Solver composition across the PDE/linear algebra barrier

Lawrence Mitchell\textsuperscript{1}  Rob Kirby\textsuperscript{2}

\textsuperscript{1}Departments of Computing and Mathematics, Imperial College London

\textsuperscript{2}Department of Mathematics, Baylor University

\textbf{16 15 June 2017}
Thanks

- NSF CCF/SHF program
- PRISM at Imperial College, London (http://prism.ac.uk)
Software libraries simplify writing models

Stationary Rayleigh-Bénard convection

\[-\Delta u + u \cdot \nabla u + \nabla p + \frac{Ra}{Pr} \hat{g} T = 0\]
\[\nabla \cdot u = 0\]
\[-\frac{1}{Pr} \Delta T + u \cdot \nabla T = 0\]

```python
from firedrake import *

mesh = Mesh(...)  
V = VectorFunctionSpace(mesh, "CG", 2) 
W = FunctionSpace(mesh, "CG", 1) 
Q = FunctionSpace(mesh, "CG", 1) 
Z = V * W * Q

upT = Function(Z) 
u, p, T = split(upT) 
v, q, S = TestFunctions(Z)

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=bc)
```
What about the solver?

- Black boxes (LU, AMG) rarely work on coupled problems at scale.
- Block factorizations common.
- Gives many knobs to tweak, may require problem-specific auxiliary operators.
Block preconditioning

- (Approximate) block LU factorizations.
- Why? Krylov methods converge fast on operators with low-degree minimal polynomials.

Murphy, Golub, and A. J. Wathen 2000

\[
\begin{bmatrix}
A & 0 \\
0 & CA^{-1}B^T
\end{bmatrix}^{-1}
\begin{bmatrix}
A & B^T \\
C & 0
\end{bmatrix}
\]

has at most four distinct eigenvalues.
What does this mean?

- Invert diagonal blocks (smaller problems).
- Off-diagonal blocks: only require *action*
- The most efficient (time to solution) strategy is problem and parameter dependent:
  - Do I invert the blocks well?
  - Do I invert them inexacty?
- Need to be able to *configure* the solver without changing the code (e.g. eliminating first row or second?)
- Need to treat nested problems.
Jacobian from Newton linearization:

\[
J = \begin{bmatrix}
F & B^T & M_1 \\
C & 0 & 0 \\
M_2 & 0 & K
\end{bmatrix}.
\]

- Navier-Stokes (top left $2 \times 2$)
- Convection-diffusion for temperature (bottom right)
- Coupling in $M_1$ and $M_2$ (non-symmetric).
Write

\[ N = \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix} \quad \tilde{M}_1 = \begin{bmatrix} M_1 \\ 0 \end{bmatrix} \quad \tilde{M}_2 = \begin{bmatrix} M_2 & 0 \end{bmatrix} \]

and eliminate the Navier-Stokes block, giving system for temperature:

\[ S_T = K - \tilde{M}_2 N^{-1} \tilde{M}_1. \]

Howle and Kirby 2012 $K^{-1}$ is a good preconditioner for $S_T$. 
Krylov iterations

Solve the (left-)preconditioned system

\[ P^{-1}Jx = P^{-1}b. \]

Write \( K(A, A) \sim A^{-1} \) to denote an iteration \( K \) on \( A \) using \( A \) to construct the preconditioner. Then

\[
P^{-1} = \begin{bmatrix}
K(N, N) & 0 \\
0 & I
\end{bmatrix}
\begin{bmatrix}
I & -\tilde{M}_1 \\
0 & I
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
0 & K(K, K)
\end{bmatrix}
\]

is a block-multiplicative combination of \( K(N, N) \) and \( K(K, K) \).
A lower Schur complement factorisation of $N$ is a good option. Requires $\mathcal{K}(F, \mathbb{F})$ and $\mathcal{K}(S_p, \mathbb{S})$ where $S_p = -CF^{-1}B^T$. One option is the *pressure convection-diffusion* approximation:

$$\mathcal{K}(S_p, \mathbb{S}) = \mathcal{K}(L_p, L) F_p \mathcal{K}(M_p, M),$$

giving

$$\mathcal{K}(N, N) = \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(L_p, L) F_p \mathcal{K}(M_p, M) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix}.$$
Things to note

- Only inverses are diagonal blocks.
- Off-diagonal: matrix-free?
- How to control nested iterations?
- PCD drives an axe through the horizontal split between the PDE library and the solver library.

