From symbolic mathematics to fast solvers for finite element problems

Lawrence Mitchell[1,*]

C.J. Cotter, P.E. Farrell, D.A. Ham, M. Homolya, P.H.J. Kelly, R.C. Kirby, F. Wechsung ...

18th September 2018

[1]Department of Computer Science, Durham University
*lawrence.mitchell@durham.ac.uk

Outline

- Automated finite elements
  - Fast implementations thereof
- Multilevel solvers for coupled problems
  - Stationary Navier-Stokes
- Future directions

Code that looks like maths

Find $(u, p, T) \in V \times W \times Q$ s.t.

$$\int \nabla u \cdot \nabla v + (u \cdot \nabla u) \cdot v$$

$$-p\nabla \cdot v + \frac{\mathrm{Ra}}{\mathrm{Pr}} T g\hat{z} \cdot v \,\mathrm{d}x = 0$$

$$\int \nabla \cdot u q \,\mathrm{d}x = 0$$

$$\int (u \cdot \nabla T) S + \mathrm{Pr}^{-1} \nabla T \cdot \nabla S \,\mathrm{d}x = 0$$

$$\forall\, (v, q, T) \in V \times W \times Q$$

```
from firedrake import *
mesh = Mesh(...)
V = VectorFunctionSpace(mesh, "CG", 2)
W = FunctionSpace(mesh, "CG", 1)
Q = FunctionSpace(mesh, "CG", 1)
Z = V * W * Q
Ra = Constant(...)
Pr = Constant(...)
upT = Function(Z)
u, p, T = split(upT)
v, q, S = TestFunctions(Z)
bcs = [...]

F = (inner(grad(u), grad(v))
     + inner(dot(grad(u), u), v)
     - inner(p, div(v))
     + (Ra/Pr)*inner(T*g, v)
     + inner(div(u), q)
     + inner(dot(grad(T), u), S)
     + (1/Pr) * inner(grad(T), grad(S)))*dx

solve(F == 0, upT, bcs=bcs)
```

Finite elements I

$$F(u) = 0 \text{ in } \Omega$$

$$+ \text{ boundary conditions}$$

Seek weak solution in some space of functions $V(\Omega)$.

Now we need to solve the (infinite dimensional) problem, find $u \in V$ s.t.

$$\int_\Omega F(u) v \, \mathrm{d}x = 0 \quad \forall \, v \in V$$

Choose finite dimensional subspace $V_h \subset V$, find $u_h \in V_h$ s.t.

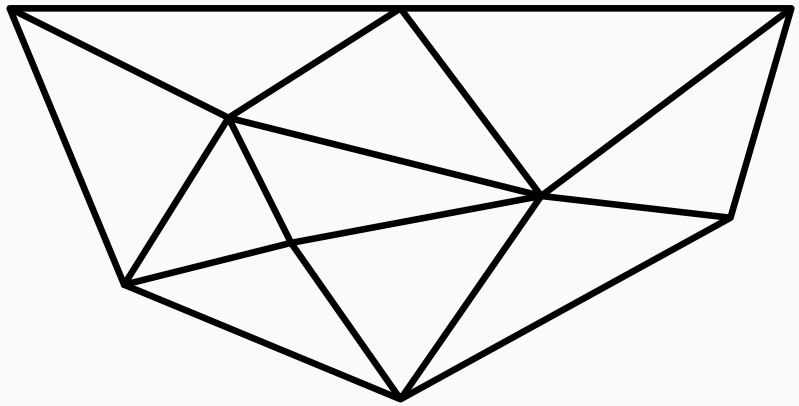
$$\int_\Omega F(u_h) v_h \, \mathrm{d}x = 0 \quad \forall \, v_h \in V_h$$

Finite elements II

Divide domain Ω...

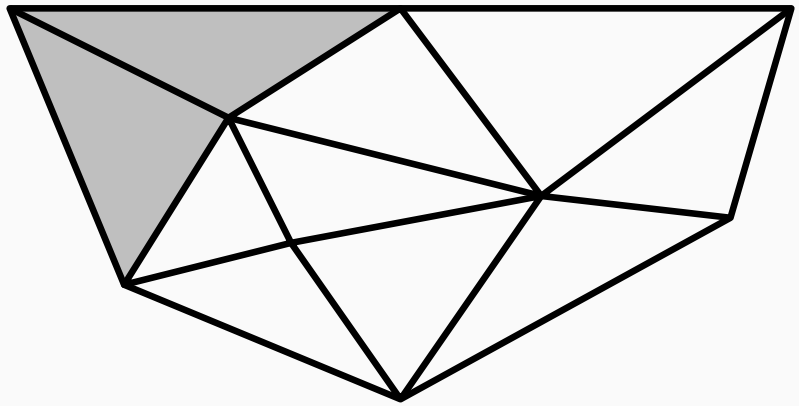
Finite elements II

...into triangulation $\mathcal{T}$...



