Composable multilevel solvers in PETSc

Jed Brown[1],
Mark Adams[2], Peter Brune[1], Matt Knepley[3], Dave May[4],
Barry Smith[1]

[1]Mathematics and Computer Science Division, Argonne National Laboratory
[2]Columbia University
[3]Computation Institute, University of Chicago
[4]ETH Zürich

Oak Ridge 2012-11-14

Multiphysics problems

Examples

- Saddle-point problems (e.g. incompressibility, contact)
- Stiff waves (e.g. low-Mach combustion)
- Mixed type (e.g. radiation hydrodynamics, ALE free-surface flows)
- Multi-domain problems (e.g. fluid-structure interaction)
- Full space PDE-constrained optimization

Software/algorithmic considerations

- Separate groups develop different “physics” components
- Do not know a priori which methods will have good algorithmic properties
- Achieving high throughput is more complicated
- Multiple time and/or spatial scales
  - Splitting methods are delicate, often not in asymptotic regime
  - Strongest nonlinearities usually non-stiff: prefer explicit for TVD limiters/shocks

The Great Solver Schism: Monolithic or Split?

Monolithic

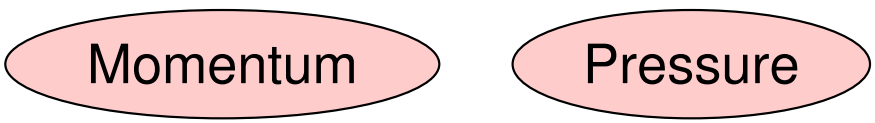
- Direct solvers
- Coupled Schwarz
- Coupled Neumann-Neumann (need unassembled matrices)
- Coupled multigrid
- X Need to understand local spectral and compatibility properties of the coupled system

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

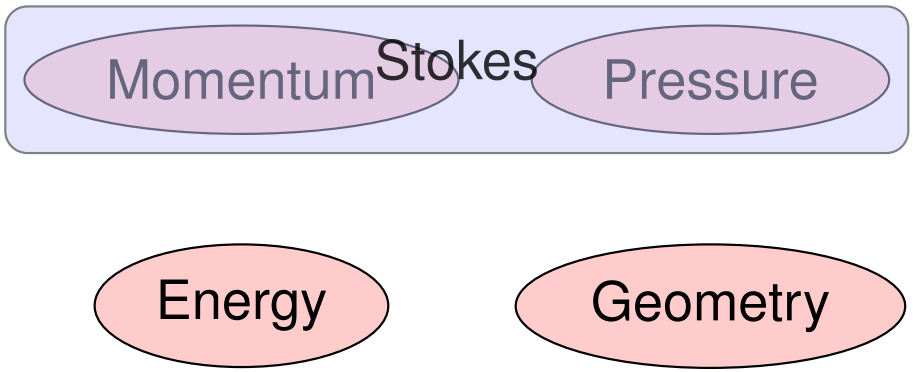


- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc

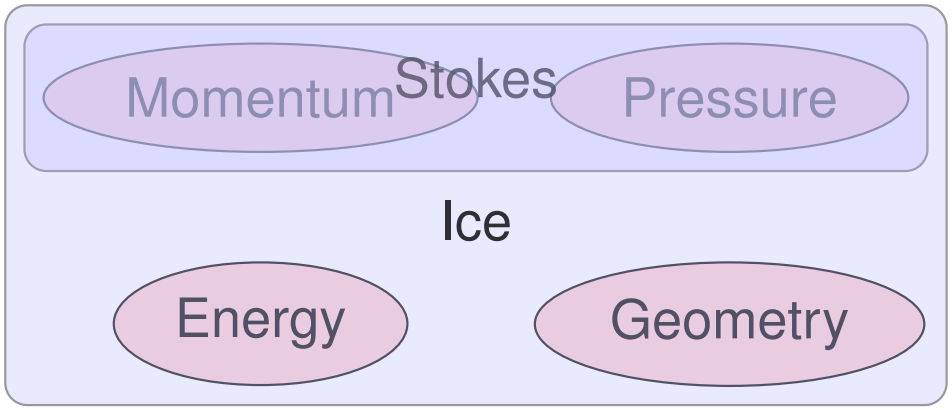
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc



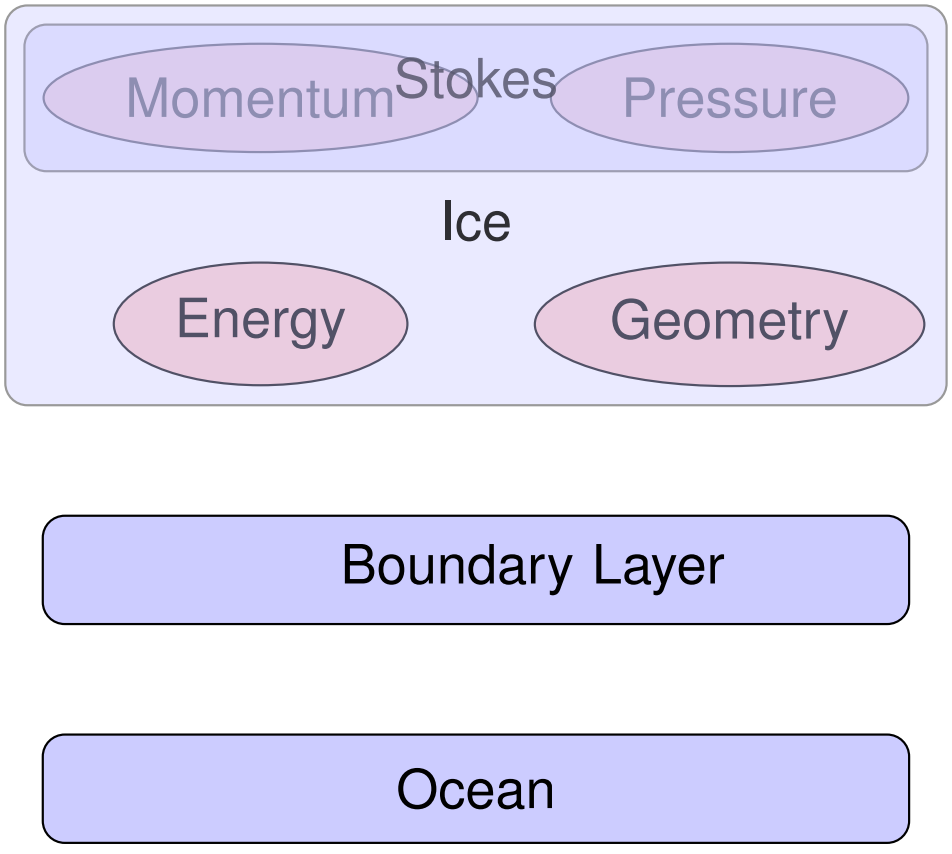
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

