The Journal of Open Source Software

PyMPDATA v1: Numba-accelerated implementation of MPDATA with examples in Python, Julia and Matlab

Piotr Bartman [1], **Jakub Banaśkiewicz**[1], **Szymon Drenda**[1], **Maciej Manna**[1], **Michael A. Olesik** [1], **Paweł Rozwoda**[1], **Michał Sadowski** [1], **and Sylwester Arabas** [1,2]

1 Jagiellonian University, Kraków, Poland **2** University of Illinois at Urbana-Champaign, IL, USA

DOI: 10.21105/joss.03896

Software

- Review
- Repository
- Archive

Editor: Arfon Smith

Reviewers:

- @Chiil
- @wdeconinck

Submitted: 25 October 2021
Published: 05 September 2022

Statement of need

Convection-diffusion problems arise across a wide range of pure and applied research, in particular in geosciences, aerospace engineering, and financial modelling (for an overview of applications, see, e.g., section 1.1 in Morton (1996)). One of the key challenges in numerical solutions of problems involving advective transport is sign preservation of the advected field (for an overview of this and other aspects of numerical solutions to advection problems, see, e.g., Røed (2019)). The Multidimensional Positive Definite Advection Transport Algorithm (MPDATA) is a robust, explicit-in-time, and sign-preserving solver introduced in Smolarkiewicz (1983) and Smolarkiewicz (1984). MPDATA has been subsequently developed into a family of numerical schemes with numerous variants and solution procedures addressing a diverse set of problems in geophysical fluid dynamics and beyond. For reviews of MPDATA applications and variants, see, e.g., Smolarkiewicz & Margolin (1998) and Smolarkiewicz (2006).

The PyMPDATA project introduced herein constitutes a high-performance and multi-threaded implementation of structured-mesh MPDATA in Python. PyMPDATA is aimed to address several aspects which steepen the learning curve and limit collaborative usage and development of existing C++ (e.g., Jaruga et al., 2015) and Fortran (e.g., Kühnlein et al., 2019) implementations of MPDATA. Performance on par with compiled-language implementations is targeted by employing just-in-time (JIT) compilation using Numba (Lam et al., 2015), which translates Python code into fast machine code using the Low Level Virtual Machine (LLVM, https://llvm.org/) compiler infrastructure (for a discussion of another JIT implementation of MPDATA using PyPy, see Arabas et al., 2014). PyMPDATA is engineered aiming at both performance and usability, the latter encompassing research user's, developer's and maintainer's perspectives. From researcher's perspective, PyMPDATA offers hassle-free installation on Linux, macOS, and Windows. It also eliminates the compilation stage from the perspective of the user. From developer's and maintainer's perspectives, PyMPDATA offers a suite of unit tests, multi-platform continuous integration setup, seamless integration with Python development tools including debuggers, profilers, and code analysers.

Summary

PyMPDATA interface uses NumPy for array-oriented input and output. Usage of PyMPDATA from Julia (https://julialang.org) and Matlab (https://mathworks.com) through PyCall and the built-in Python interoperability tools, respectively, is depicted in the PyMPDATA README file. The appendices of the present paper include Python, Julia and Matlab minimal code snippets covering steps needed to complete a basic PyMPDATA simulation depicted in Fig. (Figure 1)

and based on Fig. 5 from Arabas et al. (2014).

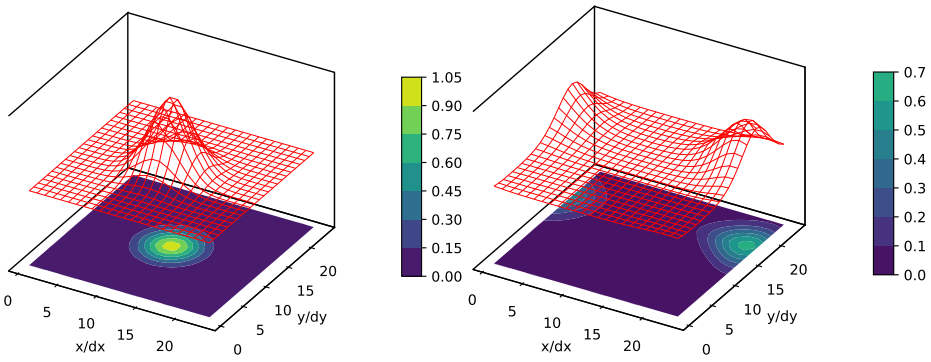

Figure 1: Visualisation of the initial condition (left) and simulation state after 75 timesteps (right) from the 2D simulation for which sample codes are given in the appendices, based on Fig. 5 from Arabas et al. (2014).

