On the design of Monte-Carlo particle coagulation solver interface

a CPU/GPU Super-Droplet Method case study with PySDM

Piotr Bartman and Sylwester Arabas

Jagiellonian University

Context: aerosol-cloud-precipitation interactions

"Cloud and ship. Ukraine, Crimea, Black sea, view from Ai-Petri mountain"
(photo: Yevgen Timashov / National Geographic)

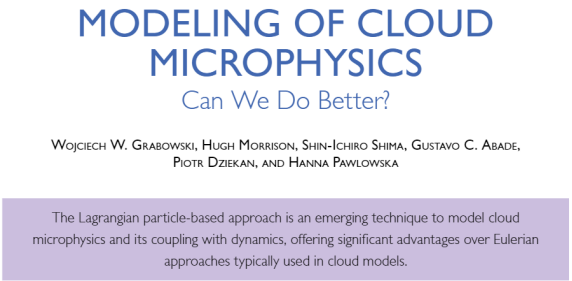
Context: particle-based microphysics

Cloud microphysics challenges:

- enormous range of scales: from micrometer-size particles to kilometer-size cloudfield structures,
- multiplicity and complexity of processes which are hard to represent in non-particle (continuous media) approaches: coalescence, condensation/evaporation, freezing, sublimation, turbulence, chemical reactions etc.

Particle-based approach:

- applicable in other domains, e.g., astrophysics, oceanology, aerosol/hydrosol technology including combustion.

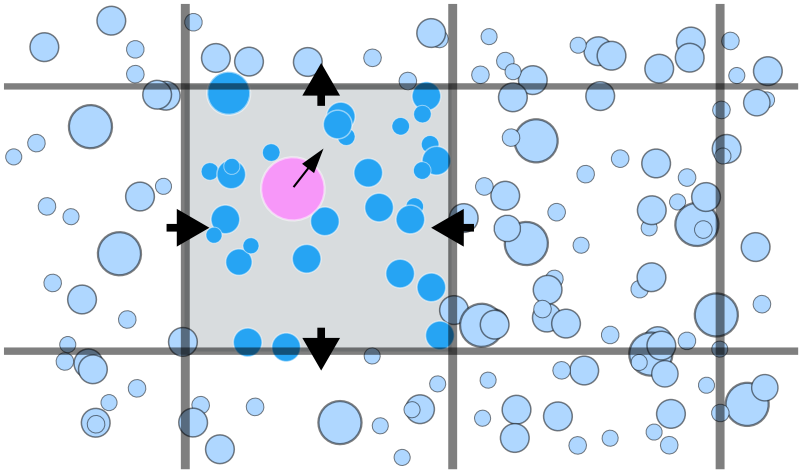

MODELING OF CLOUD MICROPHYSICS

Can We Do Better?

Wojciech W. Grabowski, Hugh Morrison, Shin-Ichiro Shima, Gustavo C. Abade, Piotr Dziekan, and Hanna Pawlowska

The Lagrangian particle-based approach is an emerging technique to model cloud microphysics and its coupling with dynamics, offering significant advantages over Eulerian approaches typically used in cloud models.

Context: particle-based microphysics (CFD coupled)



PySDM (https://github.com/atmos-cloud-sim-uj/PySDM)

PySDM is a new open-source cloud microphysics simulation package for modeling the dynamics of particle population in moist air using the super-droplet approach. The package core is a Pythonic implementation of the Super-Droplet Method (SDM), a Monte-Carlo algorithm for representing cloud droplet collisional growth.

Motivation:

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

arXiv.org > physics > arXiv:2103.17238

Physics > Atmospheric and Oceanic Physics

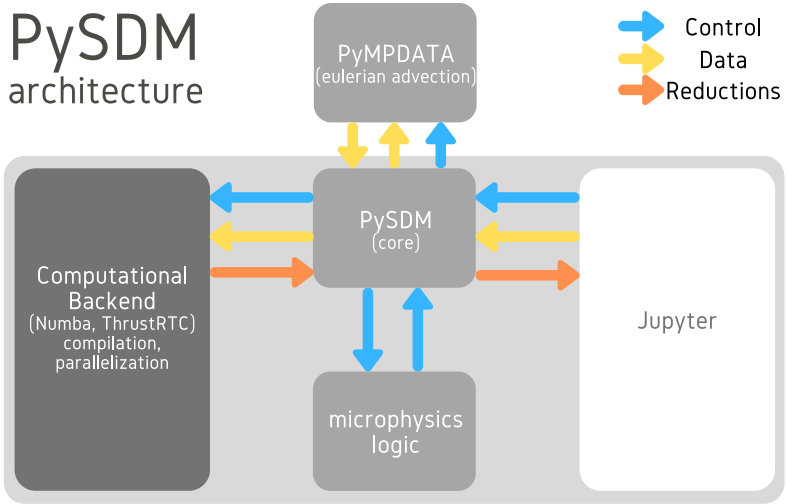
[Submitted on 31 Mar 2021]

PySDM v1: particle-based cloud modelling package for warm-rain microphysics and aqueous chemistry

Piotr Bartman, Sylwester Arabas, Kamil Górski, Anna Jaruga, Grzegorz Łazarski, Michael Olesik, Bartosz Piasecki, Aleksandra Talar

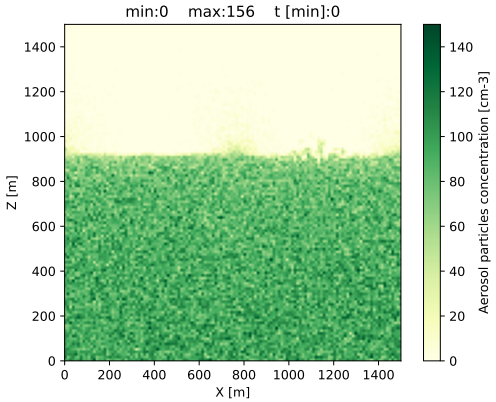
PySDM is an open-source Python package for simulating the dynamics of particles undergoing condensational and collisional growth, interacting with a fluid flow and subject to chemical composition changes. It is intended to serve as a building block for process-level as well as computational-fluid-dynamics simulation systems involving representation of a continuous phase (air) and a dispersed phase (aerosol), with PySDM being responsible for representation of the dispersed phase. The PySDM package core is a Pythonic high-performance implementation of the Super-Droplet Method (SDM) Monte-Carlo algorithm for representing collisional growth, hence the name. PySDM has two alternative parallel number-crunching backends available: multi-threaded CPU backend based on Numba and GPU-resident backend built on top of ThrustRTC. The usage examples are built on top of four simple atmospheric cloud modelling frameworks: box, adiabatic parcel, single-column and 2D prescribed flow kinematic models. In addition, the package ships with tutorial code depicting how PySDM can be used from Julia and Matlab.

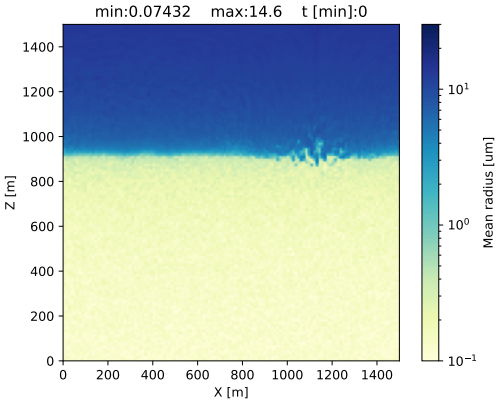
PySDM: workflow



PySDM: example prescribed-flow simulation

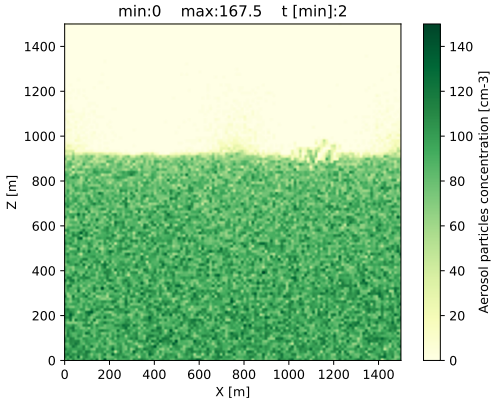
Computational grid: 128x128
Computational particles: 2^{21}

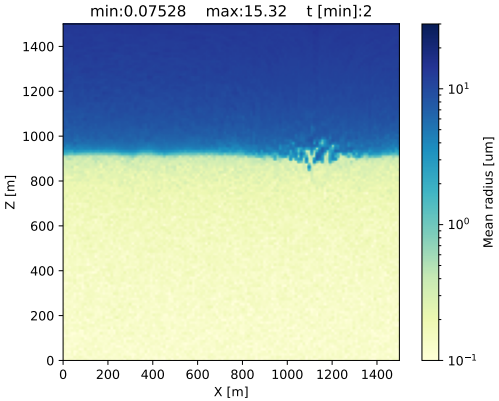




PySDM: example prescribed-flow simulation

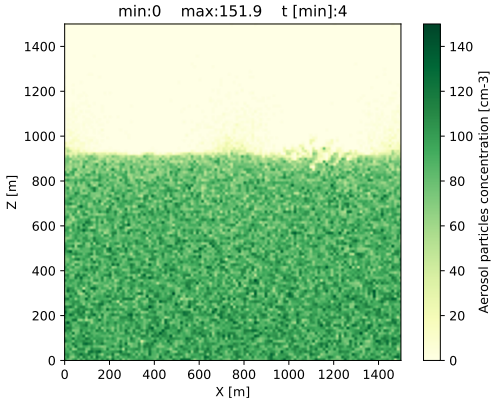
Computational grid: 128x128
Computational particles: 2^{21}

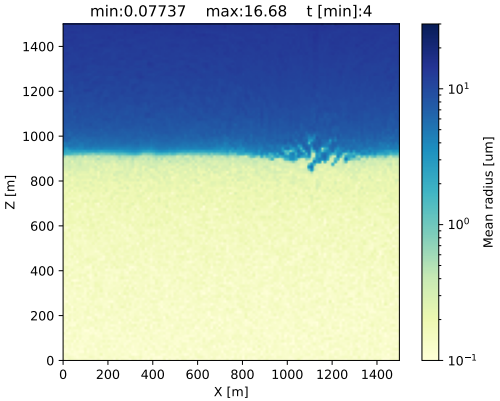




PySDM: example prescribed-flow simulation

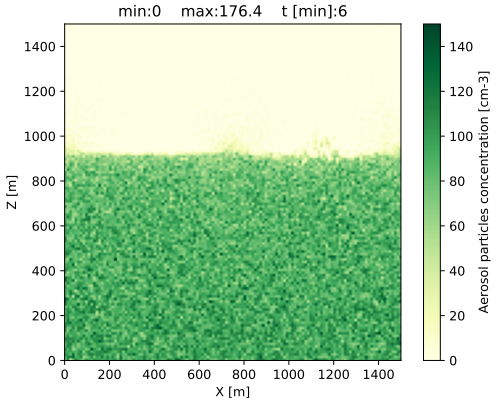
Computational grid: 128x128
Computational particles: 2^{21}

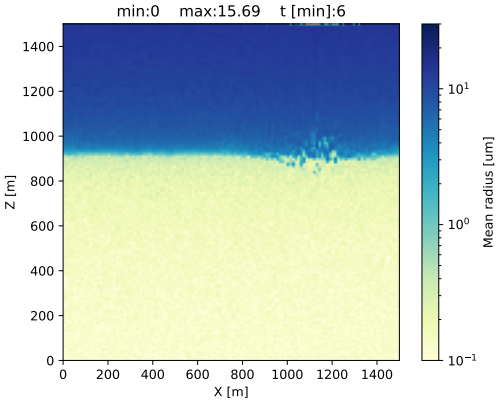




PySDM: example prescribed-flow simulation

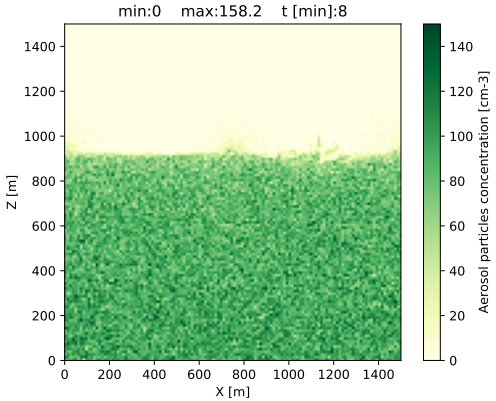
Computational grid: 128x128
Computational particles: 2^{21}

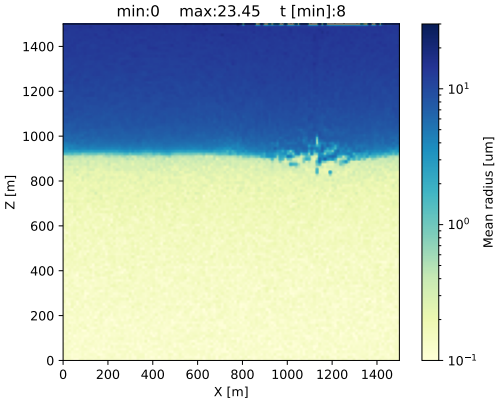




PySDM: example prescribed-flow simulation

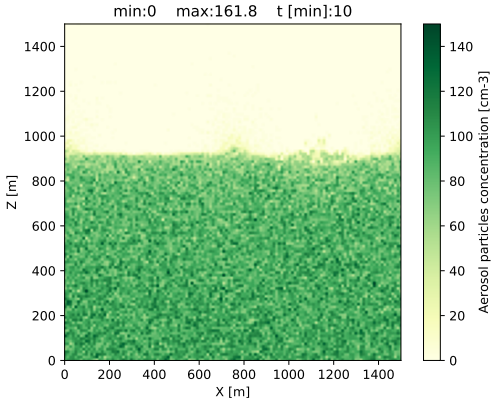
Computational grid: 128x128
Computational particles: 2^{21}

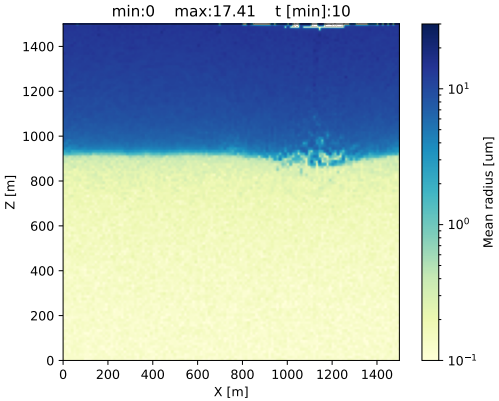




PySDM: example prescribed-flow simulation

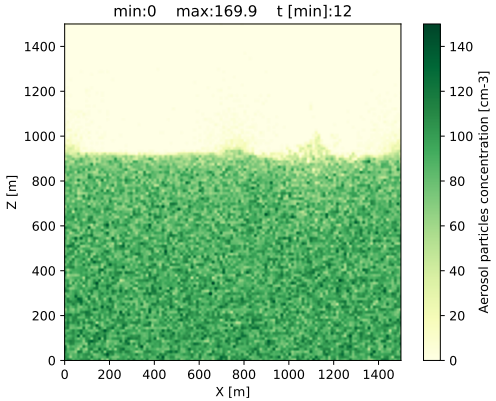
Computational grid: 128x128
Computational particles: 2^{21}

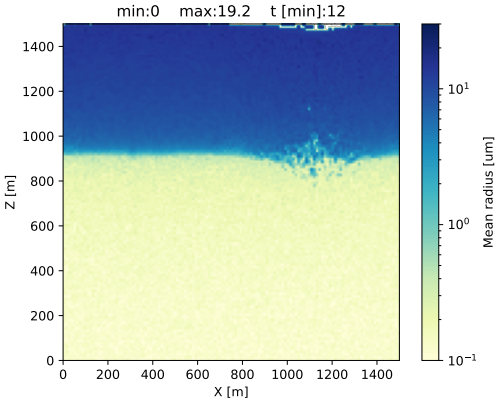




PySDM: example prescribed-flow simulation

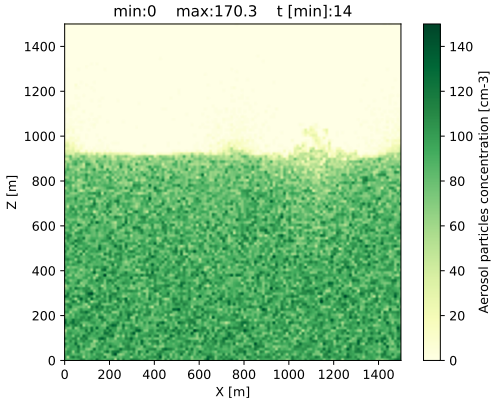
Computational grid: 128x128
Computational particles: 2^{21}

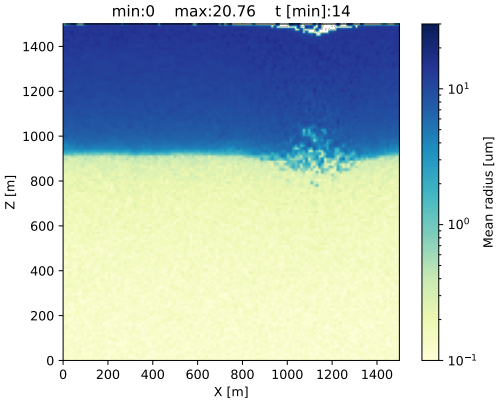




PySDM: example prescribed-flow simulation

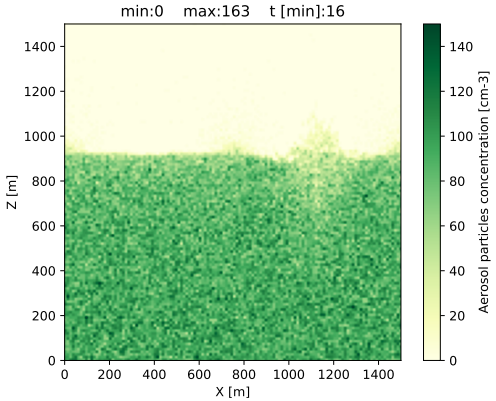
Computational grid: 128x128
Computational particles: 2^{21}

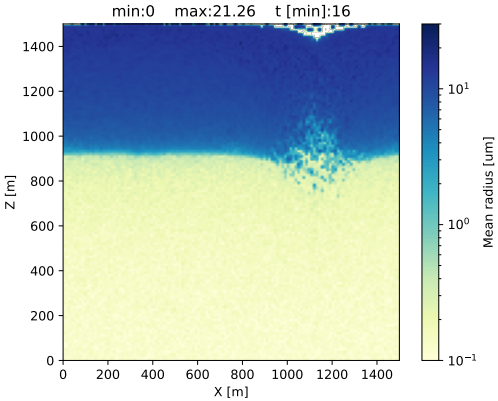




PySDM: example prescribed-flow simulation

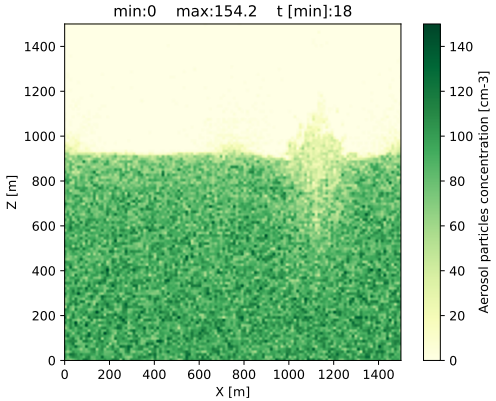
Computational grid: 128x128
Computational particles: 2^{21}

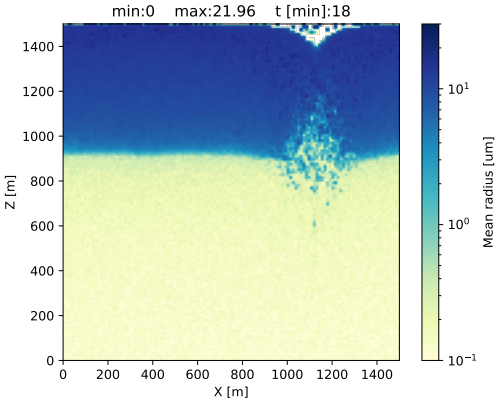




PySDM: example prescribed-flow simulation

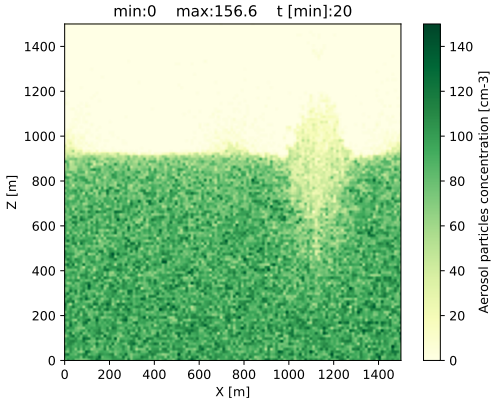
Computational grid: 128x128
Computational particles: 2^{21}

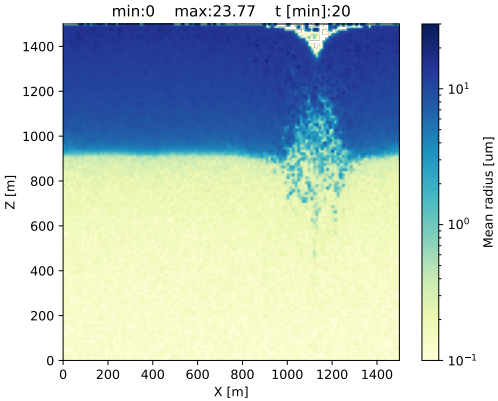




PySDM: example prescribed-flow simulation

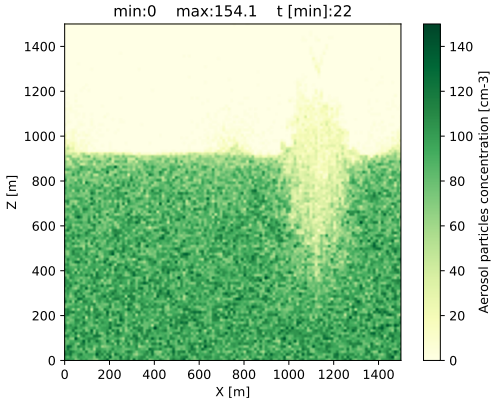
Computational grid: 128x128
Computational particles: 2^{21}

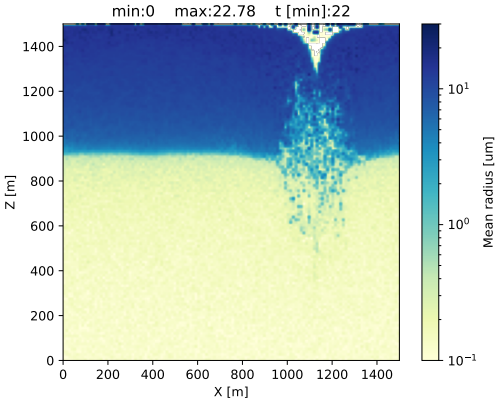




PySDM: example prescribed-flow simulation

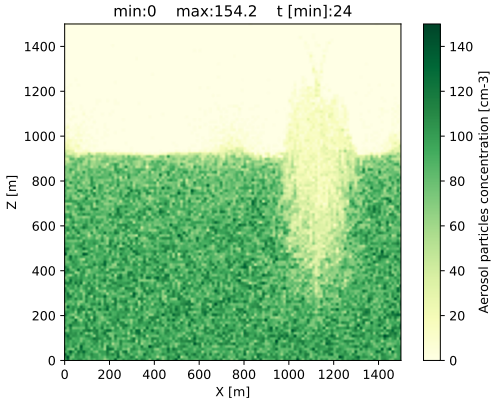
Computational grid: 128x128
Computational particles: 2^{21}

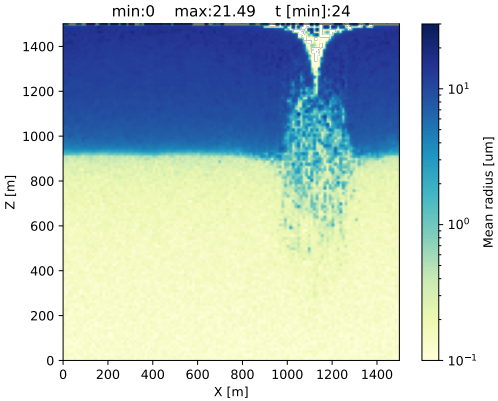




PySDM: example prescribed-flow simulation

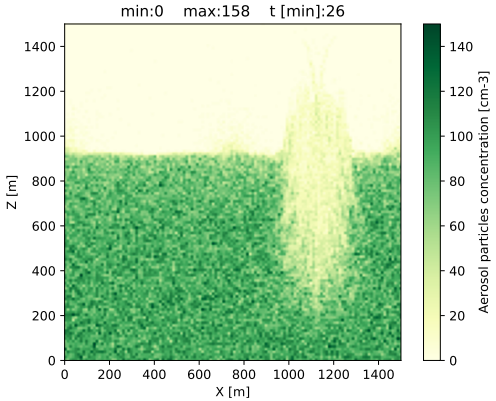
Computational grid: 128x128
Computational particles: 2^{21}

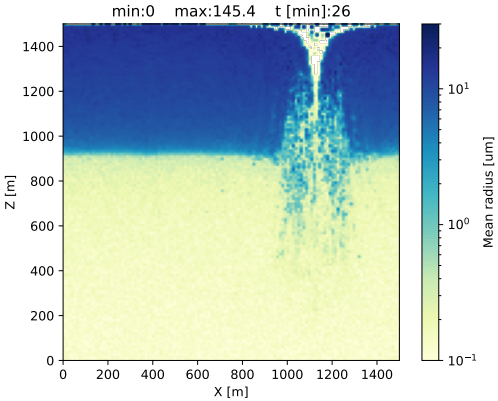




PySDM: example prescribed-flow simulation

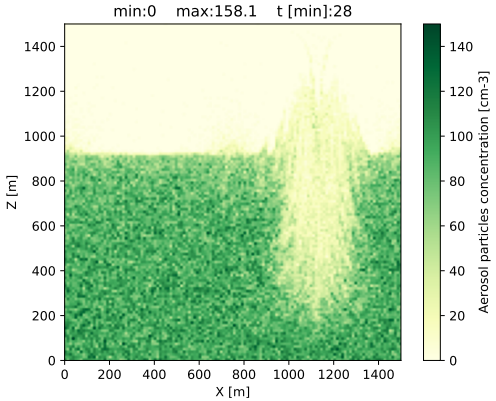
Computational grid: 128x128
Computational particles: 2^{21}

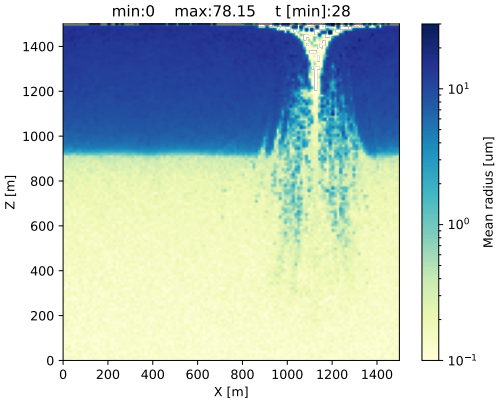




PySDM: example prescribed-flow simulation

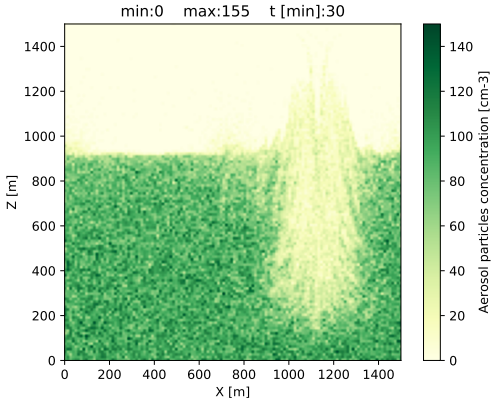
Computational grid: 128x128
Computational particles: 2^{21}

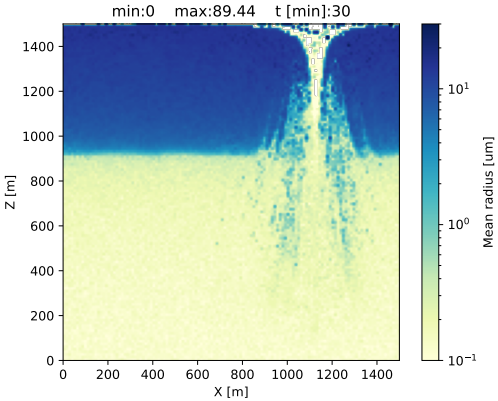




PySDM: example prescribed-flow simulation

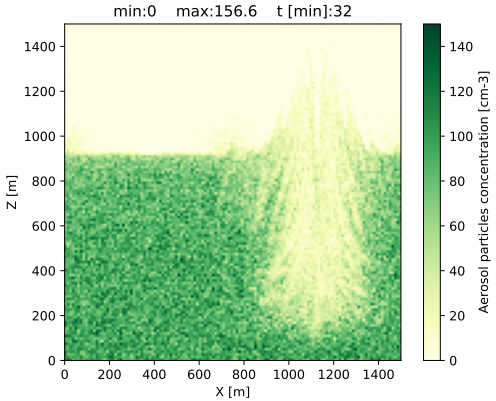
Computational grid: 128x128
Computational particles: 2^{21}

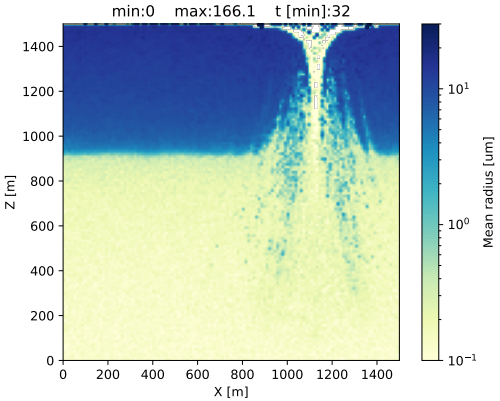




PySDM: example prescribed-flow simulation

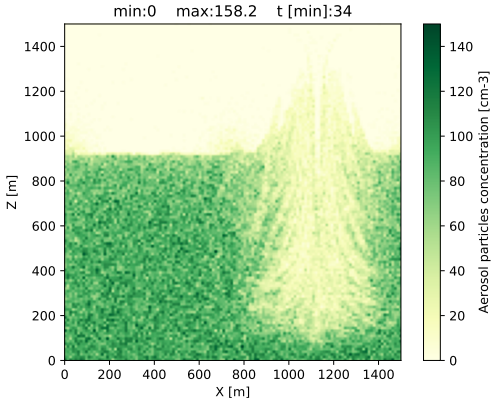
Computational grid: 128x128
Computational particles: 2^{21}

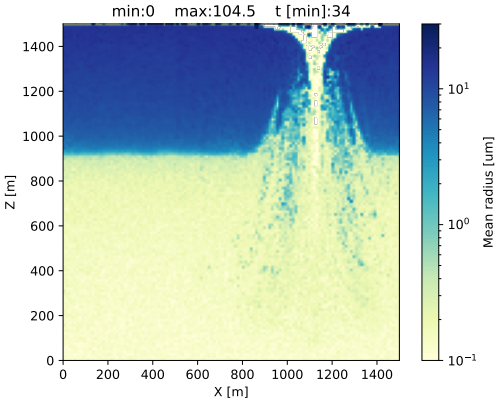




PySDM: example prescribed-flow simulation

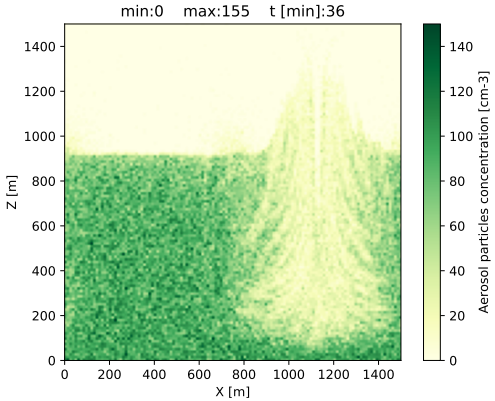
Computational grid: 128x128
Computational particles: 2^{21}

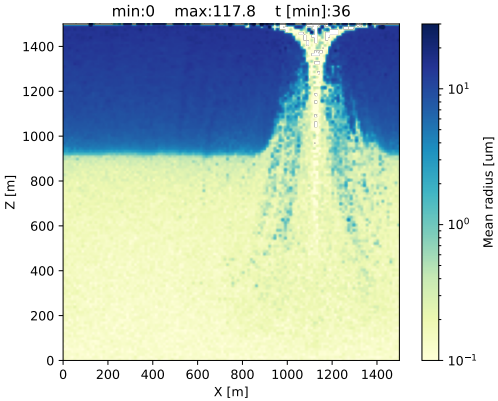




PySDM: example prescribed-flow simulation

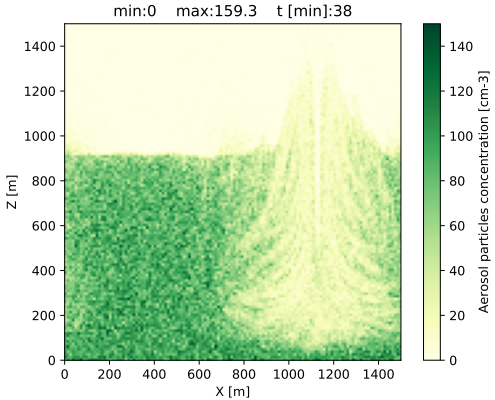
Computational grid: 128x128
Computational particles: 2^{21}

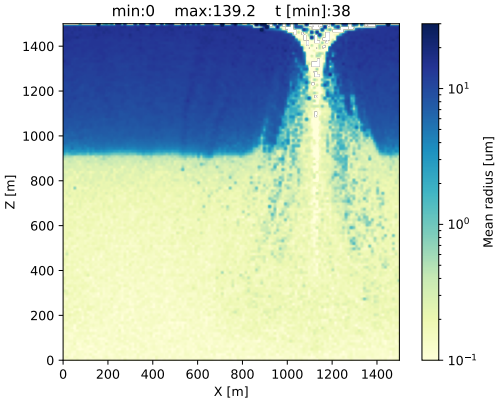




PySDM: example prescribed-flow simulation

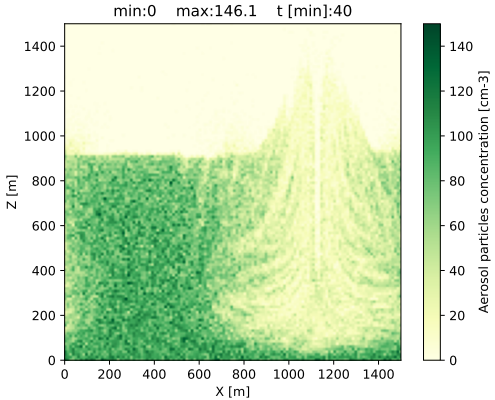
Computational grid: 128x128
Computational particles: 2^{21}

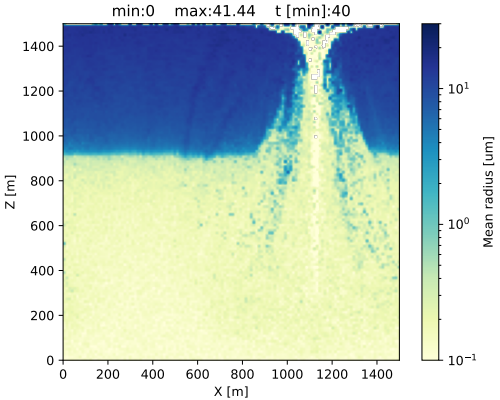




PySDM: example prescribed-flow simulation

Computational grid: 128x128
Computational particles: 2^{21}





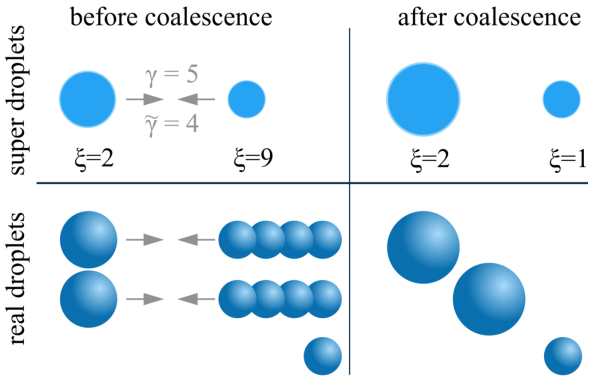
Super droplet method (Shima et al. 2009)

- Probabilistic collision representation
- $\mathcal{O}(n(\Omega))$
- collide all of $\min\{\xi_{[i]}, \xi_{[j]}\}$ (all or nothing)
- constant number of super-droplets

$$p = a(v_{[j]}, v_{[k]}) \max\{\xi_{[j]}, \xi_{[k]}\} \frac{(n(\Omega)^2 - n(\Omega))/2}{\lfloor n(\Omega)/2 \rfloor} \frac{\Delta t}{\Delta V}$$

$$\gamma = \lceil p - \phi_\gamma \rceil \text{ where } \phi_\gamma \sim \mathit{Uniform}[0, 1)$$

and for $\xi_{[j]} > \xi_{[k]}$ and $\tilde{\gamma} = \min\{\gamma, \lfloor \xi_{[j]}/\xi_{[k]} \rfloor\}$



PySDM validation vs. Golovin analytical solution (Smoluchowski eq.)

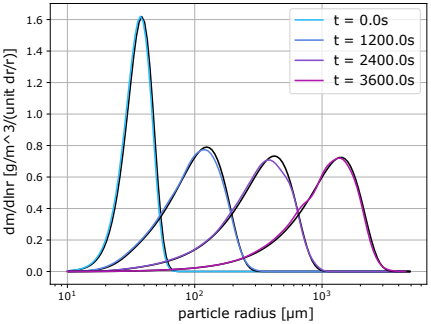


Fig. 6 Particle mass spectrum: SDM results (colour) vs. analytic solution (black)

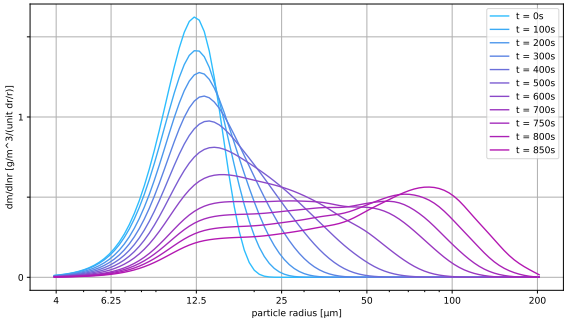


Fig. 8 Particle mass spectrum for gravitational kernel

SDM and parallel computing

hybrid hardware computation

- SDM is almost embarrassingly parallel
- GPU-resident data to avoid data transfer bottleneck
- simultaneous computation on CPU (Eulerian, CFD, simulation control) and GPU (particles, dispersed phase)

cell-wise parallelization vs. particle-wise parallelization

- GPU thread pool size $>$ grid size
- balancing

Monte-Carlo parallelization

- random numbers generation
- permutations (MergeShuffle)

PySDM: backend interface and storage layer

model state

```
idx = Index(N_SD, int)
multiplicities = IndexedStorage(idx, N_SD, int)
attributes = IndexedStorage(idx, (N_ATTR, N_SD), float)
volume_view = attributes[0:1, :]

cell_id = IndexedStorage(idx, N_SD, int)
cell_idx = Index(N_CELL, int)
cell_start = Storage(N_CELL + 1, int)

pair_prob = Storage(N_SD//2, float)
pair_flag = PairIndicator(N_SD, bool)

u01 = Storage(N_SD, float)
```

algorithm flow

```
remove_if_equal_0(idx, multiplicities)
urand(u01)
shuffle_per_cell(cell_start, idx, cell_idx, cell_id, u01)
flag_pairs(pair_flag, cell_id, cell_idx, cell_start)
coalescence_kernel(pair_prob, pair_flag, volume_view)
times_max(pair_prob, multiplicities, pair_flag)
normalize(pair_prob, dt, dv, cell_id, cell_idx, cell_start)
urand(u01[:N_SD//2])
compute_gamma(pair_prob, u01[:N_SD//2])
update_attributes(multiplicities, attributes, pair_prob)
```

Storage - a base container which is intended to adapt the interfaces of the underlying implementation-dependent array containers.

IndexedStorage - subclass of Storage is intended as container for super-particle attributes.

Index - class features permutation and array-shrinking logic.

PairIndicator - class provides an abstraction layer facilitating pairwise operations inherent to several steps of the SDM workflow.

PySDM CPU/threads backend (built on top of Numba/LLVM)

```python
@numba.njit()
def coalescence_body(n, idx, length, attributes, gamma, is_first_in_pair):
    for i in prange(length // 2):
        if gamma[i] == 0:
            continue
        j, k = pair_indices(i, idx, is_first_in_pair)

        new_n = n[j] - gamma[i] * n[k]
        if new_n > 0:
            n[j] = new_n
            for a in range(0, len(attributes)):
                attributes[a, k] += gamma[i] * attributes[a, j]
        else:  # new_n == 0
            n[j] = n[k] // 2
            n[k] = n[k] - n[j]
            for a in range(0, len(attributes)):
                attributes[a, j] = gamma[i] * attributes[a, j] + attributes[a, k]
                attributes[a, k] = attributes[a, j]
```

- *just-in-time* compilation for Python functions
- CPU parallelization (multi-threading)
- generates optimized machine code from pure Python code using the LLVM compiler infrastructure
- on-the-fly native code generation
- integration with the Python scientific software stack (thanks to Numpy)

PySDM GPU backend (built on top of ThrustRTC/NVRTC)

```
def coalescence(n, idx, length, attributes, gamma, healthy):
    n_sd = ThrustRTC.DVInt64(attributes.shape[1])
    n_attr = ThrustRTC.DVInt64(attributes.shape[0])
    kernel = ThrustRTC.For(['n', 'idx', 'n_sd', 'attributes', 'n_attr', 'gamma', 'healthy'], "i", '''
            if (gamma[i] == 0) return;
            auto j = idx[2 * i];
            auto k = idx[2 * i + 1];
            auto new_n = n[j] - gamma[i] * n[k];
            if (new_n > 0) {
                n[j] = new_n;
                for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd)
                    attributes[attr + k] += gamma[i] * attributes[attr + j];
            }
            else {  // new_n == 0
                n[j] = (int64_t)(n[k] / 2);
                n[k] = n[k] - n[j];
                for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd) {
                    attributes[attr + j] = gamma[i] * attributes[attr + j] + attributes[attr + k];
                    attributes[attr + k] = attributes[attr + j];
                }
            }
            ''')
    kernel.launch_n(length // 2, [n.data, idx.data, n_sd, attributes.data, n_attr, gamma.data, healthy.data])
```

- *run-time-compilation*
- library of general GPU algorithms, similar to Thrust, can be used in C-interop languages
- flexibility wrt level of indirection:
  – built-in algorithms or
  – CUDA-free loop bodies or
  – custom CUDA kernels
- CUDA only
- CURandRTC: random number generator (XORWOW)

Take-home messages (expounded in the paper)

- particle-based microphysics & SDM for coagulation
- SDM & separation of concerns (e.g., parallelisation logic, algorithm workflow, data dependencies, CFD coupling)
- PySDM: new implementation of SDM with two parallel backends (sharing an API):
  - targeting different hardware (CPU/threads vs. GPU),
  - based on different JIT technology (Numba/LLVM, ThrustRTC/NVRTC)
  - Numpy-independent storage layer with particle permutation indirection
- simultaneous CPU+GPU usage capability to fully leverage hybrid platforms (async functionality in ThrustRTC)
- journal-review-level reproduciblity (goal: $<$1h time-to-reproduce)

PySDM (github.com/atmos-cloud-sim-uj/PySDM)

README.md

build passing coverage 61%

PySDM

PySDM is a package for simulating the dynamics of population of particles immersed in moist air
based (a.k.a. super-droplet) approach to represent aerosol/cloud/rain microphysics. The package
high-performance implementation of the Super-Droplet Method (SDM) Monte-Carlo algorithm f
collisional growth (Shima et al. 2009), hence the name. PySDM has two alternative parallel numb
available: multi-threaded CPU backend based on Numba and GPU-resident backend built on top

Demos:

- Shima et al. 2009 Fig. 2 launch binder Open in Colab
  (Box model, coalescence only, test case employing Golovin analytical solution)
- Berry 1967 Figs. 6, 8, 10 launch binder Open in Colab
  (Box model, coalescence only, test cases for realistic kernels)

Thank you for your attention!

more:

`http://atmos.ii.uj.edu.pl`

piotr.bartman@doctoral.uj.edu.pl
sylwester.arabas@uj.edu.pl

funding acknowledgment:
Foundation for Polish Science / European Union