Bridging Pythonicity with performance:
Monte-Carlo on GPU case study using ThrustRTC and CURandRTC

Piotr Bartman and Sylwester Arabas

Jagiellonian University

Jagiellonian University, Kraków, Poland

- founded in 1364
- among 20 oldest continuously operating univ. in the world
- ca. 40 000 students, 7000 staff (4000 acad.), 16 faculties

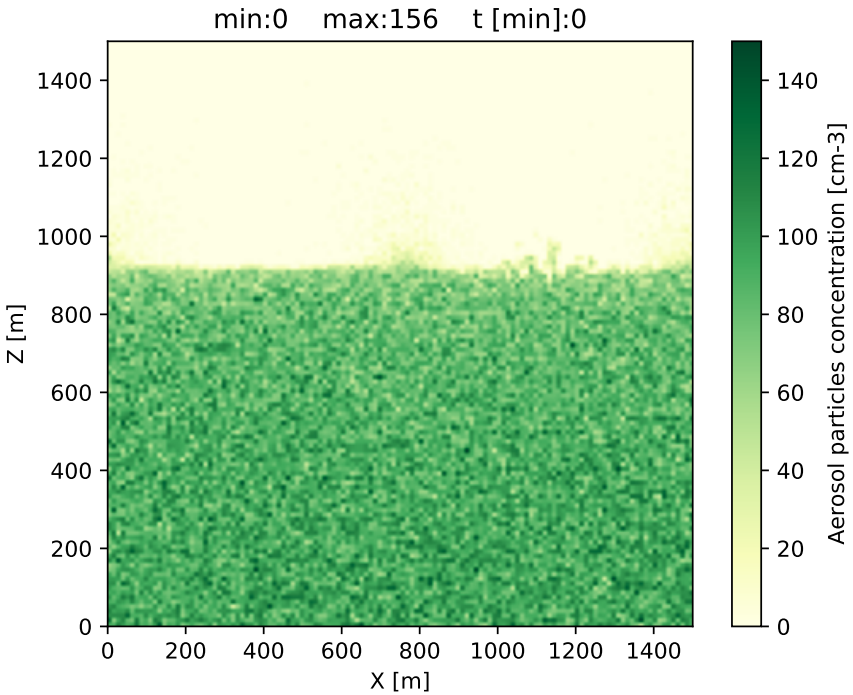
Scientific context

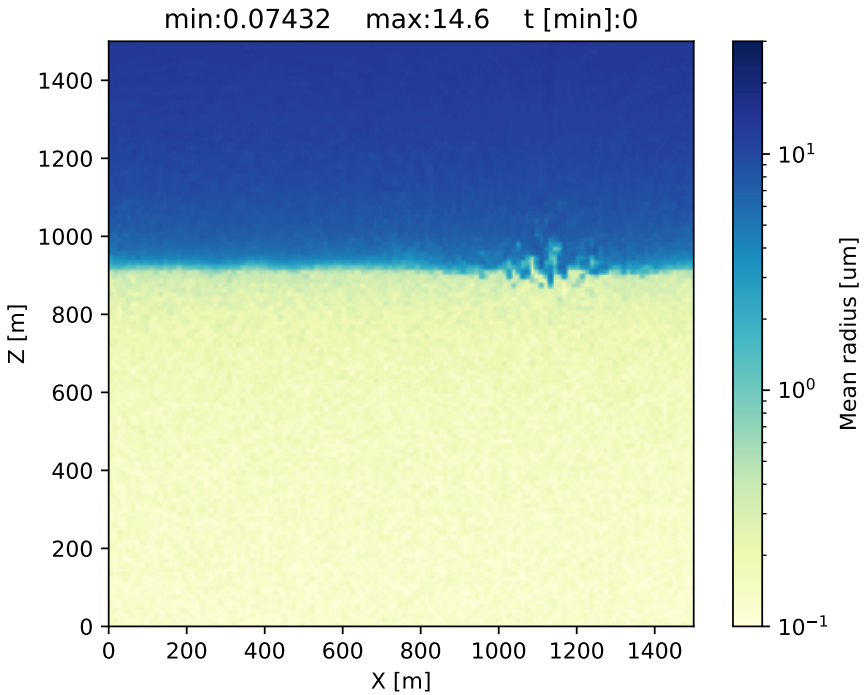
Context: aerosol-cloud-precipitation interactions

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

Context: aerosol-cloud-precipitation interactions

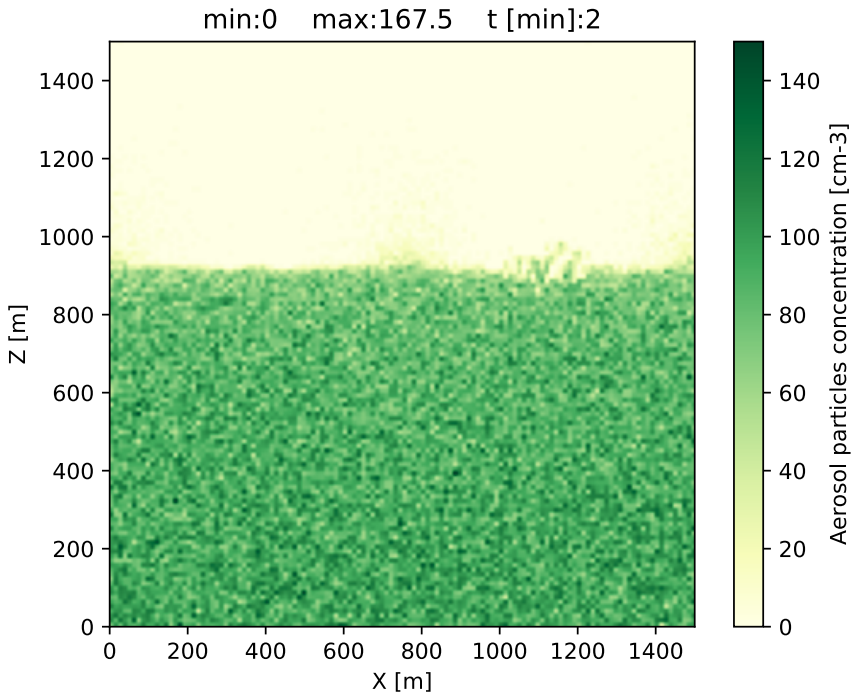
Computational grid: 128x128
Computational particles: 2^{21}

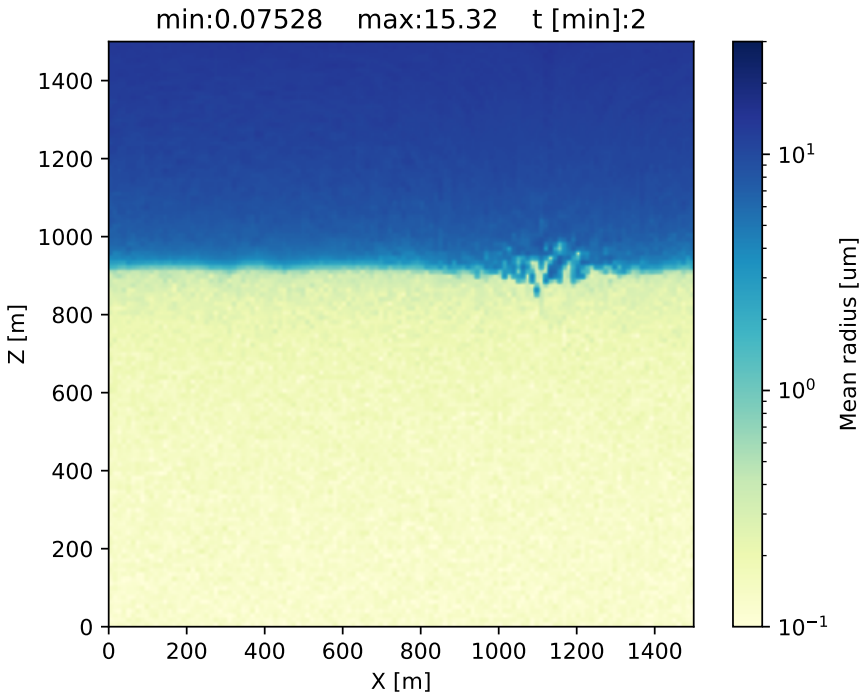




Context: aerosol-cloud-precipitation interactions

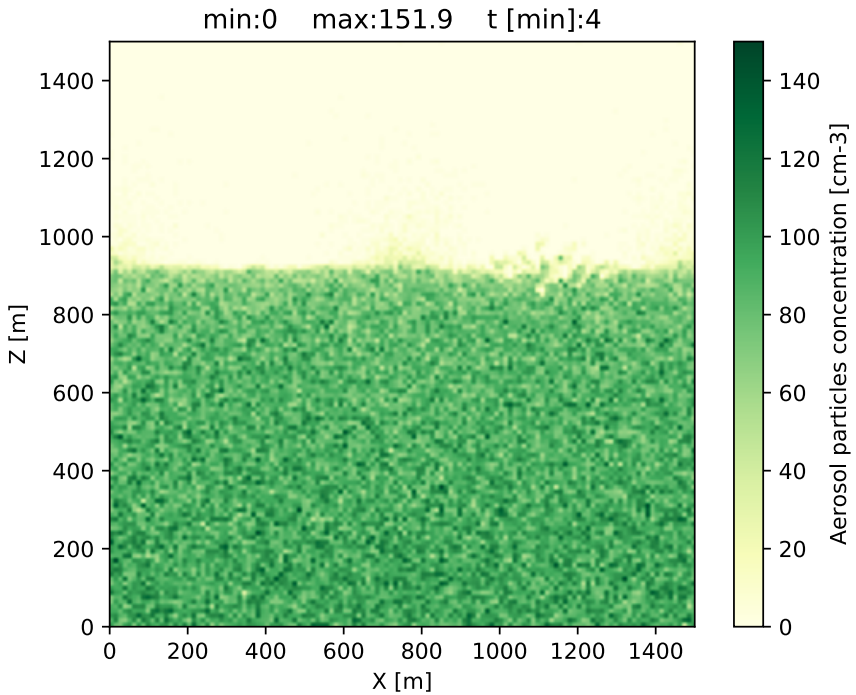
Computational grid: 128x128
Computational particles: 2^{21}

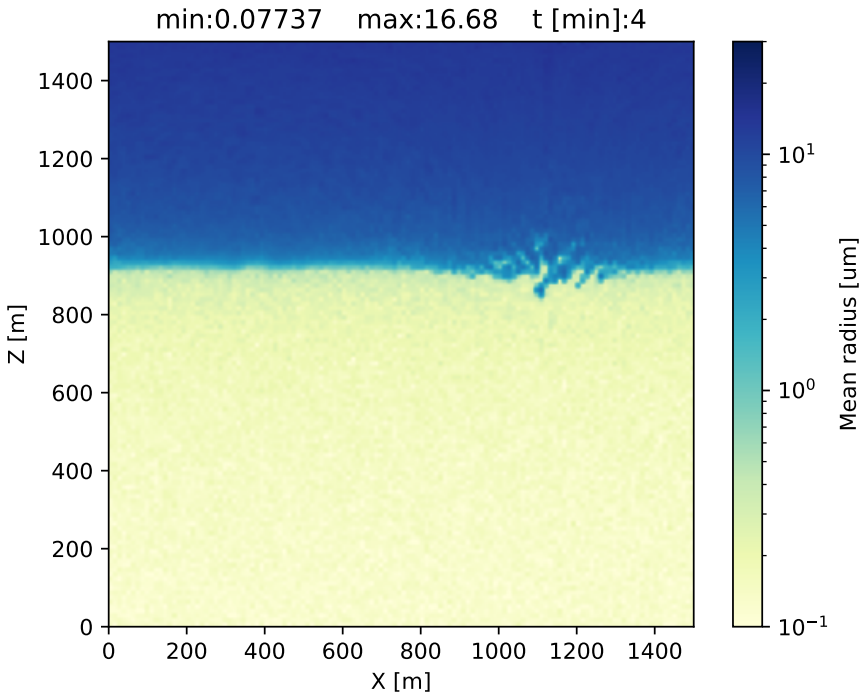




Context: aerosol-cloud-precipitation interactions

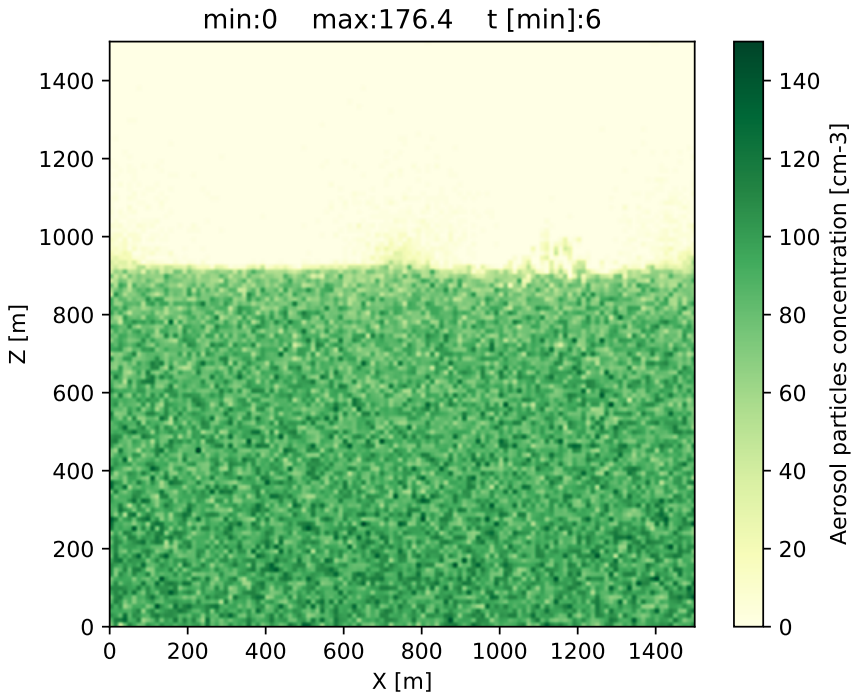
Computational grid: 128x128
Computational particles: 2^{21}

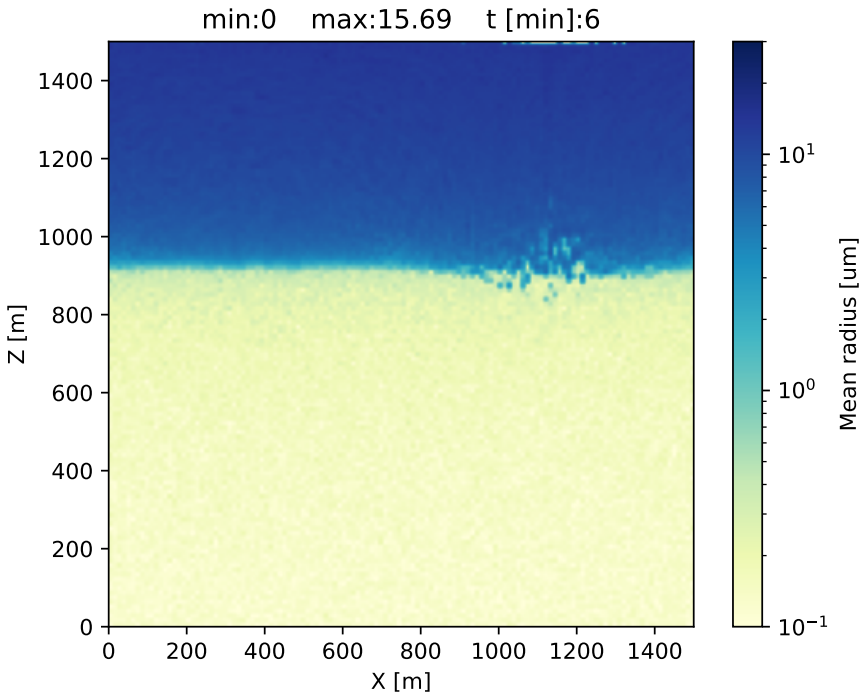




Context: aerosol-cloud-precipitation interactions

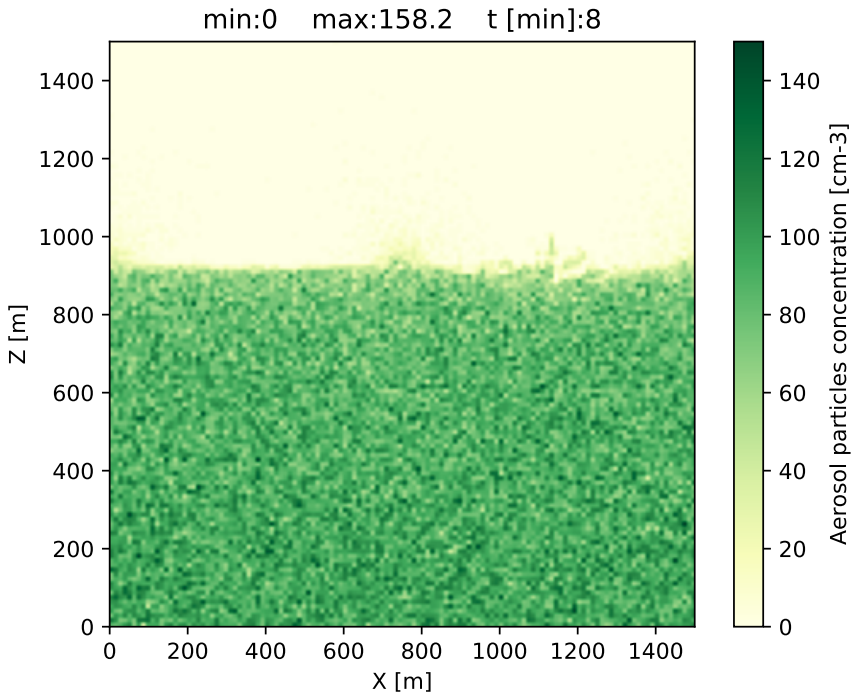
Computational grid: 128x128
Computational particles: 2^{21}