Finite elements II

...and choose basis with finite support.



Finite elements III

Integrals become sum over element integrals

$$\int_\Omega F(u_h)v_h \,\mathrm{d}x = \sum_{e\in\mathcal{T}} \int_e F(u_h)v_h \,\mathrm{d}x$$

(Usually) perform element integrals with numerical quadrature

$$\int_e F(u_h)v_h \,\mathrm{d}x = \sum_q w_q F(u_h(q))v_h(q) \,\mathrm{d}x$$

Replace $u_h(q), v_h(q)$ with expansion in finite element basis

$$u_h(q) = \sum_i u_h^i \phi_i(q)$$

$$v_h(q) = \phi_j(q)$$

Looking for abstractions

- Mathmatics just says “here is the integral to compute on each element, do that everywhere”
- Computer implementation chooses how to do so

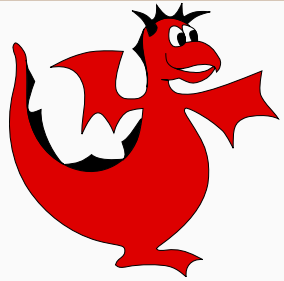
Assertion(s)
Having chosen a discretisation, writing the element integral is “mechanical”.

With an element integral in hand, integrating over a mesh is “mechanical”.

Corollary
Computers are good at mechanical things, why don’t we get the computer to write them for us?

Firedrake `www.firedrakeproject.org`

[...] *an automated system for the solution of partial differential equations using the finite element method.*



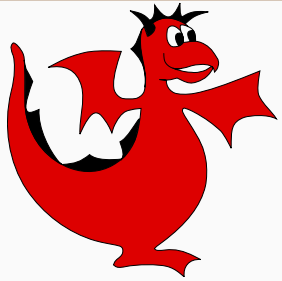
- Written in Python.
- Finite element problems specified with *embedded* domain specific language, UFL (Alnæs, Logg, Ølgaard, Rognes, and Wells 2014) from the FEniCS project.
- *Runtime* compilation to optimised, low-level (C) code.
- PETSc for meshes and (algebraic) solvers.

F. Rathgeber, D.A. Ham, **LM**, M. Lange, F. Luporini, A.T.T. McRae, G.-T. Bercea, G.R. Markall, P.H.J. Kelly. ACM Transactions on Mathematical Software, 2016.

`arXiv:1501.01809[cs.MS]`

Firedrake `www.firedrakeproject.org`

> [...] *an automated system for the solution of partial differential equations using the finite element method.*



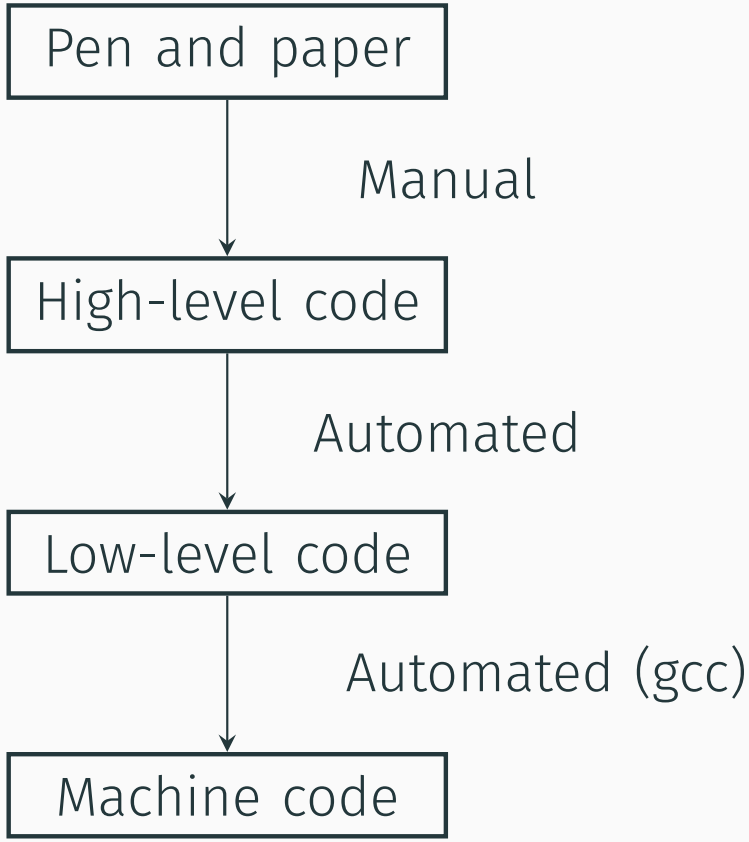
User groups at
Imperial, Oxford, Bath, Leeds, Durham, Kiel, Rice, Houston, Exeter, Buffalo, Waterloo, Minnesota, Baylor, Texas A&M, ...

