Multilevel Stokes flow solvers
Adapting to heterogeneity and rheology

Jed Brown
Mathematics and Computer Science Division, Argonne National Laboratory

CIG Mantle/Lithosphere 2012-07-30
Intent of this talk

- observation: solver scalability is the bottleneck at scale
- "black box" solvers are not sustainable
  - optimal solvers must accurately handle all scales
  - optimality is crucial for large-scale problems
  - hardware puts up a spirited fight to abstraction
- introduce multilevel solver concepts
- outline ingredients that discretizations can provide to solvers
- discuss algorithmic trade-offs
- current state of solver software and what we are working on
Outline

Introduction

Multiscale Toolbox
   Coarse grids
   Smoothing

Software and performance
   Coupling software
   Performance considerations
Outline

Introduction

Multiscale Toolbox
  Coarse grids
  Smoothing

Software and performance
  Coupling software
  Performance considerations
It’s *all* about algorithms (at the petascale)

- **Given, for example:**
  - a “physics” phase that scales as $O(N)$
  - a “solver” phase that scales as $O(N^{3/2})$
  - computation is almost all solver after several doublings

- **Most applications groups have not yet “felt” this curve in their gut**
  - as users actually get into queues with more than 4K processors, this will change

*(c/o David Keyes)*
Challenges for elliptic solvers

▶ multiscale material coefficients
  ▶ long, thin high viscosity: transmit stresses long distances
  ▶ “jelly sandwich”: release long-range stresses locally

▶ nonlinearity
  ▶ plasticity: creates “jelly sandwich”
  ▶ Newton linearization produces local anisotropy
  ▶ heating: localization
  ▶ coupling to other physical processes

▶ multilevel methods
  ▶ need accurate coarse grids
  ▶ need effective smoothers
Multigrid separates scales, feedback between scales.

*The Multigrid V-cycle*

- **Smoothing (relaxation)**
- **Restriction**
- **Prolongation (interpolation)**

*Error on the fine grid*

*Error approximated on a smaller coarse grid*
The Great Solver Schism: Monolithic or Split?

Monolithic

- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid

X Need to understand local spectral and compatibility properties of the coupled system

- Preferred data structures depend on which method is used.
- Interplay with geometric multigrid.

Split

- Physics-split Schwarz (based on relaxation)
- Physics-split Schur (based on factorization)
  - approximate commutators
  - SIMPLE, PCD, LSC
  - segregated smoothers
  - Augmented Lagrangian
  - “parabolization” for stiff waves

X Need to understand global coupling strengths
Outline

Introduction

Multiscale Toolbox
  Coarse grids
  Smoothing

Software and performance
  Coupling software
  Performance considerations
Three schools of thought

- **Multigrid (Brandt, Hackbusch, ...)**
  - originally for resolved/asymptotic spatial discretizations
  - textbook: reach discretization error in one F-cycle
  - matrix-light/free, good for bandwidth
  - FAS well-developed for nonlinear problems

- **Multilevel Domain Decomposition (Mandel, Dohrmann, Widlund)**
  - leverage direct subdomain solvers, minimize communication
  - rapid coarsening $\kappa(P^{-1}A) \sim (1 + \log \frac{H}{h})^{2(L-1)}$
  - often formulated only as two-level methods
  - typically with domain-conforming coefficients
  - lightly developed for nonlinear (e.g. ASPIN [Cai and Keyes])

- **Multiscale Finite Elements (Babuska, Arbogast, ...)**
  - local preprocessing to construct coarse space
  - rarely/never revisit fine space
  - mostly restricted to linear problems
Computable Convergence Measures

- Prolongation $P : V_{\text{coarse}} \rightarrow V_{\text{fine}}$
- Restriction $R : V_{\text{fine}} \rightarrow V_{\text{coarse}}$
- $I - PR : V_{\text{fine}} \rightarrow V_{\text{fine}}$ removes part of vector visible in coarse space
- Error iteration matrix $I - M^{-1}A$, worst-case convergence factor is $\lambda_{\text{max}}$
- “Interpolation must be able to approximate an eigenvector with error bound proportional to the size of the associated eigenvalue.”

