Firedrake

Composable block preconditioning for multiphysics problems

. . . or, programming your solver

Lawrence Mitchell[1,*] Rob Kirby[2]

27th March 2017

[1]Departments of Computing and Mathematics, Imperial College London
*lawrence.mitchell@imperial.ac.uk

[2]Department of Mathematics, Baylor University

Imperial College London

Firedrake simplifies writing models

Stationary Rayleigh-Bénard convection

$$-\Delta u + u \cdot \nabla u + \nabla p + \frac{\mathrm{Ra}}{\mathrm{Pr}} \hat{g} T = 0$$

$$\nabla \cdot u = 0$$

$$-\frac{1}{\mathrm{Pr}} \Delta T + u \cdot \nabla T = 0$$

```
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)
bcs = [...] # no-flow + temp gradient
nullspace = MixedVectorSpaceBasis(
   Z, [Z.sub(0), VectorSpaceBasis(constant=True),
       Z.sub(2)])
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, nullspace=nullspace)
```

Krylov solvers are not solvers

It’s all about preconditioning

- Coupled problems are (typically) not amenable to black box solution methods.
- For small problems, can just use LU factorisation.
- For large problems, often use approximate block factorisations.
- Many configuration options, may require problem-specific auxiliary operators.
- Important to capture the abstraction so that automated model manipulation is still possible.

Block preconditioning

Most state of the art preconditioning for multi-variable problems is based on block LU factorisations.

Block preconditioning

Most state of the art preconditioning for multi-variable problems is based on block LU factorisations.

$$T = \begin{bmatrix} A & 0 \\ 0 & CA^{-1}B^T \end{bmatrix}^{-1} \begin{bmatrix} A & B^T \\ C & D = 0 \end{bmatrix}$$

has minimal polynomial $T(T-I)(T^2-T-I) = 0$ (Murphy, Golub, and A. J. Wathen 2000). Ipsen (2001) treats case of $D \neq 0$.

Alternate approach: "function space" preconditioning (Kirby 2010; Mardal and Winther 2011; Málek and Strakoš 2014).

Consequences

- The most efficient (time to solution) strategy is problem and parameter dependent:
  - Do I invert the blocks well or not?
  - How many couping terms should I drop?
- Need to be able to *configure* the solver without changing the code (e.g. eliminating first row or second?)
- Need to treat nested problems (Navier-Stokes inside Rayleigh-Bénard).
- Much larger configuration space than single-variable, fully "algebraic" preconditioning.

Back to Rayleigh-Bénard

Newton updates need inverse of Jacobian:

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

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

Preconditioning

We will invert J with a Krylov method, so we need a preconditioner. Let

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

and block eliminate N in J, giving system for temperature:

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

Howle and Kirby (2012) show that K^{-1} is a good preconditioner for S_T.

Newton update

Solve for the update

$$\delta x = J^{-1} F(x).$$

$$x \leftarrow x + \delta x$$

Write $\mathcal{K}(J, \mathbb{J})$ to denote approximating J^{-1} using an iteration $\mathcal{K}$ on J using $\mathbb{J}$ as a preconditioner. Then the iteration

$$\mathcal{K}\left(J, \begin{bmatrix} \mathcal{K}(N, \mathbb{N}) & 0 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & -\tilde{M}_1 \\ 0 & I \end{bmatrix} \begin{bmatrix} I & 0 \\ 0 & \mathcal{K}(K, \mathbb{K}) \end{bmatrix}\right)$$

empirically converges swiftly, and requires only $\mathbb{N}$ and $\mathbb{K}$.

Navier-Stokes block (Elman, Silvester, and A. Wathen 2014)

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:

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

so our recipe for $\mathcal{K}(N, \mathbb{N})$ is:

$$\mathcal{K}\left(N, \begin{bmatrix} F & 0 \\ 0 & \mathcal{K}(S_p, \mathcal{K}(L_p, \mathbb{L})\ F_p\, \mathcal{K}(M_p, \mathbb{M})) \end{bmatrix} \begin{bmatrix} I & 0 \\ -C & I \end{bmatrix} \begin{bmatrix} \mathcal{K}(F, \mathbb{F}) & 0 \\ 0 & I \end{bmatrix}\right).$$

Things to note

- We only ever need inverses of diagonal blocks.
- Can save memory by applying operators matrix-free.
- The inverses are nested, we need ways of controlling the inner iterations.

