Toward less synchronous composable multilevel methods for implicit multiphysics simulation

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

2012-02-06

Outline

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.

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

How much nesting?

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

- 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.

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

- 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`.

Example 3×3 problem with nested 2×2 split

```
-fieldsplit_s_ksp_type gcr
-fieldsplit_s_ksp_rtol 1e-1
-fieldsplit_s_ksp_monitor_vht
-fieldsplit_s_ksp_monitor_singular_value
-fieldsplit_s_pc_type fieldsplit
-fieldsplit_s_pc_fieldsplit_type schur
-fieldsplit_s_pc_fieldsplit_real_diagonal
-fieldsplit_s_pc_fieldsplit_schur_factorization_type lower
-fieldsplit_s_fieldsplit_u_ksp_type gmres
-fieldsplit_s_fieldsplit_u_ksp_max_it 10
-fieldsplit_s_fieldsplit_u_pc_type asm
-fieldsplit_s_fieldsplit_u_sub_pc_type ilu
-fieldsplit_s_fieldsplit_u_sub_pc_factor_levels 1
-fieldsplit_s_fieldsplit_u_ksp_converged_reason
-fieldsplit_s_fieldsplit_p_ksp_type preonly
-fieldsplit_s_fieldsplit_p_ksp_max_it 1
-fieldsplit_s_fieldsplit_p_pc_type jacobi
-fieldsplit_e_ksp_type gmres
-fieldsplit_e_ksp_converged_reason
-fieldsplit_e_pc_type asm
-fieldsplit_e_sub_pc_type ilu
-fieldsplit_e_sub_pc_factor_levels 2
```

Monolithic approaches

Parallel direct solver

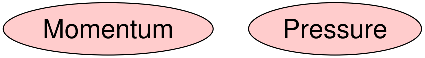
```
-dm_mat_type aij -pc_type lu -pc_factor_mat_solver_package mumps
```

