Manuscript prepared for Geosci. Model Dev.
with version 2014/07/29 7.12 Copernicus papers of the LaTeX class copernicus.cls.
Date: 10 April 2015

libcloudph++ 1.0: single-moment bulk, double-moment bulk, and particle-based warm-rain microphysics library in C++

S. Arabas[1], **A. Jaruga**[1], **H. Pawlowska**[1], **and W. W. Grabowski**[2,3]

[1]Institute of Geophysics, Faculty of Physics, University of Warsaw, Warsaw, Poland
[2]National Center for Atmospheric Research (NCAR), Boulder, Colorado, USA
[3]Affiliate Professor at the University of Warsaw, Warsaw, Poland

Correspondence to: S. Arabas (sarabas@igf.fuw.edu.pl)
and H. Pawlowska (hanna.pawlowska@igf.fuw.edu.pl)

Abstract. This paper introduces a **lib**rary of algorithms for representing **cloud** micro**ph**ysics in numerical models. The library is written in C++, hence the name *libcloudph++*. In the current release, the library covers three warm-rain schemes: the single- and double-moment bulk schemes, and the particle-based scheme with Monte-Carlo coalescence. The three schemes are intended for modelling frameworks of different dimensionality and complexity, ranging from parcel models to multi-dimensional cloud-resolving (e.g. large-eddy) simulations. A two-dimensional prescribed-flow framework is used in the paper to illustrate the library features. The *libcloudph++* and all its mandatory dependencies are free and open-source software. The Boost.units library is used for zero-overhead dimensional analysis of the code at compile time. The particle-based scheme is implemented using the Thrust library that allows to leverage the power of graphics processing units (GPU), retaining the possibility to compile the unchanged code for execution on single or multiple standard processors (CPUs). The paper includes complete description of the programming interface (API) of the library and a performance analysis including comparison of GPU and CPU set-ups.

1 Introduction

Representation of cloud processes in numerical models is crucial for weather and climate prediction. Taking climate modelling as an example, one may learn that numerous distinct modelling systems are designed in similar ways, sharing not only the concepts but also the implementations of some of their components (Pennell and Reichler, 2010). This creates a perfect opportunity for code reuse which is one of the key "best practices" for scientific computing (Wilson et al., 2014, Sect. 6). The reality, however, is that the code to be shared is often "transplanted" from one model to another (Easterbrook and Johns, 2009, Sect. 4.6) rather than reused in a way enabling the users to benefit from ongoing development and updates of the shared code. From the authors' experience, this practise is not uncommon in development of limited-area models as well (yet, such software-engineering issues are rarely the subject of discussion in literature). As a consequence, there exist multiple implementations of the same algorithms but it is difficult to dissect and attribute the differences among them. Avoiding "transplants" in the code is not easy, as numerous software projects in atmospheric modelling feature monolithic design that hampers code reuse.

This brings us to the conclusion that there is a potential demand for a library-type cloud-microphysics software package that could be readily reused and that would enable its users to easily benefit from developments of other researchers (by gaining access to enhancements, corrections, or entirely new schemes). Library approach would not only facilitate collaboration, but also reduce development time and maintenance effort by imposing a separation of cloud microphysics logic from other model components such as the dynamical core or the parallelisation logic. Such strict separation of concerns is also a prerequisite for genuine software testing.

Popularity of several geoscientific-modelling software packages that offer shared-library functionality suggests soundness of such approach – e.g., libRadtran (Mayer and Kylling, 2005) and CLUBB (Golaz et al., 2002), cited nearly 350 and 100 times, respectively.

The motivation behind the development of the libclouph++ library introduced herein is twofold. First, we intend to exemplify the possibilities of library-based code reuse in the context of cloud modelling. Second, in the long run, we intend to offer the community a range of tools applicable for research on some of the key topics in atmospheric science such as the interactions between aerosol, clouds and precipitation – phenomena that still pose significant challenges for the existing tools and methodologies (Stevens and Feingold, 2009).

The library can be used in simulation frameworks of different dimensionality, different dynamical cores, different parallelisation strategies, and in principle models written in different programming languages. Presented library is written in C++, a choice motivated by the availability of high-performance object-oriented libraries and the built-in "template" mechanism. C++ templates allow the implemented algorithms not to be bound to a single data type, single array dimensionality, or single hardware type (e.g. CPU/GPU choice). The library code and documentation are released as free (meaning both gratis & libre) and open-source software – a prerequisite for use in auditable and reproducible research (Morin et al., 2012; Ince et al., 2012).

Openness, together with code brevity and documentation, are also crucial for enabling the users not to treat the library as a "black box". While a self-contained package with well-defined interface is black-box approach compatible, the authors encourage users to inspect and test the code.

Modelling of atmospheric clouds and precipitation employs computational techniques for particle-laden flows. These are divided into Eulerian and Lagrangian approaches (see e.g. Crowe et al., 2012, Chapter 8). In the Eulerian approach, the cloud and precipitation properties are assumed to be continuous in space, like those of a fluid. In the Lagrangian approach, the so-called computational particles are tracked through the model domain. Information associated with those particles travels along their trajectories. The local properties of a given volume are diagnosed by taking into account the properties of particles contained within it. The Eulerian approach is well suited for modelling transport of gaseous species in the atmosphere and is the most common choice for modelling atmospheric flows. This is why most cloud microphysics models are built using the Eulerian concept (Straka, 2009, e.g chapter 9.1). However, it is the Lagrangian approach that is particularly well suited for dilute flows such as those of cloud droplets and rain drops in the atmosphere.

In the current release, *libcloudph++* is equipped with implementations of three distinct models of cloud microphysics. All three belong to the so-called warm-rain class of schemes, meaning they cover representation of processes leading to formation of rain but they do not cover representation of the ice phase (snow, hail, graupel, etc.). The so-called single-moment bulk and double-moment bulk schemes described in Sects. 3 and 4 belong to the Eulerian class of methods. In Sect. 5, a coupled Eulerian–Lagrangian particle-based scheme is presented. In the particle-based scheme, Lagrangian tracking is used to represent the dispersed phase (atmospheric aerosol, cloud droplets, rain drops), while the continuous phase (moisture, heat) is represented with the Eulerian approach. Description of each of the three schemes includes:

- discussion of the key assumptions,
- formulation of the scheme,
- definition of the programming interface (API),
- overview of the implementation,
- example results.

The particle-based scheme, being a novel approach to modelling clouds and precipitation, is discussed in more detail than the bulk schemes.

Description of the programming interface of *libcloudph++* includes C++ code listings of all data-structure definitions and function signatures needed to use the library. In those sections, C++ nomenclature is used without introduction (for reference, see e.g. Brokken, 2013, that covers the C++11 version of the language used in the presented code). Sections covering scheme formulation feature cloud-modelling nomenclature which is briefly introduced in Appendix A.

The library is equipped with Python bindings allowing to use all of *libcloudph++* features from the Python programming language. The bindings are described in a separate technical note (Jarecka et al., 2015) which includes also an example solution for interfacing *libcloudph++* from Fortran using the Python bindings.

The paper is structured as follows. Formulation of an example modelling framework is presented in Sect. 2. The three implemented schemes are described in Sects. 3–5. Section 6 presents a performance evaluation of all three schemes. Section 7 provides a summary of the key features of *libcloudph++*.

Appendix A contains an outline of governing equations for moist atmospheric flow. Appendix B contains a list of symbols used throughout the text. Appendix C covers the description of an example program based on *libcloudph++* that was developed to perform the simulations presented throughout the text.

The *libcloudph++* and the program used to generate all results presented in the paper are released as free and open-source software – see the section on code availability at the end of the paper.

2 Example framework

Being a library, *libcloudph++* does not constitute a complete modelling system. It is a set of reusable software components

that need to be coupled with a model representing air motion. In this section, we describe a simple example framework in which the library may be used. The three following subsections cover the description of a 2-D kinematic flow model, a set-up including initial conditions, and a conceptual numerical solver. Example results obtained with these simulation components are presented alongside the description of microphysics schemes in Sects. 3, 4 and 5.

A simple 2-D kinematic framework allows, and limits, one to study cloud microphysical processes decoupled from cloud dynamics. In fact, the differences between simulations when feedback on the dynamics is taken out can lead to a better understanding of the role of flow dynamics (e.g. Slawinska et al., 2009). Such an approach results in a computationally cheap, yet still insightful, set-up of potential use in: (i) development and testing of cloud-processes parameterisations for larger scale models, (ii) studying such processes as cloud processing of aerosols; and (iii) developing remote-sensing retrieval procedures involving detailed treatment of cloud microphysics.

2.1 2-D kinematic flow model

The flow model formulation is inspired by the 2-D framework described in Szumowski et al. (1998), Morrison and Grabowski (2007) and Rasinski et al. (2011). The primary assumption is that the dry-air density does not change in time (here, a vertical profile $\rho_\mathrm{d}(z)$ is used) which allows to prescribe the 2-D velocity field using a streamfunction:

$$\begin{cases} \rho_\mathrm{d} \cdot u = -\partial_z \psi \\ \rho_\mathrm{d} \cdot w = \partial_x \psi \end{cases} \tag{1}$$

where $\psi = \psi(x, z; t)$ is the streamfunction and u and w denote horizontal and vertical components of the velocity field $\boldsymbol{u}$.

One may notice that the stationarity of the dry-air density field, together with phase-change-related variations in time of the temperature and the water vapour mixing ratio, imply time variations of the pressure profile. The deviations from the initial (hydrostatic) profile are insignificant, though.

2.2 8th ICMW VOCALS set-up

Example results presented in the following sections are based on a modelling set-up designed for the 8th International Cloud Modelling Workshop (ICMW, Muhlbauer et al., 2013, case 1). It was designed as the simplest scenario applicable for benchmarking models representing aerosol processing by clouds. The cloud depth and aerosol characteristics are chosen to mimic a drizzling stratocumulus cloud.

The definition of the streamfunction $\psi(x, z)$ is the same as in Rasinski et al. (2011, Eq. 2):

$$\psi(x, z) = -w_\mathrm{max} \frac{X}{\pi} \sin\left(\pi \frac{z}{Z}\right) \cos\left(2\pi \frac{x}{X}\right) \tag{2}$$

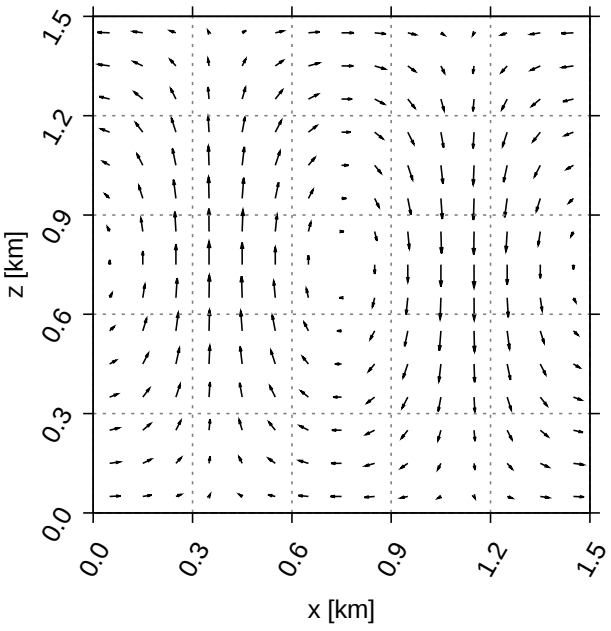

Figure 1. The constant-in-time 2-D velocity field used in the presented simulations. See discussion of Eqs. (1) and (2).

with $w_\mathrm{max} = 0.6\,\mathrm{m\,s^{-1}}$, domain width $X = 1.5\,\mathrm{km}$ and domain height $Z = 1.5\,\mathrm{km}$. The resulting velocity field (depicted in Fig. 1) mimics an eddy spanning the whole domain, and thus covering an updraught and a downdraught region. The domain is periodic in horizontal direction. To maintain flow incompressibility up to round-off error, velocity components (cf. Eq. 1) are derived from Eq. (2) using numerical differentiation formulæ for a given grid type (Arakawa-C grid is used in the examples presented in the paper).

The initial profiles of liquid-water potential temperature θ_l and the total water mixing ratio r_t are defined as constant with altitude ($\theta_\mathrm{l} = 289\,\mathrm{K}$; $r_\mathrm{t} = 7.5\,\mathrm{g\,kg^{-1}}$). The initial air-density profile corresponds to the hydrostatic equilibrium with a pressure of $1015\,\mathrm{hPa}$ at the bottom of the domain. This results in supersaturation in the upper part of the domain, where a cloud deck is formed in the simulations.

The domain is assumed to contain aerosol particles. Their dry size spectrum is a bi-modal log-normal distribution:

$$N(r_\mathrm{d}) = \sum_m \frac{N_m}{\sqrt{2\,\pi}\ln(\sigma_m)} \frac{1}{r_\mathrm{d}} \exp\left[-\left(\frac{\ln(\frac{r_\mathrm{d}}{r_m})}{\sqrt{2}\ln(\sigma_m)} \right)^2 \right] \tag{3}$$

with the following parameters (values based on the VOCALS campaign measurements, Allen et al., 2011, Table 4):

$$\sigma_1 = 1.4;\ \ d_1 = 0.04\,\mu\mathrm{m};\ \ N_1 = 60\,\mathrm{cm^{-3}}$$
$$\sigma_2 = 1.6;\ \ d_2 = 0.15\,\mu\mathrm{m};\ \ N_2 = 40\,\mathrm{cm^{-3}}$$

where $\sigma_{1,2}$ is the geometric standard deviation, $d_{1,2} = 2 \cdot r_{1,2}$ is the mode diameter and $N_{1,2}$ is the particle concentration at standard conditions ($T = 20\,^\circ\mathrm{C}$ and $p = 1013.25\,\mathrm{hPa}$). This

corresponds to a vertical gradient of concentration due to the vertical gradient of air density, and a gradual shift towards larger sizes of the wet particle spectrum due to vertical gradient of relative humidity. Both modes of the distribution are assumed to be composed of ammonium sulphate.

In the examples presented in this paper, the model was initialised with $\theta = \theta_\mathrm{l}$ and $r_\mathrm{v} = r_\mathrm{t}$ (i.e., no condensed water). To avoid unrealistic supersaturation, an arbitrary limit of $5\,\%$ (RH $= 1.05$) was imposed when evaluating drop growth equation during the spin-up.

To maintain steady mean temperature and moisture profiles (i.e. to compensate for gradual water loss due to precipitation and warming of the boundary layer due to latent heating), mean temperature and moisture profiles are relaxed to the initial profile. The temperature and moisture equations include an additional source term in the form $-(\phi_0 - \langle\phi\rangle)/\tau$, where ϕ_0, $\langle\phi\rangle$ and τ are the initial profile, the horizontal mean of ϕ at a given height and the relaxation time scale, respectively. The relaxation time scale τ is height-dependant (mimicking effects of surface heat fluxes) and is prescribed as $\tau = \tau_\mathrm{rlx} \cdot \exp(z/z_\mathrm{rlx})$ with $\tau_\mathrm{rlx} = 300\,\mathrm{s}$ and $z_\mathrm{rlx} = 200\,\mathrm{m}$. Note that such formulation does not dampen small-scale perturbations of ϕ, but simply shifts the horizontal mean toward ϕ_0.

For models that include a description of the cloud droplet size spectrum, the initial data for the droplet concentration and size are obtained by initialising the simulation with a two-hour-long spin-up period. During the spin-up, precipitation formation, cloud drop sedimentation and the relaxation terms are switched off. The spin-up period is intended to adjust the initial cloud droplet size spectrum (not specified by the set-up) to an equilibrium with the initial condition.

The grid steps are 20 m in both directions. The advection-solver timestep is one second. Shorter sub-timesteps may be used for the microphysics.

2.3 A conceptual solver

The conceptual solver is meant to perform numerical integration of a system of heterogeneous transport equations, each equation of the form:

$$\partial_t r_i + \frac{1}{\rho_\mathrm{d}} \nabla \cdot (\boldsymbol{u} \rho_\mathrm{d} r_i) = \dot{r}_i \tag{4}$$

where r_i is the mixing ratio of the advected constituent, ρ_d is the dry-air density, $\boldsymbol{u}$ is the velocity field, and the dotted right-hand-side term $\dot{r}_i$ depicts sources (see also Appendix A). The solver logic consists of five steps executed in a loop, with each loop repetition advancing the solution by one timestep. Each of the first four integration steps is annotated in Fig. 2 and described in the following paragraph. The final step does data output and is performed conditionally every few timesteps.

The proposed solver features uncentered-in-time integration of the right-hand-side terms. The source terms that are

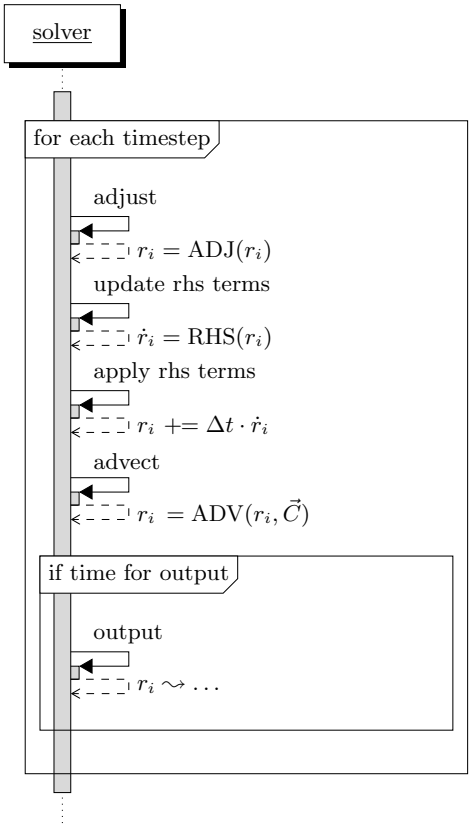

Figure 2. A sequence diagram depicting control flow in a conceptual solver described in Sect. 2.3. This solver design is extended with *libcloudph++* API calls in diagrams presented in Figs. 3, 5, and 7. The diagram structure is modelled after the Unified Modeling Language (UML) sequence diagrams. Arrows with solid lines depict calls, while the dashed arrows depict returns from the called code. Individual solver steps are annotated with labels expressed in semi-mathematical notation and depicting key data dependencies. Model state variables are named r_i, their corresponding right-hand-side terms are named $\dot{r}_i$. If a symbol appears on both sides of the equation, a programming-like assignment notation is meant, in which the old value of the symbol is used prior to assignment, e.g. $r_i = \mathrm{ADV}(r_i, \boldsymbol{C})$. ADV, ADJ, and RHS depict all operations the solver does during the advection, adjustment, and right-hand-side update steps, respectively.

not formulated as time derivatives are referred to as adjustments. The adjustments are applied after advection but before updating the right-hand-side terms.

