PyMPDATA: A Just-in-Time Compiled Implementation of MPDATA

Sylwester Arabas
AGH University of Krakow, Poland

June 26 2025 (M3ODEL Lunch Talk @uni-mainz.de)

AGH

ENVIRONMENTAL PHYSICS GROUP

WIKIPEDIA
Die freie Enzyklopädie

AGH Wissenschaftlich-Technische Universität

Die **AGH Wissenschaftlich-Technische Universität**[4] (polnisch: *Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie* = Akademie für Bergbau und Hüttenwesen „Stanisław Staszic“ zu Krakau) – kurz **AGH –** ist die größte technische Universität in der Stadt Krakau mit 23.570 Studenten und 2.154 wissenschaftlichen Mitarbeitern.[2] Sie hat den Ruf einer der besten Universitäten in Polen.[5]

Die AGH Wissenschaftlich-Technische Universität wurde 1919 gegründet und das Hauptgebäude wurde in den Jahren 1923–1935 in Czarna Wieś errichtet. Rektor der Universität ist seit Mai 2020 Jerzy Lis.[6][7]

AGH Wissenschaftlich-Technische Universität
Akademia Górniczo-Hutnicza im. Stanisława Staszica w Krakowie

AGH

Motto	„Labore creata, labori et scientiae servis“
Gründung	1919
Trägerschaft	staatlich
Ort	Krakau, Polen
Rektor	Jerzy Lis
Studierende	18,074[1] (12/2023)
Mitarbeiter	4,167[2] (31. Dezember 2017)
davon Professoren	157[2] (31. Dezember 2017)
Netzwerke	IAU[3]
Website	www.agh.edu.pl (https://www.agh.edu.pl/)

Fakultäten

Die AGH Universität gliedert sich in 16 Fakultäten[7]:

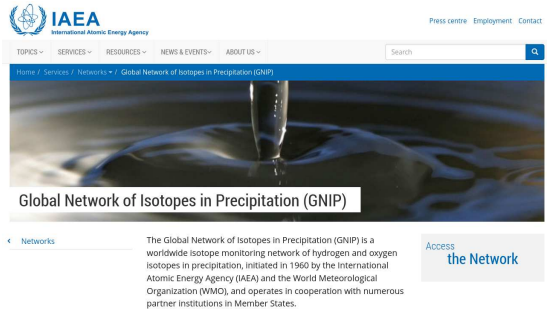
- Fakultät für Bauingenieurwesen und Ressourcenmanagement (*Wydział Inżynierii Lądowej i Gospodarki Zasobami*)
- Fakultät für Metalltechnik und Industrieinformatik (*Wydział Inżynierii Metali i Informatyki Przemysłowej*)
- Fakultät für Elektrotechnik, Automatik, Informatik und Biomedizintechnik (*Wydział Elektrotechniki, Automatyki, Informatyki i Inżynierii Biomedycznej*)
- Fakultät für Informatik, Elektronik und Telekommunikation (*Wydział Informatyki, Elektroniki i Telekomunikacji*)
- Fakultät für Maschinenbau und Robotik (*Wydział Inżynierii Mechanicznej i Robotyki*)
- Fakultät für Geologie, Geophysik und Umweltschutz (*Wydział Geologii, Geofizyki i Ochrony Środowiska*)
- Fakultät für montane Geodäsie und Umweltingenieuswesen (*Wydział Geodezji Górniczej i Inżynierii Środowiska*)
- Fakultät für Material- und Keramikwissenschaften (*Wydział Inżynierii Materiałowej i Ceramiki*)
- Fakultät für Gießereitechnik (*Wydział Odlewnictwa*)
- Fakultät für Nichteisenmetalle (*Wydział Metali Nieżelaznych*)
- Fakultät für Bohr-, Öl- und Gaswesen (*Wydział Wiertnictwa, Nafty i Gazu*)
- Fakultät für Management (*Wydział Zarządzania*)
- Fakultät für Energie und Kraftstoffe (*Wydział Energetyki i Paliw*)
- Fakultät für Physik und Angewandte Informatik (*Wydział Fizyki i Informatyki Stosowanej*)
- Fakultät für angewandte Mathematik (*Wydział Matematyki Stosowanej*)
- Fakultät für Geisteswissenschaften (*Wydział Humanistyczny*)

Hauptgebäude der AGH

AGH

ENVIRONMENTAL PHYSICS GROUP

- IAEA/GNIP site in Kraków



AGH

ENVIRONMENTAL
PHYSICS
GROUP

- IAEA/GNIP site in Kraków
- 50-year precip isotopic data record

AGH

ENVIRONMENTAL
PHYSICS
GROUP

- IAEA/GNIP site in Kraków
- 50-year precip isotopic data record
- high-altitude lab (clouds in-situ) @Kasprowy Wierch (6500 ft AMSL)

photo: naukaoklimacie.pl

AGH

ENVIRONMENTAL
PHYSICS
GROUP

- IAEA/GNIP site in Kraków
- 50-year precip isotopic data record
- high-altitude lab (clouds in-situ) @Kasprowy Wierch (6500 ft AMSL)
- zfs.agh.edu.pl

photo: naukaoklimacie.pl

plan of the talk

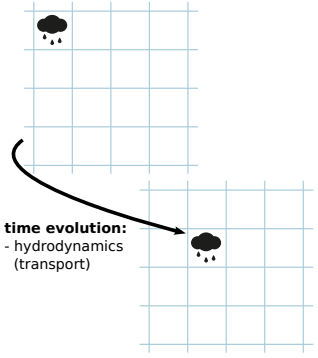
- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$: advected scalar field (advectee),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: flow velocity vector field (advector),
$G(\mathbf{x})$: fluid density, Jacobian of coordinate transformation, or their product

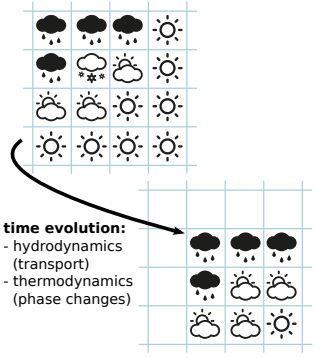


MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$: advected scalar field (advectee),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: flow velocity vector field (advector),
$G(\mathbf{x})$: fluid density, Jacobian of coordinate transformation, or their product



MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

  $\psi(\mathbf{x}, t)$: advected scalar field (advectee),
  $\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: flow velocity vector field (advector),
  $G(\mathbf{x})$: fluid density, Jacobian of coordinate transformation, or their product

- homogeneous problem in 1D and with $G = 1$:

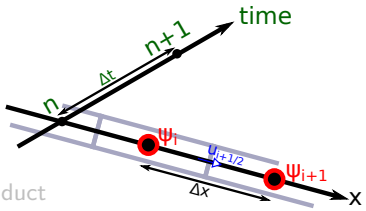
$$\partial_t \psi + \partial_x (u\psi) = 0$$

MPDATA key concepts: UPWIND discretisation

- advection equation / scalar conservation law:

$$\partial_t(G\psi) + \nabla \cdot (\mathbf{v}\psi) = GR$$

$\psi(\mathbf{x}, t)$: advected scalar field (advectee),
$\mathbf{v} = \{u, \ldots\} = G\dot{\mathbf{x}}$: flow velocity vector field (advector),
$G(\mathbf{x})$: fluid density, Jacobian of coordinate transformation, or their product



- homogeneous problem in 1D and with $G = 1$:

$$\partial_t \psi + \partial_x(u\psi) = 0$$

- UPWIND discretisation on a spatially staggered grid (n numbers time steps, i numbers grid steps):

$$\frac{\psi_i^{n+1} - \psi_i^n}{\Delta t} + \frac{\overbrace{f(\psi_i^n, \psi_{i+1}^n, u_{i+1/2}^n)}^{\text{right-hand wall flux}} - \overbrace{f(\psi_{i-1}^n, \psi_i^n, u_{i-1/2}^n)}^{\text{left-hand wall flux}}}{\Delta x} = 0$$

$$f(\psi_l, \psi_r, u) = \frac{\overbrace{u + |u|}^{\text{positive part}}}{2}\psi_l + \frac{\overbrace{u - |u|}^{\text{negative part}}}{2}\psi_r$$

MPDATA key concepts: Courant number & UPWIND stability criterion

- introducing non-dimensional Courant number $C = u\frac{\Delta t}{\Delta x}$:

$$\psi_i^{n+1} = \psi_i^n - \left[f(\psi_i^n, \psi_{i+1}^n, C_{i+1/2}^n) - f(\psi_{i-1}^n, \psi_i^n, C_{i-1/2}^n) \right]$$

yields a conservative and sing-preserving "UPWIND" hello-world scheme stable for $|C| \leqslant 1$.

```
1  def f(psi_l, psi_r, C):
2      return .5 * (C + abs(C)) * psi_l + \
3             .5 * (C - abs(C)) * psi_r
4  def step(psi: np.ndarray, i: slice, C: np.ndarray):
5      psi[i] = psi[i] - (
6          f(psi[i       ], psi[i + one], C[i + hlf]) -
7          f(psi[i - one], psi[i       ], C[i - hlf])
8      )
9  def upwind(nt: int, C: np.ndarray, psi: np.ndarray):
10     i = slice(1, len(psi) - 1)
11     for _ in range(nt):
12         step(psi, i, C)
```

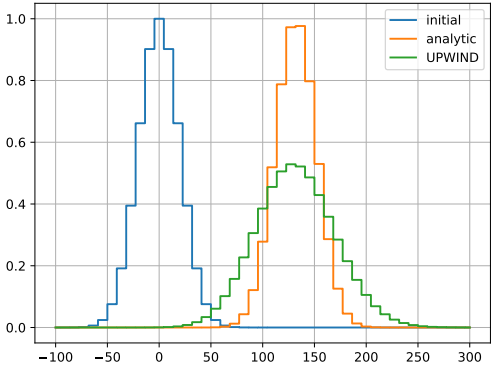
MPDATA key concepts: Courant number & UPWIND stability criterion

- introducing non-dimensional Courant number $C = u\frac{\Delta t}{\Delta x}$:

$$\psi_i^{n+1} = \psi_i^n - \Big[f(\psi_i^n, \psi_{i+1}^n, C_{i+1/2}^n) - f(\psi_{i-1}^n, \psi_i^n, C_{i-1/2}^n)\Big]$$

yields a conservative and sing-preserving "UPWIND" hello-world scheme stable for $|C| \leqslant 1$.

```
1  def f(psi_l, psi_r, C):
2      return .5 * (C + abs(C)) * psi_l + \
3             .5 * (C - abs(C)) * psi_r
4  def step(psi: np.ndarray, i: slice, C: np.ndarray):
5      psi[i] = psi[i] - (
6          f(psi[i       ], psi[i + one], C[i + hlf]) -
7          f(psi[i - one], psi[i       ], C[i - hlf])
8      )
9  def upwind(nt: int, C: np.ndarray, psi: np.ndarray):
10     i = slice(1, len(psi) - 1)
11     for _ in range(nt):
12         step(psi, i, C)
```



MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const. C for simplicity):

$$\psi_i^{n+1} = \psi_i^n + \partial_t\psi|_i^n\,(+\Delta t) + \frac{1}{2}\,\partial_t^2\psi\Big|_i^n\,(+\Delta t)^2 + O(\Delta t^3)$$
$$\psi_{i+1}^n = \psi_i^n + \partial_x\psi|_i^n\,(+\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(+\Delta x)^2 + O(\Delta x^3)$$
$$\psi_{i-1}^n = \psi_i^n + \partial_x\psi|_i^n\,(-\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(-\Delta x)^2 + O(\Delta x^3)$$

$$\rightsquigarrow \quad \psi_i^{n+1} = \psi_i^n - \left[\frac{C+|C|}{2}\,(\psi_i^n - \psi_{i-1}^n) + \frac{C-|C|}{2}\,(\psi_{i+1}^n - \psi_i^n)\right]$$

MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const. C for simplicity):

$$\psi_i^{n+1} = \psi_i^n + \partial_t\psi|_i^n\,(+\Delta t) + \frac{1}{2}\,\partial_t^2\psi\Big|_i^n\,(+\Delta t)^2 + O(\Delta t^3)$$
$$\psi_{i+1}^n = \psi_i^n + \partial_x\psi|_i^n\,(+\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(+\Delta x)^2 + O(\Delta x^3)$$
$$\psi_{i-1}^n = \psi_i^n + \partial_x\psi|_i^n\,(-\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(-\Delta x)^2 + O(\Delta x^3)$$

$$\rightsquigarrow \quad \psi_i^{n+1} = \psi_i^n - \left[\frac{C+|C|}{2}\,(\psi_i^n - \psi_{i-1}^n) + \frac{C-|C|}{2}\,(\psi_{i+1}^n - \psi_i^n)\right]$$

- which substituted to the UPWIND formulæ yields (up to second-order terms):

$$\partial_t\psi|_i^n\,\Delta t + \underbrace{\partial_t^2\psi}_{u^2\partial_x^2\psi}\Big|_i^n \frac{\Delta t^2}{2} = -\,C\,\Delta x\,\partial_x\,\psi|_i^n + \frac{|C|}{2}\,\Delta x^2\,\partial_x^2\,\psi|_i^n$$

MPDATA key concepts: numerical diffusion & modified-equation analysis

- UPWIND incurs **numerical diffusion**, quantifiable using Taylor expansion (const. C for simplicity):

$$\psi_i^{n+1} = \psi_i^n + \partial_t\psi|_i^n\,(+\Delta t) + \frac{1}{2}\,\partial_t^2\psi\Big|_i^n\,(+\Delta t)^2 + O(\Delta t^3)$$
$$\psi_{i+1}^n = \psi_i^n + \partial_x\psi|_i^n\,(+\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(+\Delta x)^2 + O(\Delta x^3)$$
$$\psi_{i-1}^n = \psi_i^n + \partial_x\psi|_i^n\,(-\Delta x) + \frac{1}{2}\,\partial_x^2\psi\Big|_i^n\,(-\Delta x)^2 + O(\Delta x^3)$$

$$\rightsquigarrow \quad \psi_i^{n+1} = \psi_i^n - \left[\frac{C+|C|}{2}\,(\psi_i^n - \psi_{i-1}^n) + \frac{C-|C|}{2}\,(\psi_{i+1}^n - \psi_i^n)\right]$$

- which substituted to the UPWIND formulæ yields (up to second-order terms):

$$\partial_t\psi|_i^n\,\Delta t + \underbrace{\partial_t^2\psi}_{u^2\partial_x^2\psi}\Big|_i^n\frac{\Delta t^2}{2} = -\,C\,\Delta x\,\partial_x\,\psi|_i^n + \frac{|C|}{2}\,\Delta x^2\,\partial_x^2\,\psi|_i^n$$

where $\partial_t^2\psi$ can be replaced with spatial derivative using the Cauchy-Kovalevskaya procedure:

$$\partial_t\psi|_i^n + u\partial_x\psi|_i^n = \underbrace{\left(|u|\frac{\Delta x}{2} - u^2\frac{\Delta t}{2}\right)}_{k\ -\ \text{numerical diffusion}}\partial_x^2\psi|_i^n$$

(e.g., Roberts & Weiss 1966, doi:10.2307/2003507)

MPDATA key concepts: antidiffusive pseudo-velocities

- diffusion can be cast as advection with a pseudo-velocity:

$$\partial_t \psi = k \partial_x^2 \psi + \ldots \quad \rightsquigarrow \quad \partial_t \psi + \partial_x (k \underbrace{\frac{\partial_x \psi}{\psi}}_{\text{pseudo-velocity}} \psi) = \ldots$$

(e.g., Lange 1973, doi:10.2172/4308175)

MPDATA key concepts: antidiffusive pseudo-velocities

- diffusion can be cast as advection with a pseudo-velocity:

$$\partial_t \psi = k \partial_x^2 \psi + \ldots \quad \rightsquigarrow \quad \partial_t \psi + \partial_x (\underbrace{k \frac{\partial_x \psi}{\psi}}_{\text{pseudo-velocity}} \psi) = \ldots$$

(e.g., Lange 1973, doi:10.2172/4308175)