$$\max_x \|x\|(I - PR)S(I - PR) / \|x\|_A$$

- What can we do before we have prolongation $P$?
Compatible Relaxation

- Apply smoother subject to constraint $\hat{R}x = 0$
  1. $\tilde{x}_n = x_{n-1} + S_A^{-1}(r(x_{n-1}))$
  2. $x_n = \tilde{x}_n + S_R^{-1}(\hat{R}\tilde{x}_n)$

- Method to determine when coarse space is rich enough
- Slow to relax points/regions good candidates for coarse points/aggregates
- If subdomain solves used for smoothing, only interfaces are candidates

[Livne 2004]
Coarse basis functions

- $\|PRx\|_A + \|(I-PR)x\|_A \leq C \|x\|_A$
- “decompose any $x$ into parts without increasing energy much”
- near-null spaces must be represented exactly (partition of unity)
- number of rows of $R$ determined already, usually $P = R^T$
- energy minimization with specified support [Wan, Chan, Smith; Mandel, Brezina, Vanek]
- smoothed aggregation: $P_{\text{smooth}} = (I - \omega D^{-1}A)P_{\text{agg}}$
- classical AMG: each fine point processed independently
- domain decomposition/multiscale FEM: solve subdomain problems
Example: BDDC/FETI-DP coarse basis function

- only low-order continuity between subdomains
- corrected by more technical subdomain smoother

[Mandel and Sousedik 2010]
Why I like subdomain problems

- subassembly avoids explicit matrix triple product $A_{coarse} \leftarrow P^T A_{fine} P$
- can update the coarse operator locally (e.g. local nonlinearity)
- need not assemble entire fine grid operator
- can coarsen very rapidly (at least in smooth regions)
- lower communication setup phase

[Arbogast 2011]
Complication for saddle point problems

\[
\begin{pmatrix}
A & B^T \\
B & 0
\end{pmatrix}
\]

- want uniform stability for coarse problem
  - respect inf-sup condition, similar to fine grid
- want exact representation of volumetric mode
  - i.e. we can’t cheat on conservation while upscaling
- to be rigorous, we need to evaluate face integrals
  - self-similar coarse discretizations are attractive
- heuristic algebraic coarsening also possible [Adams 2004]
Nonlinear problems

- matrix-based smoothers require global linearization
- nonlinearity often more efficiently resolved locally
- nonlinear additive or multiplicative Schwarz
- nonlinear/matrix-free is good if

\[ C = \frac{(\text{cost to evaluate residual at one point}) \cdot N}{(\text{cost of global residual})} \sim 1 \]

- finite difference: \( C < 2 \)
- finite volume: \( C \sim 2 \), depends on reconstruction
- finite element: \( C \sim \text{number of vertices per cell} \)
- larger block smoothers help reduce \( C \)
The equation \[
\begin{pmatrix}
A & B^T \\
B & 0
\end{pmatrix}
\]
represents the structure of a saddle point system. The text discusses the following points:

- Pressure has no self-coupling.
- Pressure error modes are not spectrally separated.
- Approaches:
  - Block smoothers (Vanka)
  - Splitting with approximate Schur complement
  - Amplify fine-grid modes
Vanka block smoothers

- solve pressure-centered cell problems (better for discontinuous pressure)
- robust convergence factor $\sim 0.3$ if coarse grids are accurate
- 1D energy minimizing interpolants easy and effective
- can use assembled sparse matrices, but more efficient without
Changing Associativity: Distributive Smoothing

\[ PAx = Pb \quad APy = b, \quad x = Py \]

- Normal Preconditioning: make \( PA \) or \( AP \) well-conditioned
- Alternative: amplify high-frequency modes
  - Multigrid smoothers only need to relax high-frequency modes
  - Easier to do when spectrally separated: \( h \)-ellipticity
    - pointwise smoothers (Gauss-Seidel) and polynomial/multistage methods
- Mechanics: form the product \( PA \) or \( AP \) and apply “normal” method
- Example (Stokes)

\[
A \sim \begin{pmatrix} -\nabla^2 & \nabla \\ \nabla \cdot & 0 \end{pmatrix} \quad P \sim \begin{pmatrix} 1 & -\nabla \\ 0 & -\nabla^2 \end{pmatrix} \quad AP \sim \begin{pmatrix} -\nabla^2 & "0" \\ \nabla \cdot & -\nabla^2 \end{pmatrix}
\]