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_max_it 1
-npc_fas_ksp_type preonly
-npc_fas_pc_type pbjacobi
-npc_fas_snes_ms_type m62
-npc_fas_snes_max_it 1
```

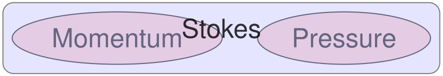
Outline

Multi-physics coupling in PETSc



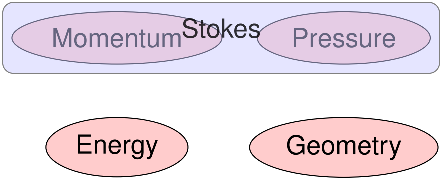
- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc



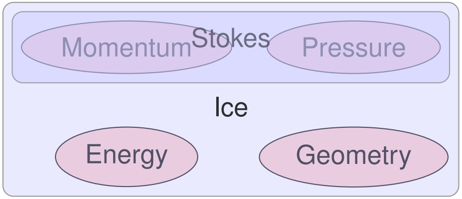
- package each "physics" independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc



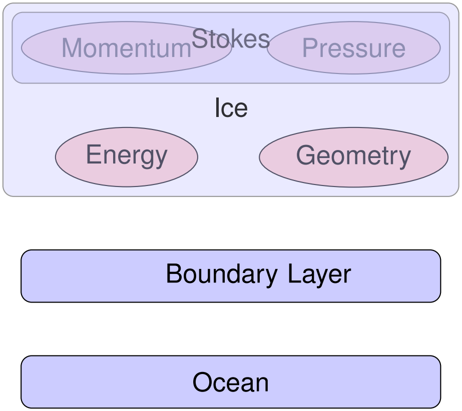
- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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

Multi-physics coupling in PETSc

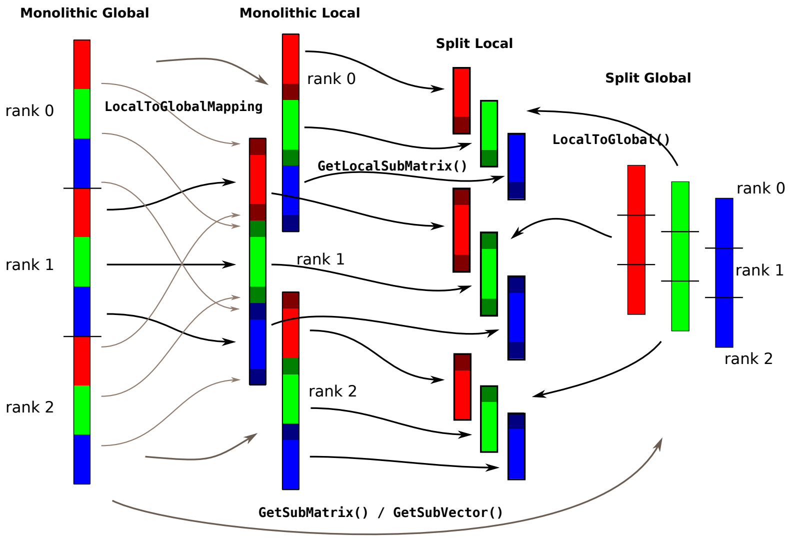


- package each “physics” independently
- solve single-physics and coupled problems
- semi-implicit and fully implicit
- reuse residual and Jacobian evaluation unmodified
- direct solvers, fieldsplit inside multigrid, multigrid inside fieldsplit without recompilation
- use the best possible matrix format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



Work in Split Local space, matrix data structures reside in any space.

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

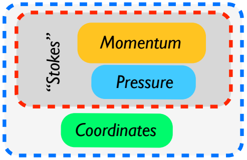
- Primarily for assembly
  - `B` is not guaranteed to implement `MatMult`
  - The communicator for `B` is not specified, only safe to use non-collective ops (unless you check)
- `IS` represents an index set, includes a block size and communicator
- `MatSetValuesBlockedLocal()` is implemented
- MatNest 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 + 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]

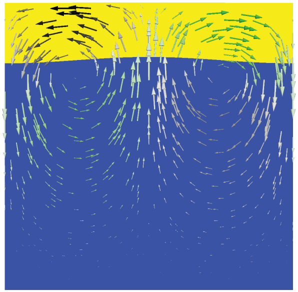
JACOBIAN

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

Reuse stokes operators and saddle point preconditioners

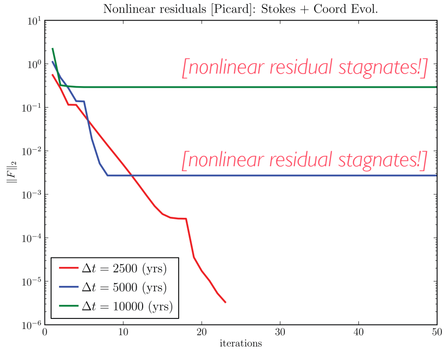
NESTED PRECONDITIONER

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

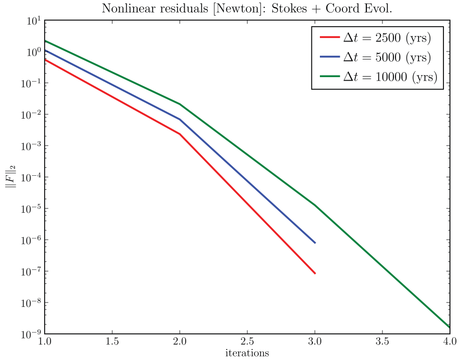


"Drunken seaman", Rayleigh Taylor instability test case from Kaus et al., 2010. Dense, viscous material (yellow) overlying less dense, less viscous material (blue).

Stokes + Implicit Free Surface

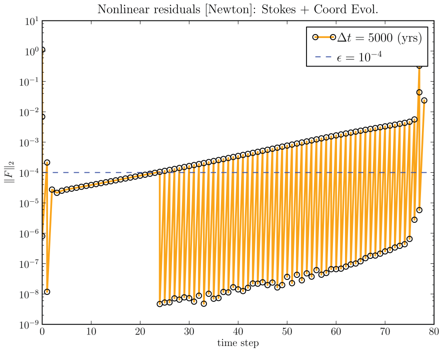


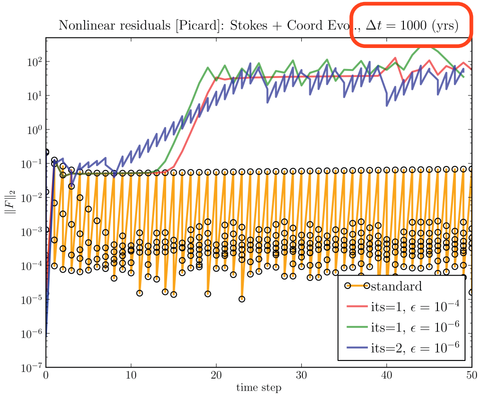
* Picard fails to converge for large time steps sizes.



* 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.*

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

Outline

On-node hardware roadmap

Hardware trends

- More cores (keep hearing $\mathscr{O}(1000)$ per node)