The library code is not bound to this particular solver logic – it is just an example intended to present the library API. We refer the reader to Grabowski and Smolarkiewicz (2002) for the discussion of higher-order integration techniques for moist atmospheric flows.

3 Single-moment bulk scheme

A common approach to represent cloud water and precipitation in a numerical simulation is the so-called single-moment bulk approach. The concepts behind it date back to the seminal works of Kessler (1995, Sect. 3, and earlier works cited therein). The constituting assumption of the scheme is the division of water condensate into two categories: cloud water and rain water. The term single-moment refers to the fact that only the total mass (proportional to the third moment of the particle size distribution) of water per category (cloud or rain) is considered in the model formulation.

In an Eulerian framework, two transport equations for the cloud water mixing ratio r_c and the rain water mixing ratio r_r are solved in addition to the state variables θ and r_v representing heat and moisture content, respectively (see Table 1 for a list of model-state variables in all schemes discussed in the paper).

Single-moment bulk microphysics is a simplistic approach. Without information about the shape of droplet size distribution, the model is hardly capable of being coupled with a description of aerosol- or radiative-transfer processes.

3.1 Formulation

3.1.1 Key assumptions

The basic idea is to maintain saturation in the presence of cloud water. Condensation/evaporation of cloud water triggered by supersaturation/subsaturation occurs instantaneously. Rain water forms through autoconversion of cloud water into rain (the negligible condensation of rain water is not considered). Autoconversion occurs only after a prescribed threshold of the cloud water mixing ratio is reached. Subsequent increase in rain water is possible through the accretion of cloud water by rain.

Cloud water is assumed to follow the airflow, whereas rain water falls relative to the air with a sedimentation velocity. Rain water evaporates only after all available cloud water has been evaporated and saturation is still not reached. In contrast to cloud water, rain water evaporation does not occur instantaneously. The rain evaporation rate is a function of relative humidity and is parameterised with an assumed shape of the raindrop size distribution.

3.1.2 Phase changes

Phase changes of water are represented with the so-called saturation adjustment procedure. Unlike in several other formulations of the saturation adjustment procedure (cf. Straka, 2009, chapt. 4.2), the implemented in *libcloudph++* covers not only cloud water condensation and evaporation, but also rain water evaporation.

Any excess of water vapour with respect to saturation is instantly converted into cloud water, bringing the relative humidity to 100 %. Similarly, any deficit with respect to saturation causes instantaneous evaporation of liquid water. The formulation of the saturation adjustment procedure takes the latent heat release equation as a starting point. The heat source depicted with $\Delta\theta$ is defined through two integrals, the first representing condensation or evaporation of cloud water, and the second one representing rain evaporation:

$$\Delta\theta = \int_{r_\mathrm{v}}^{r'_v} \frac{\mathrm{d}\theta}{\mathrm{d}r_\mathrm{v}}\mathrm{d}r_\mathrm{v} + \int_{r'_\mathrm{v}}^{r''_\mathrm{v}} \frac{\mathrm{d}\theta}{\mathrm{d}r_\mathrm{v}}\mathrm{d}r_\mathrm{v} \tag{5}$$

$$\Delta r_\mathrm{v} = \underbrace{(r'_\mathrm{v} - r_\mathrm{v})}_{-\Delta r_\mathrm{c}} + \underbrace{(r''_\mathrm{v} - r'_\mathrm{v})}_{-\Delta r_\mathrm{r}} \tag{6}$$

where $\frac{\mathrm{d}\theta}{\mathrm{d}r_\mathrm{v}} = \frac{-\theta l_v}{c_\mathrm{pd} T}$ (cf. Eq. A13 in Appendix A) and the integration limit r'_v for cloud water condensation/evaporation is:

$$r'_\mathrm{v} = \begin{cases} r'_\mathrm{vs} & r_\mathrm{v} > r_\mathrm{vs} \\ r'_\mathrm{vs} & r_\mathrm{v} \le r_\mathrm{vs} \wedge r_\mathrm{c} \ge r'_\mathrm{vs} - r_\mathrm{v} \\ r_\mathrm{v} + r_\mathrm{c} & r_\mathrm{v} \le r_\mathrm{vs} \wedge r_\mathrm{c} < r'_\mathrm{vs} - r_\mathrm{v} \end{cases} \tag{7}$$

where $r'_\mathrm{vs} = r_\mathrm{vs}(\rho_\mathrm{d}, \theta', r'_\mathrm{v})$ is the saturation vapour density evaluated after the adjustment. The first case in Eq. 7 corresponds to supersaturation. The second and the third cases correspond to subsaturation with either sufficient or insufficient amount of cloud water to bring the air back to saturation.

When saturation is reached through condensation or evaporation of the cloud water, the second integral in Eq. (7) vanishes. If there is not enough cloud water available to reach saturation through evaporation, the integration continues with the limit r''_v defined as follows:

$$r''_\mathrm{v} = \begin{cases} r'_\mathrm{v} & r'_\mathrm{v} = r'_\mathrm{vs} \\ r''_\mathrm{vs} & r'_\mathrm{v} < r'_\mathrm{vs} \wedge \delta r_\mathrm{r} \ge r''_\mathrm{vs} - r'_\mathrm{v} \\ r'_\mathrm{v} + \delta r_\mathrm{r} & r'_\mathrm{v} < r'_\mathrm{vs} \wedge \delta r_\mathrm{r} < r''_\mathrm{vs} - r'_\mathrm{v} \end{cases} \tag{8}$$

where δr_r depicts the limit of evaporation of rain within one timestep. Here, it is parameterised as $\delta r_\mathrm{r} = \min(r_\mathrm{r}, \Delta t \cdot E_\mathrm{r})$ with E_r being the rain evaporation rate estimated following Grabowski and Smolarkiewicz (1996, Eq. 5c) using the formula of Ogura and Takahashi (1971, Eq. 25). As with r'_vs, here $r''_\mathrm{vs} = r_\mathrm{vs}(\rho_\mathrm{d}, \theta'', r''_\mathrm{v})$.

Table 1. State variables for the three implemented schemes. Number of state variables times the number of Eulerian grid cells plus number of particle attributes times the number of Lagrangian computational particles gives an estimation of the memory requirement of a given scheme. See Appendix B for symbol definitions.

	Eulerian (PDE) state variables	Lagrangian (ODE) particle attributes
1-moment bulk	θ, r_v, r_c, r_r	–
2-moment bulk	θ, r_v, r_c, r_r, n_c, n_r	–
particle-based	θ, r_v	r_d^3, r_w^2, N, κ

Noteworthy, the name adjustment reserved in Sect. 2.3 for source terms not formulated as time derivative suits the above-defined procedure which is formulated through integration over vapour mixing ratio and not over time (see also discussion of Eq. 3a in Grabowski and Smolarkiewicz, 1990).

3.1.3 Coalescence

The collisions and coalescence of droplets are modelled with two separate processes: autoconversion and accretion. Autoconversion represents collisions between cloud droplets only, while accretion refers to collisions between rain drops and cloud droplets. Both are parameterised in a phenomenological manner as right-hand-side (rhs) terms following Grabowski and Smolarkiewicz (1996, Eq. 5a,b) using the Kessler's formulæ. See Wood (2005) for a review of how these formulations compare with other bulk warm-rain schemes.

In the Kessler's formulation, autoconversion source term is proportional to $\max(r_\mathrm{c} - r_\mathrm{c0}, 0)$, where the value of the mixing-ratio threshold r_c0 controls the onset of precipitation. Values of r_c0 found in the literature vary from 10^{-4} to $10^{-3}\,\mathrm{kg\,kg^{-1}}$ (Grabowski and Smolarkiewicz, 1996).

3.1.4 Sedimentation

Representation of sedimentation of rain water is formulated as a rhs term[1]. The rhs term is formulated employing the upstream advection scheme:

$$\dot{r}_\mathrm{r}^\mathrm{new} = \dot{r}_\mathrm{r}^\mathrm{old} - (F_\mathrm{in} - F_\mathrm{out})/\rho_\mathrm{d} \tag{9}$$

$$F_\mathrm{in} = F_\mathrm{out}|_\mathrm{above} \tag{10}$$

$$F_\mathrm{out} = -\frac{r_\mathrm{r}}{\Delta z}\frac{[\rho_\mathrm{d}|_\mathrm{below} v_\mathrm{t}(r_\mathrm{r}|_\mathrm{below}) + \rho_\mathrm{d} v_\mathrm{t}(r_\mathrm{r})]}{2} \tag{11}$$

where $^\mathrm{old}$ and $^\mathrm{new}$ superscripts are introduced to indicate that $\dot{r}_\mathrm{r}$ is a sum of multiple terms. The $|_\mathrm{above}$ and $|_\mathrm{below}$ symbols refer to the grid cell sequence in a column, v_t is the rain terminal velocity parameterised as a function of rain water mixing ratio (Eq. 5d in Grabowski and Smolarkiewicz, 1996), and F_in and F_out symbolise fluxes of r_r through the grid cell edges.

Employment of the upstream scheme brings several consequences. First, unlike the cellwise formulation of phase changes and coalescence, the sedimentation scheme is defined over a grid column. Second, the combination of terminal velocity, vertical grid cell spacing Δz and the timestep Δt must adhere to the Courant condition (cf. discussion in Grabowski and Smolarkiewicz, 2002). Last, but not least, the upstream algorithm introduces numerical diffusion that can be alleviated by application of a higher-order advection scheme Smolarkiewicz (e.g., MPDATA, cf. 2006, and references therein).

3.2 Programming interface

3.2.1 API elements

The single-moment bulk scheme's API consists of one structure (composite data type) and three functions, which are all defined within the **libcloudph::blk_1m** namespace. The separation of the scheme's logic into the three functions is done first according to the conceptual solver design (i.e. separation of rhs terms and adjustments), and second according to a data-dependency criterion (i.e. cellwise or columnwise calculations). In case of the single-moment bulk scheme, the three functions correspond to the three represented processes, namely phase changes (cellwise adjustments), coalescence (cellwise rhs terms), and sedimentation (columnwise rhs term). Sedimentation is the only process involving columnwise traversal of the domain (note the $|_\mathrm{above}$ and $|_\mathrm{below}$ symbols in Eqs. 9–11).

The **blk_1m::opts_t** structure (Listing 1) is intended for storing options of the scheme for a given simulation. The template parameter **real_t** controls floating point format (e.g., float, double, ...). The structure fields include flags for toggling individual processes, a value of autoconversion threshold r_c0, and an absolute tolerance used in numerical integration of Eq. (7). By default, all processes are enabled, $r_\mathrm{c0} = 5 \times 10^{-4}\,\mathrm{kg\,kg^{-1}}$ and the tolerance is set to $2 \times 10^{-5}\,\mathrm{kg\,kg^{-1}}$. All three functions from the single-moment bulk scheme's API expect an instance of **opts_t** as their first parameter (see Listings 2–4).

[1] Another commonly used approach is to alter the vertical component of the Courant number when calculating advection

```
template<typename real_t>
struct opts_t {
  bool
    cond = true,    // condensation
    cevp = true,    // evaporation of cloud
    revp = true,    // evaporation of rain
    conv = true,    // autoconversion
    accr = true,    // accretion
    sedi = true;    // sedimentation
  real_t
    r_c0  = 5e-4,   // autoconv. threshold
    r_eps = 2e-5;   // absolute tolerance
};
```

Listing 1: **blk_1m::opts_t** definition.

The saturation adjustment of state variables (cf. Sect. 3.1.2) is obtained through a call to the **blk_1m::adj_cellwise()** function (signature in Listing 2). The additional template parameter **cont_t** specifies the type of data container used for passing model state variables. The function expects **cont_t** to implement an STL-style[2] iterator interface (e.g., the standard **std::vector** class or a Blitz++ array slice as used in the example code described in Appendix C). The function arguments include references

```
template <typename real_t, class cont_t>
void adj_cellwise(
  const opts_t<real_t> &opts,
  const cont_t &rhod_cont,
  cont_t &th_cont,
  cont_t &rv_cont,
  cont_t &rc_cont,
  cont_t &rr_cont,
  const real_t &dt
)
```

Listing 2: **blk_1m::adj_cellwise()** signature.

```
template <typename real_t, class cont_t>
void rhs_cellwise(
  const opts_t<real_t> &opts,
  cont_t &dot_rc_cont,
  cont_t &dot_rr_cont,
  const cont_t &rc_cont,
  const cont_t &rr_cont
)
```

Listing 3: **blk_1m::rhs_cellwise()** signature.

[2]C++ Standard Template Library

```
template <typename real_t, class cont_t>
real_t rhs_columnwise(
  const opts_t<real_t> &opts,
  cont_t &dot_rr_cont,
  const cont_t &rhod_cont,
  const cont_t &rr_cont,
  const real_t &dz
)
```

Listing 4: **blk_1m::rhs_columnwise()** signature.

to containers storing ρ_d (read-only) and $\theta, r_\mathrm{v}, r_\mathrm{c}, r_\mathrm{r}$ (to be adjusted). The last argument **dt** is the timestep needed to calculate the precipitation evaporation limit (see discussion of Eq. 8).

Forcings due to autoconversion and accretion are obtained through a call to the **blk_1m::rhs_cellwise()** function whose signature is given in Listing 3. The function modifies $\dot{r}_c$ and $\dot{r}_\mathrm{r}$ by adding the computed rhs terms to the values already present in $\dot{r}_c$ and $\dot{r}_\mathrm{r}$. Read-only access is required for ρ_d, r_c and r_r passed as the last three arguments.

Representation of sedimentation is included in a separate function **rhs_columnwise()** (signature in Listing 4). The **cont_t** references passed as arguments are assumed to point to containers storing vertical columns of data with the last element placed at the top of the domain. The last argument **dz** is the vertical grid spacing. The function returns the value of F_out (see Eq. 9) for the lowermost grid cell within a column.

3.2.2 Example calling sequence

With the prototype solver concept defined in Sect. 2.3, all three functions described above are called once per each timestep. The diagram in Fig. 3 depicts the sequence of calls. As suggested by its name, the **adj_cellwise()** function (covering representation of phase changes) is called within the adjustments step. Functions **rhs_cellwise()** and **rhs_columnwise()** covering representation of coalescence and sedimentation, respectively, are both called during the rhs-update step.

3.3 Implementation overview

The single-moment bulk scheme is implemented as a header-only C++ library (i.e. one does not have to build it separately and link with it, just the header files are needed to use it). The implementation of the single-moment bulk scheme requires a C++ compiler compliant with the C++11 version of the language.

Variables, function arguments, and return values of physical meaning are all typed using the Boost.units classes (Schabel and Watanabe, 2008). Consequently, all expressions involving them are subject to dimensional analysis at compile time – incurring no runtime overhead. This reduces the risk

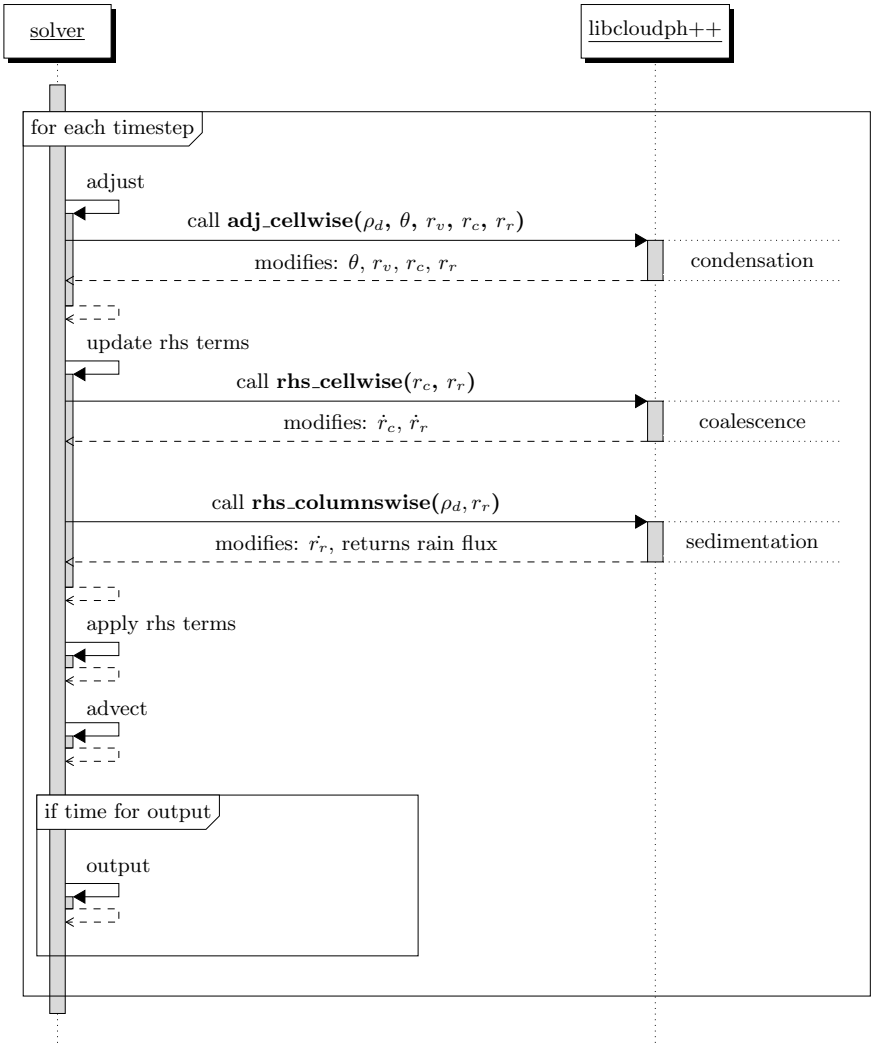

Figure 3. Sequence diagram of *libcloudph++* API calls for the single-moment bulk scheme and a prototype transport equation solver. See discussion in Sect. 3.2.2 and the caption of Fig. 2 for description or the diagram elements.

of typo-like bugs (e.g. divide instead of multiply by density) and contributes to readibility and hence maintainability of the code.

The integrals in Eq. (7) defining the saturation adjustment procedure are computed using the Boost.Numeric.Odeint library (Ahnert and Mulansky, 2013). The container traversals (e.g., iteration over elements of a set of array slices or a set of vectors) are performed using the Boost.Iterator library.