- Convergence factor 0.32 (as good as Laplace) for smooth problems
Coupled MG for Stokes, split smoothers

\[ J = \begin{pmatrix} A & B^T \\ B & C \end{pmatrix} \]

\[ P_{\text{smooth}} = \begin{pmatrix} A_{\text{SOR}} & 0 \\ B & M \end{pmatrix} \]

- `pc_type mg` - `pc_mg_levels 5` - `pc_mg_galerkin`
- `mg_levels_pc_type fieldsplit`
- `mg_levels_pc_fieldsplit_block_size 3`
- `mg_levels_pc_fieldsplit_0_fields 0,1`
- `mg_levels_pc_fieldsplit_1_fields 2`
- `mg_levels_fieldsplit_0_pc_type sor`
Outline

Introduction

Multiscale Toolbox
  Coarse grids
  Smoothing

Software and performance
  Coupling software
  Performance considerations
Multi-physics coupling in PETSc

- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting
Multi-physics coupling in PETSc

- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting
Multi-physics coupling in PETSc

- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting
Multi-physics coupling in PETSc

- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting
Multi-physics coupling in PETSc

- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format for each physics (e.g. symmetric block size 3)
- matrix-free anywhere
- multiple levels of nesting
Splitting for Multiphysics

\[
\begin{bmatrix}
A & B \\
C & D
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix} =
\begin{bmatrix}
f \\
g
\end{bmatrix}
\]

- **Relaxation:** `-pc_fieldsplit_type`
  - [additive, multiplicative, symmetric_multiplicative]

\[
\begin{bmatrix}
A & D \\
C & D
\end{bmatrix}^{-1} \begin{bmatrix}
A & D
\end{bmatrix}^{-1} \begin{bmatrix}
A & 1
\end{bmatrix}^{-1} \left(1 - \begin{bmatrix}
A & B \\
1 & C & D
\end{bmatrix}^{-1}\right)
\]

- Gauss-Seidel inspired, works when fields are loosely coupled
- **Factorization:** `-pc_fieldsplit_type schur`

\[
\begin{bmatrix}
A & B \\
S & 1
\end{bmatrix}^{-1} \begin{bmatrix}
1 & C A^{-1} \\
CA^{-1} & 1
\end{bmatrix}^{-1}, \quad S = D - CA^{-1}B
\]

- robust (exact factorization), can often drop lower block
- how to precondition $S$ which is usually dense?
  - interpret as differential operators, use approximate commutators
Work in Split Local space, matrix data structures reside in any space.
Multiphysics Assembly Code: Jacobians

FormJacobian_Coupled(SNES snes, Vec X, Mat J, Mat B, ...) {
    // Access components as for residuals
    MatGetLocalSubMatrix(B, is[0], is[0], &Buu);
    MatGetLocalSubMatrix(B, is[0], is[1], &Buk);
    MatGetLocalSubMatrix(B, is[1], is[0], &Bku);
    MatGetLocalSubMatrix(B, is[1], is[1], &Bkk);
    FormJacobianLocal_U(user, &infou, u, k, Buu); // single physics
    FormJacobianLocal_UK(user, &infou, &infok, u, k, Buk); // coupling
    FormJacobianLocal_KU(user, &infou, &infok, u, k, Bku); // coupling
    FormJacobianLocal_K(user, &infok, u, k, Bkk); // single physics
    MatRestoreLocalSubMatrix(B, is[0], is[0], &Buu);
    // More restores

    ▶ Assembly code is independent of matrix format
    ▶ Single-physics code is used unmodified for coupled problem
    ▶ No-copy fieldsplit:
      -pack_dm_mat_type nest -pc_type fieldsplit
    ▶ Coupled direct solve:
      -pack_dm_mat_type aij -pc_type lu -pc_factor_mat_solver_package mumps
Quasi-Newton revisited: ameliorating setup costs