PCD

Needs the auxiliary discretised operator F_p and approximate inverses of the auxiliary operators L_p and M_p.

Communication between PDE and solver libraries can no longer be *unidirectional*.

Firedrake & PETSc to the rescue

- PETSc already provides a highly runtime-configurable library for *algebraically* composing solvers (Brown et al. 2012).
- Firedrake makes it straightforward to build auxiliary operators.
- We combine these to allow simple development of complex preconditioners.

Two new pieces

A new matrix type

A PETSc shell matrix that implements matrix-free actions using Firedrake, and contains the UFL of the bilinear form.

Custom preconditioners

These matrices do not have entries, so we create custom preconditioners that can inspect the UFL and do the appropriate thing.

Two new pieces

PCD

```python
class PCDPC(PCBase):
    def initialize(self, pc):
        _, P = pc.getOperators()
        ctx = P.getContext()
        appctx = ctx.appctx
        p, q = ctx.arguments()
        [...] # Handling of boundary conditions elided
        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
        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 ← 𝒦(L_p, 𝕃) F_p 𝒦(M_p, 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.

The elided boilerplate

```
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 are solving

$$\mathcal{K}\left(\begin{bmatrix} F & B^T & M_1 \\ C & 0 & 0 \\ M_2 & 0 & K \end{bmatrix}, \mathbb{J}\right)$$

using

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

with

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

and

$$S_p = -C\mathcal{K}(F, \mathbb{F})\, B^T.$$

First, the temperature solve

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

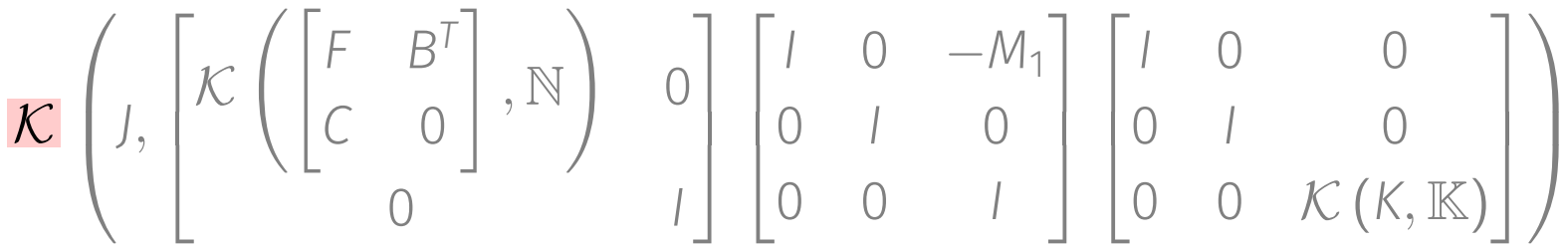
```
-mat_type matfree
-ksp_type fgmres
-pc_type fieldsplit
-pc_fieldsplit_type multiplicative
-pc_fieldsplit_0_fields 0,1
-pc_fieldsplit_1_fields 2
-fieldsplit_1_ksp_type gmres
-fieldsplit_1_pc_type python
-fieldsplit_1_pc_python_type firedrake.AssembledPC
-fieldsplit_1_assembled_pc_type hypre
```

First, the temperature solve

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

```
-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



```
-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

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

```
-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

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

```
-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

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

```
-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

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

```
-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

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

```
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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

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