- "**Smolarkiewicz** algorithm" (MPDATA): upwind-integrate backwards-in-time, with an anti-diffusive pseudo velocity to reverse the effects of numerical diffusion, iteratively (m numbers iteration)

$$C^{m+1}_{i-1/2} = \frac{\Delta t}{\Delta x} k^m_{i-1/2} \left. \frac{\partial_x \psi}{\psi} \right|^m_{i-1/2} \approx \begin{cases} 0 & \text{if } \psi^m_i + \psi^m_{i-1} = 0 \\ \left[|C^m_{i-1/2}| - (C^m_{i-1/2})^2 \right] \frac{\psi^m_i - \psi^m_{i-1}}{\psi^m_i + \psi^m_{i-1}} & \text{otherwise} \end{cases}$$

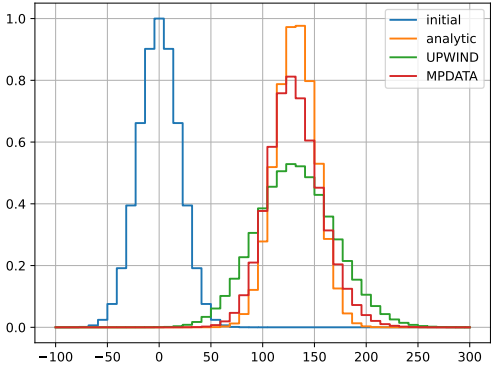
(Smolarkiewicz 1983 MWR, 1984 JCP: doi:10.1016/0021-9991(84)90121-9)

MPDATA hello-world (1D, single iteration) implementation

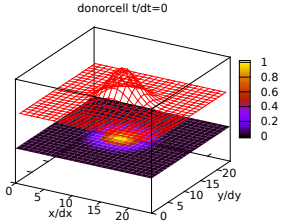
```
1  def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
2      return (abs(C[i - hlf]) - C[i - hlf] ** 2) * (
3          psi[i] - psi[i - one]
4      ) / (
5          psi[i - one] + psi[i]
6      )
7  def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
8      i = slice(1, len(psi) - 1)
9      i_ext = slice(1, len(psi))
10     for _ in range(nt):
11         upwind(psi, i, C)
12         upwind(psi, i, C_corr(C, i_ext, psi))
```

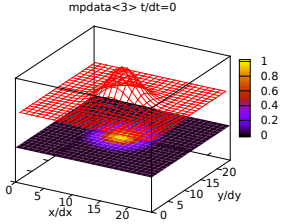
MPDATA hello-world (1D, single iteration) implementation

```
1  def C_corr(C: np.ndarray, i: slice, psi: np.ndarray):
2      return (abs(C[i − hlf]) − C[i − hlf] ** 2) * (
3          psi[i] − psi[i − one]
4      ) / (
5          psi[i − one] + psi[i]
6      )
7  def mpdata(nt: int, C: np.ndarray, psi: np.ndarray):
8      i = slice(1, len(psi) − 1)
9      i_ext = slice(1, len(psi))
10     for _ in range(nt):
11         upwind(psi, i, C)
12         upwind(psi, i, C_corr(C, i_ext, psi))
```

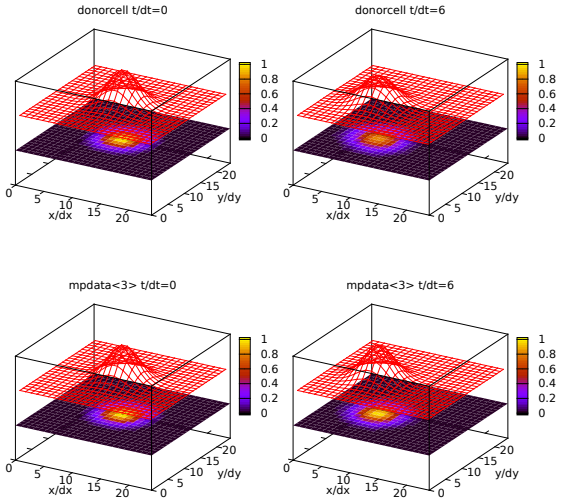


MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)

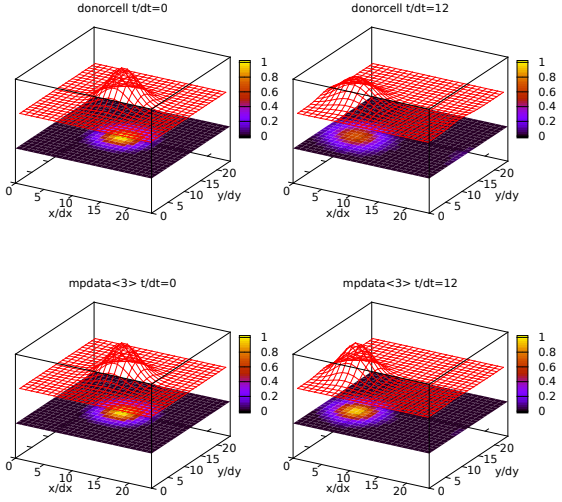




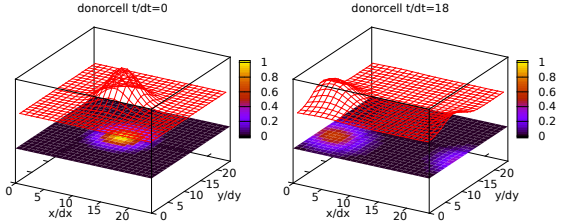
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



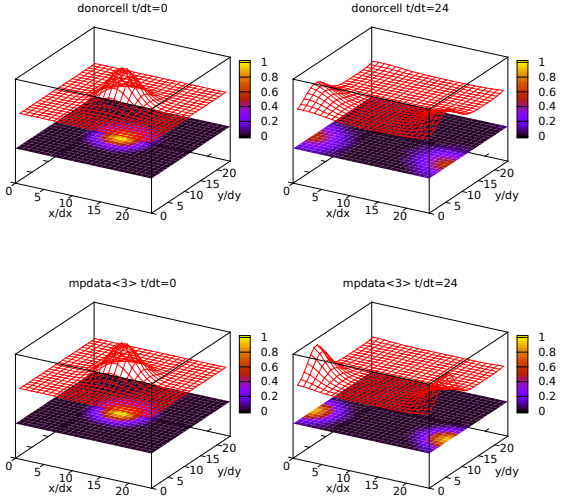
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



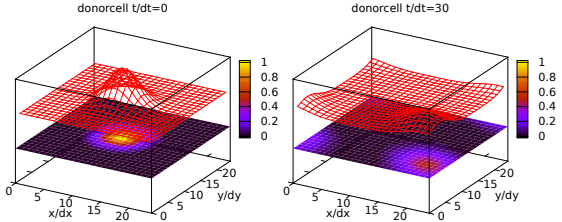
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



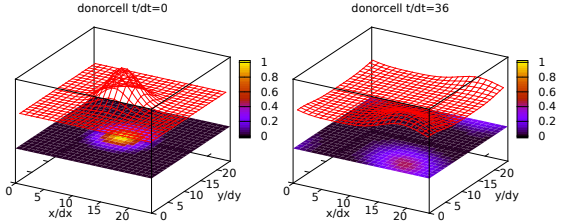
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



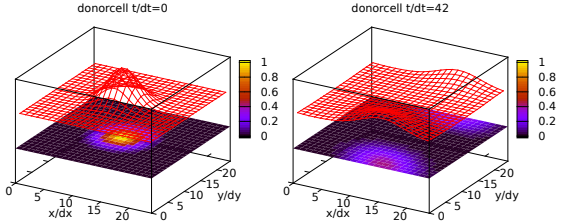
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



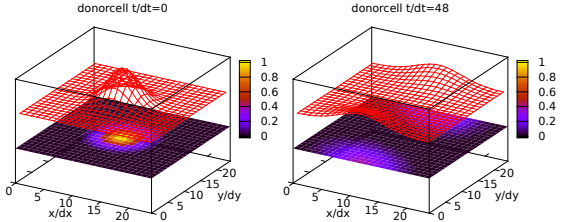
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



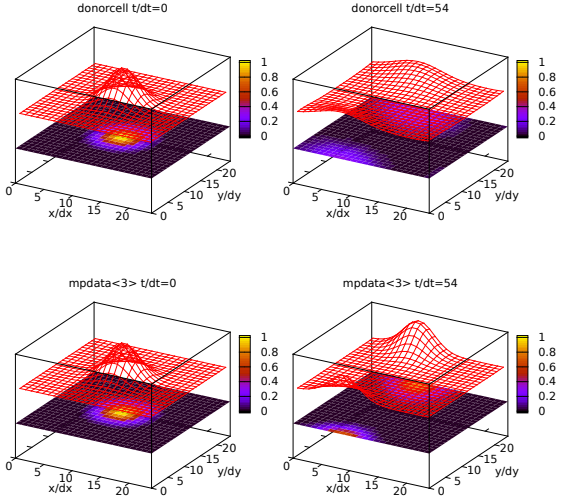
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



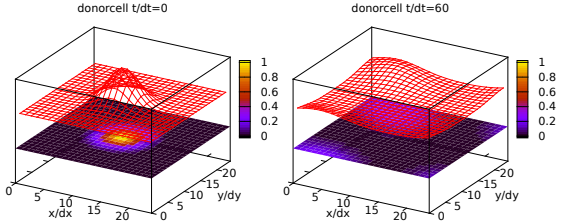
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



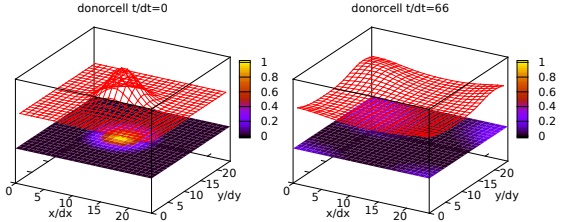
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



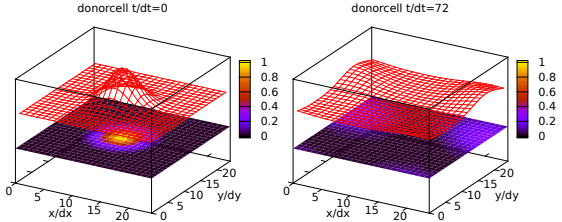
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



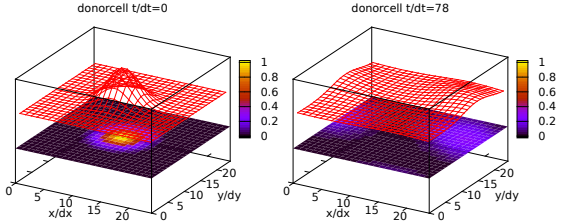
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



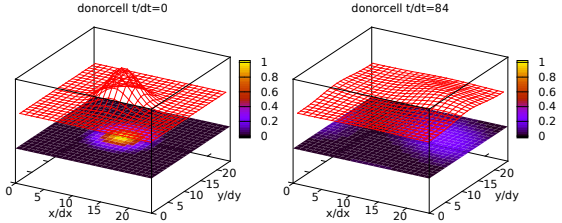
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



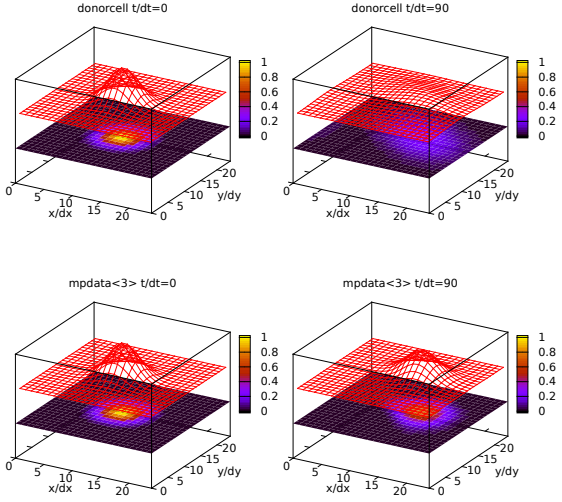
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



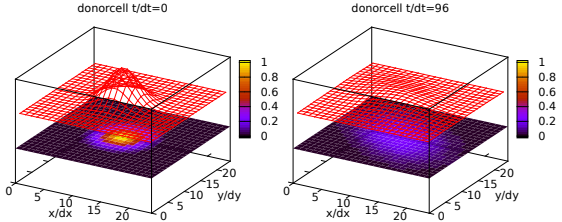
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



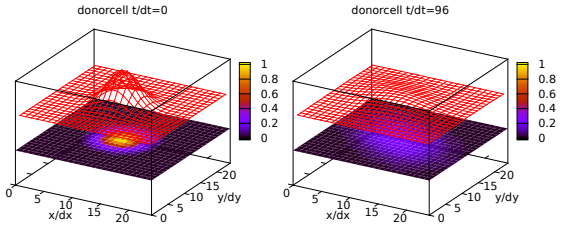
MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



MPDATA: ”M” for multi-dimensional (2D is not a sum of two 1D MPDATAs)



MPDATA: "M" for multi-dimensional (2D is not a sum of two 1D MPDATAs)



$$\sum_{d=0}^{1} \psi_{[i,j]+\pi^d_{1,0}} \equiv \psi_{[i+1,j]} + \psi_{[i,j+1]}$$

$$C'^{[d]}_{[i,j]+\pi^d_{1/2,0}} = \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \cdot \left[1 - \left| C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \right| \right] \cdot A^{[d]}_{[i,j]}(\psi) - \sum_{q=0, q\neq d}^{N} C^{[d]}_{[i,j]+\pi^d_{1/2,0}} \cdot \overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} \cdot B^{[d]}_{[i,j]}(\psi)$$

$$\overline{C}^{[q]}_{[i,j]+\pi^d_{1/2,0}} = \frac{1}{4} \cdot \left(C^{[q]}_{[i,j]+\pi^d_{1,1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,1/2}} + C^{[q]}_{[i,j]+\pi^d_{1,-1/2}} + C^{[q]}_{[i,j]+\pi^d_{0,-1/2}} \right)$$

$$A^{[d]}_{[i,j]} = \frac{\psi_{[i,j]+\pi^d_{1,0}} - \psi_{[i,j]}}{\psi_{[i,j]+\pi^d_{1,0}} + \psi_{[i,j]}}$$

$$B^{[d]}_{[i,j]} = \frac{1}{2} \frac{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} - \psi_{[i,j]+\pi^d_{1,-1}} - \psi_{[i,j]+\pi^d_{0,-1}}}{\psi_{[i,j]+\pi^d_{1,1}} + \psi_{[i,j]+\pi^d_{0,1}} + \psi_{[i,j]+\pi^d_{1,-1}} + \psi_{[i,j]+\pi^d_{0,-1}}}$$

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993
- resursive anti-diffusive velocities: Beason & Margolin 1988

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993
- resursive anti-diffusive velocities: Beason & Margolin 1988
- flux-corrected transport: Smolarkiewicz and Grabowski 1990

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993
- resursive anti-diffusive velocities: Beason & Margolin 1988
- flux-corrected transport: Smolarkiewicz and Grabowski 1990
- third-order terms: Smolarkiewicz and Margolin 1998

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993
- resursive anti-diffusive velocities: Beason & Margolin 1988
- flux-corrected transport: Smolarkiewicz and Grabowski 1990
- third-order terms: Smolarkiewicz and Margolin 1998
- infinite-gauge variant: Smolarkiewicz 2006

MPDATA variants (structured grid)

- basic (+iterations): Smolarkiewicz 1983
- n-dimensions, diffusion, divergent flow corrections: Smolarkiewicz 1984
- coordinate transformation: Smolarkiewicz and Clark 1986, Smolarkiewicz and Margolin 1993
- resursive anti-diffusive velocities: Beason & Margolin 1988
- flux-corrected transport: Smolarkiewicz and Grabowski 1990
- third-order terms: Smolarkiewicz and Margolin 1998
- infinite-gauge variant: Smolarkiewicz 2006
- fully third-order variant: Waruszewski et al. 2018

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)
PISM	C++	2D	/pism	U. Alaska Fairbanks

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)
PISM	C++	2D	/pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	/igfuw/bE_SDs	NCAR

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)
PISM	C++	2D	/pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	/igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	/apc-llc	RAS (?)

