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

Scientific Simulations
Models ➔ Algorithms ➔ Algebraic Systems

Scientific Simulations

- CFD Simulation
- Electromagnetism
- Continuum mechanics
### Problems

<table>
<thead>
<tr>
<th>Problems</th>
<th>Partial Differential Equations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisson</td>
<td>$-\Delta u = f$</td>
</tr>
<tr>
<td>Reaction-Diffusion</td>
<td>$- \nabla \cdot (\mu(x)\nabla u) + c(x)u = f$</td>
</tr>
<tr>
<td>H(curl)</td>
<td>$\nabla \times \nabla \times u + u = f$</td>
</tr>
<tr>
<td>H(div)</td>
<td>$- \nabla \nabla \cdot u + u = f$</td>
</tr>
<tr>
<td>Maxwell</td>
<td>$B_t + \nabla \times E = 0$, $\nabla \cdot D = \rho$, $D_t - \nabla \times H = -J$, $\nabla \cdot B = 0$</td>
</tr>
<tr>
<td>Darcy</td>
<td>$- \nabla \cdot u = f$, $u = \mu \nabla p$</td>
</tr>
<tr>
<td>Stokes</td>
<td>$- \Delta u + \nabla p = f$, $\nabla \cdot u = 0$</td>
</tr>
<tr>
<td>Brinkman</td>
<td>$- \nabla \cdot (\mu(x)\nabla u) + c(x)u + \nabla p = f$, $\nabla \cdot u = g$</td>
</tr>
<tr>
<td>Linear Elasticity</td>
<td>$- \Delta u - \frac{1}{1-2\nu} \nabla \nabla \cdot u = f$</td>
</tr>
<tr>
<td>Biharmonic (plate)</td>
<td>$\Delta^2 u = f$</td>
</tr>
<tr>
<td>Reissner-Mindlin</td>
<td>$- \text{div}(CD(\phi)) = \lambda t^{-2}(\phi - \nabla \omega) = 0$, $\lambda t^{-2}(\nabla \omega + \text{div} \phi) = g$</td>
</tr>
<tr>
<td>R–C–D</td>
<td>$u_t + v(x) \cdot \nabla u - \nabla \cdot (\mu(x)\nabla u) + c(x)u = f$</td>
</tr>
<tr>
<td>Navier–Stokes</td>
<td>$u_t + (u \cdot \nabla)u - \mu \Delta u + \nabla p = f$, $\nabla \cdot u = 0$</td>
</tr>
<tr>
<td>Johnson-Segelman</td>
<td>$u_t + (u \cdot \nabla)u - \mu S \Delta u + \nabla p = \nabla \cdot \tau$, $\nabla \cdot u = 0$, $\tau + W[iu_t + (u \cdot \nabla)u - R\tau - \tau R^T] = \frac{1}{2} \mu p \nabla u$</td>
</tr>
<tr>
<td></td>
<td>$R = \frac{a+1}{2} \nabla u + \frac{a-1}{2} \nabla u^T$</td>
</tr>
<tr>
<td>MHD</td>
<td>$u_t + (u \cdot \nabla)u + \nabla p = \mu \Delta u + \mu_0 J \times B$, $\nabla \cdot u = 0$, $B_t + \nabla \times E = 0$, $\nabla \times B = \mu_0 J$, $\nabla \cdot B = 0$, $E + u \times B = \sigma_0^{-1} J$</td>
</tr>
</tbody>
</table>

**Black-oil model**

---

**Scientific Simulations**

**Wednesday, March 20, 13**
Models $\rightarrow$ Algorithms $\rightarrow$ Algebraic Systems

Scientific Simulations
PDE model

Algorithms
Scientific Simulations ➞ Algorithms ➞ Algebraic Systems

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

PDE model
Scientific Simulations

Models $\rightarrow$ Algorithms $\rightarrow$ Algebraic Systems

Discretization

PDE model

Algorithms

Algebraic Equations
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:

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
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 ($O(N^3)$)
**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 ($O(N^3)$)**

<table>
<thead>
<tr>
<th>$N=10^6$</th>
<th>$N=10^7$</th>
<th>$N=10^8$</th>
<th>$N=10^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.98s</td>
<td>5.27hr</td>
<td>220 days</td>
<td>602 Years</td>
</tr>
</tbody>
</table>