Annual user & developer meeting this year had 45 attendees.

More automation

$$a(u, v) = \int_\Omega \nabla u \cdot \nabla v \, \mathrm{d}x \quad \forall v \in V$$

```
V = FiniteElement("Lagrange",
                  triangle, 1)
u = TrialFunction(V)
v = TestFunction(V)
F = dot(grad(u), grad(v))*dx
```



Automation enables optimisations

Compile UFL (symbolics) into low-level code (implementation).

M. Homolya, **LM**, F. Luporini, D.A. Ham. SIAM SISC (2018).

`arXiv:1705.03667[cs.MS]`

- Element integral

$$\int_e \nabla u \cdot \nabla v \, \mathrm{d}x$$

```
V = FiniteElement("Lagrange", triangle, 1)
u = TrialFunction(V)
v = TestFunction(V)
a = dot(grad(u), grad(v))*dx
```

- Is transformed to a tensor algebra expression

$$\sum_q w_q \, |d| \sum_{i_5} \left(\sum_{i_3} K_{i_3,i_5} \left[E^{(1)}_{q,k} \quad E^{(2)}_{q,k} \right]_{i_3} \right) \left(\sum_{i_4} K_{i_4,i_5} \left[E^{(1)}_{q,j} \quad E^{(2)}_{q,j} \right]_{i_4} \right)$$

- Multiple optimisation passes aim to minimise FLOPs required to evaluate this expression.

Optimisation passes

Sum factorisation

Spectral elements typically use *tensor product* basis functions

$$\phi_{i,q} := \phi_{(j,k),(p,r)} = \varphi_{j,p}\varphi_{k,r}$$

Tensor algebra maintains this structure, compiler exploits it for low-complexity evaluation of integrals.

Loop transformations

Solve ILP problem to reorder loops, and decide how to split complicated expressions.

Vectorisation

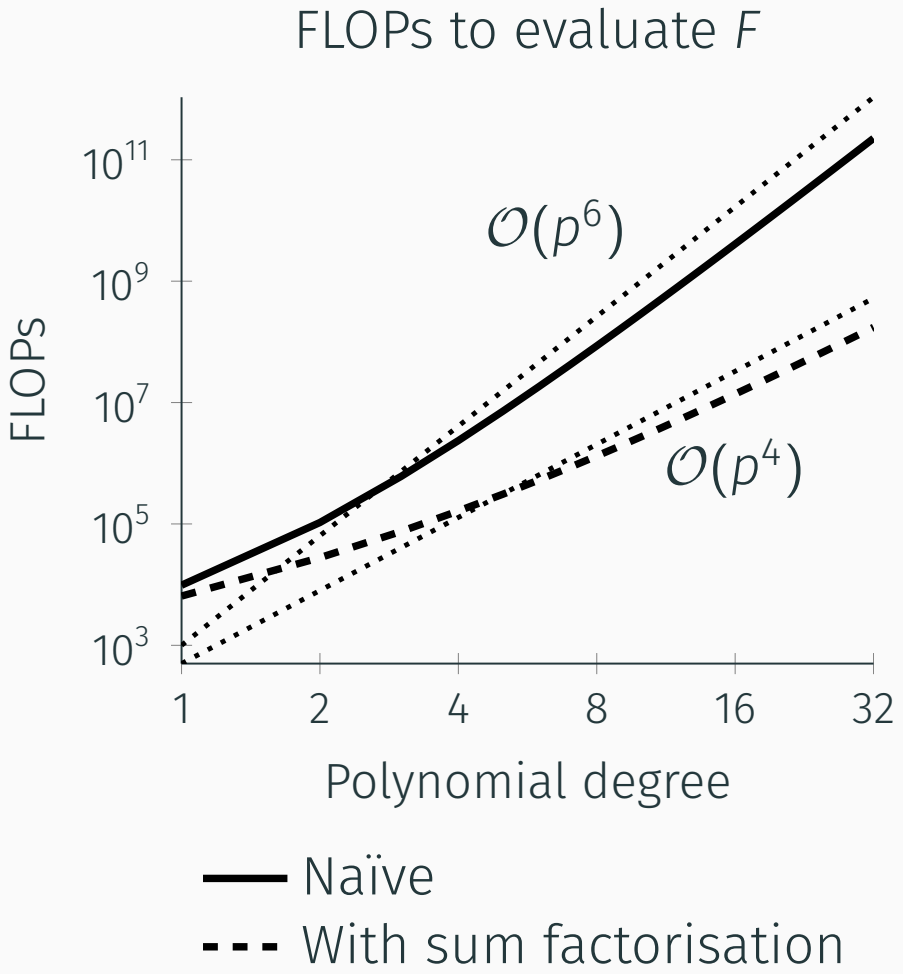
Low-level rearrangement to ease the job of C compilers in generating efficient code.