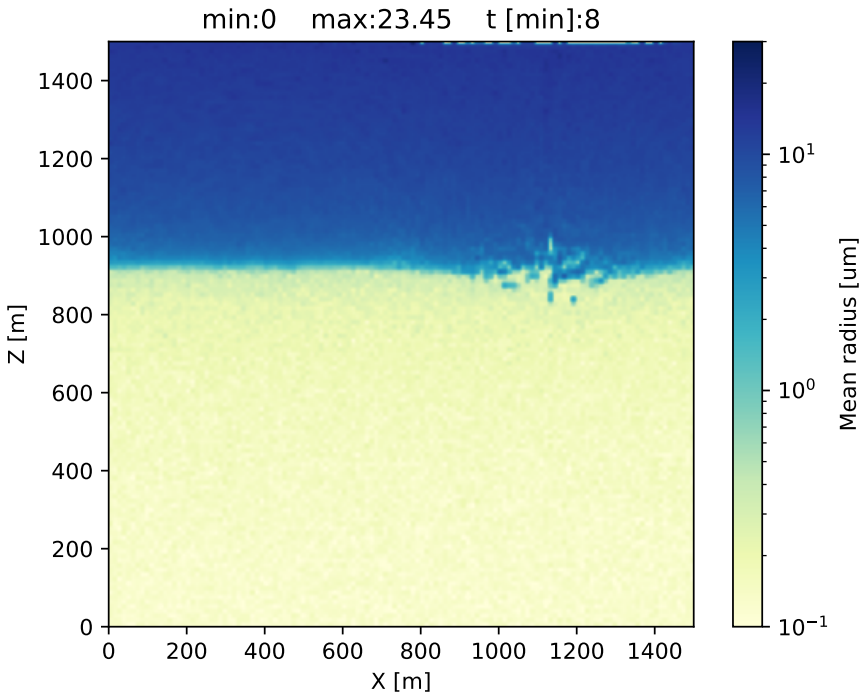




Context: aerosol-cloud-precipitation interactions

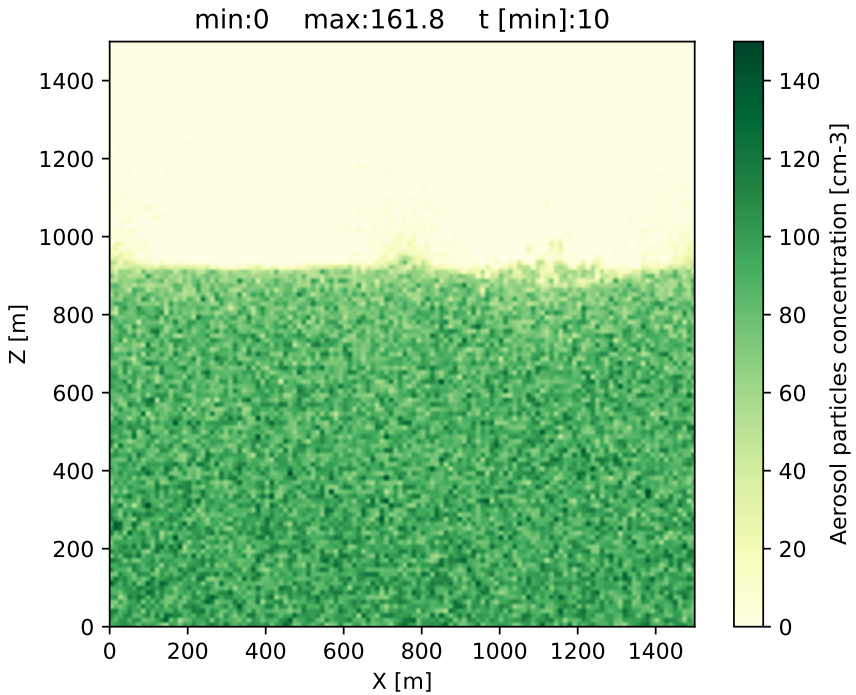
Computational grid: 128x128
Computational particles: 2^{21}

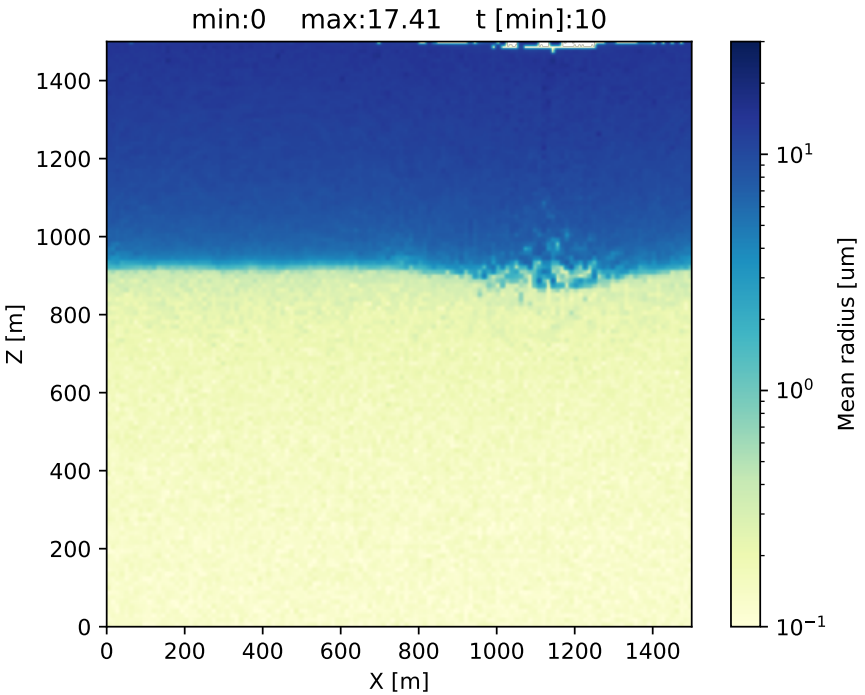




Context: aerosol-cloud-precipitation interactions

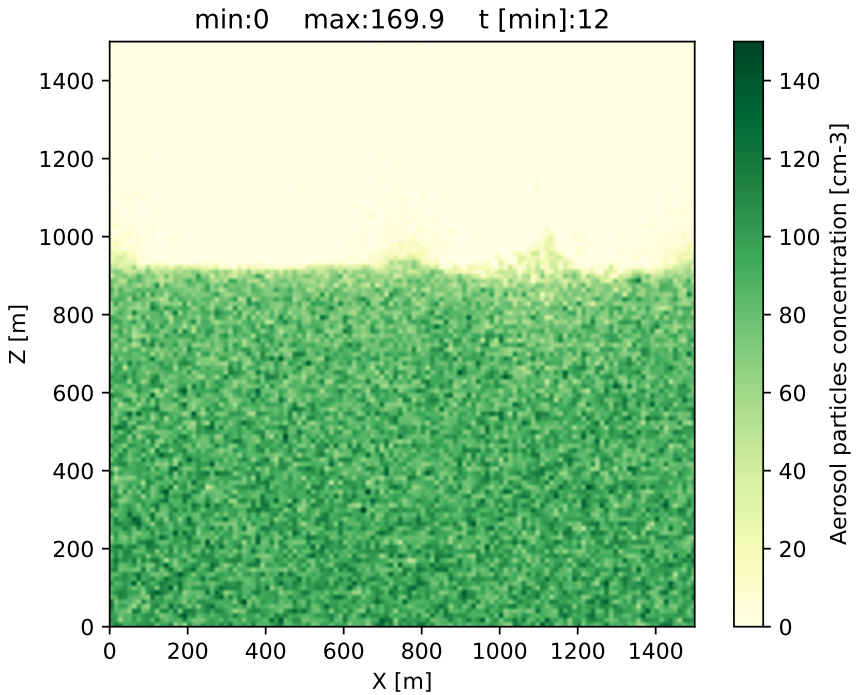
Computational grid: 128x128
Computational particles: 2^{21}

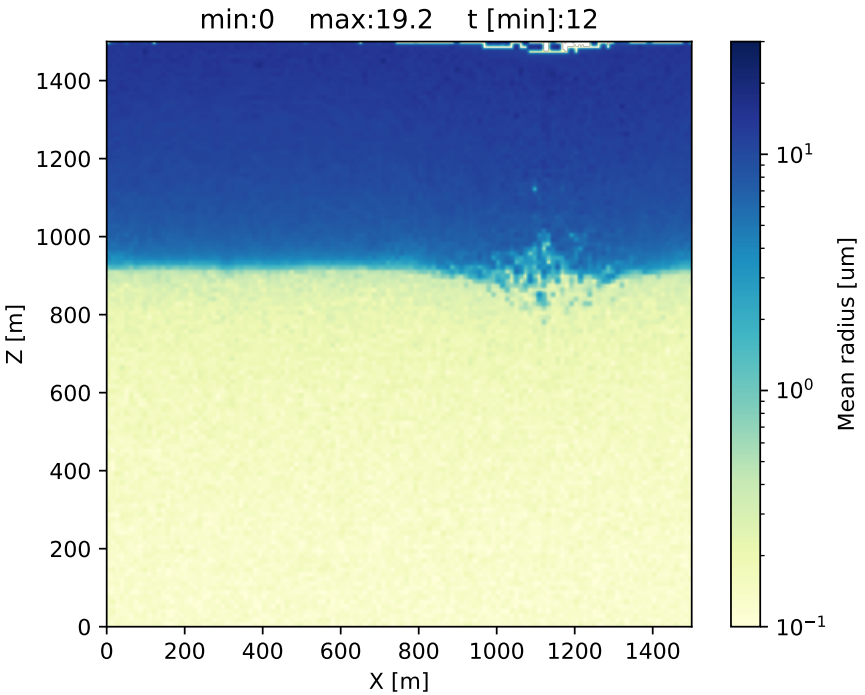




Context: aerosol-cloud-precipitation interactions

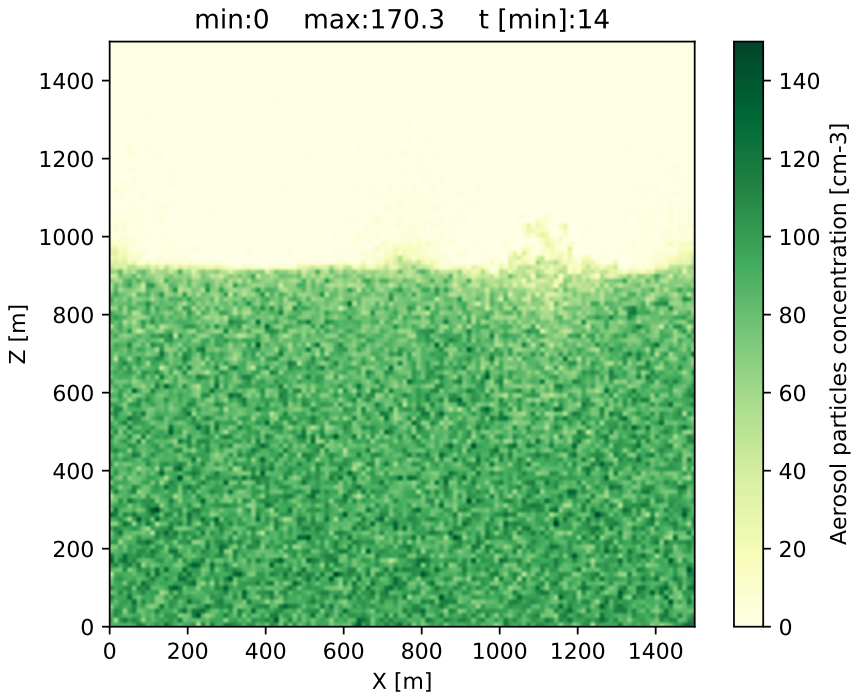
Computational grid: 128x128
Computational particles: 2^{21}

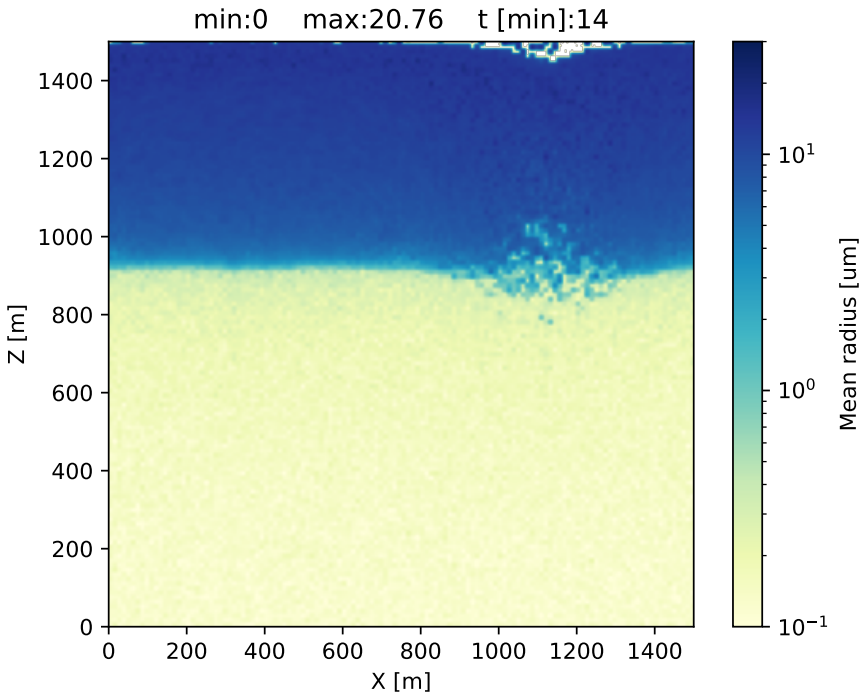




Context: aerosol-cloud-precipitation interactions

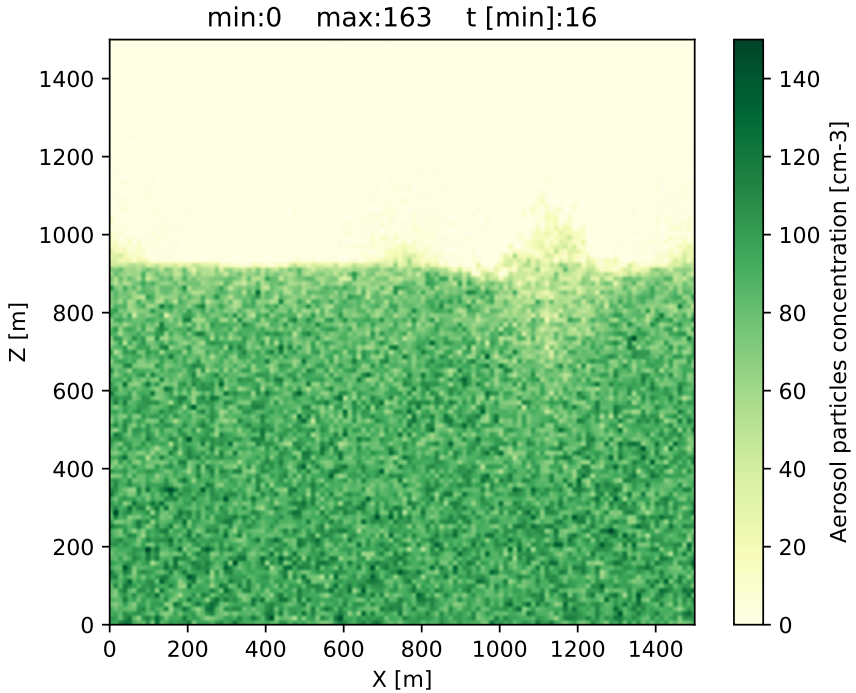
Computational grid: 128x128
Computational particles: 2^{21}

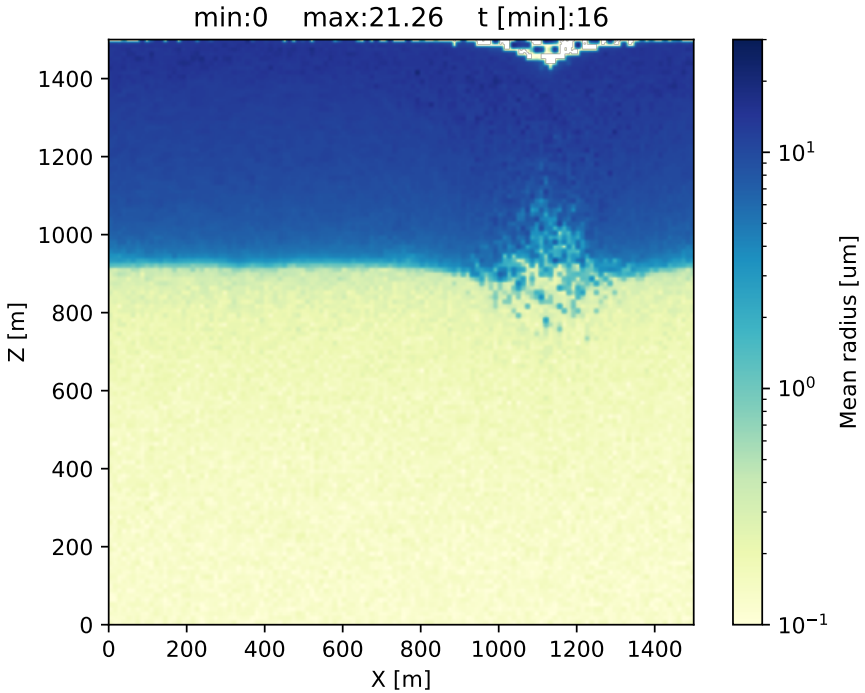




Context: aerosol-cloud-precipitation interactions

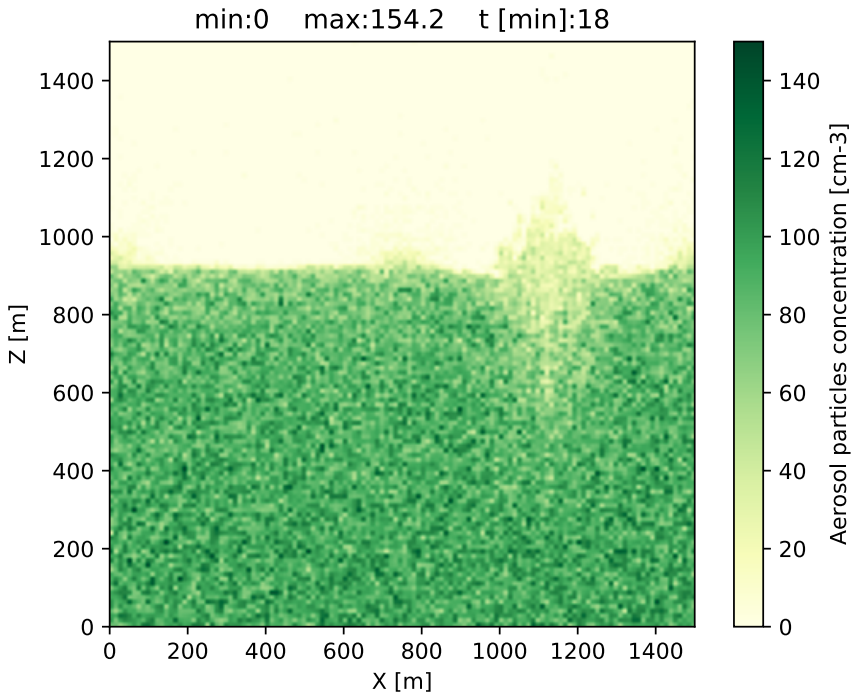
Computational grid: 128x128
Computational particles: 2^{21}