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)
PISM	C++	2D	/pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	/igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	/apc-llc	RAS (?)

reusable:

libmpdata++	C++/Blitz++	1,2,3D	/igfuw	IGF FUW

MPDATA implementations

known closed-source (Numerical Weather Prediction):

- COSMO
- ECMWF IFS

open-source:

integrated into CFD packages:

AtmosFOAM	C++ & Python/Numba	3D	/AtmosFOAM	U. Reading
ROMS	FORTRAN	3D	/myroms	UCLA (?)
PISM	C++	2D	/pism	U. Alaska Fairbanks
babyEULAG	FORTRAN	2D	/igfuw/bE_SDs	NCAR
apc-llc/mpdata	C/CUDA	3D	/apc-llc	RAS (?)

reusable:

libmpdata++	C++/Blitz++	1,2,3D	/igfuw	IGF FUW
PyMPDATA	Python/Numba	1,2,3D	/open-atmos	UJ, AGH

plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)

PyMPDATA: 100% Python codebase

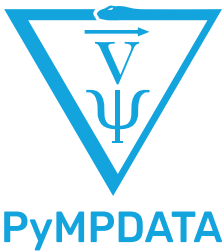
```python
@numba.njit(**options.jit_flags)
def a_term(psi):
    """eq. 13 in [Smolarkiewicz 1984](https://doi.org/10.1016/0021-9991(84)90121-9);
    eq. 17a in [Smolarkiewicz & Margolin 1998](https://doi.org/10.1006/jcph.1998.5901)"""
    result = ats(*psi, 1) - ats(*psi, 0)
    if infinite_gauge:
        return result / 2
    return result / (ats(*psi, 1) + ats(*psi, 0) + epsilon)



@numba.njit(**options.jit_flags)
def b_term(psi):
    """eq. 13 in [Smolarkiewicz 1984](https://doi.org/10.1016/0021-9991(84)90121-9);
    eq. 17b in [Smolarkiewicz & Margolin 1998](https://doi.org/10.1006/jcph.1998.5901)"""
    result = ats(*psi, 1, 1) + ats(*psi, 0, 1) - ats(*psi, 1, -1) - ats(*psi, 0, -1)
    if infinite_gauge:
        return result / 4

    return result / (
        ats(*psi, 1, 1)
        + ats(*psi, 0, 1)
        + ats(*psi, 1, -1)
        + ats(*psi, 0, -1)
        + epsilon
    )
```

PyMPDATA: design goals, tech stack, features



- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")

PyMPDATA: design goals, tech stack, features

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...

PyMPDATA: design goals, tech stack, features

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...
- API: 5 classes & single "advance" method

PyMPDATA: design goals, tech stack, features

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...
- API: 5 classes & single "advance" method
- array-traversal abstractions avoiding explicit fill-halo or barrier calls

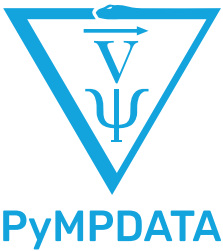
PyMPDATA: design goals, tech stack, features

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...
- API: 5 classes & single "advance" method
- array-traversal abstractions avoiding explicit fill-halo or barrier calls
- docs (and CI) covers usage from Python, Julia, Matlab and Rust

PyMPDATA: design goals, tech stack, features

- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...
- API: 5 classes & single "advance" method
- array-traversal abstractions avoiding explicit fill-halo or barrier calls
- docs (and CI) covers usage from Python, Julia, Matlab and Rust
- suite of 20+ Jupyter notebook examples maintained with the project all with badges enabling **single-click execution on Colab**

PyMPDATA: design goals, tech stack, features



- Numba JIT $\rightsquigarrow$ pure-Python code with compiled-language performance (plus OpenMP-like multi-threading, but no profiling tools)
- runs on Linux, macOS & Windows (no compilation, just "pip install")
- $> 90\%$ unit-test coverage (codecov) and growing...
- API: 5 classes & single "advance" method
- array-traversal abstractions avoiding explicit fill-halo or barrier calls
- docs (and CI) covers usage from Python, Julia, Matlab and Rust
- suite of 20+ Jupyter notebook examples maintained with the project all with badges enabling **single-click execution on Colab**
- examples in 1D, 2D & 3D: advection-diffusion, bin cloud μ-physics, spherical coordinates, shallow-water, Black-Scholes, Burgers, Boussinesq

plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

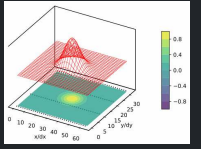
PyMPDATA & PyMPDATA-examples docs: open-atmos.io/PyMPDATA

PyMPDATA

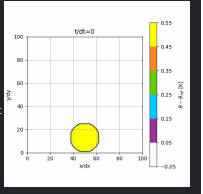
Documentation

What is PyMPDATA?

PyMPDATA is a **Numba-accelerated** multi-threaded Pythonic implementation of the **MPDATA algorithm** of Smolarkiewicz et al. used in geophysical fluid dynamics and beyond for **numerically solving generalised convection-diffusion PDEs**. PyMPDATA supports integration in 1D, 2D and 3D structured meshes with optional coordinate transformations. The first animation shown depicts a "hello-world" 2D advection-only simulation with dotted lines indicating domain decomposition across three threads. The second animation depicts an MPDATA solution to coupled mass and momentum conservation equations for a buoyancy-driven flow in Boussinesq approximation (see Jaruga et al. 2015 example).



A separate project called **PyMPDATA-MPI** depicts how **numba-mpi** can be used to enable distributed memory parallelism in PyMPDATA.



What is the difference between PyMPDATA and PyMPDATA-examples?

PyMPDATA is a Python package that provides the MPDATA algorithm implementation. It is a library that can be used in your own projects.

PyMPDATA-examples is a Python package that provides examples of how to use PyMPDATA. It includes common Python modules used in PyMPDATA smoke tests and in example Jupyter notebooks (but the package wheels do not include the notebooks, only .py files imported from the notebooks and PyMPDATA tests).

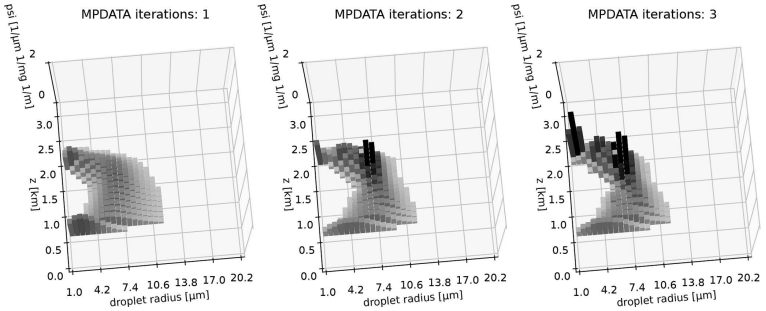
MPDATA for condensational growth in bin μ-physics (Olesik et al. 2022)

Geosci. Model Dev., 15, 3879–3899, 2022
https://doi.org/10.5194/gmd-15-3879-2022

Geoscientific Model Development
Open Access
EGU

On numerical broadening of particle-size spectra: a condensational growth study using PyMPDATA 1.0

Michael A. Olesik[1], Jakub Banaśkiewicz[2], Piotr Bartman[2], Manuel Baumgartner[3,4], Simon Unterstrasser[5], and Sylwester Arabas[6,2]



- spectro-spatial advection (single-column model)
- spectral broadening vs. MPDATA options

MPDATA for Asian option pricing using 2D PDE (Magnuszewski et al. 2025)

arXiv > q-fin > arXiv:2505.24435

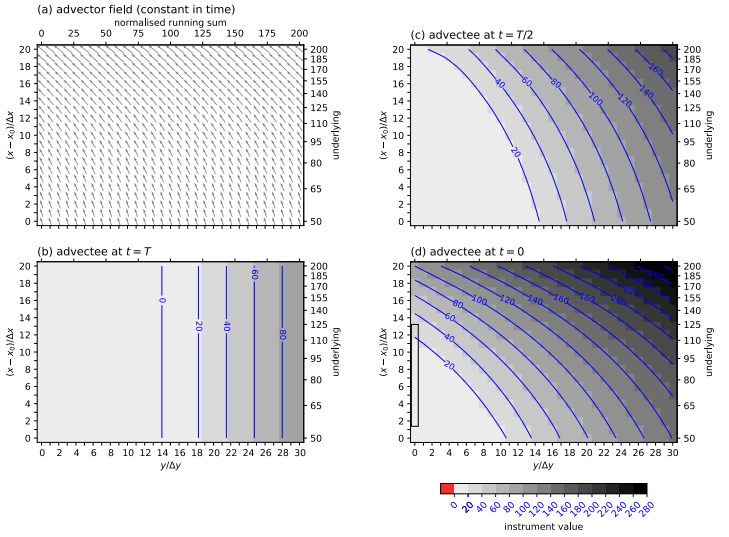
Quantitative Finance > Computational Finance

[Submitted on 30 May 2025]

Path-dependent option pricing with two-dimensional PDE using MPDATA

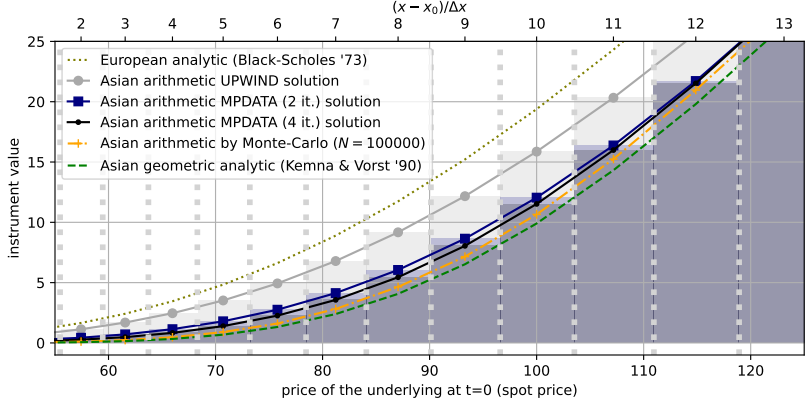
Paweł Magnuszewski, Sylwester Arabas

In this paper, we discuss a simple yet robust PDE method for evaluating path-dependent Asian-style options using the non-oscillatory forward-in-time second-order MPDATA finite-difference scheme. The valuation methodology involves casting the Black-Merton-Scholes equation as a transport problem by first transforming it into a homogeneous advection-diffusion PDE via variable substitution, and then expressing the diffusion term as an advective flux using the pseudo-velocity technique.



MPDATA for Asian option pricing using 2D PDE (Magnuszewski et al. 2025)

$$\frac{\partial f}{\partial t} + rS\frac{\partial f}{\partial S} + \frac{\sigma^2}{2}S^2\frac{\partial^2 f}{\partial S^2} + v\frac{\partial f}{\partial A} - rf = 0$$

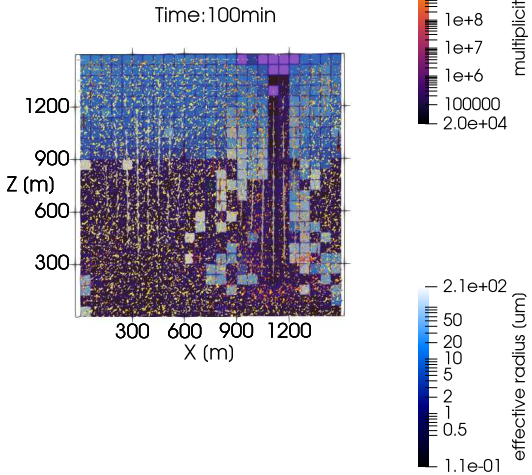


arXiv:2505.24435

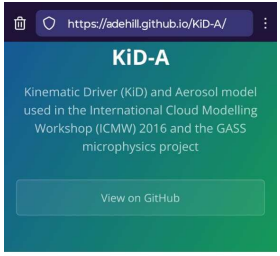
Eulerian transport for PySDM (the original reason for PyMPDATA dev)

https://pypi.org/p/PySDM

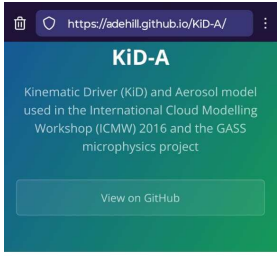
PySDM



Eulerian transport for PySDM: pure-Python implementation of MetOffice KiD



Eulerian transport for PySDM: pure-Python implementation of MetOffice KiD



Geosci. Model Dev., 16, 4193–4211, 2023
https://doi.org/10.5194/gmd-16-4193-2023

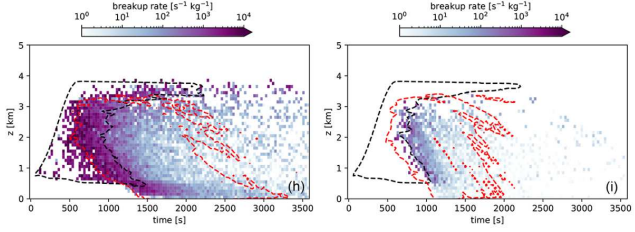
Geoscientific Model Development
Open Access
EGU

Development and technical paper

Breakups are complicated: an efficient representation of collisional breakup in the superdroplet method

Emily de Jong[1], John Ben Mackay[2,a], Oleksii Bulenok[3], Anna Jaruga[4], and Sylwester Arabas[5,b,c]

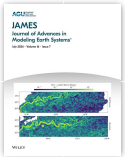
[1]Department of Mechanical and Civil Engineering, California Institute of Technology, Pasadena, CA, USA
[2]Scripps Institution of Oceanography, San Diego, CA, USA
[3]Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland
[4]Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA
[5]Faculty of Physics and Applied Computer Science, AGH University of Krakow, Kraków, Poland
[a]formerly at: Department of Environmental Science and Engineering, California Institute of Technology, Pasadena, CA, USA
[b]formerly at: Department of Atmospheric Sciences, University of Illinois Urbana-Champaign, Urbana, IL, USA
[c]formerly at: Faculty of Mathematics and Computer Science, Jagiellonian University, Kraków, Poland



Eulerian transport for PySDM: pure-Python implementation of MetOffice KiD

JAMES | Journal of Advances in Modeling Earth Systems

Research Article | Open Access



Volume 16, Issue 7
July 2024
e2023MS004028
This article also appears in:
The CliMA Earth System Model

Training Warm-Rain Bulk Microphysics Schemes Using Super-Droplet Simulations

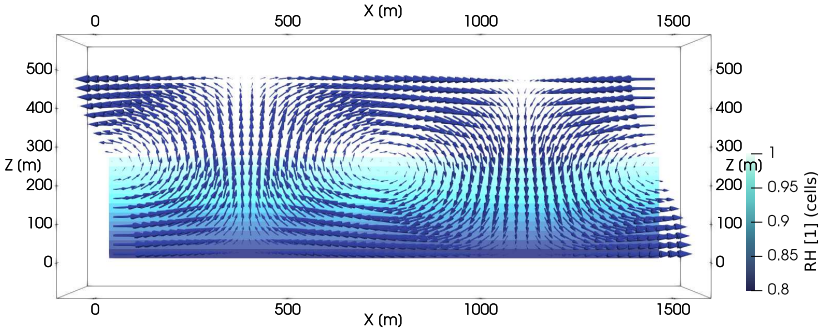
Sajjad Azimi, Anna Jaruga, Emily de Jong, Sylwester Arabas, Tapio Schneider

First published: 26 July 2024 | https://doi.org/10.1029/2023MS004028

Abstract

Cloud microphysics is a critical aspect of the Earth's climate system, which involves processes at the nano- and micrometer scales of droplets and ice particles. In climate modeling, cloud microphysics is commonly represented by bulk models, which contain simplified process rates that require calibration. This study presents a framework for calibrating warm-rain bulk schemes using high-fidelity super-droplet simulations that provide a more accurate and physically based representation of cloud and precipitation processes. The calibration framework employs ensemble Kalman methods including Ensemble Kalman Inversion and Unscented Kalman Inversion to calibrate bulk microphysics schemes with probabilistic super-droplet simulations. We demonstrate the framework's effectiveness by calibrating a single-moment bulk scheme, resulting in a reduction of data-model mismatch by more than 75% compared to the model with initial parameters. Thus, this study demonstrates a powerful tool for enhancing the accuracy of bulk microphysics schemes in atmospheric models and improving climate modeling.