3.4 Example results

The simulation framework described in Sect. 2 and implemented as described in Appendix C was used to perform an example simulation with the single-moment bulk scheme. Integration of the transport equations was performed using the nonoscillatory variant of the MPDATA advection scheme (Smolarkiewicz, 2006). Figure 4 presents a snapshot of the cloud and the rain water fields after 30 min simulation time (excluding the spin-up period). The cloud deck is located in the upper part of the domain with the cloud water content increasing from the cloud base up to the upper boundary of the domain. The model has reached a quasi-stationary state and features a drizzle shaft that forms in the updraught region in the left-hand side of the domain. The quasi-stationary state was preceded by a transient rainfall across the entire domain in the first minutes of the simulation. This was caused by the initial cloud water content exceeding the autoconversion threshold in the upper part of the entire cloud deck.

4 Double-moment bulk scheme

A common extension of the single-moment bulk approach is a double-moment bulk scheme. Similarly to the single-moment approach, the double-moment warm-rain scheme

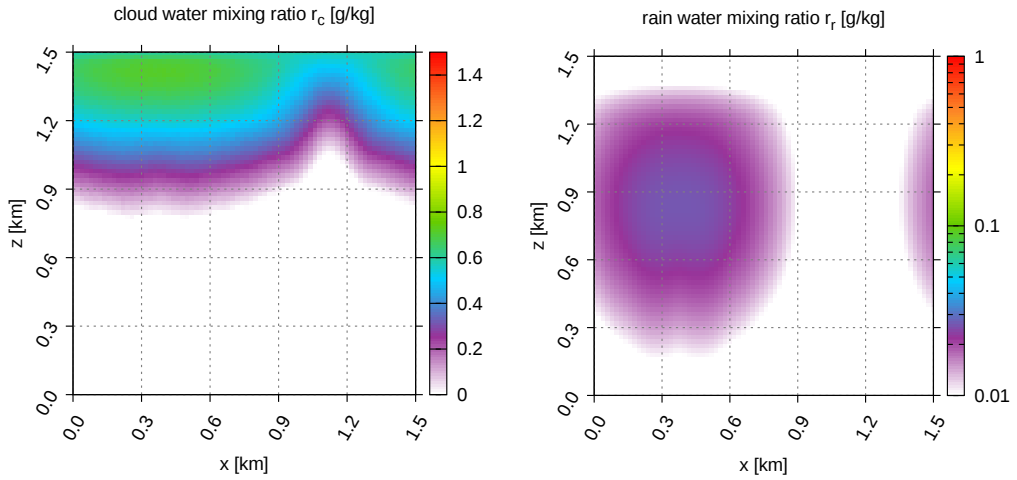

Figure 4. Example results from a 2-D kinematic simulation using the single-moment bulk scheme. All panels depict model state after 30 min simulation time (excluding the spin-up period). Note the logarithmic colour scale for rain water plots. See Sect. 3.4 for discussion.

assumes that condensed water is divided into two categories: cloud water and rain water. In addition to the total mass of water in both categories, concentrations of droplets and drops are also predicted. As a result, the scheme considers two moments of particle size distribution, hence the name. In the Eulerian framework, four transport equations for cloud droplet concentration n_c, cloud water mixing ratio r_c, rain drop concentration n_r and rain water mixing ratio r_r are solved (see Table 1 for a list of model-state variables). With additional information on the concentration of cloud droplets and rain drops, the double-moment bulk microphysics scheme is better suited than the single-moment scheme for coupling to aerosol and radiative-transfer models.

The double-moment scheme implemented in *libcloudph++* was introduced by Morrison and Grabowski (2007). The scheme includes prediction of the supersaturation, making it well suited for depicting impacts of aerosol on clouds and precipitation. However, the scheme does not keep track of the changes of aerosol size distribution, and hence excludes impacts of clouds and precipitation on aerosol.

4.1 Formulation

4.1.1 Key assumptions

The formulation of the double-moment bulk scheme assumes aerosol, cloud, and rain spectra shapes (lognormal, gamma, and exponential, respectively). Aerosol is assumed to be well mixed throughout the whole domain and throughout the whole simulation time (uniform concentration per unit mass of dry air). Cloud water forms only if some of the aerosol particles are activated due to supersaturation. Activation and subsequent growth by condensation are calculated applying the predicted supersaturation. As in the single-moment scheme, rain water forms through autoconversion and is further increased by accretion. Prediction of the mean size of cloud droplets and rain drops allows to better link the parameterisation of autoconversion and accretion to the solutions of the collision-coalescence equation. As in the single-moment scheme, cloud water is assumed to follow the airflow, whereas rain water falls relative to the air. Evaporation of cloud and rain water is included in the formulation of phase changes (and hence includes the negligible diffusional growth of rain water).

4.1.2 Phase changes

Cloud droplets form from activated aerosol. The number of activated droplets is derived by applying the Köhler theory to the assumed multi-modal lognormal size distribution of aerosols. Freshly activated cloud droplets are assumed to have the radius of $1\,\mu\mathrm{m}$; for full derivation see Morrison and Grabowski (2007, Eqs. 9–13) and Khvorostyanov and Curry (2006). The concentrations of activated droplets are computed separately for each mode of the aerosol size distribution and then summed.

The size distribution of aerosols is not resolved by the model. To take into account the decrease of aerosol concentration due to previous activation, in each timestep the number of available aerosols is approximated by the difference between the initial aerosol concentration and the concentration of preexisting cloud droplets. Note that this approximation is valid for weakly precipitating clouds only. For a strongly raining cloud, the model should include an additional variable, the concentration of activated cloud droplets. It differs from the droplet concentration because of collision-coalescence (see Eqs. (7) and (8) in Morrison and Grabowski, 2008).

The changes in cloud and rain water due to condensation and evaporation follow Eq. (8) in Morrison and Grabowski

(2007) with the phase relaxation times computed following Eq. (4) in Morrison et al. (2005) adapted to fall speed parameterisation used in Morrison and Grabowski (2007).

The decrease in number concentration due to evaporation of cloud droplets and rain drops is computed following the approach of Khairoutdinov and Kogan (2000). Cloud droplet concentration is kept constant during evaporation, until all cloud water has to be removed. Rain drop concentration decreases during evaporation preserving the mean size of rain (drizzle) drops.

4.1.3 Coalescence

Parameterisation of autoconversion and accretion follows Khairoutdinov and Kogan (2000). In contrast to the single-moment scheme, the autoconversion rate is a continuous function, and the rain onset is not controlled by a single threshold. Drizzle drops formed due to autoconversion are assumed to have initial radius of $25\,\mu\mathrm{m}$.

4.1.4 Sedimentation

Sedimentation is calculated in the same way as in the single-moment scheme (see Sect. 3.1.4), employing upstream advection. Sedimentation velocities (mass-weighted for the rain density and number-weighted for the rain drop concentration) are calculated by applying the terminal velocity formulation given in Simmel et al. (2002, Table 2). Sedimentation velocity is multiplied by $\rho_{\mathrm{d0}}/\rho_{\mathrm{d}}$ to follow Eq. (A4) in Morrison et al. (2005), where ρ_{d0} is the density of dry air at standard conditions.

4.2 Programming interface

4.2.1 API elements

The double-moment bulk scheme's API consists of one structure and two functions, all defined within the **libcloudphxx::blk_2m** namespace. The structure **blk_2m::opts_t** holds the scheme's options and its definition is provided in Listing 5. Among the options, there are process-toggling Boolean fields, parameters of the aerosol lognormal size distribution (see Eq. 3) and the parameter β defining the solubility of aerosol (see Khvorostyanov and Curry, 2006, Sect. 2.1).

All processes are represented as right-hand-side terms. Contributions to the rhs terms, due to phase changes and coalescence, are obtained through a call to **blk_2m::rhs_cellwise()** (see Listing 6). As in the single-moment bulk scheme's API, contribution from sedimentation to the rhs terms can be computed by calling **blk_2m::rhs_columnwise()** (Listing 7).

The meaning of the template parameters and the function arguments is analogous to the single-moment bulk scheme's API (see Sect. 3.2). The computed values of rhs terms are

```
template<typename real_t>
struct opts_t
{
  bool
    acti = true, // activation
    cond = true, // condensation
    acnv = true, // autoconversion
    accr = true, // accretion
    sedi = true; // sedimentation

  // RH limit for activation
  real_t RH_max = 44;

  // aerosol spectrum
  struct lognormal_mode_t
  {
    real_t
      mean_rd,   // [m]
      sdev_rd,   // [1]
      N_stp,     // [m-3] @STP
      chem_b;    // [1]
  };
  std::vector<lognormal_mode_t> dry_distros;
};
```

Listing 5: **blk_2m::opts_t** definition.

```
template <typename real_t, class cont_t>
void rhs_cellwise(
  const opts_t<real_t> &opts,
  cont_t &dot_th_cont,
  cont_t &dot_rv_cont,
  cont_t &dot_rc_cont,
  cont_t &dot_nc_cont,
  cont_t &dot_rr_cont,
  cont_t &dot_nr_cont,
  const cont_t &rhod_cont,
  const cont_t &th_cont,
  const cont_t &rv_cont,
  const cont_t &rc_cont,
  const cont_t &nc_cont,
  const cont_t &rr_cont,
  const cont_t &nr_cont,
  const real_t &dt
)
```

Listing 6: **blk_2m::rhs_cellwise()** signature.

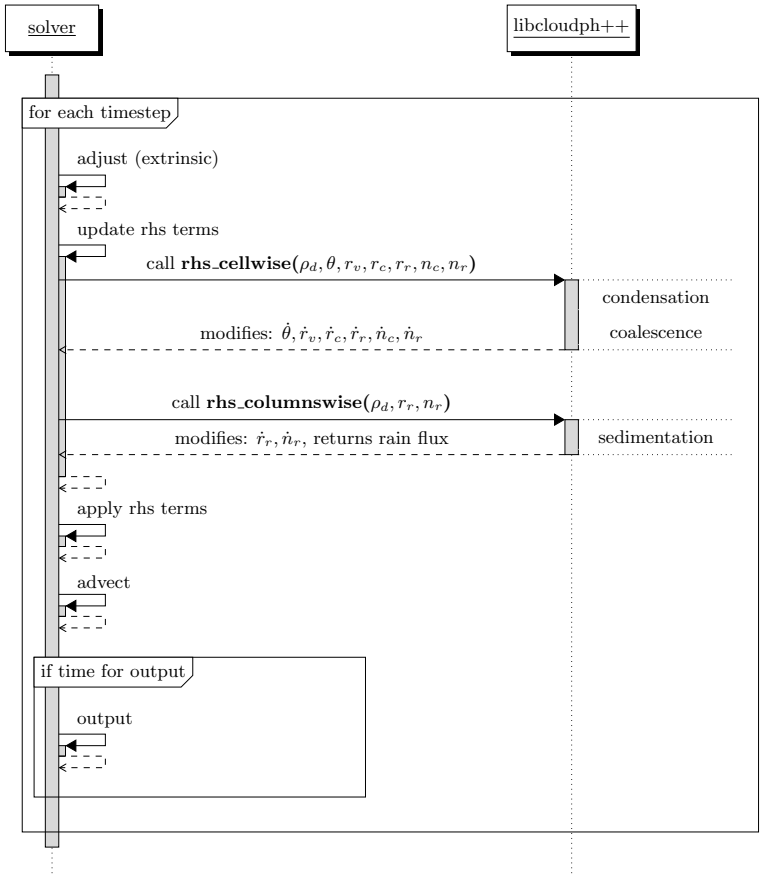

Figure 5. Sequence diagram of *libcloudph++* API calls for the double-moment bulk scheme and a prototype transport equation solver. See discussion in Sect. 4.2.2 and the caption of Fig. 2 for description of the diagram elements.

```
template <typename real_t, class cont_t>
real_t rhs_columnwise(
  const opts_t<real_t> &opts,
  cont_t &dot_rr_cont,
  cont_t &dot_nr_cont,
  const cont_t &rhod_cont,
  const cont_t &rr_cont,
  const cont_t &nr_cont,
  const real_t &dt,
  const real_t &dz
)
```

Listing 7: **blk_1m::rhs_columnwise()** signature.

added to the values already present in the arrays passed as arguments.

The cellwise-formulated processes are handled in the following order: activation, condensation/evaporation of cloud droplets, autoconversion, accretion, and condensation/evaporation of rain. In principle, there are no guarantees that the summed contributions from all of those processes, multiplied by the timestep, are smaller than the available water contents or droplet concentrations. To prevent negative values of water contents or concentrations, each contribution to the rhs term evaluated within **rhs_cellwise()** is added to the array $\dot{r}_i$ passed as argument using the following rule:

$$\dot{r}_i^{\text{new}} = \min\left(\dot{r}_i^{\star}, \frac{r_i + \Delta t \cdot \dot{r}_i^{\text{old}}}{\Delta t}\right) \tag{12}$$

where $\dot{r}_i^{\text{old}}$ is the value obtained in evaluation of previously-handled processes, $\dot{r}_i^{\star}$ is the value computed using the model formulæ, and $\dot{r}_i^{\text{new}}$ is the augmented value of rhs term that guarantees non-negative values of r_i after its application. The same rule is applied when evaluating values of outgoing fluxes F_{out} from Eq. (9) when calculating rhs term within **rhs_columnwise()**. The **rhs_columnwise()** returns the value of the F_{out} flux from the lowermost grid cell within a column.

4.2.2 Example calling sequence

A diagram with an example calling sequence for the double-moment scheme is presented in Fig. 5. The only difference from the single-moment bulk scheme's calling sequence presented in Sect. 3.2.2 is the lack of an adjust-

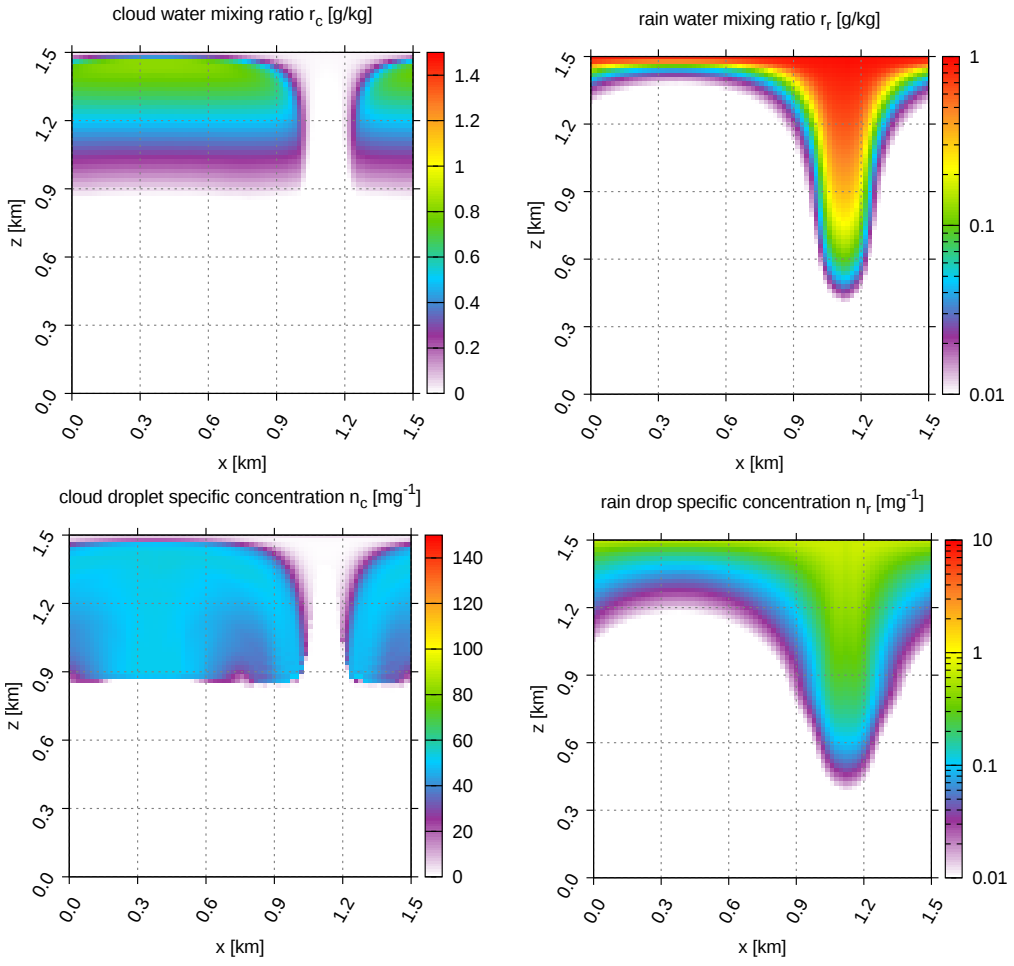

Figure 6. Example results from a 2-D kinematic simulation using the double-moment bulk scheme. All panels depict model state after 30 min simulation time (excluding the spin-up period). Note the logarithmic colour scale used for plotting rain water. See Sect. 4.4 for discussion.

ments step. Condensation is represented using right-hand-side terms and is computed together with coalescence by calling **blk_2m::rhs_cellwise()**.

4.3 Implementation overview

The implementation of the double-moment scheme follows closely the implementation of the single-moment scheme (see Sect. 3.3). It's a header-only C++ library, using Boost.units classes for dimensional analysis and Boost.Iterator for iterating over sets of array slices.

4.4 Example results

Simulations presented in Sect. 3.4 were repeated with the double-moment scheme. Figure 6 presents snapshots of the cloud and rain water content as well as the cloud and rain drop concentration fields after 30 min simulated time (excluding the spin-up period). Because of large differences in the predicted rain, rain water content, and drop concentration are plotted using logarithmic colour scale in order to keep the same colour range for all three presented schemes.

Similarly to the results from the single-moment scheme presented in Fig. 4, cloud water content increases from the cloud base almost up to the upper boundary of the domain. However, unlike in the case of the single-moment scheme, the cloud deck in Fig. 6 features a "cloud hole" above the downdraught region. The rain forms in the upper part of the updraught and is advected into the downdraught region in the right-hand side of the domain. The double-moment simulation at the thirtieth minute is still to reach the quasi-stationary state. This occurs because of the differences in the parameterisation of autoconversion that lead to different timings of the onset of precipitation.

The cloud droplet concentration plot reveals that the model captures the impact of the cloud-base vertical velocity (and hence the supersaturation) on the concentration of activated cloud droplets. The highest concentrations are found near the updraught axis, and the lowest near the updraught edges. There is a difference in shape between the rain drop concen-

tration field n_{r} and the rain water mixing ratio field r_{r}. This corresponds to the different fall velocities for the two fields – number- and mass-weighted for n_{r} and r_{r}, respectively.