Algorithmically optimal implementation

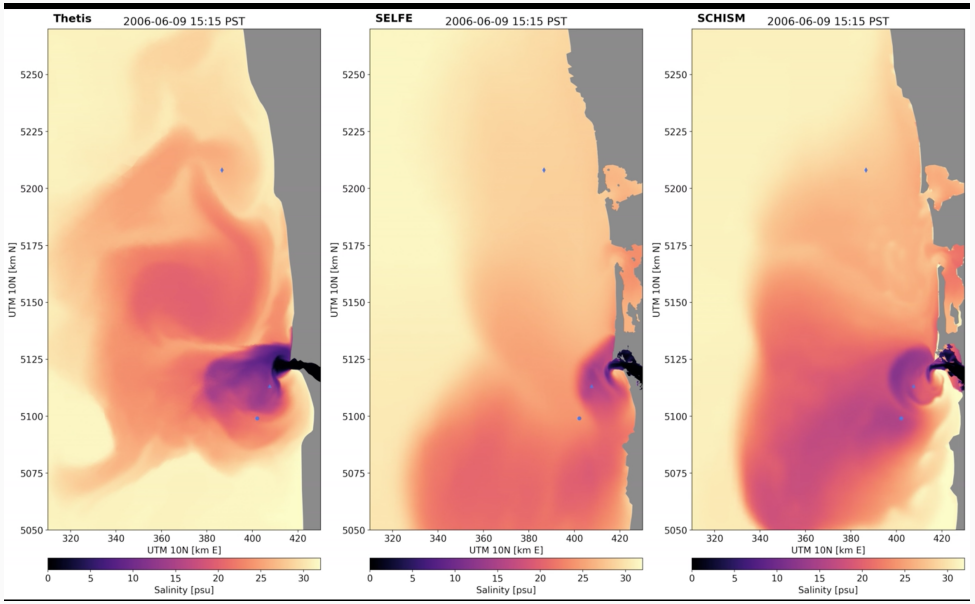
Computing curl − curl action
Find $u \in V \subset H(\mathbf{curl})$ s.t.

$$\int \mathbf{curl}\, u \cdot \mathbf{curl}\, v \, \mathrm{d}x = \int B \cdot v \, \mathrm{d}x \quad \forall v \in V.$$

```
NCE = FiniteElement("NCE", hexahedron, degree)
Q = VectorElement("Q", hexahedron, degree)
u = Coefficient(NCE)
B = Coefficient(Q)
v = TestFunction(NCE)
F = (dot(curl(u), curl(v)) - dot(B, v))*dx
```



Application: coastal ocean modelling



Surface salinity of the Columbia river plume. Credit T. Kärnä, Finnish Meteorological Institute

`thetisproject.org`

- Lower numerical mixing, improved results.
- 3-8x faster than models with similar quality of results.
- Differentiable: efficient adjoint.

THETIS

T. Kärnä, S.C. Kramer, **LM**, D.A. Ham, M.D. Piggott, and A.M. Baptista. To appear in Geoscientific Model Development, 2018. `arXiv:1711.08552 [physics.ao-ph]`

Solving sparse systems

Application challenge

Stationary incompressible Navier-Stokes
Find $(u, p) \in H^1(\Omega; \mathbb{R}^d) \times L^2(\Omega)$ such that

$$\begin{aligned} -\nu\nabla^2 u + (u \cdot \nabla)u + \nabla p &= f && \text{in } \Omega, \\ \nabla \cdot u &= 0 && \text{in } \Omega, \\ u &= g && \text{on } \Gamma_D, \\ \nu\nabla u \cdot n &= pn && \text{on } \Gamma_N, \end{aligned}$$

Discretise and linearise
Need to solve linear saddle point system

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

Challenges for solvers

Desired properties

- Growth in time to solution is (at worst) $\mathcal{O}(n \log n)$.
- Convergence does not degree with $\nu \to 0$.