- Long vector registers (32B for AVX and BG/Q, 64B for MIC)
- Must use SMT to hide memory latency (POWER7)
- 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

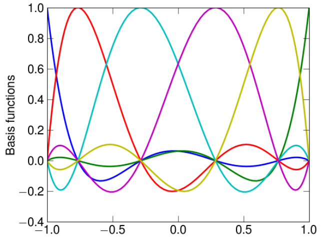
How to program this beast?

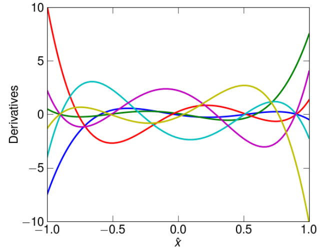
- Decouple physics from discretization
  - Expose small, embarrassingly parallel operations to user
  - Library schedules user threads for reuse between kernels
  - User provides physics in kernels run at each quadrature point
  - Continuous weak form: find $u \in \mathcal{V}_D$

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

  - Similar form at faces, may involve Riemann solve
- Library manages reductions
  - Interpolation and differentiation on elements
  - 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: $\hat{B}, \hat{D}$

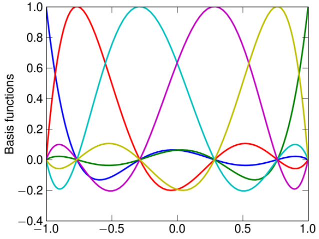
3D reference element

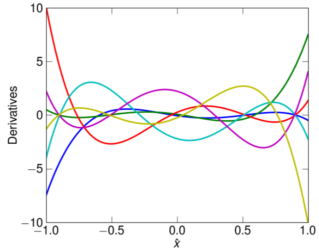
$$\hat{W} = \hat{W} \otimes \hat{W} \otimes \hat{W}$$
$$\hat{B} = \hat{B} \otimes \hat{B} \otimes \hat{B}$$

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

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

$$\begin{aligned} B^e &= \hat{B} \qquad\qquad W^e = \hat{W} \Lambda(|J^e(r)|) \\ D^e_i &= \Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right)\hat{D}_0 + \Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right)\hat{D}_1 + \Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right)\hat{D}_2 \\ (D^e_i)^T &= \hat{D}^T_0 \Lambda\left(\frac{\partial \hat{x}_0}{\partial x_i}\right) + \hat{D}^T_1 \Lambda\left(\frac{\partial \hat{x}_1}{\partial x_i}\right) + \hat{D}^T_2 \Lambda\left(\frac{\partial \hat{x}_2}{\partial x_i}\right) \end{aligned}$$

Global problem is defined by assembly

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

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

Representation of Jacobians, Automation

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

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

- 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.

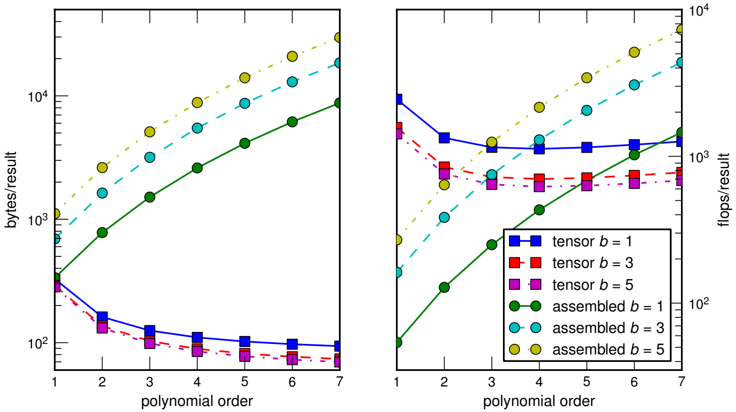
Conservative (non-Boussinesq) two-phase ice flow

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

$$
\begin{gathered}
(\rho u)_t + \nabla\cdot(\rho u \otimes u - \eta D u_i + p1) - \rho g = 0 \\
\rho_t + \nabla\cdot\rho u = 0 \\
E_t + \nabla\cdot\big((E+p)u - k_T \nabla T - k_\omega \nabla\omega\big) - \eta D u_i : D u_i - \rho u\cdot g = 0
\end{gathered}
$$

- 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