*Wednesday, March 20, 13*
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 ($O(N^3)$)
  
<table>
<thead>
<tr>
<th>$N=10^6$</th>
<th>$N=10^7$</th>
<th>$N=10^8$</th>
<th>$N=10^9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>18.98s</td>
<td>5.27hr</td>
<td>220 days</td>
<td>602 Years</td>
</tr>
</tbody>
</table>

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

- **Optimal solvers, e.g. Multigrid (MG) methods (O(N))**
  - $N=10^6$: 0.5s
  - $N=10^7$: 5s
  - $N=10^8$: 50s
  - $N=10^9$: 500s
Geometric Multigrid Method
Geometric Multigrid Method
Geometric Multigrid Method

$A_h$
$A_h \Rightarrow (GS)_h$
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

\[ A_h \Rightarrow (GS)_h \]

\[ + \]

\[ A_{2h} \Rightarrow (GS)_{2h} \]

\[ + \]

\[ A_{4h} \Rightarrow (GS)_{4h} \]

\[ + \]

\[ A_{8h} \Rightarrow \ldots \]
Geometric Multigrid Method

\[ A_h \Rightarrow (GS)_h \]
\[ A_{2h} \Rightarrow (GS)_{2h} \]
\[ A_{4h} \Rightarrow (GS)_{4h} \]
\[ A_{8h} \Rightarrow \ldots \]

\[ O(N_h) + O(N_{2h}) + O(N_{4h}) + \ldots = O(N_h) \]
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 / ... \((R = P^T)\)
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)
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:
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
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

$$I - B^{MG} A = (I - S^T A)(I - PB^{MG}_c RA)(I - SA)$$
Parallel AMG
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

Grid_baltic

Baltic Lake
MG for Unstructured Grid

Grid_baltic

Baltic Lake
MG for Unstructured Grid

Grid_baltic

Baltic Lake
\[ B_h = S_h + \Pi_h \bar{B}_h \Pi_h^T \]
FASP Preconditioner

<table>
<thead>
<tr>
<th>Problems</th>
<th>Partial Differential Equations</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poisson</td>
<td>$-\nabla u = f$</td>
<td>1</td>
</tr>
<tr>
<td>Reaction-Diffusion</td>
<td>$-\nabla \cdot (\mu(x)\nabla u) + c(x)u = f$</td>
<td>1</td>
</tr>
<tr>
<td>H(curl)</td>
<td>$\nabla \times \nabla \times u + u = f$</td>
<td>4</td>
</tr>
<tr>
<td>H(div)</td>
<td>$-\nabla \nabla \cdot u + u = f$</td>
<td>3–6</td>
</tr>
<tr>
<td>Maxwell</td>
<td>$B_t + \nabla \times E = 0, \nabla \cdot D = \rho,$ $D_t - \nabla \times H = -J,$ $\nabla \cdot B = 0$</td>
<td>4</td>
</tr>
<tr>
<td>Darcy</td>
<td>$-\nabla \cdot u = f, \ u = \mu \nabla p$</td>
<td>3–6</td>
</tr>
<tr>
<td>Stokes</td>
<td>$-\Delta u + \nabla p = f, \ \nabla \cdot u = 0$</td>
<td>3</td>
</tr>
<tr>
<td>Brinkman</td>
<td>$-\nabla \cdot (\mu(x)\nabla u) + c(x)u + \nabla p = f, \ \nabla \cdot u = g$</td>
<td>3</td>
</tr>
<tr>
<td>Linear Elasticity</td>
<td>$-\Delta u - \frac{\nu}{1-2\nu} \nabla \nabla \cdot u = f$</td>
<td>3</td>
</tr>
<tr>
<td>Biharmonic (plate)</td>
<td>$\Delta^2 u = f$</td>
<td>2</td>
</tr>
<tr>
<td>Reissner-Mindlin</td>
<td>$-\text{div}(CD(\phi)) = \lambda t^{-2}(\phi - \nabla \omega) = 0,$ $\lambda t^{-2}(-\Delta \omega + \text{div} \phi) = g$</td>
<td>4</td>
</tr>
<tr>
<td>R–C–D</td>
<td>$u_t + v(x) \cdot \nabla u - \nabla \cdot (\mu(x)\nabla u) + c(x)u = f$</td>
<td>1</td>
</tr>
<tr>
<td>Navier–Stokes</td>
<td>$u_t + (u \cdot \nabla)u - \mu \Delta u + \nabla p = f, \ \nabla \cdot u = 0$</td>
<td>4</td>
</tr>
<tr>
<td>Johnson–Seglman</td>
<td>$u_t + (u \cdot \nabla)u - \mu_s \Delta u + \nabla p = \nabla \cdot \tau, \ \nabla \cdot u = 0,$ $\tau + \text{Wi}[u_t + (u \cdot \nabla)u - R\tau - \tau R^T] = \frac{1}{2} \mu p \nabla \tau$ $R = \frac{3+1}{2} \nabla u + \frac{3-1}{2} \nabla u^T$</td>
<td>3</td>
</tr>
<tr>
<td>MHD</td>
<td>$u_t + (u \cdot \nabla)u + \nabla p = \mu \Delta u + \mu_0 J \times B, \ \nabla \cdot u = 0,$ $B_t + \nabla \times E = 0, \ \nabla \times B = \mu_0 J,$ $\nabla \cdot B = 0, \ E + u \times B = \sigma_0^{-1} J$</td>
<td>7</td>
</tr>
<tr>
<td>Black-oil model</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unsmoothed Aggregation AMG
(Braess, 1995; Kim, Xu, and Ludmil 2003)
Aggregation (coarsening): Find a non-overlapping partition \( \{G_j\} \), which are referred as aggregates.
Aggregation (coarsening): Find a non-overlapping partition \( \{G_{ij}\} \), 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