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

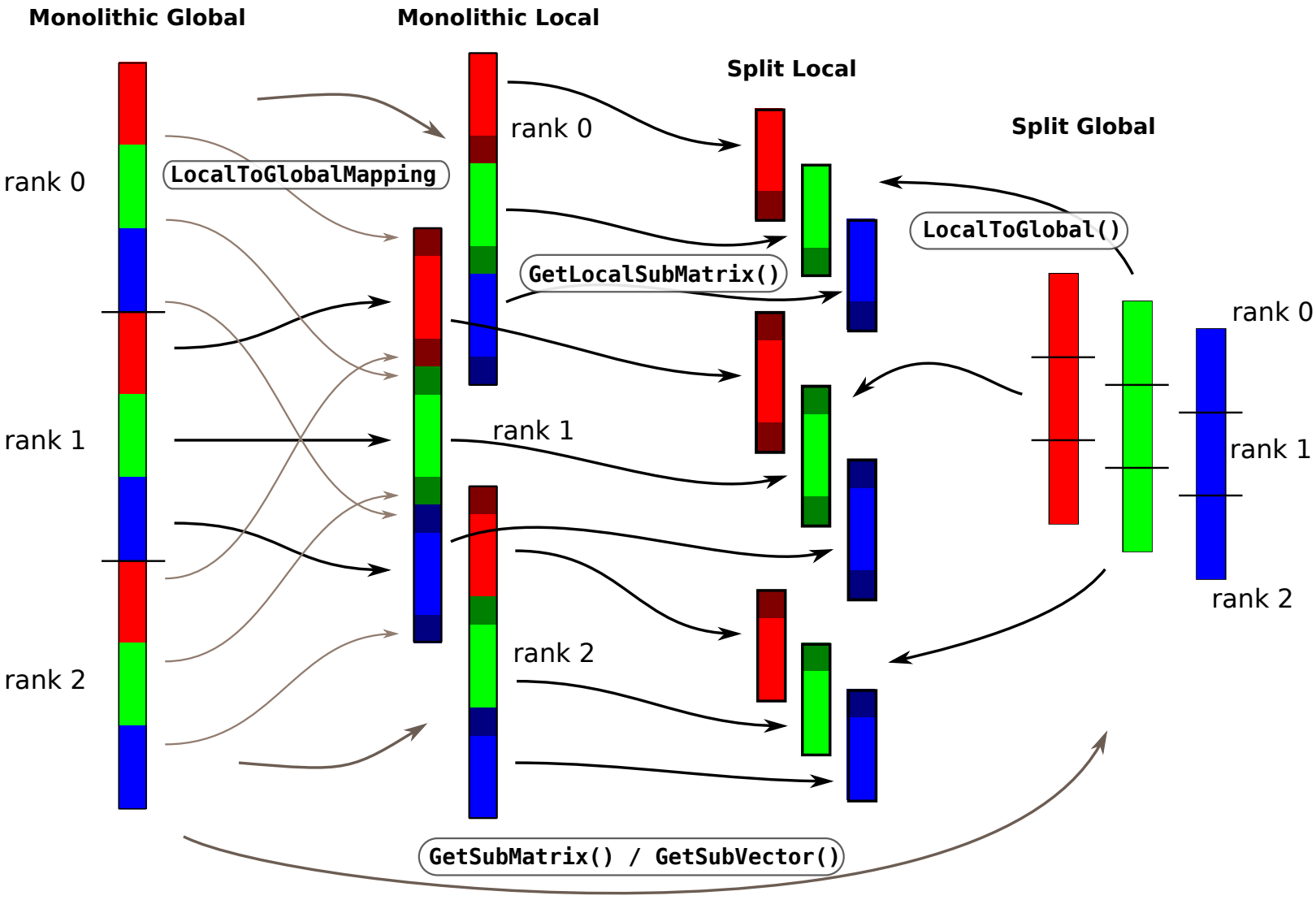
- Relaxation: `-pc_fieldsplit_type [additive,multiplicative,symmetric_multiplicative]`

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

  - 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`

`MatGetLocalSubMatrix(Mat A,IS rows,IS cols,Mat *B);`

- Primarily for assembly
  - `B` is not guaranteed to implement `MatMult`
  - The communicator for `B` is not specified,
    only safe to use non-collective ops (unless you check)
- `IS` represents an index set, includes a block size and communicator
- `MatSetValuesBlockedLocal()` is implemented
- MatNest returns nested submatrix, no-copy
- No-copy for Neumann-Neumann formats
  (unassembled across procs, e.g. BDDC, FETI-DP)
- Most other matrices return a lightweight proxy `Mat`
  - `COMM_SELF`
  - Values not copied, does not implement `MatMult`
  - Translates indices to the language of the parent matrix
  - Multiple levels of nesting are flattened

Stokes example

The common block preconditioners for Stokes require only options:

The Stokes System

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

```
-pc_type fieldsplit
-pc_field_split_type

-fieldsplit_0_ksp_type preonly
```

Stokes example

The common block preconditioners for Stokes require only options:

```
-pc_type fieldsplit
-pc_field_split_type additive

-fieldsplit_0_pc_type ml
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

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

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_field_split_type
multiplicative

-fieldsplit_0_pc_type hypre
-fieldsplit_0_ksp_type preonly

-fieldsplit_1_pc_type jacobi
-fieldsplit_1_ksp_type preonly
```

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

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_field_split_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
```

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

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_field_split_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
```

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

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_field_split_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
```

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

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_field_split_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
```

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

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_field_split_type schur
 -pc_fieldsplit_schur_factorization_type full
```

$$\text{PC}$$

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

Stokes example

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

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

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

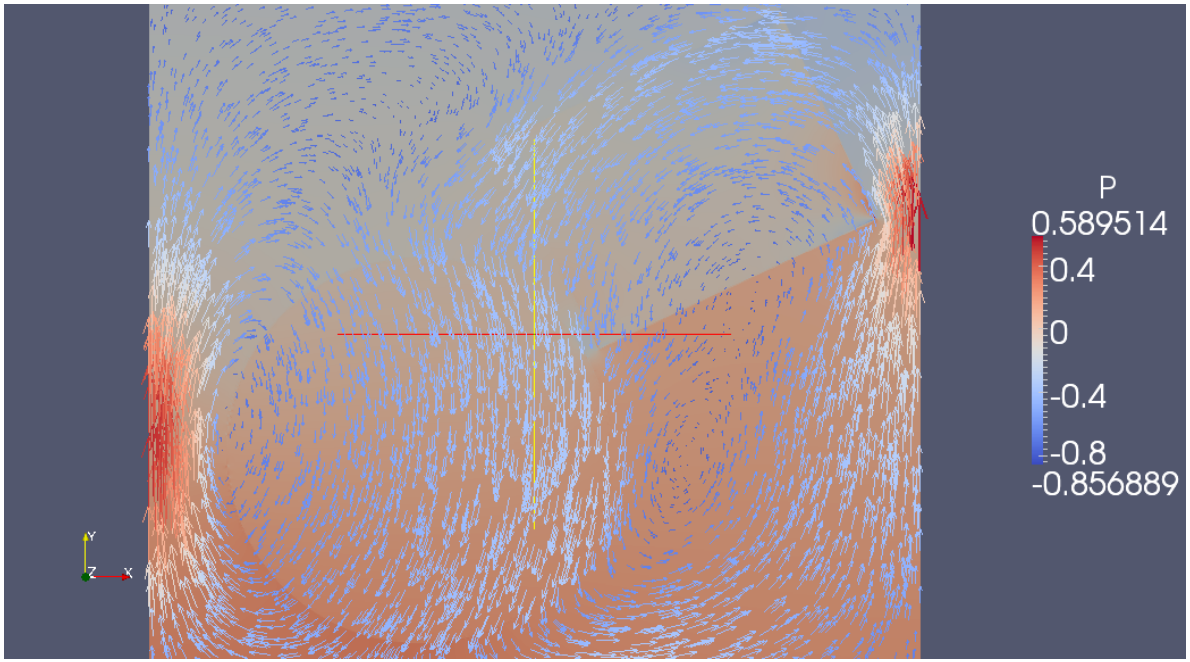
$$\begin{pmatrix} \hat{A} & B \\ 0 & \hat{S}_{\mathrm{LSC}} \end{pmatrix}$$