PCD

Needs the auxiliary discretised operator $F_p$ and approximate inverses of the auxiliary operators $L_p$ and $M_p$. 
PETSc already provides a highly runtime-configurable library for \textit{algebraically} composing solvers (Brown et al. 2012). Firedrake makes it straightforward to build any auxiliary operators.

\textbf{A new matrix type}

We create a new PETSc matrix type in Firedrake that remembers the symbolic Jacobian it comes from, and implements matrix-free actions.

\textbf{Firedrake-aware preconditioners}

Algebraic preconditioners cannot work on shell matrices, so we create custom preconditioners that can inspect the symbolic information and do the appropriate thing.
Contexts: deal between friends
Poisson Matrix-free works: Lower memory usage

![Graph showing memory usage comparison between AIJ and Matrix-free methods for 2D and 3D problems as a function of polynomial degree.](image)

- **AIJ [2D]**
- **AIJ [3D]**
- **Matrix-free [2D]**
- **Matrix-free [3D]**
Poisson Matrix-free works: Runtime isn’t much worse

![Graph showing the comparison of different matrix assembly and multiplication methods for 2D and 3D cases. The x-axis represents the polynomial degree, ranging from 1 to 7, and the y-axis represents the time per degree of freedom (time/dof) in seconds (s), ranging from $10^{-9}$ to $10^{-7}$. The graph compares the Assemble AIJ, MatMult AIJ, and MatMult matrix-free methods for both 2D and 3D cases, showing how runtime increases with higher polynomial degrees.]
RB Matrix-free works: Lower memory usage

Polynomial degree of scalar space

|        | Bytes/dof |
|--------|-----------|
| AIJ [2D] |           |
| Matrix-free [2D] |           |
| Nest [2D] |           |
| AIJ [3D] |           |
| Matrix-free [3D] |           |
| Nest [3D] |           |
RB Matrix-free works: Runtime isn’t much worse

Polynomial degree of scalar space

Time/dof [s]

Assemble AIJ [2D]
MatMult AIJ [2D]
MatMult matrix-free [2D]
Assemble Nest [2D]
MatMult Nest [2D]

Assemble AIJ [3D]
MatMult AIJ [3D]
MatMult matrix-free [3D]
Assemble Nest [3D]
MatMult Nest [3D]
```python
class PCDPC(PCBase):
    def initialize(self, pc):
        _, P = pc.getOperators()
        ctx = P.getContext()
        appctx = ctx.appctx
        p, q = ctx.arguments()
        M_p = assemble(p*q*dx)
        L_p = assemble(inner(grad(p), grad(q))*dx)
        M_ksp = KSP().create()
        M_ksp.setOperators(M_p)
        L_ksp = KSP().create()
        L_ksp.setOperators(L_p)
        # Some boilerplate elided
        [\ldots]
        u0 = split(appctx["state"])[appctx["velocity_space"]]
        F_p = assemble(inner(grad(p), grad(q))*dx + inner(u0, grad(p))*q*dx)

    def apply(self, pc, x, y):
        # y \leftarrow \mathcal{K}(L_p, \mathbb{L}) F_p \mathcal{K}(M_p, \mathbb{M}) x
        a, b = self.workspace
        self.M_ksp.solve(x, a)
        self.F_p.mult(a, b)
        self.L_ksp.solve(b, y)
```

How to configure things

PETSc provides a “programming language” for configuring objects at runtime. It has two operations

1. Value assignment
2. String concatenation

Every object has an options prefix which controls where in the options database it looks for configuration values. This is verbose, but a very powerful idea. We can control the types of individual solves by ensuring that they have different prefixes.
class PCDPC(PCBase):
    def initialize(self, pc):
        ...
        prefix = pc.getOptionsPrefix()
        M_ksp.setOptionsPrefix(prefix + "pcd_Mp")
        M_ksp.setFromOptions()
        L_ksp.setOptionsPrefix(prefix + "pcd_Lp")
        L_ksp.setFromOptions()
Back to the main event