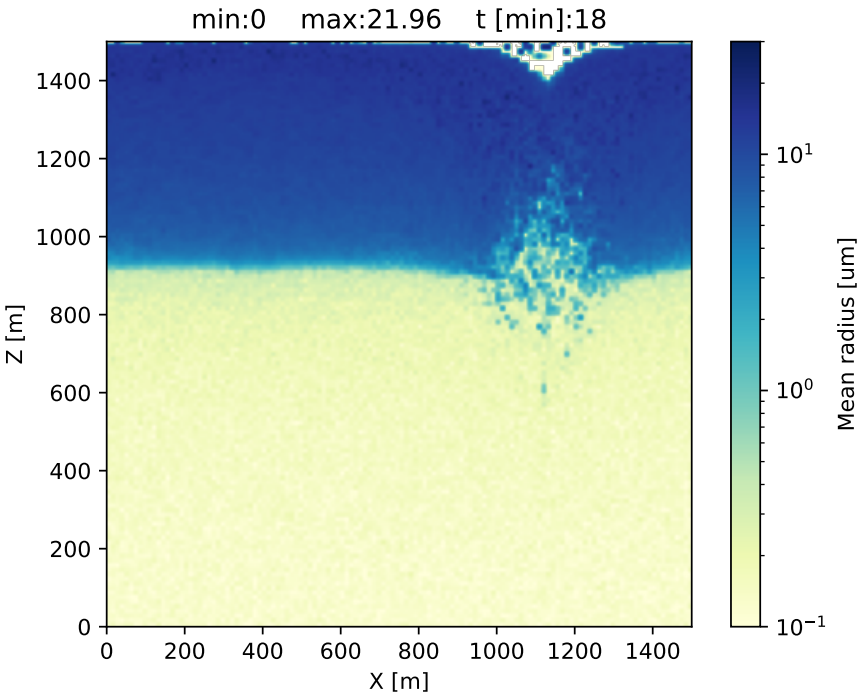




Context: aerosol-cloud-precipitation interactions

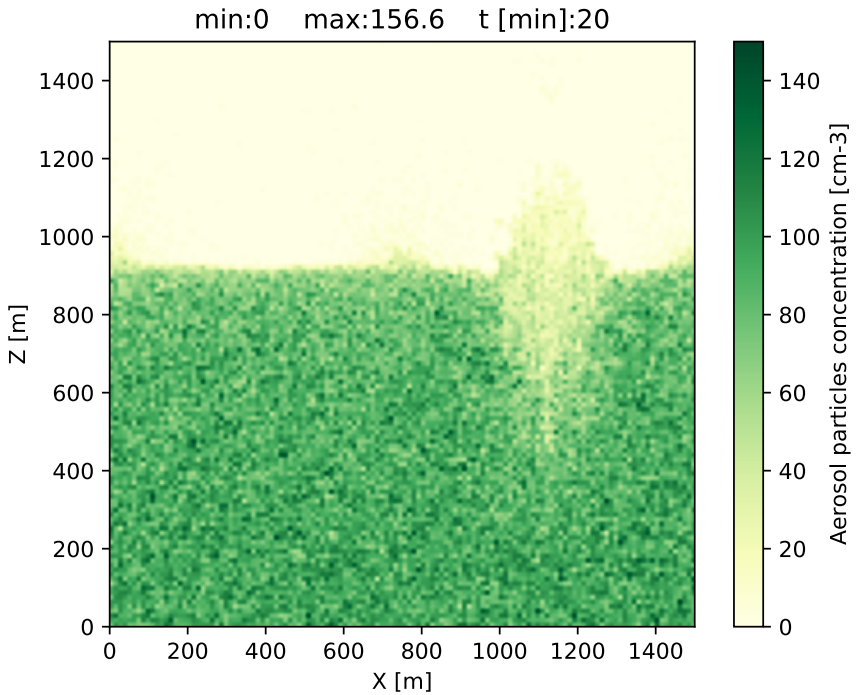
Computational grid: 128x128
Computational particles: 2^{21}

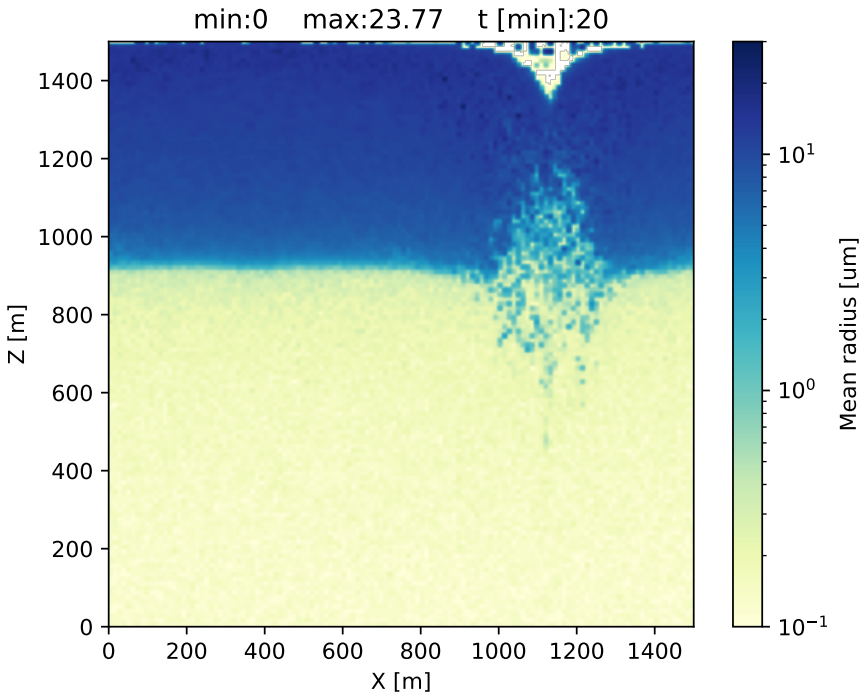




Context: aerosol-cloud-precipitation interactions

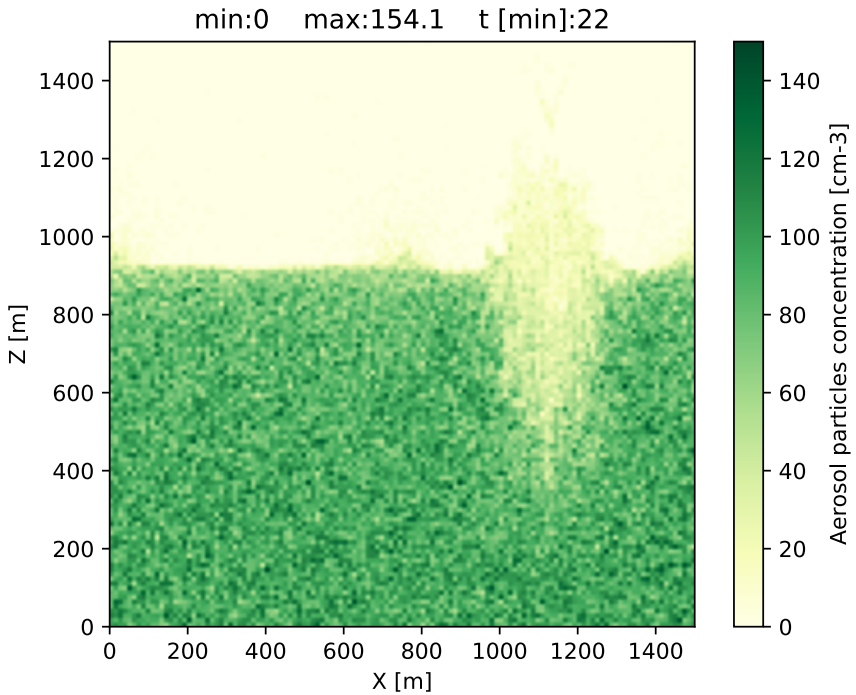
Computational grid: 128x128
Computational particles: 2^{21}

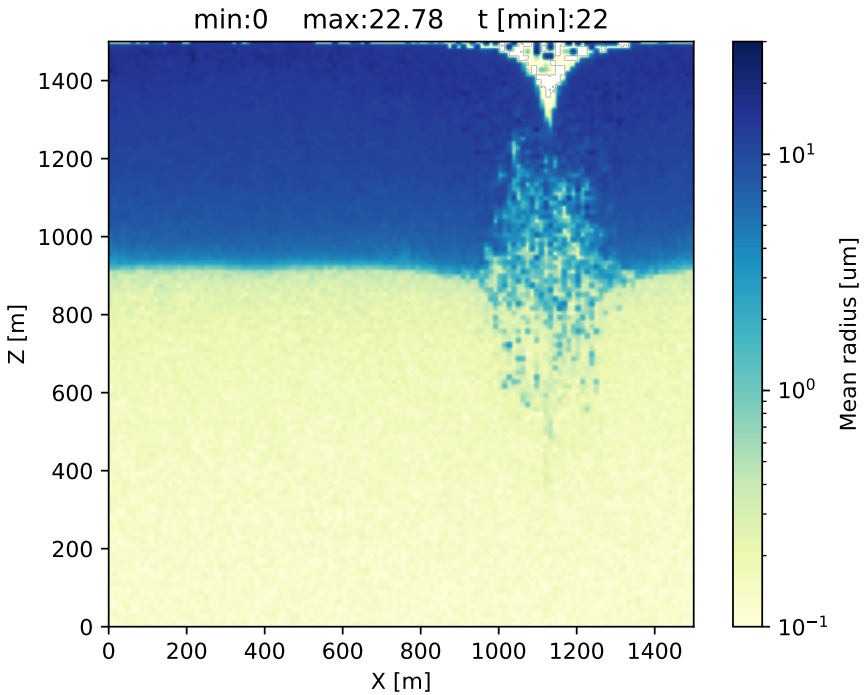




Context: aerosol-cloud-precipitation interactions

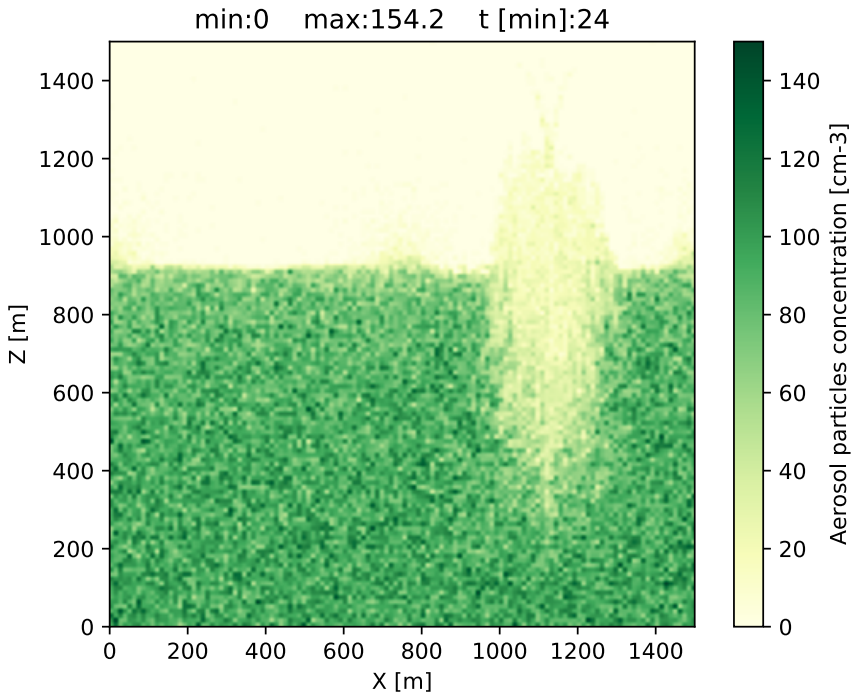
Computational grid: 128x128
Computational particles: 2^{21}

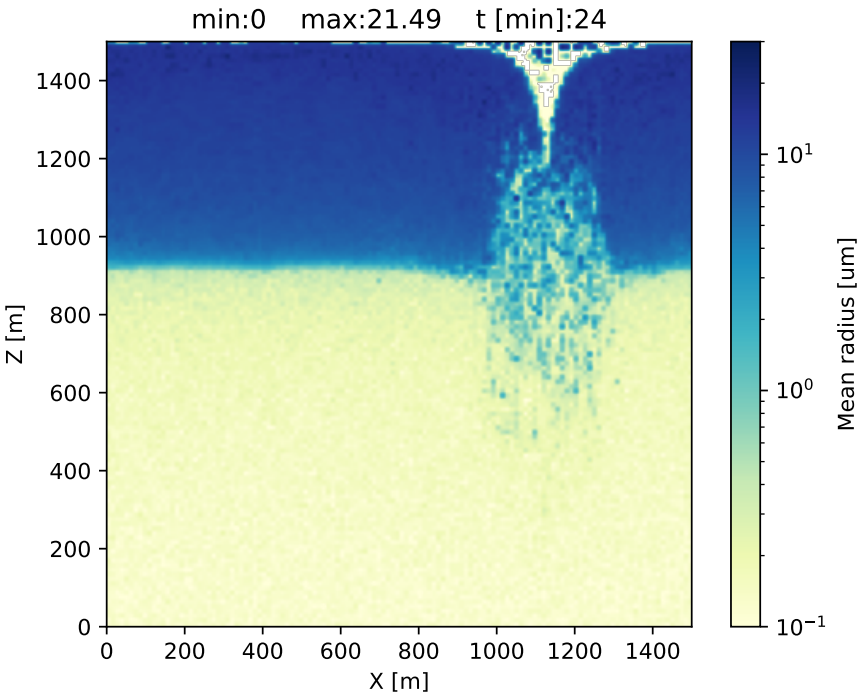




Context: aerosol-cloud-precipitation interactions

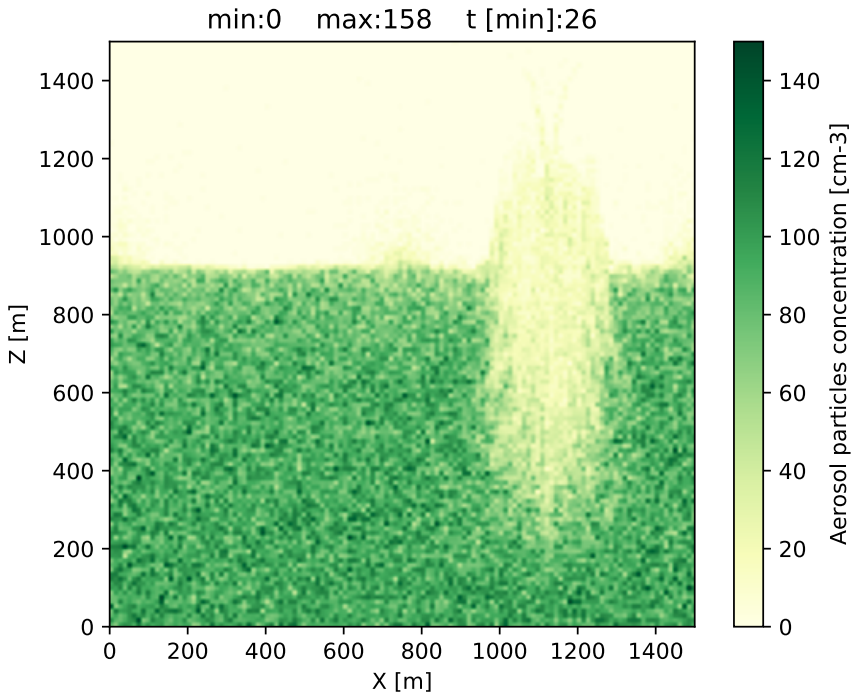
Computational grid: 128x128
Computational particles: 2^{21}

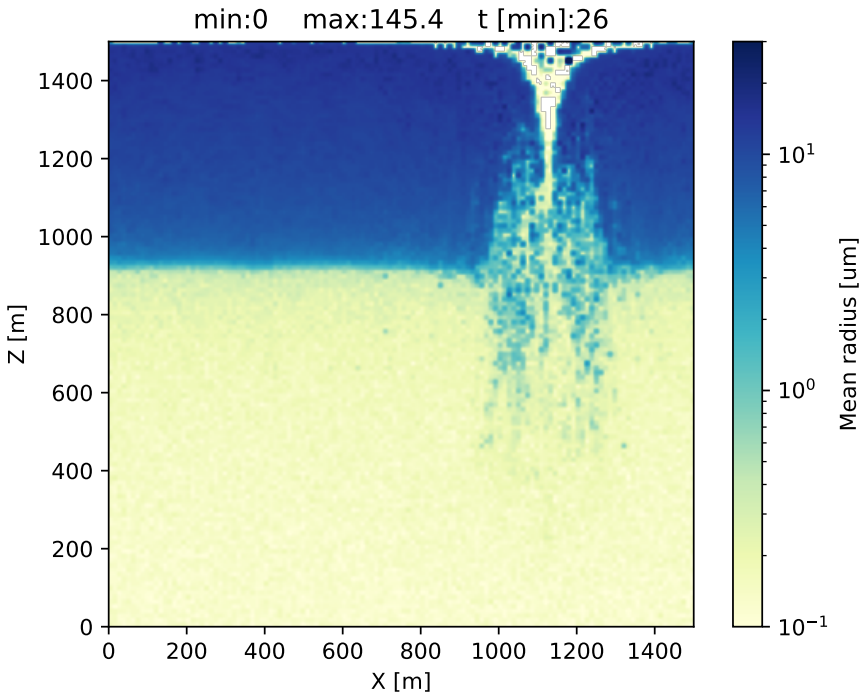




Context: aerosol-cloud-precipitation interactions

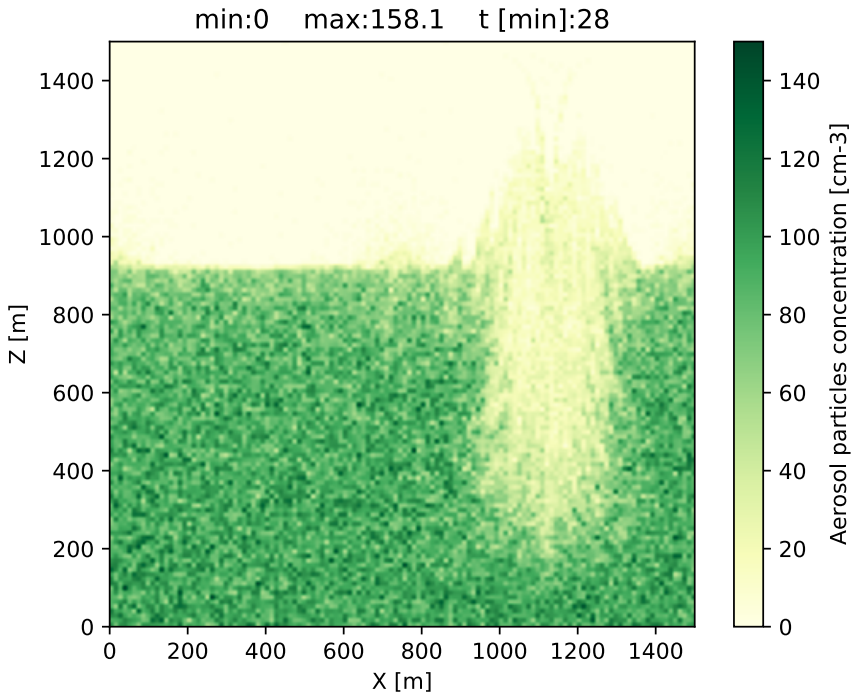
Computational grid: 128x128
Computational particles: 2^{21}