```
-mg_levels_pc_fieldsplit_schur_factorization_type upper
```

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
```

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

Rediscretized Multigrid using DM

- DM manages problem data beyond purely algebraic objects
  - structured, redundant, and (less mature) unstructured implementations in PETSc
  - third-party implementations
- `DMCoarsen(dmfine,coarse_comm,&coarsedm)` to create "geometric" coarse level
  - Also `DMRefine()` for grid sequencing and convenience
  - `DMCoarsenHookAdd()` for external clients to move resolution-dependent data for rediscretization and FAS
- `DMCreateInterpolation(dmcoarse,dmfine,&Interp,&Rscale)`
  - Usually uses geometric information, can be operator-dependent
  - Can be improved subsequently, e.g. using energy-minimization from AMG
- Resolution-dependent solver-specific callbacks use attribute caching on DM.
  - Managed by solvers, not visible to users unless they need exotic things (e.g. custom homogenization, reduced models)

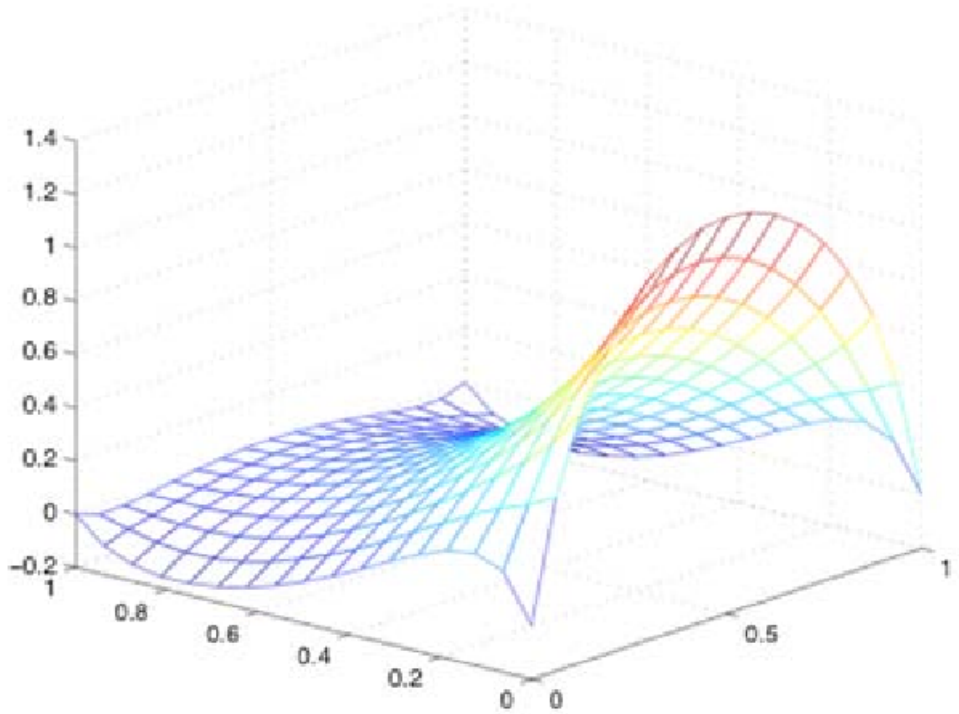
Three schools of thought

- 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 $\kappa(P^{-1}A) \sim \left(1 + \log \frac{H}{h}\right)^{2(L-1)}$
  - 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

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

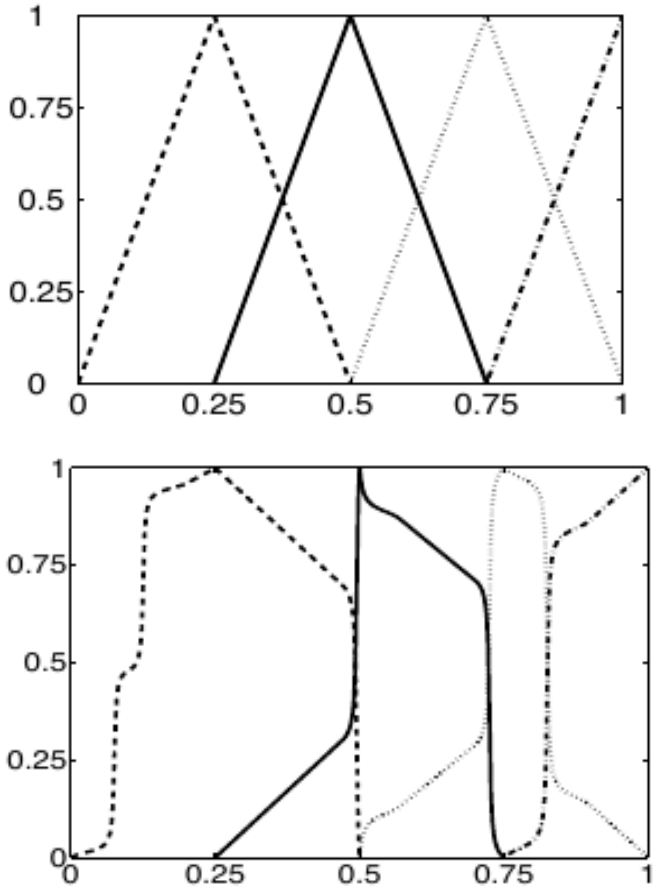
Example: BDDC/FETI-DP coarse basis function



[Mandel and Sousedik 2010]

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

Why I like subdomain problems



[Arbogast 2011]

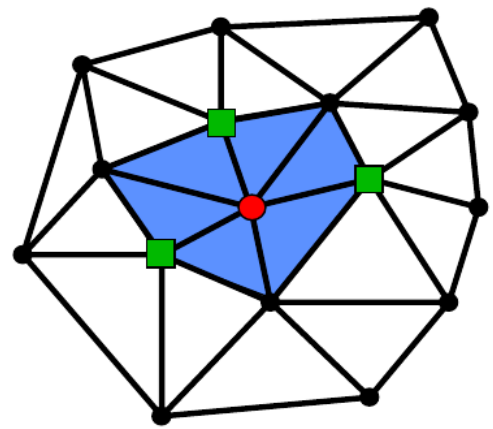
- subassembly avoids explicit matrix triple product $A_{\text{coarse}} \leftarrow P^T A_{\text{fine}} P$
- can update the coarse operator locally (e.g. local nonlinearity)
- need not assemble entire fine grid operator
- if repetitive structure, need not store entire fine grid state
- can coarsen very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

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: $C \sim$ number of vertices per cell
- larger block smoothers help reduce C



Monolithic nonlinear solvers

Coupled nonlinear multigrid accelerated by NGMRES with multi-stage smoothers

```
-lidvelocity 200 -grashof 1e4
-snes_grid_sequence 5 -snes_monitor -snes_view
-snes_type ngmres
-npc_snes_type fas
-npc_snes_max_it 1
-npc_fas_coarse_snes_type ls
-npc_fas_coarse_ksp_type preonly
-npc_fas_snes_type ms
-npc_fas_snes_ms_type vltp61
-npc_fas_snes_max_it 1
-npc_fas_ksp_type preonly
-npc_fas_pc_type pbjacobi
-npc_fas_snes_max_it 1
```

- Uses only residuals and point-block diagonal
- High arithmetic intensity and parallelism