```
-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 matfree
-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
```

Not shown

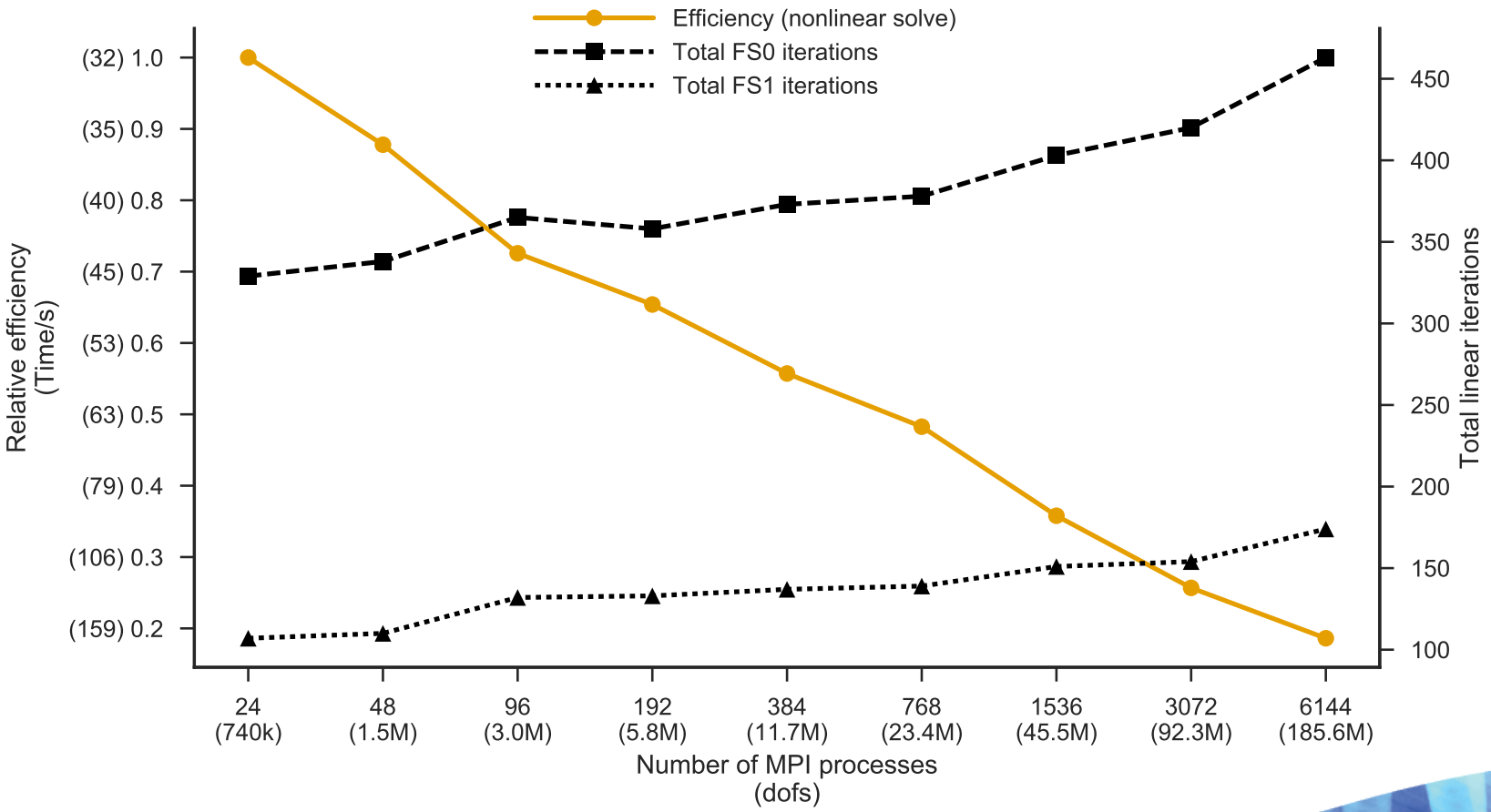
- Setting solver tolerances
- Configuring $\mathcal{K}(F, \mathbb{F})$ inside application of S_p (not needed because $\mathcal{K}(S_p, \mathbb{S})$ is applied `preonly`).
- More complex configurations for elliptic solves (e.g. *hp*-independent iterations using subspace corrections for high order).
- ...

Runtime composability

- Can tune implicit solve for Navier-Stokes on its own, then drop in where-ever such a block wants inverted.
- Model formulation doesn't care about variable splittings. 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!

Some results

Weak scaling for Rayleigh-Bénard. Ra = 200, Pr = 6.8. Three nonlinear iterations, 10 outer Krylov iterations.



Easy extensibility I

A preconditioner for the Ohta–Kawasaki equation (Farrell and Pearson 2016)

$$u_t - \Delta w + \sigma(u - m) = 0$$
$$w + \epsilon^2 \Delta u - u(u^2 - 1) = 0$$

Newton iteration at each timestep solves

$$\begin{bmatrix} (1 + \Delta t \theta \sigma) M & \Delta t \theta K \\ -\epsilon^2 K - M_E & M \end{bmatrix} \begin{bmatrix} \delta u \\ \delta w \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \end{bmatrix}$$

Easy extensibility II

Preconditioning

$$P^{-1} = \begin{bmatrix} (1+\Delta t\theta\sigma)M & 0 \\ -\epsilon^2 K - M_E & S \end{bmatrix}^{-1} = \begin{bmatrix} A^{-1} & 0 \\ 0 & S^{-1} \end{bmatrix} \begin{bmatrix} I & 0 \\ -CA^{-1} & I \end{bmatrix}.$$

Use

$$S^{-1} \approx \hat{S}^{-1} M \hat{S}^{-1}$$

where

$$\hat{S} = M + \epsilon\sqrt{(\Delta t\theta)/(1+\Delta t\theta\sigma)}K.$$

Implementation