As of the current version, `PyMPDATA` supports homogeneous transport in one (1D), two (2D), and three dimensions (3D) using structured meshes, optionally generalised by coordinate transformation (Smolarkiewicz & Clark, 1986; Smolarkiewicz & Margolin, 1993). `PyMPDATA` includes implementation of a subset of `MPDATA` variants, such as the non-oscillatory option (Smolarkiewicz & Grabowski, 1990), the infinite-gauge variant (Margolin & Shashkov, 2006; Smolarkiewicz & Clark, 1986), the divergent-flow option (Margolin & Smolarkiewicz, 1998; Smolarkiewicz, 1984), the double-pass donor cell (DPDC) flavour (Beason & Margolin, 1988; Margolin & Shashkov, 2006; Margolin & Smolarkiewicz, 1998), and the third-order-terms options (Margolin & Smolarkiewicz, 1998). It also features support for integration of Fickian-terms in advection-diffusion problems using the pseudo-transport velocity approach (Smolarkiewicz & Clark, 1986; Smolarkiewicz & Szmelter, 2005).

A companion package named `PyMPDATA-examples` contains a set of Jupyter notebooks, which reproduce results from literature using `PyMPDATA`. These examples are also executed within continuous integration runs. Several of the examples feature comparisons against analytical solution, and these are also included in the test suite of `PyMPDATA`. The `PyMPDATA-examples` README file includes links (badges) offering single-click deployment in the cloud using either the Binder (`https://mybinder.org`) or the Colab (`https://colab.research.google.com`) platforms.

A separate project named `numba-mpi` has been developed to set the stage for future Message Passing Interface (`MPI`) distributed memory parallelism in `PyMPDATA`. The `PyMPDATA`, the `PyMPDATA-examples` and the `numba-mpi` packages are available in the `PyPI` package repository, and installation of these packages reduces to typing `pip install package_name`. Development of all three packages is hosted on `GitHub` at: https://github.com/atmos-cloud-sim-uj/ and continuous integration runs on `Linux`, `macOS` and `Windows` using `GitHub Actions` and `Appveyor` platforms (the latter used for 32-bit runs on Windows). Auto-generated documentation sites built with `pdoc3`
are hosted at https://atmos-cloud-sim-uj.github.io/PyMPDATA/, https://atmos-cloud-sim-uj.github.io/PyMPDATA-examples/, and https://atmos-cloud-sim-uj.github.io/numba-mpi/.

`PyMPDATA` is a free and open-source software released under the terms of the GNU General Public License 3.0 (https://www.gnu.org/licenses/gpl-3.0).

Usage examples

Simulations included in the `PyMPDATA-examples` package are listed below, labelled with the paper reference on which the example setup is based. Each example is annotated with the dimensionality, number of equations constituting the system, and an outline of setup.

- 1D:
  - Smolarkiewicz (2006): single-equation advection-only homogeneous problem with different algorithm options depicted with constant velocity field
  - Arabas & Farhat (2020): single-equation advection-diffusion problem resulting from a transformation of the Black-Scholes equation into either homogeneous or heterogeneous problem for European or American option valuation, respectively
  - Olesik et al. (2022): single-equation advection-only homogeneous problem with coordinate transformation depicting application of `MPDATA` for condensational growth of a population of particles
- 2D:
  - Molenkamp (1968): single-equation homogeneous transport with different algorithm options
  - Jarecka et al. (2015): shallow-water system with three equations representing conservation of mass and two components of momentum (with the momentum equations featuring source terms) modelling spreading under gravity of a three-dimensional elliptic drop on a two-dimensional plane
  - Williamson & Rasch (1989): advection on a spherical plane depicting transformation to spherical coordinates
  - Shipway & Hill (2012): coupled system of water vapour mass (single spatial dimension) and water droplet number conservation (spatial and spectral dimensions) with the latter featuring source term modelling activation of water droplets on aerosol particles, coordinate transformation used for representation of air density profile
- 3D:
  - Smolarkiewicz (1984): homogeneous single-equation example depicting revolution of a spherical signal in a constant angular velocity rotational velocity field

In addition, `PyMPDATA` is used in a two-dimensional setup in of the examples in the sister PySDM package (Bartman et al., 2022).