Nonlinear solvers in PETSc SNES

LS, TR	Newton-type with line search and trust region
NRichardson	Nonlinear Richardson, usually preconditioned
VIRS, VISS	reduced space and semi-smooth methods for variational inequalities
QN	Quasi-Newton methods like BFGS
NGMRES	Nonlinear GMRES
NCG	Nonlinear Conjugate Gradients
GS	Nonlinear Gauss-Seidel/multiplicative Schwarz sweeps
FAS	Full approximation scheme (nonlinear multigrid)
MS	Multi-stage smoothers, often used with FAS for hyperbolic problems
Shell	Your method, often used as a (nonlinear) preconditioner

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

- 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

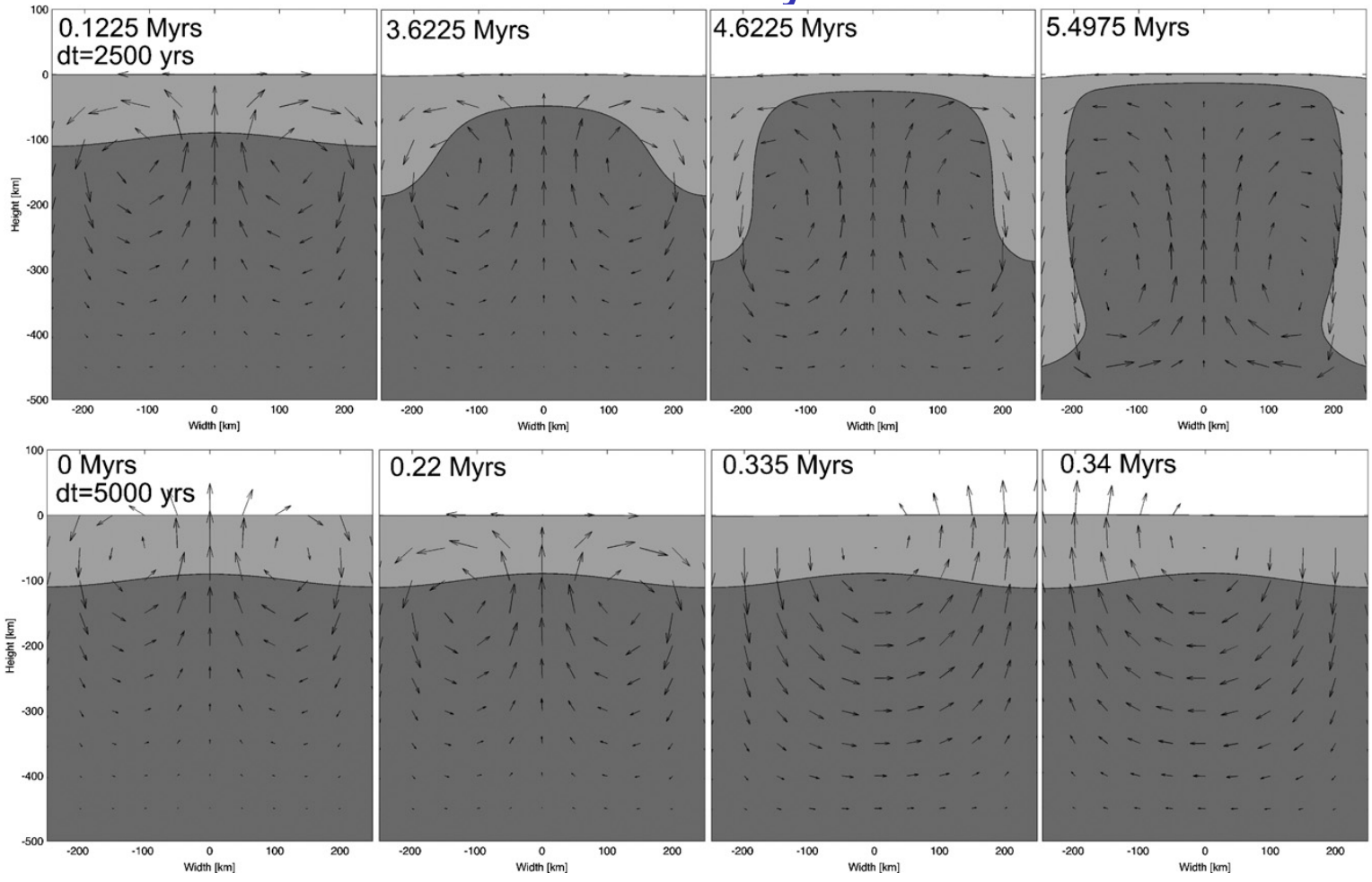
- Limited-memory Quasi-Newton/BFGS with lagged solve for H_0

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

pseudo-plastic rheology

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

The Drunken Seaman instability



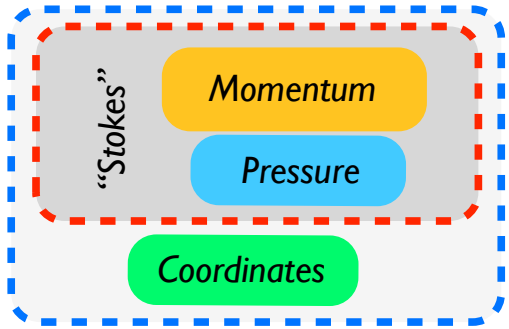
- Subduction and mantle convection with a free surface.
- Free surface critical to long-term dynamics (e.g. mountain range formation)
- Advective 0.01 CFL for stability.
- Semi-implicit helps: Kaus, Mühlhaus, and May, 2010

Stokes + Implicit Free Surface

$$\left[\eta D_{ij}(\boldsymbol{u})\right]_{,j} - p_{,i} = f_i$$

$$u_{k,k} = 0$$

$$\hat{x}_i = \hat{x}_i^{t-\Delta t} + \Delta t\, u_i(\hat{x}_i)$$



COORDINATE RESIDUALS

$$F_x := -u_i + \frac{\hat{x}_i}{\Delta t} - \frac{\hat{x}_i^{t-\Delta t}}{\Delta t}$$

[We use a full Lagrangian update of our mesh, with no remeshing]

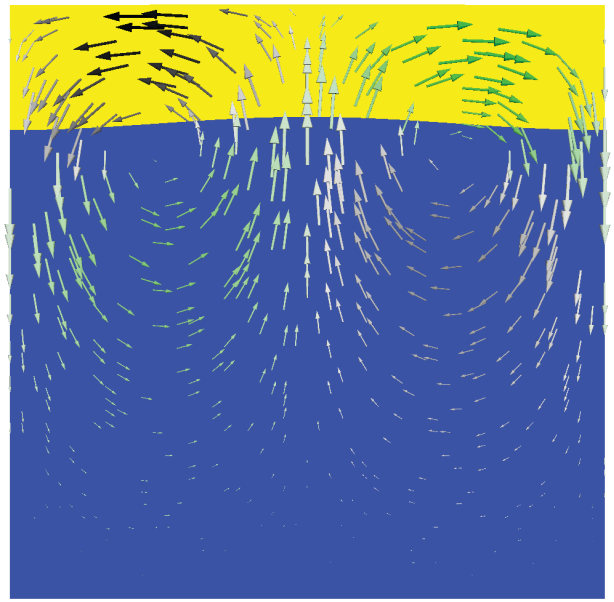
JACOBIAN

$$\mathcal{J}_{si} = \begin{bmatrix} A + \delta_{\hat{x}} A & B + \delta_{\hat{x}} B & J_{ac} \\ B^T + \delta_{\hat{x}} B^T & 0 & J_{bc} \\ -I & 0 & \frac{I}{\Delta t} \end{bmatrix}$$