Direct methods

✓ Convergence independent of ν

✗ Time to solution $\mathcal{O}(n^2)$ (3D).

Krylov methods

✓ Time to solution $\mathcal{O}(n \log n)$ (with multilevel preconditioner)

✗ Convergence independent of ν challenging

Challenges for solvers

Want ν- and h-independent convergence.

Monolithic multigrid: Vanka, Braess-Sarazin smoothing

- ✓ h-independent
- ✗ Convergence degrades with $\nu \to 0$.

Block factorisations

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

Challenges, good approximations to A^{-1} and S^{-1}.

Block factorisations, approximating S^{-1}

Pressure mass ($S^{-1} \approx M_p^{-1}$)

- ✓ h-independent
- ✗ Convergence degrades rapidly with $\nu \to 0$

PCD, LSC ($S^{-1} \approx M_p^{-1} F_p L_p^{-1}$)

- ✓ h-independent
- ✗ Convergence degrades with $\nu \to 0$.

Block factorisations, approximating S^{-1}

Augmented Lagrangian
Replace $A \to A + \gamma B^T M_p^{-1} B =: A_\gamma$:

$$P_\gamma := \begin{pmatrix} A + \gamma B^T M_p^{-1} B & B^T \\ B & 0 \end{pmatrix} \begin{pmatrix} \delta u \\ \delta p \end{pmatrix} = \begin{pmatrix} b \\ 0 \end{pmatrix}.$$

Discrete solution unchanged, since $B\delta u = 0$.

Then $S^{-1} \approx (\nu + \gamma) M_p^{-1}$ (gets better as $\gamma \to \infty$).

- ✓ h- and ν-independent
- ✗ A_γ is *much* harder to invert than A.

Developed 3D multigrid scheme for A_γ extending 2D preconditioner of Benzi and Olshanskii (2006).

A multigrid solver for A_γ I: robust smoother

grad – div term has large kernel: point smoothers not robust.

Theorem (Schöberl (1999), Benzi & Olshanksii (2006))
A smoother robust wrt ν and γ requires a subspace decomposition

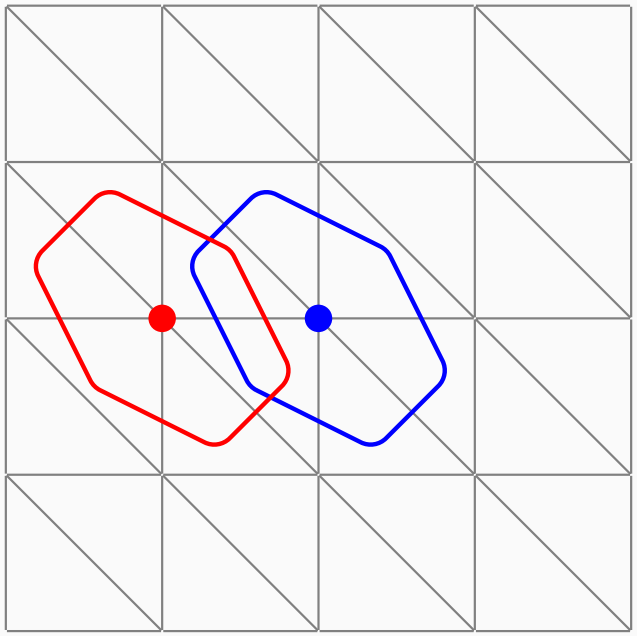
$$V = \sum_i V_i$$

such that the kernel is spanned by the subspaces

$$\{u \in V : (\nabla \cdot u, \nabla \cdot v) = 0 \; \forall \; v \in V\} =: \mathcal{N} = \sum_i (V_i \cap \mathcal{N})$$

Star smoother

- Want a *local basis* for the kernel: $\mathbb{P}_0$ pressure.
- Kernel spanned by functions on patch of elements around each vertex.
- Use overlapping additive Schwarz smoother of “star” patches.

