PySDM:
Bridging performance and pythonicity with Numba, Pythran and ThrustRTC

Piotr Bartman

6th ENES HPC Workshop, 26 May 2020

ABOUT THE PROJECT

Atmospheric Cloud Simulation Group
Faculty of Mathematics and Computer Science,
Jagiellonian University in Kraków, Poland

Super-droplet method (SDM) in Python: PySDM

Our projects:
http://github.com/atmos-cloud-sim-uj/PySDM

Jagiellonian University

PySDM

CFD/Monte-Carlo simulations in aerosol-cloud-rain physics

Potential users:

- cloud physicists
- developers of parameterisations for weather & climate models

Motivation:

- novel modelling methods (probabilistic particle-based simulation)
- leveraging modern hardware and cloud computing
- maintainability and reproducibility requirements (of journals and scientific method)

PySDM

reproducibility goals

“(...) code must be made accessible during the review process”
„(...) no manual processing of the data: models are run by a script, and all pre- and post-processing is scripted”
„(...) figures and tables must be scientifically reproducible from the scripts”
„(...) if the code is not ready, then neither is the manuscript”

new 2019 GMD journal policy, doi:10.5194/gmd-12-2215-2019

Geoscientific Model Development

An interactive open-access journal of the European Geosciences Union

Pythonicity

Code is read much more often than it is written

Code readable - performance trade off

„'two-language problem' — researchers often prototype algorithmsin a user-friendly language such as Python but then have to rewritethem in a faster language (...)"*

```
def proj(vx, vy, vz, kx, ky, kz, inv_k_square_nozero):
   tmp = (kx * vx + ky * vy + kz * vz) * inv_k_square_nozero
   return vx - kx * tmp, vy - ky * tmp, vz - kz * tmp
```

=

```
subroutine proj(res, vx, vy, vz, kx, ky, kz, inv_k_square_nozero, N0, N1, N2)

  implicit none

  ! Input/Output
  integer, intent(in) :: N0, N1, N2
  double precision, intent(in) :: vx(2, N2, N1, N0), vy(2, N2, N1, N0), vz(2, N2, N1, N0)
  double precision, intent(in) :: kx(N2, N1, N0), ky(N2, N1, N0), kz(N2, N1, N0)
  double precision, intent(in) :: inv_k_square_nozero(N2, N1, N0)
  double precision, intent(out) :: res(2, 3, N2, N1, N0)

  ! Locals
  double precision :: tmp(2)
  integer:: i, j, k

  do k = 1, N0
    do j = 1, N1
      do i = 1, N2
        tmp(1:2) = (kx(i,j,k) * vx(1:2,i,j,k) &
             + ky(i,j,k) * vy(1:2,i,j,k) &
             + kz(i,j,k) * vz(1:2,i,j,k)) * inv_k_square_nozero(i,j,k)

        res(1:2,1,i,j,k) = vx(1:2,i,j,k) - kx(i,j,k) * tmp(1:2)
        res(1:2,2,i,j,k) = vy(1:2,i,j,k) - ky(i,j,k) * tmp(1:2)
        res(1:2,3,i,j,k) = vz(1:2,i,j,k) - kz(i,j,k) * tmp(1:2)
      enddo
    enddo
  enddo

end subroutine proj
```