Reuse stokes operators and saddle point preconditioners

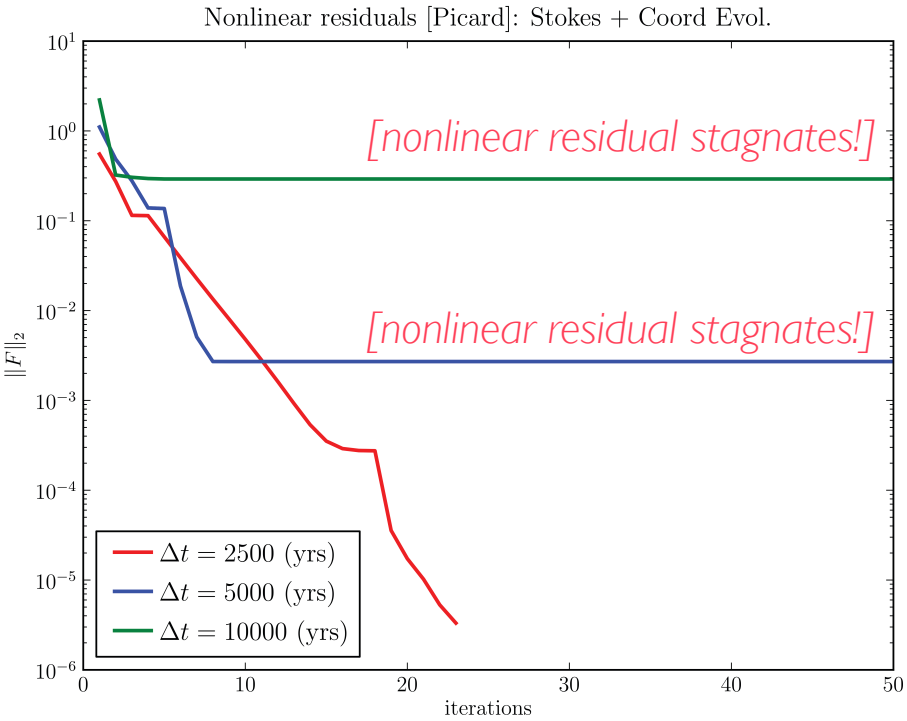
NESTED PRECONDITIONER

$$\mathcal{P}_{si} = \begin{bmatrix} \left[\ \mathcal{P}_s^l\ \right] \\ \left[\ I\ \right]\left[\ -\frac{I}{\Delta t}\ \right] \end{bmatrix} \qquad \mathcal{P}_s^l = \begin{bmatrix} A & 0 \\ B^T & -S \end{bmatrix}$$

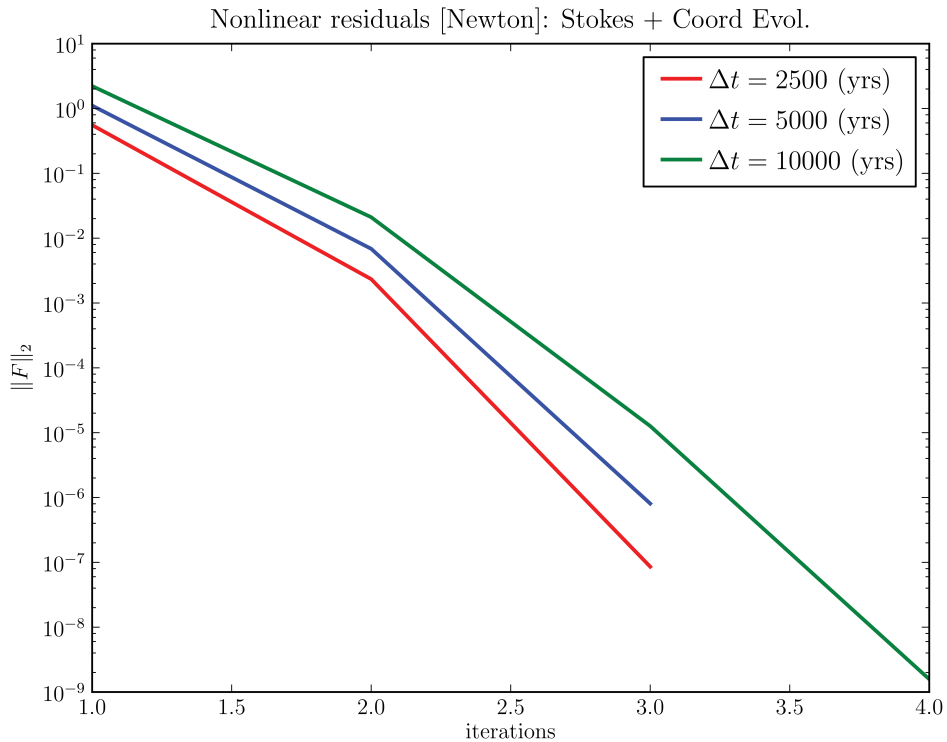


"Drunken seaman", Rayleigh Taylor instability test case from Kaus et al., 2010. Dense, viscous material (yellow) overlying less dense, less viscous material (blue).