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)



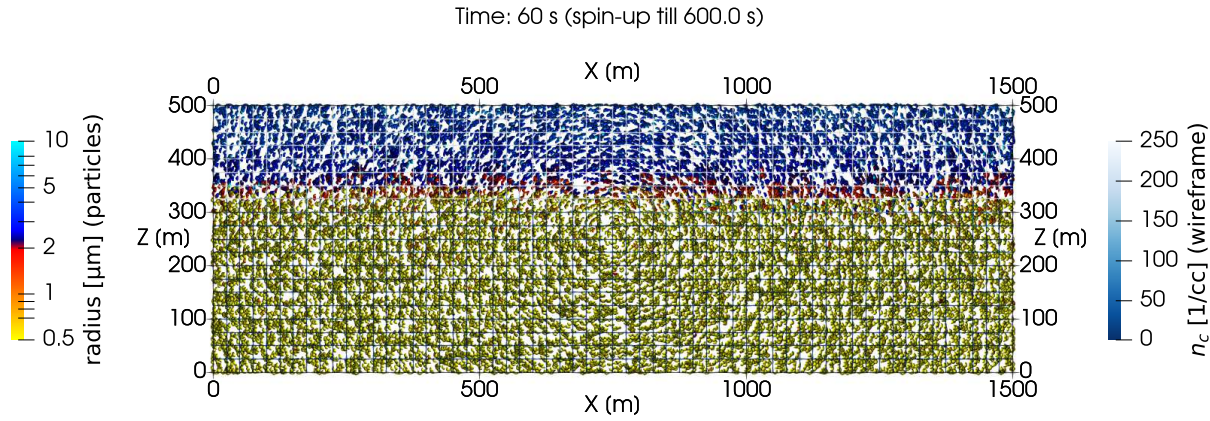
Journal of Advances in Modeling Earth Systems / Volume 17, Issue 4 / e2024MS004770

Immersion Freezing in Particle-Based Aerosol-Cloud Microphysics: A Probabilistic Perspective on Singular and Time-Dependent Models

Sylwester Arabas, Jeffrey H. Curtis, Israel Silber, Ann M. Fridlind, Daniel A. Knopf, Matthew West, Nicole Riemer

First published: 12 April 2025
https://doi.org/10.1029/2024MS004770

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

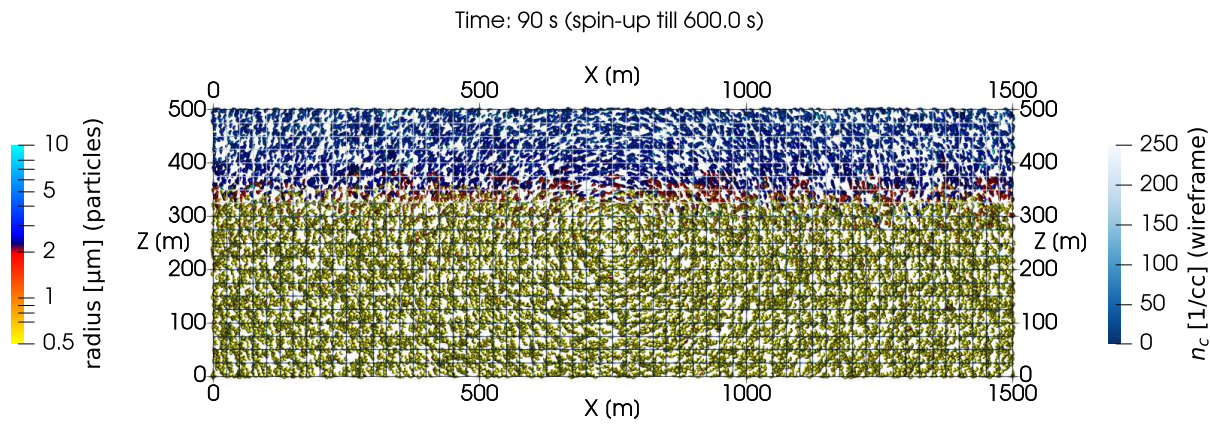


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

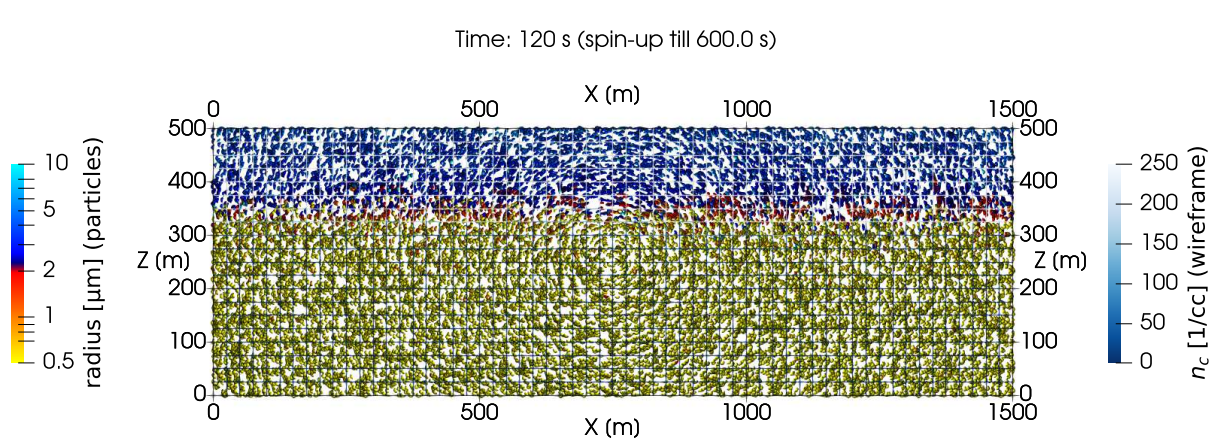


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

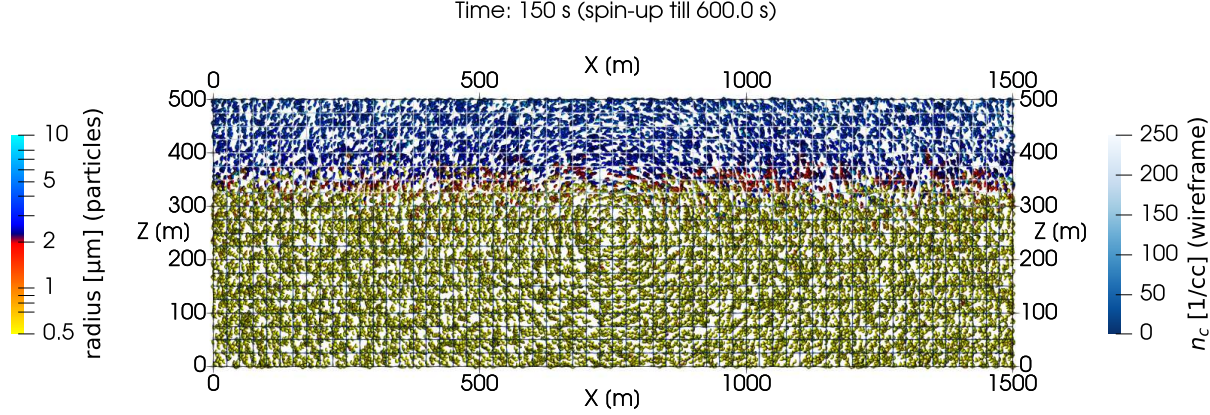


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

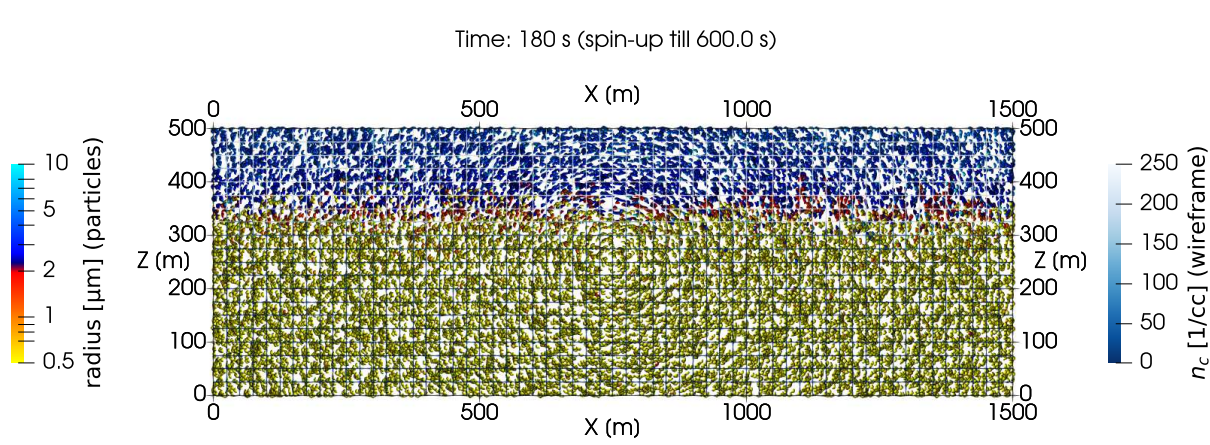


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

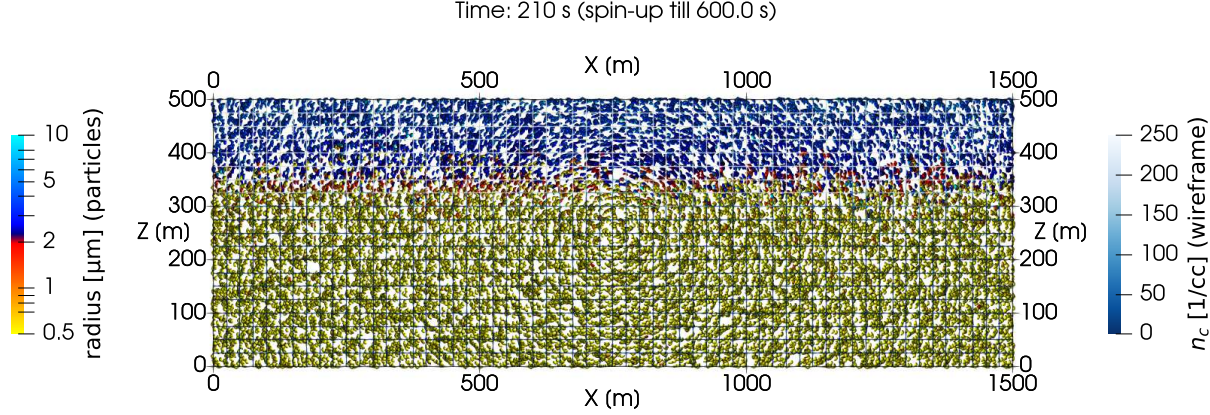


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

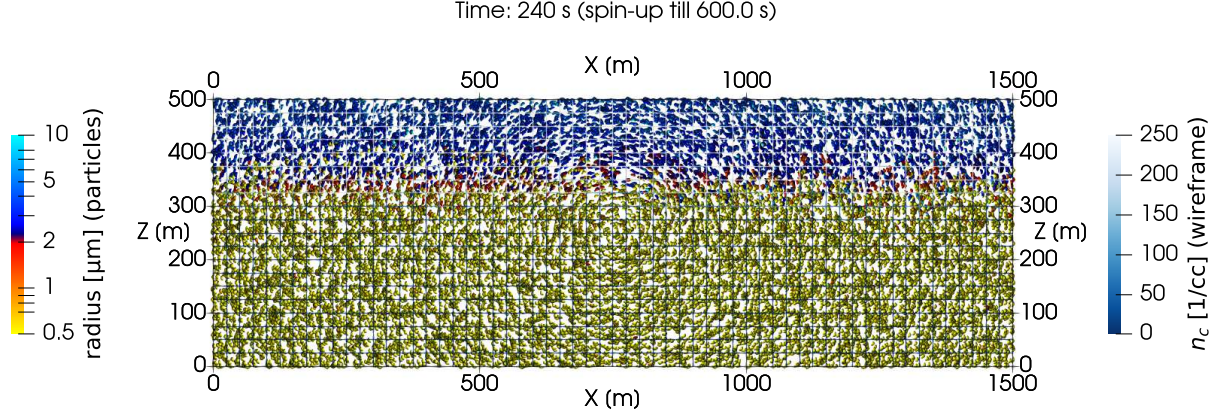


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

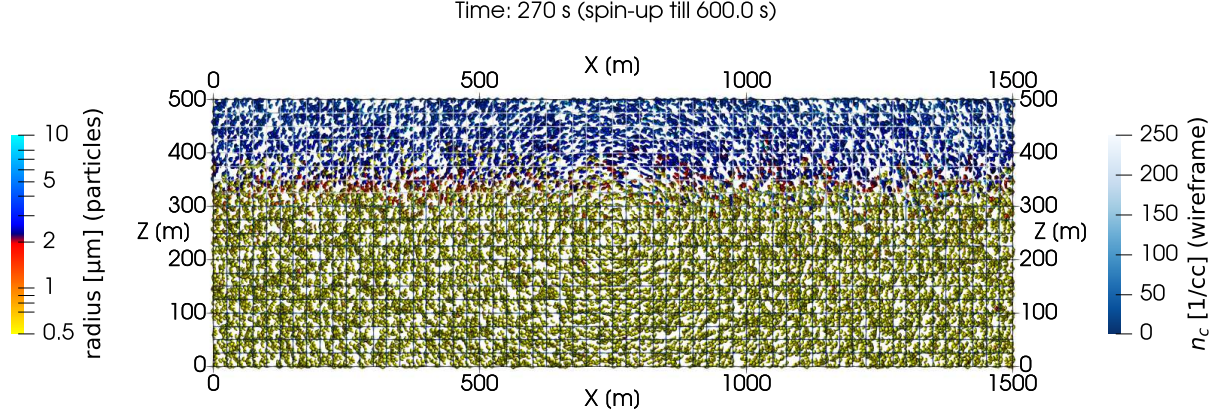


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_{\text{g}} = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