We want to invert $J$ using $\mathcal{K}(J, \mathbb{J})$ where $J = \begin{bmatrix} F & B^T & M_1 \\ C & 0 & 0 \\ M_2 & 0 & K \end{bmatrix}$

and we are using

$$\mathcal{K}(J, \mathbb{J}) = \begin{bmatrix} \mathcal{K}\left(\begin{bmatrix} F & B^T \\ C & 0 \\ 0 \end{bmatrix}, \mathbb{N}\right) \ 0 \\ 0 \ 0 \ 0 \\ 0 \ 0 \ 0 \end{bmatrix} \begin{bmatrix} l & 0 & -M_1 \\ 0 & l & 0 \\ 0 & 0 & l \end{bmatrix} \begin{bmatrix} l & 0 & 0 \\ 0 & l & 0 \\ 0 & 0 & \mathcal{K}(K, \mathbb{K}) \end{bmatrix}$$

with

$$\mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(L_p, \mathbb{L}) \ F_p \mathcal{K}(M_p, \mathbb{M}) \end{bmatrix} \begin{bmatrix} l & 0 \\ -C & l \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & l \end{bmatrix}$$
First, the temperature solve

\( \mathcal{K}(J, J) = \begin{bmatrix} \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, \mathbb{N} \right) & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \mathcal{K}(K, \mathbb{K}) \)
First, the temperature solve

\[ \mathcal{K}(J, J) = \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, N \right) 0 \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ I & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \mathcal{K}(K, K) \end{bmatrix} \]
First, the temperature solve

\[ \mathcal{K}(J, J) = \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, N \right) \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right] \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ I & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \mathcal{K}(K, K) \end{bmatrix} \]

- \texttt{mat_type matfree}
- \texttt{ksp_type fgmres}
- \texttt{pc_type fieldsplit}
- \texttt{pc_fieldsplit_type multiplicative}
- \texttt{pc_fieldsplit_0_fields 0,1}
- \texttt{pc_fieldsplit_1_fields 2}
- \texttt{fieldsplit_1_ksp_type gmres}
- \texttt{fieldsplit_1_pc_type python}
- \texttt{fieldsplit_1_pc_python_type firedrake.AssembledPC}
- \texttt{fieldsplit_1_assembled_pc_type hypre}
First, the temperature solve

\[ \mathcal{K}(J, \mathbb{J}) = \left[ \begin{array}{c}
\mathcal{K} \left( \left[ \begin{array}{cc}
F & B^T \\
C & 0
\end{array} \right], \mathbb{N} \right) \mathbb{I} \\
0 & 0
\end{array} \right] \left[ \begin{array}{ccc}
I & 0 & -M_1 \\
0 & I & 0 \\
0 & 0 & I
\end{array} \right] \left[ \begin{array}{ccc}
I & 0 & 0 \\
0 & I & 0 \\
0 & 0 & \mathcal{K}(K, \mathbb{K})
\end{array} \right] \right] \]
First, the temperature solve

\[ K(J, J) = K \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, N \right) \begin{bmatrix} 0 \\ 0 \end{bmatrix} + \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & K(K, K) \end{bmatrix} \]

-mat_type matfree
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
-fieldsplit_1_ksp_type gmres
-fieldsplit_1_pc_type python
-fieldsplit_1_pc_python_type firedrake.AssembledPC
-fieldsplit_1_assembled_pc_type hypre
First, the temperature solve

\[
\mathcal{K}(J, \mathbb{J}) = \begin{bmatrix} \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, \mathbb{N} \right) 0 \end{bmatrix} \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \end{bmatrix} \mathcal{K} (K, \mathbb{K})
\]

