High Performance Implicit Solvers for Geodynamics

These slides:
http://59A2.org/files/20130110-CIGWebinar.pdf

Jed Brown

jedbrown@mcs.anl.gov

Mathematics and Computer Science Division, Argonne National Laboratory

CIG Webinar 2013-01-10

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

Stokes problems

Performance considerations

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

Stokes problems

Performance considerations

Why do we need solvers?

Definition (Stiffness)

A discretized PDE is stiff if the true physics propagates information much more than one grid cell over a time step length desirable for resolving transient dynamics.

Why do we need solvers?

Definition (Stiffness)

A discretized PDE is stiff if the true physics propagates information much more than one grid cell over a time step length desirable for resolving transient dynamics.

$$(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta D u + p 1) - \rho g = 0$$

$$\rho_t + \nabla\cdot\rho u = 0$$

Why do we need solvers?

Definition (Stiffness)

A discretized PDE is stiff if the true physics propagates information much more than one grid cell over a time step length desirable for resolving transient dynamics.

$$(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta D u + p 1) - \rho g = 0$$
$$\rho_t + \nabla\cdot\rho u = 0$$

1. Incompressibility: acoustic wave travels much faster than mantle or lithosphere time scale (anelastic; Mach number)

Why do we need solvers?

Definition (Stiffness)

A discretized PDE is stiff if the true physics propagates information much more than one grid cell over a time step length desirable for resolving transient dynamics.

$$(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta D u + p 1) - \rho g = 0$$
$$\rho_t + \nabla\cdot \rho u = 0$$

1. Incompressibility: acoustic wave travels much faster than mantle or lithosphere time scale (anelastic; Mach number)
2. Convection insignificant compared to viscosity (unrelated to stiffness; Reynolds number)

Why do we need solvers?

Definition (Stiffness)

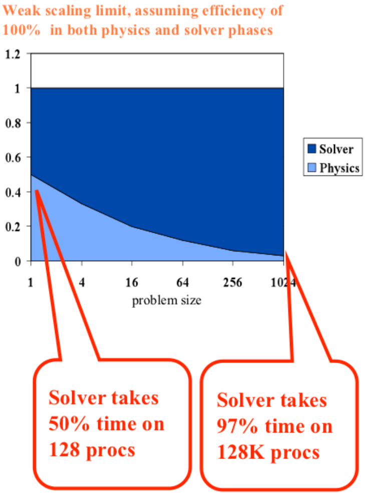
A discretized PDE is stiff if the true physics propagates information much more than one grid cell over a time step length desirable for resolving transient dynamics.

$$(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta D u + p 1) - \rho g = 0$$
$$\rho_t + \nabla\cdot \rho u = 0$$

1. Incompressibility: acoustic wave travels much faster than mantle or lithosphere time scale (anelastic; Mach number)
2. Convection insignificant compared to viscosity (unrelated to stiffness; Reynolds number)
3. Relaxation fast compared to dynamical time scale (depends on observational scale)

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



Weak scaling limit, assuming efficiency of 100% in both physics and solver phases

Solver takes 50% time on 128 procs

Solver takes 97% time on 128K procs

(c/o David Keyes)

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

Stokes problems

Performance considerations

Evaluating methods

- Performance of methods will depend on grid resolution and model parameters (regime and heterogeneity).
- A method is:
  - scalable (also “optimal”) if its performance is independent of resolution and parallelism
  - robust if its performance is (nearly) independent of model parameters
  - efficient if it solves the problem in a small multiple of the cost to evaluate the residual[1]

[1] We’ll settle for “as fast as the best known method”.

Evaluating methods

- Performance of methods will depend on grid resolution and model parameters (regime and heterogeneity).
- A method is:
  - scalable (also “optimal”) if its performance is independent of resolution and parallelism
  - robust if its performance is (nearly) independent of model parameters
  - efficient if it solves the problem in a small multiple of the cost to evaluate the residual[1]
- Linear problems typically arise from linearizing a nonlinear problem. This step is not necessary, but it is convenient for reusing software and for debugging.

[1] We’ll settle for “as fast as the best known method”.

Taxonomy of implicit solvers

Global linearization: Picard and Newton

- Linear solve “$J(u)w = -F(u)$”
  - (sparse) direct vs. iterative (Krylov) with preconditioning
  - classical relaxation (Jacobi, Gauss-Seidel), incomplete factorization (ILU)
  - domain decomposition and multigrid
- Globalization: “$u_{\text{next}} = u + \alpha w$”
  - Line search, trust region, continuation

Inherently nonlinear methods

- Nonlinear GMRES, Nonlinear CG (can use preconditioning)
- Nonlinear domain decomposition
- Nonlinear multigrid: Full Approximation Scheme (FAS)

Taxonomy of implicit solvers

Global linearization: Picard and Newton

- Linear solve "$J(u)w = -F(u)$"
  - (sparse) direct vs. iterative (Krylov) with preconditioning
  - classical relaxation (Jacobi, Gauss-Seidel), incomplete factorization (ILU)
  - domain decomposition and multigrid
- Globalization: "$u_{\text{next}} = u + \alpha w$"
  - Line search, trust region, continuation

Inherently nonlinear methods

- Nonlinear GMRES, Nonlinear CG (can use preconditioning)
- Nonlinear domain decomposition
- Nonlinear multigrid: Full Approximation Scheme (FAS)

- These methods can be **scalable**.

