From Description to Code Generation: Building High-Performance Tools in Python Part 1: Introduction

Andreas Klöckner
Simon Garcia de Gonzalo

Computer Science
University of Illinois at Urbana-Champaign

Outline

1 Outline

2 Why Python?

Setting

High-performance code is **challenging**:

- designed to push machines, models, and methods to the limits of their capabilities
- often put together on a (comparatively) shoestring budget
- often repurposed $\rightarrow$ high demands on flexibility

Goals

- Build Mathematically-oriented mini-languages ('DSLs')
- Apply domain-specific optimizations and transformations
- Leverage tools to generate GPU/multi-core code from DSL
- Create glue that ties components together

Goals

'Don't be limited by what's available.'

- Build Mathematically-oriented mini-languages ('DSLs')
- Apply domain-specific optimizations and transformations
- Leverage tools to generate GPU/multi-core code from DSL
- Create glue that ties components together

- Introduction
  - Why Python?
  - IPython
  - Python
  - `numpy`
- Building languages
  - Syntax trees
  - Expression languages
  - Operations on expression trees
  - A first glimpse of code generation
- OpenCL as a vehicle for code generation
  - Execution model
  - OpenCL + Python
  - High-performance primitives
- Case studies
  - `numpy`: broadcasting
  - `numpy`: `einsum`
  - UFL
- Generating C
  - Using templating engines
  - Types and hybrid code
  - Structured code generation (ASTs)
- Code generation via Loopy
  - Loop polyhedra
  - Instructions and ordering
  - Loop transformation, and data layout
  - Generating instructions from DSLs

Outline

1 Outline

2 Why Python?

Why Python?

Python: One example of a modern scripting language

- Mature
- Large and active community
- Emphasizes readability
- Written in widely-portable C
- A 'multi-paradigm' language
- Rich ecosystem of sci-comp related software

Why Python for HPC?

Python is unique as an HPC language:

- approachable
- safe
- gentle learning curve
- principled
- performant enough for large, complicated systems

Getting the software

Core packages:

- Python: `https://www.python.org`
- numpy: `https://www.numpy.org`
- pymbolic: `https://github.com/inducer/pymbolic`
- PyOpenCL: `https://github.com/pyopencl/pyopencl`
- loopy: `https://github.com/inducer/loopy`

Supporting packages:

- matplotlib: `http://www.matplotlib.org`
- mako: `http://www.makotemplates.org`
- cgen: `https://github.com/inducer/cgen`

All open-source under MIT/BSD licenses.

Installing the software

- Demo: `virtualenv`
- Demo: `pip`

DEMO TIME