Chapter 1

Introduction

A common criticism of domain specific languages is that they are superfluous since any code they generate could have been written in C, C++ or Fortran by hand instead. This is especially true for `pyop3` since it generates a simple subset of C code focussed on reading/writing from array-like data structures. This is the sort of code that is bread-and-butter for undergraduate courses on high-performance computing; classical optimisations like loop tiling and AoS-SoA are very simple to write by hand.

The counter point, of course, is that `pyop3` is not primarily intended to be used as a simulation front-end, but instead as an intermediate representation of some higher level abstraction. DSLs can more or less only express problems that are also expressible in all of their IRs. The only exception to this is if the abstractions provide an escape hatch where one can inject custom code to perform a particular task. Escape hatches, however, are difficult to produce and "hacky". Also, the cost of implementing one can be prohibitive. It would be preferable for problems to be fully expressible in all IRs.

The key contribution of `pyop3` is that it increases the number of representable states of a finite-element-like code (whilst also cutting down on boilerplate). This increased expressiveness is beneficial as it enables higher-level DSLs (e.g. UFL) to express more problems without having to resort to abstraction-breaking internal code changes. As a consequence, implementing novel numerical methods that would have previously been infeasible are now tractable.

The remainder of this paper is laid out as follows…