Implementation highlights

In 2D and 3D simulations, domain-decomposition is used for multi-threaded parallelism. Domain decomposition is performed along the outer dimension only and is realised using the `numba.prange()` functionality.

`PyMPDATA` design features a custom-built multi-dimensional Arakawa-C staggered grid layer, allowing to concisely represent multi-dimensional stencil operations on both scalar and vector fields. The grid layer is built on top of `NumPy`'s `ndarrays` (using "C" ordering) using the `Numba`'s `@njit` functionality for high-performance multi-threaded array traversals. The array-traversal layer enables to code once for multiple dimensions (i.e. one set of `MPDATA` formulae for 1D, 2D, and 3D), and automatically handles (if needed) any halo-filling logic related to boundary conditions.

The `Numba`'s deviation from `Python` semantics rendering closure variables as compile-time constants is extensively exploited within `PyMPDATA` code base enabling the just-in-time compilation to benefit from information on domain extents, algorithm variant used and problem characteristics (e.g., coordinate transformation used, or lack thereof).

In general, the numerical and concurrency aspects of `PyMPDATA` implementation follow the

libmpdata++ open-source C++ implementation of MPDATA (Jaruga et al., 2015).

Performance

A basic performance analysis is carried out comparing PyMPDATA execution (wall) times: (i) with or without Numba JIT, as well as (ii) comparing performance against the C++ implementation of MPDATA in libmpdata++. The tests are carried out using a 3D simulation based on the revolving sphere case from Smolarkiewicz (1984) (figs. 13-16 therein) as used in Jaruga et al. (2015) (fig. 13 therein). The simulation setup involves solution of a homogeneous advection problem in a cubic domain with a non-divergent rotational flow. Here, for simplicity, all simulations are carried out for 64 timesteps, and the measured wall-time is divided by the number of steps and reported as wall-time per timestep. In all reported runs, both for libmpdata++ and PyMPDATA, two corrective iterations of MPDATA are used, and the basic flavour of the algorithm is employed. Wall-time measurements are carried out using the timers built-in into libmpdata++, and using Python's `timeit` routines, respectively. Timing applies to integration only excluding initial condition evaluation or output handling. Simulations are repeated four times and the minimal value is reported in order to filter out JIT-compilation and caching overhead and to minimise thread-schedulling differences. For PyMPDATA runs with Numba JIT disabled, the number of repetitions is reduced from four to two. Simulations are carried out on one, two or three threads on a machine with four physical cores.

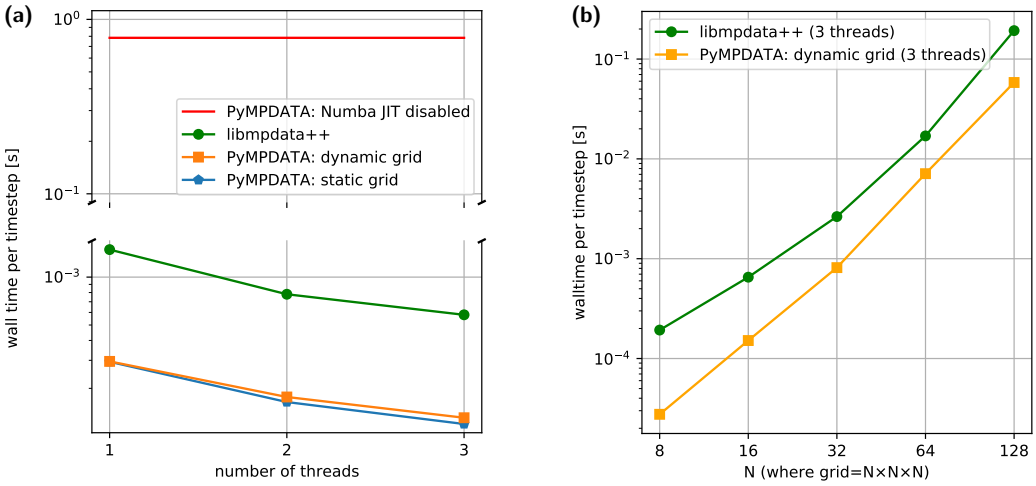

Figure 2: Comparison of wall-time measurements results for a 3D simulation using PyMPDATA with JIT disabled (red line) and enabled (connected points) corroborated against timings of analogous simulation performed with libmpdata++. Panel (a) presents scaling with the number of threads used, for the case of 16 by 16 by 16 domain. Panel (b) depicts scaling with domain size for simulations using three threads.