*Nature 2019 ``toolbox'' column (on Julia), doi: 10.1038/d41586-019-02310-3

PySDM

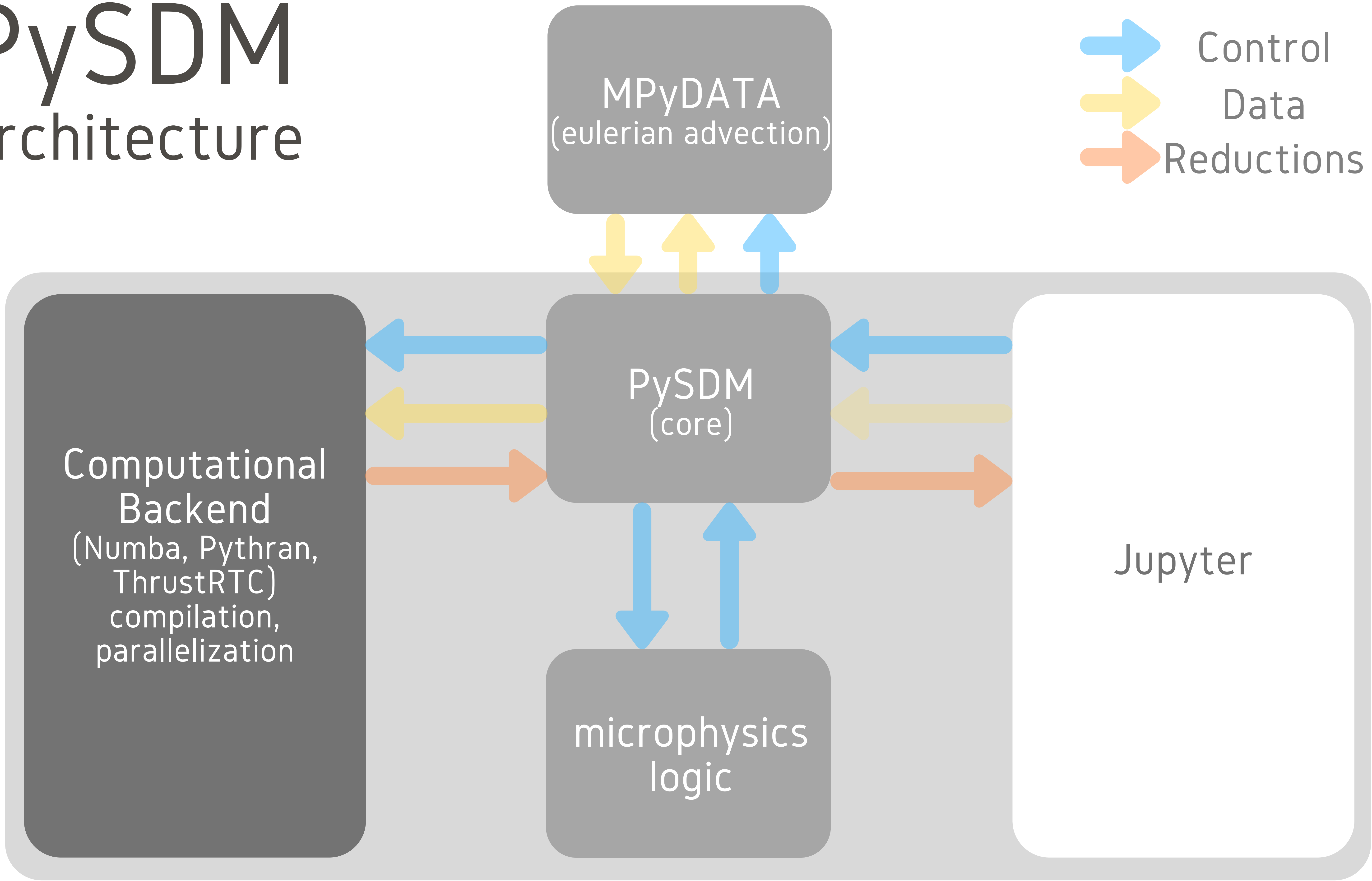
technological stack and workflows

General:
- open source
- automated tests
- code coverage

Python acceleration:
- LLVM
- multi-threading
- GPU computation

User interface:
- interactive exapmles & tutorials

PySDM architecture



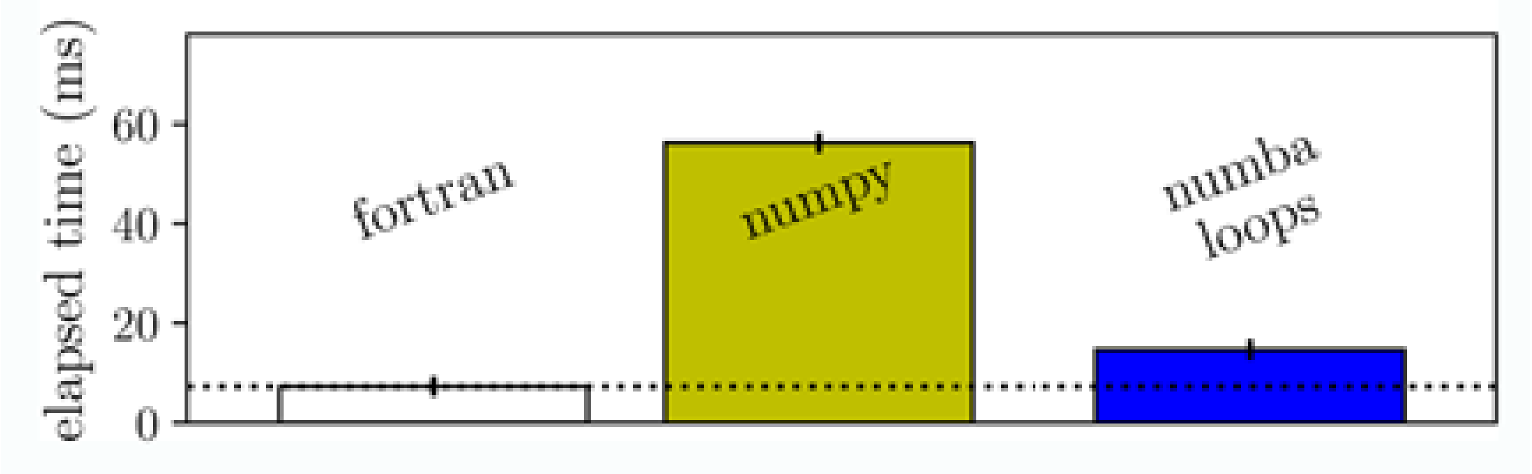
Numba

numba.pydata.org | github.com/numba/numba/

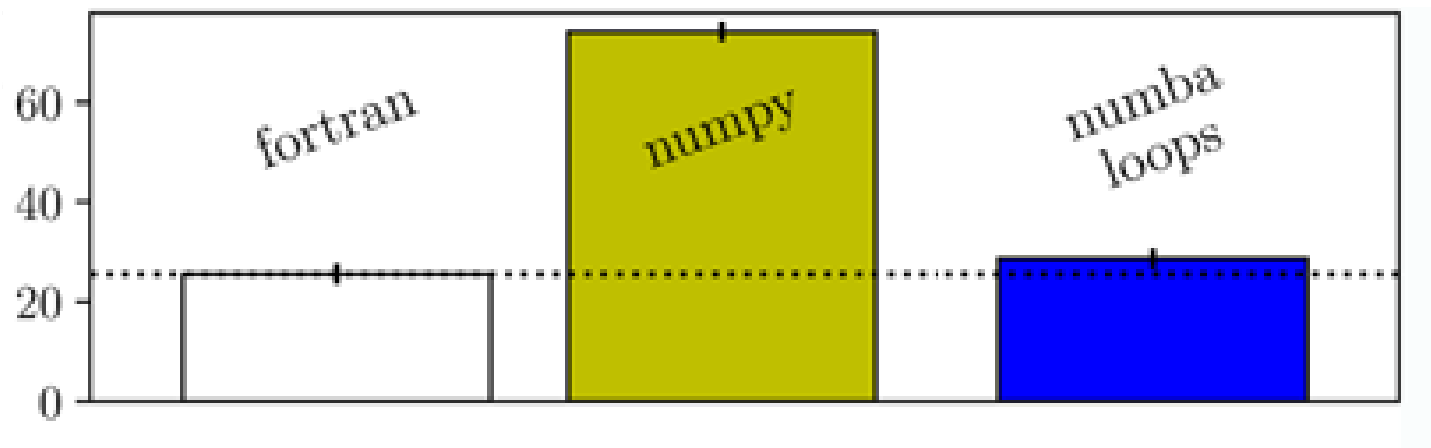
- compiler for Python functions
- generates optimized machine code from pure Python code using the LLVM compiler infrastructure
- on-the-fly code generation
- native code generation for the CPU and GPU hardware
- integration with the Python scientific software stack (thanks to Numpy)

Mohanan et al. 2019, doi:10.5334/jors.238

in-place



out-of-place (with memory allocation)



PySDM - Numba

maintainability & performance

```python
@numba.njit(**conf.JIT_FLAGS)
def lv(T):
    return const.l_tri + \
           (const.c_pv - const.c_pw) * \
           (T - const.T_tri)
```

```python
def test_lv():
    with DimensionalAnalysis():
        # Arrange
        si = constants.si
        T = 300 * si.kelvins

        # Act
        latent_heat = formulae.lv(T)

        # Assert
        assert latent_heat.check('[energy]/[mass]')
```