- **mat_type** matfree
- **ksp_type** fgmres
- **pc_type** fieldsplit
- **pc_fieldsplit_type** multiplicative
- **pc_fieldsplit_0_fields** 0,1
- **pc_fieldsplit_1_fields** 2
- **fieldsplit_1_ksp_type** gmres
- **fieldsplit_1_pc_type** python
- **fieldsplit_1_pc_python_type** firedrake.AssembledPC
- **fieldsplit_1_assembled_pc_type** hypre
First, the temperature solve

\[ \mathcal{K}(J, \mathbb{J}) = \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, \mathbb{N} \right) 0 \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} I & 0 & 0 \\ 0 & I & 0 \\ 0 & 0 & \mathcal{K}(K, \mathbb{K}) \end{bmatrix} \]

- mat_type matfree
- ksp_type fgmres
- pc_type fieldsplit
- pc_fieldsplit_type multiplicative
- pc_fieldsplit_0_fields 0,1
- pc_fieldsplit_1_fields 2
- fieldsplit_1_ksp_type gmres
- fieldsplit_1_pc_type python
- fieldsplit_1_pc_python_type firedrake.AssembledPC
- fieldsplit_1_assembled_pc_type hypre
First, the temperature solve

\[ \mathcal{K}(J, \mathcal{J}) = \left[ \mathcal{K} \left( \begin{bmatrix} F & B^T \\ C & 0 \end{bmatrix}, \mathcal{N} \right) \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right] \begin{bmatrix} I & 0 & -M_1 \\ 0 & I & 0 \end{bmatrix} \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix} \begin{bmatrix} I \\ 0 \\ 0 \end{bmatrix} \mathcal{K}(K, \mathcal{K}) \]

-mat_type matfree
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
-fieldsplit_1_ksp_type gmres
-fieldsplit_1_pc_type python
-fieldsplit_1_pc_python_type firedrake.AssembledPC
-fieldsplit_1_assembled_pc_type hypre
Now the Navier-Stokes block

\[
\mathcal{K}(N, \bar{N}) = \begin{bmatrix}
F & 0 \\
0 & \underbrace{\mathcal{K}(L_p, \bar{L}) F_p \mathcal{K}(M_p, \bar{M})}_{\mathcal{K}(S_p, \bar{S})}
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix}
\begin{bmatrix}
\mathcal{K}(F, \bar{F}) & 0 \\
0 & I
\end{bmatrix}
\]

- fieldsplit_0_ksp_type gmres
- fieldsplit_0_pc_type fieldsplit
- fieldsplit_0_pc_fieldsplit_type schur
- fieldsplit_0_pc_fieldsplit_schur_fact_type lower
- fieldsplit_0_fieldsplit_0_ksp_type preonly
- fieldsplit_0_fieldsplit_0_pc_type python
- fieldsplit_0_fieldsplit_0_assembled_pc_type hypre
- fieldsplit_0_fieldsplit_1_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pc_type python
- fieldsplit_0_fieldsplit_1_pc_python_type firedrake.PCDPC
- fieldsplit_0_fieldsplit_1_pcd_Fp_mat_type aij
- fieldsplit_0_fieldsplit_1_pcd_Mp_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pcd_Mp_pc_type ilu
- fieldsplit_0_fieldsplit_1_pcd_Kp_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pcd_Kp_pc_type hypre
Now the Navier-Stokes block

\[
\mathcal{K}(N, \mathbb{N}) = \begin{bmatrix}
F \\
0 \\
\mathcal{K}(L_p, L) F_p \mathcal{K}(M_p, M) \\
\mathcal{K}(S_p, S)
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix}
\begin{bmatrix}
\mathcal{K}(F, F) & 0 \\
0 & I
\end{bmatrix}
\]

-\texttt{fieldsplit\_0\_ksp\_type gmres}
-\texttt{fieldsplit\_0\_pc\_type fieldsplit}
-\texttt{fieldsplit\_0\_pc\_fieldsplit\_type schur}
-\texttt{fieldsplit\_0\_pc\_fieldsplit\_schur\_fact\_type lower}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_type python}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_python\_type firedrake.AssembledPC}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_assembled\_pc\_type hypre}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_type python}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_python\_type firedrake.PCDPC}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_Fp\_mat\_type aij}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_Mp\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_Mp\_pc\_type ilu}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_Kp\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_Kp\_pc\_type hypre}
Now the Navier-Stokes block