Taxonomy of implicit solvers

Global linearization: Picard and Newton

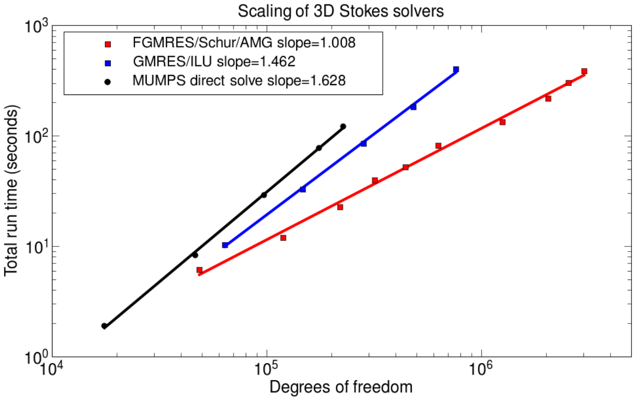
- Linear solve “$J(u)w = -F(u)$”
  - (sparse) direct vs. iterative (Krylov) with preconditioning
  - classical relaxation (Jacobi, Gauss-Seidel), incomplete factorization (ILU)
  - domain decomposition and multigrid
- Globalization: “$u_{\text{next}} = u + \alpha w$”
  - Line search, trust region, continuation

Inherently nonlinear methods

- Nonlinear GMRES, Nonlinear CG (can use preconditioning)
- Nonlinear domain decomposition
- Nonlinear multigrid: Full Approximation Scheme (FAS)

- How nonlinear are the scales? How expensive is setup?

What about direct linear solvers?



- By all means, start with a direct solver
- Direct solvers are robust, but not scalable
- **2D**: $\mathcal{O}(n^{1.5})$ flops, $\mathcal{O}(n \log n)$ memory.
- **3D**: $\mathcal{O}(n^2)$ flops, $\mathcal{O}(n^{4/3})$ memory
- We will focus on iterative linear solvers

Matrices

Definition (Matrix)

A matrix is a linear transformation between finite dimensional vector spaces.

Matrices

Definition (Matrix)

A matrix is a linear transformation between finite dimensional vector spaces.

Definition (Forming a matrix)

Forming or assembling a matrix means defining it's action in terms of entries (usually stored in a sparse format).

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

- These matrices are dense. Never form them.

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\varepsilon \to 0} \frac{F(x+\varepsilon y) - F(x)}{\varepsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

- These are not very sparse. Don't form them.

Important matrices

1. Sparse (e.g. discretization of a PDE operator)
2. Inverse of *anything* interesting $B = A^{-1}$
3. Jacobian of a nonlinear function $Jy = \lim_{\epsilon \to 0} \frac{F(x+\epsilon y)-F(x)}{\epsilon}$
4. Fourier transform $\mathscr{F}, \mathscr{F}^{-1}$
5. Other fast transforms, e.g. Fast Multipole Method
6. Low rank correction $B = A + uv^T$
7. Schur complement $S = D - CA^{-1}B$
8. Tensor product $A = \sum_e A_x^e \otimes A_y^e \otimes A_z^e$
9. Linearization of a few steps of an explicit integrator

- None of these matrices “have entries”

What can we do with a matrix that doesn't have entries?

Krylov solvers for $Ax = b$

- Krylov subspace: $\{b, Ab, A^2b, A^3b, \dots\}$
- Convergence rate depends on the spectral properties of the matrix
  - Existance of small polynomials $p_n(A) < \varepsilon$ where $p_n(0) = 1$.
  - condition number $\kappa(A) = \|A\| \left\|A^{-1}\right\| = \sigma_{\text{max}}/\sigma_{\text{min}}$
  - distribution of singular values, spectrum Λ, pseudospectrum Λ_ε
- For any popular Krylov method $\mathscr{K}$, there is a matrix of size m, such that $\mathscr{K}$ outperforms all other methods by a factor at least $\mathscr{O}(\sqrt{m})$ [Nachtigal et. al., 1992]

Typically...

