pyop3: A (semi-)novel abstraction for automating mesh-based computations

Connor Ward

September 2022

State of play

PyOP2 is a framework for automating mesh-based computations.

- *"automating"* means *"using code generation"*
- *"mesh-based computations"* = *Next slide...*
- Integral[1] part of the Firedrake finite element system.
- Support for iteration over extruded meshes (discussed later).

[1] Pun intended.

1. Loop over a set of mesh entities (e.g. cells).
2. Gather (pack) local data[2] from global data structure(s) into temporary array(s).
3. Execute a local computation on this packed data.
4. Scatter (unpack) the result of the local computation back into some global object.

[2] e.g. “all DoFs contained within the cell’s closure”

DMPlex

- PETSc's unstructured mesh topology abstraction.
- Dimension-independent.

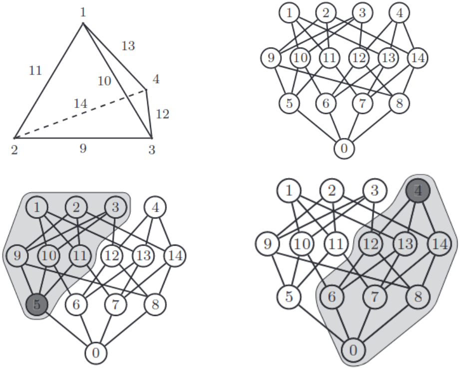

Figure 1: Source: Lange et al. (2016)

What I’m doing

pyop3 is a total rewrite of PyOP2 that...

- Has improved composability (e.g. nested loops, map composition, multiple kernels)
- Facilitates some performance optimisations (e.g. loop tiling, loop fusion, data layout transformations)[3]
- **Can generate fast code for a wide variety of composed meshes**

[3] Not yet implemented.

pyop3 interface

```
# cell assembly
loop(c := mesh.cells.index, kernel(dat1[closure(c)], dat2[closure(c)]))

# interior facet assembly
loop(f := mesh.interior_facets.index, [
  kernel(dat1[closure(support(f))], dat2[closure(support(f))])
])

# patches
loop(v := mesh.verts.index, [
  loop(p := star(v), kernel(dat1[closure(p)], dat2[closure(p)])),
  ...
])
```

What does “fast code” mean?

- Accessing data on unstructured meshes is slower than for structured since we need to use indirection maps (i.e. `dat[map[i]]` instead of `dat[f(i)]`).
- There are circumstances where one can have meshes with structured and unstructured bits (a.k.a. a *composed* mesh).
- We want to be able to generate code that uses direct addressing for the structured parts and indirection maps for the unstructured parts.

- Tensor product of an unstructured base mesh with a 1D interval mesh (structured).
- Iteration up columns is fast.
- Hackily supported in PyOP2.

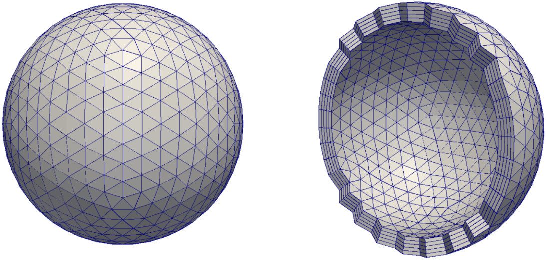

Figure 2: Extruding a sphere (source: firedrakeproject.org).

Some example meshes: cubed-sphere

- Mesh made of 6 structured panels stuck together at the boundaries.
- Looping over the panels is fast.

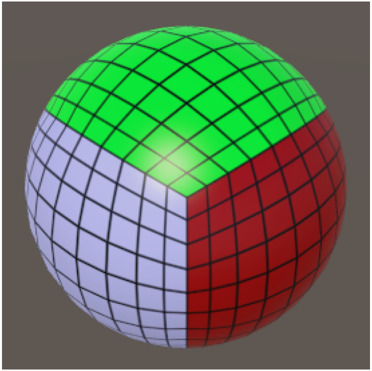

Figure 3: Source: catlikecoding.com

Some more examples

- Block-structured[4] (e.g. aerofoils)
- Hybrid
- **Multigrid?**

[4] I *think* this is the correct term.

Example generated code

The pyop3 code:

```
loop(
  c := extruded_mesh.cells.index,
  [
    mylocalkernel(dat1[cone(c)], dat2[c])
  ]
)
```

gets turned into:

```
double t0[16];
double t1[1];

for (int32_t i0 = 0; i0 < ncells; ++i0)  // loop over base cells
  for (int32_t i1 = 0; i1 < nlayers; ++i1)  // loop up column
  {
    for (int32_t i2 = 0; i2 < 3; ++i2)  // loop over cone of base mesh (triangles)
      for (int32_t i3 = 0; i3 < 2; ++i3)  // loop over cone of interval mesh
        for (int32_t i4 = 0; i4 < ndofs; ++i4)  // loop over DoFs
          t0[4 * i2 + 2 * i3 + i4] = dat1[
            (map0[3 * i0 + i2] * nlayers  // base cone
            + f(i1, i3)]) * ndofs  // interval cone (no map needed!)
            + i4  // DoFs
          ];
    t1[0] = 0.0;  // initialise output to zero
    mylocalkernel(&(t0[0]), &(t1[0]));  // local computation
    dat2[i0 * nlayers + i1] = t1[0];  // write to output
```

Imperial College London

Being rigorous

- We want a nice way to describe these sorts of composed meshes such that the code generation can exploit any structure.
- We, possibly erroneously, think that they might form a *ring*. In other words, this would mean that one could *add* and *multiply* meshes with one another.
- For example, an extruded mesh is clearly the product of some base mesh with an interval.

My questions

- Is there a unified way to describe these types of mesh composition?
- How do we propagate structural information such that I can generate code to exploit it?
- Does any of the code I write belong in PETSc instead of pyop3?