5 Particle-based scheme

The third scheme available in *libcloudph++* differs substantially from the two bulk schemes. It does not treat water condensate as continuous medium. Instead, the scheme employs Lagrangian tracking of particles that represent atmospheric aerosol, cloud and drizzle droplets, and rain drops. Volumes relevant to atmospheric flows contain far too many particles to be individually represented in a numerical model. Consequently, each "computational particle" represents a multiplicity of particles of identical properties (i.e. spatial coordinates and physicochemical properties). Such an approach was recently applied for modelling precipitating clouds by Andrejczuk et al. (2010), Sölch and Kärcher (2010), Riechelmann et al. (2012) and Arabas and Shima (2013). Formulation of the scheme presented here follows the Super Droplet Method of Shima et al. (2007, 2009) to represent collisions and coalescence of particles.

5.1 Formulation

The formulated particle-based model involves a Lagrangian component and an Eulerian component (that is not part of the library). The Eulerian component is responsible for advecting θ and r_{v} (see Appendix A). The Lagrangian component is responsible for tracking the computational particles, each having the following attributes:

- multiplicity N
- location (i.e. spatial coordinates with $0, 1, 2$ or 3 components)
- wet radius squared r_{w}^2
- dry radius cubed r_{d}^3
- hygroscopicity parameter κ

Multiplicity depicts the number of particles represented by a computational particle. All particles represented by one computational particle are assumed to be spherical water solution droplets of radius r_{w}. Following Shima et al. (2009), the model is formulated in r_{w}^2 for numerical reasons.

The amount of solvent is represented by the dry radius r_{d} (third power is used in the model code because most often r_{d}^3 serves as a proxy for volume of the solvent). The hygroscopicity of the solvent is parameterised using the single-parameter approach of Petters and Kreidenweis (2007).

The list of particle attributes can be extended. For example, parameters describing chemical composition of the solution or the electrical charge of a particle can be added. Adding new particle attributes does not increase the computational expense of the Eulerian component of the solver. However, extension of the phase space by a new dimension (the added attribute) potentially requires using more computational particles to achieve sufficient coverage of the phase space.

5.1.1 Key assumptions

Most of the assumptions of the bulk models described in Sects. 3 and 4 are no longer necessary. All particles are subject to the same set of processes. As a result, the model represents even dry deposition and collisions between aerosol particles (both being effectively negligible). The supersaturation is resolved taking into account phase-change kinetics (i.e. condensation and evaporation are not instantaneous). There are no assumptions on the shape of the particle size spectrum. Aerosol particles may be internally or externally mixed (i.e. have the same or different solubility for particles of different sizes).

There are, however, two notable consequences of the assumptions of all particles being spherical and composed of water solution. First, the humidity within the domain and the hygroscopicity of the aerosol substance must both be high enough for the solution to be dilute. For tropospheric conditions and typical complex-composition internally-mixed aerosol, this assumption is generally sound (Fernández-Díaz et al., 1999; Marcolli et al., 2004). Second, the nonsphericity of large precipitation particles has to be negligible. It is a valid assumption for drops smaller than $1\,\mathrm{mm}$ (Szakáll et al., 2010).

It is also assumed that a particle never breaks up into multiple particles. It is a reasonable assumption for the evaporation of cloud particles into aerosol (Mitra et al., 1992). However, both collision-induced and spontaneous breakup become significant (the latter to a much smaller extent) for larger droplets (McFarquhar, 2010) and hence the scheme requires an extension in order to allow for diagnosing rain spectra for strongly precipitating clouds.

There is not yet any mechanism built into the model to represent aerosol sources (other than regeneration of aerosol by evaporation of cloud droplets).

5.1.2 Advection

In the current version of the library, it is assumed that particle motion has two components: advection by the fluid flow and gravitational sedimentation with the terminal velocity. The library interface expects that the user passes information on the flow velocity in the form of Courant number fields, one per each dimension. The Courant number is defined as the flow velocity times the ratio of the timestep to the grid step in a given dimension. The Arakawa-C staggered grid is used and hence the Courant numbers represent velocities at the edges of the Eulerian grid cells.

Transport of particles by the flow is computed using the backward Euler scheme:

$$x^{[n+1]} = x^{[n]} + \Delta x \cdot C(x^{[n+1]}) \tag{13}$$

where C is the Courant number field component, and Δx is the grid step (formulæ are given for the x dimension, but are applicable to other dimensions as well). Evaluation of $C(x^{[n+1]})$ is performed using linear approximation (interpolation/extrapolation of the particle velocities using fluid velocity values at the grid cell edges):

$$C(x^{[n+1]}) = (1-\omega) \cdot C_{[i-\frac{1}{2}]} + \omega \cdot C_{[i+\frac{1}{2}]} \tag{14}$$

where the fractional indices $i - \frac{1}{2}$ and $i + \frac{1}{2}$ denote left and right edges of a grid cell i in which a given particle is located at time level n. The weight ω is defined as:

$$\omega = x^{[n+1]}/\Delta x - \lfloor x^{[n]}/\Delta x \rfloor \tag{15}$$

where $\lfloor x \rfloor$ depicts the largest integer not greater than x. Substituting Eqs. (14) and (15) into Eq. (13) results in an analytic solution for $x^{[n+1]}$:

$$x^{[n+1]} = \frac{x^{[n]} + \Delta x \left(C_{i-\frac{1}{2}} - \lfloor x^{[n]}/\Delta x \rfloor \cdot \Delta C \right)}{1 - \Delta C} \tag{16}$$

where $\Delta C = C_{i+\frac{1}{2}} - C_{i-\frac{1}{2}}$.

The same procedure is repeated in other spatial dimensions if applicable (i.e. depending on the dimensionality of the Eulerian component). Periodic horizontal boundary conditions are assumed.

5.1.3 Phase changes

The growth rate of particles is calculated using the single-equation (so-called Maxwell–Mason) approximation to the heat and vapour diffusion process (Straka, 2009, rearranged Eq. 5.106):

$$r_{\mathrm{w}} \frac{\mathrm{d}r_{\mathrm{w}}}{\mathrm{d}t} = \frac{D_{\mathrm{eff}}}{\rho_{\mathrm{w}}} (\rho_{\mathrm{v}} - \rho_{\circ}) \tag{17}$$

where the effective diffusion coefficient is:

$$D_{\mathrm{eff}}^{-1} = D^{-1} + K^{-1} \frac{\rho_{\mathrm{vs}} l_{\mathrm{v}}}{T} \left(\frac{l_{\mathrm{v}}}{R_{\mathrm{v}} T} - 1 \right) \tag{18}$$

and ρ_{vs} stands for the density of water vapour at saturation with respect to a plane surface of pure water. The vapour density at drop surface $\rho_{\circ}$ is modelled as:

$$\rho_{\circ} = \rho_{\mathrm{vs}} \cdot a_{\mathrm{w}}(r_{\mathrm{w}}, r_{\mathrm{d}}) \cdot \exp(A/r_{\mathrm{w}}) \tag{19}$$

where water activity a_{w} and the so-called Kelvin term $\exp(A/r_{\mathrm{w}})$ are evaluated using the κ-Köhler parameterisation of Petters and Kreidenweis (2007). See Arabas and Pawlowska (2011) for the formulæ for A, l_{v} and ρ_{vs}.

Vapour and heat diffusion coefficients D and K are evaluated as:

$$D = D_0 \cdot \beta_M \cdot \frac{\mathrm{Sh}}{2} \tag{20}$$

$$K = K_0 \cdot \beta_T \cdot \frac{\mathrm{Nu}}{2} \tag{21}$$

The Fuchs–Sutugin transition-régime correction factors $\beta_M(r_{\mathrm{w}}, T)$ and $\beta_T(r_{\mathrm{w}}, T, p)$ are used in the form recommended for cloud modelling by Laaksonen et al. (2005, i.e. employing mass and heat accommodation coefficients of unity). The Sherwood number Sh and the Nusselt number Nu (twice the mean ventilation coefficients) are modelled following Clift et al. (1978) as advocated by Smolík et al. (2001).

As in the particle-based ice-microphysics model of Sölch and Kärcher (2010), no interpolation of the Eulerian state variables to particle positions is done (in contrast to the approach employed in warm-rain models of Andrejczuk et al., 2008; Shima et al., 2009; Riechelmann et al., 2012). It is therefore assumed, likely in compliance with the logic of an Eulerian solver component, that the heat and moisture are homogeneous within a grid cell. Consequently, the effects of subgrid-scale mixing on the particles follow the so-called homogeneous-mixing scenario (see Jarecka et al., 2013, and references therein). Furthermore, no effects of vapour field inhomogeneity around particles are taken into consideration (see Vaillancourt et al., 2001; Castellano and Ávila, 2011).

Particle terminal velocities used to evaluate Sh and Nu are calculated using the parameterisation of Khvorostyanov and Curry (2002, see also 5.1.5 herein).

The representation of condensation and evaporation in the Lagrangian component involves a sub-stepping logic in which the Eulerian component timestep Δt is divided into a number of equal sub-steps. This is intended to cope with potentially large difference between the characteristic timescales of condensation (notably during aerosol activation) and of the large-scale air flow solved by the Eulerian component of the solver. Presently, the number of subtimesteps is kept constant throughout the domain and throughout the simulation time. However, the actual constraints for timestep length $\Delta t'$ differ substantially, particularly with the distance from cloud base (see Fig. 2 in Arabas and Pawlowska, 2011). An adaptive timestep choice mechanism is planned for a future release. For simplicity, the substepping procedure is not depicted explicitly in the following formulæ. It is only hinted by labelling subtimestep as $\Delta t'$ and the subtimestep number as n'. If the user enables substepping, the advective tendencies of θ_d and r_v are applied fractionally in each sub-step.

Within each sub-step, the drop growth equation is solved for each computational particle with an implicit scheme with

respect to wet radius but explicit with respect to r_v and θ:

$$r_\mathrm{w}^{2^{[n'+1]}} = r_\mathrm{w}^{2^{[n']}} + \Delta t' \cdot \left.\frac{\mathrm{d}r_w^2}{\mathrm{d}t}\right|_{r_\mathrm{w}^{2^{[n'+1]}}, r_\mathrm{v}^{[n']}, \theta^{[n']}} \tag{22}$$

Solution to the above equation is sought by employing a predictor-corrector type procedure. First, the value of the $\frac{\mathrm{d}r_w^2}{\mathrm{d}t}$ derivative evaluated at $r_\mathrm{w}^{2^{[n']}}$ is used to construct an initial-guess range $a < r_w^{2^{[n'+1]}} < b$ in which roots of Eq. (22) are to be sought, with:

$$a = \max\left(r_d^2, r_\mathrm{w}^{2^{[n']}} + \min\left(2 \cdot \left.\frac{\mathrm{d}r_\mathrm{w}^2}{\mathrm{d}t}\right|_{r_\mathrm{w}^{[n']}}, 0\right)\right) \tag{23}$$

$$b = r_\mathrm{w}^{2^{[n']}} + \max\left(2 \cdot \left.\frac{\mathrm{d}r_\mathrm{w}^2}{\mathrm{d}t}\right|_{r_\mathrm{w}^{[n']}}, 0\right) \tag{24}$$

Second, $r_\mathrm{w}^{2^{n'+1}}$ is iteratively searched using the bisection algorithm. If the initial-guess range choice makes bisection search ill-posed (minimisation function having the same sign at a and b), the algorithm stops after first iteration returning $(a+b)/2$, what reduces the whole procedure to the standard Euler scheme (due to the use of factor 2 in the definition of a and b). It is worth noting, that such treatment of drop growth (i.e. Lagrangian in radius space, also called moving sectional or method of lines approach) incurs no numerical diffusion.

After each sub-step, in addition to application of a fraction of advective tendency, the thermodynamic fields r_v and θ are adjusted to account for water vapour content change due to condensation or evaporation on particles within a given grid cell and within a given sub-step by evaluating:

$$r_\mathrm{v}^{[n'+1]} - r_\mathrm{v}^{[n']} = \rho_\mathrm{d}^{-1} \frac{-4\pi\rho_\mathrm{w}}{3\Delta V} \sum_{i \in \text{grid cell}} N_{[i]} \left[r_{\mathrm{w}_{[i]}}^{3^{[n'+1]}} - r_{\mathrm{w}_{[i]}}^{3^{[n']}} \right] \tag{25}$$

$$\theta^{[n'+1]} - \theta^{[n']} = \left(r_\mathrm{v}^{[n'+1]} - r_\mathrm{v}^{[n']}\right) \left.\frac{\mathrm{d}\theta}{\mathrm{d}r_\mathrm{v}}\right|_{r_\mathrm{v}^{[n']}, \theta^{[n']}} \tag{26}$$

where ΔV is the grid cell volume, and ρ_w is the density of liquid water. Noteworthy, such formulation maintains conservation of heat and moisture in the domain regardless of the accuracy of integration of the drop growth equation.

Phase-change calculations are performed before any other processes. This is because condensation and evaporation are the only process modifying the r_v and θ fields of the Eulerian component. Consequently, Eulerian component of the solver may continue integration as soon as phase-change calculations are completed. Such asynchronous logic is applicable when using a GPU – particle advection, sedimentation, and collisions can be calculated by the Lagrangian component of the solver using a GPU while the Eulerian component advects model state variables using a CPU.

5.1.4 Coalescence

The coalescence scheme is an implementation of the Super Droplet Method (SDM) described in Shima et al. (2009). SDM is a Monte-Carlo type algorithm for representing particle collisions. As it is done for phase changes, coalescence of particles is solved using subtimesteps $\Delta t''$. In each subtimestep, all computational particles within a given grid cell are randomly grouped into non-overlapping pairs (i.e. no computational particle may belong to more than one pair). Then, the probability of collisions between computational particles i and j in each pair is evaluated as:

$$P_{ij} = \max(N_i, N_j) K(r_i, r_j) \frac{\Delta t''}{\Delta V} \frac{n(n-1)}{2\lfloor n/2 \rfloor} \tag{27}$$

where n is the total number of computational particles within a grid cell in a given timestep and $K(r_i, r_j)$ is the collection kernel. In analogy to a target-projectile configuration, scaling the probability of collisions with the larger of the two multiplicities $\max(N_i, N_j)$ (target size) implies that if a collision happens, $\min(N_i, N_j)$ of particles will collide (number of projectiles). The last term in Eq. (27) upscales the probability to account for the fact that not all $(n(n-1)/2)$ possible pairs of computational particles are examined but only $\lfloor n/2 \rfloor$ of them. Evaluation of collision probability for non-overlapping pairs only, instead of for all possible pairs of particles, makes the computational cost of the algorithm scale linearly, instead of quadratically, with the number of computational particles (at the cost of increasing the sampling error of the Monte-Carlo scheme).

If geometric collisions are considered, the coalescence kernel has the following form

$$K(r_i, r_j) = E(r_i, r_j) \cdot \pi (r_i + r_j)^2 \cdot |v_i - v_j| \tag{28}$$

where $E(r_i, r_j)$ is the collection efficiency and v is the terminal velocity of particles (i.e. their flow-relative sedimentation velocity). The collection efficiency differs from unity if hydrodynamic effects (e.g. Vohl et al., 2007) or van der Waals forces (Rogers and Davis, 1990) are considered. The whole coalescence kernel may take different form (in particular may be nonzero for drops of equal terminal velocity) if turbulence effects are taken into account (Grabowski and Wang, 2013, and references therein).

In each subtimestep, the evaluated probability P_{ij} is compared to a random number from a uniform distribution over the (0,1) interval. If the probability is larger than the random number, a collision event is triggered. During a collision event, all $\min(N_i, N_j)$ particles collide (Shima et al., 2009, see Fig. 1 and Sect. 4.1.4 in). One of the colliding computational particles (the one with the smaller multiplicity) retains its multiplicity but changes its dry and wet radii to those of the newly formed particles. The second colliding computational particle (the one with the larger multiplicity) retains its dry and wet radii but changes its multiplicity to the difference between N_i and N_j. Other particle parameters are

either summed (i.e. extensive parameters such as r_{d}^3) or averaged (i.e. intensive parameters such as κ).

Unlike in the formulation of Shima et al. (2009), particles with equal multiplicities collide using the same scheme, leaving one of the particles with zero multiplicity. Particles with zero multiplicity are "recycled" at the beginning of each timestep. The recycling procedure first looks for computational particles with highest multiplicities and then assigns their properties to the recycled particles halving the multiplicity.

The "multiple coalescence" feature of SDM introduced in Shima et al. (2009) to robustly cope with an undersampled condition of $P_{ij} > 1$ is implemented. It is also planned to use the values of P_{ij} to control an adaptive timestep logic to be introduced in a future release.

Noteworthy, the collisional growth is represented in a numerical-diffusion-free manner, that is, Lagrangian in particle radius space (both dry and wet radius). This is an advantage over the Eulerian-type schemes based on the Smoluchowski equation which exhibit numerical diffusion (see e.g. Bott, 1998).

5.1.5 Sedimentation

Sedimentation velocity is computed using the formula of Khvorostyanov and Curry (2002, Eqs. 2.7, 2.12, 2.13, 3.1). The explicit Euler scheme is used for adjusting particle positions Sedimentation may result in the particles leaving the domain (i.e. dry deposition or ground-reaching rainfall). Computational particles that left the domain undergo the same recycling procedure as described in section 3.1.3 for equal-multiplicity collisions.

5.1.6 Initialisation

One of the key parameters of the particle-based simulation is the number of computational particles used. As in several recent cloud-studies employing particle-based techniques, the initial particle spatial coordinates are chosen randomly using a uniform distribution. Consequently, the initial condition has a uniform initial mean density of computational particles per cell (assuming all cells have the same volume). The value of this initial mean density defines the sampling error in the particle parameter space, particularly in the context of phase changes and coalescence which are both formulated on cellwise basis. The ranges of values used in the recent studies are: 30–250 (Sölch and Kärcher, 2010, particles injected throughout simulation), 100–200 (Andrejczuk et al., 2010, grid cell size variable in height, particles added throughout simulation), 26–186 (Riechelmann et al., 2012), 8–512 (Arabas and Shima, 2013), 30–260 (Unterstrasser and Sölch, 2014).

The dry radii of the computational particles are chosen randomly with a uniform distribution in the logarithm of radius. The minimal and the maximal values of dry radius are chosen automatically by evaluating the initial dry-size distribution. The criterion is that the particle multiplicity (i.e. the number of particles represented by a computational particle) for both the minimal and the maximal radii be greater or equal one.