- The action $y \leftarrow Ax$ can be computed in $\mathscr{O}(m)$
- Aside from matrix multiply, the n^{th} iteration requires at most $\mathscr{O}(mn)$

The $\mathfrak{p}$-Bratu equation

- 2-dimensional model problem

  $$-\nabla\cdot\big(|\nabla u|^{\mathfrak{p}-2}\nabla u\big) - \lambda e^u - f = 0, \qquad 1 \le \mathfrak{p} \le \infty, \quad \lambda < \lambda_{\text{crit}}(\mathfrak{p})$$

  Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.
- Regularized variant

  $$-\nabla\cdot(\eta\nabla u) - \lambda e^u - f = 0$$

  $$\eta(\gamma) = (\epsilon^2 + \gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u) = \frac{1}{2}|\nabla u|^2$$
- Jacobian

  $$J(u)w \sim -\nabla\cdot\big[(\eta 1 + \eta' \nabla u \otimes \nabla u)\nabla w\big] - \lambda e^u w$$

  $$\eta' = \frac{\mathfrak{p}-2}{2}\eta/(\epsilon^2 + \gamma)$$

  Interpretation: conductivity tensor flattened in direction ∇u
- Simple finite difference discretization in PETSc:

  ```
  $ cd petsc/src/snes/examples/tutorials/; make ex15
  ```

The $\mathfrak{p}$-Bratu equation

- 2-dimensional model problem

  $$-\nabla\cdot\left(|\nabla u|^{\mathfrak{p}-2}\nabla u\right)-\lambda e^u - f = 0, \qquad 1\le\mathfrak{p}\le\infty, \quad \lambda<\lambda_{\text{crit}}(\mathfrak{p})$$

  Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.
- Regularized variant

  $$-\nabla\cdot(\eta\nabla u)-\lambda e^u - f = 0$$

  $$\eta(\gamma)=(\epsilon^2+\gamma)^{\frac{\mathfrak{p}-2}{2}} \qquad \gamma(u)=\frac{1}{2}|\nabla u|^2$$
- Jacobian

  $$J(u)w \sim -\nabla\cdot\left[(\eta 1+\eta'\nabla u\otimes\nabla u)\nabla w\right]-\lambda e^u w$$

  $$\eta' = \frac{\mathfrak{p}-2}{2}\eta/(\epsilon^2+\gamma)$$

  Interpretation: conductivity tensor flattened in direction ∇u
- Simple finite difference discretization in PETSc:

  ```
  $ cd petsc/src/snes/examples/tutorials/; make ex15
  ```
- Step 1: Write the residual.

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ ./ex15 -da_refine 1 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 2 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 4 -snes_mf -snes_monitor -ksp_converged_reason`

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`

-
  ```
  $ ./ex15 -da_refine 1 -snes_mf -snes_monitor
  -ksp_converged_reason
  ```

  ```
  0 SNES Function norm 9.324361041196e-01
  Linear solve converged due to CONVERGED_RTOL iterations 7
  1 SNES Function norm 4.534365556764e-09

  CONVERGED_FNORM_RELATIVE Number of nonlinear iterations = 1
  ```

-
  ```
  $ ./pbratu -da_refine 2 -snes_mf -snes_monitor
  -ksp_converged_reason
  ```

-
  ```
  $ ./pbratu -da_refine 3 -snes_mf -snes_monitor
  -ksp_converged_reason
  ```

-
  ```
  $ ./pbratu -da_refine 4 -snes_mf -snes_monitor
  -ksp_converged_reason
  ```

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ ./ex15 -da_refine 1 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 2 -snes_mf -snes_monitor -ksp_converged_reason`

  ```
  0 SNES Function norm 5.363535697720e-01
  Linear solve converged due to CONVERGED_RTOL iterations 18
  1 SNES Function norm 1.276738526722e-06
  Linear solve converged due to CONVERGED_RTOL iterations 18
  2 SNES Function norm 1.263046904535e-11

  CONVERGED_FNORM_RELATIVE Number of nonlinear iterations = 2
  ```
- `$ ./pbratu -da_refine 3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 4 -snes_mf -snes_monitor -ksp_converged_reason`

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ ./ex15 -da_refine 1 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 2 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 3 -snes_mf -snes_monitor -ksp_converged_reason`

  ```
  0 SNES Function norm 2.820917170607e-01
  Linear solve converged due to CONVERGED_RTOL iterations 42
  1 SNES Function norm 2.782839451653e-06
  Linear solve converged due to CONVERGED_RTOL iterations 45
  2 SNES Function norm 2.682642095006e-11

  CONVERGED_FNORM_RELATIVE Number of nonlinear iterations = 2
  ```

- `$ ./pbratu -da_refine 4 -snes_mf -snes_monitor -ksp_converged_reason`

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ ./ex15 -da_refine 1 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 2 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 4 -snes_mf -snes_monitor -ksp_converged_reason`

```
0 SNES Function norm 1.441189193029e-01
Linear solve converged due to CONVERGED_RTOL iterations 101
1 SNES Function norm 1.409860069506e-06
Linear solve converged due to CONVERGED_RTOL iterations 154
2 SNES Function norm 1.390912345257e-11

CONVERGED_FNORM_RELATIVE Number of nonlinear iterations = 2
```

