Latest developments in pyop3

Connor Ward

13 September 2023

Overview

What is **pyop3**?

A simple-ish example

What is pyop3?

TL;DR

- A programming language for mathematicians
- Comes with a compiler
- The language lets you express how to read and write from complicated data structures

In more detail

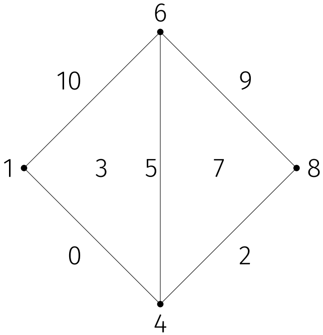
- A **domain-specific** programming language for mathematicians **embedded in Python**
- Comes with a **just-in-time** compiler **that targets loopy and then C/CUDA/OpenCL**
- The language lets you express how to read and write from complicated data structures
- **Never need to create a `PetscSection` ever again!**

Why is this hard?

- FEM codes have diverse and complicated data structures
- These data structures also need to be accessed in non-trivial ways

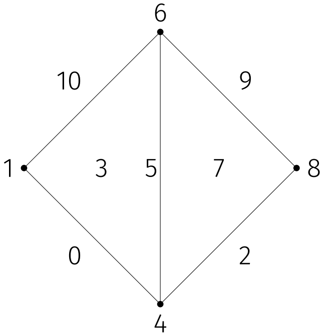
A simple-ish example

Creating a data layout for a mesh



```
mesh_axis = Axis(
  [
    AxisComponent(2, "cells"),
    AxisComponent(5, "edges"),
    AxisComponent(4, "verts"),
  ],
  "mesh"
  permutation=[3, 7, 0, 10, 5, 9, 2, 1, 6, 4, 8],
)
```

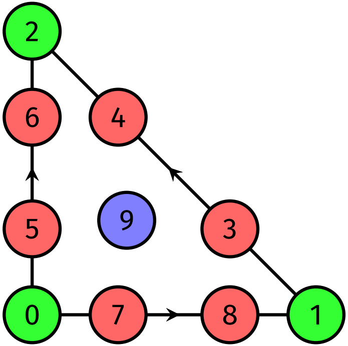
Creating a data layout for a mesh



```
mesh_axis = Axis(
  [
    AxisComponent(2, ”cells”),
    AxisComponent(5, ”edges”),
    AxisComponent(4, ”verts”),
  ],
  ”mesh”
  permutation=[3, 7, 0, 10, 5, 9, 2, 1, 6, 4, 8],
)
```

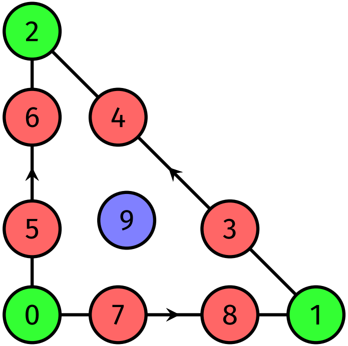
🟥 0	🟩 0	🟥 4	🟦 0	🟩 2	🟥 2	🟩 1	🟦 1	🟩 3	🟥 3	🟥 1

Now make it P3

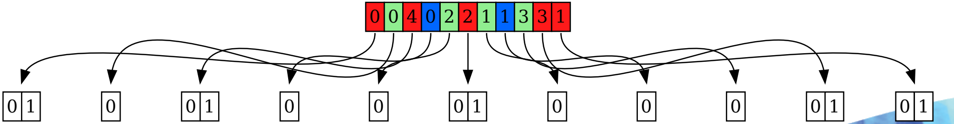


```
axes = (
  AxisTree(mesh_axis)
    .add_subaxis(Axis(1), mesh_axis.id, "cells")
    .add_subaxis(Axis(2), mesh_axis.id, "edges")
    .add_subaxis(Axis(1), mesh_axis.id, "verts")
)
```

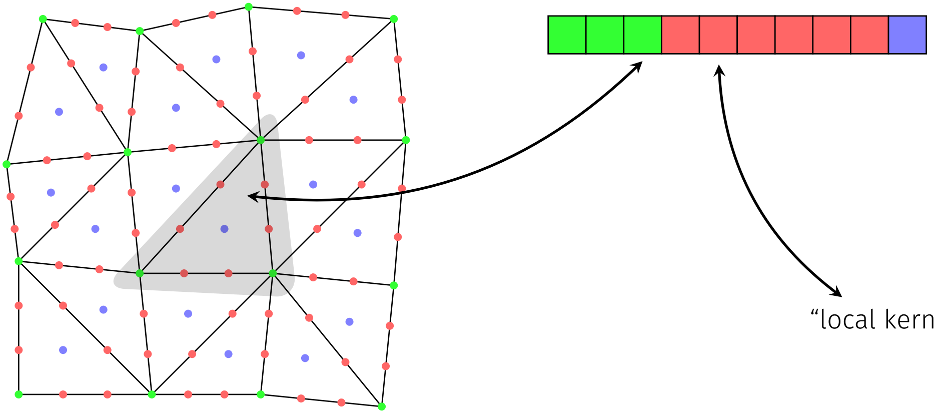
Now make it P3



```
axes = (
  AxisTree(mesh_axis)
    .add_subaxis(Axis(1), mesh_axis.id, "cells")
    .add_subaxis(Axis(2), mesh_axis.id, "edges")
    .add_subaxis(Axis(1), mesh_axis.id, "verts")
)
```



A typical non-trivial data access pattern: residual assembly



Residual assembly in pyop3

```
for every cell in the mesh:
  collect DoFs found in the cell's closure
  call a local kernel with these DoFs
  scatter the result to a global vector
```

Residual assembly in pyop3

```
for every cell in the mesh:
  collect DoFs found in the cell's closure
  call a local kernel with these DoFs
  scatter the result to a global vector
```

```
loop(
  c := mesh.cells.index(),
  kernel(func0[closure(c)], ...)
)
```

Code generation!

```c
void my_loop(double *func0, int *map0, int *map1, int *layout0, int *layout1, int *layout2, ...) {
  // to store the "packed" data
  double t_0[10];
  // loop over cells
  for (int32_t i_0 = 0; i_0 < 2; ++i_0) {
    // pack cell DoFs
    t_0[0] = func0[layout0[i_0]];
    // pack edge DoFs
    for (int32_t i_5 = 0; i_5 < 3; ++i_5) {  // loop over edges
      for (int32_t i_6 = 0; i_6 < 2; ++i_6) {  // loop over edge DoFs
        j_3 = map0[i_0 * 3 + i_5];  // select the right edge
        t_0[i_5*2 + i_6 + 1] = func0[layout1[j_3] + i_6];  // pack DoF
      }
    }
    // pack vertex DoFs
    for (int32_t i_7 = 0; i_7 < 3; ++i_7) {  // loop over vertices
      j_5 = map1[i_0 * 3 + i_7];  // select the right vertex
      t_0[i_7 + 7] = func0[layout2[j_5]];  // pack DoF
    }
    // execute the local kernel
    kernel(t_0, ...);
    // now unpack the result in the same way
  }
}
```