Stokes + Implicit Free Surface

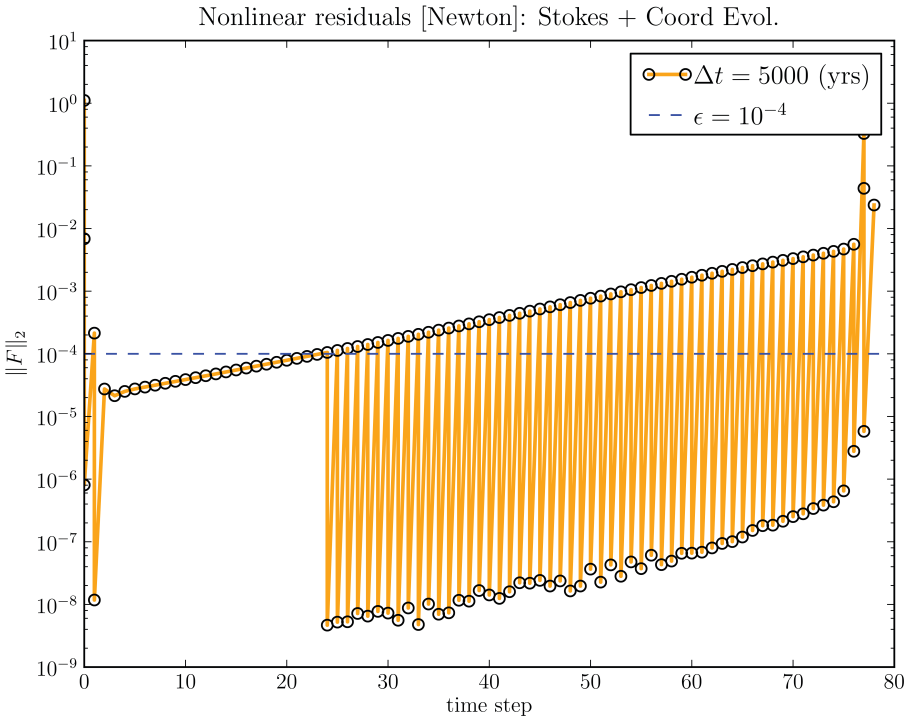


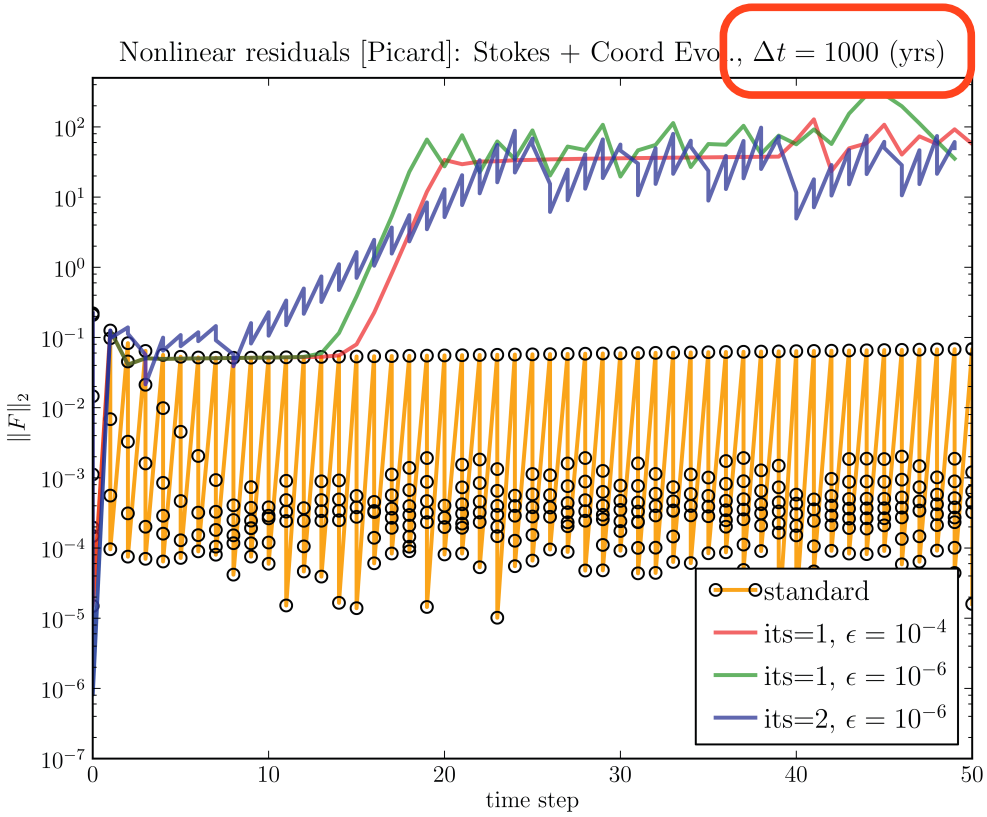
* Picard fails to converge for large time steps sizes.



* Newton is robust for a wide range of time step sizes.

Stokes + Implicit Free Surface





* The nonlinear residual *ALWAYS* increases from one step to the next.

* A nonlinear solve is required to control the error.

* An accurate nonlinear solve on the first time step, combined with 1 or 2 nonlinear iterations on subsequent steps still results in severe errors. *This is true even when dt is small.*

Conservative (non-Boussinesq) two-phase ice flow

Find momentum density ρu, pressure p, and total energy density E:

$$(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta Du_i + p1) - \rho g = 0$$
$$\rho_t + \nabla\cdot\rho u = 0$$
$$E_t + \nabla\cdot\big((E+p)u - k_T\nabla T - k_\omega\nabla\omega\big) - \eta Du_i : Du_i - \rho u\cdot g = 0$$

- Solve for density ρ, ice velocity u_i, temperature T, and melt fraction ω using constitutive relations.
  - Simplified constitutive relations can be solved explicitly.
  - Temperature, moisture, and strain-rate dependent rheology η.
  - High order FEM, typically Q_3 momentum & energy
- DAEs solved implicitly after semidiscretizing in space.
- Preconditioning using nested fieldsplit
- Thermomechanical steady state in about 10 nonlinear iterations

Relative effect of the blocks

$$J = \begin{pmatrix} J_{uu} & J_{up} & J_{uE} \\ J_{pu} & 0 & 0 \\ J_{Eu} & J_{Ep} & J_{EE} \end{pmatrix}.$$

J_{uu} Viscous/momentum terms, nearly symmetric, variable coefficionts, anisotropy from Newton.

J_{up} Weak pressure gradient, viscosity dependence on pressure (small), gravitational contribution (pressure-induced density variation). Large, nearly balanced by gravitational forcing.

J_{uE} Viscous dependence on energy, very nonlinear, not very large.

J_{pu} Divergence (mass conservation), nearly equal to J_{up}^T.

J_{Eu} Sensitivity of energy on momentum, mostly advective transport. Large in boundary layers with large thermal/moisture gradients.

J_{Ep} Thermal/moisture diffusion due to pressure-melting, $u \cdot \nabla$.

J_{EE} Advection-diffusion for energy, very nonlinear at small regularization. Advection-dominated except in boundary layers and stagnant ice, often balanced in vertical.

How much nesting?

$$P_1 = \begin{pmatrix} J_{uu} & J_{up} & J_{uE} \\ 0 & B_{pp} & 0 \\ 0 & 0 & J_{EE} \end{pmatrix}$$

$$P = \begin{bmatrix} \begin{pmatrix} J_{uu} & J_{up} \\ J_{pu} & 0 \end{pmatrix} & \\ \begin{pmatrix} J_{Eu} & J_{Ep} \end{pmatrix} & J_{EE} \end{bmatrix}$$

- B_{pp} is a mass matrix in the pressure space weighted by inverse of kinematic viscosity.
- Elman, Mihajlović, Wathen, JCP 2011 for non-dimensional isoviscous Boussinesq.
- Works well for non-dimensional problems on the cube, not for realistic parameters.
- Inexact inner solve using upper-triangular with B_{pp} for Schur.
- Another level of nesting.
- GCR tolerant of inexact inner solves.
- Outer converges in 1 or 2 iterations.
  - Low-order preconditioning full-accuracy unassembled high order operator.
  - Build these on command line with PETSc `PCFieldSplit`.

Phase field models

State variables $u = (u_1, \ldots, u_N)^T$ are concentrations of different phases satisfying the inequality and sum constraints

$$u(x,t) \in G = \{v \in \mathbb{R}^d | v_i \geq 0, \sum_{i=1}^{N} v_i = 1\}, \quad \forall (x,t) \in Q.$$

Minimize free energy, reduced space active set method

$$J = \begin{pmatrix} A & 0 & 0 & -I \\ 0 & A & 0 & -I \\ 0 & 0 & A & -I \\ -I & -I & -I & 0 \end{pmatrix}, \qquad P = \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & A & 0 & 0 \\ 0 & 0 & A & 0 \\ -I & -I & -I & S_{\mathrm{LSC}} \end{pmatrix}$$

```
-ksp_type fgmres -pc_type fieldsplit
-pc_fieldsplit_detect_saddle_point
-pc_fieldsplit_type schur
-pc_fieldsplit_schur_precondition self
-fieldsplit_0_ksp_type preonly
-fieldsplit_0_pc_type hypre
-fieldsplit_1_ksp_type fgmres
-fieldsplit_1_pc_type lsc
```

IMEX time integration in PETSc

- Additive Runge-Kutta IMEX methods

$$G(t,x,\dot{x}) = F(t,x)$$
$$J_\alpha = \alpha G_{\dot{x}} + G_x$$

  - User provides:
    - `FormRHSFunction(ts,`t`,`x`,`F`,void *ctx);`
    - `FormIFunction(ts,`t`,`x`,`$\dot{x}$`,`G`,void *ctx);`
    - `FormIJacobian(ts,`t`,`x`,`$\dot{x}$`,`α`,`J`,`J_p`,mstr,void *ctx);`
  - L-stable DIRK for stiff part G
  - Choice of explicit method, e.g. SSP
  - Orders 2 through 5, embedded error estimates
  - Dense output, hot starts for Newton
  - More accurate methods if G is linear, also Rosenbrock-W
  - Can use preconditioner from classical “semi-implicit” methods
  - FAS nonlinear solves supported
  - Extensible adaptive controllers, can change order within a family
  - Easy to register new methods: `TSARKIMEXRegister()`
- Eliminated many interface quirks in PETSc 3.3
- Single step interface so user can have own time loop

The Roadmap

Hardware trends

- More cores (keep hearing $\mathcal{O}(1000)$ per node)
- Long vector registers (already 32 bytes for AVX and BG/Q)
- Must use SMT to hide memory latency
- Must use SMT for floating point performance (GPU, BG/Q)
- Large penalty for non-contiguous memory access

“Free flops”, but how can we use them?