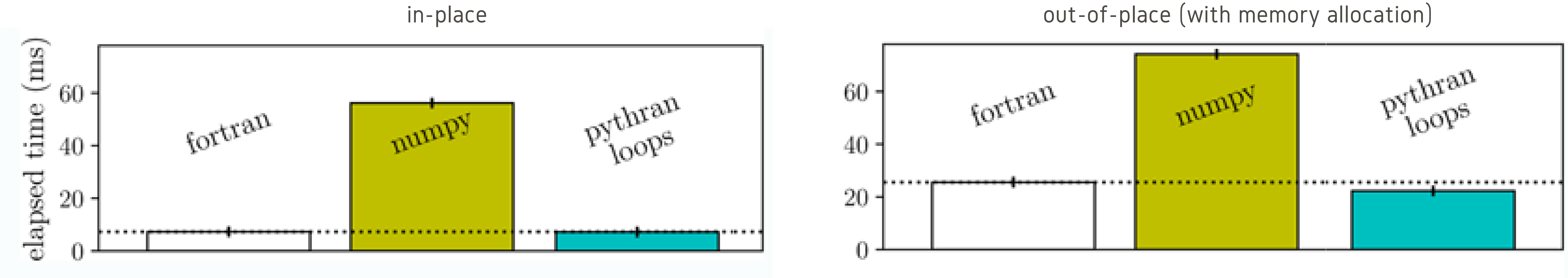
Pythran

pythran.readthedocs.io | github.com/serge-sans-paille/pythran

Pythran

- ahead of time compiler for a subset of the Python language, with a focus on scientific computing
- takes a Python module annotated with a few interface description and turns it into a native Python module with the same interface
- meant to efficiently compile scientific programs
- focus on of multi-cores and SIMD instruction units

Mohanan et al. 2019, doi:10.5334/jors.238



ThrustRTC + CURandRTC

github.com/fynv/ThrustRTC | github.com/fynv/CurandRTC

- library of general GPU algorithms, functionally similar to Thrust, that can be used in non-C++ programming launguages (Python)
- Thrust (docs.nvidia.com/cuda/thrust):
  - high-level interface enabling performance portability between GPUs and multicore CPUs
  - Interoperability with established technologies (such as CUDA, TBB, and OpenMP)
- CURandRTC: random number generator using the XORWOW algorithm

PySDM - ThrustRTC

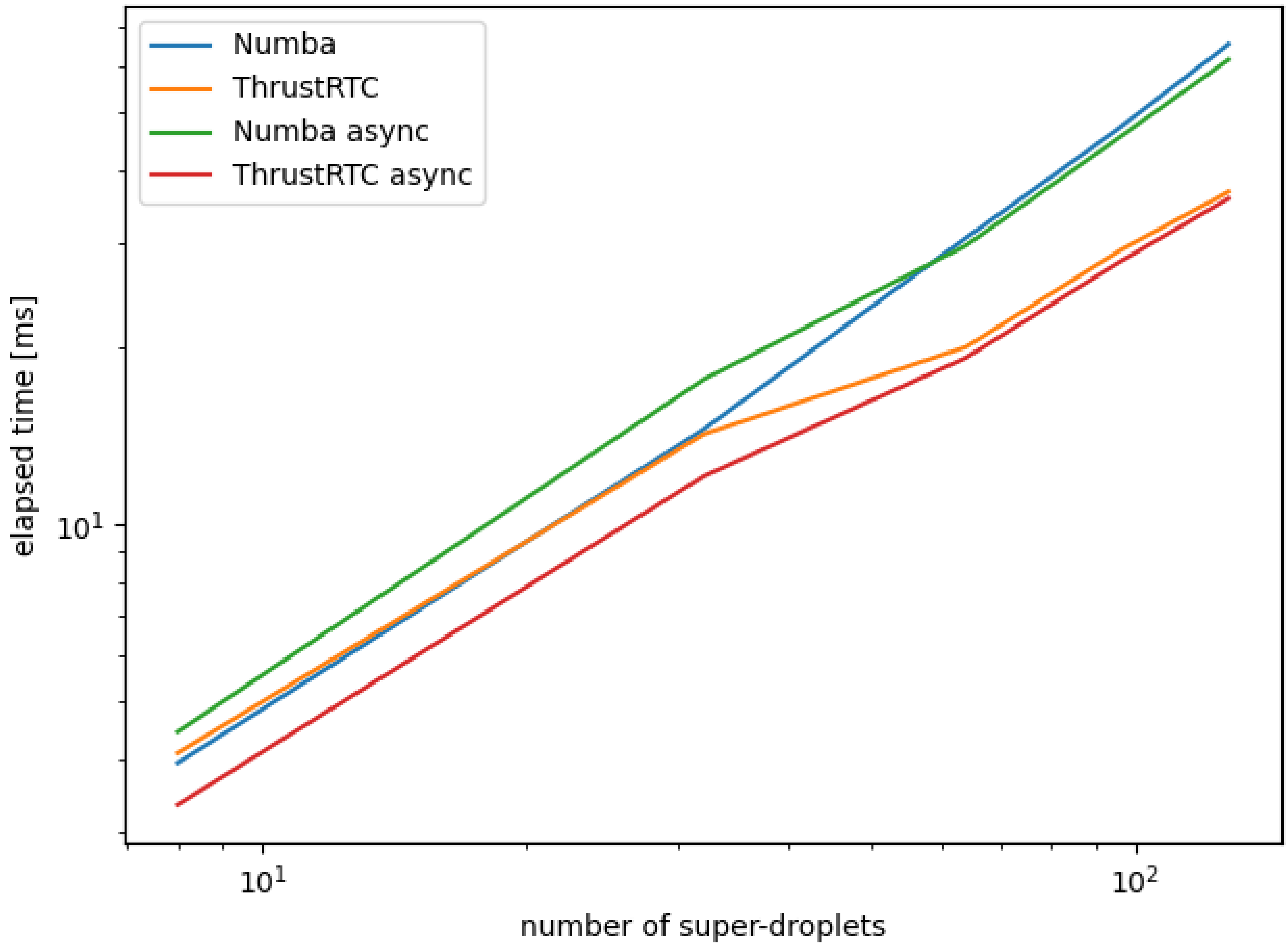
maintainability & performance

```python
def add(output, addend):
    trtc.Transform_Binary(addend, output, output, trtc.Plus())
```

```python
def flag_precipitated(cell, position, idx, length, healthy):
    idx_length = trtc.DVInt64(idx.size())
    n_dims = trtc.DVInt64(len(cell.shape))
    kernel = trtc.For(
        ['idx', 'idx_lth', 'n_dims', 'healthy', 'cell', 'position'], "i", '''
        id = idx_lth * (n_dims-1) + i;
        if (cell[id] == 0 && position[id] < 0) {
            idx[i] = idx_lth;
            healthy[0] = 0;
        }
        ''')
    kernel.launch_n(length, [idx, idx_length, n_dims, healthy, cell, position])
```

