Firedrake

pyop3 is coming

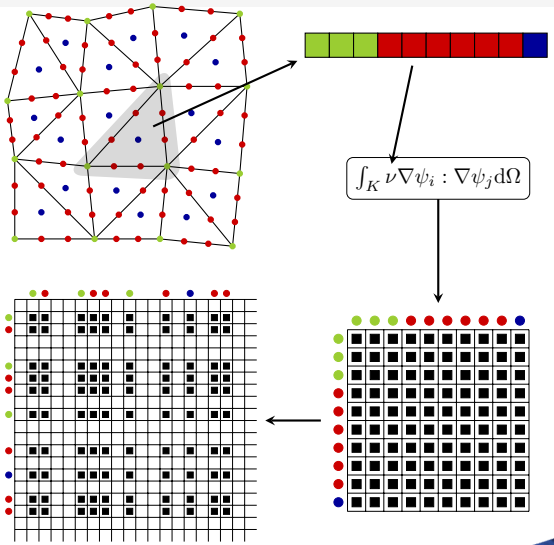
Connor Ward, David Ham, Jack Betteridge

16/09/2024

Imperial College London

- We have made a new package for **mesh stencil calculations**.
- It is called **pyop3**.
- It will **soon replace PyOP2** in Firedrake.
- This presentation will focus on the impact this will have on you, Firedrake users and developers.
- And hopefully inspire some of you to give it a try.

Mesh stencil calculation example: FEM assembly



Introducing PyOP2: motivation

- Writing one assembly loop is easy, writing many is hard.
- We want to support a wide range of different discretisations and kernels.
- We don't want to write these all by hand. Instead we want to automatically generate fast code from a high-level representation.

Introducing PyOP2

PyOP2 code:

```
op2.par_loop(
  local_kernel,
  mesh.cell_set,
  dat(op2.READ, cell_node_map),
  mat(op2.INC, (cell_node_map, cell_node_map)),
)
```

- Generates, compiles, and executes C code.
- Automatically ensures parallel correctness.
- Responsible for coordinating FEM assembly in Firedrake (among others).
- All global Firedrake data structures wrap PyOP2 ones.

Why do we need something new?

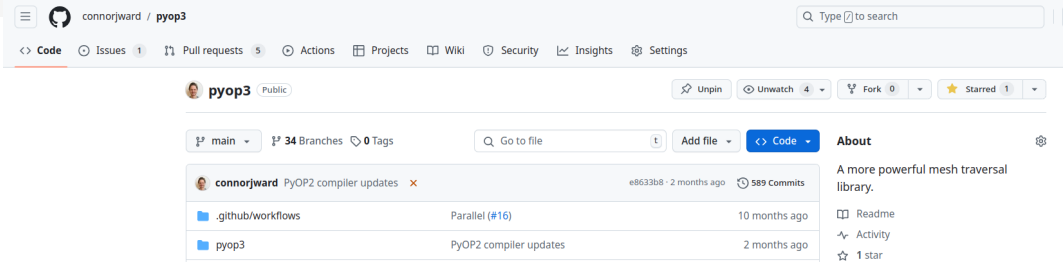
PyOP2 is a wonderful tool, but...

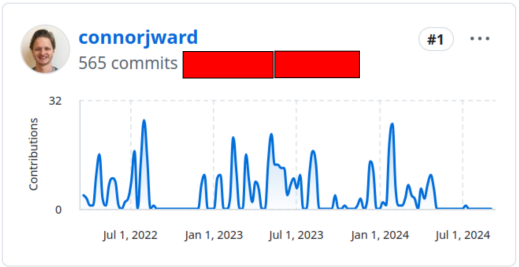
- It was designed specifically for FEM.
- Some important classes of algorithm are not expressible or require non-composable ‘hacky’ solutions (e.g. SLATE, PatchPC).

Introducing pyop3

- A near-total rewrite of PyOP2.
- Still generates C code from a Python DSL.
- Has a much more flexible interface.
- Introduces a novel, flexible way of describing data layouts (later).

