A Strategy for Optimal Solvers in PyLith

Matthew Knepley\textsuperscript{1}  Peter Brune\textsuperscript{2}

\textsuperscript{1} Mathematics and Computer Science Division
Argonne National Laboratory

\textsuperscript{2} Computer Science Department
University of Chicago

AMR Tutorial Workshop
Boulder, CO
“A very popular error – having the courage of one’s convictions: Rather it is a matter of having the courage for an attack upon one’s convictions.”

— Friedrich Wilhelm Nietzsche
Outline

1. Why Optimal Solvers?
2. Multigrid for Unstructured Meshes
3. Coupled Problems
4. Actual Work
The more powerful the computer, the greater the importance of optimality.

Example:
- Suppose \(\text{Alg}_1\) solves a problem in time \(CN^2\), where \(N\) is the input size.
- Suppose \(\text{Alg}_2\) solves the same problem in time \(CN\).
- Suppose \(\text{Alg}_1\) and \(\text{Alg}_2\) are able to use 10,000 processors.

In constant time compared to serial,
- \(\text{Alg}_1\) can run a problem 100X larger.
- \(\text{Alg}_2\) can run a problem 10,000X larger.

Alternatively, filling the machine’s memory,
- \(\text{Alg}_1\) requires 100X time.
- \(\text{Alg}_2\) runs in constant time.
Convergence to $||r|| < 10^{-9}||b||$ using GMRES(30)/ILU

<table>
<thead>
<tr>
<th>Elements</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>10</td>
</tr>
<tr>
<td>256</td>
<td>17</td>
</tr>
<tr>
<td>512</td>
<td>24</td>
</tr>
<tr>
<td>1024</td>
<td>34</td>
</tr>
<tr>
<td>2048</td>
<td>67</td>
</tr>
<tr>
<td>4096</td>
<td>116</td>
</tr>
<tr>
<td>8192</td>
<td>167</td>
</tr>
<tr>
<td>16384</td>
<td>329</td>
</tr>
<tr>
<td>32768</td>
<td>558</td>
</tr>
<tr>
<td>65536</td>
<td>920</td>
</tr>
<tr>
<td>131072</td>
<td>1730</td>
</tr>
</tbody>
</table>
Linear Convergence

Convergence to $\|r\| < 10^{-9}\|b\|$ using GMRES(30)/MG

<table>
<thead>
<tr>
<th>Elements</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>5</td>
</tr>
<tr>
<td>256</td>
<td>7</td>
</tr>
<tr>
<td>512</td>
<td>6</td>
</tr>
<tr>
<td>1024</td>
<td>7</td>
</tr>
<tr>
<td>2048</td>
<td>6</td>
</tr>
<tr>
<td>4096</td>
<td>7</td>
</tr>
<tr>
<td>8192</td>
<td>6</td>
</tr>
<tr>
<td>16384</td>
<td>7</td>
</tr>
<tr>
<td>32768</td>
<td>6</td>
</tr>
<tr>
<td>65536</td>
<td>7</td>
</tr>
<tr>
<td>131072</td>
<td>6</td>
</tr>
</tbody>
</table>
Outline

1. Why Optimal Solvers?

2. Multigrid for Unstructured Meshes

3. Coupled Problems

4. Actual Work
Why not use AMG?
Why not use AMG?

- Of course we will try AMG
Why not use AMG?

- Of course we will try AMG
  - BoomerAMG, ML, SAMG, ASA
Why not use AMG?

- Of course we will try AMG
  - BoomerAMG, ML, SAMG, ASA
- Problems with vector character
Why not use AMG?

- Of course we will try AMG
  - BoomerAMG, ML, SAMG, ASA
- Problems with vector character
- Geometric aspects to the problem
Why not use AMG?

- Of course we will try AMG
  - BoomerAMG, ML, SAMG, ASA
- Problems with vector character
- Geometric aspects to the problem
  - Material property variation
Why not use AMG?

- Of course we will try AMG
  - BoomerAMG, ML, SAMG, ASA
- Problems with vector character
- Geometric aspects to the problem
  - Material property variation
  - Faults
Users want to control the mesh

Developed efficient, topological coarsening
  - Miller, Talmor, Teng algorithm

Provably well-shaped hierarchy
Simple Coarsening

1. Compute a spacing function $f$ for the mesh (Koebe)
Simple Coarsening

1. Compute a **spacing function** $f$ for the mesh (Koebe)
2. Scale $f$ by a factor $C > 1$
Simple Coarsening

1. Compute a spacing function $f$ for the mesh (Koebe)
2. Scale $f$ by a factor $C > 1$
3. Choose a maximal independent set of vertices for new $f$
Simple Coarsening

