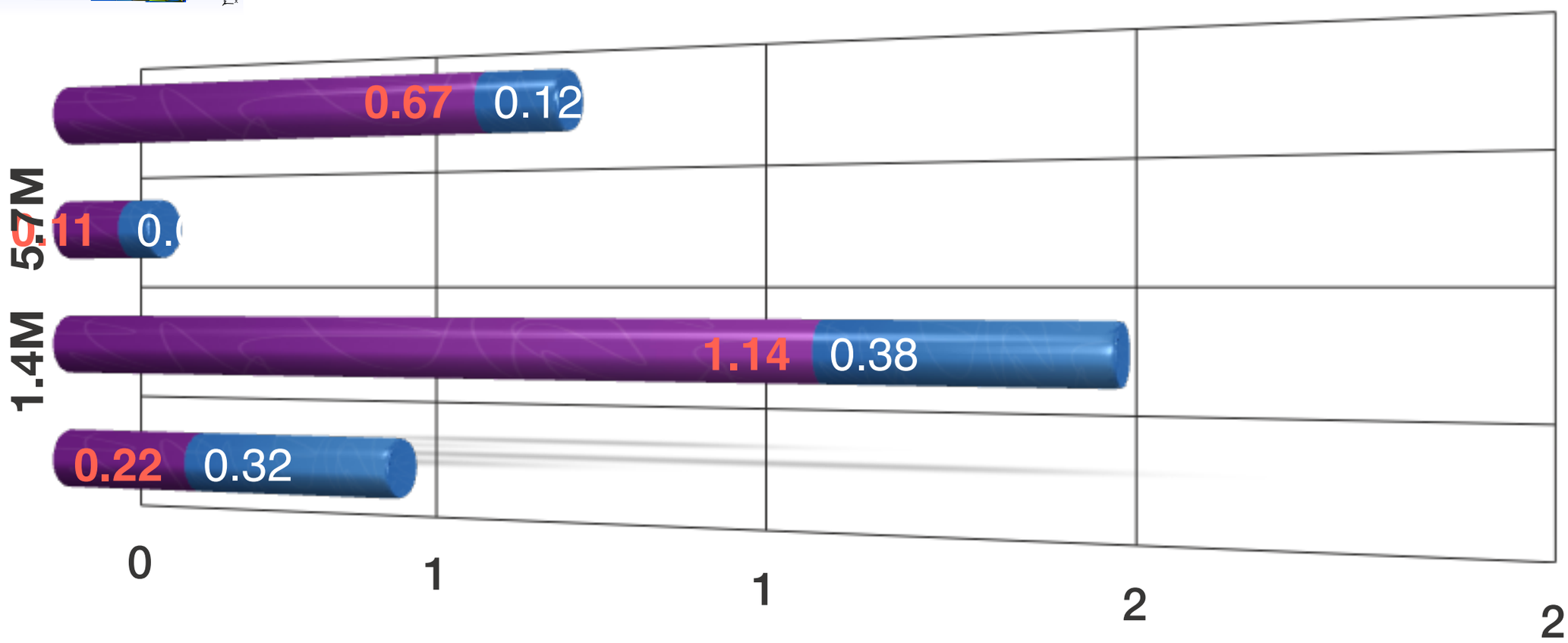
Heat Transfer Problem on Quasi-uniform Grid



Setup

Solve

Total: 2.8× CUSP



Conclusion

Conclusion

- We developed a parallel AMG method based on an auxiliary grid

Conclusion

- We developed a parallel AMG method based on an auxiliary grid
- We implemented the algorithm on GPU

Conclusion

- We developed a parallel AMG method based on an auxiliary grid
- We implemented the algorithm on GPU
- Numerical results show that our new method can speed up the existing GPU code (CUSP) at least 2 for isotropic problems

Conclusion

- We developed a parallel AMG method based on an auxiliary grid
- We implemented the algorithm on GPU
- Numerical results show that our new method can speed up the existing GPU code (CUSP) at least 2 for isotropic problems

- Fast way to construct the adaptive tree

Conclusion

- We developed a parallel AMG method based on an auxiliary grid
- We implemented the algorithm on GPU
- Numerical results show that our new method can speed up the existing GPU code (CUSP) at least 2 for isotropic problems

- Fast way to construct the adaptive tree
- Efficient cycles implementation

Conclusion

- We developed a parallel AMG method based on an auxiliary grid
- We implemented the algorithm on GPU
- Numerical results show that our new method can speed up the existing GPU code (CUSP) at least 2 for isotropic problems

- Fast way to construct the adaptive tree
- Efficient cycles implementation

L. Wang, X. Hu, J. Cohen, and J. Xu, A *Parallel Auxiliary Grid AMG method for GPU*, accepted by SISC, 2013
X. Hu, P. Vassilevski, and J. Xu, *Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid*, accepted by SINUM, 2012

Thank you

Acknowledgements: NVIVIA Academic Partership Award