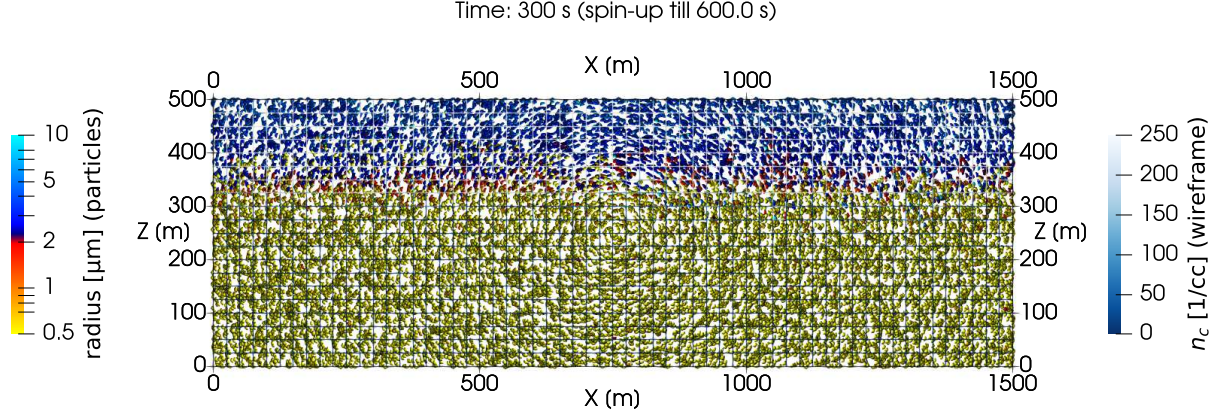


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

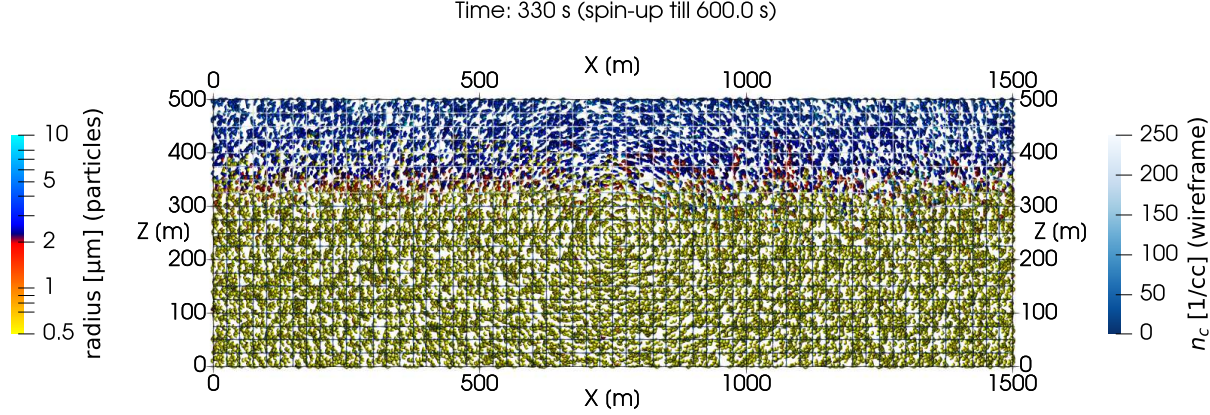


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

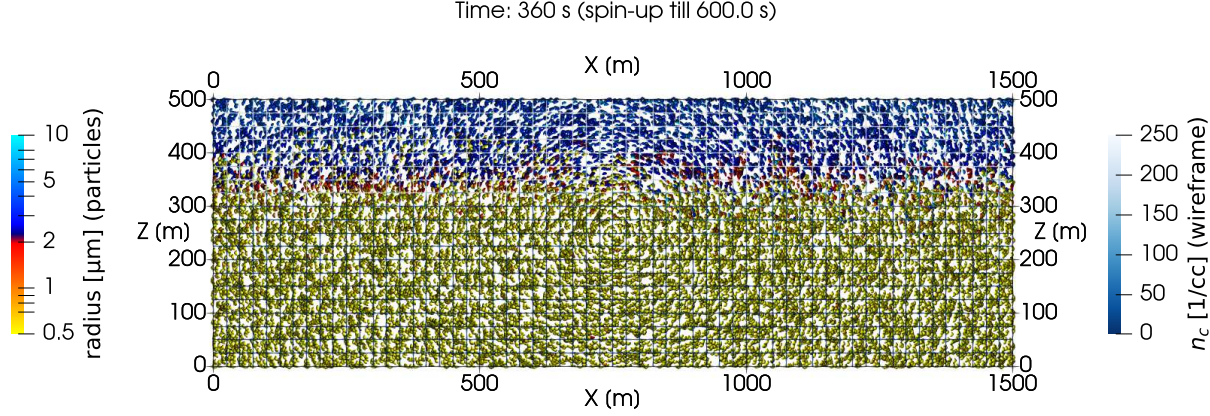


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

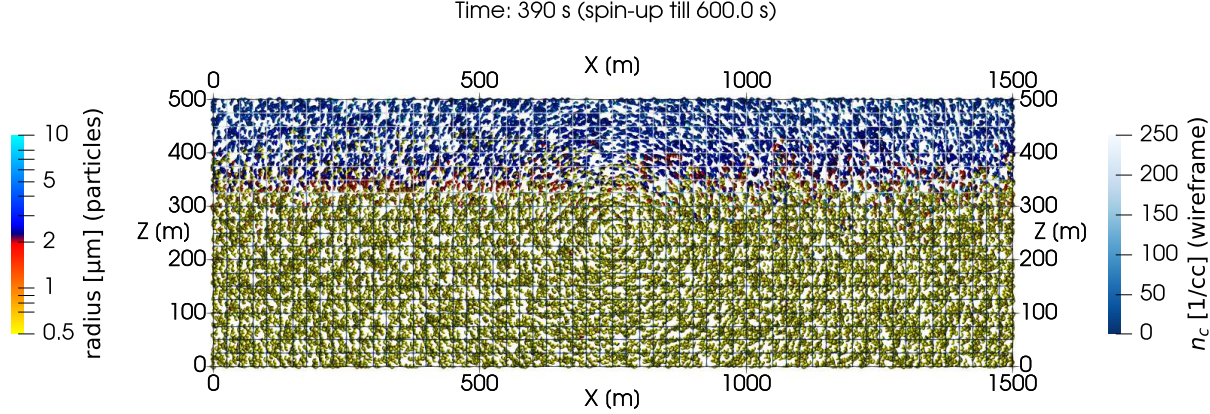


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

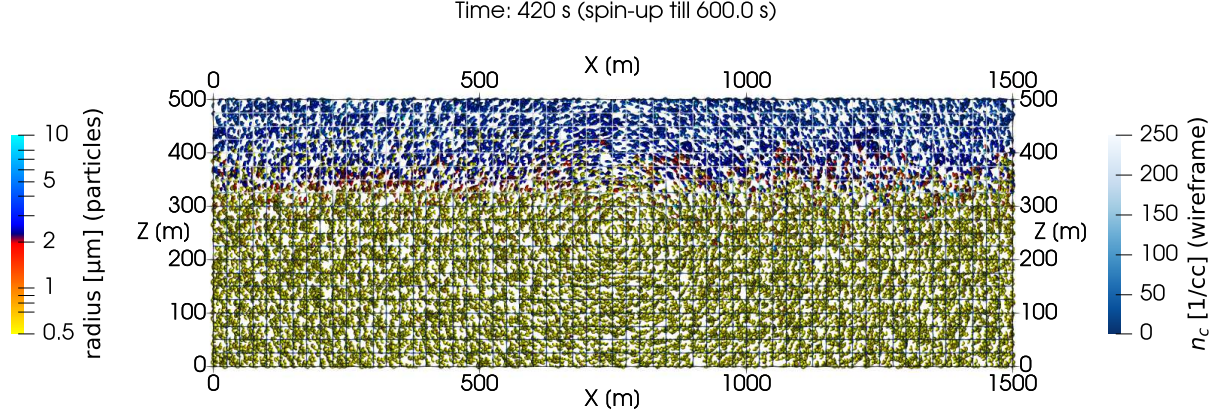


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

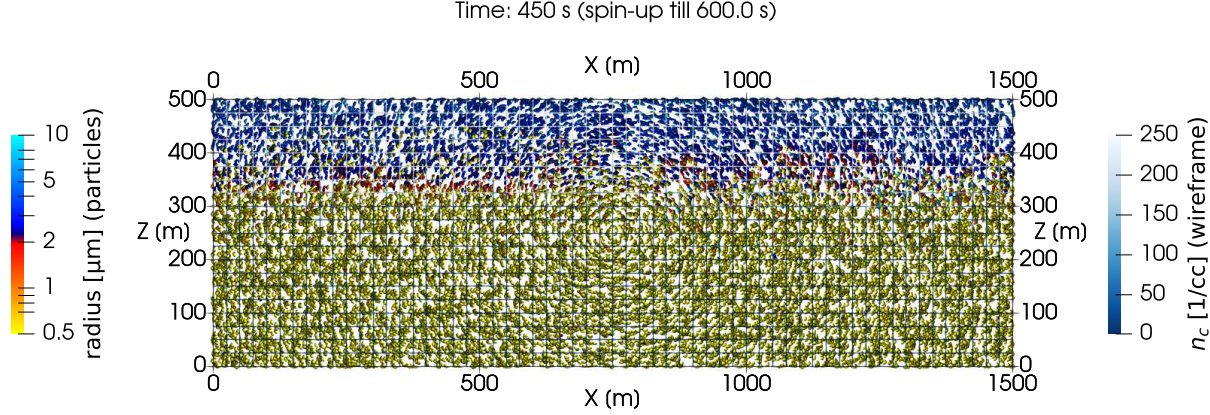


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

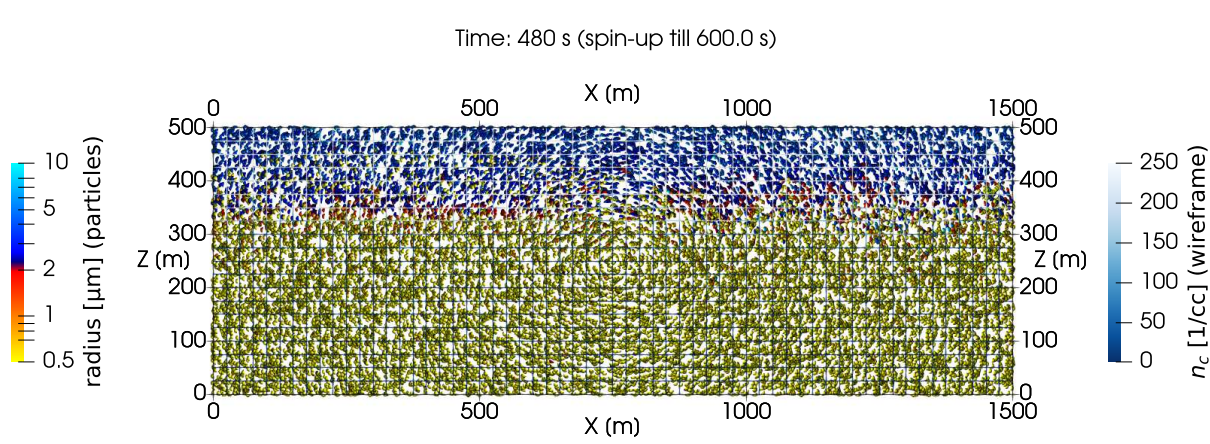


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

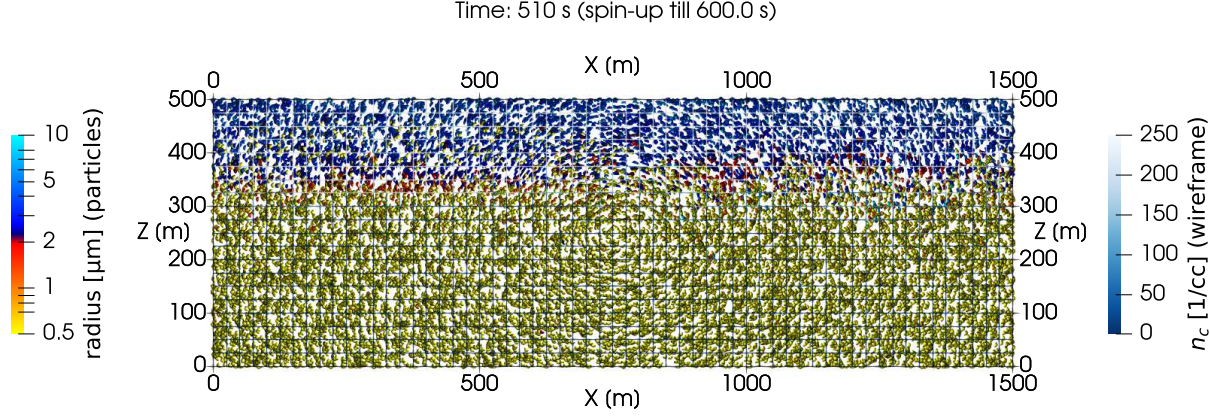


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

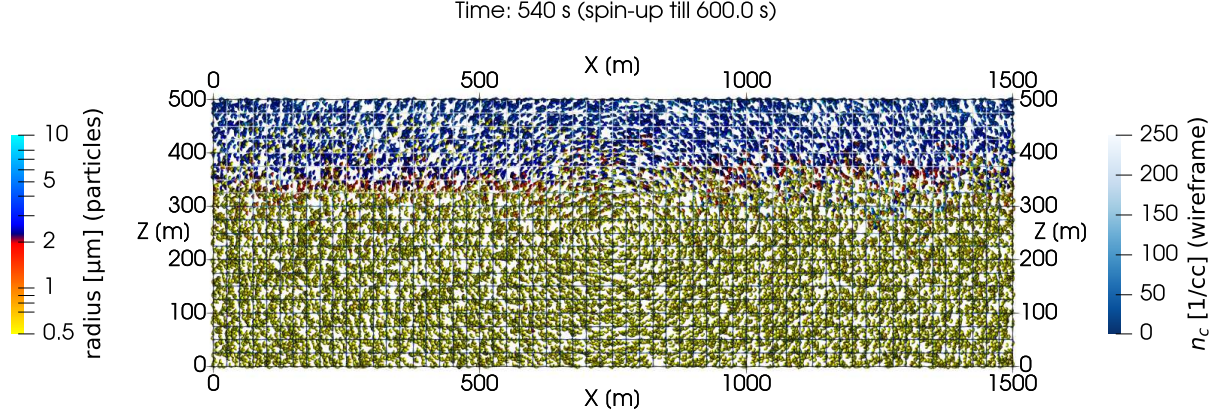


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

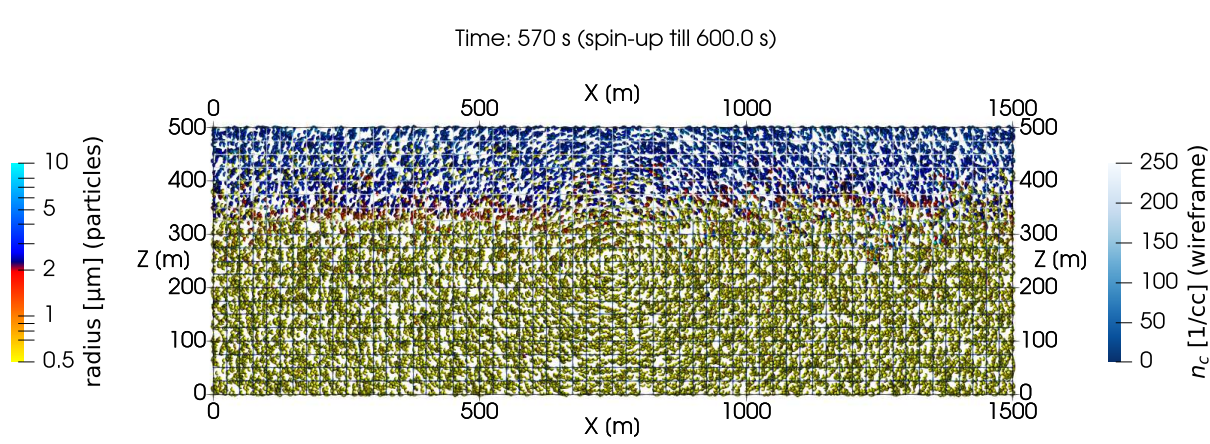


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

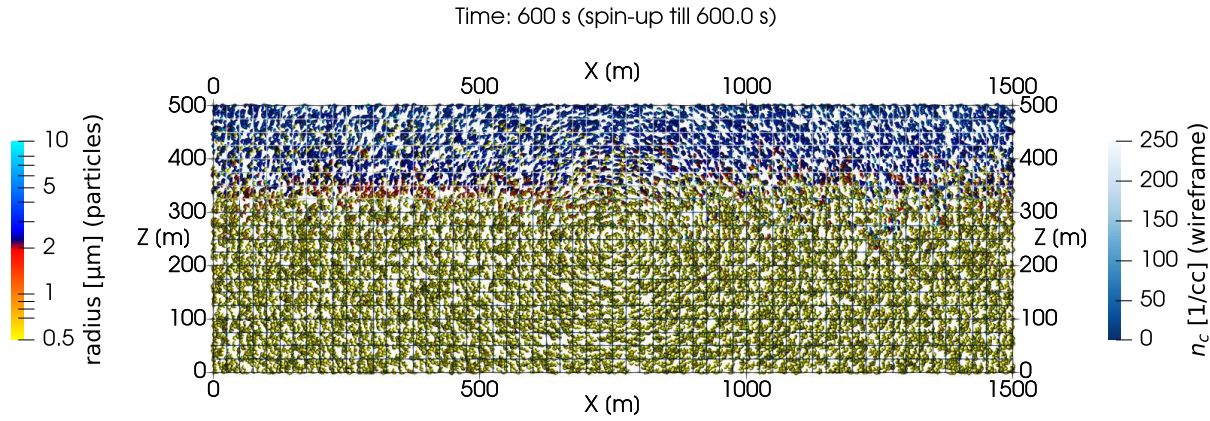