\[ \mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(L_p, L) F_p \mathcal{K}(M_p, M) \mathcal{K}(S_p, S) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, F) & 0 \\ 0 & I \end{bmatrix} \]
Now the Navier-Stokes block

\[
\mathcal{K}(\mathbb{N}, \mathbb{N}) = \begin{bmatrix}
F & 0 \\
0 & \mathcal{K}(L_p, L_p) F_p \mathcal{K}(M_p, M_p)
\end{bmatrix} \begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix} \begin{bmatrix}
\mathcal{K}(F, F) & 0 \\
0 & \mathcal{K}(S_p, S_p)
\end{bmatrix}
\]
Now the Navier-Stokes block

\[ \mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} F & 0 & K(L_p, \mathbb{L}) & F_p & K(M_p, \mathbb{M}) & \mathbb{K}(S_p, \mathbb{S}) \\ 0 & I & 0 & 0 & I & 0 & I \end{bmatrix} \]

- \texttt{fieldsplit.0.ksp.type gmres}
- \texttt{fieldsplit.0.pc.type fieldsplit}
- \texttt{fieldsplit.0.pc_fieldsplit_type schur}
- \texttt{fieldsplit.0.pc_fieldsplit_schur_fact_type lower}
- \texttt{fieldsplit.0_fieldsplit.0.ksp.type preonly}
- \texttt{fieldsplit.0_fieldsplit.0_pc.type python}
- \texttt{fieldsplit.0_fieldsplit.0_pc.python_type firedrake.AssembledPC}
- \texttt{fieldsplit.0_fieldsplit.0_assembled_pc.type hypre}
- \texttt{fieldsplit.0_fieldsplit.1_ksp.type preonly}
- \texttt{fieldsplit.0_fieldsplit.1_pc_type python}
- \texttt{fieldsplit.0_fieldsplit.1_pc_python_type firedrake.PCDPC}
- \texttt{fieldsplit.0_fieldsplit.1_pcd_Fp_mat_type aij}
- \texttt{fieldsplit.0_fieldsplit.1_pcd_Mp_ksp.type preonly}
- \texttt{fieldsplit.0_fieldsplit.1_pcd_Mp_pc_type ilu}
- \texttt{fieldsplit.0_fieldsplit.1_pcd_Kp_ksp.type preonly}
- \texttt{fieldsplit.0_fieldsplit.1_pcd_Kp_pc_type hypre}
Now the Navier-Stokes block

\[
\mathcal{K}(N, \bar{N}) = \begin{bmatrix}
F & 0 \\
0 & \begin{bmatrix}
\mathcal{K}(L_p, \mathbb{L}) & F_p \mathcal{K}(M_p, \mathbb{M}) & \mathcal{K}(S_p, \mathbb{S})
\end{bmatrix}
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix}
\begin{bmatrix}
\mathcal{K}(F, \mathbb{F}) & 0 \\
0 & I
\end{bmatrix}
\]
Now the Navier-Stokes block

\[
\mathcal{K}(N, N) = \begin{bmatrix}
F & 0 \\
0 & \mathcal{K}(L_p, L) F_p \mathcal{K}(M_p, M)
\end{bmatrix} \left[
\begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix} \right] \begin{bmatrix}
\mathcal{K}(F, F) & 0 \\
0 & I
\end{bmatrix}
\]

- \texttt{fieldsplit\_0\_ksp\_type gmres}
- \texttt{fieldsplit\_0\_pc\_type fieldsplit}
- \texttt{fieldsplit\_0\_pc\_fieldsplit\_type schur}
- \texttt{fieldsplit\_0\_pc\_fieldsplit\_schur\_fact\_type lower}
- \texttt{fieldsplit\_0\_fieldsplit\_0\_ksp\_type preonly}
- \texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_type python}
- \texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_python\_type firedrake.AssembledPC}
- \texttt{fieldsplit\_0\_fieldsplit\_0\_assembled\_pc\_type hypre}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_ksp\_type preonly}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_type python}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_python\_type firedrake.PCDPC}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pcd\_Fp\_mat\_type aij}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pcd\_Mp\_ksp\_type preonly}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pcd\_Mp\_pc\_type ilu}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pcd\_Kp\_ksp\_type preonly}
- \texttt{fieldsplit\_0\_fieldsplit\_1\_pcd\_Kp\_pc\_type hypre}
Now the Navier-Stokes block

$$\mathcal{K}(N, N) = \begin{bmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{K}(L_p, L) \mathcal{F}_p \mathcal{K}(M_p, M) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, F) & 0 \\ 0 & I \end{bmatrix}$$