Figure Figure 2 (a) depicts wall-times measured with a domain of 16 by 16 by 16, and for: PyMPDATA with Numba JIT disabled (red line), libmpdata++ (green connected points), and PyMPDATA with JIT enabled for both dynamic grid (i.e., grid extents specified at run-time, plotted with orange connected points) and static grid (i.e., grid extents specified ahead of JIT compilation, blue connected points). First, an over three orders of magnitude speedup is depicted comparing wall-times with JIT disabled and enabled. Comparison of PyMPDATA and libmpdata++ reveals comparable performance and scaling with number of threads with consistently shorter wall-times for PyMPDATA, and a slight further improvement when switching from dynamic to static grid.

Figure Figure 2 (b) depicts wall-time dependence on the domain size for the case of three

threads, and confirms that the observed higher performance of PyMPDATA as compared with libmpdata++ can be observed over a range of domain sizes starting from 16 by 16 by 16 up to 128 by 128 by 128. While more comprehensive tests and analyses would be needed to identify the cause of this superior performance, two possible factors include overhead from employment of the Blitz++ library in libmpdata++ as well as explicit (potentially superfluous) halo-filling triggers in libmpdata++ as opposed to on-demand halo-filling design implemented in PyMPDATA. Noteworthy, as reported in Jaruga et al. (2015) (section 7 therein), for the very test case discussed herein, and for small grid sizes (59 by 59 by 59), libmpdata++ had up to five times longer execution times compared with the original FORTRAN-77 serial implementaion of MPDATA. This indicates that the measured performance of PyMPDATA approaches the performance of the original FORTRAN-77 implementation, at the same time offering multi-threading concurrency, hiding the compilation and linking stages from the user, and featuring interoperability with the Python package ecosystem.

Appendix P: Python sample code

```python
import numpy as np
from PyMPDATA import Options, ScalarField, VectorField, Stepper, Solver
from PyMPDATA.boundary_conditions import Periodic

options = Options(n_iters=2)
nx, ny = 24, 24
Cx, Cy = -.5, -.25
halo = options.n_halo

xi, yi = np.indices((nx, ny), dtype=float)
advectee = ScalarField(
  data=np.exp(
    -(xi+.5-nx/2)**2 / (2*(nx/10)**2)
    -(yi+.5-ny/2)**2 / (2*(ny/10)**2)
  ),
  halo=halo,
  boundary_conditions=(Periodic(), Periodic())
)
advector = VectorField(
  data=(np.full((nx + 1, ny), Cx), np.full((nx, ny + 1), Cy)),
  halo=halo,
  boundary_conditions=(Periodic(), Periodic())
)
stepper = Stepper(options=options, grid=(nx, ny))
solver = Solver(stepper=stepper, advectee=advectee, advector=advector)
solver.advance(n_steps=75)
state = solver.advectee.get()
```

Appendix J: Julia sample code

```julia
using PyCall
Options = pyimport("PyMPDATA").Options
ScalarField = pyimport("PyMPDATA").ScalarField
VectorField = pyimport("PyMPDATA").VectorField
Stepper = pyimport("PyMPDATA").Stepper
Solver = pyimport("PyMPDATA").Solver
Periodic = pyimport("PyMPDATA.boundary_conditions").Periodic
```

```
options = Options(n_iters=2)
nx, ny = 24, 24
Cx, Cy = -.5, -.25
idx = CartesianIndices((nx, ny))
halo = options.n_halo
advectee = ScalarField(
    data=exp.(
        -(getindex.(idx, 1) .- .5 .- nx/2).^2 / (2*(nx/10)^2)
        -(getindex.(idx, 2) .- .5 .- ny/2).^2 / (2*(ny/10)^2)
    ),
    halo=halo,
    boundary_conditions=(Periodic(), Periodic())
)
advector = VectorField(
    data=(fill(Cx, (nx+1, ny)), fill(Cy, (nx, ny+1))),
    halo=halo,
    boundary_conditions=(Periodic(), Periodic())
)

stepper = Stepper(options=options, grid=(nx, ny))
solver = Solver(stepper=stepper, advectee=advectee, advector=advector)
solver.advance(n_steps=75)
state = solver.advectee.get()
```