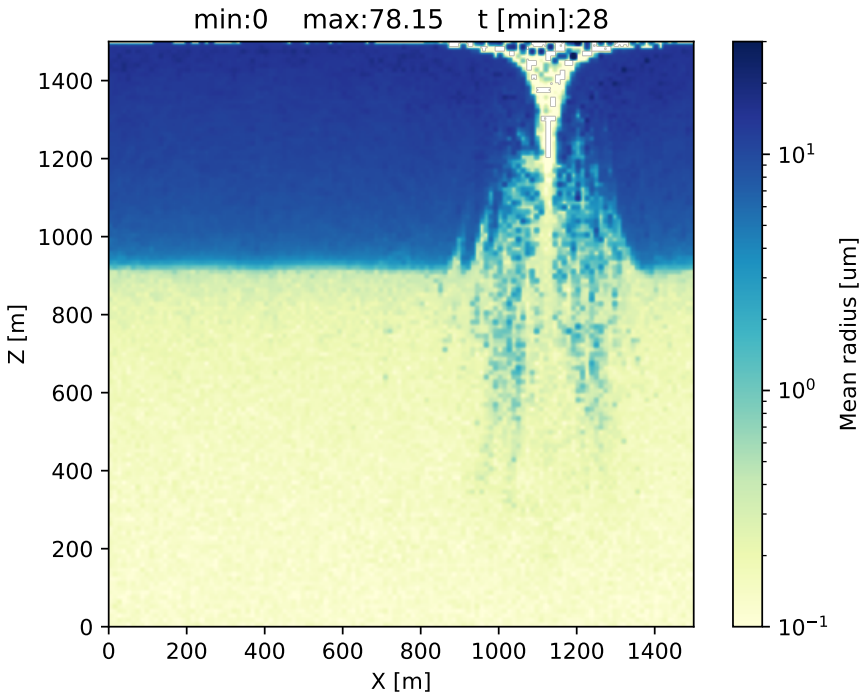




Context: aerosol-cloud-precipitation interactions

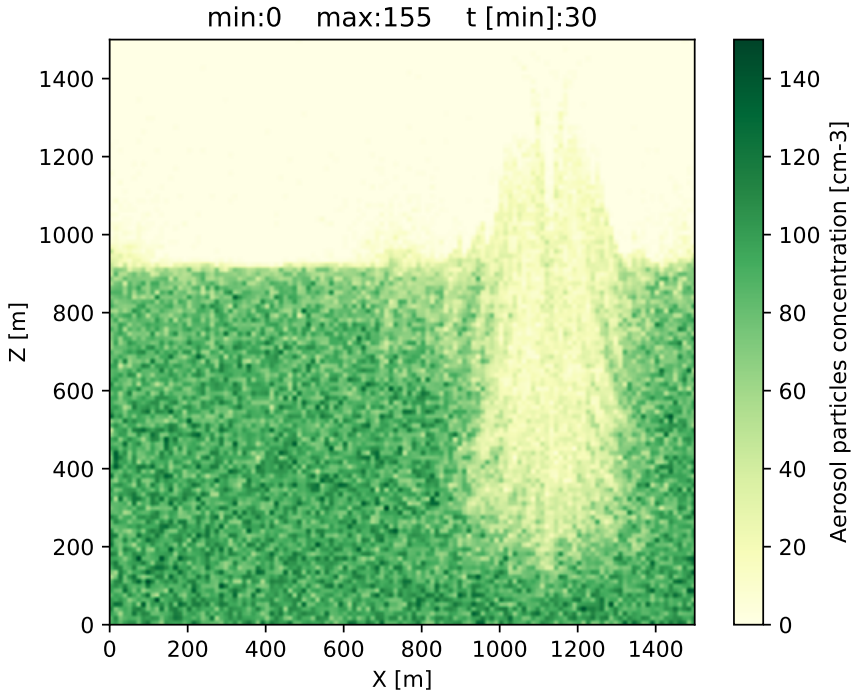
Computational grid: 128x128
Computational particles: 2^{21}

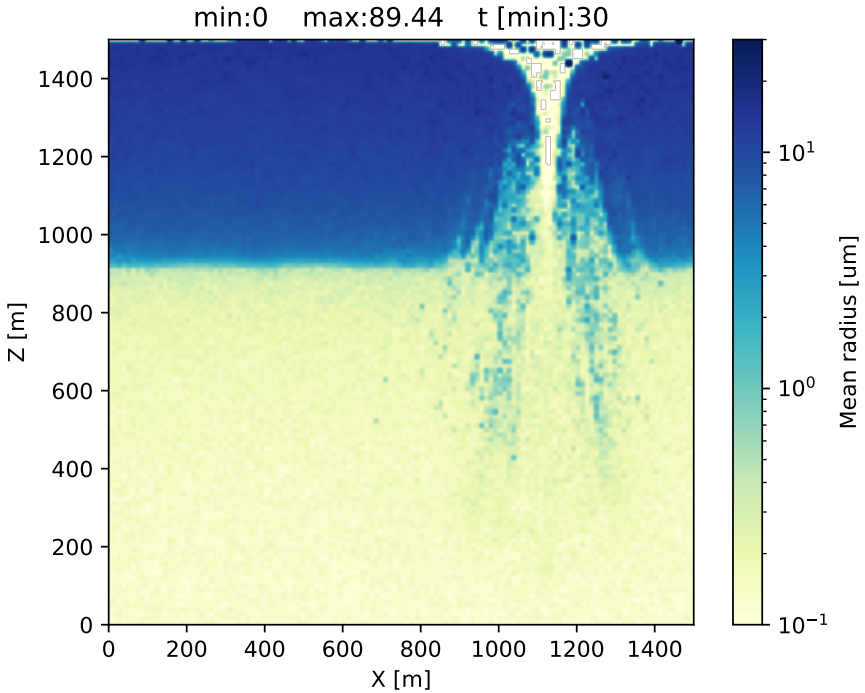




Context: aerosol-cloud-precipitation interactions

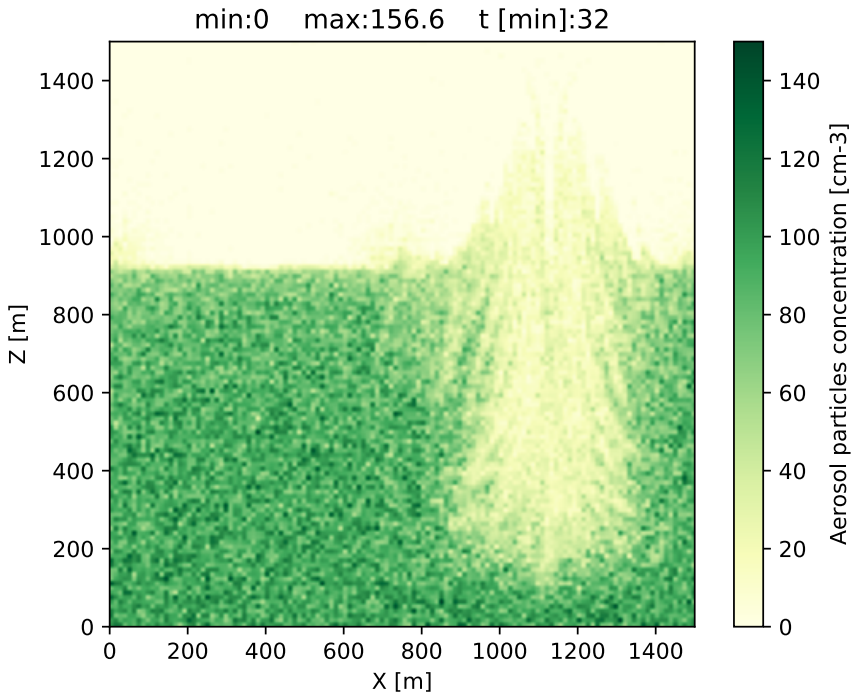
Computational grid: 128x128
Computational particles: 2^{21}

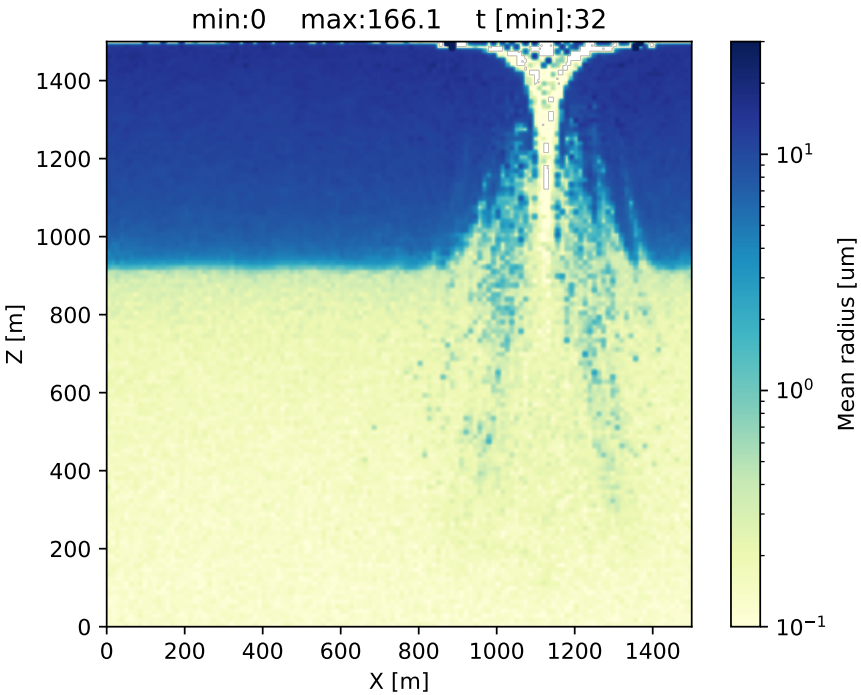




Context: aerosol-cloud-precipitation interactions

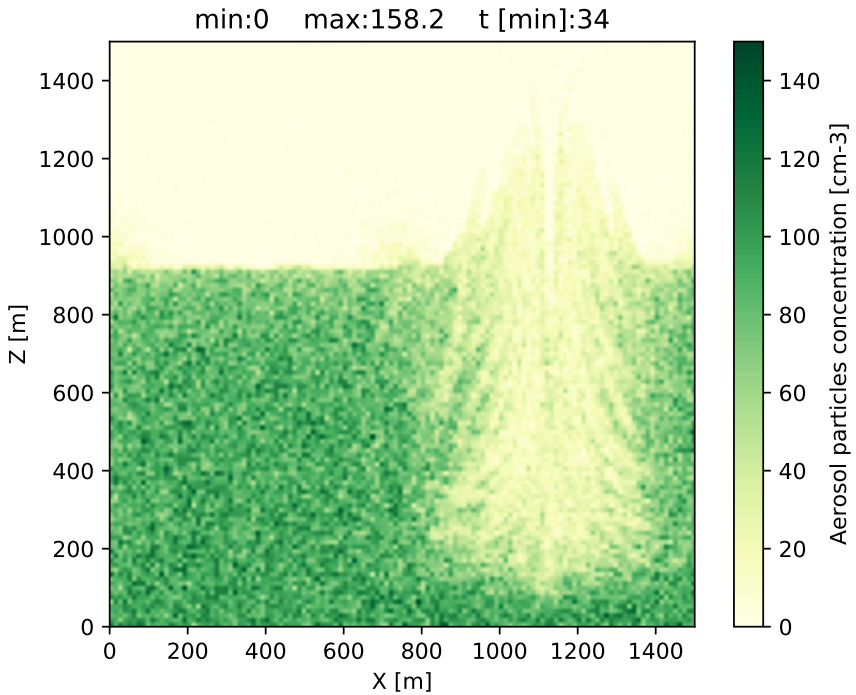
Computational grid: 128x128
Computational particles: 2^{21}

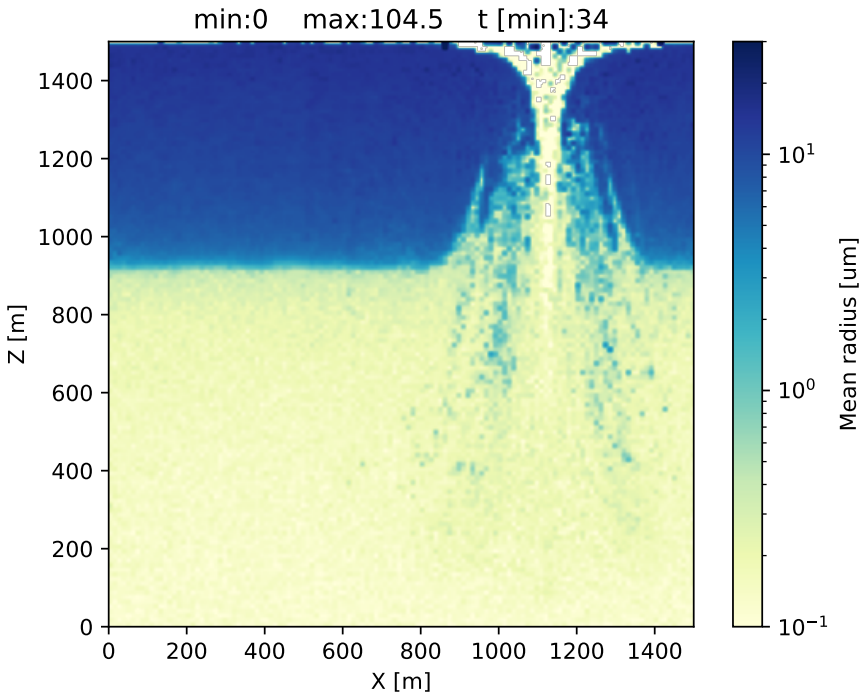




Context: aerosol-cloud-precipitation interactions

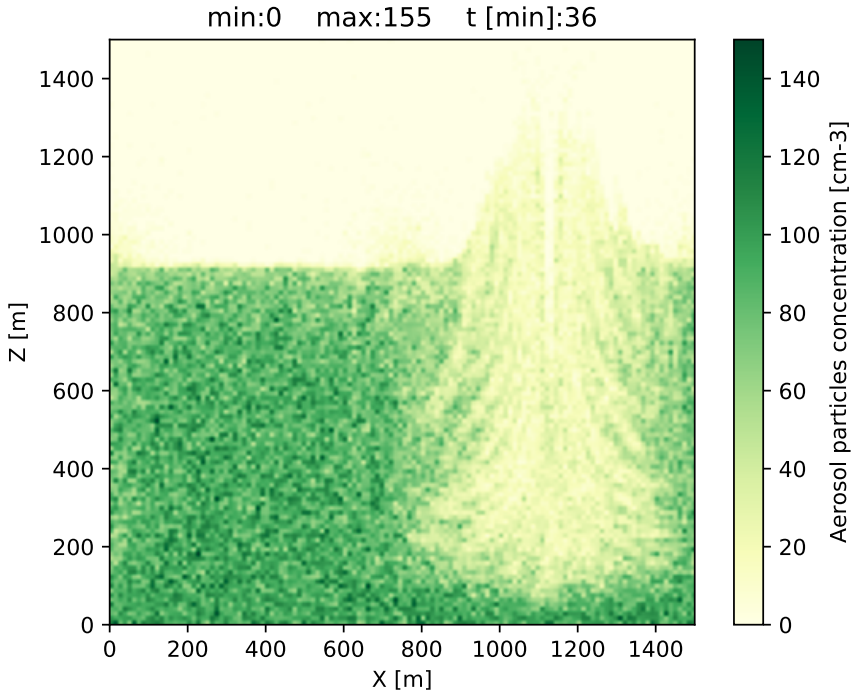
Computational grid: 128x128
Computational particles: 2^{21}

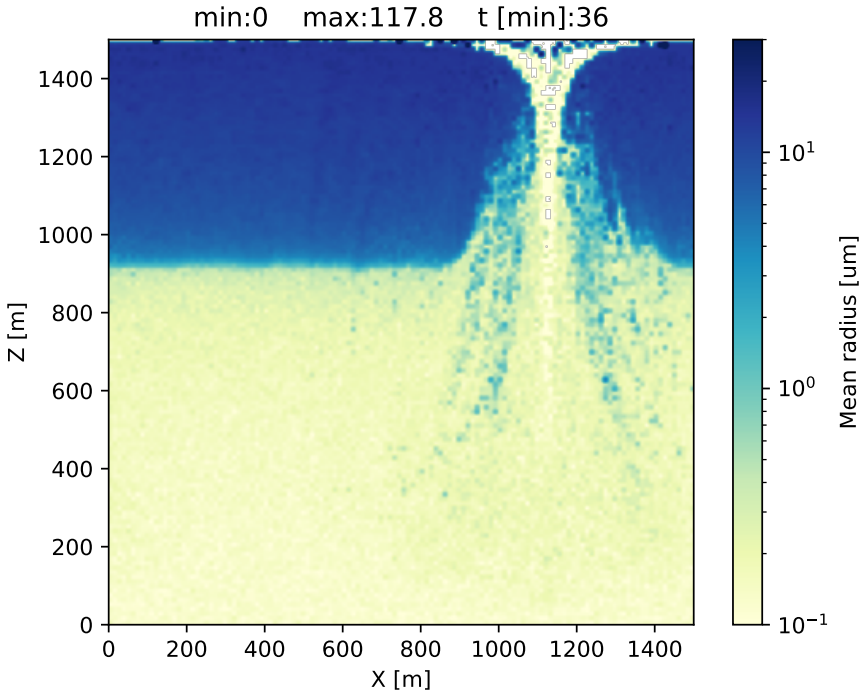




Context: aerosol-cloud-precipitation interactions

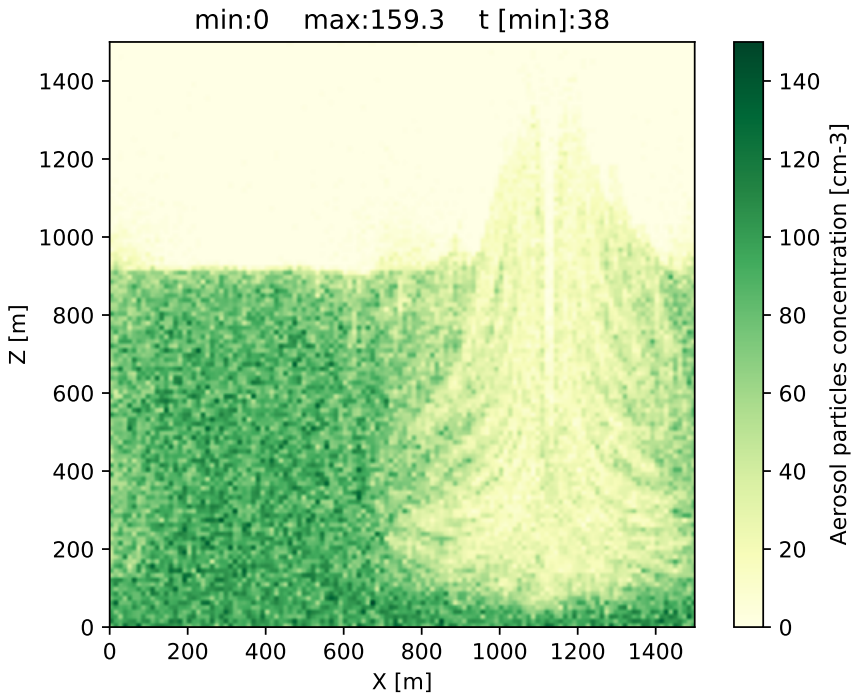
Computational grid: 128x128
Computational particles: 2^{21}