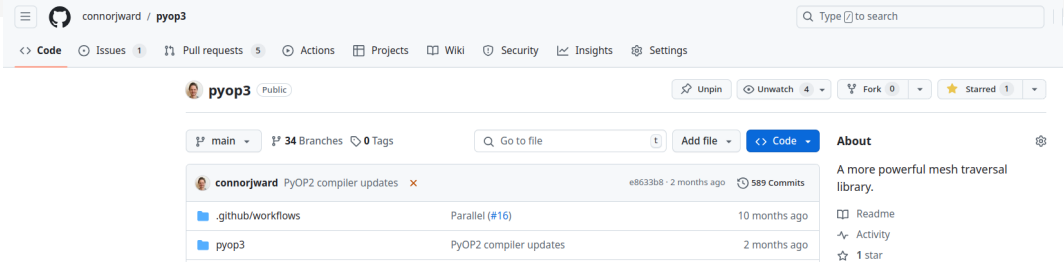
Introducing pyop3

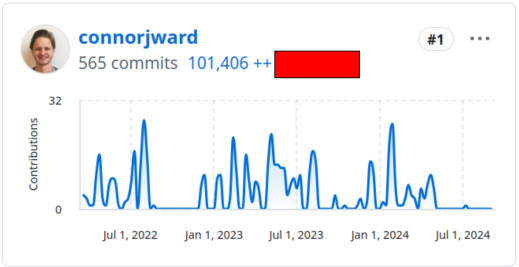




Imperial College London

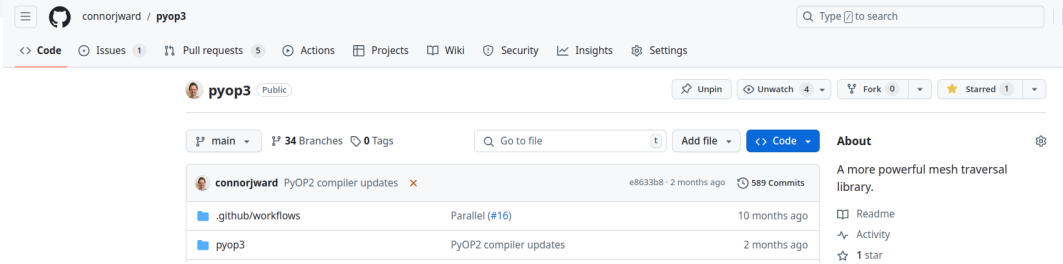
Introducing pyop3





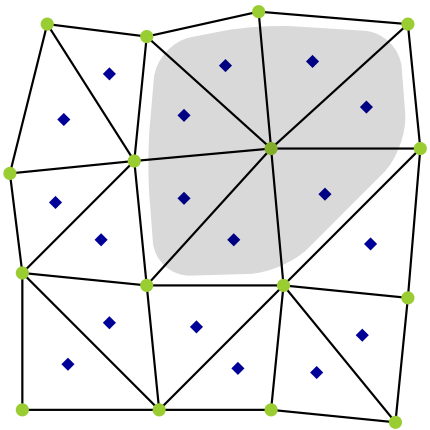
Imperial College London

Introducing pyop3



Imperial College London

An example: vertex-based slope limiter



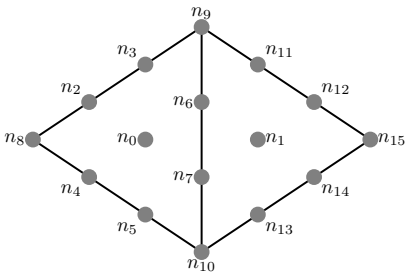
```
1  mesh = UnitSquareMesh(...)
2  V_cg = FunctionSpace(mesh, "CG", 1)
3  V_dg = FunctionSpace(mesh, "DG", 0)
4  cg = Function(V_cg)
5  dg = Function(V_dg)
6
7  loop = op3.loop(
8    v := mesh.vertices.index(),
9    op3.loop(
10     c := mesh.star(v, k=2).index(),
11     max_kernel(dg.dat[c], cg.dat[v]),
12   ),
13 )
14 loop()
```