- **Newton-Krylov with analytic Jacobian**

<table>
<thead>
<tr>
<th>Lag</th>
<th>FunctionEval</th>
<th>JacobianEval</th>
<th>PCSetUp</th>
<th>PCApply</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bt</td>
<td>12</td>
<td>8</td>
<td>8</td>
<td>31</td>
</tr>
<tr>
<td>1 cp</td>
<td>31</td>
<td>6</td>
<td>6</td>
<td>24</td>
</tr>
<tr>
<td>2 bt</td>
<td>—</td>
<td>— diverged</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 cp</td>
<td>41</td>
<td>4</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>3 cp</td>
<td>50</td>
<td>4</td>
<td>4</td>
<td>44</td>
</tr>
</tbody>
</table>

- **Jacobian-free Newton-Krylov with lagged preconditioner**

<table>
<thead>
<tr>
<th>Lag</th>
<th>FunctionEval</th>
<th>JacobianEval</th>
<th>PCSetUp</th>
<th>PCApply</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bt</td>
<td>23</td>
<td>11</td>
<td>11</td>
<td>31</td>
</tr>
<tr>
<td>2 bt</td>
<td>48</td>
<td>4</td>
<td>4</td>
<td>36</td>
</tr>
<tr>
<td>3 bt</td>
<td>64</td>
<td>3</td>
<td>3</td>
<td>52</td>
</tr>
<tr>
<td>4 bt</td>
<td>87</td>
<td>3</td>
<td>3</td>
<td>75</td>
</tr>
</tbody>
</table>

- **Limited-memory Quasi-Newton/BFGS with lagged solve for $H_0$**

<table>
<thead>
<tr>
<th>Restart</th>
<th>$H_0$</th>
<th>FunctionEval</th>
<th>JacobianEval</th>
<th>PCSetUp</th>
<th>PCApply</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 cp</td>
<td>$10^{-5}$</td>
<td>17</td>
<td>4</td>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>1 cp</td>
<td>preonly</td>
<td>21</td>
<td>5</td>
<td>5</td>
<td>10</td>
</tr>
<tr>
<td>3 cp</td>
<td>$10^{-5}$</td>
<td>21</td>
<td>3</td>
<td>3</td>
<td>43</td>
</tr>
<tr>
<td>3 cp</td>
<td>preonly</td>
<td>23</td>
<td>3</td>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>6 cp</td>
<td>$10^{-5}$</td>
<td>29</td>
<td>2</td>
<td>2</td>
<td>60</td>
</tr>
<tr>
<td>6 cp</td>
<td>preonly</td>
<td>29</td>
<td>2</td>
<td>2</td>
<td>14</td>
</tr>
</tbody>
</table>

pseudo-plastic rheology

- snes_type qn
- snes_qn_scale_type jacobian
Performance of assembled versus unassembled

- High order Jacobian stored unassembled using coefficients at quadrature points, can use local AD
- Choose approximation order at run-time, independent for each field
- Precondition high order using assembled lowest order method
- Implementation > 70% of FPU peak, SpMV bandwidth wall < 4%
Coarse levels may not be cheaper than fine levels

- latency for longer-range communication outweighs smaller data
- very aggressive coarsening important to limit number of levels
- alternatives: additive multigrid, redundant coarse grids

[Gahvari, Schulz, Yang, Jordan, Gropp 2011]
Multilevel Solvers are a *Way of Life*

- ingredients that discretizations can provide
  - identify “fields”
  - topological coarsening, possibly for fields
  - near-null space information
  - “natural” subdomains
  - subdomain integration, face integration
  - element or subdomain assembly/matrix-free smoothing
- solver composition
  - most splitting methods accessible from command line
  - energy optimization for tentative coarse basis functions
  - algebraic form of distributive relaxation
  - generic assembly for large systems and components
  - working on flexible “library-assisted” nonlinear multigrid
  - adding support for interactive eigenanalysis