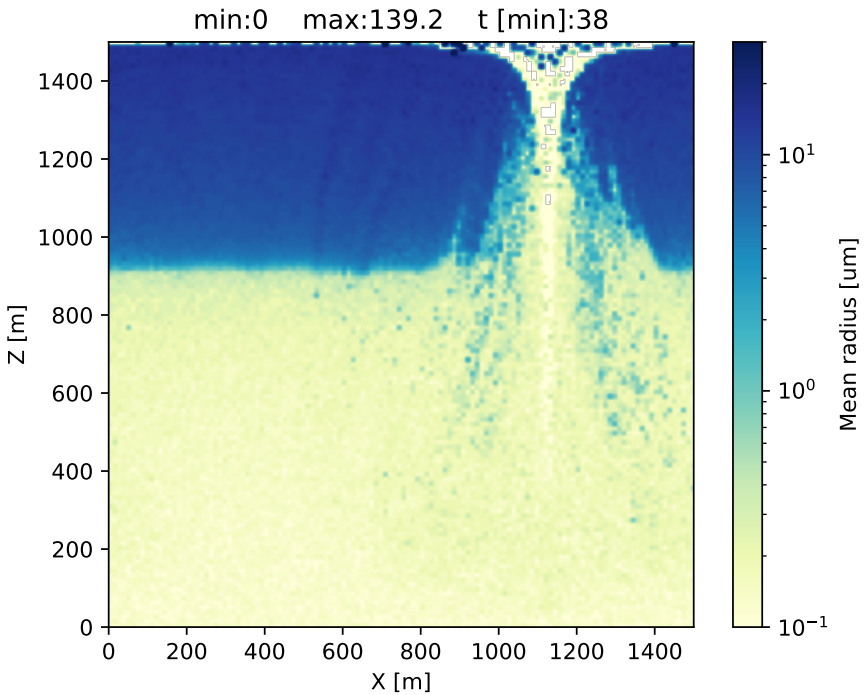




Context: aerosol-cloud-precipitation interactions

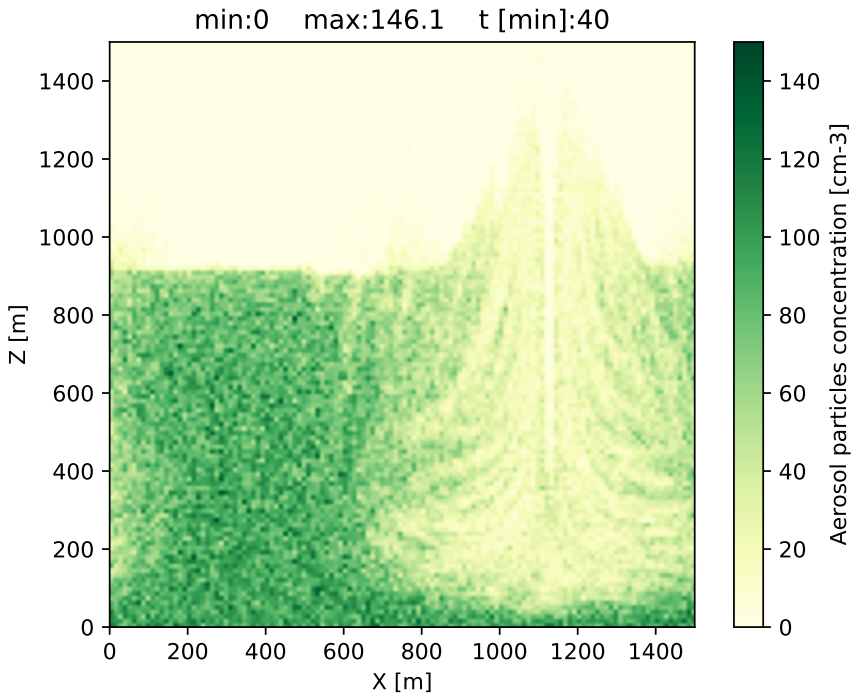
Computational grid: 128x128
Computational particles: 2^{21}

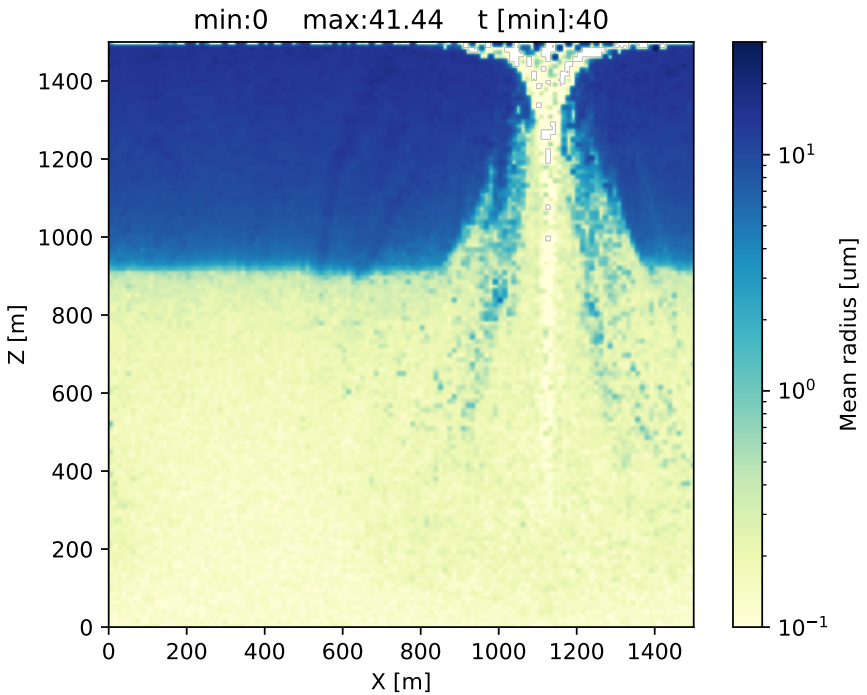




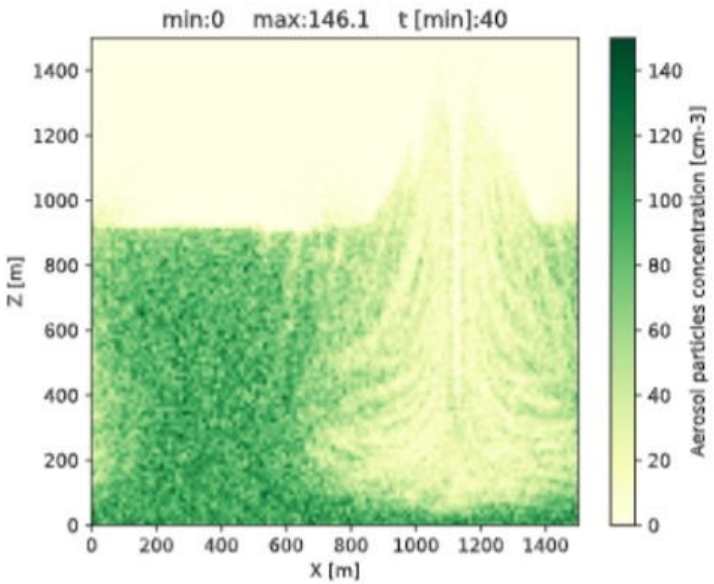
Context: aerosol-cloud-precipitation interactions

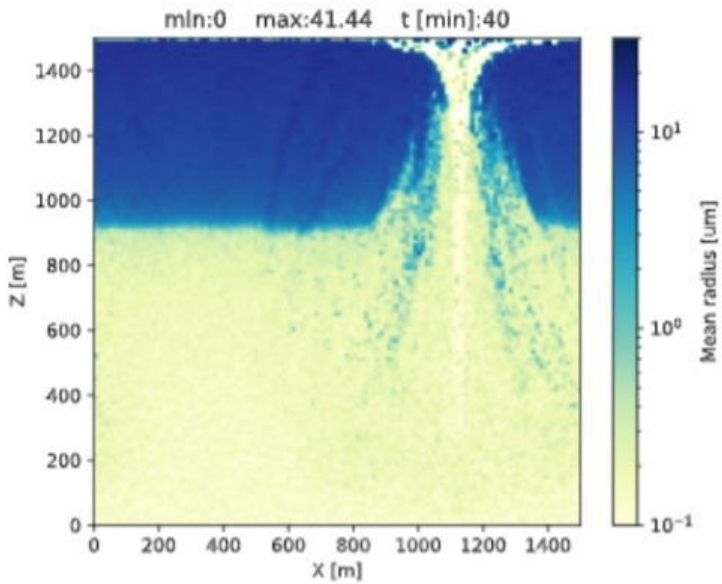
Computational grid: 128x128
Computational particles: 2^{21}





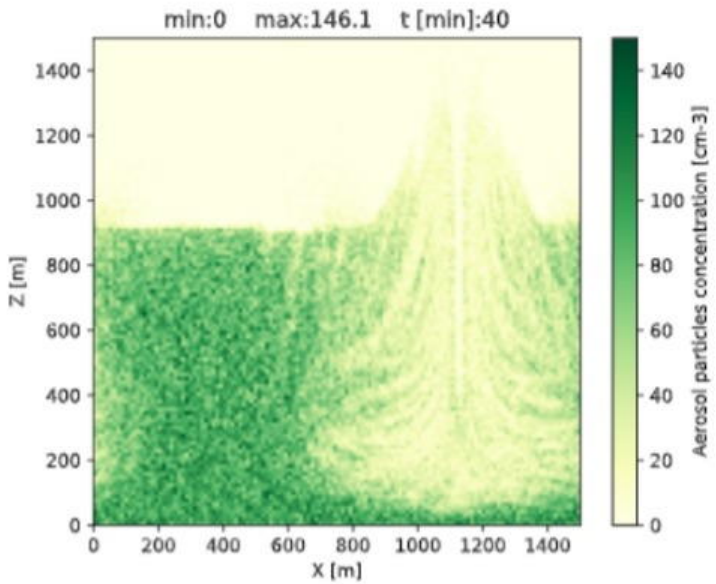
Needs:

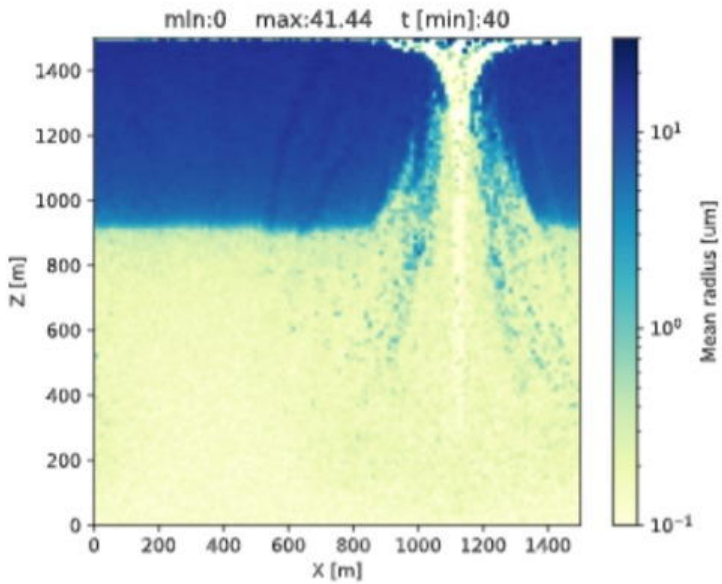

min:0 max:146.1 t [min]:40
Z [m]
X [m]
Aerosol particles concentration [cm-3]


min:0 max:41.44 t [min]:40
Z [m]
X [m]
Mean radius [um]

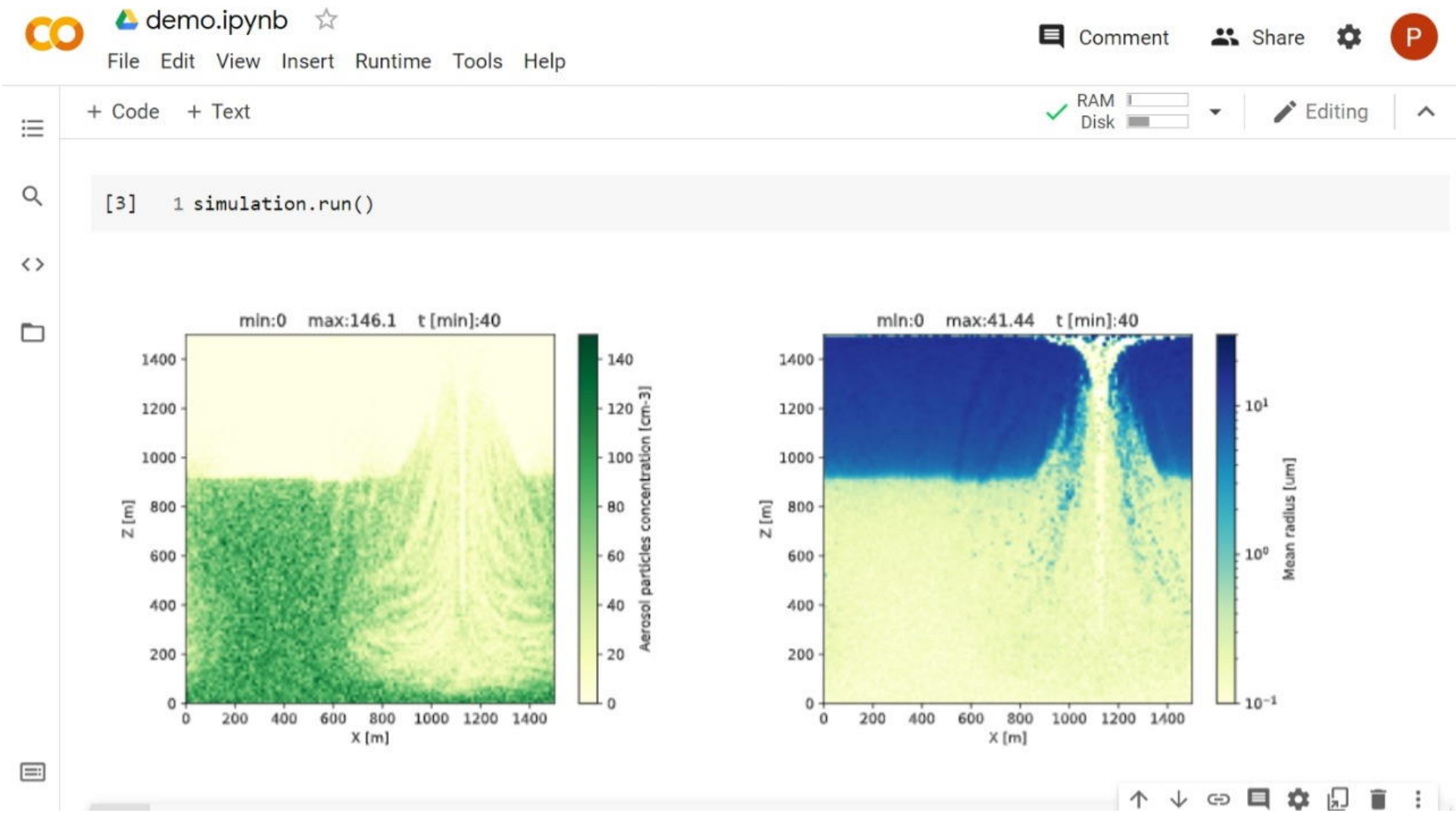
Needs: Pythonic

```
[3]   1 simulation.run()
```