Step 1: Write the residual

- Start with $\mathfrak{p} = 2$ (standard Laplacian), define only residuals
- Matrix-free Jacobians, no preconditioning `-snes_mf`
- `$ ./ex15 -da_refine 1 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 2 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 3 -snes_mf -snes_monitor -ksp_converged_reason`
- `$ ./pbratu -da_refine 4 -snes_mf -snes_monitor -ksp_converged_reason`

  ```
  0 SNES Function norm 1.441189193029e-01
  Linear solve converged due to CONVERGED_RTOL iterations 101
  1 SNES Function norm 1.409860069506e-06
  Linear solve converged due to CONVERGED_RTOL iterations 154
  2 SNES Function norm 1.390912345257e-11

  CONVERGED_FNORM_RELATIVE Number of nonlinear iterations = 2
  ```

- The number of iterations is growing with grid refinement.

Experimenting with algorithms

- `-pc_type asm -sub_pc_type lu`
- `-pc_type gamg -pc_gamg_agg_nsmooths 1`
- `-jtype PICARD -pc_type lu`
- `-snes_mf_operator -jtype PICARD -pc_type ml`

-
  ```
  -snes_type ngmres -snes_ngmres_m 10
      -npc_snes_max_it 1 -npc_snes_type fas
  ```

Barriers

- Krylov method: $(\text{iteration count}) \sim \sqrt{\text{condition number}}$
- Elliptic ill-conditioning
  - $\kappa(A) \sim h^{-2}$ for second order elliptic problems
  - *Asymptotics* not improved for standard methods:
    `-pc_type jacobi`, `-pc_type sor`, `-pc_type ilu`
  - 1-level Domain Decomposition: $\kappa \sim H^{-2}\phi(H/h)$
    `-pc_type bjacobi`, `-pc_type asm`
  - Multilevel/multigrid: $\kappa \sim 1$
    `-pc_type gamg`, `-pc_type ml`, `-pc_type hypre`, `-pc_type mg`
- Heterogeneity
  - Conditioning proportional to maximum material contrast
  - In friendly circumstances, a local preconditioner restores $\sim h^{-2}$ ill-conditioning
  - Coarse approximations and subdomain transmission conditions become difficult
  - Fine grids necessary *because of* heterogeneity
  - Coarse grid must accurately represent long-range coupling

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

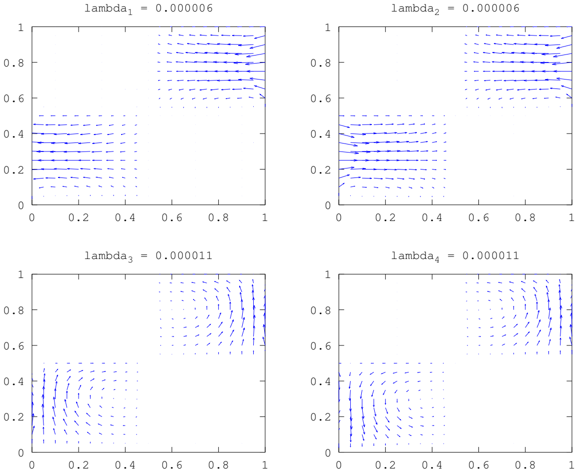
Stokes problems

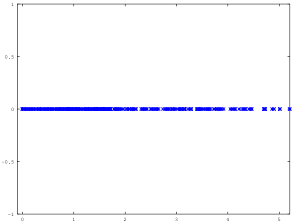
Performance considerations

Low energy modes of preconditioned operator $P^{-1}A$

2×2 checkerboard elasticity problem, Neumann condition on right boundary

- Original operator, stiff blocks don't move

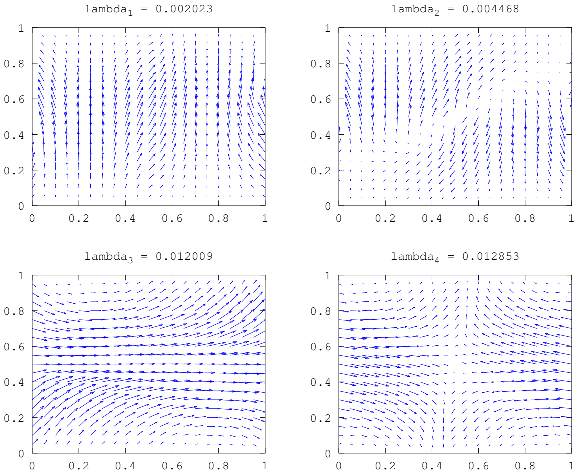


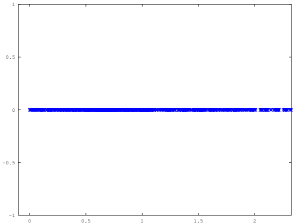


Low energy modes of preconditioned operator $P^{-1}A$

2×2 checkerboard elasticity problem, Neumann condition on right boundary

- With Jacobi preconditioning, balances stiffness, not $\mathcal{O}(\Delta x^2)$ elliptic ill-conditioning