- High order methods good: better accuracy per storage
- High order methods bad: work unit gets larger
- GPU threads have very little memory, must keep work unit small
- Need library composability, keep user contribution embarrassingly parallel

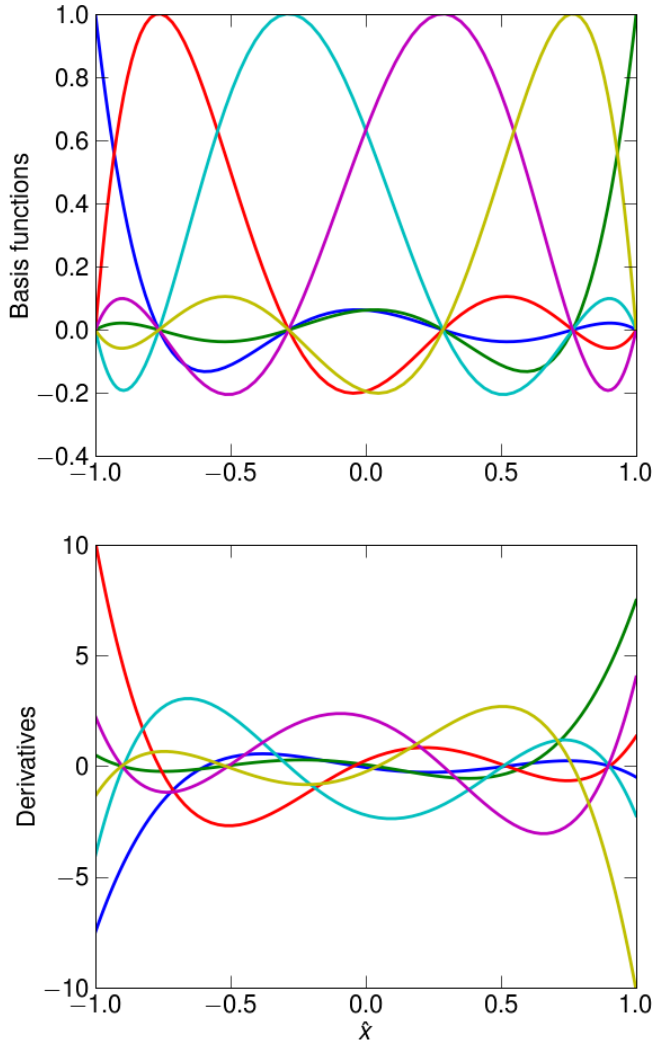
How to program this beast?

- Decouple physics from discretization
  - Expose small, embarrassingly parallel operations to user
  - Library schedules user threads for reuse between kernels
  - User provides physics in kernels run at each quadrature point
  - Continuous weak form: find $u \in \mathscr{V}_D$

$$v^T F(u) \sim \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u) = 0, \qquad \forall v \in \mathscr{V}_0$$

  - Similar form at faces, but may involve Riemann solve
- Library manages reductions
  - Interpolation and differentiation on elements
  - Interaction with neighbors (limiting, edge stabilization)
  - Exploit tensor product structure to keep working set small
  - Assembly into solution/residual vector (sum over elements)

Nodal hp-version finite element methods



1D reference element

- Lagrange interpolants on Legendre-Gauss-Lobatto points
- Quadrature $\hat{R}$, weights $\hat{W}$
- Evaluation: $\hat{B}, \hat{D}$

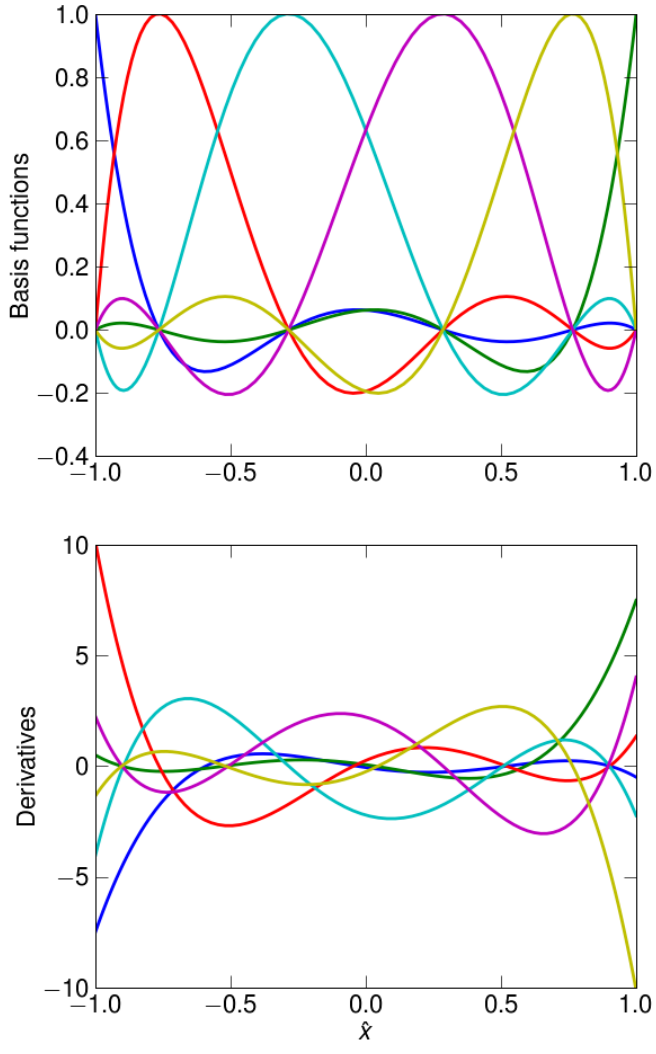
3D reference element

$$\hat{W} = \hat{W} \otimes \hat{W} \otimes \hat{W}$$
$$\hat{B} = \hat{B} \otimes \hat{B} \otimes \hat{B}$$

$$\hat{D}_0 = \hat{D} \otimes \hat{B} \otimes \hat{B}$$
$$\hat{D}_1 = \hat{B} \otimes \hat{D} \otimes \hat{B}$$
$$\hat{D}_2 = \hat{B} \otimes \hat{B} \otimes \hat{D}$$

These tensor product operations are very efficient, 70% of peak flop/s

Nodal hp-version finite element methods



1D reference element

- Lagrange interpolants on Legendre-Gauss-Lobatto points
- Quadrature $\hat{R}$, weights $\hat{W}$
- Evaluation: $\hat{B}, \hat{D}$

3D reference element

$$\hat{W} = \hat{W} \otimes \hat{W} \otimes \hat{W}$$
$$\hat{B} = \hat{B} \otimes \hat{B} \otimes \hat{B}$$

$$\hat{D}_0 = \hat{D} \otimes \hat{B} \otimes \hat{B}$$
$$\hat{D}_1 = \hat{B} \otimes \hat{D} \otimes \hat{B}$$
$$\hat{D}_2 = \hat{B} \otimes \hat{B} \otimes \hat{D}$$

