Object-oriented programming with NumPy using CPython & PyPy; comparison with C++ & modern Fortran

**Introduction**

The poster describes and extends key findings presented in paper:


All codes can be found in the repository: [https://github.com/slavoo/mpdata](https://github.com/slavoo/mpdata)

**Object-oriented programming (OOP)**

Application of OOP for scientific software may help to:

- maintain modularity and separation of program logic layers (e.g. separation of numerical algorithms, parallelisation mechanisms, error handling and the description of physical processes)
- shorten and simplify the source code by reproducing the mathematical notation used in the literature.

**Advevtive transport**

The advection equation for a conserved quantity described by a scalar field

$$\frac{\partial \psi}{\partial t} = -\nabla \cdot (\overrightarrow{\mathbf{v}} \psi)$$

**The Arakawa-C staggered grid**

A natural choice for some advective schemes.

The discretised representations of the scalar field, and each component of the velocity vector field are defined over different grid point locations. In mathematical notation this can be indicated by usage of fractional indices.

**Performance evaluation**

A series of simulations with different grid sizes was executed. The test set-ups and procedures are described in detail in [arXiv:1301.1334](http://arxiv.org/abs/1301.1334).

**Improving Python performance**

What **is** PyPy and why is it faster:

- Alternative implementation of Python equipped with a just-in-time compiler;
- developed with the aim of improving Python's performance while maintaining compatibility with CPython;
- PyPy's built-in NumPy implementation features lazy-evolution mechanism (v. 1.9),
- switching to PyPy does not require code modifications!

Other tools to improve Python performance:

In addition to PyPy, Cython and Numba were tested. Both Cython and Numba require large number of lines of code to be added or changed. A (much longer) version of the code with the most time-consuming statements rewritten by explicitly coding loops over spatial indices was tested. Initial tests suggest that still only PyPy (version 2.0) is the fastest Python solution.

**Python added values**

- Python has the most brief and clear syntax, encouraging OOP, easy to learn.
- Python is the easiest extension to use at vectorial level, and now combine architectures and systems.
- Python (and C++) is general-purpose language with large users‘ community.

---

**Advection equation:**

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n + \frac{1}{\Delta t} \left[ \nabla \cdot (\overrightarrow{\mathbf{v}} \psi) \right]_{i,j}$$

Equations for model variables (e.g., temperature, liquid water content) are solved for each grid-point of model domain.

Performance evaluation of Object-oriented programming with NumPy using CPython & PyPy; comparison with C++ & modern Fortran provide comparable functionalities for compact representation of mathematical abstractions within scientific code.

---

**Representations of blackboard abstractions within the code**

- Loop-free array arithmetic
- Fractional indexing
- Permutation of array indexes
- Array-valued expressions and functions

Example of reproducing notation in Python - 2D donor-cell advection

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n + \frac{1}{\Delta t} \left[ \nabla \cdot (\overrightarrow{\mathbf{v}} \psi) \right]_{i,j}$$

$$\psi_{i,j}^{n+1} = \psi_{i,j}^n + \frac{1}{\Delta t} \left[ \nabla \cdot (\overrightarrow{\mathbf{v}} \psi) \right]_{i,j}$$

**Advection equation**

$$\frac{\partial \psi}{\partial t} = -\nabla \cdot (\overrightarrow{\mathbf{v}} \psi)$$

**Memory consumption**

CPU time

Memory consumption

- The PyPy (and C++) set-ups offered significantly smaller memory consumption than Fortran and CPython for larger domains.
- The PyPy set-up was up to twice faster than CPython and roughly twice slower than C++.