- fieldsplit_0_ksp_type gmres
- fieldsplit_0_pc_type fieldsplit
- fieldsplit_0_pc_fieldsplit_type schur
- fieldsplit_0_pc_fieldsplit_schur_fact_type lower
- fieldsplit_0_fieldsplit_0_ksp_type preonly
- fieldsplit_0_fieldsplit_0_pc_type python
- fieldsplit_0_fieldsplit_0_pc_python_type firedrake.AssembledPC
- fieldsplit_0_fieldsplit_0_assembled_pc_type hypre
- fieldsplit_0_fieldsplit_1_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pc_type python
- fieldsplit_0_fieldsplit_1_pc_python_type firedrake.PCDPC
- fieldsplit_0_fieldsplit_1_pcd_Fp_mat_type aij
- fieldsplit_0_fieldsplit_1_pcd_Mp_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pcd_Mp_pc_type ilu
- fieldsplit_0_fieldsplit_1_pcd_Kp_ksp_type preonly
- fieldsplit_0_fieldsplit_1_pcd_Kp_pc_type hypre
Now the Navier-Stokes block

\[
\mathcal{K}(N,N) = \begin{bmatrix}
F & 0 & \mathcal{K}(L_p, L) F_p & \mathcal{K}(M_p, M) \\
0 & \mathcal{K}(S_p, S) & \mathcal{K}(F, F)
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
-C & I
\end{bmatrix}
\begin{bmatrix}
I & 0 \\
0 & I
\end{bmatrix}
\]

-fieldsplit_0_ksp_type gmres
-fieldsplit_0_pc_type fieldsplit
-fieldsplit_0_pc_fieldsplit_type schur
-fieldsplit_0_pc_fieldsplit_schur_fact_type lower
-fieldsplit_0_fieldsplit_0_ksp_type preonly
-fieldsplit_0_fieldsplit_0_pc_type python
-fieldsplit_0_fieldsplit_0_pc_python_type firedrake.AssembledPC
-fieldsplit_0_fieldsplit_0_assembled_pc_type hypre
-fieldsplit_0_fieldsplit_1_ksp_type preonly
-fieldsplit_0_fieldsplit_1_pc_type python
-fieldsplit_0_fieldsplit_1_pc_python_type firedrake.PCDPC
-fieldsplit_0_fieldsplit_1_pcd_Fp_mat_type aij
-fieldsplit_0_fieldsplit_1_pcd_Mp_ksp_type preonly
-fieldsplit_0_fieldsplit_1_pcd_Mp_pc_type ilu
-fieldsplit_0_fieldsplit_1_pcd_Kp_ksp_type preonly
-fieldsplit_0_fieldsplit_1_pcd_Kp_pc_type hypre
Now the Navier-Stokes block

\[ \mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(L_p, \mathbb{L}) \mathcal{F}_p \mathcal{K}(M_p, \mathbb{M}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix} \]

- `fieldsplit_0_ksp_type` gmres
- `fieldsplit_0_pc_type` fieldsplit
- `fieldsplit_0_pc_fieldsplit_type` schur
- `fieldsplit_0_pc_fieldsplit_schur_fact_type` lower
- `fieldsplit_0_fieldsplit_0_ksp_type` preonly
- `fieldsplit_0_fieldsplit_0_pc_type` python
- `fieldsplit_0_fieldsplit_0_pc_python_type` firedrake.AssembledPC
- `fieldsplit_0_fieldsplit_0_assembled_pc_type` hypre
- `fieldsplit_0_fieldsplit_1_ksp_type` preonly
- `fieldsplit_0_fieldsplit_1_pc_type` python
- `fieldsplit_0_fieldsplit_1_pc_python_type` firedrake.PCDPC
- `fieldsplit_0_fieldsplit_1_pcd_Fp_mat_type` aij
- `fieldsplit_0_fieldsplit_1_pcd_Mp_ksp_type` preonly
- `fieldsplit_0_fieldsplit_1_pcd_Mp_pc_type` ilu
- `fieldsplit_0_fieldsplit_1_pcd_Kp_ksp_type` preonly
- `fieldsplit_0_fieldsplit_1_pcd_Kp_pc_type` hypre
Now the Navier-Stokes block