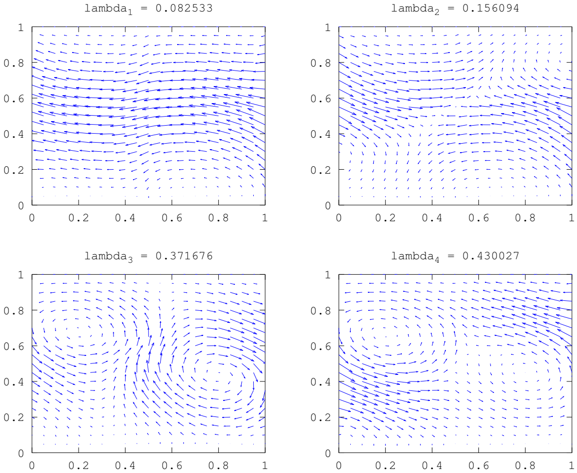


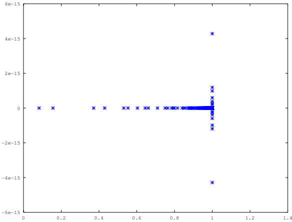


Low energy modes of preconditioned operator $P^{-1}A$

2×2 checkerboard elasticity problem, Neumann condition on right boundary

- With BoomerAMG preconditioning, does not find all rotations

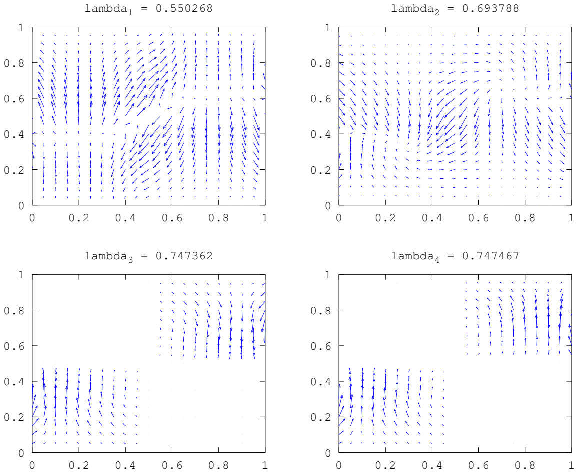


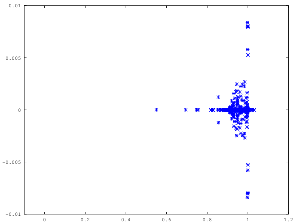


Low energy modes of preconditioned operator $P^{-1}A$

2×2 checkerboard elasticity problem, Neumann condition on right boundary

- With geometric MG, Galerkin coarse operators, Chebychev smoother

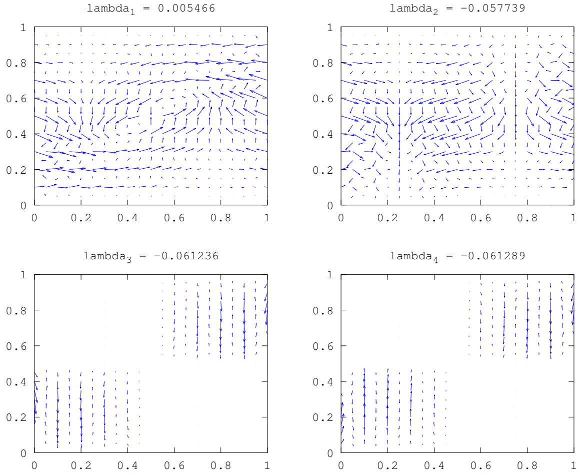


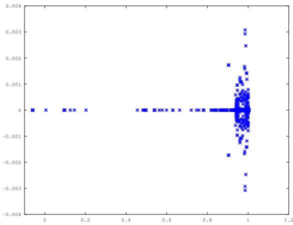


Low energy modes of preconditioned operator $P^{-1}A$

2×2 checkerboard elasticity problem, Neumann condition on right boundary

- With geometric MG, Galerkin coarse operators, unstable Chebychev smoother





Linear solver convergence problems[2]

- Watch the true residual `-ksp_monitor_true_residual`
- Make the problem small and create an environment to test rapidly
- Are boundary conditions correct?
  `-pc_type svd -pc_svd_monitor` and `-pc_type lu`
- Is the system singular? Known nullspace?
- Is the condition number reasonable?
  `-ksp_monitor_singular_value`
- Compare preconditioned residual to true residual (unstable preconditioner)
- Is GMRES restart a problem? `-ksp_gmres_restart 300`
- Is preconditioner nonlinear? `-ksp_type gcr`,
  `-ksp_type fgmres`
- Geometric multigrid with rediscretization: boundary condition scaling.