1. Compute a \textit{spacing function} $f$ for the mesh (Koebe)
2. Scale $f$ by a factor $C > 1$
3. Choose a maximal independent set of vertices for new $f$
4. Retriangulate
Caveats

1. Must generate coarsest grid in hierarchy first
Multigrid for Unstructured Meshes

Miller-Talmor-Teng Algorithm

Caveats

1. Must generate coarsest grid in hierarchy first
2. Must choose boundary vertices first (and protect boundary)
Caveats

1. Must generate coarsest grid in hierarchy first
2. Must choose boundary vertices first (and protect boundary)
3. Must account for boundary geometry
GMG Performance

For simple domains, everything works as expected:
Linear solver iterates are constant as system size increases:
GMG Performance

For simple domains, everything works as expected:

Work to build the preconditioner is constant as system size increases:
Reentrant Problems

- Reentrant corners need nonuniform refinement to maintain accuracy.
- Coarsening preserves accuracy in MG without user intervention.
Reentrant Problems

- Reentrant corners need nonuniform refinement to maintain accuracy.
- Coarsening preserves accuracy in MG without user intervention.

![Reentrant Corner Error Plot](image-url)
Exact Solution for reentrant problem: \( u(x, y) = r^{\frac{2}{3}} \sin\left(\frac{2}{3} \theta\right) \)
Reentrant Problems

Exact Solution for reentrant problem: \( u(x, y) = r^\frac{2}{3} \sin(\frac{2}{3} \theta) \)
Linear solver iterates are constant as system size increases:
GMG Performance

Work to build the preconditioner is constant as system size increases:

![Comparison Performance Graph]

- Pacman
- L

Mesh Size (Vertices)

Total Error

Comparison Performance
Outline

1. Why Optimal Solvers?
2. Multigrid for Unstructured Meshes
3. Coupled Problems
4. Actual Work
We would like to couple
- dynamic rupture
- quasi-static relaxation
We would like to couple

- dynamic rupture
- quasi-static relaxation

However, we must cope with

- different length scales
- different time scales
- solving full algebraic system
We assume that

- we can produce adequate meshes for each subproblem
- these meshes will change slowly during the simulation
- each subproblem admits an efficient, multilevel solver
Length Scales

We assume that

- we can produce adequate meshes for each subproblem
- these meshes will change slowly during the simulation
- each subproblem admits an efficient, multilevel solver

Our proposed solution is then

1. Create a single, adapted mesh
We assume that

- we can produce adequate meshes for each subproblem
- these meshes will change slowly during the simulation
- each subproblem admits an efficient, multilevel solver

Our proposed solution is then

1. Create a single, adapted mesh
2. For multiple coarse hierarchies
   - Can quickly coarsen where solution is smooth
Coupled Problems

Length Scales

We assume that

- we can produce adequate meshes for each subproblem
- these meshes will change slowly during the simulation
- each subproblem admits an efficient, multilevel solver

Our proposed solution is then

1. Create a single, adapted mesh
2. For *multiple* coarse hierarchies
   - Can quickly coarsen where solution is smooth
3. Solve each subproblem on its own hierarchy
We assume that

- we can separate time scales \textit{a priori}
- change slowly over the simulation
We assume that

- we can separate time scales \textit{a priori}
- change slowly over the simulation

Our proposed solution is then

1. to formulate the problem as spacetime FEM
We assume that

- we can separate time scales \textit{a priori}
- change slowly over the simulation

Our proposed solution is then

1. to formulate the problem as spacetime FEM
2. use Multi-Adaptive Galerkin timestepping

\textit{Multi-Adaptive Galerkin Methods for ODEs I},
We assume that

- each subproblem has an efficient, scalable solver
We assume that
  - each subproblem has an efficient, scalable solver

Our solution is then
  - use Jacobian-free Newton
We assume that
- each subproblem has an efficient, scalable solver

Our solution is then
- use Jacobian-free Newton
- use *physics-based* (Schwartz) preconditioning
We assume that
- each subproblem has an efficient, scalable solver

Our solution is then
- use Jacobian-free Newton
- use *physics-based* (Schwartz) preconditioning
- could also do this with FAS
Algebraic Solution

We assume that

- each subproblem has an efficient, scalable solver

Our solution is then

- use Jacobian-free Newton
- use *physics-based* (Schwartz) preconditioning
- could also do this with FAS

Outline

1. Why Optimal Solvers?
2. Multigrid for Unstructured Meshes
3. Coupled Problems
4. Actual Work
- New boundary protection scheme
  - Better preservation of embedded boundaries

- Replace mesh generator call with flips
  - Do not need point insertion
PyLith Work

- Replace KSP solver with DMMG
  - Enables GMG
  - Enables nonlinear solver

- Fix MG interpolation for cohesive elements
  - Collocation
  - Weighted average in a ball