  \[
  (A_c)_{kl} = \sum_{i \in G_i} \sum_{j \in G_j} a_{ij}
  \]
A drawback: Only two-grid method converges uniformly
A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:
A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid \( \Rightarrow \) Optimal multigrid
A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid $\Rightarrow$ Optimal multigrid
  - AMLI-cycle
A drawback: Only two-grid method converges uniformly

Enhanced MG cycles could help:

- Optimal two-grid $\Rightarrow$ 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 $\Rightarrow$ 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 $\Rightarrow$ 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-cycle method converges uniformly
Example

\[ \begin{aligned} -\Delta u &= f, \quad \text{in } \Omega \\
    u &= 0, \quad \text{on } \partial\Omega \end{aligned} \]
Example

\[
\begin{cases}
-\Delta u = f, & \text{in } \Omega \\
u = 0, & \text{on } \partial \Omega
\end{cases}
\]
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
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 = P v^c \Rightarrow v_i = v_j^c,
Prolongation & Restriction

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

Restriction

For $i \in G_j$, $v = Pu^c \Rightarrow v_i = u^c_j$. 
Prolongation & Restriction

**Prolongation**
- Value of a leaf node
  \[= \text{Value of its parent node}\]

**Restriction**
- Value of parent node
  \[= \text{sum of its children nodes}\]

For \(i \in G_j\),
\[v = P v^c \Rightarrow v_i = v^c_j,\]
Prolongation & Restriction

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

For \( i \in G_j \),
\[
v = P^t v^c \Rightarrow v_i = v^c_j,
\]

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

- 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
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)

Figure 12: Bandwidth results for structured matrices with single precision values.

While the peak rate of double precision arithmetic on the GTX 285 is an order of magnitude less than the peak single precision rate, SpMV performance is generally bandwidth-limited, and therefore does not approach the limits of floating point throughput. Indeed, DIA double precision (uncached) performance in the 27-point example is 45.9% that of the single precision result, which nearly matches the ratio of bytes per FLOP of the two kernels (50%) as listed in Table 1. For the ELL kernel, a similar analysis suggests a relative performance of 60.0% (double precision to single precision) which again agrees with the 59.2% observed.

Since the remaining kernels are not immediately bandwidth-limited, we cannot expect a direct correspondence between relative performance and computational intensity. In particular, the CSR (scalar) kernel retains 92.5% of its single precision performance, while computational intensity would suggest a 60.0% figure. This anomaly is explained by the fact that uncoalesced double-word memory accesses are inherently more efficient than uncoalesced single-word accesses on a memory bandwidth basis (cf. Figure 1).