[2] `http://scicomp.stackexchange.com/questions/513`

Nonlinear solver convergence problems[3]

- Is the Jacobian assembled correctly?
  - `-snes_mf_operator -pc_type lu`
  - `-snes_type test` or `-snes_compare_explicit`
  - `-snes_mf_type ds`
- Is the linear system solved accurately enough?
- Does the linear system become singular?
- Is there a bug in residual evaluation?
- Is the residual function discontinuous?
- `-snes_linesearch_monitor`
- `./configure --with-precision=__float128`

[3] http://scicomp.stackexchange.com/questions/30

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

Stokes problems

Performance considerations

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

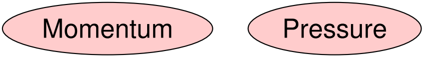
Split

- 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

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

Multi-physics coupling in PETSc

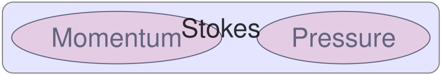


Momentum

Pressure

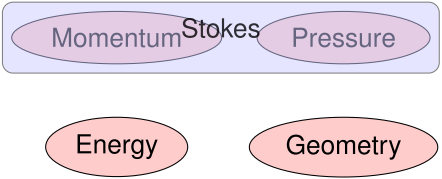
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc



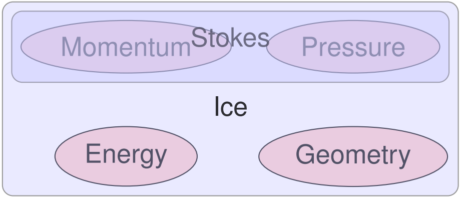
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc



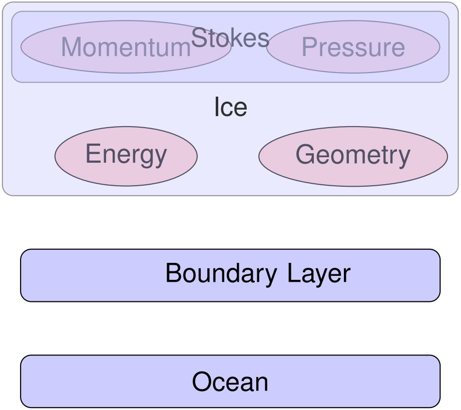
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format 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

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

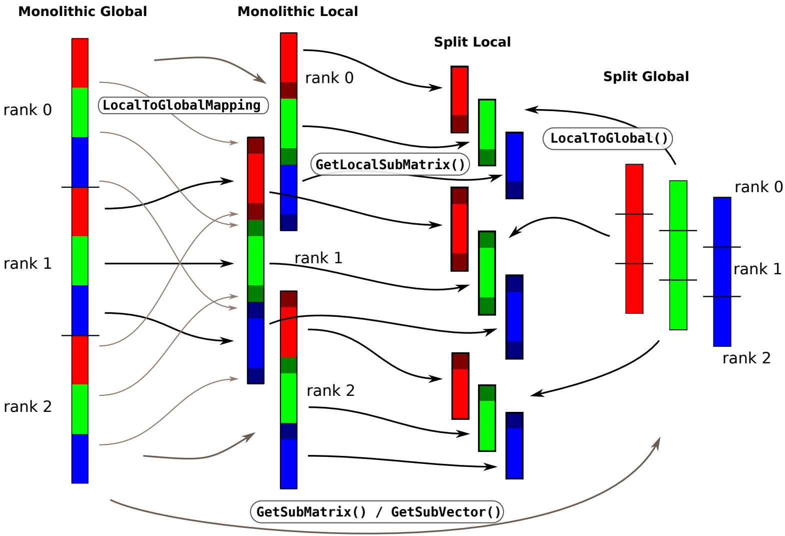
- Relaxation: `-pc_fieldsplit_type [additive,multiplicative,symmetric_multiplicative]`

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

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

$$\begin{bmatrix} A & B \\ & S \end{bmatrix}^{-1} \begin{bmatrix} 1 & \\ CA^{-1} & 1 \end{bmatrix}^{-1}, \qquad S = D - CA^{-1}B$$