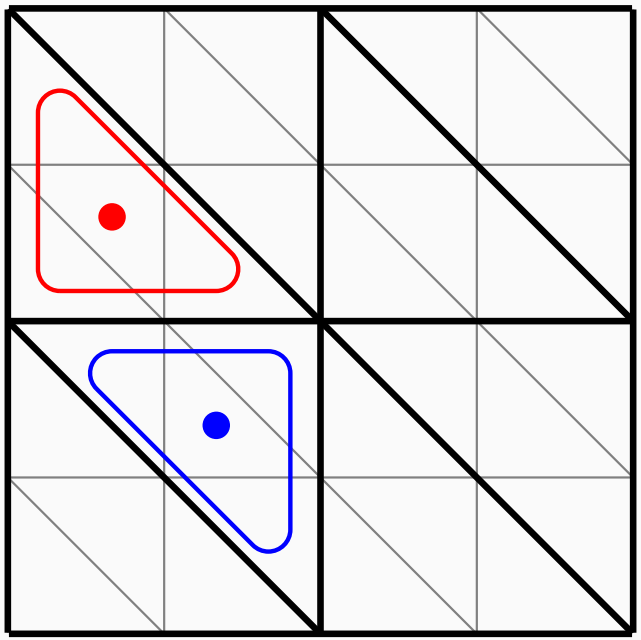


A multigrid solver for A_γ II: robust prolongation

Theorem (Schöberl (1999))
Let $E_H : V_H \to V_h$, a robust multigrid cycle requires, $\forall u_H \in V_H$:

$$\nu\|\nabla(E_H u_H)\|_{L^2}^2 + \gamma\|\nabla \cdot (E_H u_H)\|_{L^2}^2 \leq c(\nu\|\nabla(u_H)\|_{L^2}^2 + \gamma\|\nabla \cdot u_H\|_{L^2}^2)$$

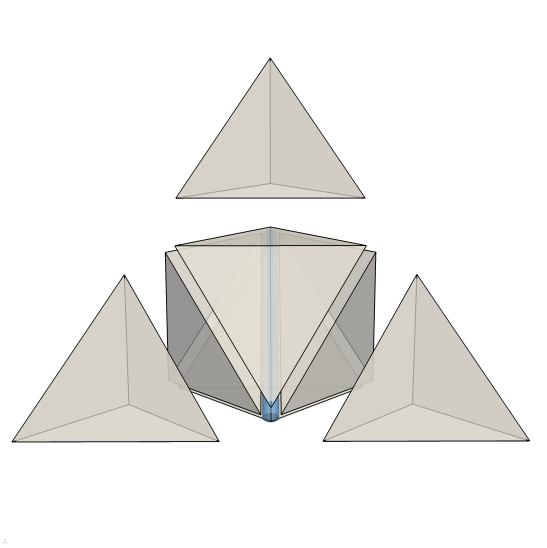
Can fix by solving a small Stokes problem for a correction in the *interior* of each coarse cell.



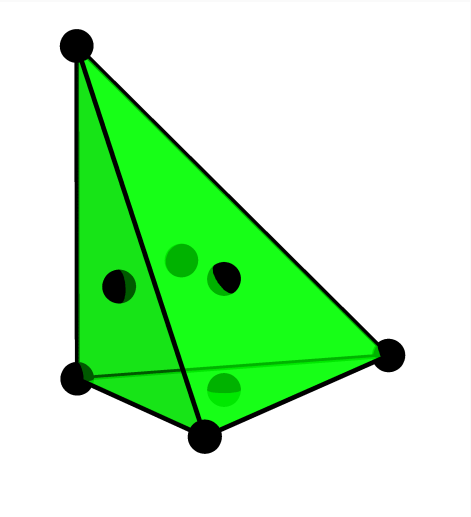
3D element pair

Problem
$[\mathbb{P}_2]^3 - \mathbb{P}_0$ is *not* inf-sup for the local Stokes problems.

$[\mathbb{P}_3]^3 - \mathbb{P}_0$ is, but is tremendously expensive for only second order convergence in the velocity.



Solution
We use an affine-equivalent analog of the Bernard-Raugel element: $\left[\mathbb{P}_1 \oplus B_3^F\right]^3 - \mathbb{P}_0$.