These tensor product operations are very efficient, 70% of peak flop/s

Operations on physical elements

Mapping to physical space

$$x^e : \hat{K} \to K^e, \quad J^e_{ij} = \partial x^e_i / \partial \hat{x}_j, \quad (J^e)^{-1} = \partial \hat{x} / \partial x^e$$

Element operations in physical space

$$B^e = \hat{B} \qquad W^e = \hat{W}\Lambda(|J^e(r)|)$$

$$D^e_i = \Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right)\hat{D}_0 + \Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right)\hat{D}_1 + \Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right)\hat{D}_2$$

$$(D^e_i)^T = \hat{D}^T_0\Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right) + \hat{D}^T_1\Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right) + \hat{D}^T_2\Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right)$$

Global problem is defined by assembly

$$F(u) = \sum_e \mathscr{E}^T_e \left[(B^e)^T W^e \Lambda(f_0(u^e, \nabla u^e)) + \sum_{i=0}^{d} (D^e_i)^T W^e \Lambda(f_{1,i}(u^e, \nabla u^e)) \right] = 0$$

where $u^e = B^e \mathscr{E}^e u$ and $\nabla u^e = \{D^e_i \mathscr{E}^e u\}^2_{i=0}$

Representation of Jacobians, Automation

- For unassembled representations, decomposition, and assembly
- Continuous weak form: find u

$$v^T F(u) \sim \int_\Omega v \cdot f_0(u, \nabla u) + \nabla v : f_1(u, \nabla u) = 0, \qquad \forall v \in \mathscr{V}_0$$

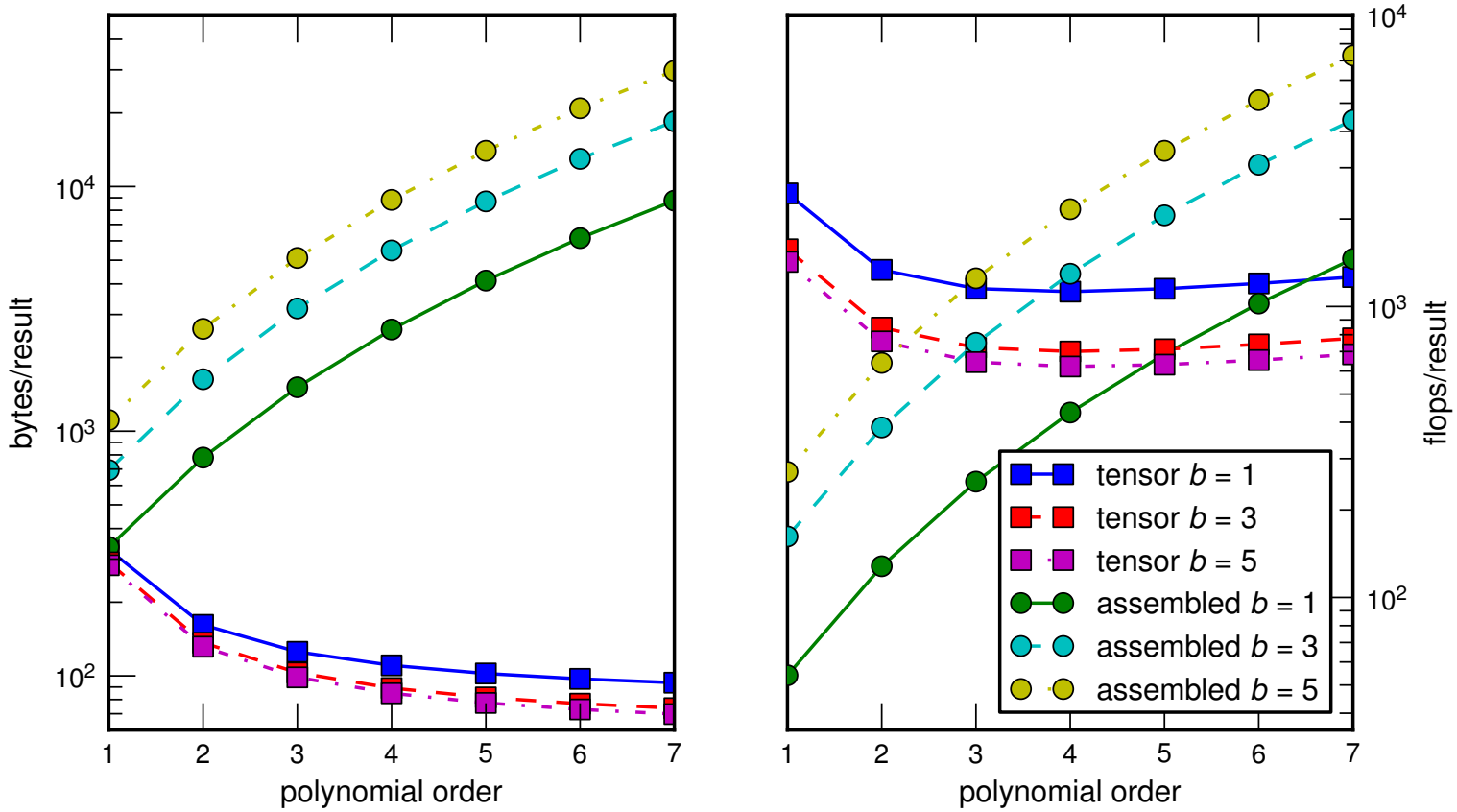
- Weak form of the Jacobian $J(u)$: find w

$$v^T J(u) w \sim \int_\Omega \begin{bmatrix} v^T & \nabla v^T \end{bmatrix} \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} w \\ \nabla w \end{bmatrix}$$

$$[f_{i,j}] = \begin{bmatrix} \frac{\partial f_0}{\partial u} & \frac{\partial f_0}{\partial \nabla u} \\ \frac{\partial f_1}{\partial u} & \frac{\partial f_1}{\partial \nabla u} \end{bmatrix} (u, \nabla u)$$

- Terms in $[f_{i,j}]$ easy to compute symbolically, AD more scalable.
- Nonlinear terms f_0, f_1 usually have the most expensive nonlinearities in the computation of scalar material parameters
  - Equations of state, effective viscosity, “star” region in Riemann solve
  - Compute gradient with reverse-mode, store at quadrature points.
  - Perturb scalars, then use forward-mode to complete the Jacobian.
  - Flip for action of the adjoint.

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)
Sandy Bridge 6-core	21*	150	7.2
Magny Cours 16-core	42*	281	6.7
Blue Gene/Q node	43	205	4.8
Tesla M2050	144	515	3.6

Outlook on Solver Composition

- Unintrusive composition of multigrid and block preconditioning
- Build preconditioners from literature
  on the command line
- User code does not depend on matrix format, preconditioning method, nonlinear solution method, time integration method (implicit or IMEX), or size of coupled system (except for driver).

In development

- Distributive relaxation, Vanka smoothers, coarsening for “dual” variables
- Improving operator-dependent semi-geometric multigrid
- More automatic spectral analysis and smoother optimization
- Automated support for mixing analysis into MG levels

Performance challenges

- Expressing NUMA distribution between libraries
- Cache sharing/granularity
- Small dense linear algebra for MG smoothers
- Flexibility in GPU kernels

- Maximize science per Watt
- Huge scope remains at problem formulation
- Raise level of abstraction at which a problem is formally specified
- Algorithmic optimality is crucial