  - 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
  ```

Outline

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

Stokes problems

Performance considerations

Stokes example

The common block preconditioners for Stokes require only options:

The Stokes System

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

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type additive

-fieldsplit_0_pc_type ml
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

Cohouet and Chabard, *Some fast 3D finite element solvers for the generalized Stokes problem*, 1988.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type
multiplicative
-fieldsplit_0_pc_type hypre
-fieldsplit_0_ksp_type preonly
-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

Elman, *Multigrid and Krylov subspace methods for the discrete Stokes equations*, 1994.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
 -pc_fieldsplit_schur_factorization_type diag
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

May and Moresi, *Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics*, 2008.

Olshanskii, Peters, and Reusken, *Uniform preconditioners for a parameter dependent saddle point problem with application to generalized Stokes interface equations*, 2006.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
 -pc_fieldsplit_schur_factorization_type lower
```

PC

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

May and Moresi, *Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics*, 2008.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type none
-fieldsplit_1_ksp_type minres
 -pc_fieldsplit_schur_factorization_type upper
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S} \end{pmatrix}$$

May and Moresi, *Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics*, 2008.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur

-fieldsplit_0_pc_type gamg
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type lsc
-fieldsplit_1_ksp_type minres
 -pc_fieldsplit_schur_factorization_type upper
```

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

May and Moresi, *Preconditioned iterative methods for Stokes flow problems arising in computational geodynamics*, 2008.

Kay, Loghin and Wathen, *A Preconditioner for the Steady-State N-S Equations*, 2002.
Elman, Howle, Shadid, Shuttleworth, and Tuminaro, *Block preconditioners based on approximate commutators*, 2006.

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_fieldsplit_type schur
 -pc_fieldsplit_schur_factorization_type full
```

PC

$$\begin{pmatrix} I & 0 \\ B^T A^{-1} & I \end{pmatrix} \begin{pmatrix} \hat{A} & 0 \\ 0 & \hat{S} \end{pmatrix} \begin{pmatrix} I & A^{-1}B \\ 0 & I \end{pmatrix}$$

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
```

System on each Coarse Level

$$R \begin{pmatrix} A & B \\ B^T & 0 \end{pmatrix} P$$

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type additive

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type jacobi
-mg_levels_fieldsplit_1_ksp_type preonly
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & I \end{pmatrix}$$

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type
multiplicative

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type jacobi
-mg_levels_fieldsplit_1_ksp_type preonly
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & I \end{pmatrix}$$

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & 0 \\ 0 & -\hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type diag
```

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & 0 \\ B^T & \hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type lower
```

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type none
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type upper
```

Stokes example

All block preconditioners can be *embedded* in MG using only options:

```
-pc_type mg -pc_mg_levels 5 -pc_mg_galerkin
-mg_levels_pc_type fieldsplit
-mg_levels_pc_fieldsplit_type schur

-mg_levels_fieldsplit_0_pc_type sor
-mg_levels_fieldsplit_0_ksp_type preonly

-mg_levels_fieldsplit_1_pc_type lsc
-mg_levels_fieldsplit_1_ksp_type minres
```

Smoother

PC

$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

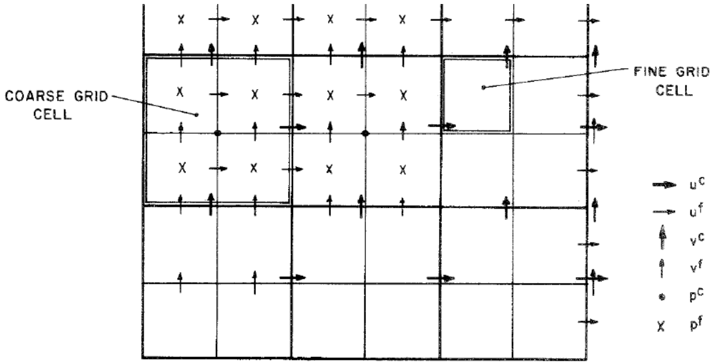
```
-mg_levels_pc_fieldsplit_schur_factorization_type upper
```

Smoothing for saddle point systems

$$\begin{pmatrix} A & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}$$

- pressure has no self-coupling
- pressure error modes not spectrally separated
- approaches
  - block smoothers (Vanka)
  - amplify fine-grid modes (distributive relaxation)
  - splitting with approximate Schur complement

Vanka block smoothers



- solve pressure-centered cell problems
  (better for discontinuous pressure)
- robust convergence factor ~ 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

$$PAx = Pb \qquad\qquad APy = b, \quad x = Py$$

- 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)

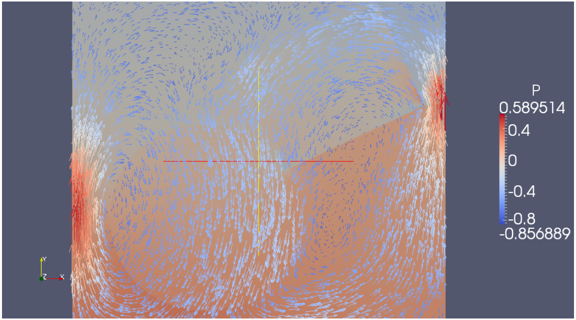
$$A \sim \begin{pmatrix} -\nabla^2 & \nabla \\ \nabla\cdot & 0 \end{pmatrix} \quad P \sim \begin{pmatrix} 1 & -\nabla \\ 0 & -\nabla^2 \end{pmatrix} \quad AP \sim \begin{pmatrix} -\nabla^2 & \text{“0”} \\ \nabla\cdot & -\nabla^2 \end{pmatrix}$$