The initial spectrum shape is arbitrary. Externally mixed aerosol may be represented using multiple spectra, each characterised by different value of κ. The initial particle multiplicities are evaluated treating the input spectra as corresponding to the standard atmospheric conditions (STP) and hence the concentrations are multiplied by the ratio of the dry-air density in a given grid cell to the air density at STP.

In one and two dimensions, the grid cell volume ΔV used to derive multiplicities from the concentrations is defined assuming a unit length of $1\,\mathrm{m}$ in the omitted dimensions. This assumption has effectively no impact on the computed rates of condensation or coalescence (Eqs. 25 and 27, respectively), as in their formulation, the multiplicities always appear divided by ΔV. In a zero-dimensional configuration intended for parcel-like frameworks the ΔV is updated in every timestep to match changes in the dry air density by maintaining a constant total mass of dry air of $1\,\mathrm{kg}$.

Equation (17) defines the relationships between the dry and the wet spectra in the model. These should, in principle, be fulfilled by the initial condition imposed on the model state variables. For cloud-free air, it is obtained by assuming an equilibrium defined by putting zero on the left-hand side of Eq. (17). This allows to diagnose the wet spectrum from the dry one. Bringing all particles to equilibrium at a given humidity is done without changing θ and r_{v} to resemble bulk models' initial state. A small amount of water needed to obtain equilibrium is thus added to the system.

For set-ups assuming initial presence of cloud water within the model domain, the equilibrium condition may be applied only to subsaturated regions within the model domain. The initial wet radius of particles within the supersaturated regions is set to its equilibrium value at a given threshold relative humidity (e.g. $\mathrm{RH} = 95\,\%$ as used in Lebo and Seinfeld, 2011). Subsequent growth is computed within the first few minutes of the simulation. Optionally, in order to avoid activation of all available aerosol, the drop growth Eq. (17) is evaluated limiting the value of the supersaturation to a given threshold, e.g. $5\,\%$ as used in the set-up defined in Sect. 2.2 (see also discussion on particle-based simulation initialisation in Andrejczuk et al., 2010, Sect. 2.2).

5.2 Programming interface

5.2.1 API elements

The particle-based scheme's API differs substantially from bulk schemes' APIs. It features object-oriented approach of equipping data structures (referred to as classes) with functions (referred to as methods). Furthermore, unlike the bulk schemes' APIs, the particle-based scheme is not

implemented as a header-only library but requires linking with **libcloudphxx_lgrngn** shared library. The particle-based scheme's API consists of four structures (classes), one function and two enumerations, all defined within the **libcloudphxx::lgrngn** namespace. The often occurring template parameter **real_t** controls the floating point format.

As in the case of bulk schemes, the scheme options are stored in a separate structure **lgrngn::opts_t** whose definition is given in Listing 8. The first Boolean fields provide

```
template<typename real_t>
struct opts_t
{
  // process toggling
  bool adve, sedi, cond, coal;

  // RH limit for drop growth
  real_t RH_max;
```

Listing 8: **lgrngn::opts_t** definition.

control over process toggling. The **RH_max** field defines the RH limit for evaluating drop growth equation (e.g. during a spin-up period, see Sect. 2.2).

Other options of the particle-based scheme not meant to be altered during simulation are grouped into a structure named **lgrngn::opts_init_t** (Listing 9). The initial dry size spectrum of aerosol is represented with the **dry_distros** map. The map associates values of the solubility parameter κ with particle size distributions. The size distributions are specified as pointers to functors returning concentration of particles at STP as a function of logarithm of dry radius. Subsequent fields specify the geometry of the Eulerian grid and the timestep. It is assumed that the Eulerian component operates on a rectilinear grid with a constant grid cell spacing, although this assumption may easily be lifted in future releases if needed. The parameters **x0**, **y0**, **z0**, **x1**, **y1**, **z1** are intended for defining a subregion of the Eulerian domain to be covered with computational particles. The number of sub-steps to be taken within one Eulerian timestep when calculating condensation and coalescence is defined by **sstp_cond** and **sstp_coal**, respectively. The last two fields provide control of the initial mean concentration of computational particles per grid cell and the type of the coalescence kernel to be used. As of the current release, two options are available: the geometric kernel and the Golovin kernel, see Listing 10).

```
template<typename real_t>
struct opts_init_t
{
  // initial dry sizes of aerosol
  typedef boost::ptr_unordered_map<
    real_t,                 // kappa
    unary_function<real_t> // n(ln(rd)) @ STP
  > dry_distros_t;
  dry_distros_t dry_distros;

  // Eulerian component parameters
  int nx, ny, nz;
  real_t dx, dy, dz, dt;

  // no. of substeps
  int sstp_cond, sstp_coal;

  // Lagrangian domain extents
  real_t x0, y0, z0, x1, y1, z1;

  // mean no. of super-droplets per cell
  real_t sd_conc_mean;

  // coalescence Kernel type
  kernel_t kernel;
```

Listing 9: **lgrngn::opts_init_t** definition.

```
enum kernel_t { geometric, golovin };
```

Listing 10: **lgrngn::kernel_t** definition.

Unlike in the case of the bulk schemes, here the actual geometry and memory layout of the Eulerian grid need to be known to map the particle spatial coordinates to the Eulerian grid cell indices. The memory layout of array data is represented in the API using the **lgrngn::arrinfo_t** structure (Listing 11). The meaning of **dataZero** and **strides** fields

```
template <typename real_t>
struct arrinfo_t
{
  // member fields:
  real_t * const dataZero;
  const ptrdiff_t *strides;
```

Listing 11: **lgrngn::arrinfo_t** definition.

match those of equally-named methods of the Blitz++ Array class. Quoting Blitz++ documentation (Veldhuizen, 2005): „**dataZero** is a pointer to the element $(0,0,\ldots,0)$, even if such an element does not exist in the array. What's the point of having such a pointer? Say you want to access the element (i,j,k). If you add to the pointer the dot product of (i,j,k) with the stride vector **stride**, you get a pointer to the element (i,j,k)." Using **arrinfo_t** as the type for API function arguments makes the library potentially compatible with a wide range of array containers, Blitz++ being just an example. In addition, no assumptions are made with respect to array index ranges or dimension ordering, what allows the li-

brary to operate on array slabs (e.g. array segments excluding the so-called halo regions) and both row- and column-major storage.

The state of the Lagrangian component of the model (notably, the values of particle attributes) is stored in an instance of the **lgrngn::particles_t** class (see Listing 12). Internally, the Lagrangian calculations are implemented using the Thrust library[3] which, among other, allows to run the particle-based simulations either on CPU[s] or on a GPU. The second template parameter of **lgrngn::particles_t** is the type of the backend to be used by the Thrust library, and as of current release it has three possible values: serial, OpenMP, or CUDA (cf. Listing 13 with definition of the **backend_t** enumeration). The OpenMP[4] backend offers multi-threading using multiple CPU cores and/or multiple CPUs. The CUDA[5] backend enables the user to perform the computations on a GPU. The serial backend does single-thread computations on a CPU. The "backend-aware" **particles_t<real_t, backend>** inherits from "backend-unaware" **particles_proto_t<real_t>** (definition not shown) what allows to use a single pointer to **particles_proto_t** with different backends (as used in the return value of **lgrngn::factory()** discussed below).

Initialisation, time-stepping, and data output is performed by calling **particles_t**'s methods whose signatures are given in Listing 12 and discussed in the following three paragraphs.

The **particles_t::init()** method performs the initialisation steps described in Sect. 5.1.6 and is intended to be called once at the beginning of the simulation. The first three arguments are mandatory and should point to the θ, r_v and ρ_d fields of the Eulerian component of the solver. The next arguments should point to the Courant number field components. The number of components depends on the dimensionality of the modelling framework, and ranges from zero (parcel framework) up to three (3-D simulation). The Courant number components are expected to be discretised on the Arakwa-C grid, thus for the 2-D case **courant_1**'s shape is **(nx+1)** $\times$ **nz** and **courant_2**'s shape is **nx** $\times$ **(nz+1)**.

Time-stepping is split into two methods: **particles_t::step_sync()** and **particles_t::step_async()**. The former covers representation of the processes that alter the Eulerian fields (i.e. phase changes). The latter covers all other processes (transport of particles, sedimentation, and coalescence) which may be computed asynchronously, for example, while the Eulerian model calculates advection of the Eulerian fields. Both methods take a reference to an instance of **lgrngn::opts_t** as their first argument. Among arguments of **step_sync()**, only the first three are mandatory. The passed θ and r_v fields will be overwritten by the method. The Courant field components need to be specified only if

[3] http://thrust.github.io/
[4] http://openmp.org/
[5] http://nvidia.com/

```
template <typename real_t, backend_t backend>
struct particles_t: particles_proto_t<real_t>
{
  // initialisation
  void init(
    const arrinfo_t<real_t> th,
    const arrinfo_t<real_t> rv,
    const arrinfo_t<real_t> rhod,
    const arrinfo_t<real_t> courant_1,
    const arrinfo_t<real_t> courant_2,
    const arrinfo_t<real_t> courant_3
  );

  // time-stepping methods
  void step_sync(
    const opts_t<real_t> &,
    arrinfo_t<real_t> th,
    arrinfo_t<real_t> rv,
    const arrinfo_t<real_t> courant_1,
    const arrinfo_t<real_t> courant_2,
    const arrinfo_t<real_t> courant_3,
    const arrinfo_t<real_t> rhod
  );
  real_t step_async(
    const opts_t<real_t> &
  );

  // diagnostic methods
  void diag_sd_conc();
  void diag_dry_rng(
    const real_t &r_mi, const real_t &r_mx
  );
  void diag_wet_rng(
    const real_t &r_mi, const real_t &r_mx
  );
  void diag_dry_mom(const int &k);
  void diag_wet_mom(const int &k);
  real_t *outbuf();

  // ...
```

Listing 12: **lgrngn::particles_t** definition.

```
enum backend_t { serial, OpenMP, CUDA };
```

Listing 13: **lgrngn::backend_t** definition.

```
template <typename real_t>
particles_proto_t<real_t> *factory(
  const backend_t,
  const opts_init_t<real_t> &
);
```

Listing 14: **lgrngn::factory()** signature.

the Eulerian component of the model solves air dynamics (they are omitted in the case of the kinematic framework used in examples in this paper). The last argument pointing to a ρ_{d} array is also optional and needs to be specified only if the Eulerian framework allows the density to vary in time. The **step_async()** method returns accumulated rain flux through the bottom of the domain.

The **particles_t**'s methods prefixed with **diag_** provide a mechanism for obtaining statistical information on the droplet parameters gridded on the Eulerian component mesh. The **particles_t::diag_sd_conc()** method calculates the concentration of computational particles per cell. The **particles_t::diag_dry_mom()** and **particles_t::diag_wet_mom()** calculate statistical moments of the dry and wet size spectra respectively. The kth moment M of the dry (d) or wet (w) spectrum is defined here as:

$$M^{[k]}_{\mathrm{d,\,w}} = (\rho_{\mathrm{d}}\Delta V)^{-1} \sum_{\substack{i \in \text{grid cell} \\ r_{\mathrm{d},w_{[i]}} \in [r_{\mathrm{mi}}, r_{\mathrm{mx}}]}} N_{[i]} r^k_{\mathrm{d},w_{[i]}} \tag{29}$$

where the index i traverses all computational particles and N is the particle multiplicity. The moment number k is chosen through the methods' argument **k**. The range of radii $[r_{\mathrm{mi}}, r_{\mathrm{mx}}]$ over which the moments are calculated is chosen by calling **diag_dry_rng()** or **diag_wet_rng()** before calls to **diag_dry_mom()** and **diag_wet_mom()**, respectively. The **particles_t::outbuf()** method stores the calculated fields in an output buffer and returns a pointer to the first element of this buffer.

The last element of the particle-based scheme's API is the **factory()** function. It returns a pointer to a newly allocated instance of the **particles_t** class. Its arguments are the backend type (see Listing 13) and the scheme's options grouped in the **opts_init_t** structure (see Listing 9). The purpose of introducing the **lgrngn::factory()** function is twofold. First, it makes the backend choice a runtime mechanism rather than a compile-time one (backend is one of the compile-time template parameters of **particles_t**). Second, it does report an error if the library was compiled without CUDA (GPU) or OpenMP (multi-threading) backend support.

5.2.2 Example calling sequence

Figure 7 depicts an example calling sequence for the particle-based scheme's API. The API calls are split among the adjustments and output steps of the solver. The rhs steps are presented in the diagram, but here they refer to forcings extrinsic with respect to the cloud microphysics scheme (e.g. the relaxation terms in the set-up described in Sect. 2.2).

In the case of bulk schemes (Figs. 3 and 5) both the solver and library flow control was handled by a single thread (or a group of threads performing the same operations in case of domain decomposition). Here, there are two separate threads (or a group of solver threads plus one library thread in case of domain decomposition). The synchronisation between the solver and the library threads is depicted in the diagram with "wait for ..." labels.

In the presented calling sequence, the diagnostic methods are only called within the output step. Depending on the modelling framework, such calls may also be needed in every timestep, for example, to provide data on particle surface for a radiative-transfer component, or the data on particle mass for a dynamical component of the solver. Note that a single call to **diag_dry/wet_rng()** may be followed by multiple calls to **diag_dry/wet_mom()** as depicted by nesting the "for each moment" loop within the "for each size range" loop.

5.3 Implementation overview

The Lagrangian component of the model is implemented using the Thrust library (Hoberock and Bell, 2010). Consequently, all parallelisation logic is hidden behind the Thrust API calls. The parallelisation is obtained by splitting the computational-particle population among several computational units using shared memory. Thrust allows to compile the same code for execution on multiple parallel architectures including general-purpose GPUs (via CUDA) and multi-core CPUs (via OpenMP). The implemented particle-based scheme is particularly well suited for running in a set-up where the Eulerian computations are carried out on a CPU, and the Lagrangian computations are delegated to a GPU. That is due to:

- the low data exchange rate between these two components (there is never a need to transfer the state of all computational particles to the Eulerian component residing in the main memory, only the aggregated size spectrum parameters defined per each grid box are needed);
- the possibility to perform part of the microphysics computations asynchronously, simultaneously with other computations carried out on CPU(s) (cf. Sec. 5.1.3).

Since the version of CUDA compiler available at the time of development did not support C++11, the particle-based scheme was implemented using C++03 constructs only. Furthermore, the CUDA compiler does not support all C++ constructs used by the Boost.units library. For this reason, a **fake_units** drop-in replacement for Boost.units was written and is shipped with *libcloudph++*. It causes all quantities in the program to behave as dimensionless. It is included instead of Boost.units only if compiling the CUDA backend. Consequently, the particle-based scheme's code is checked for unit correctness while compiling other backends.

The asynchronous launch/wait logic is left to be handled by the caller. In the example program *icicle* (see Appendix C), it is implemented using the C++11's **std::async()** call.

Both in the case of GPU and CPU configurations, the Mersenne Twister (Matsumoto and Nishimura, 1998) ran-

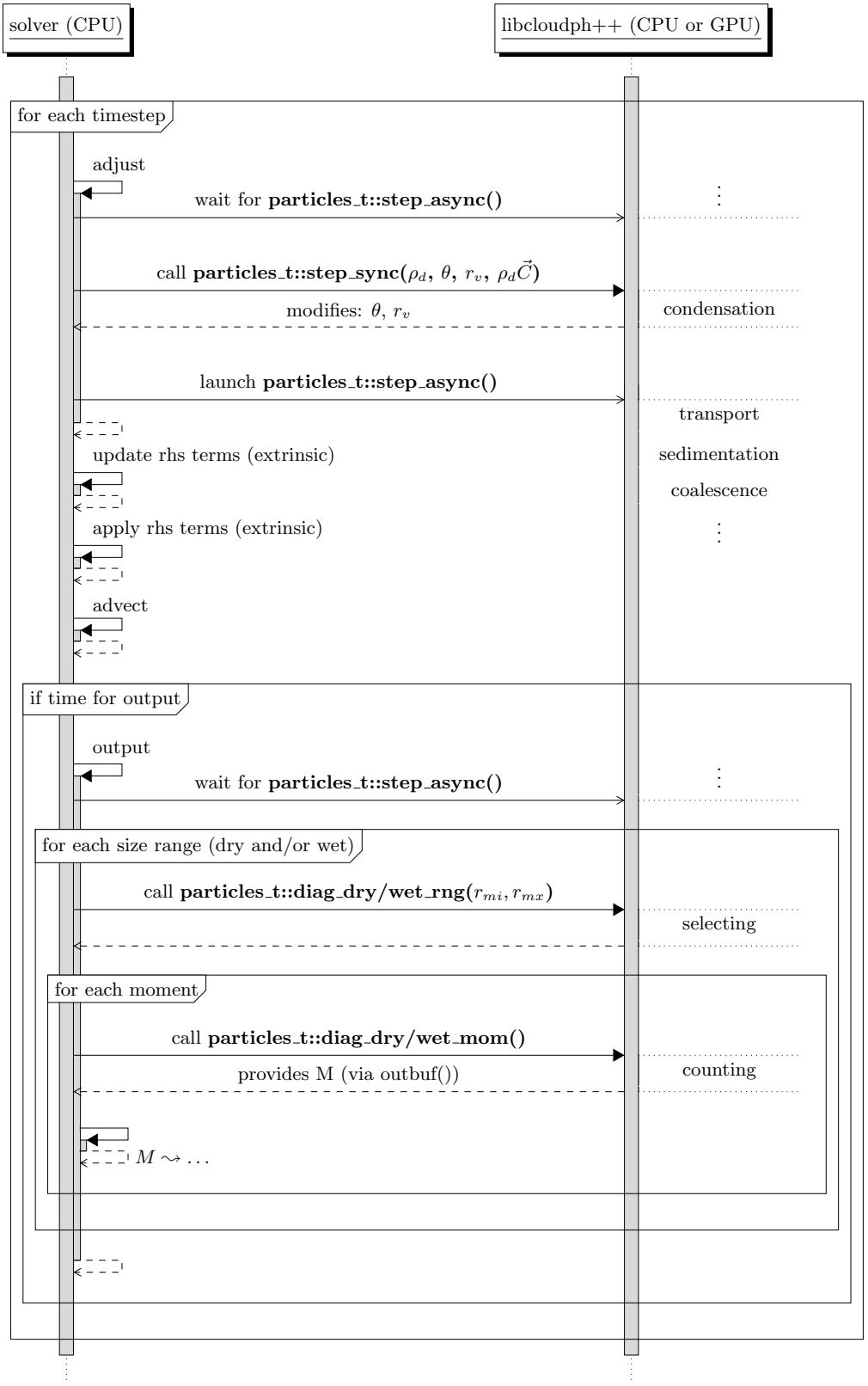

Figure 7. Sequence diagram of *libcloudph++* API calls for the particle-based scheme and a prototype transport equation solver. Diagram discussed in Sect. 5.2.2. See also caption of Fig. 2 for description or diagram elements.

dom number generator is used. If using GPU, the CUDA cuRAND's **MTGP32** is used offering parallel execution with multiple random number streams. If not using GPU, the C++11 **std::mt19937** is used and the random number generation is done by a single thread only, even if using OpenMP.

5.4 Example results

Figures 8 and 9 present results from an example simulation with the particle-based scheme performed using the framework described in section 2. The simulations are analogous to those discussed in Sects. 3.4 (single-moment) and 4.4 (double-moment). As before, the plots are for the thirtieth minute of the simulation time (excluding the two-hour-long spin-up period). The initial mean concentration of computational particles was set to 64 per cell. The number of substeps was set to 10 for both condensation and coalescence. The geometric coalescence kernel was used.

Figure 8 depicts aerosol, cloud, and rain properties obtained by calculating moments of the particle size distribution in each grid cell. In addition to quantities corresponding to the bulk model variables r_c, r_r (cf. Figs 4 and 6) and n_c and n_r (cf. Fig. 6), Fig. 8 features plots of the effective radius (ratio of the third to the second moment of the size spectrum) and the aerosol concentration. The distinction between aerosol particles, cloud droplets, and rain drops is made using radius thresholds of 0.5 and 25 $\mu\mathrm{m}$ for aerosol/cloud and cloud/rain boundaries, respectively. The noise in most panels comes from sampling errors of the particle-based scheme; these errors get smaller with increasing number of computational particles used (not shown). The cloud water content and cloud droplet concentration plots both show strong similarities to the results of simulation using the double-moment scheme (Fig. 6). The increase with height of cloud water content, the approximately constant with height drop concentration, presence of the maximum droplet concentration near the updraught axis, and presence of the cloud hole are all evident in both the particle-based and the double-moment simulations. The range of values of the rain water content and the rain drop concentration predicted by the particle-based model roughly matches those of the double-moment scheme, yet the level of agreement is much smaller than in the case of cloud water. For example, the maximum rain water content in the double-moment simulation is located in the centre of the downdraught, whereas this location features virtually no rain in the particle-based simulation. The two schemes agree with respect to the vertical extent of the drizzle shaft as it vanishes at about 300 m above the bottom boundary of the domain in both cases.

The plot of the effective radius in Fig. 8 shows the gradual increase of drop sizes from the cloud base up to the top of the cloud. The effective radius plot features the smoothest gradients among all presented plots. This is likely due to the fact that unlike other plotted quantities, the effective radius is an intensive parameter and hence is not proportional to the drop concentration which inherits random fluctuations of the initial aerosol concentrations. The aerosol concentration demonstrates anticipated presence of the interstitial aerosol within the cloud. The regions of largest rain water content correspond to regions of lowered aerosol concentrations, both within and below the cloud. This likely demonstrates the effect of scavenging of aerosol particles by the drizzle drops, most likely overpredicted by the geometric collision kernel applied in the simulation.

The ten black squares overlaid on each plot in Fig. 8 show locations of the regions for which the wet and dry particle size spectra are plotted in Fig. 9. The ten locations are composed of 3×3 grid cells each. The spectra plotted in Fig. 9 are all averages over the 3×3 cell regions. The dry spectra are composed of 40 bins in an isologarithmic layout from 1 nm to 10 $\mu\mathrm{m}$. The wet spectra are composed of 25 bins extending the above range up to 100 $\mu\mathrm{m}$. Each square in the Fig. 8 and its corresponding panel in Fig. 9 is labelled with a letter (a to j). All panels in Fig. 9 contain two vertical lines at 0.5 and 25 $\mu\mathrm{m}$ that depict the threshold values of particle wet radius used to differentiate between aerosol, cloud droplets, and rain drops.

To match the pathway of cloud evolution, we shall discuss the panels in Fig. 9 counterclockwise, starting from panel (i) which presents data on the aerosol size spectrum in the updraught below cloud base. There, the wet spectrum plotted with the thick blue line is slightly shifted towards larger sizes than the dry spectrum plotted with the thin red line. This shift corresponds to humidification of the hygroscopic aerosol. Panels (g) and (e) show how the wet spectrum evolves while the updraught lifts the particles across the cloud base causing the largest aerosol to be activated and to form cloud droplets. Panel (c) shows a bimodal wet spectrum with an unactivated aerosol mode to the left and the cloud droplet mode just below 10 $\mu\mathrm{m}$. Panel (a) depicts the near-cloud-top conditions and reveals that some of the cloud droplets had already grown pass the 25 $\mu\mathrm{m}$ threshold, likely through collisional growth. Such drops have significant fall velocities which causes the air in the upper part of the domain to become void of the largest aerosol. This is evident from the shape of the dry spectrum in panel (b) depicting conditions above the downdraught. Panel (d) and panel (c) show size spectra at the same altitude of about 100 m above cloud base. Their comparison reveals that the spectrum of cloud droplets in the downdraught (panel d, edge of the cloud hole) is much wider than near the updraught axis (panel c). Finally, panels (f), (h), and (j) show gradual evaporation of drizzle and cloud droplets back to aerosol-sized particles.

6 Performance evaluation

Computational cost of a microphysics scheme is one of the key factors determining its practical applicability. Here, we present a basic analysis of the computational cost of the three

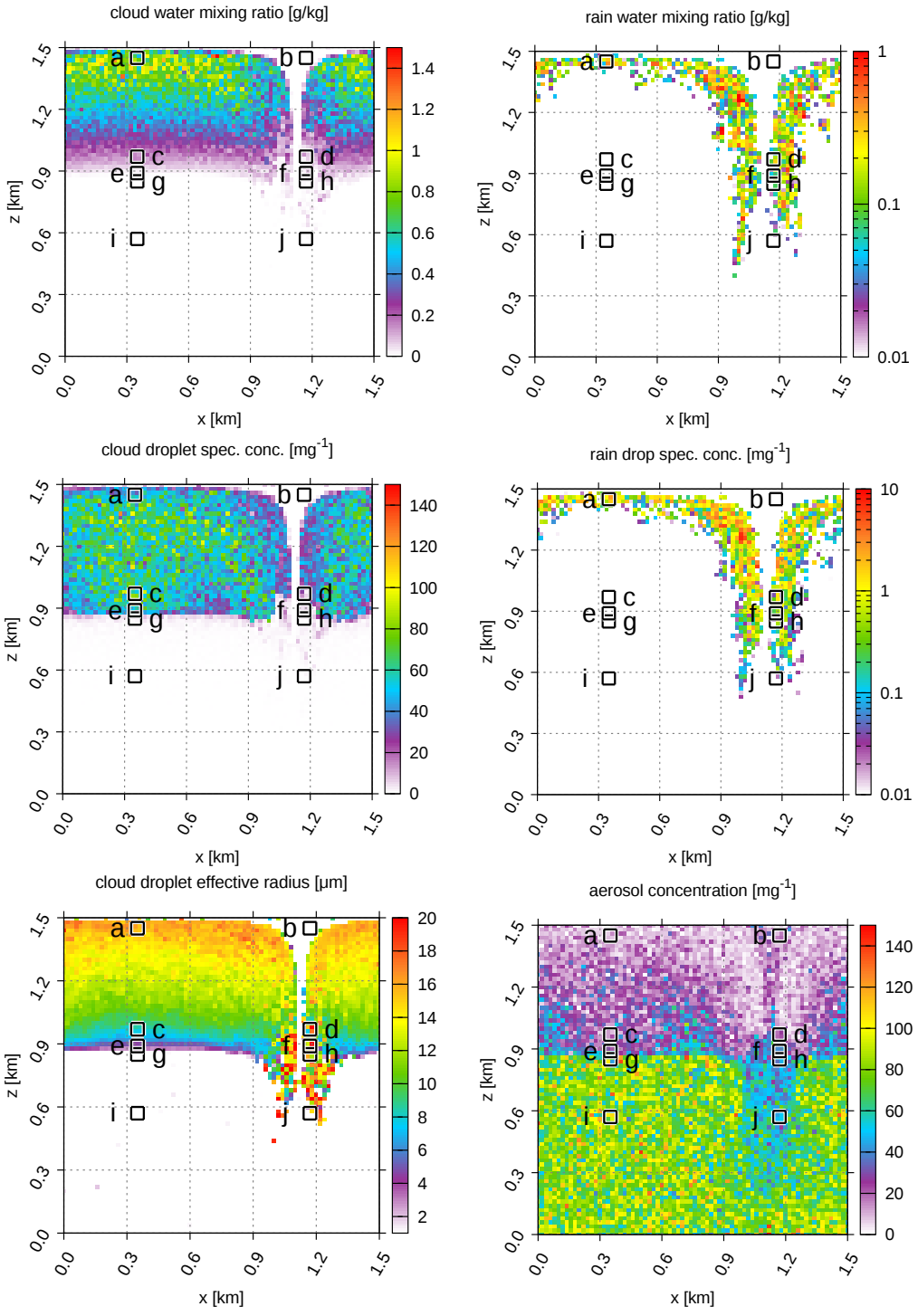

Figure 8. Example results from a 2-D kinematic simulation using the particle-based scheme. All panels depict model state after 30 min simulation time (excluding the spin-up period). The black overlaid squares mark grid cells for which the dry and wet size spectra are shown in Fig. 9. See Sect. 5.4 for discussion.

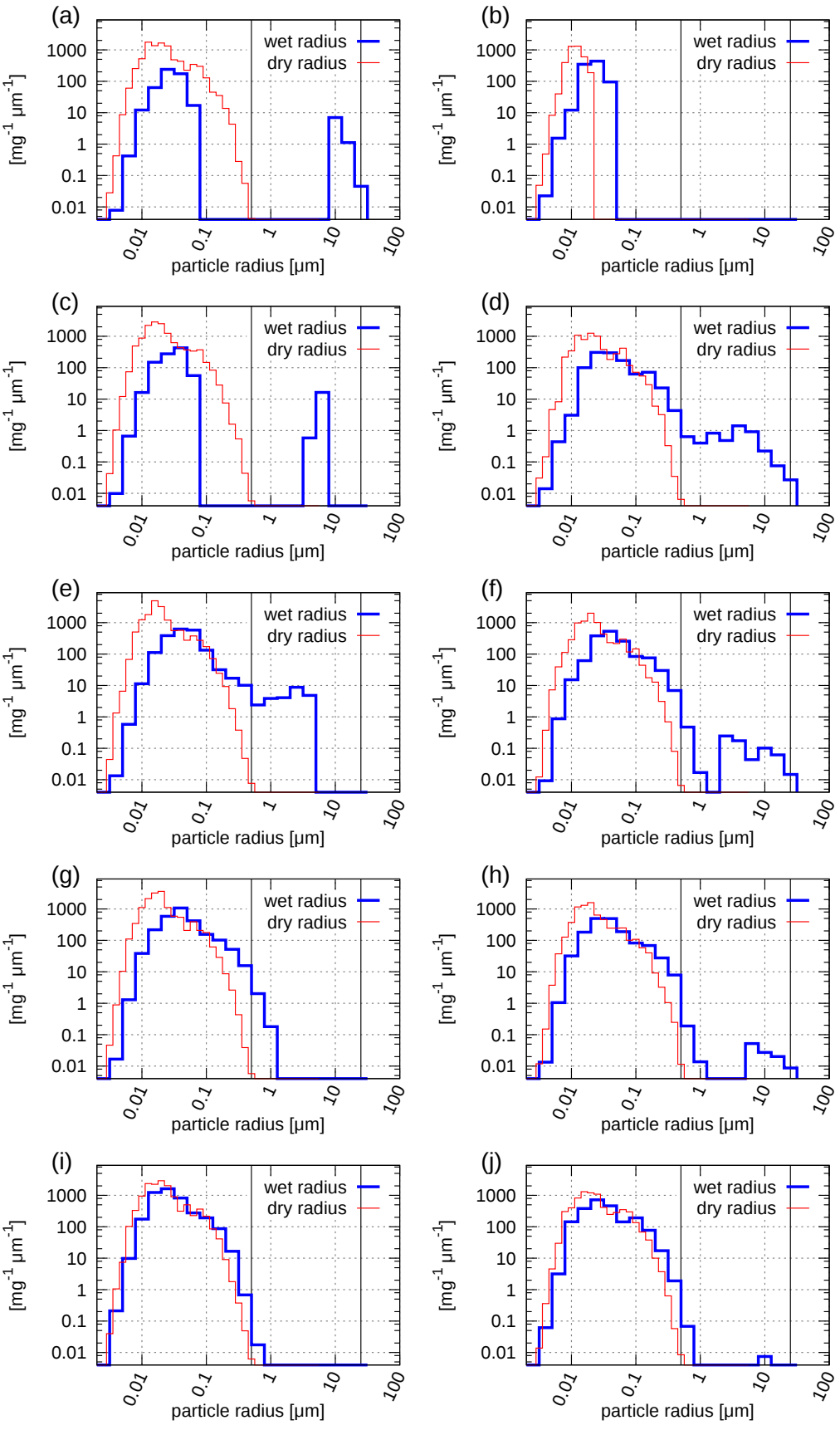

Figure 9. Plots of dry and wet size spectra for ten locations within the model domain. The locations and their labels **(a–j)** are overlaid on plots in Fig. 8. The vertical bars at 0.5 and 25 µm indicate the range of particle wet radii which is associated with cloud droplets. See Sect. 5.4 for discussion.

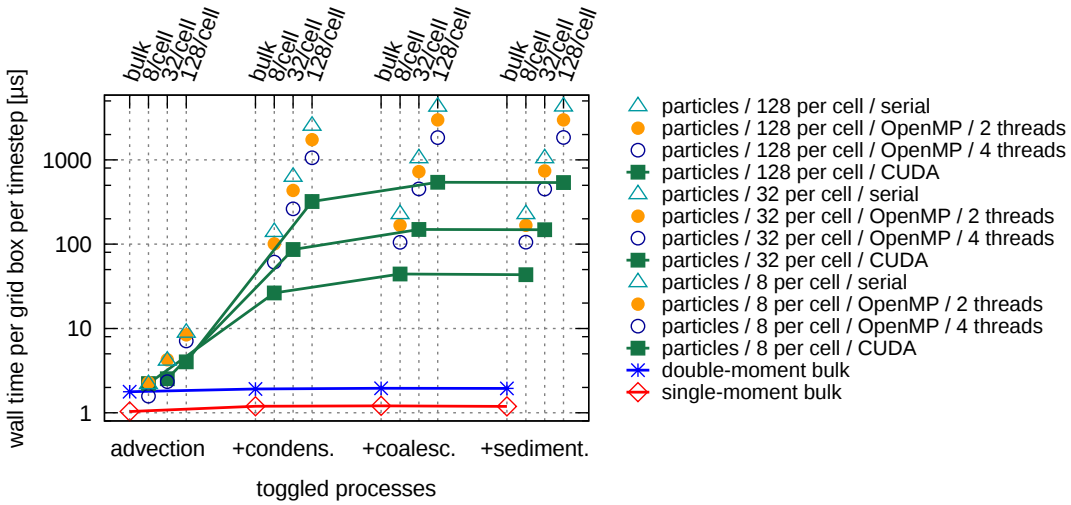

Figure 10. Computational cost of the three microphysics schemes expressed as wall-clock time per timestep per grid box. Values measured for different settings of process-toggling options shown (bottom horizontal axis). Results obtained with the particle-based scheme are grouped by the number of computational particles used (upper helper horizontal axes). See Sect. 6 for discussion.

schemes presented in this paper. The analysis is based on timing of simulations carried out with the kinematic framework and the simulation set-up described in Sect. 2.1 using the *icicle* tool described in Appendix C. In order to depict the contributions of individual elements of the schemes, all simulations were repeated with four sets of process-toggling options:

- advection only,
- advection and phase changes,
- advection, phase changes, and coalescence,
- all above plus sedimentation.

For the particle-based scheme, the advection-only runs include transport of particles and the Eulerian fields (moisture and heat).

Simulations were performed with a 6-core AMD Phenom II CPU and a 96-core nVidia Quadro 600 GPU (an example 2010 prosumer desktop computer). The CPU code was compiled using GCC 4.8 with -Ofast, -march = native and -DNDEBUG options enabled. The GPU code was compiled with nvcc 5.5 with -arch = sm_20 and -DNDEBUG options enabled. No data output was performed.

In order to eliminate from the reported values the time spent on simulation startup, all simulations were repeated twice, performing a few timesteps in the first run and a dozen timesteps in the second run. The long and short run times were subtracted and the result was normalised by the difference in number of timesteps.

In order to reduce the influence of other processes on the wall-clock timing, all simulations were additionally repeated three times, and the shortest measured time is reported.

The particle-based simulations were performed with three different mean densities of computation particles, 8, 32 and 128 per grid cell, and with four "backend" settings:

- serial backend,
- OpenMP backend using 2 threads,
- OpenMP backend using 4 threads,
- CUDA backend using the GPU.

The test was completed for single-precision arithmetics. The GPU used offered about three times higher performance at single precision. Higher-performance GPU hardware available in computing centres is expected to deliver similar performance for double precision. Execution times for CPU-only calculations hardly change when switching from double to single precision.

Figure 10 presents measured wall-clock times for the four sets of processes (bottom x axis labels) and for all three schemes (different colours and symbols). For simulations with all processes turned on, it takes the double-moment scheme roughly twice longer than the single-moment scheme to advance the solution by one timestep. The particle-based scheme may be anything from about ten- to over hundred-times more costly than the double-moment bulk scheme depending on its settings.

Figure 10 also shows how the execution time of the particle-based scheme depends on the backend choice and on the number of computational particles used. The execution time is also dependent on the number of subtimesteps used for phase changes and coalescence (not shown, 10 subtimesteps per one advective step were used here). It is also evident that computations of phase changes take most

of the simulation time for particle-based simulations. The code responsible for the iterative implicit solution of the drop-growth equation is thus the first candidate for optimisation (e.g., through employment of a faster-converging root-finding algorithm and through introduction of adaptive time-stepping).

Arguably, the most striking feature depicted in Fig. 10 is the order-of-magnitude speedup between serial execution times for CPU and the GPU execution times. Even compared to the four-thread OpenMP runs, the GPU backend offers a threefold speedup. It is worth reiterating here the two reasons why the particle-based scheme is particularly well-suited for GPUs. First, the large body of data defining the state of all particles never leaves the GPU memory (the GPU-CPU transfer bandwidth is often a major issue for the performance of GPU codes). Here, all data that are transferred from the GPU are first gridded onto the Eulerian mesh before being sent from GPU to the main memory. Second, a significant part of the computations (i.e. everything but phase changes) may be computed asynchronously, leaving all but one CPU available for other tasks of the solver (one thread is busy controlling the GPU).

Finally, Fig. 10 also depicts the linear scaling of the computational cost of the particle-based method with the number of computational particles (cf. Sect. 5). Regardless of the backend choice, increasing the mean number of particles per cell from 8 to 32 to 128 gives a linear increase of wall-time as seen in the logarithmic scale of the plot.

The library is still at its initial stage of development, and improvements in performance are expected.

7 Summary

The main goal of developing *libcloudph++*, has been to offer the community a set of reusable software components of applicability in modern cloud modelling. Incorporation of the double-moment bulk and the particle-based schemes makes the library applicable for research on the widely discussed indirect effects of aerosol on climate.

The implementation of the library was carried out having maintainability and auditability as priorities. This is reflected in:

- the choice of C++ with its concise and modularity-encouraging syntax[6] ;
- the separation of code elements related to the schemes' formulation (formulæ) from other elements of the library (API, numerics);
- the adoption of compile-time dimensional analysis for all physically-meaningful expressions in the code;
- the delegation of substantial part of the library implementation to external libraries (including the dimensional analysis, algorithm parallelisation and GPU hardware handling);
- the hosting of library development and handling of code dissemination through a public code repository.

All above, supported by the choice of the GNU General Public License, underpins our goal of offering reusable code.

[6] As of current release, *libcloudph++* consists of ca. 100 files with a total of ca. 8000 lines of code (LOC) of which ca. 1000 LOC are common to all schemes; ca. 500, 1000, and 4500 LOC are pertaining to the single-moment, double-moment, and particle-based schemes, respectively; ca. 1000 LOC define the Python bindings.

Code availability

The library is released under the GNU General Public License v3.0. The 1.0 release of the library accompanying this publication is available for download as an electronic supplement to the paper and tagged as "1.0.0" at the project repository. See project website for a list of pointers to relevant resources: http://libcloudphxx.igf.fuw.edu.pl/.

In the current development workflow, we employ continuous integration on Linux with GNU *g++*[7] and LLVM *clang++*[8] compilers and on Apple OSX with the Apple *clang++*[9] compiler. Consequently, these are considered the supported platforms.

The Supplement related to this article is available online at doi:10.5194/gmd-0-1-2015-supplement.

Appendix A: Common concepts and nomenclature

This section presents some key elements of a mostly standard approach to analytic description of motion of moist air, particularly in the context of modelling of the warm-rain processes. It is given for the sake of completeness of the formulation and to ease referencing particular equations from within the text and the source code.

Governing equations

There are three key types of matter considered in the model formulation and their densities ρ_i and mass mixing ratios r_i are defined as follows:

ρ_d	dry air	
$\rho_\mathrm{v} = r_\mathrm{v}\rho_\mathrm{d}$	water vapour	(A1)
$\rho_\mathrm{l} = r_\mathrm{l}\rho_\mathrm{d}$	liquid water	(A2)

[7] http://gcc.gnu.org/
[8] http://llvm.org/
[9] http://apple.com/xcode

The governing equations are the continuity equation for dry air, a conservation law for water vapour, and the thermodynamic equation (see e.g. Vallis, 2006, Sect. 1.6):

$$\partial_t \rho_\mathrm{d} + \nabla \cdot (\boldsymbol{u} \rho_\mathrm{d}) = 0 \tag{A3}$$

$$\frac{D r_\mathrm{v}}{D t} = \dot{r}_\mathrm{v} \tag{A4}$$

$$\frac{D s}{D t} = \frac{\dot{q}}{T} \tag{A5}$$

where s and $\dot{q}$ represent entropy and heat sources, respectively (both defined per unit mass of dry air). The dot notation is used to distinguish variations due to transport and due to thermodynamic processes.

It is assumed already in Eq. (A3) that the presence of moisture and its transformations through phase changes do not influence the density of dry air. Dry-air flow is assumed to act as a carrier flow for trace constituents. This assumption is corroborated by the fact that in the Earth's atmosphere $1 \gg r_\mathrm{v} > r_\mathrm{l}$.

System of transport equations

Equations (A4) and (A5) may be conveniently expressed as a pair of transport equations of a similar form to Eq. (A3).

A continuity equation for water vapour density ρ_v is obtained by summing Eq. (A4) $\cdot \rho_\mathrm{d} + r_\mathrm{v} \cdot$ Eq. (A3):

$$\partial_t (\rho_\mathrm{d} r_\mathrm{v}) + \nabla \cdot (\boldsymbol{u} \rho_\mathrm{d} r_\mathrm{v}) = \rho_\mathrm{d} \dot{r}_\mathrm{v} \tag{A6}$$

Combining Eq. (A5) with the definition of potential temperature $\theta^\star$:

$$\mathrm{d}s = c_p^\star \mathrm{d}(\ln \theta^\star) \tag{A7}$$

gives:

$$c_p^\star \frac{\mathrm{d}\theta^\star}{\mathrm{d}t} = \frac{\theta^\star}{T} \dot{q} \tag{A8}$$

At this point, no assumption is made on the exact form of $\theta^\star$ or $c_p^\star$. Summing Eq. (A3) $\cdot \theta^\star c_p^\star$ and Eq. (A8) $\cdot \rho_\mathrm{d}$ and $\rho_\mathrm{d} \theta^\star \cdot \frac{D}{Dt} c_p^\star = \rho_\mathrm{d} \theta^\star \dot{c}_p^\star$ results in a continuity equation for $\rho_\mathrm{d} c_p^\star \theta^\star$ (akin to energy density):

$$\partial_t (\rho_\mathrm{d} c_p^\star \theta^\star) + \nabla \cdot (\boldsymbol{u} \rho_\mathrm{d} c_p^\star \theta^\star) = \rho_\mathrm{d} \theta^\star \left[\dot{c}_p^\star + \dot{q}/T \right] \tag{A9}$$

Resultant Eqs. (A6) and (A9) share the form of a generalised transport equation (see Smolarkiewicz, 2006, Sect. 4.1):

$$\partial_t (\rho_\mathrm{d} \phi) + \nabla \cdot (\rho_\mathrm{d} \boldsymbol{u} \phi) = \rho_\mathrm{d} \dot{\phi} \tag{A10}$$

representing transport of a quantity ϕ (equal to r_v or $c_p^\star \theta^\star$) by a dry-air carrier flow.

Dry air potential temperature

The way the potential temperature was defined in the preceding section gives a degree of freedom in the choice of $\theta^\star$ and $\dot{q}$. For moist air containing suspended water aerosol, assuming thermodynamic equilibrium and neglecting the expansion work of liquid water, $\mathrm{d}s$ may be expressed as (Eqs. 6.10–6.11 in Curry and Webster, 1999):

$$\mathrm{d}s = \overbrace{c_\mathrm{pd} d(\ln T) - R_\mathrm{d} d(\ln p_d)}^{c_\mathrm{pd} d(ln\theta)} + \underbrace{\left[l_\mathrm{v} \mathrm{d}r_\mathrm{v} + (r_\mathrm{v} c_\mathrm{pv} + r_\mathrm{l} c_\mathrm{l} + r_\mathrm{v} l_\mathrm{v}/T)\, \mathrm{d}T \right]}_{-\mathrm{d}q} / \tag{A11}$$

where $p_\mathrm{d} = \rho_\mathrm{d} R_\mathrm{d} T$ is the partial pressure of dry air, and the potential temperature θ is defined here as:

$$\theta = T \left(\frac{p_{1000}}{p_\mathrm{d}} \right)^{\frac{R_\mathrm{d}}{c_\mathrm{pd}}} \tag{A12}$$

($p_{1000} = 1000\,\mathrm{hPa}$, note that the definition features the dry air pressure as opposed to the total pressure, see e.g. Bryan, 2008; Duarte et al., 2014).

Substituting $c_p^\star = c_\mathrm{pd} = \text{const}$ and $\theta^\star = \theta$ into Eq. (A9) and employing the form of $\dot{q}$ hinted with $-\mathrm{d}q$ in Eq. (A11) gives:

$$\partial_t (\rho_\mathrm{d} \theta) + \nabla \cdot (\boldsymbol{u} \rho_\mathrm{d} \theta) = \frac{-\rho_\mathrm{d} \theta}{c_\mathrm{pd} T} \left[l_\mathrm{v} \dot{r_\mathrm{v}} + \dot{T} \left(r_\mathrm{v} c_\mathrm{pv} + r_\mathrm{l} c_\mathrm{l} + \frac{r_\mathrm{v} l_\mathrm{v}}{T} \right) \right] \tag{A13}$$

Neglecting of all but the $l_\mathrm{v} \dot{r}$ terms on the right-hand side results in an approximation akin to the one employed in Grabowski and Smolarkiewicz (1996) and used herein as well.

Another common choice of $\theta^\star$ and $\dot{q}$ is obtained by putting $\theta^\star = \theta \cdot \exp\left(\frac{-r_\mathrm{v} l_\mathrm{v}}{c_\mathrm{pd} T} \right)$, what results in the $l_\mathrm{v} \mathrm{d}r_\mathrm{v}$ term becoming a part of $c_\mathrm{pd} \mathrm{d}(\ln \theta^\star)$ instead of $-\mathrm{d}q$ in Eq. (A11) (see e.g. Grabowski and Smolarkiewicz, 1990, Sect. 3).

Diagnosing T and p from state variables

The principal role of any cloud-microphysics scheme is to close the equation system defined by Eqs. (A6) and (A13) with a definition of $\dot{r}_v$ linked with a representation of liquid water within the model domain. This requires representation of various thermodynamic processes that depend on temperature and pressure which are diagnosed from the model state variables (i.e. the quantities for which the transport equations are solved). With the approach outlined above, the model state variables are:

r_v water vapour mixing ratio

θ potential temperature

Assuming ρ_d is known (solved by a dynamical core of a model), temperature and pressure may be diagnosed from r_v and θ with:

$$T = \left[\theta \left(\frac{\rho_\mathrm{d} R_\mathrm{d}}{p_{1000}} \right)^{\frac{R_\mathrm{d}}{c_\mathrm{pd}}} \right]^{c_\mathrm{pd}/(c_\mathrm{pd} - R_\mathrm{d})} \tag{A14}$$

$$p = \rho_\mathrm{d} \left(R_\mathrm{d} + r_\mathrm{v} R_\mathrm{v} \right) T \tag{A15}$$

Appendix B: List of symbols

A list of symbols is provided in Table A1.

Appendix C: Example program "icicle"

The example simulations discussed in the text were performed with *icicle*–an implementation of all elements of the example modelling framework presented in Sect. 2, that is: the transport equation solver, the 2-D kinematic framework and the simulation set-up.

Dependencies

The code of *icicle* depends on *libcloudph++*, *libcloudph++*'s sister project *libmpdata++* (Jaruga et al., 2015) and several components of the Boost[10] collection. The *libmpdata++* components solve the transport equations for the Eulerian fields using the MPDATA algorithm (Smolarkiewicz, 2006) and provide data output facility using the HDF5 library[11]. Figure C1 presents dependency tree of *icicle*. Source code of *icicle*, *libmpdata++* and *libcloudph++* is available for download at http://foss.igf.fuw.edu.pl/. The 1.0.0 release tarballs for both *libcloudph++* and *icicle* are provided as an electronic supplement to the paper. All other *icicle* dependencies are available, for instance, as Debian[12] packages. All *icicle* dependencies are free (gratis) software, and all but CUDA (which is an optional dependency) are additionally libre–open sourced, and released under freedom-ensuring licenses.

Compilation

Build automation for *icicle*, *libmpdata++* and *libcloudph++* is handled in a standard way using CMake[13]. In all three cases, a possible command sequence will resemble:

```
$ mkdir build
$ cd build
$ cmake ..
$ make
$ make test
$ sudo make install
```

Usage

Control over simulation options of *icicle* is available via command-line parameters. Most of the options correspond to the fields of the **opts_t** structures of the three microphysics schemes discussed in the paper. A list of general options may be obtained by calling:

```
$ icicle --help
```

and includes, in particular, the **--micro** option that selects the microphysics scheme. Options specific to each of the three available schemes are listed as in the following example:

```
$ icicle --micro=lgrngn --help
```

For the particle-based scheme, the options include such settings as the backend type (serial, OpenMP or CUDA) and the size ranges for which to output the moments of the particle size distribution.

Simulations may be stopped at any time by sending the process a SIGTERM or SIGINT signal (e.g., using the **kill** utility or with **Ctrl+C**). It causes the solver to continue integration up to the end of the current timestep, close the output file, and exit. After executing the simulation, its progress may be monitored for example with **top -H** as the process threads' names are continuously updated with the percentage of work completed.

The Supplement related to this article is available online at doi:10.5194/gmd-0-1-2015-supplement.

Acknowledgements. S. Arabas thanks Shin-ichiro Shima (University of Hyogo, Japan) for introducing to particle-based simulations. We thank Dorota Jarecka (University of Warsaw) and Graham Feingold (NOAA) for insightful discussions and comments to the initial version of the manuscript. We acknowledge contributions to *libcloudph++* code from Piotr Dziekan, Dorota Jarecka and Maciej Waruszewski. Development of *libcloudph++*, *libmpdata++* and *icicle* has been supported by Poland's National Science Centre (Narodowe Centrum Nauki) [decisions no. 2011/01/N/ST10/01483 and 2012/06/M/ST10/00434]. Additional support was provided by the European Union 7 FP ACTRIS (Aerosol, Clouds, and Trace gases Research InfraStructure network) No. 262254. WWG's institution NCAR is operated by the University Corporation for Atmospheric Research under sponsorship of the US National Science Foundation. The authors express their appreciation of the work of the developers of the free/libre/open-source software which served as a basis for implementation of the presented library (see Sect. C for a list). We would like to express our admiration to the way the Clang[14] C++ compiler improved the comfort of development and debugging of heavily-templated code based on libraries such as Boost.units and Blitz++. All figures were generated using gnuplot[15]. Development of *libcloudph++* continuesly benefits from the computational services offered by Travis at their continuous-integration platform.

References

Ahnert, K. and Mulansky, M.: Boost.Numeric.Odeint: solving ordinary differential equations, in: Boost Library Documenta-

[10] http://boost.org/
[11] http://hdfgroup.org/
[12] http://debian.org/
[13] http://cmake.org/
[14] http://clang.llvm.org/
[15] http://gnuplot.info/

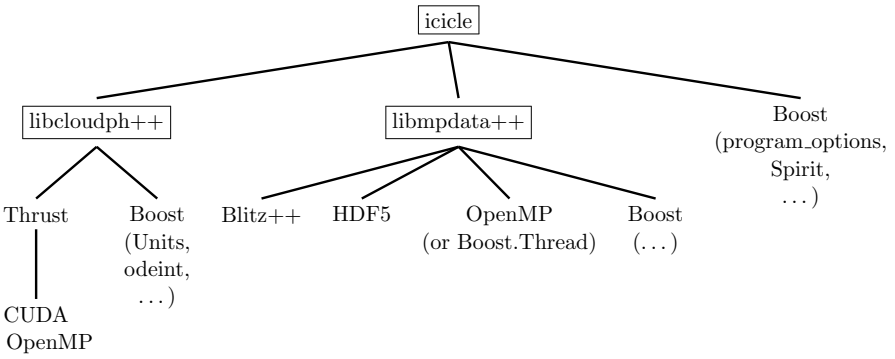

Figure C1. A tree of libcloudph++'s and icicle's major dependencies. In addition to these libraries, several components require C++11 compiler and CMake at build time.

tion, available at: http://www.boost.org/doc/libs/ (last access: 15 November 2014), 2013.

Allen, G., Coe, H., Clarke, A., Bretherton, C., Wood, R., Abel, S. J., Barrett, P., Brown, P., George, R., Freitag, S., McNaughton, C., Howell, S., Shank, L., Kapustin, V., Brekhovskikh, V., Kleinman, L., Lee, Y.-N., Springston, S., Toniazzo, T., Krejci, R., Fochesatto, J., Shaw, G., Krecl, P., Brooks, B., McMeeking, G., Bower, K. N., Williams, P. I., Crosier, J., Crawford, I., Connolly, P., Allan, J. D., Covert, D., Bandy, A. R., Russell, L. M., Trembath, J., Bart, M., McQuaid, J. B., Wang, J., and Chand, D.: South East Pacific atmospheric composition and variability sampled along 20° S during VOCALS-REx, Atmos. Chem. Phys., 11, 5237–5262, doi:10.5194/acp-11-5237-2011, 2011.

Andrejczuk, M., Reisner, J., Henson, B., Dubey, M., and Jeffery, C.: The potential impacts of pollution on a nondrizzling stratus deck: does aerosol number matter more than type?, J. Geophys. Res., 113, D19204, doi:10.1029/2007JD009445, 2008.

Andrejczuk, M., Grabowski, W., Reisner, J., and Gadian, A.: Cloud-aerosol interactions for boundary layer stratocumulus in the Lagrangian Cloud Model, J. Geophys. Res., 115, D22214, doi:10.1029/2010JD014248, 2010.

Arabas, S. and Pawlowska, H.: Adaptive method of lines for multi-component aerosol condensational growth and CCN activation, Geosci. Model Dev., 4, 15–31, doi:10.5194/gmd-4-15-2011, 2011.

Arabas, S. and Shima, S.: Large Eddy simulations of trade-wind cumuli using particle-based microphysics with Monte-Carlo coalescence, J. Atmos. Sci., 70, 2768–2777, doi:10.1175/JAS-D-12-0295.1, 2013.

Bott, A.: A flux method for the numerical solution of the stochastic collection equation, J. Atmos. Sci., 55, 2284–2293, doi:10.1175/1520-0469(1998)055<2284:AFMFTN>2.0.CO;2, 1998.

Brokken, F.: C++ Annotations, Center of Information Technology, University of Groningen, available at: http://cppannotations.sf.net/ (last access: 15 November 2014), 2013.

Bryan, G.: On the computation of pseudoadiabatic entropy and equivalent potential temperature, Mon. Weather Rev., 136, 5239–5245, doi:10.1175/2008MWR2593.1, 2008.

Castellano, N. E., and Ávila, E. E.: Vapour density field of a population of cloud droplets, J. Atmos. Sol.-Terr. Phy., 73, 2423–2428, doi:10.1016/j.jastp.2011.08.013, 2011.

Clift, R., Grace, J., and Weber, M.: Bubbles, Drops, and Particles, Academic Press, New York, 1978, reprinted by Dover Publications, 2005.

Crowe, C., Schwarzkopf, J., Sommerfeld, M., and Tsuji, Y.: Multiphase flows with droplets and particles, 2nd edn., CRC Press, Boca Raton, FL, USA, 2012.

Curry, J. and Webster, P.: Thermodynamics of Atmospheres and Oceans, Academic Press, 1999.

Duarte, M., Almgren, A., Balakrishnan, K., Bell, J., and Romps, D.: A Numerical Study of Methods for Moist Atmospheric Flows: Compressible Equations, Mon. Weather Rev., 142, 4269–4283, doi:10.1175/MWR-D-13-00368.1, 2014.

Easterbrook, S. M. and Johns, T. C.: Engineering the software for understanding climate change, Comput. Sci. Eng., 11, 65–74, doi:10.1109/MCSE.2009.193, 2009.

Fernández-Díaz, J. M., Braña, M. A. R., García, B. A., Muñiz, C. G.-P., and Nieto, P. J. G.: The goodness of the internally mixed aerosol assumption under condensation-evaporation, Aerosol Sci. Tech., 31, 17–23, doi:10.1080/027868299304327, 1999.

Golaz, J.-C., Larson, V., and Cotton, W.: A PDF-based model for boundary layer clouds. Part 1: Method and model description, J. Atmos. Sci., 59, 3540–3551, doi:10.1175/1520-0469(2002)059<3540:APBMFB>2.0.CO;2, 2002.

Grabowski, W. and Smolarkiewicz, P.: Monotone finite-difference approximations to the advection-condensation problem, Mon. Weather Rev., 118, 2082–2097, doi:10.1175/1520-0493(1990)118<2082:MFDATT>2.0.CO;2, 1990.

Grabowski, W. and Smolarkiewicz, P.: Two-time-level semi-lagrangian modeling of precipitating clouds, Mon. Weather Rev., 124, 487–497, doi:10.1175/1520-0493(1996)124<0487:TTLSLM>2.0.CO;2, 1996.

Grabowski, W. and Smolarkiewicz, P.: A multiscale anelastic model for meteorological research, Mon. Weather Rev., 130, 939–956, doi:10.1175/1520-0493(2002)130<0939:AMAMFM>2.0.CO;2, 2002.

Grabowski, W. W. and Wang, L.-P.: Growth of cloud droplets in a turbulent environment, Annu. Rev. Fluid Mech., 45, 293–324, doi:10.1146/annurev-fluid-011212-140750, 2013.

Hoberock, J. and Bell, N.: Thrust: a parallel template library, available at: http://thrust.github.io/ (last access: 15 November 2014), 2010.

Ince, D., Hatton, L., and Graham-Cumming, J.: The case for open computer programs, Nature, 482, 485–488, doi:10.1038/nature10836, 2012.

Jarecka, D., Grabowski, W., Morrison, H., and Pawlowska, H.: Homogeneity of the subgrid-scale turbulent mixing in large-Eddy simulation of shallow convection, J. Atmos. Sci., 70, 2751–2767, doi:10.1175/JAS-D-13-042.1, 2013.

Jarecka, D., Arabas, S., Del Vento, D.: Python bindings for libcloudph++, arXiv:1504.01161, 2015.

Jaruga, A., Arabas, S., Jarecka, D., Pawlowska, H., Smolarkiewicz, P., and Waruszewski, M.: llibmpdata++ 1.0: a library of parallel MPDATA solvers for systems of generalised transport equations, Geosci. Model Dev., 8, 1005–1032, doi:10.5194/gmd-8-1005-2015, 2015.

Kessler, E.: On the continuity and distribution of water substance in atmospheric circulations, Atmos. Res., 38, 109–145, doi:10.1016/0169-8095(94)00090-Z, 1995.

Khairoutdinov, M. and Kogan, Y.: A new cloud physics parameterization in a large-Eddy simulation model of marine stratocumulus, Mon. Weather Rev., 128, 229–243, doi:10.1175/1520-0493(2000)128<0229:ANCPPI>2.0.CO;2, 2000.

Khvorostyanov, V. and Curry, J.: Terminal velocities of droplets and crystals: power laws with continuous parameters over the size spectrum, J. Atmos. Sci., 59, 1872–1884, doi:10.1175/1520-0469(2002)059<1872:TVODAC>2.0.CO;2, 2002.

Khvorostyanov, V. and Curry, J.: Aerosol size spectra and CCN activity spectra: Reconciling the lognormal, algebraic, and power laws, J. Geophys. Res., 111, D12202, doi:10.1029/2005JD006532, 2006.

Laaksonen, A., Vesala, T., Kulmala, M., Winkler, P. M., and Wagner, P. E.: Commentary on cloud modelling and the mass accommodation coefficient of water, Atmos. Chem. Phys., 5, 461–464, doi:10.5194/acp-5-461-2005, 2005.

Lebo, Z. J. and Seinfeld, J. H.: A continuous spectral aerosol-droplet microphysics model, Atmos. Chem. Phys., 11, 12297–12316, doi:10.5194/acp-11-12297-2011, 2011.

Marcolli, C., Luo, B. P., Peter, Th., and Wienhold, F. G.: Internal mixing of the organic aerosol by gas phase diffusion of semivolatile organic compounds, Atmos. Chem. Phys., 4, 2593–2599, doi:10.5194/acp-4-2593-2004, 2004.

Matsumoto, M. and Nishimura, T.: Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM T. Model. Comput. S., 8, 3–30, doi:10.1145/272991.272995, 1998.

Mayer, B. and Kylling, A.: Technical note: The libRadtran software package for radiative transfer calculations–description and examples of use, Atmos. Chem. Phys., 5, 1855–1877, doi:10.5194/acp-5-1855-2005, 2005.

McFarquhar, G.: Raindrop size distribution and evolution, in: Rainfall: State of the Science, edited by: Testik, F. Y. and Gebremichael, M., Washington, D. C., USA, AGU, 49–59, doi:10.1029/GM191, 2010.

Mitra, S., Brinkmann, J., and Pruppacher, H.: A wind tunnel study on the drop-to-particle conversion, J. Aerosol Sci., 23, 245–256, doi:10.1016/0021-8502(92)90326-Q, 1992.

Morin, A., Urban, J., Adams, P., Foster, I., Sali, A., Baker, D., and Sliz, P.: Shining light into black boxes, Science, 336, 159–160, doi:10.1126/science.1218263, 2012.

Morrison, H. and Grabowski, W.: Comparison of bulk and bin warm-rain microphysics models using a kinematic framework, J. Atmos. Sci., 64, 2839–2861, doi:10.1175/JAS3980, 2007.

Morrison, H. and Grabowski, W.: Modeling supersaturation and subgrid-scale mixing with two-moment bulk warm microphysics, J. Atmos. Sci., 65, 792–812, doi:10.1175/2007JAS2374.1, 2008.

Morrison, H., Curry, J., and Khvorostyanov, V.: A new double-moment microphysics parameterization for application in cloud and climate models. Part 1: Description, J. Atmos. Sci., 62, 1665–1677, doi:10.1175/JAS3446.1, 2005.

Muhlbauer, A., Grabowski, W. W., Malinowski, S. P., Ackerman, T. P., Bryan, G. H., Lebo, Z. J., Milbrandt, J. A., Morrison, H., Ovchinnikov, M., Tessendorf, S., Thériault, J. M., and Thompson, G.: Reexamination of the State-of-the-art of cloud modeling shows real improvements, B. Am. Meteorol. Soc., 94, ES45–ES48, doi:10.1175/BAMS-D-12-00188.1, 2013.

Ogura, Y. and Takahashi, T.: Numerical simulation of the life cycle of a thunderstorm cell, Mon. Weather Rev., 99, 895–911, doi:10.1175/1520-0493(1971)099<0895:NSOTLC>2.3.CO;2, 1971.

Pennell, C. and Reichler, T.: On the effective number of climate models, J. Climate, 24, 2358–2367, doi:10.1175/2010JCLI3814.1, 2010.

Petters, M. D. and Kreidenweis, S. M.: A single parameter representation of hygroscopic growth and cloud condensation nucleus activity, Atmos. Chem. Phys., 7, 1961–1971, doi:10.5194/acp-7-1961-2007, 2007.

Rasinski, P., Pawlowska, H., and Grabowski, W.: Observations and kinematic modeling of drizzling marine stratocumulus, Atmos. Res., 102, 120–135, doi:10.1016/j.atmosres.2011.06.020, 2011.

Riechelmann, T., Noh, Y., and Raasch, S.: A new method for large-eddy simulations of clouds with Lagrangian droplets including the effects of turbulent collision, New J. Phys., 14, 065008, doi:10.1088/1367-2630/14/6/065008, 2012.

Rogers, J. and Davis, R.: The effects of van der Waals attractions on cloud droplet growth by coalescence, J. Atmos. Sci., 47, 1075–1080, doi:10.1175/1520-0469(1990)047<1075:TEOVDW>2.0.CO;2, 1990.

Schabel, M. and Watanabe, S.: Boost.Units: Zero-overhead dimensional analysis and unit/quantity manipulation and conversion, in: Boost Library Documentation, available at: http://www.boost.org/doc/libs/ (last access: 15 November 2014), 2008.

Shima, S., Sugiyama, T., Kusano, K., Kawano, A., and Hirose, S.: Simulation method, simulation program, and simulator, European Patent EP1847939, 2007.

Shima, S., Kusano, K., Kawano, A., Sugiyama, T., and Kawahara, S.: The super-droplet method for the numerical simulation of clouds and precipitation: a particle-based and probabilistic microphysics model coupled with a non-hydrostatic model, Q. J. Roy. Meteor. Soc., 135, 1307–1320, doi:10.1002/qj.441, 2009.

Simmel, M., Trautmann, T., and Tetzlaff, G.: Numerical solution of the stochastic collection equation–comparison of the Linear Discrete Method with other methods, Atmos. Res., 61, 135–148, doi:10.1016/S0169-8095(01)00131-4, 2002.

Slawinska, J., Grabowski, W. W., and Morrison, H.: The impact of atmospheric aerosols on precipitation from deep organized convection: a prescribed-flow model study using double-moment

bulk microphysics, Q. J. Roy. Meteor. Soc., 135, 1906–1913, doi:10.1002/qj.450, 2009.

Smolarkiewicz, P.: Multidimensional positive definite advection transport algorithm: an overview, Int. J. Numer. Meth. Fl., 50, 1123–1144, doi:10.1002/fld.1071, 2006.

Smolík, J., Džumbová, L., Schwarz, J., and Kulmala, M.: Evaporation of ventilated water droplet: connection between heat and mass transfer, J. Aerosol Sci., 32, 739–748, doi:10.1016/S0021-8502(00)00118-X, 2001.

Sölch, I. and Kärcher, B.: A large-eddy model for cirrus clouds with explicit aerosol and ice microphysics and Lagrangian ice particle tracking, Q. J. Roy. Meteor. Soc., 136, 2074–2093, doi:10.1002/qj.689, 2010.

Stevens, B. and Feingold, G.: Untangling aerosol effects on clouds and precipitation in a buffered system, Nature, 461, 607–613, doi:10.1038/nature08281, 2009.

Straka, J.: Cloud and Precipitation Microphysics: Principles and Parameterizations, Cambridge University Press, 2009.

Szakáll, M., Mitra, S. K., Diehl, K., and Borrmann, S.: Shapes and oscillations of falling raindrops – a review, Atmos. Res., 97, 416–425, doi:10.1016/j.atmosres.2010.03.024, 2010.

Szumowski, M., Grabowski, W., and Ochs III, H.: Simple two-dimensional kinematic framework designed to test warm rain microphysical models, Atmos. Res., 45, 299–326, doi:10.1016/S0169-8095(97)00082-3, 1998.

Unterstrasser, S. and Sölch, I.: Optimisation of the simulation particle number in a Lagrangian ice microphysical model, Geosci. Model Dev., 7, 695–709, doi:10.5194/gmd-7-695-2014, 2014.

Vaillancourt, P., Yau, M., and Grabowski, W.: Microscopic approach to cloud droplet growth by condensation. Part I: Model description and results without turbulence, J. Atmos. Sci., 58, 1945–1964, doi:10.1175/1520-0469(2001)058<1945:MATCDG>2.0.CO;2, 2001.

Vallis, G.: Atmospheric and oceanic fluid dynamics: fundamentals and large-scale circulation, Cambridge University Press, Cambridge, 2006.

Veldhuizen, T.: Blitz++ User's Guide: a C++ class library for scientific computing, version 0.9, available at: http://blitz.sf.net/resources/blitz-0.9.pdf (last access: 15 November 2014), 2005.

Vohl, O., Mitra, S., Wurzler, S., Diehl, K., and Pruppacher, H.: Collision efficiencies empirically determined from laboratory investigations of collisional growth of small raindrops in a laminar flow field, Atmos. Res., 85, 120–125, doi:10.1016/j.atmosres.2006.12.001, 2007.

Wilson, G., Aruliah, D. A., Titus Brown, C., Chue Hong, N. P., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K., Mitchell, I. M., Plumbley, M., Waugh, B., White, E. P., and Wilson, P.: Best practices for scientific computing, PLoS Biol., 12, e1001745, doi:10.1371/journal.pbio.1001745, 2014.

Wood, R.: Drizzle in stratiform boundary layer clouds. Part II: Microphysical aspects, J. Atmos. Sci., 62, 3034–3050, doi:10.1175/JAS3530.1, 2005.

Table A1. List of symbols.

Symbol	SI unit	Description
$A = 2\sigma_w/(R_\mathrm{v} T \rho_w)$	[m]	Kelvin term exponent parameter
β_M, β_T	[1]	transition-régime correction factors
Δt, Δx, Δz, ΔV	[s] or [m] or [m^3]	timestep, grid cell dimensions and volume
θ_l	[K]	liquid water potential temperature (cf. Sect. 2.2)
θ	[K]	potential temperature
κ	[1]	hygroscopicity parameter
ρ_i	depends on i	any state variable (density)
ρ_d, ρ_v	[$\mathrm{kg\,m^{-3}}$]	densities of dry air and vapour vapour
ρ_c, ρ_r	[$\mathrm{kg\,m^{-3}}$]	cloud and rain water densities/content
$\rho_w = 1000$	[$\mathrm{kg\,m^{-3}}$]	density of liquid water
ρ_vs	[$\mathrm{kg\,m^{-3}}$]	saturation vapour density
$\rho_\circ$	[$\mathrm{kg\,m^{-3}}$]	vapour density at drop surface
$\dot\rho_i$, $\dot\rho_c$, $\dot\rho_r$	depends on i	rhs terms (any, cloud water, rain water)
σ_m	[1]	geometric standard deviation (lognormal spectrum)
$\sigma_w = 0.072$	[$\mathrm{J\,m^{-2}}$]	surface tension coefficient of water
τ, τ_rlx	[s]	relaxation time scale (cf. Sect. 2.2)
ϕ_i	depends on i	any advected specific quantity (e.g. mixing ratio)
ψ	[$\mathrm{kg\,m^{-1}\,s^{-1}}$]	streamfunction
$a_w = (r_\mathrm{w}^3 - r_\mathrm{d}^3)/(r_\mathrm{w}^3 - r_\mathrm{d}^3 \cdot (1-\kappa))$	[1]	water activity
a, b	[m^2]	initial interval for bisection algorithm
$c_\mathrm{pd} = 1005$, $c_\mathrm{pv} = 1850$, $c_\mathrm{l} = 4218$	[$\mathrm{J\,kg^{-1}\,K^{-1}}$]	specific heat at const. pressure (dry air, vapour & liquid water)
C	[1]	Courant number
d_m, r_m	[m]	mode diameter and radius (lognormal spectrum)
D, D_eff, D_0	[$\mathrm{m^2\,s^{-1}}$]	diffusion coefficients for water vapour in air
E_r	[$\mathrm{kg\,m^{-3}\,s^{-1}}$]	evaporation rate of rain (single-moment scheme)
$E(r_i, r_j)$	[1]	collection efficiency
F_in, F_out	[$\mathrm{kg\,m^{-3}\,s^{-1}}$]	fluxes of ρ_r through the grid cell edges
K, K_0	[$\mathrm{J\,m^{-1}\,s^{-1}\,K^{-1}}$]	thermal conductivities of air
$K(r_i, r_j)$	[$\mathrm{m^3\,s^{-1}}$]	collection kernel
$l_{v0} = 2.5 \times 10^6$	[$\mathrm{J\,kg^{-1}}$]	latent heat of evaporation at the triple point
$l_v(T) = l_{v0} + (c_\mathrm{pv} - c_l) \cdot (T - T_0)$	[$\mathrm{J\,kg^{-1}}$]	latent heat of evaporation at a given temperature
$M^{[k]}$	[$\mathrm{m^{-3+k}}$]	kth moment of size spectrum
n	[1]	total number of computational particles
n_c, n_r	[$\mathrm{m^{-3}}$]	cloud droplet and rain drop concentrations
N	[1]	multiplicity (attribute of computational particle)
N_m	[$\mathrm{m^{-3}}$]	particle concentration (lognormal spectrum)
p, p_d	[Pa]	pressure, dry air partial pressure
P_{ij}	[1]	probability of collisions
Q, q	[$\mathrm{J\,m^{-3}}$], [$\mathrm{J\,kg^{-1}}$]	heat per unit volume and mass
r_d, r_w	[m]	particle dry and wet radii
r_c0	[$\mathrm{kg\,kg^{-1}}$]	autoconversion threshold (mixing ratio)
r_v, r_l, $r_\mathrm{t} = r_\mathrm{v} + r_\mathrm{l}$	[$\mathrm{kg\,kg^{-1}}$]	mixing ratios (vapour, liquid, total)
R_v, R_d	[$\mathrm{J\,K^{-1}\,kg^{-1}}$]	gas constants for water vapour and dry air
S, s	[$\mathrm{J\,K^{-1}\,m^{-3}}$], [$\mathrm{J\,K^{-1}\,kg^{-1}}$]	entropy per unit volume and mass
T	[K]	temperature
$\boldsymbol{u} = (u, v)$	[$\mathrm{m\,s^{-1}}$]	velocity field
v_t, v_i, v_j	[$\mathrm{m\,s^{-1}}$]	terminal velocity
w_max	[$\mathrm{m\,s^{-1}}$]	maximum velocity (cf. amplitude of ψ)
w	[1]	averaging weight in particle advection scheme
x, z	[m]	spatial coordinate
X, Z	[m]	domain extent