4.3 Unstructured Matrices

Our unstructured matrix performance study considers the same 14 matrix corpus (cf. Table 3) used by Williams et al. [23] in their multicore benchmarking study. Figure 13 reports single and double precision performance. In contrast to the structured case, the unstructured performance results are varied, with no single kernel outperforming all others. The HYB format achieves the highest...
Performance (N. Bell and M. Garland, 2009)

- CSR is commonly used but has low performance

![Graph showing performance comparison of different formats](image-url)

Figure 12: Bandwidth results for structured matrices with single precision values.

![Graph showing performance comparison of different formats](image-url)

Figure 13: SpMV throughput on unstructured matrices.
Performance (N. Bell and M. Garland, 2009)

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

![Graph showing performance comparison between different matrix formats (COO, CSR, DIA, ELL) for structured matrices. The graph includes data points for different precision levels (single and double). The x-axis represents the matrix types (Laplacian 3pt, Laplacian 5pt, Laplacian 7pt, Laplacian 9pt, Laplacian 27pt), and the y-axis represents the throughput in GB/s. The graph indicates varying performance across different formats and matrix types.](image)
- 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
__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
__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**

\[ v = P \mathbf{v}^c \Rightarrow v_i = v_j^c, \text{ if } i \in G_j. \]
Prolongation and Restriction

**Prolongation**

\[ v = P v^c \Rightarrow v_i = v_j^c, \text{ if } i \in G_j. \]

```c
__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**

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

```c
__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 \[ A_c = P^T A P \]

**Triple matrices multiplication**

In general, triple matrices multiplication consists of 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. \((1.65x)\)
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

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

- 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
- 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

Value Title

<table>
<thead>
<tr>
<th>2048*2048</th>
<th>1024*1024</th>
<th>2048*1024</th>
<th>1024*1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.11</td>
<td>0.16</td>
<td>0.35</td>
<td>0.13</td>
</tr>
<tr>
<td>1.6</td>
<td>3.98</td>
<td>0.98</td>
<td>0.63</td>
</tr>
<tr>
<td>2.75</td>
<td>6.03</td>
<td>1.49</td>
<td>0.69</td>
</tr>
<tr>
<td>3.28</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6.03</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

CPU(i7) | CUSP | NEW

2048*2048    1024*1024

CPU(i7) | CUSP | NEW

Wednesday, March 20, 13
Poisson Problem on Uniform Grid

Setup: 28.2× CPU, 21.3× CUSP
Solve: 5.9× CPU, 3.2× CUSP
Total: 10.2× CPU, 6.7× CUSP
Poisson Problem on Quasi-uniform Grid
Poisson Problem on Quasi-uniform Grid

Wednesday, March 20, 13
Poisson Problem on Quasi-uniform Grid

**Setup**
- 0.16M: 0.49
- 1.4M: 1.09 / 0.82
- 5.7M: 1.52

**Solve**
- 0.16M: 0.49
- 1.4M: 3.65 / 4.11
- 5.7M: 1.52

**Total**: 3.7× CUSP

**Setup**: 6.8× CUSP

**Solve**: 2.2× CUSP

Wednesday, March 20, 13
Poisson Problem on Shape-regular Grid

Setup

Solve
Poisson Problem on Shape-regular Grid

Setup

Solve

Wednesday, March 20, 13
Poisson Problem on Shape-regular Grid

Setup: 6.4× CUSP
Solve: 1.2× CUSP
Total: 1.9× CUSP

Wednesday, March 20, 13
Poisson Problem on Disk Grid
Poisson Problem on Disk Grid

Setup vs. Solve

- 0.30: 1.65
- 2.9M: 2.19, 1.68
- 11.5M: 1.37, 7.21

Wednesday, March 20, 13
Poisson Problem on Disk Grid

Total: 2.1× CUSP
Heat Transfer Problem Problem on a cube
Heat Transfer Problem Problem on a cube

Wednesday, March 20, 13
Heat Transfer Problem Problem on a cube

Total: 2.4× CUSP
Heat Transfer Problem on Quasi-uniform Grid
Heat Transfer Problem on Quasi-uniform Grid
Heat Transfer Problem on Quasi-uniform Grid

Setup

Solve

Total: 2.8× CUSP
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

X. Hu, P. Vassilevski, and J. Xu, Comparative Convergence Analysis of Nonlinear AMLI-cycle Multigrid, accepted by SINUM, 2012
Thank you

Acknowledgements: NVIVIA Academic Partnership Award