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

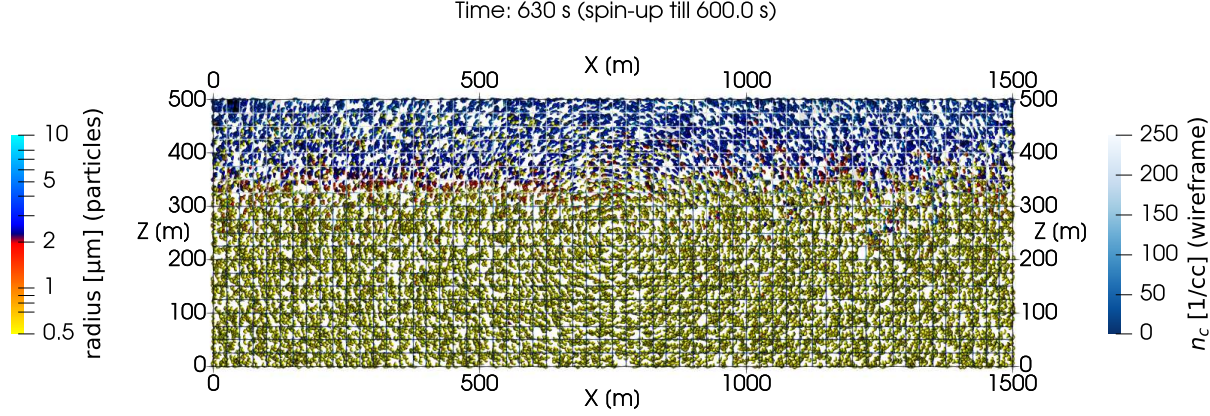


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

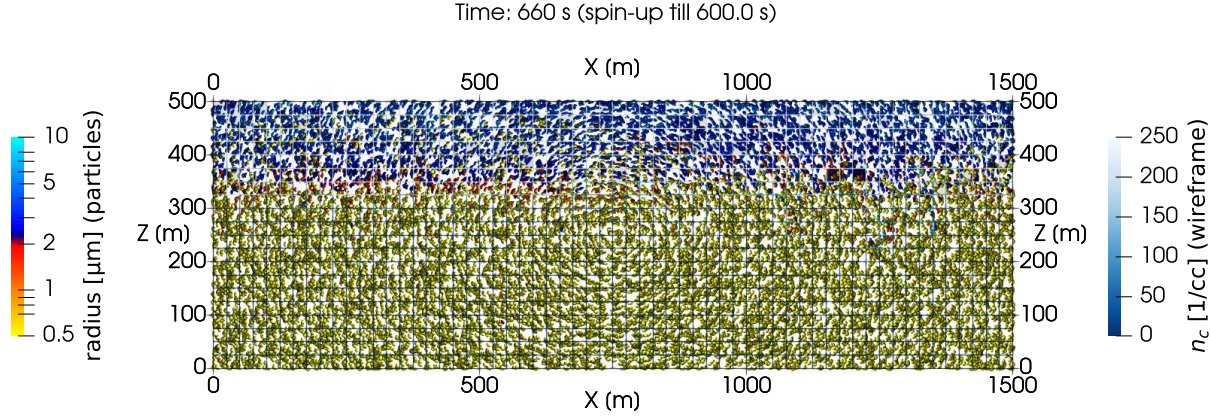


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

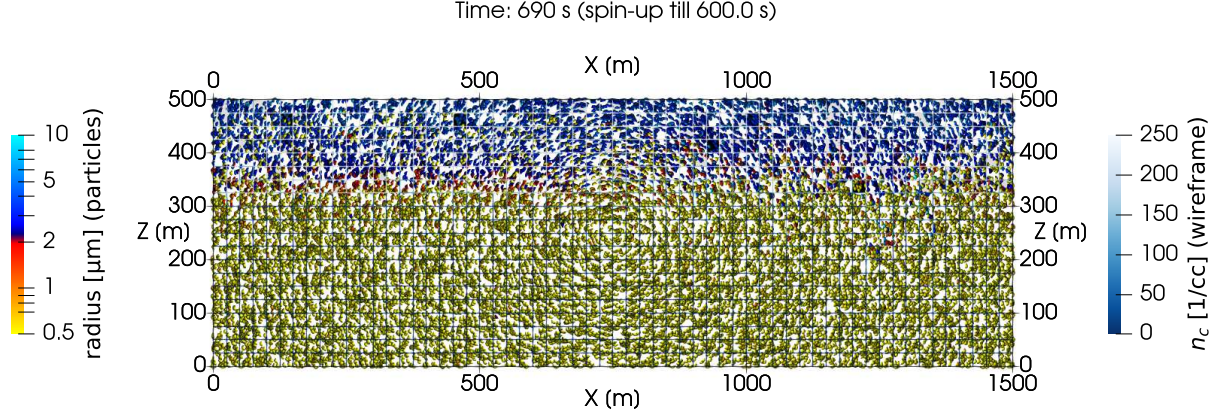


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

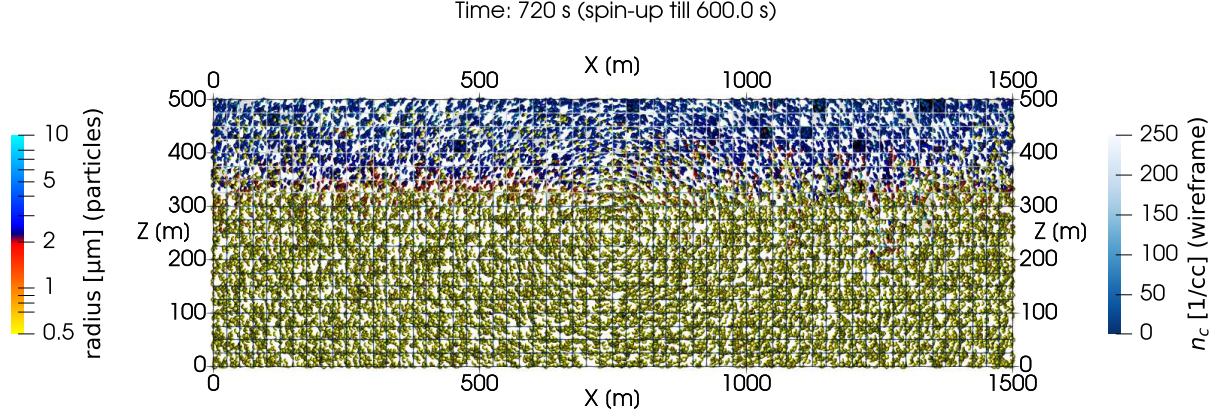


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

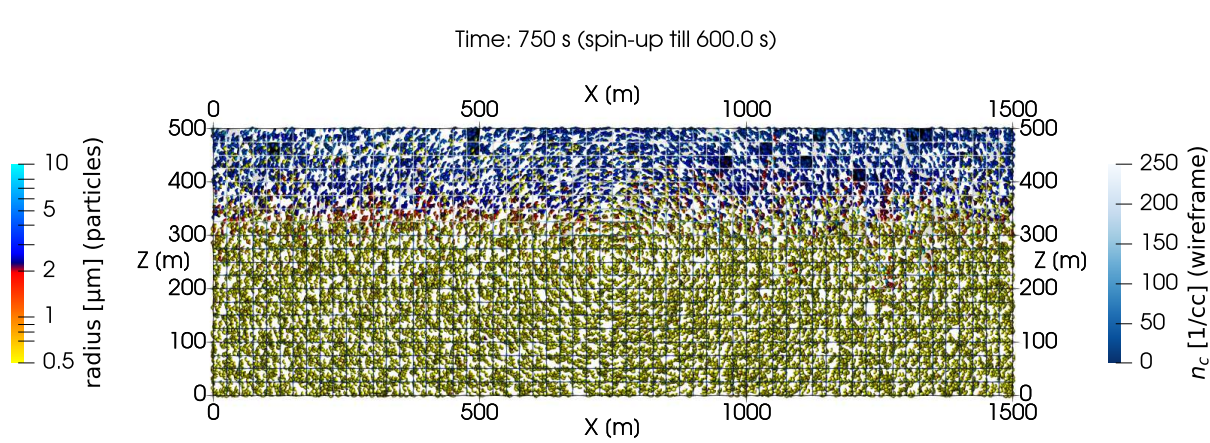


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

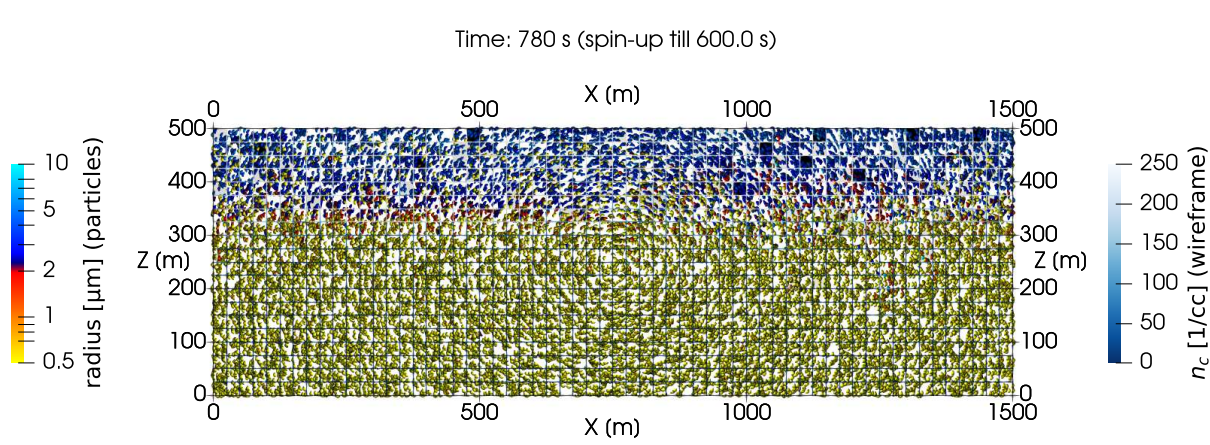


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

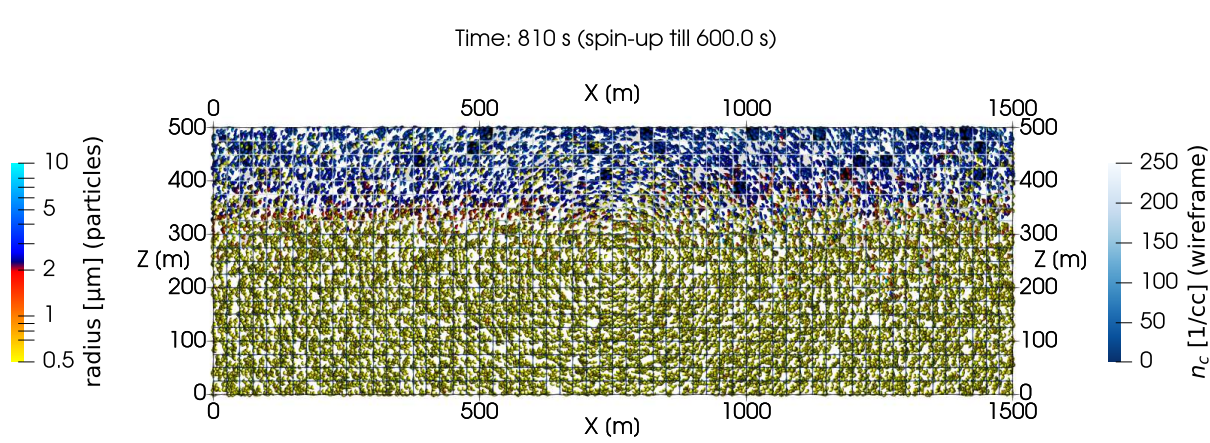


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

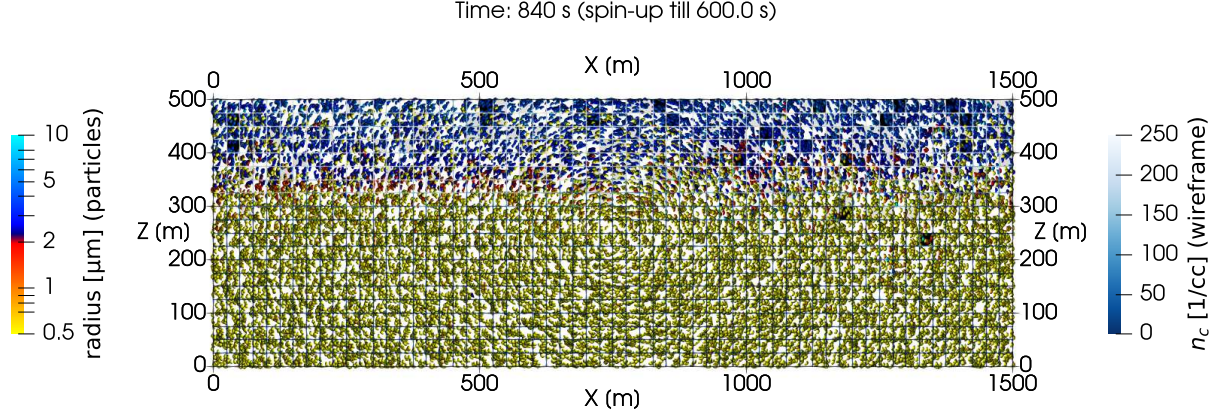


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

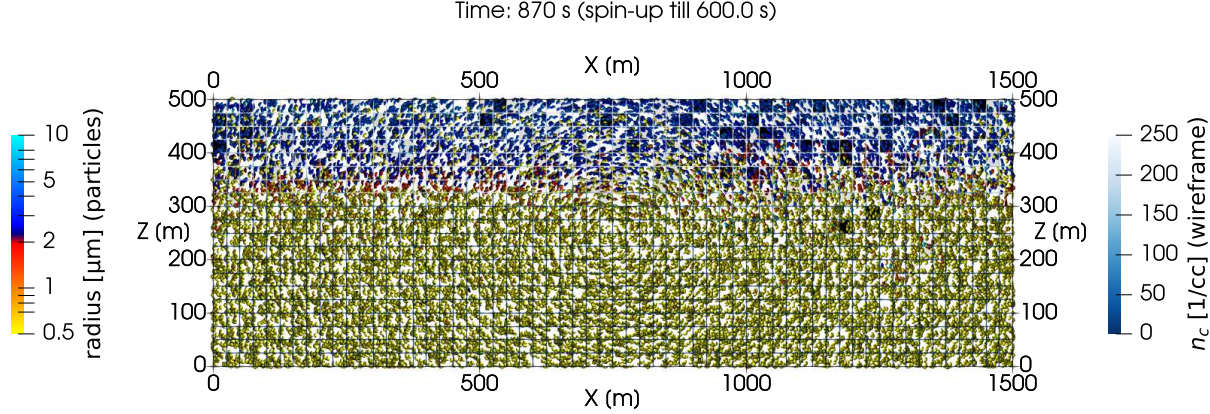


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

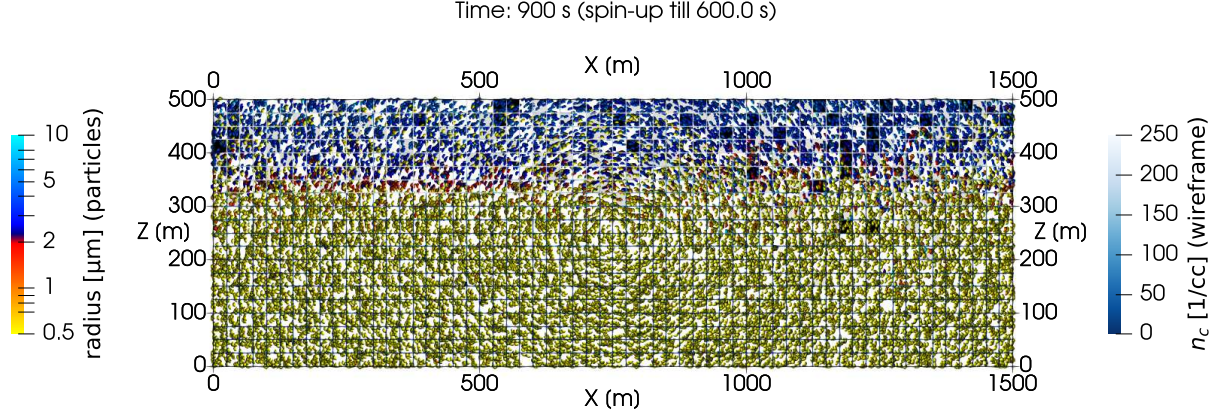