Preconditioner for flexible additive Schwarz smoothers

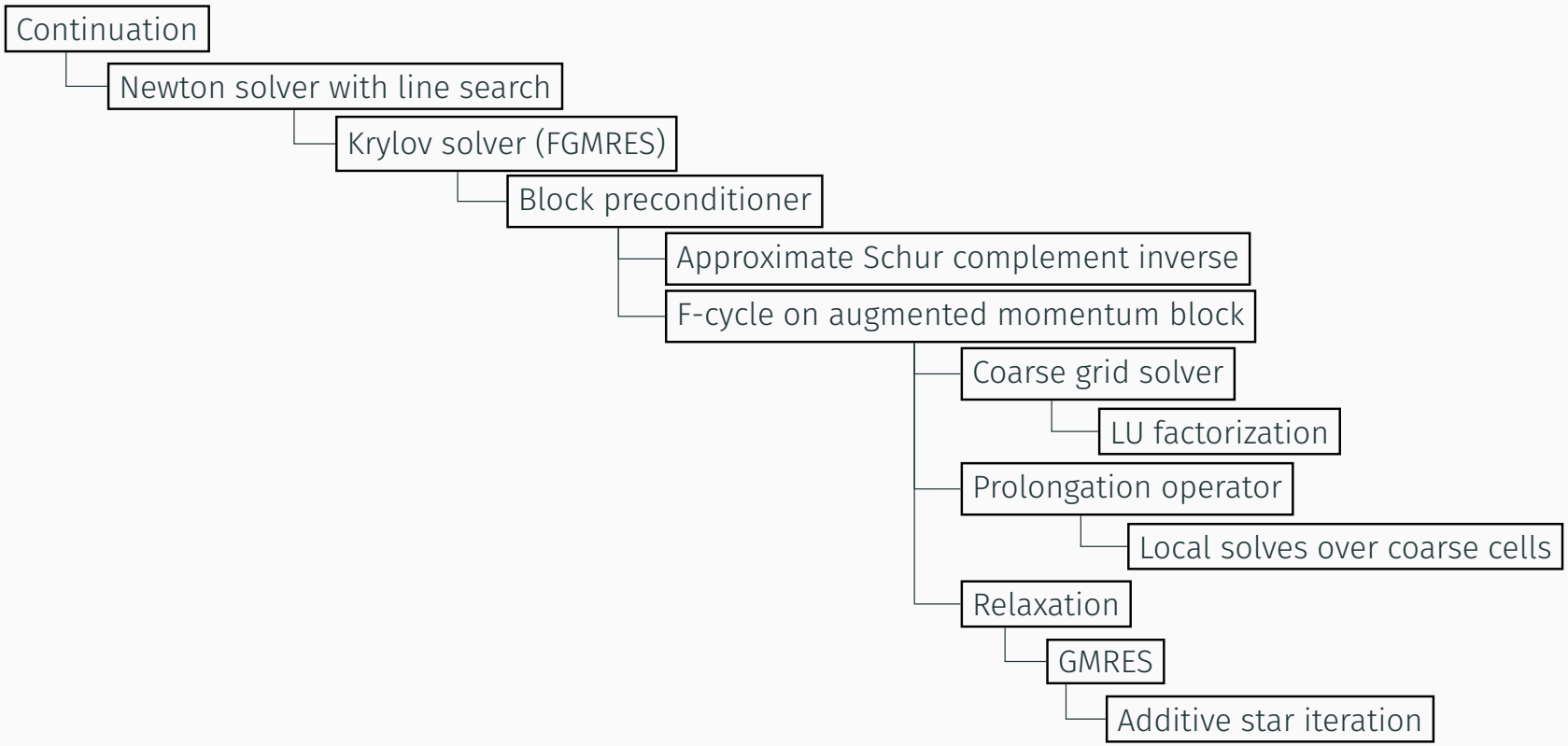
- Solver needs additive Schwarz smoothers in two places.
- New in PETSc 3.10 `-pc_type patch`

Extensible callback interface

- Separate subspace decompostion (topology + discretisation)
- from equation assembly
- Extensible mechanism for defining patches
- Flexibly supports Vanka, line- and plane-smoothers, ...
- Easy experimentation!

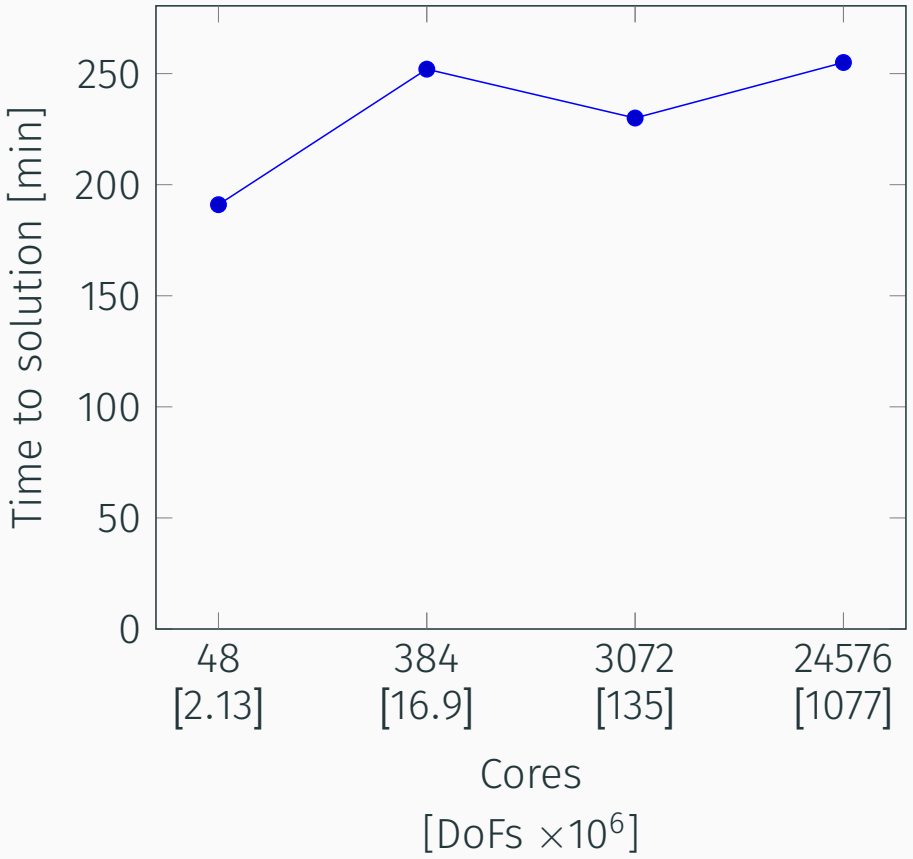
LM, M.G. Knepley, P.E. Farrell, In preparation.

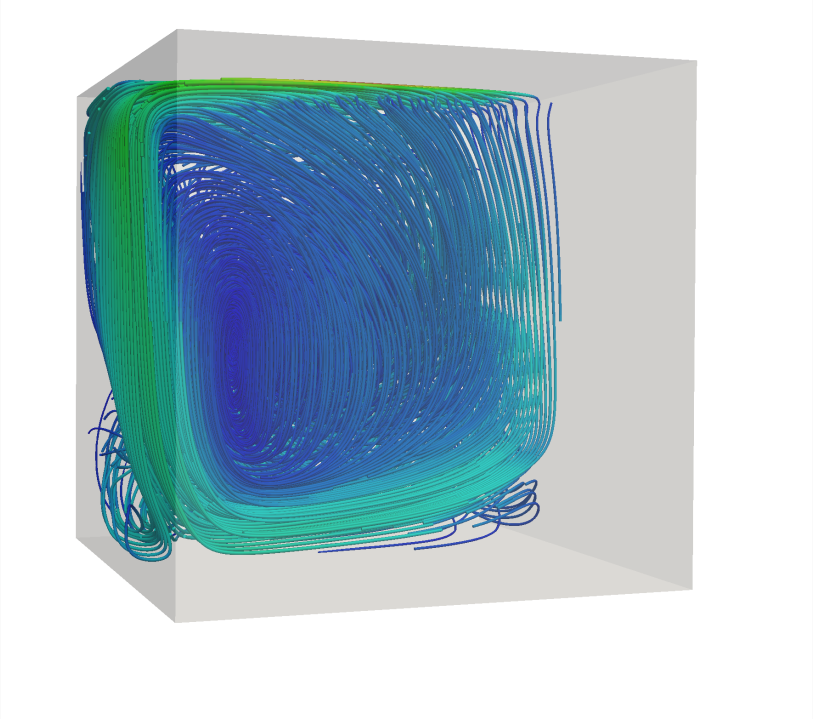
Multilevel solver




```
$ count-lines .
Language  files  blank  comment  code
Python        4    108      153   558
```

Results: 3D lid-driven cavity





Velocity streamlines at Reynolds number 5000. Credit F. Wechsung, University of Oxford.

P.E. Farrell, **LM**, and F. Wechsung. In preparation

Results: 3D lid-driven cavity

# refinements	# dofs	Reynolds number				
		10	100	1000	2500	5000
1	2.1×10^6	7.50	7.33	7.50	7.00	6.50
2	1.7×10^7	8.50	7.00	7.50	6.50	5.50
3	1.3×10^8	7.00	7.00	6.50	5.00	6.50
4	1.1×10^9	7.00	7.33	5.50	4.00	9.00

Table 1: Average Krylov iterations per Newton step

P.E. Farrell, **LM**, and F. Wechsung. In preparation

Going forward

- $\left[\mathbb{P}_1 \oplus B_3^F\right]^3 - \mathbb{P}_0$ is not a great discretisation
- Same ideas should apply to *pressure robust* schemes (e.g. Scott-Vogelius, $H(\mathbf{div}) - L^2$ mixed methods)
- Application of same ideas directly to nonlinear multigrid?

Idea

- Preconditioners are formulated in mathematics on paper
- They *should* be formulated in mathematics on computer

Conclusions & directions

Automated finite elements

- Enable significant experimentation
- Better code than many humans
- Productive mechanism to collaborate with computer science

Fast solvers

- How to capture mathematical “building blocks” in code?
- Each class PDE presents different challenges, can we avoid doing everything “from scratch” each time?