- 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

Role of implicit solvers

Common methods and algorithmic barriers

Failure modes and troubleshooting

Coupling approaches

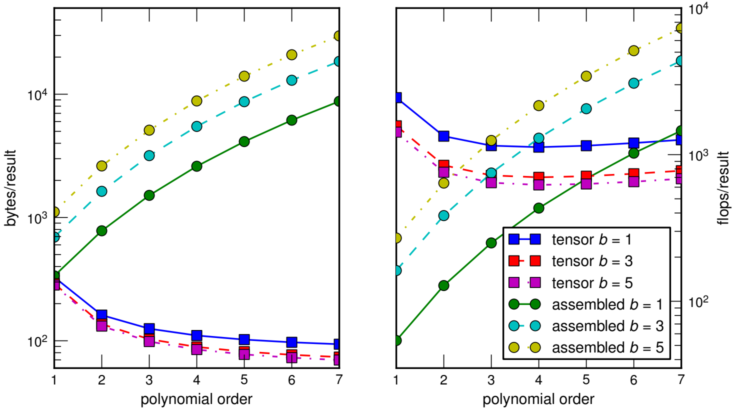
Stokes problems

Performance considerations

Profiling basics

- Get the math right
  - Choose an algorithm that gives robust iteration counts and really converges
- Look at where the time is spent
  - Run with `-log_summary` and look at events
  - `VecNorm`, `VecDot` measures latency
  - `MatMult` measures neighbor exchange and memory bandwidth
  - `PCSetUp` factorization, aggregation, matrix-matrix products, ...
  - `PCApply` V-cycles, triangular solves, ...
  - `KSPSolve` linear solve
  - `SNESFunctionEval` residual evaluation (user code)
  - `SNESJacobianEval` matrix assembly (user code)

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 $< 4\%$

Hardware Arithmetic Intensity

Operation	Arithmetic Intensity (flops/B)
Sparse matrix-vector product	1/6
Dense matrix-vector product	1/4
Unassembled matrix-vector product	≈ 8
High-order residual evaluation	> 5

Processor	BW (GB/s)	Peak (GF/s)	Balanced AI (F/B)
E5-2670 8-core	35	166	4.7
Magny Cours 16-core	49	281	5.7
Blue Gene/Q node	43	205	4.8
Tesla M2090	120	665	5.5
Kepler K20Xm	160	1310	8.2
Xeon Phi	150	1248	8.3

Quasi-Newton revisited: ameliorating setup costs

- Newton-Krylov with analytic Jacobian

Lag	FunctionEval	JacobianEval	PCSetUp	PCApply
1 bt	12	8	8	31
1 cp	31	6	6	24
2 bt		— diverged —		
2 cp	41	4	4	35
3 cp	50	4	4	44

pseudo-plastic rheology

```
-snes_type qn
-snes_qn_scale_type
jacobian
```

- Jacobian-free Newton-Krylov with lagged preconditioner

Lag	FunctionEval	JacobianEval	PCSetUp	PCApply
1 bt	23	11	11	31
2 bt	48	4	4	36
3 bt	64	3	3	52
4 bt	87	3	3	75

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

Restart	H_0	FunctionEval	JacobianEval	PCSetUp	PCApply
1 cp	10^{-5}	17	4	4	35
1 cp	preonly	21	5	5	10
3 cp	10^{-5}	21	3	3	43
3 cp	preonly	23	3	3	11
6 cp	10^{-5}	29	2	2	60
6 cp	preonly	29	2	2	14

Network latency

`MPI_Allreduce` is slow at large scale

- True on many machines, not on Blue Gene ($\sim 100\,\mu s$)
- Bottleneck for Krylov methods
- Pipelining allows overlap, uses `MPI_Iallreduce` from MPI-3
  `-ksp_type pgmres`, `-ksp_type pipecg`, `-ksp_type pipecr`

Coarse grid solves for multigrid

- Need to restrict active processor set
- Coarse levels have similar cost to finer levels
- Aggressive coarsening more important than tight iteration count
- Additive multigrid possible, but less robust

Outlook

- PETSc `http://mcs.anl.gov/petsc`
- Trilinos `http://trilinos.sandia.gov`
- Think about solution algorithms when designing discretization
- Learn how to evaluate solver quality and experiment
- Expect the best method to change with problem instance and machine

Contact

- `petsc-maint@mcs.anl.gov`
- `http://lists.mcs.anl.gov/pipermail/petsc-users/`