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

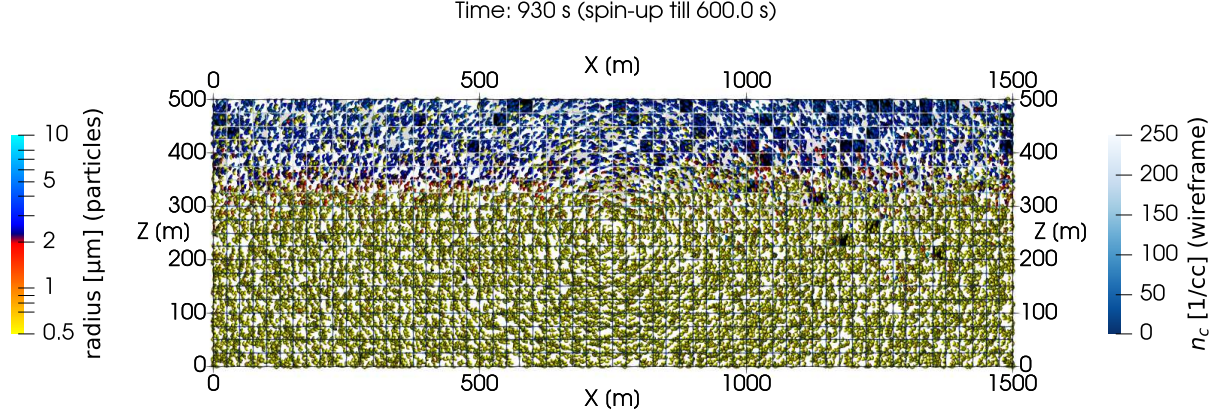


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

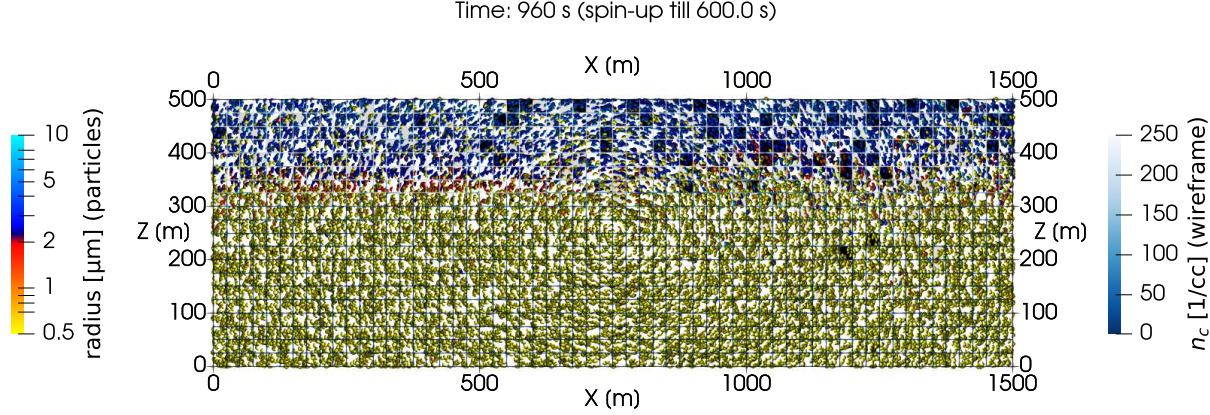


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

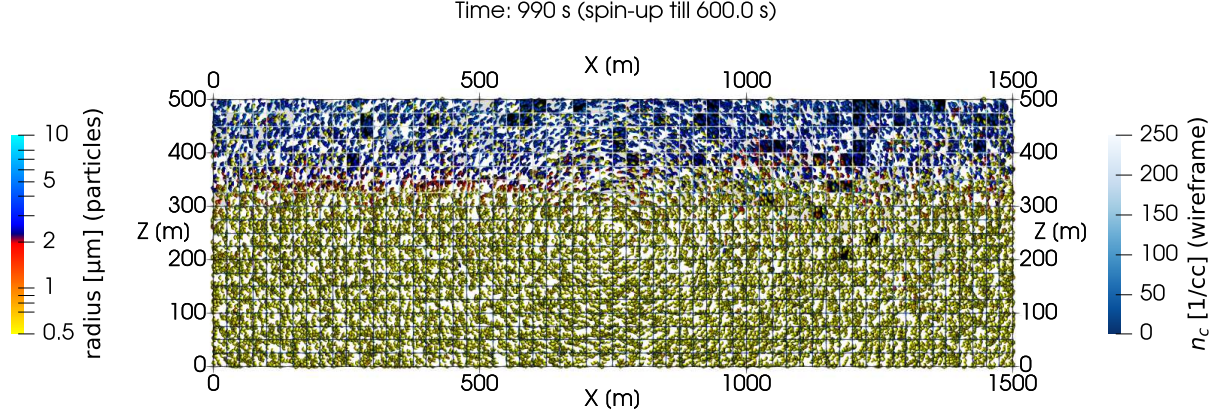


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

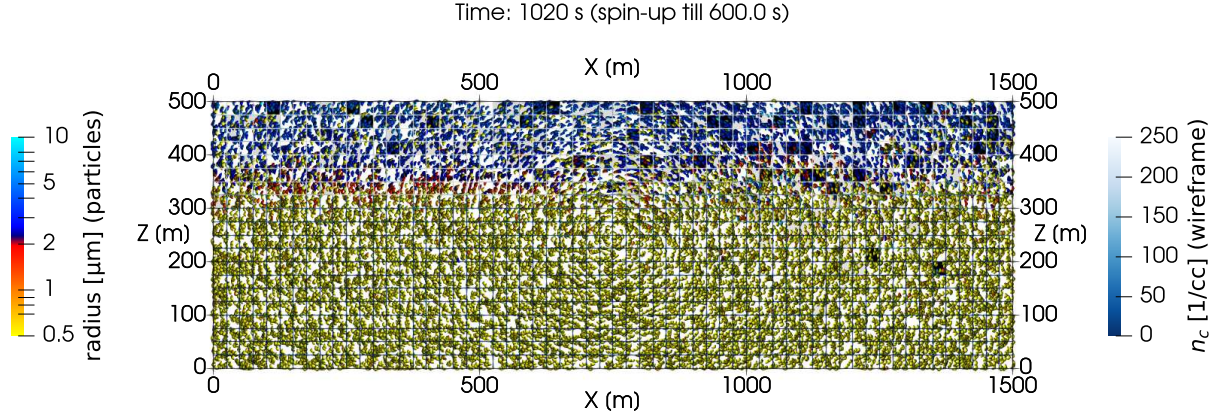


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

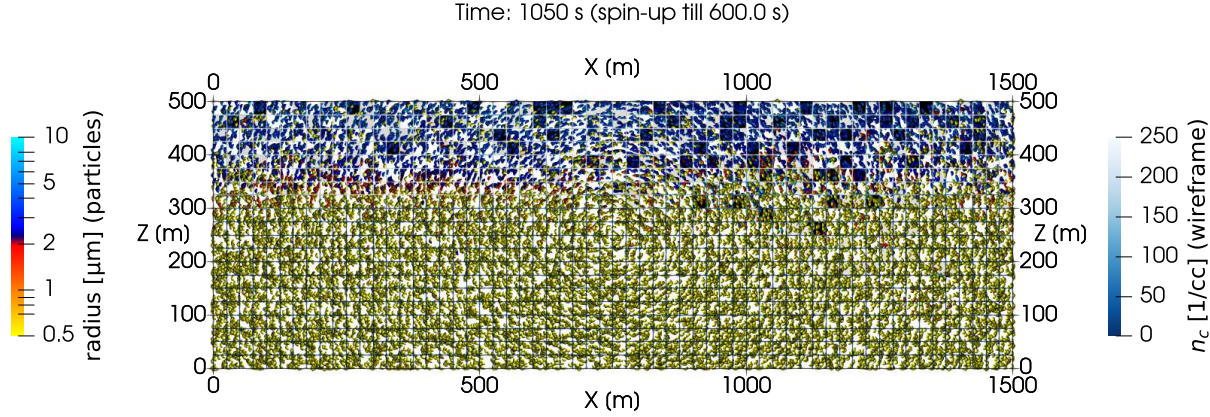


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

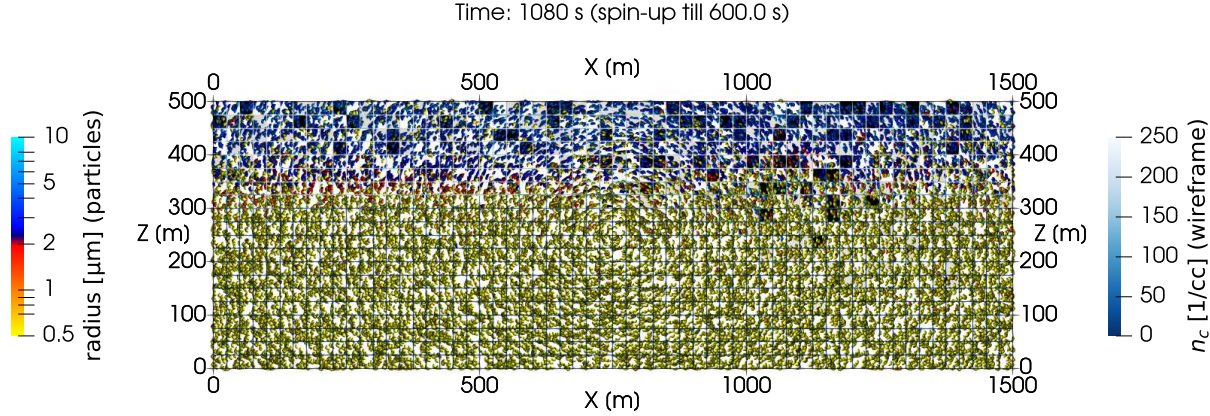


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

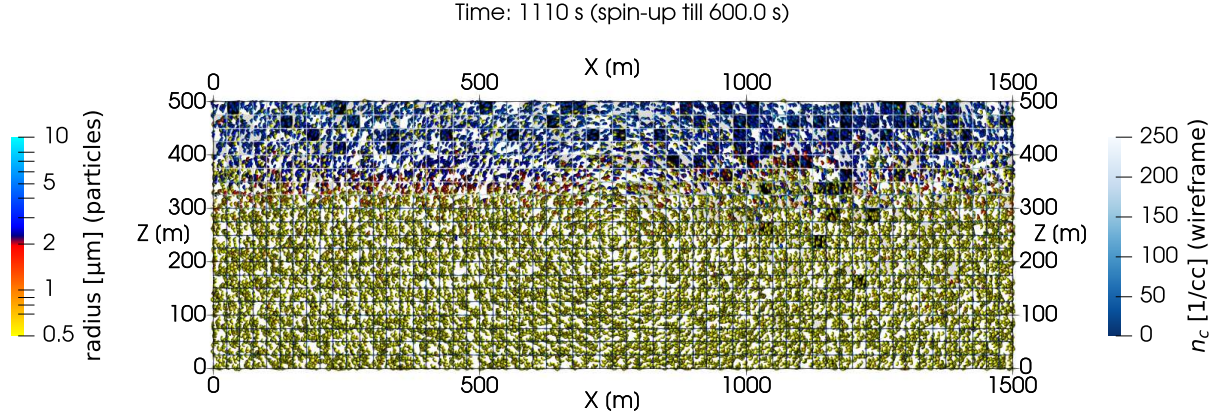


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

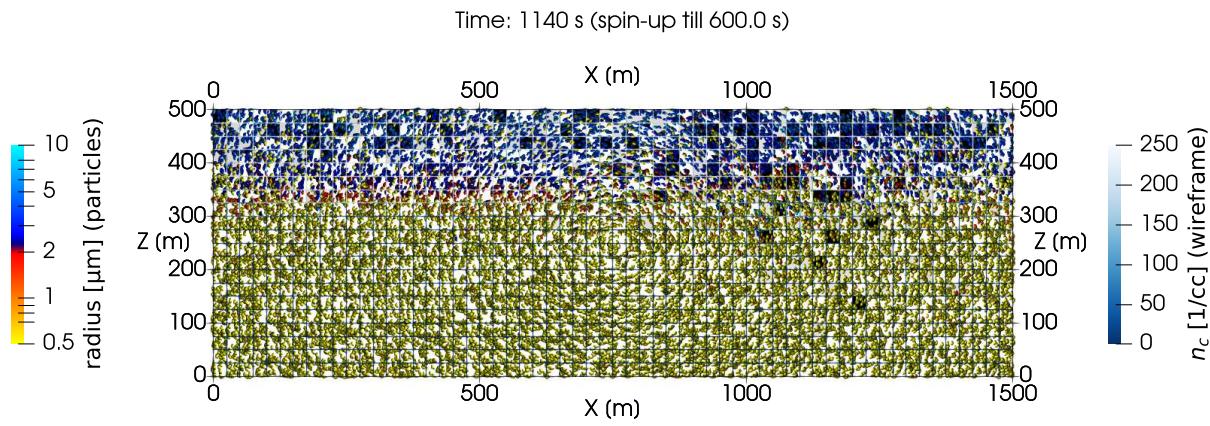


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

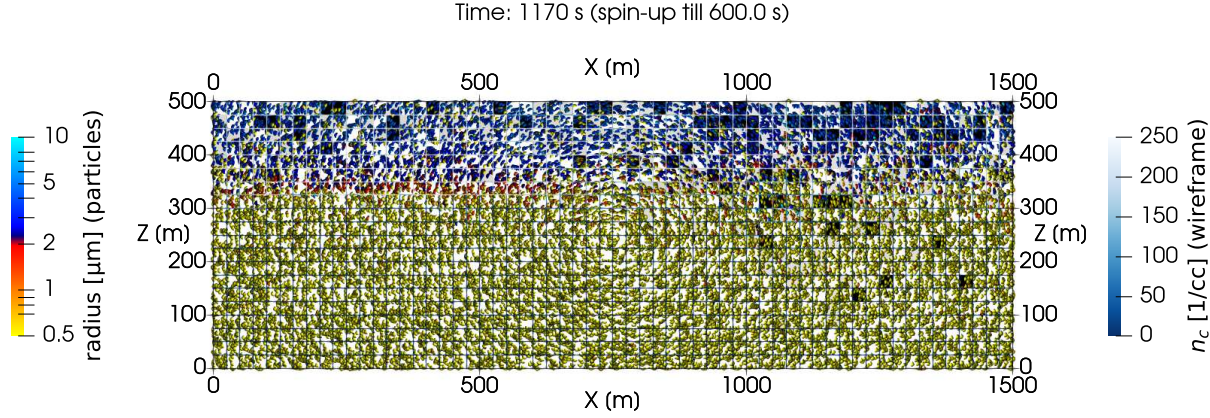


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

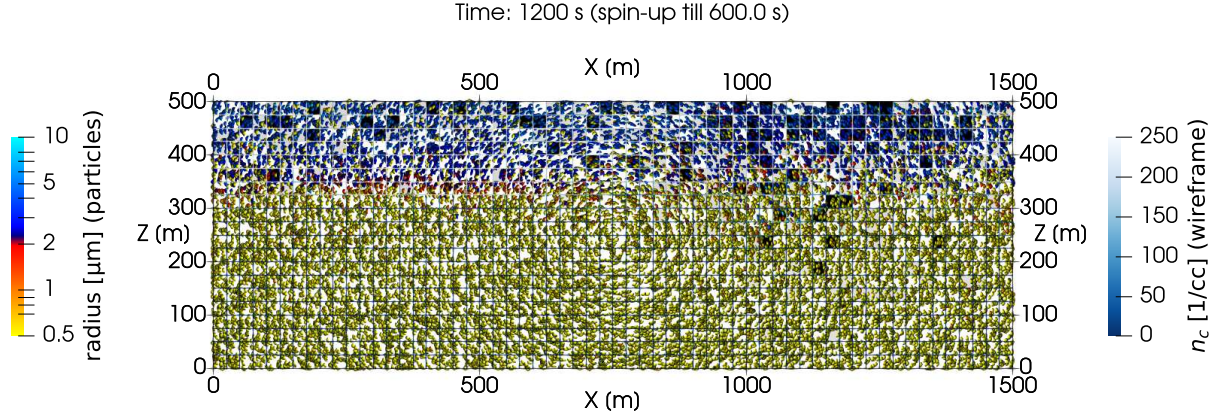


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)