ThrustRTC vs Numba

github.com/atmos-cloud-sim-uj/PySDM/tree/master/PySDM_examples/ICMW_2012_case_1



async - computations on the CPU are overlapped with computations on the accelerator

Numba - CPU multi-threading
ThrustRTC - GPU resident

PySDM

portability

Build jobs | View config

✓	# 182.1	AMD64	Linux	Python 3.8 with newest packages on Linux	19 min 45 sec
✓	# 182.2	AMD64	Linux	Python 3.7 on Linux numba::parallel=False	21 min 10 sec
✓	# 182.3	AMD64	Linux	Python 3.7 on Linux numba::parallel=True	18 min 39 sec
✓	# 182.4	AMD64	OSX	Python 3.7 on OSX	30 min 11 sec
✓	# 182.5	AMD64	Windows	Python 3.7 on Windows	32 min 20 sec

README.md

build passing coverage 66%

PySDM

PySDM simulates the dynamics of population of particles immersed in moist air using the particle-based (a.k.a. super-droplet) approach to represent aerosol/cloud/rain microphysics. The package features a Pythonic implementation of the Super-Droplet Method (SDM) Monte-Carlo algorithm for representing collisinal growth (Shima et al. 2009), hence the name.

Demos:

- Shima et al. 2009 Fig. 2 launch binder
- Arabas & Shima 2017 Fig. 5 launch binder
- Yang et al. 2018 Fig. 2: launch binder
- ICMW 2012 case 1 (work in progress) launch binder

Tutorials:

- Introduction launch binder
- Coalescence launch binder

Credits:

Development of PySDM is supported by the EU through a grant of the Foundation for Polish Science (POIR.04.04.00-00-5E1C/18).

PySDM
lessons learned

Numba Pythran

Numba:

- portable and pure-python (yet not fully pythonic; OOP kills performance)
- easy debbuging
- little-to-no control over (and huge performance sensitivity to) inlining/optimization

Pythran:

- essentially equivalent performance and parallelization features as Numba (both based on LLVM and OpenMP)
- cross-platform support is low-priority

PySDM
lessons learned

ThrustRTC (+CURandRTC):

- viable high-level Python abstractions for parallel GPU computations
- tricky debugging for custom kernels (CUDA-free C code in Python strings)
- no option to execute on CPU/threads (unlike original Thrust)

Special thanks for Numba, Pythran, ThrustRTC developers for quickly responses (2, 5, 1 fixed issues respectively)

MORE:

github.com/atmos-cloud-sim-uj/PySDM
github.com/piotrbartman
mail: piotr.bartman@student.uj.edu.pl