Appendix M: Matlab sample code

```
Options = py.importlib.import_module('PyMPDATA').Options;
ScalarField = py.importlib.import_module('PyMPDATA').ScalarField;
VectorField = py.importlib.import_module('PyMPDATA').VectorField;
Stepper = py.importlib.import_module('PyMPDATA').Stepper;
Solver = py.importlib.import_module('PyMPDATA').Solver;
Periodic = ...
    py.importlib.import_module('PyMPDATA.boundary_conditions').Periodic;

options = Options(pyargs('n_iters', 2));
nx = int32(24);
ny = int32(24);

Cx = -.5;
Cy = -.25;

[xi, yi] = meshgrid(double(0:1:nx-1), double(0:1:ny-1));

halo = options.n_halo;
advectee = ScalarField(pyargs(...
    'data', py.numpy.array(exp( ...
        -(xi+.5-double(nx)/2).^2 / (2*(double(nx)/10)^2) ...
        -(yi+.5-double(ny)/2).^2 / (2*(double(ny)/10)^2) ...
    )), ...
    'halo', halo, ...
    'boundary_conditions', py.tuple({Periodic(), Periodic()}) ...
));
advector = VectorField(pyargs(...
    'data', py.tuple({ ...
```

```
        Cx * py.numpy.ones(int32([nx+1 ny])), ...
        Cy * py.numpy.ones(int32([nx ny+1])) ...
     }), ...
    'halo', halo, ...
    'boundary_conditions', py.tuple({Periodic(), Periodic()}) ...
));

stepper = Stepper(pyargs(...
  'options', options, ...
  'grid', py.tuple({nx, ny}) ...
));
solver = Solver(...
    pyargs('stepper', ...
    stepper, 'advectee', ...
    advectee, 'advector', ...
    advector ...
));
solver.advance(pyargs('n_steps', 75));
state = solver.advectee.get();
```

Author contributions

PB had been the architect of `PyMPDATA` with SA taking the role of main developer and maintainer over the time. MO participated in the package core development and led the development of the condensational-growth example, which was the basis of his MSc thesis. JB contributed the DPDC algorithm variant handling. SD contributed the advection-diffusion example. MM contributed to the `numba-mpi` package. PR contributed the shallow-water example. MS contributed the advection-on-a-sphere example. The paper was composed by SA and is based on the contents of the README files of the `PyMPDATA`, `PyMPDATA-examples`, and `numba-mpi` packages.

Acknowledgements

Development of `PyMPDATA` has been carried out within the POWROTY/REINTEGRATION programme of the Foundation for Polish Science, co-financed by the European Union under the European Regional Development Fund (POIR.04.04.00-00-5E1C/18).

References

Arabas, S., & Farhat, A. (2020). Derivative pricing as a transport problem: MPDATA solutions to Black–Scholes-type equations. *J. Comput. Appl. Math.*, *373*. https://doi.org/10.1016/j.cam.2019.05.023

Arabas, S., Jarecka, D., Jaruga, A., & Fijałkowski, M. (2014). Formula translation in Blitz++, NumPy and modern Fortran: A case study of the language choice tradeoffs. *Sci. Prog.*, *22*. https://doi.org/10.3233/SPR-140379

Bartman, P., Bulenok, O., Górski, K., Jaruga, A., G., Ł., Olesik, M., Piasecki, B., Singer, C. E., Talar, A., & Arabas, S. (2022). PySDM v1: Particle-based cloud modelling package for warm-rain microphysics and aqueous chemistry. *J. Open Source Softw.*, *7*(72). https://doi.org/10.21105/joss.03219

Beason, C. W., & Margolin, L. G. (1988). DPDC (double-pass donor cell): A second-order monotone scheme for advection. *Fifth Nuclear Code Developers' Conference*. https:

//www.osti.gov/servlets/purl/7049237

Jarecka, D., Jaruga, A., & Smolarkiewicz, P. K. (2015). A spreading drop of shallow water. *J. Comp. Phys.*, *289*. https://doi.org/10.1016/j.jcp.2015.02.003

Jaruga, A., Arabas, S., Jarecka, D., Pawlowska, H., Smolarkiewicz, P. K., & Waruszewski, M. (2015). libmpdata++ 1.0: A library of parallel MPDATA solvers for systems of generalised transport equations. *Geosci. Model Dev.*, *8*. https://doi.org/10.5194/gmd-8-1005-2015