\[ \mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} \mathcal{F} & 0 \\ 0 & \mathcal{K}(L_p, \mathbb{L}) F_p \mathcal{K}(M_p, \mathbb{M}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix} \]

-fieldsplit_0_ksp_type gmres
-fieldsplit_0_pc_type fieldsplit
-fieldsplit_0_pc_fieldsplit_type schur
-fieldsplit_0_pc_fieldsplit_schur_fact_type lower
-fieldsplit_0_fieldsplit_0_ksp_type preonly
-fieldsplit_0_fieldsplit_0_pc_type python
-fieldsplit_0_fieldsplit_0_pc_python_type firedrake.AssembledPC
-fieldsplit_0_fieldsplit_0_assembled_pc_type hypre
-fieldsplit_0_fieldsplit_1_ksp_type preonly
-fieldsplit_0_fieldsplit_1_pc_type python
-fieldsplit_0_fieldsplit_1_pc_python_type firedrake.PCDPC
-fieldsplit_0_fieldsplit_1_pcd_Fp_mat_type aij
-fieldsplit_0_fieldsplit_1_pcd_Mp_ksp_type preonly
-fieldsplit_0_fieldsplit_1_pcd_Mp_pc_type ilu
-\textbf{-fieldsplit_0_fieldsplit_1_pcd_Kp_ksp_type preonly}
-fieldsplit_0_fieldsplit_1_pcd_Kp_pc_type hypre
Now the Navier-Stokes block

\[ \mathcal{K}(N, \mathbb{N}) = \begin{bmatrix} F & I \\ 0 & \mathcal{K}(L_p, \mathbb{L}) F_p \mathcal{K}(M_p, \mathbb{M}) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix} \]

-\texttt{fieldsplit\_0\_ksp\_type gmres}
-\texttt{fieldsplit\_0\_pc\_type fieldsplit}
-\texttt{fieldsplit\_0\_pc\_fieldsplit\_type schur}
-\texttt{fieldsplit\_0\_pc\_fieldsplit\_schur\_fact\_type lower}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_type python}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_pc\_python\_type firedrake.AssembledPC}
-\texttt{fieldsplit\_0\_fieldsplit\_0\_assembled\_pc\_type hypre}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_type python}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_python\_type firedrake.PCDPC}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_cd_Fp_mat\_type aij}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_cd_Mp_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_cd_Mp_pc\_type ilu}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_cd_Kp_ksp\_type preonly}
-\texttt{fieldsplit\_0\_fieldsplit\_1\_pc\_cd_Kp_pc\_type hypre}
Can tune implicit solve for Navier-Stokes on its own, then drop in for Navier-Stokes inverse.

Don’t need to know ahead of time how to split, maybe we wanted to eliminate temperature first. Do so, without changing the code.

Composes with nonlinear solvers that need linearisations.

Automatically take advantage of any improvements in Firedrake (fast matrix actions, etc...)

No need to worry about parallel!
Further extensibility: High-order?

- Single parameter controls degree (FIAT)
- Memory/operation costs vary greatly
- Linear systems much harder to solve.
- What to do?
Further extensibility: High-order?