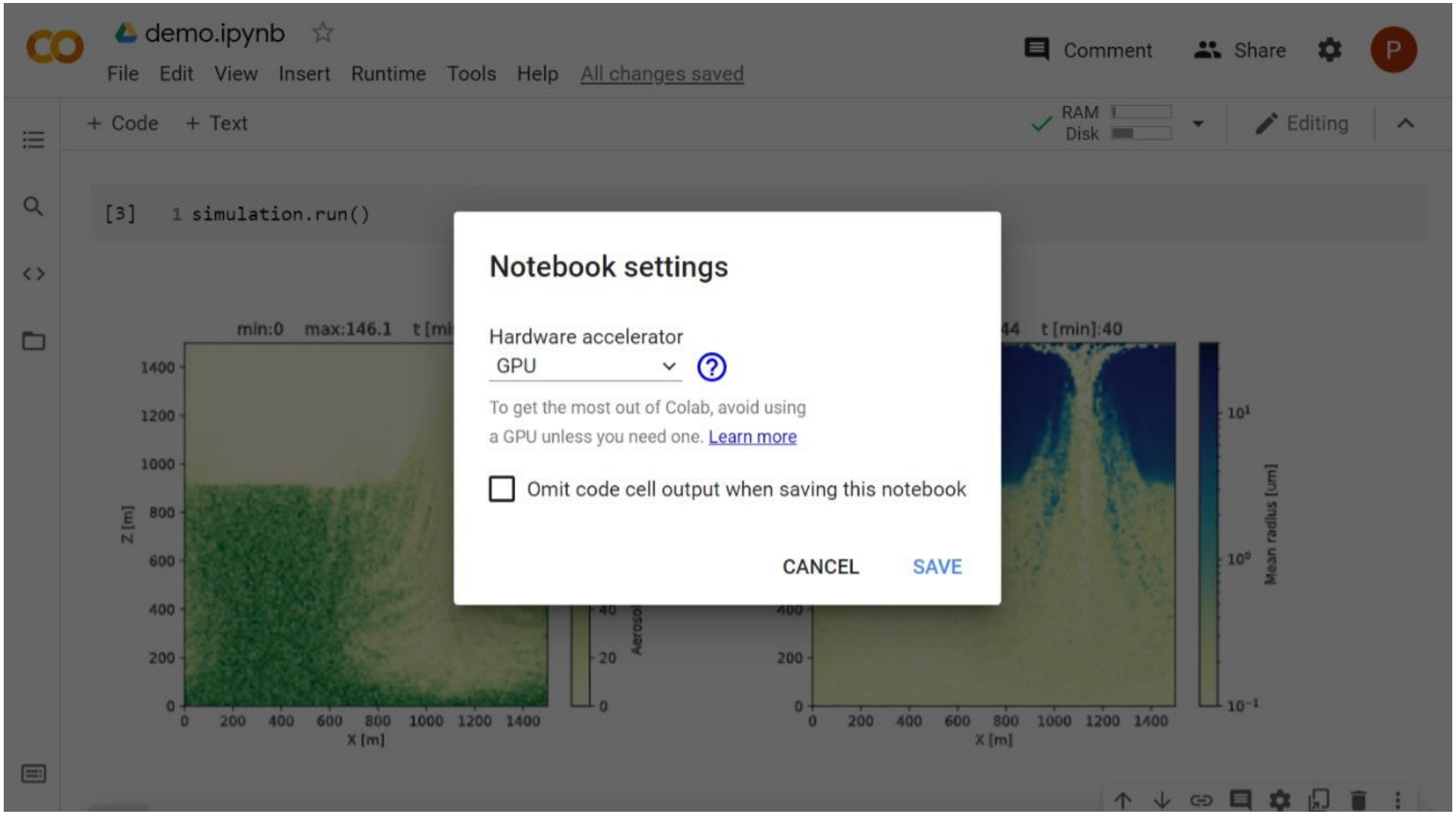




Needs: Pythonic, Colab friendly



Needs: Pythonic, Colab friendly, GPU-enabled



SDM & PySDM

Cloud droplet collisional growth

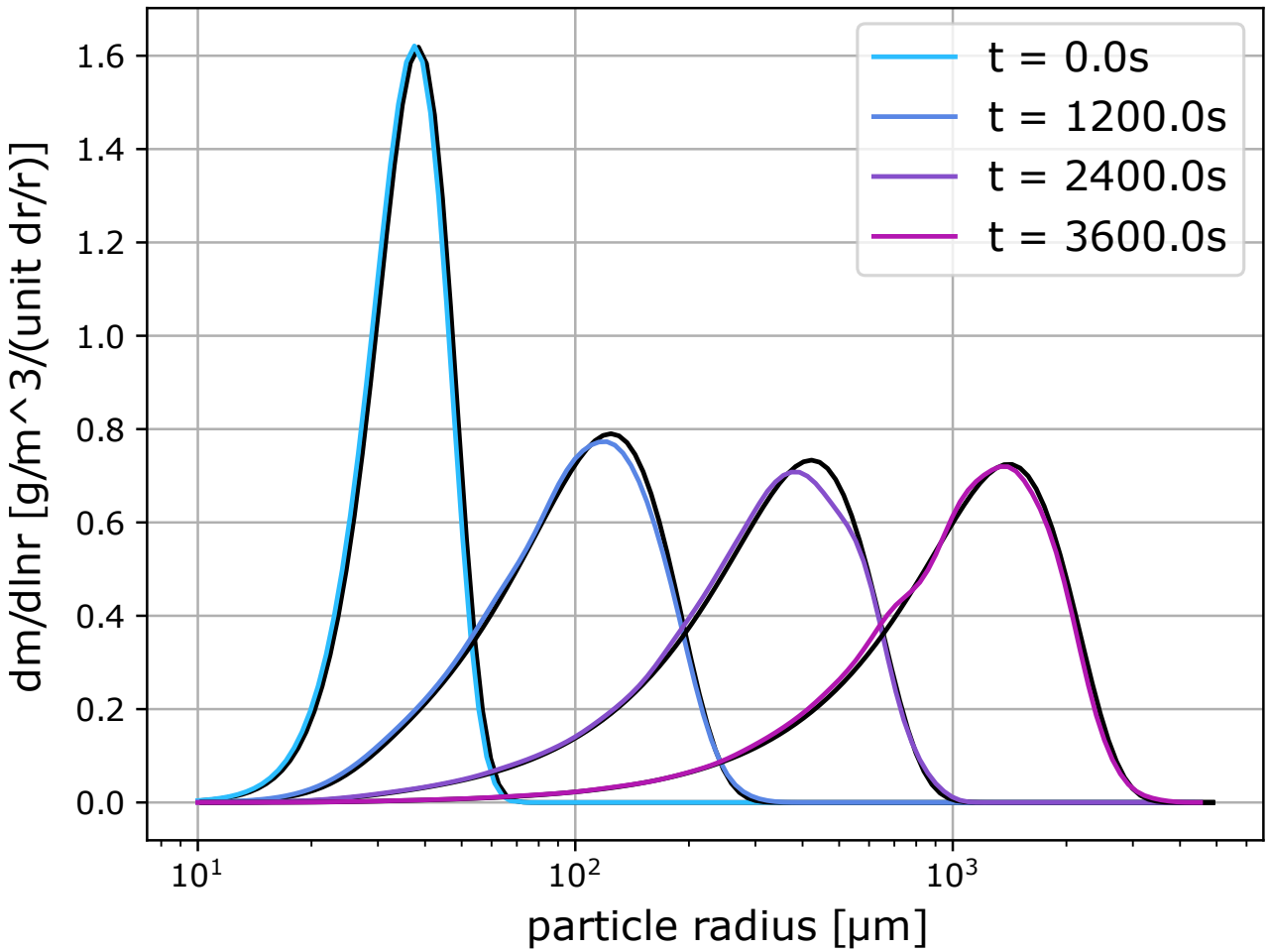
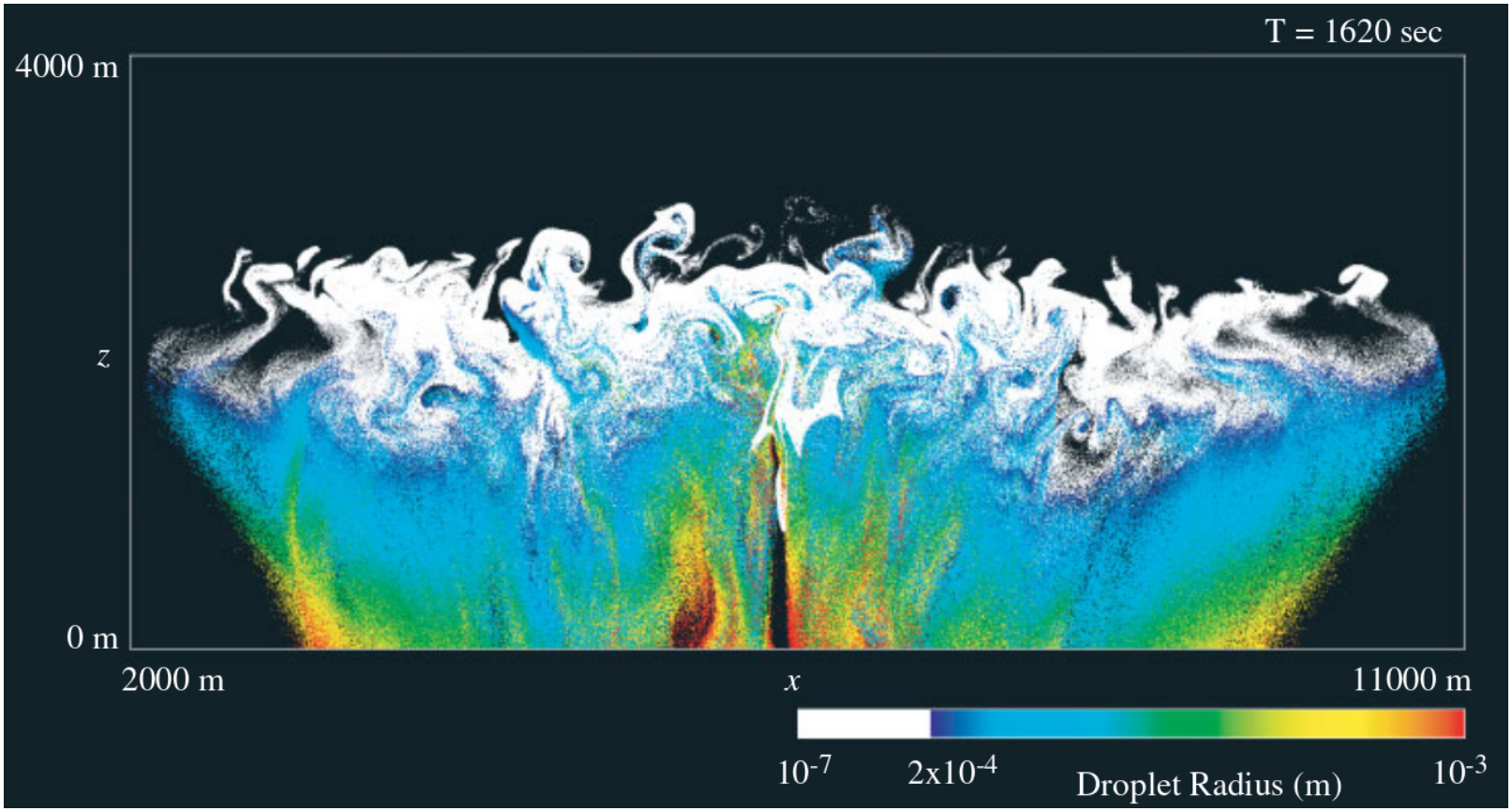


Fig. 2 from Shima et al. 2009, reproduced by PySDM.

Super-Droplet Method (SDM)



Super-droplet simulation of a shallow convective cloud
(figure: Shima et al. 2009, QJRMS)

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

README.md

build passing coverage 61%

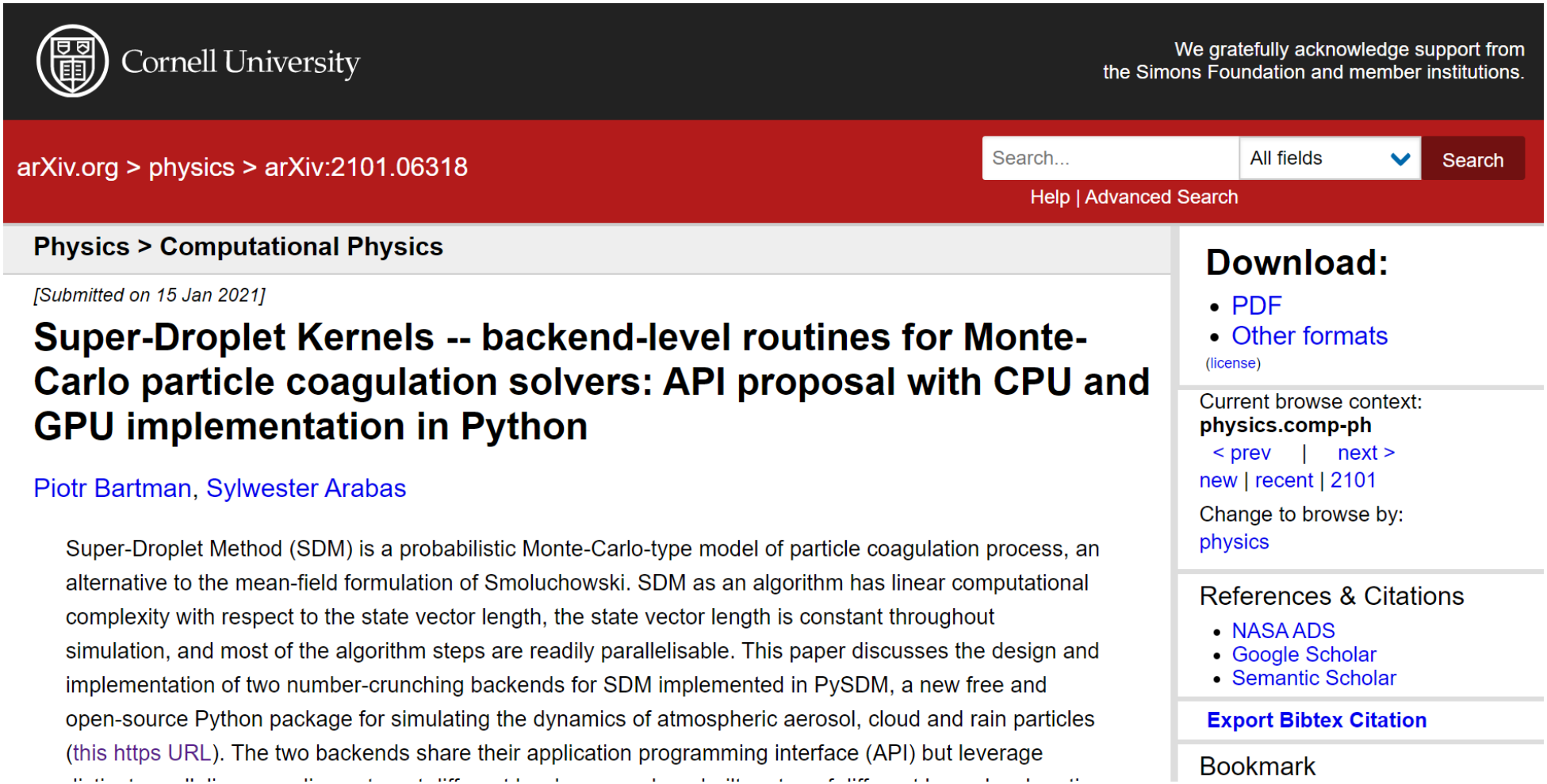
PySDM

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)

Super Droplet Kernels: CPU and GPU backends



Cornell University

We gratefully acknowledge support from the Simons Foundation and member institutions.

arXiv.org > physics > arXiv:2101.06318

Search... All fields Search

Help | Advanced Search

Physics > Computational Physics

[Submitted on 15 Jan 2021]

Super-Droplet Kernels -- backend-level routines for Monte-Carlo particle coagulation solvers: API proposal with CPU and GPU implementation in Python

Piotr Bartman, Sylwester Arabas

Super-Droplet Method (SDM) is a probabilistic Monte-Carlo-type model of particle coagulation process, an alternative to the mean-field formulation of Smoluchowski. SDM as an algorithm has linear computational complexity with respect to the state vector length, the state vector length is constant throughout simulation, and most of the algorithm steps are readily parallelisable. This paper discusses the design and implementation of two number-crunching backends for SDM implemented in PySDM, a new free and open-source Python package for simulating the dynamics of atmospheric aerosol, cloud and rain particles (this https URL). The two backends share their application programming interface (API) but leverage

Download:

- PDF
- Other formats

(license)

Current browse context: **physics.comp-ph**

< prev | next >

new | recent | 2101

Change to browse by: physics

References & Citations

- NASA ADS
- Google Scholar
- Semantic Scholar

Export Bibtex Citation

Bookmark

PySDM backends

```
151 def coalescence(n, idx, length, attributes, gamma, healthy):
152     n_sd = ThrustRTC.DVInt64(attributes.shape[1])
153     n_attr = ThrustRTC.DVInt64(attributes.shape[0])
154     kernel = ThrustRTC.For(['n', 'idx', 'n_sd', 'attributes', 'n_attr', 'gamma', 'healthy'], "i", '''
155             if (gamma[i] == 0) return;
156             auto j = idx[2 * i];
157             auto k = idx[2 * i + 1];
158             auto new_n = n[j] - gamma[i] * n[k];
159             if (new_n > 0) {
160                 n[j] = new_n;
161                 for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd)
162                     attributes[attr + k] += gamma[i] * attributes[attr + j];
163             }
164             else {  // new_n == 0
165                 n[j] = (int64_t)(n[k] / 2);
166                 n[k] = n[k] - n[j];
167                 for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd) {
168                     attributes[attr + j] = gamma[i] * attributes[attr + j] + attributes[attr + k];
169                     attributes[attr + k] = attributes[attr + j];
170                 }
171             }
172             ''')
173     kernel.launch_n(length // 2, [n.data, idx.data, n_sd, attributes.data, n_attr, gamma.data, healthy.data])
```

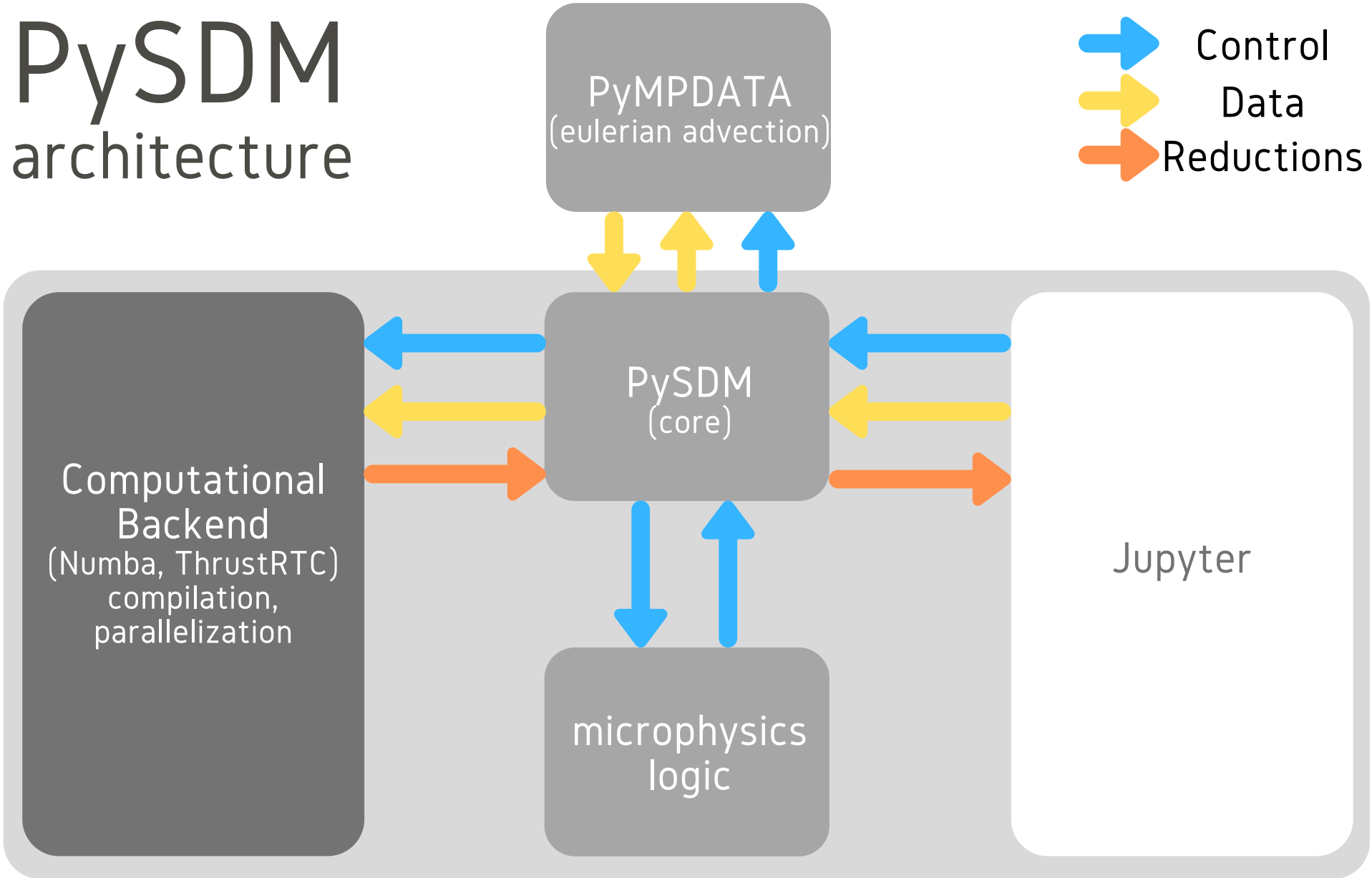
```
81  @numba.njit()
82  def coalescence_body(n, idx, length, attributes, gamma, is_first_in_pair):
83      for i in prange(length // 2):
84          if gamma[i] == 0:
85              continue
86          j, k = pair_indices(i, idx, is_first_in_pair)
87
88          new_n = n[j] - gamma[i] * n[k]
89          if new_n > 0:
90              n[j] = new_n
91              for a in range(0, len(attributes)):
92                  attributes[a, k] += gamma[i] * attributes[a, j]
93          else:  # new_n == 0
94              n[j] = n[k] // 2
95              n[k] = n[k] - n[j]
96              for a in range(0, len(attributes)):
97                  attributes[a, j] = gamma[i] * attributes[a, j] + attributes[a, k]
98                  attributes[a, k] = attributes[a, j]
```

- *run-time-compilation*
- GPU parallelization
- ???

- *just-in-time* compilation
- CPU parallelization
- ∼100x faster than pure Python

PySDM: workflow

PySDM architecture

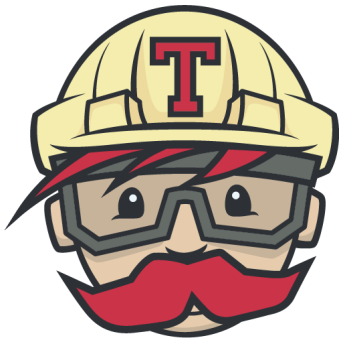


PySDM: technological stack and workflows

- Python `python.org`
- Numba `numba.pydata.org`
- ThrustRTC `pypi.org/project/ThrustRTC`

PySDM: technological stack and workflows

- Python python.org
- Numba numba.pydata.org
- ThrustRTC pypi.org/project/ThrustRTC
- GitHub & GitHub Actions github.com
- TravisCI travis-ci.org
- AppVeyor appveyor.com



PySDM: technological stack and workflows

- Python python.org
- Numba numba.pydata.org
- ThrustRTC pypi.org/project/ThrustRTC

- GitHub & GitHub Actions github.com
- TravisCI travis-ci.org
- AppVeyor appveyor.com

- Jupyter jupyter.org
- Binder mybinder.org
- Colab colab.research.google.com

ThrustRTC & CURandRTC

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...
- open-source, included in the NVIDIA HPC SDK & CUDA Toolkit

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...
- open-source, included in the NVIDIA HPC SDK & CUDA Toolkit
- high-level interface (STL-like, CUDA-boilerplate-free)

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...
- open-source, included in the NVIDIA HPC SDK & CUDA Toolkit
- high-level interface (STL-like, CUDA-boilerplate-free)
- multiple backends: CUDA, TBB and OpenMP

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...
- open-source, included in the NVIDIA HPC SDK & CUDA Toolkit
- high-level interface (STL-like, CUDA-boilerplate-free)
- multiple backends: CUDA, TBB and OpenMP
- enabling code portability between GPUs and multicore CPUs (preprocessor define to switch between CUDA and CPU threads)

Background: C++ Thrust (thrust.github.io)

- C++ parallel programming library from NVIDIA mimicking std::algorithm of the C++ STL
- host/device vectors, iterators, reductions, prefix-sums, sorting, ...
- open-source, included in the NVIDIA HPC SDK & CUDA Toolkit
- high-level interface (STL-like, CUDA-boilerplate-free)
- multiple backends: CUDA, TBB and OpenMP
- enabling code portability between GPUs and multicore CPUs (preprocessor define to switch between CUDA and CPU threads)
- e.g., optional dependency of Boost.Numeric.Odeint

ThrustRTC: Python, C#, Java (github.com/fynv)

- library of general GPU algorithms, similar to Thrust, can be used in C-interop languages (Python, C#, Java)

ThrustRTC: Python, C#, Java (github.com/fynv)

- library of general GPU algorithms, similar to Thrust, can be used in C-interop languages (Python, C#, Java)
- uses *NVRTC + dynamic-instantiation*, instead of *CUDA runtime + static compilation + templates (Thrust)*

ThrustRTC: Python, C#, Java (github.com/fynv)

- library of general GPU algorithms, similar to Thrust, can be used in C-interop languages (Python, C#, Java)
- uses *NVRTC + dynamic-instantiation*, instead of *CUDA runtime + static compilation + templates (Thrust)*
- with Python: compilation, linking, dll-loading hidden from the user; automagical data typing; easy "pip" installation

ThrustRTC: Python, C#, Java (github.com/fynv)

- library of general GPU algorithms, similar to Thrust,
  can be used in C-interop languages (Python, C#, Java)
- uses *NVRTC + dynamic-instantiation*, instead of
  CUDA runtime + static compilation + templates (Thrust)
- with Python: compilation, linking, dll-loading hidden from the user;
  automagical data typing; easy "pip" installation
- flexibility wrt level of indirection:
  – built-in algorithms (e.g., arithmetic, reductions, sorting, ...) or
  – CUDA-free loop bodies (C++ code in strings, automatic thread control) or
  – custom CUDA kernels (C++ code in strings, explicit thread control)

ThrustRTC: Python, C#, Java (github.com/fynv)

- library of general GPU algorithms, similar to Thrust,
  can be used in C-interop languages (Python, C#, Java)
- uses *NVRTC + dynamic-instantiation*, instead of
  CUDA runtime + static compilation + templates (Thrust)
- with Python: compilation, linking, dll-loading hidden from the user;
  automagical data typing; easy "pip" installation
- flexibility wrt level of indirection:
  – built-in algorithms (e.g., arithmetic, reductions, sorting, ...) or
  – CUDA-free loop bodies (C++ code in strings, automatic thread control) or
  – custom CUDA kernels (C++ code in strings, explicit thread control)
- CUDA only

ThrustRTC: built-in algorithms

```
50  # sorting
51  dvalues = trtc.device_vector_from_list([ 1, 4, 2, 8, 5, 7 ], 'int32_t')
52  trtc.Sort(dvalues)
53
```

ThrustRTC: built-in algorithms

```
50  # sorting
51  dvalues = trtc.device_vector_from_list([ 1, 4, 2, 8, 5, 7 ], 'int32_t')
52  trtc.Sort(dvalues)
53
54  # reduction
55  vec = trtc.device_vector_from_list([ 0, 1, 0, 1, 1], 'int32_t')
56  result = trtc.Count(vec, trtc.DVInt32(1))
57
```

ThrustRTC: built-in algorithms

```
50  # sorting
51  dvalues = trtc.device_vector_from_list([ 1, 4, 2, 8, 5, 7 ], 'int32_t')
52  trtc.Sort(dvalues)
53
54  # reduction
55  vec = trtc.device_vector_from_list([ 0, 1, 0, 1, 1], 'int32_t')
56  result = trtc.Count(vec, trtc.DVInt32(1))
57
58  # transformations
59  X = trtc.device_vector('int32_t', 10)
60  Y = trtc.device_vector('int32_t', 10)
61  Z = trtc.device_vector('int32_t', 10)
62  # initialize X to 0,1,2,3, ....
63  trtc.Sequence(X)
64  # compute Y = -X
65  trtc.Transform(X, Y, trtc.Negate())
66  # fill Z with twos
67  trtc.Fill(Z, trtc.DVInt32(2))
68  # compute Y = X mod 2
69  trtc.Transform_Binary(X, Z, Y, trtc.Modulus())
70  # replace all the ones in Y with tens
71  trtc.Replace(Y, trtc.DVInt32(1), trtc.DVInt32(10))
72  # print Y
73  print(Y.to_host())
```

ThrustRTC: *generic* custom algorithms

```
10  kernel = ThrustRTC.Kernel(['arr_in', 'arr_out', 'k'], '''
11      auto i = blockIdx.x * blockDim.x + threadIdx.x;
12      if (i >= arr_in.size())
13          return;
14      arr_out[i] = arr_in[i] * k;
15      ''')
16  kernel.launch(gridDim=1, blockDim=128, args=[d_vec_in, d_vec_out, d_k])
17
18  forLoop = ThrustRTC.For(['arr_in', 'arr_out', 'k'], "i", '''
19      arr_out[i] = arr_in[i] * k;
20      ''')
21  forLoop.launch_n(n=5, args=[d_vec_in, d_vec_out, d_k])
```

ThrustRTC: *generic* custom algorithms

```
10  kernel = ThrustRTC.Kernel(['arr_in', 'arr_out', 'k'], '''
11      auto i = blockIdx.x * blockDim.x + threadIdx.x;
12      if (i >= arr_in.size())
13          return;
14      arr_out[i] = arr_in[i] * k;
15      ''')
16  kernel.launch(gridDim=1, blockDim=128, args=[d_vec_in, d_vec_out, d_k])
17
18  forLoop = ThrustRTC.For(['arr_in', 'arr_out', 'k'], "i", '''
19      arr_out[i] = arr_in[i] * k;
20      ''')
21  forLoop.launch_n(n=5, args=[d_vec_in, d_vec_out, d_k])
```

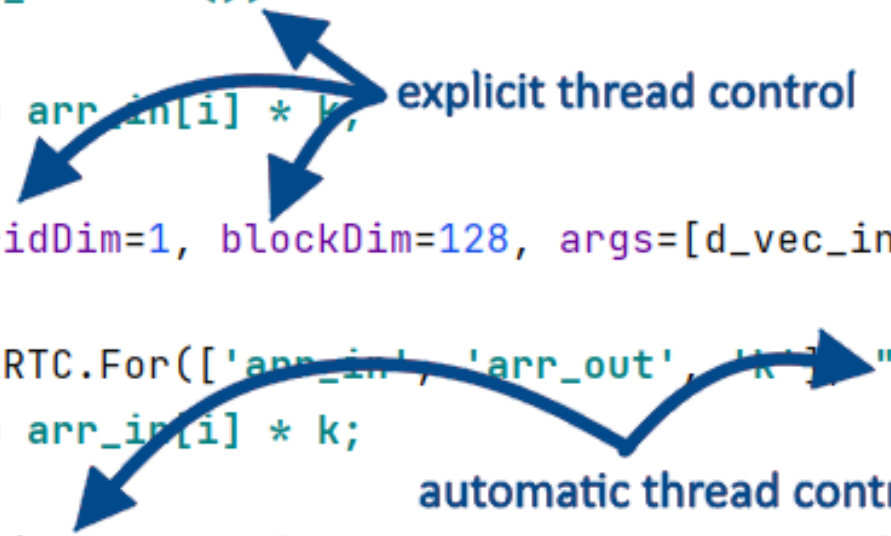
ThrustRTC: *generic* custom algorithms

```
10  kernel = ThrustRTC.Kernel(['arr_in', 'arr_out', 'k'], '''
11      auto i = blockIdx.x * blockDim.x + threadIdx.x;
12      if (i >= arr_in.size())
13          return;
14      arr_out[i] = arr_in[i] * k;
15      ''')
16  kernel.launch(gridDim=1, blockDim=128, args=[d_vec_in, d_vec_out, d_k])
17
18  forLoop = ThrustRTC.For(['arr_in', 'arr_out', 'k'], "i", '''
19      arr_out[i] = arr_in[i] * k;
20      ''')
21  forLoop.launch_n(n=5, args=[d_vec_in, d_vec_out, d_k])
```

C++11 constructs

ThrustRTC: *generic* custom algorithms

```
10  kernel = ThrustRTC.Kernel(['arr_in', 'arr_out', 'k'], '''
11      auto i = blockIdx.x * blockDim.x + threadIdx.x;
12      if (i >= arr_in.size())
13          return;
14      arr_out[i] = arr_in[i] * k;
15      ''')
16  kernel.launch(gridDim=1, blockDim=128, args=[d_vec_in, d_vec_out, d_k])
17
18  forLoop = ThrustRTC.For([...], "i", '''
19      arr_out[i] = arr_in[i] * k;
20      ''')
21  forLoop.launch_n(n=5, args=[d_vec_in, d_vec_out, d_k])
```



ThrustRTC.For

```
151     def coalescence(n, idx, length, attributes, gamma, healthy):
152         n_sd = ThrustRTC.DVInt64(attributes.shape[1])
153         n_attr = ThrustRTC.DVInt64(attributes.shape[0])
154         kernel = ThrustRTC.For(['n', 'idx', 'n_sd', 'attributes', 'n_attr', 'gamma', 'healthy'], "i", '''
155                 if (gamma[i] == 0) return;
156                 auto j = idx[2 * i];
157                 auto k = idx[2 * i + 1];
158                 auto new_n = n[j] - gamma[i] * n[k];
159                 if (new_n > 0) {
160                     n[j] = new_n;
161                     for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd)
162                         attributes[attr + k] += gamma[i] * attributes[attr + j];
163                 }
164                 else {  // new_n == 0
165                     n[j] = (int64_t)(n[k] / 2);
166                     n[k] = n[k] - n[j];
167                     for (auto attr = 0; attr < n_attr * n_sd; attr+=n_sd) {
168                         attributes[attr + j] = gamma[i] * attributes[attr + j] + attributes[attr + k];
169                         attributes[attr + k] = attributes[attr + j];
170                     }
171                 }
172                 ''')
173         kernel.launch_n(length // 2, [n.data, idx.data, n_sd, attributes.data, n_attr, gamma.data, healthy.data])
```

CURandRTC (github.com/fynv/CURandRTC)

- simple random number generator using the XORWOW algorithm

```
5   class Random:
6       def __init__(self, size, seed):
7           rng = CURandRTC.DVRNG()
8           self.generator = ThrustRTC.device_vector('RNGState', size)
9           self.size = size
10          device_seed = ThrustRTC.DVInt64(seed)
11          kernel = ThrustRTC.For(['rng', 'states', 'seed'], 'i', '''
12                  rng.state_init(seed, i, 0, states[i]);
13                  ''')
14          kernel.launch_n(size, [rng, self.generator, device_seed])
15
16          self.rand_kernel = ThrustRTC.For(['states', 'vec_rnd'], 'i', '''
17                  vec_rnd[i] = states[i].rand01();
18                  ''')
19
20      def __call__(self, storage):
21          self.rand_kernel.launch_n(len(storage), [self.generator, storage.data])
```

ThrustRTC: summary

Highlights:

- GPU + portability
- Low entry threshold for developers (high level parallel API: both built-in and custom kernels)

Challenges:

- custom kernels
- profiling
- maintainability + CI (Fake Thrust, test coverage)

Demo (role play: reviewer)

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

README.md

build passing coverage 61%

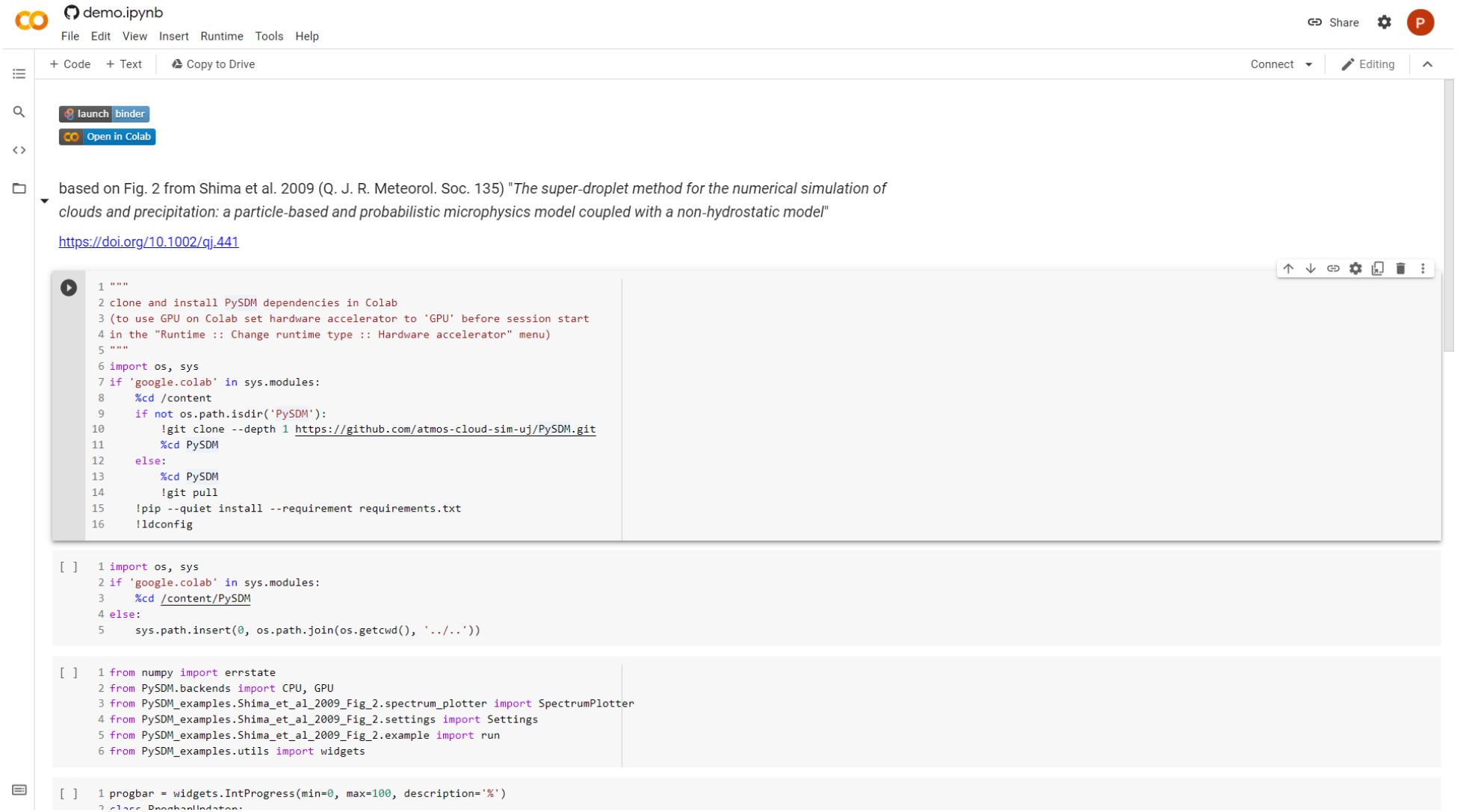
PySDM

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

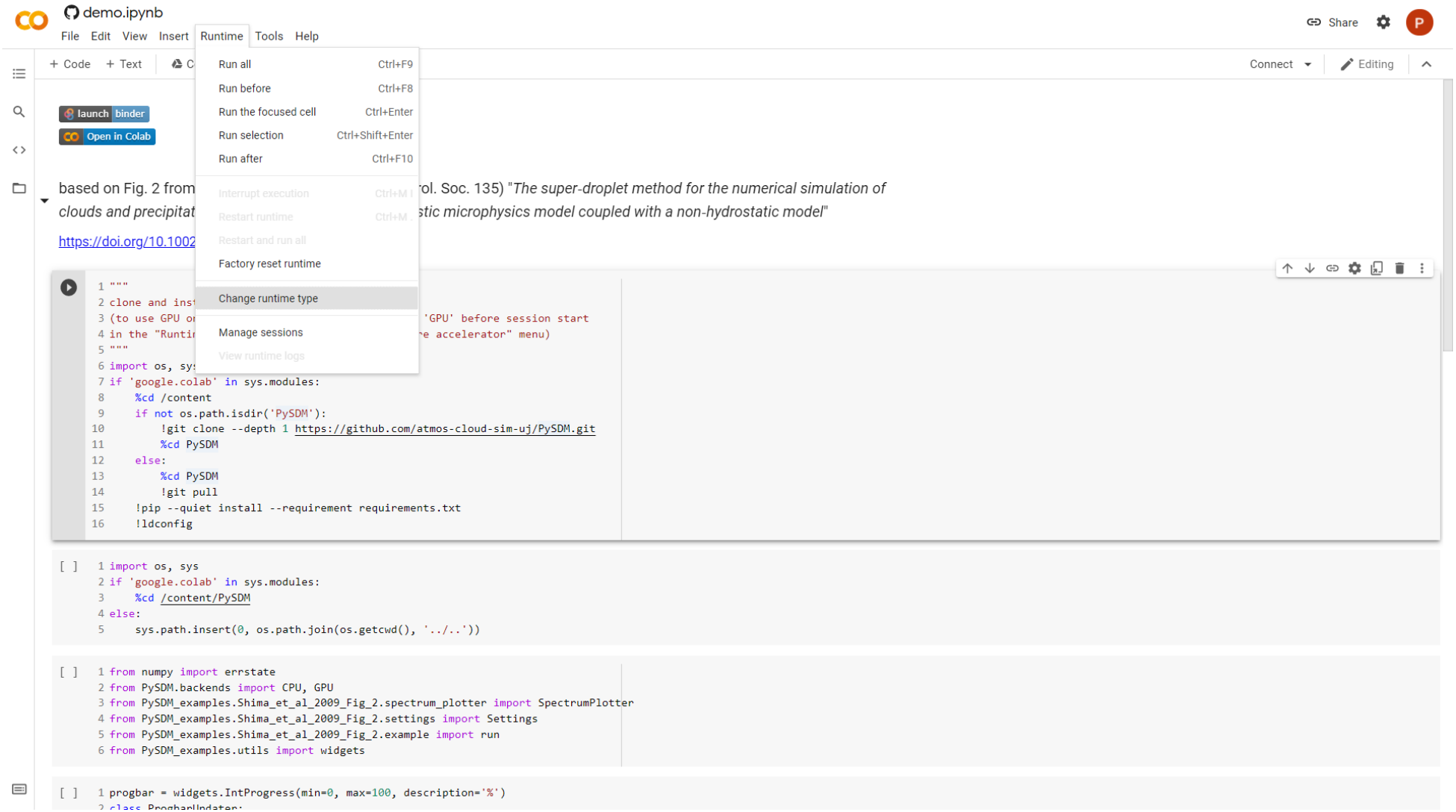
Demos:

- 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)

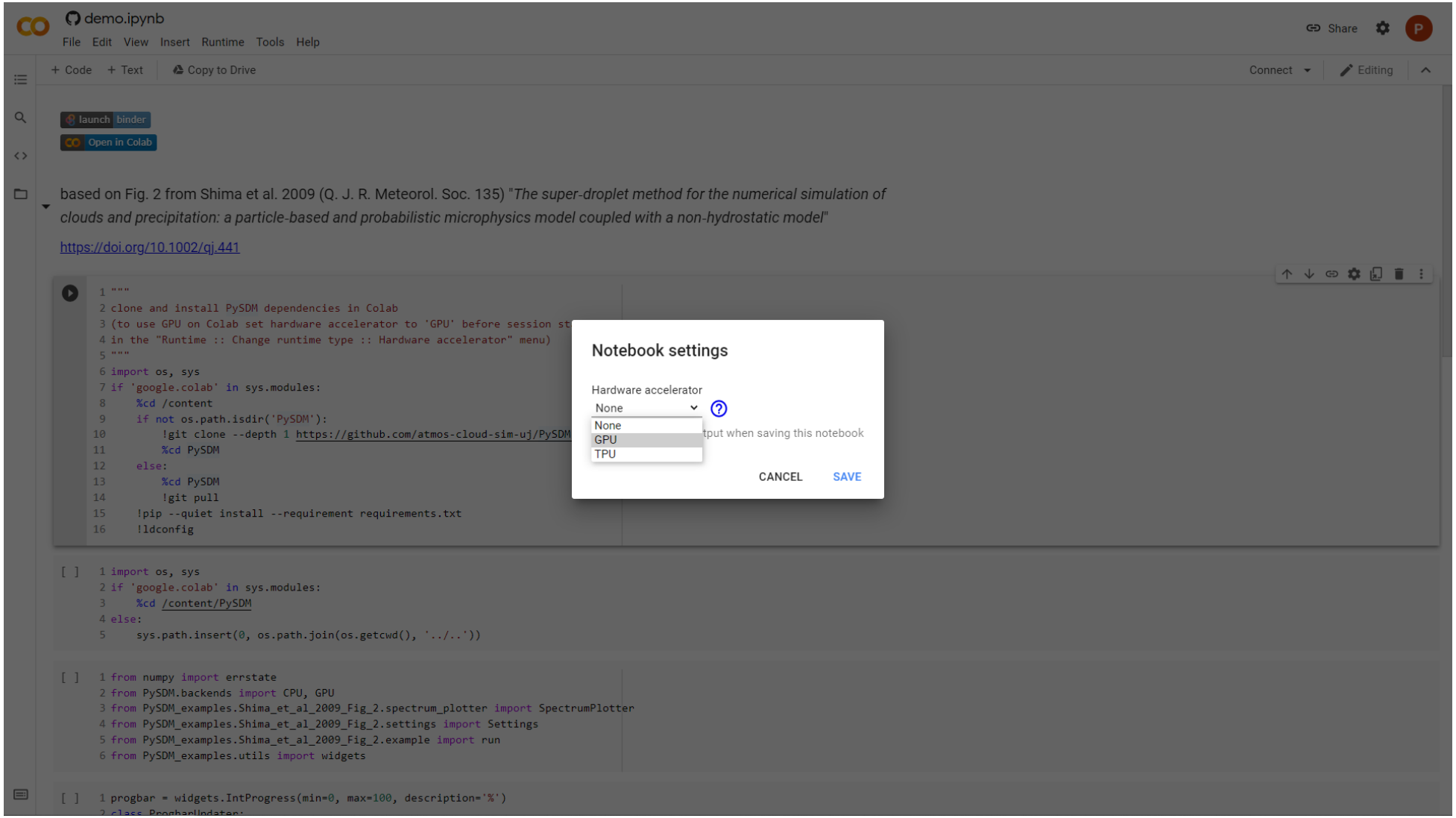
Demo (github.com/atmos-cloud-sim-uj/PySDM)



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



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



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

demo.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

RAM Disk Editing

launch binder

Open in Colab

based on Fig. 2 from Shima et al. 2009 (Q. J. R. Meteorol. Soc. 135) "*The super-droplet method for the numerical simulation of clouds and precipitation: a particle-based and probabilistic microphysics model coupled with a non-hydrostatic model*"

https://doi.org/10.1002/qj.441

```
"""
clone and install PySDM dependencies in Colab
(to use GPU on Colab set hardware accelerator to 'GPU' before session start
in the "Runtime :: Change runtime type :: Hardware accelerator" menu)
"""
import os, sys
if 'google.colab' in sys.modules:
    %cd /content
    if not os.path.isdir('PySDM'):
        !git clone --depth 1 https://github.com/atmos-cloud-sim-uj/PySDM.git
        %cd PySDM
    else:
        %cd PySDM
        !git pull
    !pip --quiet install --requirement requirements.txt
    !ldconfig
```

```
/content
Cloning into 'PySDM'...
remote: Enumerating objects: 328, done.
remote: Counting objects: 100% (328/328), done.
remote: Compressing objects: 100% (295/295), done.
remote: Total 328 (delta 18), reused 108 (delta 1), pack-reused 0
Receiving objects: 100% (328/328), 600.87 KiB | 2.30 MiB/s, done.
Resolving deltas: 100% (18/18), done.
/content/PySDM
     |████████████████████████████████| 768kB 5.8MB/s
     |████████████████████████████████| 81kB 8.3MB/s
     |████████████████████████████████| 3.4MB 37.7MB/s
     |████████████████                | 7.7MB 17.8MB/s eta 0:00:01
```

```
import os, sys
if 'google.colab' in sys.modules:
    %cd /content/PySDM
```

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

demo.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

Share

+ Code + Text Copy to Drive

RAM Disk

Editing

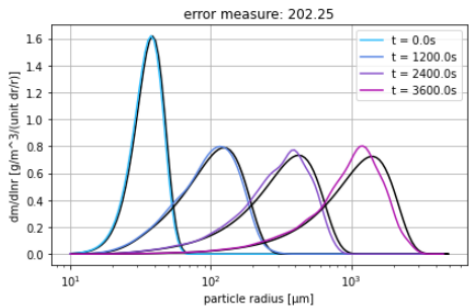
```
[5] 17        plotter.show()
```

```
1 n_SD = widgets.IntSlider(value=14, min=12, max=18, step=1, description='$log_2(n_{SD})$', continuous_update=False)
2 n_step = widgets.IntSlider(value=3600, step=100, min=0, max=3600, description='$n_{step}$', continuous_update=False)
3 n_plot = widgets.IntSlider(value=3, step=1, min=1, max=8, description='$n_{plot}$', continuous_update=False)
4 sliders = widgets.HBox([n_SD, n_step, n_plot])
5
6 adaptive = widgets.Checkbox(value=False, description='adaptive dt')
7 smooth = widgets.Checkbox(value=True, description='smooth plot')
8 gpu = widgets.Checkbox(value=False, description='GPU')
9 options = [adaptive, smooth]
10 if GPU.ENABLE:
11     options.append(gpu)
12 boxes = widgets.HBox(options)
13 # TODO #410: freezer
14 self = widgets.interactive_output(demo,
15                                   {'n_SD': n_SD, 'n_step': n_step, 'n_plot': n_plot, 'adaptive': adaptive, 'smooth': smooth, 'gpu': gpu})
16
17 display(sliders, boxes, progbar, self)
```

$log_2(n_{... 14 n_{step} 3600 n_{plot} 3

adaptive dt smooth plot GPU

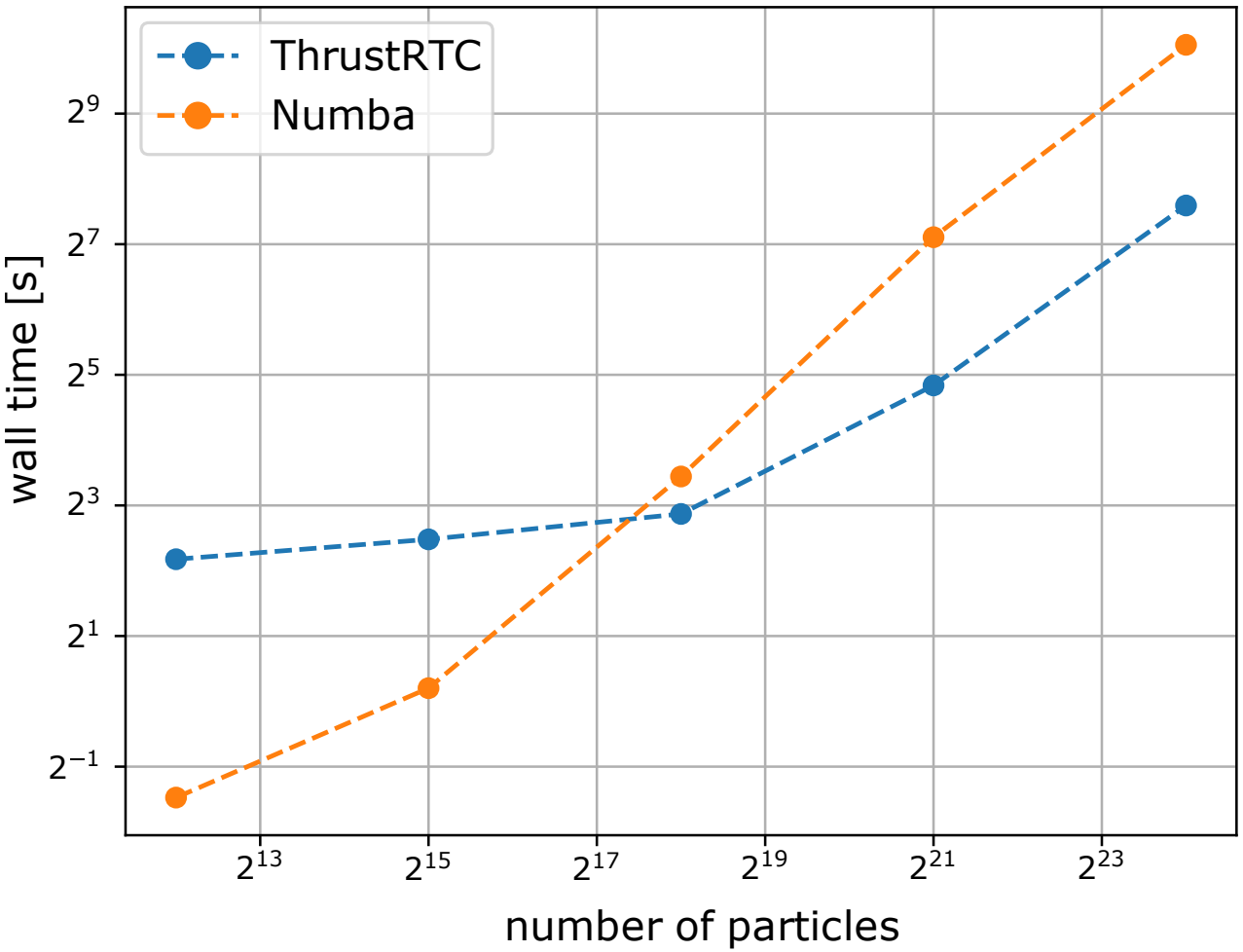
%



tmpctwqic6u.pdf

```
[ ] 1
```

Backends performance

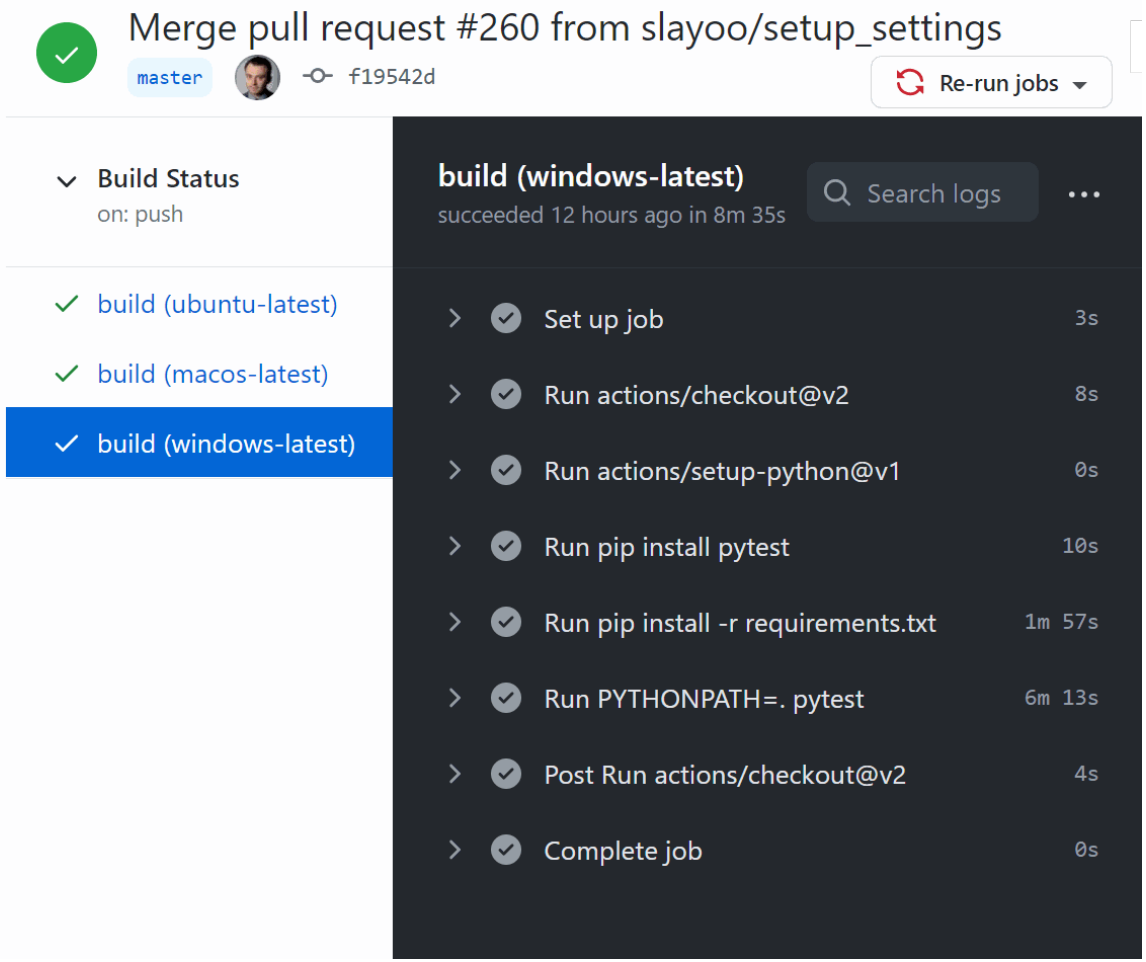


Comparison Numba and ThrustRTC backends

Portability and Continuous Integration

33 lines (26 sloc) | 553 Bytes — Raw | Blame

```yaml
name: Build Status

defaults:
  run:
    shell: bash

on:
  push:
    branches: [ master ]
  pull_request:
    branches: [ master ]

jobs:
  build:
    strategy:
      matrix:
        platform: [ubuntu-latest, macos-latest, windows-latest]
    runs-on: ${{ matrix.platform }}

    steps:
      - uses: actions/checkout@v2

      - uses: actions/setup-python@v1
        with:
          python-version: 3.8
      - run: |
          pip install pytest

      - run: |
          pip install -r requirements.txt

      - run: |
          PYTHONPATH=. pytest
```



Take-home messages

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user
  - seamless pip installation on Linux & Windows

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user
  - seamless pip installation on Linux & Windows
- **without trading off flexibility of building less-Pythonic custom algorithms:**

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user
  - seamless pip installation on Linux & Windows
- **without trading off flexibility of building less-Pythonic custom algorithms:**
  - C++ code in Python strings (incl. auto, decltype, ...)

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user
  - seamless pip installation on Linux & Windows
- **without trading off flexibility of building less-Pythonic custom algorithms:**
  - C++ code in Python strings (incl. auto, decltype, ...)
  - either explicit (ThrustRTC.Kernel) or implicit CUDA-free thread management (ThrustRTC.For)

Take-home messages: key advantages of ThurstRTC

- **portable Pythonic GPU solution with low entry threshold for (research) developers and users:**
  - built-in parallel algorithm library (similar to std::algorithm, Thrust, ...)
  - NVRTC-based: CUDA compilation, parallelisation logic, data typing entirely hidden from the user
  - seamless pip installation on Linux & Windows
- **without trading off flexibility of building less-Pythonic custom algorithms:**
  - C++ code in Python strings (incl. auto, decltype, ...)
  - either explicit (ThrustRTC.Kernel) or implicit CUDA-free thread management (ThrustRTC.For)
  - CUDA interoperable (e.g. for atomics, synchronisation)

Take-home messages: ThurstRTC in PySDM

- rapid development of GPU backend offering significant speedup

Take-home messages: ThurstRTC in PySDM

- rapid development of GPU backend offering significant speedup
- CUDA-free code with high-level parallelisation abstractions

Take-home messages: ThurstRTC in PySDM

- rapid development of GPU backend offering significant speedup
- CUDA-free code with high-level parallelisation abstractions
- simultaneous CPU+GPU usage capability to fully leverage hybrid platforms (async functionality in ThrustRTC)

Take-home messages: ThurstRTC in PySDM

- rapid development of GPU backend offering significant speedup
- CUDA-free code with high-level parallelisation abstractions
- simultaneous CPU+GPU usage capability to fully leverage hybrid platforms (async functionality in ThrustRTC)
- result: portable GPU-enabled tools runnable also on Colab (Google's web-based Jupyter platform)

Take-home messages: ThurstRTC in PySDM

- rapid development of GPU backend offering significant speedup
- CUDA-free code with high-level parallelisation abstractions
- simultaneous CPU+GPU usage capability to fully leverage hybrid platforms (async functionality in ThrustRTC)
- result: portable GPU-enabled tools runnable also on Colab (Google's web-based Jupyter platform)
- journal-review-level reproduciblity (goal: $<$1h time-to-reproduce)

Acknowledgments

Overview | Repositories 25 | Projects | Packages

RTC | Type: All | Language: All

2 results for repositories matching **RTC** | Clear filter

Fei Yang

fynv

Unfollow

...

Former software engineer at NVIDIA. Currently: 宙予科技, Beijing

38 followers · **41** following · ☆ **71**

hyangfeih@gmail.com

ThrustRTC

CUDA tool set for non-C++ languages that provides similar functionality like Thrust, with NVRTC at its core.

Unstar

cuda | nvrtc | thrust

C++ ☆ 29 3 Other Updated on 6 Feb

CURandRTC

CURandRTC is a GPU random number generation module based on ThrustRTC.

Unstar

cuda | random-number-generators | thrust | nvrtc

C++ Other Updated on 4 Jun 2020

Thank you for your attention!

more:
arxiv.org/abs/2101.06318
github.com/atmos-cloud-sim-uj/PySDM

piotr.bartman@doctoral.uj.edu.pl

funding acknowledgment:
Foundation for Polish Science / European Union