Kühnlein, C., Deconinck, W., Klein, R., Malardel, S., Piotrowski, Z. P., Smolarkiewicz, P. K., Szmelter, J., & Wedi, N. P. (2019). FVM 1.0: A nonhydrostatic finite-volume dynamical core for the IFS. *Geosci. Model Dev.*, *12*. https://doi.org/10.5194/gmd-12-651-2019

Lam, S. K., Pitrou, A., & Seibert, S. (2015). Numba: A LLVM-based Python JIT compiler. *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. https://doi.org/10.1145/2833157.2833162

Margolin, L. G., & Shashkov, M. (2006). MPDATA: Gauge transformations, limiters and monotonicity. *Int. J. Numer. Methods Fluids*, *50*(10). https://doi.org/10.1002/fld.1070

Margolin, L. G., & Smolarkiewicz, P. K. (1998). Antidiffusive velocities for multipass donor cell advection. *SIAM J. Sci. Comput.*, *20*(3). https://doi.org/10.1137/S106482759324700X

Molenkamp, C. R. (1968). Accuracy of finite-difference methods applied to the advection equation. *J. Appl. Meteorol. Climatol.*, *7*. https://doi.org/10.1175/1520-0450(1968)007%3C0160:AOFDMA%3E2.0.CO;2

Morton, K. W. (1996). *Numerical solution of convection-diffusion problems*. CRC Press. https://doi.org/10.1201/9780203711194

Olesik, M., Banaśkiewicz, J., Bartman, P., Baumgartner, M., Unterstrasser, S., & Arabas, S. (2022). On numerical broadening of particle-size spectra: A condensational growth study using PyMPDATA 1.0. *Geosci. Model Dev.*, *15*. https://doi.org/10.5194/gmd-15-3879-2022

Røed, L. P. (2019). Advection problem. In *Atmospheres and oceans on computers*. https://doi.org/10.1007/978-3-319-93864-6_5

Shipway, B. J., & Hill, A. A. (2012). Diagnosis of systematic differences between multiple parametrizations of warm rain microphysics using a kinematic framework. *Q. J. R. Meteorol. Soc.*, *138*(669). https://doi.org/10.1002/qj.1913

Smolarkiewicz, P. K. (1983). A simple positive definite advection scheme with small implicit diffusion. *Mon. Weather Rev.*, *111*. https://doi.org/10.1175/1520-0493(1983)111%3C0479:ASPDAS%3E2.0.CO;2

Smolarkiewicz, P. K. (1984). A fully multidimensional positive definite advection transport algorithm with small implicit diffusion. *J. Comp. Phys.*, *54*. https://doi.org/10.1016/0021-9991(84)90121-9

Smolarkiewicz, P. K. (2006). Multidimensional positive definite advection transport algorithm: An overview. *Int. J. Numer. Methods Fluids*, *50*(10). https://doi.org/10.1002/fld.1071

Smolarkiewicz, P. K., & Clark, T. L. (1986). The multidimensional positive definite advection transport algorithm: Further development and applications. *J. Comp. Phys.*, *67*. https://doi.org/10.1016/0021-9991(86)90270-6

Smolarkiewicz, P. K., & Grabowski, W. W. (1990). The multidimensional positive definite advection transport algorithm: Nonoscillatory option. *J. Comp. Phys.*, *86*. https://doi.org/10.1016/0021-9991(90)90105-A

Smolarkiewicz, P. K., & Margolin, L. G. (1993). On forward-in-time differencing for fluids: Extension to a curvilinear framework. *Mon. Weather Rev.*, *121*. https://doi.org/10.1175/1520-0493(1993)121%3C1847:OFITDF%3E2.0.CO;2

Smolarkiewicz, P. K., & Margolin, L. G. (1998). MPDATA: A finite-difference solverfor geophysical flows. *J. Comp. Phys.*, *140*. https://doi.org/10.1006/jcph.1998.5901

Smolarkiewicz, P. K., & Szmelter, J. (2005). MPDATA: An edge-based unstructured-grid formulation. *J. Comp. Phys.*, *206*(2). https://doi.org/10.1016/j.jcp.2004.12.021

Williamson, D. L., & Rasch, P. J. (1989). Two-dimensional semi-Lagrangian transport with shape-preserving interpolation. *Mon. Weather Rev.*, *117*. https://doi.org/10.1175/1520-0493(1989)117%3C0102:TDSLTW%3E2.0.CO;2