```python
class OKPC(PCBase):

    def initialize(self, pc):
        # Approximate S^{-1} ~ Ŝ^{-1} M Ŝ^{-1} where Ŝ = ⟨q, w⟩ + ε√c ⟨∇q, ∇w⟩
        _, P = pc.getOperators()
        ctx = P.getPythonContext()
        # User information about Δt, θ, etc...
        dt, theta, eps, sigma = ctx.appctx["parameters"]
        V = ctx.a.arguments()[0].function_space()
        c = (dt * theta * eps**2)/(1 + dt * theta * sigma)
        w = TrialFunction(V)
        q = TestFunction(V)
        op = assemble(inner(w, q)*dx + sqrt(c) * inner(grad(w), grad(q))*dx)
        self.ksp = KSP().create(comm=pc.comm)
        self.ksp.setOptionsPrefix(pc.getOptionsPrefix + "shat_")
        self.ksp.setOperators(op.petscmat, op.petscmat)
        [...] # boilerplate elided
        mass = assemble(w*q*dx)
        self.mass = mass.petscmat
        work = self.mass.createVecLeft()
        self.work = (work, work.duplicate())

    def apply(self, pc, x, y):
        tmp1, tmp2 = self.work
        self.ksp.solve(x, tmp1)
        self.mass.mult(tmp1, tmp2)
        self.ksp.solve(tmp2, y)
```

Future directions

- Include ability to nest multigrid solves: matrix-free multigrid.
- Extend approach to nonlinear preconditioning, DD (needs PyOP2++)

All of this is available right now

`http://www.firedrakeproject.org/`

Questions?

References I

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.

Farrell, P. E. and J. W. Pearson (2016). *A preconditioner for the Ohta–Kawasaki equation*. arXiv: 1603.04570 [ma.NA].

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.

References II

Ipsen, I. C. F. (2001). “A Note on Preconditioning Nonsymmetric Matrices”. *SIAM Journal on Scientific Computing* **23**. doi:10.1137/S1064827500377435.

Kirby, R. C. (2010). “From Functional Analysis to Iterative Methods”. *SIAM Review* **52**. doi:10.1137/070706914.

Málek, J. and Z. Strakoš (2014). *Preconditioning and the Conjugate Gradient Method in the Context of Solving PDEs*. Philadelphia, PA: Society for Industrial and Applied Mathematics. doi:10.1137/1.9781611973846.

Mardal, K.-A. and R. Winther (2011). “Preconditioning discretizations of systems of partial differential equations”. *Numerical Linear Algebra with Applications* **18**. doi:10.1002/nla.716.

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.