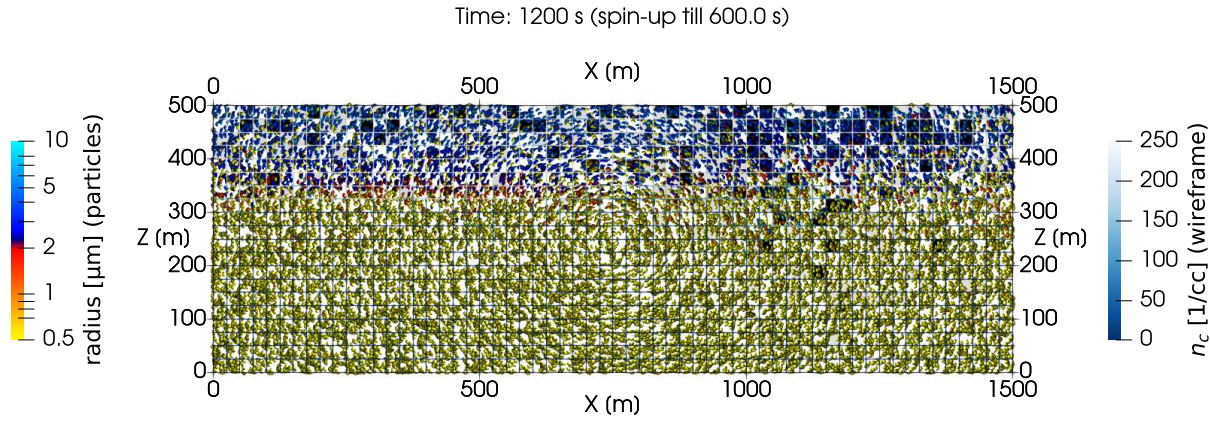


16+16 super-particles/cell for INP-rich + INP-free particles

$N_{\text{aer}} = 300/cc$ (two-mode lognormal) $N_{\text{INP}} = 150/L$ (lognormal, $D_g = 0.74$ μm, $\sigma_g = 2.55$)

spin-up = freezing off; subsequently frozen particles act as tracers

Eulerian transport for PySDM: 2D prescribed-flow framework (Kessler '69)



100% Python, 100% open-source, 100% runs "in the cloud" (Google Colab, jupyterhub, ...)

$\rightsquigarrow$ **actually reproducible by readers & reviewers**

PyMPDATA & Trixi.jl in one notebook (basic 2D advection)

Introduction

Trixi.jl is a numerical simulation framework for conservation laws written in Julia. It is based on the Discontinuous Galerkin (DG) method and for the purpose of this comparison, we will use the StructuredMesh for data representation.

This notebook compares the results of a simple advection equation solved in 2D by PyMPDATA and Trixi.jl. The general flow of the notebook is as follows:

1. define the advection equation and the common settings for both PyMPDATA and Trixi.jl in the JSON file;
2. run the simulation in Trixi.jl and save the results;
3. use Trixi2Vtk to convert the results to a vtk file;
4. reshape the results from Trixi.jl to match the shape of the results from PyMPDATA;
5. run the simulation in PyMPDATA for a bigger nx and ny, to account for the polynomial degree in Trixi.jl;
6. compare the results from PyMPDATA and Trixi.jl;
7. assert that the results are close to each other, this is to ensure that the implementation of PyMPDATA is correct.

To run the notebook, Julia and the following Julia packages are required:

- JSON
- Trixi
- OrdinaryDiffEq
- Trixi2Vtk
- Pkg

In [1]:
```
import sys
if 'google.colab' in sys.modules:
    !pip --quiet install open-atmos-jupyter-utils
    from open_atmos_jupyter_utils import pip_install_on_colab
    pip_install_on_colab('PyMPDATA-examples')
```

In [ ]:
```
if 'google.colab' in sys.modules:
    JULIA_URL = "https://julialang-s3.julialang.org/bin/linux/x64/1.11/julia-1.11.1-linux-x86_64.tar.gz"
    !wget -nv $JULIA_URL -O /tmp/julia.tar.gz
    !tar -x -f /tmp/julia.tar.gz -C /usr/local --strip-components 1
    !rm /tmp/julia.tar.gz
```

plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

Geosci. Model Dev., 8, 1005–1032, 2015
www.geosci-model-dev.net/8/1005/2015/
doi:10.5194/gmd-8-1005-2015

Geoscientific
Model Development
Open Access

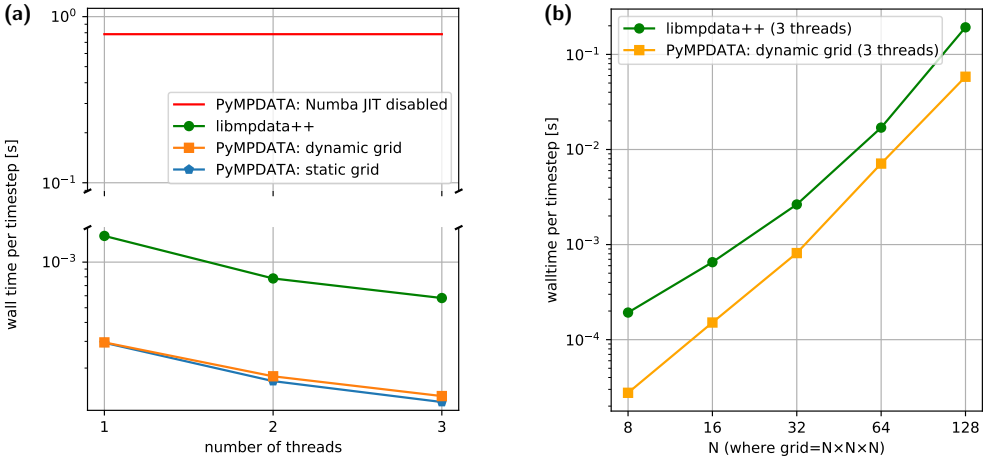
libmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations

A. Jaruga[1], S. Arabas[1], D. Jarecka[1,2], H. Pawlowska[1], P. K. Smolarkiewicz[3], and M. Waruszewski[1]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[2]National Center for Atmospheric Research, Boulder, CO, USA
[3]European Centre for Medium-Range Weather Forecasts, Reading, UK

Correspondence to: A. Jaruga (ajaruga@igf.fuw.edu.pl) and H. Pawlowska (hanna.pawlowska@igf.fuw.edu.pl)

Numba JIT & multi-threading: PyMPDATA vs. libmpdata++ performance



Bartman et al. 2022 (JOSS, doi:10.21105/joss.03896)

plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

introducing Numba-MPI (now a dependency of py-pde)

ScienceDirect

SoftwareX

Volume 28, December 2024, 101897

Original software publication

Numba-MPI v1.0: Enabling MPI communication within Numba/LLVM JIT-compiled Python code

Kacper Derlatka [a 1], Maciej Manna [a 2], Oleksii Bulenok [a 3], David Zwicker [b], Sylwester Arabas [c]

[a] Faculty of Mathematics and Computer Science, Jagiellonian University in Kraków, Poland

[b] Max Planck Institute for Dynamics and Self-Organization, Göttingen, Germany

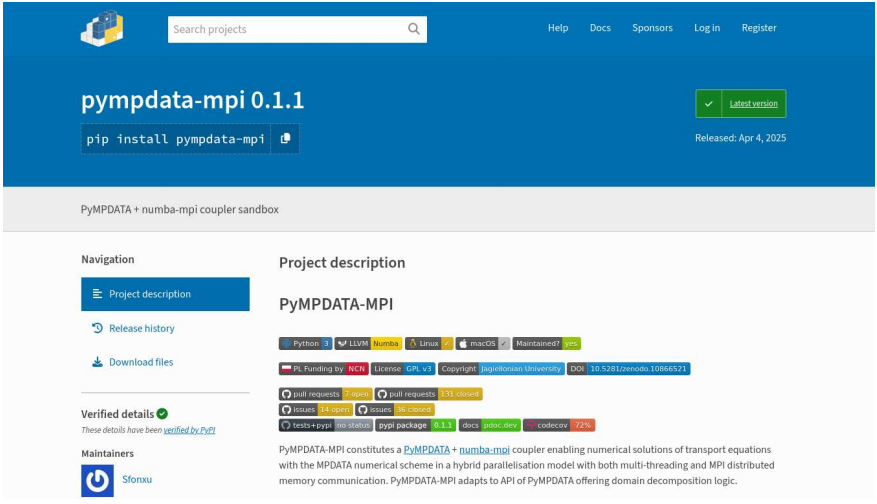
[c] Faculty of Physics and Applied Computer Science, AGH University of Krakow, Poland

https://doi.org/10.1016/j.softx.2024.101897

Abstract

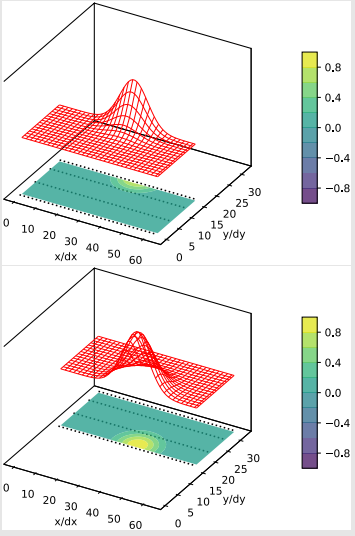
The numba-mpi package offers access to the Message Passing Interface (MPI) routines from Python code that uses the Numba just-in-time (JIT) compiler. As a result, high-performance and multi-threaded Python code may utilize MPI communication facilities without leaving the JIT-compiled code blocks, which is not possible with the mpi4py package, a higher-level Python interface to MPI. For debugging or code-coverage analysis purposes, numba-mpi retains full functionality of the code even if the JIT compilation is disabled.

PyMPDATA-MPI

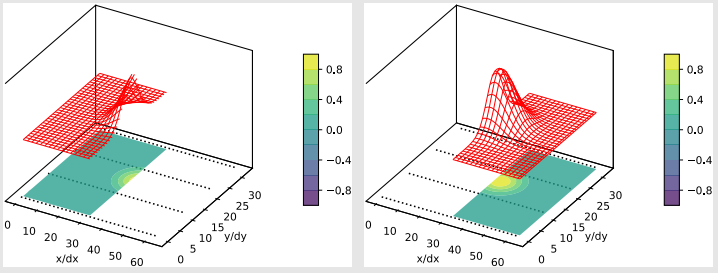


PyMPDATA-MPI: customisable hybrid threading + MPI parallelisation

threading dim = MPI dim



threading dimension ≠ MPI dimension



Derlatka et al. 2024 (SoftwareX, doi:10.1016/j.softx.2024.101897)

plan of the talk

- MPDATA scheme and its implementations
- PyMPDATA: pure-Python just-in-time compiled MPDATA
- PyMPDATA documentation and "examples"
- PyMPDATA performance vs. C++
- MPI, HPC & distributed-memory parallelisation?
- PyMPDATA in teaching (i.e., implemented by students!)

code contributors (CS, math & physics students):

Jakub Banaśkiewicz (UJ), Piotr Bartman (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygieł (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), new contributors very welcome!

code contributors (CS, math & physics students):

Jakub Banaśkiewicz (UJ), Piotr Bartman (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygieł (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), new contributors very welcome!

FNP Foundation for Polish Science

NATIONAL SCIENCE CENTRE POLAND

code contributors (CS, math & physics students):

Jakub Banaśkiewicz (UJ), Piotr Bartman (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygieł (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), new contributors very welcome!

FNP Foundation for Polish Science

NATIONAL SCIENCE CENTRE POLAND

6-month postdoc position at AGH available

code contributors (CS, math & physics students):

Jakub Banaśkiewicz (UJ), Piotr Bartman (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygieł (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), new contributors very welcome!

FNP Foundation for Polish Science

NATIONAL SCIENCE CENTRE POLAND

6-month postdoc position at AGH available

AMS Annual Meeting @Houston (25-29 Jan 2026) session

18th Symposium on Aerosol–Cloud– Climate Interactions
Third Symposium on Cloud Physics

Abstracts are due by **14 August 2025 at 5:00 PM ET**

SUBMIT ABSTRACT

Joint Sessions

- Advances in numerical modeling of aerosol-cloud interactions: moment-, bin- and particle-resolved methods and beyond

code contributors (CS, math & physics students):

Jakub Banaśkiewicz (UJ), Piotr Bartman (UJ), Kacper Derlatka (UJ, Pega), Szymon Drenda (UJ), Adrian Jaśkowiec (AGH), Piotr Karaś (AGH), Norbert Klockiewicz (AGH), Michał Kowalczyk (AGH), Kacper Majchrzak (AGH), Paweł Magnuszewski (AGH), Maciej Manna (UJ, Autodesk), Wojciech Neuman (AGH), Michael Olesik (UJ), Arkadiusz Paterak (AGH), Paulina Pojda (AGH), Wiktor Prosowicz (AGH), Weronika Romaniec (AGH), Paweł Rozwoda (UJ), Michał Sadowski (UJ), Jan Stryszewski (AGH), Michał Szczygieł (AGH), Michał Wroński (AGH), Joanna Wójcicka (AGH), Antoni Zięciak (AGH), Agnieszka Żaba (AGH), new contributors very welcome!

FNP Foundation for Polish Science

NATIONAL SCIENCE CENTRE POLAND

6-month postdoc position at AGH available

AMS Annual Meeting @Houston (25-29 Jan 2026) session

Thank you for your attention!

sylwester.arabas@agh.edu.pl

MPDATA Wikipedia article: contributions welcome!

https://en.wikipedia.org/wiki/Draft:MPDATA

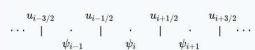
Contents hide

Description of the basic scheme in 1D [edit]

MPDATA is inherently multi-dimensional, and primarily used in computational fluid dynamics where the advective velocities and problem geometries are variable in time. Still, the key idea underlying the MPDATA approach can be conveyed with a basic example of solenoidal stationary flow in one dimension[11] (i.e., $\mathbf{v} = [u]$ constant in time and space), without coordinate transformation ($G = 1$), for the case of homogeneous advection ($R = 0$) of a nonnegative scalar field ($\psi \geq 0$), with the following flux form of the advection equation:

$$\partial_t \psi + \partial_x (u\psi) = 0. \tag{2}$$

Upwind discretisation of the problem on a regular staggered grid with a time step Δt and a grid step Δx, with $n = t/\Delta t$, $i = x/\Delta x$, and the half-integer spatial indices corresponding to grid-cell walls:



can be formulated with:

$$\frac{\psi_i^{n+1} - \psi_i^n}{\Delta t} + \frac{\overbrace{f(\psi_i^n, \psi_{i+1}^n, u_{i+1/2}^n)}^{\text{right-hand wall flux}} - \overbrace{f(\psi_{i-1}^n, \psi_i^n, u_{i-1/2}^n)}^{\text{left-hand wall flux}}}{\Delta x} = 0 \tag{3}$$

with the flux function defined using positive and negative parts of $u_{i\pm1/2}$ as:

$$f(\psi_l, \psi_r, u) = \overbrace{\frac{u + |u|}{2}}^{\text{positive part}} \psi_l + \overbrace{\frac{u - |u|}{2}}^{\text{negative part}} \psi_r. \tag{4}$$

Introducing the non-dimensional Courant number $C = u\Delta t/\Delta x$, the resultant explicit-in-time scheme (referred to as "upwind", "upstream" or "donor-cell"), for a constant C reads: