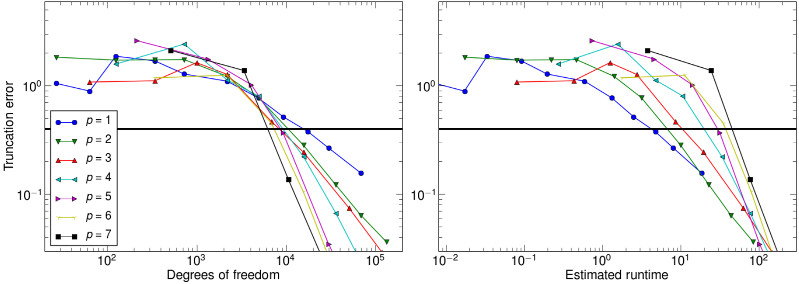
Making high-order finite elements fast, robust, and easy

Jed Brown

VAW, ETH Zürich

Columbia 2009-07-02

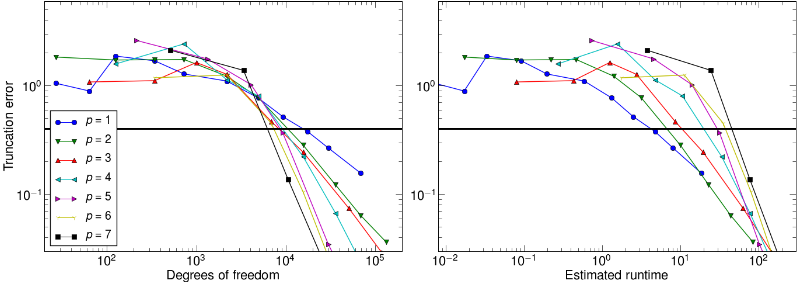
High order methods are expensive



Order $p - 1$ elements:

- element matrices have p^6 nonzeros
- cost $\mathcal{O}(p^7) - \mathcal{O}(p^9)$ to assemble
- **Very** expensive to precondition and solve with

High order methods are expensive



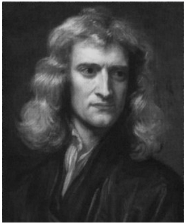
Order $p-1$ elements:

- element matrices have p^6 nonzeros
- cost $\mathcal{O}(p^7) - \mathcal{O}(p^9)$ to assemble
- **Very** expensive to precondition and solve with
- High order methods must be competitive **per degree of freedom** in order to be practical

What do we want from hp solvers?

- h independence
- p independence
- robust for high aspect and deformed elements
- robust to jumps in material coefficients/nonlinearity
- leverage existing code for preconditioning
- minimal problem-specific code development
- runtime within a small constant factor of the best problem-specific solver

Newton iteration



- Standard form of a nonlinear system

$$F(x) = 0$$

- Iteration

$$\text{Solve:} \quad J(x^n)s^n = -F(x^n)$$

$$\text{Update:} \quad x^{n+1} \leftarrow x^n + s^n$$

Stokes problem

$$\begin{bmatrix} \boldsymbol{v} \\ q \end{bmatrix}^T F(\boldsymbol{u}, p) \sim \int_\Omega \eta D\boldsymbol{v} : D\boldsymbol{u} - p\nabla \cdot \boldsymbol{v} - q\nabla \cdot \boldsymbol{u} - \boldsymbol{f} \cdot \boldsymbol{v} = 0 \quad \forall(\boldsymbol{v}, q)$$

$$\begin{bmatrix} \boldsymbol{v} \\ q \end{bmatrix}^T J(\boldsymbol{w}) \begin{bmatrix} \boldsymbol{u} \\ p \end{bmatrix} \sim \int_\Omega {\color{red}\eta D\boldsymbol{v} : D\boldsymbol{u} + \eta'(D\boldsymbol{v} : D\boldsymbol{w})(D\boldsymbol{w} : D\boldsymbol{u})} - {\color{green}p\nabla \cdot \boldsymbol{v}} - {\color{blue}q\nabla \cdot \boldsymbol{u}}$$

$$J(\boldsymbol{w}) = \begin{bmatrix} {\color{red}A(\boldsymbol{w})} & {\color{green}B^T} \\ {\color{blue}B} & \end{bmatrix}$$

Matrices and Preconditioners

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).

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathscr{P}(\hat{J})$ using information $\hat{J}$, such that $P^{-1}J$ (or JP^{-1}) has favorable spectral properties.

Left preconditioning in a Krylov iteration

$$(P^{-1}J)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}J)P^{-1}b, (P^{-1}J)^2 P^{-1}b, \dots\}$$

Matrices and Preconditioners

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).

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathscr{P}(\hat{J})$ using information $\hat{J}$, such that $P^{-1}J$ (or JP^{-1}) has favorable spectral properties.

Left preconditioning in a Krylov iteration

$$(P^{-1}J)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}J)P^{-1}b, (P^{-1}J)^2 P^{-1}b, \dots\}$$

Matrices and Preconditioners

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).

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathscr{P}(\hat{J})$ using information $\hat{J}$, such that $P^{-1}J$ (or JP^{-1}) has favorable spectral properties.

Left preconditioning in a Krylov iteration

$$(P^{-1}J)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}J)P^{-1}b, (P^{-1}J)^2P^{-1}b, \dots\}$$

Matrices and Preconditioners

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).

Definition (Preconditioner)

A preconditioner $\mathscr{P}$ is a method for constructing a matrix (just a linear function, not assembled!) $P^{-1} = \mathscr{P}(\hat{J})$ using information $\hat{J}$, such that $P^{-1}J$ (or JP^{-1}) has favorable spectral properties.

Left preconditioning in a Krylov iteration

$$(P^{-1}J)x = P^{-1}b$$

$$\{P^{-1}b, (P^{-1}J)P^{-1}b, (P^{-1}J)^2 P^{-1}b, \dots\}$$

Physics-based preconditioning

Working definition

Matrices assembled for preconditioning do not correspond to differential operators present in the continuum equations.

Examples

- Indefinite problems [1]
    - Incompressible flow/elasticity
    - Mixed formulations of flow in porous media
    - PDE-constrained optimization
- Stiff-wave problems [2]
    - Surface gravity waves in geophysical flows
    - Whistler waves in Hall MHD

General construction

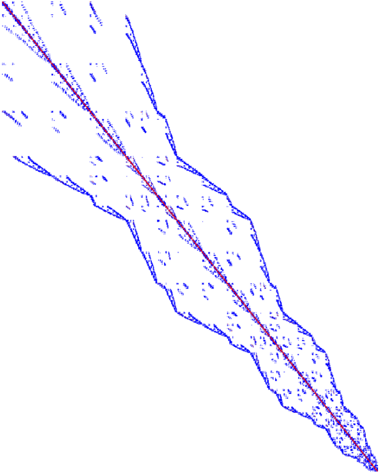
Block incomplete factorization of the Jacobian, reinterpret resulting Schur complements.

---

[1] Benzi, Golub, Liesen. *Numerical solution of saddle point problems*, 2005.

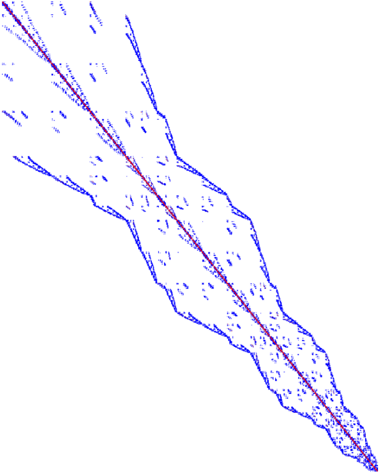
[2] Knoll, Mousseau, Chacón, Reisner. *Jacobian-Free Newton-Krylov methods for the accurate time integration of stiff wave systems*, 2005.

Bottlenecks of (Jacobian-free) Newton-Krylov



- Matrix assembly
  - integration: FPU
  - insertion: memory/branching
- Preconditioner setup
  - coarse level operators
  - overlapping subdomains
  - (incomplete) factorization
- Preconditioner application
  - triangular solves/relaxation: memory
  - coarse levels: network latency
- Matrix multiplication
  - Sparse storage: memory
  - Matrix-free: FPU

Bottlenecks of (Jacobian-free) Newton-Krylov



- Matrix assembly
  - integration: FPU
  - insertion: memory/branching
- Preconditioner setup
  - coarse level operators
  - overlapping subdomains
  - (incomplete) factorization
- Preconditioner application
  - triangular solves/relaxation: memory
  - coarse levels: network latency
- Matrix multiplication
  - Sparse storage: memory
  - Matrix-free: FPU
- Globalization

Hardware capabilities

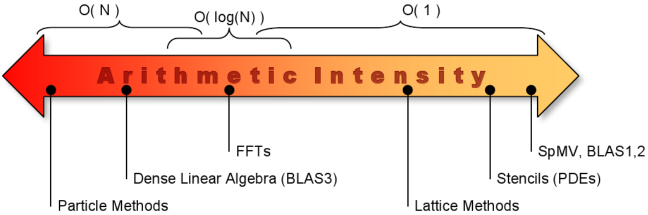
Floating point unit

Recent Intel: each core can issue

- 1 packed add (latency 3)
- 1 packed mult (latency 5)
- One can include a read
- Peak: 10 Gflop/s (`double`)

Memory

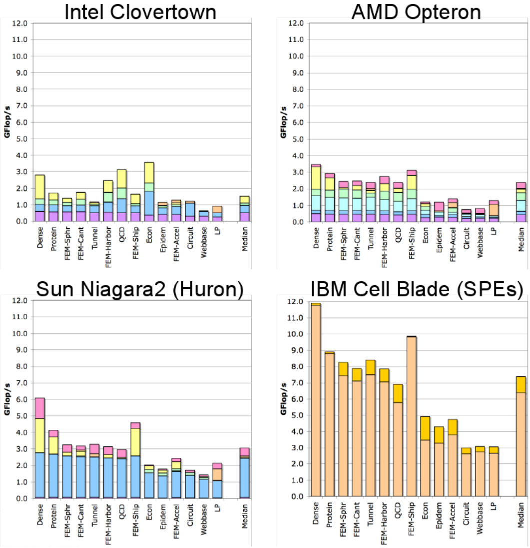
- ~ 250 cycle latency
- 5.3 GB/s bandwidth (per channel)
- 1 `double` load / 3.7 cycles



(Oliker et al. 2008)

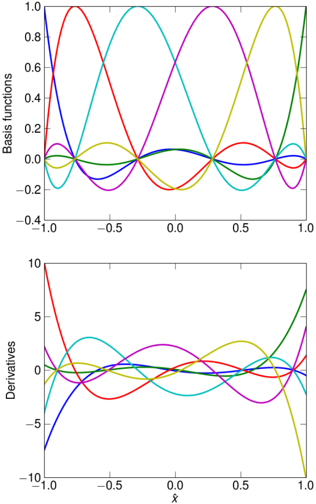
Sparse Mat-Vec

- Model faster cores by commenting out the inner kernel calls, but still performing all DMAs
- Enabled 1x1 BCOO
- ~16% improvement



(Oliker et al. *Multi-core Optimization of Sparse Matrix Vector Multiplication*, 2008)

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

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

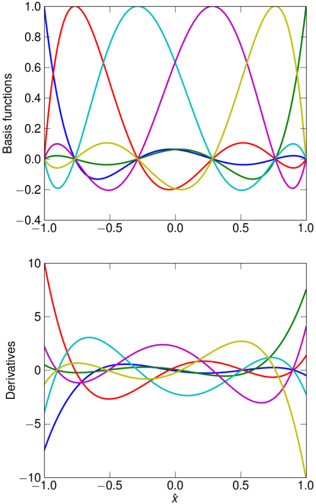
$$\hat{\boldsymbol{B}} = \hat{B} \otimes \hat{B} \otimes \hat{B}$$

$$\hat{\boldsymbol{D}}_0 = \hat{D} \otimes \hat{B} \otimes \hat{B}$$

$$\hat{\boldsymbol{D}}_1 = \hat{B} \otimes \hat{D} \otimes \hat{B}$$

$$\hat{\boldsymbol{D}}_2 = \hat{B} \otimes \hat{B} \otimes \hat{D}$$

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

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

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

$$\hat{\boldsymbol{D}}_0 = \hat{D} \otimes \hat{B} \otimes \hat{B}$$

$$\hat{\boldsymbol{D}}_1 = \hat{B} \otimes \hat{D} \otimes \hat{B}$$

$$\hat{\boldsymbol{D}}_2 = \hat{B} \otimes \hat{B} \otimes \hat{D}$$

These tensor product operations are very efficient, 10–20+ times faster than sparse mat-vec

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

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

Global problem is defined by assembly

$$\mathcal{E} = [\mathcal{E}^e]$$

where $\mathcal{E}^e$ maps global dofs to element dofs

Residuals

- Continuous weak form: find $u \in \boldsymbol{\mathcal{V}}_D$ such that

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

- Fully discrete form

$$\sum_e \mathcal{E}_e^T \Big[(\boldsymbol{B}^e)^T \boldsymbol{W}^e \Lambda(f_0(u^e, \nabla u^e)) + \sum_{i=0}^{2} (\boldsymbol{D}_i^e)^T \boldsymbol{W}^e \Lambda(f_1(u^e, \nabla u^e)) \Big] = \boldsymbol{0}$$

with $u^e = \boldsymbol{B}^e \mathcal{E}^e u$ and $\nabla u^e = \{\boldsymbol{D}_i^e \mathcal{E}^e u\}_{i=0}^2$.

1. Get element dofs with $\mathcal{E}^e$, evaluate $u, \nabla u$ at quadrature points
2. Apply the pointwise operations f_0, f_1
3. Weight the residuals with $\boldsymbol{W}^e$
4. Contribute weighted residuals via $\mathcal{E}_e^T (\boldsymbol{B}^e)^T$ and $\mathcal{E}_e^T (\boldsymbol{D}^e)^T$

Jacobians

- Continuous weak form: find $u \in \boldsymbol{\mathcal{V}}_D$ such that

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

- Weak form of the Jacobian

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

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

- Frequently much of $[f_{i,j}]$ is computed while evaluating f_i.
  - Inexpensive *taping* for full-accuracy matrix-free Jacobian
  - Code reuse in preconditioner assembly
- The terms in $[f_{i,j}]$ are easy to compute with symbolic math. Possible to automatically generate code.

Fast diagonalization

Strengths

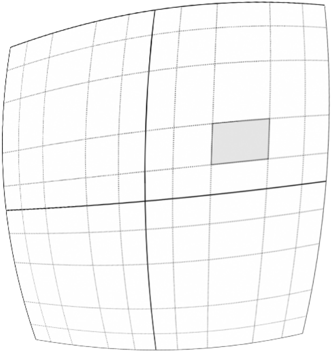
- Very robust for high order
- Avoids any sparse matrices (on finest level)

Weaknesses

- only available for special equations
- non-affine elements
- variable coefficients
- custom software development

(Lottes, Fischer. *Hybrid Multigrid/Schwarz Algorithms for the Spectral Element Method*, 2005)

"Dual order"



- any system of equations
- robust on non-affine elements
- robust to variable coefficients
- leverages existing software
- requires very little coding
- weak (bounded) p-dependence

Changing the inner product

Consider the problem $Ax = b$ discretized with high-order elements. The consistent formulation of this problem with low-order elements is $\hat{A}x = \hat{M}M^{-1}b$.

What code do you need to write?

Conventional FEM

- Residuals $v^T F(u)$

  ```
  for each quadrature point:
    sum basis functions:  $u, \nabla u$
    evaluate $f_0, f_1$
    weight residuals against $v, \nabla v$
  ```

- Assembly $J(w)$

  ```
  for each quadrature point:
    sum basis functions:  $w, \nabla w$
    evaluate $[f_{i,j}]$
    for each test function:
      for each trial function:
        sum into $K^e[\text{test}, \text{trial}]$
  insert $K^e$ into global matrix
  ```

Dual-order hp-FEM

- Residuals $v^T F(u)$

  ```
  evaluate $u, \nabla u$ at quad pts
  evaluate $f_0, f_1$, tape for $[f_{i,j}]$
  weight residuals and transpose
  ```

- Matrix-free $v^T J(w) u$

  ```
  evaluate $u, \nabla u$ at quad points
  restore $[f_{i,j}](w)$ from tape
  weight residuals
    $\begin{bmatrix} v \\ \nabla v \end{bmatrix}^T \begin{bmatrix} f_{0,0} & f_{0,1} \\ f_{1,0} & f_{1,1} \end{bmatrix} \begin{bmatrix} u \\ \nabla u \end{bmatrix}$
  transpose
  ```

- Assemble one or more matrices for preconditioning

$\mathfrak{p}$-Bratu

$$-\nabla \cdot \left(|\nabla u|^{\mathfrak{p}-2} \nabla u \right) - \lambda e^u - f = 0$$

for $1 \le \mathfrak{p} \le \infty$, $\lambda < \lambda_{\text{crit}}$. Singular or degenerate when $\nabla u = 0$, turning point at λ_{crit}.

Regularized variant in weak form

Find $u \in \boldsymbol{\mathcal{V}}_D$ such that

$$\int_\Omega \eta \nabla v \cdot \nabla u - \lambda v e^u - f v = 0 \qquad \forall v \in \boldsymbol{\mathcal{V}}_0$$

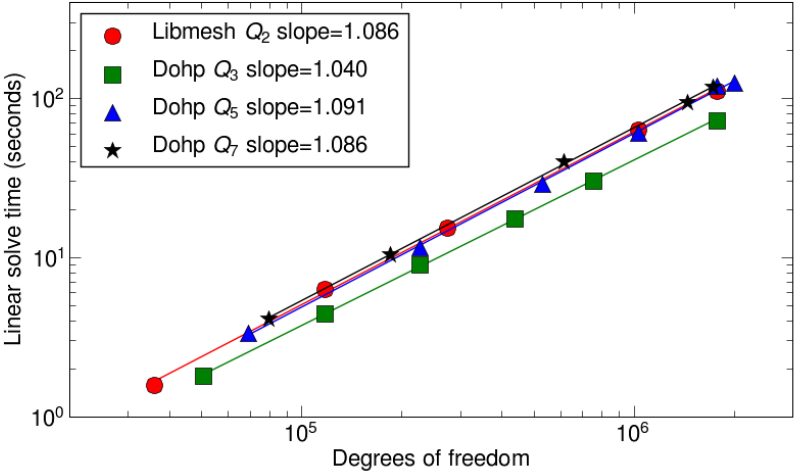
where $\eta(\gamma) = (\epsilon + \gamma)^{\frac{\mathfrak{p}-2}{2}}$, $\gamma = \frac{1}{2} \nabla u \cdot \nabla u$

Jacobian

$$v^T J(w) u \sim \int_\Omega \nabla v \cdot \left[\eta \mathbf{1} + \eta' \nabla w \otimes \nabla w \right] \cdot \nabla u - v \left[\lambda e^w \right] u$$

where $\eta' = \frac{\mathfrak{p}-2}{2} \eta / (\epsilon + \gamma)$.
Efficient matrix-free form: $[f_{i,j}] \sim (\eta, \sqrt{\eta'} \nabla w, \lambda e^w)$

Performance compared to conventional quadratic elements

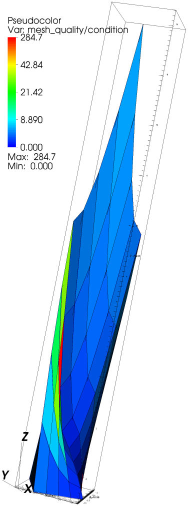


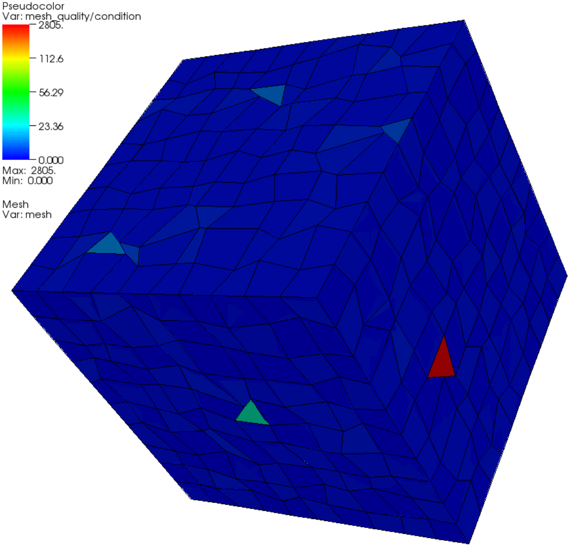
Profiling

Event	Libmesh Q_2	Dohp Q_3	Dohp Q_5	Dohp Q_7
Assembly	41	25	24	23
Krylov	111	73	119	117
MF MatMult	–	36	55	55
PCSetUp	16	10	12	9
PCApply	82	27	51	52
CG its	34	23	41	49
Mat nonzeros	111 M	44.7 M	44.7 M	44.3 M

Assembly and solve time (seconds) for a 3D Poisson problem with 121^3 degrees of freedom (120^3 for Q_7), relative tolerance of 10^{-8}.

Deformed meshes





Deformed meshes

order	8^3 brick ML/BMG	`twist` ML/BMG	`random` ML/BMG
Q_1	4/4	4/4	8/–
Q_2	24/24	25/23	27/27
Q_3	24/24	29/27	28/27
Q_4	29/28	39/30	34/33
Q_5	35/27	47/34	42/35
Q_6	35/29	60/40	43/41

Iteration counts for three different meshes using ML and BoomerAMG (BMG) preconditioning. For the Q_1 case, BoomerAMG failed, producing NaN.

Large variation in coefficients

Event	R-ML	R-ILU(0)	R-ILU(1)	E-ILU(0)	E-ILU(1)
Residual	65	35	29	57	51
Assembly	136	78	68	127	110
Krylov	287	295	274	187	166
MF MatMult	173	259	190	155	110
PCSetup	27	2	11	5	17
PCApply	86	26	67	27	39
Total time	496	413	374	377	333
Newton #	26	15	13	24	21
Residual #	46	25	20	40	36
Krylov #	606	911	667	545	386

Full nonlinear solve, 10^8 variation in η, Q_3 elements.

Power-law Stokes

- Strong form: Find $(\boldsymbol{u}, p) \in \boldsymbol{\mathcal{V}}_D \times \mathcal{P}$ such that

$$\begin{aligned} -\nabla\cdot(\eta D\boldsymbol{u}) + \nabla p - \boldsymbol{f} &= 0 \\ \nabla\cdot\boldsymbol{u} &= 0 \end{aligned}$$

where

$$\begin{aligned} D\boldsymbol{u} &= \tfrac{1}{2}\left(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^T\right) \\ \gamma(D\boldsymbol{u}) &= \tfrac{1}{2} D\boldsymbol{u} : D\boldsymbol{u} \\ \eta(\gamma) &= B(\Theta,\dots)\left(\epsilon + \gamma\right)^{\frac{\mathfrak{p}-2}{2}}, \quad \mathfrak{p} = 1 + \tfrac{1}{\mathfrak{n}} \approx \tfrac{4}{3} \end{aligned}$$

Power-law Stokes

Weak form of the Newton step

Find $(\boldsymbol{u}, p)$ such that

$$\int_\Omega D\boldsymbol{v} : \left[\eta \mathbf{1} + \eta' D\boldsymbol{w} \otimes D\boldsymbol{w}\right] : D\boldsymbol{u} \\ - p\nabla \cdot \boldsymbol{v} - q\nabla \cdot \boldsymbol{u} = -v \cdot F(\boldsymbol{w}) \qquad \forall (\boldsymbol{v}, q)$$

Matrix form

$$\begin{bmatrix} A(\boldsymbol{w}) & B^T \\ B & \end{bmatrix} \begin{pmatrix} u \\ p \end{pmatrix} = -\begin{pmatrix} F_u(\boldsymbol{w}) \\ 0 \end{pmatrix}$$

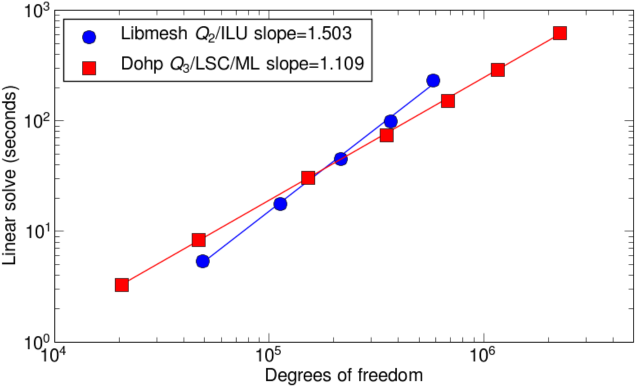
Block factorization

$$\begin{bmatrix} A & B^T \\ B & \end{bmatrix} = \begin{bmatrix} 1 & \\ BA^{-1} & 1 \end{bmatrix} \begin{bmatrix} A & B^T \\ & S \end{bmatrix} = \begin{bmatrix} A & \\ B & S \end{bmatrix} \begin{bmatrix} 1 & A^{-1}B^T \\ & 1 \end{bmatrix}$$

where the Schur complement is

$$S = -BA^{-1}B^T.$$

Power-law Stokes Scaling



Power-law Stokes Scaling

	6^3	12^3	16^3	20^3	24^3	30^3
Velocity dofs	20577	151959	352947	680943	1167051	2260713
Pressure dofs	343	2197	4913	9261	15625	29791
Assembly (s)	0.5	4.2	10	20	34	66
Krylov (s)	3.3	31	74	152	292	623
GMRES its	34	36	36	37	41	42

Assembly and linear-solve time (seconds) for a relative tolerance of 10^{-6} on the Stokes problem with mixed $Q_3 - Q_1$ elements using GMRES right-preconditioned with LSC and ML.

Conclusions

- High-order elements are fast, robust, and easy.
- They fit naturally into the JFNK framework.
- They are effective at utilizing modern hardware.
- Assembling matrices based on Q_2 or higher elements is rarely cost-effective (memory or time).

Ongoing work

- Code generation/AD
- Stabilized methods
- Moving domains
- Iterative substructuring (BDDC/FETI-DP)

Tools

- PETSc `http://mcs.anl.gov/petsc`
  - ML, Hypre, MUMPS
- ITAPS `http://itaps.org`
  - MOAB, CGM, Lasso