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 $< 4\%$

Memory Bandwidth

Operation	Arithmetic Intensity (flops per byte)
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
GeForce 9400M	21	54	2.6
GTX 285	159	1062	6.8
Tesla M2050	144	1030	7.1

Prospects for reducing synchronization

- Dot products and norms
  - orthogonality is a powerful concept
  - dot product/norm fusion in CG variants
  - latency-tolerant Krylov methods, TSQR for GMRES
  - nonlinear methods (e.g. NGMRES, BFGS, line searches)
  - hierarchical methods to limit system-wide norms
  - setting up smoothers and coarsening rates for AMG
- additive coarse grids
- subphysics on subcommunicators, even within multigrid context
- s-step methods (and other fusion)
  - often spoiled by algorithmic requirements of preconditioning
  - relevant for multigrid smoothers
  - difficult crossovers for 3D problems

Multigrid is *always* strong scaling

- Finest level is chosen by the application (might have big subdomains)
- All coarsened levels choose communicator size based on strong scaling limit
- Optimizing the strong scaling limit pays off consistently
- Rapid coarsening is important (2:1 semi-coarsening not okay any more)

Advanced Analysis

- Uncertainty quantification
  - intrusive vs unintrusive methods, multilevel
  - uncertainty in modeling error
  - use subdifferentials for non-smooth processes
  - unified handling of heterogeneous observational data
- PDE-constrained optimization
  - multi-objective
  - robustness
  - rich problem description
  - fusing algorithmic steps (LNK and coupled DD fuse gradients with progress)
- Exploring stability manifolds
  - solving bordered linear and nonlinear systems
- nearly time-periodic nonlinear problems
  - identifying cycles in ocean models, turbomachinery

Software challenges

- Which interfaces do users have to interact with?
  - “F”ramework vs library
  - Extensibility is critical for multiphysics
- Asynchronous interfaces crossing module boundaries
  - How to ensure progress?
- Merge communication on multiple levels or between multiple physics
- Fusing coarse level operations
- Working with non-nested communicators is tricky
- Current solutions for hierarchical memory are bad for libraries
  - I want a communicator-like object
  - I want a way to allocate memory explicitly/relative to algorithmic dependencies instead of implicit “first touch”
- Time integration: IMEX, multirate, parallel in time
  - method of lines: $g(\dot{u}, u, t) = f(u, t)$
  - Lax-Wendroff time integration is harder for composable software