- Single parameter controls degree (FIAT)
- Memory/operation costs vary greatly
- Linear systems much harder to solve.
- What to do?
Additive Schwarz/subspace correction

Preconditioner is

\[ P = \sum_{i=1}^{J} R_i Q_i, \]

where

- \( Q_i \): projection onto subspace \( i \)
- \( R_i \): (approximate) subspace solver

See (Xu 1992), (Schöberl et al. 2008)
Quadratic DOFS
$P^1$ subspace
A typical vertex patch
### $P^4$ Laplacian, 3D

| DoFs ($\times 10^6$) | MPI processes | Krylov its | Time to solution (s) |
|----------------------|---------------|------------|----------------------|
|                      |               | hypre      | schwarz              |
|                      |               |            | hypre                | schwarz            |
| 2.571                | 24            | 19         | 19                   | 5.62               | 9.48               |
| 5.545                | 48            | 20         | 19                   | 6.45               | 10.6               |
| 10.22                | 96            | 20         | 19                   | 6.17               | 10.3               |
| 20.35                | 192           | 21         | 18                   | 6.53               | 10.7               |
| 43.99                | 384           | 22         | 19                   | 7.53               | 11.9               |
| 81.18                | 768           | 22         | 19                   | 7.52               | 11.7               |
| 161.9                | 1536          | 23         | 19                   | 8.98               | 13                 |
| 350.4                | 3072          | 24         | 19                   | 8.56               | 14                 |
| 647.2                | 6144          | 26         | 19                   | 9.32               | 13.9               |
| 1291                 | 12288         | 28         | 19                   | 10.2               | 17.3               |
| 2797                 | 24576         | 29         | 19                   | 13                 | 22.5               |
## RB iteration results

| DoFs ($\times 10^6$) | MPI processes | Newton its | Krylov its | Time to solution (s) |
|----------------------|---------------|------------|------------|---------------------|
| 0.7405               | 24            | 3          | 16         | 31.7                |
| 1.488                | 48            | 3          | 16         | 36.3                |
| 2.973                | 96            | 3          | 17         | 43.9                |
| 5.769                | 192           | 3          | 17         | 47.3                |
| 11.66                | 384           | 3          | 17         | 56                  |
| 23.39                | 768           | 3          | 17         | 64.9                |
| 45.54                | 1536          | 3          | 18         | 85.2                |
| 92.28                | 3072          | 3          | 18         | 120                 |
| 185.6                | 6144          | 3          | 19         | 167                 |
Inner iterations degrade with mesh refinement – affects scaling!

| DoFs ($\times 10^6$) | Navier-Stokes iterations | Temperature iterations |
|---------------------|--------------------------|------------------------|
| 0.7405              | 329 (20.6)               | 107 (6.7)              |
| 1.488               | 338 (21.1)               | 110 (6.9)              |
| 2.973               | 365 (21.5)               | 132 (7.8)              |
| 5.769               | 358 (21.1)               | 133 (7.8)              |
| 11.66               | 373 (21.9)               | 137 (8.1)              |
| 23.39               | 378 (22.2)               | 139 (8.2)              |
| 45.54               | 403 (22.4)               | 151 (8.4)              |
| 92.28               | 420 (23.3)               | 154 (8.6)              |
| 185.6               | 463 (24.4)               | 174 (9.2)              |
Future directions

▶ Integrate with GMG solves: matrix-free multigrid.
▶ Take advantage of new hybridisation opportunities in Firedrake
▶ Extend approach to nonlinear preconditioning?

All of this is available as part of the Firedrake project
http://www.firedrakeproject.org/
https://github.com/wence-/ssc/
Brown, J. et al. (2012). “Composable Linear Solvers for Multiphysics”. Proceedings of the 2012 11th International Symposium on Parallel and Distributed Computing. ISPDC ’12. Washington, DC, USA: IEEE Computer Society. doi:10.1109/ISPDC.2012.16.

Elman, H., D. Silvester, and A. Wathen (2014). Finite elements and fast iterative solvers. Second edition. Oxford University Press.

Howle, V. E. and R. C. Kirby (2012). “Block preconditioners for finite element discretization of incompressible flow with thermal convection”. Numerical Linear Algebra with Applications 19. doi:10.1002/nla.1814.

Murphy, M. F., G. H. Golub, and A. J. Wathen (2000). “A Note on Preconditioning for Indefinite Linear Systems”. SIAM Journal on Scientific Computing 21. doi:10.1137/S1064827599355153.

Schöberl, J. et al. (2008). “Additive Schwarz preconditioning for $p$-version triangular and tetrahedral finite elements”. IMA Journal of Numerical Analysis 28.

Xu, J. (1992). “Iterative methods by space decomposition and subspace correction”. SIAM review 34.