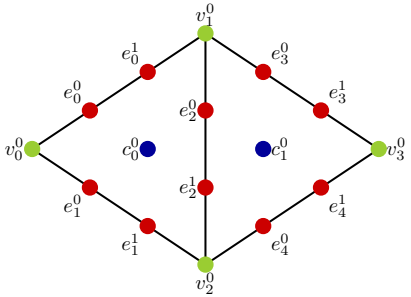
An example: vertex-based slope limiter

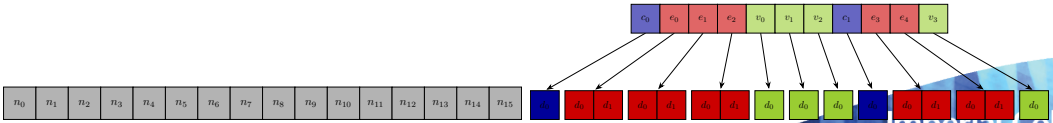
```
1  void pyop3_loop(...)
2  {
3    int32_t p_0;
4    double t_0[1];
5    double t_1[1];
6
7    for (int32_t i_0 = 0; i_0 <= end; ++i_0)
8    {
9      p_0 = array_0[i_0];
10     for (int32_t i_1 = 0; i_1 <= -1 + p_0; ++i_1)
11     {
12       t_0[0] = array_3[array_4[array_1[array_2[i_0] + i_1]]];
13       t_1[0] = array_5[array_6[i_0]];
14       max_kernel(&(t_0[0]), &(t_1[0]));
15       array_5[array_6[i_0]] = t_1[0];
16     }
17   }
```

Flexibly describing data layouts

Data layouts are **trees**, instead of N-dimensional arrays.



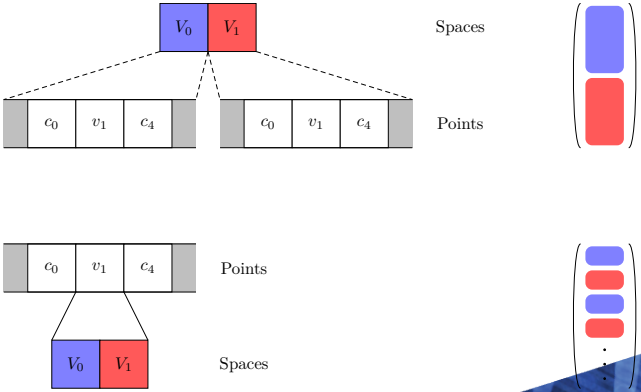




Imperial College London

Alternative data layouts

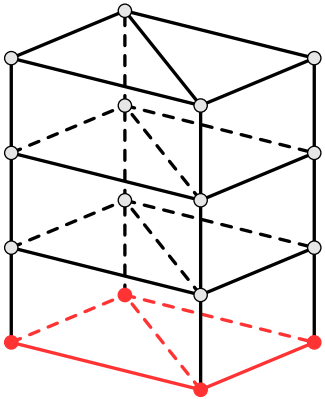
- pyop3 can freely swap around parts of the tree to give different, equivalent, data layouts.
- It is very similar to AoS/SoA optimisations.



Impact on Firedrake users/developers

- Hopefully extremely minimal.
- The top-level API is unchanged.
- Code performance should be largely the same.
- Obviously any PyOP2 code will need porting.
- If you are frequently interacting directly with the arrays (i.e. `function.dat.data`) then you *may* notice changes.

Farewell extruded mesh

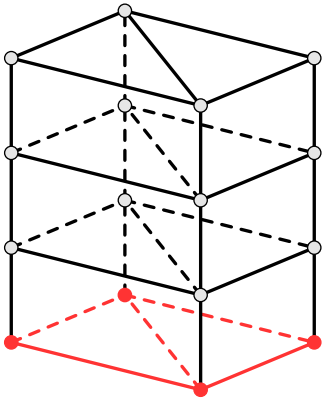


Fast for 2 reasons:

1. Regular data layout
2. Smaller indirection maps:

$$\text{loc} = \texttt{map0[}i_{\,base}\texttt{]} + i_{\text{column}} \times \text{offset}$$

Farewell extruded mesh



Fast for 2 reasons:

1. Regular data layout
2. Smaller indirection maps:

$$\text{loc} = \texttt{map0}[i_{\textit{base}}] + i_{\text{column}} \times \text{offset}$$

- It turns out that (2) is not very important.
- An extruded mesh will just be a PETSc DMPlex. Lots of code can be simplified.
- Again, Firedrake users should not notice anything different.

Additional benefits

- 'Easy' to implement new preconditioners: SLATE (hybridisation) and PatchPC (additive Schwartz) are expressible in `pyop3`.
- Can generate the transformation code necessary for arbitrary mesh entity orientations.
- Support for monolithic assembly of matrices (i.e. `"mataij"`) containing Real blocks (incidental).

More speculative:

- Structured meshes
- *hp*-adaptivity
- Low-level compiler optimisations (e.g. loop fusion)
- And more...

Summary

- `pyop3` will replace PyOP2 soon.
- It should enable a variety of new numerical methods.
- User-facing changes should be limited.

- We are currently seeking funding to extend `pyop3` to support assembly on GPUs.
- Please consider giving it a try! (not just yet)